INSTRUCTIONS FOR USE

PLC Automation

Automation Builder, AC500

Automation Builder 2.5.0, AC500 V2, AC500-eCo V2, AC500-XC V2

Table of contents

1 PLC Automation with V2 CPUs

1.1 About this document

1.1.1 Documentation structure

Chapter 1.1.4 “Use the "magic button" to display your current position in the table of contents” on page 12.

See also chapter " Your tasks - documentation from the user's point of view" on page 10.

Table 1: Guidance for this documentation: Main chapters

Getting started Basic information to start with Automation Builder and AC500 PLC, e.g., licensing, GUI explanations, example projects.	*Chapter 1.2 “Getting started” on page 12*
Automation Builder installation manager Add, remove or modify software packages in Automation Builder.	*Chapter 1.3 “Automation Builder installation manager” on page 138*
Programming with CODESYS Information about IEC programming in Automation Builder, including description of CODESYS libraries.	*Chapter 1.4 “Programming with CODESYS” on page 145*
Libraries and solutions ABB libraries. Overview and description of integrated standard libraries and solution libraries available as library packages. Explanation of the concept of solution libraries ("system technology"). Description of the library elements, like function blocks and functions.	*AC500 V3 library descriptions: Chapter 1.5 " Libraries and solutions" on page 735*
PLC integration (hardware) Hardware description and specifications. Overview on module variants, connections, technical data, order data, assembly of modules. Device configuration in Automation Builder. Explanation of system behavior ("system technology"), interaction between PLC behavior (firmware), configuration, programming and use cases.	*Chapter 1.6 “PLC integration (hardware)” on page 3697*
Diagnosis and debugging Explanation of the diagnosis system in the PLC, the display of error messages at the CPU and in IEC applications. Online diagnosis in Automation Builder. List of diagnosis and error messages.	*Chapter 1.7 “Diagnosis and debugging for AC500 V2 products” on page 6365*
Engineering interfaces and tools Information on add-on packages, e.g., for security static analysis or Project Version Control. Mainly for advanced users.	*Chapter 1.8 “Engineering interfaces and tools” on page 6550*
Human machine interface (HMI) Information on HMI with Automation Builder. Configuration of HMI devices in Automation Builder.	*Chapter 1.9 “Human machine interface” on page 6628*
Contact ABB Contact information about our sales and support teams.	*Chapter 1.10 “Contact ABB” on page 6640*

1.1.2 Your tasks - documentation from the user's point of view

All information about **AC500**, **AC500-XC** and **AC500-eCo** is available in this manual.

All information about **AC500-S** and **AC500-S-XC** is available online in the *safety user manual*.

↳ *Chapter 1.1.4 “Use the "magic button" to display your current position in the table of contents” on page 12.*

↳ *See also chapter " Documentation structure" on page 9.*

As a mechanical/electrical designer

PLC system description	↳ *Chapter 1.6.1.8 “PLC system description” on page 3731*
Hardware descriptions	↳ *Chapter 1.6.2 “Device specifications” on page 3785*
Comparison of product features	*For AC500 V2 available via product catalog (online)*

As a switchgear cabinet manufacturer

Assembly of modules	↳ *Chapter 1.6.3 “System assembly, construction and connection” on page 5213*
Connection of modules	In the device specifications, select the desired product to access the connection for this device ↳ *Chapter 1.6.2 “Device specifications” on page 3785*. *“Device specifications ➔ Product group ➔ Product type ➔ Electrical connection”*
Installation instructions	*AC500 V2 + V3 (online)*

As a programming engineer

Getting started with Automation Builder	↳ *Chapter 1.2 “Getting started” on page 12*
Installation of Automation Builder	*AC500 V2 + V3 (online)*
License management for Automation Builder	↳ *Chapter 1.2.4 “Managing your licenses” on page 20*
Getting started with example projects	↳ *Chapter 1.2.18 “Example projects” on page 67*
Firmware update	↳ *Chapter 1.6.5.1.7 “Firmware identification and update” on page 5786*
Configuration of PLC hardware in Automation Builder	↳ *Chapter 1.6.5 “Configuration in Automation Builder for AC500 V2 products” on page 5757*
Programming with CODESYS	↳ *Chapter 1.4 “Programming with CODESYS” on page 145*
Function block libraries	Libraries by ABB ↳ *Chapter 1.5 “Libraries and solutions” on page 735* CODESYS libraries by 3S ↳ *Chapter 1.4.2 “Libraries” on page 532*
System behavior ("system technology"), interaction between PLC (firmware), configuration, programming and use cases.	↳ *Chapter 1.6.4 “System technology for AC500 V2 products” on page 5394*

Visualization and web visualization: Example projects	*Chapter 1.2.18.1.2 “Example project for central I/O expansion” on page 70*
Visualization and web visualization	*Chapter 1.4.3 “Visualization” on page 636*
Add, remove or modify software packages in Automation Builder via installation manager	*Chapter 1.3 “Automation Builder installation manager” on page 138*
Add-on software packages	*Chapter 1.8 “Engineering interfaces and tools” on page 6550*
HMI, e.g., interface to control panels and SCADA systems	*Chapter 1.9 “Human machine interface” on page 6628*

As a commissioning engineer

Function block libraries	Libraries by ABB *Chapter 1.5 “Libraries and solutions” on page 735* CODESYS libraries by 3S *Chapter 1.4.2 “Libraries” on page 532*
Hardware descriptions	*Chapter 1.6.2 “Device specifications” on page 3785*
Diagnosis and debugging	*Chapter 1.7 “Diagnosis and debugging for AC500 V2 products” on page 6365*

As a service engineer

Diagnosis and debugging	*Chapter 1.7 “Diagnosis and debugging for AC500 V2 products” on page 6365*
List of diagnosis and error messages	*Chapter 1.7.3 “Diagnosis messages” on page 6429*
Contact the PLC support team	*Chapter 1.10 “Contact ABB” on page 6640*

As a specialist for AC500 V2 CPU range, new to AC500 V3 CPU range

AC500 V3 CPU specifications	*Chapter 1.6.2.3 “Processor modules” on page 3803*
Comparison of product features	*For AC500 V2 available via product catalog (online)*
Convert an AC500 V2 project to an AC500 V3 project	*Chapter 1.2.17 “Converting an AC500 V2 project to an AC500 V3 project” on page 67*
Compatible modules with AC500 CPUs	*Chapter 1.6.2 “Device specifications” on page 3785*
Documentation for AC500 V3	*AC500 V3 (online)*

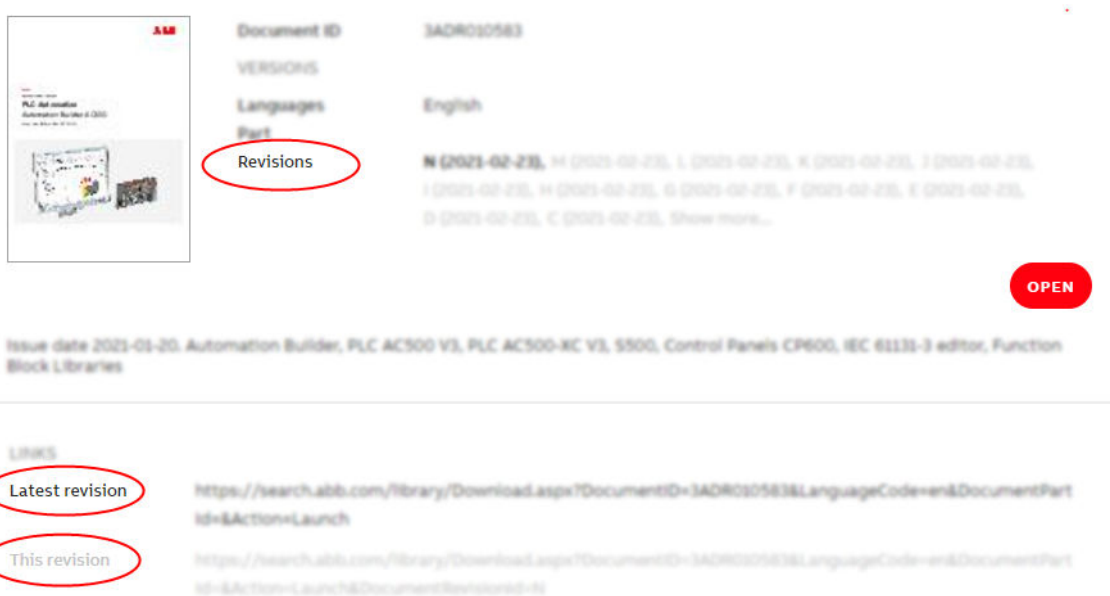
As a specialist for PLCs, new to AC500 PLC

Getting started with engineering suite Automation Builder	*Chapter 1.2 “Getting started” on page 12*
PLC system description	*Chapter 1.6.1.8 “PLC system description” on page 3731*
Hardware descriptions	*Chapter 1.6.2 “Device specifications” on page 3785*
System technology: System behavior, interaction between PLC behavior (firmware), configuration, programming and use cases.	*Chapter 1.6.4 “System technology for AC500 V2 products” on page 5394*

1.1.3 Older revisions of this document

You can always find all revisions of our documents on our website.

AC500 V2 (online)



Revisions	Select any of the revisions
Latest revision	Get a link to the always latest revision
This revision	Get a direct link to the selected revision

1.1.4 Use the "magic button" to display your current position in the table of contents

☑ Documentation is opened in a PDF reader. PDF readers often provide a button to synchronize with the table of contents. Usually, you can find the "magic button" in the bookmarks tab. For example, it looks like this: /

▷ Select the "magic button".

⇨ Your current position will be highlighted in the bookmark tab.

1.2 Getting started

ABB Automation Builder - One holistic suite

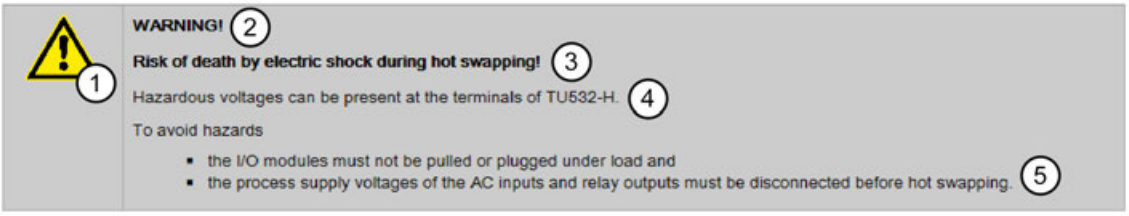
ABB Automation Builder is the integrated software suite for machine builders and system integrators wanting to automate their machines and systems in a productive way. Combining the tools required for configuring, programming, debugging and maintaining automation projects from a common intuitive interface, Automation Builder addresses the largest single cost element of most of today's industrial automation projects: software. ABB Automation Builder covers the engineering of ABB PLCs, Safety PLCs, control panels, drives, motion and robots.

Automation Builder ReadMe

Before starting Automation Builder configuration read the version specific information provided in the Automation Builder readme file. It describes new features and functions as well as workarounds on known problems. The readme file is stored in the installation directory of Automation Builder, however can be downloaded as well from ABB website http://new.abb.com/plc/automationbuilder.

1.2.1 Safety notice

Throughout the documentation we use the following types of safety and information notices according to ANSI Z535 make you aware of safety considerations or advice on AC500 products usage.



1 **Safety alert symbol** indicates the danger.
2 **Signal word** classifies the danger.
3 **Type and source of the risk** are mentioned.
4 **Possible consequences** of the risk are described.
5 **Measures to avoid these consequences** (enumerations).

Signal words

DANGER!
DANGER indicates a hazardous situation which, if not avoided, will result in death or serious injury.

Ensure to take measures to prevent the described impending danger.

WARNING!
WARNING indicates a hazardous situation which, if not avoided, could result in death or serious injury.

Ensure to take measures to prevent the described dangerous situation.

CAUTION!
CAUTION indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

Ensure to take measures to prevent the described dangerous situation.

NOTICE!
NOTICE is used to address practices not related to physical injury but might lead to property damage for example damage of the product.

Ensure to take measures to prevent the described dangerous situation.

NOTE provides additional information on the product, e.g. advices for configuration or best practice scenarios.

1.2.2 Cyber security

Cyber security disclaimer

This product is designed to be connected to and to communicate information and data via a network interface. It is your sole responsibility to provide and continuously ensure a secure connection between the product and your network or any other network (as the case may be). You shall establish and maintain any appropriate measures (such as but not limited to the installation of firewalls, application of authentication measures, encryption of data, installation of anti-virus programs, etc.) to protect the product, the network, its system and the interface against any kind of security breaches, unauthorized access, interference, intrusion, leakage and/or theft of data or information. ABB Ltd and its affiliates are not liable for damages and/or losses related to such security breaches, any unauthorized access, interference, intrusion, leakage and/or theft of data or information.

Although ABB provides functionality testing on the products and updates that we release, you should institute your own testing program for any product updates or other major system updates (to include but not limited to code changes, configuration file changes, third party software updates or patches, hardware exchanges, etc.) to ensure that the security measures that you have implemented have not been compromised and system functionality in your environment is as expected. This also applies to the operating system. Security measures (such as but not limited to the installation of latest patches, installation of firewalls, application of authentication measures, installation of anti-virus programs, etc.) are in your responsibility. You have to be aware that operating systems provide a considerable number of open ports that should be monitored carefully for any threats.

It has to be considered that online connections to any devices are not secured. It is your responsibility to assure that connections are established to the correct device (and e.g. not to an unknown device pretending to be a known device type). Furthermore you have to take care that confidential data exchanged with the PLC is either compiled or encrypted.

Security related deployment guidelines for industrial automation

Security details for industrial automation is provided in a *whitepaper* on ABB website.

Secure communication

Whenever possible, use an encrypted communication between AC500 V3 devices and third party devices, such as HMI devices. This is necessary to protect passwords and other data.

Frequently asked questions

For more information around cyber security please see our *FAQ*.

1.2.2.1 Defense in depth

The defense in depth approach implements multi-layer IT security measures. Each layer provides its special security measures. All deployed security mechanisms in the system must be updated regularly. It is also important to follow the system vendor’s recommendations on how to configure and use these mechanisms. As a basis, the components must include security functions such as:

- Virus protection
- Firewall protection
- Strong and regularly changed passwords
- User management
- Using VPN tunnels for connections between networks

Additional security components such as routers and switches with integrated firewalls should be available. A defined user and rights concept managing access to the controllers and their networks is mandatory. Finally, the manufacturer of the components should be able to quickly discover weaknesses and provide patches.

> *Only used services/ports should be enabled (e.g. to enable the functionality of an FTPS server).*

References: *CODESYS Security Whitepaper*

Security zones

IT resources vary in the extent to which they can be trusted. A common security architecture is therefore based on a layered approach that uses zones of trust to provide increasing levels of security according to increasing security needs. Less-trusted zones contain more-trusted zones and connections between the zones are only possible through secure interconnections such as firewallsFig. 1. All resources in the same zone must have the same minimum level of trust. The inner layers, where communication interaction needs to flow freely between nodes, must have the highest level of trust. This is the approach described in the IEC 62443 series of standards.

Firewalls, gateways, and proxies are used to control network traffic between zones of different security levels, and to filter out any undesirable or dangerous material. Traffic that is allowed to pass between zones should be limited to what is absolutely necessary because each type of service call or information exchange translates into a possible route that an intruder may be able to exploit. Different types of services represent different risks. Internet access, incoming e-mail and instant messaging, for example, represent very high risks.

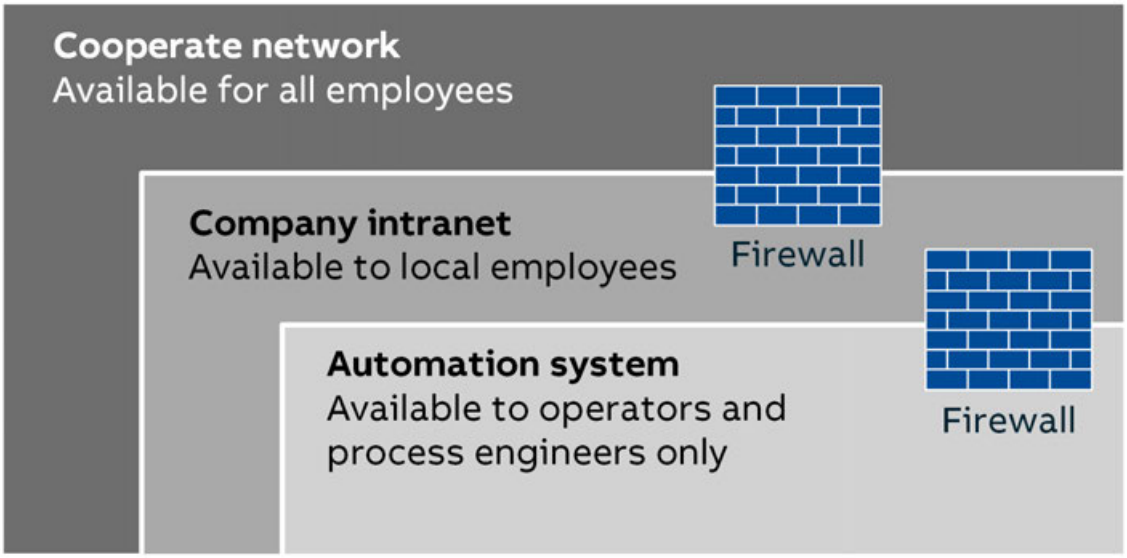


Fig. 1: Security zones

Fig. 1 shows three security zones, but the number of zones does not have to be as many or as few as three. The use of multiple zones allows access between zones of different trust levels to be controlled to protect a trusted resource from attack by a less trusted one.

High-security zones should be kept small and independent. They need to be physically protected, i.e. physical access to computers, network equipment and network cables must be limited by physical means to authorized persons only. A high-security zone should obviously not depend on resources in a less secure zone for its security. Therefore, it should form its own domain that is administered from the inside, and not depend on, e.g., a domain controller in a less secure network.

Even if a network zone is regarded as trusted, an attack is still possible: by a user or compromised resource that is inside the trusted zone, or by an outside user or resource that succeeds to penetrate the secure interconnection. Trust therefore depends also upon the types of measures taken to detect and prevent compromise of resources and violation of the security policy.

References: *Security for Industrial Automation and Control Systems*

1.2.2.2 Secure operation

The controller must be located in a protected environment in order to avoid accidental or intended access to the controller or the application.

A protected environment can be:

- Locked control cabinets without connection from outside
- No direct internet connection
- Use firewalls and VPN to separate different networks
- Separate different production areas with different access controls

To increase security, physical access protection measures such as fences, turnstiles, cameras or card readers can be added.

Follow these rules for the protected environment:

- Keep the trusted network as small as possible and independent from other networks.
- Protect the cross-communication of controllers and the communication between controllers and field devices via standard communication protocols (fieldbus systems) using appropriate measures.
- Protect such networks from unauthorized physical access.
- Use fieldbus systems only in protected environments. They are not protected by additional measures, such as encryption. Open physical or data access to fieldbus systems and their components is a serious security risk.
- Physically protect all equipment, i.e., ensure that physical access to computers, network equipment and cables, controllers, I/O systems, power supplies, etc., is limited to authorized persons
- When connecting a trusted network zone to outer networks, make sure that all connections are through properly configured secure interconnections only, such as a firewall or a system of firewalls, which is configured for “deny by default”, i.e., blocks everything except traffic that is explicitly needed to fulfill operational requirements.
- Allow only authorized users to log on to the system, and enforce strong passwords that are changed regularly.
- Continuously maintain the definitions of authorized users, user groups, and access rights, to properly reflect the current authorities and responsibilities of all individuals at all times. Users should not have more privileges than they need to do their job.
- Do not use the system for e-mail, instant messaging, or internet browsing. Use separate computers and networks for these functions if they are needed.
- Do not allow installation of any unauthorized software in the system.
- Restrict temporary connection of portable computers, USB memory sticks and other removable data carriers. Computers that can be physically accessed by regular users should have ports for removable data carriers disabled.
- If portable computers need to be connected, e.g., for service or maintenance purposes, they should be carefully scanned for viruses immediately before connection.
- All CDs, DVDs, USB memory sticks and other removable data carriers, and files with software or software updates, should also be checked for viruses before being introduced into the trusted zone.
- Continuously monitor the system for intrusion attempts.
- Define and maintain plans for incident response, including how to recover from potential disasters.
- Regularly review the organization as well as technical systems and installations with respect to compliance with security policies, procedures and practices.

A protected local control cabinet could look like in figure 2, page 17. This network is not connected to any external network. Security is primarily a matter of physically protecting the automation system and preventing unauthorized users from accessing the system and from connecting or installing unauthorized hardware and software.

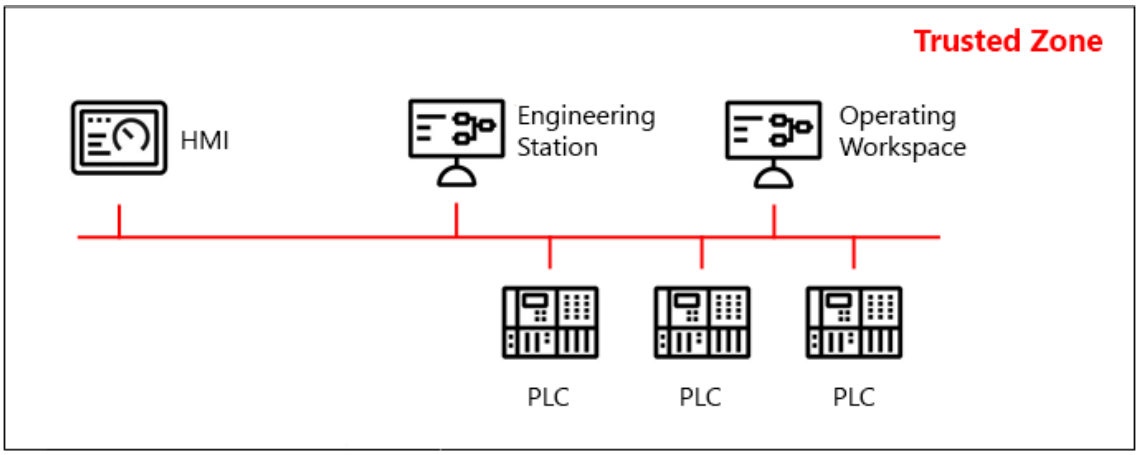


Fig. 2: Isolated automation system

Servers and workplaces that are not directly involved in the control and monitoring of the process should preferably be connected to a subnet that is separated from the automation system network by means of a router/firewall. This makes it possible to better control the network load and to limit access to certain servers on the automation system network. Note that servers and workplaces on this subnet are part of the trusted zone and thus need to be subject to the same security precautions as the nodes on the automation system network.

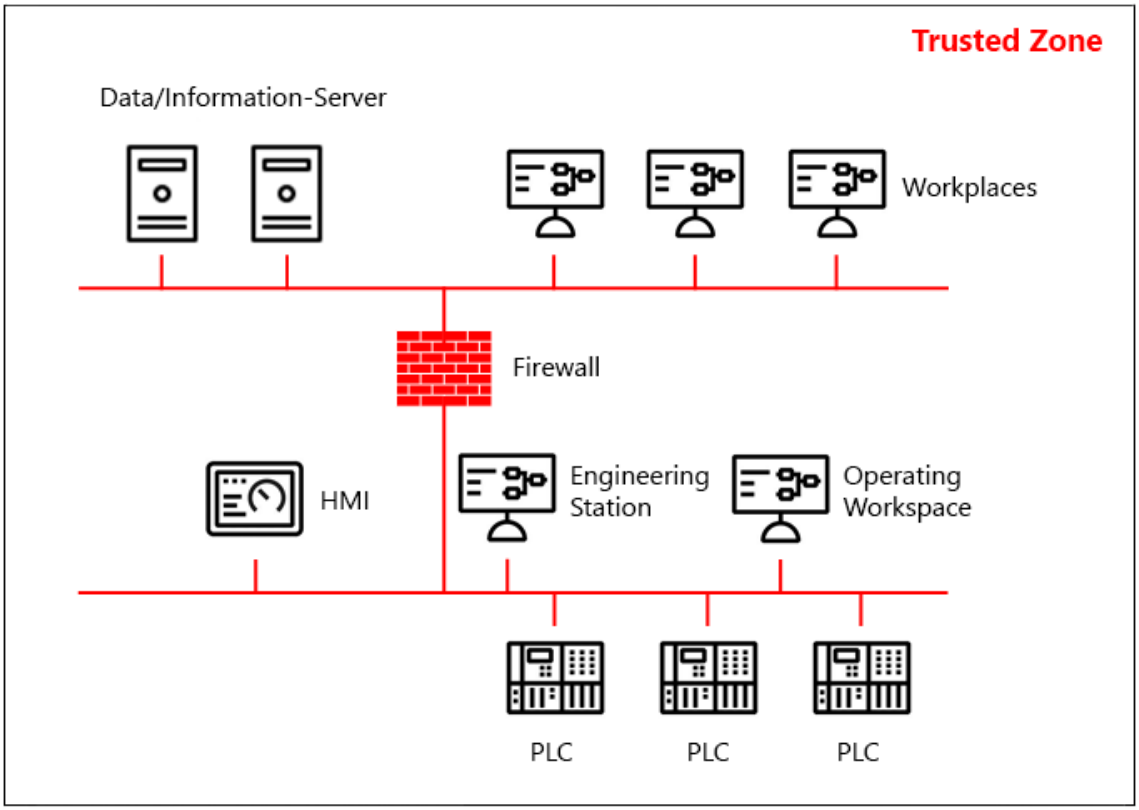


Fig. 3: Plant information network connected to an automation system

For the purposes of process control security, a general-purpose information system (IS) network should not be considered a trusted network, not the least since such networks are normally further connected to the Internet or other external networks. The IS network is therefore a different lower-security zone, and it should be separated from the automation system by means of a firewall. The IS and automation system networks should form separate domains.

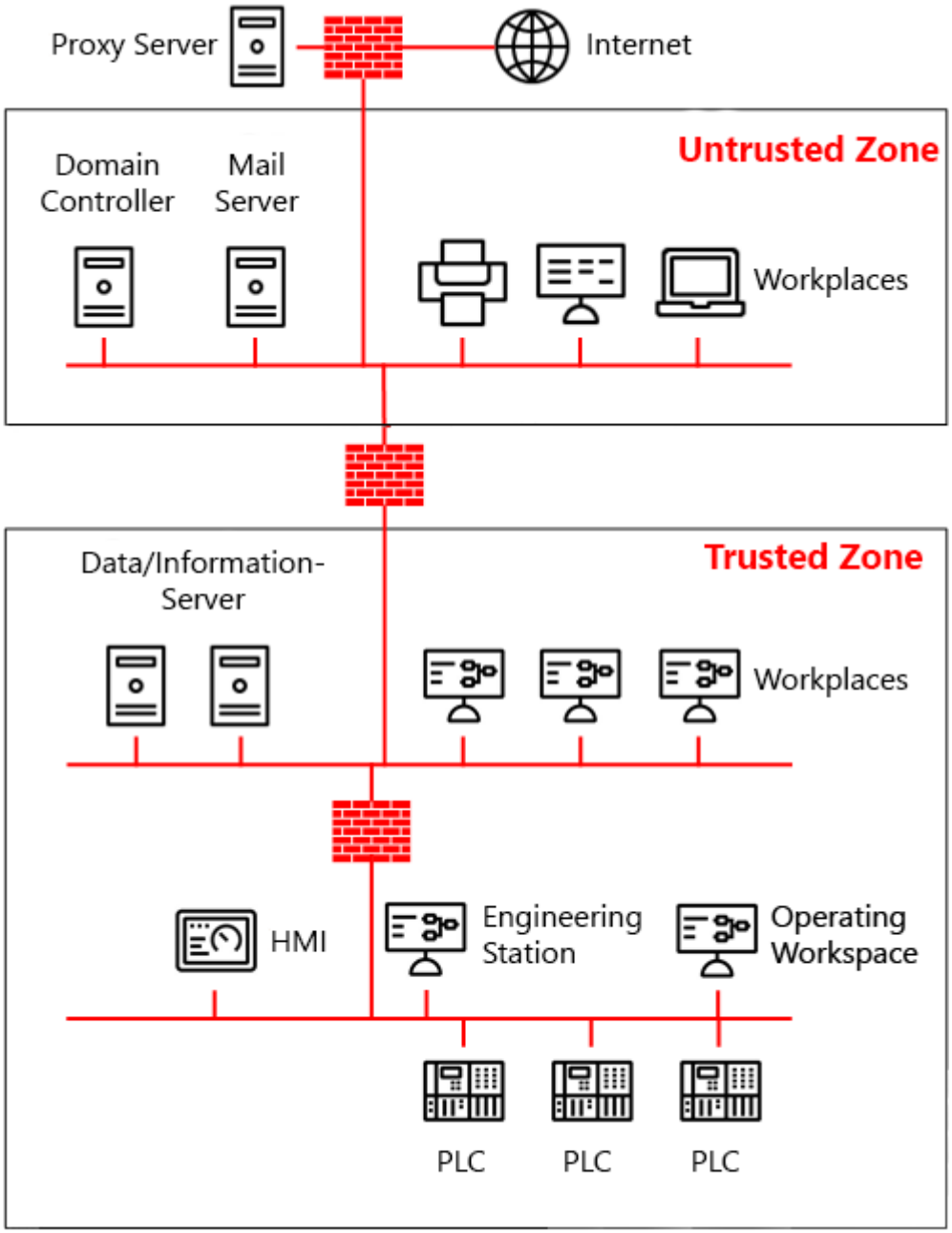


Fig. 4: Automation system and IS network

1.2.2.3 Hardening

System hardening means to eliminate as many security risks as possible. Hardening your system is an important step to protect your personal data and information. This process intends to eliminate attacks by patching vulnerabilities and turning off inessential services. Hardening a system involves several steps to form layers of protection.

Commissioning phase

- Protect the hardware from unauthorized access
- Be sure the hardware is based on a secure environment
- Disable unused software and services (network ports)
- Install firewalls
- Disallow file sharing among programs
- Install virus and spyware protection

- Use containers or virtual machines
- Create strong passwords by applying a strong password policy
- Create and keep backups
- Use encryption when possible
- Disable weak encryption algorithms
- Separate data and programs
- Enable and use disk quotas
- Strong logical access control
- Adjust default settings, especially passwords

Verification phase

- Verification of antivirus - Check antivirus is active and updated
- Verification of the identification - Check that test and unauthorized accounts are removed
- Verification of intrusion detection systems - Check malicious traffic is blocked
- Verification of audit logging - Check audit log is enabled
- You can use the checklist out of the *cyber security white paper*

Operation phase

- Keep software up-to-date, especially by applying security patches
- Keep antivirus up and running
- Keep antivirus definitions up-to-date
- Delete unused user accounts
- Lock an active session whenever it is unattended, e.g., lock the screen of the PC or of the control panel (HMI)

Decommissioning phase

- Delete all credentials stored in the device like certificates and user data *Chapter 1.6.3.4.6 “Decommissioning” on page 5233*.

References: *Hardening in Wikipedia (2021)*

1.2.2.4 Open Ports and Services

Overview of minimum cyber security requirements for open ports and services settings.

Port	Protocol	Description
1217	TCP	CODESYS Gateway V3
1210	TCP	CODESYS Gateway V2
1211	TCP	CODESYS Gateway V2
22350	TCP/UDP	CodeMeter License Server (runtime) – license
22352	HTTP	CodeMeter License Server (runtime) – WebAdmin
22353	HTTPS	CodeMeter License Server (runtime) – WebAdmin
11030	HTTP	Python editor server

1.2.3 Automation Builder update notification

A notification dialog will be shown if there are any updates available for the currently installed version on every launch of the Automation Builder.

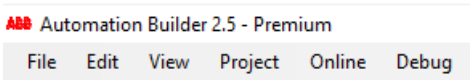
Chapter 1.3.3 “Automation Builder update notification” on page 141

1.2.4 Managing your licenses

After installing and licensing the Automation Builder you can manage your licenses in various ways.

1.2.4.1 Identifying the installed license

Since Automation Builder Version 1.1.1 the title bar or Automation Builder shows a license information:



Be aware of the following rule for this information:

The info in the menubar is taken in this order from the first found license

- local licenses (on PC)
- on dongle (USB key)
- network licenses (since AB1.2)

So if a local license is only basic and a dongle with premium is inserted:

- the information in the menubar is basic
- the functionality is premium (the highest available)

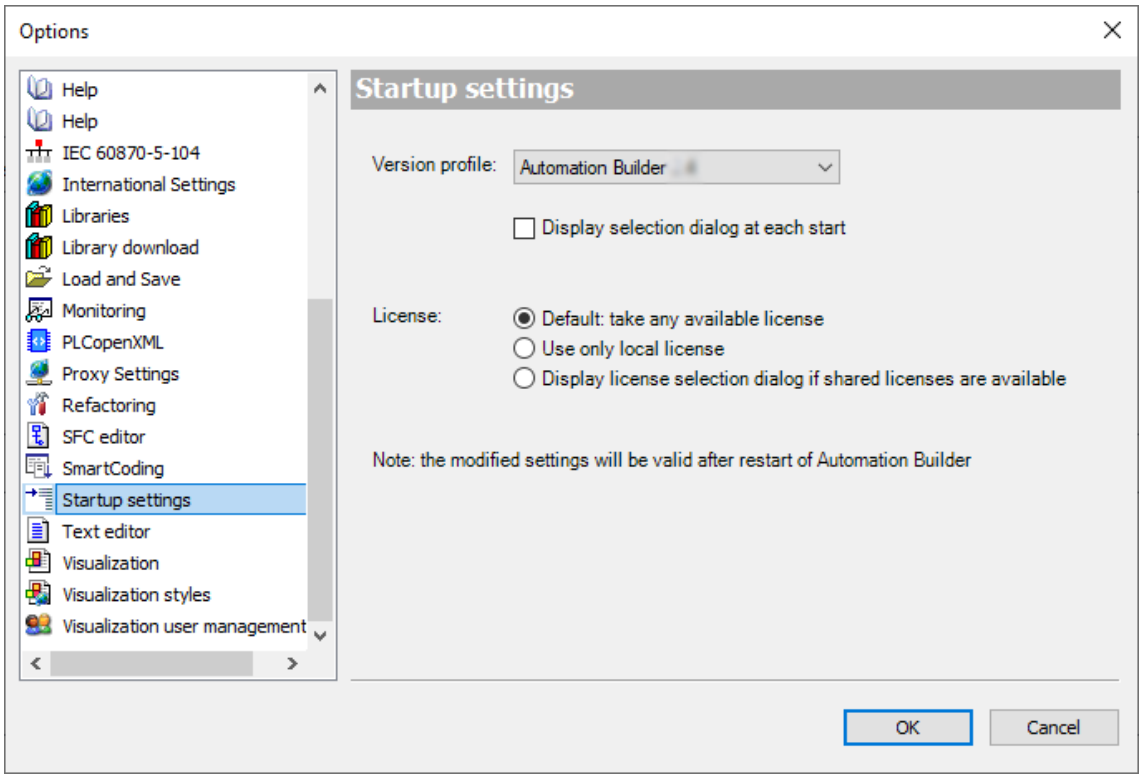
To check your installed licenses, the CodeMeter Control Center tool can be used ↳ *Chapter 1.2.4.3 “Checking licenses with CodeMeter control center” on page 22*.

1.2.4.2 Selecting the license used on Automation Builder startup

You can select, which license the Automation Builder should use on startup.

To select which license should be used:

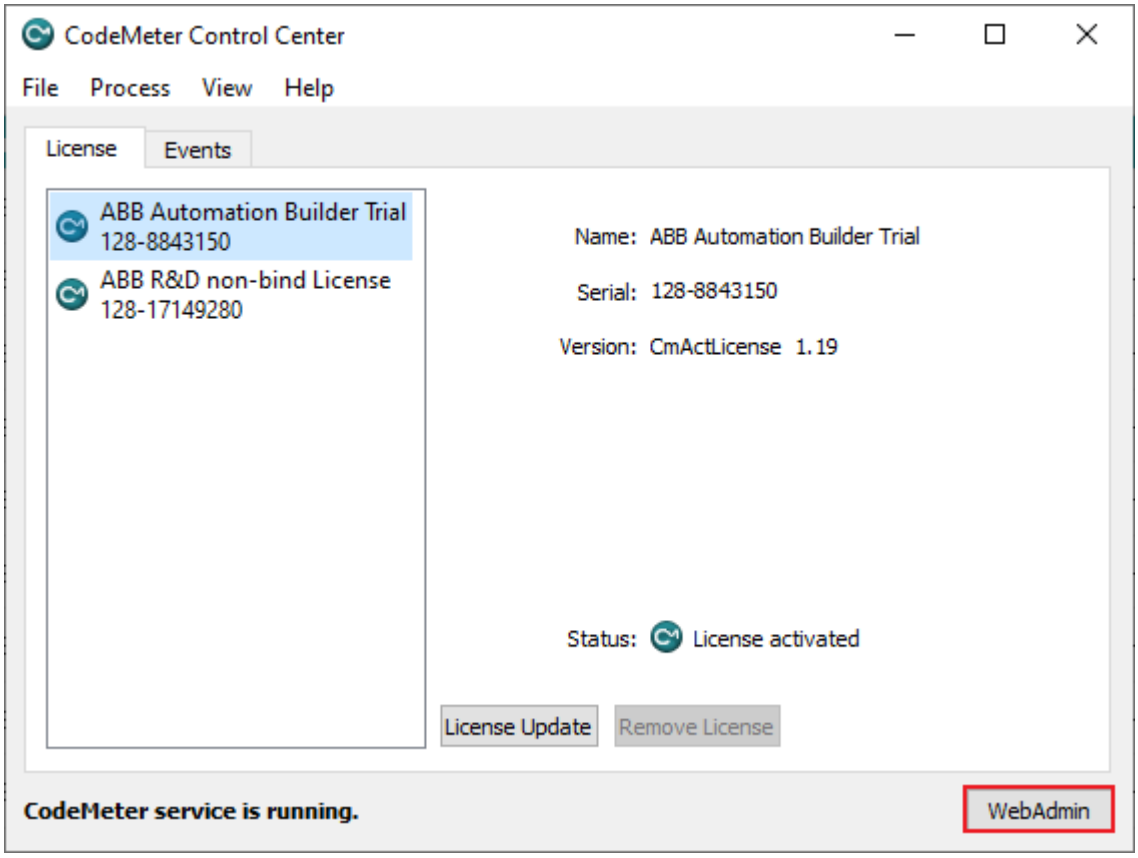
1. In the Automation Builder menu select *“Tools ➔ Options”*.
   ⇨ The Options window is opened.
2. In *“Startup settings”* under *“License”* select which license should be used.
   - Default: The most comprehensive available license will be selected
   - Use only local license: Network licenses will never be selected
   - Display licenses selection dialog if shared licenses are available: On every Automation Builder startup, you will be asked to select a license



3. To apply the setting select *“OK”*.

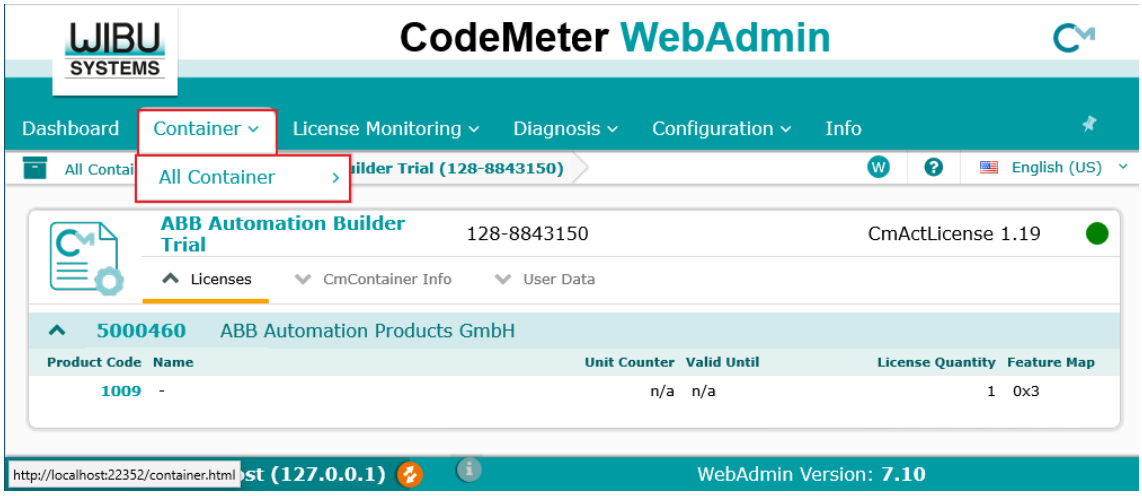
1.2.4.3 Checking licenses with CodeMeter control center

1. Open the CodeMeter Control Center via the *“Windows start menu ➔ CodeMeter ➔ CodeMeter Control Center”*.



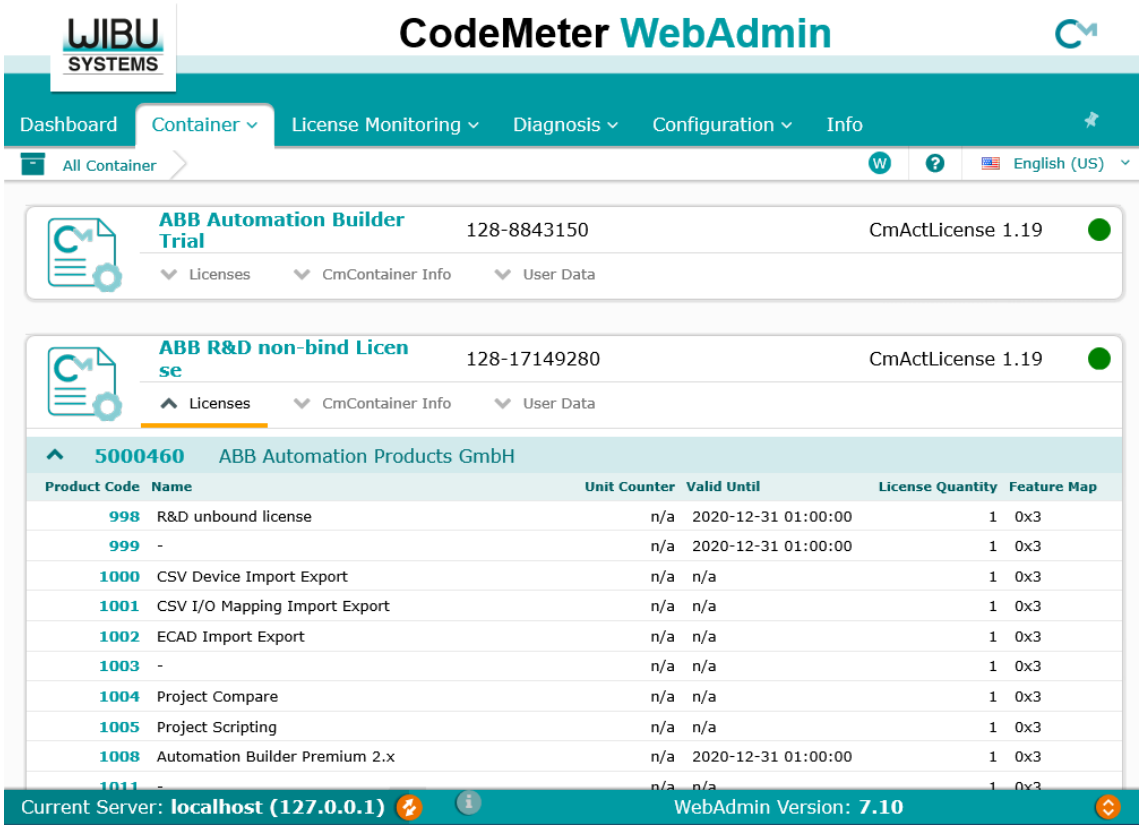
2. In the CodeMeter Control Center window you can see the different license “tickets” / “CmContainers” that are installed on your PC.

   To see more details, open the CodeMeter WebAdmin by selecting *“WebAdmin”*



3. Select *“Container ➔ All Container”*

   ⇨ Here the details of the license containers can be checked.

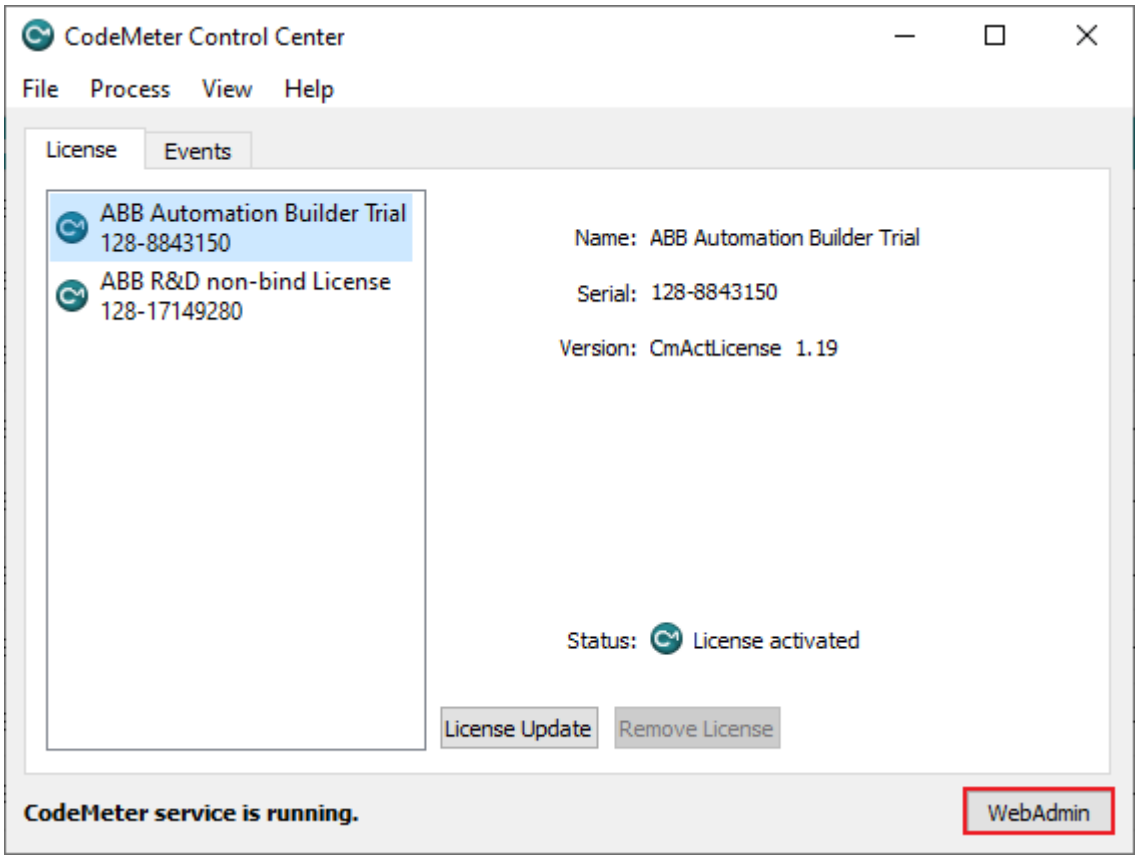


1.2.4.4 Setting dedicated network servers in search list

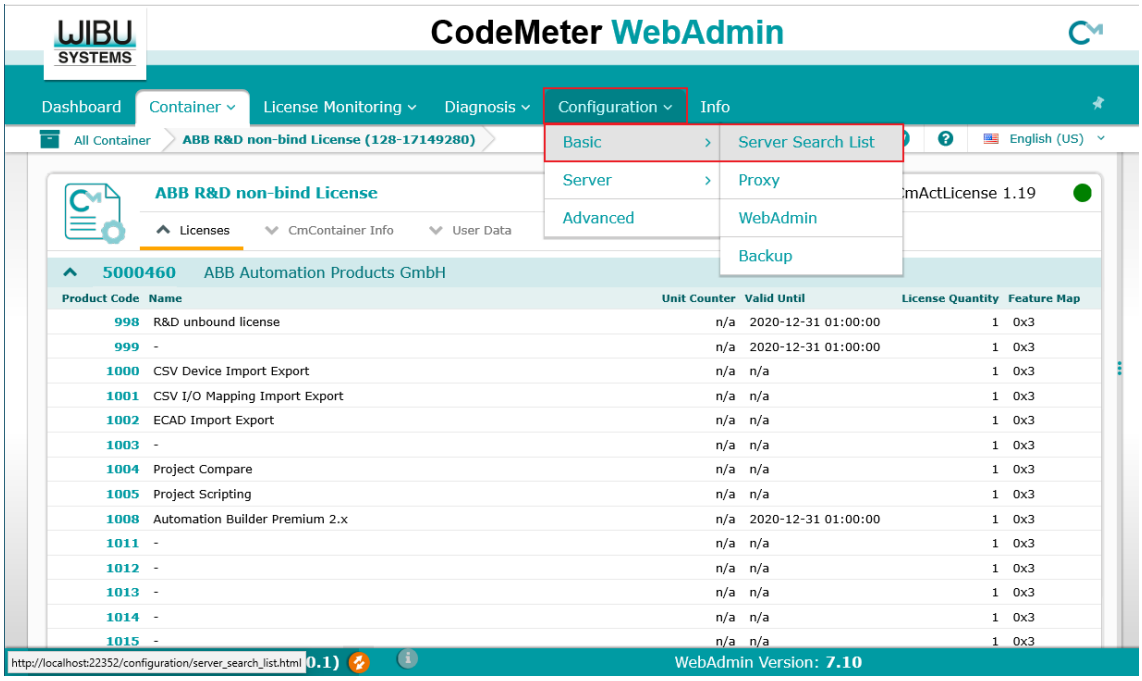
In case of a new installation CodeMeter will check for licenses also in the network. If there is a run-out or wrongly installed license found, the service is closed without any further hint. This looks like Automation Builder starts and closes after a few moments.

To set the search for licenses to your local machine only follow these steps:

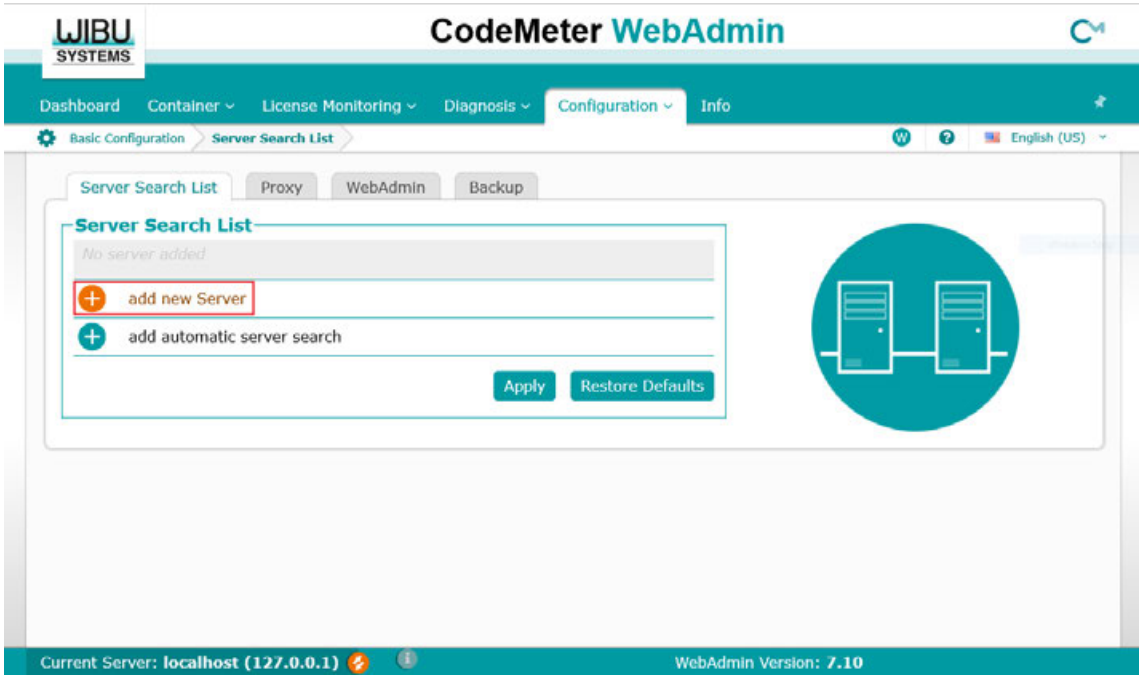
1. Open the CodeMeter Control Center. See ↳ *Chapter 1.2.4.3 “Checking licenses with CodeMeter control center” on page 22*
2. Open the CodeMeter WebAdmin by selecting *“WebAdmin”*



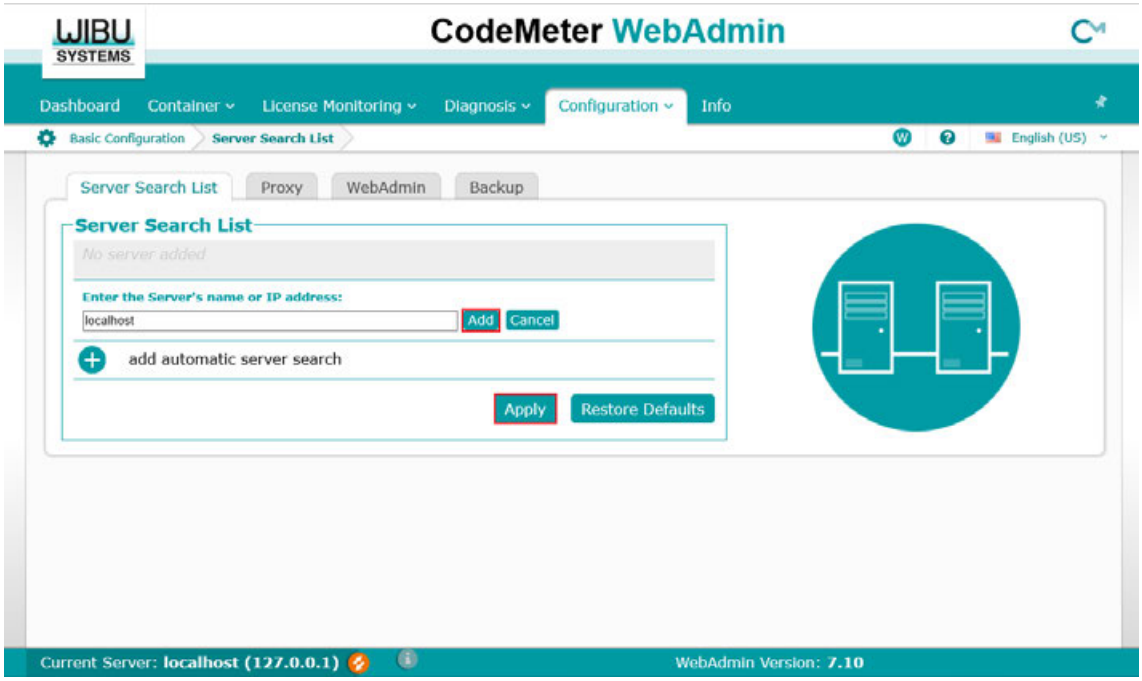
3. Select *"Configuration ➔ Basic ➔ Server Search List"*



4. Select *"add new Server"*



5. Enter "localhost" in the Server's names field
6. Select *“Add”*
7. Confirm by selecting *“Apply”*
   ⇨ The "localhost" is added to the Server Search List

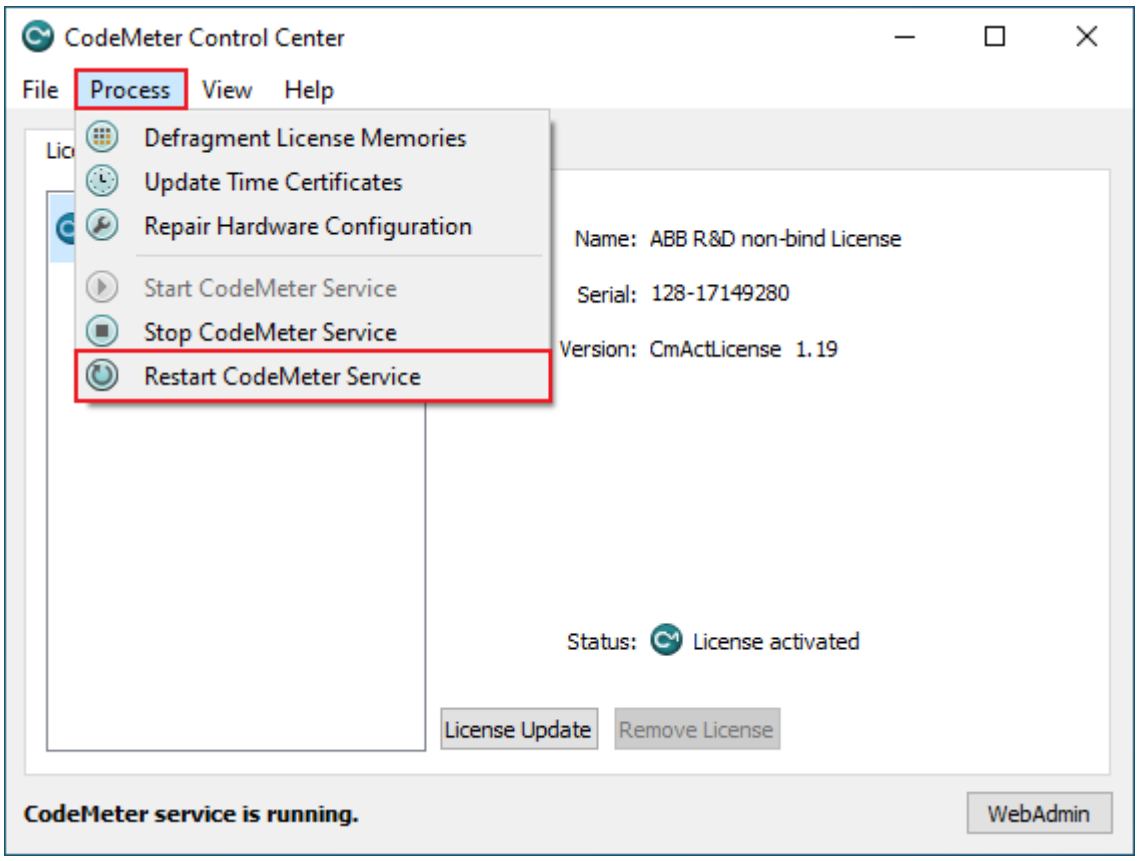


8. Restart the license check
9. Add more servers to the search list by entering the IP-Adress or name of the license servers you know.

1.2.4.5 Restarting license check with a dongle bound license

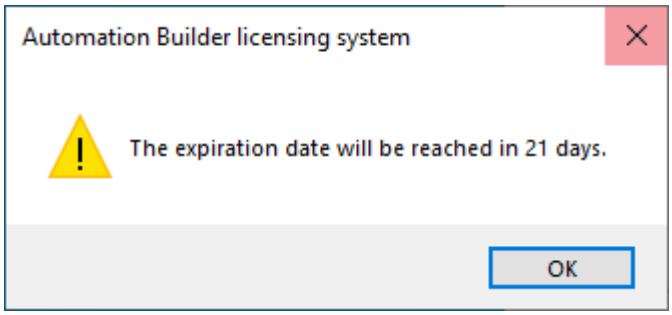
In case of using a dongle bound license it might be necessary to restart the check for license on the PC, e.g. if the dongle was removed and reinserted.

1. Open the CodeMeter Control Center. See *Chapter 1.2.4.3 “Checking licenses with CodeMeter control center” on page 22*
2. Select *“Process ➔ Restart CodeMeter Service”*



1.2.4.6 Removing trial license to remove expiring message

If an unlimited license is installed after having a trial license activated, the warning message for expiring date of the trial license still pops up at the Startup of the Automation Builder.

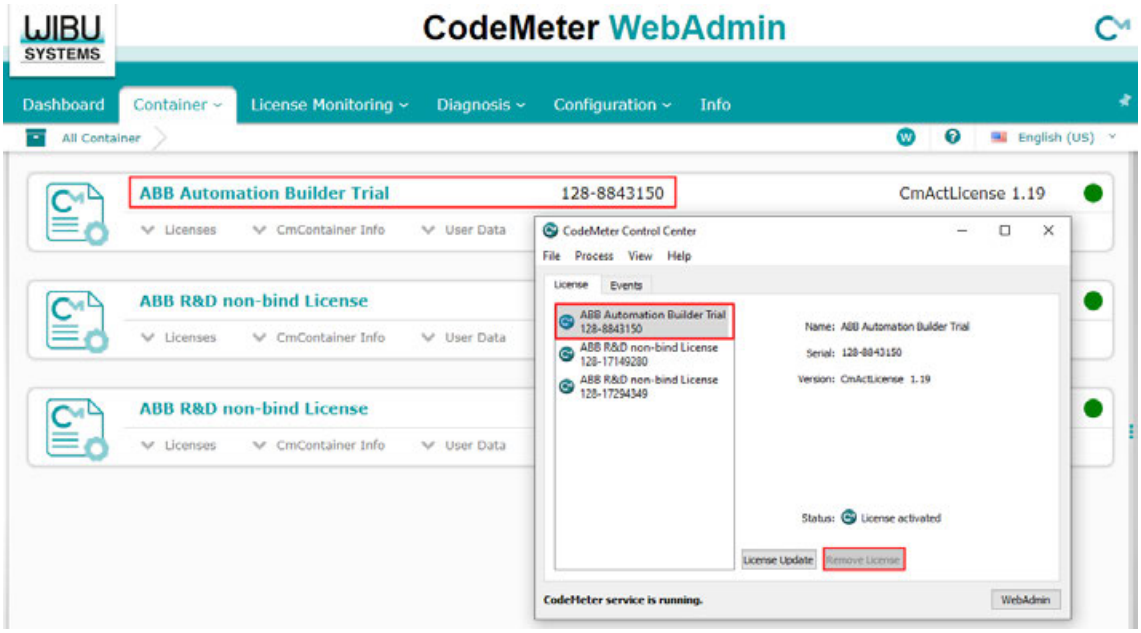


To avoid this message the trial license can be removed.

CAUTION!

- If you remove a license from your PC it will be permanently lost.
- Be aware that the trial license includes all premium functionalities, which will not be available any more if your unlimited license is not a premium license, e.g. standard.

1. Check for the Trial CmContainer number in CodeMeter WebAdmin InterfaceCheck for the Trial CmContainer number in CodeMeter WebAdmin Interface
2. Search CmContainer number in CodeMeter Control Center



3. Remove this selected license in CodeMeter Control Center

1.2.4.7 Network licenses

Starting from Automation Builder 1.2.0 network licenses can be used with Automation Builder.

This allows sharing of licenses between team members, easy switchover between several workstations with a single license and allows centralized administration (ordering, registration, activation).

The Automation Builder License Manager and CodeMeter need to be used to configure the Network server.

- *In a typical office LAN (Local Area Network) setup on Client side the default settings of the Automation Builder (and CodeMeter) are sufficient to get the Network Licenses working.*
- *If an opened Automation Builder is loosing contact to the network server (e.g. due to network problems) Automation Builder will prompt the user to restore the network. After 30 minutes without connection to the network server Automation Builder will fall back to basic edition. Opened editors for non-basic features stay open and usable. So your work will not be lost in case of troubles with the network.*

1.2.4.7.1 Setting up a network license

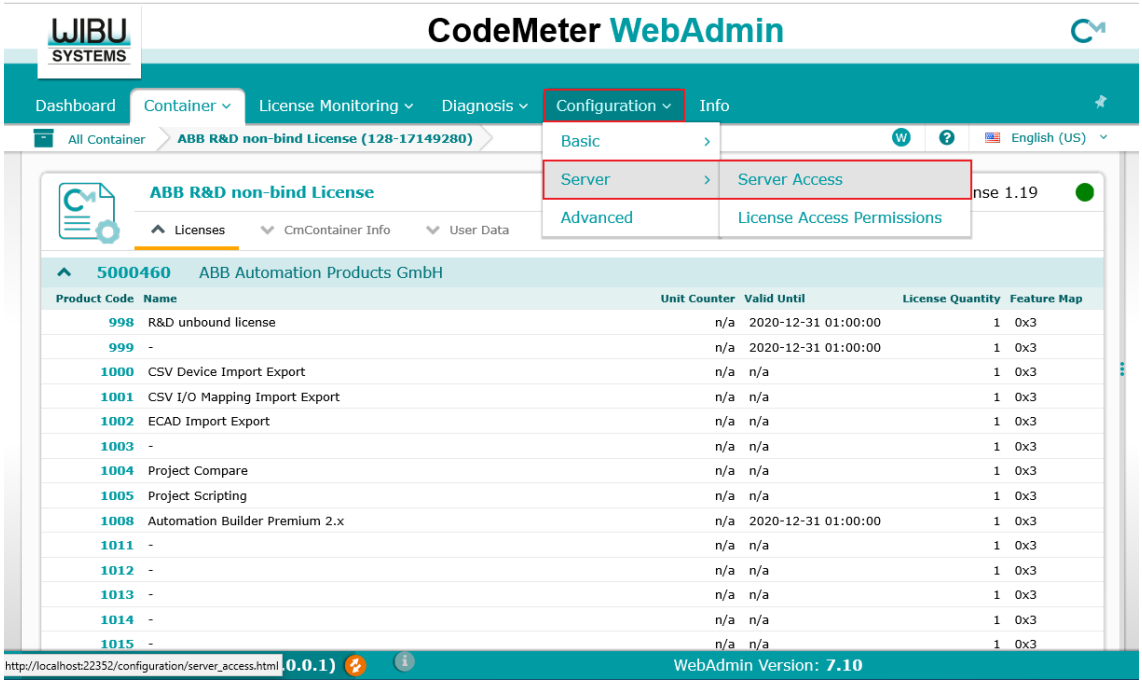
The following setup works in typical environments.

Configuring a network license server

☑ Network license must be registered.

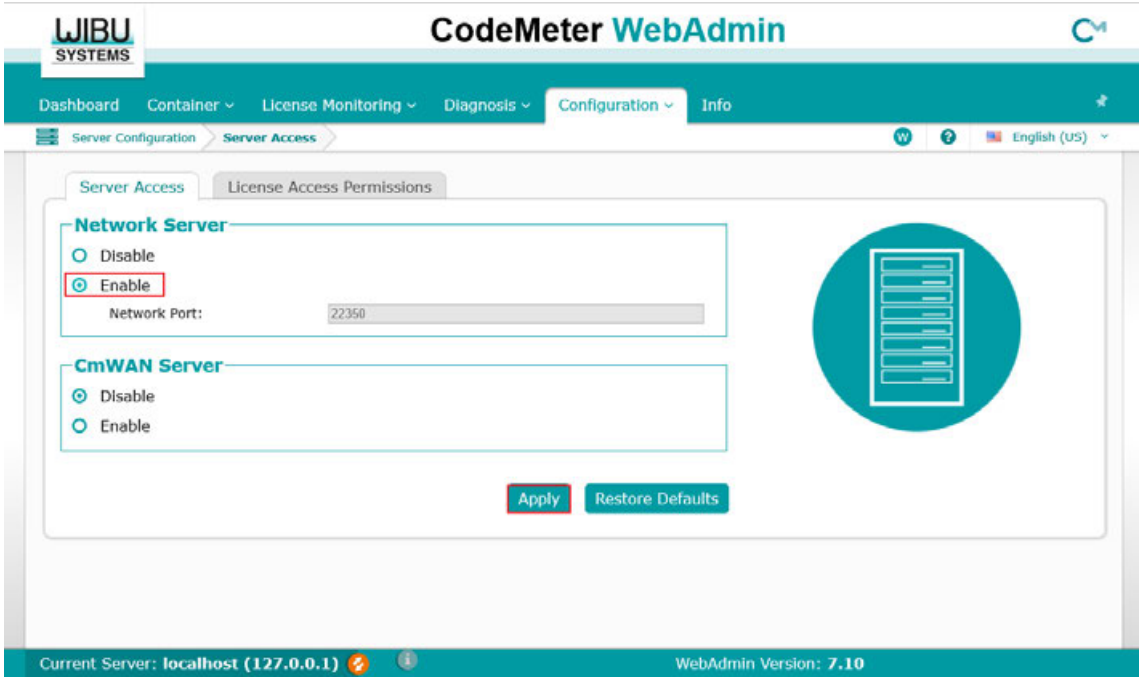
☑ Automation Builder license must be activated.

1. Launch CodeMeter WebAdmin as described in *Chapter 1.2.4 "Managing your licenses" on page 20*
2. Select *"Configuration ➔ Server ➔ Server Access"*



3. Enable Network Server

   Keep the default port settings. These should work in most cases.
4. Select *"Apply"*



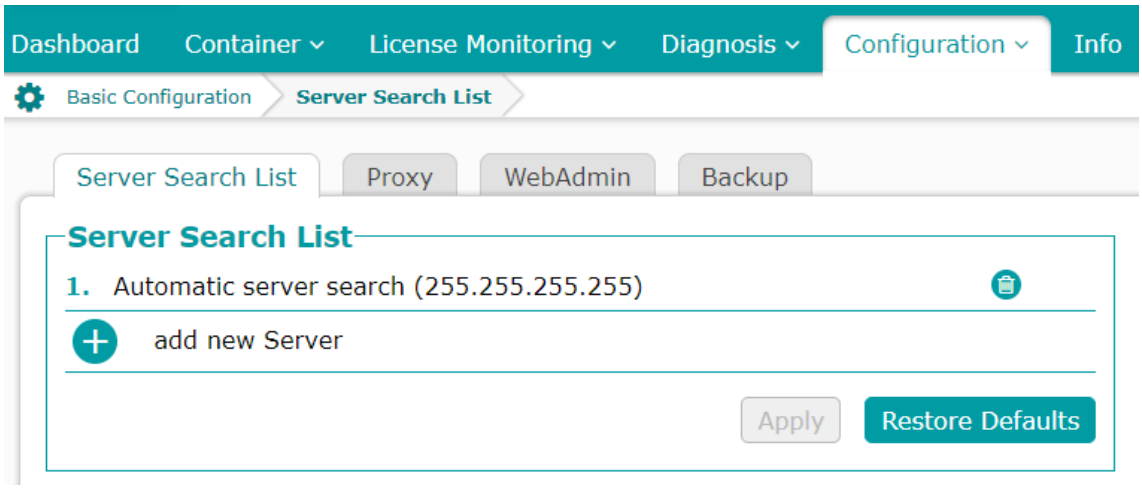
5. For the changes to take effect, restart CodeMeter Control Center see *Chapter 1.2.4.5 "Restarting license check with a dongle bound license" on page 25*

NOTICE!

– In case you want to control usage of network licenses please refer to
↳ *Chapter 1.2.4.7.3 “View network server license usage” on page 30*
– Activation keys for network licenses are valid for one network license each. This one license can be shared among many people but only one Automation Builder instance at the same time. If you want to run more than one Automation Builder instance at the same time you have to activate more than one network license. This means you have to purchase and enter more than one activation key (one per license).

Configuring the client side

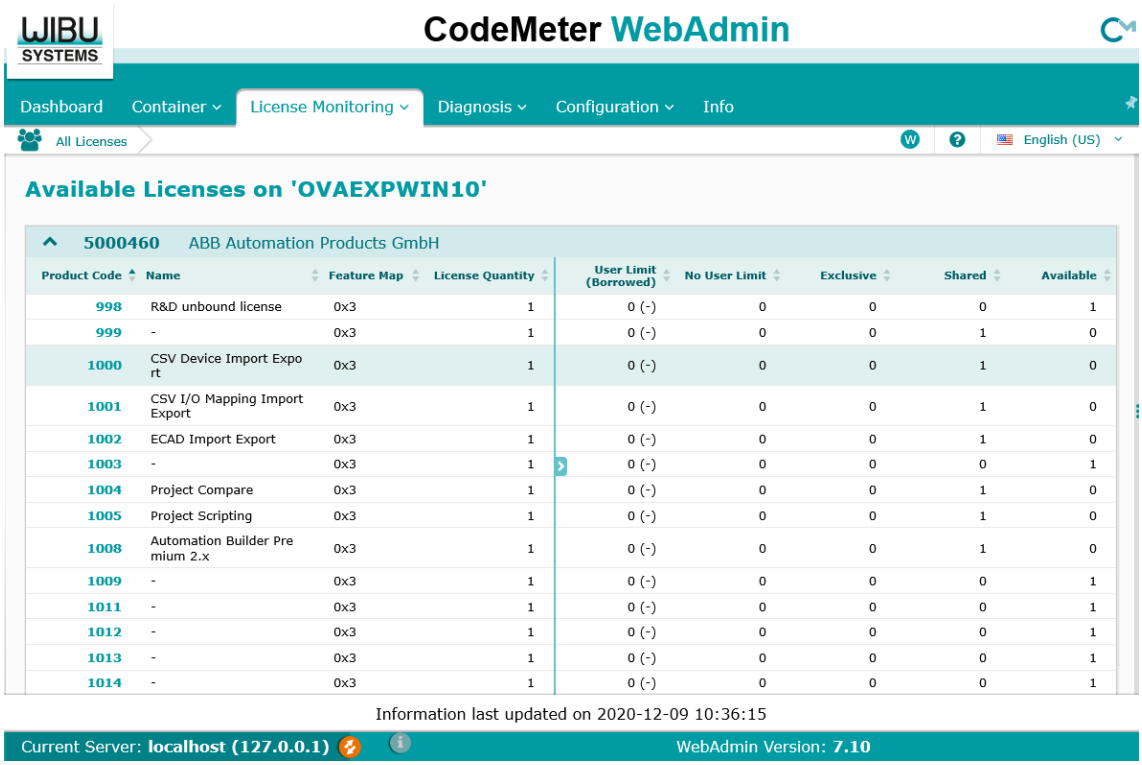
The default settings of Automation Builder and the CodeMeter (on client side) are sufficient in most cases to get the network licenses working. In case of problems accessing the network license, please set the server search list on the client side.



1.2.4.7.2 View network server licenses

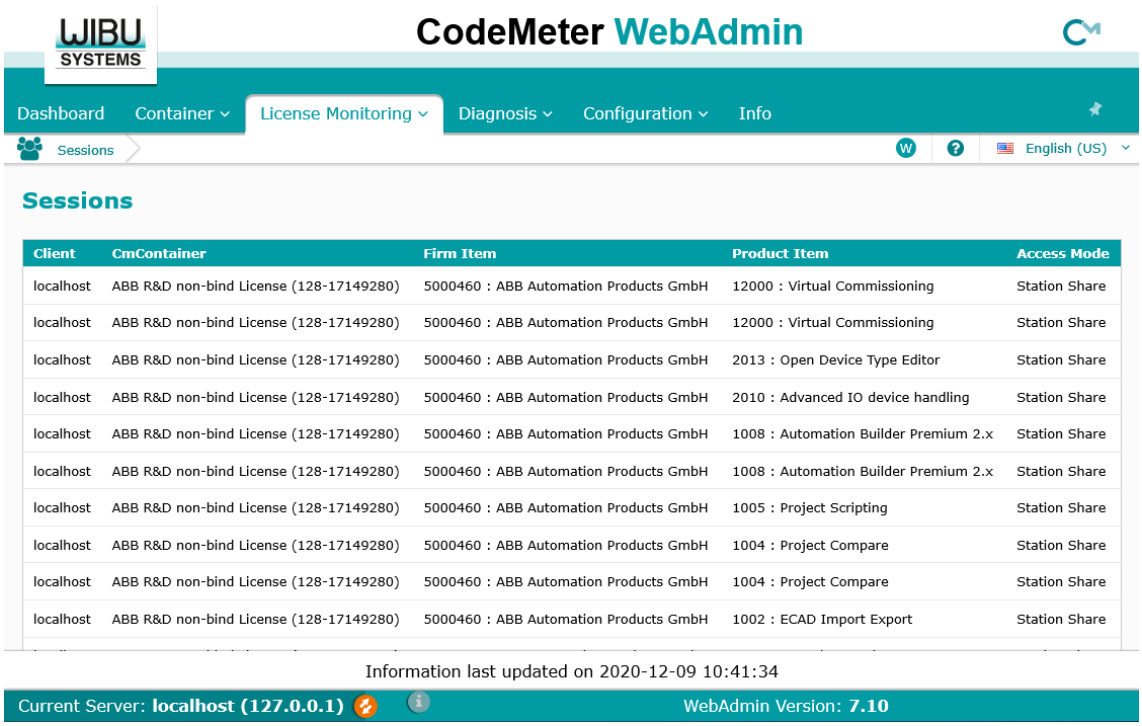
On the Network Server side you can find information on existing network licenses and their current allocation.

1. Launch CodeMeter WebAdmin. See ↳ *Chapter 1.2.4.3 "Checking licenses with CodeMeter control center" on page 22*
2. Select *"License Monitoring ➔ All Licenses "*



1.2.4.7.3 View network server license usage

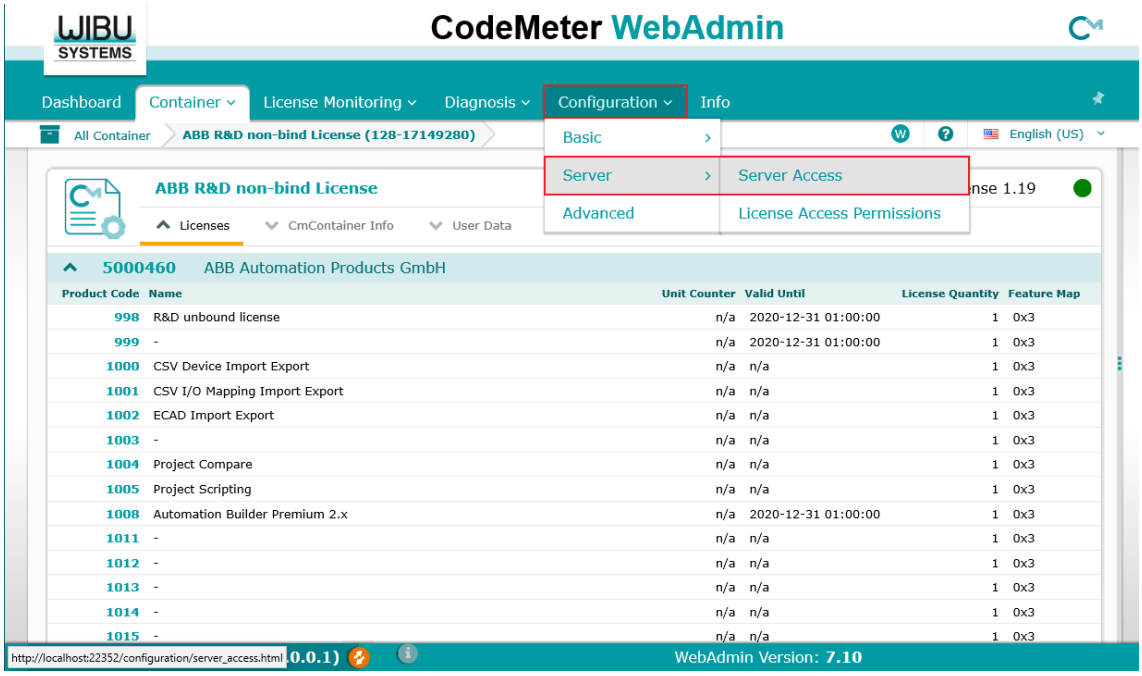
1. Launch CodeMeter WebAdmin. See ↳ *Chapter 1.2.4.3 "Checking licenses with CodeMeter control center" on page 22*
2. Select *"License Monitoring ➔ Sessions "*



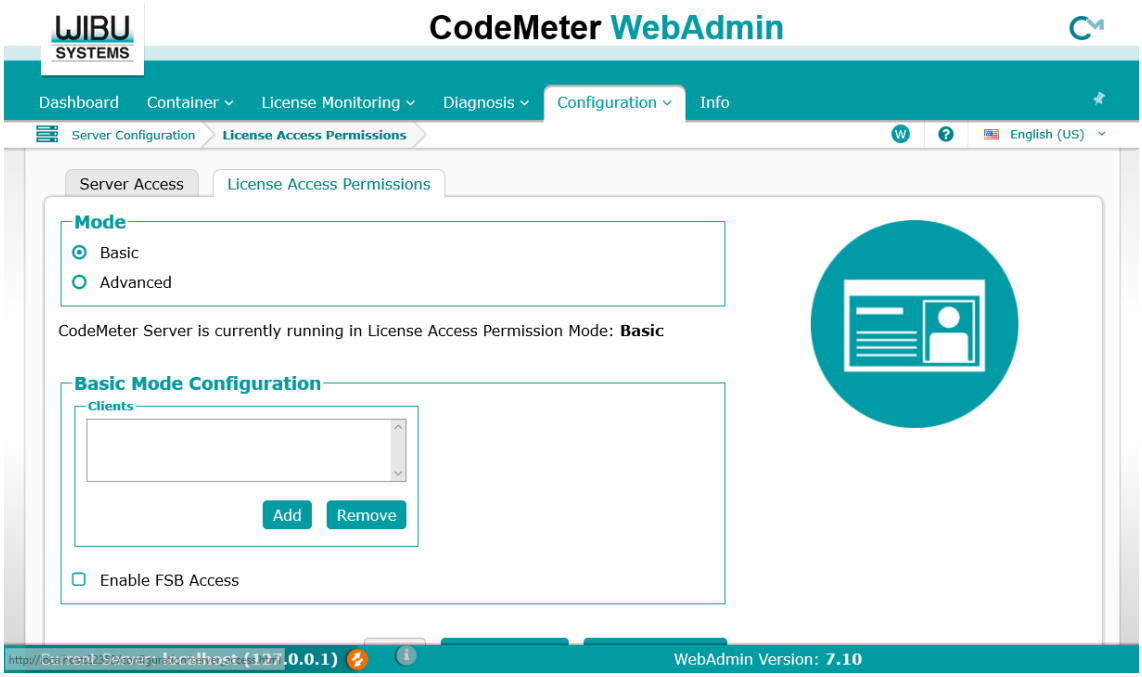
1.2.4.7.4 Controlling network server license usage

On the Network Server side you can define settings managing the client access to CodeMeter License Server on a network.

1. Launch CodeMeter WebAdmin. See *Chapter 1.2.4.3 "Checking licenses with CodeMeter control center" on page 22*
2. Select *"Configuration ➔ Server ➔ Server Access"*



3. Add entries of PCs you want to share licenses with by adding client computers and IP addresses for accessing CodeMeter License Server on a network.



1.2.4.8 License borrowing manager

The license borrowing manager allows you, to borrow a network license for offline use and return it.

> *The license borrowing manager is not part of the default software distribution, but will be handed out on request.*

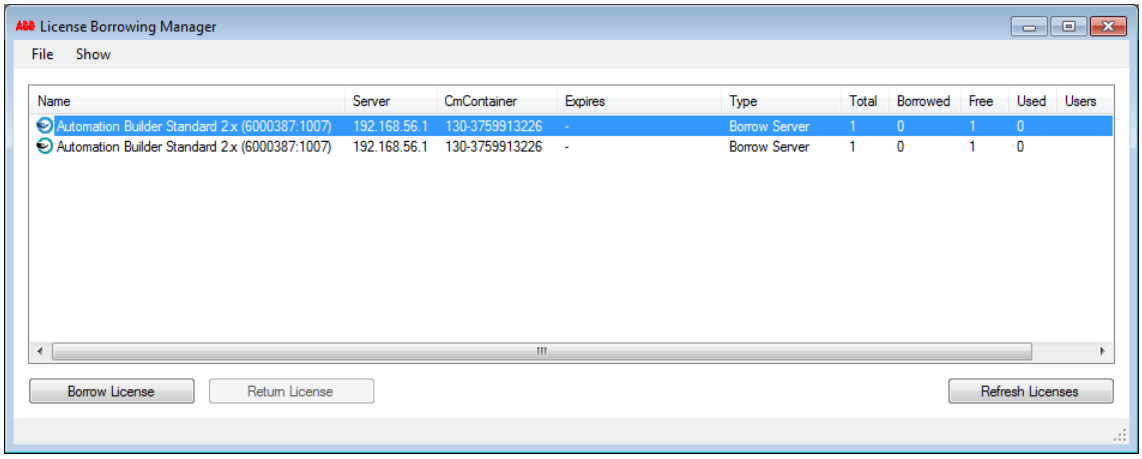
> *The license borrowing manager is only supported by Automation Builder 2.2.3 and later.*

1.2.4.8.1 Borrowing a network license

☑ Network access to the license server required.

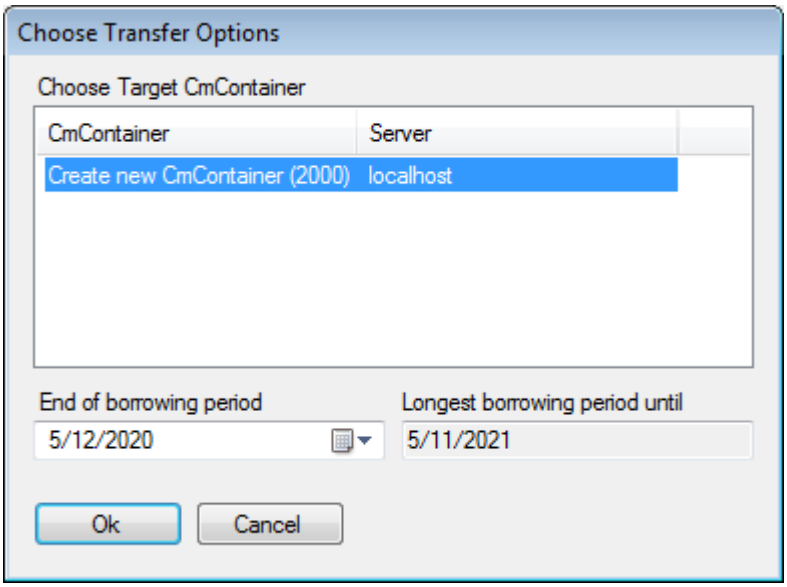
☑ Opened the license borrowing manager.

1. Select the license you want to borrow.



2. Select *“Borrow License”*.
3. Select the taget CmContainer.

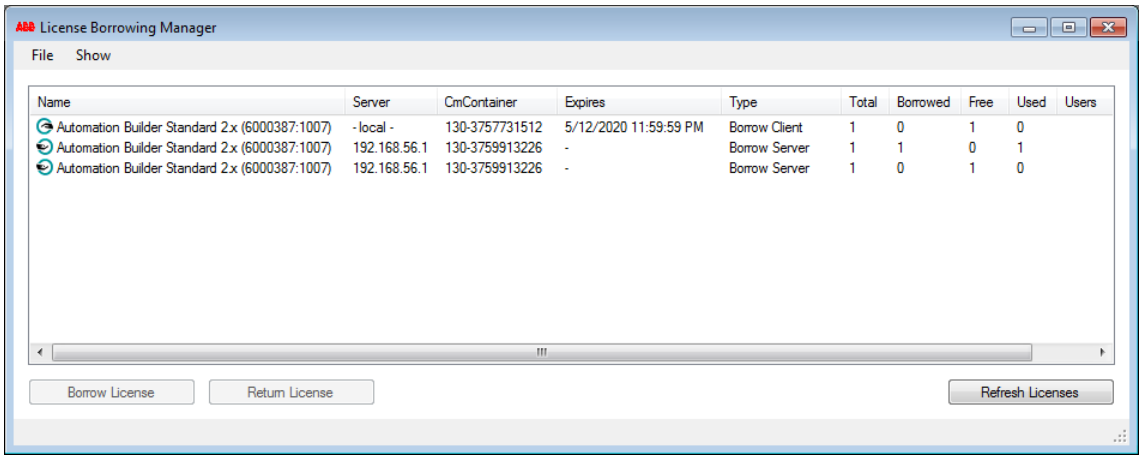
   Alternatively a new CmContainer can be created.
4. Select the end of the borrowing period.



5. Select *“OK”*.

   ⇨ The license has sucessfully been borrowed.

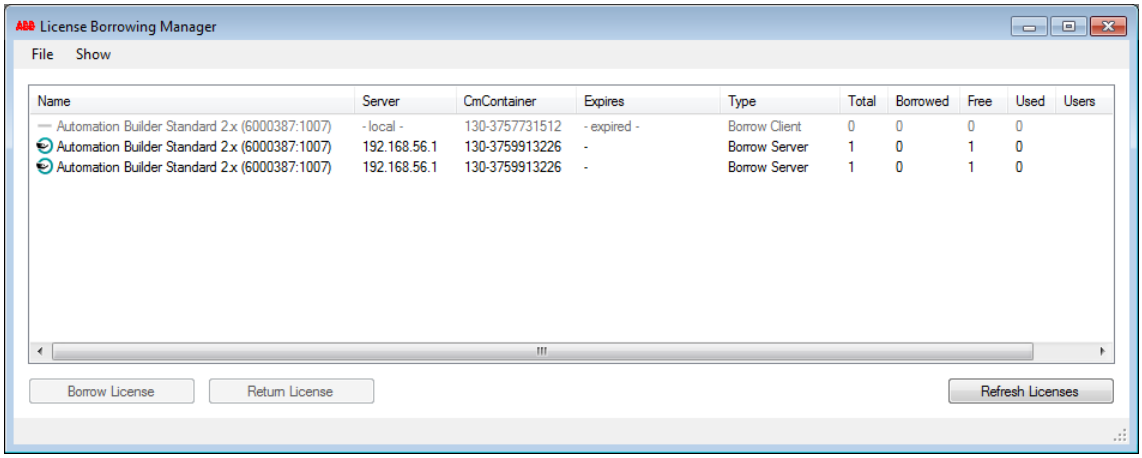
   The list of available licenses has been updated.



1.2.4.8.2 Returning a network license

Automatical return of a license

Network licenses will be returned automatically after the expiration of the maximum borrwoing period. No licenses server access is required.



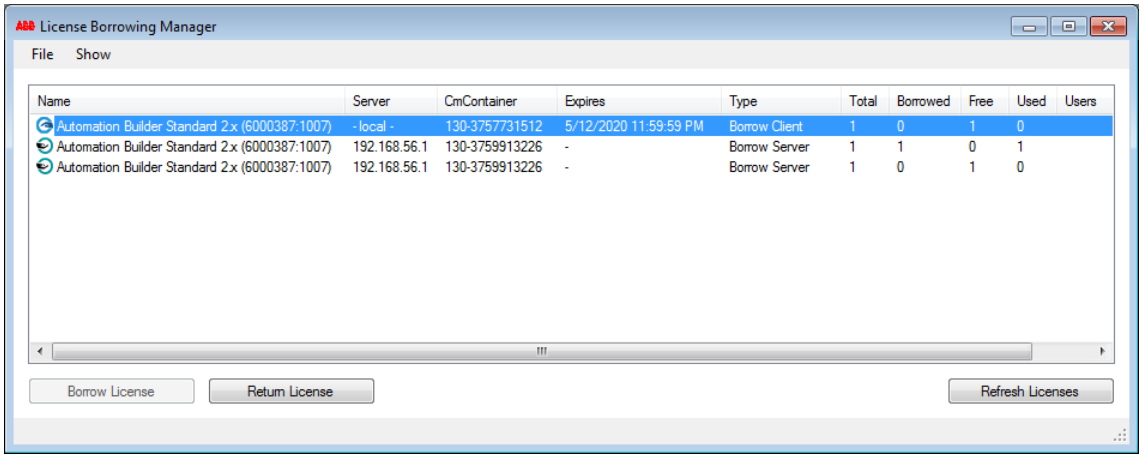
Manual return of a network license

Network licenses can be returned anytime manually.

☑ Network access to the license server required.

☑ Opened the license borrowing manager.

1. Select a borrowed license.



2. Select *“Return License”*

   ⇨ The license has sucessfully been returned.

1.2.4.9 Transfering an Automation Builder license

1.2.4.9.1 General

It is possible to transfer normal licenses from a PC to another PC or dongle (DM-Key).

This is not possible for ABB internal or temporary licenses, e.g. the 30 day Trial license.

The process consists of two main steps:

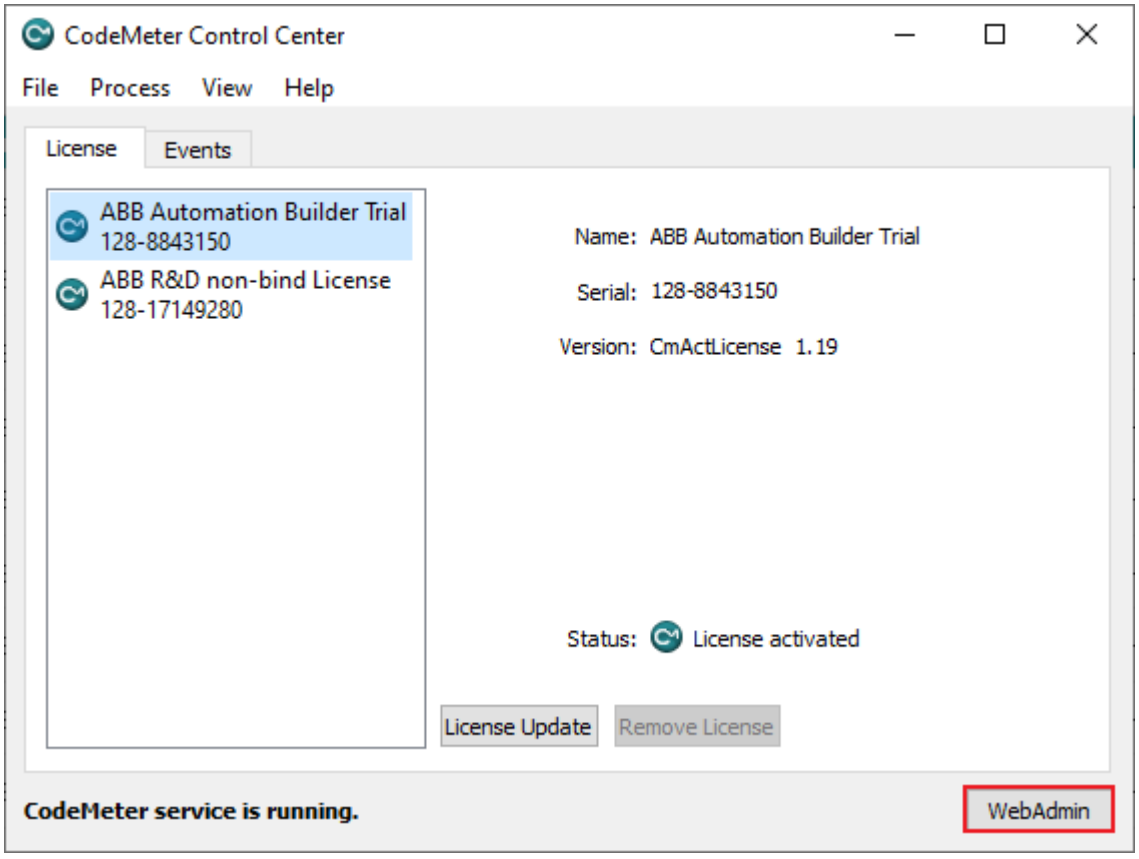
1. Return the actual license from the actual PC
2. Reactivate the license on the new PC

1.2.4.9.2 Getting activation code

For all license transfer processes the activation code is required. It is available from the license paper from purchasing the license.

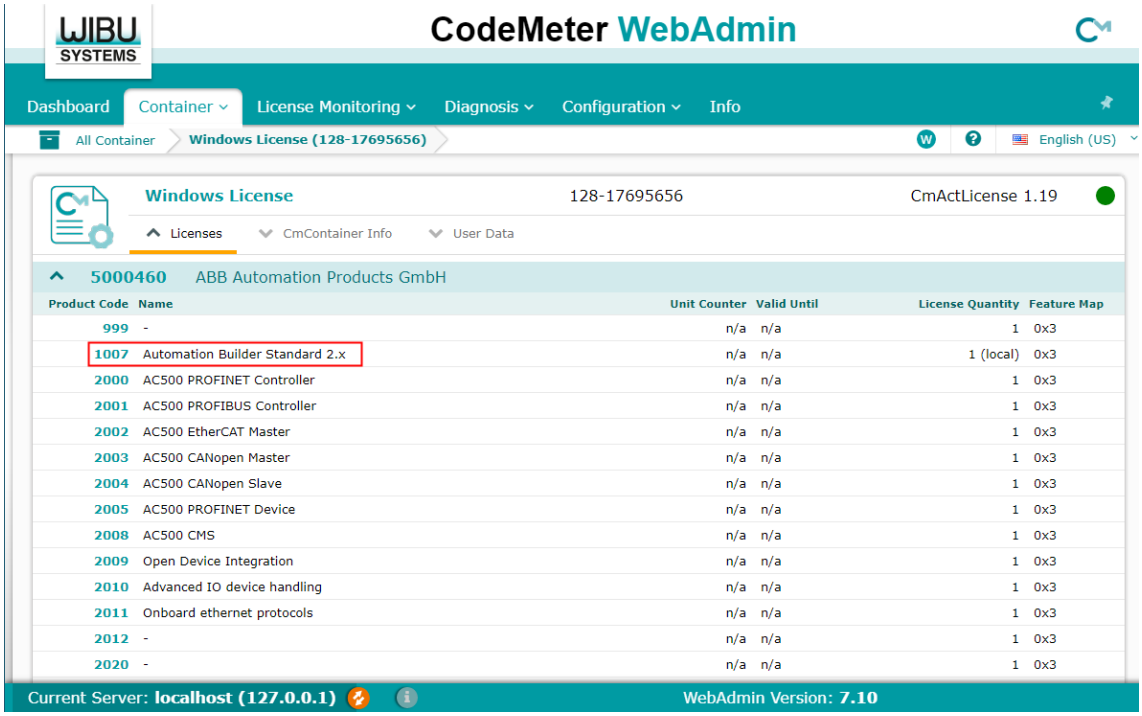
For Automation Builder licenses purchased April 2020 or later, the activation code is available from the activated license:

1. Open CodeMeter Control Center and navigate to the *"WebAdmin"*.

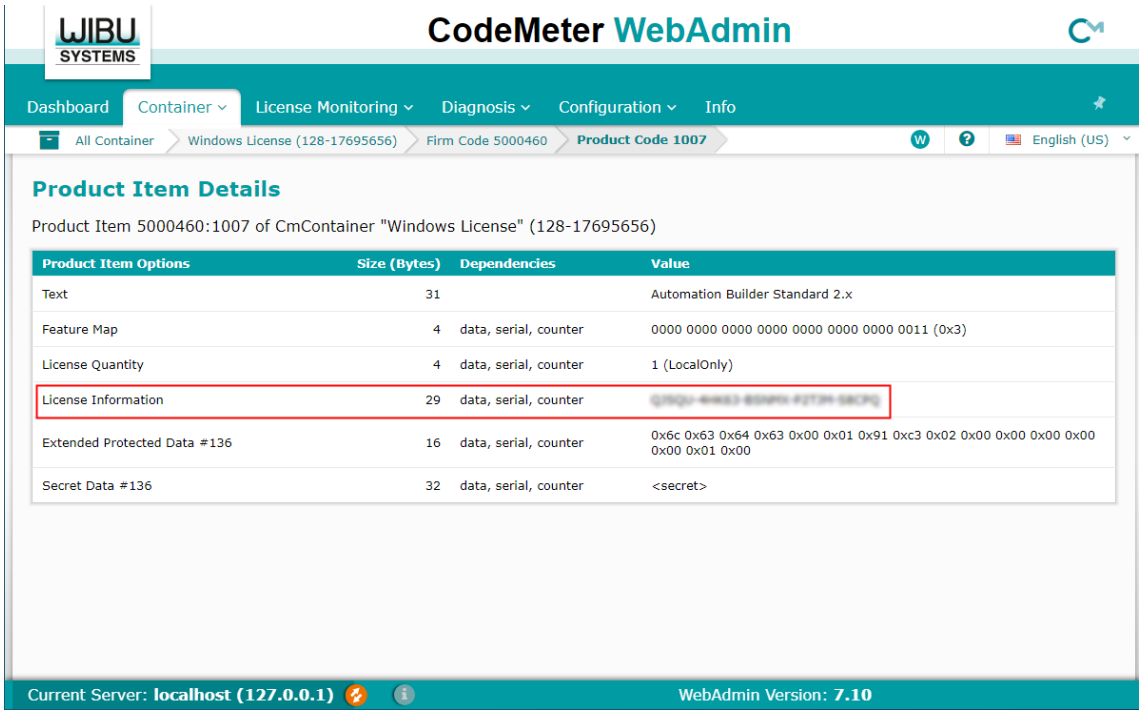


2. Identify the right product code.

   Automation Builder editions consist of multiple product codes. The activation ID is available from the product code containing the edition name, e.g. "Automation Builder Standard".



3. Select product code to access the product code details. Under *"License Information"* you can find the activation code.



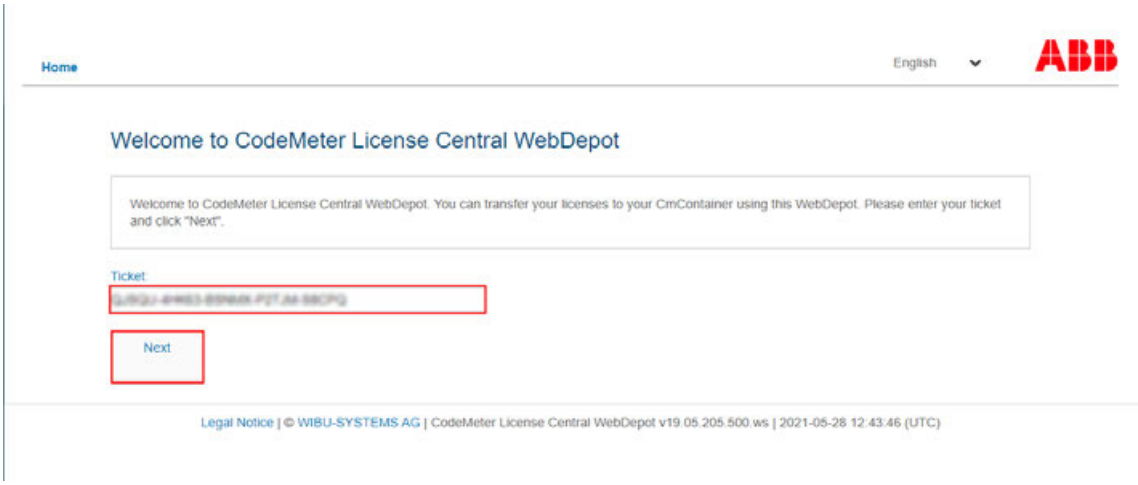
1.2.4.9.3 Returning an Automation Builder license

☑ You need the License Activation code of the license you want to return.

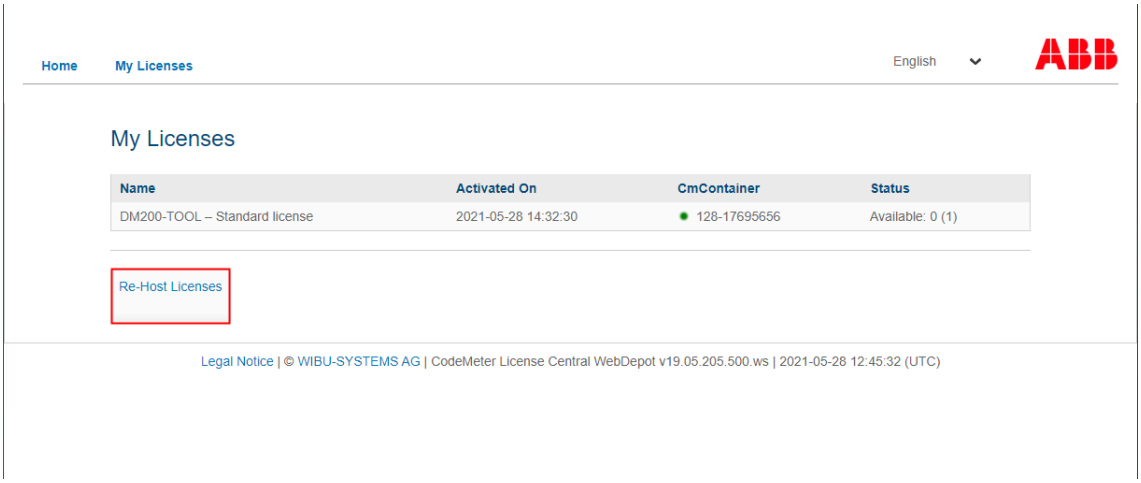
1. Go to the following website: *http://lc.codemeter.com/32838/depot-return/index.php*

> *The website is also availaible through the Automation Builder menu under "Help ➔ Return of Automation Builder license".*

2. Insert your Activation code in the field "Ticket"
3. Select *"Next"*



4. Select *"Re-Host License"*

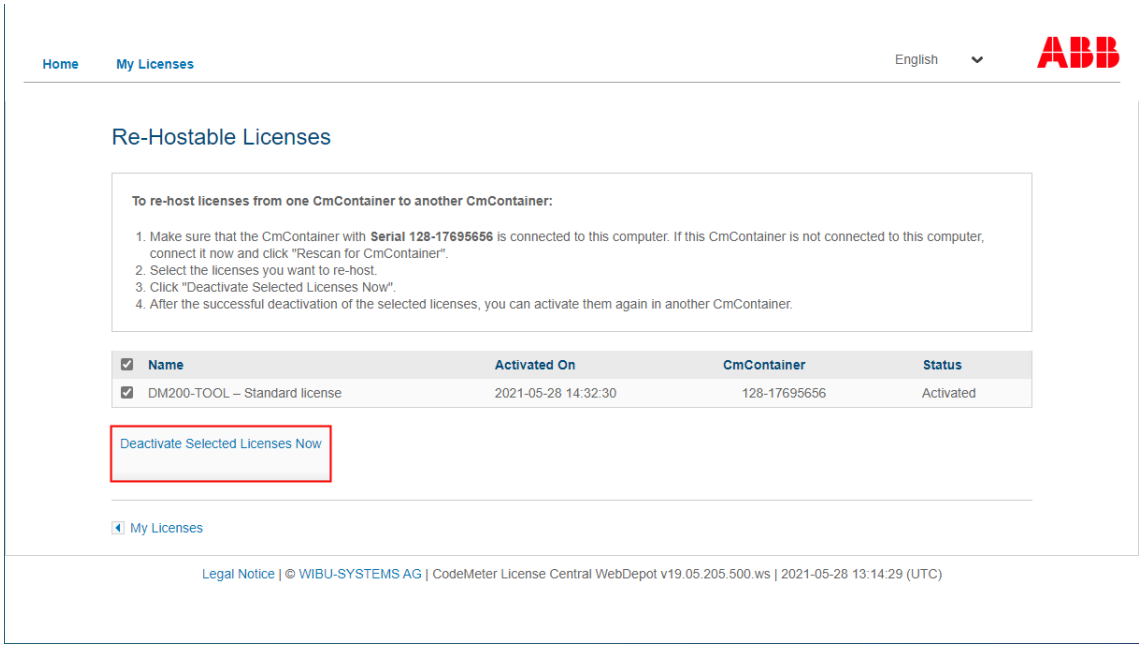


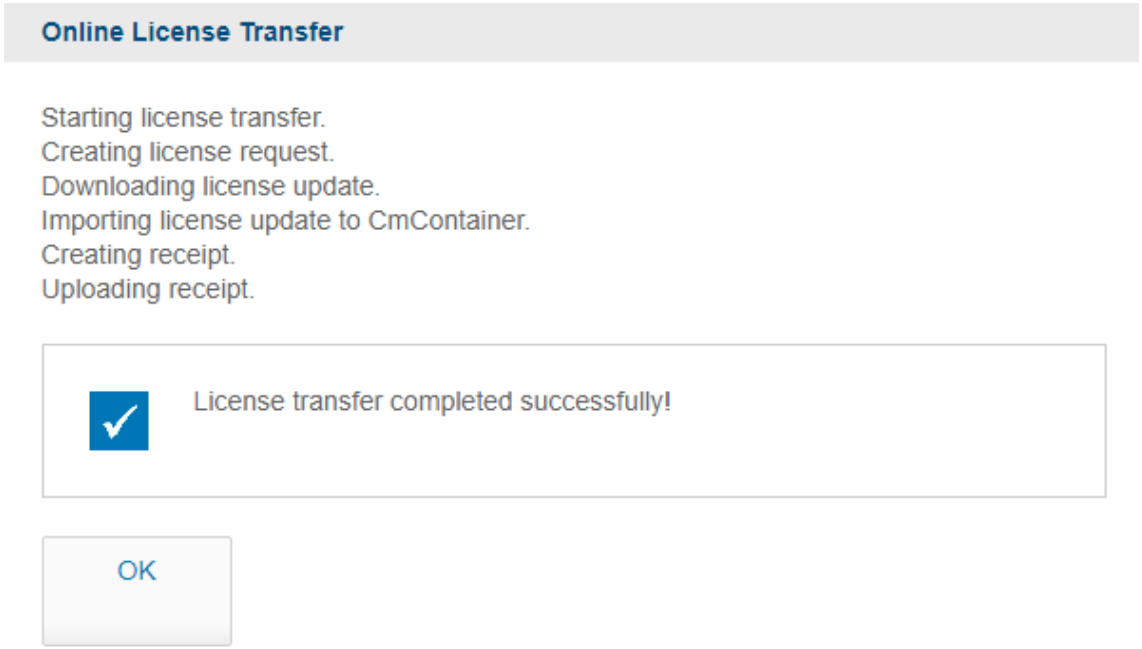
⇨ If the CmContainer is found, continue with Online licenses transfer ↯ *Chapter 1.2.4.9.3.1 "Online license transfer" on page 37*

⇨ If the CmContainer is not found, continue with Offline license transfer ↯ *Chapter 1.2.4.9.3.2 "Offline license transfer" on page 39*

Online license transfer

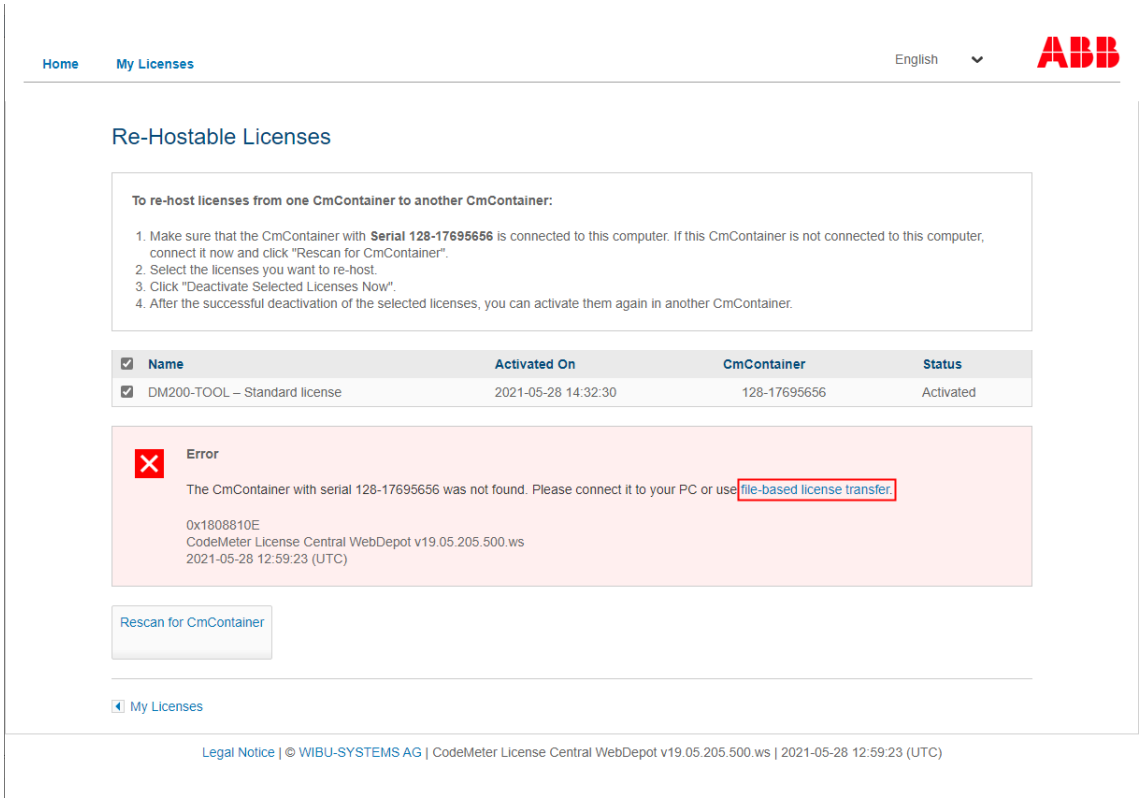
▷ Wait till the CmContainer is found, then select *"Deactivate Selected License Now"*


Home
My Licenses
English
Re-Hostable Licenses
To re-host licenses from one CmContainer to another CmContainer:
1. Make sure that the CmContainer with Serial 128-17695656 is connected to this computer. If this CmContainer is not connected to this computer, connect it now and click "Rescan for CmContainer".
2. Select the licenses you want to re-host.
3. Click "Deactivate Selected Licenses Now".
4. After the successful deactivation of the selected licenses, you can activate them again in another CmContainer.
Name
Activated On
CmContainer
Status
DM200-TOOL – Standard license
2021-05-28 14:32:30
128-17695656
Activated
Deactivate Selected Licenses Now
My Licenses
Legal Notice | © WIBU-SYSTEMS AG | CodeMeter License Central WebDepot v19.05.205.500.ws | 2021-05-28 13:14:29 (UTC)

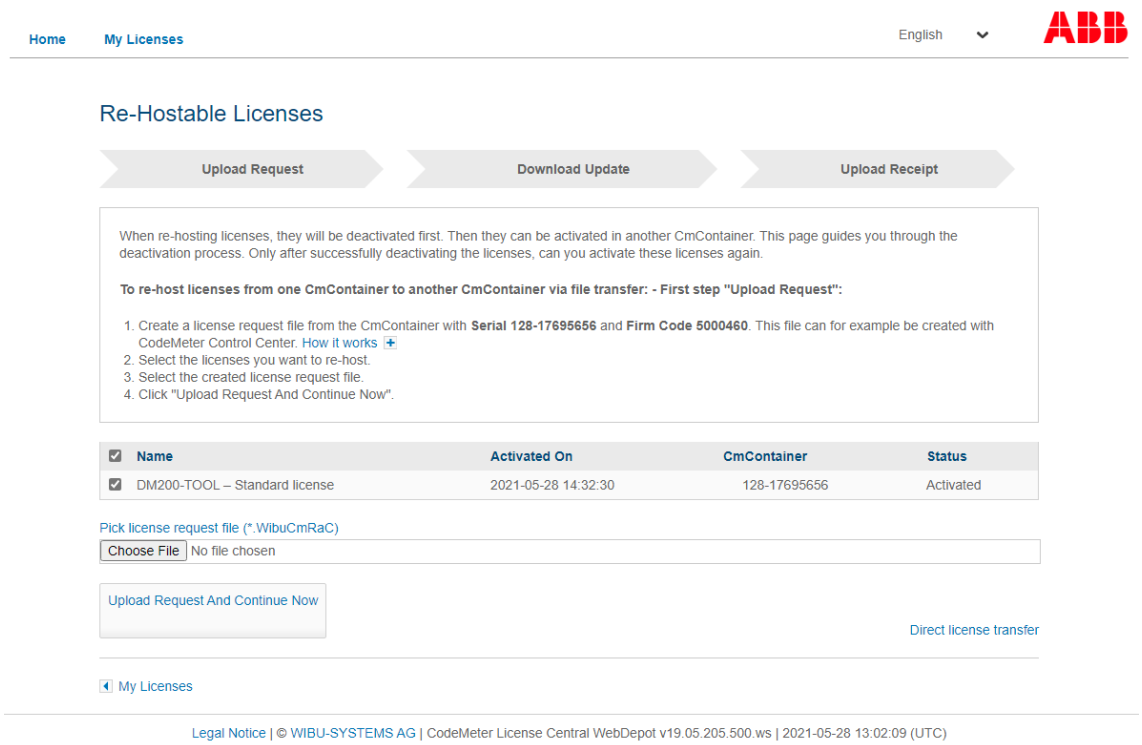

Online License Transfer
Starting license transfer.
Creating license request.
Downloading license update.
Importing license update to CmContainer.
Creating receipt.
Uploading receipt.
License transfer completed successfully!
OK

Offline license transfer

▷ If the CmContainer is not found on this PC, select file-based license transfer workflow.

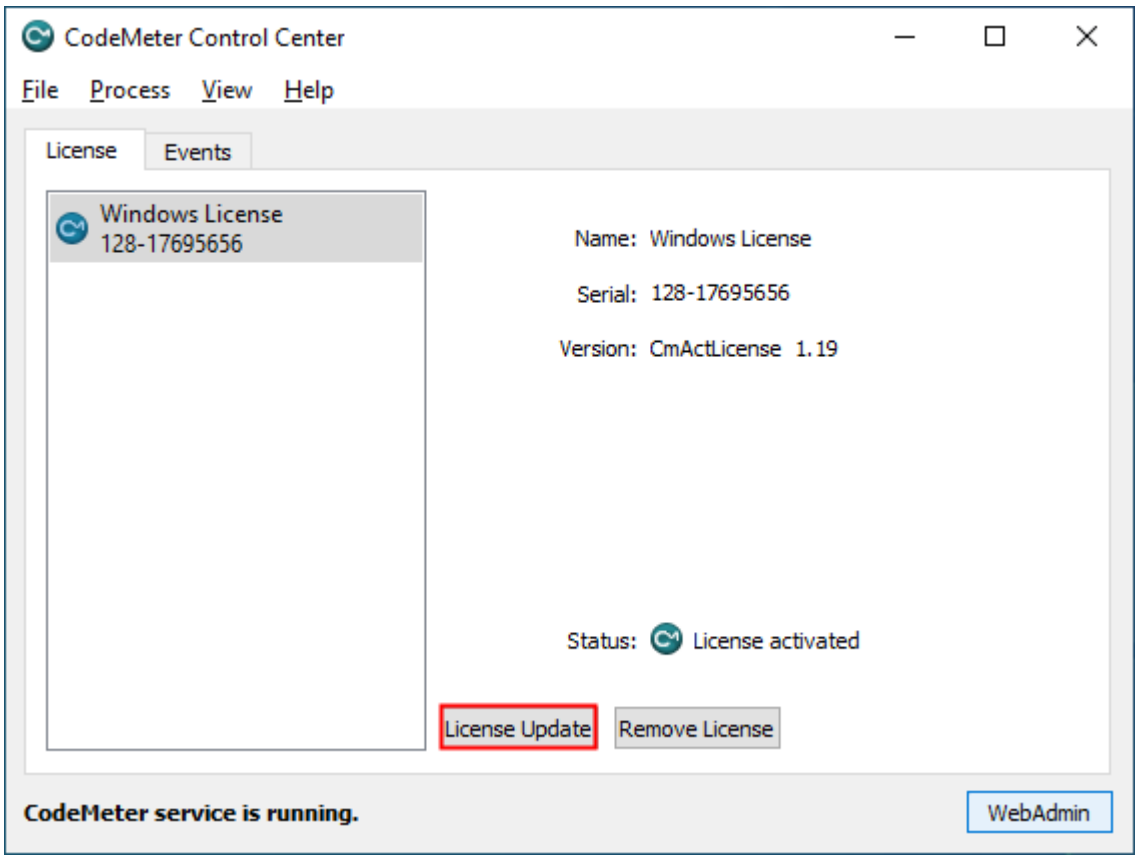


⇨ The following dialog opens

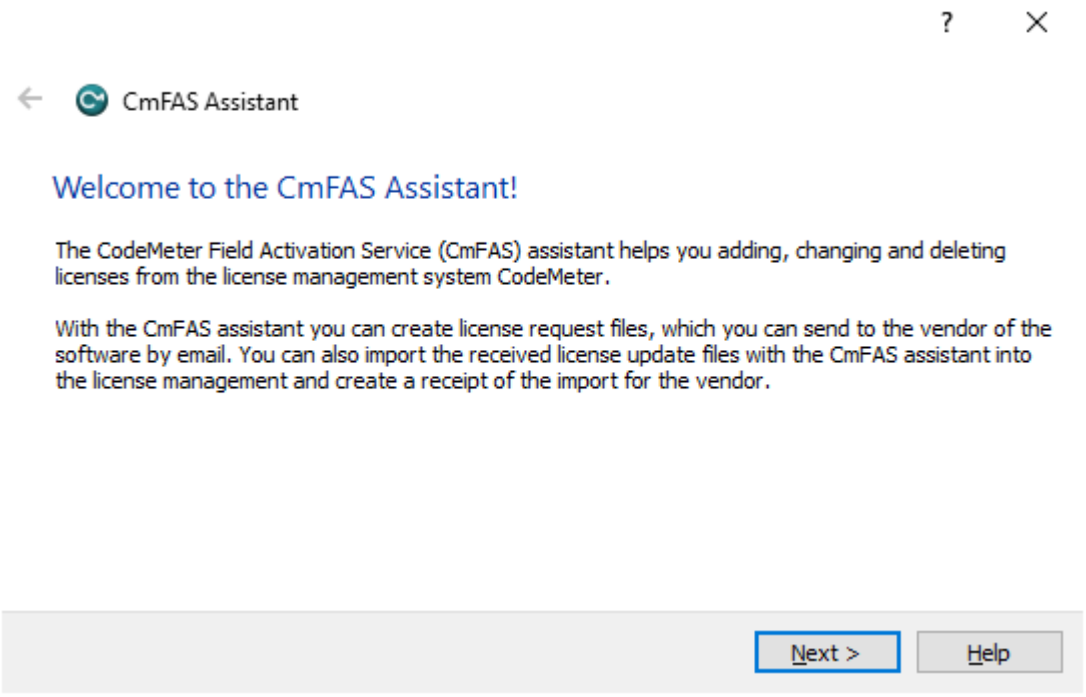


The instructions will lead you through the main steps of the offline license transfer:

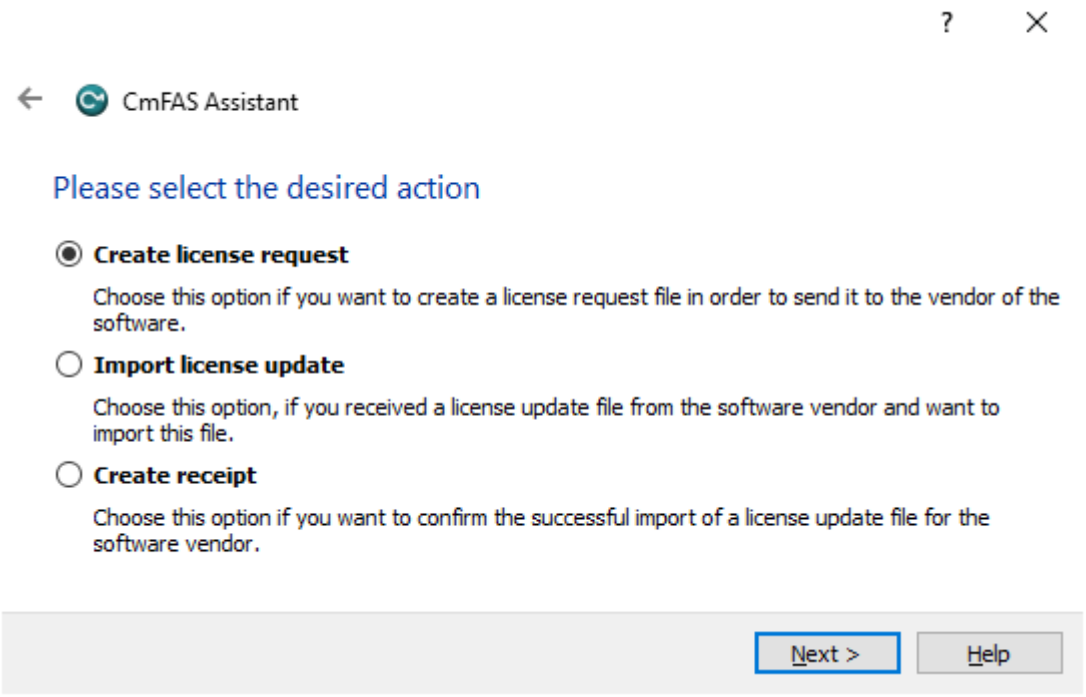
1. On the offline PC open the CodeMeter Control Center.
2. Select *“License Update”*.



⇨ The CmFAS Assistant opens.

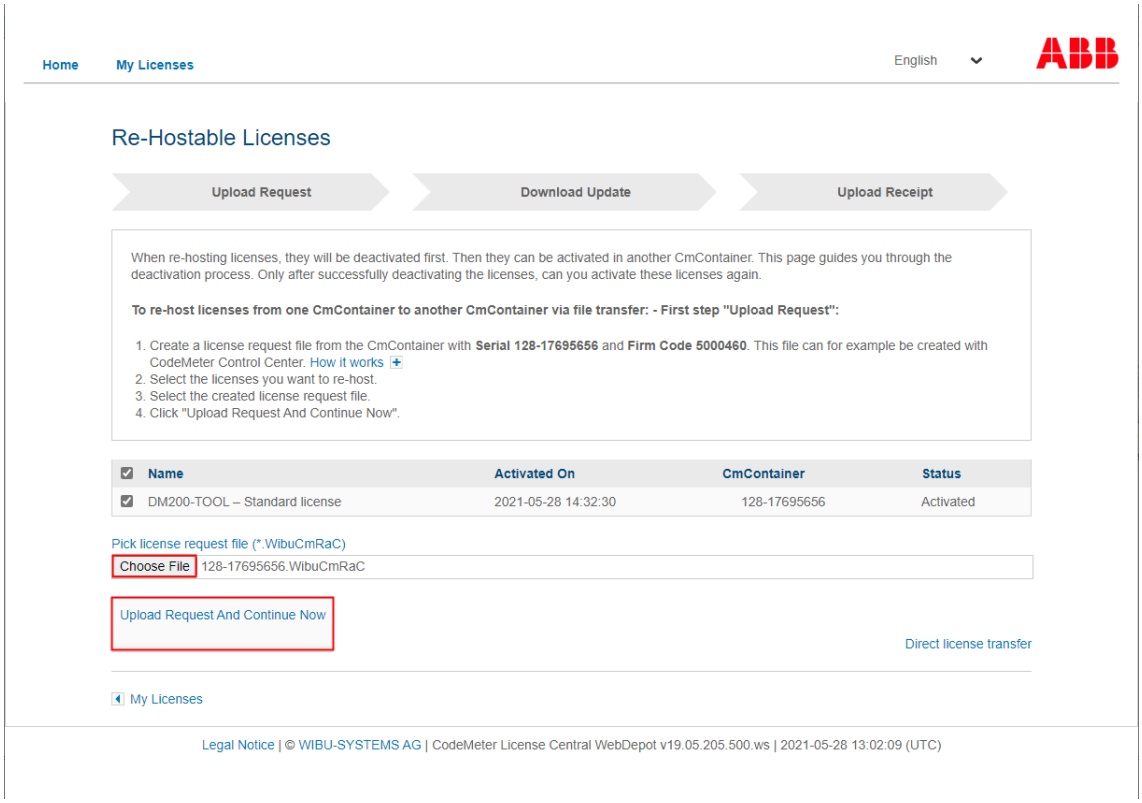


3. Select *“Create a license request file”*.

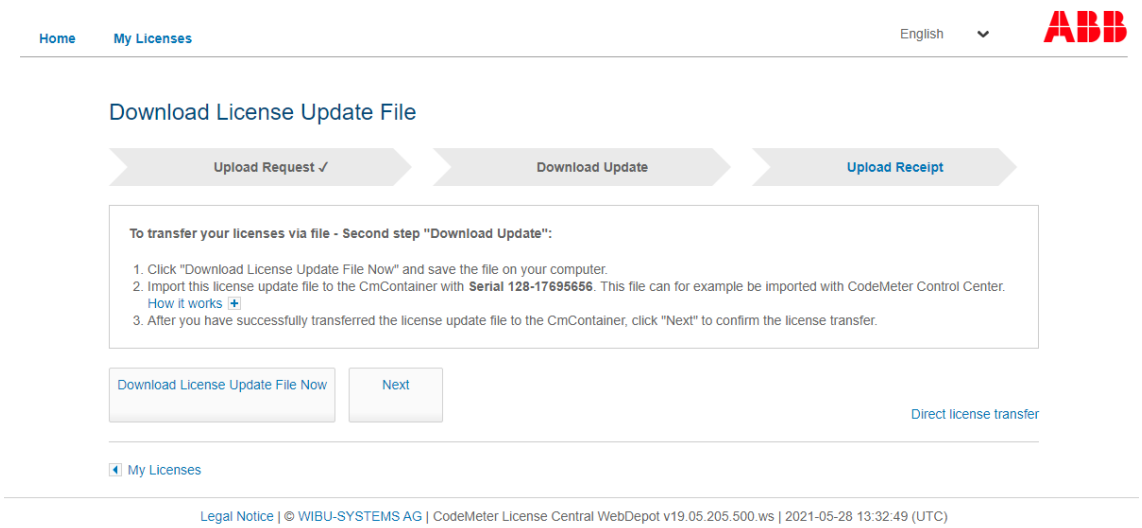


4. Select a location to store the license request file.
5. Transfer the license request file from the offline PC to an online PC.

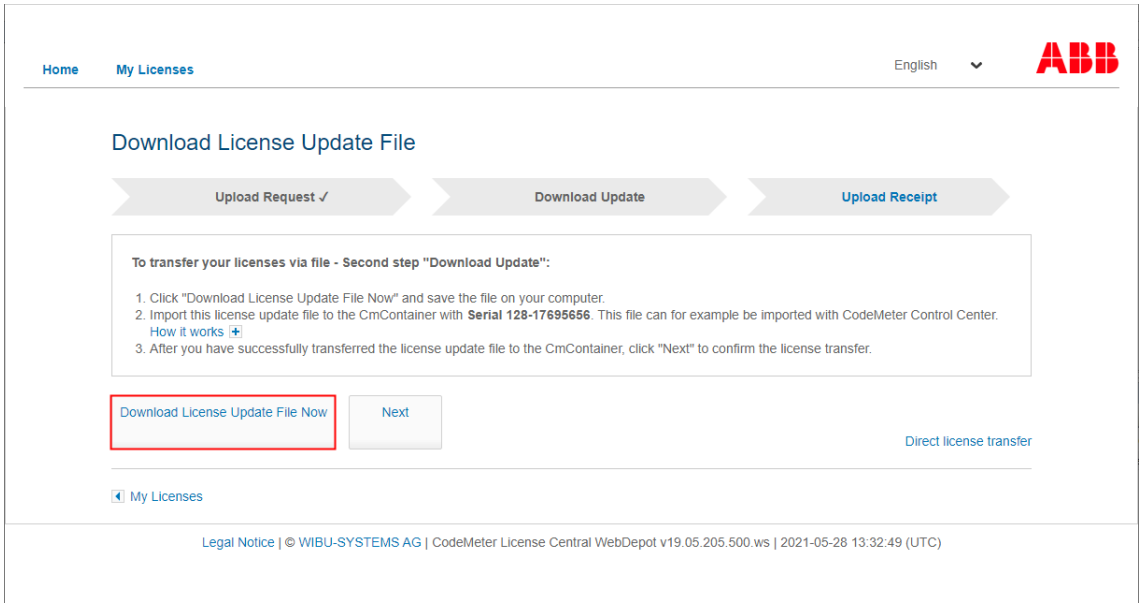
6. On the online PC choose the license request file and select *“Upload Request And Continue Now”*.



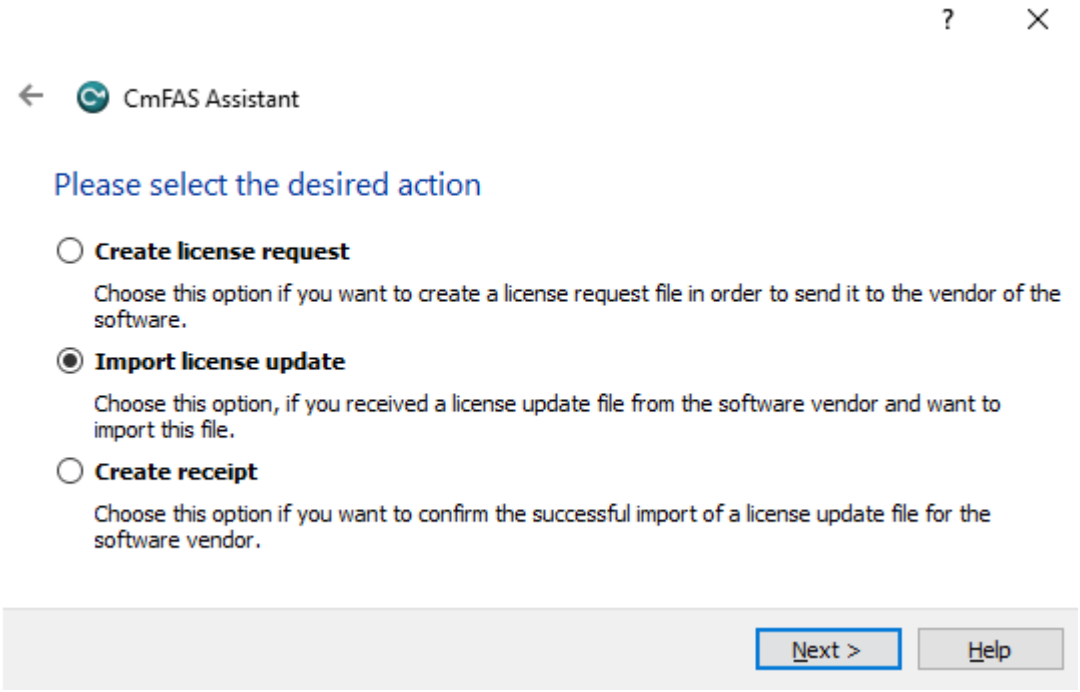
⇨ The next dialog is opened



7. Select *“Download License Update File Now”*.

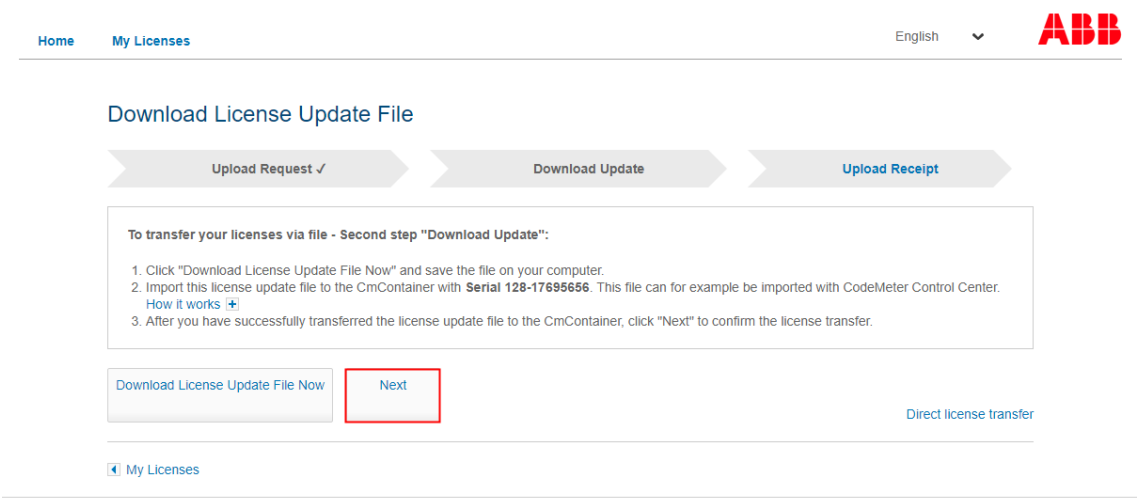


8. Save the license update file to a location on your computer.
9. Transfer the license upate file from the online PC to the offline PC.
10. On the offline PC open the CmFAS Assistant.
11. Select *“Import license update”*.

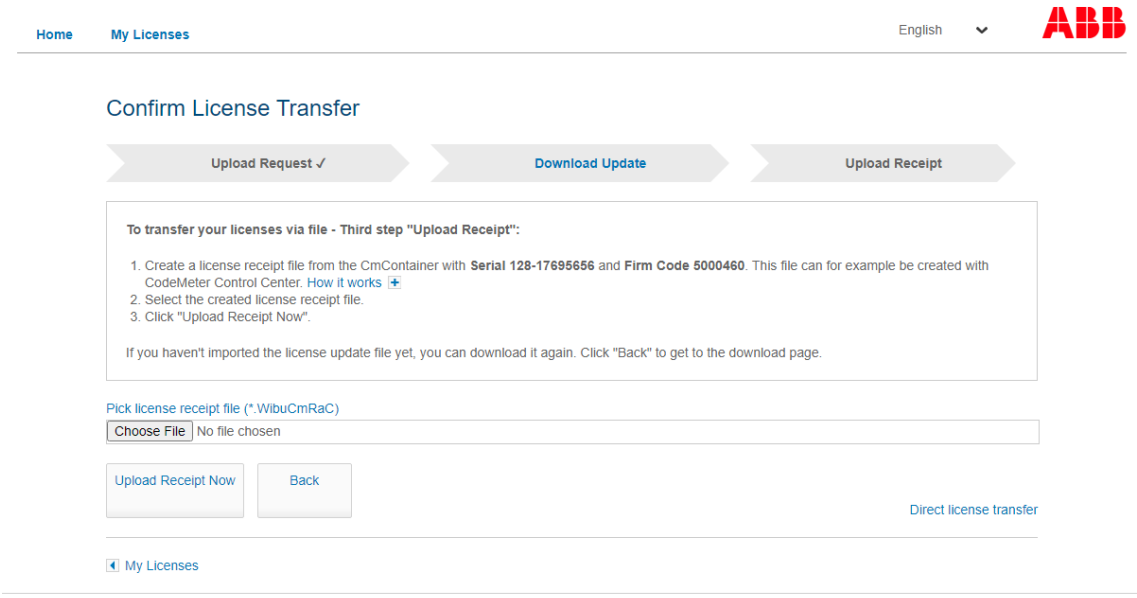


12. Select the license update file, to import the new license to the offline PC

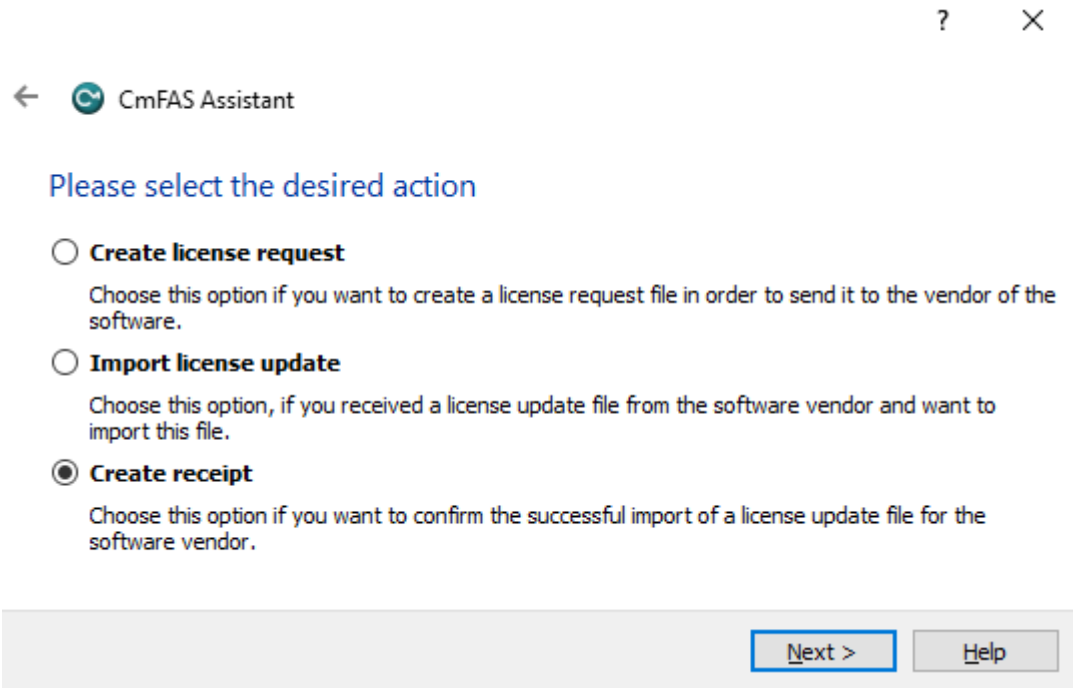
13. To confirm a succesful license transfer return to the online PC and select *“Next”*.



⇨ The last dialog is opened

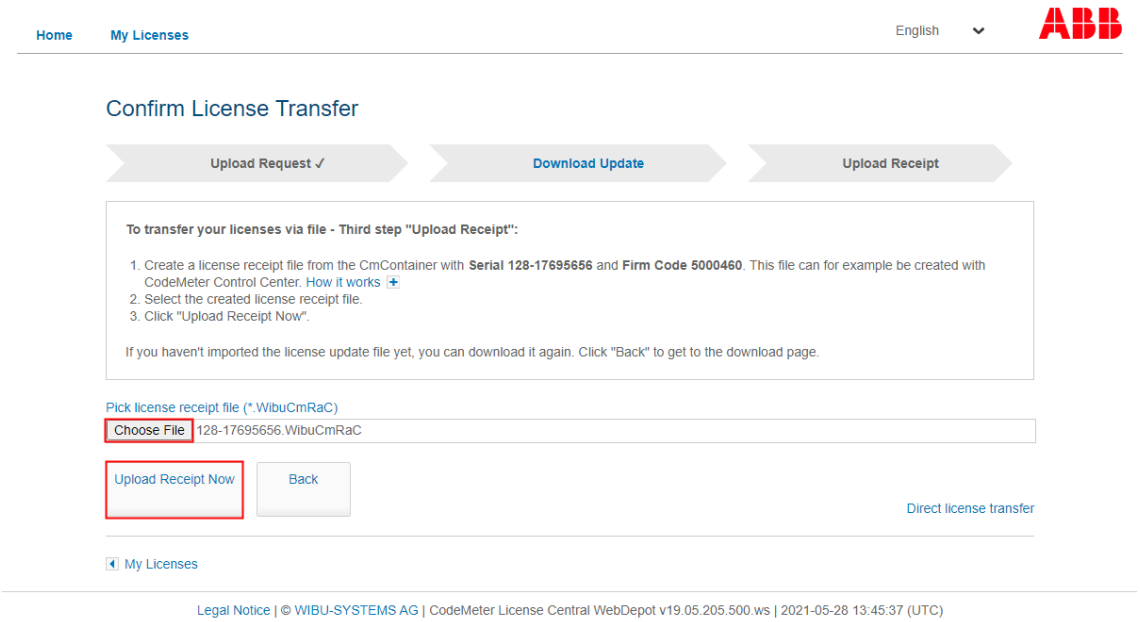


14. On the offline PC open the CmFAS Assistant.
15. Select *“Create receipt”*.

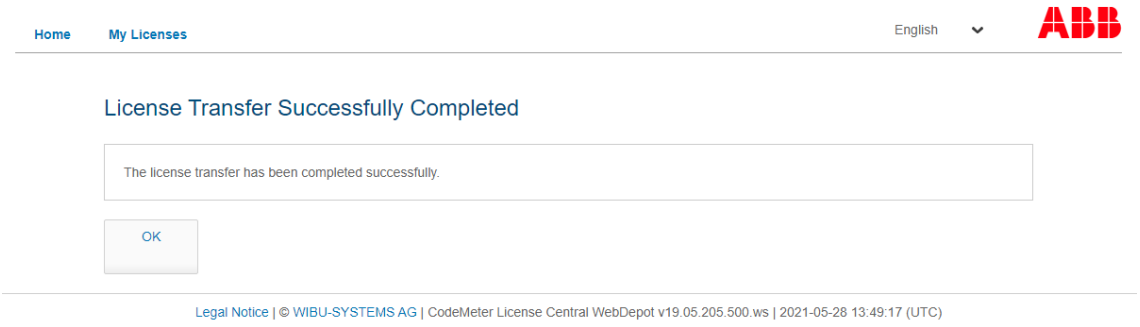


16. Choose a location to save the license receipt file.
17. Transfer the license receipt file from the offline PC to the online PC.

18. On the online PC choose the license receipt file and select *“Upload Receipt Now”*.



⇨ After a succesful license transfer you will receive the following message



1.2.4.10 Generating license information file for support

To create a license information file which includes all license information for the support:

1. Select *“Windows start menu* ➔ *CodeMeter* ➔ *Tools* ➔ *DmDust”*.
   ⇨ The explorer window opens and shows the folder where the created log file *“CmDust-Result.log”* is stored.
2. Please attach this file to any support request regarding your licenses.

1.2.4.10.1 Log files

Sometimes more detailed log files are needed to analyse a situation.

Then please also zip the following folder and attach it to your support request.

C:\ProgramData\CodeMeter\Logs

This folder includes

- CmActDiagLogyyyy-mm-dd-hhmmss.log
- CodeMeteryyyy-mm-dd-hhmmss.log

To make it easier to distinguish when the files were created, they are named as follows:

- yyyy – year, mm – month, hh – hour; mm – minutes, ss – seconds.

1.2.5 Set-up communication parameters in Windows

Set-up communication parameters

To set-up the communication between the PC and the PLC, e.g., for downloading the compiled program, you have to set-up the communication parameters.

The IP address of your PC must be in the same class as the IP address of the CPU.

The factory setting of the IP address of the CPU is 192.168.0.10.

The IP address of your PC should be 192.168.0.X. Avoid X = 10 in order to prevent an IP conflict with the CPU.

Subnet mask should be 255.255.255.0.

Change the IP address

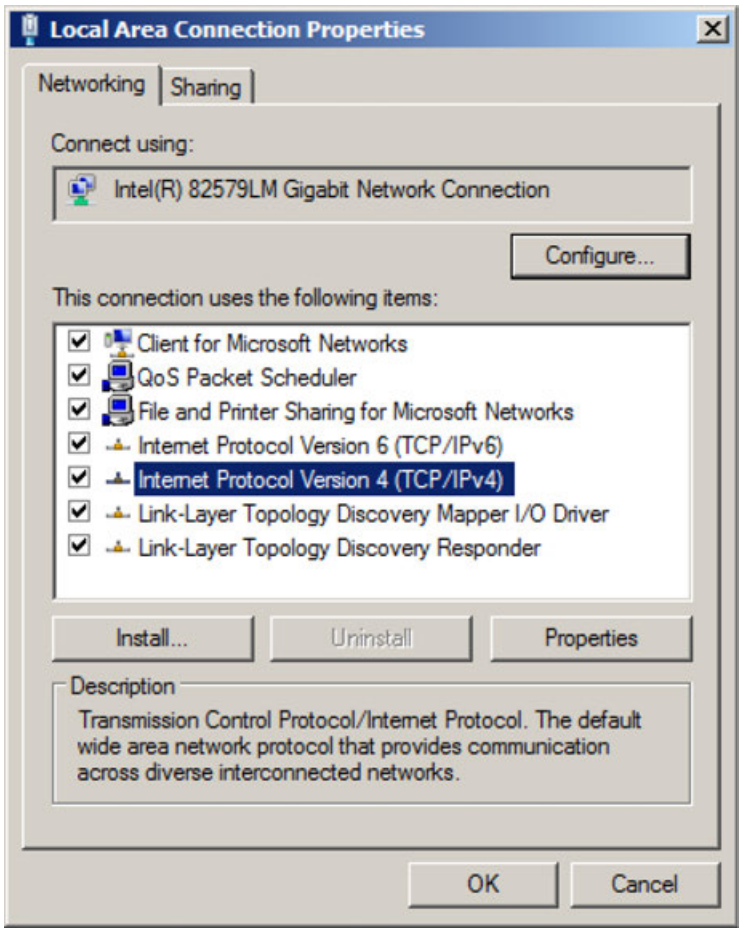
1. Open Windows **Control Panel**. Click *“Network and Internet ➔ Network and Sharing Center”*.
2. Click **Change adapter settings**.

   ⇨

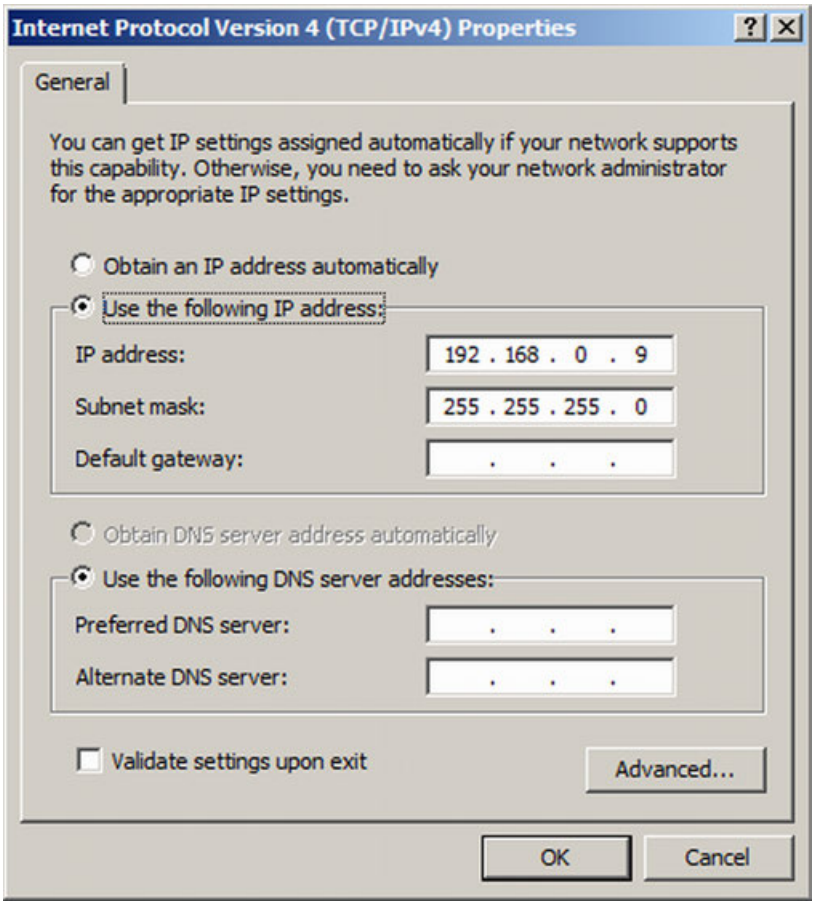
   

   If using existing network with several devices, please pay attention on given network rules or contact your system administrator.

3. Right-click **Local Area Connection (Ethernet)** and select **Properties**.



4. Double-click **Internet Protocol Version 4 (TCP/IPv4)**.



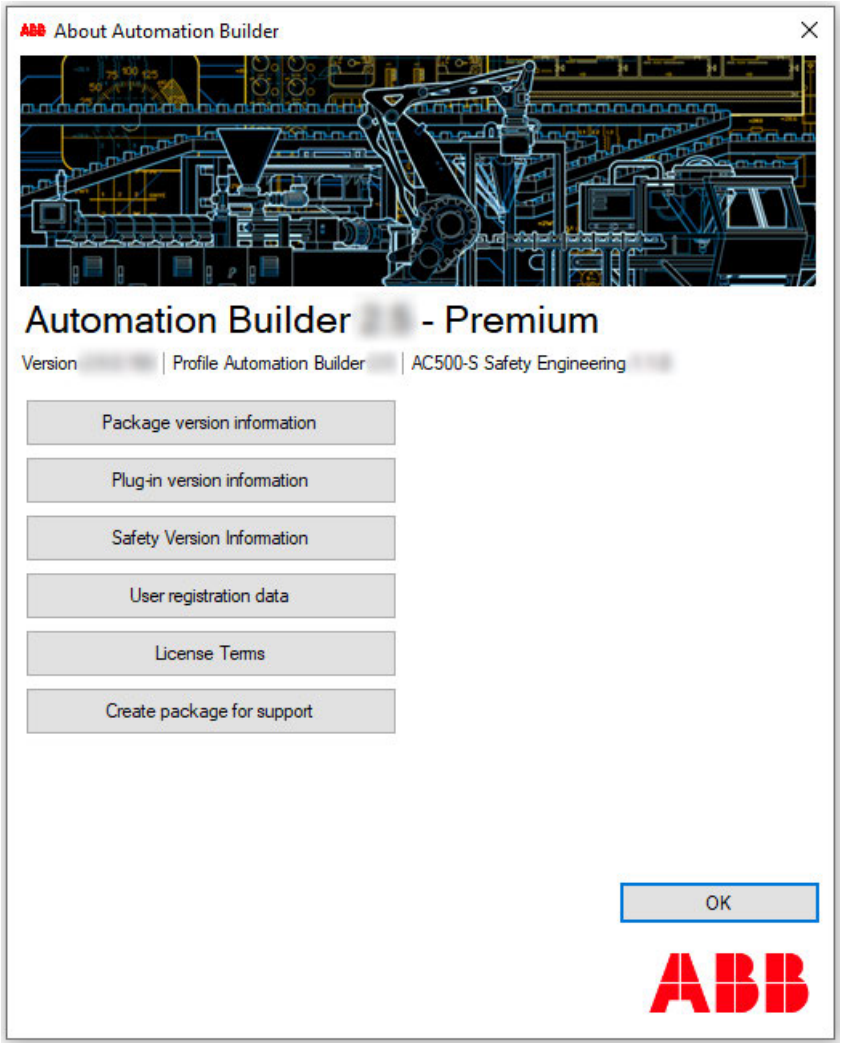
5. Enter your desired IP address and subnet mask.

1.2.6 Further information

Further information on the installed Automation Builder version such as installed packages or license terms can be found on the "About" page (help menu).

Safety Version is visible if safety option is installed. Safety Version Information shows the versions of all safety components.

- Package version information: Further information about all installed package versions is shown.
- Plug-in version infromation: Further information about all installed plug-in versions is shown.
- Safety version infromation: Further information about all safety component versions is shown.
- User registration data: Enter or change your registration data.
- License Terms: Information about the license terms.
- Create package for support: Creates a package which can be saved or sent to support
  ↳ *Chapter 1.2.7 “Create log files for support” on page 50*.



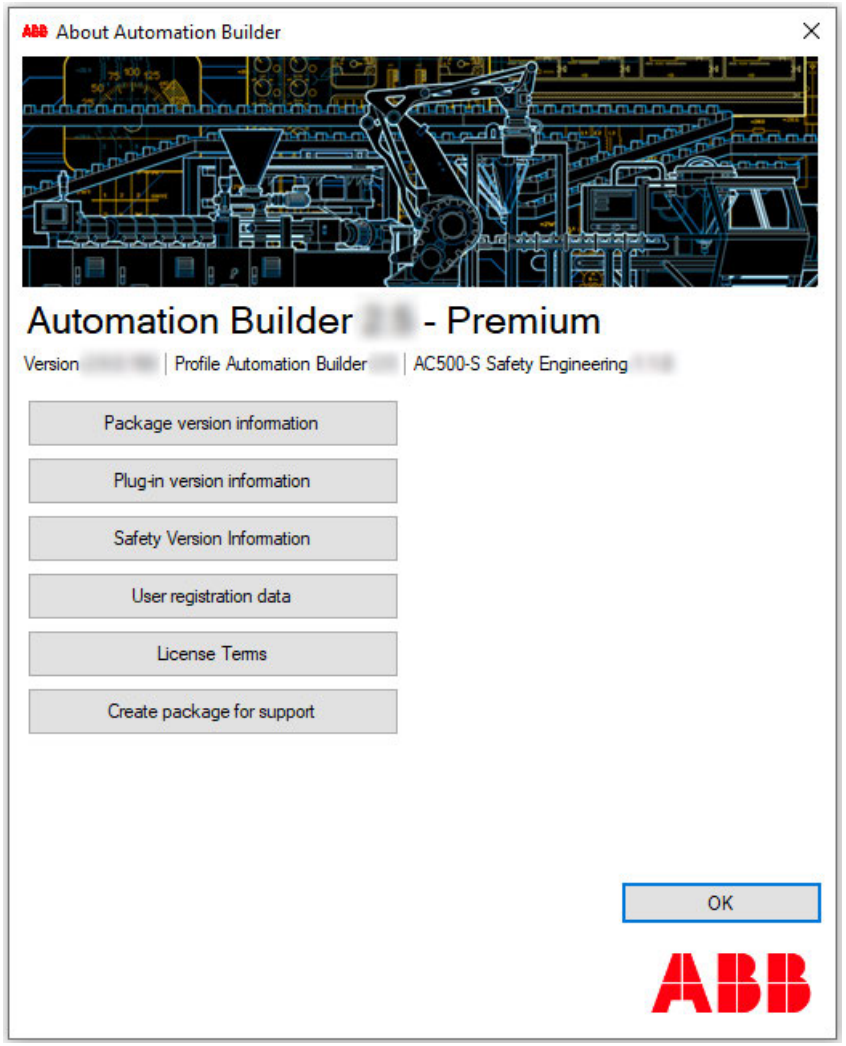
It is possible to either continue working with a project on an older Automation Builder version or to update a project to the latest Automation Builder version.

1.2.7 Create log files for support

Professional support requires some information about the project and the devices.

To collect this information proceed as follows:

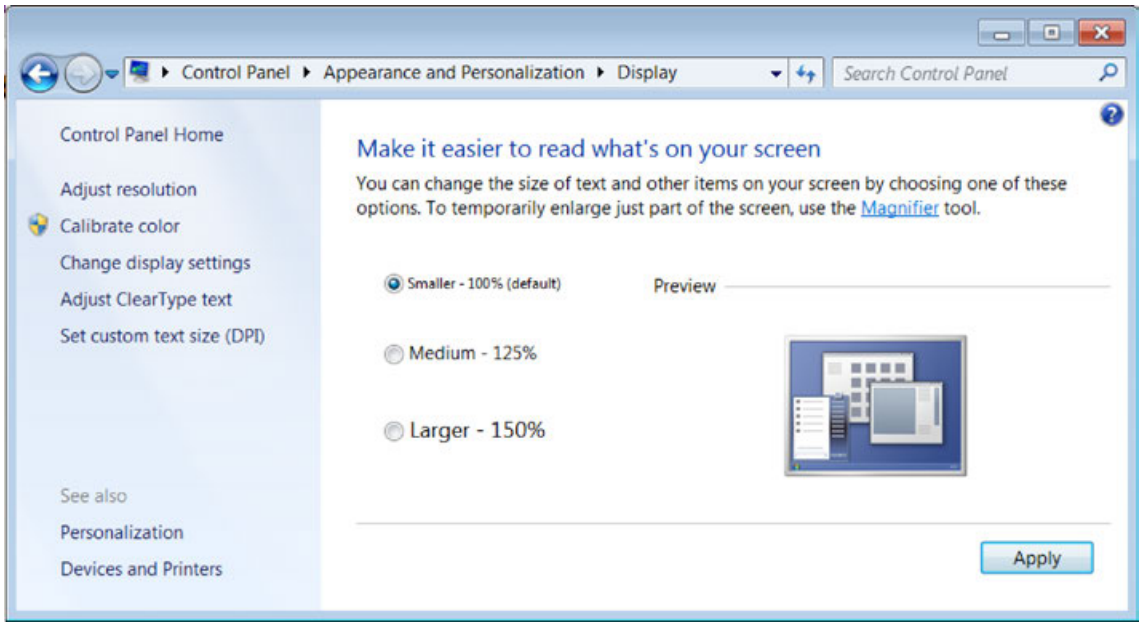
1. Click *"Help ➔ About"* in the main menu of Automation Builder.



2. Click *[Create package for support]* and wait until a list of log files is displayed.
3. Click *[Save package]* to store the zipped log files to your disk, or click *[Send package]* to send the zipped log files to ABB support.
4. Click *[OK]*.

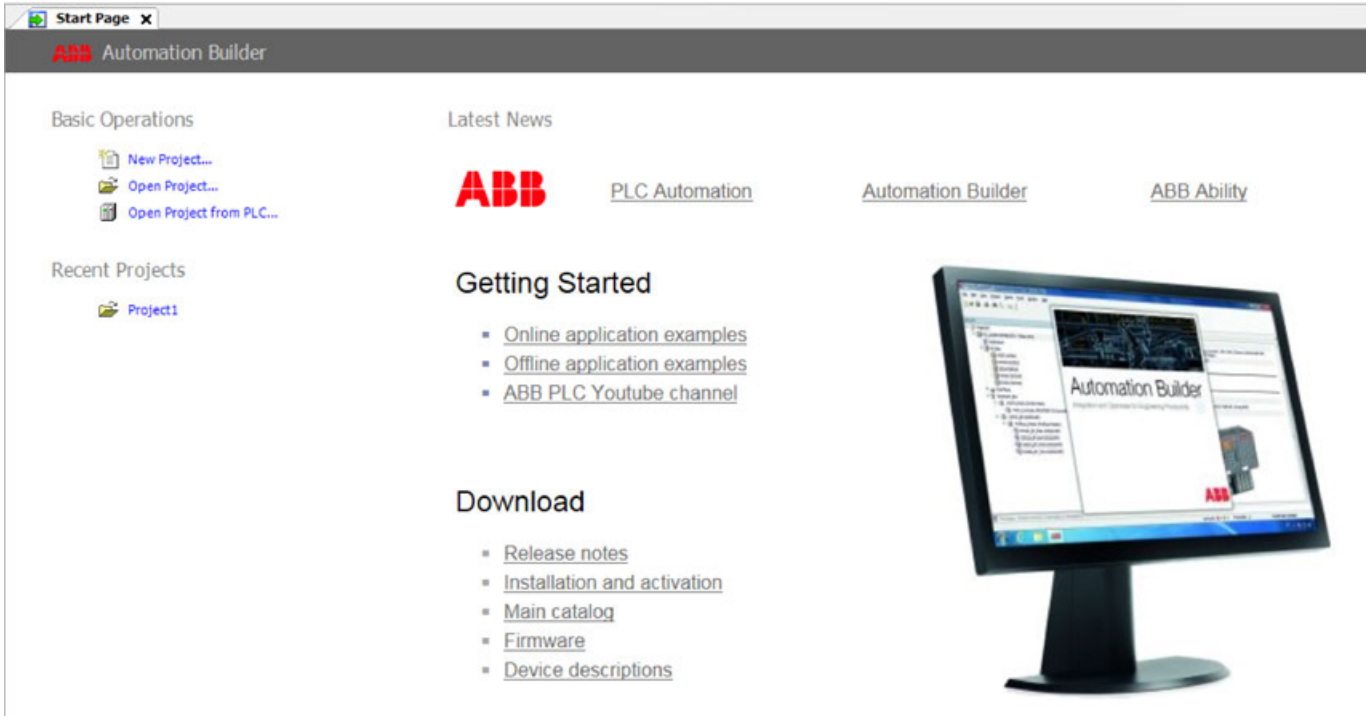
1.2.8 Menues, views, windows

Ensure the full display of Automation Builder editors by choosing the option Smaller - 100 % (default) in "Start ➔ Control Panel ➔ Appearance and Personalization ➔ Display".



1.2.8.1 Start page and menus

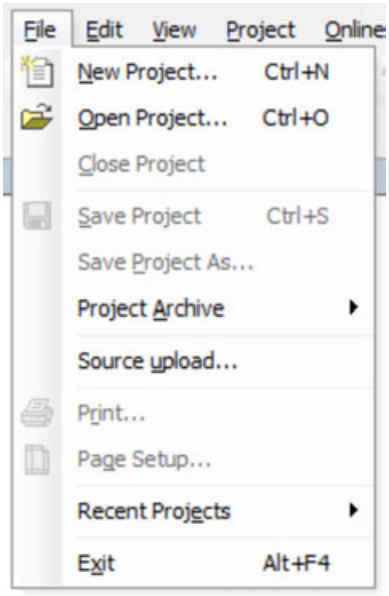
After start-up of Automation Builder software the start page is displayed.



All items of the Automation Builder user interface are described in the CODESYS documentation:

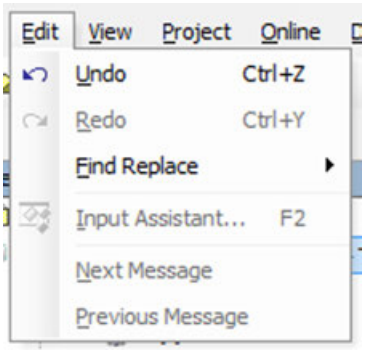
- *Chapter 1.4.1 “Development system” on page 145*

“ File” menu:



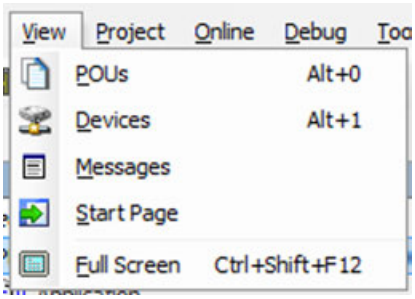
Command	Description
New Project	Creating a new project with the dialog New Project. ↳ *Chapter 1.6.5.1.1 “Project handling” on page 5757*
Open Project	Opening an existing project file.
Close Project	Closing a project file while Automation Builder keeps running. If the current projects keeps information which have not been saved before, you will be asked if the changes should be saved or not.
Save Project	Saving the project at the currently defined location. It is only available if changes have been done to the project since the last saving. This is indicated by an * (asterisk) behind the project name in title bar.
Save Project As	Saving the project whereby project name and location can be defined in a dialog box.
Project Archive	Creating a project archive. ↳ *Chapter 1.6.5.1.1 “Project handling” on page 5757*
Source upload	This command loads the project source code.
Print	Printing the content of the tab which is currently shown in editor window.
Page Setup	Opening the Page setup dialog where the page setup can be configured for printing.
Recent Projects	Reopening the most recently worked projects. The number of projects kept in the list can be defined.
Exit	Closing Automation Builder. If the current projects keeps information which have not been saved before, you will be asked if the changes should be saved or not.

“Edit” menu:



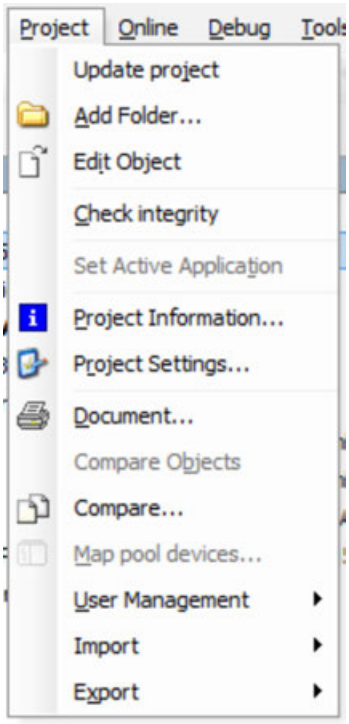
Command	Description
Undo	The last action is reversed.
Redo	A reversed action is redone.
Find Replace	Full text search and text replacement functions.
Input Assistant	This command opens the “Input Assistant” dialog box, guiding you in inserting a program element valid at the current cursor position.
Next Message	This command selects the next message in the messages view.
Previous Message	This command selects the previous message in the messages view.

“View” menu:



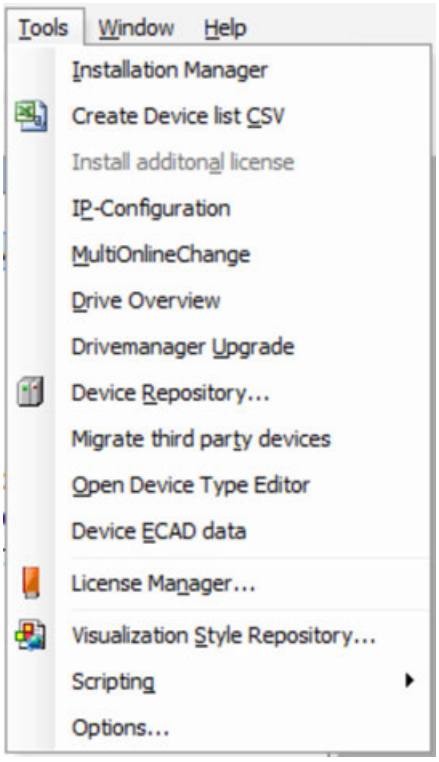
Command	Description
POUs	This command opens the “POUs” view in the CODESYS main window. POUs located here are available in the entire project.
Devices	Displaying the device tree on the left side of Automation Builder.
Messages	Displaying the Messages window.
Start Page	Displaying the Start Page tab. It shows the recently used projects, links to useful commands and the ABB PLC Product Website.
Full Screen	Effecting that CODESYS frame window is displayed in full screen mode. Use *[Ctrl]* + *[Shift]* + *[F12]* to toggle back to normal mode.

“Project” menu:



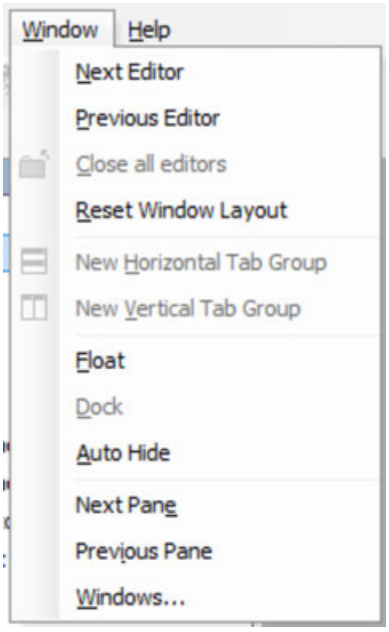
Command	Description
Update project	Updates the project
Add Folder	This command opens a dialog box for defining a new folder in the Devices or POUs view.
Edit Object	This command opens the object in its editor.
Project Information	Opening the "Project Information" dialog window which shows detailed information of the currently opened project.
Project Settings	This command opens the “Project settings” dialog box.
Document	This command opens the “Document Project” dialog box, where you can define the project documentation. This includes the selection of objects in the open project that you want to print.
Compare	This command opens the “Project compare” dialog. In this dialog, you define the reference project to compare with the current project. You configure the comparison process by means of options. When the dialog is exited, the comparison starts and the result is shown in the view “Project Compare - Differences”.
Import	This command opens a dialog box for importing objects from an XML file.
Export	This command opens a dialog box for exporting objects from a project to an XML file.

“Tools” menu:



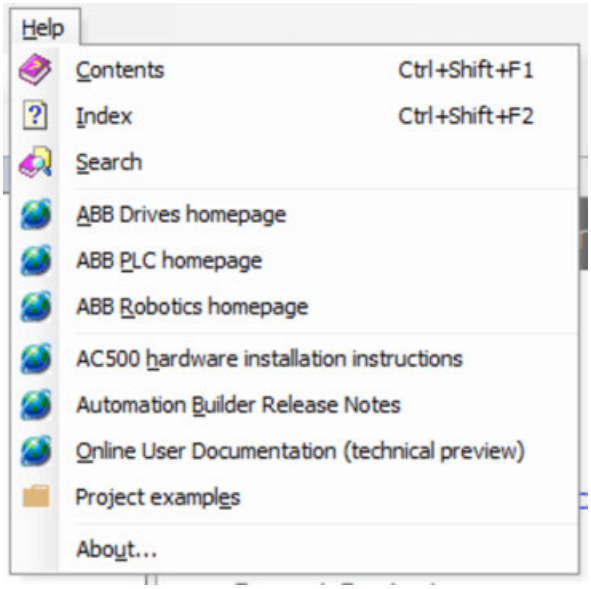
Command	Description
Installation Manager	Installation Manager allows installation and maintenance of Automation Builder packages.
Create Device list CSV	Creates an Exel file with a device list.
Install additional license	Allows to install an additional license.
IP-Configuration	Allows to configure the IP address.
Drive Overview	Shows a Drive overview
Drivemanager Upgrade	Allows a Drivemanager upgrade
Device Repository	Opening the Device Repository. This is a database for all devices which have been installed on the local system for Automation Builder. The default settings are defined by the current profile.
Options	Opening the "Options" dialog. This dialog contains sub-dialogs for configuring the appearance and behavior of Automation Builder. The default settings are defined by the current profile.

“Window” menu:



Command	Description
Next Editor	Opening the next editor window
Previous Editor	Opening the previous editor window
Close All Editors	Closing all opened editor windows and showing the start page of Automation Builder.
Reset Window Layout	This command resets all currently open windows and views to their default positions. You are prompted for a confirmation before the command is executed.
New Horizontal Tab Group	Adding a horizontal aligned tab view. Editor windows can be placed on the different views by drag-and-drop.
New Vertical Tab Group	Adding a vertical aligned tab view. Editor windows can be placed on the different views by drag-and-drop.
Float	This command releases a docked view from its frame in the user interface and repositions it on the screen as a floating window.
Dock	This command returns a floating window, which was released by the “Float” command, to the frame of the user interface.
Auto Hide	This command shows or hides a view.
Next Pane	This command sets the focus on the next pane.
Previous Pane	This command sets the focus on the previous pane.
Windows	Opening the Windows dialog box. This dialog box provides a list of active editor windows which can be accessed via button Activate or closed with button Close window(s). It is possible to close several windows.

“Help” menu:



Command	Description
Contents	Opening the table of contents of Automation Builder online help
Index	Opening the index of Automation Builder online help
Search	Opening the full-text search of Automation Builder online help
ABB Drives / PLC / Robotics home page	Opening the product specific home page
Automation Builder Release Notes	Opening the Release Notes of the current Automation Builder version
Online User Documentation	Opening the Automation Builder webhelp
Project examples	Opening the directory in which project examples are stored
About	Showing the about dialog with details of version

1.2.8.2 'All Messages' window

Errors, warning and success messages are written to the *“All messages”* window:

1.2.9 Device repository

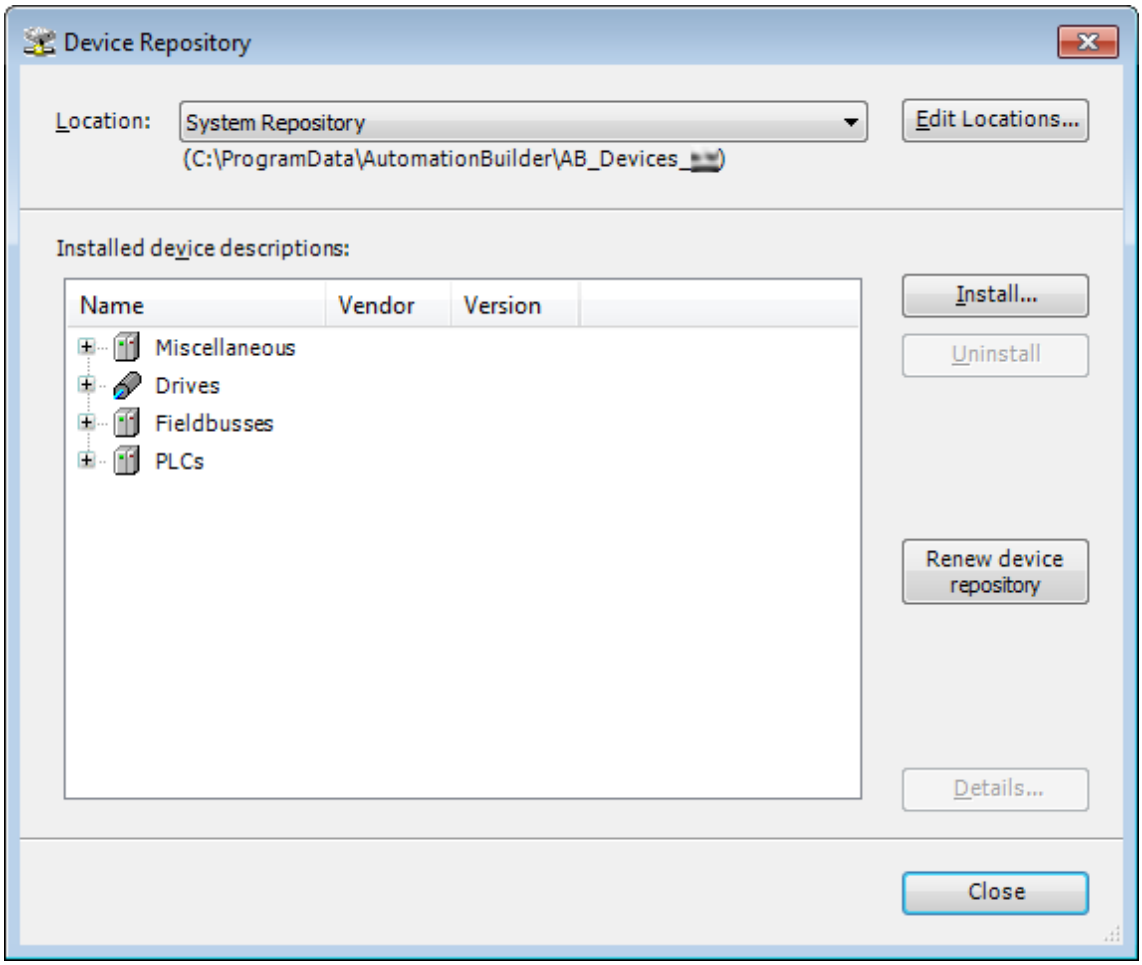
The Device Repository of Automation Builder manages the pool of devices that can be used in the PLC configuration.

You install or uninstall devices in the *"Device Repository"* dialog box. The system installs a device by reading the device description files, which define the device properties for configurability, programmability, and possible connections to other devices.

You can use the devices provided in the device repository by adding them to the device tree of your project.

Dialog device repository

1. Click *"Tools ➔ Device Repository"*.
   ⇨ The *"Device Repository"* dialog box opens.



[Edit Locations]: Changes the default repository location. The devices can be managed at different locations.

[Install] / *[Uninstall]*: Installs or uninstalls devices.

[Renew device repository]: Updates the device list, e.g. after uninstallation of a device.

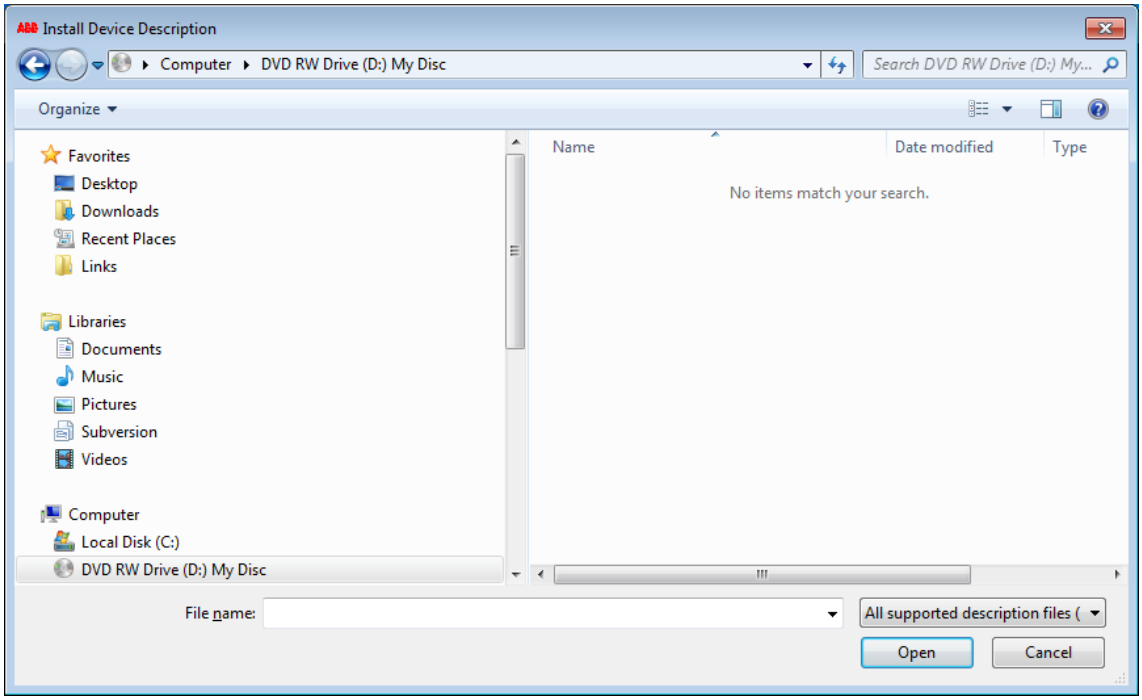
[Details]: Provides technical details on the selected device.

2. Select the install location. *"System Repository"* is set by default.

Installing devices

The device repository cannot be changed manually, e.g. by copying or deleting files. Use always the Device Repository dialog to add or remove devices.

1. Click *[Install]* and select the appropriate file format.
   ⇨ The *"Install Device Description"* dialog box opens.



2. Select the file path of the device description.
3. Select the file type filter of the required device description.
   ⇨ All device descriptions of the selected file type are listed.
4. Select the required device description and click *"Open"*.
   ⇨ Automation Builder adds the device description to the matching category of your device repository.

   If errors occur during installation (for example, missing files that are referenced by the device description), then Automation Builder displays them in the lower part of the device repository dialog box.

> *During the installation the device description files and all additional files referenced by that description will be copied to an internal location. Altering the original files will have no further effects to an internal location.*
>
> *The changes take only effect after reinstalling the corresponding device(s). The version number shown in the information section of the device should be verified.*

Uninstalling devices

Select the device you want to remove and click *[Uninstall]*.

The device is removed from the list.

> *Uninstalled devices which are used in existing projects are indicated by the symbol ❓. The device will not be configured properly.*

1.2.10 Creating and configuring projects

What is a project?

- A project contains the objects which are necessary to create a controller program ("application"):
  - Pure POUs, for example programs, function blocks, functions, and GVLs.
  - Objects that are also required to be able to run the application on a PLC. For example, task configuration, Library Manager, symbol configuration, device configuration, visualizations, and external files.
- In a project, you can program multiple applications and connect multiple controller devices.
- CODESYS manages device-specific and application-specific POUs in the *"Devices"* view ("device tree") and project-wide POUs in the *"POUs"* view.
- For the creation of projects, there are templates that already contain certain objects.
- Basic configurations and information for the project are defined in the *"Project Settings"* and *"Project Information"*. For example:
  - Compiler settings
  - User management
  - Author
  - Data about the project file

  There are settings for the version compatibility of the project in the configuration dialogs in the *"Project Environment"*.
- You save a project as a file in the file system. As an option, you can pack it together with project-relevant files and information into a project archive. It is also possible to save files in a source code management system such as SVN.
- Each project contains the information about the CODESYS version with which it was created. When you open it in another version, CODESYS will notify you about possible or necessary updates regarding file format, library versions, etc.
- You can compare, import/export projects, and create documentation for them.
- You can protect a project from being changed, or even completely protect it from being read. By using user management, you can selectively control the access to the project and even to individual objects in the project.

Handling of AC500 projects such as project creation, export/import, comparison of projects etc. is described in the sections for AC500 V2 products.

Chapter 1.6.5.1.1 "Project handling" on page 5757

1.2.11 Handling of AC500 projects

Handling of AC500 projects such as project creation, export/import, comparison of projects etc. is described in the sections for AC500 V2 products.

Chapter 1.6.5.1.1 "Project handling" on page 5757

Copy-and-paste from one project to another project in two different Automation Builder instances is possible. After copying parts of a project to a higher Automation Builder version the copied components have to be updated.

It is not possible to downgrade a project to an earlier Automation Builder version.

- *Import of export files is only allowed in the same profile version.*
- *Copy-and-paste of configurations must not be used to copy objects to an earlier version.*

> *Automation Builder performs an integrity check for the PLC configuration before generating the configuration.*

Project archive

Automation Builder supports the creation and the import of project archive files. Archive files contain all relevant project data including the PLC configuration, the CODESYS project files and all device descriptions. This allows exchanging Automation Builder projects without taking care of the target environment General Settings. *Chapter 1.6.5.1 “General settings” on page 5757*

User and access rights of a project

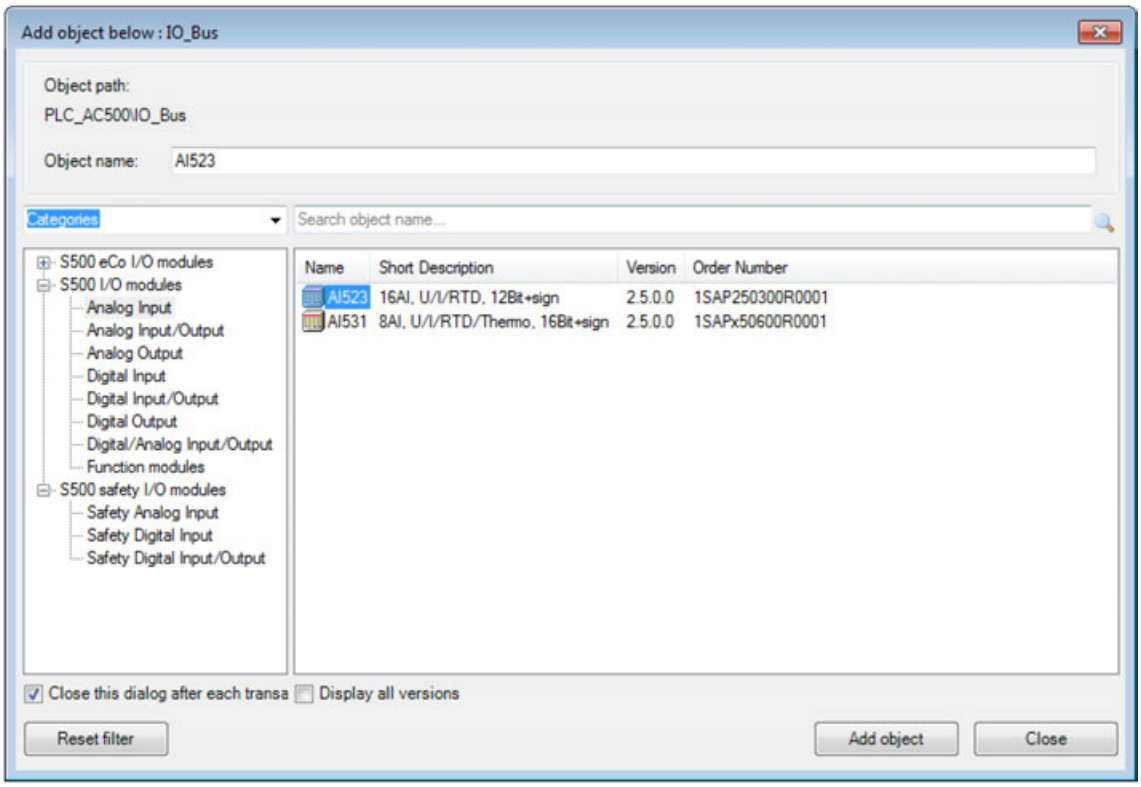
The 'User Management' provides functions for defining user accounts and configure the access rights within a project. The rights to access project objects via specified actions are assigned only to user groups, not to a single user account. So each user must be member of a group General Settings. *Chapter 1.6.5.1 “General settings” on page 5757*

1.2.12 Connection of devices

1.2.12.1 Configuring devices

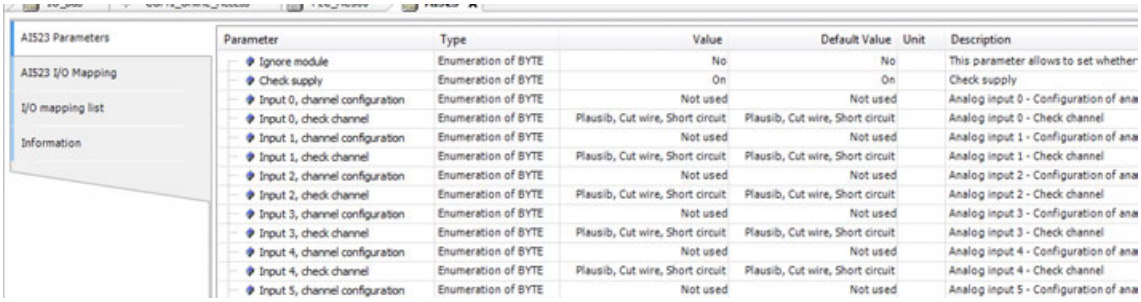
Modify your Automation Builder project by adding device objects. Preset items can be replaced in the same way.

1. In the device tree, right-click an item node. Select *“Add object”*.



2. Select the desired object and click *[Add object]*.

3. Double-click the new object in the device tree to configure the device settings. Depending on the selected item different configuration tabs are available.



1.2.12.2 Symbolic names for variables, inputs and outputs

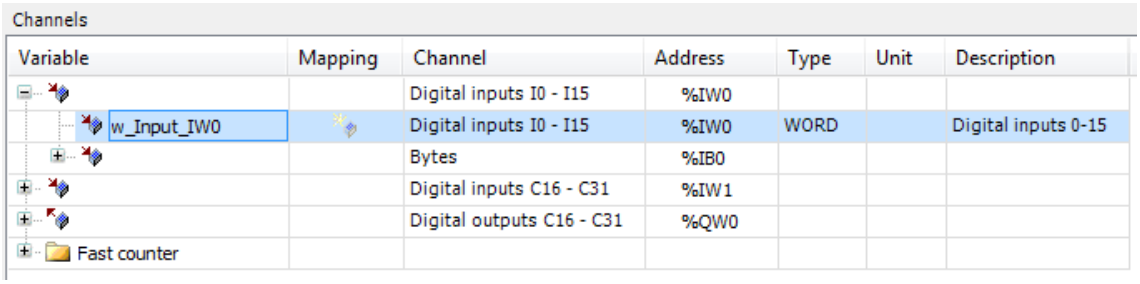
> *The IEC naming rules are not checked during input in Automation Builder.*

Input and output mapping

Devices with I/Os provide an I/O Mapping tab in their configuration editor where the available I/O channels can directly be mapped to a global variable.

The corresponding variable declarations are automatically created in a global variables object in a subfolder of the Global Variables section in the CODESYS project.

All available I/O channels can easily be assigned to a variable.

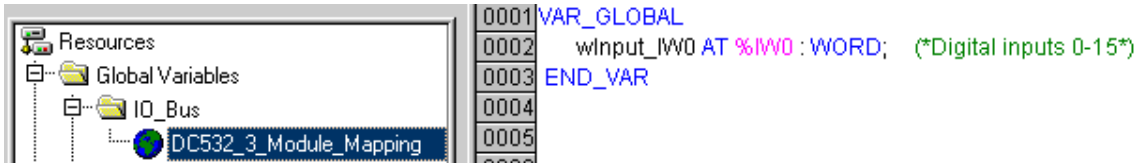


The variable is automatically added to the Global Variables in the CODESYS project after recreating the configuration data *Chapter 1.2.15.1 “Creating configuration data” on page 65*.

> *AC500 uses Motorola Byte Order (Big Endian).*
>
> *The numbers in column Channel correspond to the channel numbers only and not to the bit position inside the WORD variable.*

> *Only entries with a data type set in column "Type" can be mapped. These entries can be expanded to show the available I/O channels.*
>
> *If the project has been imported from a previous Automation Builder version, all variables should be checked to avoid inconsistencies concerning the I/O mapping.*

The variable is automatically added to the global variables in the CODESYS project after (re)creating the configuration data:



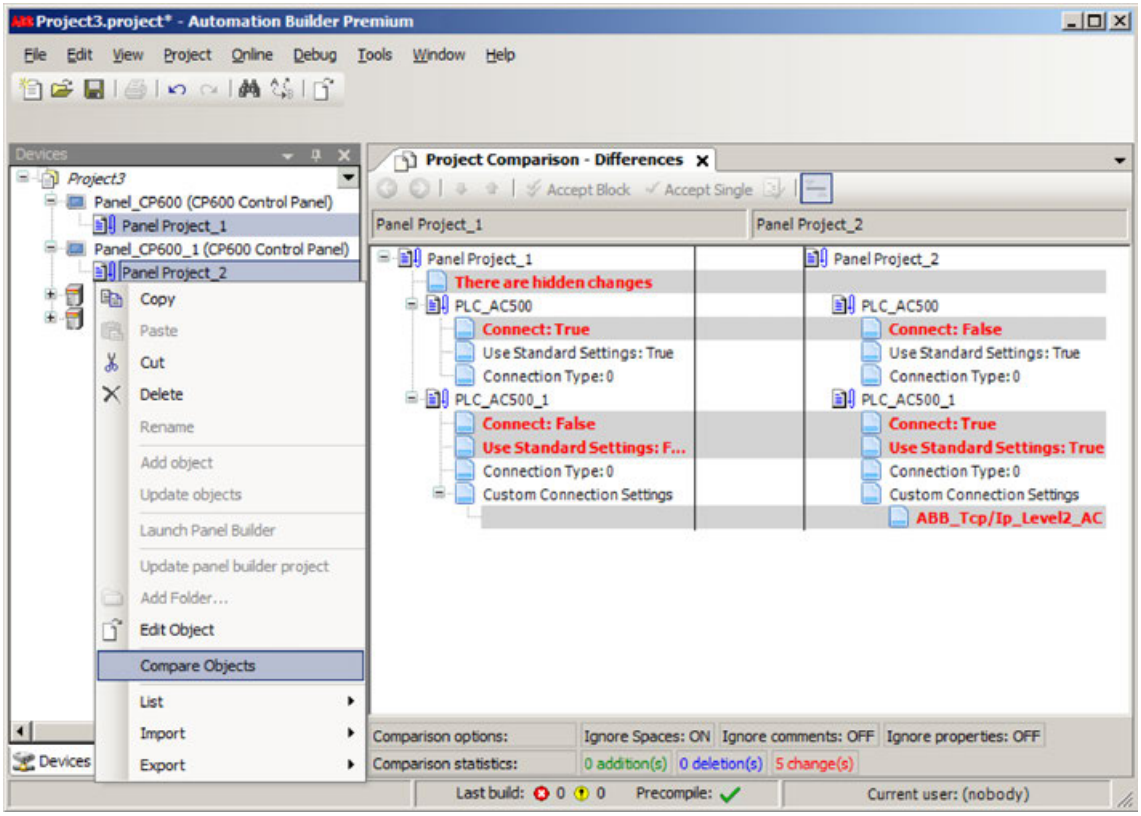
An additional GVL (Global Variables List) can be created and transferred to CODESYS V2.3. Editing of lists created in CODESYS V2.3 is not possible.

1.2.12.3 Update of AC500 devices

Perform a firmware update to update AC500 V2 devices. *Chapter 1.6.5.1.7 “Firmware identification and update” on page 5786*

1.2.12.4 Comparing objects

To compare similar objects within a project (such as the project configuration) select both objects. Right-click and select **Compare Objects** to see the differences.



1.2.13 Connection of serial interfaces

Depending on the device type, the configuration of serial interfaces is different.

AC500 V2 Products: *Chapter 1.6.5.2.11 “Serial interfaces COM1 and COM2” on page 6098*

1.2.13.1 Programming of applications

To create an application program which can be run on the controller, you fill POUs with declarations and implementation code (source code), establish the link from the controller I/Os to application variables, and configure the task assignment. After checking and debugging, the CODESYS compiler creates the application code which can be downloaded to the controller.

The programming of the application POUs is supported by the programming language editors and other features such as text lists, image pools, alarm configurations, pragmas, refactoring, and ready-to-use POUs from CODESYS Development System or libraries.

There are features for syntax checking and code analysis, for achieving data persistence, and for encrypting the application code which is downloaded to the controller.

1.2.14 I/O mapping

For all connected I/O devices perform an I/O Mapping.

↬ *Chapter 1.6.5.1.4 “I/O mapping list” on page 5781*

1.2.15 Data transfer and CODESYS programming

1.2.15.1 Creating configuration data

☑ If the setup of all devices has been finished, you can create the configuration file.

▷ Right-click on *“Application”* and select *“Create configuration data”* from context menu.

⇨ The generation process starts. The progress is shown in the status bar of Automation Builder.

Check integrity

Within a Automation Builder version only device versions of this Automation Builder version can be used. By following the use case described in the chapter Use Cases the system takes care about this project integrity. Additionally Automation Builder performs an integrity check for the selected PLC before generating the configuration.

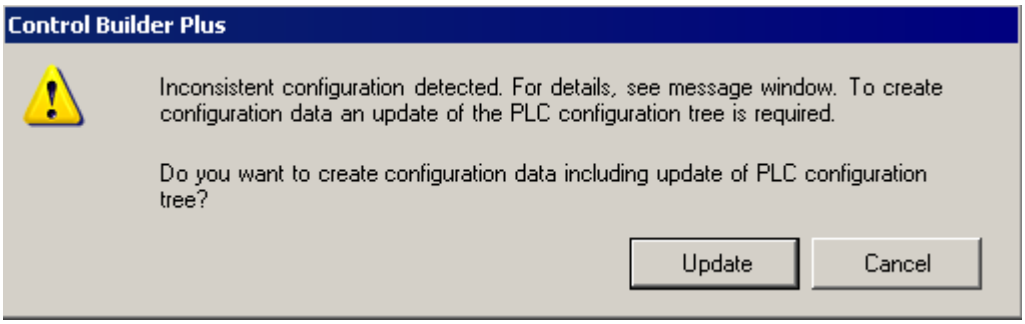
The integrity check can manually be called for the complete project:

1. Select the menu item *“Project ➔ Check integrity”*.

   ⇨ Automation Builder checks the project integrity for the complete project ("Project integrity" checks if all devices in the device tree are installed in the device repository).

   If the integrity check was successful a success message is displayed.

   If the integrity check was not successful the following message is displayed:



2. Click *“Update”* button:

   ⇨ Automation Builder updates the configuration to the latest version, creates configuration data and starts CODESYS V2.3.

3. Click *“Cancel”* button:

   ⇨ Creates configuration data and Automation Builder does not start CODESYS V2.3.

If the integrity check fails the Message Window displays further details.

> *In case of inconsistent configuration data update correct your project. Update all devices and install all 3rd party devices as used in the project.*

Following error messages indicating that devices are not installed properly: "The device XXXXXX was not found in the device repository! Please reinstall this device using the menu item *"Tools ➔ DeviceRepository"*.

Install the device. *Chapter 1.2.9 "Device repository" on page 58*.

"The version of device XXXXXX is not compatible with the current version."

Update to new version.

1.2.15.2 Launching programming system CODESYS V2.3.9.x

In your PLC project, double-click on *"Application"* to open the CODESYS V2.3.9.x project.

CAUTION!

Risk of damaged Automation Builder projects!

Projects created with Automation Builder are incompatible with CODESYS V2.3.9.x. Opening these projects directly with CODESYS V2.3.9.x may cause corrupted projects. Always use Automation Builder to open these projects.

> *If several instances of CODESYS V2.3.9.x are opened, double-clicking on the target item brings the corresponding CODESYS V2.3.9.x instance to the front.*

CPU name

The name of the used AC500 CPU is shown in the main node of the device tree.

The names of the devices can be edited by selecting the corresponding entry and clicking the entry. The name must be unique and cannot be reused in the same project. Automation Builder automatically appends double entries with an up counting number as an suffix.

> *The IEC naming rules are not checked during input in Automation Builder.*

1.2.15.3 Source download/upload in Automation Builder

Source download

The *"Source download"* downloads and stores the current project as a Zip-file on the PLC's memory card.

- Right-click *"PLC_AC500_V2 <...>"* node and click *"Source download"*.

Source upload

The *"Source upload"* retrieves a previously downloaded project from memory card to a selectable directory on the Automation Builder PC and optionally opens Automation Builder with this uploaded project.

- Click *"File"* in the main menu of Automation Builder and click *"Source upload"*.

Internally both commands are invoking a hidden instance of CODESYS V2.3 which is then working in batch mode to accomplish the download/upload.

Most of the error messages which could occur during the process are raised by CODESYS V2.3 and will then be displayed in Automation Builder as a result message box (the language of the message box is the installed language of CODESYS V2.3, the messages are sometimes quite "basic").

Before using the commands, check the following:

- *No CODESYS V2.3 instance started from within Automation Builder on the same PLC node should be running.*
- *To perform the download/upload, CODESYS V2.3 must be able to log in/on to the PLC. Therefore the gateway settings should be correct and saved to project file. It is recommended to test once the ability of CODESYS V2.3 to log in/on to the PLC.*
- *A formatted memory card must be mounted in the PLC's memory card slot.*

1.2.16 AC500 PLC configuration

See Getting Started for AC500 V2 products. *Chapter 1.2.18.1.1 "Hardware AC500 V2" on page 67*

See Starter Kit for AC500-eCo products. *Chapter 1.6.1.10 "AC500-eCo starter kit" on page 3741*

1.2.17 Converting an AC500 V2 project to an AC500 V3 project

A project that has been configured for an AC500 V2 PLC can be converted to a project for an AC500 V3 PLC.

Essentially, the conversion is done in Automation Builder, however, some additional actions have to be executed manually. The complete procedure is described in the application example

Instructions on how to convert a V2 project to a V3 project and differences between V2 and V3.

1.2.18 Example projects

1.2.18.1 Example projects for AC500 V2

1.2.18.1.1 Hardware AC500 V2

Configuration for example projects

The example projects require a small PLC configuration with I/O devices, e.g., as available in the training case TA515-CASE. *https://to.abb/AfO9-ftT*

Table 2: Modules for example projects to get started with AC500 V2 PLC

Product name	Type	First project ↳ *Chapter 1.2.18.1.2 "Example project for central I/O expansion" on page 70*	Second project ↳ *Chapter 1.2.18.1.3 "Example project for remote I/O expansion with PROFINET" on page 109*
PM585-ETH	AC500 V2 CPU	x	x
TB521-ETH	terminal base for CPU	x	x
DA501	analog/digital mixed input/output (I/O) module	x	x
TU516-H	terminal unit for I/O module	x	x
CM579-PNIO	PROFINET communication module	--	x
CI502-PNIO	PROFINET communication interface module	--	x
TU508-ETH	terminal unit for communication interface module	--	x
TA524	blind cap for terminal base	x	x

Connections

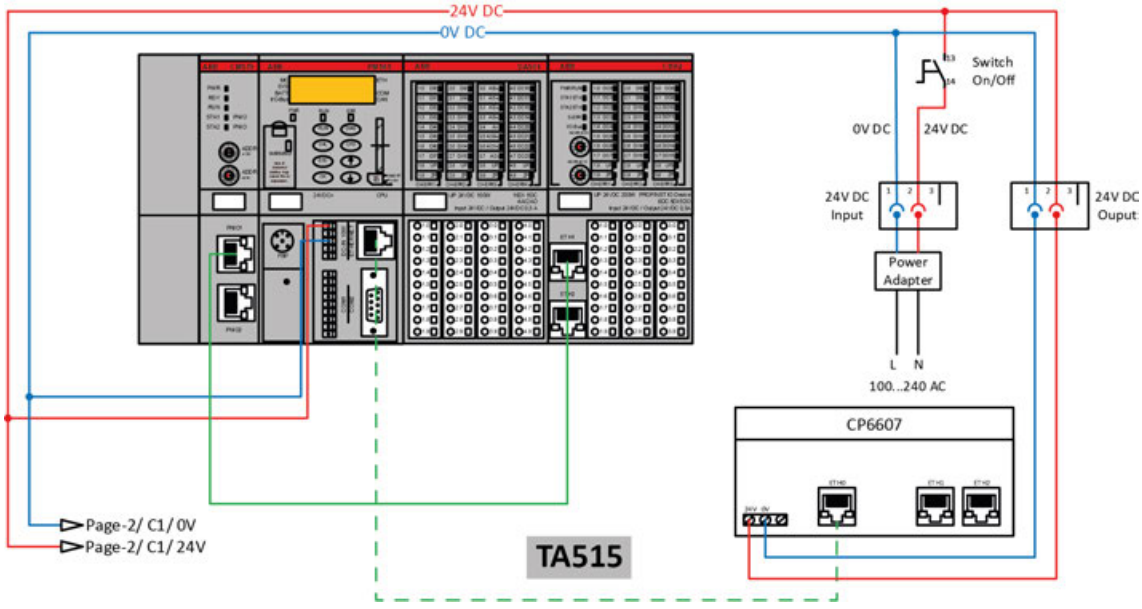


Fig. 5: Training case TA515

In the training case, the control panel CP6607 is included. A control panel is not needed for the example projects.

For testing the example project some inputs require to be connected as follows:

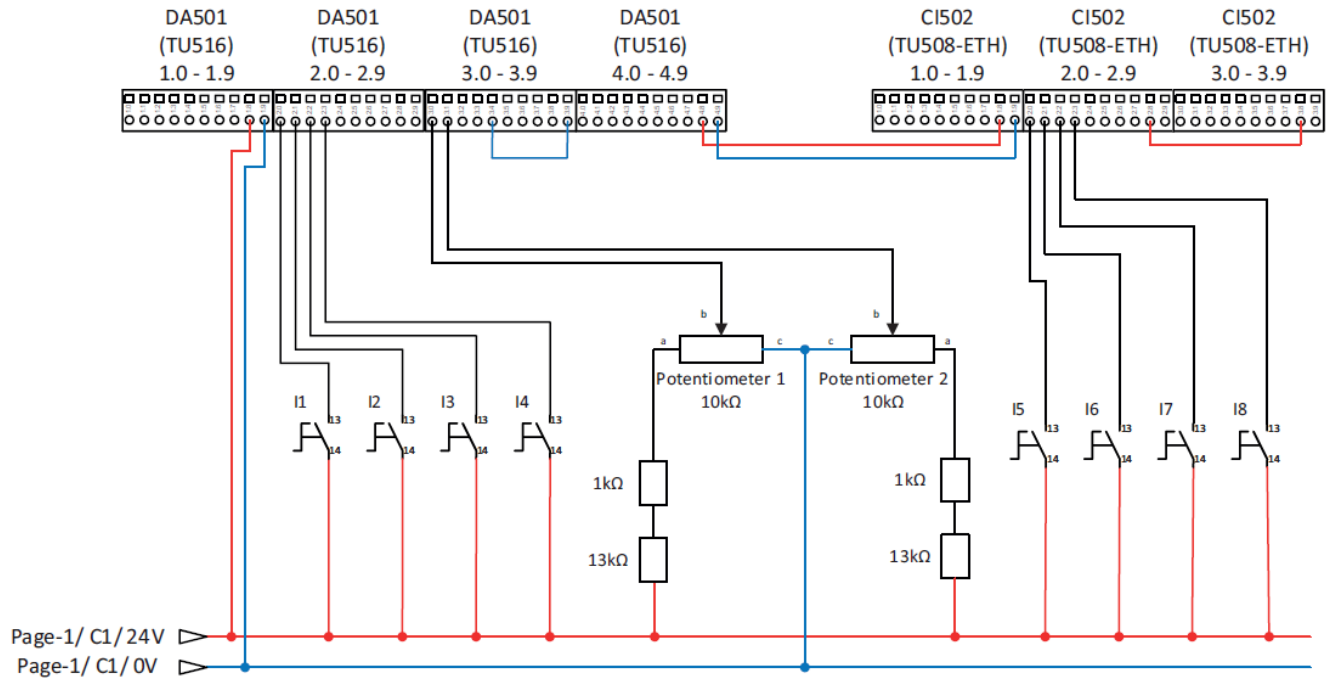


Fig. 6: Wiring of training case

> *For the example projects, not all input switches and none of the potentiometers included in training case are necessary.*
>
> *You will need switch I1 for the example project for central I/O expansion.*
>
> *You will need switch I5 for the example project for remote I/O expansion.*

System assembly, construction and connection

> **NOTICE!**
>
> **Avoidance of electrostatic charging**
>
> PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:
>
> – Touch a grounded object to discharge potential static.
> – Wear an approved grounding wrist strap.
> – Do not touch connectors or pins on component boards.
> – Do not touch circuit components inside the equipment.
> – If available, use a static-safe workstation.
> – When not in use, store the equipment in appropriate static-safe packaging.

You can mount AC500 PLC either to DIN rail or to a metal plate ↬ *Chapter 1.6.3.6.3 “Mounting and demounting” on page 5325*. Here, we recommend to mount on DIN rail.

1. Snap the terminal base onto DIN rail.
2. Snap the additional terminal units for I/O modules onto DIN rail.
3. Make the sensor/actuator wire connections according to the dedicated electronic module you want to use. Provide external process power supply as required.
4. If required, make the fieldbus connections according to the dedicated master communication module you want to use.

5. Plug the appropriate electronic and I/O modules in the correct locations (processor module, communication modules on terminal base, and eventually also communication interface modules and I/O modules onto dedicated terminal units).
6. Connect a programming cable (Ethernet cable between ETH port of CPU and PC with engineering software).

1.2.18.1.2 Example project for central I/O expansion

The following steps show how to set-up an application project and configure the hardware. A simple logic is used as example to introduce in programming and commissioning of the PLC. The workflow for creation of a visualization is explained, as well as how to set-up a web server for visualization.

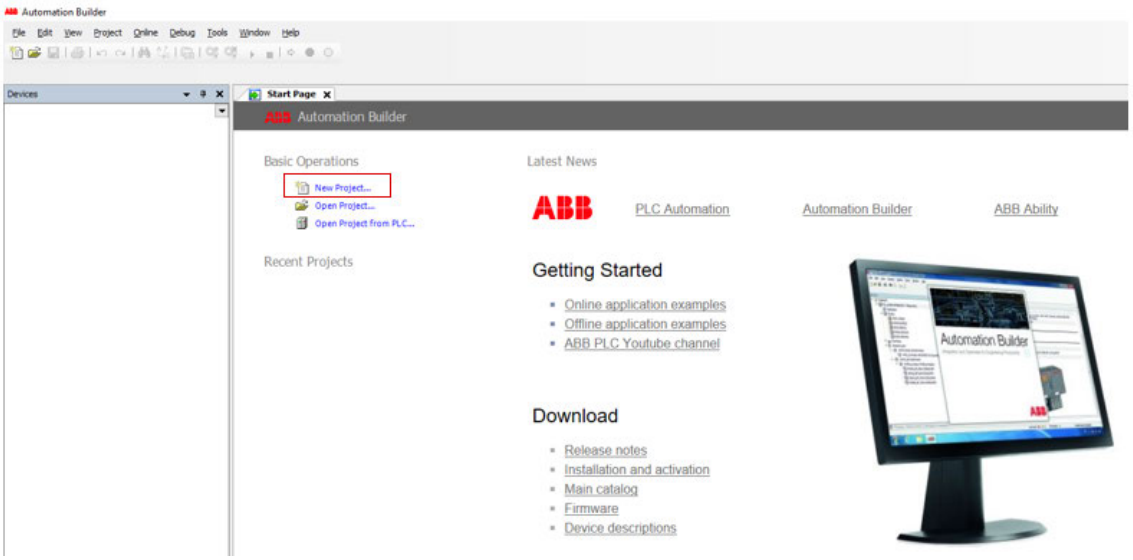
Preconditions

- Automation Builder is installed and licensed as, at least, basic edition ↬ *Chapter 1.2.4 "Managing your licenses" on page 20*.
- AC500 V2 CPU is assembled and connected to the PC ↬ *Chapter 1.2.18.1.1 "Hardware AC500 V2" on page 67*.

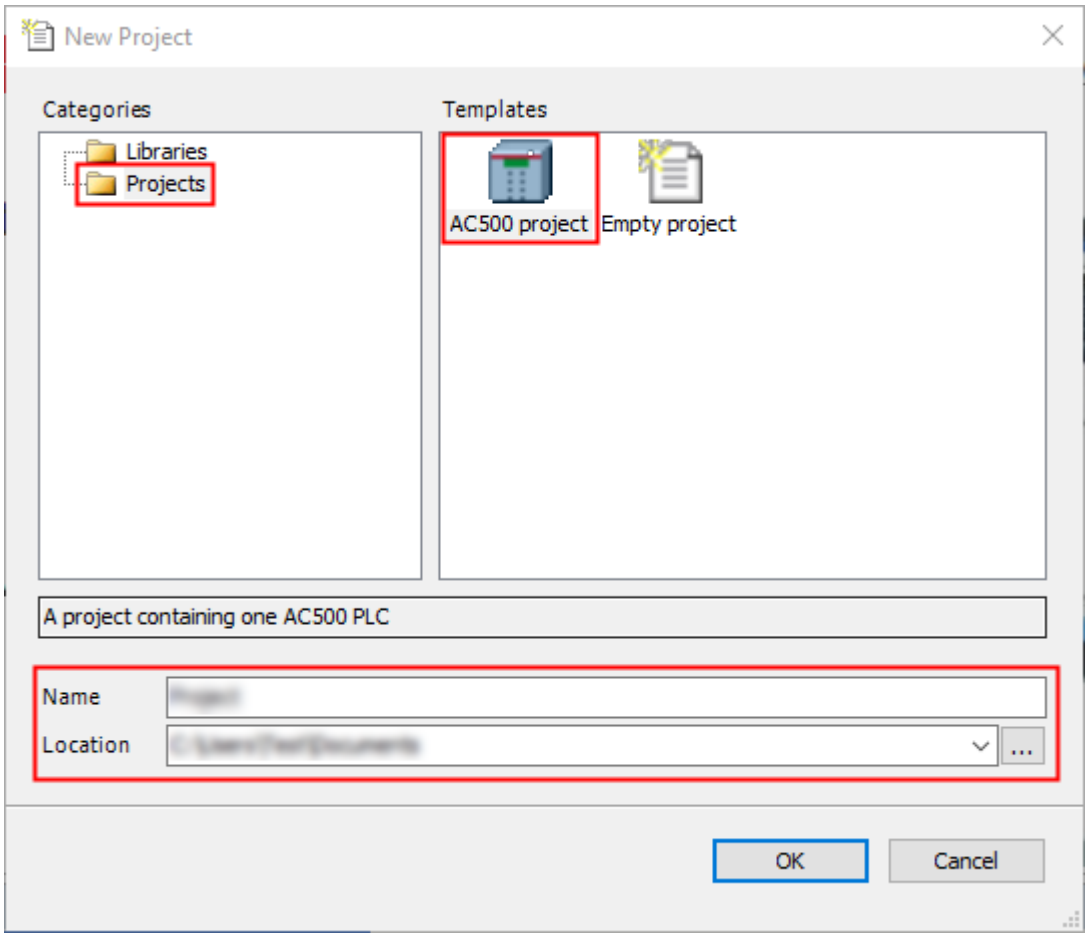
Create, set-up and save your AC500 V2 project

Create a project

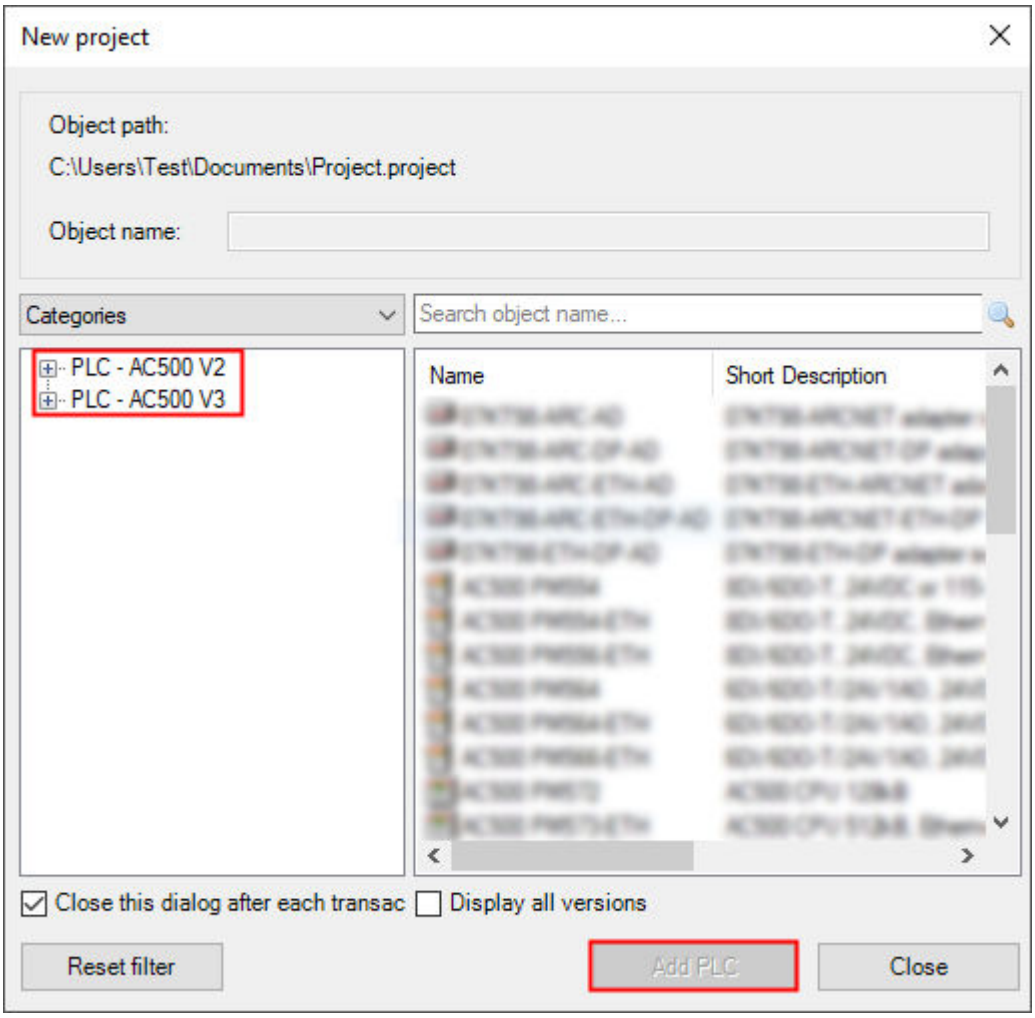
1. Launch Automation Builder either out of the desktop icon or out of the Windows menu.



2. Select *"New Project"* or go to menu *"File ➔ New Project"*.

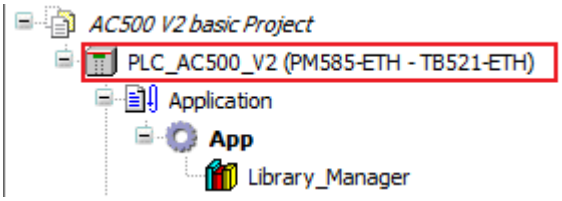


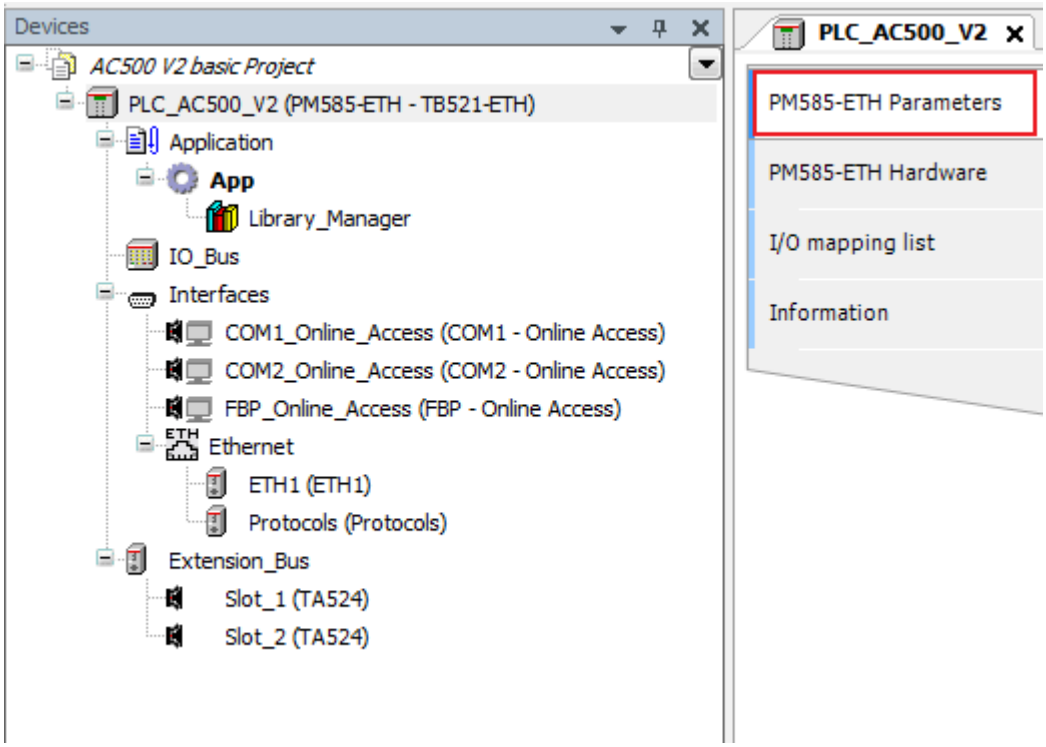
3. Select *“Projects”*.
4. Select *“AC500 project”*.
5. Fill in project name.
6. Choose a location to save the project to.
7. Select *“OK”*.
8. Select *“PLC - AC500 V2”*.
9. Select the CPU according to your hardware set-up.



10. Select *"Add PLC"* to add the CPU to your application.

Configure your CPU

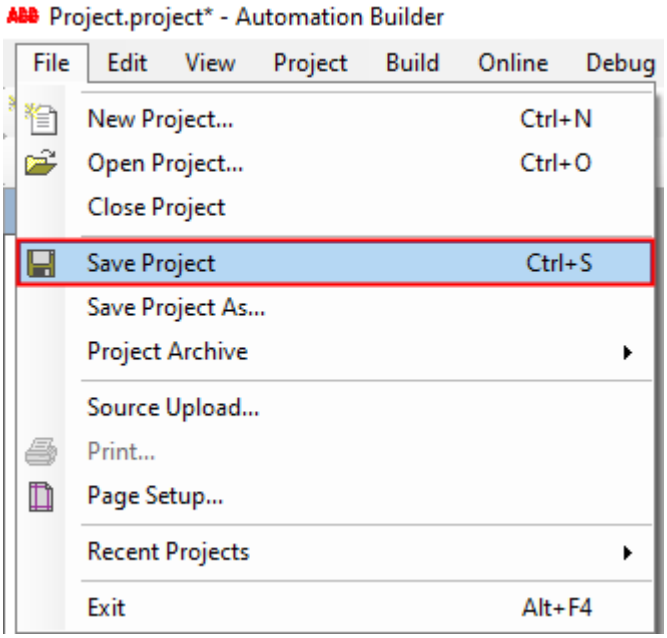




Parameter	Type	Value	Default Value
Auto run	Enumeration of BYTE	On	On
Error LED / Failsafe function	Enumeration of BYTE	On	On
Check battery	Enumeration of BYTE	On	On

1. Double-click *“PLC_AC500_V2”*.
   ⇨ A tab opens in the editor view.
2. Select *“CPU-Parameters Parameters”*.
3. Under parameter *“Check battery”*, choose the value *“Off”* since there is no battery present inside the CPU module.
4. Keep the default values for all other parameters.

Save the project



▷ Select menu *"File ➔ Save Project"*.

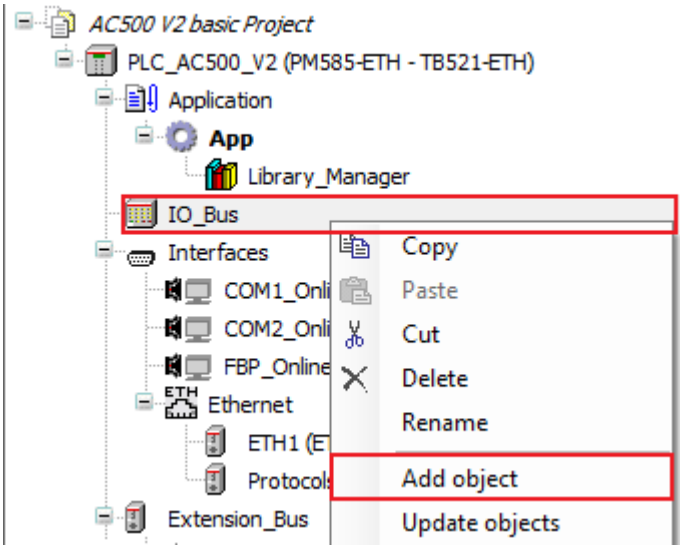
Alternatively, select the save icon in the tool bar.

Alternatively, press *[Ctrl]* + *[S]*.

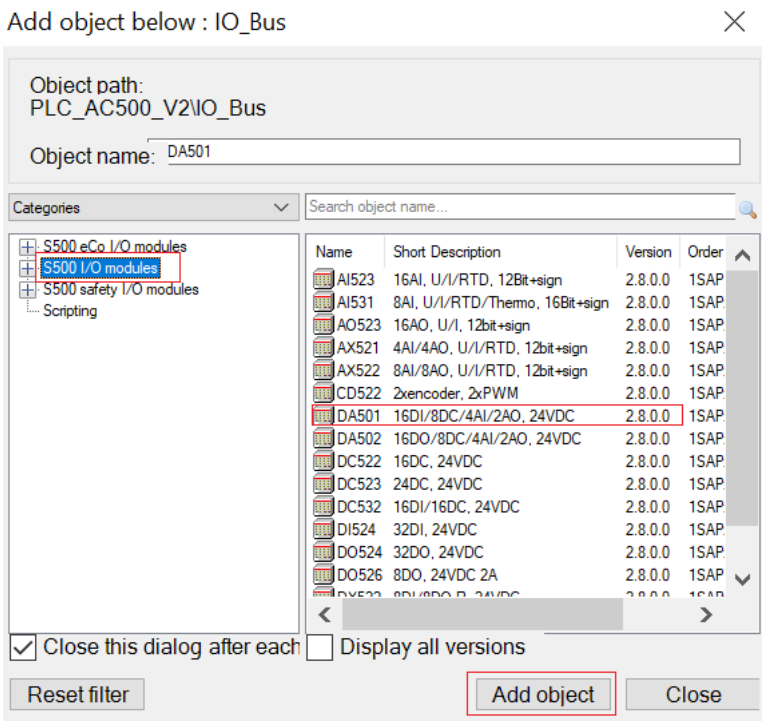
Configure the I/O module

- The types and order of modules in the Automation Builder project must match the real hardware configuration.
- The position of the modules in the device tree can be changed by drag and drop.

Add an I/O bus module

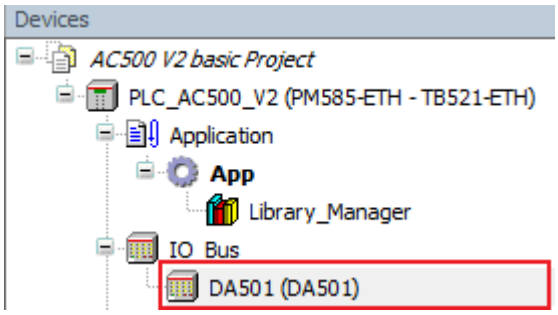


1. Right-click *"IO_Bus"* in the device tree.
2. Select *"Add object"*.

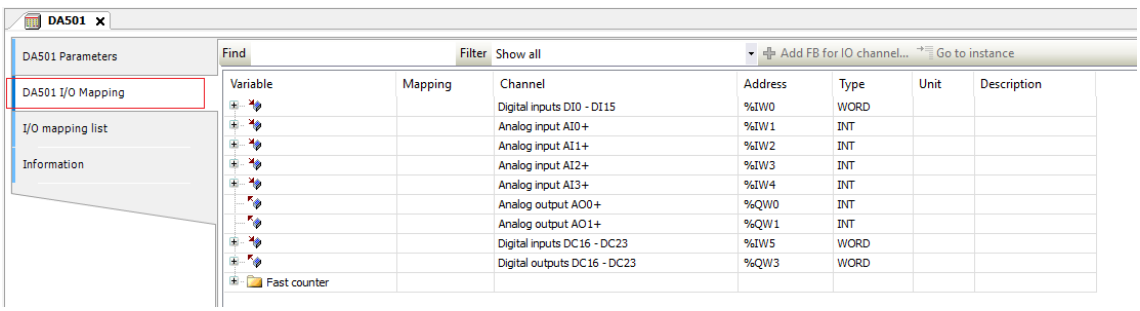


3. Select *“S500 I/O modules”*.
4. Select *“DA501”* module.
5. Select *“Add object”* to add the module to the I/O bus.

DA501 variable mapping

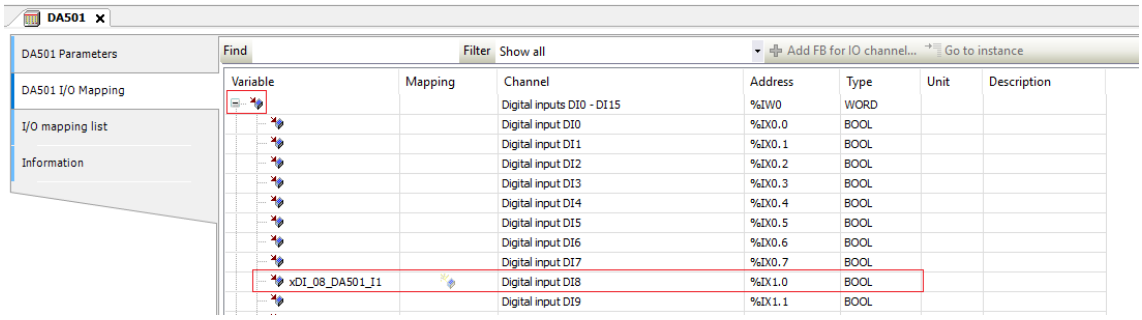


1. Double-click *“DA501”* in the device tree.
   ⇨ A tab opens in the editor view.
2. Select *“DA501 I/O Mapping”*
   ⇨ Here, you will map variable names (symbols) for the channels you will need in the program.



The suggested name convention is based on "Hungarian notation". A name prefix is describing variable type: e.g., "x" = variable of type BOOL, "w" = WORD, "i" = INT (integer) etc. This increases the code readability and is helpful for program analysis.

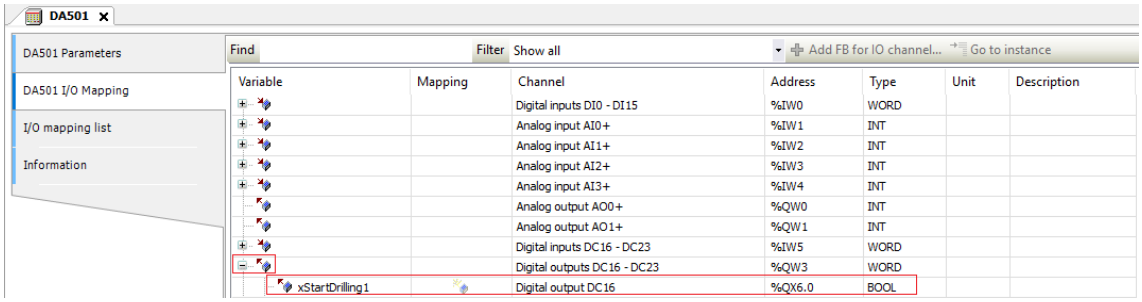
Handle the digital input variables



1. Open the list of the digital inputs.
2. Fill in the variable names:

Channel	Type	Variable
Digital input DI8	BOOL	xDI_08_DA501_I1

Handle the digital output variables



1. Open the list of the digital outputs.
2. Fill in the variable names:

Channel	Type	Variable
Digital output DC16	BOOL	xStartDrilling1

Programming and compiling

You write the program code in a separate IEC 61131-3 editor (CODESYS). You can open CODESYS out of Automation Builder.

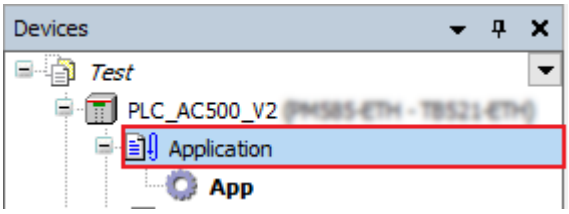
Supported programming languages:

- ST (Structured Text)
- IL (Instruction List)
- FBD (Function Block Diagram)
- LD (Ladder Diagram)
- SFC (Sequential Function Chart)
- CFC (Continuous Function Chart)

Starting the IEC 61131 programm editor CODESYS

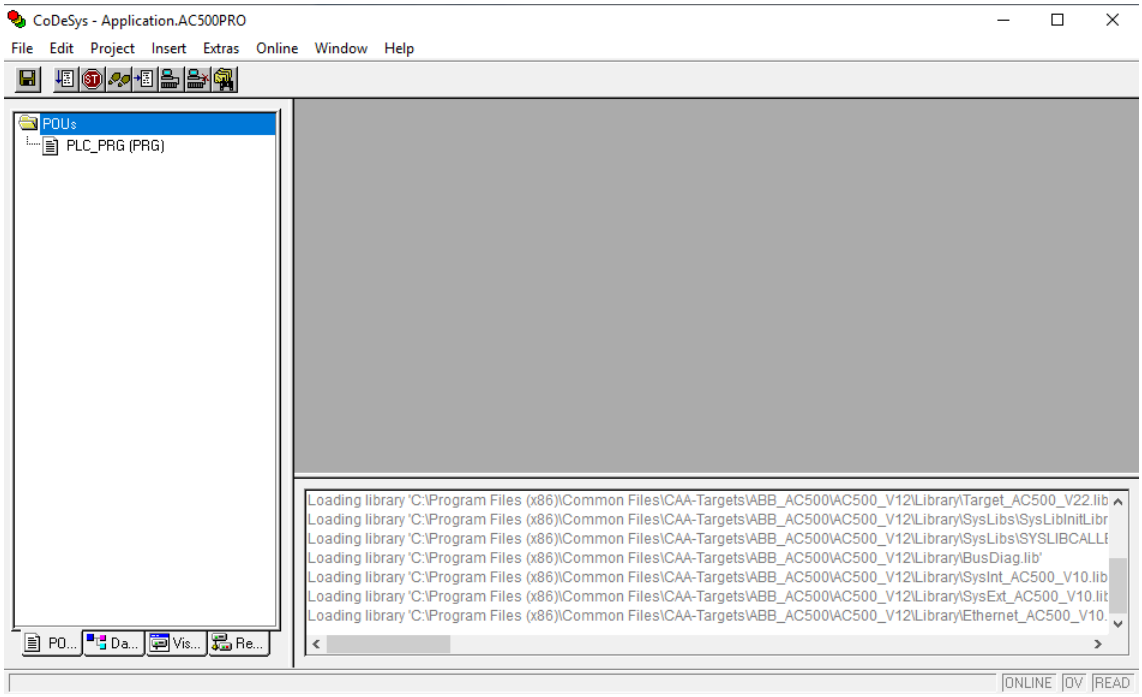
To start the IEC 61131 programm editor CODESYS:

☑ Open an AC500 V2 project in Automation Builder



▷ In the Automation Builder device tree double-click *“Application”*

⇨ This will start the IEC 61131 programm editor CODESYS



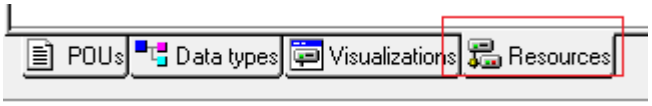
Task configuration

A task is a time unit in the processing of a user program (IEC application), which defines by parameters the way and the speed the CPU is executing the user program.

For this project you will use only one cycling task.

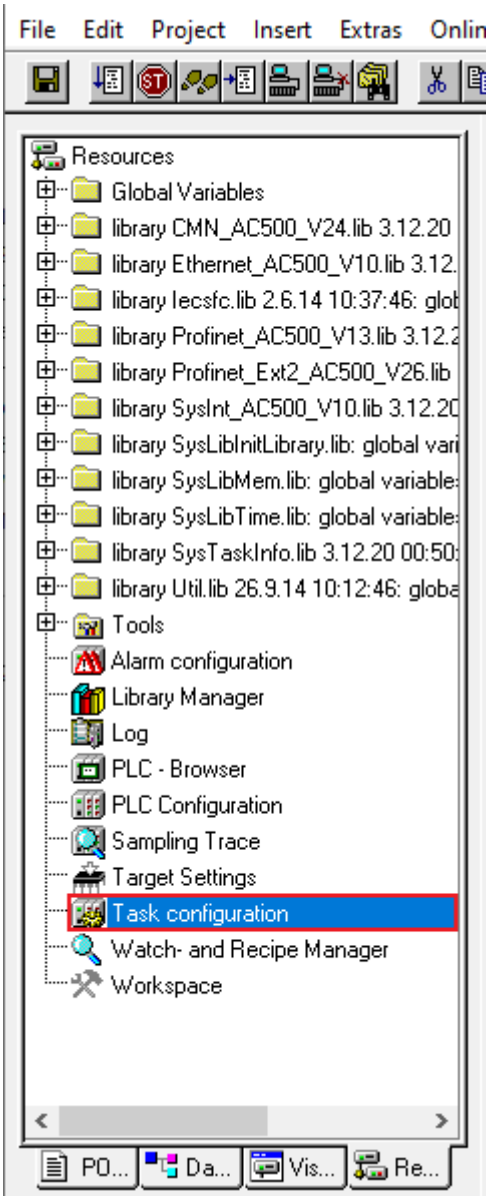
☑ Open CODESYS editor ↳ *Chapter 1.2.18.1.2.4.1 “Starting the IEC 61131 programm editor CODESYS” on page 77*

1. In the CODESYS editor menu select the *“Resources”* tab.

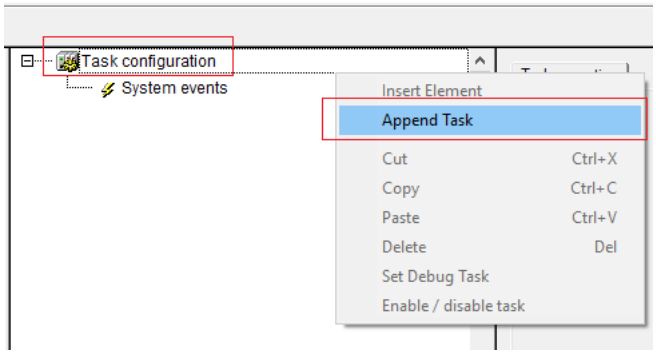


2. Double click *“Task configuration”*.

⇨ The Task configuration window opens.

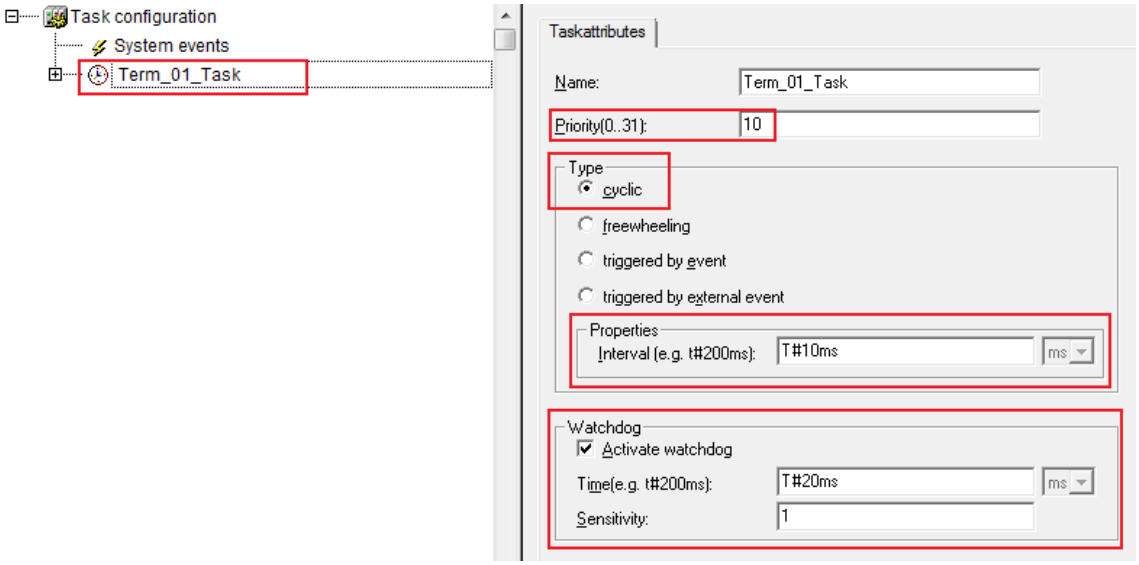


3. Right-click on *"Task configuration"*.
4. Select *"Append Task"*.



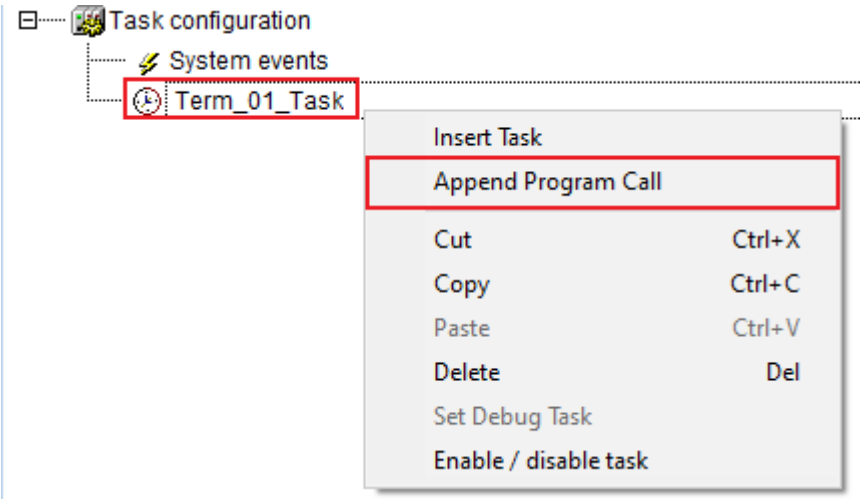
5. Enter a name.
6. Set *"Priority"* to 15
7. Set *"Type"* to *"cyclic"*.
8. Set *"Interval"* to *"T#10ms"*.

9. Activate Watchdog.
10. Set *“Time”* to *“T#20ms”*.

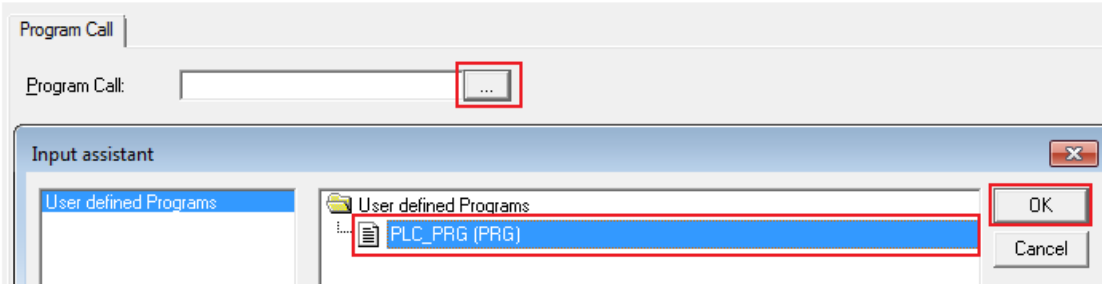


Priority type interval	This is how the CPU prioritizes the task, when more than one task is defined. Type In the CPU you can run tasks dependent on the demands of the process Interval For cyclic tasks you can set the cyclical execution time. It is usually set in milliseconds with IEC time syntax
Watchdog calls	To keep track of the time it takes to complete the task Calls You can call in one or more program POUs in one single task

11. Right click the watch icon next to *“Term_01_Task”*.
12. Select *“Append Program Call”*.

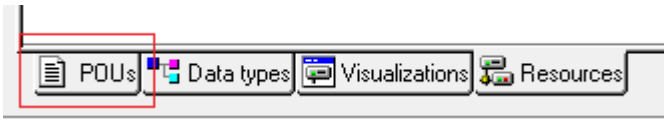


13. Select *“[…]”*.
    ⇨ The input assistant opens.
14. Select *“PLC_PRG (PRG)”*.
15. Select *“OK”*.
    ⇨ The task has been appended.

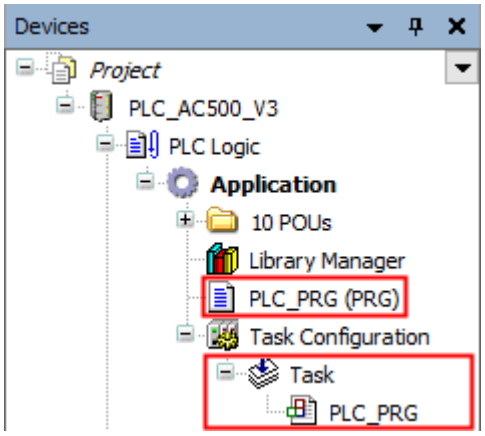


Main program PLC_PRG

In CODESYS menu select tab POUs, there is one call of a POU (program organization unit) i.e. "PLC_PRG".



In your project the "PLC_PRG" will become a main program containing calls to other programs (POUs) which you will create one by one.



The PLC_PRG POU has been defined by default in ST (Structured Text) editor. Keep this setting because of good visibility of the instructions at a glance and good handling for trouble-shooting.

Boolean logic "NOT"

Application example "driller"

Recognizing of a driller by a photo sensor. "TRUE" input signal from sensor indicates that a driller is broken. If driller has been found correct, then start drilling.

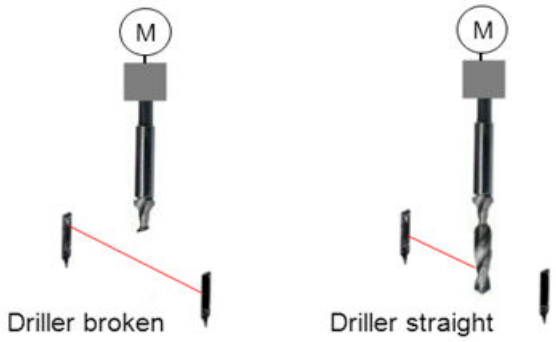


Table 3: Required behavior

Signal from photo sensor	Required signal of motor ON
FALSE	TRUE
TRUE	FALSE

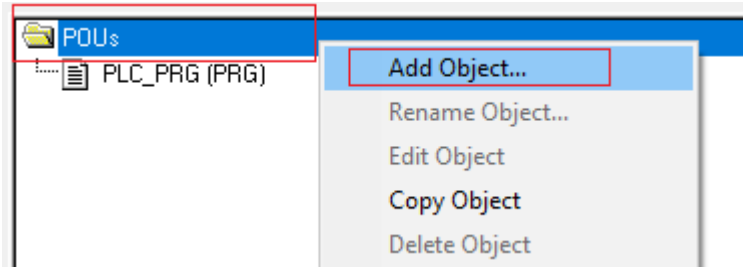
Table 4: Hardware set-up

Element	HW channel	Symbol	Description
Switch I1	DA501 DI8	xDI_08_DA501_I1	Photo sensor
LED output DC16	DA501 DC16	xStartDrilling1	Motor on

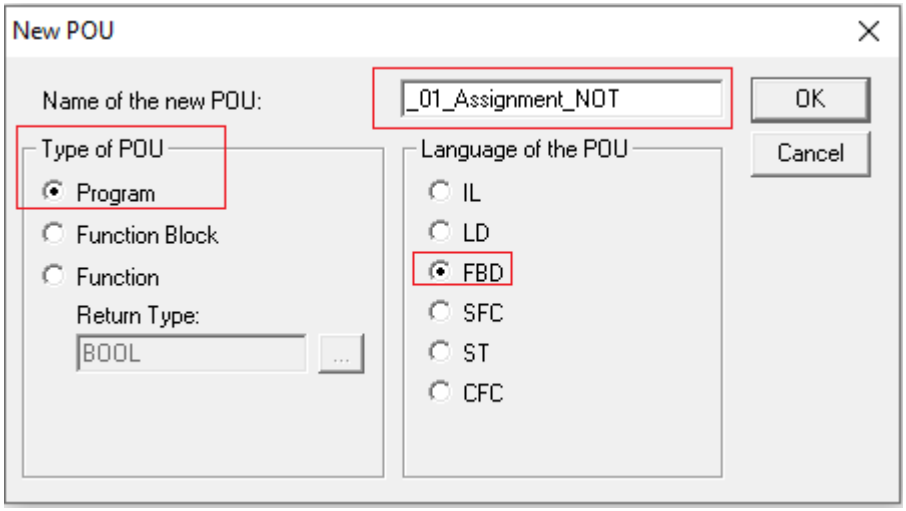
Implementation

Create a new program POU in the project

1. In the CODESYS menu select POUs
2. Select *“Add object”*

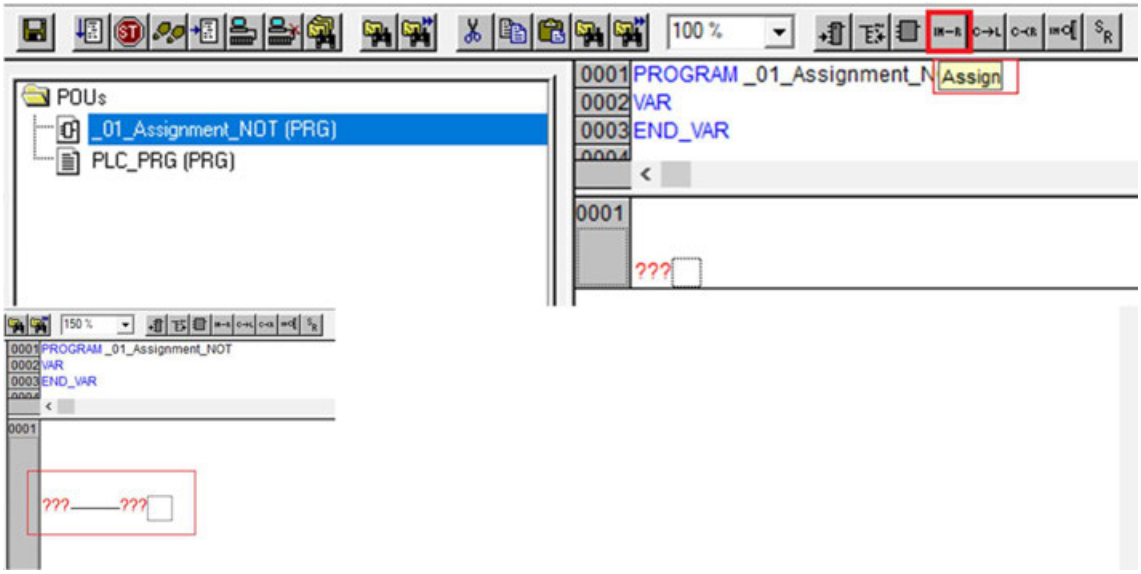


3. Enter *“_01_Assignment_NOT”*.
4. Set *“Type of POU”* to *“Program”*.
5. Set *“Language of the POU”* to Function Block Diagram *“FBD”*.
6. Select *“OK”*.
   ⇨ POU has been added.

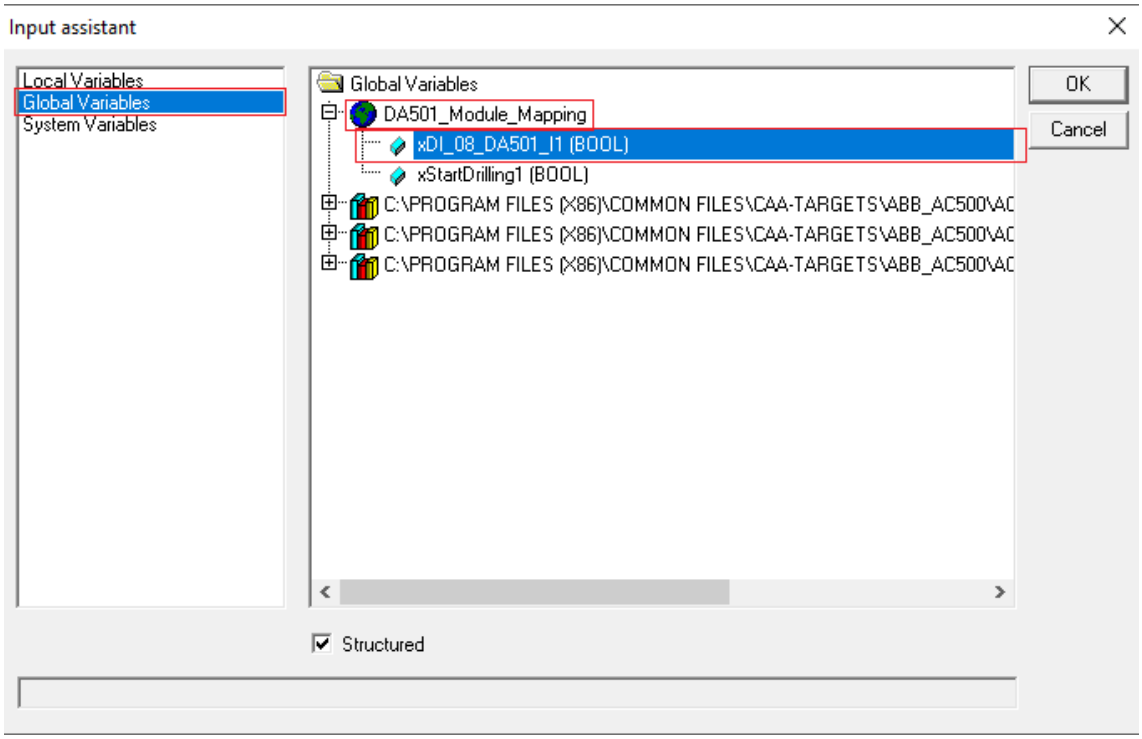


Assign the hardware DI signals to local variables

1. In the CODESYS device tree double-click POU *"_01_Assignement_NOT"*.
2. Click inside of the first Network.
3. Select *"Assign"* from Tools.

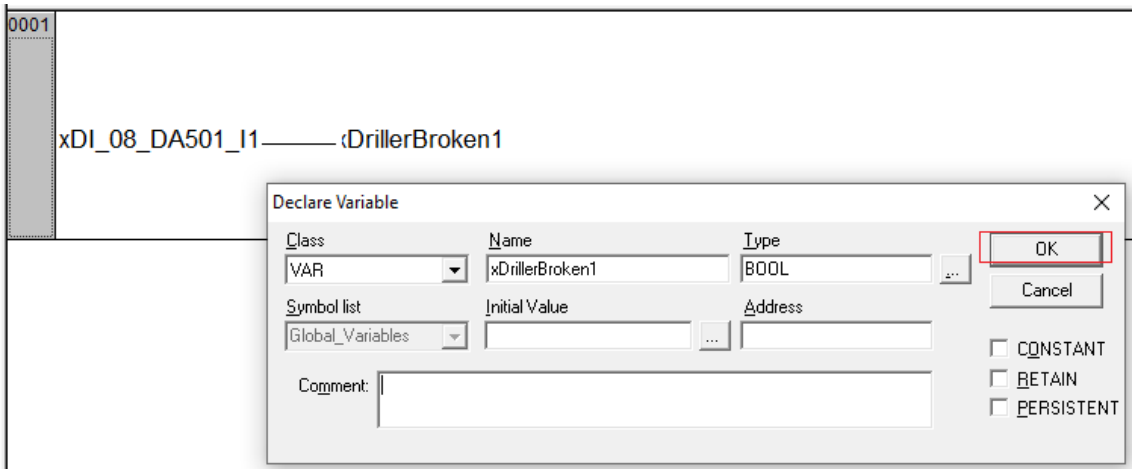


4. Select *"???"* on the left side of the assignment and press *[F2]*.
   ⇨ *"Input Assistant"* opens.
5. Under *"Global Variables"* open *"DA501_Module_Mapping"*.



6. Select *"xDI_08_DA501_I1"*.
7. Select *"???"* on the right side of the assignment connector.
8. Create a new local variable by typing in *"xDrillerBroken1"* which will replace the *"???"*.
9. Press *[Enter]*.

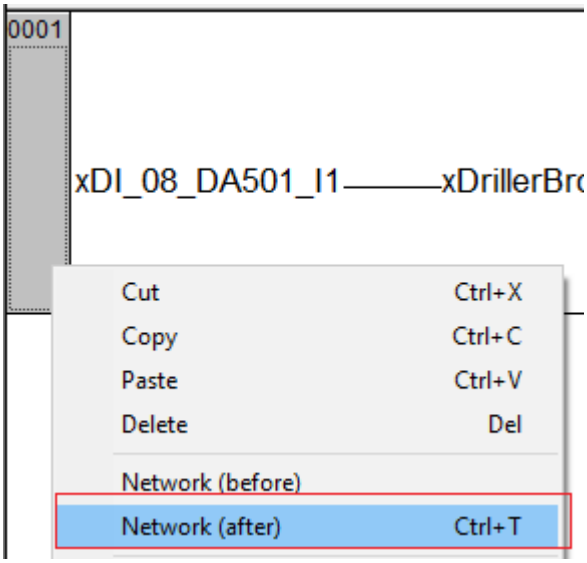
   ⇨ *"Declare Variable"* opens. You see the written variable name and the data type BOOL. The scope is "VAR". It means it is a local variable within this POU.



10. Select *"[OK]"* to accept the entries.

11. Right-click on network 1 and select Network (after)

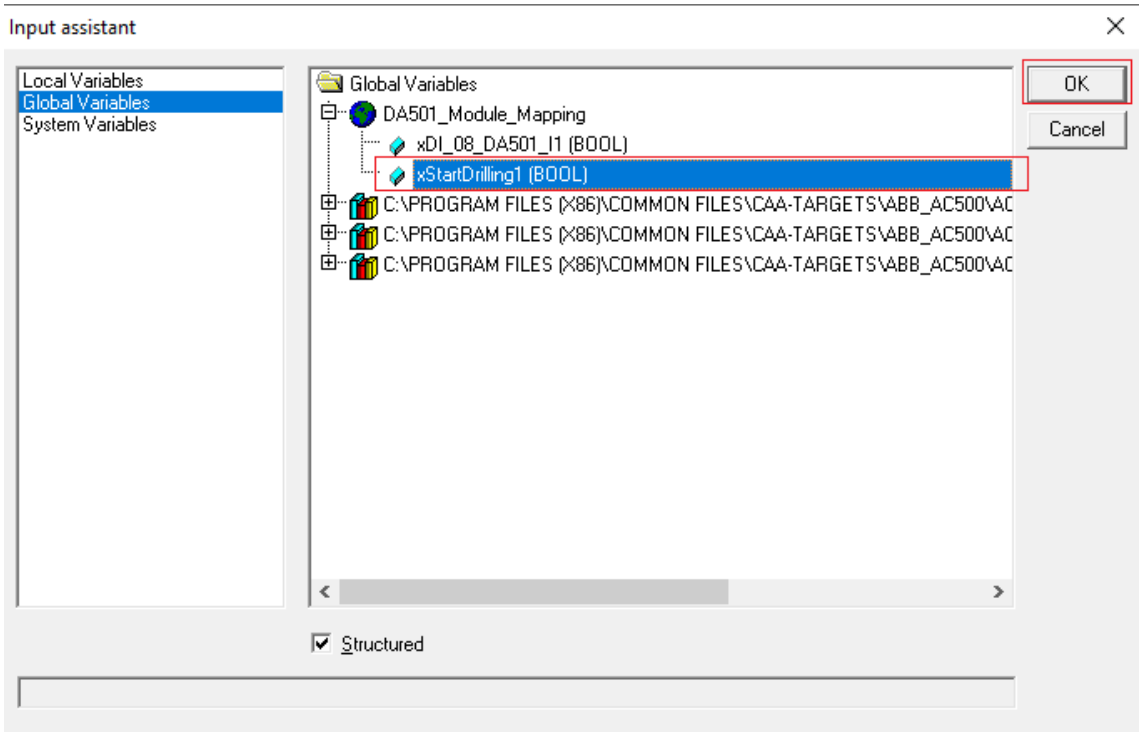
    ⇨ You added a network *“2”* below network 1.



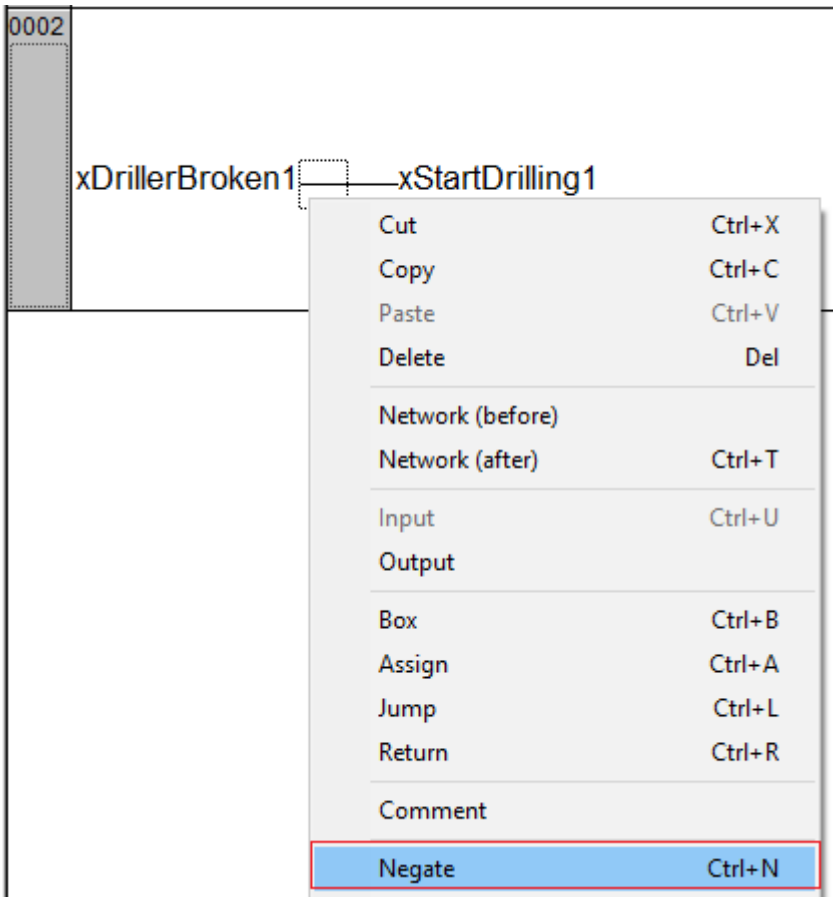
Add assignments and a Boolean NOT to the DO signals

1. Add an assingment from the Tools.
2. Type in or copy & paste "xDrillerBroken1" to the left side of the instruction line.
3. Select *“???”* on the right side of the instruction line, then press F2.

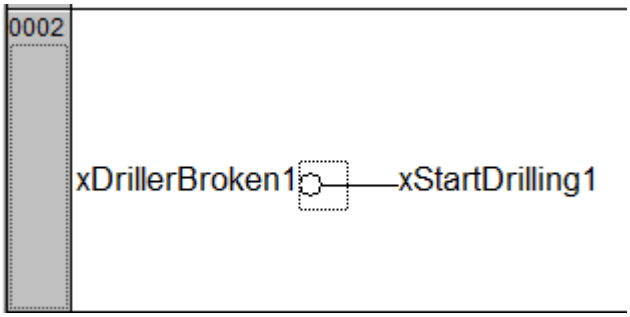
    ⇨ *“Input Assistant”* opens.
4. Under “Global Variables” open “DA501_Module_Mapping”
5. Select *“xStartDrilling1”*
6. Select *“[OK]”* to close the dialog.



7. Right-click the left side of assignment PIN.

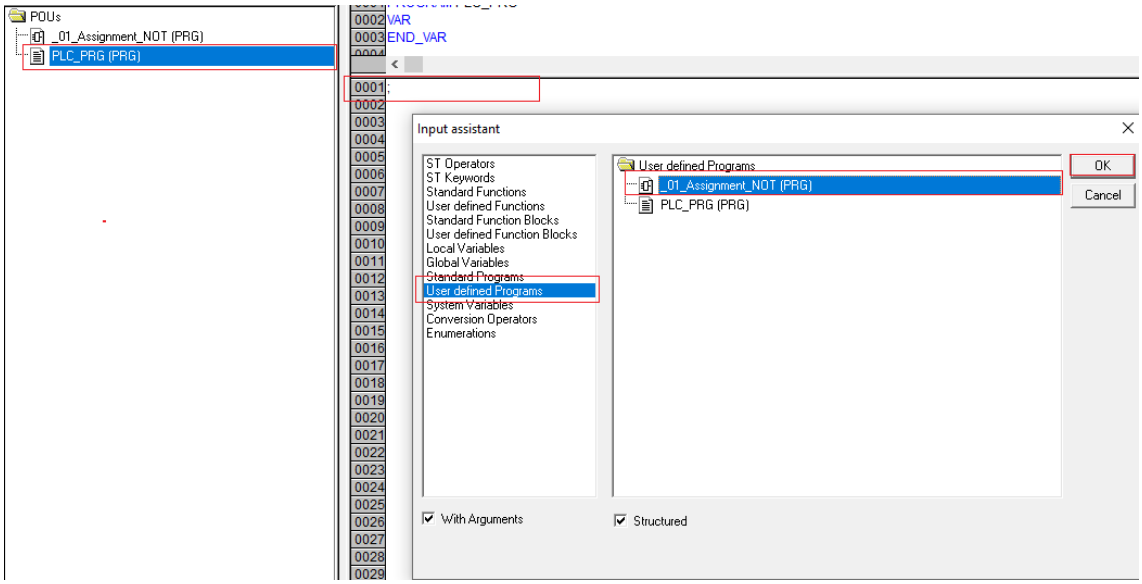


8. Select *“Negate”* to add a negation to the assignment



Call the POU in the PLC_PRG

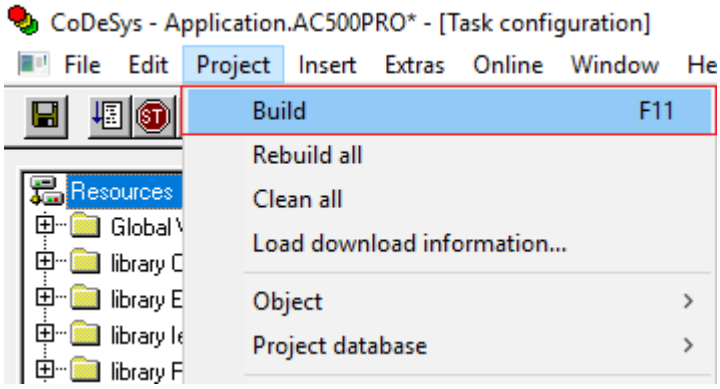
1. Double-click *“PLC_PRG”*.
2. Select the first line in *“PLC_PRG”* and press *[F2]*
   ⇨ *“Input Assistant”* opens.



3. Open *“User defined Program”*
4. Select *“_01_Assignment_NOT”*
5. Select *“OK”* to close the dialog

Compile the project

Before logging-in to the CPU, you need to compile the complete code without any errors.

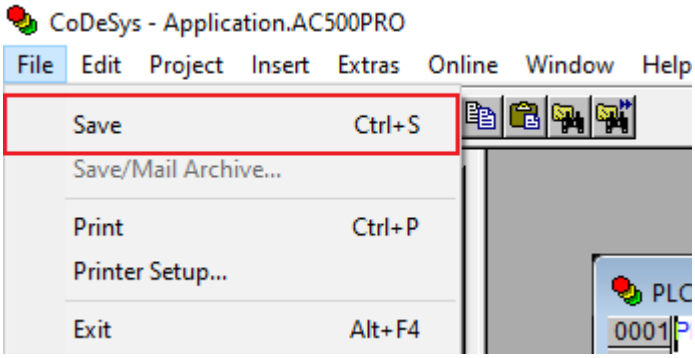


▷ In the CODESYS editor menu select *“Project ➔ Build”*

⇨ The result of the compiling is shown in the *“Messages”* field at the bottom of the screen.

If you skip the compiling and select *“Login”*, the Automation Builder will automatically trigger compiling in advance to logging-in.

Save CODESYS project



▷ In the CODESYS editor menu select *"File ➔ Save"*.

Alternatively, select the save icon in the tool bar.

Alternatively, press *[Ctrl]* + *[S]*.

Set-up the communication gateway

Set-up communication parameters

To set-up the communication between the PC and the PLC, e.g., for downloading the compiled program, you have to set-up the communication parameters.

The IP address of your PC must be in the same class as the IP address of the CPU.

The factory setting of the IP address of the CPU is 192.168.0.10.

The IP address of your PC should be 192.168.0.X. Avoid X = 10 in order to prevent an IP conflict with the CPU.

Subnet mask should be 255.255.255.0.

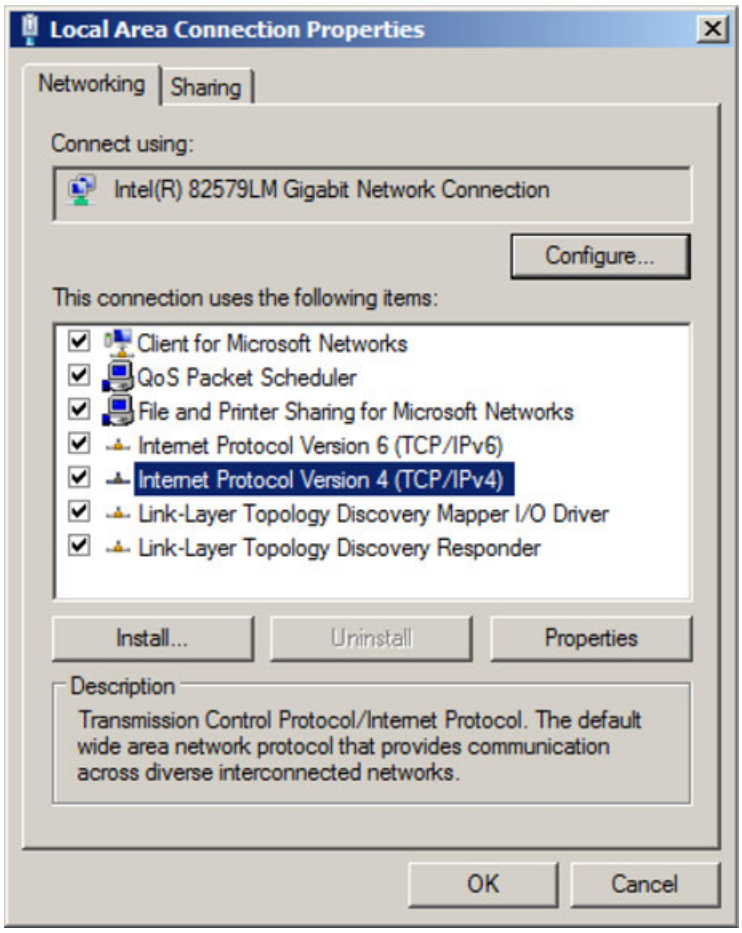
Change the IP address

1. Open Windows **Control Panel**. Click *"Network and Internet ➔ Network and Sharing Center"*.
2. Click **Change adapter settings**.

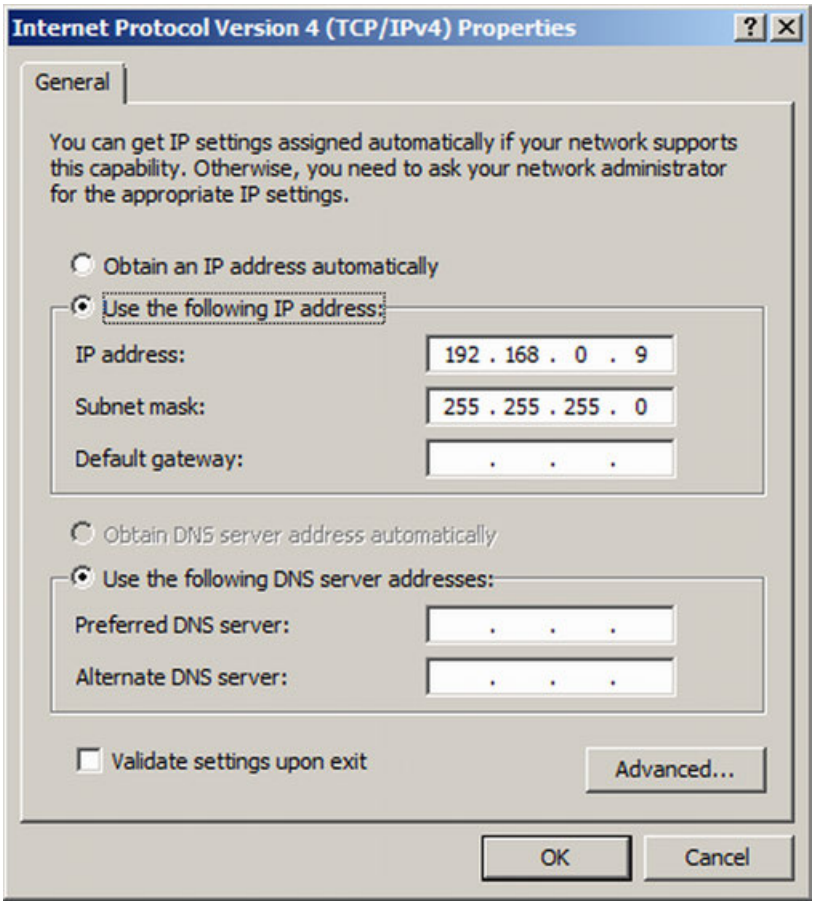
⇨

> *If using existing network with several devices, please pay attention on given network rules or contact your system administrator.*

3. Right-click **Local Area Connection (Ethernet)** and select **Properties**.



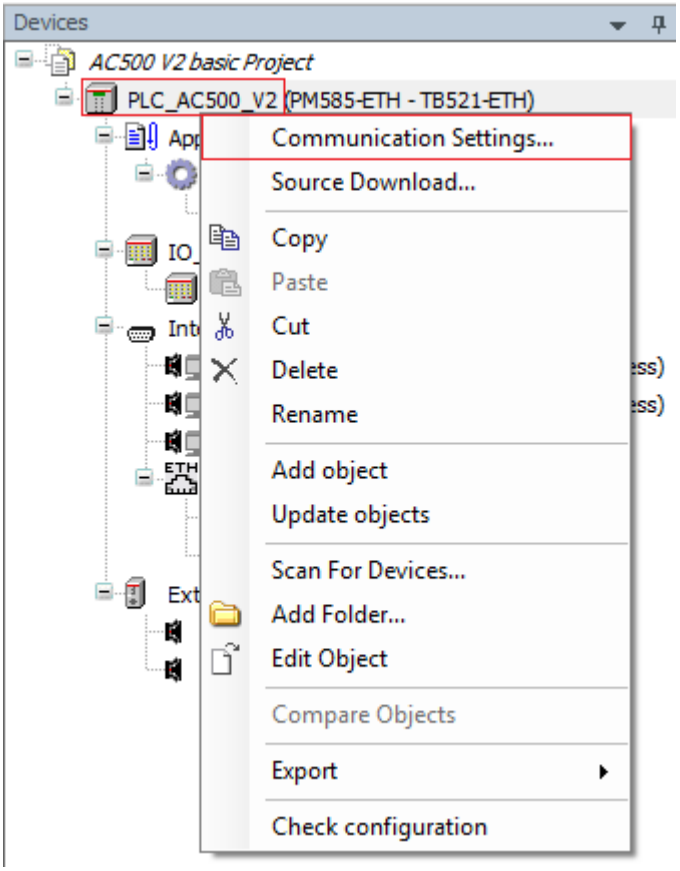
4. Double-click **Internet Protocol Version 4 (TCP/IPv4)**.



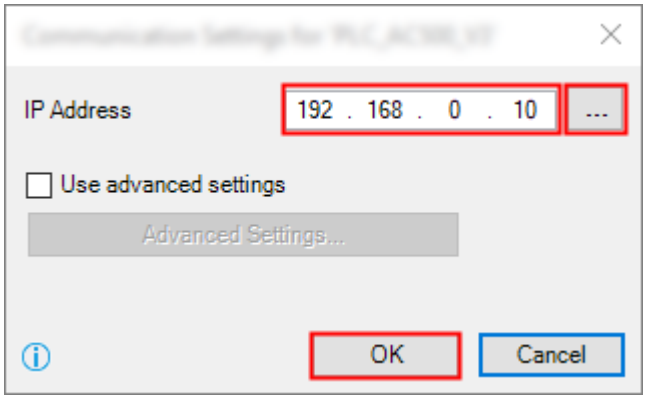
5. Enter your desired IP address and subnet mask.

Set-up the communication gateway

☑ CPU and PC are connected with an Ethernet cable.

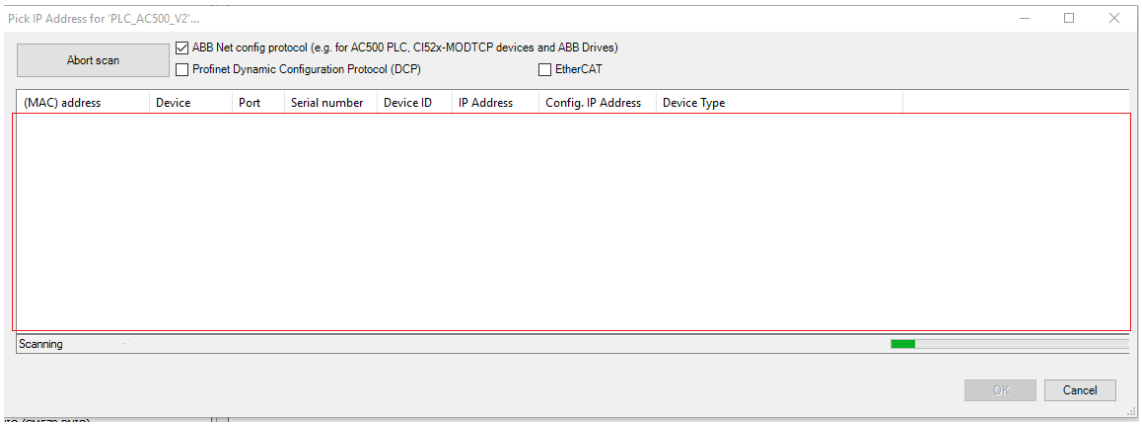


1. In the Automation Builder device tree right-click *"PLC_AC500_V2"*.
2. Select *"Communication Settings"*.



3. Keep the default value in the IP address of the CPU or type in the current IP address, if differs.

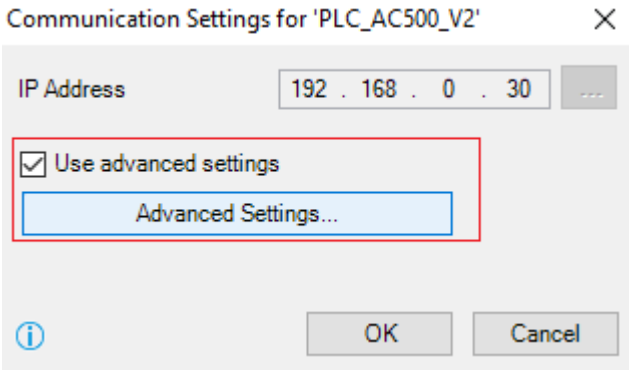
> *The standard (default) IP address of the port ETH1 is: 192.168.0.10*
> *The standard (default) IP address of the port ETH2 is: 192.168.1.10*

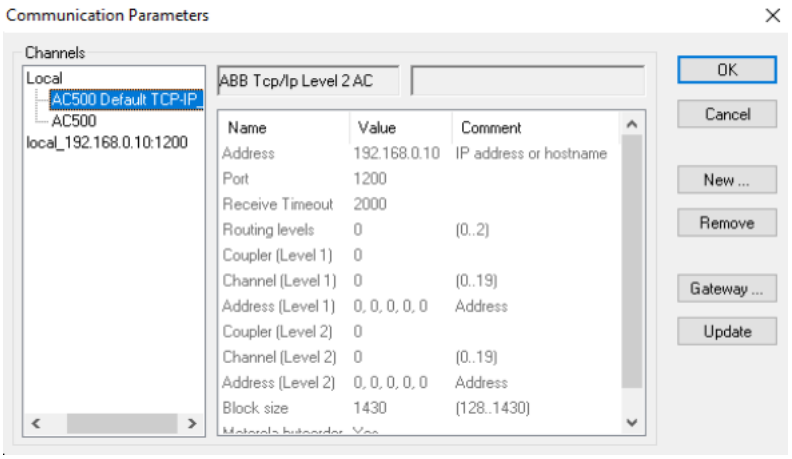


4. Select *"OK"* to implement the IP address.

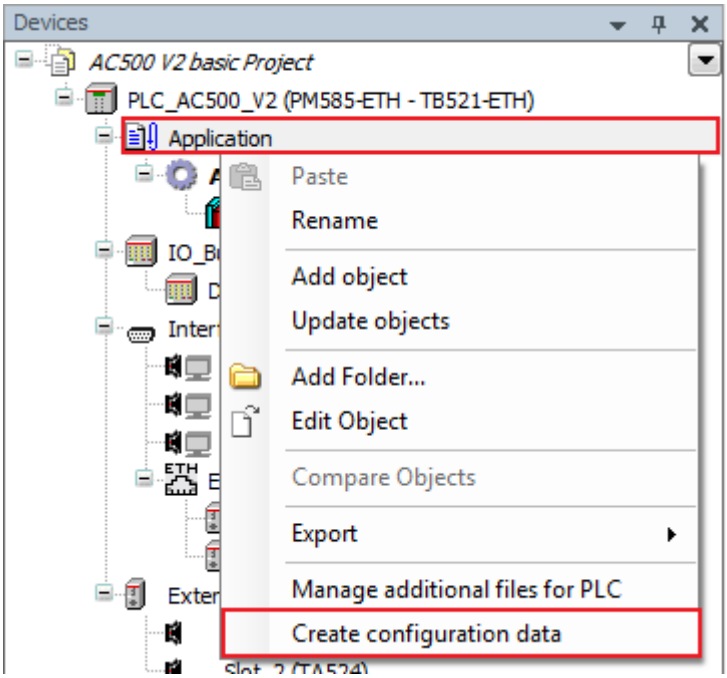
Check communication settings

If you need to check the communications settings or if you want to see more information about the current selected CPU.



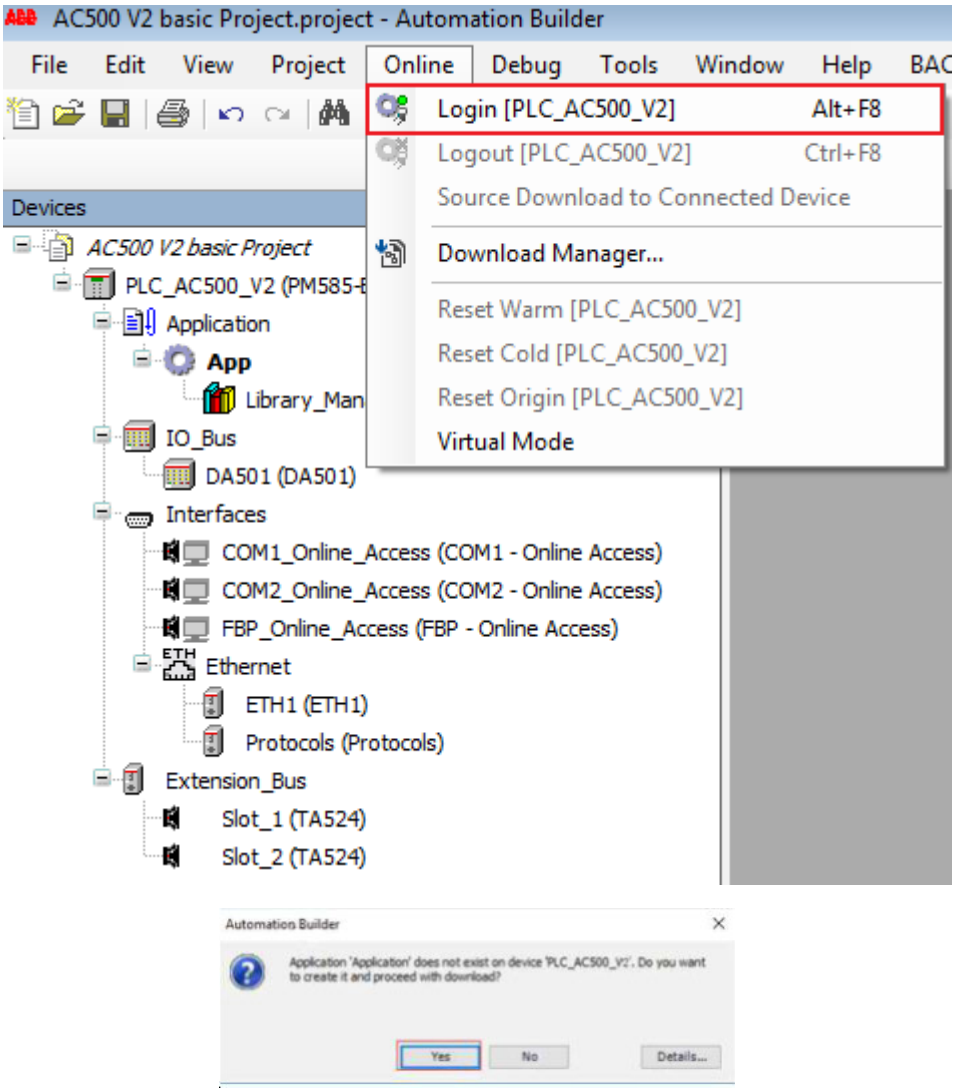


After changing the IP Address either double click the Application or right-click and *"Create Configuration Data"*



Log-in to CPU and download the program

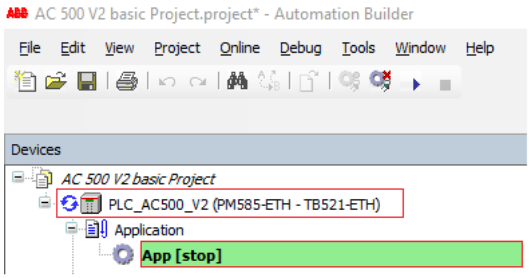
Logging-in to the CPU will load the project into the AC500 V2 CPU. The first log-in will also load the hardware set-up.



1. In the Automation Builder menu select *“Online ➔ Login [PLC_AC500_V2]”*.

   ⇨ A pop-up will appear.

2. Select *“Yes”* to download the application to the AC500V2 CPU.



   ⇨ PLC is in "stop" mode.

3. Start the PLC ↳ *Chapter 1.2.18.1.2.7.1 “Start the program execution” on page 94*.

> *Generally, if the CPU is in RUN mode, i.e. in program execution mode, a download will always cause the mode change to "stop". In stop mode the CPU is not controlling the system!*
>
> *Always, after selecting the "Login" command, read carefully the dialog box text to ensure that you are aware of the CPU's behavior after the command confirmation.*

By default, a download generates following actions in the CPU:

- The project is stored in the RAM memory.
- The project is stored in the flash EEPROM, if boot application was created.

Test the program

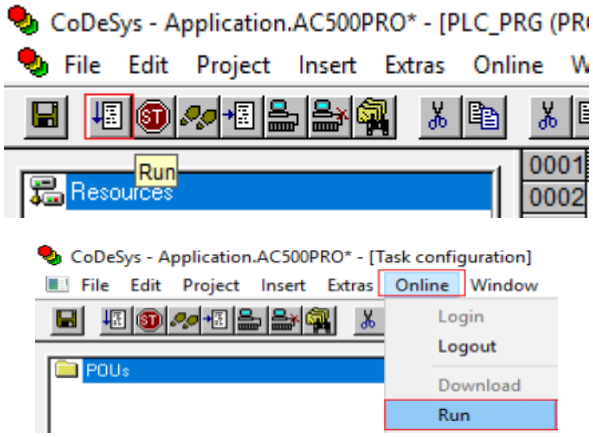
Start the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.

☑ Open CODESYS *Chapter 1.2.18.1.2.8.1 "Starting the IEC 61131 programm editor CODESYS" on page 97*.



▷ In the CODESYS editor menu select *"Online ➔ Run"*

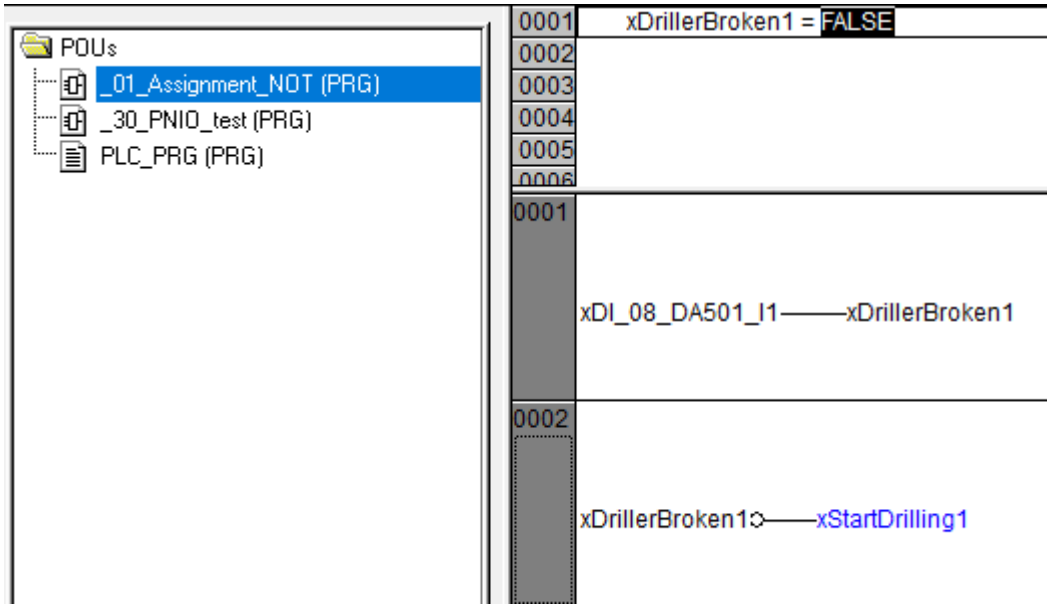
Alternatively, select the "run" icon in the tool bar.

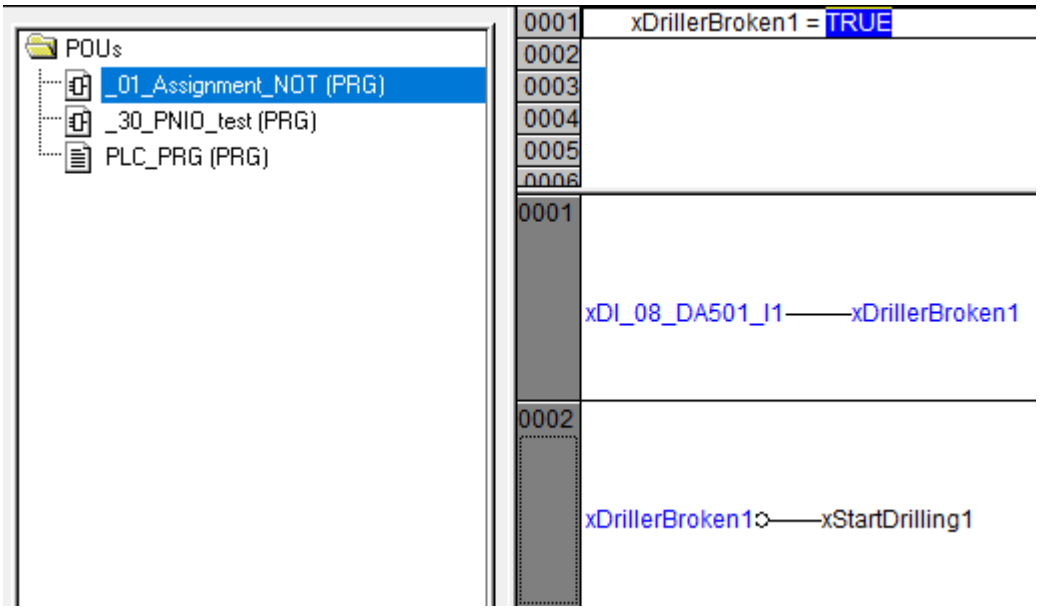
Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I1 and in the CODESYS editor observe:

- The online status of inputs and outputs within the POU.



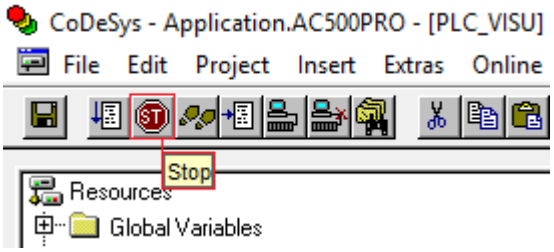


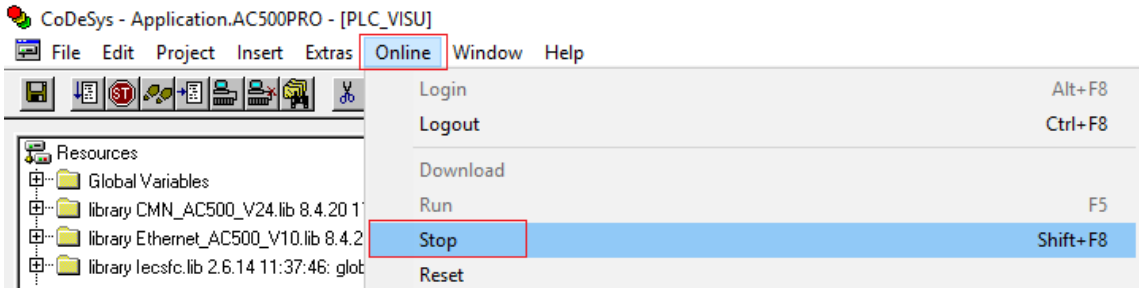
Stop the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in RUN mode.





▷ In the CODESYS editor menu select *“"Online ➔ Stop [PLC_AC500_V2]"”*

Alternatively, select the "stop” icon in the tool bar.

Alternatively, press *[Shift]* + *[F8]*.

Set-up visualization

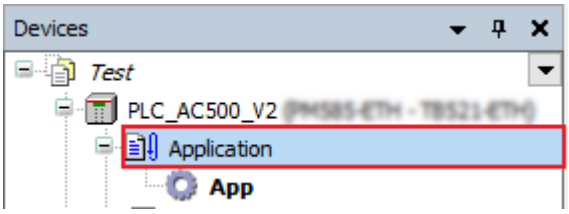
The visualization allows designing a graphical representation of project variables. In online mode, the graphical elements can change, for example, their color, size or position according to the actual variable status.

Visualization for your project is done via CODESYS editor.

Starting the IEC 61131 programm editor CODESYS

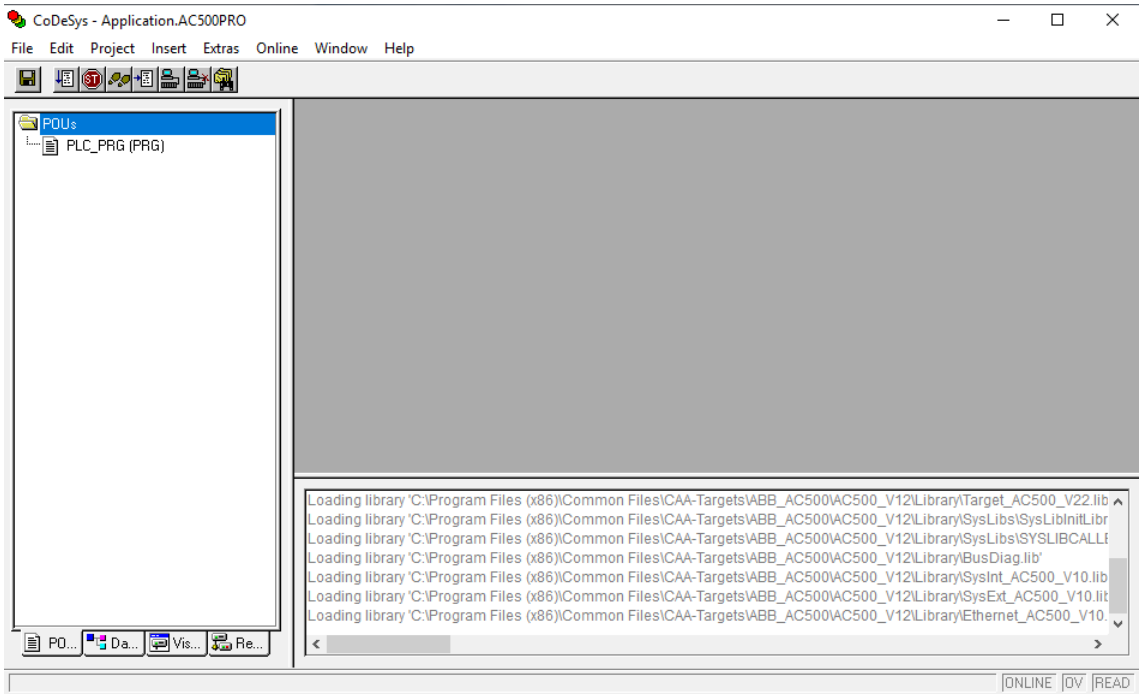
To start the IEC 61131 programm editor CODESYS:

☑ Open an AC500 V2 project in Automation Builder



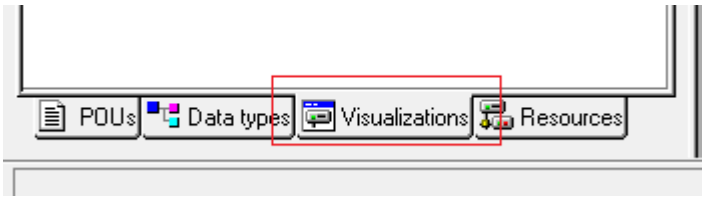
▷ In the Automation Builder device tree double-click *“Application”*

⇨ This will start the IEC 61131 programm editor CODESYS

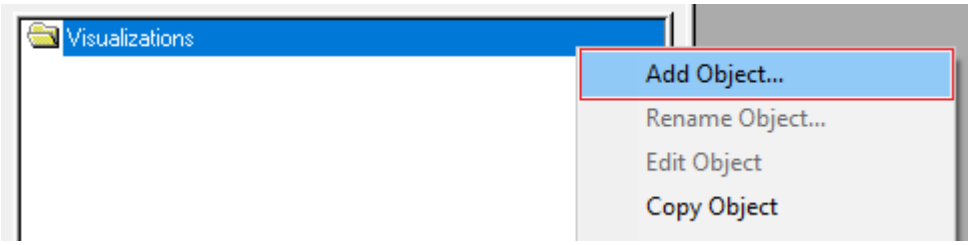


Insert visualization object

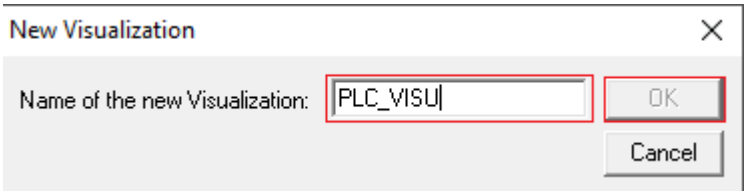
1. In the CODESYS menu select *“Visualization”*



2. Right-click Visualizations.
3. Select *“Add object”*.



4. Type in *“PLC_VISU”*
5. Select *“OK”* to add Visualization

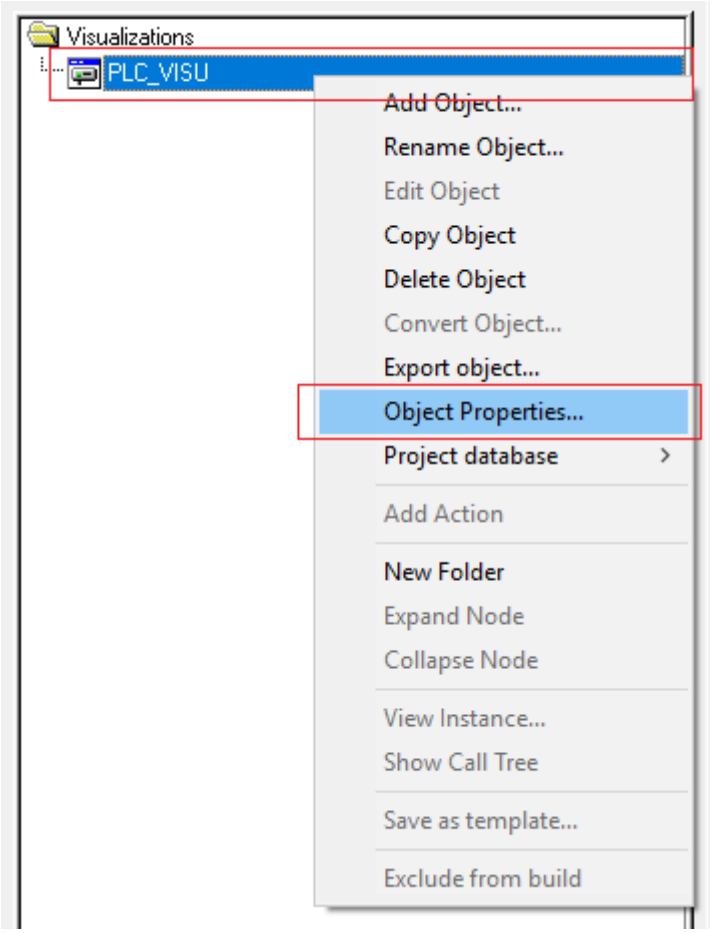


⇨ The new visualization object is inserted.

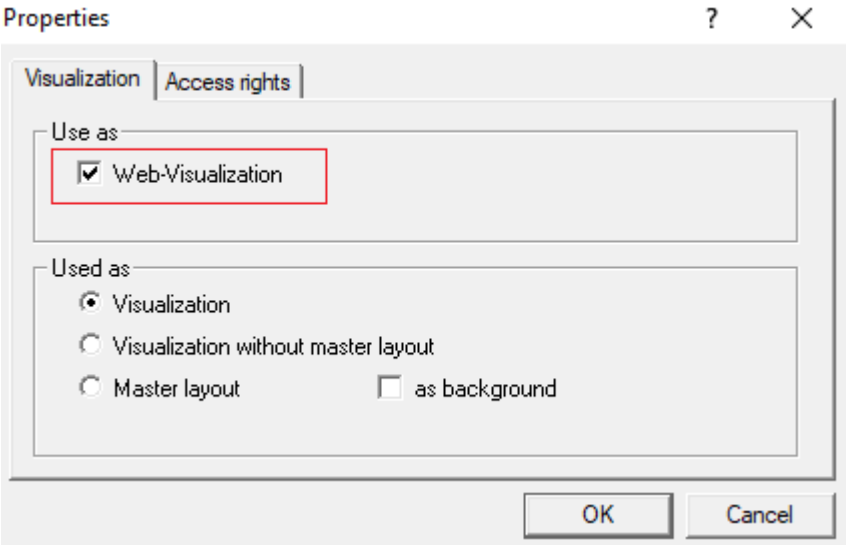
The name "PLC_VISU" has been chosen, because it is the default name for a home screen in a web visualization. If you have more than one visualization object in your project, it will be useful to choose another name, e.g. "_01_Assignment_NOT_v" and to choose "PLC_VISU" as a home screen to access all available visualization screens. The name of a visualization object can be modified afterwards.

Creating and configuring of visualization

1. In the CODESYS device tree right-click *“PLC_VISU”*
2. Select *“Object Properties”*



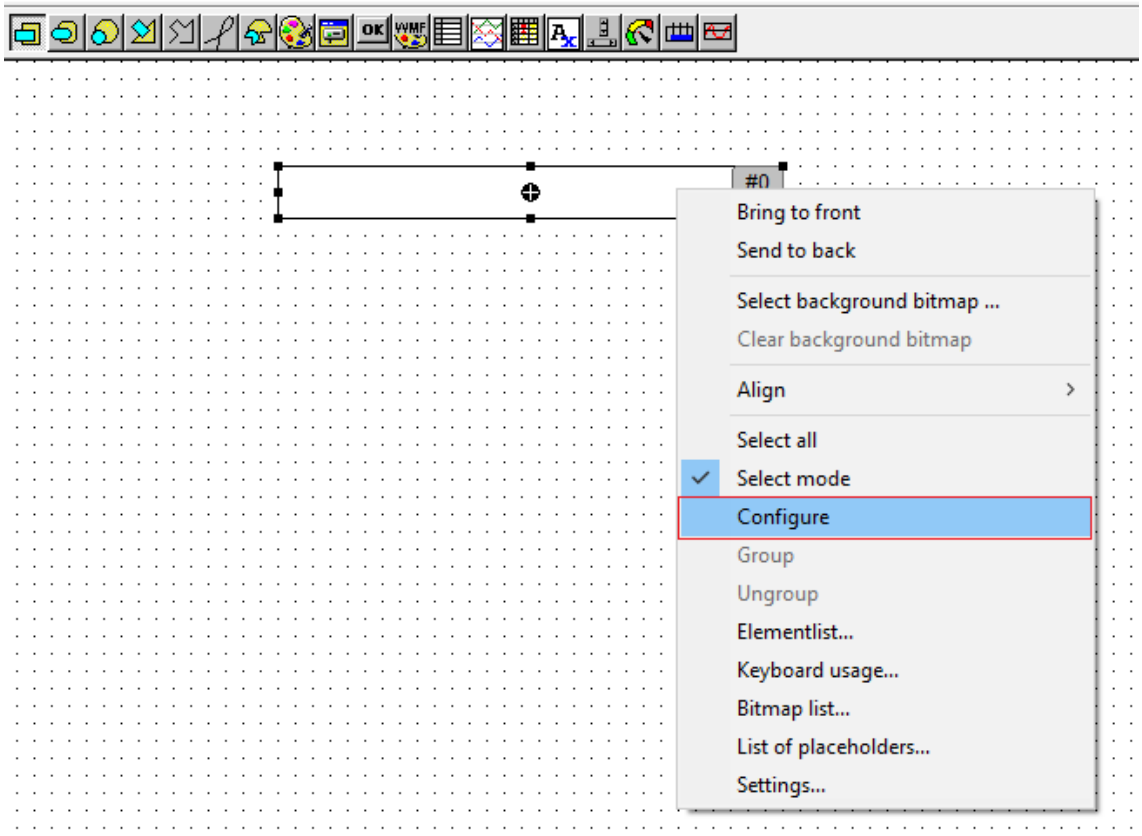
3. Enable *"Web-Visualization"*



Add a screen title

1. In the CODESYS editor toolbar select *“Rectangle”*.

2. Now by dragging your mouse anywhere on the Visualization you can create a rectangle.



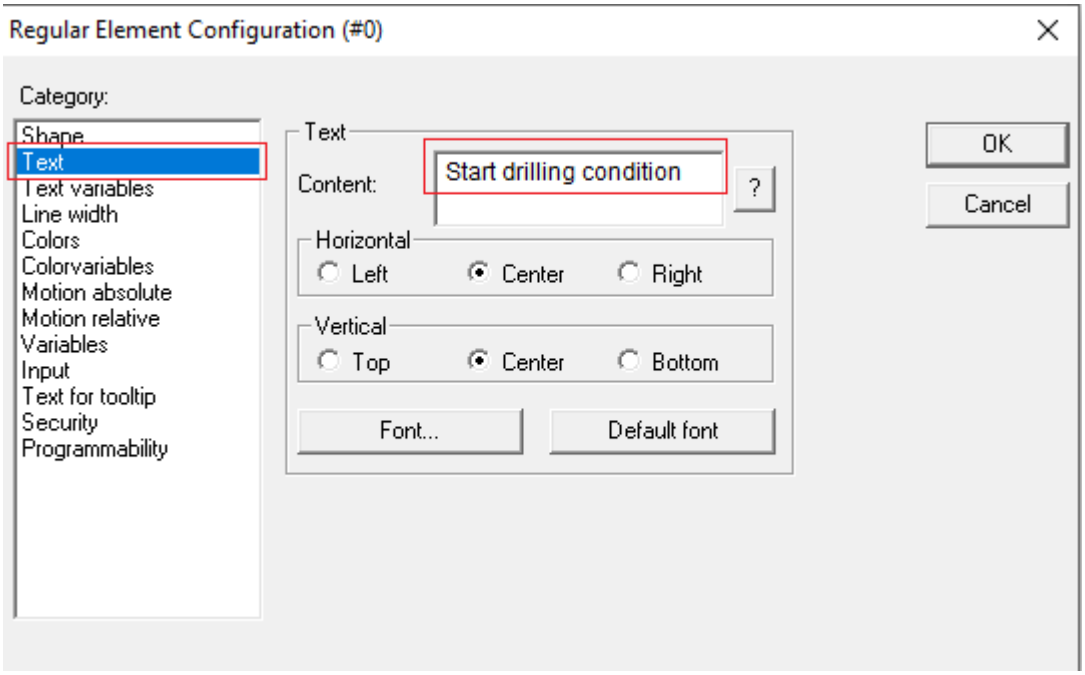
3. Double-click the shape.

   Alternatively right-click and then select *“Configure”*

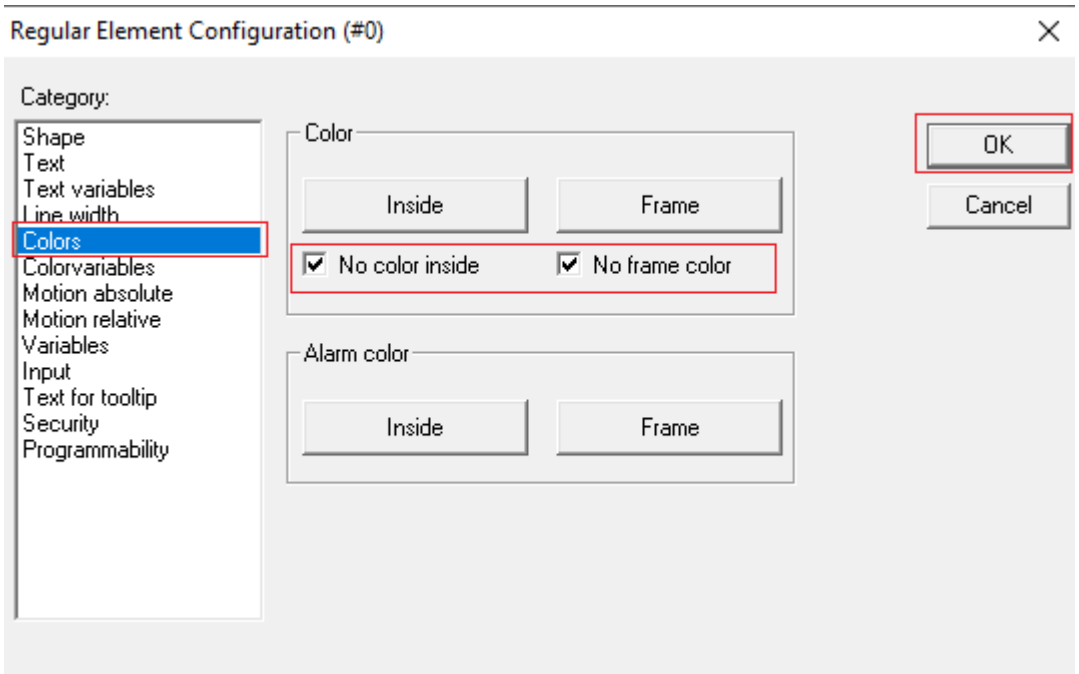
   ⇨ The *“Regular Element Configuration”* window opens.

4. Under *“Category”* select *“Text”*

5. Under *“Content”* type in *“Start drilling condition”*

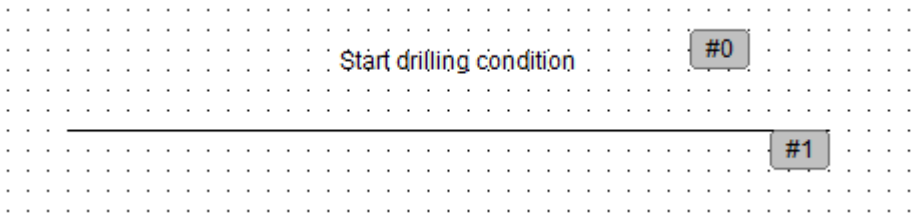


6. Under Category select *“Colors”*.
7. Under *“Color”* enable *“No color inside”* and *“No frame color”* this will help create a cleaner look later on
8. Select *“[OK]”* to implement changes.

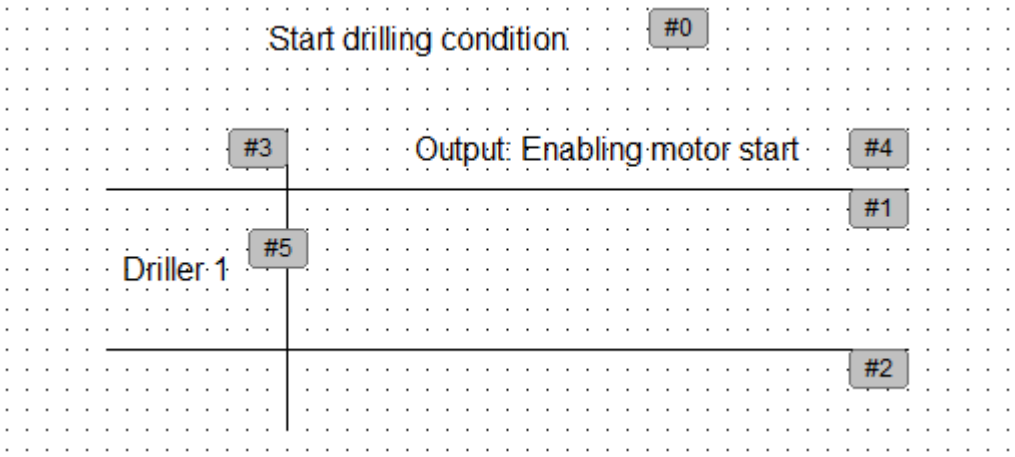


Further lines and labels

1. In the CODESYS editor toolbox select *“Polyline”*
2. Create a line by left-clicking and holding the mouse button. Drag the line to your desired lenght then double-click to end the line.

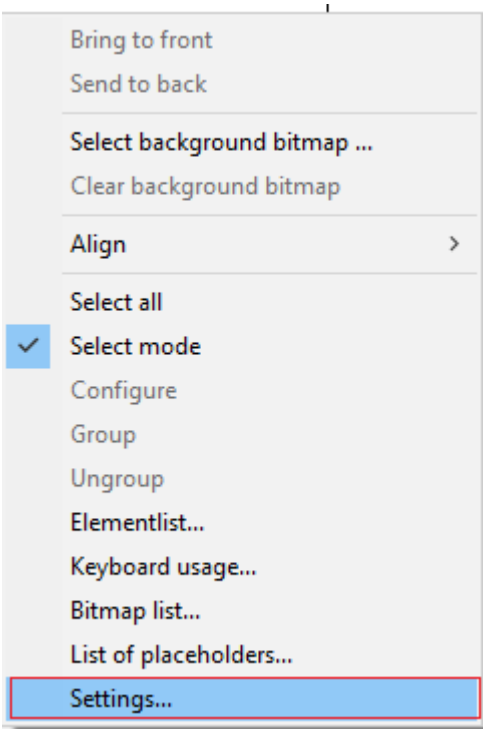


3. Follow the same procedure to create the other shapes and labels.

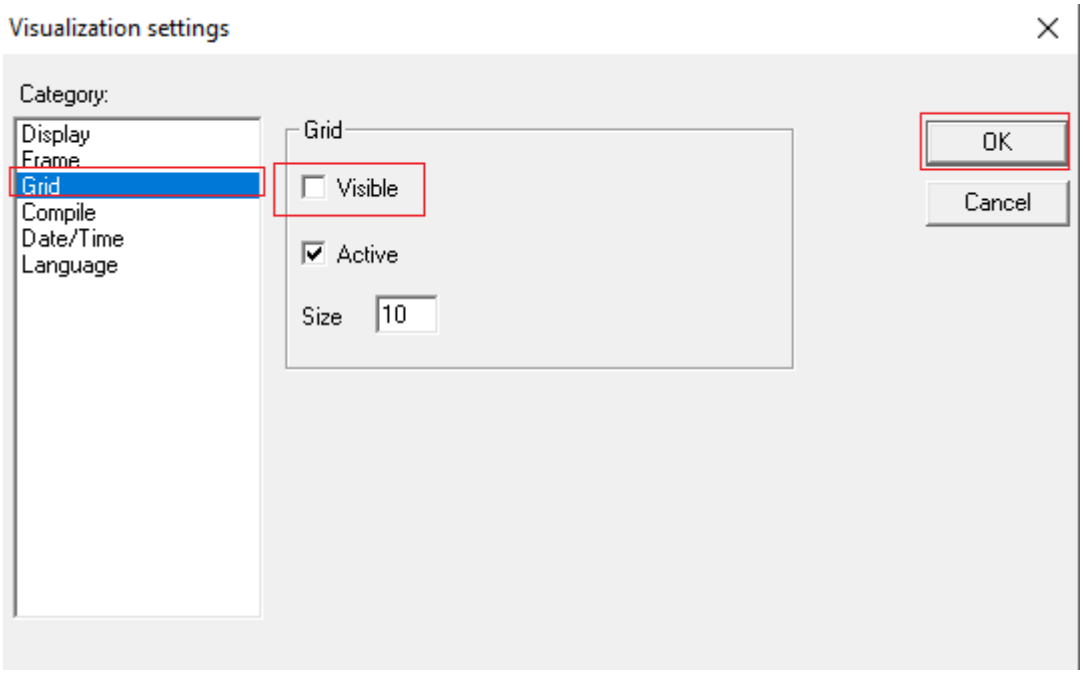


Disable Grid and check Settings

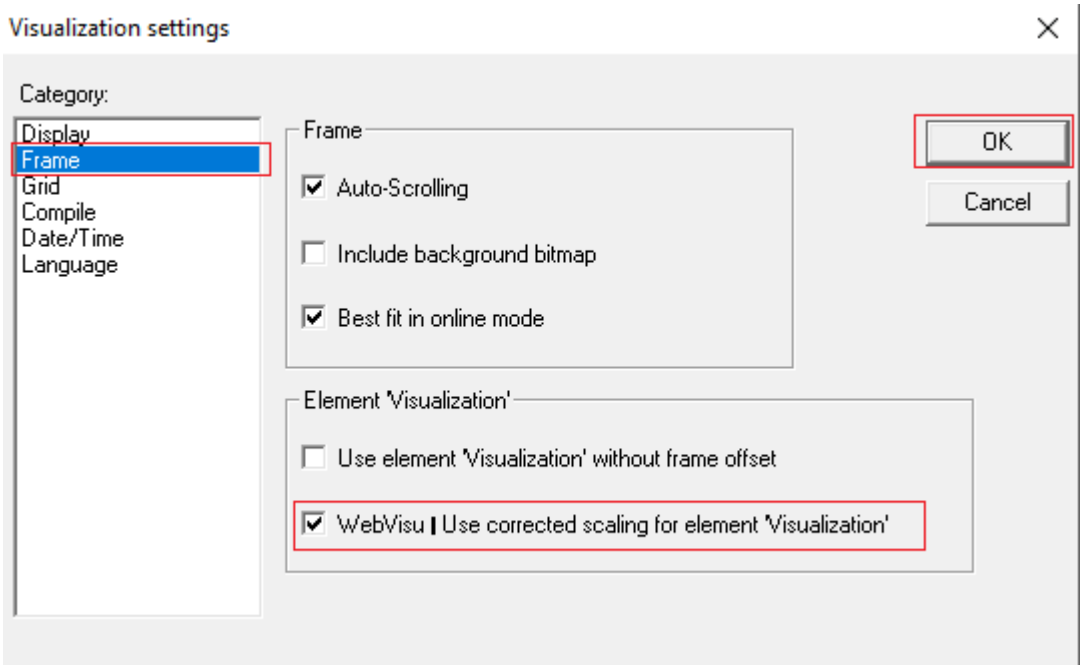
1. In the CODESYS editor right-click anywhere on the visualization and select *“Settings”*.

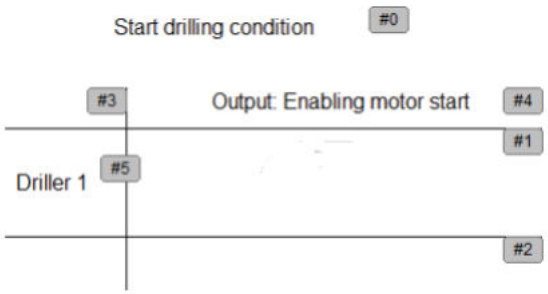


2. Select *“Grid”* and unmark *“Visible”*.



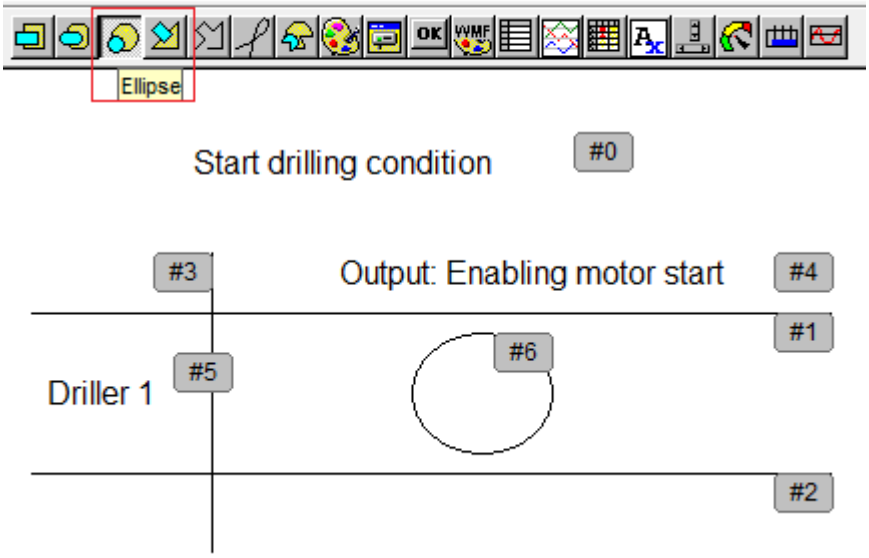
3. Select *"Frame"* and make sure *"WebVisu"* is activated.



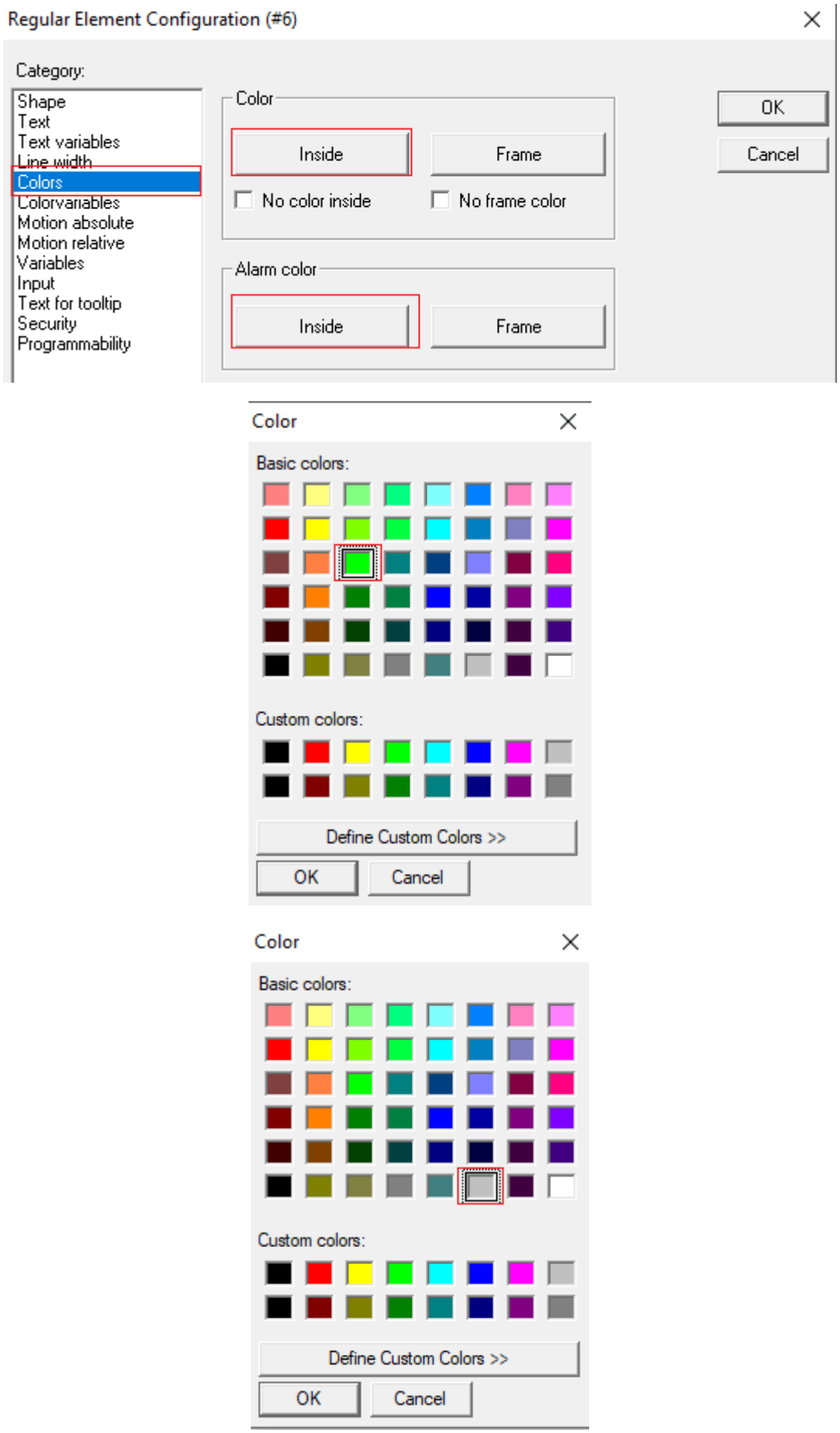


Lamp element for signal indication

1. In the CODESYS editor tool bar select Ellipse and adapt size, if required.
2. Double-click on the ellipse to open the configuration.

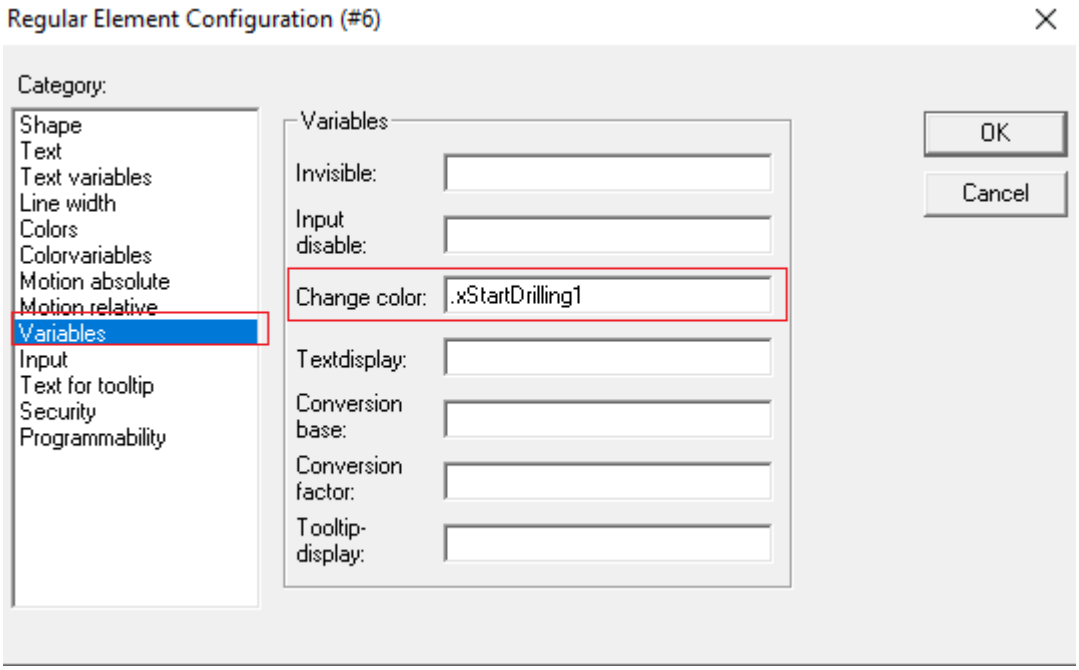


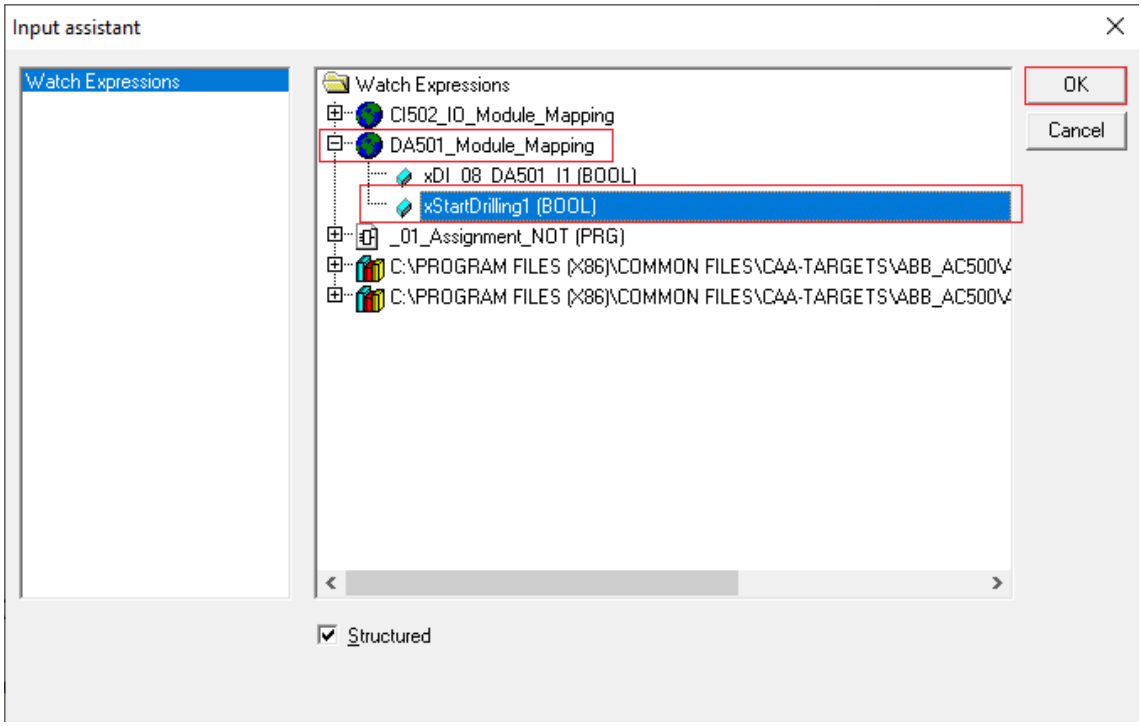
3. Select *“Colors”* and set two different colors for "Color" and "Alarm color".



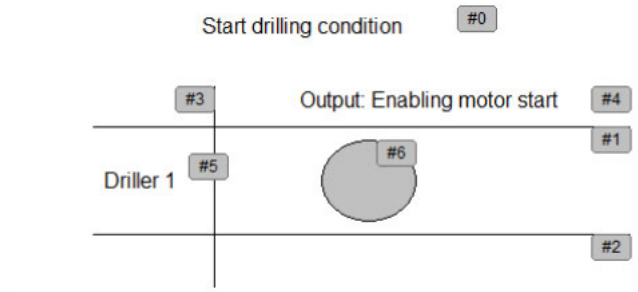
4. Open Variables and left-click *"Change color"*.

5. Press *[F2]*, this will open the *"Input assistant"*.



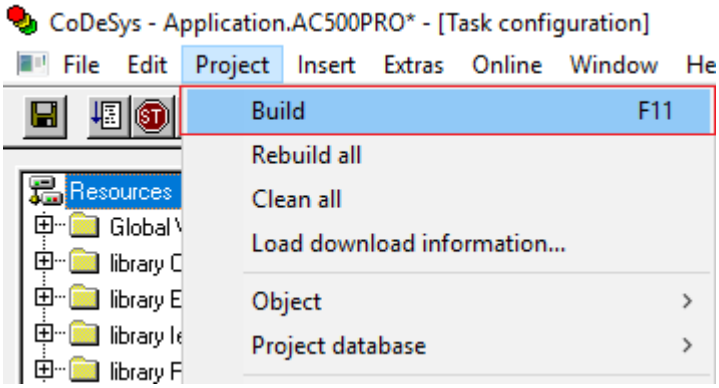


6. In *"DA501_Module_Mapping"* select *"xStartDrilling1"*
7. Select *"[OK]"*



Compile the project

Before logging-in to the CPU, you need to compile the complete code without any errors.

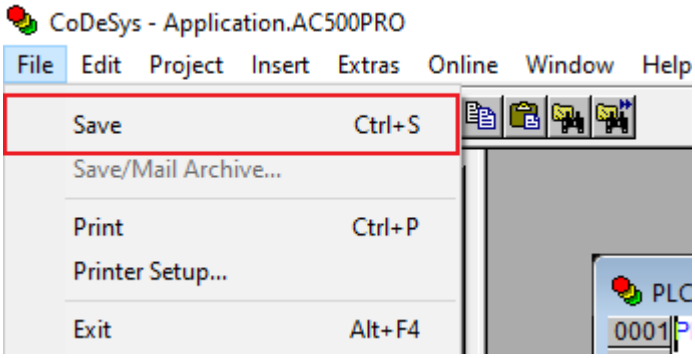


▷ In the CODESYS editor menu select *"Project ➔ Build"*

⇨ The result of the compiling is shown in the *"Messages"* field at the bottom of the screen.

If you skip the compiling and select *"Login"*, the Automation Builder will automatically trigger compiling in advance to logging-in.

Save CODESYS project



▷ In the CODESYS editor menu select *"File ➔ Save"*.

Alternatively, select the save icon in the tool bar.

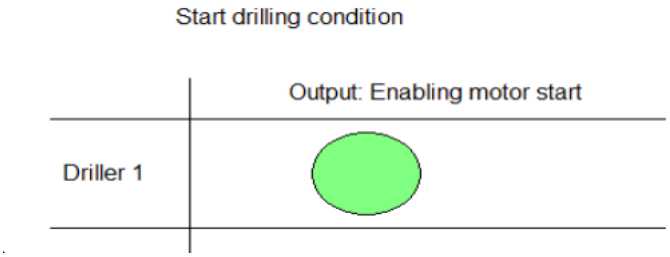
Alternatively, press *[Ctrl]* + *[S]*.

Loading the project to the CPU

▷ Download the project to the CPU ↬ *as described in Chapter 1.2.18.1.2.6 , on page 92*.

Test the program

Operate the switches and observe the visualization screen.



Reset the CPU

Reset values and parameters

In some cases, it could be required to do a CPU reset, e.g., for resetting of counter values, parameters etc.

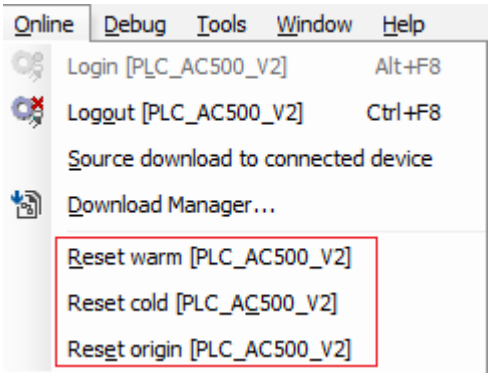


Fig. 7: Reset commands in “Online” menu

Reset warm	All variables are reset, except R% variables.
Reset cold	Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except R% variables.
Reset origin	All variables and the application project are reset.

Table 5: Behavior of variables of type VAR (local or global) and variables of type PERSISTENT RETAIN

	VAR	VAR PERSISTENT RETAIN
After online command 'Online change'	no change	no change
After online command 'Download'	initialization	no change
After online command 'Reset warm'	initialization	no change
After online command 'Reset cold'	initialization	no change
After online command 'Reset origin'	initialization	initialization
After power supply off	initialization	no change

1.2.18.1.3 Example project for remote I/O expansion with PROFINET

This example introduces the configuration of the PLC with remote I/O. The use of I/O channels in a program and commissioning of the configuration is shown.

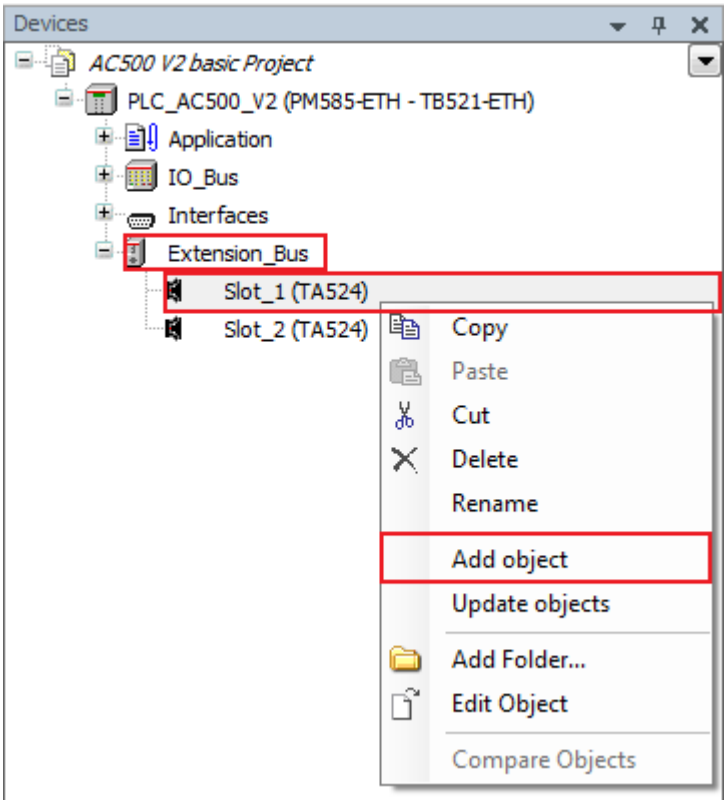
Preconditions

- Automation Builder is installed and licensed as, at least, standard edition ↳ *Chapter 1.2.4 "Managing your licenses" on page 20*.
- AC500 V2 CPU is assembled and connected to the PC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V2" on page 67*.
- Configuration and programming of this example project will be made in the existing example project for central I/O expansion ↳ *Chapter 1.2.18.1.2 "Example project for central I/O expansion" on page 70*.
- CM579-PNIO communication module is inserted in terminal base and connected to the PLC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V2" on page 67*.
- CI502-PNIO communication interface module is inserted in terminal unit and connected to the PLC ↳ *Chapter 1.2.18.1.1 "Hardware AC500 V2" on page 67*.

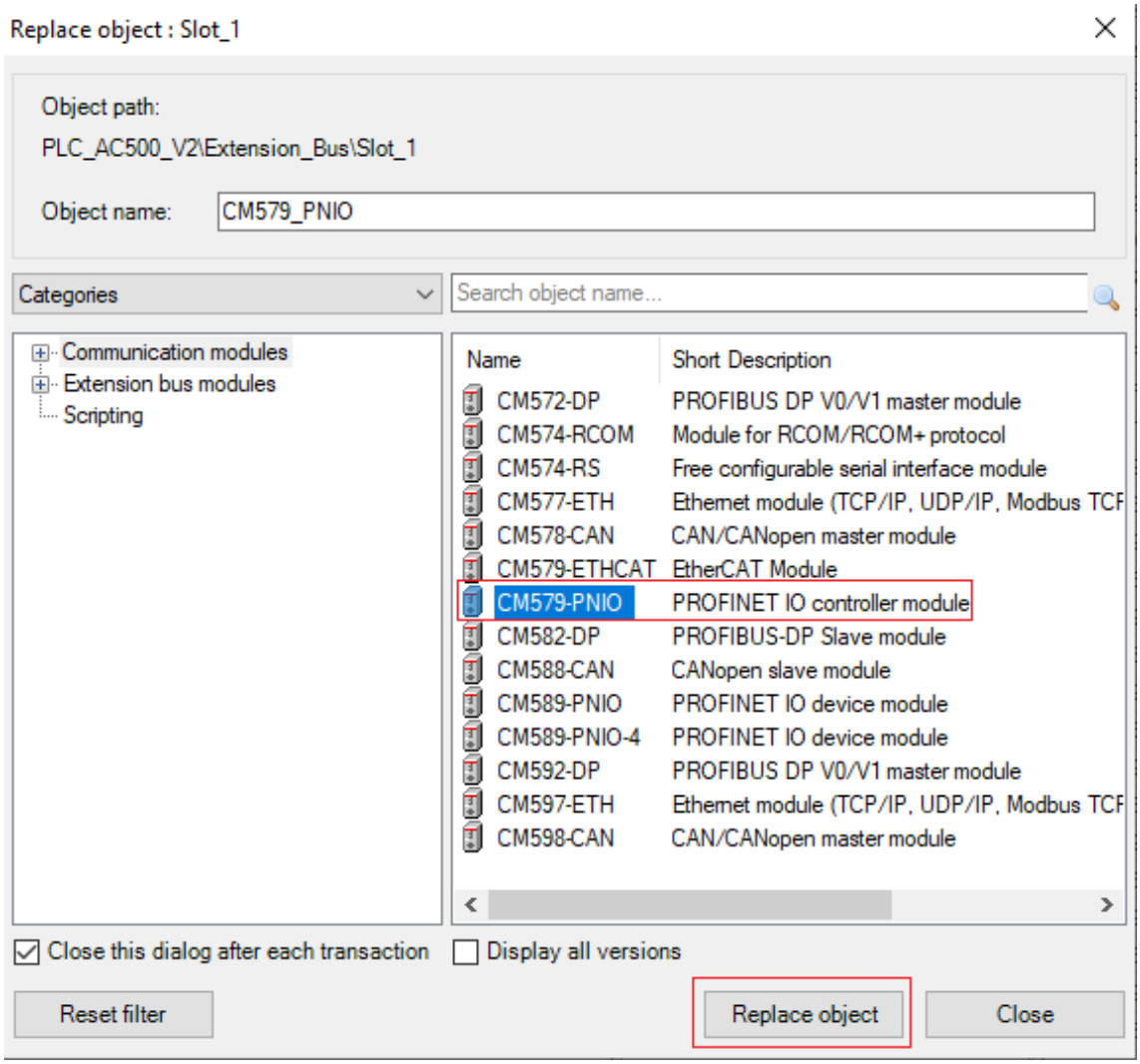
Set-up PROFINET controller

Add the CM579-PNIO to the device tree

1. In the Automation Builder device tree under *"Extension_Bus"*, right-click *"Slot_1"*.
2. Select *"Add object"*.

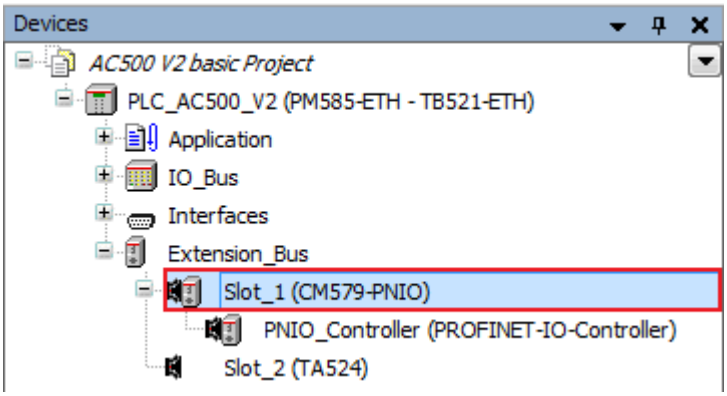


3. Select *"CM579-PNIO"*.



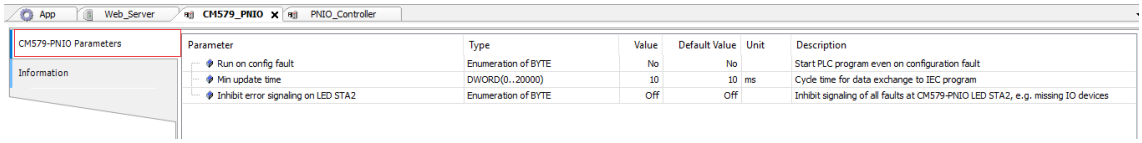
4. Select *“Replace object”* to add the CM579-PNIO.

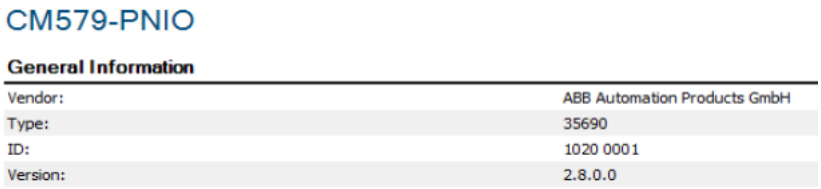
Set-up the general behavior



1. Under *“Extension_Bus”*, double-click *“CM579_PNIO”* in the device tree.
   ⇨ A tab opens in the editor view.

2. Select *“CM579-PNIO Parameters”*.

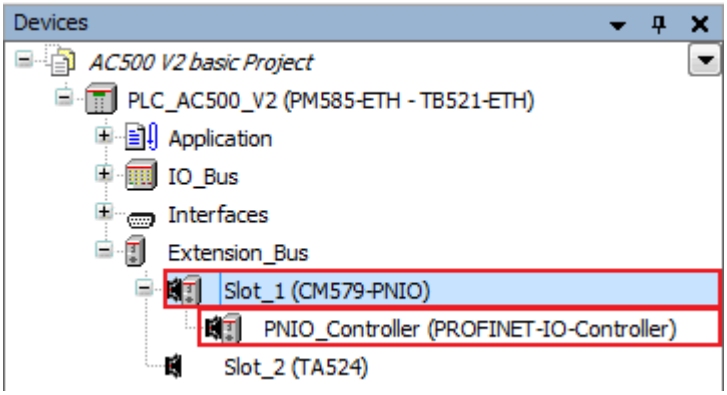




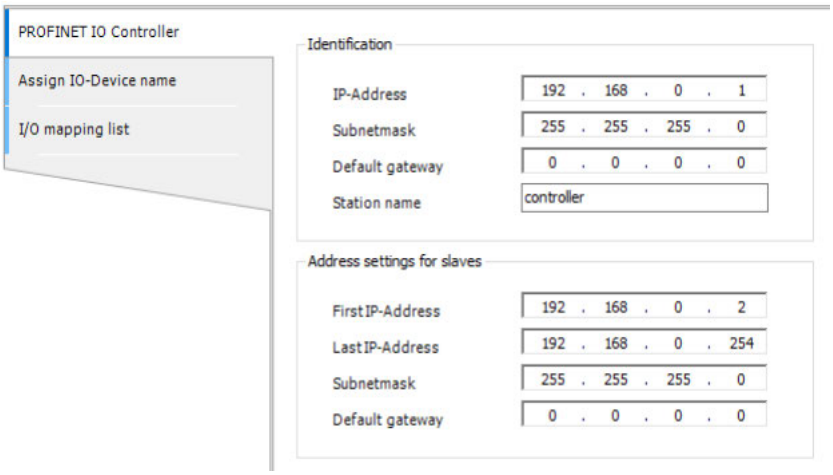
3. Select *“Information”*.
   ⇨ This page contains general information about the CM579-PNIO.
4. For the example project, you can keep the default settings.

Set-up the PROFINET IO controller

☑ To edit settings for the controller, you must not be logged-in to the PLC.



1. Under *“CM579_PNIO”*, double-click *“PNIO_Controller”* in the device tree.
   ⇨ A tab opens in the editor view.
2. Select *“PROFINET IO CONTROLLER”*



3. Here, you can set-up the way, IP addresses are distributed out to the industrial bus network. You can even set, what IP-address and DNS name (station name) the PROFINET controller has.

   For the example project, keep the default settings.

Set-up PROFINET device

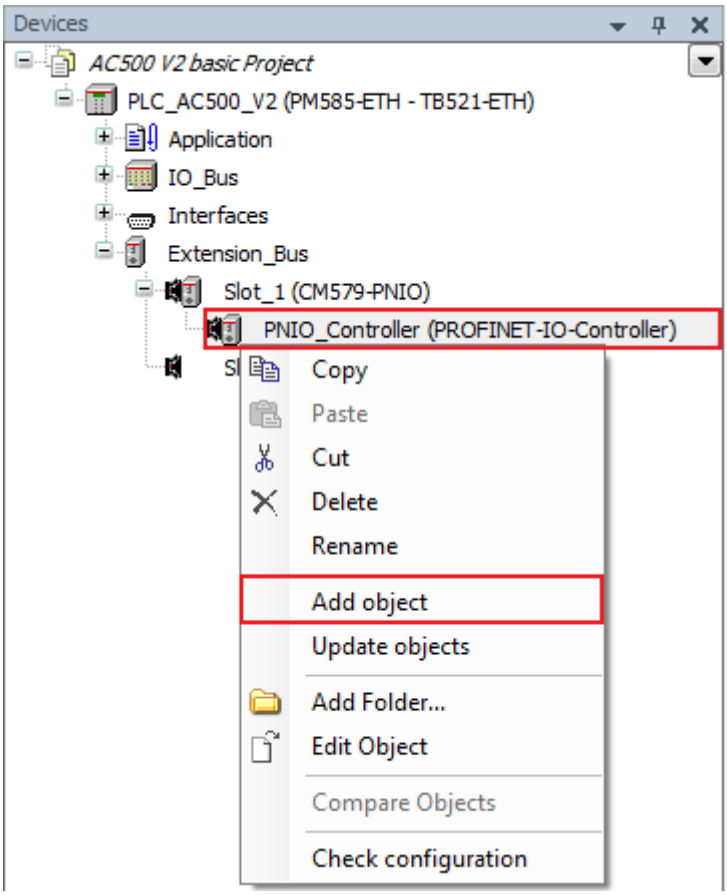
Hardware preparation

1. Switch off the power supply of your PLC.

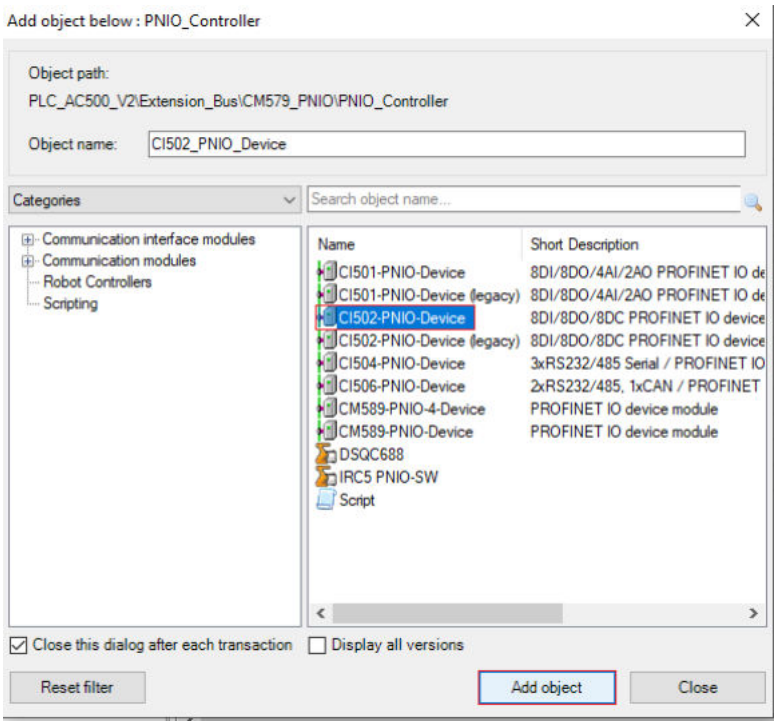
2. Use a screw driver to set the CI502 module address to "02" by positioning of the upper rotary switch to "0" and lower switch to "2". Note, that the numbers have hexadecimal format.
3. Switch on the power supply.

Add the CI502-PNIO to the device tree

1. Right-click *“PNIO_Controller”* in the device tree.
2. Select *“Add object”*.



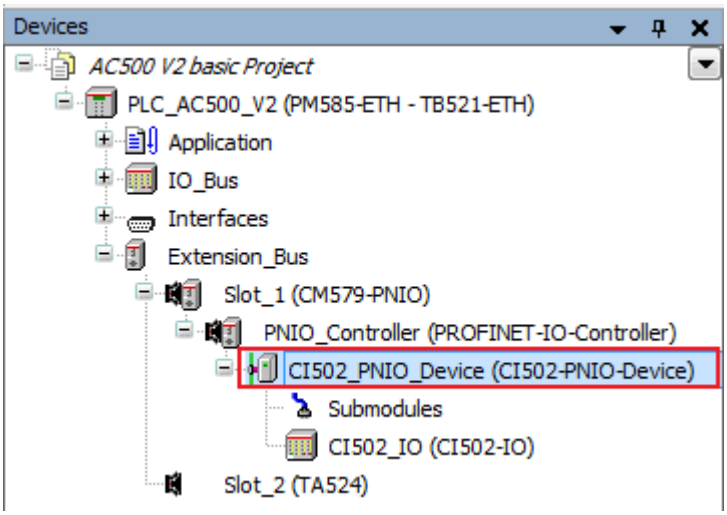
3. Select *"CI502-PNIO-Device"*.



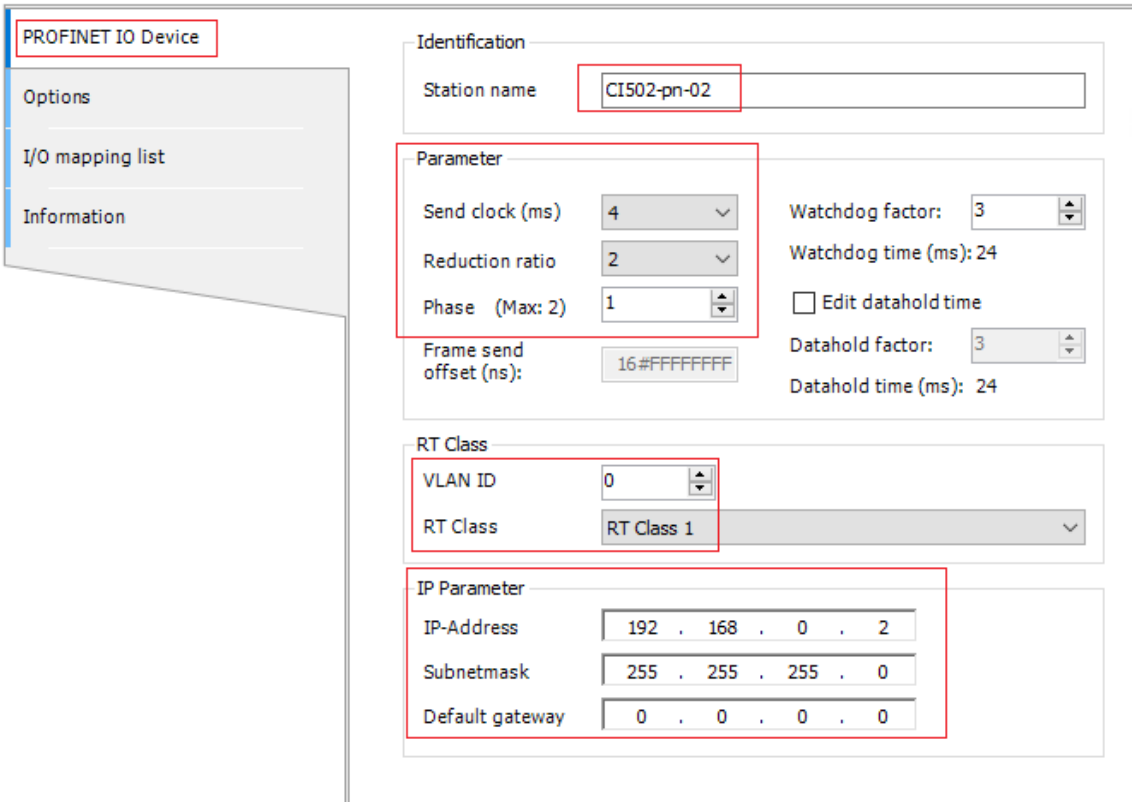
4. Select *"Add object"* to add the device.

Configure the CI502-PNIO device

Configure the CI502-PNIO PROFINET IO device



1. Double-click *“CI502_PNIO_Device”*.
   ⇨ A tab opens in the editor view.
2. Select *“PROFINET IO Device”*.



Station name	Default station name
Parameter	Communication time set-up
VLAN	Virtual local area network ID
RT Class	PROFINET IO RT (real time) type settings
IP Parameter	IP-addressing parameters of the node. If modifications are required for *“IP Parameter”*, they must be done also for CM579-PNIO and all other devices in this PROFINET line.

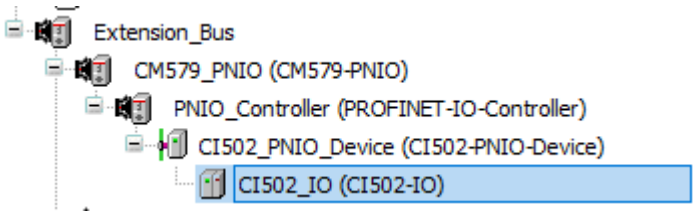
3. Set station name to "ci502-pn-02" according to hardware settings.

   For numbers greater than 09 always make sure, that the last two **decimal** digits of the node's *"Station Name"* in Automation Builder correspond to the position of module's rotary switches (**hexadecimal** values): e.g., "ci502-pn-10" <-> "0A" or "ci502-pn-16" <-> "10".

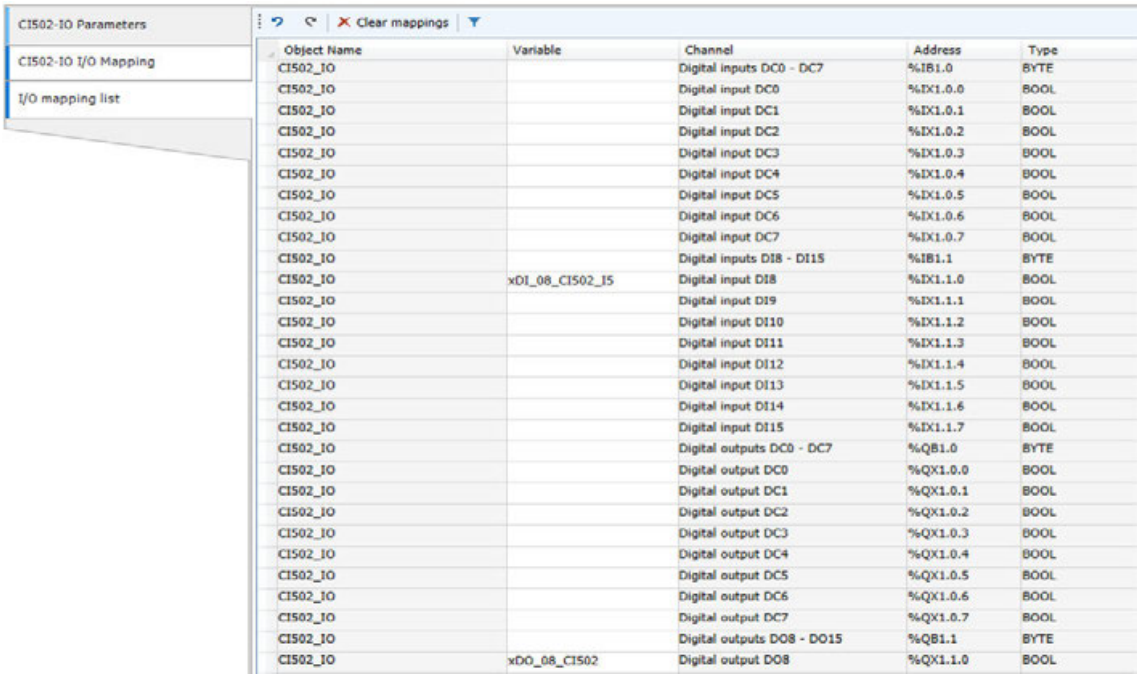
4. Leave the default settings for *"IP Parameter"*.
5. Adjust the communication time settings to get a Watchdog (ms) 24:
   - *"Send clock (ms)"*: 4
   - *"Reduction ratio"*: 2
   - *"Phase"*: 1
6. Leave the default settings for *"VLAN ID"*.
7. Leave the default settings for *"RT Class"*.

> *If the node has the same device address (the last two digits of the device name) as set by means of the rotary switches on the module, all the node parameters will be loaded automatically upon initialization scan of the CI50x module. This allows, e.g., the module exchange without an engineering tool.*

Create CI502-PNIO I/O mapping to symbols



1. Double-click *"CI502_IO"*.



Object Name	Variable	Channel	Address	Type
CI502_IO		Digital inputs DC0 - DC7	%IB1.0	BYTE
CI502_IO		Digital input DC0	%IX1.0.0	BOOL
CI502_IO		Digital input DC1	%IX1.0.1	BOOL
CI502_IO		Digital input DC2	%IX1.0.2	BOOL
CI502_IO		Digital input DC3	%IX1.0.3	BOOL
CI502_IO		Digital input DC4	%IX1.0.4	BOOL
CI502_IO		Digital input DC5	%IX1.0.5	BOOL
CI502_IO		Digital input DC6	%IX1.0.6	BOOL
CI502_IO		Digital input DC7	%IX1.0.7	BOOL
CI502_IO		Digital inputs DI8 - DI15	%IB1.1	BYTE
CI502_IO	xDI_08_CI502_I5	Digital input DI8	%IX1.1.0	BOOL
CI502_IO		Digital input DI9	%IX1.1.1	BOOL
CI502_IO		Digital input DI10	%IX1.1.2	BOOL
CI502_IO		Digital input DI11	%IX1.1.3	BOOL
CI502_IO		Digital input DI12	%IX1.1.4	BOOL
CI502_IO		Digital input DI13	%IX1.1.5	BOOL
CI502_IO		Digital input DI14	%IX1.1.6	BOOL
CI502_IO		Digital input DI15	%IX1.1.7	BOOL
CI502_IO		Digital outputs DC0 - DC7	%QB1.0	BYTE
CI502_IO		Digital output DC0	%QX1.0.0	BOOL
CI502_IO		Digital output DC1	%QX1.0.1	BOOL
CI502_IO		Digital output DC2	%QX1.0.2	BOOL
CI502_IO		Digital output DC3	%QX1.0.3	BOOL
CI502_IO		Digital output DC4	%QX1.0.4	BOOL
CI502_IO		Digital output DC5	%QX1.0.5	BOOL
CI502_IO		Digital output DC6	%QX1.0.6	BOOL
CI502_IO		Digital output DC7	%QX1.0.7	BOOL
CI502_IO		Digital outputs DO8 - DO15	%QB1.1	BYTE
CI502_IO	xDO_08_CI502	Digital output DO8	%QX1.1.0	BOOL

2. Select *"PNIO Module I/O Mapping"*.

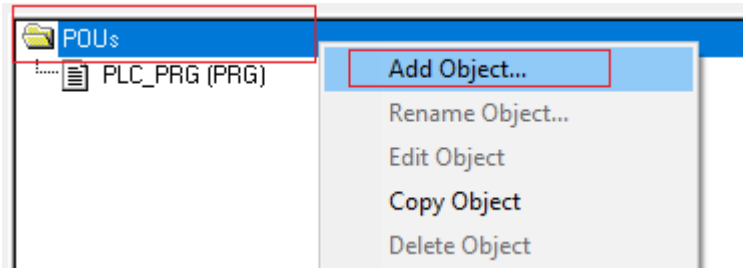
3. Fill in the variable names:

Element	Hardware channel	Symbol
Switch I5	CI502 DI8	xDI_08_CI502_I5
LED output DO8	CI502 DO 8	xDO_08_CI502

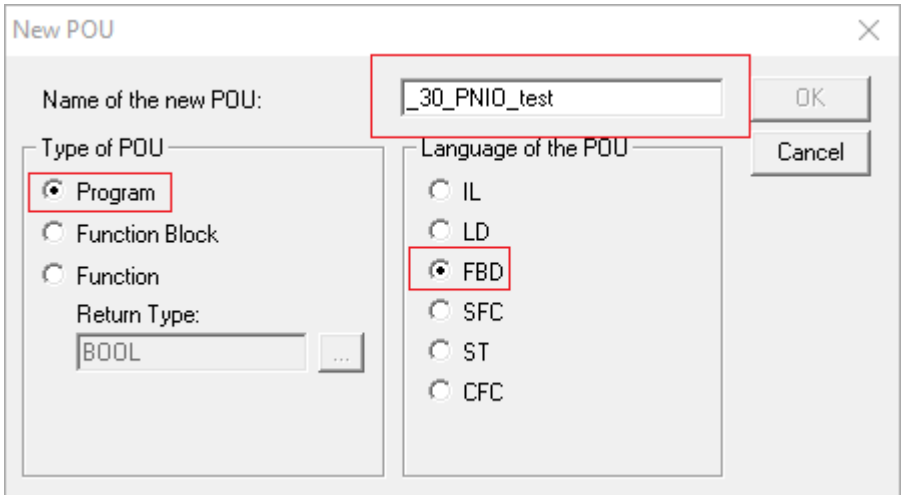
Add remote I/O expansion to project

Add a program POU to the project

1. Double click *“Application”* in the device tree to create the application file.
   ⇨ This will open the IEC 61131 programm editor CODESYS (A configuration file will be created.)(*Chapter 1.2.18.2.2.4.1 “Starting the IEC 61131 programm editor CODESYS” on page 127*).
2. In the CODESYS editor device tree right-click “POUs”.
3. Select *“Add object”*.

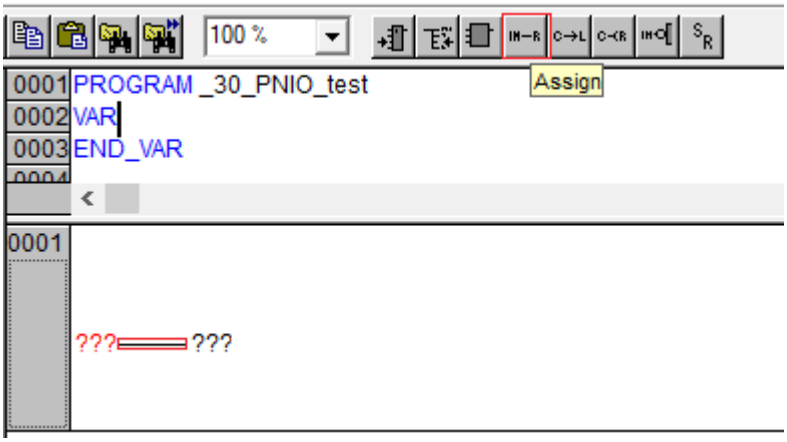


4. Enter *“_30_PNIO_test”*.
5. Select *“Program”*.
6. Select Function Block Diagram *“FBD”*.
7. Select *“[OK]”* to add POU.

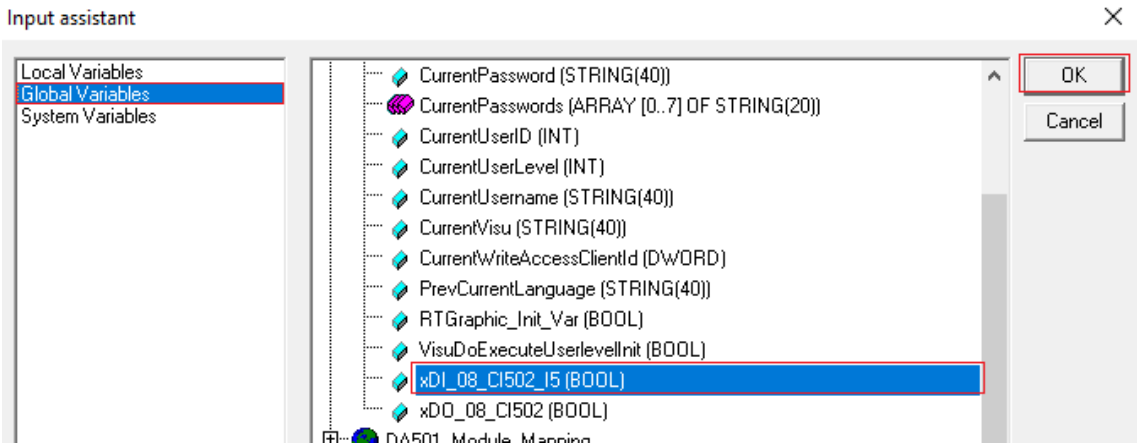


Create POU logic

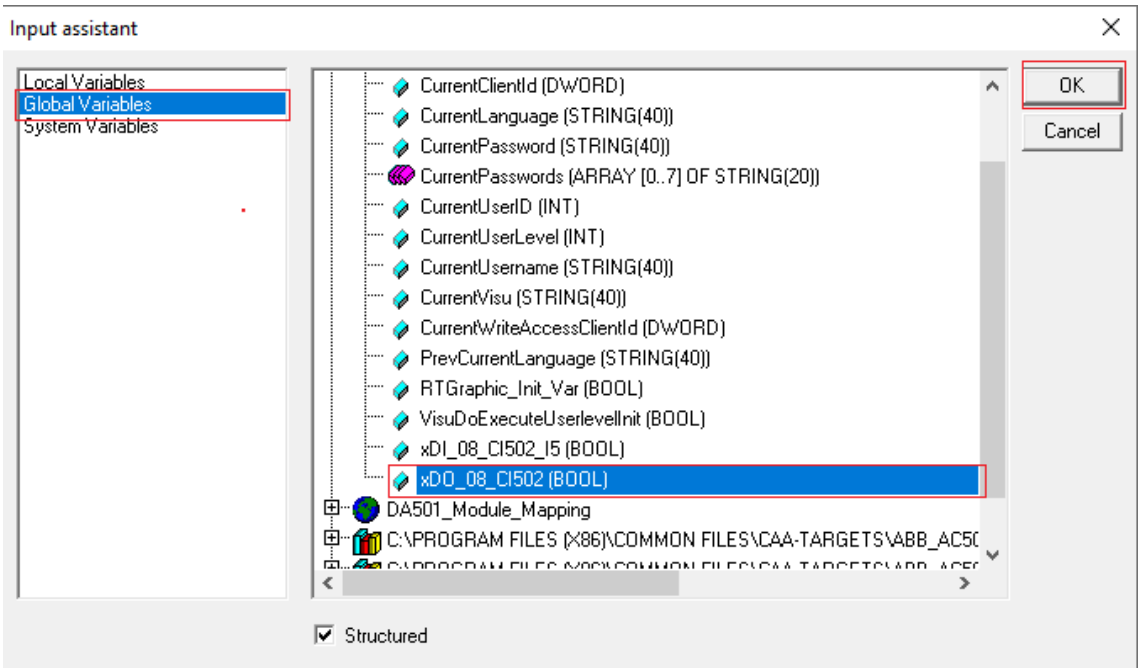
1. In the CODESYS editor device tree double-click *“_30_PNIO_test”*
2. Select *“Assign”* from Tools.



3. Select *"???"* on the left side of the assignment, press *[F2]*.
4. Select *"Global Variables"*.
5. In *"CI502_IO_Module_Mapping"* list, select *"xDI_08_CI502_I5"*.
6. Select *"[OK]"* to add this variable to the left side of the assignment connector.

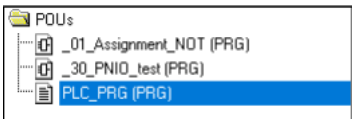


7. Select *"???"* on the right side of the assignment, then press *[F2]*.
8. In *"CI502_IO_Module_Mapping"* list" , select *"xDO_08_CI502"*.
9. Select *"[OK]."*

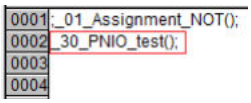


Call the POU in PLC_PRG

1. In the CODESYS editor device tree double-click *“PLC_PRG”*.

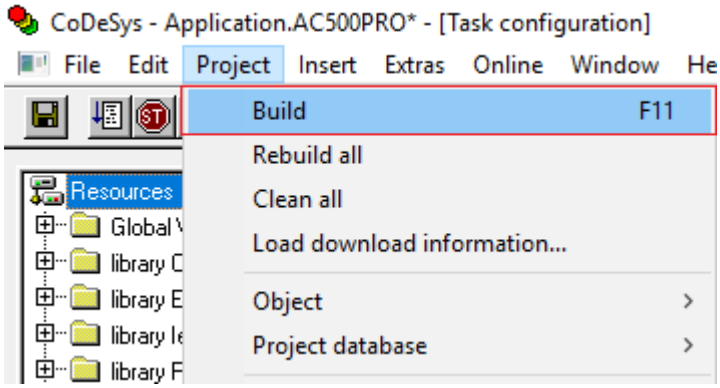


2. Select the next free line in *“PLC_PRG”* and press *[F2]*.
   ⇨ *“Input Assistant”* opens.
3. Select *“User defined Program”*.
4. Select *“_30_PNIO test”*.
5. Select *“[OK]”* to close the dialog.



Compile the project

Before logging-in to the CPU, you need to compile the complete code without any errors.

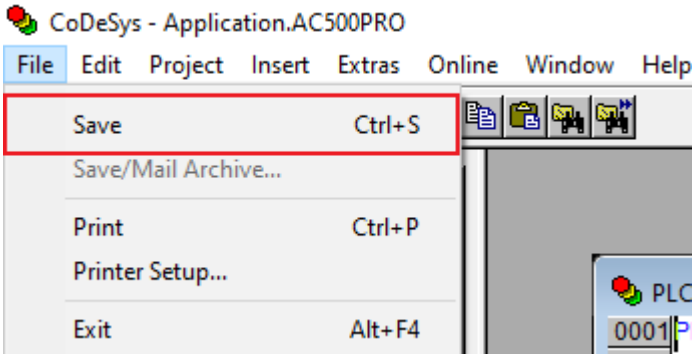


▷ In the CODESYS editor menu select *"Project ➔ Build"*

⇨ The result of the compiling is shown in the *"Messages"* field at the bottom of the screen.

If you skip the compiling and select *"Login"*, the Automation Builder will automatically trigger compiling in advance to logging-in.

Save CODESYS project



▷ In the CODESYS editor menu select *"File ➔ Save"*.

Alternatively, select the save icon in the tool bar.

Alternatively, press *[Ctrl]* + *[S]*.

Loading the project to the CPU

▷ Download the project to the CPU ↬ *as described in Chapter 1.2.18.2.2.5 , on page 135*.

Test the program

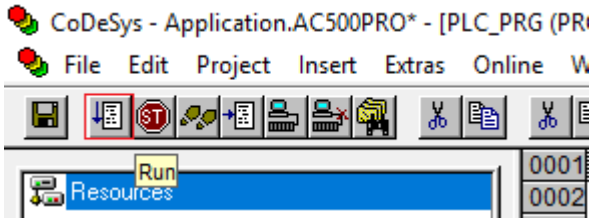
Start the program execution

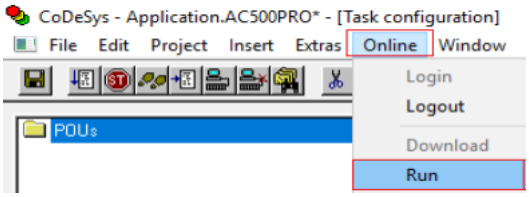
☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.

☑ Open CODESYS ↬ *Chapter 1.2.18.2.2.4.1 “Starting the IEC 61131 programm editor CODESYS” on page 127*.





▷ In the CODESYS editor menu select *“Online ➔ Run”*

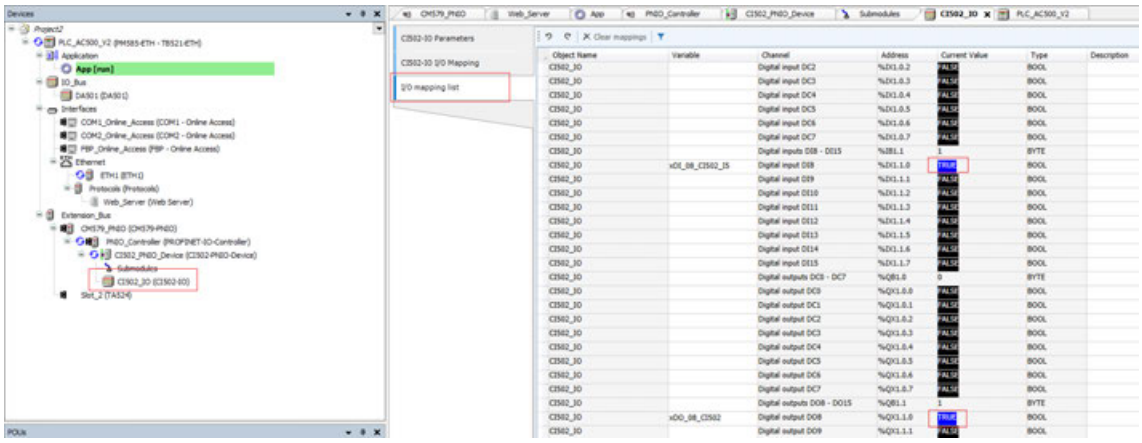
Alternatively, select the "run" icon in the tool bar.

Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I5 and observe:

- The LEDs of the relevant CI502 inputs and outputs.
- The online status of inputs and outputs within the POU.



Reset the CPU

Reset values and parameters

In some cases, it could be required to do a CPU reset, e.g., for resetting of counter values, parameters etc.

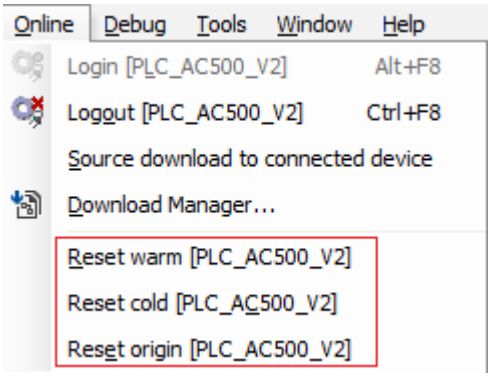


Fig. 8: Reset commands in “Online” menu

Reset warm All variables are reset, except R% variables.
Reset cold Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except R% variables.
Reset origin All variables and the application project are reset.

Table 6: Behavior of variables of type VAR (local or global) and variables of type PERSISTENT RETAIN

	VAR	VAR PERSISTENT RETAIN
After online command 'Online change'	no change	no change
After online command 'Download'	initialization	no change
After online command 'Reset warm'	initialization	no change
After online command 'Reset cold'	initialization	no change
After online command 'Reset origin'	initialization	initialization
After power supply off	initialization	no change

1.2.18.2 Example projects for AC500-eCo V2

1.2.18.2.1 Hardware AC500-eCo V2

Contents of the AC500-eCo V2 starter kit:

- 1 x AC500-eCo V2 CPU PM554-TP-ETH with 2 terminal blocks plugged on the CPU: 1 x 11 pin, 1 x 9 pin
- 1 x digital input simulator
- 1 x programming cable

System assembly, construction and connection

1. Assemble the CPU.

   Enclosed to the CPU, you find the installation instructions. Refer to these installation instructions for assembling the CPU.

2. Connect the input simulator to the CPU.

   Enclosed to the input simulator, you find the installation instructions. Refer to these installation instructions to connect the input simulator to the CPU.

DANGER!

Risk of death through electric shock.

Before using the CPU refer to the “Regulations Concerning the Setting up of Installations” for safety instructions. *http://search-ext.abb.com/library/Download.aspx?DocumentID=3ADR025003*&Action=Launch*

CAUTION!

Risk of damaging the PLC modules.

The CPU can be damaged by overvoltages and short circuits. Make sure that all voltage sources (supply and process voltage) are switched off before you begin with operations on the system.

3. The CPU needs to be powered by 24 V DC. Connect the CPU to the power, by using the 5-pin screw-type terminal block.

CAUTION!

Risk of damaging the CPU and the connected modules.

Voltages > 35 V DC can destroy the CPU and the connected modules. Make sure that the supply voltage never exceeds 35 V DC.

4. Plug the programming cable into the CPU and your PC.

1.2.18.2.2 Example project

The following steps show how to set-up an application project and configure the hardware. A simple logic is used as example to introduce in programming and commissioning of the PLC. The workflow for creation of a visualization is explained, as well as how to set-up a web server for visualization.

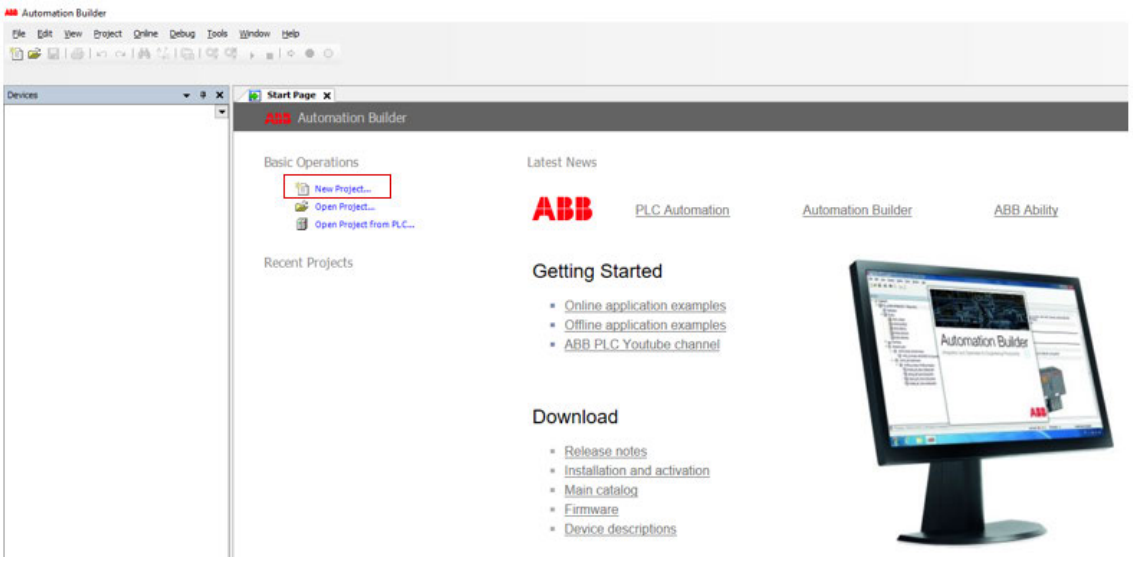
Preconditions

- Automation Builder is installed and licensed as, at least, basic edition ↬ *Chapter 1.2.4 “Managing your licenses” on page 20*.
- AC500 V2 CPU is assembled and connected to the PC ↬ *Chapter 1.2.18.2.1 “Hardware AC500-eCo V2” on page 121*.

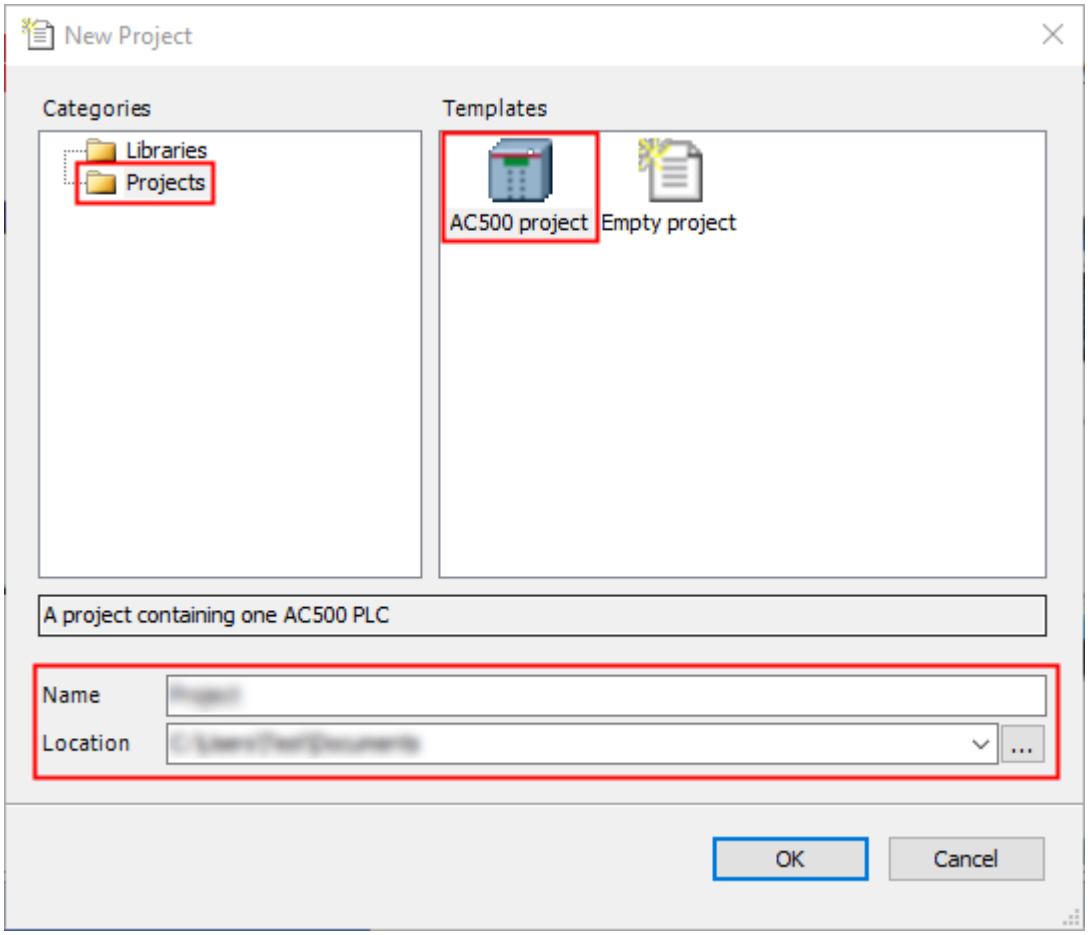
Create, set-up and save your AC500 V2 project

Create a project

1. Launch Automation Builder either out of the desktop icon or out of the Windows menu.

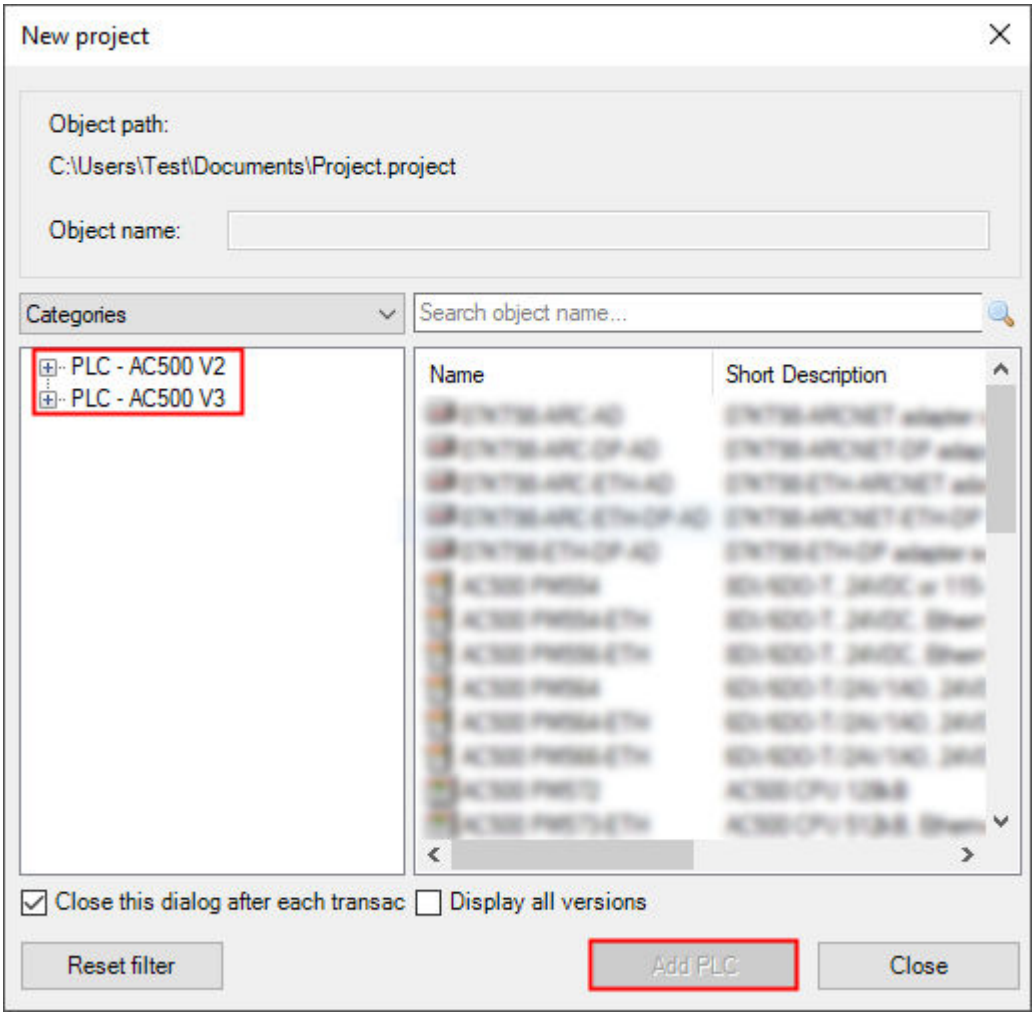


2. Select *"New Project"* or go to menu *"File ➔ New Project"*.



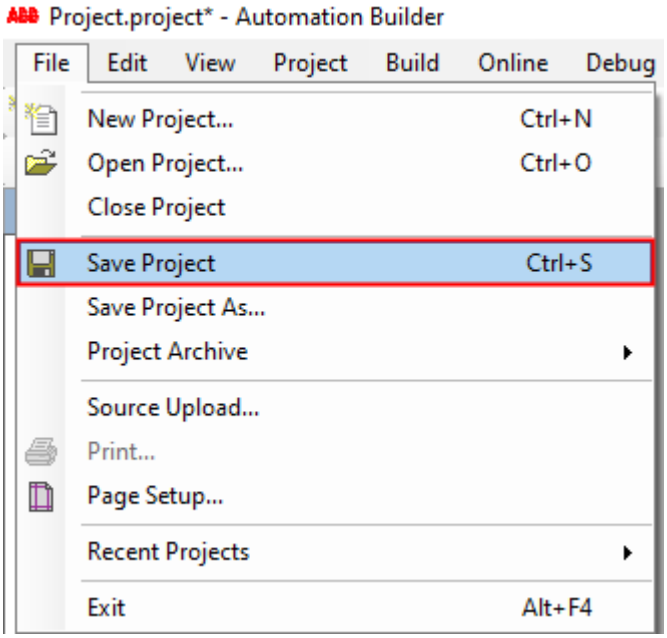
3. Select *"Projects"*.
4. Select *"AC500 project"*.
5. Fill in project name.
6. Choose a location to save the project to.

7. Select *“OK”*.
8. Select *“PLC - AC500 V2”*.
9. Select the CPU according to your hardware set-up.



10. Select *“Add PLC”* to add the CPU to your application.

Save the project



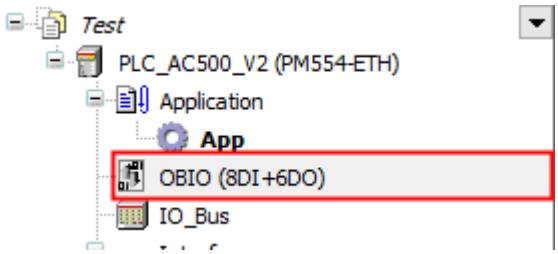
▷ Select menu *"File ➔ Save Project"*.

Alternatively, select the save icon in the tool bar.

Alternatively, press *[Ctrl]* + *[S]*.

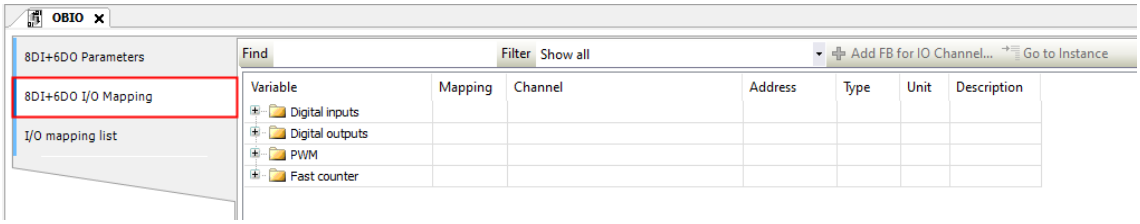
Configure the onboard I/O channels

Onboard I/O variable mapping



1. Double-click *"OBIO"* in the device tree.

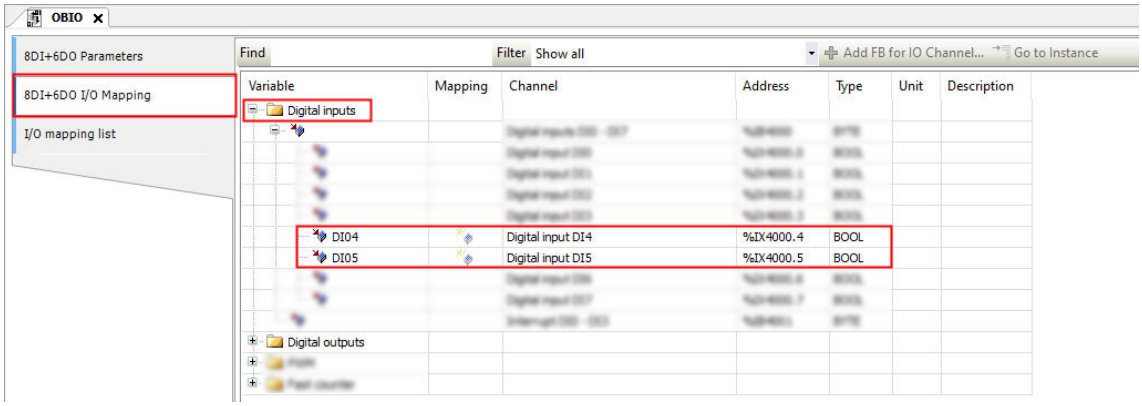
   ⇨ A tab opens in the editor view.



2. Select *"8DI+6DO I/O Mapping"*.

   ⇨ Here, you will map variable names (symbols) for the channels you will need in the program.

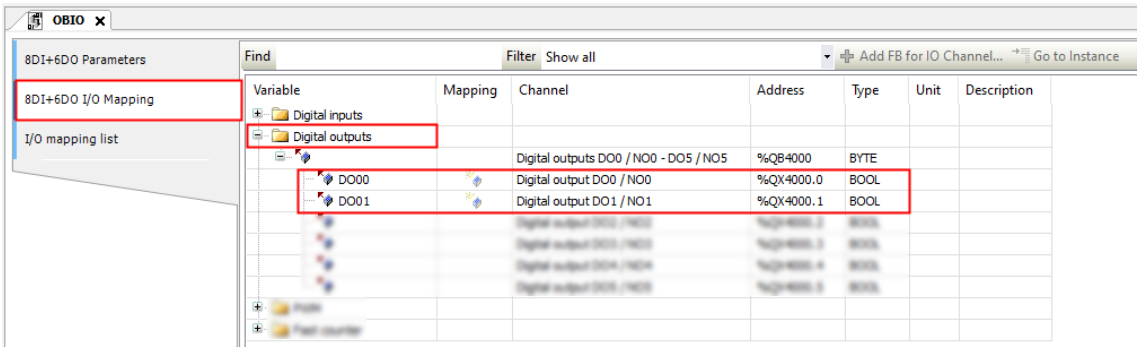
Handle the digital input variables



1. Open the list of the digital inputs.
2. Fill in the variable names:

Channel	Type	Variable
Digital input DI4	BOOL	DI04
Digital input DI5	BOOL	DI05

Handle the digital output variables



1. Open the list of the digital outputs.
2. Fill in the variable names:

Channel	Type	Variable
Digital output DO0 / NO0	BOOL	DO00
Digital output DO1 / NO1	BOOL	DO01

Programming and compiling

You write the program code in a separate IEC 61131-3 editor (CODESYS). You can open CODESYS out of Automation Builder.

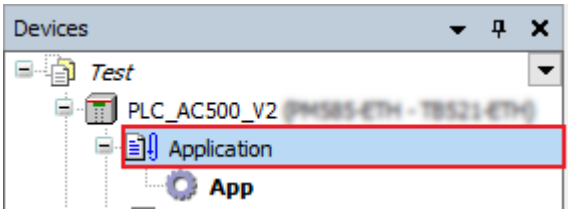
Supported programming languages:

- ST (Structured Text)
- IL (Instruction List)
- FBD (Function Block Diagram)
- LD (Ladder Diagram)
- SFC (Sequential Function Chart)
- CFC (Continuous Function Chart)

Starting the IEC 61131 programm editor CODESYS

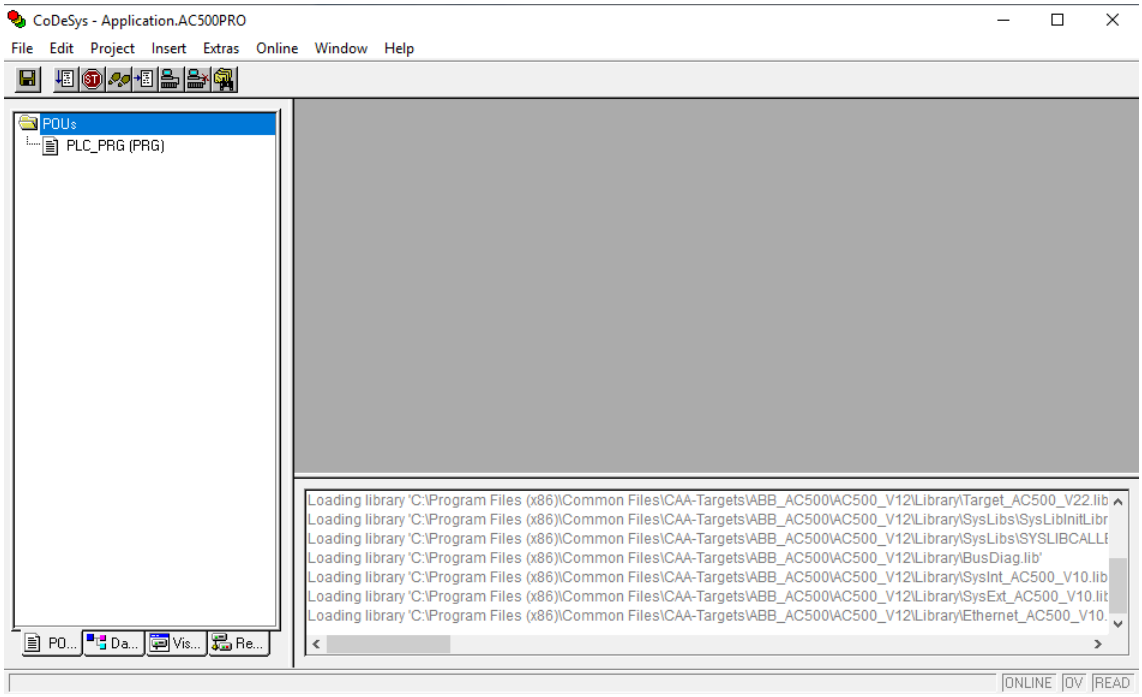
To start the IEC 61131 programm editor CODESYS:

☑ Open an AC500 V2 project in Automation Builder



▷ In the Automation Builder device tree double-click *“Application”*

⇨ This will start the IEC 61131 programm editor CODESYS

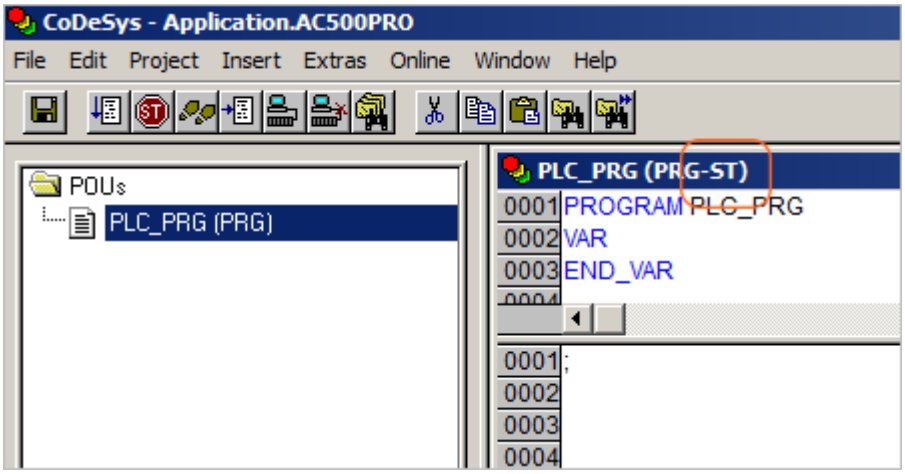


Change the programming language into FBD

The default programming language is ST.

1. Select *“PLC_PRG (PRG)”*.

> *If you made a mistake during the process, you can always undo. In the CODESYS menu select “Edit ➔ Undo”.*

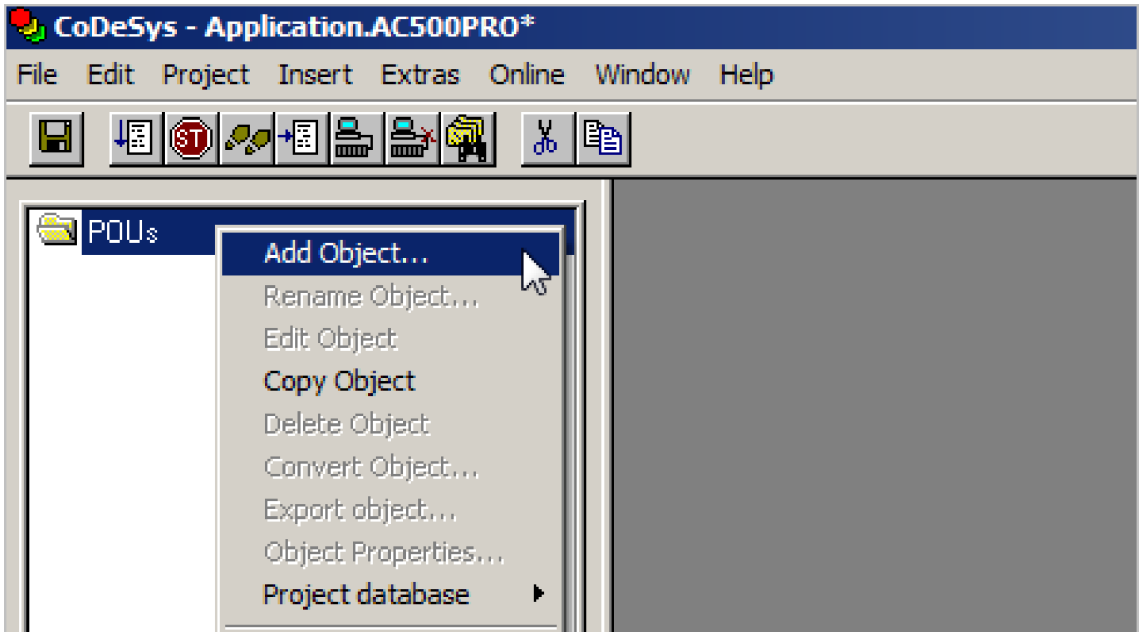


2. Press *[DELETE]*.

   ⇨ *“PLC_PRG (PRG)”* has been removed.

3. Right-click *“POUs”*.

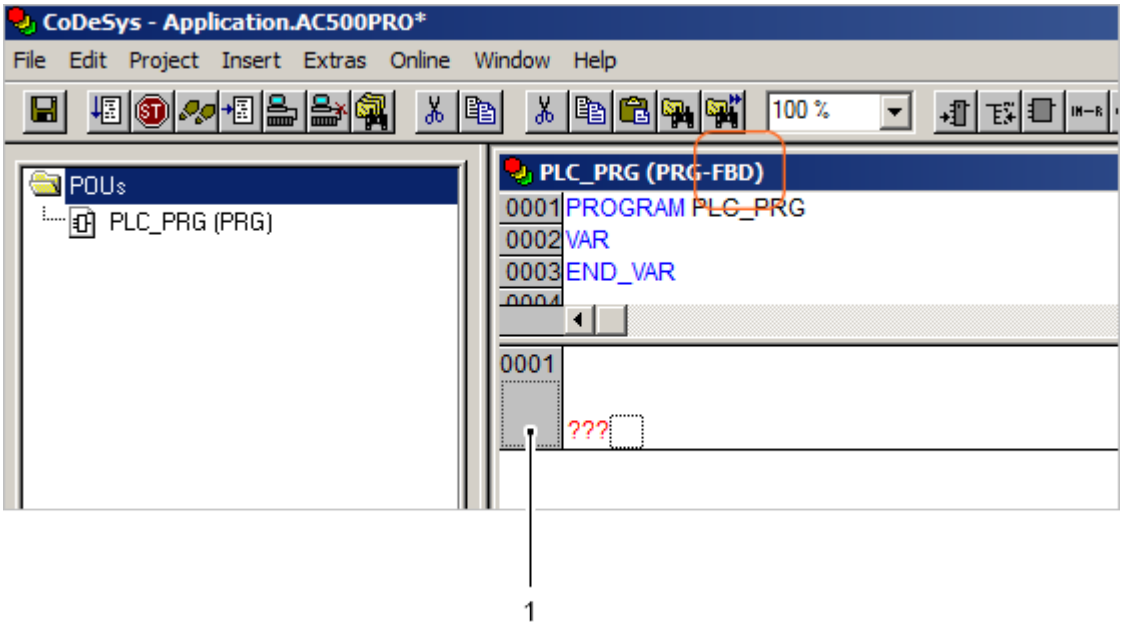
   Select *“Add Object”*.



4. Under *“Language of the POU”* select *“FBD”*.

5. Select *“OK”*.

   ⇨ A newly generated POU opens. The programming language of the POU is FBD.



1 Network (here network number 0001)

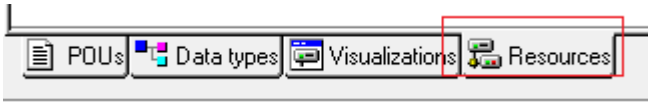
Task configuration

A task is a time unit in the processing of a user program (IEC application), which defines by parameters the way and the speed the CPU is executing the user program.

For this project you will use only one cycling task.

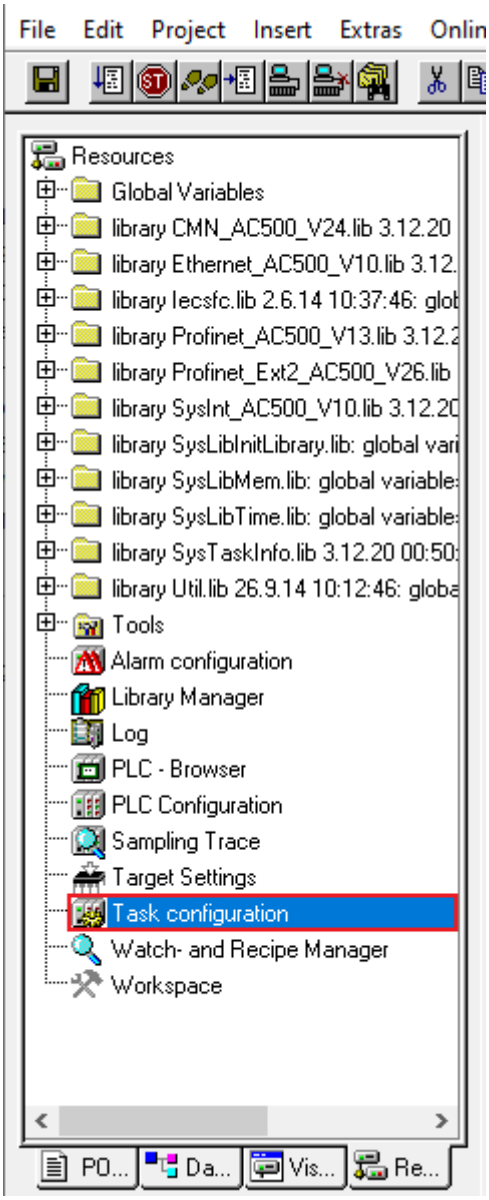
☑ Open CODESYS editor ↯ *Chapter 1.2.18.2.2.4.1 “Starting the IEC 61131 programm editor CODESYS” on page 127*

1. In the CODESYS editor menu select the *“Resources”* tab.

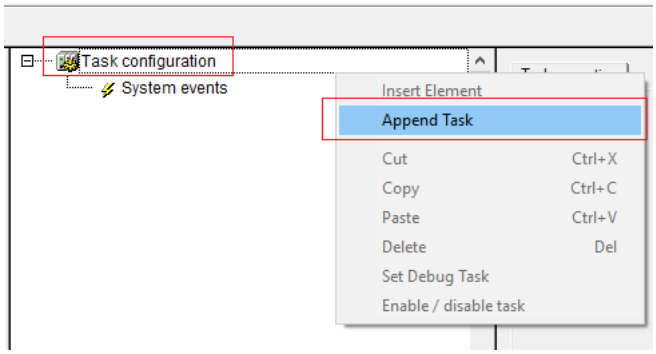


2. Double click *“Task configuration”*.

   ⇨ The Task configuration window opens.

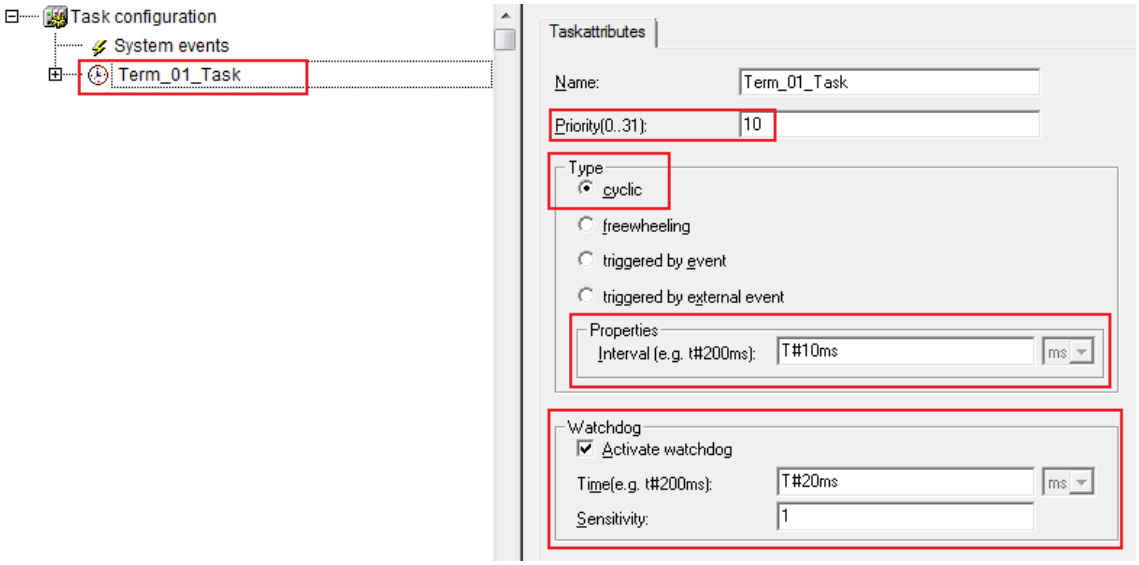


3. Right-click on *"Task configuration"*.
4. Select *"Append Task"*.



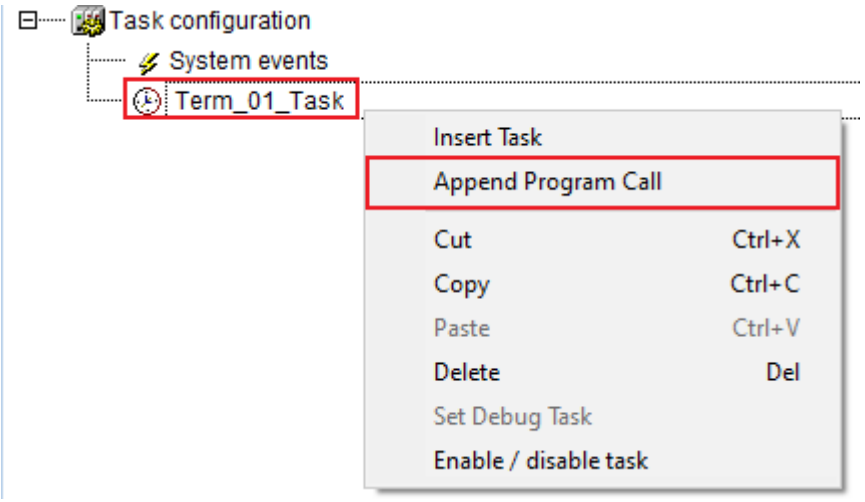
5. Enter a name.
6. Set *"Priority"* to 15
7. Set *"Type"* to *"cyclic"*.
8. Set *"Interval"* to *"T#10ms"*.

9. Activate Watchdog.
10. Set *“Time”* to *“T#20ms”*.

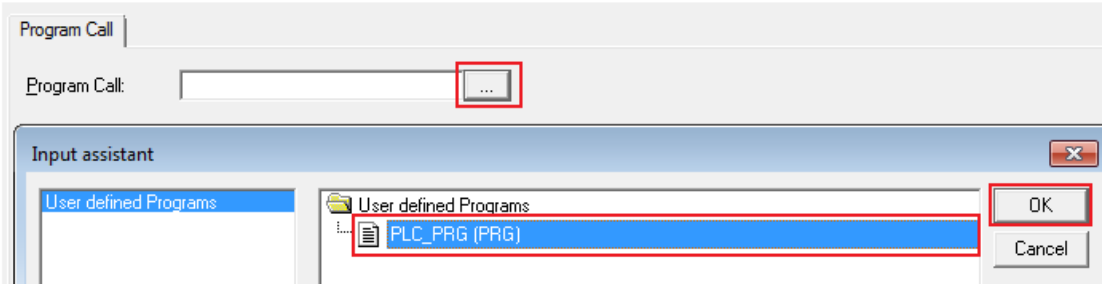


Priority type interval	This is how the CPU prioritizes the task, when more than one task is defined. Type In the CPU you can run tasks dependent on the demands of the process Interval For cyclic tasks you can set the cyclical execution time. It is usually set in milliseconds with IEC time syntax
Watchdog calls	To keep track of the time it takes to complete the task Calls You can call in one or more program POUs in one single task

11. Right click the watch icon next to *“Term_01_Task”*.
12. Select *“Append Program Call”*.

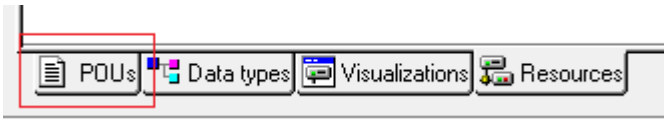


13. Select *“[…]”*.
    ⇨ The input assistant opens.
14. Select *“PLC_PRG (PRG)”*.
15. Select *“OK”*.
    ⇨ The task has been appended.

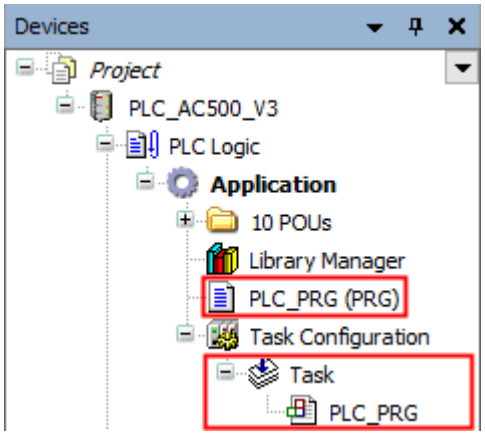


Main program PLC_PRG

In CODESYS menu select tab POUs, there is one call of a POU (program organization unit) i.e. "PLC_PRG".



In your project the "PLC_PRG" will become a main program containing calls to other programs (POUs) which you will create one by one.



The PLC_PRG POU has been defined by default in ST (Structured Text) editor. Keep this setting because of good visibility of the instructions at a glance and good handling for trouble-shooting.

Boolean logic AND

Implementation

Add Boolean "AND"

1. Select the dotted rectangle behind“ *???*”. The dotted rectangle represents the cursor position in this network.
2. In the toolbar select the Box icon.

   ⇨ A box with the AND operator will be added to your network.

When inserting a new box, it always appears as an AND operator. To change the type of the box click on the AND text and enter another name.

To get an overview of all accepted operators, functions and function blocks click on the AND text and press [F2].

Assign input and output variables to the AND operator

Assign input variables to the AND operator

The *"???"* on the left side of the AND operator represent your two inputs.

To assign variables to each input:

1. Select *"???"*.
2. Press *[F2]*.
   ⇨ The input assistant opens.
3. Under Global Variables open *"OBIO_Module_Mapping"*.
4. For each *"???"* select *"DI04"* and *"DI05"*.

Assign output variable to the AND operator

The *"???"* on the right side of the AND operator represent your output.

To assign a variable to the output:

1. Select *"???"*.
2. Press *[F2]*.
   ⇨ The input assistant opens.
3. Under Global Variables open *"OBIO_Module_Mapping"*.
4. Select *"DO00"*.

Add another network

To add another network:

1. Right-click on network 0001.
2. Select *"Network (after)"*.
   ⇨ You added a new network after network 0001. The new network number is 0002.

Add assignments and variables

To add an assignments to your network:

▷ From the toolbar select the assign symbol.

Alternatively right-click on the dotted rectangle and select *"Assign"*.

⇨ You added another assignment to the network.

To add variables to your assignments:

1. Select *"???"*.
2. Press *[F2]*.
   ⇨ The input assistant opens.

3. Under Global Variables open *“OBIO_Module_Mapping”*.
4. For the input select DO00.

   For the output select DO01.

Add Boolean NOT

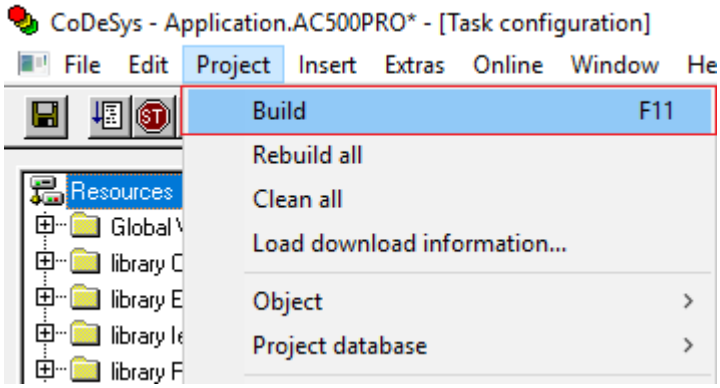
To negate the function:

1. Select the line behind the input. The dotted rectangle represents the cursor postion.
2. Right click on the dotted rectangle and select Negate.

   ⇨ You inserted a negation between your assignments.

Compile the project

Before logging-in to the CPU, you need to compile the complete code without any errors.

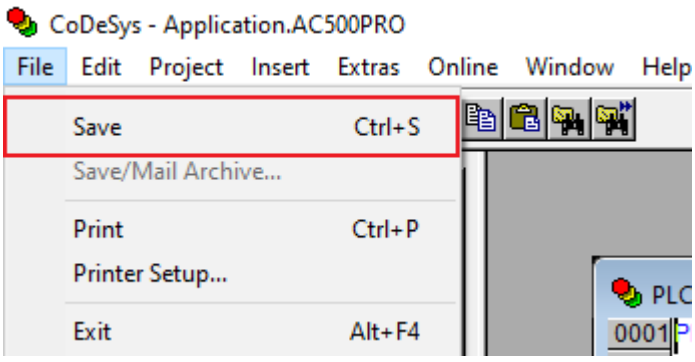


▷ In the CODESYS editor menu select *“Project ➔ Build”*

⇨ The result of the compiling is shown in the *“Messages”* field at the bottom of the screen.

If you skip the compiling and select *“Login”*, the Automation Builder will automatically trigger compiling in advance to logging-in.

Save CODESYS project



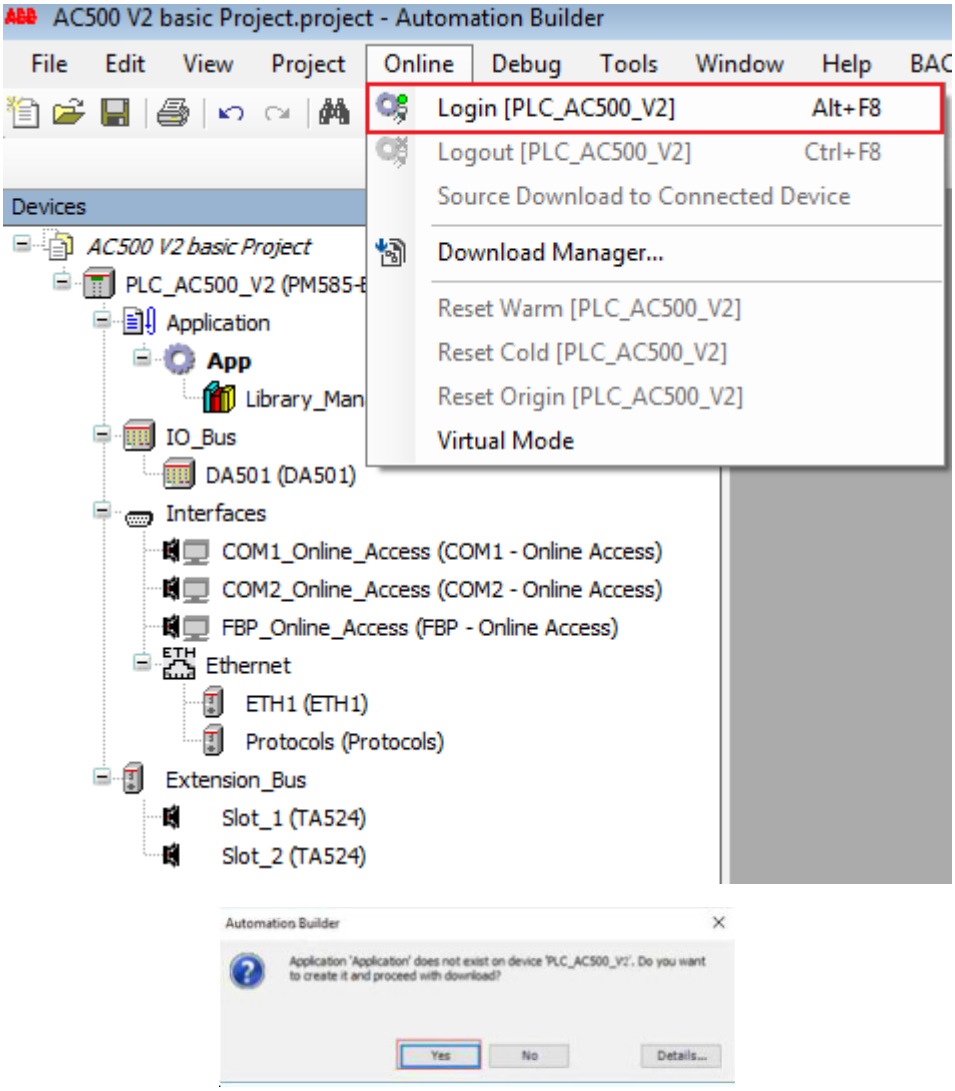
▷ In the CODESYS editor menu select *“File ➔ Save”*.

Alternatively, select the save icon in the tool bar.

Alternatively, press *[Ctrl] + [S]*.

Log-in to CPU and download the program

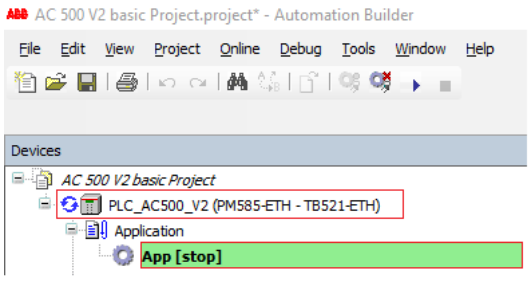
Logging-in to the CPU will load the project into the AC500 V2 CPU. The first log-in will also load the hardware set-up.



1. In the Automation Builder menu select *“Online ➔ Login [PLC_AC500_V2]”*.

   ⇨ A pop-up will appear.

2. Select *“Yes”* to download the application to the AC500V2 CPU.



   ⇨ PLC is in "stop" mode.

3. Start the PLC ↳ *Chapter 1.2.18.2.2.6.1 “Start the program execution” on page 136*.

> *Generally, if the CPU is in RUN mode, i.e. in program execution mode, a download will always cause the mode change to "stop". In stop mode the CPU is not controlling the system!*
>
> *Always, after selecting the "Login" command, read carefully the dialog box text to ensure that you are aware of the CPU's behavior after the command confirmation.*

By default, a download generates following actions in the CPU:

- The project is stored in the RAM memory.
- The project is stored in the flash EEPROM, if boot application was created.

Test the program

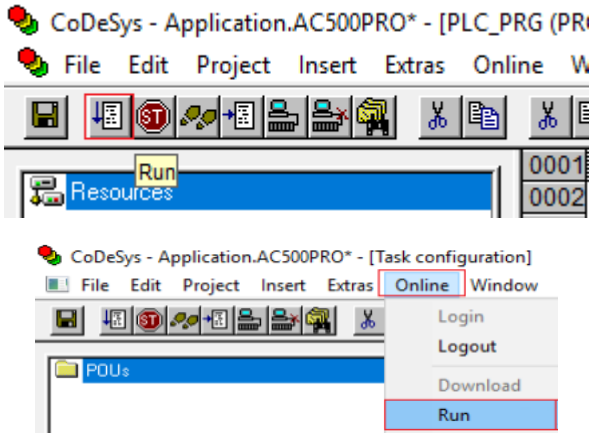
Start the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in "stop" mode.

☑ Open CODESYS *Chapter 1.2.18.2.2.4.1 "Starting the IEC 61131 programm editor CODESYS" on page 127*.



▷ In the CODESYS editor menu select *"Online ➔ Run"*

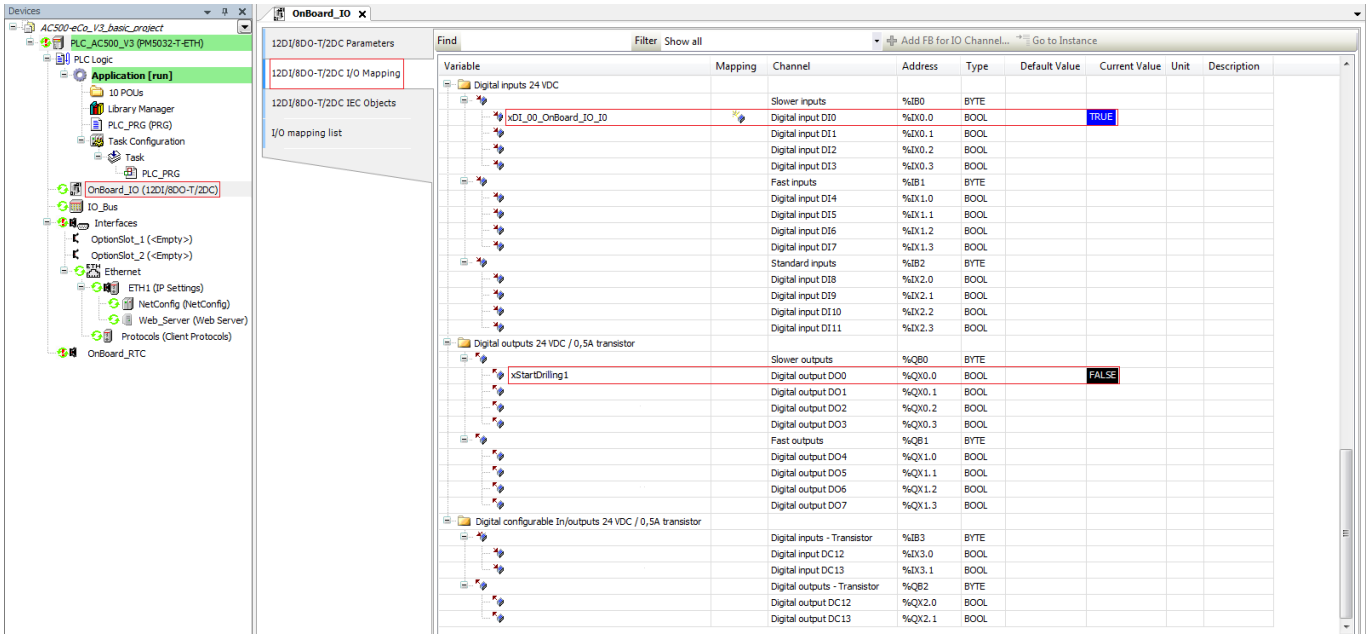
Alternatively, select the "run" icon in the tool bar.

Alternatively, press *[F5]*.

Test the function

▷ Operate the switch I1 and observe:

- The LEDs of the relevant onboard I/O inputs and outputs.
- The online status of inputs and outputs within the POU.

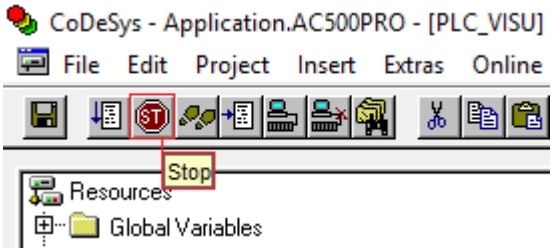


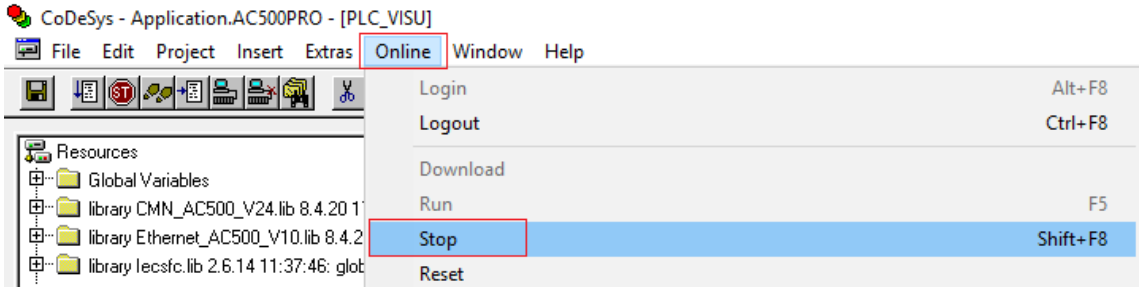
Stop the program execution

☑ You are logged in the CPU.

☑ An executable project is loaded to the CPU.

☑ The CPU is in RUN mode.





▷ In the CODESYS editor menu select *“"Online ➔ Stop [PLC_AC500_V2]"”*

Alternatively, select the "stop” icon in the tool bar.

Alternatively, press *[Shift]* + *[F8]*.

Reset the CPU

Reset values and parameters

In some cases, it could be required to do a CPU reset, e.g., for resetting of counter values, parameters etc.

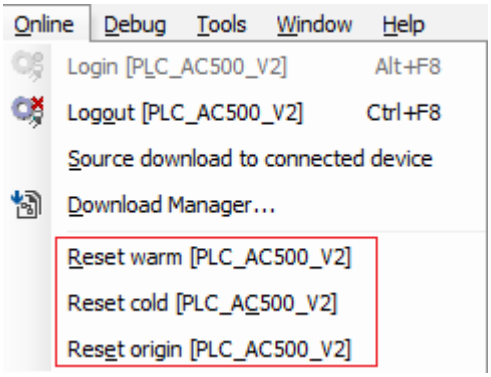


Fig. 9: Reset commands in “Online” menu

Reset warm — All variables are reset, except R% variables.
Reset cold — Causes initialization of all variables, except PERSISTENT variables. By recommended creation of remanent variables always with both properties: PERSISITENT and RETAIN, this command resets all variables, except R% variables.
Reset origin — All variables and the application project are reset.

Table 7: Behavior of variables of type VAR (local or global) and variables of type PERSISTENT RETAIN

	VAR	VAR PERSISTENT RETAIN
After online command 'Online change'	no change	no change
After online command 'Download'	initialization	no change
After online command 'Reset warm'	initialization	no change
After online command 'Reset cold'	initialization	no change
After online command 'Reset origin'	initialization	initialization
After power supply off	initialization	no change

1.3 Automation Builder installation manager

Automation Builder installation manager allows you to install customer specific software packages, modify the existing installation, update installation information and to uninstall Automation Builder software packages in a comfortable and flexible way.

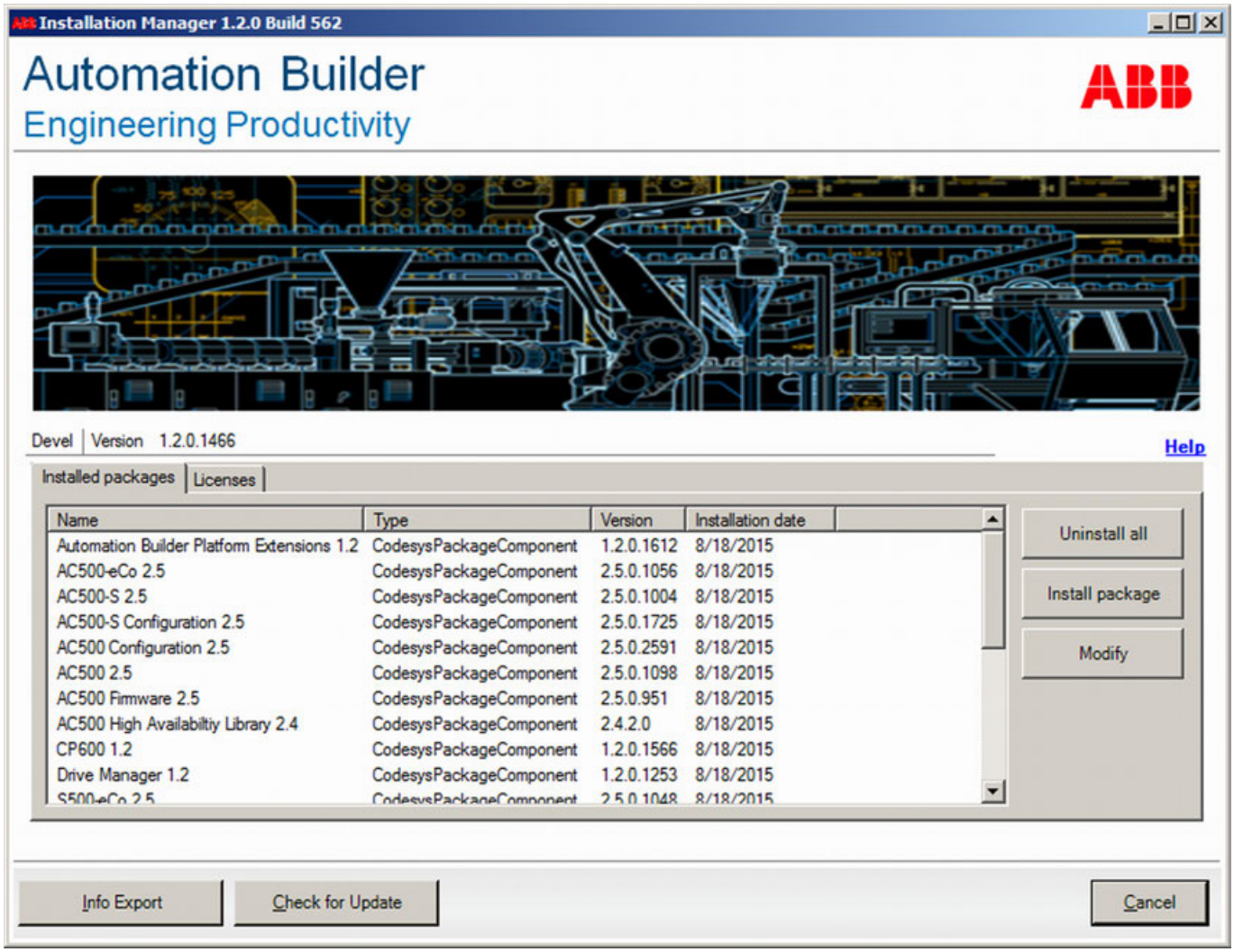
You can launch installation manager from the main menu of Automation Builder or from Windows start menu.

1. Open Automation Builder software.

   From the **Tools** menu, select **Installation Manager**.

2. As an alternative, launch installation manager from Windows start menu: *"Start menu ➔ All Programs ➔ ABB ➔ Automation Builder ➔ ABB Automation Builder Installation Manager"*.

   ⇨ Installation manager starts.



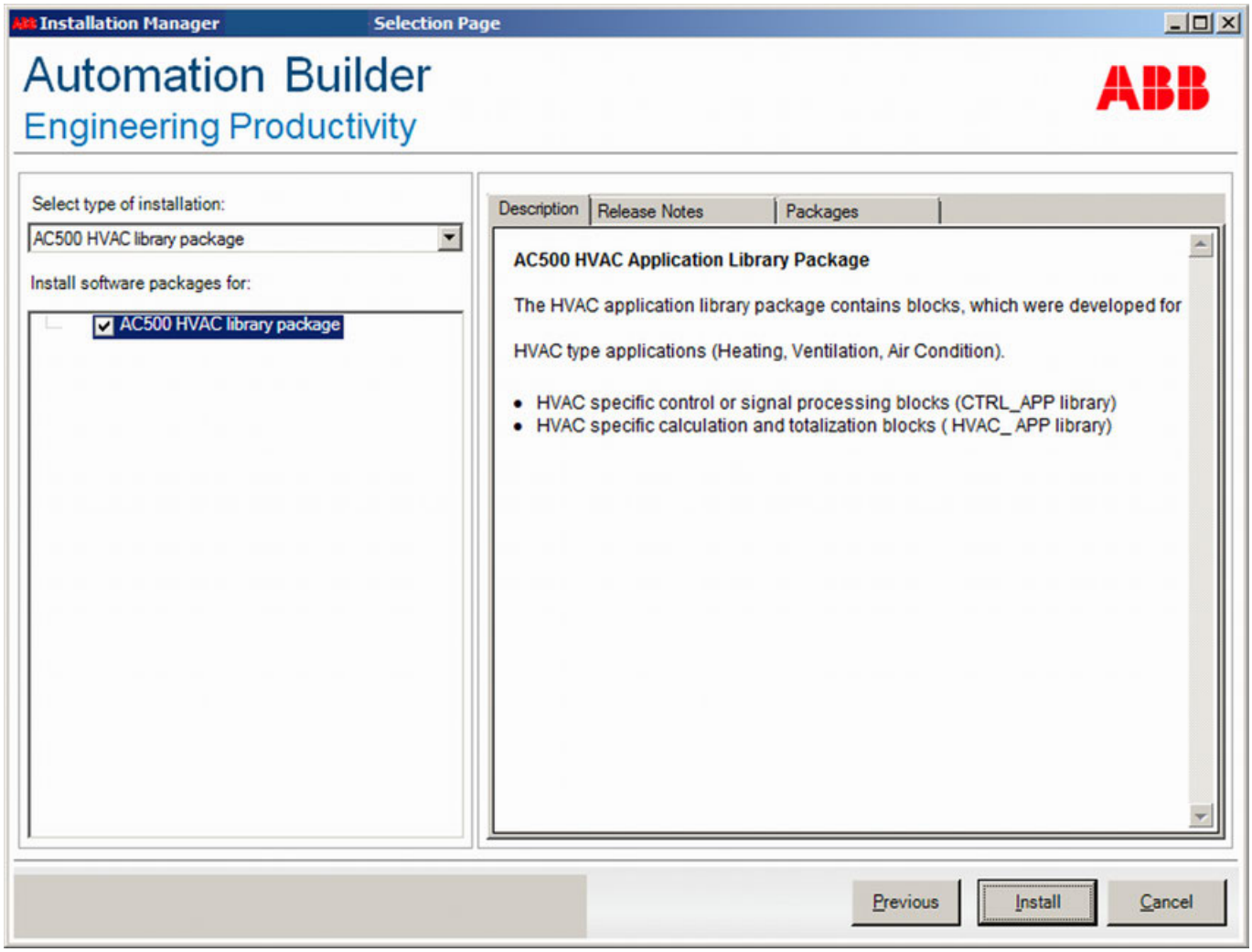
Options:

- **Installed packages**: Shows all installed packages of Automation Builder.
- **Licenses**: Displays the detailed license information of installed Automation Builder packages in the CodeMeter WebAdmin page. For more information, see *http://localhost:22350/$help/CmUserHelp/us/index.html?controlcenter.htm*.
- **Uninstall all**: Uninstalls the currently installed Automation Builder software.
- **Install Package**: Installs customer specific software packages.
- **Modify**: Adds or removes installed software packages.
- **Info Export**: Exports detailed information of installed packages in a notepad.
- **Check for Update**: Checks if your installed version of Automation Builder is up to date and checks for updates.

1.3.1 Installing customer specific package

Installation manager allows you to install customer specific software packages (CABPKG files). These packages are separately distributed to the customer based on the customer requirement.

1. In the installation manager, click **Install Package**.
2. Select the package to be installed (.cabpkg file) from the file system.



3. Select the components to be installed.
4. Click **Install**.

   ⇨ Data installation starts.
5. Successfully installed components are indicated with .

   Errors during data download are indicated with . Errors during download of any package component aborts the installation. In this case click **Show Log** and save the log data. Send the log file to ABB support team.

   Click **Finish** to end the wizard.

1.3.2 Adding or removing installed software packages

1. In the installation manager, click **Modify**.

   ⇨ The selection page opens.

   The selected software packages are installed already.

   The not selected software packages are not installed.

2. Select the software packages you want to install.

   Unselect the software packages you want to uninstall.

   > *You cannot unselect the main **ABB Automation Builder** software package.*

   If also an older Automation Builder version or Control Builder Plus version shall be installed for compatibility reasons, select the appropriate options under **Install also previous product versions**. This allows to open and edit a corresponding project in the original version without a previous project upgrade.

3. Click **Continue**.

   The following three cases are possible:

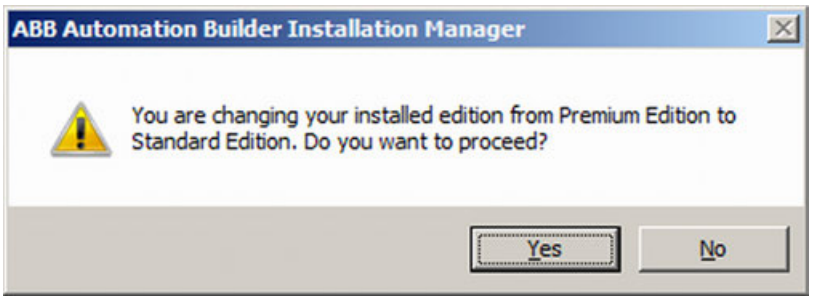
   - The selected software package starts downloading and installing.
   - The unselected software package will uninstall.
   - The unselected software package will uninstall first and then download and install the selected software package.

4. Successfully downloaded components are indicated with .

   Errors during data download are indicated with . Errors during download of any package component aborts the installation. In this case click **Show Log** and save the log data. Send the log file to ABB support team.

   Click **Finish** to end the wizard.

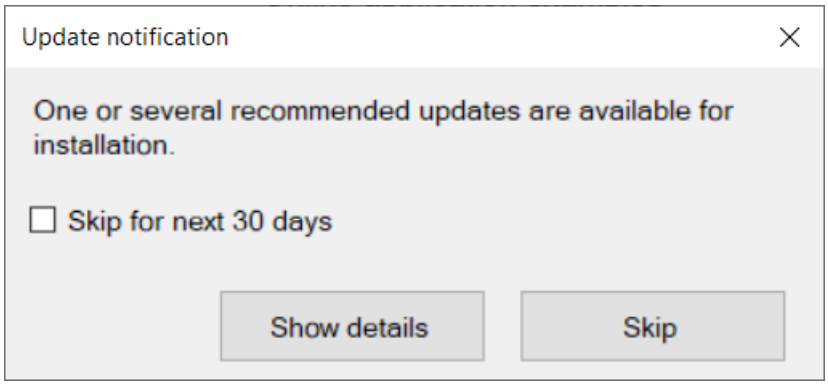
If you modify the type of installed edition, a warning message is displayed.



1.3.3 Automation Builder update notification

An update notification dialog will be shown during Automation Builder startup in case there are any updates available for the currently installed version.

- Notification on available major, minor, or service release version
- Notification on recommend software updates (Bug fixes, CM FW, V2 FW, LIB updates, documentation updates, ...), Automation Builder 2.5 and next future versions will show notification on updates.

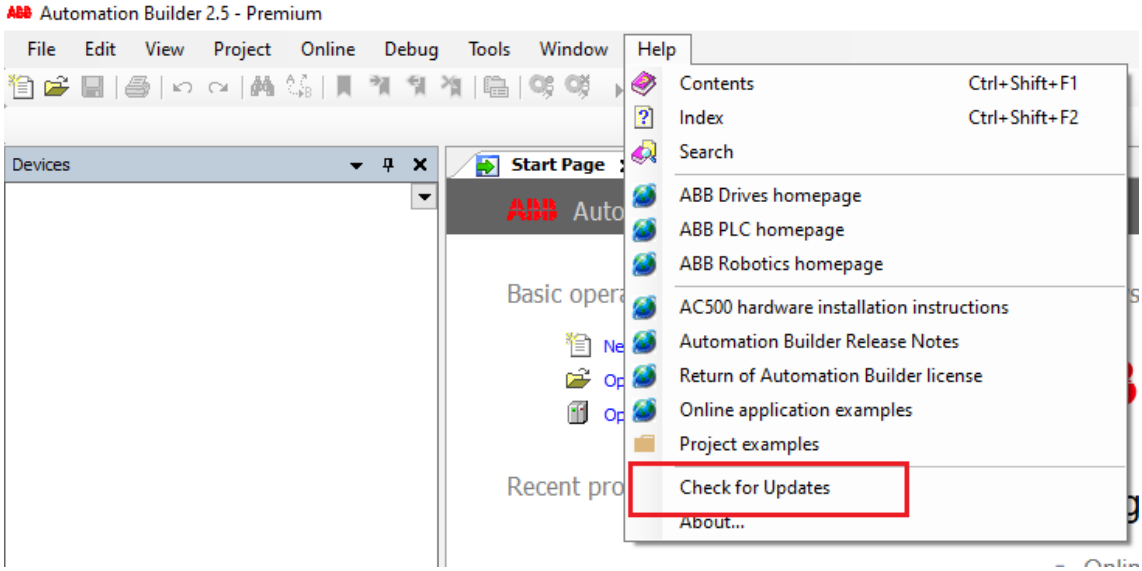


Skip of next 30 days + Skip:	Close the notification dialog. Notification dialog will not be shown for next 30 days.
Show details:	Show details will show the updates details page.
Skip:	Close the notification dialog. Next time launch of Automation Builder will show the notification dialog.

Update notifications will only be shown in the latest installed Automation Builder version profile.

"Help" - "Check for Updates" menu item

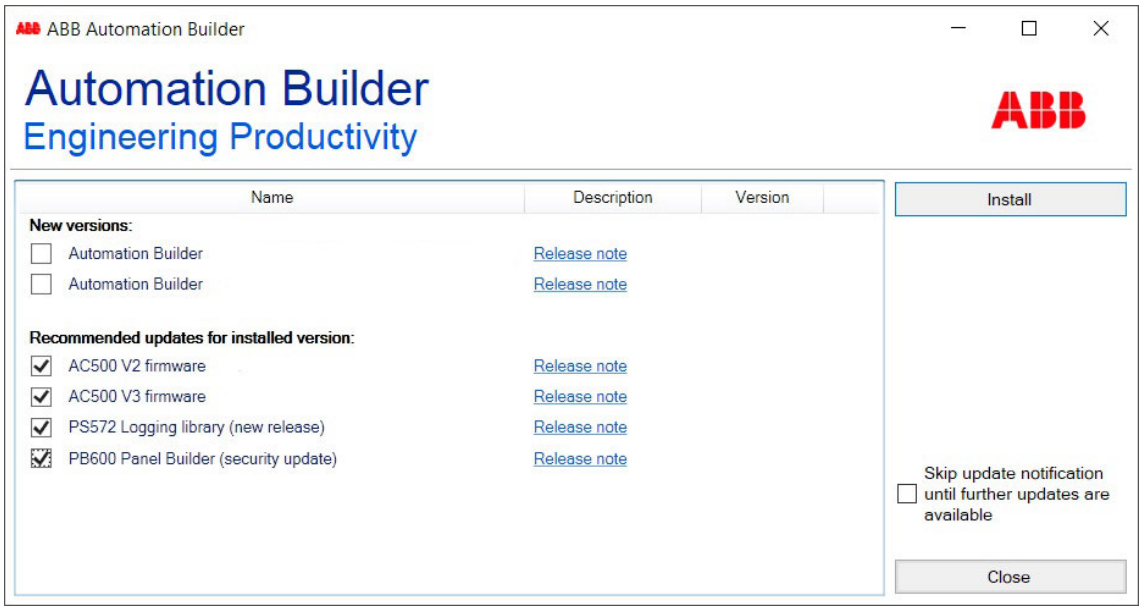
The *"Check for Updates"* menu item has been added to the *"Help"* menu. The user has the possibility to check for updates manually.



Check for Updates: Will launch the Automation Builder update details window.

Automation Builder update details window

The Automation Builder update window provides information about all available updates for the currently installed Automation Builder version and features. Detailed information is provided via the description links.



Skip update notification until further updates are available:	If this option is selected and the update details page is *"Close"*, no notification is displayed at startup until new updates are available.
New versions:	New releases of Automation Builder will be shown this section which will list hotfix version for the currently installed version or recent major version released, if any.
Recommended updates for installed version:	Updates for the currently installed options will be shown.

User can only select any one of the new versions and install.

Installed updates in the Installation Manager start page

All the installed updates will be shown in the Installation Manager start page in the *"Installed updates"* tab.

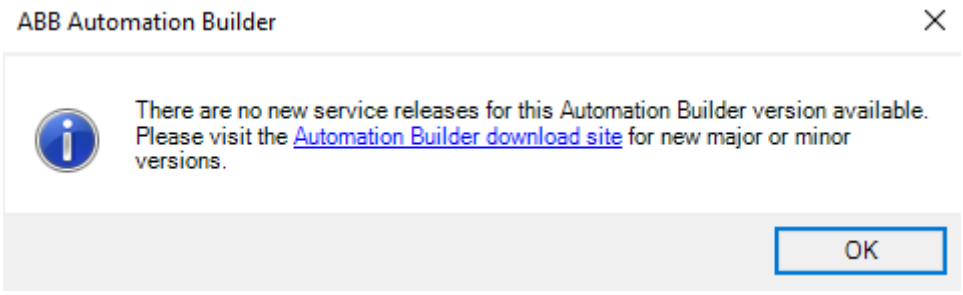
Installation Manager selection page

All the newly installed updates package version information will be updated and shown in the packages tab.

1.3.4 Checking for updates

▷ In the installation manager, click *“Check for new service release”*.

⇨ If the installed Automation Builder version is up-to-date, the following message will appear.



If a newer Automation Builder version is available, you will get an option to download and install the new version.

> *Create a project archive before updating Automation Builder. Project archives contain all project data, including data that is not stored with a *.project file, e.g. device description files for third party devices.*
>
> *Chapter 1.6.5.1.1.7.1 “Creation of an archive ” on page 5768*

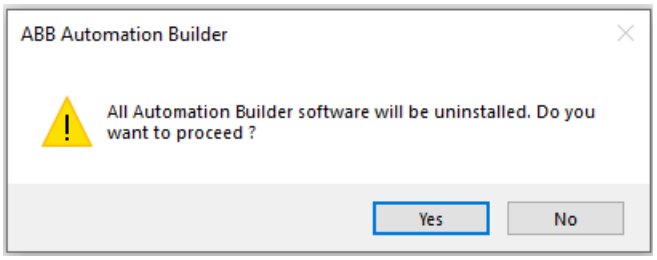
1.3.5 Uninstalling Automation Builder

Installation manager offers a comfortable way to uninstall Automation Builder software. This will uninstall all related packages of Automation Builder platform as well, such as Mint Plug-in, Automation Builder Extensions, Drive Manager etc.

1. In the installation manager, click *“Uninstall all”*.

⇨ A warning message is displayed to uninstall Automation Builder software.

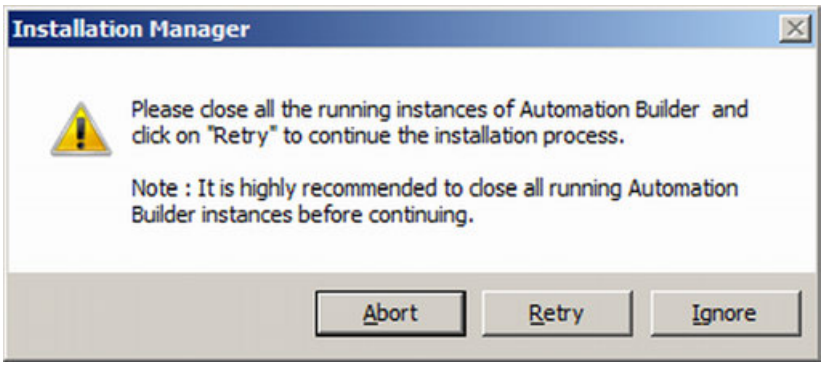
Click **Yes** to continue.



2. If Automation Builder instances are running, a warning message is displayed.

   Close running instances of Automation Builder and click **Retry** to continue uninstallation.

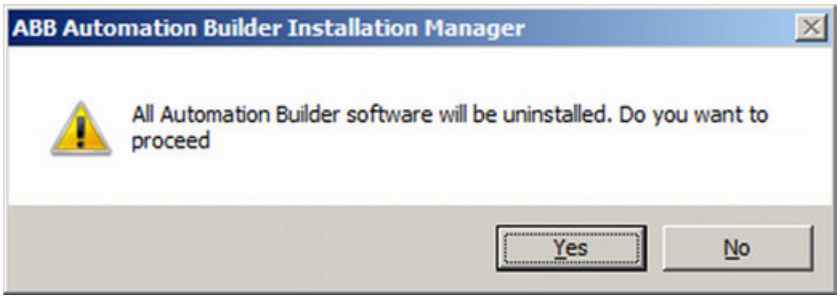
   With **Abort** uninstallation of the current package is stopped. Uninstallation is continued with the next package. With **Ignore**, uninstallation is forced. As this can lead to an erroneous uninstallation, we recommend you, *not* to use this option.



3. If installation manager was launched with *"Tools ➔ Installation Manager"*, the following message is displayed as Automation Builder is still running:

   With **Yes** Automation Builder software is closed to continue uninstallation procedure.

   With **No** uninstallation of the current package is stopped. Uninstallation is continued with the next package.



4. For each of the packages being uninstalled, system may prompt to continue uninstallation.
5. Successfully uninstalled components are indicated with ✔.

   Errors during uninstallation are indicated with ✖. Errors during uninstallation of any package component aborts the uninstallation. In this case click **Show Log** and save the log data. Send the log file to ABB support team.

   Click **Finish** to end the wizard.

1.4 Programming with CODESYS

1.4.1 Development system

1.4.1.1 Overview

1.4.1.1.1 How is a project structured?

A project is put into a file named after the project ↳ *Chapter 1.4.1.1.9.1 "Project" on page 151*. The first program organization unit created in a new project will automatically be named PLC_PRG ↳ *Chapter 1.4.1.1.9.2 "POU (program organization unit)" on page 151* ↳ *Chapter 1.4.1.1.9.8 "PLC_PRG" on page 158*. The process begins here (in compliance with the main function in a C program), and other POUs can be accessed from the same point: programs, function blocks and functions ↳ *Chapter 1.4.1.1.9.7 "Program" on page 156* ↳ *Chapter 1.4.1.1.9.4 "Function block" on page 153* ↳ *Chapter 1.4.1.1.9.3 "Function" on page 151*.

Once you have defined a Task Configuration, it is no longer necessary to create a program named PLC_PRG ↳ *Chapter 1.4.1.4.8.1 "Overview" on page 390*. You will find more about this in the Task Configuration chapter.

There are different kinds of objects in a project: POUs, data types, display elements (visualizations) and resources.

The Object organizer contains a list of all the objects in your project *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*.

1.4.1.1.2 How do I set up my project?

First you should configure your PLC in order to check the accuracy of the addresses used in the project.

Then you can create the POUs needed to solve your problem.

Now you can program the POUs you need in the desired languages.

Once the programming is complete, you can compile the project and remove errors should there be any.

1.4.1.1.3 How can I test my project?

Once all errors have been removed, activate the simulation, log in to the simulated PLC and "load" your project in the PLC *Chapter 1.4.1.1.11.8 “Simulation” on page 184*. Now you are in Online mode.

Now open the window with your PLC Configuration and test your project for correct sequence. To do this, enter input variables manually and observe whether outputs are as expected. You can also observe the value sequence of the local variables in the POUs. In the 'Watch and Recipe Manager' you can configure data records whose values you wish to examine.

1.4.1.1.4 Comparing projects

You can compare the currently open project with another project – a reference project. The differences in contents, properties, or access rights are detected and shown in a comparison view.

Clicking *“Project ➔ Compare”* opens the *“Project Compare”* dialog for you to configure and run the comparison. Then the result is shown in the comparison view *“Project Compare - Differences”* where the objects are aligned in a tree structure. Objects that indicate differences from the respective reference object are identified by colors and symbols. This is how you detect whether or not the contents, properties, or access rights are different.

For differences in the contents, you can also open the detailed compare view *“Project Compare - <object name> Differences”* in order to zoom into the object. In the detailed compare view, the contents of the object and reference object are displayed or their source code aligned. The detected differences are marked. Previously opened views are not closed. In this way, you can have any number of comparison views open and read them, in addition to the project compare view.

You can accept the detected differences from the reference project into the current project. This is possible only from the reference project into the open project. To do this, you activate differences (for example in the code) that should be accepted in the current project with the commands , , or in the active comparison view for accepting. These positions are highlighted in yellow. Make sure that any other open compare views are inactive (write-protected, read-only). therefore, you can activate differences to be accepted in exactly one comparison view only. When exiting the active compare view, if you confirm that the differences that are activated for acceptance are actually accepted into the current project, then the current project is modified.

In order to exit the project comparison completely, close the project compare view.

Creating a comparison view

Requirement: You have made changes in your current project and wish, for example, to compare it with the last-saved version. In the meantime, for example, you have added further POUs, removed a POU, changed single lines of code or the object properties in function blocks.

1. Select the command *"Project ➔ Compare"*.
   ⇨ The *"Project Comparison"* dialog box opens.
2. Enter the path to the reference project, for example the path to the last-saved version of your current project.
3. Leave the activation of the comparison option *"Ignore Spaces"* as it is.
4. Click on *"OK"*.
   ⇨ The comparison view opens. Title: *" Project Comparison – Differences"*. The Device trees of the current project and the reference project are displayed alongside each other and the changed objects are marked in color.
5. Select an object marked in blue in the tree of the reference project (right). The current project no longer contains this object.
   Click on *"Accept Single"*
   ⇨ The object is added to the tree of the current project (left). The line has a yellow background. appears in the middle column.
6. Select an object marked in green in the tree of the current project (left). The reference project does not contain this object.
   Click on *"Accept Single"*
   ⇨ The object is removed again from the tree of the current project (left). The line has a yellow background. appears in the middle column.
7. If changes are detected in the content of an object that is contained in both the current project and the reference project, this is indicated by red lettering. You can then switch to the detailed comparison view for the object by double-clicking on the object.
8. Close the comparison view and answer the query whether the changes made are to be saved with *"Yes"*.
   ⇨ The changes become effective in the project.

Opening the detailed compare view

Requirement: For example, a user modified the code in a POU of the current project. You have performed the project comparison by clicking *"Project ➔ Compare"*. The project compare view shows this POU highlighted in red in the aligned in the project tree.

1. Double-click the line of the aligned POU versions.
   ⇨ The compare view switches to the detailed compare view of the POU. The modified code lines are highlighted in gray and written in red.
2. Click .
   ⇨ Code lines with changes (red) are extended by two lines: an line with insert (left, green) and a line with delete (right, blue).
3. Click again.
   ⇨ The code line is marked again as modified.
4. Move the mouse pointer to the code line marked as modified and click *"Accept Single"*.
   ⇨ The code line from the reference project is activated for acceptance into the current project.

5. Click .
   ⇨ The project compare view opens for the entire project. It is write-protected (read-only) to prevent you from activating differences for acceptance. The link highlighted in yellow above the tree view also indicates this.
6. Click the link: *“Project compare view is read only because there are uncommitted changes in another view. Click here to switch to the modified view.”*
   ⇨ The detailed compare view opens again. The unconfirmed changes are highlighted in yellow.
7. Click ✕ in the tab of the view and confirm that the changes should be saved.
   ⇨ The detail project view is closed and the POU is overwritten. Now it corresponds to the POU of the reference project. The project view is active again so that you can continue working with project compare.

If you do not click the link, but click ✕ instead to close the editor of the project compare view, then you will also confirm the acceptance of changes into the current project. The detail changes are accepted and then the project compare is closed completely.

See also

- *Chapter 1.6.5.1.1.6 “Comparing projects” on page 5765*
- *Chapter 1.4.1.1.4.1 “Creating a comparison view” on page 147*

1.4.1.1.5 Debugging

In case of a programming error you can set a breakpoint *Chapter 1.4.1.1.11.3 “Breakpoint” on page 182*. If the process stops at such a breakpoint, you can examine the values of all project variables at this point in time. By working through sequentially (single step) you can check the logical correctness of your program.

1.4.1.1.6 Additional online functions

Further debugging functions:

You can set program variables and inputs and outputs at certain values.

You can use the flow control to check which program lines have been run.

A window log records operations, user actions and internal processes during an online session in a chronological order *Chapter 1.4.1.4.4.2 “'Window' 'Log'” on page 374*.

If activated in the target settings the Sampling Trace allows you to trace and display the actual course of variables over an extended period of time.

Also a target specific function is the PLC Browser which can serve to request certain information from the PLC.

Once the project has been set up and tested, it can be downloaded to the hardware and tested as well. The same online functions as you used with the simulation will be available.

1.4.1.1.7 Additional features

The entire project can be documented or exported to a text file at any time.

For communication purposes there is a symbolic interface and a DDE interface. A gateway server plus OPC server and DDE server are components of the standard installation packet.

A visualization can be processed target specifically to be available as web visualization and/or target visualization. This allows to run and view the visualization via internet or on a PLC monitor.

1.4.1.1.8 File types

The following file types can be created:

File extension	Example	Description	Format	Path (default)
*.pro	project01.pro	Project file	binary	project directory
*.ci	project01<number>.ci	Information on the last build (compilation) of the project → incremental compile possible; only created when project gets saved	number: coded target ID	binary
project directory	*.eci	project01<number>.eci	external Compile-Information; Subset of the ci-file in eci-format; can be read via an access-dll	number: coded target ID
PE	project directory	*.cic	project01<number>.cic	target-dependant information on the last build (compilation) of the project -> incremental compile possible; only created when project gets saved
number: coded target ID	binary	project directory	*.cit	project01<number>.cit
temporary *.ci-file; created at target change, transformed to a ci-file at next save of the project	number: coded target ID	binary	project directory	*.ri
project01<number>.ri	information on the last download, important for Online Change; created at each download	number: coded target ID	binary	project directory
*.exp	project01.exp, PLC_PRG.exp	export file ('Project' 'Export')	Export format (Text)	project directory
*.tlt	*.txt	project01.tlt	project01.txt	translation file (defined in 'Project' 'Translate in another language')
Text		*.sym	project01.sym	symbol file
Text	project directory	*.sdb	project01.sdb	symbol file
binary	project directory	*.sym_xml	project01.sym_xml	symbol file
XML	project directory	*.asd	project01.asd	save file (temporary, 'Auto save', 'Auto save before compile')

File extension	Example	Description	Format	Path (default)
binary	project directory	*.asl	lib01.asl	save file for a library opened as project (temporary, 'Auto save', 'Auto save before compile')
binary	library resp. project directory	*.bak	project01.bak	backup file for project (permanent, 'Create backup')
binary	project directory	*.prg	*.bin	default.prg
project01.prg	boot project, file name depending on target	binary	target system (created online)	project directory (created offline)
*.chk		default.chk	project01.chk	checksum for boot projekt code
binary	target system (created online)	project directory (created offline)	*.ini	codesys.ini
ini-file for various settings	Text	with codesys.exe	*.dfr	default.dfr
project01.dfr	frame file (Printer setup)	binary	with codesys.exe	*.asm
code386.asm	assembler-Listing of the created project code	Text	compile directory	*.lst
project01.lst	assembler-Listing of the created project code	Text	compile directory	
*.bpl	project01.bpl	debug-files (breakpoint-information)	Text	compile directory
	*.st	PLC_PRG.st	debug-files, implicit ST-code	Text
compile directory		*.map	project01.map	map-file; information on memory organization and variable locations
Text	compile directory		*.hex	*.h86
project01.hex (Output) resp. standard.hex (Lib)	.hex for Intel or Motorola, .h86 for Intel; compiler output or input for external library	Intel or Motorola hex-files	compile directory	resp. library directory
*.trd	projectxy0.trd	trend logging (number befor the dot is counted up, if file is full and another must be created)	Text	project directory
*.log	projectxy.log	log file (Log)	binary	project directory
*.wtc	projx_watch1.wtc	watch list (Watch- and Recipe-Manager)	Text	user defined directory

File extension	Example	Description	Format	Path (default)
*.alm	alarmlog0.alm	alarm log-file		user defined directory resp. download-directory of the controller
*.zip	projectxy.zip	project archive file; zip-file with files belonging to the project , 'File' 'Save/ Mail Archive'		user defined directory
*.trc	project01_tr1.trc	trace recording	binary	with codesys.exe
*.mon	project01_tr1.mon	trace recording	XML	with codesys.exe
*.tcf	project01_tr1.tcf	trace configuration	binary	with codesys.exe

1.4.1.1.9 Project components

Project

A project contains all of the objects in a PLC program. A project is saved in a file named after the project. The following objects are included in a project:

POUs (program organization units), data types, visualizations, resources, and libraries.

POU (program organization unit)

Functions, function blocks, and programs are POUs which can be supplemented by actions.

Each POU consists of a declaration part and a body. The body is written in one of the IEC programming languages which include IL, ST, SFC, FBD, LD or CFC. All IEC standard POUs are supported. If you want to use these POUs in your project, you must include the library standard.lib in your project.

POUs can call up other POUs. However, recursions are not allowed.

Function

A function is a POU, which yields exactly one data element (which can consist of several elements, such as fields or structures) when it is processed, and whose call in textual languages can occur as an operator in expressions.

When declaring a function do not forget that the function must receive a type. This means, after the function name, you must enter a colon followed by a type.

Example A correct function declaration can look like this:

```
FUNCTION Fct: INT
```

In addition, a result must be assigned to the function. That means that function name is used as an output variable.

A function declaration begins with the keyword FUNCTION. Regard the recommendations on the naming *Chapter 1.4.1.3.9.2.1 “Recommendations on the naming of identifiers” on page 297*.

Example of a function Fct in IL, in which three input variables are declared:

The first both input variables get multiplicated and then divided by the third one. The function returns the result of this operation:

Declaration part:

```
FUNCTION Fct: INT
VAR_INPUT
PAR1:INT;
PAR2:INT;
PAR3:INT;
END_VAR
```

Implementation part:

```
LD PAR1
MUL PAR2
DIV PAR3
ST Fct
```

Calling a function:

In ST a function call can be used as operand in an expression.

In SFC a function call can only take place within actions of a step or a transition.

CODESYS allows using global variables within a function. This intentionally deviates from the IEC61131-3 standard, which prescribes that return value of a function only will be modified by the input parameters. Thus the difference between functions and programs is just, that functions return only one return value and that their parameters and return values are handled over the stack.

Examples for calling the above described function Fct:

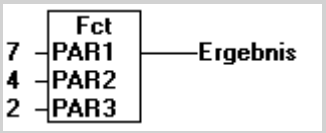
In IL:

```
LD 7
Fct 2,4
ST  Ergebnis
```

In ST:

```
Ergebnis := Fct(7, 2, 4);
```

In FUP:



NOTICE!

If a local variable in a function is declared as RETAIN, this will be without any effect. The variable will not be saved in the Retain area!

- *If you define a function in your project with the name CheckBounds, you can use it to check for range overflows in arrays ↳ Chapter 1.4.1.8.2.1 “ARRAY” on page 445.*
- *If you define functions in your project with the names CheckDivByte, CheckDivWord, CheckDivDWord and CheckDivReal, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0 ↳ Chapter 1.4.1.6.2.4 “DIV” on page 408.*
- *If you define functions with the names CheckRangeSigned and CheckRangeUnsigned, then range exceeding of variables declared with subrange types can be intercepted ↳ Chapter 1.4.1.8.2.7 “Subrange types” on page 450.*

All these check function names are reserved for the described usage.

Function block

A function block is a POU which provides one or more values during the procedure.

As opposed to a function, a function block provides no return value ↳ *Chapter 1.4.1.1.9.3 “Function” on page 151*.

A function block declaration begins with the keyword FUNCTION_BLOCK. Regard the recommendations on the naming ↳ *Chapter 1.4.1.3.9.2.1 “Recommendations on the naming of identifiers” on page 297*.

Reproductions or instances (copies) of a function block can be created ↳ *Chapter 1.4.1.1.9.5 “Function block instances” on page 153*.

The call of a function block is done via a function block instance ↳ *Chapter 1.4.1.1.9.6 “Calling a function block” on page 154*.

Example in IL of a function block with two input variables and two output variables:

One output (iMulErg) is the product of the two inputs, the other (xVergl) is a comparison for equality:

Declaration part:

```
FUNCTION_BLOCK FUB
VAR_INPUT
    iPar1:INT;
    iPar2:INT;
END_VAR
VAR_OUTPUT
    iMulErg:INT;
    xVergl:BOOL;
END_VAR
```

Implementation part in IL:

```
LD iPar1
MUL iPar2
ST iMulErg

LD iPar1
EQ iPar2
ST xVergl
```

Function block instances

Reproductions or instances (copies) of a function block can be created ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*.

Each instance possesses its own identifier (the Instance name), and a data structure which contains its inputs, outputs, and internal variables. Instances are declared locally or globally as variables, whereas the name of the function block is indicated as the type of an identifier.

Regard the Recommendations on the naming *Chapter 1.4.1.3.9.2.1 “Recommendations on the naming of identifiers” on page 297*.

Example

Example of an instance with the name INSTANCE of the FUB function block:

```
fubInstance: FUB;
```

Function blocks are always called through the instances described above.

Only the input and output parameters can be accessed from outside of an function block instance, not its internal variables.

Example for accessing an input variable

The function block FB has an input variable in1 of the type INT.

```
PROGRAM prog

VAR
fbinst1:fb;
END_VAR

LD 17
ST fbinst1.in1
CAL fbinst1

END_PROGRAM
```

The declaration parts of function blocks and programs can contain instance declarations *Chapter 1.4.1.1.9.4 “Function block” on page 153*. Instance declarations are not permitted in functions *Chapter 1.4.1.1.9.3 “Function” on page 151*.

Access to a function block instance is limited to the POU in which it was declared unless it was declared globally.

The instance name of a function block instance can be used as the input for a function or a function block.

All values are retained after processing a function block until the next it is processed. Therefore, function block calls with the same arguments do not always return the same output values!

If at least one of the function block variables is a retain variable, the total instance is stored in the retain area.

Calling a function block

The input and output variables of a function block can be accessed from another POU by setting up an instance of the function block and specifying the desired variable using the following syntax *Chapter 1.4.1.1.9.5 “Function block instances” on page 153*:

<Instance name>.<Variable name>

Assigning parameters at call:

If you would like to set input and/or output parameters when you call the function block, you can do this in the text languages IL and ST by assigning values to the parameters after the instance name of the function block in parentheses (for input parameters this assignment takes place using ":=" just as with the initialization of variables at the declaration position, for output parameters "=>" is to be used) ↳ *Chapter 1.4.1.3.9.11 “Variables declaration” on page 303*.

If the instance is inserted via input assistant (<F2>) with option With arguments in the implementation window of a ST or IL POU, it will be displayed automatically according to this syntax with all of its parameters. But you not necessarily must assign these parameters.

Example:

FBINST is a local variable of type of a function block, which contains the input variable xx and the output variable yy. When FBINST is inserted in a ST program via input assistant, the call will be displayed as follows and then can be supplemented with the desired values: FBINST1(xx:= , yy=>);

InOutVariables at call:

Please regard, that the InOutVariables (VAR_IN_OUT) of a function block are handed over as pointers. For this reason in a call of a function block no constants can be assigned to VAR_IN_OUTs and there is no read or write access from outside to them.

Example

Calling a VAR_IN_OUT variable inout1 of function block fubo in a ST module:

```
VAR
 fuboinst:fubo;
 iVar1:int;
END_VAR
iVar1:=2;
fuboinst(iInOut1:=iVar1);
```

not allowed in this case: "fuboinst(iInOut1:=2);" or "fuboinst.iInOut1:=2;"

Examples for calling function block FUB:

Declaration part:

```
FUNCTION_BLOCK FUB
VAR_INPUT
    iPAR1:INT;
    iPAR2:INT;
END_VAR
VAR_OUTPUT
  iMELERG:INT;
  xVERGL:BOOL;
END_VAR
```

Implementation part in AWL:

```
LD iPar1
MUL iPar2
ST iMulErg

LD iPar1
EQ iPar2
ST xVergl
```

The multiplication result is saved in the variable ERG, and the result of the comparison is saved in QUAD. An instance of FUB with the name INSTANCE is declared.

Calling FUB in IL:

Declaration part:

```
PROGRAM AWLaufruf
VAR
      xQuad :      BOOL;
      fubinstanz : FUB;
      iErg:        INT:=0;
END_VAR
```

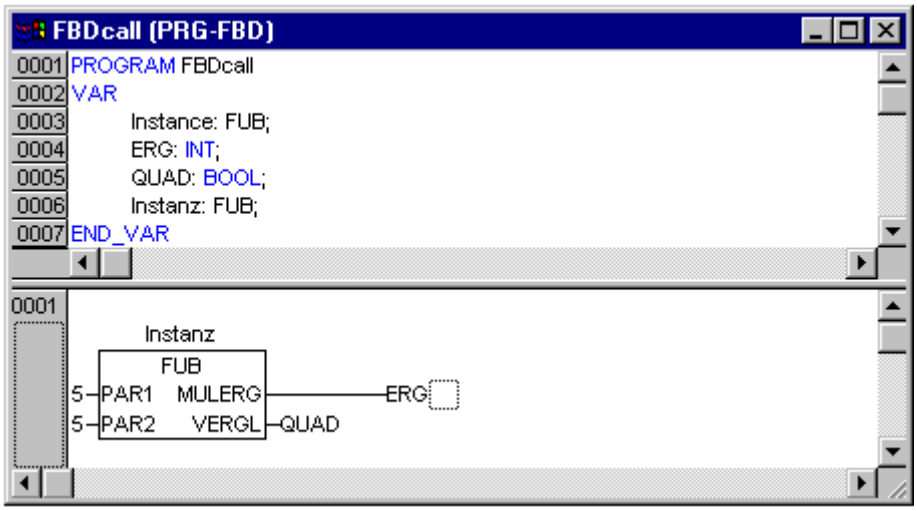
Implementation part:

```
CAL  fubinstanz(iPar1:=5;iPar2:=5);
LD    fubinstanz.xVergl
ST    xQuad
LD    fubinstanz.iMulErg
ST    iErg
```

Calling FUB in ST (declaration part as shown above for IL):

```
PROGRAM STaufruf
fubinstanz(iPar1:=5;iPar2:=5);   bzw. fubinstanz;
QUAD:=fubinstanz.xVergl;
ERG:=funbinstanz.iMulErg;
```

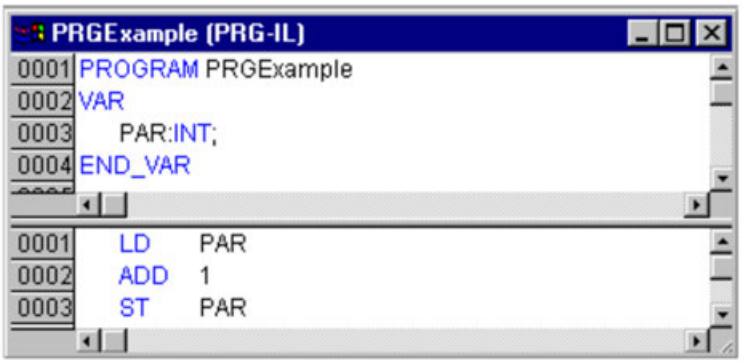
Calling FUB in FBD (declaration part as shown above for IL):



In SFC function block calls can only take place in steps.

Program

A program is a POU which returns several values during operation. Programs are recognized globally throughout the project. All values are retained from the last time the program was run until the next.



Programs can be called. A program call in a function is not allowed. There are also no instances of programs ↳ *Chapter 1.4.1.1.9.3 “Function” on page 151*.

If a POU calls a program, and if thereby values of the program are changed, then these changes are retained the next time the program is called, even if the program has been called from within another POU.

This is different from calling a function block. There only the values in the given instance of a function block are changed.

These changes therefore play a role only when the same instance is called.

A program declaration begins with the keyword PROGRAM. Regard the Recommendations on the naming of identifiers ↳ *Chapter 1.4.1.3.9.2.1 “Recommendations on the naming of identifiers” on page 297*.

If you would like to set input and/or output parameters when you call the program, you can do this in the text languages IL and ST by assigning values to the parameters after the program name in parentheses (for input parameters this assignment takes place using ":=" just as with the initialization of variables at the declaration position, for output parameters "=>" is to be used) ↳ *Chapter 1.4.1.3.9.11 “Variables declaration” on page 303*.

If the program is inserted via input assistant (<F2>) with option With arguments in the implementation window of a ST or IL POU, it will be displayed automatically according to this syntax with all of its parameters. But you not necessarily must assign these parameters.

Examples for program calls

In IL

```
CAL PRGexample2

LD PRGexample2.out_var

ST  ERG

or with assigning the parameters (input assistant "With arguments",
see above):

CAL PRGexample2(in_var:=33, out_var=>erg )
```

In ST

```
PRGexample2;

Erg := PRGexample2.out_var;

or with assigning the parameters (input assistant "With arguments",
see above):

PRGexample2(in_var:=33, out_var=>erg );
```

In FBD

In IL

```
CAL PRGexample2

LD PRGexample2.out_var

ST  ERG

or with assigning the parameters (input assistant "With arguments",
see above):

CAL PRGexample2(in_var:=33, out_var=>erg )
```

Example for a possible call sequence for PLC_PRG

See the program PRGexample shown in the picture at top of this chapter:

```
LD 0

ST  PRGexample.PAR (*Default setting for PAR is 0*)

CAL IL call (*ERG in IL call results in 1*)

CAL ST call (*ERG in ST call results in 2*)

CAL FBD call (*ERG in FBD call results in 3*)
```

If the variable PAR from the program PRGexample is initialized by a main program with 0, and then one after the other programs are called with above named program calls, then the ERG result in the programs will have the values 1, 2, and 3. If one exchanges the sequence of the calls, then the values of the given result parameters also change in a corresponding fashion.

PLC_PRG

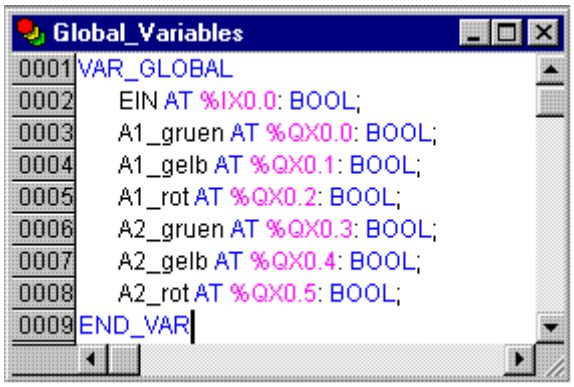
We have defined and correlated the time sequencing of the phases for both sets of traffic lights in the block SEQUENCE. Since, however, we see the traffic lights system as a module of a bus system, e.g. CAN bus, we have to make input and output variables available in the block PLC_PRG. We want to start-up the traffic lights system over an ON switch and we want to send each of the six lamps (each traffic light red, green, yellow) the corresponding "signal command" for each step of the SEQUENCE. We are now declaring appropriate Boolean variables for these six outputs and one input, before we create the programme in the editor, and are allocating them, at the same time, to the corresponding IEC addresses.

The next step is declare the variables Light1 and Light2 of the type Phases in the declaration editor.

These deliver the Boolean value of each of the six lights to the above mentioned six outputs for each step of the block SEQUENCE. We are not, however, declaring the output variables which are foreseen within the PLC_PRG block but under Resources for Global Variables instead. The Boolean input variable IN, which is used to set the variable START in the block SEQUENCE to TRUE, can be set in the same way. ON is also allocated to an IEC address.

Select the tab Resources and open the list Global Variables.

Make the declaration as follows:



The name of the variable (e.g. IN) is followed, after AT, by a percent sign which begins the IEC address. I stands for input, Q for output, B (used in the example) stands for byte and the individual bits of the module are addressed using 0.0 (0.1, 0.2, etc.). We will not do the needed controller configuration here in the example, because it depends on which target package you have available on your computer.

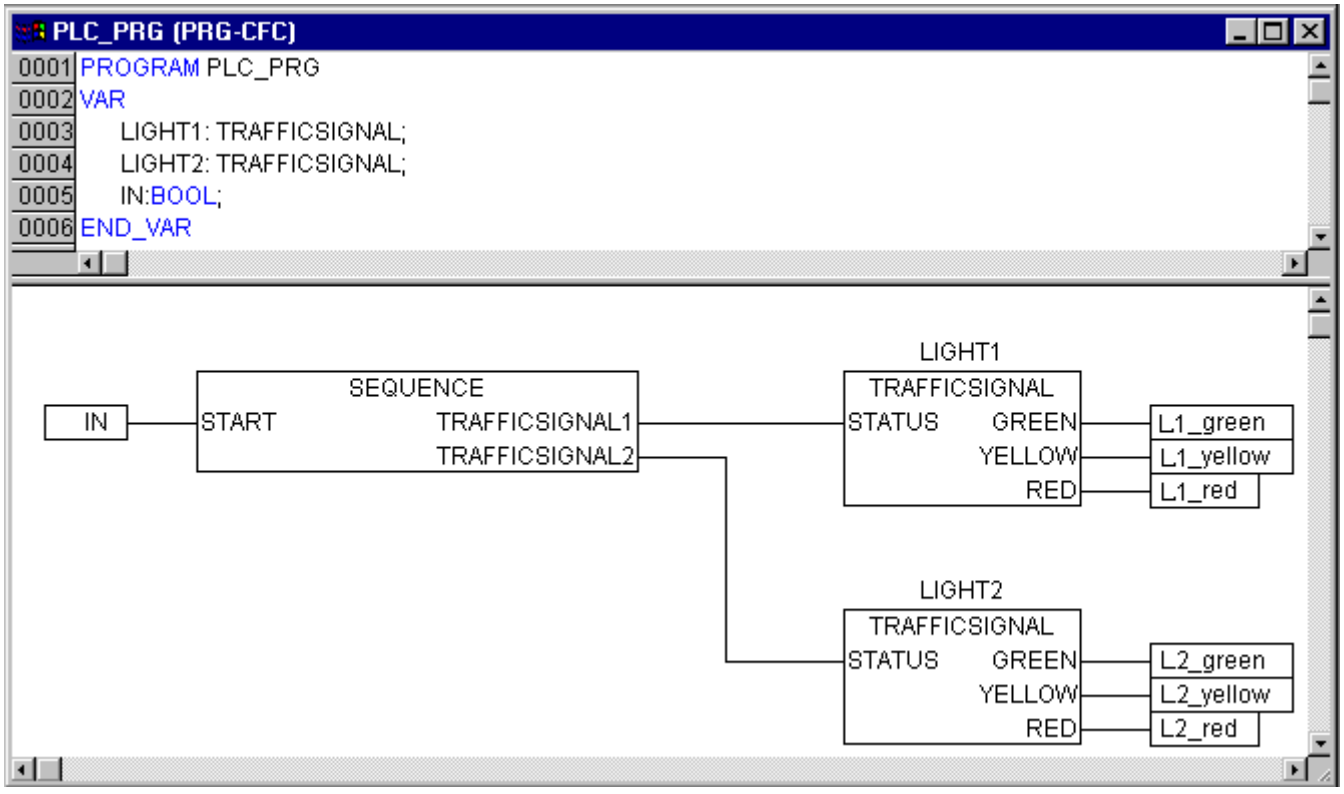
We now want to finish off the block PLC_PRG.

For this we go into the editor window. We have selected the Continuous Function Chart editor and we consequently obtain, under the menu bar, a CFC symbol bar with all of the available elements.

Click on the right mouse key in the editor window and select the element Box. Click on the text AND and write "SEQUENCE" instead. This brings up the block SEQUENCE with all of the already defined input and output variables. Insert two further block elements which you name PHASES. Phases is a function block and this causes you to obtain three red question marks over the block which you replace with the already locally declared variables LIGHT1 and LIGHT2. Now set an element of the type Input, which award the title ON and six elements of the type Output which you award variable names to, as described, namely L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red.

All of the elements of the programme are now in place and you can connect the inputs and outputs, by clicking on the short line at the input/output of an element and dragging this with a constantly depressed mouse key to the input/output of the desired element.

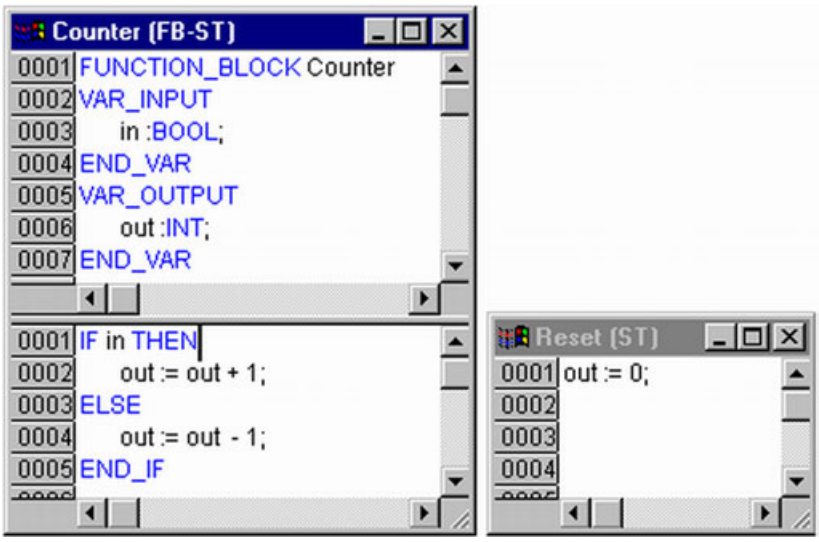
Your program should finally look like the example shown here.



Action

Actions can be defined and assigned to function blocks and programs ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153* ↳ *Chapter 1.4.1.1.9.7 “Program” on page 156* ↳ *Chapter 1.4.1.2.4.14 “'Project' 'Add action'” on page 263*. The action represents a further implementation which can be entirely created in another language as the "normal" implementation. Each action is given a name.

An action works with the data from the function block or programme which it belongs to. The action uses the same input/output variables and local variables as the "normal" implementation uses.



In the example given, calling up the function block Counter increases or decreases the output variable "out", depending on the value of the input variable "in". Calling up the action Reset of the function block sets the output variable to zero. The same variable "out" is written in both cases.

Calling an action:

An action is called up with <Program_name>.<Action_name> or <Instance_name>.<Action_name>. Regard the notation in FBD (see example below) ! If it is required to call up the action within its own block, one just uses the name of the action in the text editors and in the graphic form the function block call up without instance information ↳ *Chapter 1.4.1.1.9.5 “Function block instances” on page 153*.

Examples for calls of the above described action from another POU

Declaration for all examples:

```
PROGRAM PLC_PRG
  VAR
   Inst : Counter;
  END_VAR
```

Call of action 'Reset' in another POU, which is programmed in IL:

```
CAL Inst.Reset(In := FALSE)
  LD Inst.out
  ST  ERG
```

Call of action 'Reset' in another POU, which is programmed in ST:

```
Inst.Reset(In := FALSE);
  Erg := Inst.out;
```

Call of action 'Reset' in another POU, which is programmed in FBD:

Actions play an important role in blocks in sequential function charts. The IEC standard does not recognise actions other than actions of the sequential function chart (SFC) ↳ *Chapter 1.4.1.1.10.5.1 “Overview” on page 171.*

Resources

You need the resources for configuring and organizing your project and for tracing variable values:

- Global variables which can be used throughout the project or network
- Library manager for adding libraries to the project ↳ *Chapter 1.4.1.4.3.1 “Overview” on page 371*
- Log for recording the actions during an online session ↳ *Chapter 1.4.1.4.4.1 “Overview” on page 374*
- Alarm configuration for the configuration of the alarm handling in the project ↳ *Chapter 1.4.1.4.2.1 “Overview” on page 363*
- PLC configuration for configuring your hardware
- Task configuration for guiding your program through tasks ↳ *Chapter 1.4.1.4.8.1 “Overview” on page 390*
- Watch and receipt manager for displaying variable values and setting default variable values ↳ *Chapter 1.4.1.4.9.1 “Overview” on page 395*
- Target settings for selection and if necessary final configuration of the target system ↳ *Chapter 1.4.1.4.7.1 “Overview” on page 387*
- Workspace as an image of the project options ↳ *Chapter 1.4.1.2.1.5 “Workspace” on page 199*

Depending on the target system and on the target settings further resources might be available in your project. ↳ *Chapter 1.4.1.4 “The 'Resources' tab” on page 357*

Libraries

You can include in your project a series of libraries whose POUs, data types, and global variables you can use just like user-defined variables. The libraries standard.lib and util.lib are standard parts of the program and are always at your disposal *Chapter 1.4.1.4.3.1 “Overview” on page 371*.

Data types

Along with the standard data types the user can define his own data types. Structures, enumeration types and references can be created *Chapter 1.4.1.8.1.1 “Data types” on page 443*.

Visualization

With visualizations you can display your project variables. You can plot geometric elements off-line with the help of the visualization. In Online mode, these can then change their form/color/text output in response to specified variable values.

A visualization can be used as a pure operating interface for a PLC with HMI or as a web visualization or target visualization running via internet resp. directly on the PLC *Chapter 1.4.3.1 “Overview” on page 636*.

1.4.1.1.10 Languages

Supported languages

All languages described by the standard IEC-61131 are supported.

Textual languages:

- Instruction list (IL) *Chapter 1.4.1.1.10.3.1 “Overview” on page 163*
- Structured Text (ST) *Chapter 1.4.1.1.10.4.1 “Overview” on page 165*

Graphic languages:

- Sequential function chart (SFC) *Chapter 1.4.1.1.10.5.1 “Overview” on page 171*
- Function Block Diagram (FBD) *Chapter 1.4.1.1.10.2 “Function Block Diagram (FBD)” on page 162*
- Ladder Diagram (LD) *Chapter 1.4.1.1.10.7.1 “Overview” on page 176*

Additionally, there is available the graphic continuous function chart (CFC), based on the Function Block Diagram *Chapter 1.4.1.1.10.6 “The continuous function chart (CFC)” on page 176*.

Function Block Diagram (FBD)

The Function Block Diagram is a graphically oriented programming language. It works with a list of networks whereby each network contains a structure which represents either a logical or arithmetic expression, the call of a function block, a jump, or a return instruction *Chapter 1.4.1.1.9.4 “Function block” on page 153*.

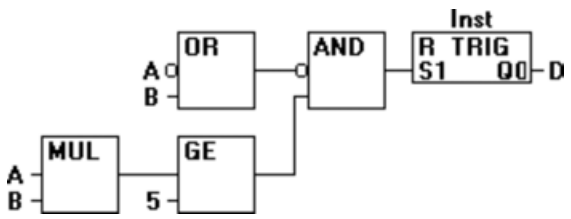


Fig. 10: Example of a network in the Function Block Diagram

See also:

- The graphic editors *Chapter 1.4.1.3.11.1 “Overview” on page 314*
- The Function Block Diagram Editor *Chapter 1.4.1.3.11.7.1 “Overview” on page 317*

Instruction list (IL)

Overview

An instruction list (IL) consists of a series of instructions. Each instruction begins in a new line and contains an operator and, depending on the type of operation, one or more operands separated by commas.

In front of an instruction there can be an identification mark (label) followed by a colon (:).

A comment must be the last element in a line. Empty lines can be inserted between instructions.

Example:

```
LD 17
ST lint (* Kommentar *)
GE 5
JMPC next
LD idword
EQ istruct.sdword
STN test
next:
```

See also:

Modifiers and operators in IL *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

Modifiers and operators in IL

In the IL language the following operators and modifiers can be used.

Table 8: Modifiers

C	with JMP, CAL, RET:	The instruction is only then executed if the result of the preceding expression is TRUE.
N	with JMPC, CALC, RETC:	The instruction is only then executed if the result of the preceding expression is FALSE.
N	otherwise:	Negation of the operand (not of the accumulator) Below you find a table of all operators in IL with their possible modifiers and the relevant meaning:

Table 9: Operators in IL

Operator	Modifiers	Meaning
LD	N	Make current result equal to the operand
ST	N	Save current result at the position of the operand
S		Then put the Boolean operand exactly at TRUE if the current result is TRUE
R		Then put the Boolean operand exactly at FALSE if the current result is TRUE
AND	N,(	Bitwise AND
OR	N,(	Bitwise OR
XOR	N,(	Bitwise exclusive OR
ADD	(	Addition
SUB	(	Subtraction
MUL	(	Multiplication
DIV	(	Division
GT	(	>
GE	(	>=
EQ	(	=
NE	(	<>
LE	(	<=
LT	(	<
JMP	CN	Jump to the label
CAL	CN	Call programor function block or
RET	CN	Leave POU and return to caller.
)		Evaluate deferred operation

Example of an IL program while using some modifiers:

LD	TRUE	(*load TRUE in the accumulator*)
ANDN	BOOL1	(*execute AND with the negated value of the BOOL1 variable*)
JMPC	mark	(*if the result was TRUE, then jump to the label "mark"*)
LDN	BOOL2	(*save the negated value of *)
ST	ERG	(*BOOL2 in ERG*)
label:		
LD	BOOL2	(*save the value of *)
ST	ERG	*BOOL2 in ERG*)

It is also possible in IL to put parentheses after an operation. The value of the parenthesis is then considered as an operand.

Example

```
LD  2
MUL 2
ADD 3
Erg
```

Here is the value of Erg 7. However, if one puts parentheses:

```
LD 2
MUL (2
ADD 3
)
ST Erg
```

The resulting value for Erg is 10, the operation MUL is only then evaluated if you come to ")"; as operand for MUL 5 is then calculated.

Structured Text (ST)

Overview

The Structured Text consists of a series of instructions which, as determined in high level languages, ("IF..THEN..ELSE") or in loops (WHILE..DO) can be executed.

Example

```
If value < 7 THEN
 WHILE value < 8 DO
  value:=value+1;
 END_WHILE;
END_IF;
```

Identifier

An identifier is a sequence of letters, numbers, and underscores that begins with a letter or an underscore.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The first 32 characters are significant.

Expressions

An expression is a construction which returns a value after its evaluation.

Expressions are composed of operators and operands. An operand can be a constant, a variable, a function call, or another expression.

Valuation of expressions

The evaluation of expression takes place by means of processing the operators according to certain binding rules. The operator with the strongest binding is processed first, then the operator with the next strongest binding, etc., until all operators have been processed.

Operators with equal binding strength are processed from left to right.

Table 10: ST operators in the order of their binding strength

Operation	Symbol	Binding strength
Put in parentheses	(expression)	Strongest binding
Function call	Function name (parameter list)	
Exponentiation	EXPT	
Negate Building of complements	- NOT	
Multiply Divide Modulo	* / MOD	
Add Subtract	+ -	
Compare	<,>,<=,>=	
Equal to Not equal to	= <>	
Boolean AND	AND	
Boolean XOR	XOR	
Boolean OR	OR	Weakest binding

Table 11: Instructions in ST

Instruction type	Example
Assignment	A:=B; CV := CV + 1; C:=SIN(X);
Calling a function block and use of the FB output	CMD_TMR(IN := %IX5, PT := 300); A:=CMD_TMR.Q
RETURN	RETURN;

IF	D:=B*B;
	IF D<0.0 THEN
	C:=A;
	ELSIF D=0.0 THEN
	C:=B;
	ELSE
	C:=D;
	END_IF;

CASE	CASE INT1 OF 1: BOOL1 := TRUE; 2: BOOL2 := TRUE; ELSE BOOL1 := FALSE; BOOL2 := FALSE; END_CASE;

FOR	J:=101;
	FOR I:=1 TO 100 BY 2 DO
	IF ARR[I] = 70 THEN
	J:=I; EXIT;
	END_IF;
	END_FOR;

WHILE	J:=1; WHILE J<= 100 AND ARR[J] <> 70 DO J:=J+2; END_WHILE;
REPEAT	J:=-1; REPEAT J:=J+2; UNTIL J= 101 OR ARR[J] = 70 END_REPEAT;
EXIT	EXIT;
Empty instruction	;

Assignment operator

On the left side of an assignment there is an operand (variable, address) to which is assigned the value of the expression on the right side with the assignment operator :=

Example:

```
Var1 := Var2 * 10;
```

After completion of this line Var1 has the tenfold value of Var2.

Calling function blocks in ST

A function block is called in ST by writing the name of the instance of the function block and then assigning the values of the parameters in parentheses *Chapter 1.4.1.1.9.4 “Function block” on page 153* *Chapter 1.4.1.1.9.5 “Function block instances” on page 153*. In the following example a timer is called with assignments for the parameters IN and PT. Then the result variable Q is assigned to the variable A.

The result variable, as in IL, is addressed with the name of the function block, a following point, and the name of the variable:

```
CMD_TMR(IN := %IX5, PT := 300);
A:=CMD_TMR.Q
```

RETURN instruction

The RETURN instruction can be used to leave a POU, for example depending on a condition.

IF instruction

With the IF instruction you can check a condition and, depending upon this condition, execute instructions.

Syntax:

```
IF <Boolean_expression1> THEN
    <IF_instructions>
{ELSIF <Boolean_expression2> THEN
    <ELSIF_instructions1>
.
.
ELSIF <Boolean_expression n> THEN
    <ELSIF_instructions n-1>
ELSE
<ELSE_instructions>}
END_IF;
```

The part in braces {} is optional.

If the <Boolean_expression1> returns TRUE, then only the <IF_Instructions> are executed and none of the other instructions.

Otherwise the Boolean expressions, beginning with <Boolean_expression2>, are evaluated one after the other until one of the expressions returns TRUE. Then only those instructions after this Boolean expression and before the next ELSE or ELSIF are evaluated.

If none of the Boolean expressions produce TRUE, then only the <ELSE_instructions> are evaluated.

Example:

```
IF temp<17
THEN heating_on := TRUE;
ELSE heating_on := FALSE;
END_IF;
```

Here the heating is turned on when the temperature sinks below 17 degrees. Otherwise it remains off.

CASE instruction

With the CASE instructions one can combine several conditioned instructions with the same condition variable in one construct.

Syntax:

```
CASE <Var1> OF
<Value1>: <Instruction 1>
<Value2>: <Instruction 2>
<Value3, Value4, Value5>: <Instruction 3>
<Value6 .. Value10>: <Instruction 4>

...

<Value n>: <Instruction n>
ELSE <ELSE instruction>
END_CASE;
```

A CASE instruction is processed according to the following model:

- If the variable in <Var1> has the value <Value i>, then the instruction <Instruction i> is executed.
- If <Var 1> has none of the indicated values, then the <ELSE Instruction> is executed.
- If the same instruction is to be executed for several values of the variables, then one can write these values one after the other separated by commas, and thus condition the common execution.
- If the same instruction is to be executed for a range of values of a variable, one can write the initial value and the end value separated by two dots one after the other. So you can condition the common condition.

Example:

```
CASE INT1 OF
1, 5: BOOL1 := TRUE;
 BOOL3 := FALSE;
2: BOOL2 := FALSE;
 BOOL3 := TRUE;
10..20: BOOL1 := TRUE;
 BOOL3:= TRUE;
ELSE
 BOOL1 := NOT BOOL1;
 BOOL2 := BOOL1 OR BOOL2;
END_CASE;
```

FOR loop

With the FOR loop one can program repeated processes.

Syntax:

```
INT_Var :INT;
FOR <INT_Var> := <INIT_VALUE> TO <END_VALUE> {BY <Step size>} DO
  <Instructions>
END_FOR;
```

The part in braces {} is optional.

The <Instructions> are executed as long as the counter <INT_Var> is not greater than the <END_VALUE>. This is checked before executing the <Instructions> so that the <instructions> are never executed if <INIT_VALUE> is greater than <END_VALUE>.

When <Instructions> are executed, <INT_Var> is always increased by <Step size>. The step size can have any integer value. If it is missing, then it is set to 1. The loop must also end since <INT_Var> only becomes greater.

Example:

```
FOR Counter:=1 TO 5 BY 1 DO
Var1:=Var1*2;
END_FOR;
Erg:=Var1;
```

Let us assume that the default setting for Var1 is the value 1. Then it will have the value 32 after the FOR loop.

<END_VALUE> must not be equal to the limit value of the counter <INT_VAR>. For example: If the variable Counter is of type SINT and if <END_VALUE> is 127, you will get an endless loop.

WHILE loop

The WHILE loop can be used like the FOR loop with the difference that the break-off condition can be any Boolean expression. This means you indicate a condition which, when it is fulfilled, the loop will be executed.

Syntax:

```
WHILE <Boolean expression>
```

<Instructions>

```
END_WHILE;
```

The <Instructions> are repeatedly executed as long as the <Boolean_expression> returns TRUE. If the <Boolean_expression> is already FALSE at the first evaluation, then the <Instructions> are never executed. If <Boolean_expression> never assumes the value FALSE, then the <Instructions> are repeated endlessly which causes a relative time delay.

The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example, by counting up or down one counter.

Example:

```
WHILE counter<>0 DO
Var1 := Var1*2;
Counter := Counter-1;
END_WHILE
```

The WHILE and REPEAT loops are, in a certain sense, more powerful than the FOR loop since one doesn't need to know the number of cycles before executing the loop. In some cases one will, therefore, only be able to work with these two loop types. If, however, the number of the loop cycles is clear, then a FOR loop is preferable since it allows no endless loops.

REPEAT loop

The REPEAT loop is different from the WHILE loop because the break-off condition is checked only after the loop has been executed. This means that the loop will run through at least once, regardless of the wording of the break-off condition.

Syntax:

```
REPEAT
```

<Instructions>

```
UNTIL <Boolean expression>
END_REPEAT;
```

The <Instructions> are carried out until the <Boolean expression> returns TRUE.

If <Boolean expression> is produced already at the first TRUE evaluation, then <Instructions> are executed only once. If <Boolean_expression> never assumes the value TRUE, then the <Instructions> are repeated endlessly which causes a relative time delay.

The programmer must make sure that no endless loop is caused. He does this by changing the condition in the instruction part of the loop, for example by counting up or down one counter.

Example:

```
REPEAT
Var1 := Var1*2;
Counter := Counter-1;
UNTIL
Counter=0
END_REPEAT;
```

EXIT instruction

If the EXIT instruction appears in a FOR, WHILE, or REPEAT loop, then the innermost loop is ended, regardless of the break-off condition.

Sequential function chart (SFC)

Overview

The sequential function chart (SFC) is a graphically oriented language which makes it possible to describe the chronological order of different actions within a program. For this the actions are assigned to step elements and the sequence of processing is controlled by transition elements.

Action

An action can contain a series of instructions in IL or in ST, a lot of networks in FBD or in LD, or again in Sequential Function Chart (SFC) ↳ *Chapter 1.4.1.1.10.2 “Function Block Diagram (FBD)” on page 162* ↳ *Chapter 1.4.1.1.10.7.1 “Overview” on page 176*.

With the simplified steps an action is always connected to a step. In order to edit an action, click twice with the mouse on the step to which the action belongs. Or select the step and select the menu command 'Extras' 'Zoom Action/Transition' ↳ *Chapter 1.4.1.3.11.9.18 “'Extras' 'Zoom Action/Transition'” on page 333*. In addition, one input or output action per step is possible.

Actions of IEC steps hang in the Object Organizer directly under their SFC-POU and are loaded with a doubleclick or by pressing <Enter> in their editor ↳ *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172*. New actions can be created with 'Project' 'Add action' ↳ *Chapter 1.4.1.2.4.14 “'Project' 'Add action'” on page 263*. You can assign max. nine actions to one IEC step.

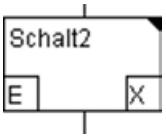
Entry or exit action

Additional to a step action you can add an entry action and an exit action to a step. An entry action is executed only once, right after the step has become active. An exit action is executed only once before the step is deactivated.

A step with entry action is indicated by an "E" in the lower left corner, the exit action by an "X" in the lower right corner.

The entry and exit action can be implemented in any language. In order to edit an entry or exit action, doubleclick in the corresponding corner in the step with the mouse.

Example of a step with entry and exit action:



Transition / Transition condition

Between the steps there are so-called transitions.

A transition condition must have the value TRUE or FALSE. Thus it can consist of either a boolean variable, a boolean address or a boolean constant. It can also contain a series of instructions having a boolean result, either in ST syntax (e.g. (i<= 100) AND b) or in any language desired ↳ *Chapter 1.4.1.3.11.9.18 “'Extras' 'Zoom Action/Transition'” on page 333*. But a transition may not contain programs, function blocks or assignments!

In the SFC-Editor a transition condition can be written directly at the transition symbol or an own editor window can be opened for entering the condition ↳ *Chapter 1.4.1.3.11.9.18 “'Extras' 'Zoom Action/Transition'” on page 333*. Regard that the instructions entered in the editor window will take precedence!

> *Besides transitions, inching mode can also be used to skip to the next step.*

Active step

After calling the SFC POU, the action (surrounded by a double border) belonging to the initial step is executed first ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. A step, whose action is being executed, is called active. In Online mode active steps are shown in blue.

In a control cycle all actions are executed which belong to active steps. Thereafter the respective following steps of the active steps become active if the transition conditions of the following steps are TRUE ↳ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172*. The currently active steps will be executed in the next cycle.

> *If the active step contains an output action, this will only be executed during the next cycle, provided that the transition following is TRUE.*
>
> *Note the possibility to use implicit variables for scanning the status and time of steps or actions ↳ Chapter 1.4.1.1.10.5.7 “Implicit variables in SFC” on page 174.*

IEC step

Along with the simplified steps the standard IEC steps in SFC are available.

In order to be able to use IEC steps, you must link the special SFC library Iecsfc.lib into your project.

Not more than nine actions can be assigned to an IEC step *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. IEC actions are not fixed as input, step or output actions to a certain step as in the simplified steps, but are stored separately from the steps and can be reused many times within a POU. For this they must be associated to the single steps with the command 'Extras''Associate action'.

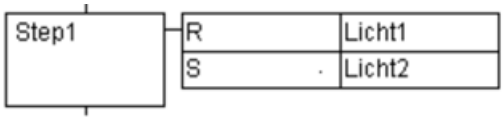
Along with actions, Boolean variables can be assigned to steps.

The activation and deactivation of actions and boolean variables can be controlled using so-called Qualifier. Time delays are possible. Since an action can still be active, if the next step has been processed, for example through the qualifier S (Set), one can achieve concurrent processes.

An associated boolean variable is set or reset with each call of the SFC block. That means, that at each call the variable gets re-assigned value TRUE resp. FALSE.

The actions associated with an IEC step are shown at the right of the step in a two-part box. The left field contains the qualifier, possibly with time constant, and the right field contains the action name respectively boolean variable name.

An example for an IEC step with two actions:



In order to make it easier to follow the processes, all active actions in online mode are shown in blue like the active steps. After each cycle a check is made to see which actions are active.

Pay attention here also to the restrictions on the use of time-qualifiers in actions that are repeatedly re-used within the same cycle.

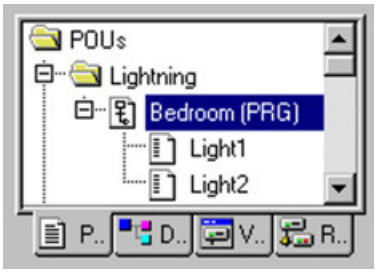
If an action has been inactivated, it will be executed once more. That means, that each action is executed at least twice (also an action with qualifier P).

In case of a call first the deactivated actions, then the active actions are executed, in alphabetical order each time.

Whether a newly inserted step is an IEC step depends upon whether the menu command 'Extras' 'Use IEC-Steps' has been chosen.

In the Object Organizer the actions hang directly underneath their respective SFC POUs. New actions can be created with 'Project' 'Add Action' *Chapter 1.4.1.2.4.14 “'Project' 'Add action'” on page 263*.

In order to use IEC steps you must include in your project the special SFC library Iecsfc.lib .



Implicit variables in SFC

In SFC implicitly declared variables ("flags") can be used to scan the status of steps and actions as well as the time of steps. These flags each are set at the beginning of a cycle. For IEC steps and IEC actions they are provided by the library iecsfc.lib (structures SFCStepType and SFCActionType), which is included automatically in a project *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172* *Chapter 1.4.1.1.10.5.2 “Action” on page 171*.

Scan of the step or action status via boolean variables:

- For IEC steps: <stepname>.x resp. <stepname>._x: <StepName>.x shows the current activation status. <StepName>._x shows the activation status for the next cycle. If <StepName>.x=TRUE, the step will be executed in the current cycle. If <StepName>._x=TRUE and <StepName>.x=FALSE, the step will be executed in the following cycle, i.e. <StepName>._x gets copied to <StepName>.x at the beginning of a cycle.
- For simplified steps: <stepname> resp. _<stepname>: If <StepName>=TRUE, the step will be executed in the current cycle. If _<StepName>=TRUE, the step will be executed in the following cycle, i.e. <StepName> gets copied to _<StepName> at the beginning of a cycle.
- For IEC-actions: <actionname>.x gets TRUE as soon as the action gets active (<actionname>._x is only for internal purposes, not for a status scan).

Time of a step via TIME variables:

The following implicit variables give the current time span which has passed since the step had got active; this is only for steps which have a minimum time configured in the step attributes *Chapter 1.4.1.3.11.9.20 “'Extras' 'Step Attributes'” on page 334*.

- For IEC steps: <stepname>.t (<stepname>._t not usable for external purposes)
- For simplified steps: _time<stepname>. BUT: If this implicit variable should be used for scan purposes, it also must be declared explicitly as a TIME variable; e.g. "_timeStep1 : TIME;"
- For IEC actions: the implicit time variables are not used.

These status flags can be used in each action and transition of the SFC module. But they can also be accessed from other programs:

Example: boolvar1:=sfc.step1.x;

step1.x in the example is an implicit boolean variable showing the status of IEC step "step1" in POU "sfc1".

SFC flags

For controlling the operation of SFC POUs flags can be used, which are created implicitly during running the project. To read this flags you have to define appropriate global or local variables as inputs or outputs. Example: If in a SFC POU a step is active for a longer time than defined in the step attributes, then a flag will be set, which is accessible by using a variable "SFCError" (SFCError gets TRUE in this case).

The following flag variables can be defined:

- SFCEnableLimit: This variable is of the type BOOL. When it has the value TRUE, the timeouts of the steps will be registered in SFCError. Other timeouts will be ignored.
- SFCInit: When this boolean variable has the value TRUE the sequential function chart is set back to the Init step. The other SFC flags are reset too (initialization). The Init step remains active, but is not executed, for as long as the variable has the value TRUE. It is only when SFCInit is again set to FALSE that the block can be processed normally.
- SFCReset: This variable, of type BOOL, behaves similarly to SFCInit. Unlike the latter, however, further processing takes place after the initialization of the Init step. Thus for example the SFCReset flag could be re-set to FALSE in the Init step.

As from compiler version 2.3.7.0 SFCReset also can be used to reset boolean actions associated to IEC steps, which was not possible before.

- SFCQuitError: Provided that the Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE whereby a possible timeout in the variable SFCError is reset. All previous times in the active steps are reset when the variable again assumes the value FALSE. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.
- SFCPause: Execution of the SFC diagram is stopped for as long as this boolean variable has the value TRUE.
- SFCError: This Boolean variable is TRUE when a timeout has occurred in a SFC diagram. If another timeout occurs in a program after the first one, it will not be registered unless the variable SFCError is reset first. It is a pre-condition that SFCError is defined, if you want to use the other time-controlling flags (SFCErrorStep, SFCErrorPOU, SFCQuitError, SFCErrorAnalyzation).
- SFCTrans: This boolean variable takes on the value TRUE when a transition is actuated.
- SFCErrorStep: This variable is of the type STRING. If SFCError registers a timeout, in this variable is stored the name of the step which has caused the timeout. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.
- SFCErrorPOU: This variable of the type STRING contains the name of the block in which a timeout has occurred. It is a pre-condition that the flag SFCError has been defined also, which registers any timeout in the SFC.
- SFCCurrentStep: This variable is of the type STRING. The name of the step is stored in this variable which is active, independently of the time monitoring. In the case of simultaneous sequences the step is stored in the branch on the outer right. No further timeout will be registered if a timeout occurs and the variable SFCError is not reset again.
- SFCErrorAnalyzationTable: This variable of type ARRAY [0..n] OF ExpressionResult provides the result of an analyzation of a transition expression. For each component of the expression, which is contributing to a FALSE of the transition and thereby to a timeout of the preceding step, the following information is written to the structure ExpressionResult: name, address, comment, current value.

  This is possible for maximum 16 components (variables), thus the array range is max. 0..15).

  The structure ExpressionResult as well as the implicitly used analyzation modules are provided with the library AnalyzationNew.lib. The analyzation modules also can be used explicitly in other POUs, which are not programmed in SFC.

  It is a pre-condition for the analyzation of a transition expression, that a timeout is registered in the preceding step. So a time monitoring must be implemented there and also the variable SFCError (see above) must be defined in the declaration window.
- SFCTip, SFCTipMode: This variables of type BOOL allow inching mode of the SFC. When this is switched on by SFCTipMode=TRUE, it is only possible to skip to the next step if SFCTip is set to TRUE. As long as SFCTipMode is set to FALSE, it is possible to skip even over transitions.

Regard also the implicit variables usable for scanning the status and time of steps/ actions ↳ *Chapter 1.4.1.1.10.5.7 “Implicit variables in SFC” on page 174.*

Alternative branch

Two or more branches in SFC can be defined as alternative branches. Each alternative branch must begin and end with a transition ↳ *Chapter 1.4.1.2.3.43 “'Project' 'Passwords for user groups'” on page 250*. Alternative branches can contain parallel branches and other alternative branches ↳ *Chapter 1.4.1.1.10.5.10 “Parallel branch” on page 176*. An alternative branch begins at a horizontal line (alternative beginning) and ends at a horizontal line (alternative end) or with a jump ↳ *Chapter 1.4.1.1.10.5.11 “Jump” on page 176*.

If the step which precedes the alternative beginning line is active, then the first transition of each alternative branch is evaluated from left to right. The first transition from the left whose transition condition has the value TRUE is opened and the following steps are activated ↳ *Chapter 1.4.1.1.10.5.5 “Active step” on page 172*.

Parallel branch

Two or more branches in SFC can be defined as parallel branches. Each parallel branch must begin and end with a step. Parallel branches can contain alternative branches or other parallel branches. A parallel branch begins with a double line (parallel beginning) and ends with a double line (parallel end) or with a jump. It can be provided with a jump label.

If the parallel beginning line of the previous step is active and the transition condition after this step has the value TRUE, then the first steps of all parallel branches become active *Chapter 1.4.1.2.3.43 “'Project' 'Passwords for user groups'” on page 250* *Chapter 1.4.1.1.10.5.5 “Active step” on page 172*. These branches are now processed parallel to one another. The step after the parallel end line becomes active when all previous steps are active and the transition condition before this step produces the value TRUE.

Jump

A jump is a connection to the step whose name is indicated under the jump symbol. Jumps are required because it is not allowed to create connections which lead upward or cross each other.

The continuous function chart (CFC)

The continuous function chart bases on the Function Block Diagram language. However it does not operate with networks, but rather with freely placeable elements. This allows feedback, for example.

Ladder Diagram (LD)

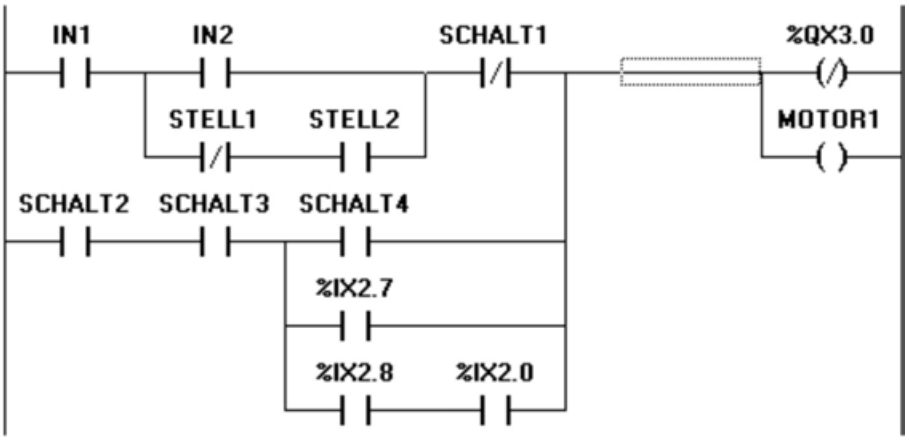
Overview

The Ladder Diagram is also a graphics oriented programming language which approaches the structure of an electric circuit.

On the one hand, the Ladder Diagram is suitable for constructing logical switches, on the other hand one can also create networks as in FBD. Therefore the LD is very useful for controlling the call of other POUs.

The Ladder Diagram consists of a series of networks. A network is limited on the left and right sides by a left and right vertical current line. In the middle is a circuit diagram made up of contacts, coils, and connecting lines.

Each network consists on the left side of a series of contacts which pass on from left to right the condition "ON" or "OFF" which correspond to the Boolean values TRUE and FALSE. To each contact belongs a Boolean variable. If this variable is TRUE, then the condition is passed from left to right along the connecting line. Otherwise the right connection receives the value OFF.



See also:

The Graphic Editors ↳ *Chapter 1.4.1.3.11.1 “Overview” on page 314*

The Ladder Editor ↳ *Chapter 1.4.1.3.11.8.1 “Overview” on page 323*

Contact

Each network in LD consists on the left side of a network of contacts (contacts are represented by two parallel lines: | |) which from left to right show the condition "On" or "Off".

These conditions correspond to the Boolean values TRUE and FALSE. A Boolean variable belongs to each contact. If this variable is TRUE, then the condition is passed on by the connecting line from left to right, otherwise the right connection receives the value "Out".

Contacts can be connected in parallel, then one of the parallel branches must transmit the value "On" so that the parallel branch transmits the value "On"; or the contacts are connected in series, then contacts must transmit the condition "On" so that the last contact transmits the "On" condition. This therefore corresponds to an electric parallel or series circuit.

A contact can also be negated, recognizable by the slash in the contact symbol: |/| ↳ *Chapter 1.4.1.3.11.8.25 “'Extras' 'Negate' in LD” on page 329*. Then the value of the line is transmitted if the variable is FALSE.

Coil

On the right side of a network in LD there can be any number of so-called coils which are represented by parentheses:( ). They can only be in parallel. A coil transmits the value of the connections from left to right and copies it in an appropriate Boolean variable. At the entry line the value ON (corresponds to the Boolean variable TRUE) or the value OFF (corresponding to FALSE) can be present.

Contacts and coils can also be negated (in the example the contact SWITCH1 and the coil %QX3.0 is negated). If a coil is negated (recognizable by the slash in the coil symbol: (/)), then it copies the negated value in the appropriate Boolean variable. If a contact is negated, then it connects through only if the appropriate Boolean value is FALSE.

Function blocks in the Ladder Diagram

Along with contacts and coils you can also enter function blocks and programs In the network they must have an input and an output with Boolean values and can be used at the same places as contacts, that is on the left side of the Ladder Diagram network ↳ *Chapter 1.4.1.1.10.7.2 “Contact” on page 177* ↳ *Chapter 1.4.1.1.10.7.3 “Coil” on page 177* ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*.

Set/Reset coils

Coils can also be defined as set or reset coils. One can recognize a set coil by the "S" in the coil symbol: (S)) It never writes over the value TRUE in the appropriate Boolean variable. That is, if the variable was once set at TRUE, then it remains so.

One can recognize a reset coil by the "R" in the coil symbol: (R)) It never writes over the value FALSE in the appropriate Boolean variable: If the variable has been once set on FALSE, then it remains so.

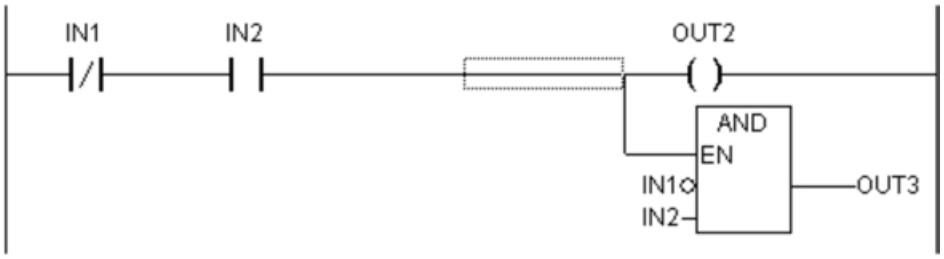
LD as FBD

When working with LD it is very possible that you will want to use the result of the contact switch for controlling other POUs. On the one hand you can use the coils to put the result in a global variable which can then be used in another place. You can, however, also insert the possible call directly into your LD network. For this you introduce a POU with EN input.

Such POUs are completely normal operands, functions, programs, or function blocks which have an additional input which is labeled with EN. The EN input is always of the BOOL type and its meaning is: The POU with EN input is evaluated when EN has the value TRUE.

An EN POU is wired parallel to the coils, whereby the EN input is connected to the connecting line between the contacts and the coils. If the ON information is transmitted through this line, this POU will be evaluated completely normally.

Starting from such an EN POU, you can create networks similar to FBD.



Reserved keywords

The following strings are reserved as keywords, i.e. they cannot be used as identifiers for variables or POUs:

ABS *Chapter 1.4.1.6.10.1 “ABS” on page 430*

ACOS *Chapter 1.4.1.6.10.10 “ACOS” on page 433*

ACTION (only used in the Export Format)

ADD *Chapter 1.4.1.6.2.1 “ADD” on page 407*

ADR *Chapter 1.4.1.6.7.1 “ADR” on page 421*

ADRINST *Chapter 1.4.1.6.7.2 “ADRINST” on page 421*

AND *Chapter 1.4.1.6.3.1 “AND” on page 410*

ANDN *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

ARRAY *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*

ASIN *Chapter 1.4.1.6.10.9 “ASIN” on page 433*

AT *Chapter 1.4.1.7.4.2 “Address” on page 441*

ATAN *Chapter 1.4.1.6.10.11 “ATAN” on page 434*

BITADR *Chapter 1.4.1.6.7.3 “BITADR” on page 421*

BOOL *Chapter 1.4.1.8.1.2 “BOOL” on page 443*

BY ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
BYTE ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
CAL ↳ *Chapter 1.4.1.6.8.1 “CAL” on page 422*
CALC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
CALCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
CASE ↳ *Chapter 1.4.1.1.10.4.9 “CASE instruction” on page 169*
CONSTANT ↳ *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*
COS ↳ *Chapter 1.4.1.6.10.7 “COS” on page 432*
DATE ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*
DINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
DIV ↳ *Chapter 1.4.1.6.2.4 “DIV” on page 408*
DO ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
DT ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*
DWORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
ELSE ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
ELSEIF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
END_ACTION (only used in the Export Format)
END_CASE ↳ *Chapter 1.4.1.1.10.4.9 “CASE instruction” on page 169*
END_FOR ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
END_FUNCTION (only used in the Export Format)
END_FUNCTION_BLOCK (only used in the Export Format)
END_IF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
END_PROGRAM (only used in the Export Format)
END_REPEAT ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*
END_STRUCT ↳ *Chapter 1.4.1.8.2.5 “Structures” on page 449*
END_TYPE ↳ *Chapter 1.4.1.8.2.4 “Enumeration” on page 448*
END_VAR ↳ *Chapter 1.4.1.3.9.6 “Local variables” on page 301*
END_WHILE ↳ *Chapter 1.4.1.1.10.4.11 “WHILE loop” on page 170*
EQ ↳ *Chapter 1.4.1.6.6.5 “EQ” on page 420*
EXIT ↳ *Chapter 1.4.1.1.10.4.13 “EXIT instruction” on page 171*
EXP ↳ *Chapter 1.4.1.6.10.5 “EXP” on page 431*
EXPT ↳ *Chapter 1.4.1.6.10.12 “EXPT” on page 434*
FALSE ↳ *Chapter 1.4.1.8.1.2 “BOOL” on page 443*
FOR ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
FUNCTION ↳ *Chapter 1.4.1.1.9.3 “Function” on page 151*
FUNCTION_BLOCK ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*
GE ↳ *Chapter 1.4.1.6.6.4 “GE” on page 419*
GT ↳ *Chapter 1.4.1.6.6.1 “GT” on page 418*
IF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
INDEXOF ↳ *Chapter 1.4.1.6.2.7 “INDEXOF” on page 410*
INI ↳ *Chapter 1.4.1.6.11.1 “INI operator” on page 434*
INT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

JMP ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
JMPC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
JMPCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LD ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LDN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LE ↳ *Chapter 1.4.1.6.6.3 “LE” on page 419*
LINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
LN ↳ *Chapter 1.4.1.6.10.3 “LN” on page 431*
LOG ↳ *Chapter 1.4.1.6.10.4 “LOG” on page 431*
LREAL ↳ *Chapter 1.4.1.8.1.4 “REAL / LREAL” on page 443*
LT ↳ *Chapter 1.4.1.6.6.2 “LT” on page 419*
LWORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
MAX ↳ *Chapter 1.4.1.6.5.3 “MAX” on page 416*
MIN ↳ *Chapter 1.4.1.6.5.4 “MIN” on page 417*
MOD ↳ *Chapter 1.4.1.6.2.5 “MOD” on page 409*
MOVE ↳ *Chapter 1.4.1.6.2.6 “MOVE” on page 409*
MUL ↳ *Chapter 1.4.1.6.2.2 “MUL” on page 407*
MUX ↳ *Chapter 1.4.1.6.5.6 “MUX” on page 418*
NE ↳ *Chapter 1.4.1.6.6.6 “NE” on page 420*
NOT ↳ *Chapter 1.4.1.6.3.4 “NOT” on page 412*
OF ↳ *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*
OR ↳ *Chapter 1.4.1.6.3.2 “OR” on page 411*
ORN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
PERSISTENT ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*
POINTER ↳ *Chapter 1.4.1.8.2.3 “Pointer” on page 447*
PROGRAM ↳ *Chapter 1.4.1.1.9.7 “Program” on page 156*
R ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
READ_ONLY
READ_WRITE
REAL ↳ *Chapter 1.4.1.8.1.4 “REAL / LREAL” on page 443*
REPEAT ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*
RET ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETAIN ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*
RETC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETURN ↳ *Chapter 1.4.1.1.10.4.7 “RETURN instruction” on page 168*
ROL ↳ *Chapter 1.4.1.6.4.3 “ROL” on page 414*
ROR ↳ *Chapter 1.4.1.6.4.4 “ROR” on page 415*
S ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
SEL ↳ *Chapter 1.4.1.6.5.2 “SEL” on page 416*
SHL ↳ *Chapter 1.4.1.6.4.1 “SHL” on page 412*
SHR ↳ *Chapter 1.4.1.6.4.2 “SHR” on page 413*

SIN ↳ *Chapter 1.4.1.6.10.6 “SIN” on page 432*

SINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

SIZEOF ↳ *Chapter 1.4.1.6.2.8 “SIZEOF” on page 410*

SQRT ↳ *Chapter 1.4.1.6.10.2 “SQRT” on page 430*

ST ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

STN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

STRING ↳ *Chapter 1.4.1.8.1.5 “STRING” on page 444*

STRUCT ↳ *Chapter 1.4.1.8.2.5 “Structures” on page 449*

SUB ↳ *Chapter 1.4.1.6.2.3 “SUB” on page 408*

TAN ↳ *Chapter 1.4.1.6.10.8 “TAN” on page 433*

THEN ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*

TIME ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*

TO ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*

TOD ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*

TRUE ↳ *Chapter 1.4.1.8.1.2 “BOOL” on page 443*

TRUNC ↳ *Chapter 1.4.1.6.9.9 “TRUNC” on page 429*

TYPE ↳ *Chapter 1.4.1.8.2.4 “Enumeration” on page 448*

UDINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

UINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

ULINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

UNTIL ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*

USINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

VAR ↳ *Chapter 1.4.1.3.9.6 “Local variables” on page 301*

VAR_ACCESS (only used very specifically, depending on the hardware)

VAR_CONFIG ↳ *Chapter 1.4.1.4.1.4.1 “Overview” on page 361*

VAR_CONSTANT ↳ *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*

VAR_EXTERNAL ↳ *Chapter 1.4.1.3.9.9 “External variables” on page 303*

VAR_GLOBAL ↳ *Chapter 1.4.1.4.1.3.2 “Several variables lists” on page 359*

VAR_IN_OUT ↳ *Chapter 1.4.1.3.9.5 “Input and output variables” on page 301*

VAR_INPUT ↳ *Chapter 1.4.1.3.9.3 “Input variable” on page 301*

VAR_OUTPUT ↳ *Chapter 1.4.1.3.9.4 “Output variable” on page 301*

WHILE ↳ *Chapter 1.4.1.1.10.4.11 “WHILE loop” on page 170*

WORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

WSTRING (IEC data type is not supported)

XOR ↳ *Chapter 1.4.1.6.3.3 “XOR” on page 411*

XORN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

Additionally all conversion operators as listed in the 'Edit' 'Input assistant' are handled as keywords ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

1.4.1.1.11 Debugging, online functions

Sampling trace

The sampling trace allows you to trace the value sequence of variables, depending upon the so-called trigger event. This is the rising edge or falling edge of a previously defined Boolean variable (trigger variable). The tracing of up to 20 variables is permitted. 500 values can be traced for each variable.

Debugging

The debugging functions make it easier for you to find errors.

In order to debug, run the command *“Project ➔ Options”* and in the dialog box that pops up under the *“Build”* options select activate option *“Debugging”*↳ *Chapter 1.4.1.2.2.9 “Options for build” on page 209*.

Breakpoint

A breakpoint is a place in the program at which the processing is stopped. Thus it is possible to look at the values of variables at specific places within the program.

Breakpoints can be set in all editors. In the text editors breakpoints are set at line numbers, in FBD and LD at network numbers, in CFC at POUs and in SFC at steps. No breakpoints can be set in function block instances.

WARNING!
Runtime system CODESYS SP 32 bit Full will deactivate the watchdog function of the concerned task as soon as the execution of the program currently is stopped at a breakpoint.

Single step

Single step means:

- In IL: Execute the program until the next CAL, LD or JMP command.
- In ST: Execute the next instruction.
- In FBD, LD: Execute the next network.
- In SFC: Continue the action until the next step.

By proceeding step by step you can check the logical correctness of your program.

Single cycle

If Single cycle has been chosen, then the execution is stopped after each cycle.

Change values online

During operations variables can be set once at a certain value or also described again with a certain value after each cycle ↳ *write value* ↳ *force value*. In online mode one also can change the variable value by double click on the value. By that boolean variables change from TRUE to FALSE or the other way round, for each other types of variables one gets the dialog Write Variable xy, where the actual value of the variable can be edited.

Monitoring

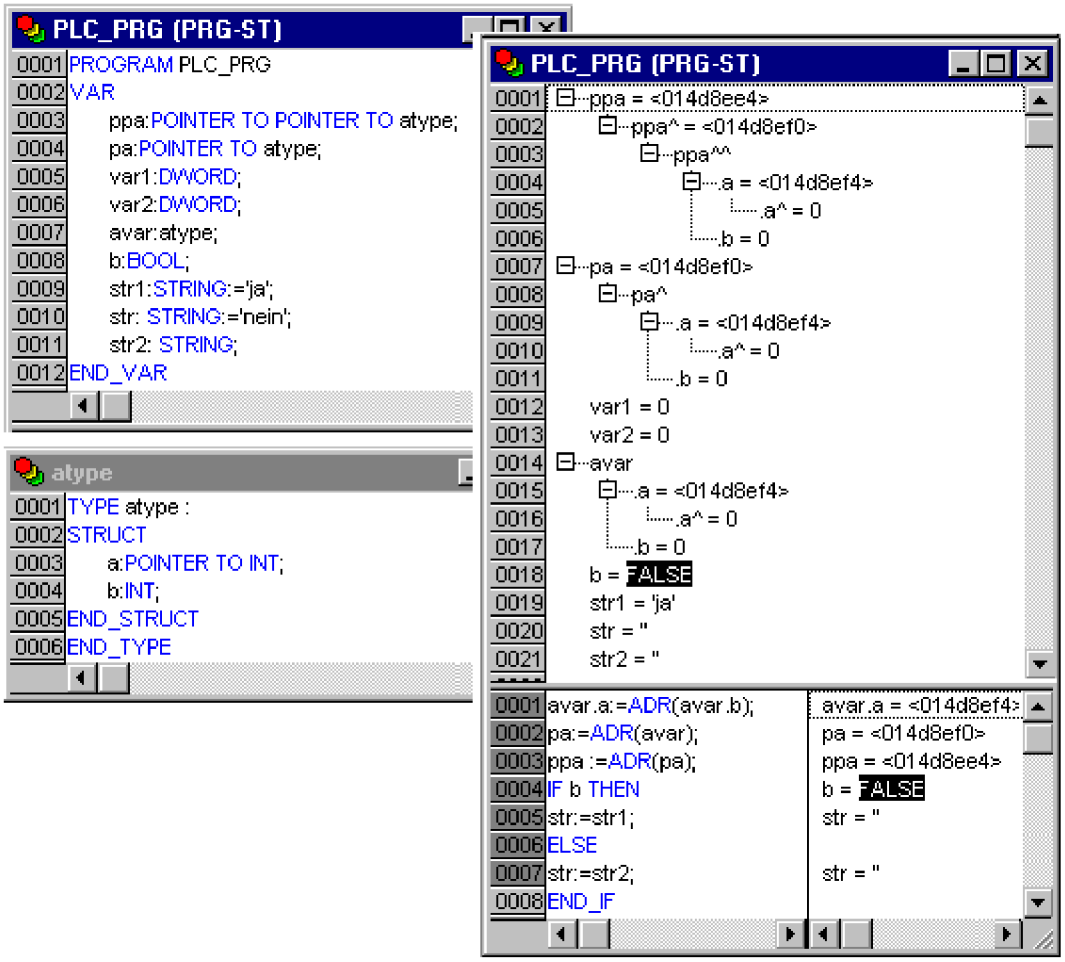
In Online mode, all displayable variables are read from the controller and displayed in real time. You will find this display in the declarations and program editor; you can also read out current values of variables in the watch and receipt manager and in a visualization. If variables from instances of function blocks are to be monitored, the corresponding instance must first be opened.

In monitoring VAR_IN_OUT variables, the de-referenced value is output.

In monitoring pointers, both the pointer and the de-referenced value are output in the declaration portion. In the program portion, only the pointer is output:

+ --pointervar = '<'pointervalue'>'

POINTERs in the de-referenced value are also displayed accordingly. With a simple click on the cross or a double-click on the line, the display is either expanded or truncated.



In the implementations, the value of the pointer is displayed. For de-referencing, however, the de-referenced value is displayed.

Monitoring of ARRAY components: In addition to array components indexed by a constant, components are also displayed which are indexed by a variable:

anarray[1] = 5

anarray[i] = 1

If the index consists of an expression (e.g. [i+j] or [i+1]), the component can not be displayed.

> *If the maximum number of variables which can be monitored, has been reached, for each further variable instead of the current value the string "Too many monitoring variables" will be displayed.*

Simulation

During the simulation the created PLC program is not processed in the PLC. All online functions are available. That allows you to test the logical correctness of your program without PLC hardware.

POUs of external libraries do not run in simulation mode.

Log

The log chronologically records user actions, internal processes, state changes and exceptions during Online mode processing ↳ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172*. It is used for monitoring and for error tracing ↳ *Chapter 1.4.1.2.6.1 “Overview” on page 278*.

1.4.1.1.12 We write a little program

Controlling a traffic signal unit

Let us now start to write a small example program. It is for a simple traffic signal unit which is supposed to control two traffic signals at an intersection. The red/green phases of both traffic signals alternate and, in order to avoid accidents, we will insert yellow or yellow/red transitional phases. The latter will be longer than the former.

In the example you will see how time dependent programs can be shown with the language resources of the IEC1131-3 standard, how one can edit the different languages of the standard, and how one can easily connect them while becoming familiar with the simulation ↳ *Chapter 1.4.1.1.11.8 “Simulation” on page 184*.

Create POU

Choose *“File ➔ New”*↳ *Chapter 1.4.1.2.3.1 “'File' 'New'” on page 222*.

In the dialog box which appears, the first POU has already been given the default name PLC_PRG. Keep this name, and the type of POU should definitely be a program. Each project needs a program with this name. In this case we choose as the language of this POU the Continuous Function Chart Editor (CFC).

Now create three more objects with the command 'Project' 'Object Add' with the menu bar or with the context menu (press right mouse button in the Object Organizer) ↳ *Chapter 1.4.1.2.4.6 “'Project' 'Object' ' Add'” on page 258*. A program in the language Sequential Function Chart named SEQUENCE, a function block ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153* in the language Function Block Diagram named TRAFFICSIGNAL, along with a POU WAIT, also of the type function block, which we want to program as an Instruction List ↳ *IL* ↳ *Chapter 1.4.1.1.10.5.1 “Overview” on page 171* ↳ *Chapter 1.4.1.1.10.2 “Function Block Diagram (FBD)” on page 162*.

What does TRAFFICSIGNAL do?

In the POU TRAFFICSIGNAL we will assign the individual trafficsignal phases to the lights, i.e. we will make sure that the red light is lit red in the red phase and in the yellow/red phase, the yellow light in the yellow and yellow/red phases, etc.

What does WAIT do?

In WAIT we will program a simple timer which as input will receive the length of the phase in milliseconds, and as output will produce TRUE as soon as the time period is finished.

What does SEQUENCE do?

In SEQUENCE all is combined so that the right light lights up at the right time for the desired time period.

What does PLC_PRG do?

In PLC_PRG the input start signal is connected to the traffic lights' sequence and the "color instructions" for each lamp are provided as outputs.

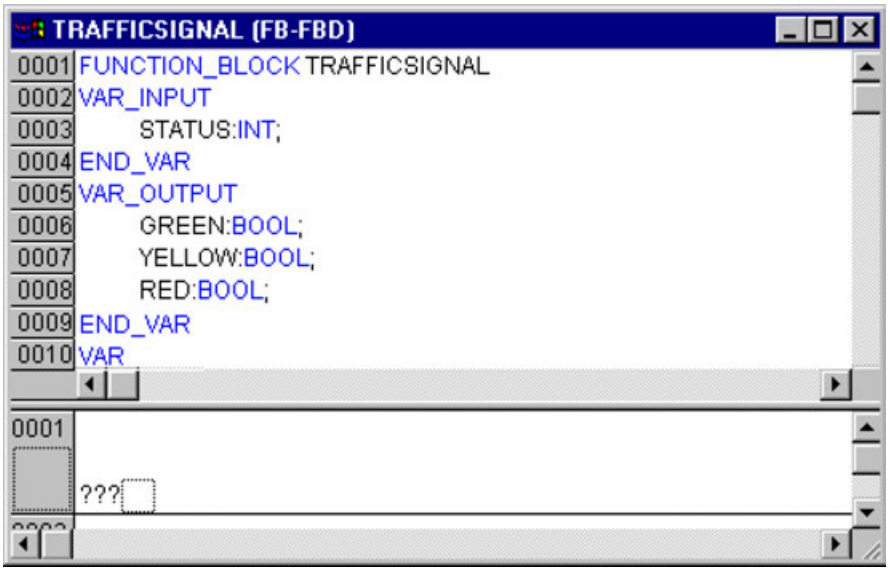
TRAFFICSIGNAL simulation

Now test your program in simulation mode. Compile ("Project" "Build") and load ("Online" "Login") it. Start the program by "Online" "Start", then set variable ON to TRUE, e.g. by a double-click on the entry "ON" in the input box of the CFC editor. This will mark the variable as prepared to be set to <TRUE>. Then press <Ctrl><F7> or command "Online" "Write values", to set the value. Now variable START in ABLAUF (which we had set to TRUE manually in the first extension level of the program) gets this value by variable ON, which is used in PLC_PRG. This will make run the traffic light cycles. PLC_PRG has changed to a monitoring window. Click twice on the plus sign in the declaration editor, the variable display drops down, and you can see the values of the individual variables.

"TRAFFICSIGNAL" declaration

Let us now turn to the POU TRAFFICSIGNAL. In the declaration editor you declare as input variable (between the keywords VAR_INPUT and END_VAR) a variable named STATUS of the type INT. STATUS will have four possible conditions, that is one for the TRAFFICSIGNAL phases green, yellow, yellow/red andred.

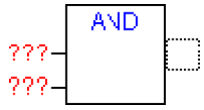
Correspondingly our TRAFFICSIGNAL has three outputs, that is RED, YELLOW and GREEN. You should declare these three variables. Then the declaration part of our TRAFFICSIGNAL will look like this *Chapter 1.4.1.1.9.4 “Function block” on page 153*:



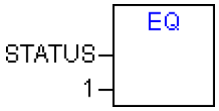
"TRAFFICSIGNAL" body

Now we determine the values of the output variables depending on the input STATUS of the POU. To do this go into the body of the POU. Click on the field to the left beside the first network (the gray field with the number 0001). You have now selected the first network. Choose the menu item 'Insert' 'Box' *Chapter 1.4.1.3.11.7.7 "'Insert' 'Box' in FBD" on page 319*.

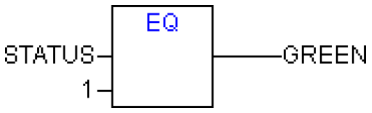
In the first network a box is inserted with the operator AND and two inputs:



Click on the text AND, so that it appears selected and change the text into EQ. Select then for each of the two inputs the three question marks and overwrite them with "STATUS" respectively "1".

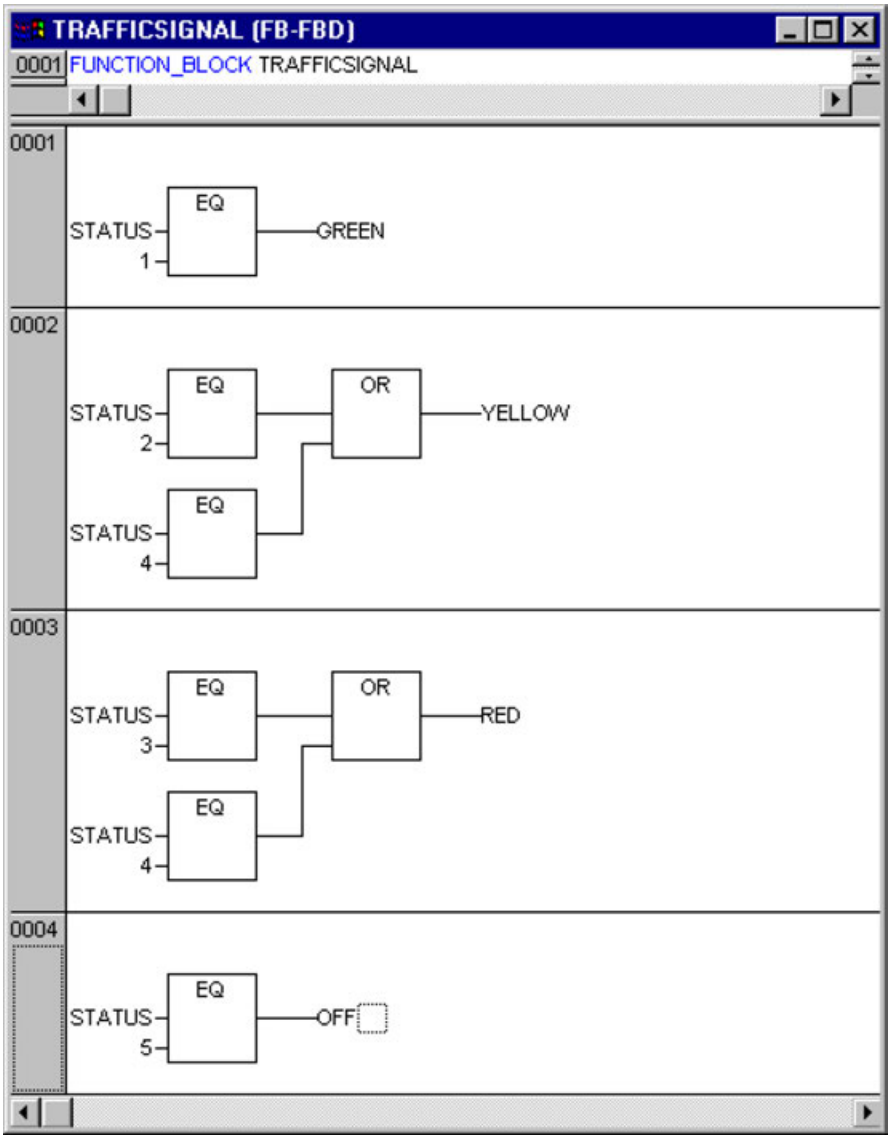


Click now on a place behind the EQ Box. Now the output of the EQ operation is selected. Choose 'Insert' 'Assign' in FBD *Chapter 1.4.1.3.11.7.4 "'Insert' 'Assign'in FBD" on page 318*. Change the three question marks ??? to GREEN. You now have created a network with the following structure:



STATUS is compared with 1, the result is assigned to GREEN. This network thus switches to GREEN if the preset state value is 1.

For the other TRAFFICSIGNAL colors we need two more networks. To create the first one execute command 'Insert' 'Network (after)' and insert an EQ-Box like described above. Then select the output pin of this box and use again command 'Insert' 'Box'. In the new box replace "AND" by "OR". Now select the first output pin of the OR-box and use command 'Insert' 'Assign' to assign it to "YELLOW". Select the second input of the OR-box by a mouse-click on the horizontal line next to the three question marks, so that it appears marked by a dotted rectangle. Now use 'Insert' 'Box' to add a further EQ-box like described above. Finally the network should look like shown in the following:



In order to insert an operator in front of another operator, you must select the place where the input to which you want to attach the operator feeds into the box.

Then use the command 'Insert' 'Box' *Chapter 1.4.1.3.11.7.7 “'Insert' 'Box' in FBD” on page 319*. Otherwise you can set up these networks in the same way as the first network.

Now our first POU has been finished. TRAFFICSIGNAL, according to the input of the value STATUS, controls whichever light color we wish.

Connecting the standard.lib

For the timer in the POU WAIT we need a POU from the standard library. Therefore, open the library manager with 'Window' 'Library Manager' *Chapter 1.4.1.4.3.1 “Overview” on page 371*. Choose 'Insert' 'Additional library' *Chapter 1.4.1.4.3.4 “'Insert' 'Additional Library'” on page 373*. The dialog box appears for opening files. From the list of the libraries choose standard.lib.

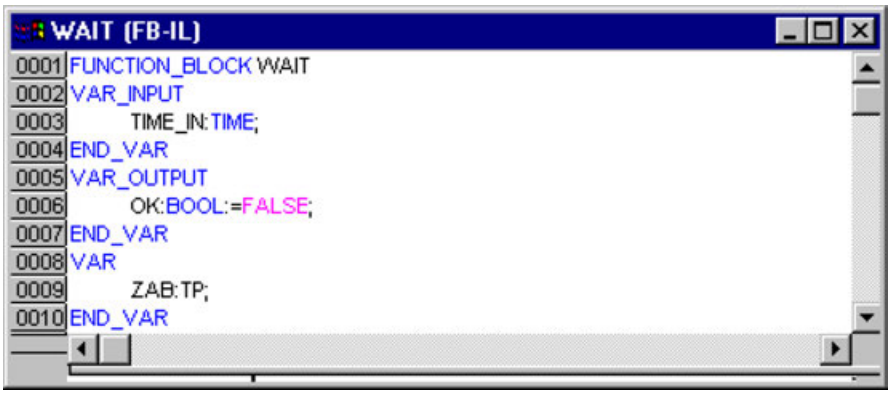
"WAIT" declaration

Now let us turn to the POU WAIT. This POU is supposed to become a timer with which we can determine the length of the time period of each TRAFFICSIGNAL phase. Our POU receives as input variable a variable TIME of the type TIME, and as output it produces a Boolean value which we want to call OK and which should be TRUE when the desired time period is finished. We set this value with FALSE by inserting at the end of the declaration (before the semicolon, however) " := FALSE ".

For our purposes we need the POU TP, a clock generator. This has two inputs (IN, PT) and two outputs (Q, ET). TP does the following:

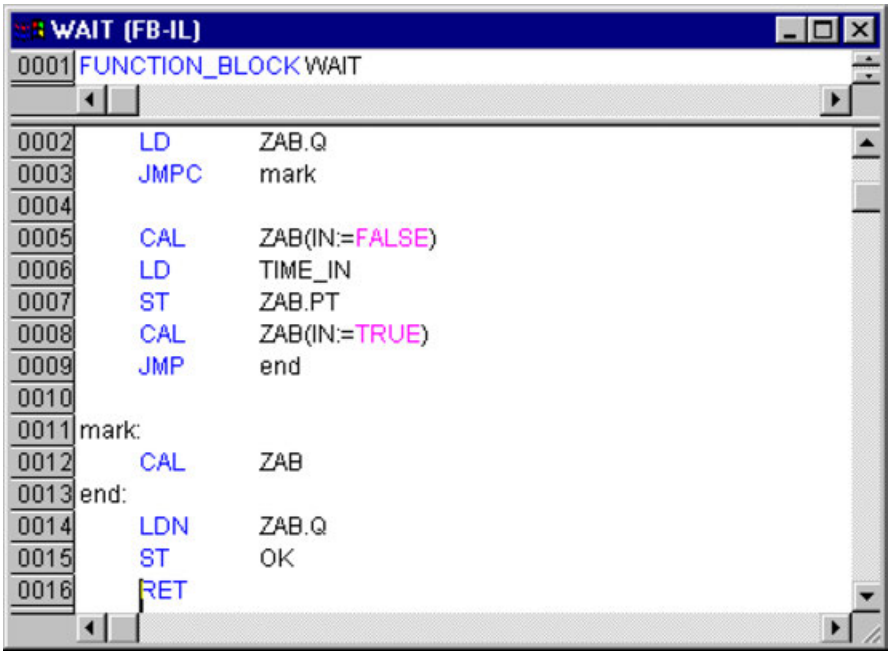
As long as IN is FALSE, ET is 0 and Q is FALSE. As soon as IN provides the value TRUE, the time is calculated at the output ET in milliseconds. When ET reaches the value PT, then ET is no longer counted. Meanwhile Q produces TRUE as long as ET is smaller than PT. As soon as the value PT has been reached, then Q produces FALSE again. See the chapter on the standard library for short descriptions of all POUs.

In order to use the POU TP in the POU WAIT we must create a local instance from TP. For this we declare a local variable ZAB (for elapsed time) of the type TP (between the keywords VAR, END_VAR). The declaration part of WAIT thus looks like this:



"WAIT" body

In order to create the desired timer, the body of the POU must be programmed as follows:



At first it is checked whether Q has already been set at TRUE (as though the counting had already been executed), in this case we change nothing with the occupation of ZAB, but we call the function block ZAB without input (in order to check whether the time period is already over) ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*.

Otherwise we set the variable IN in ZAB at FALSE, and therefore at the same time ET at 0 and Q at FALSE. In this way all variables are set at the desired initial condition. Now we assign the necessary time from the variable TIME into the variable PT, and call ZAB with IN:=TRUE. In the function block ZAB the variable ET is now calculated until it reaches the value TIME, then Q is set at FALSE.

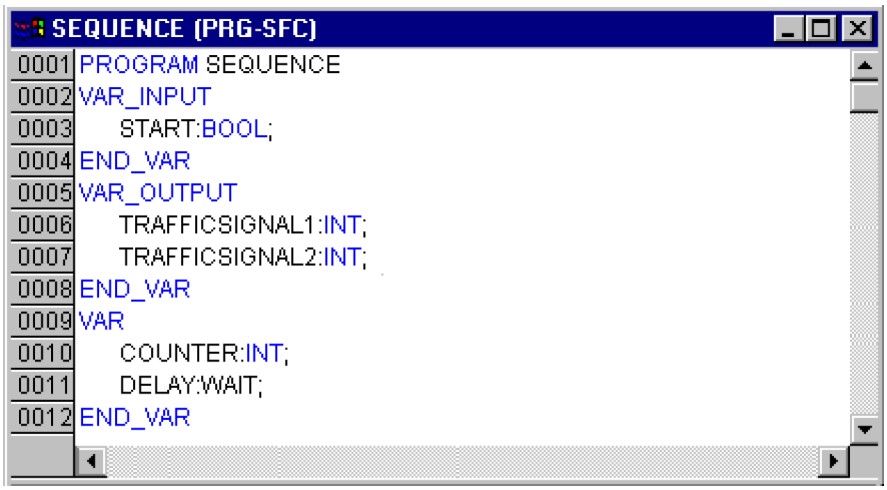
The negated value of Q is saved in OK after each execution of WAIT. As soon as Q is FALSE, then OK produces TRUE.

The timer is finished at this point. Now it is time to combine our two function blocks WAIT and SEQUENCE in the main program ↳ *Chapter 1.4.1.1.12.1.17 “PLC_PRG” on page 194*.

"SEQUENCE" first expansion level

First we declare the variables we need. They are: an input variable START of the type BOOL, two output variables TRAFFICSIGNAL1 and TRAFFICSIGNAL2 of the type INT and one of the type WAIT (DELAY as delay). The program SEQUENCE now looks like shown here.

Program SEQUENCE, First Expansion Level, Declaration Part:



Create a SFC diagram

The beginning diagram of a POU in SFC always consists of an action "Init" of a following transition "Trans0" and a jump back to Init ↳ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172*. We have to expand that.

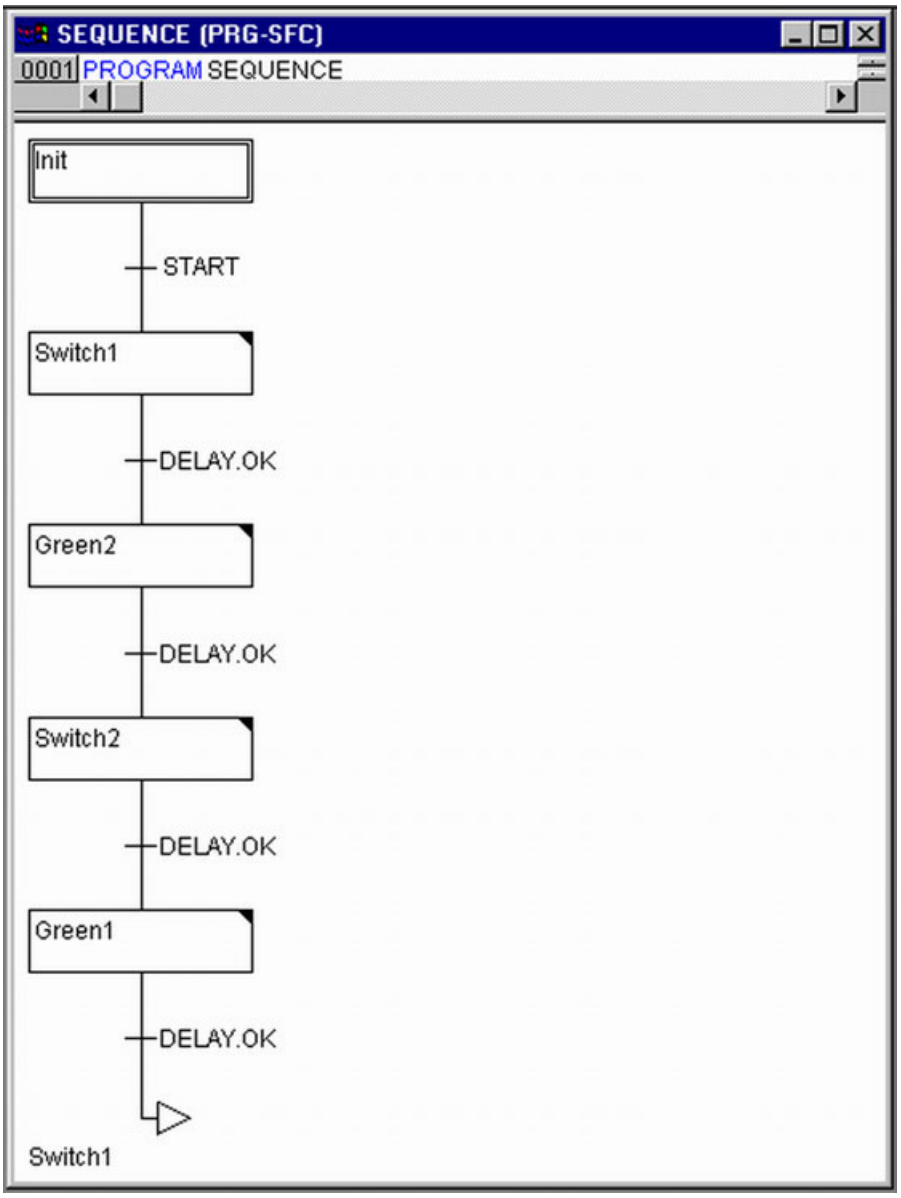
Before we program the individual action and transitions let us first determine the structure of the diagrams. We need one step for each trafficsignal phase. Insert it by marking Trans0 and choosing "Insert" "Step transition (after)". Repeat this procedure three more times.

If you click directly on the name of a transition or a step, then this is marked and you can change it. Name the first transition after Init "START", and all other transitions "DELAY.OK".

The first transition switches through when START is TRUE and all others switch through when DELAY in OK produces TRUE, i.e. when the set time period is finished.

The steps (from top to bottom) receive the names Switch1, Green2, Switch2, Green1, whereby Init of course keeps its Name. "Switch" should include a yellow phase, at Green1 TRAFFIC-SIGNAL1 will be green, at Green2 TRAFFICSIGNAL2 will be green. Finally change the return address of Init after Switch1. If you have done everything right, then the diagram should look like in the following figure:

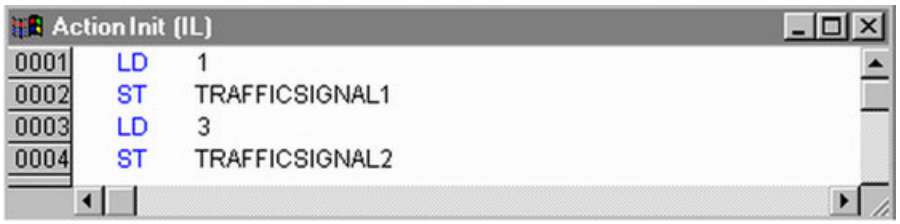
Program SEQUENCE, First Expansion Level, Instruction Part



Now we have to finish the programming of the individual steps. If you doubleclick on the field of a step, then you get a dialog for opening a new action ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. In our case we will use IL (Instruction List).

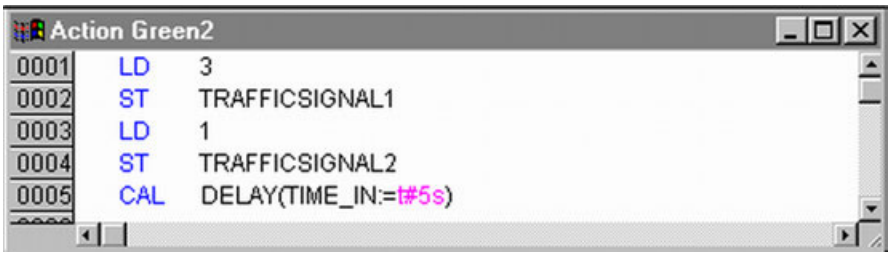
Actions and transition conditions

In the action of the step 'Init' the variables are initialized, the STATUS of TRAFFICSIGNAL1 should be 1 (green) ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. The state of TRAFFIC-SIGNAL2 should be 3 (red). The action Init then looks like in the following figure:



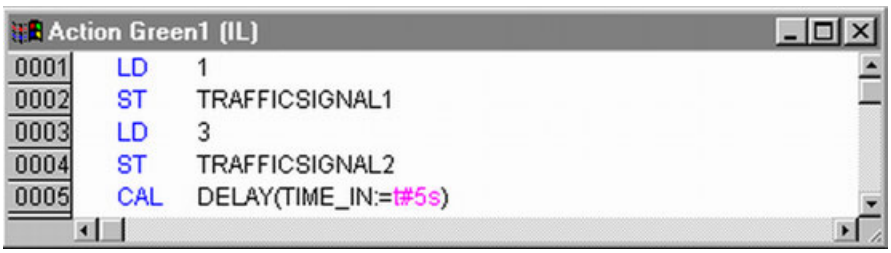
Switch1 changes the sate of TRAFFICSIGNAL1 to 2 (yellow), and that of TRAFFICSIGNAL2 to 4 (yellow-red). In addition, a time delay of 2000 milliseconds is set. The action is now as follows:

With Green2 TRAFFICSIGNAL1 is red (STATUS:=3), TRAFFICSIGNAL2 green (STATUS:=1), and the delay time is 5000 milliseconds.



At Switch2 the STATUS of TRAFFICSIGNAL1 changes to 4 (yellow-red), that of TRAFFIC-SIGNAL2 to 2 (yellow). A time delay of 2000 milliseconds is now set.

With Green1 TRAFFICSIGNAL1 is green (STATUS:=1), TRAFFICSIGNAL2 is red (STATUS:=3), and the time delay is set to 5000 milliseconds.



The first expansion phase of our program is completed.

If you want to do a first test of POU ABLAUF in simulation mode, perform the following steps:

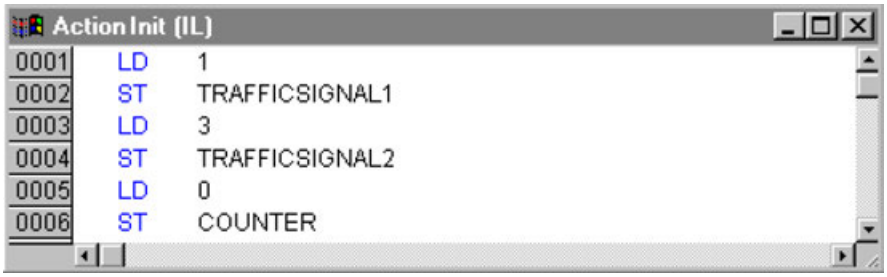
1. Open POU PLC_PRG. Each project starts running with PLC_PRG. In order to be able to provisionally start POU ABLAUF, insert a box and replace "AND" by "ABLAUF". Remain the inputs and outputs unassigned for the moment.
2. Compile the project via 'Project' 'Build'. In the message window you should get "0 Errors, 0 Warnings". Now check if option 'Online' 'Simulation' is activated and use command 'Online' 'Login' to get into simulation mode. Start program with 'Online' 'Start'. Open POU ABLAUF by a double-click on entry "ABLAUF" in the Object Organizer. The program is started now, but to get it run, variable START must be TRUE. Later this will be set by PLC_PRG but at the moment we have to set it manually within the POU. To do that, perform a double-click on the line in the declaration part, where START is defined (START=FALSE). This will set the option "<:=TRUE>" behind the variable in turquoise color. Now select command 'Online' 'Write values' to set this value. Thereupon START will be displayed in blue color in the sequence diagram and the processing of the steps will be indicated by a blue mark of the currently active step.

When you have finished this intermediate test use command 'Online' 'Logout' to leave the simulation mode and to continue programming.

"SEQUENCE" second expansion level

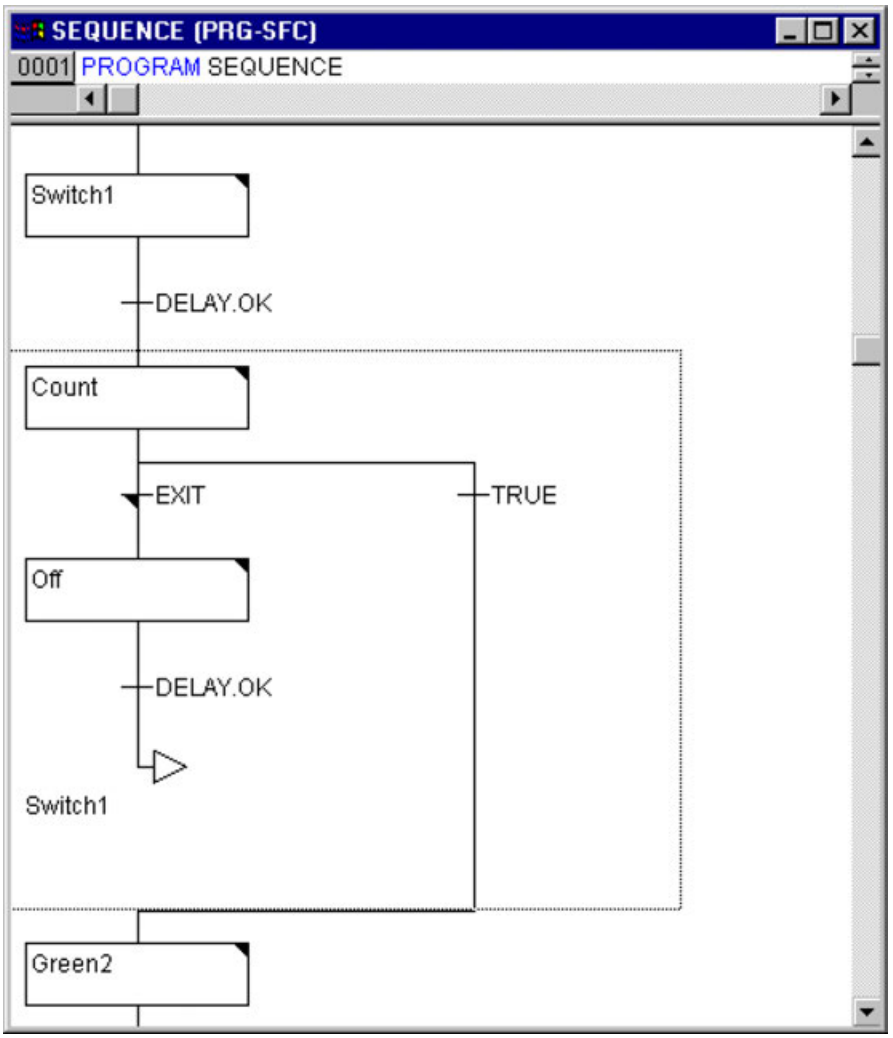
In order to ensure that our diagram has at least one alternative branch, and so that we can turn off our traffic light unit at night, we now include in our program a counter which, after a certain number of TRAFFICSIGNAL cycles, turns the unit off.

At first we need a new variable COUNTER of the type INT. Declare this as usual in the declaration part of SEQUENCE, and initialize it in Init with 0.

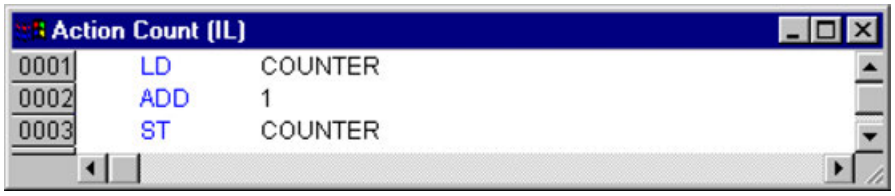


Now select the transition after Switch1 and insert a step and then a transition ↬ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172*. Select the resulting transition and insert an alternative branch to its left. After the left transition insert a step and a transition. After the resulting new transition insert a jump after Switch1.

Name the new parts as follows: the upper of the two new steps should be called "Count" and the lower "Off". The transitions are called (from top to bottom and from left to right) EXIT, TRUE and DELAY.OK. The new part should look like the part marked with the black border in the following figure:

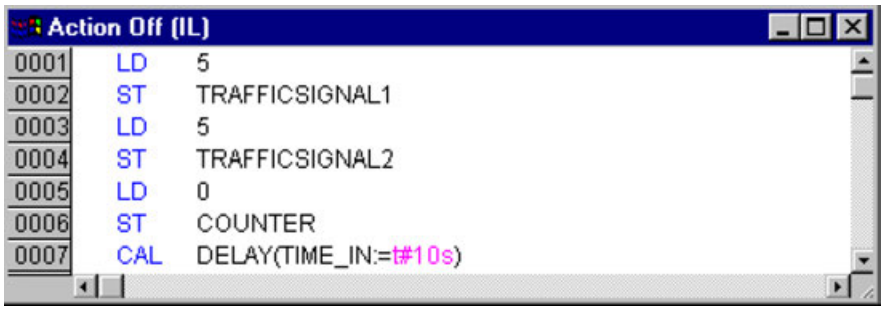


Now two new actions and a new transition condition are to be implemented ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171* ↳ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172*. At the step Count the variable COUNTER is increased by one:



The EXIT transition checks whether the counter is greater than a certain value, for example 7:

At Off the state of both lights is set at 5(OFF), (or each other number not equal 1,2,3 or 4) the COUNTER is reset to 0, and a time delay of 10 seconds is set:



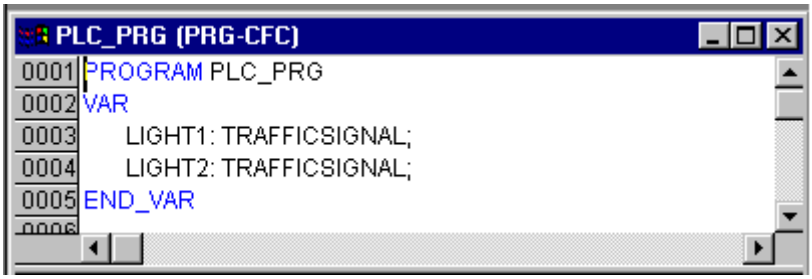
The result

In our hypothetical situation, night falls after seven trafficsignal cycles, for ten seconds the trafficsignal turns itself off, then we have daylight again, the traffic light unit turns itself on again, and the whole process starts again from the beginning. If you like, do another test of the current version of your program in simulation mode before we go on to create the POU PLC_PRG.

PLC_PRG

We have defined and correlated the time sequencing of the phases for both sets of traffic lights in the block SEQUENCE. Since, however, we see the traffic lights system as a module of a bus system, e.g. CAN bus, we have to make input and output variables available in the block PLC_PRG. We want to start-up the traffic lights system over an ON switch and we want to send each of the six lamps (each traffic light red, green, yellow) the corresponding "signal command" for each step of the SEQUENCE. We are now declaring appropriate Boolean variables for these six outputs and one input, before we create the programme in the editor, and are allocating them, at the same time, to the corresponding IEC addresses.

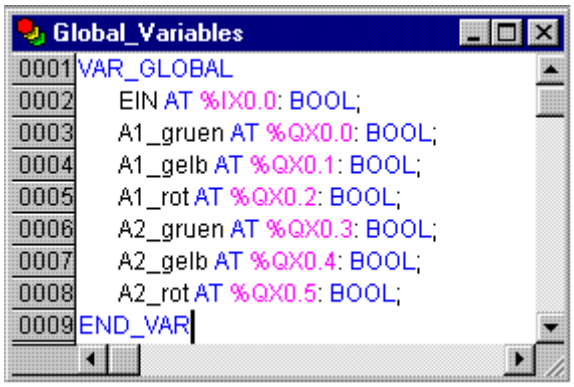
The next step is declare the variables Light1 and Light2 of the type Phases in the declaration editor.



These deliver the Boolean value of each of the six lights to the above mentioned six outputs for each step of the block SEQUENCE. We are not, however, declaring the output variables which are foreseen within the PLC_PRG block but under Resources for Global Variables instead. The Boolean input variable IN, which is used to set the variable START in the block SEQUENCE to TRUE, can be set in the same way. ON is also allocated to an IEC address.

Select the tab Resources and open the list Global Variables.

Make the declaration as follows:



The name of the variable (e.g. IN) is followed, after AT, by a percent sign which begins the IEC address. I stands for input, Q for output, B (used in the example) stands for byte and the individual bits of the module are addressed using 0.0 (0.1, 0.2, etc.). We will not do the needed controller configuration here in the example, because it depends on which target package you have available on your computer.

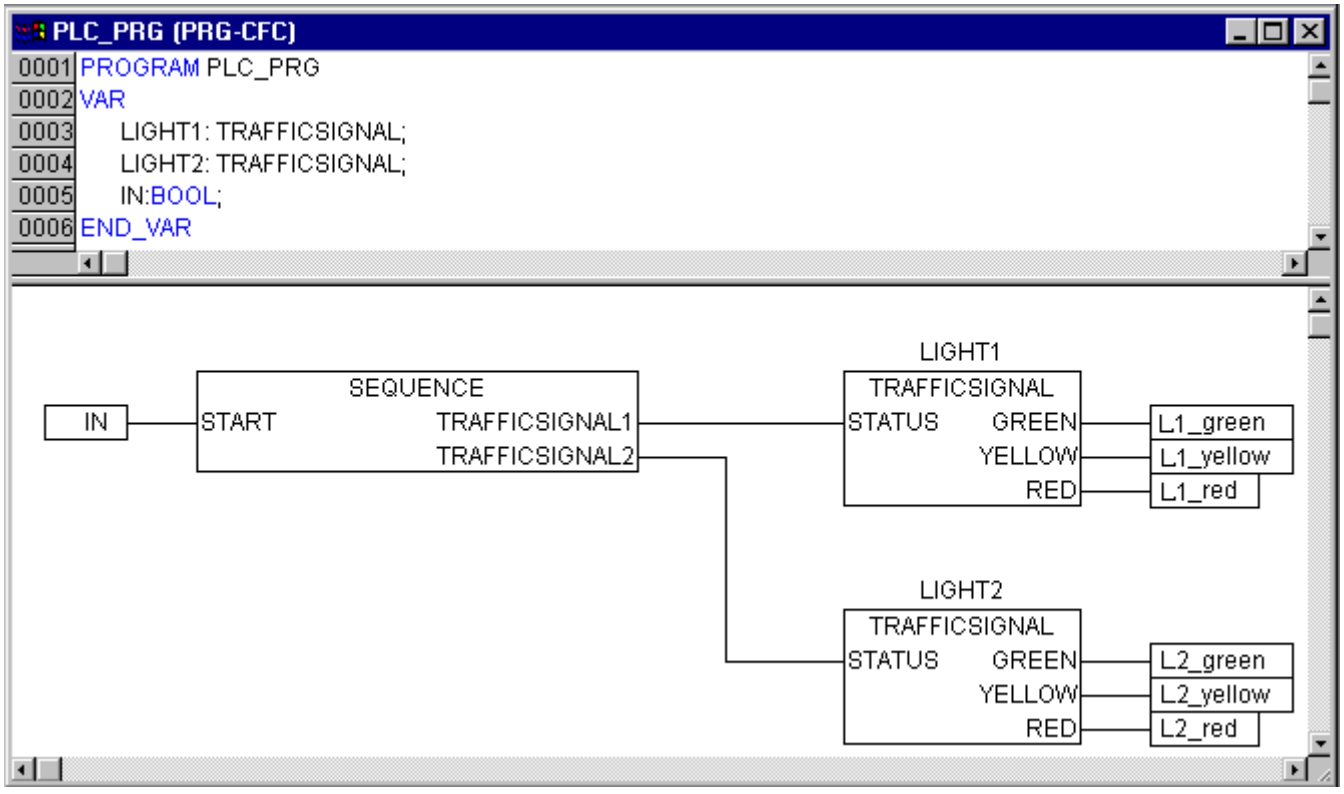
We now want to finish off the block PLC_PRG.

For this we go into the editor window. We have selected the Continuous Function Chart editor and we consequently obtain, under the menu bar, a CFC symbol bar with all of the available elements.

Click on the right mouse key in the editor window and select the element Box. Click on the text AND and write "SEQUENCE" instead. This brings up the block SEQUENCE with all of the already defined input and output variables. Insert two further block elements which you name PHASES. Phases is a function block and this causes you to obtain three red question marks over the block which you replace with the already locally declared variables LIGHT1 and LIGHT2. Now set an element of the type Input, which award the title ON and six elements of the type Output which you award variable names to, as described, namely L1_green, L1_yellow, L1_red, L2_green, L2_yellow, L2_red.

All of the elements of the programme are now in place and you can connect the inputs and outputs, by clicking on the short line at the input/output of an element and dragging this with a constantly depressed mouse key to the input/output of the desired element.

Your program should finally look like the example shown here.



TRAFFICSIGNAL simulation

Now test your program in simulation mode. Compile ("Project" "Build") and load ("Online" "Login") it. Start the program by "Online" "Start", then set variable ON to TRUE, e.g. by a double-click on the entry "ON" in the input box of the CFC editor. This will mark the variable as prepared to be set to <TRUE>. Then press <Ctrl><F7> or command "Online" "Write values", to set the value. Now variable START in ABLAUF (which we had set to TRUE manually in the first extension level of the program) gets this value by variable ON, which is used in PLC_PRG. This will make run the traffic light cycles. PLC_PRG has changed to a monitoring window. Click twice on the plus sign in the declaration editor, the variable display drops down, and you can see the values of the individual variables.

Visualizing a traffic signal unit

Creating a new visualization

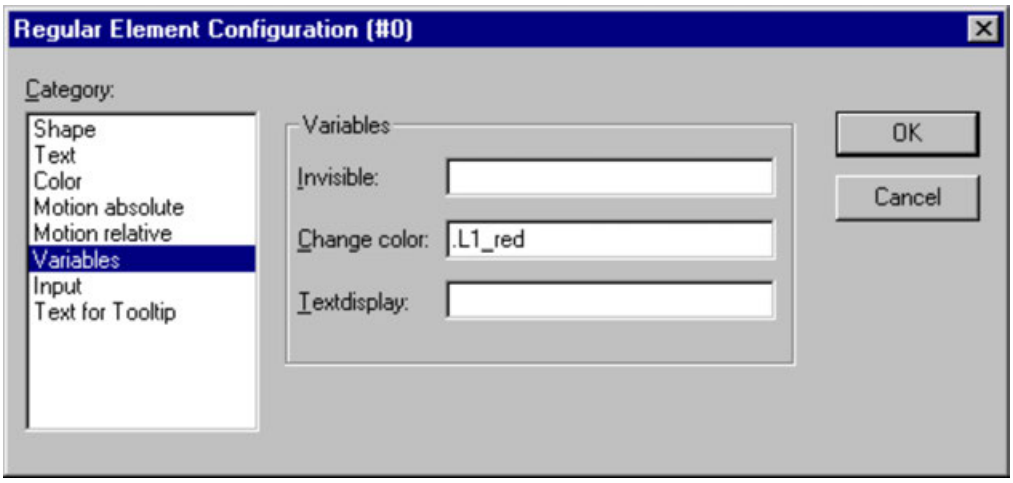
In order to create a visualization you must first select the range of Visualization in the Object Organizer. First click on the lower edge of the window on the left side with the POU on the register card with this symbol and the name Visualization. If you now choose the command 'Project' 'Object Add', then a dialog box opens *Chapter 1.4.1.2.4.6 "'Project' 'Object' ' Add'" on page 258*.

Enter here any name. When you confirm the dialog with OK, then a window opens in which you can set up your new visualization.

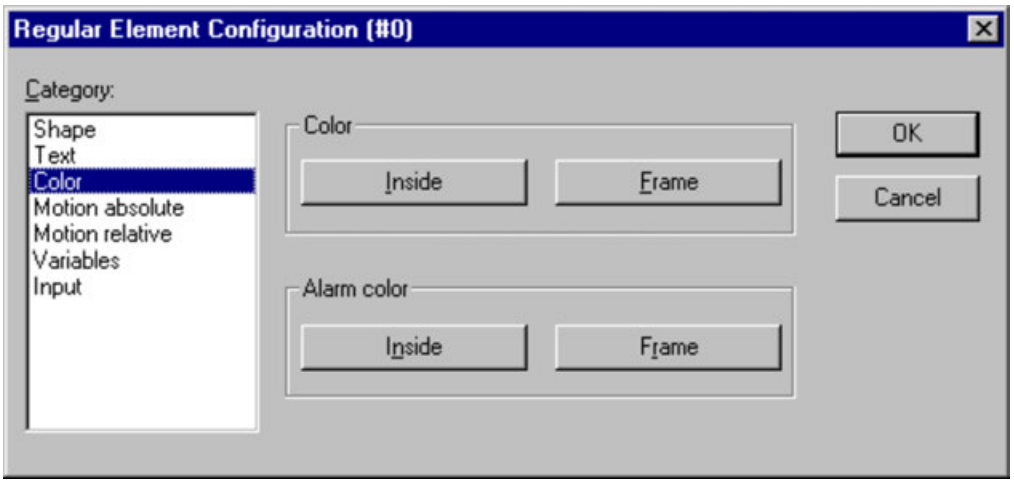
Insert element in visualization

For our TRAFFICSIGNAL visualization you should proceed as follows:

- Give the command 'Insert' 'Ellipse'.Insert..Ellipse.>Proc and try to draw a medium sized circle (?2cm). For this click in the editor field and draw with pressed left mouse button the circle in its length.
- Now doubleclick the circle. The dialog box for editing visualization elements opens
- Choose the category Variables and in the field Change color enter the variable name .L1_red or "L1_red". That means that the global variable L1_red will cause the color change as soon as it is set to TRUE. The dot before the variable name indicates that it is a global variable, but it is not mandatory.



- Then choose the category Color and click on the button Inside in the area Color. Choose as neutral a color as possible, such as black.
- Now click on the button within in the area Alarm color and choose the red which comes closest to that of a red light.



The resulting circle will normally be black, and when the variable RED from TRAFFICSIGNAL1 is TRUE, then its color will change to red. We have therefore created the first light of the first trafficsignal!

The other traffic lights

Now enter the commands 'Edit' 'Copy' (<Ctrl>+<C>) and then twice 'Edit' 'Paste' (<Ctrl>+<V>) ↳ *Chapter 1.4.1.2.5.5 “'Edit' 'Copy'” on page 273* ↳ *Chapter 1.4.1.2.5.6 “'Edit' 'Paste'” on page 274* . That gives you two more circles of the exact same size lying on top of the first one. You can move the circles by clicking on the circle and dragging it with pressed left mouse button. The desired position should, in our case, be in a vertical row in the left half of the editor window. Doubleclick on one of the other two circles in order to open the configuration dialog box again. Enter in the field Change Color of the corresponding circle the following variables:

- for the middle circle: L1_yellow
- for the lowest circle: L1-green

Now choose for the circles in the category 'Color' and in the area 'Alarm color' the corresponding color (yellow or green).

The TRAFFICSIGNAL case

Now enter the command 'Insert' 'Rectangle', and insert in the same way as the circle a rectangle which encloses the three circles. Once again choose as neutral a color as possible for the rectangle and give the command 'Extras' 'Send to back' so that the circles are visible again.

If simulation modeis not yet turned on, you can activate it with the command 'Online' 'Simulation' ↳ *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291*.

If you now start the simulation with the commands 'Online' 'Login' and 'Online' 'Run', then you can observe the color change of the first traffic signal ↳ *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279* ↳ *Chapter 1.4.1.2.6.6 “'Online' 'Run'” on page 283*.

The second traffic signal

The simplest way to create the second traffic signal is to copy all of the elements of the first traffic signal. For this you select all elements of the first traffic signal and copy them (as before with the lights of the first traffic signal) with the commands 'Edit' 'Copy' and 'Edit' 'Paste'. You then only have to change the text "TRAFFICSIGNAL1" in the respective dialog boxes into "TRAFFICSIGNAL2", and the visualization of the second traffic signal is completed.

The ON switch

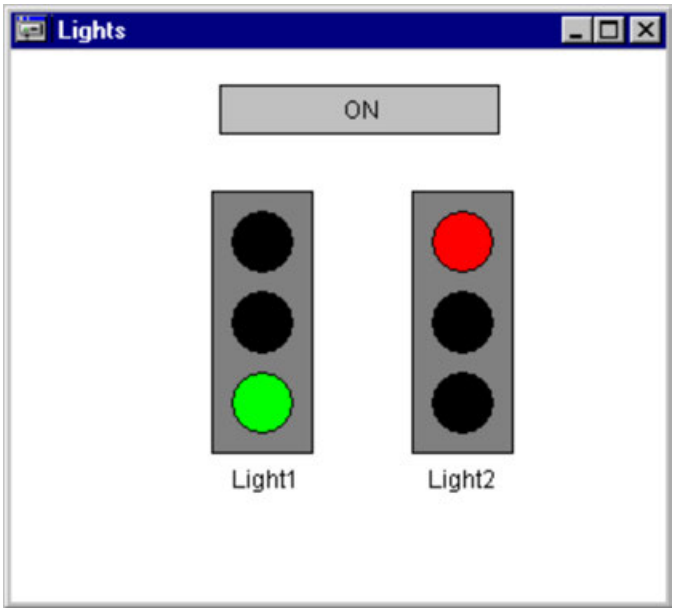
Insert a rectangle and award it, as described above, a colour for a traffic light of your choice and enter .ON at *“Variables”* for the *“Change color”*. Enter "ON" in the input field for *“Content”* in the category Text.

In order to set the variable ON to TRUE with a mouse click on the switch, activate option 'Toggle variable' in category 'Input' and enter variable name ".ON" there. Variable keying means that when a mouse click is made on the visualization element the variable .ON is set to the value TRUE but is reset to the value FALSE when the mousekey is released again (we have created hereby a simple switch-on device for our traffic lights program).

Font in the visualization

In order to complete the visualization you should first insert two more rectangles which you place underneath the traffic signals.

In the visualizations dialog box set white in the category Color for Frame and write in the category Text in the field Contents "Light1" or "Light2". Now your visualization looks like this:

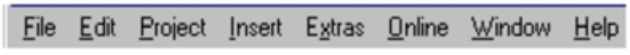


1.4.1.2 The individual components

1.4.1.2.1 The main window

Menu bar

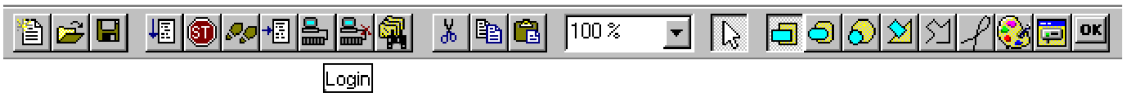
The menu bar is located at the upper edge of the main window. It contains all menu commands.



Tool bar

By clicking with the mouse on a symbol you can select a menu command more quickly. The choice of the available symbols automatically adapts itself to the active window.

The command is only carried out when the mouse button is pressed on the symbol and then released. If you hold the mouse pointer for a short time on a symbol in the tool bar, then the name of the symbol is shown in a Tooltip. In order to see a description of each symbol on the tool bar, select in *"Help"* the editor about which you want information and click on the tool bar symbol in which you are interested. The display of the tool bar is optional.



Object organizer

The Object Organizer is always located on the left side of the program. At the bottom there are four register cards with symbols for the four types of objects:

- POUs
- Data types
- Visualizations
- Resources

In order to change between the respective object types click with the mouse on the corresponding register card or use the left or right arrow key.

Screen divider

The screen divider is the border between two non-overlapping windows. There are screen dividers between the object organizer and the of the main window, between the interface (declaration part) and the implementation (instruction part) of POUs and between the work space and the message window ↳ *Chapter 1.4.1.2.1.3 "Object organizer" on page 199* ↳ *Chapter 1.4.1.2.1 "The main window" on page 198* ↳ *Chapter 1.4.1.3.9.2 "Declaration part" on page 297*.

You can move the screen divider with the mouse pointer. You do this by moving the mouse with the left mouse button pressed.

Make sure the screen divider always remains at its absolute position, even when the window size has been changed. If it seems that the screen divider is no longer present, then simply enlarge your window.

Workspace

This object in the 'Resources' tab provides an figure of the currently set project options. If you open it, you get the project options dialog with the know categories ↳ *Chapter 1.4.1.2.2.1 "'Project' 'Options'" on page 200*.

Message window

The message window is separated by a screendivider underneath the workspace in the main window ↳ *Chapter 1.4.1.2.1.4 “Screen divider” on page 199* ↳ *Chapter 1.4.1.2.1.5 “Workspace” on page 199*.

It contains all messages from the previous compilations, checks or comparisons. Search results and the cross-reference list can also be output here.

If you doubleclick with the mouse in the message window on a message or press *[Enter]*, the editor opens with the object. The relevant line of the object is selected. With the commands 'Edit' 'Next error' and 'Edit' 'Previous error' you can can quickly jump between the error messages.

Status bar

The status bar at the bottom of the window frame of the main window gives you information about the current project and about menu commands.

If an item is relevant, then the concept appears on the right side of the status bar in black script, otherwise in gray script.

When you are working in online mode, the concept Online appears in black script. If you are working in the offline mode it appears in gray script.

In Online mode you can see from the status bar whether you are in the simulation (SIM), the program is being processed (RUNS), a breakpoint is set (BP), or variables are being forced (FORCE) ↳ *Chapter 1.4.1.1.11.8 “Simulation” on page 184*.

With the text editor the line and column number of the current cursor position is indicated (e.g. Line:5, Col.:11). In online mode 'OV' is indicated black in the status bar. Pressing the *[Ins]* key switches between Overwrite and Insert mode.

If the mouse point is in a visualization, the current X and Y position of the cursor in pixels relative to the upper left corner of the screen is given. If the mouse pointer is on an Element, or if an element is being processed, then its number is indicated. If you have an element to insert, then it also appears (e.g. Rectangle).

If you have chosen a menu command but haven't yet confirmed it, then a short description appears in the status bar.

Context menu

Shortcut: *[Shift]* + *[F10]*

Instead of using the menu bar for executing a command, you can use the right mouse button. The menu which then appears contains the most frequently used commands for a selected object or for the active editor. The choice of the available commands adapts itself automatically to the active window.

1.4.1.2.2 Project options

'Project' 'Options'

With this command the dialog box for setting options is opened. The options are divided into different categories. Choose the desired category on the left side of the dialog box by means of a mouse click or using the arrow keys and change the options on the right side.

An image of the options which are set for the current project, will be found in the Resources tab in component Workspace ↳ *Chapter 1.4.1.2.1.5 “Workspace” on page 199*.

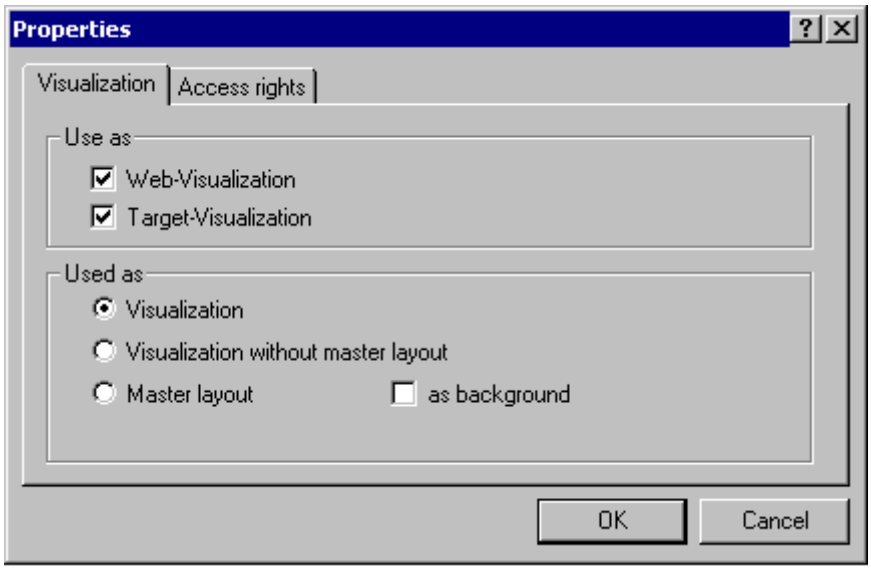
The settings amongst other things serve to configure the view of the main window. They are, unless determined otherwise, saved in the file "codesys.ini" and restored at the next startup.

You have at your disposal the following categories:

Category	Stored in CODESYS	Stored in project
Load & Save ↯ *Chapter 1.4.1.2.2.2 “Options for load & save” on page 201*	x	
User information ↯ *Chapter 1.4.1.2.2.3 “Options for user information” on page 203*	x	
Editor ↯ *Chapter 1.4.1.2.2.4 “Options for editor” on page 203*	x	
Desktop ↯ *Chapter 1.4.1.2.2.5 “Options for the desktop” on page 205*	x	
Color ↯ *Chapter 1.4.1.2.2.6 “Options for colors” on page 206*	x	
Directories ↯ *Chapter 1.4.1.2.2.7 “Options for directories” on page 207*	Cat. Common	Cat. Project
Log ↯ *Chapter 1.4.1.2.2.8 “Options for log” on page 208*	x	
Build ↯ *Chapter 1.4.1.2.2.9 “Options for build” on page 209*		
Passwords ↯ *Chapter 1.4.1.2.2.10 “Passwords” on page 211*		
Source download ↯ *Chapter 1.4.1.2.2.11 “'Source download'” on page 212*	x	
Symbol configuration ↯ *Chapter 1.4.1.2.2.12 “Options for 'Symbol configuration'” on page 214*	x	
Project Source Control ↯ *Chapter 1.4.1.2.2.13 “Options for 'Project source control'” on page 216*		x
Macros ↯ *Chapter 1.4.1.2.2.16 “Options for 'Macros'” on page 220*		x

Options for load & save

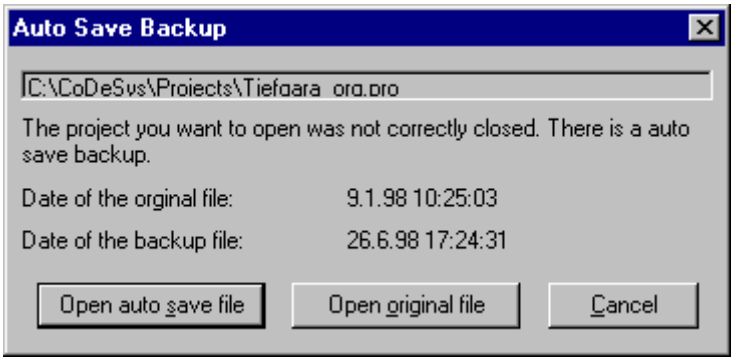
If you choose this category in the Options dialog box , then you get the following dialog box:



When activating an option, a check icon appears before the option.

Create Backup: a backup file is created at every save with the extension ".bak". Contrary to the *.asd-file (see below, 'Auto Save') this *.bak-file is kept after closing the project. So you can restore the version you had before the last project save.

Auto Save: While you are working, your project is saved according to a defined time interval (Auto Save Interval) to a temporary file with the extension ".asd" in the projects directory. This file is erased at a normal exit from the program. If for any reason Automation Builder is not shut down "normally" (e.g. due to a power failure), then the file will not get erased. When you open the file again the following message appears:



You can now decide whether you want to open the original file or the auto save file.

If a library *.lib is opened as project, a corresponding auto-save-file "*.asl" will be created.

Auto save before compile: The project will be saved before each compilation. In doing so a file with the extension ".asd" resp. ".asl" will be created, which behaves like described above for the option 'Auto Save'.

Ask for project info: When saving a new project or saving a project under a new name, the project info is automatically called. You can visualize the project info with the command 'Project' 'Project info' and also process it.

Auto Load: At the next start the last open project is automatically loaded. Loading of a project at the start of Automation Builder can also take place by entering the project in the command line.

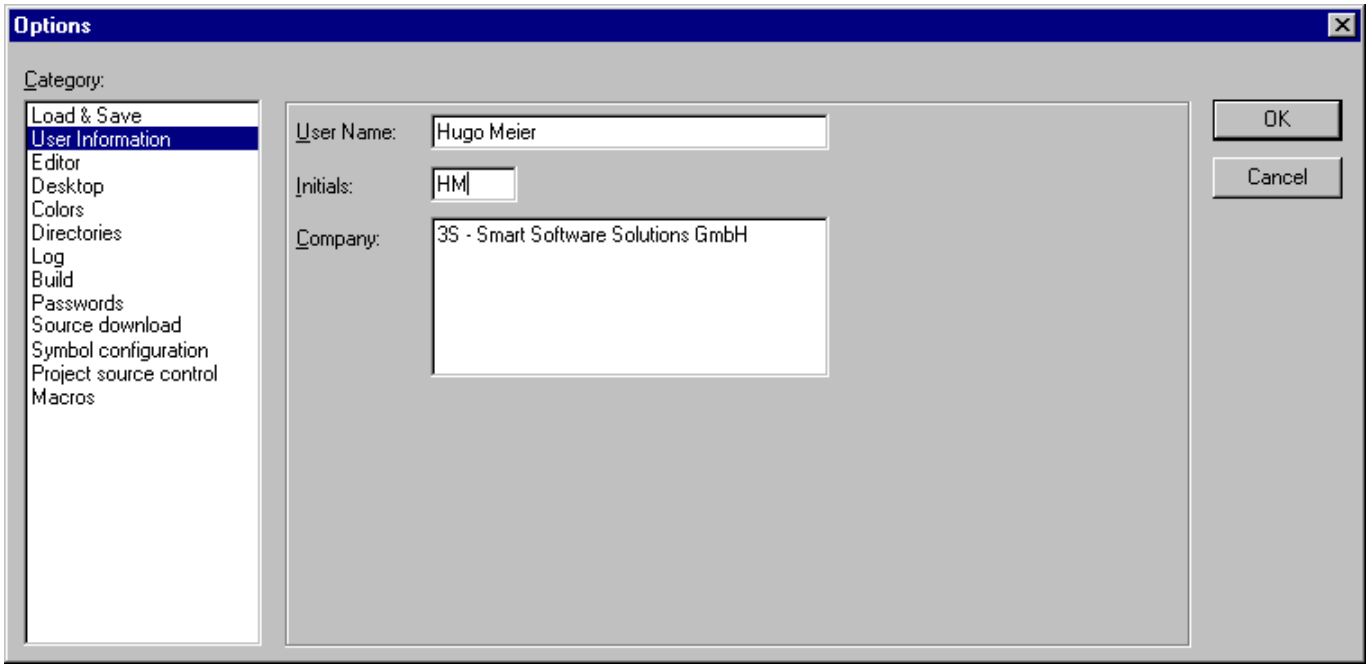
Remind of boot project on exit: If the project has been modified and downloaded without creating a new boot project since the last download of a boot project, then a dialog will advise the user before leaving the project: "No boot project created since last download. Exit anyway ?".

Save ENI credentials: User name and Password, as they might be inserted in the Login dialog for the ENI data base, will be saved. Concerning the access data, entered once by the user at 'Open project from source code manager' ('File' 'Open') in this case additionally username and password will be saved in the codesys.ini file.

The settings will be stored in CODESYS.

Options for user information

If you choose this category in the Options dialog box, then you get the following dialog box:

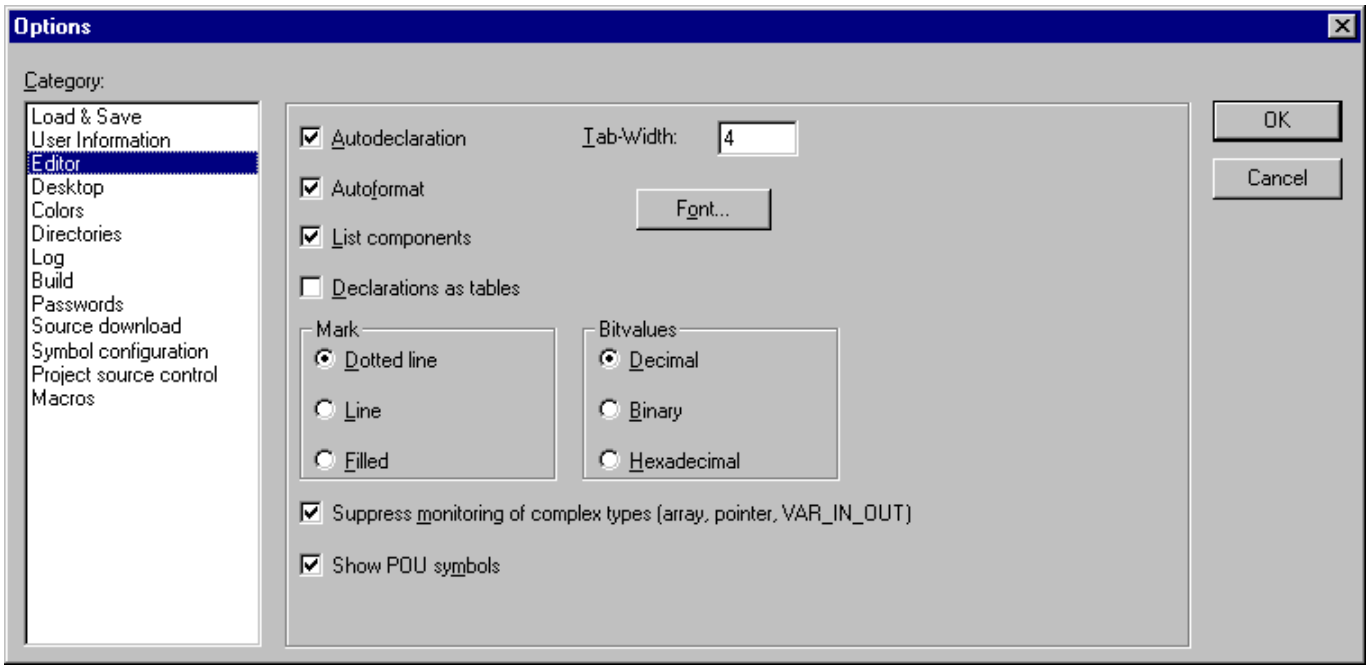


To User information belong the name of the user, his initials and the company for which he works. Each of the entries can be modified.

The settings will be stored in CODESYS.

Options for editor

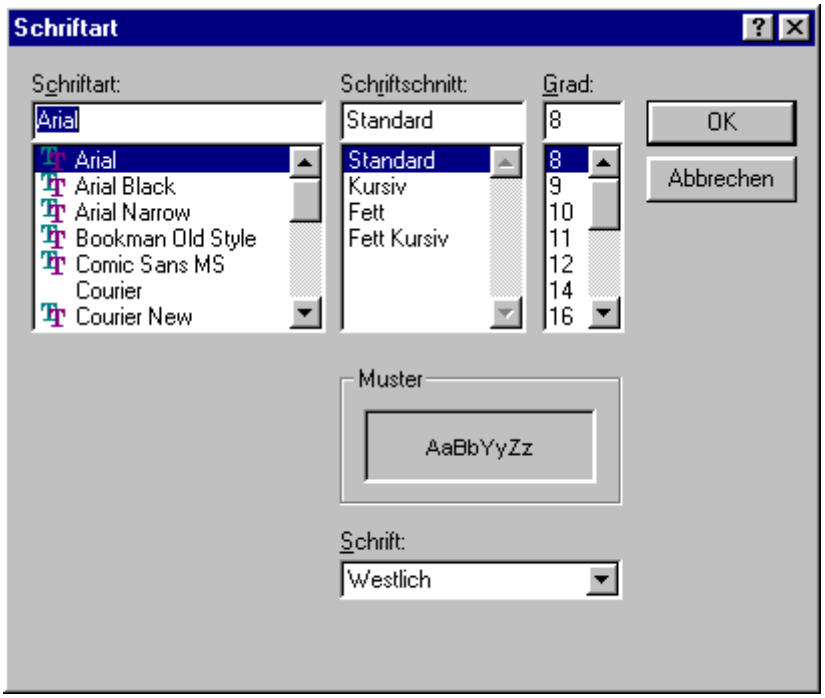
If you choose this category in the *“Options”* dialog box, then you get the following dialog box:



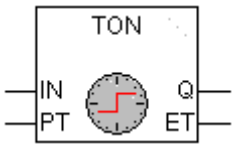
You can make the following settings for the Editors:

- Autodeclaration: If this option is activated , then after the input of a not-yet-declared variable a dialog box will appear in all editors with which this variable can be declared.
- Autoformat: If this option is activted, then automatic formatting is executed in the IL editor and in the declaration editor. When you finish with a line, the following formatting is made: 1.Operators written in small letters are shown in capitals; Tabs are inserted to that the columns are uniformly divided.
- List components: If this option is activated, then the Intellisense functionality will be available to work as an input assistant. This means that if you insert a dot at a position where a identifier should be inserted, then a selection list will open, offering all global variables which are found in the project. If you insert the name of a function block instance, then you will get a selection list of all inputs and outputs of the instanced function block. The Intellisense function is available in editors, in the Watch- and Receiptmanager, in visualizations and in the Sampling Trace.
- Declarations as tables: If this option is activated, then you can edit variables in a table instead of using the usual declaration editor. This table is sorted like a card box, where you find tabs for input variables, output variables local variables and in_out variables. For each variable there are edit fields to insert Name,Address, Type, Initial and Comment.
- Tab-Width: In the field Tab-Width in the category Editor of the Options dialog box you can determine the width of a tab as shown in the editors. The default setting is four characters, whereby the character width depends upon the font which is chosen.
- Font: By clicking on the button Font in the category Editor of the Options dialog box you can choose the font in all editors. The font size is the basic unit for all drawing operations. The choice of a larger font size thus enlarges the printout, even with each editor.

  After you have entered the command, the font dialog box opens for choosing the font, style and font size.



- Mark: When choosing Mark in the Editor category in the Options dialog box you can choose whether the current selection in your graphic editors should be represented by a dotted rectangle (Dotted), a rectangle with continuous lines (Line) or by a filled-in rectangle (Filled). In the last case the selection is shown inverted.
- Bitvalues: When choosing Bitvalues in the category Editor of the Options dialog box you can choose whether binary data (type BYTE, WORD, DWORD) during monitoring should be shown Decimal, Hexadecimal, or Binary.
- Suppress monitoring of complex types (Array, Pointer, VAR_IN_OUT): If this option is activated, complex data types like arrays, pointers, VAR_IN_OUTs will not get displayed in the monitoring window in online mode.
- Show POU symbols: If this option is activated, in the module boxes which are inserted to a graphic editor, additionally symbols will get displayed, if those are available in the library folder as bitmaps. The name of the bitmap-file must be composed of the name of the module and the extension ".bmp". Example: For module TON there is a symbol file TON.bmp available. The box will be displayed as follows:



The settings will be stored in CODESYS.

Options for the desktop

When an option is activated, a check appears in front of it.

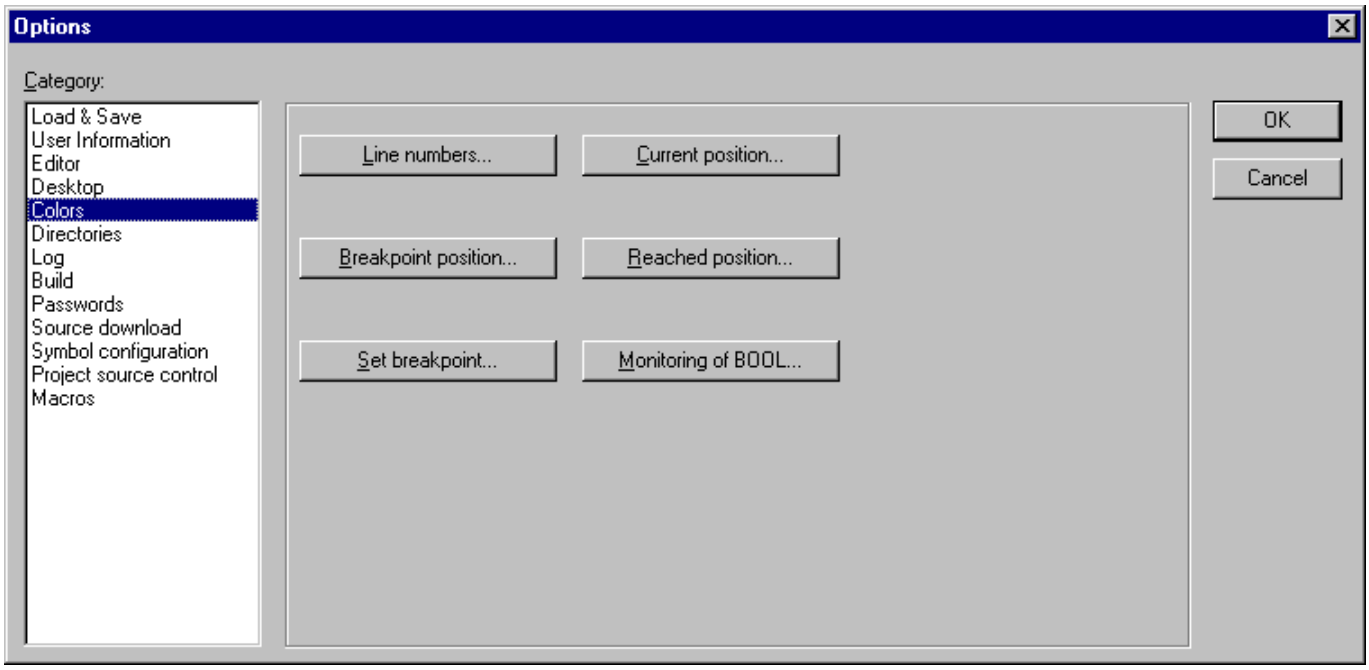
- Tool bar: The tool bar with the buttons for faster selection of menu commands becomes visible underneath the menu bar.
- Status bar: The status bar at the lower edge of the main window becomes visible.

- Online in Securitymode: In Online mode a dialog box appears with the confirmation request whether the command should really be executed. If supported by the target system, an extended dialog might be available when you want to load the actual project from the programming system to the PLC If there is already a project on the PLC, this dialog will display the project information of that project as well as the information of the project currently to be loaded. This project information also will be available in case of creating a boot project when there is already one on the PLC. This option is saved with the project.
- Query communication parameters before login: As soon as the command 'Online' 'Login' is executed, first the communication parameters dialog will open. To get in online mode you must first close this dialog with OK.
- Do not save communication parameters in project: The settings of the communication parameters dialog ('Online' 'Communication Parameters') will not be saved with the project.
- Show print area margins:In every editor window, the limits of the currently set print range are marked with red dashed lines. Their size depends on the printer characteristics (paper size, orientation) and on the size of the "Content" field of the set print layout (menu: 'File" "Documentation Settings").
- F4 ignores warnings:After compilation, when F4 is pressed in a message window, the focus jumps only to lines with error messages; warning messages are ignored.
- MDI representation: Per default this option (Multiple-Document-Interface) is activated and thus several windows can be opened at the same time. If the option is deactivated (SDI mode) only one window can be opened and will be displayed in full screen mode. Exception: The action of a program and the program itself can be displayed side by side even in MDI mode.
- Communications timeout [ms]: for standard communication services: Time span in milliseconds, after which the communication to the target system will be terminated if no more activity is detected. Possible values: 1-10000000 ms.
- Communications timeout for download [ms]: for long lasting communication services (program download, file up- and download, boot project creation and check): Time span in milliseconds after which the communication to the target system will be terminated if no more activity is detected (Download Wait Time). Possible values: 1-10000000 ms.
- XML-Encoding: The format for XML-exports can be selected. The default setting is "ISO 8859-1". This concerns the communication via ENI, Message Interface and COM Automation Interface, as well as each user-triggered XML-export. An exception is the XML-export of the Licensing Manager.
- Language: Define here, in which language the menu and dialog texts should be displayed.

The settings will be stored in CODESYS.

Options for colors

If you choose this category in the Options dialog box , then you get the following dialog box:



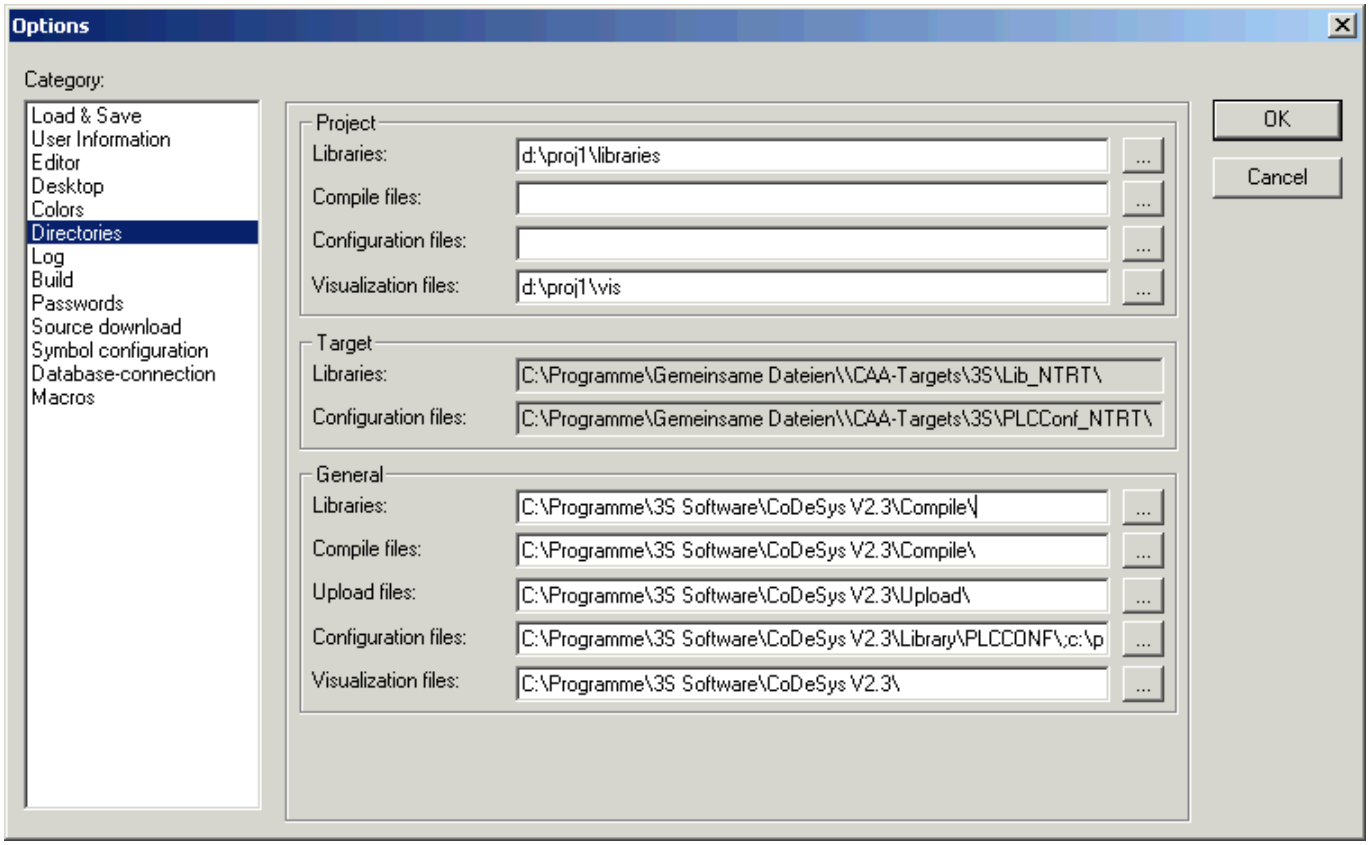
You can edit the default color setting of CODESYS. You can choose whether you want to change the color settings for Line numbers (default presetting: light gray), for Breakpoint positions (dark gray), for a Set breakpoint (light blue), for the Current position (red), for the Reached Positions (green) or for the Monitoring of Boolean values (blue).

If you have chosen one of the indicated buttons, the dialog box for the input of colors opens.

The settings will be stored in CODESYS.

Options for directories

If you choose this category in the Options dialog box, then you get the following dialog box:



Directories can be entered in the Project and Common areas for CODESYS to use in searching for Libraries, controller Configuration and Visualization files (bitmaps, XML file for dynamic texts etc.) iles, as well as for storing Compile and source-Upload files. (Compile files for example are map- and list-files, not however e.g. symbol files ! The latter will be saved in the project directory.) If you activate the button (...) behind a field, the directory selection dialog opens. For library and configuration files, several paths can be entered for each, separated by semicolons ";".

> *Please regard: Library paths can be entered based on the project file's path by prefixing a dot ".". If e.g. ".\libs" is entered, the libraries will be searched in 'C:\programs\projects\libs', if the current project is in 'C:\programs\projects'. For information on library paths see also: 'Insert' 'Additional Library'.*
>
> *Please regard: Do not use empty spaces and special characters except for "_" in the directory pathes.*

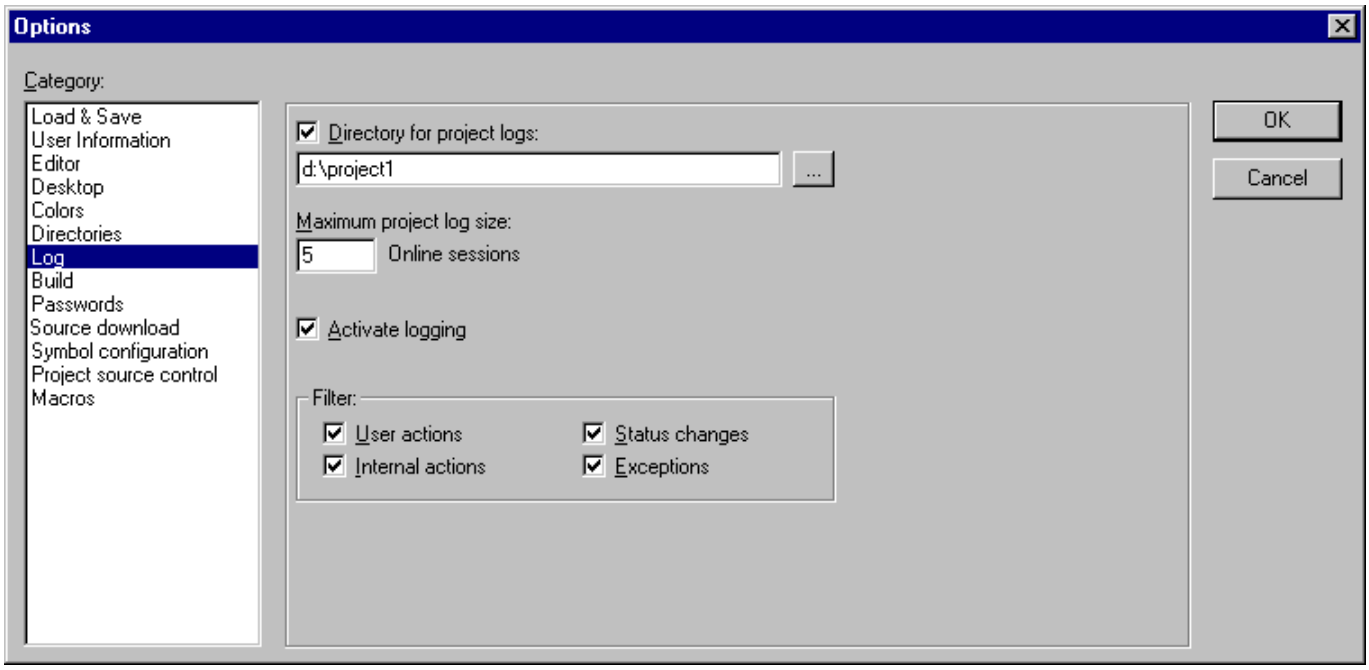
The Target area just displays the directories for libraries and configuration files set in the target system, e.g. through entries in the Target file. These fields cannot be edited, but an entry can be selected and copied (right mouse button context menu).

CODESYS generally searches first in the directories entered in 'Project', then in those in 'Target' (defined in the Target file), and finally those listed under 'Common'. If two files with identical names are found, the one in the directory that is browsed first will be used.

> *The settings for area Common will be stored in CODESYS, those for area Project in the project..*

Options for log

If you choose this category in the Options dialog box, then you get the following dialog box:



In this dialog, you can configure a file that acts as a project log, recording all user actions and internal processes during Online mode processing (see also: Log).

If an existing project is opened for which no log has yet been generated, a dialog box opens which calls attention to the fact that a log is now being set up that will receive its first input after the next login process.

The log is automatically stored as a binary file in the project directory when the project is saved. If you prefer a different target directory, you can activate the option Directory for project logs: and enter the appropriate path in the edit field. Use the button to access the "Select Directory" dialog for this purpose:

The log file is automatically assigned the name of the project with the extension .log. The maximum number of Online sessions to be recorded is determined by Maximum project log size. If this number is exceeded while recording, the oldest entry is deleted to make room for the newest.

The Log function can be switched on or off in the Option field Activate logging.

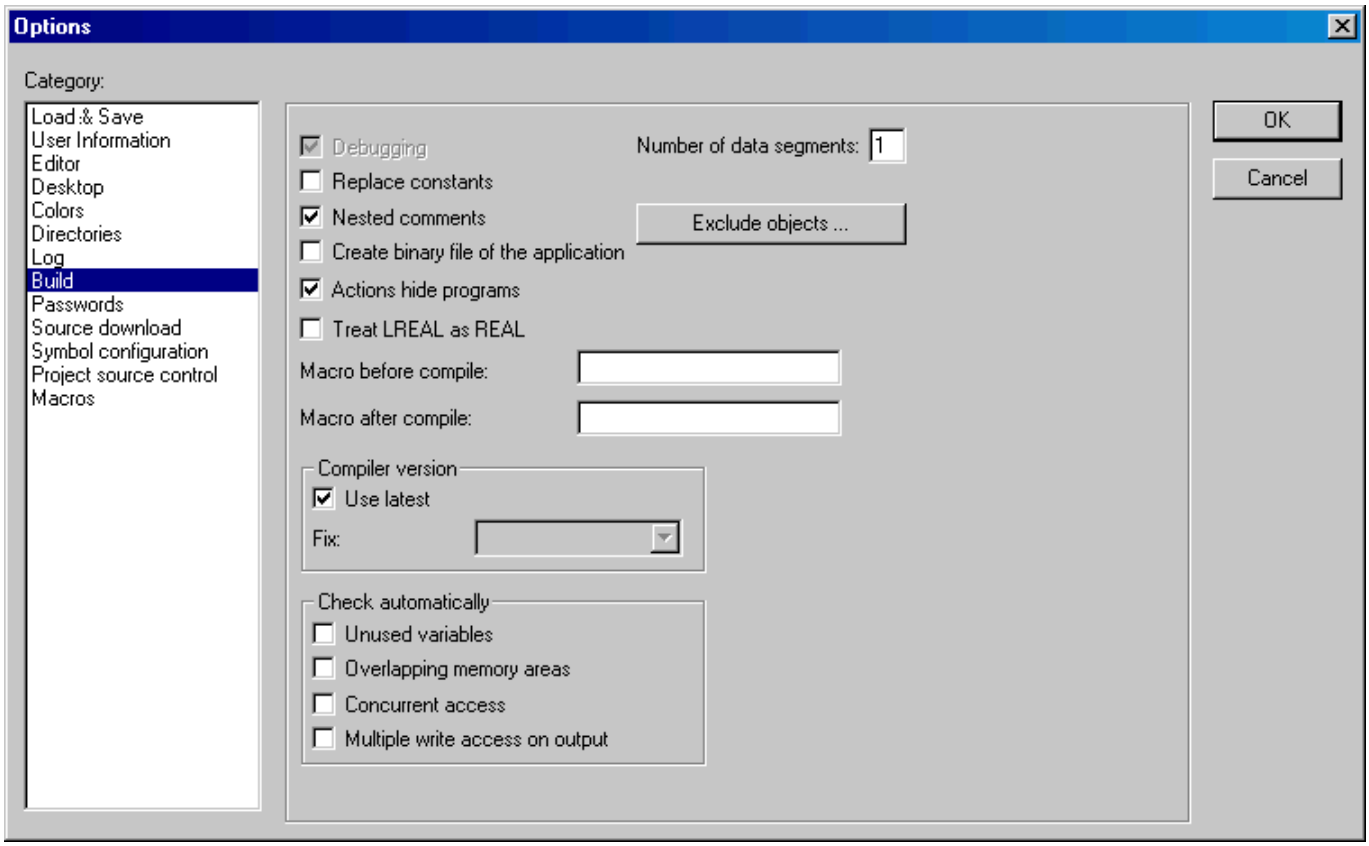
You can select in the Filter area which actions are to be recorded: User actions, Internal actions, Status changes, Exceptions. Only actions belonging to categories checked here will appear in the Log window and be written to the Log file.

The Log window can be opened with the command 'Window' 'Log'.

The settings will be stored in CODESYS.

Options for build

If you choose this category in the Options dialog box , then you get the following dialog box:



- Debugging: It depends on the target descriptions whether this option can be activated/deactivated by the user resp. which is the given setting . If it is activated, additional debugging code is created, that is the code can become considerably larger. The debugging code is needed in order to make use of the debugging functions (e.g. breakpoints). When you switch off this option, project processing becomes faster and the size of the code decreases. The option is stored with the project.
- Replace constants: The value of each constant of scalar type (thus not for strings, arrays and structures) is loaded directly, and in Online mode the constants are displayed in green. Forcing, writing and monitoring of a constant is then no longer possible. If the option is deactivated, the value is loaded into a storage location via variable access (this does in fact allow writing the variable value, but implies longer processing time).
- Nested comments: Comments can be placed within other comments.

  Example:

  (*
  a:=inst.out; (* to be checked *)
  b:=b+1;
  *)

  Here the comment that begins with the first bracket is not closed by the bracket following "checked," but only by the last bracket.

  Attention: Currently this option must be used carefully: If the setting in the project does not match the setting chosen in a library which also has been created in CODESYS and now is used is the project, compiler errors will occur, which are hard to interpret by the user and often cannot be cleared!
- Create binary file of the application: A binary image of the generated code (boot project) is created in the project directory during compilation. File name: <project_name>.bin. By comparison, the command *“Online ➔ Create boot project”* sets up the boot project on the controller *Chapter 1.4.1.2.6.25 “'Online' 'Create boot project'” on page 291*.
- Actions hide programs: This option is activated per default, when a new project is created. It means: If a local action has the same name like a global variable or a program, the following hierarchy is valid: local variable before local action before global variable before program.

  Regard: If an existing project is opened, which has been created with a previous version of CODESYS, the option will be deactivated per default. Thus the previously vlid hierarchy (local varaible before global variable before program before local action) can be kept.

- Treat LREAL as REAL: If this option is activated, (availability depends on the runtime system, default: not activated), the compile will handle LREAL values as REAL values. This can be used for creating platform independent projects.
- Number of data segments: Here you define how many memory segments should be allocated in the PLC for the project data. This space is needed to make possible online changes even if new variables have been added. If during compilation you get the message "Out of global data memory...", enter a higher number. In this regard local program variables will be handled like global variables ↳ *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*.
- Exclude objects: This button opens the dialog Exclude objects from build: In the tree of project components select those POUs which should not be regarded during compilation and activate option Exclude. Hereupon the excluded POUs will be displayed green-colored in the selection tree. Press button Exclude unused, if you just want to get displayed those POUs which are currently used in the program. Regard that a single object which is selected in the Object Organizer can also be excluded from build by using the command 'Exclude from build' from the context menu.
- Compiler Version: Here you define the compiler version to be used. CODESYS versions after V2.3.3 (version, service pack, patch) will include besides the actual compiler version also the previous compiler versions (back to V2.3.3). If you want to get the project compiled with the latest compiler version in any case, activate option Use latest. In this case however it will be checked whether the currently opened programming system is also of that version. If this is not true, the compiler version matching the actually used programming system version will be used!. If the project should be compiled with a specific version, define this via the selection list at Fix.

  In order to exert control over the compilation process you can set up two macros:

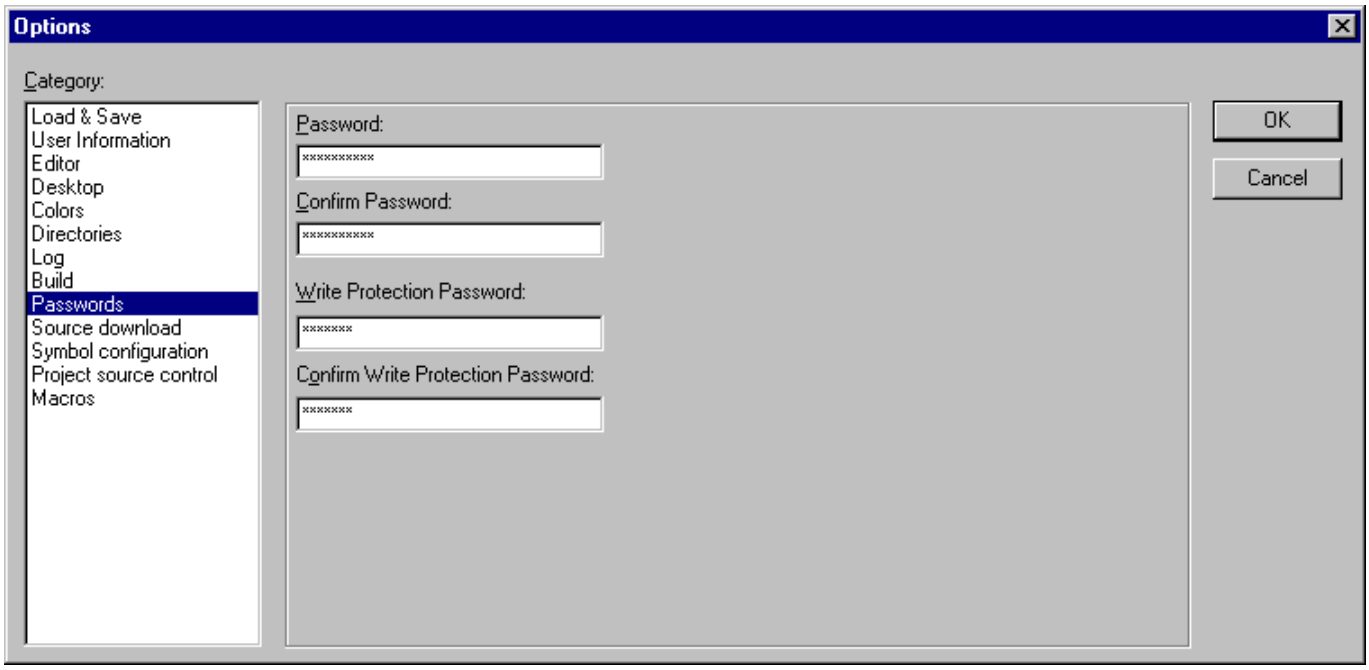
  The macro in the Macro before compile field is executed before the compilation process; the macro in the Macro after compile field afterwards. The following macro commands can not, however, be used here: file new, file open, file close, file save as, file quit, online, project compile, project check, project build, project clean, project rebuild, debug, watchlist.
- Check automatically: In order to get the semantic correctness checked at each compilation of the project the following options can be activated:
  – Unused variables
  – Overlapping memory areas
  – Concurrent access
  – Multiple write access on output

  The results will be displayed in the message window. These checks also can be initiated by the respective commands of the 'Check' submenu in the 'Project' menu. If supported by the target system, negative check results will produce compiler errors.

All entries in the Build Options dialog are stored with the project.

Passwords

If you choose this category in the Options dialog box, then you get the following dialog box:



To protect your files from unauthorized access use a password to protect against your files being opened or changed.

Enter the desired password in the field Password. For each typed character an asterisk (*) appears in the field. You must repeat the same word in the field Confirm Password. Close the dialog box with OK. If you get the message:

```
"The password does not agree with the confirmation",
```

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

If you now save the file and then reopen it, then you get a dialog box in which you are requested to enter the password. The project can then only be opened if you enter the correct password.

Along with the opening of the file, you can also use a password to protect against the file being changed. For this you must enter a password in the field Write Protection Password and confirm this entry in the field underneath.

A write-protected project can be opened without a password. For this, press the button Cancel, if a message is prompted to enter the write protection password when opening a file. Now you can compile the project, load it into the PLC, simulate, etc., but you cannot change it.

The passwords are saved with the project.

In order to create differentiated access rights you can define user groups and 'Passwords for user groups').

Additionally regard the extended possibilities to protect a project by encryption which for example can help to protect a library from getting used without having entered a key.

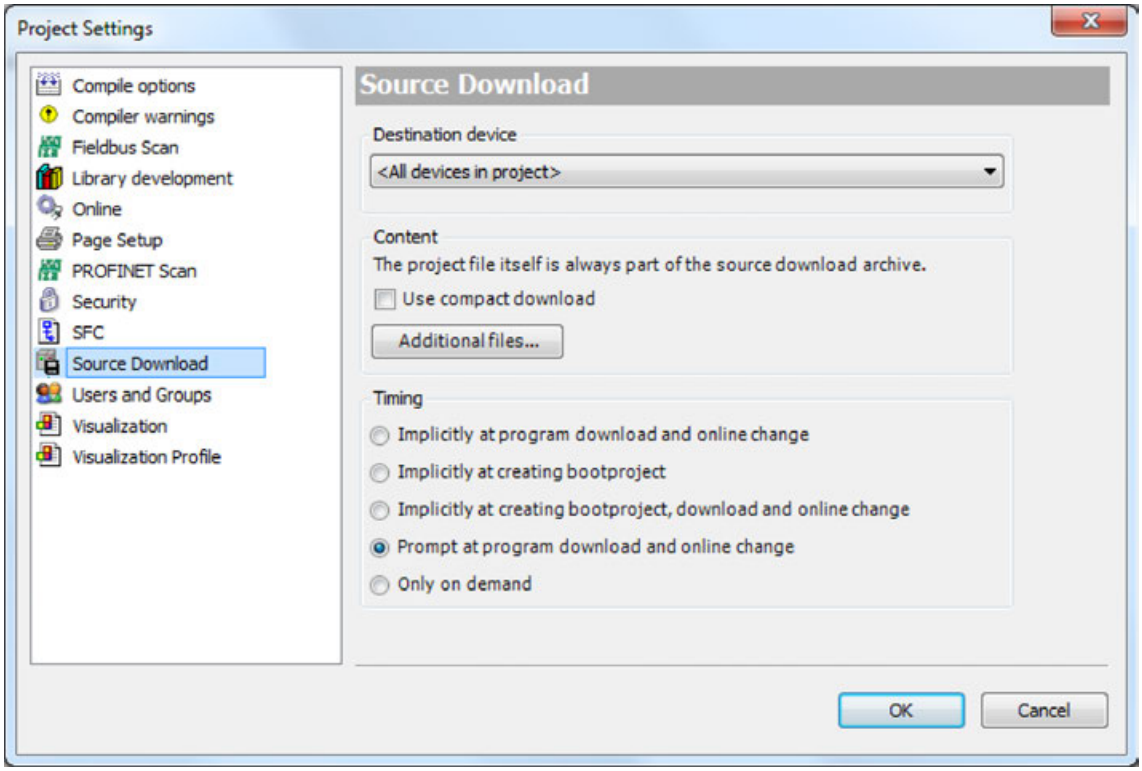
The settings will be stored in the project.

'Source download'

The user can configure a source code download (i.e. the project archive of the open project) to an memory card of any V2 PLC defined in the project. This ensures that the project on a PLC matches the current application, i.e. for maintenance purposes the project can be uploaded directly from the PLC.

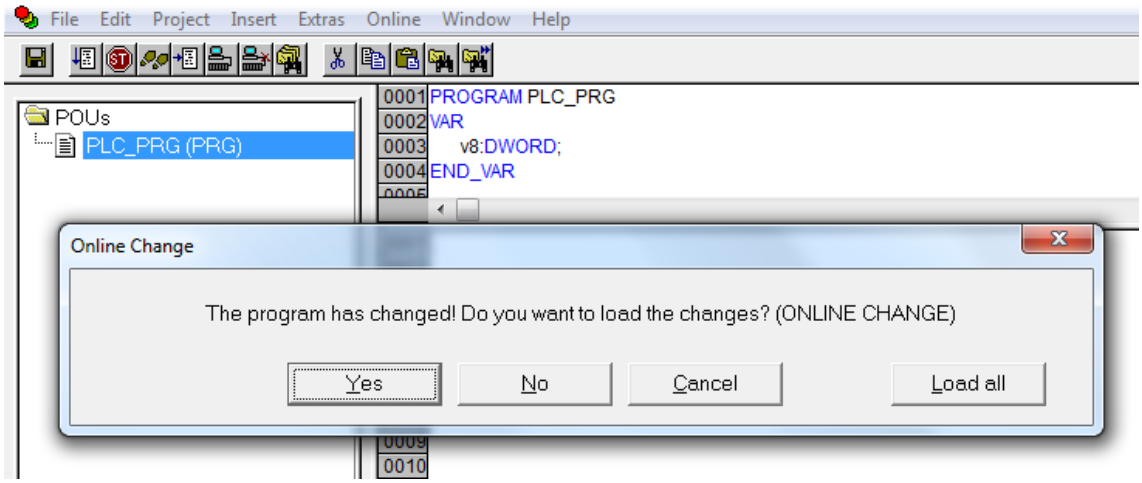
The automatic source code download ensures that the V2 PLC always has the up-to-date project archive on its memory card. With each online change or download of the application program (that can be triggered via Automation Builder or via CODESYS editor) the Automation Builder will send the current project archive to the PLCs memory card.

Enable one of the given project options in the 'Timing' section to trigger an automatic source code download on each online change or application download.



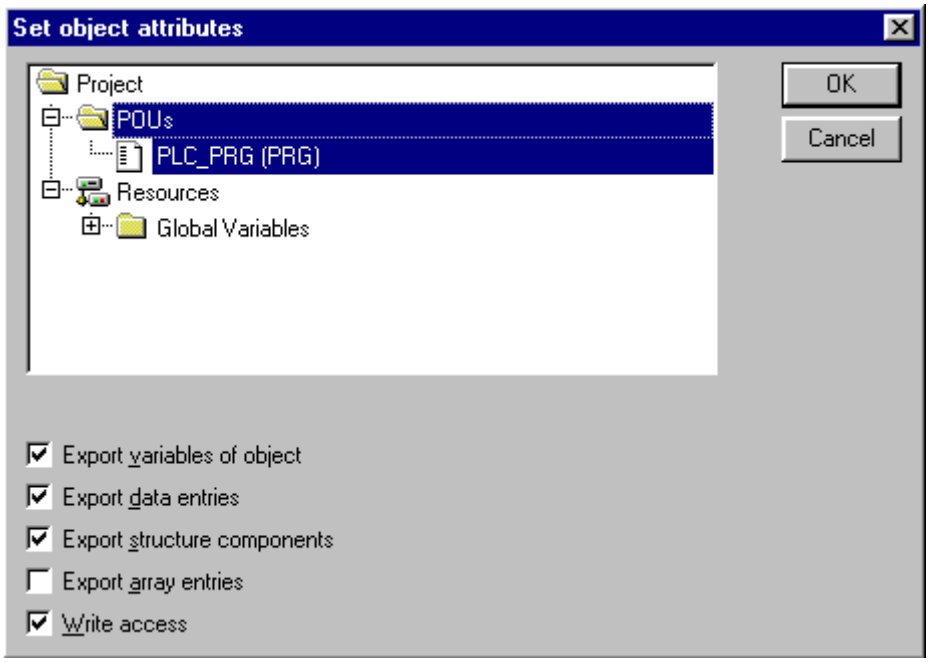
- *The option ‘Only on demand’ will never trigger any automatic download.*
- *The option ‘Prompt at program download and online change’ will prompt a dialog before downloading the source code. Only one dialog is prompted, even if multiple consecutive actions in CODESYS might have triggered a download. By this, a user can finish configuring all required changes in the application program before Automation Builder finally downloads all changes to the PLCs memory card.*
- *If no memory card is inserted in the PLC, the download fails with an error message.*

A source code download can be triggered from the CODESYS editor via online login after any change in the application program:



With a source code download always the complete project archive is downloaded. This might lead to inconsistencies between the applications and the project archive if an AC500-S device is envolved. For example, if there are recent changes in a non-safety PLC that have not been downloaded yet, while performing a download to the AC500-S PLC.

Options for 'Symbol configuration'



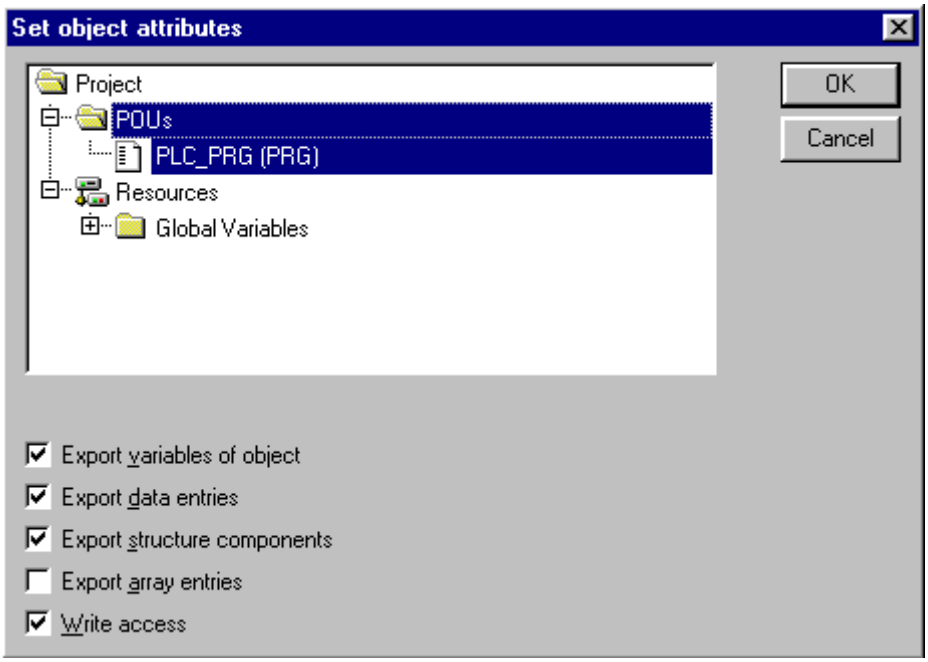
The dialog presented here is used for configuring the symbol file which will be created during each compilation of the project. The symbol file is created as a text file <project name>.sym respectively as a binary file <project name>.sdb (the format is depending on the used gateway version) in the project directory. The file is needed for data exchange with the controller via the symbolic interface and will be used for that purpose e.g. by OPC- or GatewayDDE-Server.

If the option Create symbol entries is activated, then symbol entries for the project variables will be automatically written to the symbol file. Otherwise only version info about file and project is contained.

If additionally the option Dump XML symbol table is activated, then also an XML file containing the symbol information will be created in the project directory. It will be named <project name>.SYM_XML.

Regard the following when configuring the symbol entries:

- If option 'Symbol config from INI-file' is activated in the target settings, then the symbol configuration will be read from the codesys.ini file or from another ini-file which is defined there. (In this case the dialog 'Set object attributes' cannot be edited.)
- If option 'Symbol config from INI-file' is not activated, the symbol entries will be generated in accordance with the settings you can make in the 'Set object attributes' dialog. You get there using the Configure symbol file button:



Use the tree-structured selection editor to mark the variables which should be entered in the symbol file. For this purpose you can select a POU's entry (e.g. Global Variables) which automatically will mark all variables belonging to this POU, or you can select particular variables. For the selected set of entries then activate the desired options in the lower part of the dialog box by mouse-clicks on the corresponding option boxes. Activated options are displayed checked. The following options can be set:

Export variables of object: The variables of the selected object are exported into the symbol file.

The following options can take effect only if the Export variables of object option is activated:

Export data entries: Entries for access to the global variables are created for object's structures and arrays.

Export structure components: An individual entry is created for each variable component of object's structures.

Export array entries: An individual entry is created for each variable component of object's arrays.

Write Access: Object's variables may be changed by the OPC server.

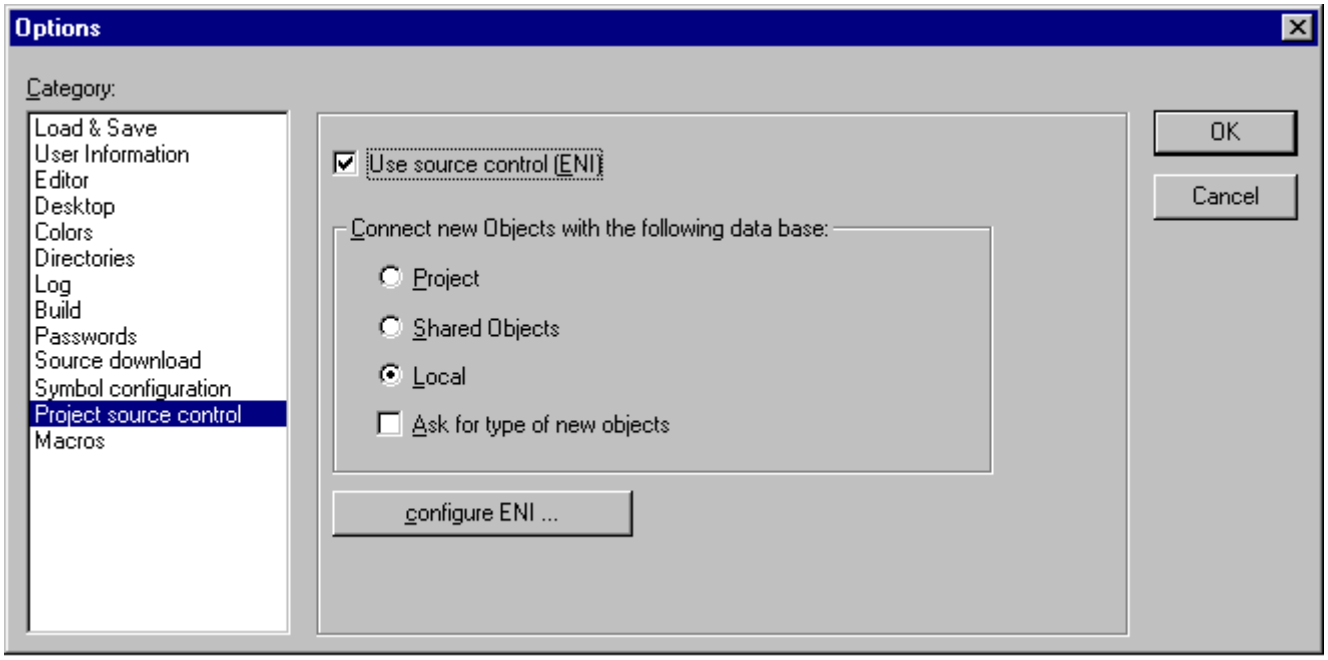
Once the option settings for the currently selected variables are complete, other POUs can be also be selected - without closing the dialog before - and given an option configuration. This can be carried out for any desired number of POU selections, one after the other.

When the dialog box is closed by selecting OK, all configurations carried out since the dialog box was opened are applied.

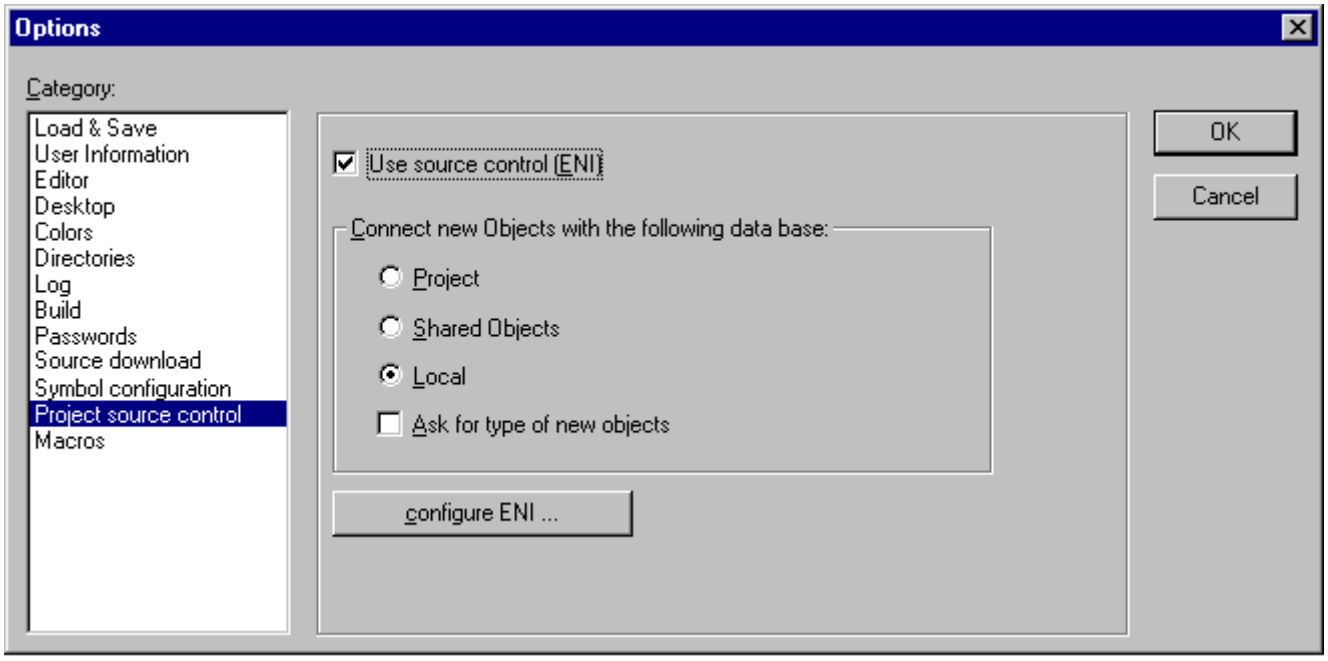
Note: Regard the possibility of using pragmas in the declaration of a variable, which define that the variable is taken to the symbol file with restricted access or that it is excluded from the symbol file.

The settings will be stored in CODESYS.

Options for 'Project source control'



This dialog is used to define whether the project should be managed in a project data base and to configure the ENI interface correspondingly.



Use source control (ENI): Activate this option, if you want to access a project data base via the ENI Server in order to administer all or a selection of POUs of the project in this data base. Preconditions: ENI Server and data base must be installed and you must be registered as an user in the database. See also the documentation for the ENI-Server resp. in chapter 'ENI'.

If the option is activated, then the data base functions (Check in, Get last version etc.) will be available for handling the project POUs. Then some of the data base functions will run automatically like defined in the options dialogs, and in the menu 'Project' 'Data Base Link' you will find the commands for calling the functions explicitly. Besides that a tab 'Data base-connection' will be added in the dialog Properties, where you can assign a POU to a particular data base catetory.

Connect new objects with the following data base:

Here you set a default: If a new object is inserted in the project ('Project' 'Object' 'Add'), then it will automatically get assigned to that object category which is defined here. This assignment will be displayed in the object properties dialog ('Project' 'Object' 'Properties') and can be modified there later. The possible assignments:

- Project: The POU will be stored in that data base folder which is defined in the dialog ENI configuration/Project objects in the field 'Project name'.
- Shared Objects: The POU will be stored in that data base folder which is defined in the dialog ENI configuration/Shared objects in the field 'Project name'.
- Local: The POU will not be managed in a ENI data base, but only will be stored locally in the project.

Besides 'Project objects' and 'Shared objects' there is a third data base category 'Compile files' for such objects which are not created until the project has been compiled. Therefore this category is not relevant for the current settings.

Ask for type of new objects: If this option is activated, then whenever a new object is added to the project, the dialog 'Object' 'Properties' will open, where you can choose to which of the three object categories mentioned above the POU should be assigned. By doing so the standard setting can be overwritten.

configure ENI: This button opens the first of three ENI configuration dialogs:

Each object of a project, which is determined to get managed in the ENI data base, can be assigned to one of the following data base categories: 'Project objects', 'Shared objects' or 'Compile files'. For each of these categories a separate dialog is available to define in which data base folder it should be stored and which presettings should be effective for certain data base functions:

- *Chapter 1.4.1.2.2.14 “Options for project objects and shared objects regarding the project data” on page 217*
- *Chapter 1.4.1.2.2.15 “Options for compile files regarding the project data base” on page 220*

Each object will be stored also locally (with project) in any case.

The dialog will open one after the other if you are doing a primary configuration. In this case a Wizard (Button Next) will guide you and the settings entered in the first dialog will be automatically copied to the other ones, so that you just have to modify them if you need different parameter values.

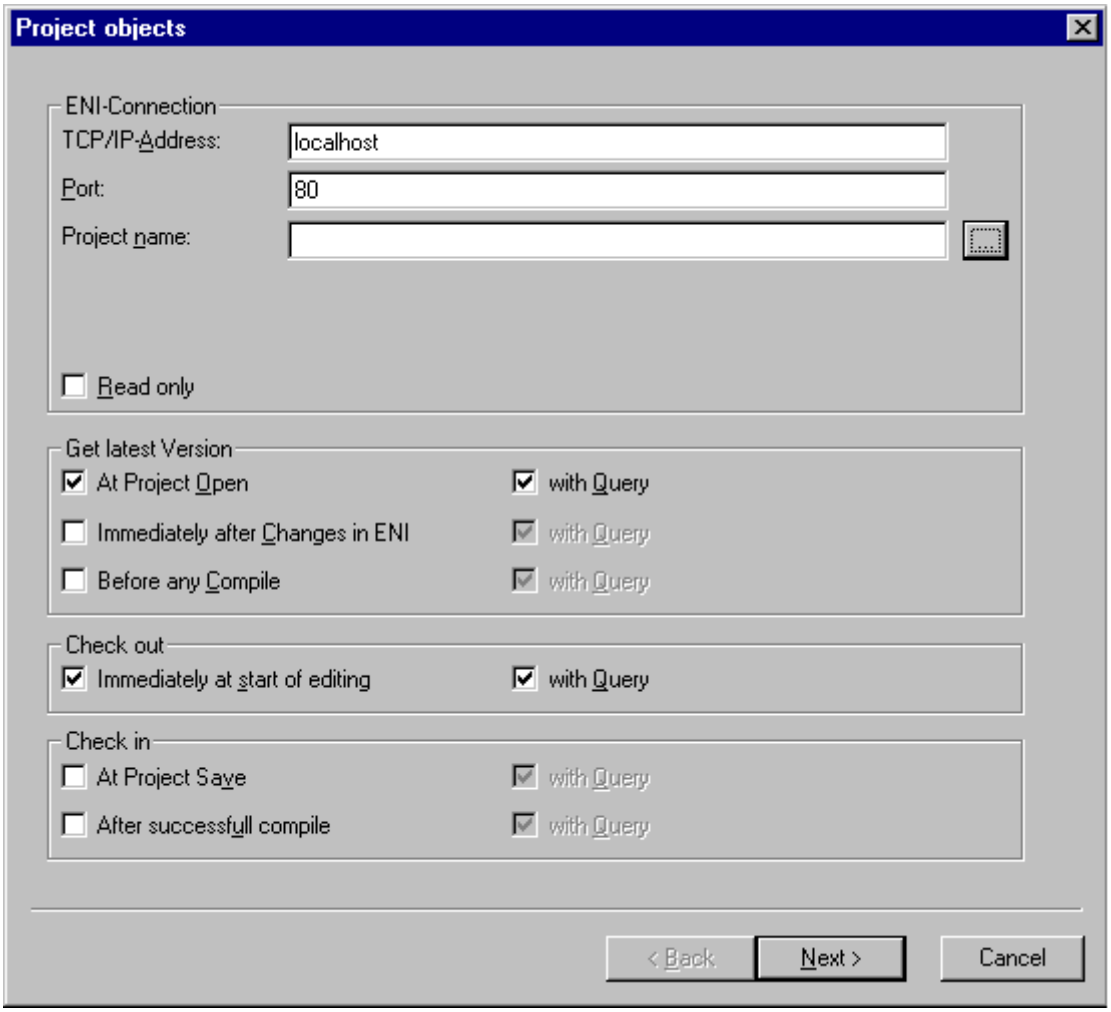
If you want to modify an existing configuration, then the three dialogs are combined in one window (three tabs).

If you have not yet logged in successfully to the data base before, then the Login dialog will be opened automatically.

The settings will be stored in the project.

Options for project objects and shared objects regarding the project data

These dialogs are part of the configuration of the project data base options ('Project' 'Options' 'Project source control'). Here you define the access parameters for the data base categories 'Project objects' und 'Shared objects'. Both dialogs contain the same items. (A third dialog is available for the configuration of the access to the data base category Compile files.



ENI-Connection

TCP/IP-Address:	Address of the computer where the ENI-Server is running
Port:	Default: 80; must be the same as set in the configuration parameters of the ENI Server
Project name:	Name of the data base folder where the objects of this category should be stored. Press button ... to open a folder tree of the already existing data base projects. If the desired folder already exists, you can select it in this tree and it will be entered in the 'Project name' edit field. If you had not logged in to the ENI Server until you try to open the folder tree by button ..., then you will first get the Login dialog where you must enter 'User name' and 'Password' as defined in your ENI user account to get access to the three data base categories.
Read only	If this option is activated, then only read access is possible to the above defined data base folder.

Get latest version

The data base function 'Get latest Version' (Menu 'Project' 'Data Base Link' copies the actual version of POUs from the above defined data base folder to the currently opened project, whereby the local version of objects will be overwritten. This will be done automatically for all objects, for which the version found in the data base differs from that in the project, as soon as one of the chosen timing conditions will meet. Choose the options by setting a check mark:

At Project Open	As soon as the project is opened in CODESYS
Immediately after Changes in ENI	As soon as a newer version of the POU is checked in to the data base (e.g. by another user); then the POU will be updated in the current project immediately and an appropriate message will pop up.
Before any Compile	Before any compile process in CODESYS

Check out

The data base function 'Check out' means that the POU will be marked as 'in the works' and will be locked for other users until it will be de-blocked again by a 'Check in' or 'Undo check out'.

If the option Immediately at start of editing is activated, then an object will be checked out automatically as soon as you start to edit it. If the object is currently already checked out by another user (indicated by a red cross before the object name in the object organizer of CODESYS), then a message will pop up.

Check in

The data base function 'Check in' means, that a new version of the object will be created in the data base. The older versions will be kept anyway.

You can activate one or both of the following options to define the time of automatic Checking in:

At Project Save	as soon as the project is saved
After successfull compile	as soon as the project has been compiled without errors

For each of the options 'Get last version', 'Check out' and 'Check in' additionally the option with Query can be activatd. In this case, before the corresponding action is carried out, a dialog opens where you still can decide to cancel the action or otherwise confirm it.

The items of the dialog 'Shared objects' are the same like in the dialog 'Project objects' described above. The settings apply to all objects which are assigned to the data base category 'Shared objects'.

If you do a primary configuration, the configuration dialogs will appear one after the other and you will be guided by a wizard (button Next). The settings made in the first dialog will automatically be inherited to the other ones. So those just have to be edited if modificiations are necessary.

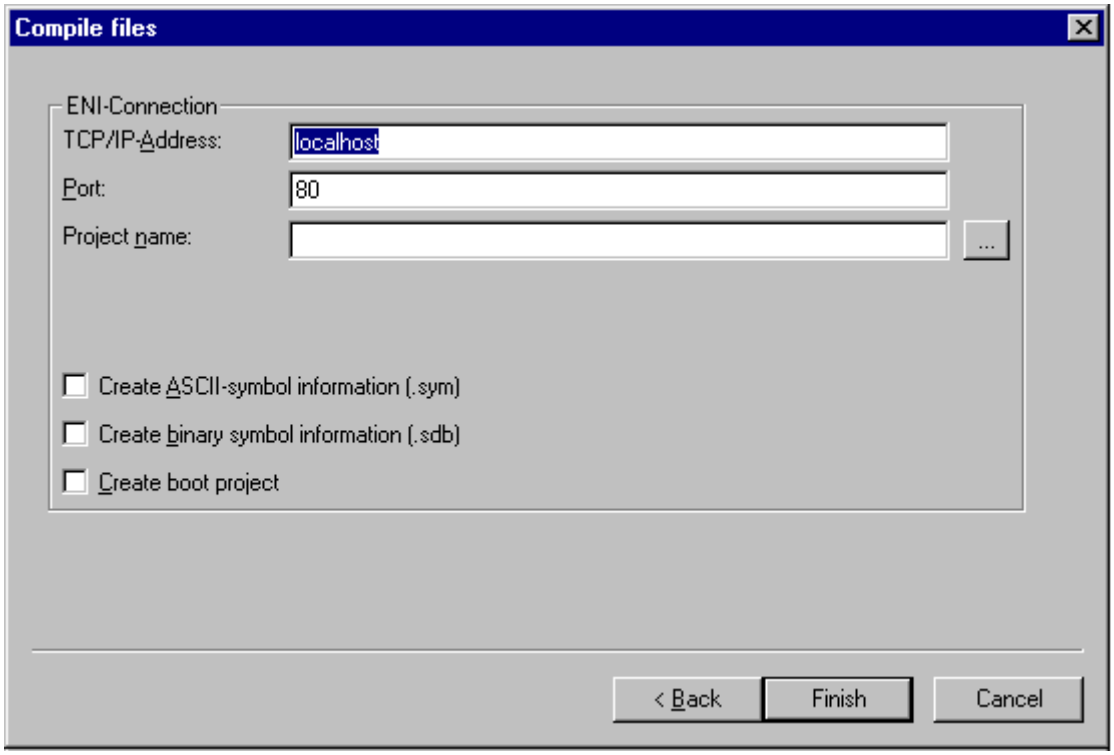
Cancel will close the dialog without saving the done modifications in the currently opened dialog. You return to the main dialog 'Options' 'Project source control'.

If an already existing configuration has been modified, then the new settings (for all three dialogs) can be saved by pressing OK. After that the dialog will be closed and you return to the main dialog 'Options' 'Project source control'.

The settings will be stored in the project.

Options for compile files regarding the project data base

This dialog is part of the option settings for the project data base ('Project' 'Options' 'Project source control'). Here you define how the objects of category 'Compile files' will be handled in the data base. (Besides that two further dialogs are available to define this for objects of category 'Project objects' and 'Shared objects'.)



For the input fields TCP/IP-Address, Port, Project namesee the description of dialog Project objects/Shared objects.

If you do a primary configuration, the configuration dialogs will appear one after the other, guided by a wizard (button Next). The settings made in the first dialog will automatically be inherited to the other ones. So those just have to be edited if modificiations are necessary.

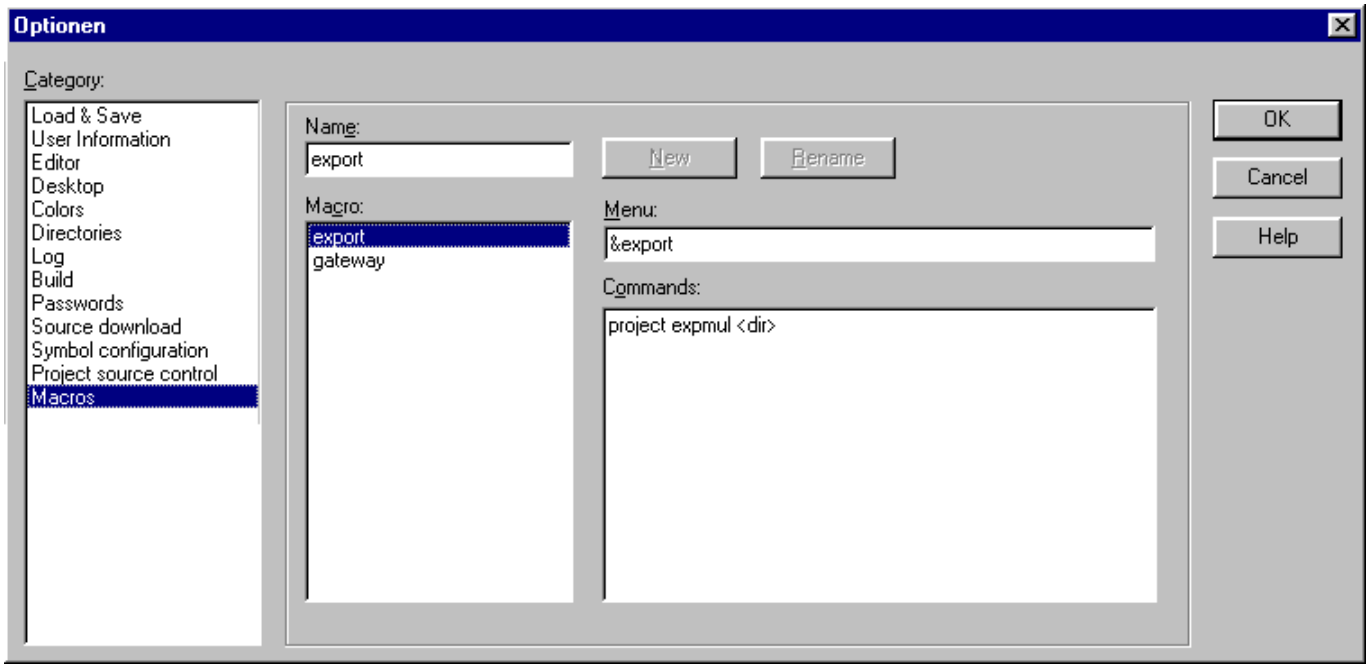
Cancel will close the dialog without saving the done modifications in the currently opened dialog (the settings made in the previous dialogs will be kept anyway). You return to the main dialog 'Options' 'Project source control'.

If an already existing configuration has been modified, then the new settings (for all three dialogs) can be saved by pressing OK. After that the dialog will be closed and you return to the main dialog 'Options' 'Project source control'.

The settings will be stored in the project.

Options for 'Macros'

If you choose this category in the Options dialog, the following dialog box opens:



In this dialog, macros can be defined using the commands of the batch mechanism, which can then be called in the 'Edit' 'Macros' menu.

Perform the following steps to define a new macro:

- In the input field Name, you enter a name for the macro to be created. After the New button is pressed, this name is transferred into the Macrolist field and marked as selected there. The macro list is represented in a tree structure. The locally defined macros are positioned one below the other. If macro libraries (see below) are integrated, then the library names will be listed and by a mouse-click on the plus- resp. minus-signs in front of those entries you can open or close a list of the library elements.
- The Menu field is used to define the menu entry with which the macro will appear in the 'Edit' 'Macros' menu. In order to be able to use a single letter as a short-cut, the letter must be preceded by the symbol '&'. Example: the name "Ma&cro 1" generates the menu entry "Macro 1". Example: the name "Ma&cro 1" will create a menu item "Macro 1".
- In the editor field Commands you define and/or edit the commands that are to constitute the newly created or selected macro. All the commands of the batch mechanism and all keywords which are valid for those are allowed. You can obtain a list by pressing the Help button. A new command line is started by pressing <Ctrl><Enter>. The context menu with the common text editor functions is obtained by pressing the right mouse button. Command components that belong together can be grouped using quotation marks.
- If you want to create further macros, perform steps 1-3 again, before you close the dialog by pressing the OK-button.

If you want to delete a macro, select it in the macro list and press button <Del>.

If you want to rename a macro, select it in the macro list, insert a new name in the edit field 'Name' and then press button Rename.

To edit an existing macro, select it in the macro list and edit the fields 'Menu' and/or 'Commands'. The modifications will be saved when pressing the OK-button.

As soon as the dialog is closed by pressing the OK-button the actual description of all macros will be saved in the project.

The macro menu entries in the 'Edit' 'Macros' menu are now displayed in the order in which they were defined. The macros are not checked until a menu selection is made.

Macro libraries: Macros can be saved in external macro libraries. These libraries can be included in other projects.

- Creating a macro library containing the macros of the currently opened project:
  Press button Create. You get the dialog Merge project, where all available macros are listed. Select the desired entries and confirm with OK. The selection dialog will close and dialog Save Macrolibrary will open. Insert here a name and path for the new library and press button Save. The library will be created named as <library name>.mac and the dialog will be closed.
- Including a macro library <library name>.mac in the currently opened project:
  Press button Include. The dialog Open Macrolibrary will open, which shows files with extension *.mac. Select the desired library and press button Open. The dialog will be closed and the library will be added to the tree of the Macrolist.

Hint: The macros of a project also can be exported ↳ *Chapter 1.4.1.2.3.22 “'Project' 'Export'” on page 239*.

The settings will be stored in the project.

1.4.1.2.3 Managing projects

'File' 'New'

Symbol: 

With this command you create an empty project with the name "Untitled". This name must be changed when saving.

'File' 'New from template'

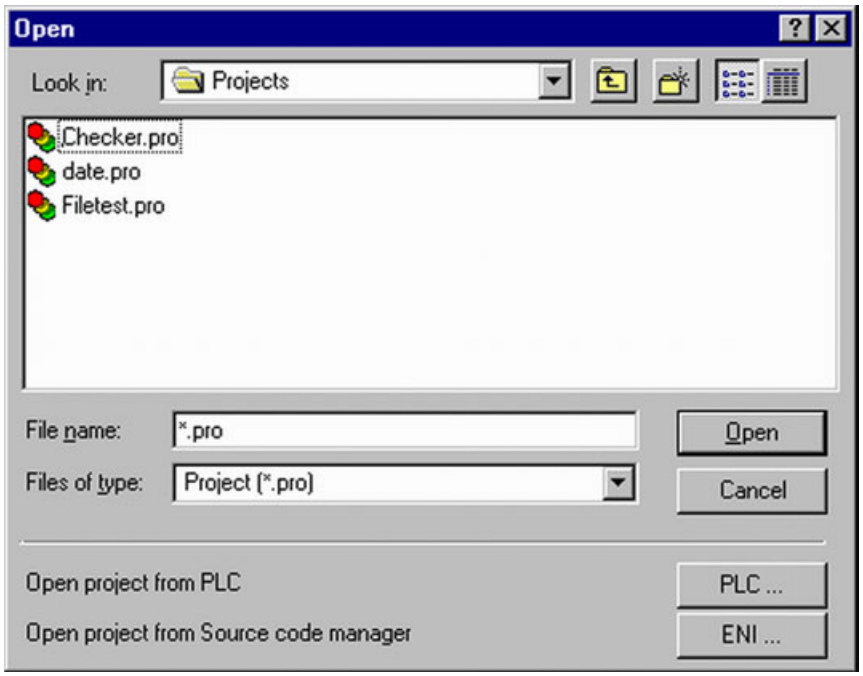
Use this command to open any desired project as a "template" project. The dialog for opening a project file will be available and the selected project will be opened with project name "Unknown" ↳ *Chapter 1.4.1.2.3.3 “'File' 'Open'” on page 222*.

'File' 'Open'

Symbol: 

With this command you open an already existing project. If a project has already been opened and changed, then you are asked whether this project should be saved or not.

The dialog box for opening a file appears, and a project file with the extension "*.pro" or a library file with the extension "*.lib" must be chosen. This file must already exist ↳ *Chapter 1.4.1.4.3.1 “Overview” on page 371*. It is not possible to create a project with the command "Open".



Open a project from the PLC

To upload a project file from the PLC, press PLC at Open project from PLC. You will obtain, as next, the dialog Communication parameters (see menu 'Online' 'Communication parameters') for setting the transmission parameters when no connection exists yet to the PLC. Once an on-line connection has been created, the system checks whether the same named project files already exist in the directory on your computer hard disc. When this is the case you receive the dialogue Load the project from the controller where you can decide whether the local files should be replaced by those being used by the controller. (This sequence is the reverse of the sequence of 'Online' 'Load source code', with which the project source file is stored in the controller. Do not confuse with 'Create Boot project'!)

Please note, that you in any case have to give a new name to a project, when you load it from the PLC to your local directory, otherwise it is unnamed. If supported by the target system, a 'Title' entered in the project info will be pre-defined as new project file name ↬ Chapter 1.4.1.2.3.34 "'Project' 'Project info'" on page 246. In this case at loading the project from the PLC the dialog for saving a file will open, where the new file name automatically is entered and can be confirmed or modified.

If there has not yet been loaded a project to the PLC, you get an error message.

↬ *Chapter 1.4.1.2.2.11 "'Source download'" on page 212*'

Open a project from Source code manager (ENI data base)

To open a project which is stored in a ENI project data base, activate option Open project from Source code manager can be used . It is a precondition that you have access to an ENI Server which serves the data base. Press button ENI..., to get a dialog where you can connect to the server concerning the data base category 'Project objects'.

Insert the appropriate access data (TCP/IP-Address, Port, Username, Password, Read only) and the data base folder (Project name) from which the objects should be get and confirm with Next. The dialog will be closed and another one will open where you have to insert the access data for the data base category 'Shared objects'. If you press button Finish the dialog will be closed and the objects of the defined folders will automatically be retrieved and displayed in the Object manager. If you want to continue to keep the project objects under data base control, then open the Project options to set the desired parameters.

The access data are stored in the codesys.ini file, username and password however only if the project option 'Save ENI credentials' is activated.

Most recently opened files

The most recently opened files are listed in the Files menu below the command 'File' 'Exit'. If you choose one of them, then this project is opened.

If passwords or User groups have been defined for the project, then a dialog box appears for entering the password.

'File' 'Close'

With this command you close the currently-open project. If the project has been changed, then you are asked if these changes are to be saved or not. If the project to be saved carries the name "Untitled", then a name must be given to it *Chapter 1.4.1.2.3.6 “'File' 'Save as'” on page 224*.

'File' 'Save'

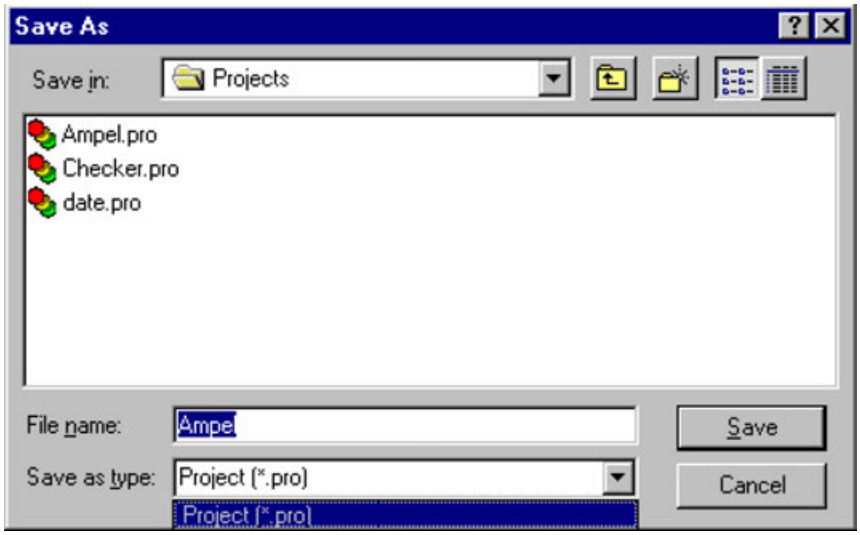
Symbol: Shortcut: <Ctrl>+<S>

With this command you save any changes in the project. If the project to be saved is called "Untitled", then you must give it a name *Chapter 1.4.1.2.3.6 “'File' 'Save as'” on page 224*.

'File' 'Save as'

With this command the current project can be saved in another file or as a library. This does not change the original project file.

After the command has been chosen the Save dialog box appears. Choose either an existing File name or enter a new file name and choose the desired file type.



If the project is to be saved under a new name, then choose the file type Project (*.pro).

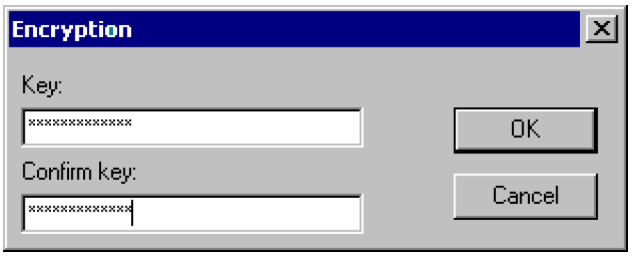
If you choose the file type Project Version 1.5 (*.pro), 2.0 (*.pro), 2.1 (*.pro) or 2.2 (*.pro), then the current project is saved as if it were created with the version 1.5, 2.0, 2.1 or 2.2. Specific data of the version 2.3 can thereby be lost! However, the project can be executed with the version 1.5, 2.0, 2.1 or 2.2.

You can also save the current project as a library in order to use it in other projects. Choose the file type Internal library (*.lib) if you have programmed your POUs in CODESYS.

Choose the file type External library (*.lib) if you want to implement and integrate POUs in other languages (e.g. C). This means that another file is also saved which receives the file name of the library, but with the extension "*.h". This file is constructed as a C header file with the declarations of all POUs, data types, and global variables. If external libraries are used, in the simulation mode the implementation, written for the POUs in CODESYS, will be executed. Working with he real hardware the implementation written in C will be executed.

Encryption of a project

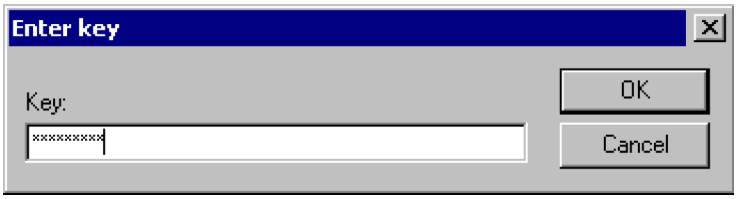
In order to save the project as an encrypted project or library, choose option Encrypted Project (*.pro) resp. Encrypted internal library (*.lib) or Encrypted external library (*.lib). In this case you get the 'Encryption' dialog , where you can define and confirm a key. The project later cannot be opened esp. a library cannot be used without this key:



The encryption extends the protection of a project, which up to now was only possible via the assignment of passwords for access and write protection ↳ *Chapter 1.4.1.2.2.10 “Passwords” on page 211*. These possibilities will exist further on, but note that they e.g. cannot avoid that a library is included in a project without the need of entering a library password (key).

A key once defined will be saved with any further savings of the project. To modify that key, you have to use again the 'Save as' dialog.

If an encrypted project should be opened resp. if an encrypted library sShould be used in a project, the dialog asking for the key will appear.



Licensing a library:

If you want save the project as a licensed library, you can add the appropriate licensing information in the dialog 'Edit Licensing Information'. Open the dialog by pressing the button Edit license info.

After having done all settings, press OK. The current project will be saved in the indicated file. If the new file name already exists, then you are asked if you want to overwrite this file.

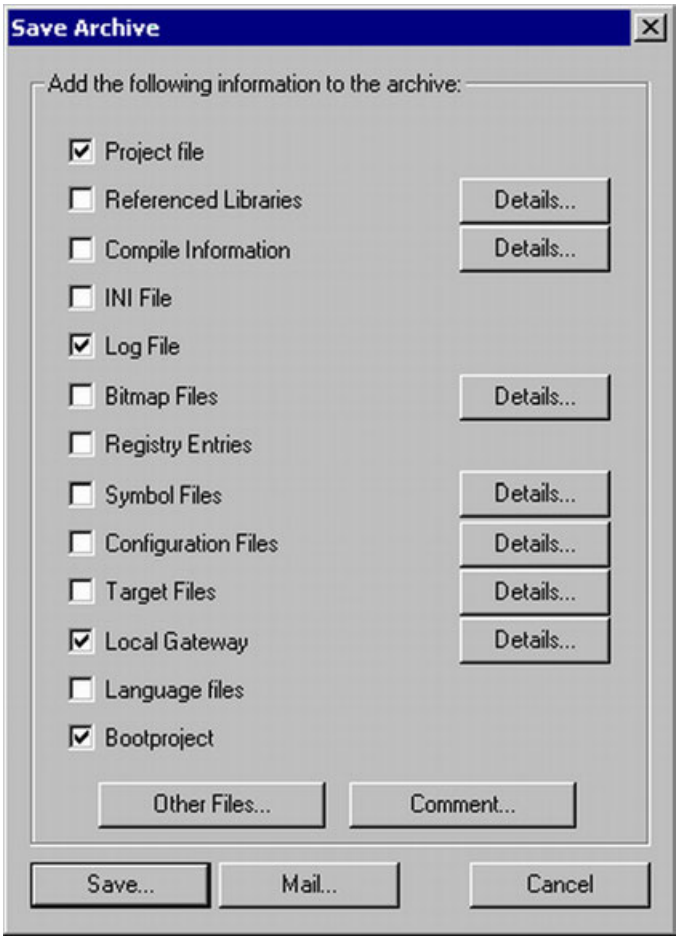
When saving as a library, the entire project is compiled. If an error occurs thereby, then you are told that a correct project is necessary in order to create a library. The project is then not saved as a library.

'File' 'Save/Mail archive'

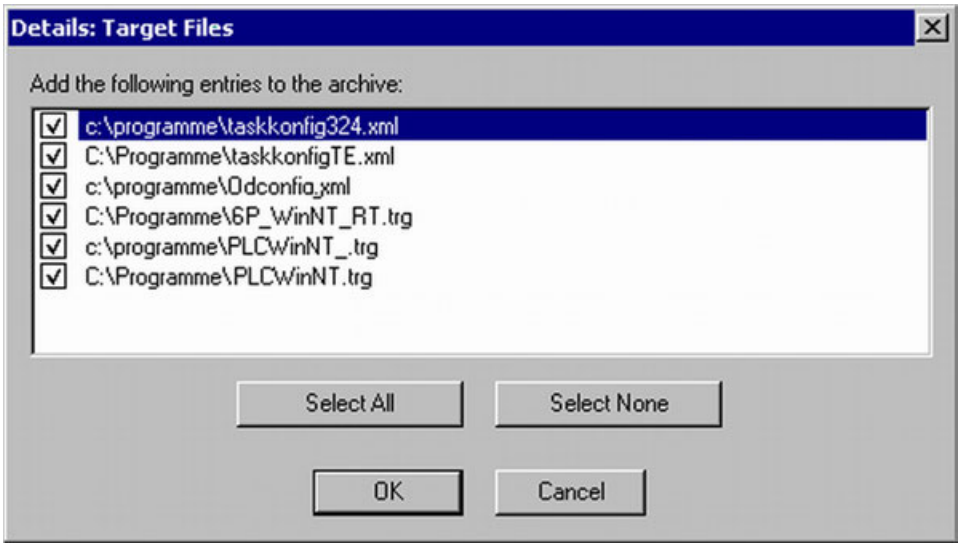
This command is used to set up and create a project archive file. All files which are referenced by and used with a CODESYS project can be packed in a compressed zip file. The zip file can be stored or directly can be sent in an email. This is useful if you want to give forward a set of all project relevant files.

The archive function is not practical for restoring a project environment. It is designated for an easy packing of all files belonging to a project. When unpacking an archive the paths of the particular files must be adapted to the actual environment!

When the command is executed, the dialog box 'Save Archive' opens:



Here you can define which file categories should be added to the archive zip file: Select or deselect a category by activating/deactivating the corresponding checkbox. Do this by a single mouseclick in the checkbox or by a doubleclick on the category name. If a category is marked with ☑, all files of this category will be added to the zip file, if it is marked with ☐, none of the files will be added. To select single files of a category press the corresponding button Details. The dialog 'Details' will open with a list of available files:



The dialog shows a list of all files which are available for the category: Automatically alle files are selected, an exception is category 'Target Files' where only the files are selected which are relevant for the currently set target system. For modifying the selection activate resp. deactivate the desired files. With the button 'Select All' all files of the list are selected, with 'Select None' none of them. A single file can be selected/deselected by a mouseclick in the checkbox, also by a doubleclick on the list entry or by pressing the spacebar when the list entry is marked.

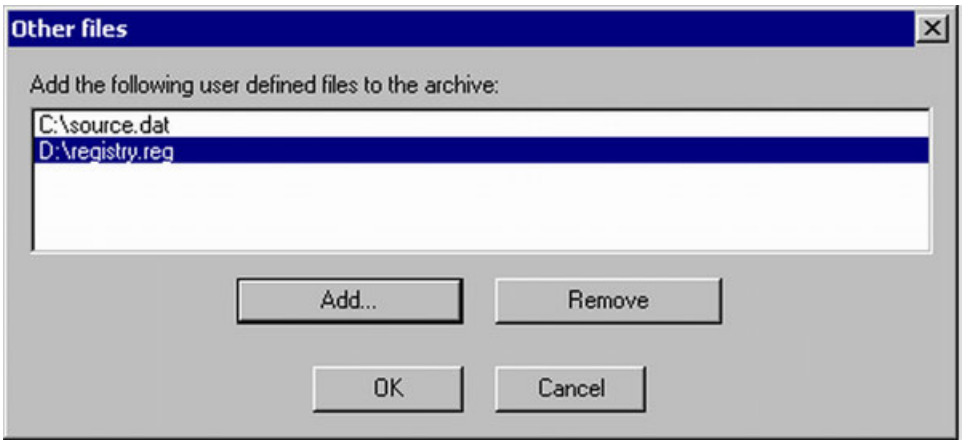
Close the Details dialog with 'Save' to store the new settings.

In the main dialog the checkbox of categories, for which not all files are selected, will appear with a grey background color ☑.

The following file categories are available, the right column of the table shows which files can be added to the zip file:

Category	Files
Project File	projectname.pro (the project file)
Referenced Libraries	*.lib, *.obj, *.hex (libraries and if available the corresponding object and hex-files)
Compile Information	*.ci (compile information) *.ri (download/reference information) <temp>.* (temporary compile and download files) also for simulation
INI File	Codesys.ini
Log File	*.log (project log file)
Bitmap Files	*.bmp (bitmaps for project POUs and visualizations)
Registry Entries	Registry.reg (Entries for Automation Alliance, Gateway and SPS; the following subtrees will be packed: HKEY_LOCAL_MACHINE\SOFTWARE\3S-Smart Software Solutions HKEY_LOCAL_MACHINE\SOFTWARE\AutomationAlliance"
Symbol Files	*.sdb, *.sym (symbolic information)
Configuration files	files used for PLC configuration (configuration files, device files, icons etc.): e.g. *.cfg, *.con, *.eds, *.dib, *.ico ....
Target Files	*.trg (target files in binary format for all installed targets) *.txt (target files for the installed targets in text format, if available)
Local Gateway	Gateway.exe, GatewayDDE.exe, GClient.dll, GDrvBase.dll, GDrvStd.dll, GHandle.dll, GSymbol.dll, GUtil.dll, further DLLs in the gateway directory if available
Language Files	language files used for visualizations (*.vis, *.xml)
Boot project	Boot project files <project name>.prg, <project name>.chk resp. the target specific boot project files.

To add any other files to the zip, press the button 'Other Files'. The dialog 'Other files' will open where you can set up a list of desired files.



Press the button Add to open the standard dialog for opening a file, where you can browse for a file. Choose one and confirm with Open. The file will be added to the list in the 'Other files' dialog. Repeat this for each file you want to add. To delete entries from the list, press the button Remove. When the list of selected files is ok, close the dialog with OK.

To add a readme file to the archive zip, press the button 'Comment'. A text editor will open, where you can enter any text. If you close the dialog with OK, during creation of the zip file a readme.txt file will be added. Additionally to the entered comments it will contain information about the build date and version of CODESYS.

If all desired selections have been made, in the main dialog press:

- Save... to create and save the archive zip file: The standard dialog for saving a file will open and you can enter the path, where the zip should be stored. The zip file per default is named <projectname>.zip. Confirm with Save to start building it. During creation the current progress status is displayed and the subsequent steps are listed in the message window. There also a message will be displayed if any file could not not been found.
- Mail... to create a temporary archive zip and to automatically generate an empty email which contains the zip as an attachment. This feature only works if the MAPI (Messaging Application Programming Interface) has been installed correctly on the system, otherwise an error message is generated. During setup of the email the progressing status is displayed and the steps of the action are listed in the message window. The temporary zip file will be removed automatically after the action has been finished.
- Cancel to cancel the action; no zip file will be generated.

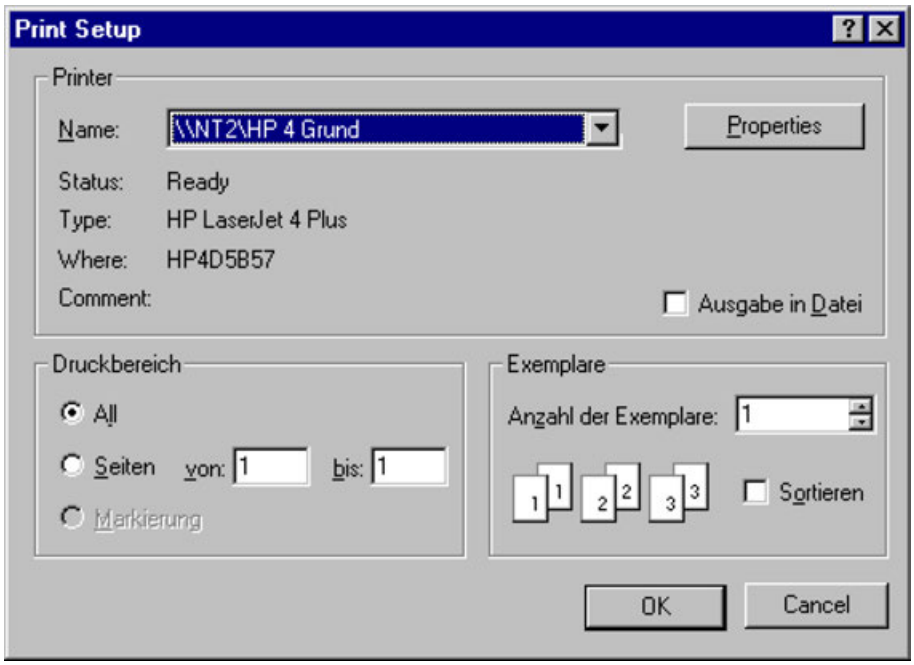
After unpacking the archive zip on a different system it might be necessary to adapt the file paths!

'File' 'Print'

Shortcut: <Ctrl>+<P>

With this command the content of the active window is printed.

After the command has been chosen, then the Print dialog box appears. Choose the desired option or configure the printer and then click OK. The active window is printed. Color output is available from all editors.



You can determine the number of the copies and print the version to a file.

With the button 'Properties' you open the dialog box to set up the printer.

You can determine the layout of your printout with the command 'File' 'Printer setup' *Chapter 1.4.1.2.3.9 “'File' 'Printer setup'” on page 229*.

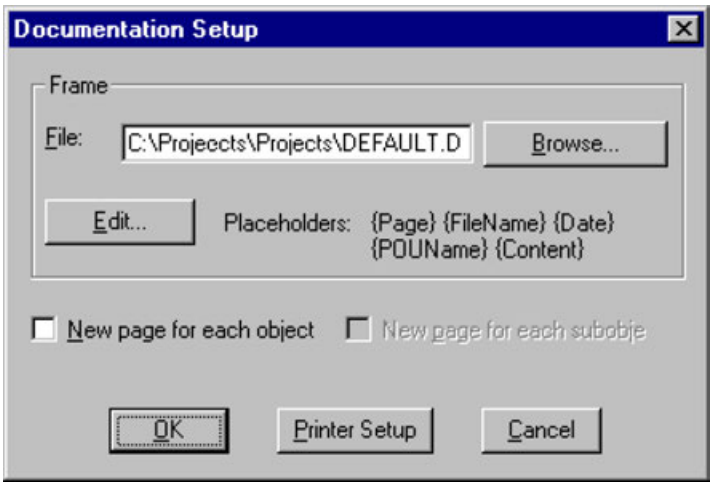
During printing the dialog box shows you the number of pages already printed. When you close this dialog box, then the printing stops after the next page.

In order to document your entire project, use the command 'Project' 'Document' *Chapter 1.4.1.2.3.21 “'Project' 'Document'” on page 238*.

If you want to create a document frame for your project, in which you can store comments regarding all the variables used in the project, then open a global variables list and use the command 'Extras' 'Make docuframe file'.

'File' 'Printer setup'

With this command you can determine the layout of the printed pages. The following dialog box is now opened:

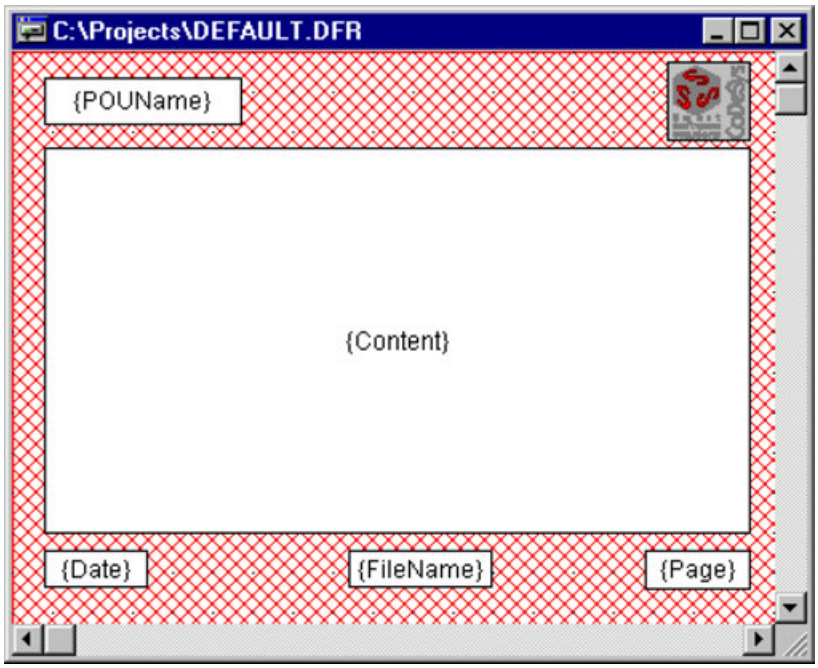


In the field File you can enter the name of the file with the extension ".dfr" in which the page layout should be saved. The default destination for the settings is the file DEFAULT.DFR.

If you would like to change an existing layout, then browse through the directory tree to find the desired file with the button 'Browse'.

You can also choose whether to begin a new page for each object and for each subobject. Use the 'Printer Setup' button to open the printer configuration.

If you click on the 'Edit' button, then the frame for setting up the page layout appears. Here you can determine the page numbers, date, file name and POU name, and also place graphics on the page and the text area in which the documentation should be printed.



With the menu item 'Insert' 'Placeholder' and subsequent selection among the five placeholders (Page, POU name, File name, Date, and Content), insert into the layout a so-called placeholder by dragging a rectangle the layout while pressing the left mouse button. In the printout they are replaced as follows:

Command	Placeholder	Effect
Page	{Page}	Here the current page number appears in the printout.
POU name	{POU Name}	Here the current name of the POU appears.
File name	{File Name}	Here the name of the project appears.
Date	{Date}	Here the current date appears.
Contents	{Contents}	Here the contents of the POU appear.

In addition, with 'Insert' 'Bitmap' you can insert a bitmap graphic (e.g. a company logo) in the page. After selecting the graphic, a rectangle should also be drawn here on the layout using the mouse. Other visualization elements can be inserted ↳ *Chapter 1.4.3 “Visualization” on page 636*.

If the template was changed, then you are asked when the window is closed if these changes should be saved or not.

> *In order to be aware of the page format which will be valid for printouts, define the layout as described above and additionally activate option 'Show print area margins' in the project options, category Desktop.*

'File' 'Exit'

Shortcut: <Alt>+<F4>

With this command you exit from CODESYS.

If a project is opened, then it is closed *Chapter 1.4.1.2.3.5 "'File' 'Save'" on page 224*.

'Project' 'Build'

Shortcut: <F11>

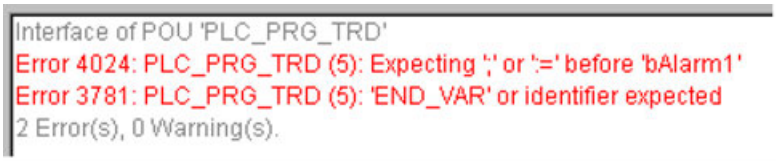
The project is compiled using 'Project' 'Build'. The compilation process is basically incremental, that is only changed POUs are recompiled. The necessary information about the last compilation is stored in a *.ci-file when the project is saved. A non-incremental compilation can also be obtained if the command 'Project' 'Clear all' is first executed.

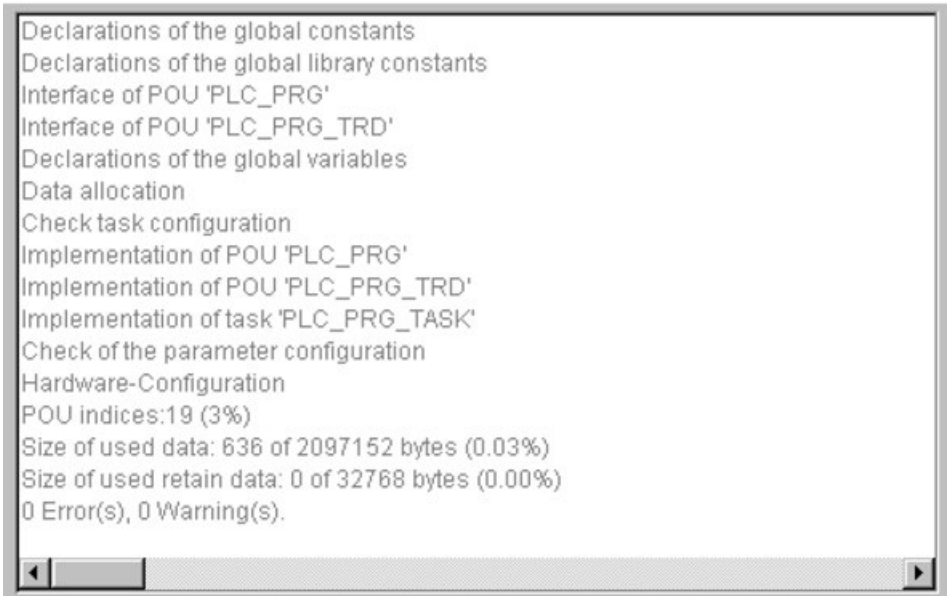
For target systems that support Online Change, all POUs that will be loaded into the controller on the next download are marked with a blue arrow in the Object Organizer after compilation.

The compilation process that is carried out with 'Project' 'Build' occurs automatically if the controller is logged-in via 'Online' 'Login' *Chapter 1.4.1.2.6.2 "'Online' 'Login'" on page 279*.

See online login for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login on the target systemFig. .

During compilation a message window is opened which shows the progress of the compilation process, any errors and warnings which may occur during compilation as well as information on the used POU indices and memory space (number and percentage). Errors and warnings are marked with numbers. Using F1 you get more information about the currently selected error.





See the listing of all available error messages and warnings ↳ *Chapter 1.4.1.10.2.1 “1100” on page 478*.

If the option 'Save before compilation' is selected in the options dialog of the Load & Save category, the project is stored before compilation.

An object, or several objects, selected in the Object Organizer can be excluded from compilation by command 'Exclude from build' which is available in the context menu, or via an appropriate configuration ('Exclude objects') in the Options for Build.

Cross references are created during compilation and are stored with the compilation information. In order to be able to use the command 'Show Call Tree', and to get up to date results with the commands 'Show Cross Reference' and 'Unused variables', 'Overlapping memory areas', '', 'Multiple Writes to output', rebuild the project after any change ↳ *Chapter 1.4.1.2.4.16 “'Project' 'Show call tree'” on page 264* ↳ *Chapter 1.4.1.2.4.17 “'Project' 'Show cross reference'” on page 264* ↳ *Chapter 1.4.1.2.3.41 “Concurrent access” on page 249* ↳ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248.*

'Project' 'Rebuild all'

With 'Project' 'Rebuild all', unlike the incremental compilation, the project is completely recompiled ↳ *Chapter 1.4.1.2.3.11 “'Project' 'Build'” on page 231*. Download information is not discarded, however, as is the case with the command 'Project' 'Clean all' ↳ *Chapter 1.4.1.2.3.13 “'Project' 'Clean all'” on page 232* ↳ *Chapter 1.4.1.2.3.14 “'Project' 'Load download information'” on page 232*. Note the possibility to exclude objects from compilation ↳ *Chapter 1.4.1.2.2.9 “Options for build” on page 209*.

See 'Online' 'Login' for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login on the target systemFig. .

'Project' 'Clean all'

With this command, all the information from the last download and from the last compilation is deleted.

After the command is selected a dialog box appears, reporting that Login without new download is no longer possible ↳ *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*. At this point the command can either be cancelled or confirmed.

Note: After having done a 'Clean all', an online change on the PLC project is only possible if the *.ri file with the project information from the last download was first renamed or saved outside the project directory and can now be loaded explicitly prior to logging-in ↳ *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279* ↳ *Chapter 1.4.1.2.3.14 “'Project' 'Load download information'” on page 232*.

See Online Login for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login on the target systemFig. .

'Project' 'Load download information'

With this command the Download-Information belonging to the project can get reloaded. After choosing the command the standard dialogue 'File Open' opens.

The Download-Information is saved automatically at each download and, dependent on the target system, potentially also at each offline creation of a boot project to a file, which is named <project name><target identifier>.ri and which is put to the project directory *Chapter 1.4.1.2.6.5 “'Online' 'Download'” on page 283*. This file gets reloaded each time the project is reopened and at login it is used to check the code of which POUs has been changed. Only these POUs will then be loaded to the PLC during online change procedure. Thus the *.ri file is a precondition for an online change *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*.

*Using command 'Project' 'Clean all' the *.ri file belonging to the current project automatically gets deleted from the projects directory, so that no online change will be possible until the *.ri file can be reloaded from another directory or from a renamed *.ri file Chapter 1.4.1.2.3.13 “'Project' 'Clean all'” on page 232.*

See online login for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login on the target systemFig. .

'Project' 'Translate into another language'

This menu item is used for translating the current project file into another language. This is carried out by reading in a translation file that was generated from the project and externally enhanced in the desired national language with the help of a text editor. The project can be just displayed or really get translated into one of the generated language versions.

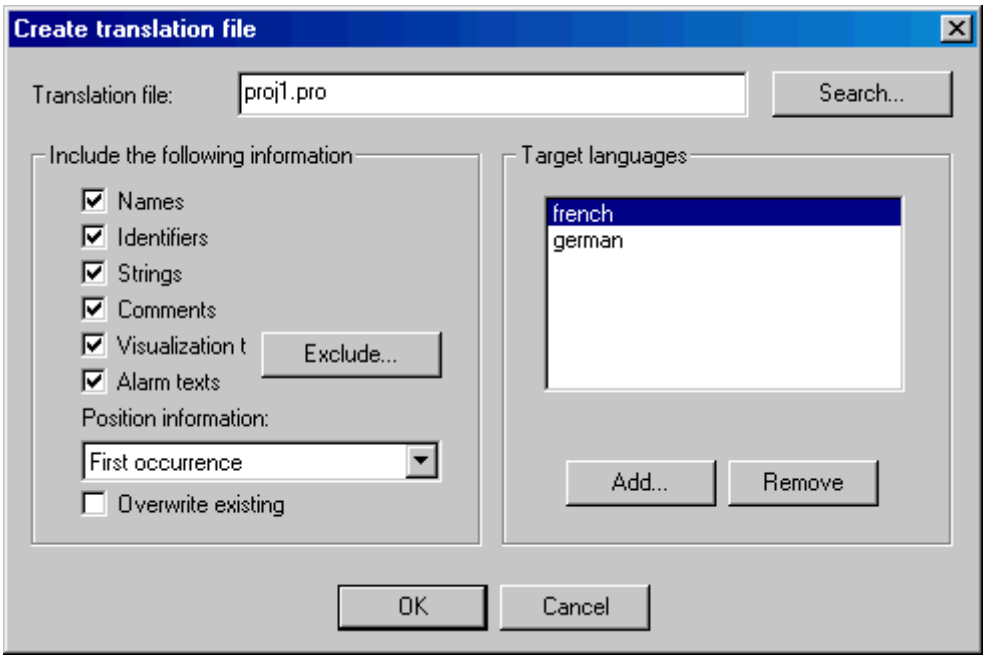
The following menu sub-items are present:

- Create translation file *Chapter 1.4.1.2.3.16 “Create translation file” on page 233*
- Translate project *Chapter 1.4.1.2.3.18 “Translate project (into another language)” on page 237*
- Show project translated *Chapter 1.4.1.2.3.19 “Show project translated” on page 238*
- Toggle translation *Chapter 1.4.1.2.3.20 “'Toggle translation'” on page 238*
- See also: 'Editing of the translation file' *Chapter 1.4.1.2.3.17 “Editing of the translation file” on page 236*

Create translation file

This command in the 'Project' 'Translate into another language' menu leads to the 'Create translation file' dialog.

Dialog for creating a translation file



In the Translation file field, enter a path that shows where the file is to be stored. The default file extension is *.tlt; this is a text file. You also can use the extension *.txt, which is recommended, if you want to work on the file in EXCEL or WORD, because in this case the data are organized in table format.

If there already exists a translation file which you want to process, give the path of this file or use the Search button to reach the standard Windows file selection dialog.

The following information from the project can optionally be passed to the translation file that is being modified or created, so that they will be available for translation: Names (names, e.g. the title 'POUs' in Object Organizer), Identifiers, Strings, Comments, Visualisation texts, Alarm texts. In addition, Position information for these project elements can be transferred.

If the corresponding options are checked, the information from the current project will be exported as language symbols into a newly created translation file or added to an already existing one. If the respective option is not selected, information belonging to the pertinent category, regardless of which project it came from, will be deleted from the translation file.

The option "Visualisation texts" only concerns 'Text' and 'Tooltip-Text' of a visualization element. Regard the following items when using a translation file for visualization texts:

- A *.tlt- or *.txt translation file only can be used with CODESYS or CODESYS HMI, not however with target visualization or web visualization. It might be better to use a special visualization language file *.vis.
- Switching to another language is only possible in online mode. This means that the visualization texts will not be translated by command 'Translate into another language' ↳ *Chapter 1.4.1.2.3.15 “'Project' 'Translate into another language'” on page 233*. A language change can only occur in Online mode if the corresponding language is entered in the 'Extras' 'Settings' dialog.
- If a *.tlt- or *.txt-file should be used for visualization texts ('Text' and 'Text for Tooltip'), the texts must be bracketed by two "#" symbols in the configuration dialog of the visualization element (e.g. #text#) in order to be transferred to the translation file.
- Language switching in visualizations ↳ *Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*.

Position information: This describes with the specifications file path, POU and line the position of the language symbol made available for translation. Three options are available for selection:

'None':	No position information is generated.
'First appearance':	The position on which the element first appears is added to the translation file.
'All':	All positions on which the corresponding element appears are specified.

If a translation file created earlier is to be edited which already contains more position information than that currently selected, it will be correspondingly truncated or deleted, regardless of which project it was generated from.

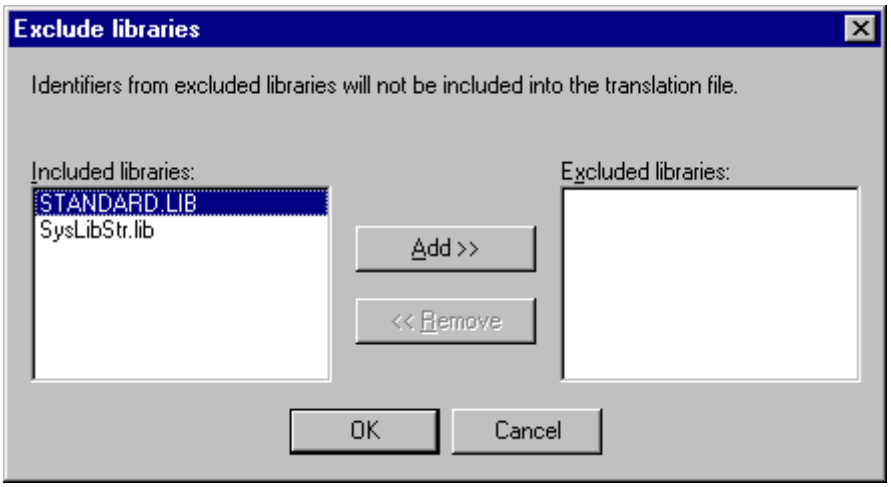
A maximum of 64 position specifications will be generated per element (language symbol), even if the user has selected "All" under "Position Information" in the 'Create Translation File' dialog.

Overwrite existing: Existing position information in the translation file, that is currently being processed, will be overwritten, regardless of which project generated it.

Target languages: This list contains identifiers for all languages which are contained in the translation file, as well as those to be added upon completion of the 'Create translation file' dialog.

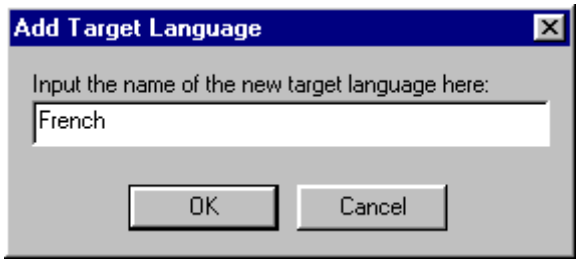
The Exclude button opens the 'Exclude libraries' dialog. Here, libraries included to the project can be selected, whose identifier information is not to be transferred to the translation file. To accomplish this, the corresponding entry in the table Included libraries on the left is selected with the mouse and placed in the Excluded libraries table to the right using the Add button. Likewise, entries already placed there can be removed using the Remove button. OK confirms the setting and closes the dialog.

Dialog for excluding library information for the translation file:



The Add button opens the 'Add Target Language' dialog:

Dialog for adding a target language (Project, Translate into Another Language):



A language identifier must be entered into the editor field; it may not have a space or an umlaut character (ä, ö, ü) at either the beginning or the end.

OK closes the 'Add Target Language' dialog and the new target language appears in the target language list.

The Remove button removes a selected entry from the list.

You may also confirm the "Create translation file" dialog via OK, in order to generate a translation file.

If a translation file of the same name already exists you will get the following confirmation message to be answered Yes or No:

" The specified translation file already exists. It will now be altered and a backup copy of the existing file will be created. Do you want to continue?"

No returns you without action to the 'Create translation file' dialog. If Yes is selected, a copy of the existing translation file with the filename "Backup_of_<translation file>.xlt" will be created in the same directory and the corresponding translation file will be modified in accordance with the options that have been entered.

The following takes place when a translation file is generated:

- For each new target language, a placeholder ("##TODO") is generated for each language symbol to be displayed *Chapter 1.4.1.2.3.17 “Editing of the translation file” on page 236*.
- If an existing translation file is processed, file entries of languages that appear in the translation file, but not in the target language list, are deleted, regardless of the project from which they were generated.

Editing of the translation file

The translation file must be opened and saved as a text file. The signs ## mark keywords. The ##TODO-placeholders in the file can be replaced by the valid translation. For each language symbol a paragraph is generated which starts and ends with a type identifier. For example ##NAME_ITEM and ##END_NAME_ITEM include a section for the name of an object as used in the object organizer. (COMMENT_ITEM marks sections for comments, IDENTIFIER_ITEM those for identifiers, STRING_ITEM those for strings and VISUALTEXT_ITEM those for visualization texts).

See in the following an example of a translation file paragraph which handles the name of one of the POUs of the project. ST_Visu. The target languages shall be English(USA) and French. In the example the position information of the project element which should be translated has been added:

before translation:

Example

```
##NAME_ITEM
[D:\codesys\projects\Bspdt_22.pro::ST_Visualisierung::0]
ST_Visualisierung
##English :: ##TODO
##French :: ##TODO
##END_NAME_ITEM
```

after translation:

The ##TODOs have been replaced by the English resp. French word for 'Visualisierung':

Example

```
##NAME_ITEM
[D:\codesys\projects\Bspdt_22.pro::ST_Visualisierung::0]
ST_Visualisierung
##English :: ST_Visualization
##French :: ST_Visu
##END_NAME_ITEM
```

Please check that the translated Identifier and Names remain valid concerning the standard and that strings and comments are in correct brackets. Example: For a comment (##COMMENT_ITEM) which is represented with "(* Kommentar 1)" in the translation file, the "##TODO" behind "##English" must be replaced by a "(* comment 1 *)". For a string (##STRING_ITEM) represented with "zeichenfolge1" the "##TODO" must be replaced by "string1".

The following parts of a translation file should not be modified without detailed knowledge: Language block, Flag block, Position information, Original texts.

Translate project (into another language)

This command in the 'Project' 'Translate into Another Language' menu opens the 'Translate Project into Another Language' dialog.

The current project can be translated into another language if an appropriate translation file is used.

If you want to save the version of the project in the language in which it was originally created, save a copy of the project prior to translation under a different name. The translation process cannot be undone. Consider in this context the possibility just to display the project in another language (in this display version then however not editable).

In the field Translation file, provide the path to the translation file to be used. By pressing Search you may access the standard Windows file selection dialog.

The field Target language contains a list of the language identifiers entered in the translation file, from which you can select the desired target language.

OK starts the translation of the current project into the chosen target language with the help of the specified translation file. During translation, a progress dialog is displayed, as well as error messages, if any. After translation, the dialog box and all open editor windows of the project are closed.

Cancel closes the dialog box without modification to the current project.

If the translation file contains erroneous entries, an error message is displayed after OK is pressed, giving the file path and the erroneous line, e.g.: "[C:\Programs\codesys\projects\visu.tlt (78)]; Translation text expected"

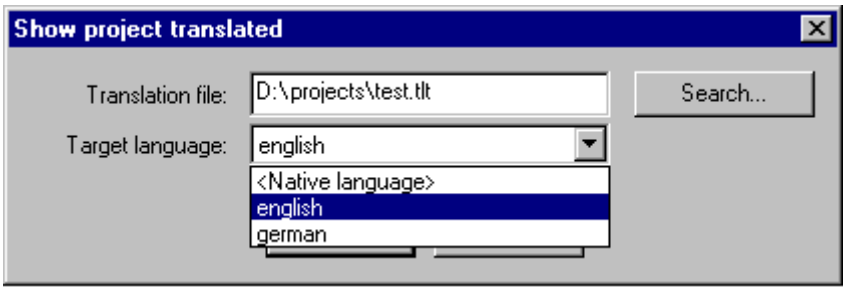
Regard the specialities for texts in visualizations.

Show project translated

If there is a translation file available for the project, you can display one of the language versions defined there, without overwriting the original language version of the project.

(Regard this possibility in comparison to the "real" translating of a project, which you would do with the command 'Translate Project', and which would mean to create a new version of the project !)

The command 'Translate this project' in menu 'Project' 'Translate into another language' opens the dialog 'Translate project into another language'.



In field Translation file insert the path of the translation file, you want to use. You can receive assistance by the standard dialog for opening a file which is opened by button Browse.

In field Target language you find a selection list, which besides the entry "<Native language>" also offers the language identifiers which are defined by the currently set translation file. The original language is that one, which is currently saved with the project. (It only could be changed by a 'Project' 'Translate'.) Choose one of the available languages and confirm the dialog with OK. Thereupon the project will be displayed in the chosen language, but cannot be edited in this view !

If you want to change back to viewing the project in its original language, use command 'Switch translation'.

Regard the specialities for texts in visualizations.

'Toggle translation'

If you have got displayed the project (not editable) in another language by command 'Show project translated', you can now switch between this language version and the (editable) original version by using the command 'Switch translation' of menu 'Project' 'Translate (into another Language)' .

Regard the specialities for texts in visualizations.

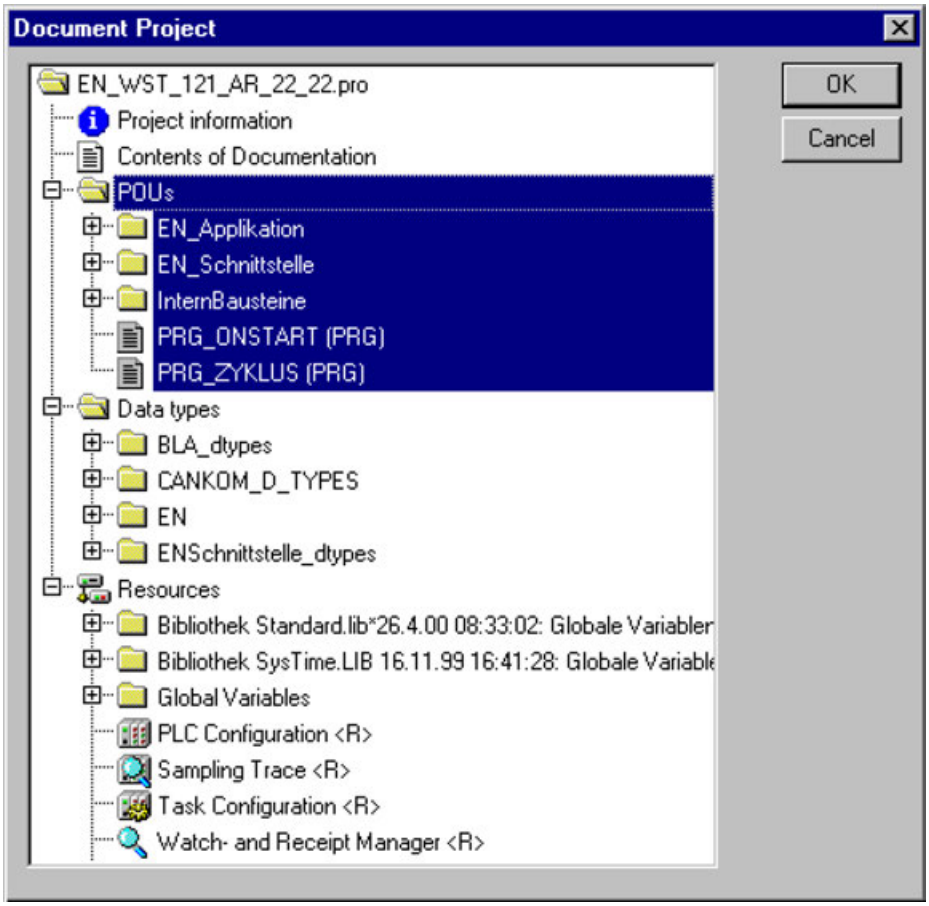
'Project' 'Document'

This command lets you print the documentation of your entire project. The elements of a complete documentation are:

- The POUs,
- the contents of the documentation,
- the datatypes *Chapter 1.4.1.8.1.1 “Data types” on page 443*,
- the visualizations *Chapter 1.4.3.1 “Overview” on page 636*,

- the resources, global variables, variables configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, the Watch and Receipt Manager),
- the call trees ↳ *Chapter 1.4.1.2.4.16 “'Project' 'Show call tree'” on page 264* of POUs and data types, as well as
- the cross-reference list ↳ *Chapter 1.4.1.2.4.17 “'Project' 'Show cross reference'” on page 264*.

For the last two items the project must have been built without errors.



Only those areas highlighted in blue in the dialog box are printed.

If you want to select the entire project, then select the name of your project in the first line.

If, on the other hand, you only want to select a single object, then click on the corresponding object or move the dotted rectangle onto the desired object with the arrow key. Objects which have a plus sign in front of their symbols are organization objects which contain other objects. With a click on a plus sign organization object is expanded, and with a click on the resulting minus sign it can be closed up again. When you select an organization object, then all relevant objects are also selected. By pressing the <Shift> key you can select a group of objects, and by pressing the <Ctrl> key you can select several individual objects.

Once you have made your selection, then click on 'OK'. The Print dialog box appears. You can determine the layout of the pages to be printed with 'File' 'Printer setup' ↳ *Chapter 1.4.1.2.3.9 “'File' 'Printer setup'” on page 229*.

'Project' 'Export'

Projects can be exported or imported ↳ *Chapter 1.4.1.2.3.23 “'Project' 'Import'” on page 240*. That allows you to exchange programs between different IEC programming systems.

There is a standardized exchange format for POUs in IL, ST, and SFC (the Common Elements format of IEC 1131-3). For the POUs in LD and FBD and the other objects, CODESYS has its own filing format since there is no text format for this in IEC 1131-3.

The selected objects are written to an ASCII file.

POUs, data types, visualizations, and the resources can be exported. In addition, entries in the library manager, that is the linking information to the libraries, can be exported (not the libraries themselves!).

WARNING!
Re-importing an exported FBD or LD POU results in an error if a comment in the graphic editor contains a single quotation mark ('), as this will be interpreted as the beginning of a string!

Once you have made your selection in the dialog box window (the same way as with 'Project' 'Document' *Chapter 1.4.1.2.3.21 “'Project' 'Document'” on page 238*), you can decide, whether you want to export the selected parts to one file or to export in separate files, one for each object. Switch on or off the option One file for each object then click on OK. The dialog box for saving files appears. Enter a file name with the expansion ".exp" respectively a directory for the object export files, which then will be saved there with the file name <objectname.exp>.

'Project' 'Import'

In the resulting dialog box for opening files select the desired export file *Chapter 1.4.1.2.3.22 “'Project' 'Export'” on page 239*.

The data is imported into the current project. If an object with the same name already exists in the same project, then a dialog box appears with the question "Do you want to replace it?": If you answer Yes, then the object in the project is replaced by the object from the import file. If you answer No, then the name of the new objects receives as a supplement an underline and a digit ("_0", "_1", ..). With Yes, all or No, all this is carried out for all objects.

If the information is imported to link with a library, the library will be loaded and appended to the end of the list in the library manager. If the library was already loaded into the project, it will not be reloaded. If, however, the export file that is being imported shows a different storage time for the library, the library name is marked with a "*" in the library manager (e.g. standard.lib*30.3.99 11:30:14), similar to the loading of a project. If the library can not be found, then an information dialog appears: "Cannot find library {<path>\}<name> <date> <time>", as when a project is loaded.

In the message window the import is registered.

'Project' 'Compare'

This command is used to compare two projects or to compare the actual version of one project with that which was saved last.

Definitions:

- actual project: Project, which you are currently working on.
- reference project: Project, which should be compared with the actual project.
- compare mode: in this mode the project will be displayed after the command 'Project' 'Compare' has been executed.
- unit: Smallest unit which can be compared. Can be a line (declaration editor, ST editor, IL editor), a network (FBD editor, LD editor) or a element/POU (CFC editor, SFC editor).

In compare mode the actual project and the reference project will be presented in a bipartited window. The names of the POUs, for which differences have been found, are marked by color. For editor POUs also the content of the POUs is displayed in a vis-a-vis way. The results and the way of presenting in compare mode depend on: 1. what filters have been activated for the compare run, affecting the consideration of whitespaces and comments during comparison; 2. whether modification within lines or networks or elements are evaluated as a completely new inserting of a POU or not.

The version of the reference project can be accepted for single differences or for 'all equally marked' differences. To accept means that the version of the reference project is taken over to the actual project.

> *In compare mode (see status bar: COMPARE) the project cannot get edited!*

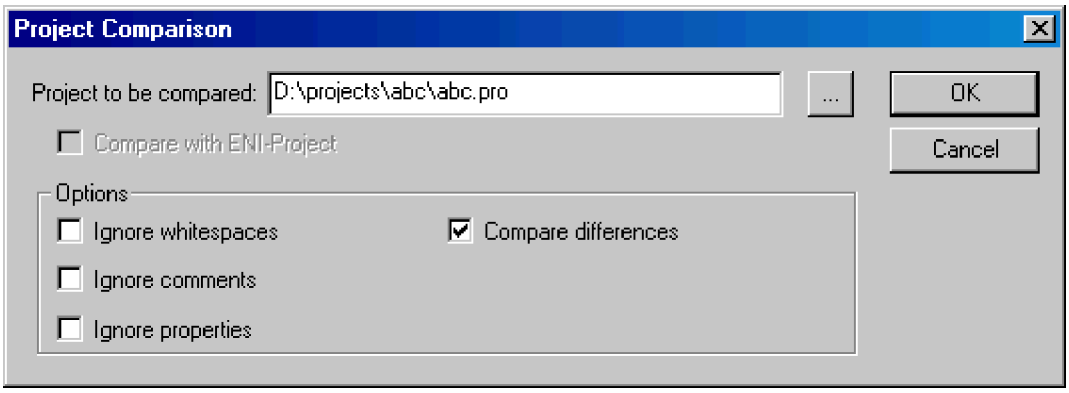
See also:

Chapter 1.4.1.2.3.25 “Execute comparison” on page 241

Chapter 1.4.1.2.3.26 “Representation of the comparison result” on page 242

Execute comparison

After executing the command 'Project' 'Compare' the dialog Project Comparison opens:



Insert the path of the reference project at Project to compare *Chapter 1.4.1.2.3.24 “'Project' 'Compare'” on page 240*. Press button if you want to use the standard dialog for opening a project. If you insert the name of the actual project, the current version of the project will be compared with the version which was saved last.

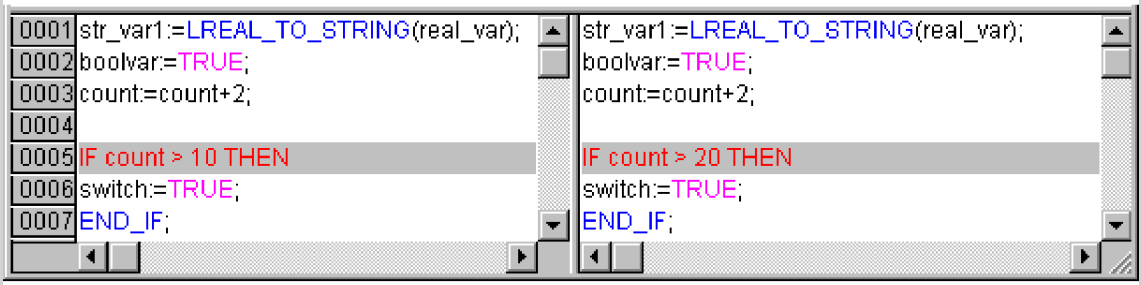
If the project is under source control in an ENI data base, then the local version can be compared with the actual version found in the data base. For this activate option Compare with ENI-Project.

The following options concerning the comparison can be activated:

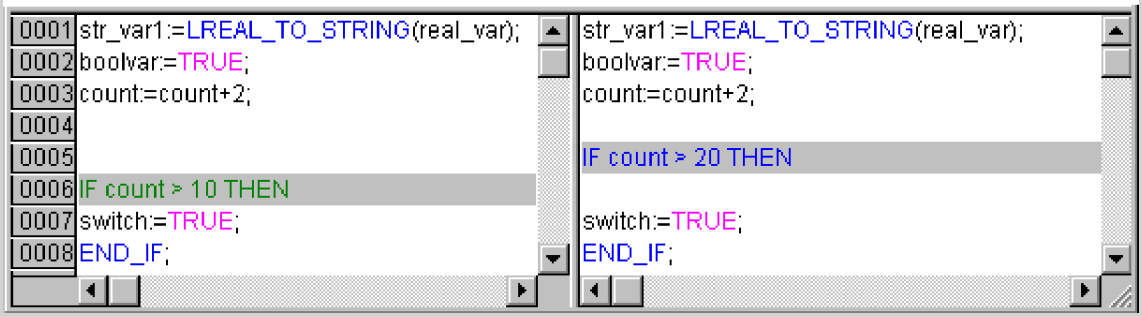
- Ignore whitespaces: There will be detected no differences which consist in a different number of whitespaces.
- Ignore comments: There will be detected no differences in comments.
- Ignore properties: There will be detected no differences in object properties.
- Compare differences: If a line, a network or an element within a POU has been modified, in compare mode it will be displayed in the bipartited window directly opposite to the version of the other project (marked red, see below). If the option is deactivated, the corresponding line will be displayed in the reference project as 'deleted' and in the actual project as 'inserted' (blue/green, see below). This means it will not be displayed directly opposite to the same line in the other project.

Example: Line 0005 has been modified in actual project (left side).

Option 'Oppose differences' activated:



Option 'Oppose differences' not activated:



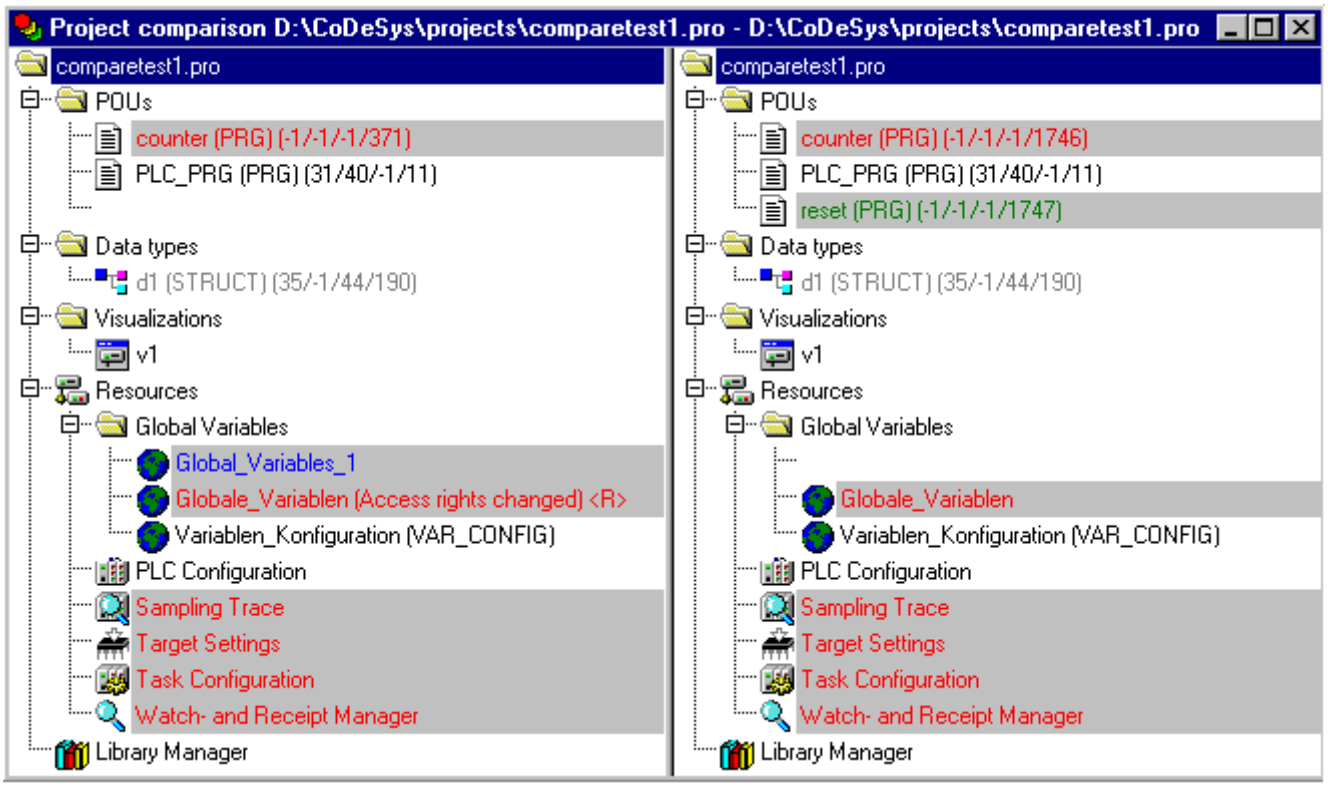
When the dialog 'Project Comparison' is closed by pressing OK, the comparison will be executed according to the settings.

Representation of the comparison result

First the structure tree of the project, titled with "Project Comparison", will be opened to display the results of the comparison. Here you can select particular POUs to see the found differences in detail.

1. Project overview in compare mode:

After the project compare has been executed, a bipartited window opens which shows the project in compare mode. In the title bar you find the project paths: "Project comparison <path of actual project> - <path of reference project>". The actual project is represented In the left half of the window, the reference project in the right one. Each structure tree shows the projects' name at the uppermost position, apart from that it corresponds to the the object organizer structure.



POUs which are different, are marked in the structure tree by a shadow, a specific color and eventually by an additional text :

- Red: Unit has been modified; is displayed with red colored letters in both partitions of the window.
- Blue: Unit only available in compare project; a gap will be inserted at the corresponding place in the structure overview of the actual project.
- Green: Unit only available in actual project; a gap will be inserted at the corresponding place in the structure overview of the actual project.
- Black: Unit for which no differences have been detected.

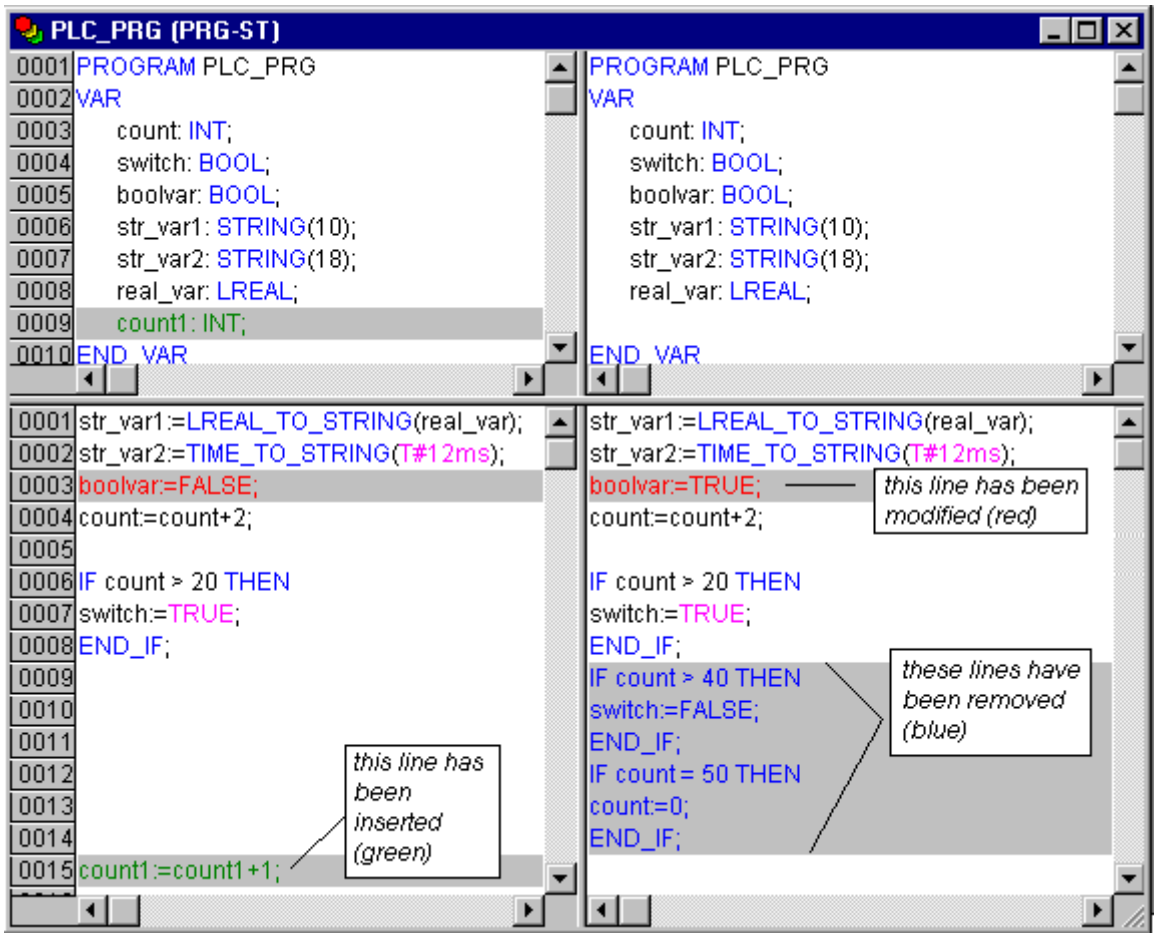
"(Properties changed)": This text is attached to the POU name in the project structure tree, if differences in the properties of the POU have been detected.

"(Access rights changed)": This text is attached to the POU name in the project structure tree, if differences in the access rights of the POU have been detected.

2. POU contents in compare mode:

By a double click on a line in the structure overview, which is marked red because of a modification, the POU is opened.

- If it is a text or graphic editor POU, it will be opened in a bipartited window. The content of the reference project (right side) is set opposite to that of the actual project (left side). The smallest unit which will be regarded during comparison, is a line (declaration editor, ST, IL), a network (FBD, LD) or an element (CFC, SFC). The same coloring will be used as described above for the project overview.
  Example:



- If it is not a editor POU, but the task configuration, the target settings etc., then the POU version of the actual and the reference project can be opened in separate windows by a double click on the respective line in the project structure. For those project POUs no further details of differences will be displayed.

'Extras' 'Next difference'

Shortcut: <F7>

This command is available in the compare mode *Chapter 1.4.1.2.3.24 "'Project' 'Compare'" on page 240*.

The cursor jumps to the next unit, where a difference is indicated. (line in project overview, line/network/element in POU)

'Extras' 'Previous difference'

Shortcut: <Shift><F7>

This command is available in the compare mode *Chapter 1.4.1.2.3.24 "'Project' 'Compare'" on page 240*.

The cursor jumps to the previous unit, where a difference is indicated (line in project overview, line/network/element in POU) .

'Extras' 'Accept change'

Shortcut: <Space>

This command is available in the compare mode ↳ *Chapter 1.4.1.2.3.24 “'Project' 'Compare'” on page 240*.

For all units, which are cohering with that one where the cursor currently is placed, and which have the same sort of difference marking, (e.g. subsequent lines), the version of the reference project will be accepted for the actual project (only possible in this direction!) ↳ *Chapter 1.4.1.2.3.25 “Execute comparison” on page 241*. The corresponding units will be shown (with the corresponding coloring) in the left side of the window.

For accepting changes of particular units please use 'Accept changed item' ↳ *Chapter 1.4.1.2.3.30 “'Extras' 'Accept changed item'” on page 245*.

The acceptance of different project parts (differences) or access right properties is only possible from the reference project of the actual project, not vice versa.

'Extras' 'Accept changed item'

Shortcut: <Ctrl> <Spacebar>

This command is available in the compare mode ↳ *Chapter 1.4.1.2.3.24 “'Project' 'Compare'” on page 240*.

Only the single unit (line, network, element) where the cursor is currently placed, will be accepted for the actual version ↳ *Chapter 1.4.1.2.3.25 “Execute comparison” on page 241*. The corresponding units will be shown (with the corresponding coloring) in the left side of the window.

If for a POU, which has got marked red-colored in the structure tree because of a change of its content, this change gets accepted, then the POU will be indicated by yellow coloring in the actual project.

POUs which are only available in the current project because of 'Accept changed item', also will be indicated by yellow coloring. POUs which have been removed from the current project because of 'Accept changed item', will be indicated by yellow coloring in the reference project ↳ *Chapter 1.4.1.2.3.25 “Execute comparison” on page 241*.

The acceptance of different project parts (differences) or access right properties is only possible from the reference project of the actual project, not vice versa.

'Extras' 'Accept properties'

This command is available in the compare mode ↳ *Chapter 1.4.1.2.3.24 “'Project' 'Compare'” on page 240*.

The object properties for the POU, where the cursor is currently placed, will be accepted for the actual project as they are set in the ence version ↳ *Chapter 1.4.1.2.3.25 “Execute comparison” on page 241*.

The acceptance of different project parts (differences) or access right properties is only possible from the reference project of the actual project, not vice versa.

'Extras' 'Accept access rights'

This command is available in the compare mode only in project overview *Chapter 1.4.1.2.3.24 "'Project' 'Compare'" on page 240*.

The object access rights for the POU, where the cursor is currently placed, will be accepted for the actual project as they are set in the reference version *Chapter 1.4.1.2.3.25 "Execute comparison" on page 241*.

'Project' 'Merge'

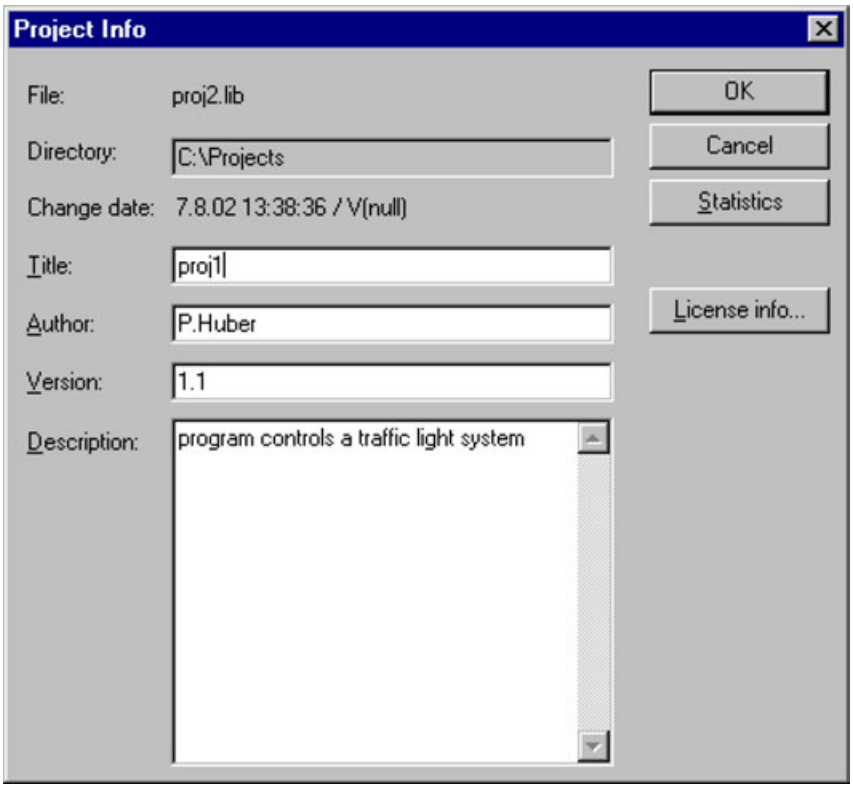
With this command you can merge objects (POUs, data types, visualizations, and resources) as well as links to libraries from other projects into your project.

When the command has been given, first the standard dialog box for opening files appears. When you have chosen a file there, a dialog box appears in which you can choose the desired object. The selection takes place as described with 'Project' 'Document' *Chapter 1.4.1.2.3.21 "'Project' 'Document'" on page 238*.

If an object with the same name already exists in the project, then the name of the new object receives the addition of an underline and a digit ("_1", "_2" ...).

'Project' 'Project info'

Under this menu item the information about your project can be saved. When the command has been given, then the following dialog box opens:



The following project information is displayed:

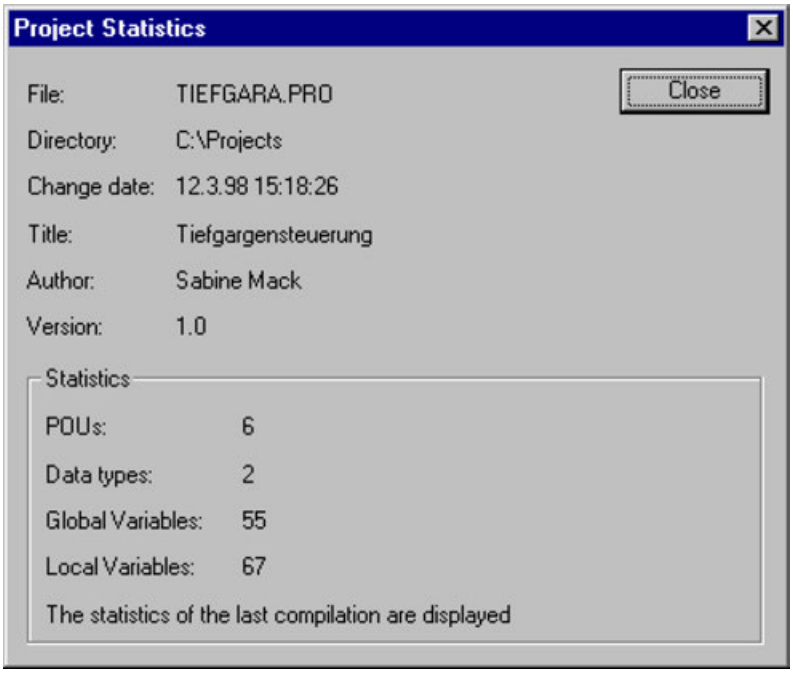
- File name
- Directory path
- The time of the most recent change (Change date)

This information cannot be changed. In addition, you can add the following information:

- A Title of the project: Please regard: If supported by the target system, this title automatically will be proposed as project file name, when the project gets loaded by command 'File' 'Open project from PLC' (In this case the dialog for saving a file will open).
- the name of the Author,
- the Version number, and
- a Description of the project.

This information is optional. When you press the button Statistics you receive statistical information about the project.

It contains information such as the number of the POUs, data types, and the local and global variables as they were traced at the last compilation.



The button 'License info' will be available, if you work on a project, which had been saved already with licensing information by the command 'File' 'Save as' *Chapter 1.4.1.2.3.6 "'File' 'Save as'" on page 224*. In this case the button opens the dialog 'Edit Licensing Information', where you can modify or remove the license.

If you choose the option 'Ask for project info' in the category 'Load & Save' in the Options dialog box, then while saving a new project, or while saving a project under a new name, the project info dialog is called automatically.

'Project' 'Global search'

With this command you can search for the location of a text in POUs, data types, or in the objects of the global variables.

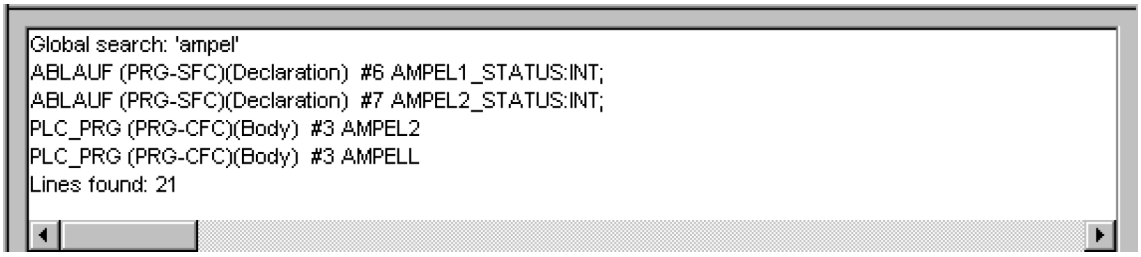
When the command is entered, a dialog box opens in which you can choose the desired object. The selection is made as in the 'Project' 'Document' description *Chapter 1.4.1.2.3.21 "'Project' 'Document'" on page 238*.

If the selection is confirmed with 'OK', the standard dialog for Search will be opened. This appears immediately when the command 'Global Search' is invoked via the symbol in the menu bar; the search is then automatically carried out in all searchable parts of the project. The most recently entered search strings can be selected through the combo box of the 'Search for' field. If a text string is found in an object, the object is loaded into the corresponding editor or in the library manager and the location where the string was found is displayed. The display of the text that is found, as well as the search and find next functions behave similarly to the command 'Edit' 'Search'.

If you select the 'In message window' button, all locations where the series of symbols searched for appears in the selected object will be listed line by line in tabular form in the message window. Afterward, the number of locations found will be displayed.

If the report window was not opened, it will be displayed. For each location that is found, the following will be displayed:

- Object name
- Location of the find in the Declaration (Decl) or in the Implementation (Impl) portion of a POU
- Line and network number if any
- The full line in the text editors
- Complete text element in the graphic editors



If you double-click the mouse on a line in the message window or press <Enter>, the editor opens with the object loaded. The line concerned in the object is marked. You can jump rapidly between display lines using the function keys <F4> and <Shift>+<F4>.

'Project' 'Global replace'

With this command you can search for the location of a text in POUs, data types, or the objects of the global variables and replace this text by another. This is executed in the same way as with 'Project' 'Global search' or 'Edit' 'Replace' ↳ *Chapter 1.4.1.2.3.35 “'Project' 'Global search'” on page 247* ↳ *Chapter 1.4.1.2.5.10 “'Edit' 'Replace'” on page 275*. The libraries, however, are not offered for selection and no display in the message window is possible.

Results are displayed in the message window.

'Project' 'Check'

This command provides commands for checking the semantic correctness of the project. The status of the most recent compilation will be regarded. If you have changed the project in the mean time, do a recompilation in order to get an up to date check result. Otherwise you will get an appropriate warning in the message window.

The results will be displayed in the message window.

In the project options, category 'Build', you can define these semantic checks to be done at each compilation of the project automatically.

Unused variables

This function in the 'Project' 'Check' menu searches for variables that have been declared but not used in the program. They are outputted by POU name and line, e.g.: PLC_PRG (4) – var1. Variables in libraries are not examined. Results are displayed in the message window.

Overlapping memory areas

This function in the 'Project' 'Check' menu tests whether in allocation of variables via the "AT" declaration overlaps have arisen at specific memory areas. For example, an overlap occurs when allocating the variables "var1 AT %QB21: INT" and "var2 AT %QD5: DWORD" because they both use byte 21. The output then appears as follows:

%QB21 is referenced by the following variables:

PLC_PRG (3): var1 AT %QB21

PLC_PRG (7): var2 AT %QD5

Results are displayed in the message window.

Multiple write acces on output

This function of the 'Project' 'Check' menu searches for memory areas to which a single project gains write access at more than one place. The output then appears as follows:

%QB24 is written to at the following locations:

PLC_PRG (3): %QB24

PLC_PRG.POU1 (8): %QB24

Results are displayed in the message window.

Concurrent access

This function in the 'Project' 'Check' menu searches for memory areas of IEC addresses which are referenced in more than one task ↬ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248*. No distinction is made here between read and write access. The output is for example:

%MB28 is referenced in the following tasks :

Task1 – PLC_PRG (6): %MB28 [read-only access]

Task2 – POU1.ACTION (1) %MB28 [write access]

Results are displayed in the message window.

User groups

Up to eight user groups with different access rights to the POUs, data types, visualizations, and resources can be set up. Access rights for single objects or all of them can be established. Only a member of a certain user group can open a project. A member of such a user group must identify himself by means of a password.

The user groups are numbered from 0 to 7, whereby the Group 0 has the administrator rights, i.e. only members of group 0 may determine passwords and access rights for all groups and/or objects.

When a new project is launched, then all passwords are initially empty. Until a password has been set for the 0 group, one enters the project automatically as a member of the 0 group.

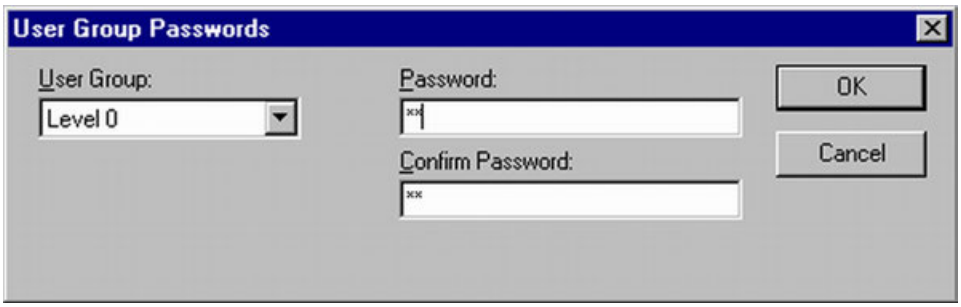
If a password for the user group 0 is existing while the project is loaded, then a password will be demanded for all groups when the project is opened.

In the combobox User group on the left side of the dialog box, enter the group to which you belong and enter on the right side the relevant password. Press OK. If the password does not agree with the saved password, then the message appears: "The password is not correct."

Only when you have entered the correct password the project can be opened. With the command 'Passwords for user group' you can assign the passwords, and with 'Object' 'Access rights' you can define the rights for single objects or for all of them.

'Project' 'Passwords for user groups'

With this command you open the dialog box for password assignment for user groups. This command can only be executed by members of group 0. When the command has been given, then the following dialog box appears:



In the left combobox 'User group' you can select the group. Enter the desired password for the group in the field 'Password'. For each typed character an asterisk (*) appears in the field. You must repeat the same password in the field 'Confirm password'. Close the dialog box after each password entry with 'OK'. If you get the message:

```
"The password does not agree with the confirmation",
```

then you made a typing error during one of the two entries. In this case repeat both entries until the dialog box closes without a message.

Then, if necessary, assign a password for the next group by calling the command again.

WARNING!
If passwords are not assigned to all user groups, a project can be opened by way of a group to which no password was assigned!

Use the command 'Object' 'Access rights' to assign the rights for single objects or all of them ↳ *Chapter 1.4.1.2.4.13 “'Project' 'Object access rights'” on page 262*.

Concerning the protection of a project see also the following:

Encryption of a project at project saving ↳ *Chapter 1.4.1.2.3.6 “'File' 'Save as'” on page 224*

'Project' 'Project database'

Overview

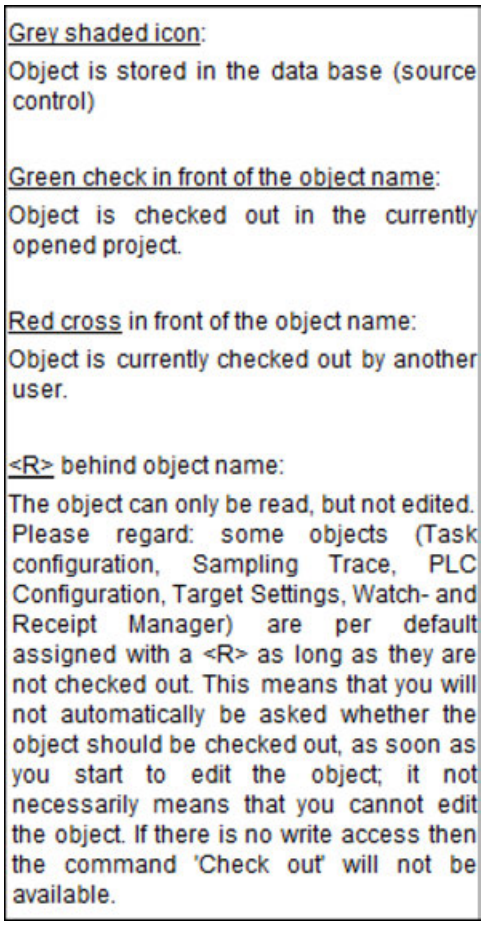
This menu item is only available if you have activated the option 'Use source control (ENI)' in the project options dialog for category 'Project source control'. A submenu is attached where you find the following commands for handling the object resp. the project in the currently connected ENI data base:

Login (The user logs in to the ENI Server)

If an object is marked in the Object Organizer and the command Data Base Link is executed (from the context menu, right mouse button), then several commands will be available for executing the corresponding data base actions. If the user had not logged in successfully to the ENI Server before, then the dialog 'Data base Login' will open automatically and the chosen command will not been executed until the login was successful.

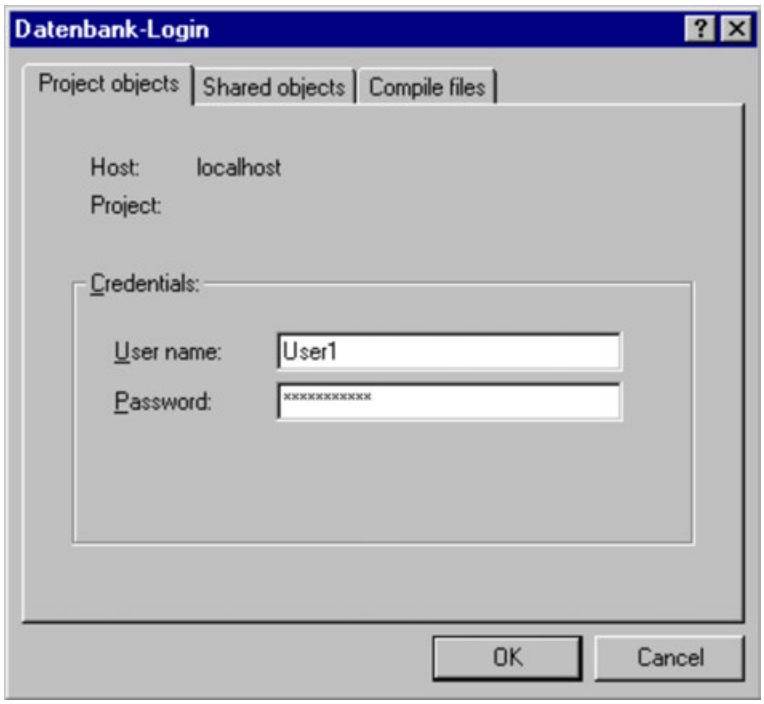
If the command 'Data Base Link' in the 'Project' menu is activated, then additional menu items will be available, which concern all objects of the project.

How the status of an object's handling in the data base is displayed in the Object Organizer:



Login

This command will open the dialog 'Login' where you can enter the access data for the ENI data base via the ENI Server. The access data also have to be defined in the ENI Server (ENI Admin, User Management) and – depending on the currently used data base – also in the user management of the data base. After the command has been executed, first the Login dialog for category 'Project objects' will open.



The following items are displayed:

- Data base: project objects
- Host: address of the computer where the ENI Server is running (must match with the entry in field 'TCP/IP address' in the project options dialog for 'Project source control').
- Project: Name of the data base project (must match with the entry in field 'Project name' in the project options dialog for 'Project source control'/category 'Project Objects').

Credentials:

- Insert username and password.
- When option Use as default for this project is activated, then the above entered access data will automatically be used for any further communication between the actual project and the data base concerning objects of the actual category.
- Press OK to confirm the settings. The dialog will be closed and automatically the Login dialog for 'Shared objects' will open. Enter the access data in the same way as described for the 'Project objects' and confirm with OK. Do the same in the third Login dialog which will be opened for category 'Compile files'.
- The Login dialog will always open as soon as you try to access the data base before having logged in successfully as described above.

If you want to save the access data with the project, activate option 'Save ENI credentials' in the project options, category 'Load & Save'.

Define

Command: 'Project' 'Data Base Link' 'Define'

Here you can define, whether the object which is currently marked in the Object organizer should be kept in the data base or just locally in the project. A dialog will open, where you can choose one of the two database categories 'Project' or 'Shared objects', or the category 'Local'.

The icons of all objects which are managed in the data base will be displayed grey-shaded in the Object organizer. Shared objects are displayed with turquoise letters.

Get latest version

Command: 'Project' Data Base Link'Get Latest Version'

The current version of the object which is marked in the Object organizer will be copied from the data base and will overwrite the local version. In contrast to the Check Out action the object will not be locked for other users in the data base *Chapter 1.4.1.2.3.44.5 “Check out” on page 253*.

Check out

Command: 'Project' 'Data Base Link' 'Check Out'

The object which is marked in the Object organizer will be checked out from the data base and by that will be locked for other users.

When executing the command the user will get a dialog 'Check out object'. A comment can be added there which will be stored in the version history of the object in the data base. Line breaks are inserted by <Ctrl>+<Enter>. If the version of the object differs from that in the local project, an appropriate message will be displayed and the user can decide whether the object should be checked out anyway.

After the dialog has been closed with OK, the checked-out object will be marked with a green check in the object organizer of the local project. For other users it will be appear marked with a red cross and will not be editable by them.

Check in

Command: 'Project' 'Data Base Link' 'Check In'

The object which is marked in the Object organizer will be checked in to the data base. Thereby a new version of the object will be created in the data base. The old versions will be kept anyway.

When executing the command the user will get a dialog 'Check in object'. There a comment can be added which will be stored in the version history of the object in the data base. Line breaks are inserted by <Ctrl>+<Enter>.

After the dialog has been closed with OK the green check in front of the object name in the Object organizer will be removed.

Undo check out

Command: 'Project' 'Data Base Link' 'Undo Check Out'

Use this command to cancel the Checking out of the object which is currently marked in the Object organizer. Thereby also the modifications of the object which have been made locally, will be canceled. No dialog will appear. The unchanged last version of the object will be kept in the data base and it will be accessible again for other users. The red cross in front of the object name in the Object organizer will disappear.

Show differences

Command: 'Project' 'Data Base Link' 'Show Differences'

The object which is currently open will be displayed in a window which is divided up in two parts. There the local version, which is currently edited by the local user, will be opposed to the last (actual) version which is kept in the data base. The differences of the versions will be marked like described for the project comparison (see 'Project' 'Compare').

Show version history

Command: 'Project' 'Data Base Link' 'Show Version History'

For the currently marked object in the Object organizer a dialog Version history of <object name> will be opened. There all versions of the object are listed which have been checked in to the data base or which have been labeled there:

The following information is given:

Version: Data base specific numbering of the versions of the object which have been checked in one after the other. Labeled versions get no version number but are marked by a label-icon.

User: Name of the user, who has executed the check-in or labeling action

Date: Date and time stamp of the action

Action: Type of the action which has been executed. Possible types: 'created' (the object has been checked in to the data base for the first time), 'checked in' (all check-ins of the object excluding the first one) and labeled with <label> (a label has been assigned to this version of the object)

The buttons:

Close: The dialog will be closed.

Display: The version which is currently marked in the table will be opened. The title bar shows: ENI: <name of the project in the data base>/<object name>

Details: The dialog 'Details of Version History' will open:

File (name of the project and the object in the data base), Version (see above), Date (see above), User (see above), Comment (Comment which has been inserted when the object has been checked in resp. has been labeled). Use the buttons Next resp. Previous to jump to the details window of the next or previous entry in the table in dialog 'Version history of ..'.

Get latest version: The version which is marked in the table will be loaded and there will overwrite the local version.

Differences: If in the table only one version of an object is marked, then this command will cause a comparison of this version with the latest (actual) data base version. If two versions are marked, then those will be compared. The differences are displayed in a bipartited window like it is done at the project comparison.

Reset version: The version which is marked in the table will be set as latest version. All versions which have been checked in later will be deleted ! This can be useful to restore an earlier status of an object.

Labels only: If this option is activated, then only those versions of the object will be displayed in the table, which are marked by a label.

Selection box below the option 'Labels only': Here you find the names of all users which have executed any data base actions for objects of the current project. Select 'All' or one of the names if you want to get the version history concerning all users or just for a certain one.

Multiple define

Command 'Project' 'Data Base Link' 'Multiple Define'

Use this command if you want to assign several objects at a single blow to a certain data base category. The dialog 'Properties' will open like described for command 'Define'. Choose the desired category and close the dialog with OK. After that the dialog 'ENI-Selection' will open, listing all POUs of the project which are considered for the chosen category (Example: if you choose category 'shared objects' then the selection window will only offer the POUs of the Resources tab). The POUs are presented in a tree structure complying to that of the Object Organizer. Select the desired POUs and confirm with OK.

Get all latest versions

Command 'Project' 'Data Base Link' 'Get All Latest Versions'

The latest version of each object of the currently opened project, which is kept under source control, will be called from the data base. Consider the following:

- If in the meantime additional objects have been stored to the data base project folder, then those will now be added to the local project.
- If objects have been deleted in the data base in the meantime, those will not be deleted in the local project, but they will automatically get assigned to category 'Local'.
- The latest version of objects of category 'Shared Objects' will only be called, if these objects are already available in the local project. For further information see command 'Get latest version' *Chapter 1.4.1.2.3.44.4 “Get latest version” on page 253*.

Multiple check out

Command 'Project' 'Data Base Link' 'Multiple Check Out'

You can check out several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those which should be checked out and confirm with OK. For further information see command 'Check Out'.

Multiple check in

Command 'Project' 'Data Base Link' 'Multiple Check In'

You can check in several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those which should be checked in and confirm with OK. For further information see command 'Check In'.

Multiple undo check out

Command 'Project' 'Data Base Link' Undo Multiple Check Out'

You can undo the check out action for several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those for which you want to cancel the check out and confirm with OK. For further information see command 'Undo Check Out'.

Project version history

Command 'Project' 'Data Base Link' 'Project Version History

If the chosen data base system supports that functionality, you can use this command to view the version history for the currently opened project.

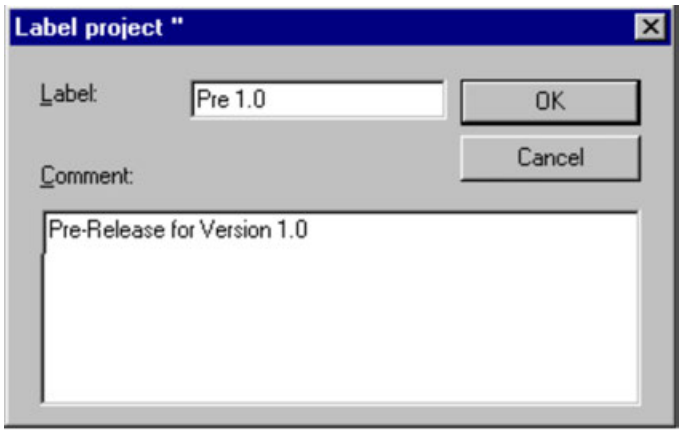
The dialog 'History of <data base project name>' will open. It shows the actions (create, check in, label) which have been performed for the particular objects of the project in a chronological order. The total number of objects is displayed behind Version history. The dialog can be handled like described for command 'Show Version History', but regard the following:

- The command 'Reset Version' is only available for single objects.
- The command 'Get latest version' means that all objects of the version of the currently marked object will be called to the local project! That means, that the objects in CODESYS will be overwritten with the older version. But: Local objects, which were not yet part of the project in that older version, will not be removed from the local project ! If a labeled version is called, which contains Shared Objects, then the user will get a dialog where he can decide whether those Shared Objects should be called also or not.

Label version

Command 'Project' 'Data Base Link' 'Label Version'

This command is used to put a "label" on the actual version of each object of a project, so that exactly this project version can be recalled later. A dialog 'Label <data base project name>' will open. Insert a label name (Label) (e.g. "Release Version") and optionally a Comment. When you confirm with OK, the dialog will close and the label and the action "labeled with <label name>" will appear in the table of the version history, as well in the history for a single object as in the history of the project. Shared Objects which are part of the project will also get that label. A labeled version of the project does not get a version number, but is just marked with a label icon in the column 'Version'. If the option 'Labels only' is activated in the Version History dialog, then only labeled versions will be listed.

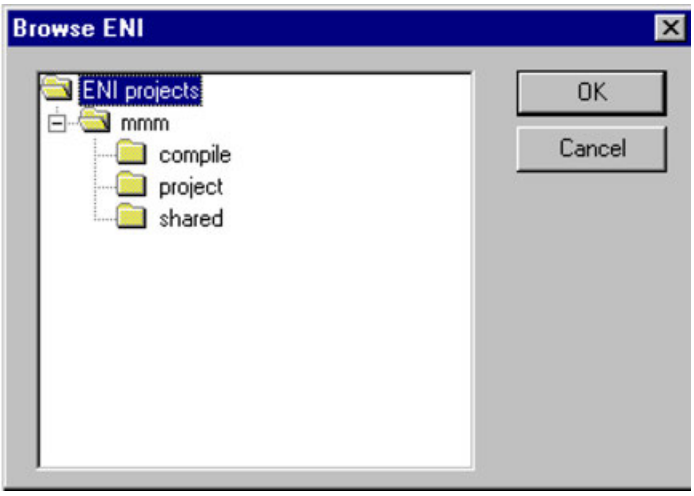


Add shared objects

Command 'Project' 'Data Base Link' 'Add Shared Objects'

Use this command if you explicitly want to add new objects of data base category 'Shared Objects' to the locally opened project. For objects of category 'Project Objects' this is not necessary, because the command 'Get (all) latest version(s)' automatically calls all objects which are found in the data base project folder, even if there are some which not yet available in the local project. But for objects of category 'Shared Objects' in this case just those objects will be called which are already available in the local project.

So execute the command 'Add Shared Objects' to open the dialog 'Browse ENI'. A list in the right part of the window shows all objects which are available in the data base folder which is currently selected in the list on the left side. Choose the desired object and press OK or do a doubleclick on the entry to insert the object to the currently opened project.



Refresh status

Command 'Project' 'Data Base Link' 'Refresh Status'

Use this command to update the display in the Object Organizer, so that you can see the actual status of the objects concerning the source control of the project.

1.4.1.2.4 Managing objects in a project

Object

POUs, data types, visualizations and the resources global variables, the variable configuration, the Sampling Trace, the PLC Configuration, the Task Configuration, and the Watch and Receipt Manager are all defined as "objects". The folders inserted for structuring the project are partially involved. All objects of a project are in the Object Organizer.

If you hold the mouse pointer for a short time on a POU in the Object Organizer, then the type of the POU (Program, Function or Function block) is shown in a Tooltip. For the global variables the tooltip shows the keyword (VAR_GLOBAL, VAR_CONFIG).

With drag & drop you can shift objects (and also folders, see 'Folder') within an object type. For this, select the object and shift it to the desired spot by holding down the left mouse button. If the shift results in a name collision, the newly introduced element will be uniquely identified by an appended, serial number (e.g. "Object_1").

Folder

In order to keep track of larger projects you should group your POUs, data types, visualizations, and global variables systematically in folders.

You can set up as many levels of folders as you want. If a plus sign is in front of a closed folder symbol  , then this folder contains objects and/or additional folders. With a click on the plus sign the folder is opened and the subordinated objects appear. With a click on the minus (which has replaced the plus sign) the folder can be closed again. In the context menu you find the commands 'Expand nodes' and 'Collapse nodes' with the same functions *Chapter 1.4.1.2.4.4 “'Expand nodes' 'Collapse nodes'” on page 258*.

With Drag&Drop you can move the objects as well as the folders within their object type. For this select the object and drag it with pressed left mouse button to the desired position.

You can create more folders with the command 'New folder' *Chapter 1.4.1.2.4.3 “'New folder'” on page 258*.

Folders have no influence on the program, but rather serve only to structure your project clearly.

'New folder'

With this command a new folder is inserted as a structural object *Chapter 1.4.1.2.4.2 “Folder” on page 257*. If a folder has been selected, then the new one is created underneath it. Otherwise it is created on the same level. If an action is selected, the new folder will be inserted at the level of the POU to which the action belongs.

The context menu of the 'Object Organizer' which contains this command appears when an object or the object type has been selected and you have pressed the right mouse button or <Shift>+<F10> *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*.

The newly inserted folder initially has the designation 'New Folder'. Observe the following naming convention for folders:

- Folders at the same level in the hierarchy must have distinct names.
- Folders on different levels can have the same name. A folder can not have the same name as an object located on the same level.

If there is already a folder with the name "New Folder" on the same level, each additional one with this name automatically receives an appended, serial number (e.g. "New Folder 1"). Renaming to a name that is already in use is not possible.

'Expand nodes' 'Collapse nodes'

With the Expand nodes command the objects which are located in the selected object are visibly unfolded. Collapse nodes hides the subordinate objects.

With folders you can open or close them with a double mouse click or by pressing <Enter> *Chapter 1.4.1.2.4.2 “Folder” on page 257*.

This command appears in the context menu of the Object Organizer when you right-click on an object, or press <Shift>+<F10>.

'Project' 'Object' 'Delete'

Shortcut: <Delete>

With this command the currently selected object (a POU, a data type, a visualization, or global variables), or a folder with the subordinated objects is removed from the Object Organizer and is thus deleted from the project. Deleting of an object can be reversed by the command 'Edit' 'Undo'.

You can get back the deleted objects by using the command 'Edit' 'Undo'.

If the editor window of the object was open, then it is automatically closed.

If you delete with the command 'Edit' 'Cut', then the object is parked on the clipboard *Chapter 1.4.1.2.5.4 “'Edit' 'Cut'” on page 273*.

'Project' 'Object' ' Add'

Shortcut: <Insert>

With this command you create a new object. The type of the object (POU, data type, visualization, or global variables) depends upon the selected register card in the Object Organizer. Note that in doing so possibly a template will be used for objects of type 'Global Variables', 'Data types', 'Function', 'Function Block' or 'Program' ., see below, chapter 'Save as template' *Chapter 1.4.1.2.4.7 “'Save as template'” on page 259*.

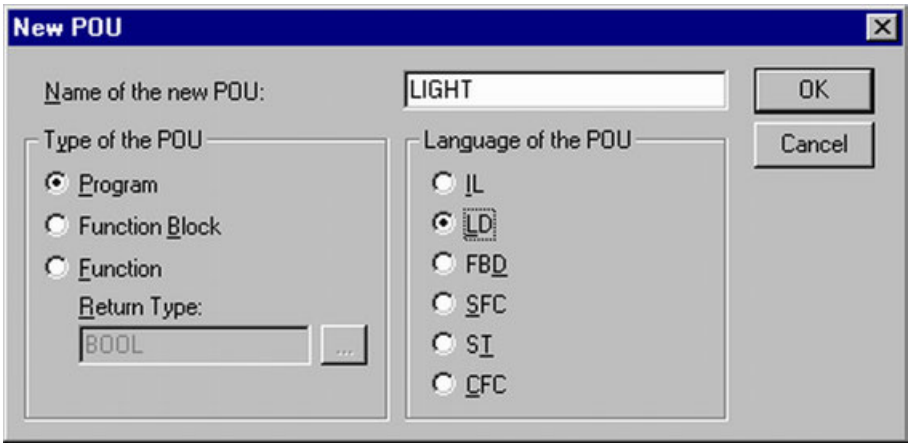
Enter the 'Name of the new POU' in the dialog box which appears. Remember that the name of the object may not have already been used.

Take note of the following restrictions:

- The name of a POU can not include any spaces
- A POU can not have the same name as another POU, a data type and should not have the same as a visualization in order to avoid problems with visualization changes.
- A data type can not receive the same name as another data type or a POU.
- A global variable list can not have the same name as another global variable list.
- An action can not have the same name as another action in the same POU.
- A visualization can not have the same name as another visualization and should not have the same as a POU in order to avoid problems with visualization changes.

In all other cases, identical naming is allowed. Thus for example actions belonging to different POUs can have the same name, and a visualization may have the same as a POU.

In the case of a POU, the POU type (program, function or function block) and the language in which it is programmed must also be selected. 'Program' is the default value of 'Type of the POU', while that of 'Language of the POU' is that of most recently created POU. If a POU of the function type is created, the desired data type must be entered in the 'Return Type' text input field. Here all elementary and defined data types (arrays, structures, enumerations, aliases) are allowed. Input assistance (e.g. via <F2>) can be used.



After pressing 'OK', which is only possible if there is no conflict with the naming conventions described above, the new object is set up in the Object Organizer and the appropriate input window appears.

If the command 'Edit' 'Insert' is used, the object currently in the clipboard is inserted and no dialog appears. If the name of the inserted object conflicts with the naming conventions (see above), it is made unique by the addition of a serial number appended with a leading underline character (e.g. "Rightturnsig_1").

If the project is under source control in an ENI data base, it may be (depends on the settings in the Project options dialog for 'Project source control') that you will be automatically asked in which data base category you want to handle the new object ↯ *Chapter 1.4.1.2.2.13 “Options for 'Project source control'” on page 216*. In this case the dialog 'Properties' will open where you can assign the object to one of the database object categories.

'Save as template'

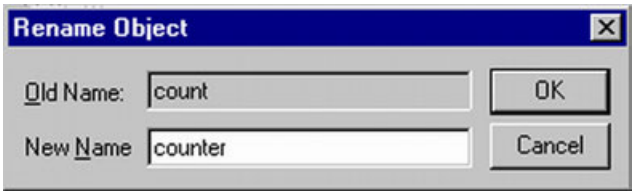
Objects of type 'Global Variables', 'Data types', 'Function', 'Function Block' or 'Program' can be saved as templates. Select the object in the Object Organizer and choose command 'Save as template' in the context menu (right mouse button). Hereupon each further new object of the same type will automatically initially get the declaration part of the template ↯ *Chapter 1.4.1.2.4.6 “'Project' 'Object' ' Add'” on page 258*. The last created template for an object type will be used.

'Project' 'Object' 'Rename'

Shortcut: <Spacebar>

With this command you give a new name to the currently-selected object or folder ↳ *Chapter 1.4.1.2.4.2 “Folder” on page 257*. Remember that the name of the object may not have already been used.

If the editing window of the object is open, then its title is changed automatically when the name is changed.



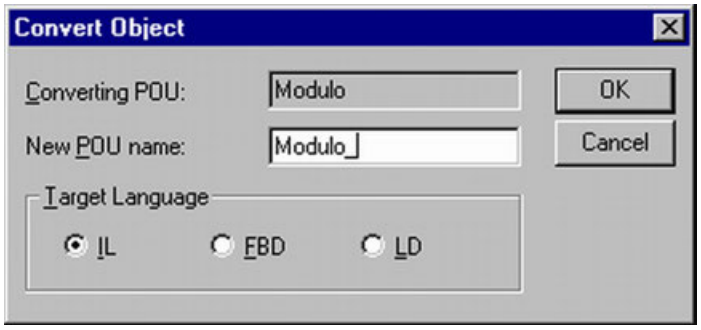
'Project' 'Object' 'Convert'

This command can only be used with POUs. You can convert POUs from the languages SFC, ST, FBD, LD, and IL into one of the three languages IL, FBD, and LD.

For this the project must be compiled. Choose the language into which you want to convert and give the POU a new name. Remember that the name of the POU may not have already been used. Then press 'OK', and the new POU is added to your POU list.

The type of processing that occurs during conversion corresponds to that which applies to compilation.

NOTICE!
Actions cannot be converted.

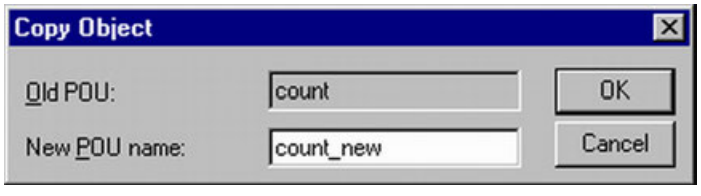


Note the following possibility: A POU which has been created in the FBD-Editor can, using the command 'Extras' 'View' be displayed and edited in the KOP-Editor as well without any conversion ↳ *Chapter 1.4.1.3.11.7.12 “'Extras' 'View'” on page 321*.

'Project' 'Object' 'Copy'

With this command a selected object is copied and saved under a new name. Enter the name of the new object in the resulting dialog box. Remember that the name of the object may not have already been used.

If, on the other hand, you used the command 'Edit' 'Copy', then the object is parked on the clipboard, and no dialog box appears.



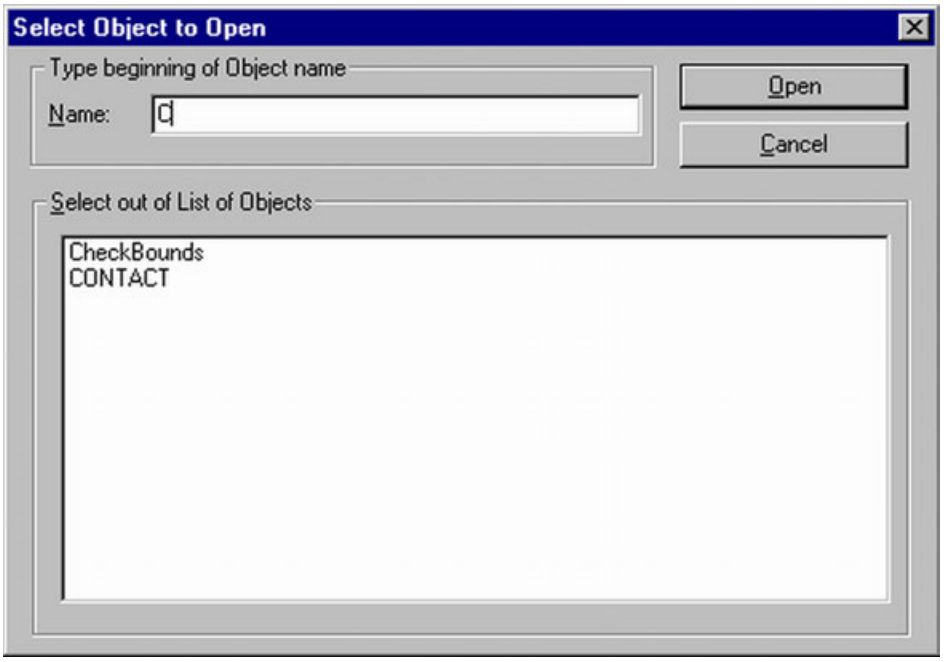
'Project' 'Object' 'Open'

Shortcut: <Enter>

With the command you load a selected object within the 'Object Organizer' into the respective editor *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*. If a window with this object is already open, then it gets a focus, is moved into the foreground and can now be edited.

There are two other ways of opening an object:

- Doubleclick with the mouse on the desired object
- in the Object Organizer, type the first letter of the object name. This will open a dialog box in which all objects of the available object types which have this initial letter are shown. Actions are listed with the notation <POU name>.<action name>. Due to the fact that the objects in the object selection dialog are listed alphabetically, the actions of a POU always get positioned below this POU. Select the desired object and click on the button 'Open' in order to load the object in its edit window. Hereupon the object gets also marked in the object organizer and all folders which are hierarchically placed above the object will get expanded. This option is supported with the object type 'Resources' only for global variables.



'Project' 'Object properties'

This command will open the dialog 'Properties' for that object which is currently marked in the Object organizer.

On the tab 'Access rights' you find the same dialog as you get when executing the command 'Project' 'Object access rights' *Chapter 1.4.1.2.4.13 “'Project' 'Object access rights'” on page 262*.

It depends on the object and the project settings, whether there are additional tabs available where you can define object properties:

Global variable list: In the tab 'Global variable list' the parameters concerning the actualization of the list and concerning the data exchange of network variables are displayed and can be modified here. This dialog also will be opened if you create a new global variable list by selecting one of the entries in section 'Global Variables' in the Object Organizer and executing the command 'Add Object' *Chapter 1.4.1.4.1.3.1 “Create a global variable list” on page 358*.

Visualization: In the tab 'Visualization' you can define for the visualization object, how it should be used *Chapter 1.4.3 “Visualization” on page 636*:

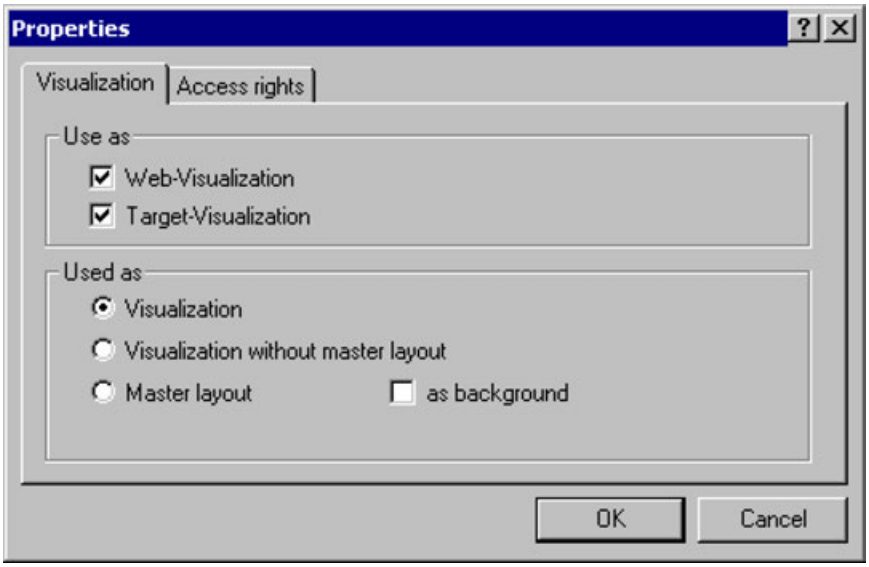
Use as:

If in the target settings the option 'web visualization' or 'target visualization' is activated, then you can choose here whether the object should be part of the web visualization or target visualization *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*.

Used as:

Activate one of these settings referring to the possibility of using "Master layouts":

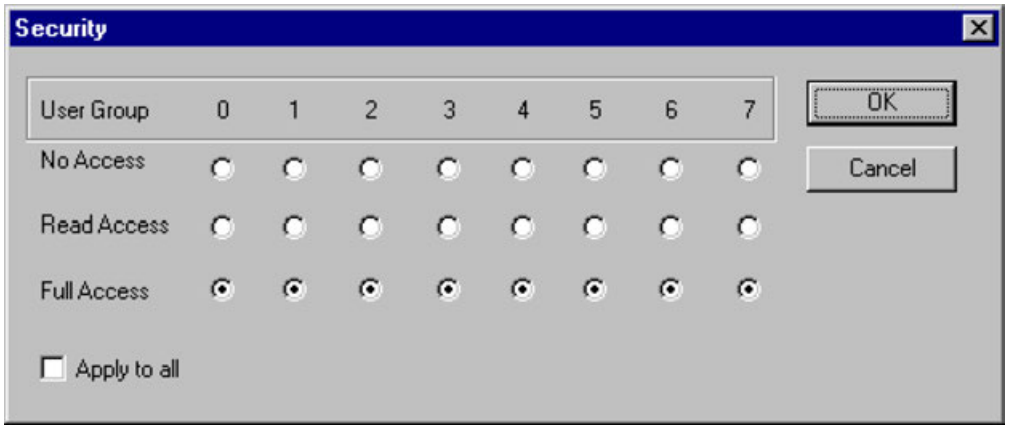
- Visualization: The object is used as a normal visualization.
- Visualization without master layout: If a Master Layout is defined in the project, it will not be applied to this visualization object.
- Master layout: The object will be used as Master Layout. Per default the master layout always will be in the foreground of a visualization, except the option as background is activated.



Database-connection: If the project is connected to an ENI data base, then a tab 'Database-connection' will be available *Chapter 1.4.1.2.2.13 “Options for 'Project source control'” on page 216*. Here you can display and modify the current assignment of the object to one of the data base categories resp. to the category 'Local'.

'Project' 'Object access rights'

With this command you open the dialog box for assigning access rights to the different 'user groups' *Chapter 1.4.1.2.3.42 “User groups” on page 249*. The following dialog box appears:



Members of the user group 0 can now assign individual access rights for each user group. There are three possible settings:

- No Access: the object may not be opened by a member of the user group.
- Read Access: the object can be opened for reading by a member of the user group but not changed.
- Full Access: the object may be opened and changed by a member of the user group.

The settings refer either to the currently-selected object in the 'Object Organizer' or, if the option 'Apply to all' is chosen, to all POUs, data types, visualizations, and resources of the project ↳ *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*.

The assignment to a user group takes place when opening the project through a password request if a password was assigned to the user group 0.

Please regard also the possibility to assign access rights concerning the operation of visualization elements (Visualization, Security).

'Project' 'Add action'

This command is used to generate an action allocated to a selected block in the 'Object Organizer' ↳ *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*. One selects the name of the action in the dialog which appears and also the language in which the action should be implemented.

The new action is placed under your block in the Object Organiser. A plus sign appears in front of the block. A simple mouse click on the plus sign causes the action objects to appear and a minus sign appears in front of the block. Renewed clicking on the minus sign causes the actions to disappear and the plus sign appears again. This can also be achieved over the context menu commands 'Expand Node' and 'Collapse Node' ↳ *Chapter 1.4.1.2.4.4 “'Expand nodes' 'Collapse nodes'” on page 258*.

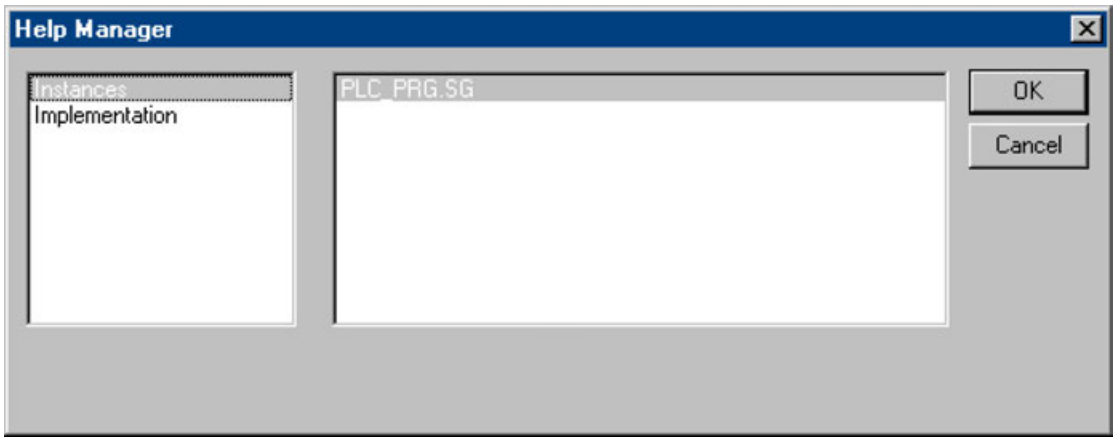
See also:

Calling actions ↳ *Chapter 1.4.1.1.9.9 “Action” on page 160*

Action in the Sequential Function Chart ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*

'Project' 'View instance'

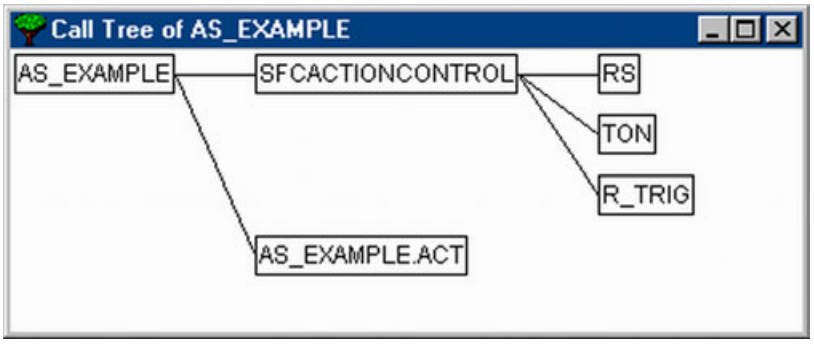
With this command it is possible to open and display the instance of the function block which is selected in the Object Organizer. In the same manner, a double click on the function block in the Object Organizer gives access to a selection dialog in which the instances of the function block as well as the implementation are listed. Select here the desired instance or the implementation and confirm using OK. The desired item is then displayed in a window.



If you want to view instances, you first have to log in! (The project has been compiled with no errors and downloaded to the PLC with 'Online' 'Login').

'Project' 'Show call tree'

With this command you open a window which shows the call tree of the object chosen in the 'Object Organizer' ↳ *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*. Before this the project must have been compiled without any error ↳ *Chapter 1.4.1.2.3.12 “'Project' 'Rebuild all'” on page 232*. The call tree contains both calls for POUs and references to data types.



'Project' 'Show cross reference'

With this command you open a dialog box which makes possible the output of all application points for a variable, address, or a POU. For this the project must be compiled ↳ *Chapter 1.4.1.2.3.11 “'Project' 'Build'” on page 231*.

At 'name' enter the name (e.g. "ivar") or path (e.g. "PLC_PRG.fbi.ivar") of the element for which you want to get listed the cross references (input assistant <F2> can be used). If you specify "*", all elements will be regarded. Also just a partial string followed by "*", e.g. "INT_*" can be specified in order to get listed all elements, the names of which start with this string.

If the project has been changed since the last compile, the term "(Not up to date)" will be displayed in the title bar of the dialog. In this case any cross references which have been created recently will not be regarded in the list unless you do a re-compile!

By clicking on the button 'Cross References' you get the list of all application points. Along with the POU and the line or network number, the variable name and the address binding, if any, are specified.

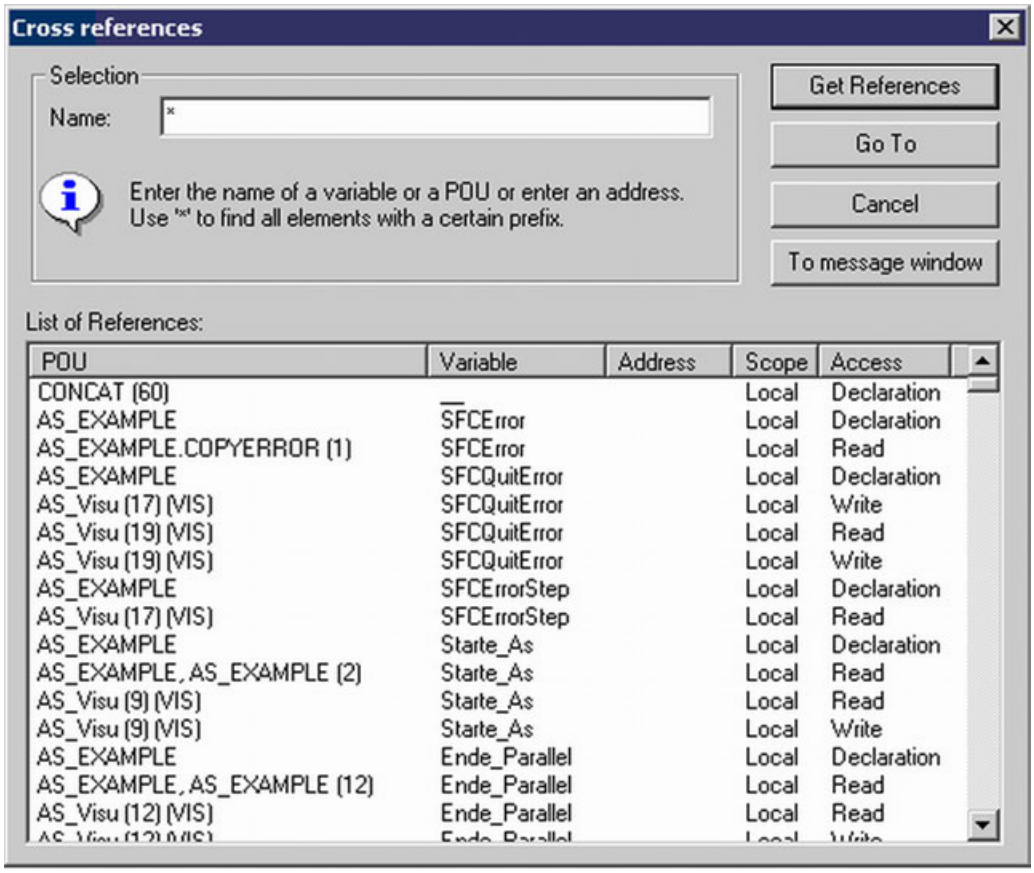
The Domain space shows whether this is a local or a global variable; the Access column shows whether the variable is to be accessed for 'reading' or 'writing' at the current location.

If the element is used within the address range of a structure or an array, this position will also be listed (Example: A variable wVar of type WORD is assigned to %MW2. A variable arrVar of type ARRAY [0..2] OF WORD is assigned to %MW0. If you call the cross-reference list for wVar, arrVar[2] will be found). The same is true for structures.

If an element is used within a visualization, the visualization name will be shown in column "POU". However note the following concerning placeholders within a visualization: The cross-reference list only regards variable name strings, which are already entered in the configuration of a visualization element, not however any names, which are generated during compilation of the project due to placeholder replacements !!

When you select a line of the cross-reference list and press the button 'Go To' or doubleclick on the line, then the POU is shown in its editor at the corresponding point. In this way you can jump to all application points without a time-consuming search.

In order to make processing easier, you can use the 'Send to message window' button to bring the current cross-reference list into the message window and from there change to the respective POU ↳ *Chapter 1.4.1.2.1.6 “Message window” on page 200*.



Direct call of a cross-reference list out of a POU editor or a watchlist:

The cross-reference list can be generated directly out of the editor (ST, FUP, KOP, AWL, CFC, AS), which is used to work on a POU, or out of a watchlist ↳ *Chapter 1.4.1.4.9.1 “Overview” on page 395*. In this case in online as well as in offline mode the command 'Show cross references' is available in the context or "Extras" menu, when a variable is selected in the editor or the watchlist ↳ *Chapter 1.4.1.3.7 “Show cross references” on page 295*.

'Project' 'Project database'

Overview

This menu item is only available if you have activated the option 'Use source control (ENI)' in the project options dialog for category 'Project source control'. A submenu is attached where you find the following commands for handling the object resp. the project in the currently connected ENI data base:

Login (The user logs in to the ENI Server)

If an object is marked in the Object Organizer and the command Data Base Link is executed (from the context menu, right mouse button), then several commands will be available for executing the corresponding data base actions. If the user had not logged in successfully to the ENI Server before, then the dialog 'Data base Login' will open automatically and the chosen command will not been executed until the login was successful.

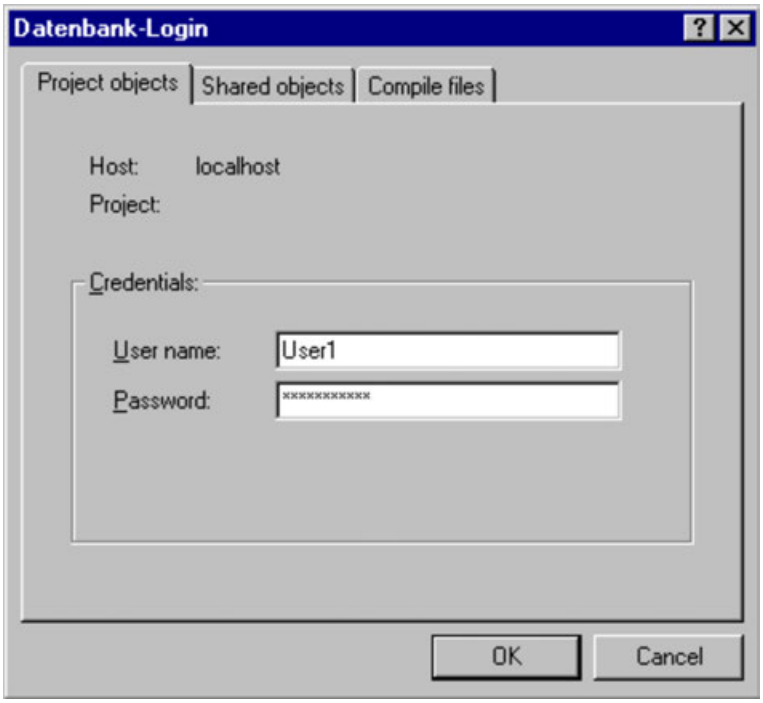
If the command 'Data Base Link' in the 'Project' menu is activated, then additional menu items will be available, which concern all objects of the project.

How the status of an object's handling in the data base is displayed in the Object Organizer:

Grey shaded icon: Object is stored in the data base (source control) Green check in front of the object name: Object is checked out in the currently opened project. Red cross in front of the object name: Object is currently checked out by another user. <R> behind object name: The object can only be read, but not edited. Please regard: some objects (Task configuration, Sampling Trace, PLC Configuration, Target Settings, Watch- and Receipt Manager) are per default assigned with a <R> as long as they are not checked out. This means that you will not automatically be asked whether the object should be checked out, as soon as you start to edit the object; it not necessarily means that you cannot edit the object. If there is no write access then the command 'Check out' will not be available.

Login

This command will open the dialog 'Login' where you can enter the access data for the ENI data base via the ENI Server. The access data also have to be defined in the ENI Server (ENI Admin, User Management) and – depending on the currently used data base – also in the user management of the data base. After the command has been executed, first the Login dialog for category 'Project objects' will open.



The following items are displayed:

- Data base: project objects
- Host: address of the computer where the ENI Server is running (must match with the entry in field 'TCP/IP address' in the project options dialog for 'Project source control').
- Project: Name of the data base project (must match with the entry in field 'Project name' in the project options dialog for 'Project source control'/category 'Project Objects').

Credentials:

- Insert username and password.
- When option Use as default for this project is activated, then the above entered access data will automatically be used for any further communication between the actual project and the data base concerning objects of the actual category.
- Press OK to confirm the settings. The dialog will be closed and automatically the Login dialog for 'Shared objects' will open. Enter the access data in the same way as described for the 'Project objects' and confirm with OK. Do the same in the third Login dialog which will be opened for category 'Compile files'.
- The Login dialog will always open as soon as you try to access the data base before having logged in successfully as described above.

If you want to save the access data with the project, activate option 'Save ENI credentials' in the project options, category 'Load & Save'.

Define

Command: 'Project' 'Data Base Link' 'Define'

Here you can define, whether the object which is currently marked in the Object organizer should be kept in the data base or just locally in the project. A dialog will open, where you can choose one of the two database categories 'Project' or 'Shared objects', or the category 'Local'.

The icons of all objects which are managed in the data base will be displayed grey-shaded in the Object organizer. Shared objects are displayed with turquoise letters.

Get latest version

Command: 'Project' Data Base Link'Get Latest Version'

The current version of the object which is marked in the Object organizer will be copied from the data base and will overwrite the local version. In contrast to the Check Out action the object will not be locked for other users in the data base *Chapter 1.4.1.2.4.18.5 “Check out” on page 268*.

Check out

Command: 'Project' 'Data Base Link' 'Check Out'

The object which is marked in the Object organizer will be checked out from the data base and by that will be locked for other users.

When executing the command the user will get a dialog 'Check out object'. A comment can be added there which will be stored in the version history of the object in the data base. Line breaks are inserted by <Ctrl>+<Enter>. If the version of the object differs from that in the local project, an appropriate message will be displayed and the user can decide whether the object should be checked out anyway.

After the dialog has been closed with OK, the checked-out object will be marked with a green check in the object organizer of the local project. For other users it will be appear marked with a red cross and will not be editable by them.

Check in

Command: 'Project' 'Data Base Link' 'Check In'

The object which is marked in the Object organizer will be checked in to the data base. Thereby a new version of the object will be created in the data base. The old versions will be kept anyway.

When executing the command the user will get a dialog 'Check in object'. There a comment can be added which will be stored in the version history of the object in the data base. Line breaks are inserted by <Ctrl>+<Enter>.

After the dialog has been closed with OK the green check in front of the object name in the Object organizer will be removed.

Undo check out

Command: 'Project' 'Data Base Link' 'Undo Check Out'

Use this command to cancel the Checking out of the object which is currently marked in the Object organizer. Thereby also the modifications of the object which have been made locally, will be canceled. No dialog will appear. The unchanged last version of the object will be kept in the data base and it will be accessible again for other users. The red cross in front of the object name in the Object organizer will disappear.

Show differences

Command: 'Project' 'Data Base Link' 'Show Differences'

The object which is currently open will be displayed in a window which is divided up in two parts. There the local version, which is currently edited by the local user, will be opposed to the last (actual) version which is kept in the data base. The differences of the versions will be marked like described for the project comparison (see 'Project' 'Compare').

Show version history

Command: 'Project' 'Data Base Link' 'Show Version History'

For the currently marked object in the Object organizer a dialog Version history of <object name> will be opened. There all versions of the object are listed which have been checked in to the data base or which have been labeled there:

The following information is given:

Version: Data base specific numbering of the versions of the object which have been checked in one after the other. Labeled versions get no version number but are marked by a label-icon.

User: Name of the user, who has executed the check-in or labeling action

Date: Date and time stamp of the action

Action: Type of the action which has been executed. Possible types: 'created' (the object has been checked in to the data base for the first time), 'checked in' (all check-ins of the object excluding the first one) and labeled with <label> (a label has been assigned to this version of the object)

The buttons:

Close: The dialog will be closed.

Display: The version which is currently marked in the table will be opened. The title bar shows: ENI: <name of the project in the data base>/<object name>

Details: The dialog 'Details of Version History' will open:

File (name of the project and the object in the data base), Version (see above), Date (see above), User (see above), Comment (Comment which has been inserted when the object has been checked in resp. has been labeled). Use the buttons Next resp. Previous to jump to the details window of the next or previous entry in the table in dialog 'Version history of ..'.

Get latest version: The version which is marked in the table will be loaded and there will overwrite the local version.

Differences: If in the table only one version of an object is marked, then this command will cause a comparison of this version with the latest (actual) data base version. If two versions are marked, then those will be compared. The differences are displayed in a bipartited window like it is done at the project comparison.

Reset version: The version which is marked in the table will be set as latest version. All versions which have been checked in later will be deleted ! This can be useful to restore an earlier status of an object.

Labels only: If this option is activated, then only those versions of the object will be displayed in the table, which are marked by a label.

Selection box below the option 'Labels only': Here you find the names of all users which have executed any data base actions for objects of the current project. Select 'All' or one of the names if you want to get the version history concerning all users or just for a certain one.

Multiple define

Command 'Project' 'Data Base Link' 'Multiple Define'

Use this command if you want to assign several objects at a single blow to a certain data base category. The dialog 'Properties' will open like described for command 'Define'. Choose the desired category and close the dialog with OK. After that the dialog 'ENI-Selection' will open, listing all POUs of the project which are considered for the chosen category (Example: if you choose category 'shared objects' then the selection window will only offer the POUs of the Resources tab). The POUs are presented in a tree structure complying to that of the Object Organizer. Select the desired POUs and confirm with OK.

Get all latest versions

Command 'Project' 'Data Base Link' 'Get All Latest Versions'

The latest version of each object of the currently opened project, which is kept under source control, will be called from the data base. Consider the following:

- If in the meantime additional objects have been stored to the data base project folder, then those will now be added to the local project.
- If objects have been deleted in the data base in the meantime, those will not be deleted in the local project, but they will automatically get assigned to category 'Local'.
- The latest version of objects of category 'Shared Objects' will only be called, if these objects are already available in the local project. For further information see command 'Get latest version' *Chapter 1.4.1.2.4.18.4 “Get latest version” on page 268*.

Multiple check out

Command 'Project' 'Data Base Link' 'Multiple Check Out'

You can check out several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those which should be checked out and confirm with OK. For further information see command 'Check Out'.

Multiple check in

Command 'Project' 'Data Base Link' 'Multiple Check In'

You can check in several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those which should be checked in and confirm with OK. For further information see command 'Check In'.

Multiple undo check out

Command 'Project' 'Data Base Link' Undo Multiple Check Out'

You can undo the check out action for several objects at a single blow. For this the dialog 'ENI-Selection' will open, listing all POUs of the project. Select those for which you want to cancel the check out and confirm with OK. For further information see command 'Undo Check Out'.

Project version history

Command 'Project' 'Data Base Link' 'Project Version History

If the chosen data base system supports that functionality, you can use this command to view the version history for the currently opened project.

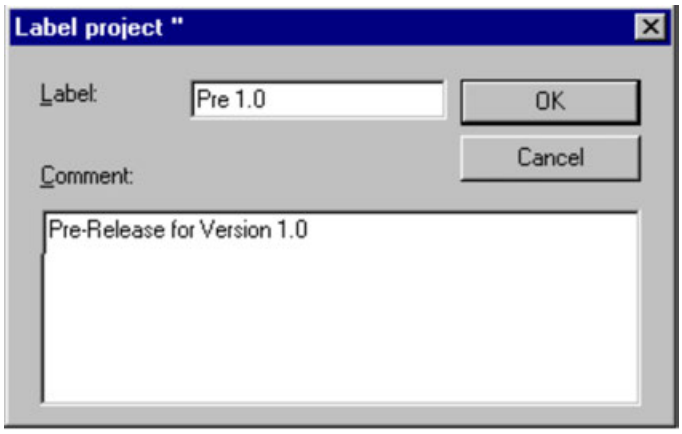
The dialog 'History of <data base project name>' will open. It shows the actions (create, check in, label) which have been performed for the particular objects of the project in a chronological order. The total number of objects is displayed behind Version history. The dialog can be handled like described for command 'Show Version History', but regard the following:

- The command 'Reset Version' is only available for single objects.
- The command 'Get latest version' means that all objects of the version of the currently marked object will be called to the local project! That means, that the objects in CODESYS will be overwritten with the older version. But: Local objects, which were not yet part of the project in that older version, will not be removed from the local project ! If a labeled version is called, which contains Shared Objects, then the user will get a dialog where he can decide whether those Shared Objects should be called also or not.

Label version

Command 'Project' 'Data Base Link' 'Label Version'

This command is used to put a "label" on the actual version of each object of a project, so that exactly this project version can be recalled later. A dialog 'Label <data base project name>' will open. Insert a label name (Label) (e.g. "Release Version") and optionally a Comment. When you confirm with OK, the dialog will close and the label and the action "labeled with <label name>" will appear in the table of the version history, as well in the history for a single object as in the history of the project. Shared Objects which are part of the project will also get that label. A labeled version of the project does not get a version number, but is just marked with a label icon in the column 'Version'. If the option 'Labels only' is activated in the Version History dialog, then only labeled versions will be listed.

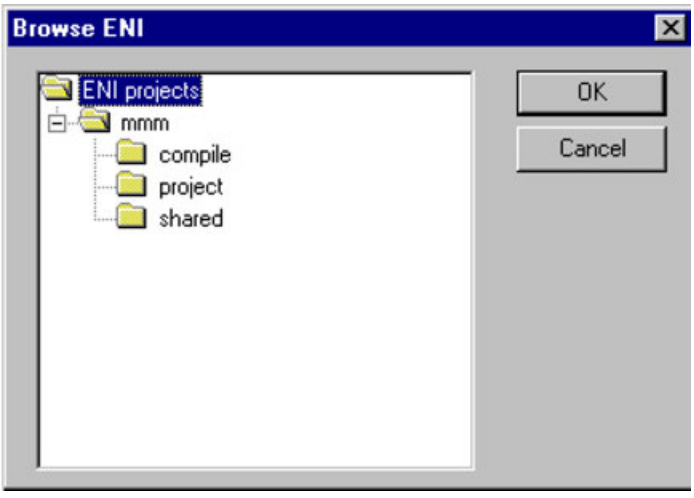


Add shared objects

Command 'Project' 'Data Base Link' 'Add Shared Objects'

Use this command if you explicitly want to add new objects of data base category 'Shared Objects' to the locally opened project. For objects of category 'Project Objects' this is not necessary, because the command 'Get (all) latest version(s)' automatically calls all objects which are found in the data base project folder, even if there are some which not yet available in the local project. But for objects of category 'Shared Objects' in this case just those objects will be called which are already available in the local project.

So execute the command 'Add Shared Objects' to open the dialog 'Browse ENI'. A list in the right part of the window shows all objects which are available in the data base folder which is currently selected in the list on the left side. Choose the desired object and press OK or do a doubleclick on the entry to insert the object to the currently opened project.



Refresh status

Command 'Project' 'Data Base Link' 'Refresh Status'

Use this command to update the display in the Object Organizer, so that you can see the actual status of the objects concerning the source control of the project.

1.4.1.2.5 General editing functions

Overview

You can use the following commands in all editors and some of them in the Object Organizer. The commands are located under the menu item 'Edit' and in the context menu that is opened with the right mouse button.

If the IntelliPoint-Software is installed on the computer, all functions of the MS IntelliMouse are supported. In all editors with zoom functionality: To magnify press the <Strg> key while rolling the wheel of the mouse, to reduce roll backwards while the <Ctrl> key is pressed.

See also:

↳ *Chapter 1.4.1.2.5.2 “'Edit' 'Undo'” on page 272*
↳ *Chapter 1.4.1.2.5.3 “'Edit' 'Redo'” on page 273*
↳ *Chapter 1.4.1.2.5.4 “'Edit' 'Cut'” on page 273*
↳ *Chapter 1.4.1.2.5.5 “'Edit' 'Copy'” on page 273*
↳ *Chapter 1.4.1.2.5.6 “'Edit' 'Paste'” on page 274*
↳ *Chapter 1.4.1.2.5.7 “'Edit' 'Delete'” on page 274*
↳ *Chapter 1.4.1.2.5.8 “'Edit' 'Find'” on page 275*
↳ *Chapter 1.4.1.2.5.9 “'Edit' 'Find next'” on page 275*
↳ *Chapter 1.4.1.2.5.10 “'Edit' 'Replace'” on page 275*
↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*
↳ *Chapter 1.4.1.2.5.12 “Unstructured display” on page 276*
↳ *Chapter 1.4.1.2.5.13 “Stuctured display” on page 277*
↳ *Chapter 1.4.1.2.5.14 “'Edit' 'Autodeclare'” on page 278*
↳ *Chapter 1.4.1.2.5.15 “'Edit' 'Next error'” on page 278*
↳ *Chapter 1.4.1.2.5.16 “'Edit' 'Previous error'” on page 278*
↳ *Chapter 1.4.1.2.5.17 “'Edit' 'Macros'” on page 278*

'Edit' 'Undo'

Shortcut: <Ctrl>+<Z>

This command undoes the action which was most recently executed in the currently-open editor window or in the Object Organizer; repeated use undoes all actions back to the time that the window was opened. This applies to all actions in the editors for POUs, data types, visualizations and global variables and in the Object Organizer.

With 'Edit' 'Redo' ↳ *Chapter 1.4.1.2.5.3 “'Edit' 'Redo'” on page 273* you can restore an action which you have undone.

The commands Undo and Redo apply to the current window. Each window carries its own action list. If you want to undo actions in several windows, then you must activate the corresponding window. When undoing or redoing in the Object Organizer the focus must lie here.

'Edit' 'Redo'

Shortcut: <Ctrl>+<Y>

With the command in the currently-open editor window or in the 'Object Organizer' you can restore an action you have undone *Chapter 1.4.1.2.1.3 “Object organizer” on page 199* *Chapter 1.4.1.2.5.2 “'Edit' 'Undo'” on page 272*.

As often as you have previously executed the command 'Undo' , you can also carry out the command 'Redo'.

The commands 'Undo' and 'Redo' apply to the current window. Each window carries its own action list. If you want to undo actions in several windows, then you must activate the corresponding window. When undoing or redoing in the Object Manager must lie there.

'Edit' 'Cut'

Symbol:  Shortcut: <Ctrl>+<X> or <Shift>+<Delete>

This command transfers the current selection from the editor to the clipboard. The selection is removed from the editor.

In the 'Object Organizer' this similarly applies to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration *Chapter 1.4.1.2.1.3 “Object organizer” on page 199* *Chapter 1.4.1.2.5.10 “'Edit' 'Replace'” on page 275*.

Remember that not all editors support the cut command, and that its use can be limited in some editors.

The form of the selection depends upon the respective editor:

In the text editors IL, ST, and declarations the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle *Chapter 1.4.1.3.11.9.1 “Overview” on page 330*.

In order to paste the content of the clipboard you use the command 'Edit' 'Paste' *Chapter 1.4.1.2.5.6 “'Edit' 'Paste'” on page 274*. In the SFC editor you can also use the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after' *Chapter 1.4.1.1.9.4 “Function block” on page 153* *Chapter 1.4.1.3.11.9.17 “'Extras' 'Paste after'” on page 333*.

In order to remove a selected area without changing the clipboard, use the command 'Edit' 'Delete' *Chapter 1.4.1.2.5.7 “'Edit' 'Delete'” on page 274*.

'Edit' 'Copy'

Symbol:  Shortcut: <Ctrl>+<C>

This command copies the current selection from the editor to the clipboard. This does not change the contents of the editor window.

In the 'Object Organizer' this similarly applies to the selected object, whereby not all objects can be deleted, e.g. the PLC configuration *Chapter 1.4.1.2.1.3 “Object organizer” on page 199* *Chapter 1.4.1.2.5.10 “'Edit' 'Replace'” on page 275*

Remember that not all editors support copying and that it can be limited with some editors.

For the type of selection the same rules apply as with 'Edit' 'Cut'.

The form of the selection depends upon the respective editor:

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field or a box with all preceding lines, boxes, and operands.

In the SFC editor the selection is a part of a series of steps steps surrounded by a dotted rectangle *Chapter 1.4.1.3.11.9.1 “Overview” on page 330*.

In order to paste the content of the clipboard you use the command 'Edit' 'Paste' *Chapter 1.4.1.2.5.6 “'Edit' 'Paste'” on page 274*. In the SFC editor you can also use the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after' *Chapter 1.4.1.1.9.4 “Function block” on page 153* *Chapter 1.4.1.3.11.9.17 “'Extras' 'Paste after'” on page 333*.

In order to delete a selected area and simultaneously put it on the clipboard, use the command 'Edit' 'Cut' *Chapter 1.4.1.2.5.4 “'Edit' 'Cut'” on page 273*.

'Edit' 'Paste'

Symbol: Shortcut: <Ctrl>+<V>

Pastes the content of the clipboard onto the current position in the editor window. In the graphic editors the command can only be executed when a correct structure results from the insertion *Chapter 1.4.1.3.11.1 “Overview” on page 314*.

With the object organizer the object is pasted from the clipboard *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*.

Remember that pasting is not supported by all editors and that its use can be limited in some editors.

The current position can be defined differently according to the type of editor:

With the text editors (and declarations) the current position is that of the blinking cursor (a vertical line) which you place by clicking with the mouse *IL, ST,*.

In the FBD and LD editors the current position is the first network with a dotted rectangle in the network number area *Chapter 1.4.1.3.11.7.1 “Overview” on page 317* *Chapter 1.4.1.3.11.8.1 “Overview” on page 323*. The contents of the clipboard are inserted in front of this network. If a partial structure has been copied, then it is inserted in front of the selected element.

In the SFCeditor the current position is determined the selection which is surrounded by a dotted rectangle *Chapter 1.4.1.3.11.9.1 “Overview” on page 330*. Depending upon the selection and the contents of the clipboard, these contents are inserted either in front of the selection or into a new branch (parallel or alternative) to the left of the selection.

In SFC the commands 'Extras' 'Insert parallel branch (right)' or 'Extras' 'Paste after' can be used in order to insert the contents of the clipboard *Chapter 1.4.1.1.9.4 “Function block” on page 153* *Chapter 1.4.1.3.11.9.17 “'Extras' 'Paste after'” on page 333*.

In order to copy a selection onto the clipboard without deleting it, use the command 'Edit' 'Copy' *Chapter 1.4.1.2.5.5 “'Edit' 'Copy'” on page 273*.

In order to remove a selected area without changing the clipboard, use the command 'Edit' 'Delete' *Chapter 1.4.1.2.5.7 “'Edit' 'Delete'” on page 274*.

'Edit' 'Delete'

Shortcut: <Del>

Deletes the selected area from the editor window. This does not change the contents of the clipboard.

In the 'Object Organizer' this similarly applies to the selected object, whereby not all objects can be deleted, e.g. the PLC Configuration *Chapter 1.4.1.2.1.3 “Object organizer” on page 199* *Chapter 1.4.1.2.5.10 “'Edit' 'Replace'” on page 275*.

For the type of selection the same rules apply as with 'Edit' 'Cut'.

The form of the selection depends upon the respective editor:

In the text editors (IL, ST, and declarations) the selection is a list of characters.

In the FBD and LD editors the choice is a number of networks which are indicated by a dotted rectangle in the network number field.

In the SFC editor the selection is a part of a series of steps surrounded by a dotted rectangle ↳ *Chapter 1.4.1.3.11.9.1 “Overview” on page 330*.

In the 'library manager' the selection is the currently selected library name ↳ *Chapter 1.4.1.4.3.1 “Overview” on page 371*.

In order to delete a selected area and simultaneously put it on the clipboard, use the command 'Edit' 'Cut' ↳ *Chapter 1.4.1.2.5.4 “'Edit' 'Cut'” on page 273*.

'Edit' 'Find'

Symbol: 

With this command you search for a certain text passage in the current editor window. The Find dialog box opens. It remains opens until the button Cancel is pressed.

In the field 'Find what' you can enter the series of characters you are looking for.

In addition, you can decide whether the text you are looking for 'Match whole word only' or not, or also whether 'Match case' is to be considered, and whether the search should proceed 'Up' or 'Down' starting from the current cursor position.

The button 'Find next' starts the search which begins at the selected position and continues in the chosen search direction. If the text passages is found, then it is highlighted. If the passage is not found, then a message announces this. The search can be repeated several times in succession until the beginning or the end of the contents of the editor window has been reached. In the CFC editor the geometrical order of the elements will be regarded, the search will run from the left upper corner of the window to the right upper corner. Please regard that FBD POUs are processed from the right to the left!

Find dialog box:

'Edit' 'Find next'

Symbol:  Shortcut: <F3>

With this command you execute a search with the same parameters as with the most recent action 'Edit' 'Find' ↳ *Chapter 1.4.1.2.5.8 “'Edit' 'Find'” on page 275*. Please note that FBD POUs are processed from the right to the left!

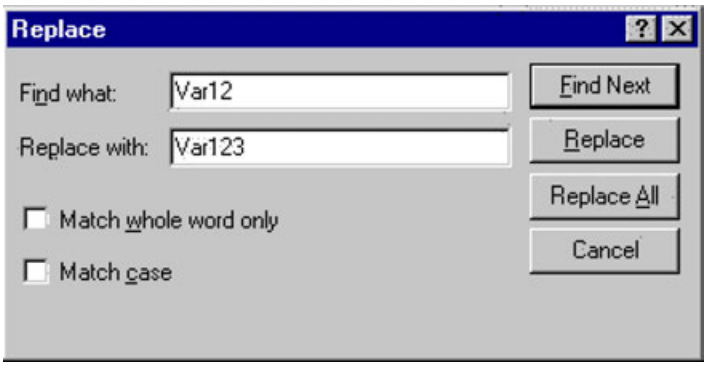
'Edit' 'Replace'

With this command you search for a certain passage just as with the command 'Edit' 'Find', and replace it with another ↳ *Chapter 1.4.1.2.5.8 “'Edit' 'Find'” on page 275*. After you have chosen the command the dialog box for find and replace appears. This dialog box remains open until the button 'Cancel' or 'Close' is pressed.

In the field behind 'Find' automatically that string will be inserted which you have marked before in the editor. You also can enter the search string manually. Pressing button 'Replace' will replace the current selection with the string which is given in the field 'Replace with'. Use the button 'Find Next' to get to the next passage where the string is found. Please regard, that FBD POUs are processed from the right to the left!

The button 'Replace all' replaces every occurrence of the text in the field 'Find next' after the current position with the text in the field 'Replace with'. At the end of the procedure a message announces how many replacements were made.

Dialog box for find and replace:



'Edit' 'Input assistant'

Shortcut: <F2>

This command provides a dialog box for choosing possible inputs at the current cursor position in the editor window. In the left column choose the desired input category, select the desired entry in the right column, and confirm your choice with OK. This inserts your choice at this position.

The categories offered depend upon the current cursor position in the editor window (e.g. variables, operators, POUs, conversions, watch variables etc.).

If the option 'With arguments' is active, then when the selected element is inserted, the arguments to be transferred are specified with it, for example: function block fu1 selected, which defines the input variable var_in: fu1(var_in:=);

Insertion of function func1, which uses var1 and var2 as transfer parameters: func1(var1,var2)

It is basically possible to switch between structured and unstructured display of the available elements ↳ *Chapter 1.4.1.2.5.13 “Stuctured display” on page 277* ↳ *Chapter 1.4.1.2.5.12 “Unstructured display” on page 276*. This occurs through activation/deactivation of the 'Structured Display' option.

For inserting identifiers you also can use the intellisense functionality ↳ *Chapter 1.4.1.3.6 “Intellisense function” on page 295.*

Unstructured display

In the left part of the window always those categories of elements are listed which are relevant for the current insert position,. e.g. ST Operators, Local Variables, Global Variables, Standard Programs, Defined Programs, Watch Expressions etc. In the right part of the window the elements of the selected category are listed simply linearly sorted in alphabetical order.

At various places (e.g. in the Watch List), multi-stage variable names are required. In that event the input assistant displays the available variables with a preceded POU name, resp. in case of structured variables and function block instances additionally with the FB name resp. data type name. Examples: PLC_PRG.ivar, PLC_PRG.Structure1.Component1. For global variables no POU name is added.

The desired element must be selected and then will be inserted at the insert position by OK.

When the non-structured input assistant is used in the Watch- and Recipe Manager, then the range of offered watch variables (Watch Expressions) can be reduced by a filter.

You can switch to structured display through activation of the Structured Display

Stuctured display

If Structured display is selected, the POUs, variables or data types will be sorted hierarchically. This is possible for standard programs, standard functions, standard function blocks, defined programs, defined functions, defined function blocks, global variables, local variables, defined types, watch variables. The visual and hierarchical display corresponds to that of the 'Object Organizer'; if elements in a library are referred to, these are inserted in alphabetical order at the very top and the pertinent hierarchy is displayed as in the Library Manager.

The in- and output variables of function blocks which are declared as local or global variables are listed in the category 'Local Variables' or 'Global Variables' under the instance name (e.g. Inst_TP ET, Inst_TP IN,...). To get there, select the instance name (e.g. Inst_TP) and confirm with OK.

If the instance of a function block is selected here, the option With arguments may be selected. In the text languages ST and IL as well as during task configuration, the instance name and the input parameters of the function block are then inserted.

For example, if Inst (DeklarationInst: TON;) is selected, the following is inserted:

Inst(IN:= ,PT:=)

If the option is not selected, only the instance name will be inserted. In the graphical languages or in the Watch window, only the instance name is generally inserted.

Components of structures are displayed in an analog fashion to function block instances.

For enumerations, the individual enumeration values are listed under the enumeration type. The order is: enumerations from libraries, enumerations from data types, local enumerations from POUs.

The general rule is that lines containing sub-objects are not selectable (except instances, see above), but can only have their hierarchy display expanded or contracted by one level, as for multi-stage variable names.

If input assistant is invoked in the Watch and Receipt Manager or in the selection of trace variables in the trace configuration dialog, it is possible to make a multiple selection. When the <Shift> key is pressed, you can select a range of variables; when the <Ctrl> key is pressed you can select many individual variables. The selected variables are so marked. If, during range selection lines are selected that do not contain valid variables (e.g. POU names), these lines will not be included in the selection. When individual selections are made, such lines can not be marked.

In the watch window and in trace configuration it is possible to transfer structures, arrays or instances from the Input Assistant dialog. As a double click with the mouse button is associated with the extension or contraction of the element's hierarchy display, selection in these cases can only be confirmed by OK.

Thereafter, the selected variables are inserted line by line in the watch window, that is each selected variable is written on a separate line. In the case of trace variables, each variable is inserted in a separate line of the trace variables list.

If the maximum number of trace variables, 20, is exceeded during insertion of the selected variables, the error message "A maximum of 20 variables is allowed" appears. Further selected variables are then not inserted in the list.

Some entries (e.g. Global Variables) are only updated in the input assistant dialog after compilation.

You can switch to unstructured display through deactivation of option Structured.

'Edit' 'Autodeclare'

Shortcut: <Shift>+<F2>

This command opens the dialog for the declaration of a variable *Chapter 1.4.1.3.9.11 “Variables declaration” on page 303*. This dialog also opens automatically when the option 'Project' 'Options' 'Editor' 'Autodeclaration' is switched on and when a new undefined variable is used the declaration editor *Chapter 1.4.1.3.9.17 “Autodeclaration” on page 306*.

'Edit' 'Next error'

Shortcut: <F4>

After the incorrect compilation of a project this command can show the next error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown *Chapter 1.4.1.2.1.6 “Message window” on page 200*.

'Edit' 'Previous error'

Shortcut: <Shift>+<F4>

After the incorrect compilation of a project this command shows the previous error. The corresponding editor window is activated and the incorrect place is selected. At the same time in the message window the corresponding error message is shown *Chapter 1.4.1.2.1.6 “Message window” on page 200*.

'Edit' 'Macros'

This menu item leads to a list of all macros, which are defined for the project. (For info on generating macros see 'Project' 'Options' 'Macros' *Chapter 1.4.1.2.2.16 “Options for 'Macros'” on page 220*). When an executable macro is selected the dialog 'Process Macro'. The name of the macro and the currently active command line are displayed. The button Cancel can be used to stop the processing of the macro. In that event the processing of the current command will be finished anyway. Then an appropriate message is displayed in the message window and in the log during Online operation: "<Macro>: Execution interrupted by user".

Macros can be executed offline and online, but in each case only those commands are executed which are available in the respective mode.

1.4.1.2.6 General online functions

Overview

The available online commands are assembled under the menu item 'Online'. The execution of some of the commands depends upon the active editor.

The online commands become available only after logging in. See the book 'General Online Functions..' in the Contents tab of this online help.See the following chapters for a description.

Thanks to 'Online Change' functionality you have the possibility of making changes to programs on the running controller. See in this connection 'Online' 'Login' ↳ *Chapter 1.4.1.2.6.2 "'Online' 'Login'" on page 279*.

See 'Online' 'Login' for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login ↳ *"Relations between Login - Build - Download - Online Change" on page 280*.

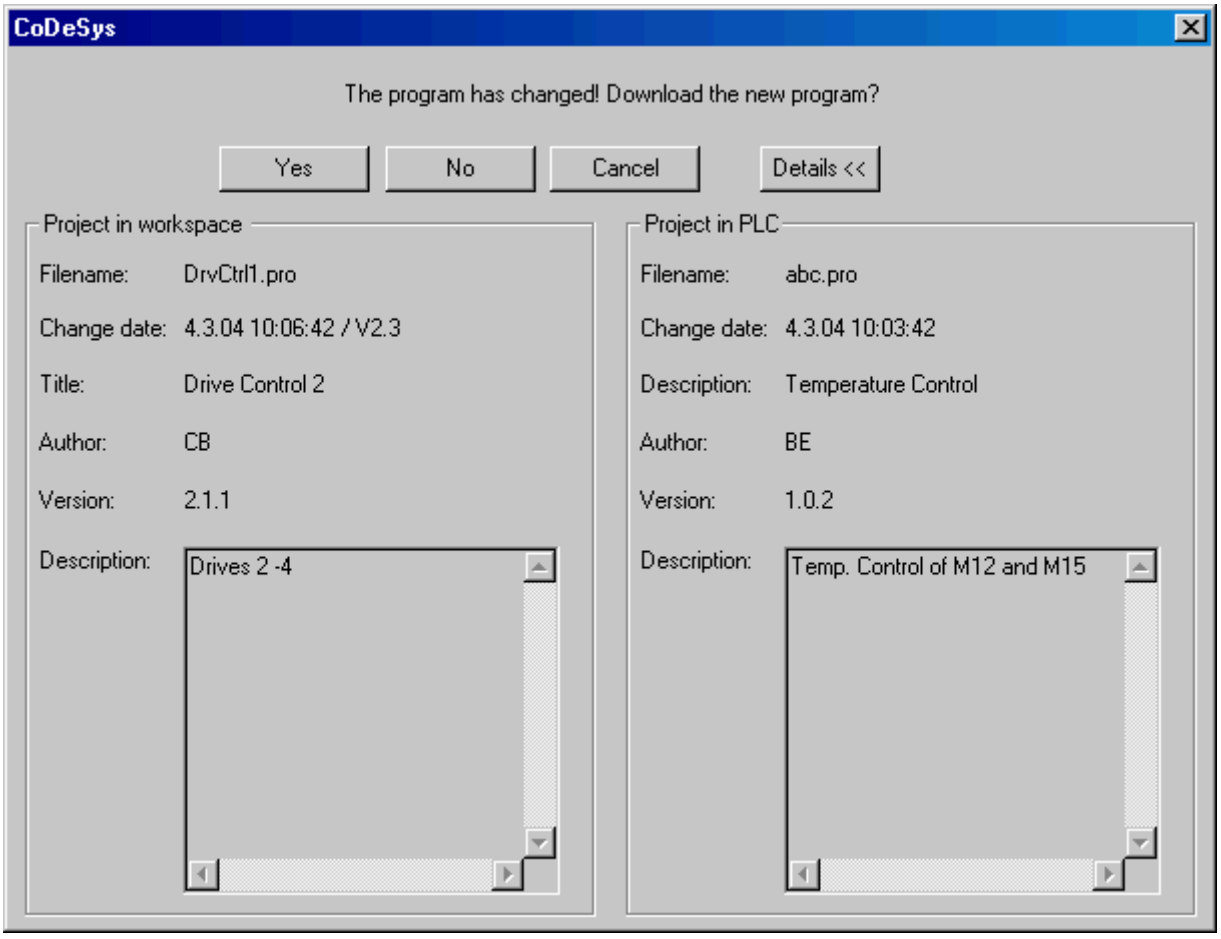
'Online' 'Login'

Symbol: Shortcut: <Alt>+<F8>

This command combines the programming system with the PLC (or starts the simulation program) and changes into the online mode ↳ *Chapter 1.4.1.1.11.8 "Simulation" on page 184*.

If the current project has not been compiled since opening or since the last modification, then it is compiled now ↳ *Chapter 1.4.1.2.3.11 "'Project' 'Build'" on page 231*. If errors occur during compilation, then Automation Builder does not change into Online mode.

If the current project was changed on the controller since the last download, but not closed, and if the last download information was not deleted with the command 'Project' 'Clear all', then after the command 'Login' a dialog opens with the question ↳ *Chapter 1.4.1.2.6.5 "'Online' 'Download'" on page 283*: "The program has been changed. Load changes? (Online Change)". By answering 'Yes' you confirm that, on log-in, the modified portions of the project are to be loaded onto the controller. (Concerning this see below a diagram on the relations between Project-Build, Project-Download, Online Change and Login and also some hints on Online Change). 'No' results in a log-in without the changes made since the last download being loaded onto the controller. 'Cancel' cancels the command. <Load all> causes the entire project to be reloaded onto the controller.



If in the project options, category Desktop, the option 'Online in security mode' is activated and if the target system supports the functionality, in the Login dialog automatically also the Project information will be displayed. This is the project information of the project which is currently opened in Automation Builder and which is already available on the controller. Via button 'Details <<' you can close this information part of the dialog. If the 'Online in security mode' option is not activated, you can explicitly open the project information display in the dialog via button 'Details >>'.

It depends on the target which button is set as default button.

Online Change does not cause a re-initialization of the variables, thus modifications of the initialization values will not be regarded ! Retain variables keep their values when an Online Change is done, they won't do that at a re-download of the project (see below, 'Online' 'Download').

After a successful login all online functions are available (if the corresponding settings in 'Project' 'Options' category have been entered ↳ *'Build'*). The current values are monitored for all visible variable declarations ↳ *Chapter 1.4.1.3.9.11 “Variables declaration” on page 303*.

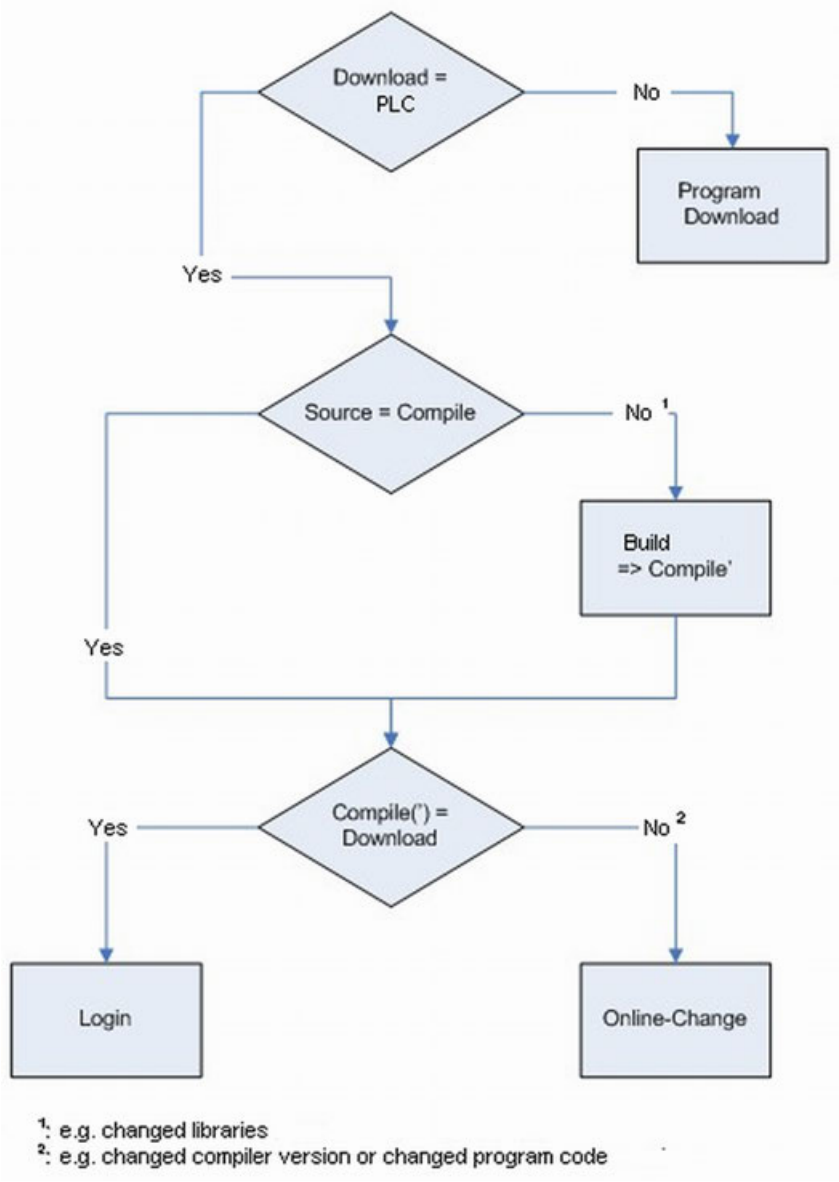
Use the 'Online' 'Logout' command to change from online back to offline mode ↳ *Chapter 1.4.1.2.6.4 “'Online' 'Logout'” on page 283*.

Relations between Login - Build - Download - Online Change

See below a diagram which shows the relations between Login, Build (Compilation), Download and Online Change:

The following terms are used:

Source	Current Automation Builder project (*.pro-file, local PC)
Compile	Compile information from the last build process, is needed for incremental compilation (*.ci-file, local PC)
Download	Information on what was loaded to the PLC at the last download (*.ri file, local PC)
PLC	Project currently available on the PLC (*.prg-file, target system)



Hints on Online Change

- Online Change is not possible after modifications in the Task or PLC Configuration, after inserting a library and after performing 'Project' 'Clean all' (see below).
- If the download Information (file <projectname><targetidentifier>.ri), which had been created at the last download (might have been an Online Change also) of the project, has been deleted meanwhile (e.g. via command 'Project' 'Clean all', then no Online Change will be possible further on, except for ↳ *Chapter 1.4.1.2.6.5 “'Online' 'Download'” on page 283* ↳ *Chapter 1.4.1.2.3.13 “'Project' 'Clean all'” on page 232*: The .ri file has been saved at another location or has been renamed and therefore now is still available and can be loaded explicitly by command 'Project' 'Load download information' ↳ *Chapter 1.4.1.2.3.14 “'Project' 'Load download information'” on page 232*. Concerning this see also below 'Online Change for a project....'.
- Online Change does not cause a re-initialization of the variables, thus modifications of the initialization values will not be regarded !
- retain variables keep their values when an Online Change is done, they won't do that at a re-download of the project ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302* ↳ *Chapter 1.4.1.2.6.5 “'Online' 'Download'” on page 283*.

Online Change for a project which is running on several PLCs:

If you want to run a project 'proj.pro' on two identical controllers PLC1 and PLC2 (same target system) and want to make sure that an update of the project on both controllers can be done via online change, do the following:

(1) Loading and starting project on PLC1, save download information for PLC1:

1. Connect the Automation Builder project proj.pro to controller PLC1 (Online/Communication parameters) and load proj.pro on PLC1 (Online/Login, Download). At the download the file proj00000001.ri will be created in the projects directory, containing download information.
2. Rename proj00000001.ri, e.g. to proj00000001_PLC1.ri. This save of the file with another file name is necessary because at a further download of proj.pro the file proj00000001.ri on another PLC would be overwritten with new download information and thus the information belonging to the download on PLC1 would be lost.
3. Start the project on PLC1 and then log out ('Online' 'Start', 'Online' 'Logout').

(2) Loading and starting project on PLC2, save download information for PLC2:

1. Now connect to controller PLC2 (using same target as PLC1) and download proj.pro on PLC2. Thus again a file proj00000001.ri will be created in the projects directory, now containing the information on the currently done download.
2. Rename the new proj00000001.ri e.g. to proj00000001_PLC2.ri in order to store it explicitly.
3. Start the project on PLC2 and log out ('Online' 'Start', 'Online' 'Logout'.

(3) Change project in Automation Builder:

Do the modifications in proj.pro which you afterwards want to transfer via Online Change to the program running on the both PLCs.

(4) Online Change on PLC1, Re-saving download information for PLC1:

In order to make possible the Online Change for proj.pro on PLC1, first the download information referring to the download of proj.pro on PLC1 must be restored. At login Automation Builder is looking for a file proj00000001.ri. But you have stored the appropriate download information in file proj00000001_PLC1.ri.

Now you have 2 possibilities:

- You can rename proj00000001_PLC1.ri again to proj00000001.ri. Thus at a login on PLC1 automatically the appropriate download information is available and Automation Builder will ask you whether you want to do an Online Change.
- Instead of this you explicitly can load the file proj00000001_PLC1.ri before login by using command 'Project' 'Load Download Information'. In this case no renaming of the .ri file is necessary.

(5) Online Change on PLC2, Re-saving download information for PLC2:

In order to make possible an Online Change concerning the modifications in proj.pro done in (3) also on PLC2 please perform the corresponding steps for proj00000001_PLC2.ri as described in step (4) .

If the system reports

Error:

```
"The selected controller profile does not match that of the target system..."
```

Check that the target system entered in the target system settings (Resources) matches the parameters entered in 'Online' 'Communications parameters'.

Error:

"Communication error. Log-out has occurred"

Check whether the controller is running. Check whether the parameters entered in 'Online' 'Communications parameters' match those of your controller. In particular, you should check whether the correct port has been entered and whether the baud rates in the controller and the programming system match. If the gateway server is used, check whether the correct channel is set.

Error:

"The program has been modified! Should the new program be loaded?"

The project which is open in the editor is incompatible with the program currently found in the PLC (or with the simulation mode program being run *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291*). Monitoring and debugging is therefore not possible *Chapter 1.4.1.1.11.2 “Debugging” on page 182*. You can either choose "No," logout, and open the right project, or use "Yes" to load the current project in the PLC.

Message:

"The program has been changed. Load changes? (ONLINE CHANGE)".

The project is running on the controller. The target system supports 'Online Change' and the project has been altered on the controller with respect to the most recent download or the most recent Online Change. You may now decide whether these changes should be loaded with the controller program running or whether the command should be cancelled. You can also, however, load the entire compiled code by selecting the Load all button.

'Online' 'Logout'

Symbol:  Shortcut <Ctrl>+<F8>

The connection to the PLC is broken, or the simulation mode program is ended and is shifted to the offline mode *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291*.

Use the 'Online' 'Login' command to change to the online mode *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*.

'Online' 'Download'

This command loads the compiled project in the PLC.

During compilation the Download-Information gets saved in a file called <project-name>0000000ar.ri , which is used during online change to compare the current program with the one most recently loaded onto the controller, so that only changed program components are reloaded *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*. This file is erased by the command 'Project' 'Clean all' *Chapter 1.4.1.2.3.13 “'Project' 'Clean all'” on page 232*! Concerning Online Change on several PLCs please see 'Online' 'Login' *Chapter 1.4.1.2.6.2 “'Online' 'Login'” on page 279*. Note that the *.ri file also gets updated during an Online Change.

Depending on the target system settings at each creation of a boot project in offline mode the *.ri file might be regenerated *Chapter 1.4.1.2.6.25 “'Online' 'Create boot project'” on page 291*.

Only persistent variables keep their values even after a download *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*.

See 'Online' 'Login' for a diagram showing the relations between Project-Build, Project-Download, Online Change and Login on the target system *Table on page 280*.

'Online' 'Run'

Symbol:  Shortcut: <F5>

This command starts the program in the PLC or in simulation mode *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291*.

This command can be executed immediately after the 'Online' 'Download' command, or after the user program in the PLC has been ended with the 'Online' 'Stop' command, or when the user program is at a break point, or when the 'Online' 'Single cycle' command has been executed *Chapter 1.4.1.2.6.7 “'Online' 'Stop'” on page 284* *Chapter 1.4.1.2.6.15 “'Online' 'Single cycle'” on page 286*.

'Online' 'Stop'

Symbol: Shortcut <Shift>+<F8>

Stops the execution of the program in the PLC or in simulation mode between two cycles *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291*.

Use the 'Online' 'Run' command to continue the program *Chapter 1.4.1.2.6.6 “'Online' 'Run'” on page 283*.

'Online' 'Reset'

This command resets – with exception of the retain variables (VAR RETAIN) - all variables to that specific value, with which they have got initialized (also those variables which have been declared as VAR PERSISTENT!). If you have initialized the variables with a specific value, then this command will reset the variables to the initialized value. All other variables are set at a standard initialization (for example, integers at 0). As a precautionary measure, you must confirm your decision before all of the variables are overwritten. The situation is that which occurs in the event of a power failure or by turning the controller off, then on (warm restart) while the program is running.

Use the 'Online' 'Run' command to restart the program *Chapter 1.4.1.2.6.6 “'Online' 'Run'” on page 283*.

See also 'Online' 'Reset (original)', 'Online' 'Reset (cold)' and - for an overview on reinitialization - Remanent variables *Chapter 1.4.1.2.6.10 “'Online' 'Reset (original)'” on page 284* *Chapter 1.4.1.2.6.9 “'Online' 'Reset (cold)'” on page 284* *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*.

'Online' 'Reset (cold)'

This command corresponds to 'Online' 'Reset', but besides of "normal" and persistent variables also sets back retain variables (!) to their initialization values *Chapter 1.4.1.2.6.8 “'Online' 'Reset'” on page 284*. The situation is that which occurs at the start of a program which has been downloaded just before to the PLC (cold start). See in this connection also 'Online' 'Reset' *Chapter 1.4.1.2.6.8 “'Online' 'Reset'” on page 284*, 'Online' 'Reset (original)' *Chapter 1.4.1.2.6.10 “'Online' 'Reset (original)'” on page 284*, and - for an overview on reinitialization - Remanent variables *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*.

'Online' 'Reset (original)'

This command resets all variables including the remanent ones (VAR RETAIN and VAR PERSISTENT) to their initialization values and erases the user program on the controller. The controller is returned to its original state *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*. See in this connection also 'Online' 'Reset' *Chapter 1.4.1.2.6.8 “'Online' 'Reset'” on page 284*, 'Online' 'Reset (cold)' *Chapter 1.4.1.2.6.9 “'Online' 'Reset (cold)'” on page 284* and - for an overview on reinitialization - Remanent variables *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*.

'Online' 'Toggle breakpoint'

Symbol: image\ebx_946809370.gif Shortcut: <F9>

This command sets a breakpoint in the present position in the active window ↳ *Chapter 1.4.1.1.11.3 “Breakpoint” on page 182*. If a breakpoint has already been set in the present position, that breakpoint will be removed.

The position at which a breakpoint can be set depends on the language in which the POU in the active window is written.

In the text editors (IL, ST), the breakpoint is set at the line where the cursor is located, if this line is a breakpoint position (recognizable by the dark-gray color of the line number field) ↳ *Chapter 1.4.1.3.12.1 “Overview” on page 352*. You can also click on the line number field to set or remove a breakpoint in the text editors.

In FBD and LD, the breakpoint is set at the currently selected network. In order to set or remove a breakpoint in the FBD or LD Editor, you can also click on the network number field.

In SFC, the breakpoint is set at the currently selected step ↳ *Chapter 1.4.1.3.11.9.1 “Overview” on page 330*. In SFC you can also use <Shift> with a doubleclick to set or remove a breakpoint.

If a breakpoint has been set, then the line number field or the network number field or the step will be displayed with a light-blue background color. If a breakpoint is reached while the program is running, the program will stop, and the corresponding field will be displayed in a red background color. In order to continue the program, use the 'Online' 'Run', 'Online' 'Step in', or 'Online' 'Step over' commands ↳ *Chapter 1.4.1.2.6.6 “'Online' 'Run'” on page 283* ↳ *Chapter 1.4.1.2.6.14 “'Online' 'Step in'” on page 286* ↳ *Chapter 1.4.1.2.6.13 “'Online' 'Step over'” on page 286*.

You can also use the Breakpoint dialog box to set or remove breakpoints.

'Online' 'Breakpoint dialog box'

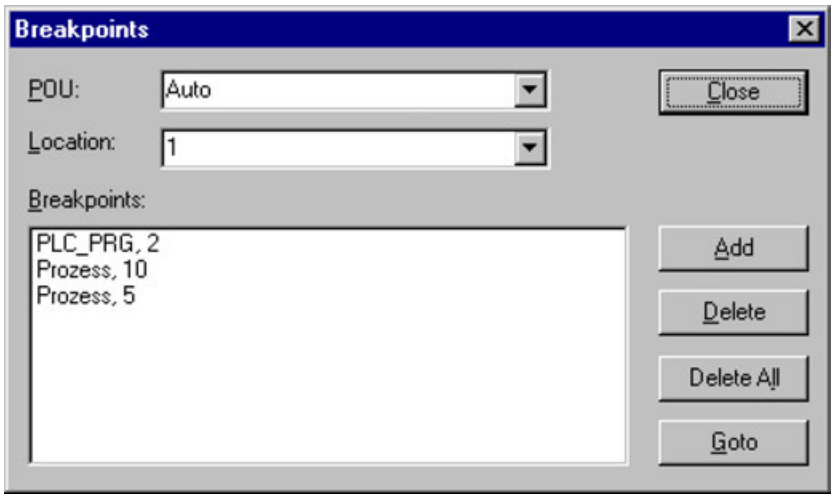
This command opens a dialog box to edit breakpoints throughout the entire project ↳ *Chapter 1.4.1.1.11.3 “Breakpoint” on page 182*. The dialog box also displays all breakpoints presently set.

In order to set a breakpoint, choose a POU in the POU combobox and the line or the network in the Location combobox where you would like to set the breakpoint; then press the Add button. The breakpoint will be added to the list.

In order to delete a breakpoint, highlight the breakpoint to be deleted from the list of the set breakpoints and press the Delete button.

The Delete All button can be used to delete all the breakpoints.

In order to go to the location in the editor where a certain breakpoint was set, highlight the respective breakpoint from the list of set breakpoints and press the Go to button.



To set or delete breakpoints, you can also use the 'Online' 'Toggle Breakpoint' command.

'Online' 'Step over'

Symbol: Shortcut: <F10>

This command causes a single step to execute *Chapter 1.4.1.1.11.4 “Single step” on page 182*. If a POU is called, the program stops after its execution. In SFC a complete action is executed *Chapter 1.4.1.1.10.5.2 “Action” on page 171*.

If the present instruction is the call-up of a function or of a function block, then the function or function block will be executed completely. Use the 'Online' 'Step in' command, in order to move to the first instruction of a called function or function block *Chapter 1.4.1.2.6.14 “'Online' 'Step in'” on page 286*.

If the last instruction has been reached, then the program will go on to the next instruction in the POU.

'Online' 'Step in'

Shortcut: <F8>

A single step is executed *Chapter 1.4.1.1.11.4 “Single step” on page 182*. The program is stopped before the first instruction of a called POU.

If necessary, there will be a changeover to an open POU.

If the present position is a call-up of a function or of a function block, then the command will proceed on to the first instruction in the called POU.

In all other situations, the command will function exactly as 'Online' 'Step over' *Chapter 1.4.1.2.6.13 “'Online' 'Step over'” on page 286*.

'Online' 'Single cycle'

Shortcut: <Ctrl>+<F5>

This command executes a single PLC Cycle and stops after this cycle *Chapter 1.4.1.1.11.5 “Single cycle” on page 182*.

This command can be repeated continuously in order to proceed in single cycles.

The Single Cycle ends when the 'Online' 'Run' command is executed *Chapter 1.4.1.2.6.6 “'Online' 'Run'” on page 283*.

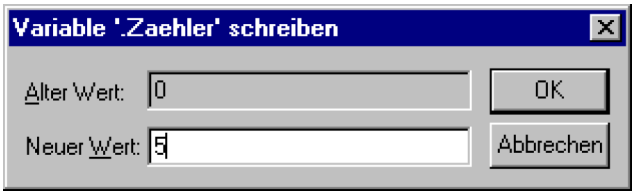
'Online' 'Write values'

Shortcut: <Ctrl>+<F7>

With this command, one or more variables are set – one time only! – to user defined values at the beginning of a cycle (see 'Online' 'Force values' for setting permanently *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*).

The values of all single-element variables can be changed, so long as they are also visible in Monitoring.

Before the command 'Write values' can be executed, a variable value must be ready to be written: For non-boolean variables a double mouse click in performed on the line in which a variable is declared, or the variable is marked and the <Enter> key is pressed. The dialog box 'Write variable <x>' then appears, in which the value to be written to the variable can be entered.



For boolean variables, the value is toggled (switched between TRUE and FALSE, with no other value allowed) by double-clicking on the line in which the variable is declared; no dialog appears.

The value set for Writing is displayed in brackets and in turquoise colour behind the former value of the variable. e.g. a=0 <:=34>.

Exception: In the FBD and LD Editor the value is shown turquoise without brackets next to the variable name.

Set the values for as many variables as you like.

The values entered to be written to variables can also be corrected or deleted in the same manner. This is likewise possible in the 'Online' 'Write/Force dialog' (see below).

The values to be written that were previously noticed are saved in a writelist (Watchlist), where they remain until they are actually written, deleted or transferred to a forcelist by the command 'Force values'.

The command to Write Values can be found at two places:

- Command 'Write Values' in the menu 'Online'.
- Button 'Write Values' in the dialog ↳ *'Editing the writelist and the forcelist'*.

When the command 'Write values' is executed, all the values contained in the writelist are written, once only , to the appropriate variables in the controller at the beginning of the cycle, then deleted from the writelist. (If the command 'Force values' is executed, the variables in question are also deleted from the writelist, and transferred to the forcelist! ↳ *Chapter 1.4.1.2.6.17 "'Online' 'Force values'" on page 287*)

In the sequential function chart language, the individual values from which a transition expression is assembled cannot be changed with 'Write values' ↳ Chapter 1.4.1.1.10.5.1 "Overview" on page 171. This is due to the fact that in monitoring the 'Total value' of the expression, not the values of the individual variables are displayed (e.g. "a AND b" is only displayed as TRUE if both variables actually have the value TRUE).

In FBD, on the other hand, only the first variable in an expression, used for example as input to a function block, is monitored ↳ *Chapter 1.4.1.1.10.2 "Function Block Diagram (FBD)" on page 162*. Thus a 'Write values' command is only possible for this variable.

'Online' 'Force values'

Shortcut: <F7>

With this command, one or more variables are permanently set (see 'Online' 'Write values' for setting only once at the beginning of a cycle ↳ *Chapter 1.4.1.2.6.16 "'Online' 'Write values'" on page 286*) to user-defined values.

The setting occurs in the runtime system, both at the beginning and at the end of the cycle. The time sequence in one cycle: 1.Read inputs, 2. Force values 3. Process code, 4. Force values 5. Write outputs.

The function remains active until it is explicitly suspended by the user (command 'Online' 'Release force') or the programming system is logged-out.

For setting the new values, a 'writelist' is first created, just as described under 'Online' 'Write values' *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*. The variables contained in the writelist are accordingly marked in Monitoring. The writelist is transferred to a 'forcelist' as soon as the command 'Online' 'Force values' is executed. It is possible that an active forcelist already exists, in which case it is updated as required. The writelist is then emptied and the new values displayed in red as 'forced'. Modifications of the forcelist are transferred to the program with the next 'Force values' command.

Note: The forcelist is created at the first forcing of the variables contained in the writelist, while the writelist existed prior to the first writing of the variables that it contains.

The command for forcing a variable, which means that it will be entered into the forcelist can be found at the following places:

- Command 'Force Values' in the menu 'Online'.
- Button 'Force Values' in the dialog *'Editing the writelist and the forcelist'*.

In the sequential function chart language, the individual values from which a transition expression is assembled cannot be changed with 'Force values'. This is due to the fact that in monitoring the 'Total value' of the expression, not the values of the individual variables are displayed (e.g. "a AND b" is only displayed as TRUE if both variables actually have the value TRUE).

In FBD, on the other hand, only the first variable in an expression, used for example as input to a function block, is monitored. Thus a 'Force values' command is only possible for this variable.

'Online' 'Release force'

Shortcut: <Shift>+<F7>

This command ends the forcing of variable values in the controller. The variable values change again in the normal way.

Forced variables can be recognized in Monitoring by the red color in which their values are displayed. You can delete the whole forcelist, but you can also mark single variables for which the forcing should be released.

To delete the whole forcelist, which means to release force for all variables, choose one of the following ways:

- Command 'Release Force' in menu 'Online'.
- Button 'Release Force' in dialog *'Editing the writelist and the forcelist'*
- Delete the whole forcelist using the command 'Release Force' in the dialog 'Remove Write-/Forcelist'. This dialog opens if you choose the command 'Release Force' while also a writelist exists *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*.
- To release force only for single variables you have to mark these variables first. Do this in one ways described in the following. After that the chosen variables are marked with an turquoise extension <Release Force>:

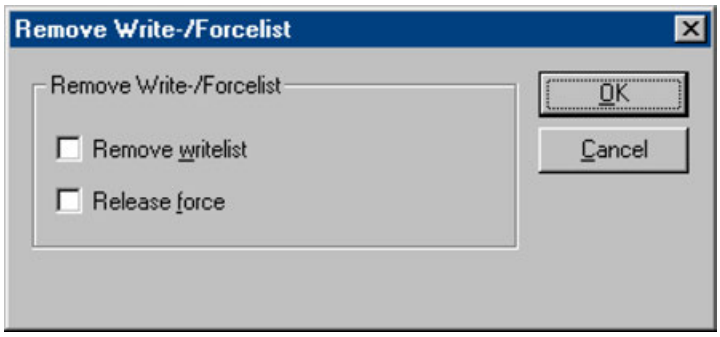
  - A double mouse click on a line, in which a non boolean variable is declared, opens the dialog 'Write variable <x>' *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*. Press button <Release Force for this variable>.

  - Repeat double mouse clicks on a line in which a boolean variable is declared to toggle to the display <Release Force> at the end of the line.

  - In the menu 'Online' open the Write/Force-Dialog and delete the value in the edit field of the column 'Forced value'.

When for all desired variables the setting "<Release Force>" is shown in the declaration window, choose the command 'Force values' to transfer the modifications of the forcelist to the program *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*.

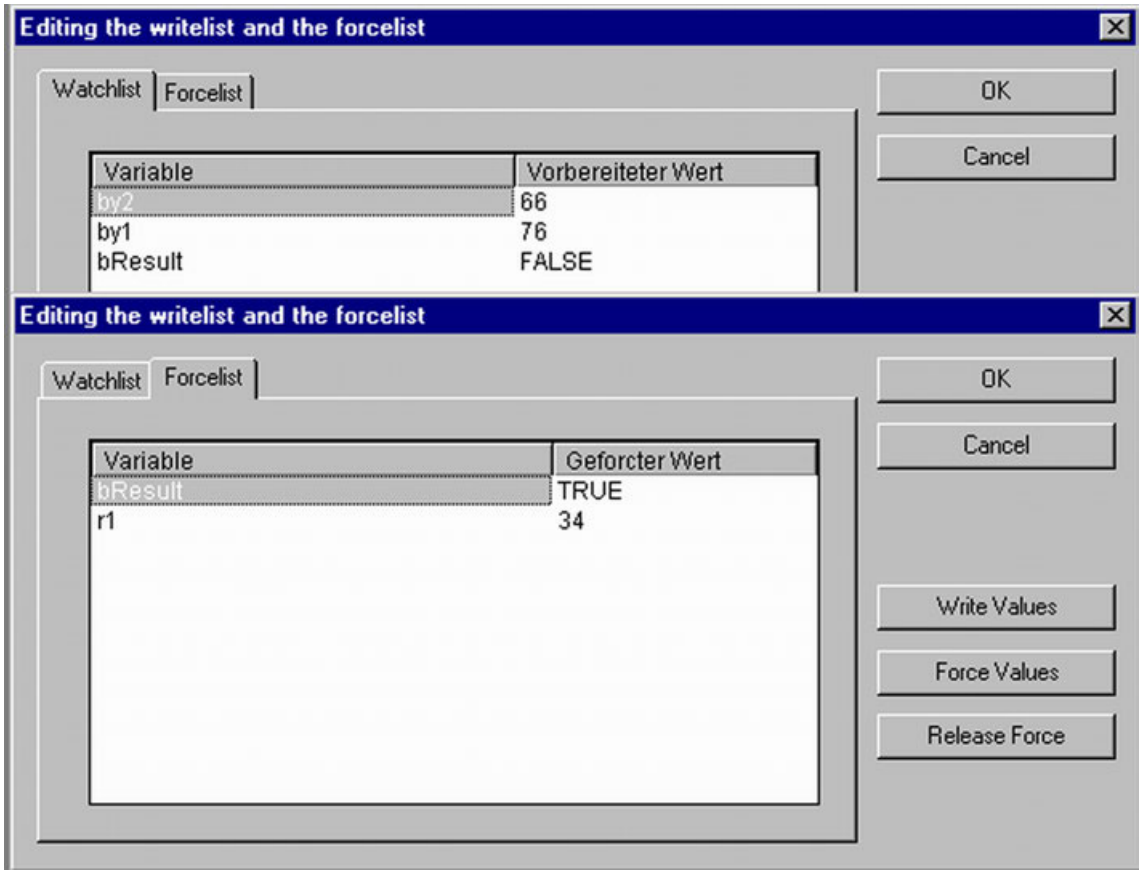
If the current writelist is not empty while you execute the command 'Release Force', the dialog 'Remove Write-/Forcelist' will be opened *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*. There the user has to decide whether he just wants to Release Force or additionally wants to Remove the writelist or if he wants to remove both lists.



'Online' 'Write/Force' Dialog'

Shortcut: <Ctrl>+<Shift>+<F7>

This command leads to a dialog which displays in two registers the current writelist (Watchlist) and forcelist (Forcelist). Each variable name and the value to be written to it or forced on it are displayed in a table.



The variables reach the watchlist via the commands 'Online' 'Write values' and are transferred to the forcelist by the command 'Online' 'Force values' ↳ *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286* ↳ *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*. The values can be edited here in the "Prepared Value" or "Forced Value" columns by clicking the mouse on an entry to open an editor field. If the entry is not type-consistent, an error message is displayed. If a value is deleted, it means that the entry is deleted from the writelist or the variable is noticed for suspension of forcing as soon as the dialog is closed with any other command than Cancel.

The following commands, corresponding to those in the Online menu, are available via buttons:

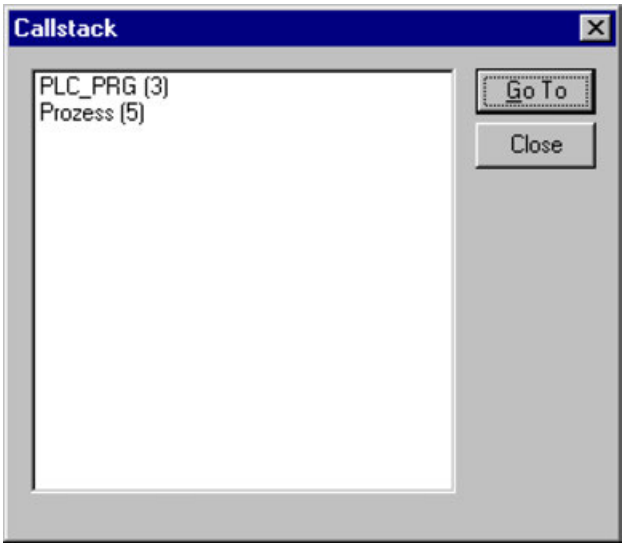
Force Values ↳ *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*: All entries in the current writelist are transferred to the forcelist, that is the values of the variables in the controller are forced. All variables marked with 'Release Force' are no longer forced. The dialog is then closed.

Write Values ↳ *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*: All entries in the current writelist are written once only to the corresponding variables in the controller. The dialog is then closed.

Release Force ↳ *Chapter 1.4.1.2.6.18 “'Online' 'Release force'” on page 288*: All entries in the forcelist will be deleted or, if a writelist is present, the dialog "Delete write-/forcelist" comes up, in which the user must decide whether he only wants to release forcing or discard the writelist, or both. The dialog will close at that point, or after the selection dialog is closed as the case may be.

'Online' 'Show call stack'

You can run this command when the simulation mode stops at a breakpoint ↳ *Chapter 1.4.1.2.6.22 “'Online' 'Simulation'” on page 291* ↳ *Chapter 1.4.1.2.3.9 “'File' 'Printer setup'” on page 229*. You will be given a dialog box with a list of the POU Call Stack.



The first POU is always PLC_PRG, because this is where the executing begins ↳ *Chapter 1.4.1.1.9.2 “POU (program organization unit)” on page 151*.

The last POU is always the POU being executed.

After you have selected a POU and have pressed the 'Go to' button, the selected POU is loaded in its editor, and it will display the line or network being processed.

'Online' 'Display flow control'

Depending on the target system settings the user can activate resp. deactivate the Flow Control function. If it is activated, a check (✓) will appear in front of the menu item. Following this, every line or every network will be marked which was executed in the last PLC Cycle.

The line number field or the network number field of the lines or networks which just run will be displayed in green. An additional field is added in the IL editor in which the present contents of the accumulator are displayed ↳ *Chapter 1.4.1.3.12.1 “Overview” on page 352*. In the graphic editors for the Function Block Diagram and Ladder Diagram, an additional field will be inserted in all connecting lines not transporting any Boolean values ↳ *Chapter 1.4.1.3.11.1 “Overview” on page 314* ↳ *Chapter 1.4.1.3.11.7.1 “Overview” on page 317* ↳ *Chapter 1.4.1.3.11.8.1 “Overview” on page 323*. When these Out- and Inputs are verified, then the value that is transported over the connecting line will be shown in this field. Connecting lines that transport only Boolean values will be shaded blue when they transport TRUE. This enables constant monitoring of the information flow.

1. The run time of a program will be increased by using flow control. This might cause timeouts in time-cyclic programs with high load.

2. At active breakpoint positions there is no flow control display.

3. If a watchdog has been defined for the concerned task, this will be switched off when the flow control is active ↳ Chapter 1.4.1.4.8.2 “'Insert' 'Insert Task' or 'Insert' 'Append Task'” on page 391.

'Online' 'Simulation'

If Simulation Mode is chosen, then a check (✓) will appear in front of the menu item.

In the simulation mode, the user program runs on the same PC under Windows. This mode is used to test the project. The communication between the PC and Simulation Mode uses the Windows Message mechanism.

If the program is not in simulation mode, then the program will run on the PLC. The communication between the PC and the PLC typically runs over the serial interface.

The status of this flag is stored with the project.

POUs of external libraries will not run in simulation mode.

'Online' 'Communication parameters'

You are offered a special dialog for setting communication parameters when the communication between the local PC and the runtime system is running over a gateway server in your system. (If the OPC or DDE server is used, the same communications parameters must be entered in its configuration).

'Online' 'Sourcecode download'

This command loads the source code for the project into the controller system. This is not to be confused with the Code that is created when the project is compiled! You can enter the options that apply to Download (time, size) in the 'Project' 'Options' 'Source download' dialog *↳ Chapter 1.4.1.2.2.1 “'Project' 'Options'” on page 200 ↳ Chapter 1.4.1.2.2.11 “'Source download'” on page 212*.

'Online' 'Create boot project'

With this command, the compiled project is set up on the controller in such a way that the controller can load it automatically when restarted. Storage of the boot project occurs differently depending on the target system. For example, on 386 systems three files are created: default.prg contains the project code, default.chk contains the code's checksum, default.sts contains the controller status after restart (start/stop).

The command 'Online' 'Create boot project' is also available in offline mode if the project has been built without errors. In this case the following files are created in the projects directory: <projektname>.prg for the boot project code, and <projektname>.chk for the checksum. These files can be renamed as necessary and then be copied to a PLC.

Depending on the target system settings at the creation of a boot project in offline mode a new *.ri file (download information) might be created *Chapter 1.4.1.2.6.5 “'Online' 'Download'” on page 283*. Also depending on the target setting a dialog will appear if this file is already existing.

If the project option "Implicit at create boot project" (category Source download) is activated, then the selected sources will be loaded automatically into the controller on the command 'Online' 'Create boot project' *Chapter 1.4.1.2.2.11 “'Source download'” on page 212.*

'Online' 'Write file to PLC'

This command is used for loading any desired file onto the controller. It opens the dialog for 'Write file to controller' in which you can select the desired file.

After the dialog is closed using the 'Open' button, the file is loaded into the controller and stored there under the same name. The loading process is accompanied by a progress dialog.

With the command 'Online' 'Load file from PLC' you can retrieve a file previously loaded on the controller *Chapter 1.4.1.2.6.27 “'Online' 'Load file from PLC'” on page 292*.

'Online' 'Load file from PLC'

With this command, you can retrieve a file previously loaded into the controller using 'Online' 'Write file to PLC' *Chapter 1.4.1.2.6.26 “'Online' 'Write file to PLC'” on page 292*. You receive the 'Load file from controller' dialog. Under Filename, provide the name of the desired file, and in the selection window enter the directory on your computer into which it is to be loaded as soon as the dialog is closed with the "Save" button.

1.4.1.2.7 Window setup

'Window' 'Tile horizontal'

With this command you can arrange all the windows horizontally in the work area so that they do not overlap and will fill the entire work area.

'Window' 'Tile vertical'

With this command you can arrange all the windows vertically in the work area so that they do not overlap and will fill the entire work area.

'Window' 'Cascade'

With this command you can arrange all the windows in the work area in a cascading fashion, one behind another.

'Window' 'Arrange symbols'

With this command you can arrange all of the minimized windows in the work area in a row at the lower end of the work area.

'Window' 'Close all'

With this command you can close all open windows in the work area.

'Window' 'Messages'

Shortcut: <Shift>+<Esc>

With this command you can open or close the Message window with the messages from the last compiling, checking, or comparing procedure *Chapter 1.4.1.2.1.6 “Message window” on page 200*.

If the messages window is open, then a check (✓) will appear in front of the command.

1.4.1.2.8 Help

'Help' 'Contents and search'

With the commands Contents or Search in the Help menu you can open the help topics window, which will be displayed via the HTML Help Viewer.

The Contents tab shows the contents tree. The books can be opened and closed by a double-click or via the plus and minus signs. That page which is currently selected in the contents tree will be displayed in the right part of the window. Hyperlinks from the text to other help pages resp. expanding hotspots are marked by a different color and an underline. A mouse-click on such texts will open the linked page resp. will show the expanded text or a figure. For example you can click on "Help Topic Window" at the end of this page in order to get displayed a figure of a help window, or you can click on the hyperlink "Context Sensitive Help" in order to get to the respective help page.

In the Index tab you can look for help pages on specific items, in the Search tab a full-text search on all help pages can be done. Follow the instructions in the register cards.

See also: Context sensitive help *Chapter 1.4.1.2.8.2 “Context sensitive help” on page 293*.

Context sensitive help

Shortcut: <F1>

You can use the <F1> key in an active window, in a dialog box, or above a menu command in order to open the 'Help' 'Contents and search' *Chapter 1.4.1.2.8.1 “'Help' 'Contents and search'” on page 293*. When you perform a command from the menu, the help for the command called up at that time is displayed. You can also highlight a text (for example, a key word or a standard function) and press <F1> to have the help displayed for that item.

1.4.1.3 Editors

1.4.1.3.1 Components of an editor

All editors for POUs (Program Organization Units) consist of a declaration part and a body. The body can consist of other a text or a graphic editor; the declaration portion is always a text editor. Body and declaration part are separated by a screen divider that can be dragged, as required, by clicking it with the mouse and moving it up or down.

1.4.1.3.2 Print margins

The vertical and horizontal margins that apply when the editor contents are printed, are shown by red dashed lines if the 'Show print range' option in the project options in the dialog 'Workspace' was selected. The properties of the printer that was entered apply, as well as the size of the print layout selected in the 'File' 'Printer Setup' menu *Chapter 1.4.1.2.3.9 “'File' 'Printer setup'” on page 229*. If no printer setup or no print layout is entered, a default configuration is used (Default.DFR and default printer). The horizontal margins are drawn as if the options 'New page for each object' or 'New page for each sub-object' were selected in 'Documentation settings'. The lowest margin is not displayed.

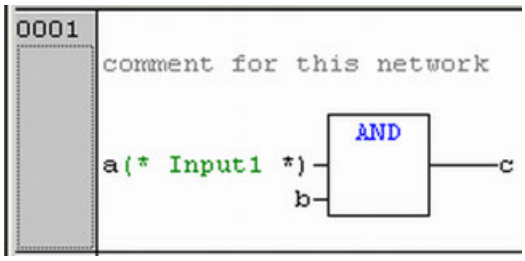
An exact display of the print margins is only possible when a zoom factor of 100% is selected.

1.4.1.3.3 Comment

User comments must be enclosed in the special symbol sequences (* and *). Example: (*This is a comment.*)

- Comments are allowed in all text editors, at any location desired, that is in all declarations, in the IL and ST languages and in self-defined data types *Chapter 1.4.1.1.10.3.1 “Overview” on page 163* *Chapter 1.4.1.1.10.4.1 “Overview” on page 165*. If the project is printed out using a template, the comment that was entered during variable declaration appears in text-based program components after each variable.
- In the FBD and LD graphic editors, comments can be entered for each network *Chapter 1.4.1.1.10.2 “Function Block Diagram (FBD)” on page 162* *Chapter 1.4.1.1.10.7.1 “Overview” on page 176*. To do this, search for the network on which you wish to comment and activate 'Insert' 'Comment'.
- Besides that comments always can be added where variable names are inserted.

  Example in FBD for a network comment and for a comment placed behind an input variable:



In KOP a comment also can be added to each contact resp. each coil, if this is configured accordingly in the display options in menu 'Extras' 'Options' *Chapter 1.4.1.3.11.9.22 “'Extras' 'Options'” on page 336*.

- In the Ladder Editor additionally a comment for each particular contact and coil can be added, if the corresponding options are activated in the menu 'Extras' 'Options'.
- In CFC there are special Comment POUs which can be placed at will *Chapter 1.4.1.1.10.6 “The continuous function chart (CFC)” on page 176* *Chapter 1.4.1.3.11.10.9 “'Insert' 'Comment' in CFC” on page 341*.
- In SFC, you can enter comments about a step in the dialog for editing step attributes *Chapter 1.4.1.1.10.5.1 “Overview” on page 171* *Chapter 1.4.1.3.11.9.20 “'Extras' 'Step Attributes'” on page 334*.

Nested comments are also allowed if the appropriate option in the 'Project' 'Options' 'Build Options' dialog is activated *Chapter 1.4.1.2.2.9 “Options for build” on page 209*.

In Online mode, if you rest the mouse cursor for a short time on a variable, the type and if applicable the address and comment of that variable are displayed in a tooltip.

1.4.1.3.4 Zoom to a POU

Shortcut: <Alt>+<Enter>

With this command a selected POU is loaded into its editor. The command is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a POU in a text editor or if the POU box is selected in a graphic editor.

If you are dealing with a POU from a library, then the library manager is called up, and the corresponding POU is displayed.

1.4.1.3.5 Open instance

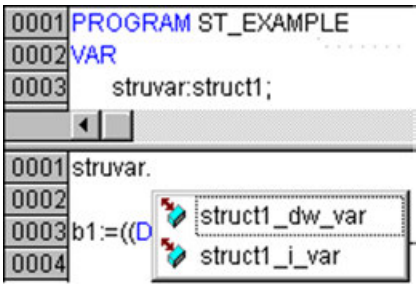
This command corresponds to the 'Project' 'View instance' command *Chapter 1.4.1.2.4.15 “'Project' 'View instance'” on page 263*.

It is available in the context menu (<F2>) or in the "Extras" menu, if the cursor is positioned on the name of a function block in a text editor or if the function block box is selected in a graphic editor.

1.4.1.3.6 Intellisense function

If the option List components is activated in the project options dialog for category 'Editor' , then the "Intellisense" functionality will be available in all editors, in the Watch- and Recipe manager, in the Visualization and in the Sampling Trace *Chapter 1.4.1.2.2.4 “Options for editor” on page 203*:

- If you insert a dot "." instead of an identifier, a selection box will appear, listing all local and global variables of the project. You can choose one of these elements and press 'Return' to insert it behind the dot. You can also insert the element by a doubleclick on the list entry.
- If you enter a function block instance or a structure variable followed by a dot, then a selection box listing all input and output variables of the corresponding function block resp. listing the structure components will appear, where you can choose the desired element and enter it by pressing 'Return' or by a doubleclick *Chapter 1.4.1.1.9.5 “Function block instances” on page 153* *Chapter 1.4.1.8.2.5 “Structures” on page 449*.
  Example: Insert "struvar." -> the components of structure struct1 will be offered:



- If you enter any string and press <Ctrl> + <Space Bar>, a selection box will appear listing all POUs and global variables available in the project. The list entry starting with the given string will be selected and can be entered to the program by pressing the <Enter> key.

1.4.1.3.7 Show cross references

Shortcut: <Strg>+<F3>

If a variable identifier is selected in an editor, then this command "Show cross references" will be available in the 'Extras' menu or the context menu. It can be used to get a list of all positions within the project where the variable is used. For information on this cross-reference list see: 'Project' 'Show cross reference'.

1.4.1.3.8 Add variables to watchlist

If one or several variables or elements are selected in one of the POU editors, these can be inserted in a new or existing watchlist via appropriate commands in the context menu. For details see Creating Watch Lists, Recipes *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

1.4.1.3.9 Declaration editor

Overview

The declaration editor is used to declare variables of POUs and global variables, for data type declarations, and in the Watch and Receipt Manager. It gives access to the usual Windows functions, and even those of the IntelliMouse can be used if the corresponding driver is installed.

In Overwrite mode, 'OV' is shown in black on the status bar; switching between Overwrite and Insert modes can be accomplished with the <Ins> key.

The declaration of variables is supported by syntax coloring *Chapter 1.4.1.3.9.15 “Syntax coloring” on page 305*.

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Consider the possibility of using pragmas to affect the properties of a variable concerning the compilation resp. precompilation process Chapter 1.4.1.3.10.1 “Pragmas, overview” on page 309.

See also:

Declaration part *Chapter 1.4.1.3.9.2 “Declaration part” on page 297*

Input variable *Chapter 1.4.1.3.9.3 “Input variable” on page 301*

Output variable *Chapter 1.4.1.3.9.4 “Output variable” on page 301*

Input and output variables *Chapter 1.4.1.3.9.5 “Input and output variables” on page 301*

Local variables *Chapter 1.4.1.3.9.6 “Local variables” on page 301*

Remanent variables *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*

Constants, typed literals *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*

External variables *Chapter 1.4.1.3.9.9 “External variables” on page 303*

Keywords *Chapter 1.4.1.3.9.10 “Keywords” on page 303*

Variables declaration *Chapter 1.4.1.3.9.11 “Variables declaration” on page 303*

AT declaration *Chapter 1.4.1.3.9.12 “AT Declaration” on page 304*

'Insert' 'Declarations keywords' *Chapter 1.4.1.3.9.13 “'Insert' 'Declaration keywords'” on page 304*

'Insert' ' Type' *Chapter 1.4.1.3.9.14 “'Insert' 'Type'” on page 304*

Syntax coloring *Chapter 1.4.1.3.9.15 “Syntax coloring” on page 305*

Shortcut mode *Chapter 1.4.1.3.9.16 “Shortcut mode” on page 305*

Autodeclaration *Chapter 1.4.1.3.9.17 “Autodeclaration” on page 306*

Line numbers in the declaration editor *Chapter 1.4.1.3.9.18 “Line numbers in the declaration editor” on page 307*

Pragma command *Chapter 1.4.1.3.10.2 “Pragma instructions for initialization, monitoring, creation of symbols, bitaccess, linking” on page 310*

Declarations as table *Chapter 1.4.1.3.9.20 “Declarations as table” on page 307*

'Insert' 'NewDeclaration' *Chapter 1.4.1.3.9.19 “'Insert' 'New declaration'” on page 307*

Declaration editors in online mode ↳ *Chapter 1.4.1.3.9.21 “Declaration editors in online mode” on page 308*

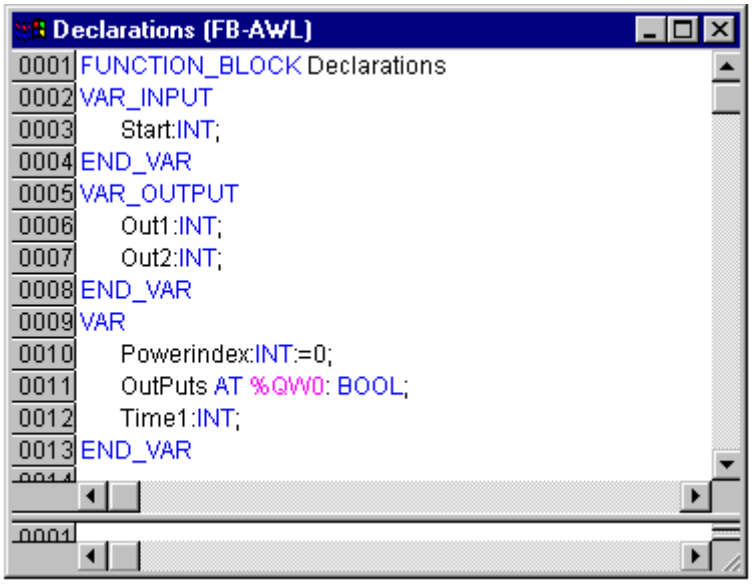
Declaration part

All variables to be used only in this POU are declared in the declaration part of the POU. These can include: input variable, output variable , input/output variables, local variables, remanent variables and constants ↳ *Chapter 1.4.1.3.9.3 “Input variable” on page 301* ↳ *Chapter 1.4.1.3.9.4 “Output variable” on page 301* ↳ *Chapter 1.4.1.3.9.5 “Input and output variables” on page 301* ↳ *Chapter 1.4.1.3.9.6 “Local variables” on page 301* ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302* ↳ *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*. The declaration syntax is based on the IEC61131-3 standard.

Regard the possibility of using templates for objects of type 'Global Variables', 'Data types', 'Function', 'Function Block' or 'Program' ↳ *Chapter 1.4.1.2.4.7 “'Save as template'” on page 259*.

> *Regard the possibility of using pragmas to affect the properties of a variable concerning the compilation resp. precompilation process ↳ pragmas.*

An example of a correct declaration of variables in the editor:



Recommendations on the naming of identifiers

Identifiers are defined at the declaration of variables (variable names), user-defined data types and at the creation of POUs (functions, function blocks, programs) and visualizations. You might follow the following recommendations concerning the naming of identifiers in order to make it as unique as possible.

(1) Variable names

The naming of variables in applications and libraries as far as possible should follow the Hungarian notation:

For each variable a meaningful, short description should be found, the base name. The first letter of each word of a base name should be a capital letter, the others should be small ones (Example: FileSize). If needed additionally an translation file for other languages can be created.

Before the base name, corresponding to the data type of the variable, prefix(es) are added in small letters.

Data type	Lower limit	Upper limit	Information content	Prefix	Comment
BOOL	FALSE	TRUE	1 bit	x*	
				b	reserved
BYTE			8 bit	by	Bit string, not for arithm. operations
WORD			16 bit	w	Bit string, not for arithm. operations
DWORD			32 bit	dw	Bit string, not for arithm. operations
LWORD			64 bit	lw	not for arithm. operations
SINT	-128	127	8 bit	si	
USINT	0	255	8 bit	usi	
INT	-32.768	32.767	16 bit	i	
UINT	0	65.535	16 bit	ui	
DINT	-2.147.483.648	2.147.483.647	32 bit	di	
UDINT	0	4.294.967.295	32 bit	udi	
LINT	-2^{63}	$2^{63} - 1$	64 bit	li	
ULINT	0	$2^{64} - 1$	64 bit	uli	
REAL			32 bit	r	
LREAL			64 bit	lr	
STRING				s	
TIME				tim	
TIME_OF_DAY				tod	
DATETIME				dt	
DATE				date	
ENUM				16 bit	
e			POINTER		
	p		ARRAY		
		a			

* pointedly for BOOLean variables x is chosen as prefix, in order to differentiate from BYTE and also in order to accommodate the perception of an IEC-programmer (see addressing %IX0.0).

Examples:

```
bySubIndex: BYTE;
sFileName: STRING;
udiCounter: UDINT;
```

In nested declarations the prefixes are attached to each other in the order of the declarations:

Example:

```
pabyTelegramData: POINTER TO ARRAY [0..7] OF BYTE;
```

Function block instances and variables of user-defined data types as a prefix get a shortcut for the FB- resp. data type name (Examle: sdo).

Example:

```
cansdoReceivedTelegram: CAN_SDOTelegram;
TYPE CAN_SDOTelegram :  (* prefix: sdo *)
STRUCT
wIndex:WORD;
bySubIndex:BYTE;
byLen:BYTE;
aby: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Locale contants (c) start with prefix c and an attached underscore, followed by the type prefix and the variable name.

Example:

```
VAR CONSTANT
c_uiSyncID: UINT := 16#80;
END_VAR
```

For Global variables (g) and Global constants (gc) an additional prefix + underscore is attached to the library prefix:

Examples:

```
VAR_GLOBAL
CAN_g_iTest: INT;
END_VAR
VAR_GLOBAL CONSTANT
CAN_gc_dwExample: DWORD;
END_VAR
```

(2) User-defined data types (DUT)

The name of each structure data type consists of a library prefix (Example: CAN), an underscore and a preferably short expressive description (Example: SDOTelegram) of the structure. The associated prefix for used variables of this structure should follow directly after the colon.

Example:

```
TYPE CAN_SDOTelegram :   (* prefix: sdo *)
STRUCT
wIndex:WORD;
bySubIndex:BYTE;
byLen:BYTE;
abyData: ARRAY [0..3] OF BYTE;
END_STRUCT
END_TYPE
```

Enumerations start with the library prefix (Example: CAL), followed by an underscore and the identifier in capital letters..

Regard that in previous versions of CODESYS ENUM values > 16#7FFF have caused errors, because they did not get converted automatically to INT values. For this reason ENUMs always should be defined with correct INT values.

Example:

```
TYPE CAL_Day :(
CAL_MONDAY,
CAL_TUESDAY,
CAL_WEDNESDAY,
CAL_THIRSDAY,
CAL_FRIDAY,
CAL_SATURDAY,
CAL_SUNDAY);
Declaration:
eToday: CAL_Day;
```

(3) Functions, Function blocks, Programs (POU)

The names of functions, function blocks and programs consist of the library prefix (Example: CAN), an underscore and an expressive short name of the POU (Example: SendTelegram). Like with variables always the first letter of a word of the POU name should be a capital letter, the others should be small letters. It is recommended to compose the name of the POU of a verb and a substantive.

Example:

```
FUNCTION_BLOCK CAN_SendTelegram (* prefix: canst *)
```

In the declaration part a short description of the POU should be provided as a comment. Further on all inputs and outputs should be provided with comments. In case of function blocks the associated prefix for set-up instances should follow directly after the name.

Actions get no prefix; just actions which should be called only internally, i.e. by the POU itself, start with prv_.

Each function - for the reason of compatibility with previous CODESYS versions - must have at least one parameter. External functions must not use structures as return values.

(4) Identifiers for Visualizations

Note: Currently you must avoid that a visualization has the same name like another POU in the project. This would lead to problems in case of changes between visualizations.

Input variable

Between the key words VAR_INPUT and END_VAR, all variables are declared that serve as input variables for a POU. That means that at the call position, the value of the variables can be given along with a call.

Example:

```
VAR_INPUT
 iIn1:INT (* 1. Inputvariable*)
END_VAR
```

Output variable

Between the key words VAR_OUTPUT and END_VAR, all variables are declared that serve as output variables of a POU. That means that these values are carried back to the POU making the call. There they can be answered and used further.

Example:

```
VAR_OUTPUT
 iOut1:INT; (* 1. Outputvariable*)
END_VAR
```

Input and output variables

Between the key words VAR_IN_OUT and END_VAR, all variables are declared that serve as input and output variables for a POU.

NOTICE!
With this variable, the value of the transferred variable is changed ("transferred as a pointer", Call-by-Reference). That means that the input value for such variables cannot be a constant. For this reason, even the VAR_IN_OUT variables of a function block can not be read or written directly from outside via <functionblockinstance><in/outputvariable>.

Example:

```
VAR_IN_OUT
 iInOut1:INT; (* 1. Inputoutputvariable *)
END_VAR
```

Local variables

Between the keywords VAR and END_VAR, all of the local variables of a POU are declared. These have no external connection; in other words, they can not be written from the outside.

Example:

```
VAR
 iLoc1:INT; (* 1. Local Variable*)
END_VAR
```

Remanent variables

Remanent variables can retain their value throughout the usual program run period. These include Retain variables and Persistent variables.

Example:

```
VAR RETAIN
 iRem1:INT; (* 1. Retain variable*)
END_VAR
```

- Retain variables are identified by the keyword RETAIN. These variables maintain their value even after an uncontrolled shutdown of the controller as well as after a normal switch off and on of the controller (corresponding to command Online Reset). When the program is run again, the stored values will be processed further. A concrete example would be an piece-counter in a production line, that recommences counting after a power failure. All other variables are newly initialized, either with their initialized values or with the standard initializations.
  Retains are reinitialized at Reset (cold) and Reset (original) and - contrary to Persistent variables - at a new download of the program.
- Persistent variables are identified by the keyword PERSISTENT. Unlike Retain variables, these variables retain their value only after a re-download, but not after an Online Reset, Online Reset (original) or Online Reset (cold), because they are not saved in the "retain area". If also persistent variables should maintain their values after a uncontrolled shutdown of the controller, then they have to be declared additionally as VAR RETAIN variables. A concrete example of "persistent Retain-Variables" would be a operations timer that recommences timing after a power failure.

x = value will be retained

- = value gets reinitialized

after Online command	VAR	VAR RETAIN	VAR PERSISTENT	VAR RETAIN PERSISTENT VAR PERSISTENT RETAIN
Reset	-	x	-	x
Reset cold	-	-	-	-
Reset origin	-	-	-	-
Download	-	-	x	x
Online Change	x	x	x	x

- *If a local variable in a program is declared as VAR RETAIN, then exactly that variable will be saved in the retain area (like a global retain variable).*
- *If a local variable in a function block is declared as VAR RETAIN, then the complete instance of the function block will be saved in the retain area (all data of the POU), whereby only the declared retain variable will be handled as a retain.*
- *If a local variable in a function is declared as VAR RETAIN, then this will be without any effect. The variable will not be saved in the retain area ! If a local variable is declared as PERSISTENT in a function, then this will be without any effect also!*

Constants, typed literals

Constants are identified by the key word CONSTANT. They can be declared locally or globally.

Syntax:

VAR CONSTANT

<Identifier>:<Type> := <initialization>;

END_VAR

Example:

```
VAR CONSTANT
 c_iCon1:INT:=12; (* 1. Constant*)
END_VAR
```

See 'Operands' for a listing of possible constants *Chapter 1.4.1.7.1 “Overview” on page 435*. See there also regarding the possibility of using typed constants *Chapter 1.4.1.7.2.9 “Typed literals” on page 437*.

External variables

Global variables which are to be imported into the POU are designated with the keyword EXTERNAL. They also appear in the Watch window of the declaration part in Online mode.

If the VAR_EXTERNAL declaration does not match the global declaration in every respect, the following error message appears: "Declaration of '<var>' does not match global declaration!"

If the global variable does not exist, the following error message appears: "Unkown global variable: '<var>'!"

Example:

```
VAR EXTERNAL
 iVarExt1:INT:=12; (* 1st external variable *)
END_VAR
```

Keywords

Keywords are to be written in uppercase letters in all editors *Chapter 1.4.1.3.9.13 “'Insert' 'Declaration keywords'” on page 304*. Keywords may not be used as variables. Examples for keywords: VAR, VAR_CONSTANT, IF, NOT, INT *Chapter 1.4.1.1.10.8 “Reserved keywords” on page 178*.

Variables declaration

A variables declaration has the following syntax:

<Identifier> {AT <Address>}:<Type> {:=<initialization>};

The parts in the braces {} are optional.

Regarding the identifier, that is the name of a variable, it should be noted that it may not contain spaces or umlaut characters, it may not be declared in duplicate and may not be identical to any keyword. Upper/lowercase writing of variables is ignored, in other words VAR1, Var1 and var1 are not different variables. Underlines in identifiers are meaningful, e.g. A_BCD and AB_CD are interpreted as different identifiers. Multiple consecutive underlines at the beginning of an identifier or within a identifier are not allowed. The length of the identifier, as well as the meaningful part of it, are unlimited.

All declarations of variables and data type elements can include initialization. They are brought about by the ":=" operator. For variables of elementary types, these initializations are constants. The default-initialization is 0 for all declarations.

Example: iVar1:INT:=12; (* Integer variable with initial value of 12*)

If you wish to link a variable directly to a definite address, then you must declare the variable with the keywordAT.

For faster input of the declarations, use the shortcut mode.

In function blocks you can also specify variables with incomplete address statements. In order for such a variable to be used in a local instance, there must be an entry for it in the variable configuration.

Pay attention to the possibility of an automatic declaration.

Regard the possibility of using pragmas to affect the properties of a variable concerning the compilation resp. precompilation process.

AT Declaration

If you wish to link a variable directly to a definite address, then you must declare the variable with the keyword AT. The advantage of such a procedure is that you can assign a meaningful name to an address, and that any necessary changes of an incoming or outgoing signal will only have to be made in one place (e.g., in the declaration).

Notice that variables requiring an input cannot be accessed by writing.

Examples:

```
xCounterHeat7 AT %QX0.0: BOOL;
wLightcabinetimpulse AT %IW2: WORD;
xDownload AT %MX2.2: BOOL;
```

If boolean variables are assigned to a Byte, Word or DWORD address, they occupy one byte with TRUE or FALSE, not just the first bit after the offset!

'Insert' 'Declaration keywords'

You can use this command to open a list of all the keywords that can be used in the declaration part of a POU. After a keyword has been chosen and the choice has been confirmed, the word will be inserted at the present cursor position *Chapter 1.4.1.1.10.8 “Reserved keywords” on page 178*.

You also receive the list when you open the input assistant (<F2>) and choose the Declarations category *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

'Insert' 'Type'

With this command you will receive a selection of the possible types for a declaration of variables. You also receive the list when you access the input assistant (<F2>).

The types are divided into these categories:

- Standard types BOOL, BYTE, etc.
- Defined types Structures, enumeration types, etc.
- Standard function blocks for instance declarations
- Defined function blocks for instance declarations

All standard types of IEC1131-3 are supported *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*.

Syntax coloring

In all editors you receive visual support in the implementation and declaration of variables. Errors are avoided, or discovered more quickly, because the text is displayed in color.

A comment left unclosed, thus annotating instructions, will be noticed immediately; keywords will not be accidentally misspelled, etc.

Table 12: Used color highlighting

Color	HIghlighted text
Blue	Keywords
Green	Comments in the text editors
Pink	Special constants (e.g. TRUE/FALSE, T#3s, %IX0.0)
Red	Input error (for example, invalid time constant, keyword, written in lower case,...)
Black	Variables, constants, assignment operators, ...

Shortcut mode

The declaration editor allows you to use the shortcut mode. This mode is activated when you end a line with <Ctrl><Enter>

Supported shortcuts:

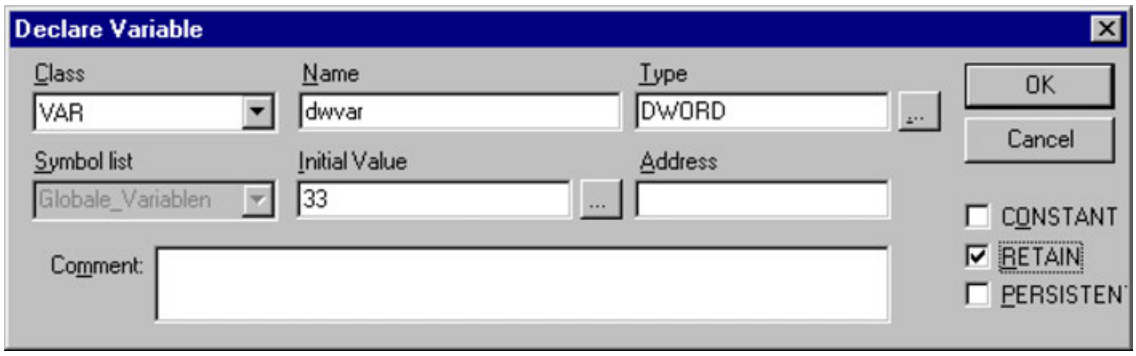
- All identifiers up to the last identifier of a line will become declaration variable identifiers
- The type of declaration is determined by the last identifier of the line. In this context, the following will apply:
  - B or BOOL gives the result - BOOL
  - I or INT gives the result - INT
  - R or REAL gives the result - REAL
  - S or string gives the result - STRING
- If no type has been established through these rules, then the type is BOOL and the last identifier will not be used as a type (Example 1.).
- Every constant, depending on the type of declaration, will turn into an initialization or a string (Examples 2. and 3.).
- An address (as in %MD12) is extended around the ATATDeclaration>Proc... attribute (Example 4.).
- A text after a semicolon (;) becomes a comment (Example 4.).
- All other characters in the line are ignored (e.g., the exclamation point in Example 5.).

Examples

Shortcut	Declaration
A	A: BOOL;
A B I 2	A, B: INT := 2;
ST S 2; A string	ST:STRING(2); (* A string *)
X %MD12 R 5 Real Number	X AT %MD12: REAL := 5.0;(* Real Number *)
B !	B: BOOL;

Autodeclaration

If the Autodeclaration option has been chosen in the Editor category of the Options dialog box , then a dialog box will appear in all editors after the input of a variable that has not yet been declared:



With the help of this dialog box, the variable can now be declared.

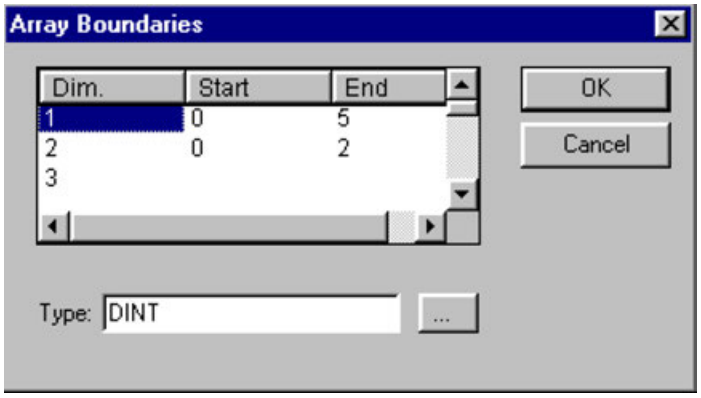
With the help of the Class combobox, select whether you are dealing with a local variable (VAR), input variable((VAR_INPUT), output variable (VAR_OUTPUT), input/output variable (VAR_INOUT), or a global variable (VAR_GLOBAL).

With the CONSTANT, RETAIN, PERSISTENT options, you can define whether you are dealing with a constant or a retain variable ↬ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*.

The variable name you entered in the editor has been entered in the Name field, BOOL has been placed in the Type field. The … button opens the input assistant dialog which allows you to select from all possible data types.

Declaration of Arrays

If ARRAY is chosen as the variable type, the dialog for entering array boundaries appears:



For each of the three possible dimensions (Dim.), array boundaries can be entered under Start and End by clicking with the mouse on the corresponding field to open an editing space. The array data type is entered in the Type field. In doing this, the … button can be used to call up an input assistant dialog.

Upon leaving the array boundaries dialog via the OK button, variable declarations in IEC format are set up based on the entries in the Type field in the dialog. Example: ARRAY [1..5, 1..3] OF INT

In the field Initial value, you may enter the initial value of the variable being declared. If this is an array or a valid structure, you can open a special initialization dialog via the … button or <F2>, or open the input assistant dialog for other variable types.

In the initialization dialog for an array you are presented a list of array elements; a mouse click on the space following ":="opens an editing field for entering the initial value of an element.

In the initialization dialog for a structure, individual components are displayed in a tree structure. The type and default initial value appear in brackets after the variable name; each is followed by ":=". A mouse click on the field following ":=" opens an editing field in which you can enter the desired initial value. If the component is an array, then the display of individual fields in the array can be expanded by a mouse click on the plus sign before the array name and the fields can be edited with initial values.

After leaving the initialization dialog with OK, the initialization of the array or the structure appears in the field Initial value of the declaration dialog in IEC format.

Example: x:=5,field:=2,3,struct2:=(a:=2,b:=3)

In the Address field, you can bind the variable being declared to an IEC address (AT declaration).

If applicable, enter a Comment. The comment can be formatted with line breaks by using the key combination <Ctrl> + <Enter>.

By pressing OK, the declaration dialog is closed and the variable is entered in the corresponding declaration editor in accordance to the IEC syntax.

The dialog box for variable declaration you also get by the command 'Edit' 'Declare Variable' ↳ *Chapter 1.4.1.2.5.14 “'Edit' 'Autodeclare'” on page 278*. *If the cursor is resting on a variable in Online mode, the Autodeclare window can be opened with <Shift><F2> with the current variable-related settings displayed.*

Line numbers in the declaration editor

In offline mode, a simple click on a special line number will mark the entire text line.

In the online mode, a single click on a specific line number will open up or close the variable in this line, in case a structural variable is involved.

'Insert' 'New declaration'

With this command you bring a new variable into the declaration table of the declaration editor ↳ *Chapter 1.4.1.3.9.20 “Declarations as table” on page 307*. If the present cursor position is located in an field of the table, then the new variable will be pasted in the preceding line; otherwise, the new variable is pasted at the end of the table. Moreover, you can paste a new declaration at the end of the table by using the right arrow key or the tab key in the last field of the table.

You will receive a variable that has "Name" located in the Name field, and "Bool" located in the Type field, as its default setting. You should change these values to the desired values. Name and type are all that is necessary for a complete declaration of variables.

Declarations as table

If the Declarations as tables option is activated in the Project Options (category ‘Editor’) or - if you already are working in the declaration editor - in the context menu, the declaration editor looks like a table. As in a card-index box, you can select the register cards of the respective variable types and edit the variables.

For each variable you are given the following entry fields.

Name:	Input the identifier of the variable.
Address:	If necessary, input the address of the variable ↳ *Chapter 1.4.1.3.9.12 “AT Declaration” on page 304*
Type:	Input the type of the variable. (Input the function block when instantiating a function block)
Initial:	Enter a possible initialization of the variable (corresponding to the ":= " assignment operator).
Comment:	Enter a comment here.

Both of the display types of the declaration editor can be changed without causing any problems. In the online mode, there are no different display types.

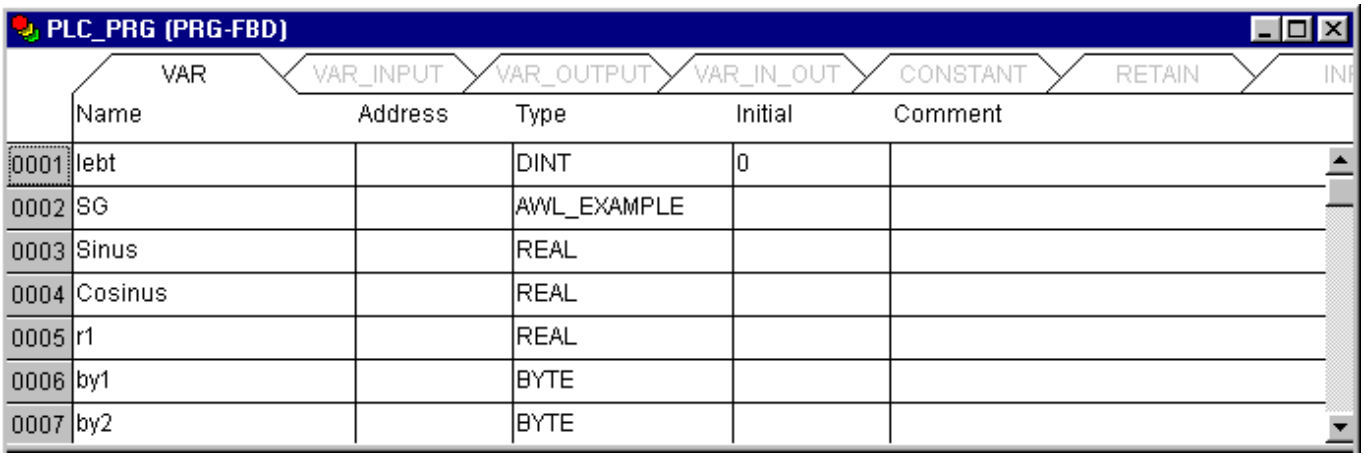
Insert new declaration:

In order to edit a new variable, select the 'Insert' 'New Declaration' command ↳ *Chapter 1.4.1.3.9.19 “'Insert' 'New declaration'” on page 307*.

Sorting the declarations:

In order to sort the table entries, set the cursor to the line number bar at the left border of the editor window and choose one of the following commands in the context menu:

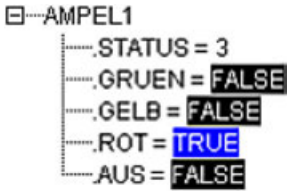
- Sort by name: All lines are sorted alphabetically according to the identifier names in column ‘Name’.
- Sort by address: All lines are sorted alphabetically according to the address entries in column ‘Address’.
- Sort by type: All lines are sorted alphabetically according to the type names in column ’Type’.
- Original order: The lines are displayed in the order in which they had been entered originally.



Declaration editors in online mode

In online mode , the declaration editor changes into a monitor window ↳ *Chapter 1.4.1.3.9.1 “Overview” on page 296*. In each line there is a variable followed by the equal sign (=) and the value of the variable. If the variable at this point is undefined, three question marks (???) will appear. For function blocks, values are displayed only for open instances (command: 'Project' 'Open instance').

In front of every multi-element variable there is a plus sign. By pressing <Enter> or after doubleclicking on such a variable, the variable is opened up. In the example, the traffic signal structure would be opened up.



When a variable is open, all of its components are listed after it. A minus sign appears in front of the variable. If you doubleclick again or press <Enter>, the variable will be closed, and the plus sign will reappear.

Pressing <Enter> or doubleclicking on a single-element variable will open the dialog box to write a variable. Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

The new value is displayed after the variable, in pointed brackets and in turquoise color, and remains unchanged. If the 'Online' 'Write values' command is given, then all variables are placed in the selected list and are once again displayed in black ↳ *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*.

If the 'Online' 'Force values' command is given, then all variables will be set to the selected values, until the 'Release force' command is given ↳ *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287* ↳ *Chapter 1.4.1.2.6.18 “'Online' 'Release force'” on page 288*. In this event, the color of the force value changes to red.

1.4.1.3.10 Pragma instructions

Pragmas, overview

The pragma instruction is used to affect the properties of a variable concerning the compilation resp. precompilation process. It can be used in with supplementary text in a program line of the declaration editor or in its own line ↳ *Chapter 1.4.1.3.9.1 “Overview” on page 296*.

The pragma instruction is enclosed in curly brackets, upper- and lower-case are ignored:
{ <Instruction text> }

If the compiler cannot meaningfully interpret the instruction text, the entire pragma is handled as a comment and read over. A warning will be issued in this case.

Depending on the type and contents of pragma, the pragma either operates on the line in which it is located or on all subsequent lines until it is ended by an appropriate pragma, or the same pragma is executed with different parameters, or the end of the file is reached. By file we mean here: declaration part, implementation portion, global variable list, type declaration.

The opening bracket may immediately follow a variable name. Opening and closing brackets must be located on the same line.

The following pragmas are currently available:

↳ *“Pragma {flag} for Initialization, Monitoring, Creation of symbols:” on page 310*

↳ *“Pragma {bitaccess...} for the Bitaccess” on page 312*

↳ *“Pragma {link} for linking a POU during code generation” on page 312*

↳ *Chapter 1.4.1.3.10.3 “Pragmas for controlling the display of library declaration parts” on page 312*

↳ *Chapter 1.4.1.3.10.4 “Pragma for nonpersistent data types” on page 313*

Pragma instructions for initialization, monitoring, creation of symbols, bitaccess, linking

Pragma {flag} for Initialization, Monitoring, Creation of symbols:

{flag [<flags>] [off|on]}

<flags> can be a combination of the following flags:

Flag	Description
noini	The variable will not be initialized.
nowatch	The variable can not be monitored
noread	The variable is exported to the symbol file without read permission
nowrite	The variable is exported to the symbol file without write permission
noread, nowrite	The variable will not get exported to the symbol file

With the "on" modifier, the pragma operates on all subsequent variable declarations until it is ended by the pragma {flag off}, or until overwritten by another {flag <flags> on} pragma.

Without the "on" or "off" modifier, the pragma operates only on the current variable declaration (that is the declaration that is closed by the next semicolon).

Example for use of pragma {flag}: Initialization and monitoring of variables

The variable a will not be initialized and will not be monitored. The variable b will not be initialized:

```
VAR
 a : INT {flag noinit, nowatch};
 b : INT {flag noinit };

END_VAR

VAR
 {flag noinit, nowatch on}
 a : INT;
 {flag noinit on}
 b : INT;
 {flag off}

END_VAR
```

Neither variable will be initialized:

```
{flag noinit on}

VAR
a : INT;
b : INT;

END_VAR

{flag off}

VAR
{flag noinit on}
a : INT;

b : INT;

{flag off}

END_VAR
```

Example for use of pragma {flag}: Getting variables to the symbol file

The flags "noread" and "nowrite" are used, in a POU that has read and/or write permission, to provide selected variables with restricted access rights. The default for the variable is the same as the setting for the POU in which the variable is declared. If a variable has neither read nor write permission, it will not be exported into the symbol file.

If the POU has read and write permission, then with the following pragmas variable a can only be exported with write permission, while variable b can not be exported at all:

```
VAR
 a : INT {flag noread};
 b : INT {flag noread, nowrite};

END_VAR

VAR
 { flag noread on}
 a : INT;
 { flag noread, nowrite on}
 b : INT;
 {flag off}

END_VAR
```

Neither variable a nor b will be exported to the symbol file:

```
{ flag noread, nowrite on }
VAR
 a : INT;
 b : INT;
END_VAR
{flag off}

VAR
 { flag noread, nowrite on}
 a : INT;
 b : INT;
 {flag off}
END_VAR
```

The pragma operates additively on all subsequent variable declarations.

Example: (all POUs in use will be exported with read and write permission)

```
a : afb;

...

FUNCTION_BLOCK afB
VAR
 b : bfb {flag nowrite};
 c : INT;
END_VAR

...

FUNCTION_BLOCK bfB
VAR
 d : INT {flag noread};
 e : INT {flag nowrite};
END_VAR
```

"a.b.d": Will not be exported

"a.b.e": Will be exported only with read permission

"a.c": Will be exported with read and write permission.

Pragma {bitaccess...} for the Bitaccess

This pragma can be used to get a correct display of a variable, which is doing a bitaccess with the help of a global constant, in the input assistant, in the intellisense function and at monitoring in the declaration window. Furtheron it will effect that, when this variable is monitored in the declaration window of the particular POU, the used global constants are shown below the respective structure variable.

Please regard: The project option 'Replace constants' (category Build) must be activated !

The pragma must be inserted in the declaration of the structure in a separate line. The line is not terminated by a semicolon.

Syntax: {bitaccess <Global Constant> <Bitnumber> '<comment>'}

<Global Constant> : Name of the global constant, which must be defined in a global variables list.

<Bitnumber> : Value of the global constant, as defined in the global variables list.

Pragma {link} for linking a POU during code generation

Normally a POU (program, function, function block) or a data unit type definition (DUT) which is not called within the project, will not be linked during code generation. But it might be desired that a function, e.g. included in the project via a library, should be available after download on the runtime system even if it is not used by the application program directly (e.g. for any check operations). For this purpose you can add the {link} pragma at any desired position in the declaration part of a POU or in a DUT in order to force a linking of the POU anyway.

Pragmas for controlling the display of library declaration parts

During creation of a library in CODESYS you can define via pragmas which parts of the declaration window should be visible resp. not visible in the Library Manager later when the library will be included in a project. The display of the implementation part of the library will not be affected by that.

Thus comments or any variables declarations can be concealed from the user. The pragmas {library private} and {library public} each affect the rest of the same line resp. the subsequent lines, as long as they are not overwritten by the each other one.

Syntax: {library public} The subsequent test will be displayed in the Library Manager. {library private} : he subsequent test will be not displayed.

Example: Declaration part of a library created in CODESYS

The comment "(* this is for all *)" should be displayed in the Library Manager after having included the library in a project., the comment "(* but this is not for all ")" however should not be displayed. The variables local and in2 also should not be displayed:

```
{library public}(* this is for all *) {library private} (* this is
not for all *)
{library public}
FUNCTION afun : BOOL
VAR_INPUT
in: BOOL;
END_VAR
{library private}
VAR
local: BOOL;
END_VAR
{library public}
VAR_INPUT
in2: BOOL;
{library private}
in3: BOOL;
{library public}
END_VAR
```

Pragma for nonpersistent data types

Normally the following is valid: Even if only one local variable in a function block or a structure is declared persistent, at usage of an instance automatically all components will be stored in the persistent information (persist.dat) on the runtime system. In order to save space you can use the pragma

{nonpersistent}

in the declaration of the function block resp. the structure. It effects that only those components of the function block resp. structure, which are declared as "persistent", will be entered to the persistent info.

Example

If an instance of the following function block is declared as persistent, only variables local and fblevel3 will be written to the persistent info. Without pragma {nonpersistent} all function block variables would be stored there.

```
FUNCTION_BLOCK FB_Level_2
{nonpersistent}
VAR_INPUT
bvar_in : BOOL;
END_VAR
VAR_OUTPUT
bvar_out : BOOL;
END_VAR
VAR
ivar2 : INT;
END_VAR
VAR PERSISTENT
local : INT := 33;
fblevel3 : FB_Level_3;
END_VAR
```

1.4.1.3.11 The graphic editors

Overview

The editors of the graphically oriented languages, Sequential Function Chart SFC, Ladder Diagram LD and Function Block Diagram FBD and of free graphic Function Block Diagram have many points in common. In the following paragraphs these features will be summarized; the specific descriptions of LD, FBD an CFC, as well as the Sequential Function Chart language SFC follow in separate sections. The implementation in the graphics editors is supported by syntax coloring.

Zoom to a POU

Shortcut: <Alt>+<Enter>

With this command a selected POU is loaded into its editor. The command is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a POU in a text editor or if the POU box is selected in a graphic editor.

If you are dealing with a POU from a library, then the library manager is called up, and the corresponding POU is displayed.

Network

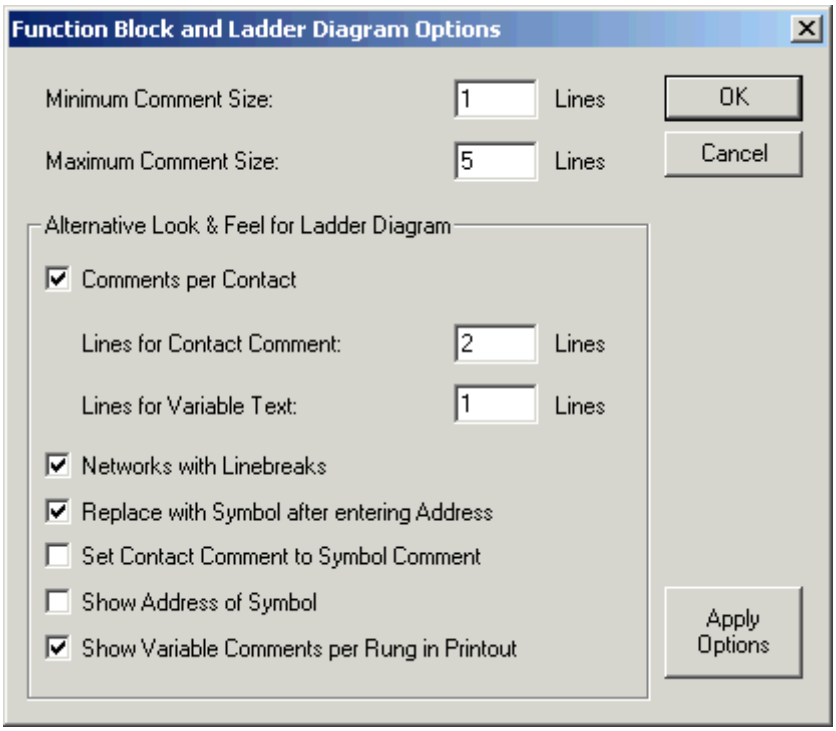
In the LD and FBD editors, the program is arranged in a list of networks. Each network is designated on the left side by a serial network number and has a structure consisting of either a logical or an arithmetic expression, a program, function or function block call, and a jump or a return instruction.

Label

Each network has a label that can optionally be left empty. This label is edited by clicking the first line of the network, directly next to the network number. Now you can enter a label, followed by a colon.

Comments, networks with linebreaks, 'Extras' 'Options'

Every network can be supplied with a multi-lined comment. In the dialog 'Function Block and Ladder Diagram Options', which can be opened by executing the command 'Extras' 'Options', you can do settings concerning comments and linebreaks.



Maximum Comment Size: Maximum number of lines to be made available for a network comment (The default value here is 4.)

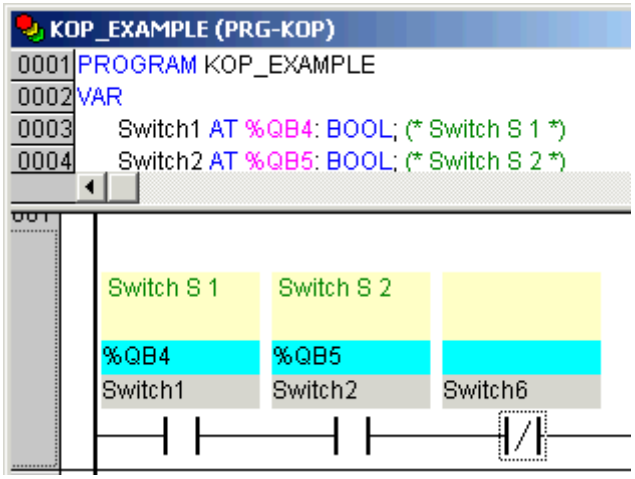
Minimum Comment Size: Number of lines that generally should be reserved for comments. If, for example, the number 2 is entered, then, at the start of each network there will be two empty lines after the label line. The default value here is 0, which has the advantage of allowing more networks to fit in the screen area.

If the minimal comment size is greater than 0, then in order to enter a comment you simply click in the comment line and then enter the comment. Otherwise you first must select the network to which a comment is to be entered, and use 'Insert' 'Comment' to insert a comment line. In contrast to the program text, comments are displayed in grey.

Alternative Look & Feel: The following options allow to define an alternative look of the networks.

Comments per Contact (only for Ladder editor): If this option is activated, you can assign an individual comment to each contact or coil . In the edit field Lines for Variable Comment enter the number of lines which should be reserved and displayed for the comment. If this setting is done, a comment field will be displayed in the editor above each contact and coil where you can insert text.

If Comments per Contact is activated, then in the Ladder editor also the number of lines (Lines for Variable Text:) can be defined which should be used for the variable name of the contact resp. coil. This is used to display even long names completely by breaking them into several lines. In the following example 2 lines are defined for the variable comment and 1 line for the variable text:



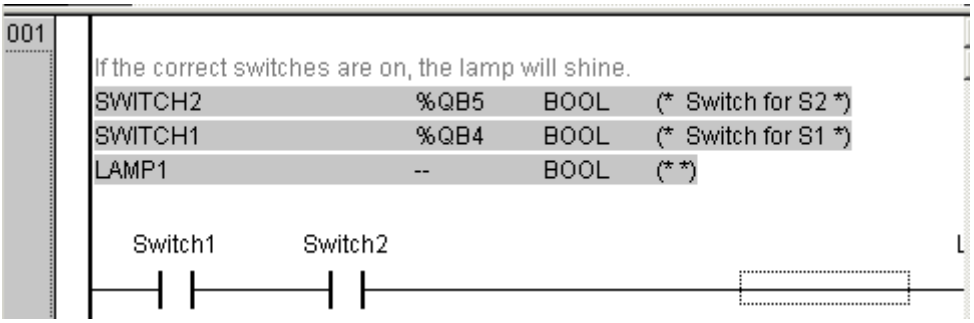
Networks with Linebreaks (only for Ladder editor): If this option is activated, linebreaks will be forced in the networks as soon as the network length exceeds the given window size and some of the elements would not be visible.

Replace with Symbol after entering Address: (only for Ladder editor): If this option is activated, you can enter an address at a box resp. at a contact or coil (e.g. "%QB4") and this address will be replaced immediately by the name of the variable which is assigned to the address. If there is no variable assigned to the entered address, the address remains displayed unchangedly.

Set Contact Comment to Symbol Comment: If this option is activated, in the comment field of a contact resp. a coil that comment will be displayed which has been defined at the declaration of the variable used for the contact or coil. The comment then can be edited. (see example in the subsequent figure). For this purpose however the option 'Comments per Contact' also must be activated. Regard that a comment which has been entered already locally at a contact or coil will be replaced automatically by the variable comment in any case, even if the variable has not got a comment in its declaration!

Show Address of Symbol: (only for Ladder editor): If this option is activated and a variable assigned to a contact or coil is assigned to an address, the address will be displayed above the variable name (see example in the figure above).

Show Variable Comments per Rung in Printout: If this option is activated, in each network for each variable used in that network there will be displayed a line showing the name, address, data type and comment for this variable, as defined in the variables declaration. This might be of useful for a documentation (printout) of the project. Example:



OK: Press this button to apply the settings on the actual POU and to close the options dialog.

Apply options: Press this button to apply the settings on the whole project. A dialog will open asking you where you have explicitly to confirm that you want to do that.

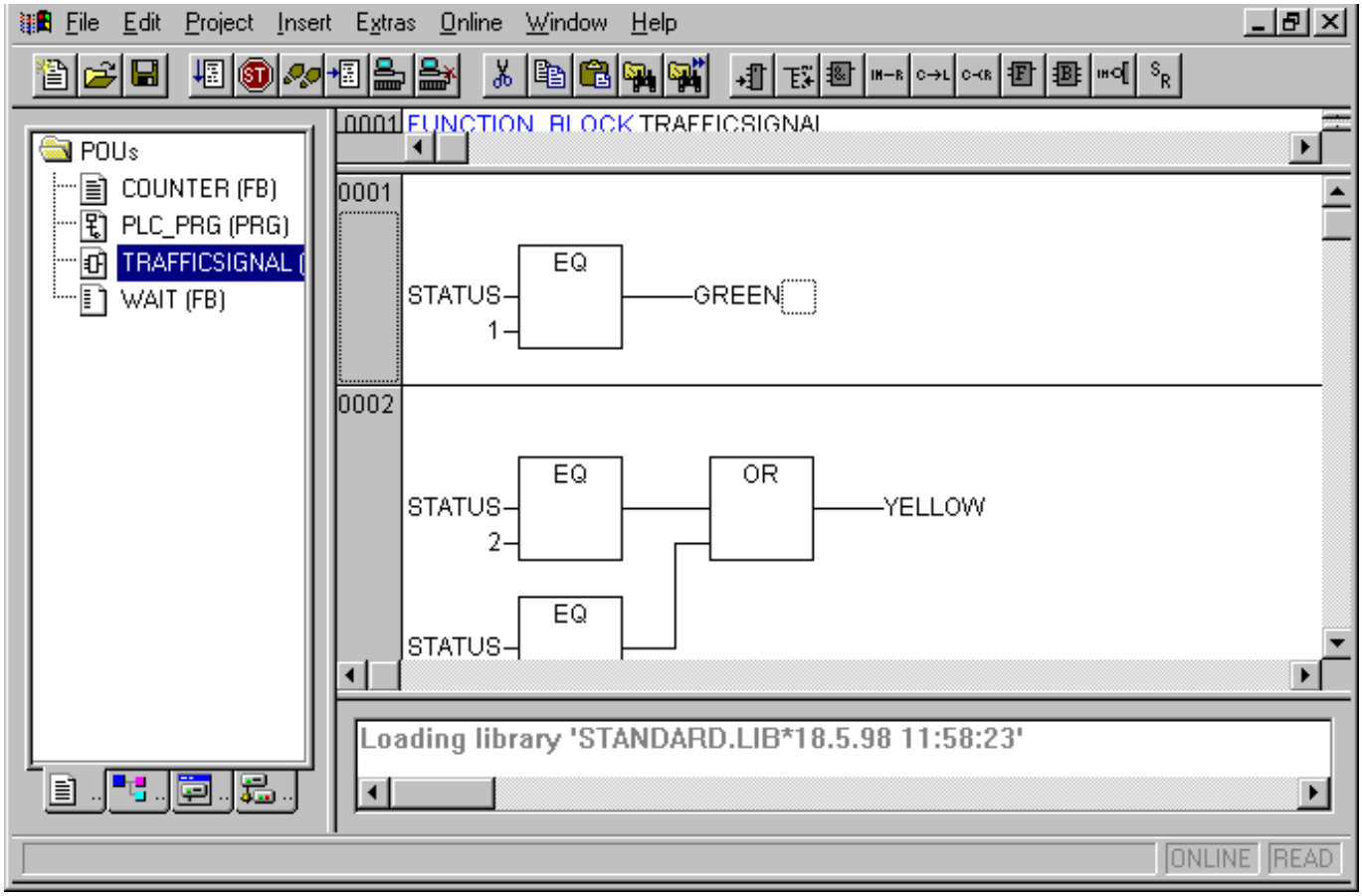
'Insert' 'Network (after)' or 'Insert' 'Network (before)'

Shortcut: <Shift>+<T> (Network after)

In order to insert a new network in the FBD or the LD editor, select the 'Insert' 'Network (after)' or the 'Insert' 'Network (before)' command, depending on whether you want to insert the new network before or after the present network. The present network can be changed by clicking the network number. You will recognize it in the dotted rectangle under the number. With the <Shift key> and a mouse click you can select from the entire area of networks, from the present one to the one clicked.

The Function Block Diagram Editor

Overview



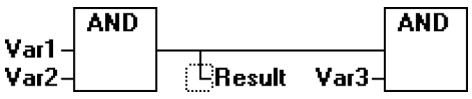
The Function Block Diagram Editor is a graphic editor. It works with a list of networks, in which every network contains a structure that displays, respectively, a logical or an arithmetical expression, the calling up of a function block, a function, a program, a jump, or a return instruction. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Cursor positions in FBD

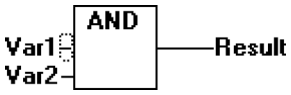
Every text is a possible cursor position. The selected text is on a blue background and can now be changed.

You can also recognize the present cursor position by a dotted rectangle. The following is a list of all possible cursor positions with an example:

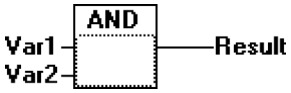
1) Every text field (possible cursor positions framed in black):



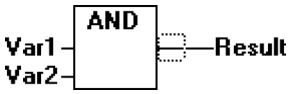
2) Every input:



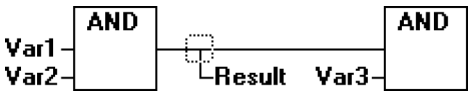
3) Every operator, function, or function block:



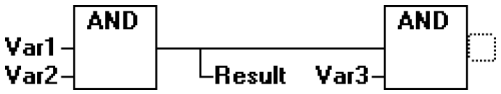
4) Outputs, if an assignment or a jump comes afterward:



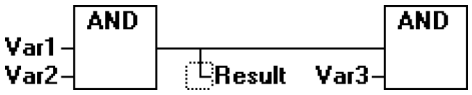
5) The lined cross above an assignment, a jump, or a return instruction:



6) Behind the outermost object on the right of every network ("last cursor position," the same cursor position that was used to select a network):



7) The lined cross directly in front of an assignment:



How to set the cursor in FBD

The cursor can be set at a certain position by clicking the mouse, or with the help of the keyboard.

Using the arrow keys, you can jump to the nearest cursor position in the selected direction at any time. All cursor positions, including the text fields, can be accessed this way. If the last cursor position is selected, then the <up> or <down> arrow keys can be used to select the last cursor position of the previous or subsequent network.

An empty network contains only three question marks "???". By clicking behind these, the last cursor position is selected.

'Insert' 'Assign'in FBD

Symbol: Shortcut: <Ctrl>+<A>

This command inserts an assignment.

Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6) *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

For an inserted assignment, a selection can be made accompanying the entered text "???", and the assignment can be replaced by the variable that is to be assigned. For this you can also use the input assistant *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*. For the possibility to enter an address instead of the variable name please see the description of the options dialog.

In order to insert an additional assignment to an existing assignment, use the 'Insert' 'Output' command.

'Insert' 'Jump' in FBD

Symbol: Shortcut: <Ctrl>+<L>

This command inserts a jump. Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2), directly after the selected output (Cursor Position 4) or at the end of the network (Cursor Position 6) *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

For an inserted jump, a selection can be made accompanying the entered text "???", and the jump can be replaced by the label to which it is to be assigned.

'Insert' 'Return' in FBD

Symbol: Shortcut: <Ctrl>+<R>

This command inserts a RETURN instruction. Depending on the selected position, insertion takes place directly in front of the selected input (Cursor Position 2),directly after the selected output (Cursor Position 4), directly before the selected line cross (Cursor Position 5), or at the end of the network (Cursor Position 6) *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

'Insert' 'Box' in FBD

Symbol: Shortcut: <Ctrl>+<B>

With this command, operators, functions, function blocks and programs can be inserted. First of all, it is always inserted an "AND" operator. This can be converted by Selection and Overwrite of the type text („AND") into every other operator, into every function, into every function block and every program. You can select the desired POU by using Input Assistant (<F2>). If the new selected block has another minimum number of inputs, these will be attached. If the new block has a smaller highest number of inputs, the last inputs will be deleted.

In functions and function blocks, the formal names of the in- and outputs are displayed.

In function blocks there exists an editable instance field above the box. If another function block that is not known is called by changing the type text, an operator box with two inputs and the given type is displayed. If the instance field is selected, input assistant can be obtained via <F2> with the categories for variable selection.

The newest POU is inserted at the selected position *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*:

- If an input is selected (Cursor Position 2), then the POU is inserted in front of this input. The first input of this POU is linked to the branch on the left of the selected input. The output of the new POU is linked to the selected input.
- If an output is selected (Cursor Position 4), then the POU is inserted after this output. The first input of the POU is connected with the selected output. The output of the new POU is linked to the branch with which the selected output was linked.
- If a POU, a function, or a function block is selected (Cursor Position 3), then the old element will be replaced by the new POU.
- As far as possible, the branches will be connected the same way as they were before the replacement. If the old element had more inputs than the new one, then the unattachable branches will be deleted. The same holds true for the outputs.
- If a jump or a return is selected, then the POU will be inserted before this jump or return. The first input of the POU is connected with the branch to the left of the selected element. The output of the POU is linked to the branch to the right of the selected element.
- If the last cursor position of a network is selected (Cursor Position 6), then the POU will be inserted following the last element. The first input of the POU is linked to the branch to the left of the selected position.

All POU inputs that could not be linked will receive the text "???". This text must be clicked and changed into the desired constant or variable.

If there is a branch to the right of an inserted POU, then the branch will be assigned to the first POU output. Otherwise the outputs remain unassigned.

'Insert' 'Input'

Symbol: Shortcut: <Ctrl>+<U>

This command inserts an operator input. With many operators, the number of inputs may vary. (For example, ADD can have 2 or more inputs.)

In order to extend such an operator by an input, you need to select the input in front of which you wish to insert an additional input (Cursor Position 1); or you must select the operator itself (Cursor Position 3), if a lowest input is to be inserted *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

The inserted input is allocated with the text "???". This text must be clicked and changed into the desired constant or variable. For this you can also use the input assistant. For the possibility to enter an address instead of the variable name please see the description of the options dialog.

'Insert' 'Output'

Symbol:

This command inserts an additional assignment into an existing assignment. This capability serves the placement of so-called assignment combs; i.e., the assignment of the value presently located at the line to several variables.

If you select the lined cross above an assignment (Cursor Position 5) or the output directly in front of it (Cursor Position 4), then there will be another assignment inserted after the ones already there *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

If the line cross directly in front of an assignment is selected (Cursor Position 4), then another assignment will be inserted in front of this one.

The inserted output is allocated with the text "???". This text must be clicked and changed into the desired variable. For this you can also use the input assistant. For the possibility to enter an address instead of the variable name please see the description of the options dialog.

'Extras' 'Negation'

Symbol: Shortcut: <Ctrl>+<N>

With this command you can negate the inputs, outputs, jumps, or RETURN instructions. The symbol for the negation is a small circle at a connection.

If an input is selected (Cursor Position 2), then this input will be negated *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

If an output is selected (Cursor Position 4), then this output will be negated.

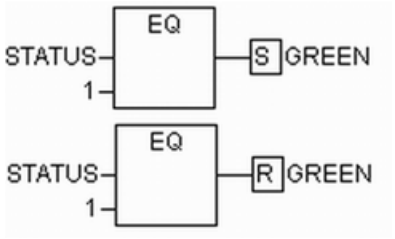
If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

'Extras' 'Set/Reset'

Symbol:

With this command you can define outputs as Set or Reset Outputs. A grid with Set Output is displayed with [S], and a grid with Reset Output is displayed with [R] :



An Output Set is set to TRUE, if the grid belonging to it returns TRUE. The output now maintains this value, even if the grid jumps back to FALSE.

An Output Reset is set to FALSE, if the grid belonging to it returns FALSE. The output maintains its value, even if the grid jumps back to FALSE.

With multiple executions of the command, the output will alternate between set, reset, and normal output.

'Extras' 'View'

Using this command for a POU created in the FBD-Editor you can choose, whether it should be displayed in the LD- (ladder logic) or in the FBD-Editor (Function Block Diagram). This is possible in offline as well as in online mode.

Open instance

This command corresponds to the 'Project' 'View instance' Command ↬ *Chapter 1.4.1.2.4.15 “'Project' 'View instance'” on page 263*.

It is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a function block in a text editor or if the function block box is selected in a graphic editor.

Cutting, copying, pasting, and deleting in FBD

The commands used to 'Cut', 'Copy', 'Paste', and 'Delete' are found under the 'Edit' menu item.

If a line cross is selected (Cursor Position 5), then the assignments, jumps, or RETURNS located below the crossed line will be cut, deleted, or copied ↬ *Chapter 1.4.1.3.11.7.2 “Cursor positions in FBD” on page 317*.

If a POU is selected (Cursor Position 3), then the selected object itself, will be cut, deleted, or copied, along with all of the branches dependent on the inputs, with the exception of the first (highest position) branch. Otherwise, the entire branch located in front of the cursor position will be cut, deleted, or copied.

After copying or cutting, the deleted or copied part is located on the clipboard and can now be pasted, as desired.

In order to do so, you must first select the pasting point. Valid pasting points include inputs and outputs.

If a POU has been loaded onto the clipboard (As a reminder: in this case all connected branches except the first are located together on the clipboard), the first input is connected with the branch before the pasting point.

Otherwise, the entire branch located in front of the pasting point will be replaced by the contents of the clipboard.

In each case, the last element pasted is connected to the branch located in front of the pasting point.

The following problem is solved by cutting and pasting: A new operator is inserted in the middle of a network. The branch located on the right of the operator is now connected with the first input, but should be connected with the second input. You can now select the first input and perform the command "'Edit' 'Cut'". Following this, you can select the second input and perform the command "'Edit' 'Paste'". This way, the branch is dependent on the second input.

The Function Block Diagram in the online mode

In the Function Block Diagram, breakpoints can only be set to networks. If a breakpoint has been set to a network, then the network numbers field will be displayed in blue. The processing then stops in front of the network where the breakpoint is located. In this case, the network numbers field will become red. Using stepping (single step), you can jump from network to network ↳ *Chapter 1.4.1.4 "The 'Resources' tab" on page 357*.

The current value is displayed for each variable. Exception: If the input to a function block is an expression, only the first variable in the expression is monitored.

Doubleclicking on a variable opens the dialog box for writing a variable. Here it is possible to change the present value of the variable. In the case of Boolean variables, no dialog box appears; these variables are toggled.

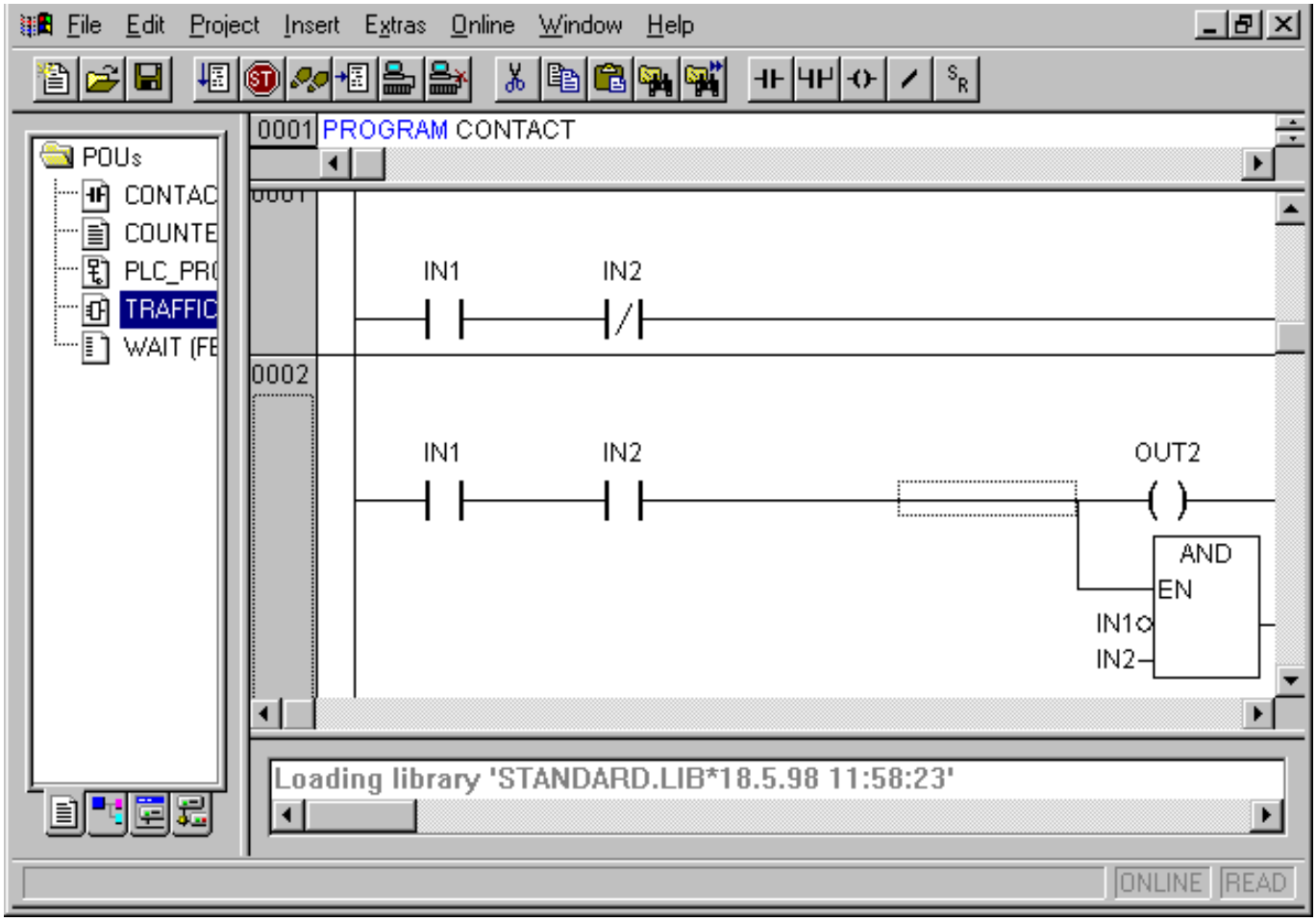
The new value will turn red and will remain unchanged. If the 'Online' 'Write values' command is given, then all variables are placed in the selected list and are once again displayed in black ↳ *Chapter 1.4.1.2.6.16 "'Online' 'Write values'" on page 286*.

The flow control is started with the 'Online' 'Flow control' command Using the flow control, you can view the present values that are being carried in the networks over the connecting lines ↳ *Chapter 1.4.1.2.6.21 "'Online' 'Display flow control'" on page 290*. If the connecting lines do not carry Boolean values, then the value will be displayed in a specially inserted field. If the lines carry Boolean values, then they will be shaded blue in the event that they carry TRUE. By this means, you can accompany the flow of information while the PLC is running.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

The ladder editor

Overview



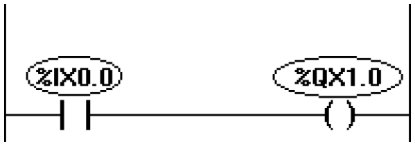
All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The LD editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

Cursor positions in the LD editors

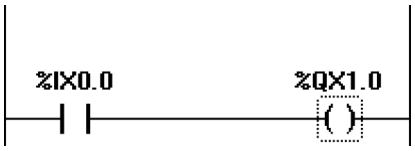
The following locations can be cursor positions, in which the function block and program accessing can be handled as contacts. POUs with EN inputs and other POUs connected to them are treated the same way as in the Function Block Diagram.

1. Every text field (possible cursor positions framed in black):

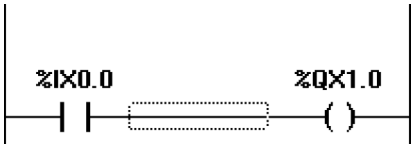


2. Every contact *Chapter 1.4.1.1.10.7.2 “Contact” on page 177* or function block:

3. Every Coil:



4. The Connecting Line between the Contacts and the Coils:



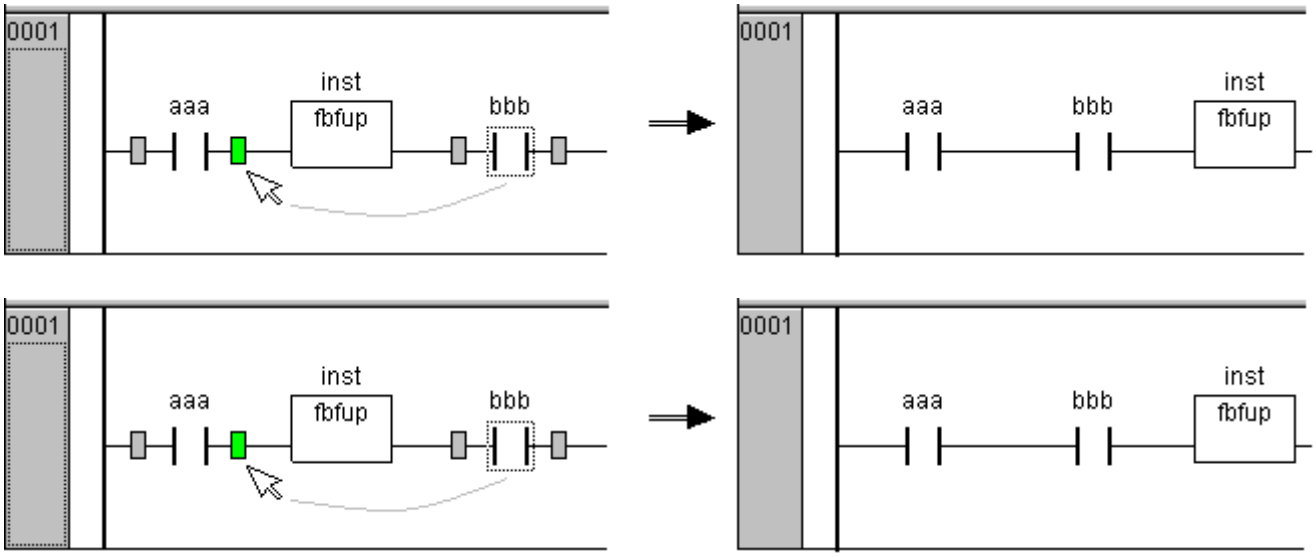
The Ladder Diagram uses the following menu commands in a special way.

Move elements or names in the LD-editor

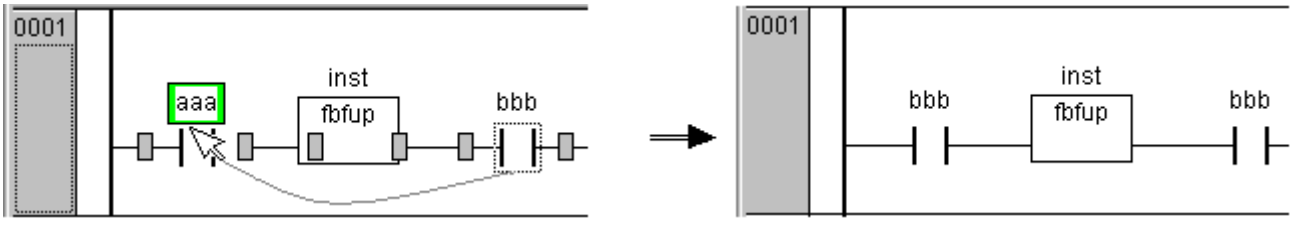
An element or just the name (variable name, address, comment) of an element can be moved to a different position within a LD POU by "drag&drop".

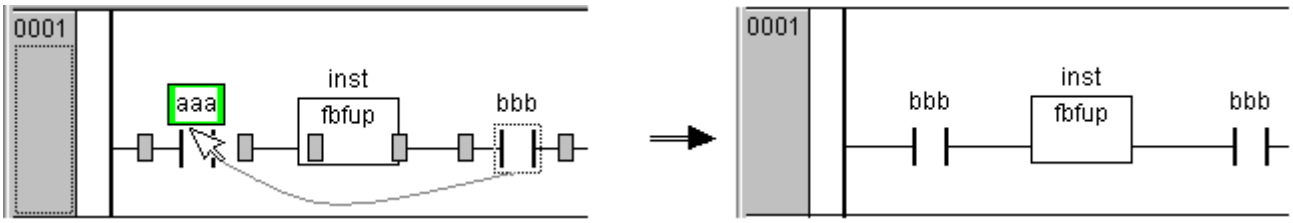
In order to do this select the desired element (contact, coil, function block) and drag it - keeping the mouse key pressed - away from the current position. Thereupon all possible positions within all networks of the POU, to which the element might be moved, will be indicated by grey-filled rectangles.

Move the element to one of these positions and let off the mouse key: the element will be inserted at the new position.



If you however move the element to the name (variable name) of another element, the name field will be shaded green. If you then let off the mouse key, the previous name will be replaced by the "dragged" one. If additionally address and comment are displayed (options), the copying also will apply to those.





'Insert' 'Network (before)' in LD

Symbol:

This command inserts a network in the Ladder editor. If there are already networks, the new one will be inserted before the currently focused.

'Insert' 'Network (after)' in LD

Symbol:

This command inserts a network in the Ladder editor. If there are already networks, the new one will be inserted after the currently focused.

'Insert' 'Contact' in LD

Symbol: Shortcut: <Ctrl>+<K>

Use this command in the LD editor in order to insert a contact in front of the marked location in the network ↳ *Chapter 1.4.1.1.10.7.2 “Contact” on page 177*.

If the marked position is a coil or the connecting line between the contacts and the coils, then the new contact will be connected serially to the previous contact connection ↳ *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323* ↳ *Chapter 1.4.1.1.10.7.3 “Coil” on page 177*.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the input assistant ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*. Note the possibility of entering an address instead of the variable name, if this is configured appropriately in the Function Block and Ladder Diagram Options.

Also in the options dialog you can activate the options Comments per Contact and Lines for variable comment and reserve a certain number of lines for the variable name. This might be useful, if long variable names are used, to keep the network short.

Also note the option 'Networks with linebreaks', which you also can activate in the Ladder Diagram Options.

'Insert' 'Contact (negated)' in LD

Symbol: Shortcut: <Ctrl> + <G>

This command inserts a negated contact ↳ *Chapter 1.4.1.1.10.7.2 “Contact” on page 177*. The same is true as for the commands 'Insert' 'Contact' in LD and 'Extras' 'Negate' in LD, which in combination also could be used to insert a negated contact ↳ *Chapter 1.4.1.3.11.8.6 “'Insert' 'Contact' in LD” on page 325* ↳ *Chapter 1.4.1.3.11.8.25 “'Extras' 'Negate' in LD” on page 329*.

'Insert' 'Parallel contact' in LD

Symbol: Shortcut: <Ctrl>+<R>

Use this command in the LD editor to insert a contact parallel to the marked position in the network ↳ *Chapter 1.4.1.1.10.7.2 “Contact” on page 177*.

If the marked position is a coil or the connection between the contacts and the coils, then the new contact will be connected in parallel to the entire previous contact connection ↳ *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323* ↳ *Chapter 1.4.1.1.10.7.3 “Coil” on page 177*.

The contact is preset with the text "???". You can click on this text and change it to the desired variable or the desired constant. For this you can also use the input assistant ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

For the possibility of entering addresses, of linebreaks for variable names and of comments per contact or coil please see the description of the Ladder Options dialog.

'Insert' 'Parallel contact (negated)' in LD

Symbol: Shortcut: <Ctrl> + <D>

This command inserts a negated parallel contact ↳ *Chapter 1.4.1.1.10.7.2 “Contact” on page 177*. The same is true as for the commands 'Insert' 'Parallel contact' in LD and 'Extras' 'Negate' in LD, which in combination also could be used to insert a negated parallel contact ↳ *Chapter 1.4.1.3.11.8.8 “'Insert' 'Parallel contact' in LD” on page 325* ↳ *Chapter 1.4.1.3.11.8.25 “'Extras' 'Negate' in LD” on page 329*.

'Insert' 'Coil' in LD

Symbol: Shortcut: <Ctrl>+<L>

You can use this command in the LD editor to insert a coil in parallel to the previous coils ↳ *Chapter 1.4.1.1.10.7.3 “Coil” on page 177*.

If the marked position is a connection between the contacts and the coils, then the new coil will be inserted as the last ↳ *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323* ↳ *Chapter 1.4.1.3.11.8.6 “'Insert' 'Contact' in LD” on page 325*. If the marked position is a coil, then the new coil will be inserted directly above it.

The coil is given the text "???" as a default setting. You can click on this text and change it to the desired variable. For this you can also use the input assistant ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

For the possibility of entering addresses, of linebreaks for variable names and of comments per coil please see the description of the Ladder Options dialog.

'Insert' 'Set' coil' in LD

Symbol: Shortcut: <Ctrl> + <I>

This command inserts a set coil ↳ *Chapter 1.4.1.1.10.7.5 “Set/Reset coils” on page 178*. The same is true as for the commands 'Insert' 'Coil' in LD and 'Extras' 'Set/Reset' in LD, which in combination also could be used to get a set coil ↳ *Chapter 1.4.1.3.11.8.10 “'Insert' 'Coil' in LD” on page 326* ↳ *Chapter 1.4.1.3.11.8.26 “'Extras' 'Set/Reset' in LD” on page 329*.

'Insert' 'Reset' coil' in LD

Symbol:

This command inserts a reset coil ↳ *Chapter 1.4.1.1.10.7.5 “Set/Reset coils” on page 178*. The same is true as for the commands 'Insert' 'Coil' in LD and 'Extras' 'Set/Reset', which in combination also could be used to get a reset coil ↳ *Chapter 1.4.1.3.11.8.10 “'Insert' 'Coil' in LD” on page 326* ↳ *Chapter 1.4.1.3.11.7.11 “'Extras' 'Set/Reset'” on page 320*.

'Insert' 'Function Block' in LD

Symbol: Shortcut: <Ctrl>+<B>

Use this command in order to insert an operator, a function block, a function or a program as a POU. For this, the connection between the contacts and the coils, or a coil, must be marked. The new POU at first has the designation AND. If you wish, you can change this designation to another one. For this you can also use the input assistant *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*. Both standard and self-defined POUs are available.

The first input to the POU is placed on the input connection, the first output on the output connection; thus these variables must definitely be of type BOOL. All other in- and outputs of the POU are filled with the text "???". These prior entries can be changed into other constants, variables or addresses. For this you can also use the input assistant *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

For the possibility of entering addresses, of linebreaks for variable names and of comments per contact, coil or function block please see the description of the Ladder Options dialog *Chapter 1.4.1.3.11.5 “Comments, networks with linebreaks, 'Extras' 'Options'” on page 314*.

POUs with EN inputs

If you want to use your LD network as a PLC for calling up other POUs , then you must merge a POU with an EN input. Such a POU is connected in parallel to the coils. Beyond such a POU you can develop the network further, as in the Function Block Diagram. You can find the commands for insertion at an EN POU under the menu item 'Insert' 'Insert at Blocks'.

An operator, a function block, a program or a function with EN input performs the same way as the corresponding POU in the Function Block Diagram, except that its execution is controlled on the EN input. This input is annexed at the connecting line between coils and contacts. If this connection carries the information "On", then the POU will be evaluated.

If a POU has been created once already with EN input, then this POU can be used to create a network. This means that data from usual operators, functions, and function blocks can flow in an EN POU and an EN POU can carry data to such usual POUs.

If, therefore, you want to program a network in the LD editor, as in FBD, you only need first to insert an EN operator in a new network. Subsequently, from this POU, you can continue to construct from your network, as in the FBD editor. A network thus formed will perform like the corresponding network in FBD.

'Insert' 'Box with EN in LD'

Symbol:

Use this command to insert a function block, an operator, a function or a program with EN input into a LD network.

The marked position must be the connection between the contacts and the coils (Cursor Position 4) or a coil (Cursor Position 3). The new POU is inserted in parallel to the coils and underneath them; it contains initially the designation "AND". If you wish, you can change this designation to another one. For this you can also use the input assistant *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

'Insert' 'Insert at blocks in LD

With this command you can insert additional elements into a POU that has already been inserted (also a POU with EN input). The commands below this menu item can be executed at the same cursor positions as the corresponding commands in the Function Block Diagram.

With 'Input' you can add a new input to the POU.

With 'Output' you can add a new output to the POU.

With 'POU', you insert a new POU. The procedure is similar to that described under 'Insert' 'POU'.

With 'Assign' you can insert an assignment to a variable. At first, this is shown by three question marks "???", which you edit and replace with the desired variable. Input assistance is available for this purpose.

'Insert' 'Rising edge detection' in LD

Symbol: 

This commands inserts a R_TRIG function block, which serves to detect a rising edge (FALSE -> TRUE) at the incoming signal. The same is true as for command 'Insert' 'Function Block' in LD which can be used to insert any available function block *Chapter 1.4.1.3.11.8.13 “'Insert' 'Function Block' in LD” on page 327*.

'Insert' 'Falling edge detection' in LD

Symbol: 

This commands inserts a F_TRIG function block, which serves to detect a rising edge (TRUE -> FALSE) at the incoming signal. The same is true as for command 'Insert' 'Function Block' in LD which can be used to insert any available function block *Chapter 1.4.1.3.11.8.13 “'Insert' 'Function Block' in LD” on page 327*.

'Insert' 'Timer (TON)' in LD

Symbol: 

This command inserts a timer Function Block of type TON. This serves to get a turn-on delay (delayed passing on of the incoming signal). For the inserting the same is true as for command 'Insert' 'Function Block' in LD, which also could be used to insert a TON module *Chapter 1.4.1.3.11.8.13 “'Insert' 'Function Block' in LD” on page 327*.

'Insert' 'Jump' in LD

With this command you can insert a parallel jump in the selected LD editor, in parallel, at the end of the previous coils *Chapter 1.4.1.1.10.7.3 “Coil” on page 177*. If the incoming line delivers the value "On", then the jump will be executed to the indicated label.

The marked position must be the connection between the contacts and the coils or a coil *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323*.

The jump is present with the text "???". You can click on this text and make a change in the desired label.

'Insert' 'Return' in LD

In the LD editor, you can use this command to insert a Return instruction in parallel at the end of the previous coils. If the incoming line delivers the value "On," then the processing of the POU in this network is broken off.

The marked position must be the connection between the contacts and the coils or a coil *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323*.

'Extras' 'Paste after' in LD

Use this command in the LD editor to paste the contents of the clipboard as serial contact below the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Paste below'in LD

Use this command in the LD editor to insert the contents of the clipboard as parallel contact below the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Paste above' in LD

Use this command in the LD editor to insert the contents of the clipboard as parallel contact above the marked position. This command is only possible if the contents of the clipboard and the marked position are networks comprised of contacts.

'Extras' 'Negate' in LD

Symbol: Shortcut: <Ctrl>+<N>

Use this command to negate a contact, a coil, a jump or return instruction, or an input or output of EN POUs at the present cursor position *Chapter 1.4.1.3.11.8.2 “Cursor positions in the LD editors” on page 323*.

Between the parentheses of the coil or between the straight lines of the contact, a slash will appear ((/) or |/|) *Chapter 1.4.1.1.10.7.3 “Coil” on page 177* *Chapter 1.4.1.1.10.7.2 “Contact” on page 177*. If there are jumps, returns, or inputs or outputs of EN POUs, a small circle will appear at the connection, just as in the FBD editor.

The coil now writes the negated value of the input connection in the respective Boolean variable. Right at this moment, a negated contact switches the status of the input to the output, if the respective Boolean variable carries the value FALSE.

If a jump or a return is marked, then the input of this jump or return will be negated.

A negation can be canceled through renewed negation.

'Extras' 'Set/Reset' in LD

Symbol:

If you execute this command on a coil, then you will receive a set coil *Chapter 1.4.1.1.10.7.3 “Coil” on page 177* *Chapter 1.4.1.1.10.7.5 “Set/Reset coils” on page 178*. Such a coil never overwrites the value TRUE in the respective Boolean variable. This means that once you have set the value of this variable to TRUE, it will always remain at TRUE. A Set Coil is designated with an "S" in the coil symbol.

If you execute this command once again, then you will be given a Reset coil. Such a coil never overwrites the value FALSE in the respective Boolean variable. This means that once you have set the value of this variable to FALSE, it will always remain at FALSE. A Reset Coil is designated with an "R" in the coil symbol.

If you execute this command repeatedly, the coil will alternate between set, reset and normal coil.

The Ladder Diagram in the online mode

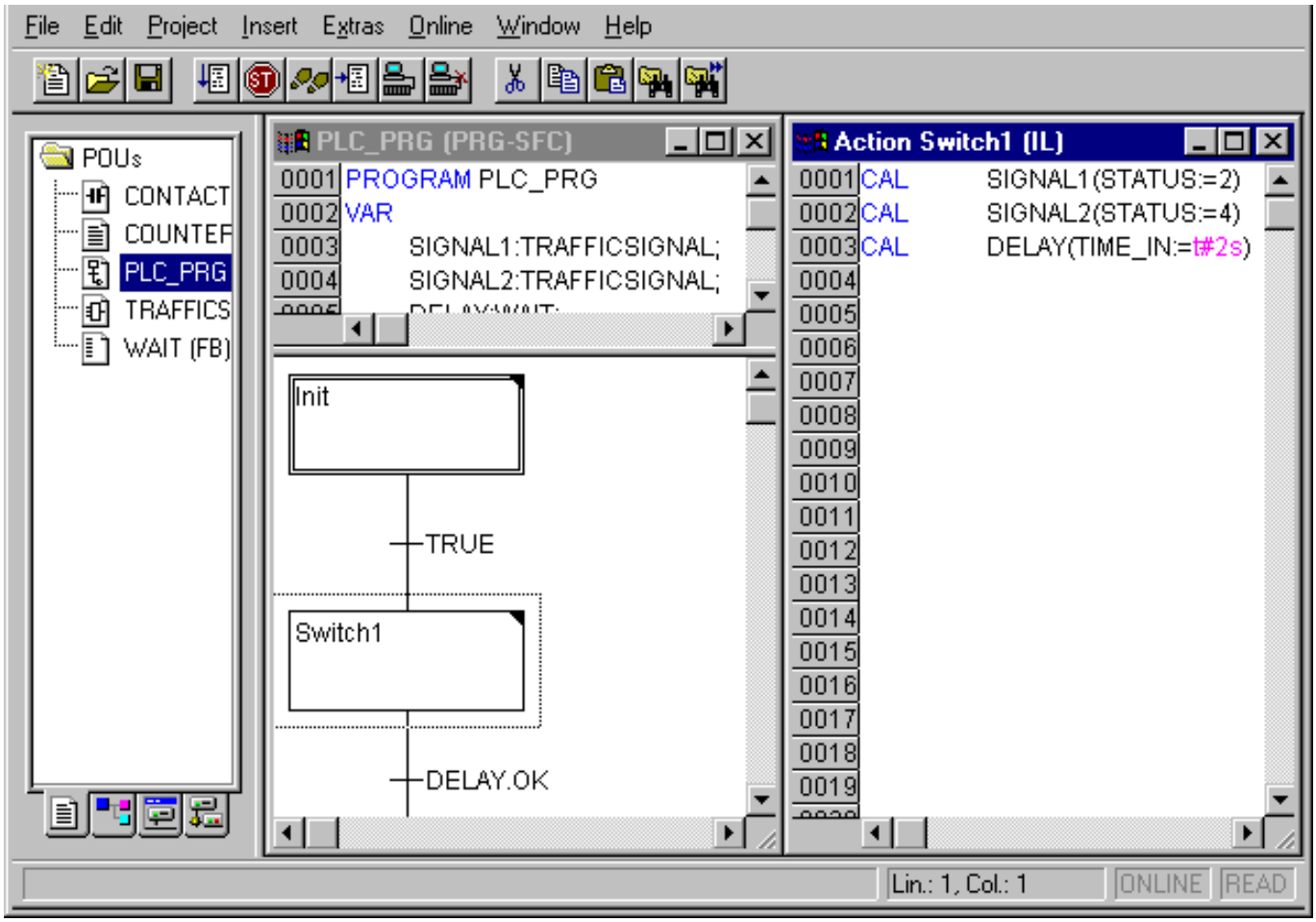
In Online mode, the contacts and coils in the Ladder Diagram that are in the "On" state are colored blue. Likewise, all lines over which the "On" is carried are also colored blue. At the inputs and outputs of function blocks, the values of the corresponding variables are indicated ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*.

Breakpoints can only be set on networks; by using stepping, you can jump from network to network ↳ *Chapter 1.4.1.1.11.3 “Breakpoint” on page 182*.

If you place the mouse pointer briefly above a variable, then the type, the address and the comment about the variable will be displayed in a Tooltip.

The sequential function chart editor

Overview



All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The Sequential Function Chart editor is a graphic editor. The most important commands are found in the context menu (right mouse button or <Ctrl><F10>). Tooltips show in Offline as well as in Online mode and in the zoomed state the full names or expressions of steps, transitions, jumps, jump labels, qualifiers or associated actions.

Marking blocks in the SFC

A marked block is a bunch of SFC elements that are enclosed in a dotted rectangle.

You can select an element (a step, a transition, or a jump) by pointing the mouse on this element and pressing the left mouse button, or you can use the arrow keys. In order to mark a group of several elements, press <Shift> for a block already marked, and select the element in the lower left or right corner of the group. The resulting selection is the smallest cohesive group of elements that includes both of these elements.

Please regard, that a step can only be deleted together with the preceding or the succeeding transition *Chapter 1.4.1.3.11.9.5 “Delete step and transition” on page 331*!

'Insert' 'Step Transition (before)'

Symbol: Shortcut: <Ctrl>+<T>

This command inserts a step in the SFC editor followed by a transition in front of the marked block.

You can select and replace the automatically specified step name "Step_<x>" by another string, also the transition name.

When renaming a step note that no comment may be added. Example: "Step_xy (counter *)" is not allowed!*

'Insert' 'Step Transition (after)'

Symbol: Shortcut: <Ctrl>+<E>

This command inserts a step in the SFC editor followed by a transition after the first transition in the marked block.

Concerning step name and transition name please see 'Insert' 'Step Transition (before)' *Chapter 1.4.1.3.11.9.3 “'Insert' 'Step Transition (before)'” on page 331*.

Delete step and transition

A step can only be deleted together with the preceding or the succeeding transition. For this purpose put a selection frame around step and transition and choose command 'Edit' 'Delete' or press the <Del> key.

'Insert' 'Alternative Branch (right)'

Symbol: Shortcut: <Ctrl>+<A>

This command inserts an alternative branch in the SFC editor as a right branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

'Insert' 'Alternative Branch (left)'

Symbol:

This command inserts an alternative branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a transition. The new branch is then made up of one transition.

'Insert' "Parallel Branch (right)"

Symbol: Shortcut: <Ctrl>+<L>

This command inserts a parallel branch in the SFC editor as the right branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step. To allow jumps to the parallel branches that have been created, these must be provided with a jump label.

'Insert' 'Parallel Branch (left)'

Symbol: This command inserts a parallel branch in the SFC editor as the left branch of the marked block. For this the marked block must both begin and end with a step. The new branch is then made up of one step. To allow jumps to the parallel branches that have been created, these must be provided with a jump label *Chapter 1.4.1.3.11.9.15 “'Extras' 'Add label to parallel branch'” on page 333*.

'Insert' 'Jump'

'This command inserts a jump in the SFC editor at the end of the branch, to which the marked block belongs. For this the branch must be an alternative branch.

The inserted text string 'Step' in the inserted jump can then be selected and replaced by the step name or the jump label of a parallel branch to be jumped to.

Concerning the step name please see 'Insert' 'Step Transition (before)' *Chapter 1.4.1.3.11.9.3 “'Insert' 'Step Transition (before)'” on page 331*.

'Insert' 'Transition-Jump'

Symbol:

This command inserts a transition in the SFC editor, followed by a jump at the end of the selected branch. For this the branch must be a parallel branch.

The inserted text string 'Step' in the inserted jump can then be selected and replaced by the step name or the jump label of a parallel branch to be jumped to.

Concerning step name and transition name please see 'Insert' 'Step Transition (before)' *Chapter 1.4.1.3.11.9.3 “'Insert' 'Step Transition (before)'” on page 331*.

'Insert' 'Add Entry-Action'

With this command you can add an entry action to a step *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. An entry-action is only executed once, right after the step has become active *Chapter 1.4.1.1.10.5.5 “Active step” on page 172*. The entry-action can be implemented in a language of your choice.

A step with an entry-action is designated by an "E" in the bottom left corner.

'Insert' 'Add Exit-Action'

With this command you can add an exit-action to a step *Chapter 1.4.1.1.10.5.3 “Entry or exit action” on page 172*. An exit-action is only executed once, before the step is deactivated. The exit-action can be implemented in a language of your choice.

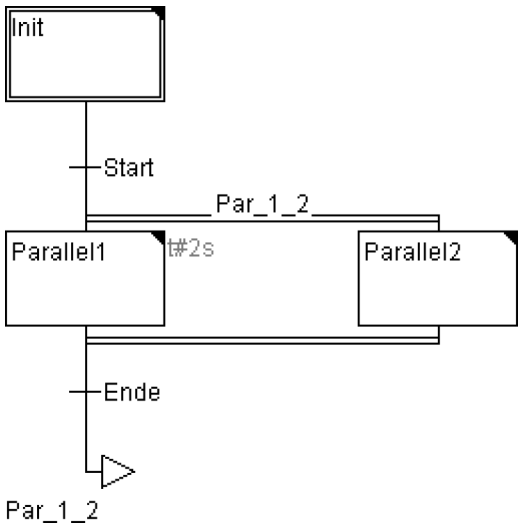
A step with an exit-action is designated by an "X" in the lower right corner.

'Extras' 'Paste Parallel Branch (right)'

This command pastes the contents of the clipboard as a right parallel branchof the marked block ↳ *Chapter 1.4.1.1.10.5.10 “Parallel branch” on page 176*. For this the marked block must both begin and end with a step. The contents of the clipboard must, likewise, be an SFC block that both begins and ends with a step.

'Extras' 'Add label to parallel branch'

In order to provide a newly inserted parallel branch with a jump label, the transition occurring before the parallel branching must be marked and the command 'Add label to parallel branch' must be executed. At that point, the parallel branch will be given a standard name consisting of "Parallel" and an appended serial number, which can be edited according to the rules for identifier names. In the following example, "Parallel" was replaced by "Par_1_2" and the jump to the transition "End" was steered to this jump label.



Delete a label

A jump label can be deleted by deleting the label name.

'Extras' 'Paste after'

This command pastes the SFC block on the clipboard after the first step or the first transition of the marked block. (Normal copying pastes it in front of the marked block.) This will now be executed, if the resulting SFC structure is correct, according to the language norms.

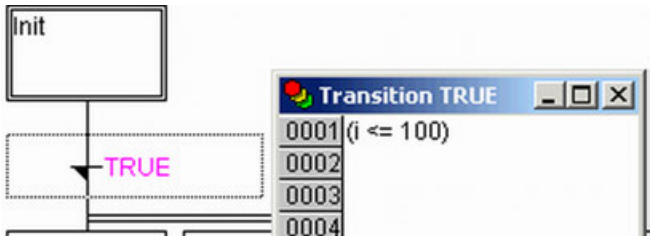
'Extras' 'Zoom Action/Transition'

Shortcut: <Alt>+<Enter>

The action of the first step of the marked block or the transition body of the first transition of the market block is loaded into the editor in the respective language, in which it has been written ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. If the action or the transition body is empty, then the language must be selected, in which it has been written.

Note that the transition condition which is written within the editor window will take precedence over a condition which might be written directly at the transition mark.

Example: If here i>100, then the transition condition will be FALSE, although TRUE has been entered at the mark!



'Extras' 'Clear Action/Transition'

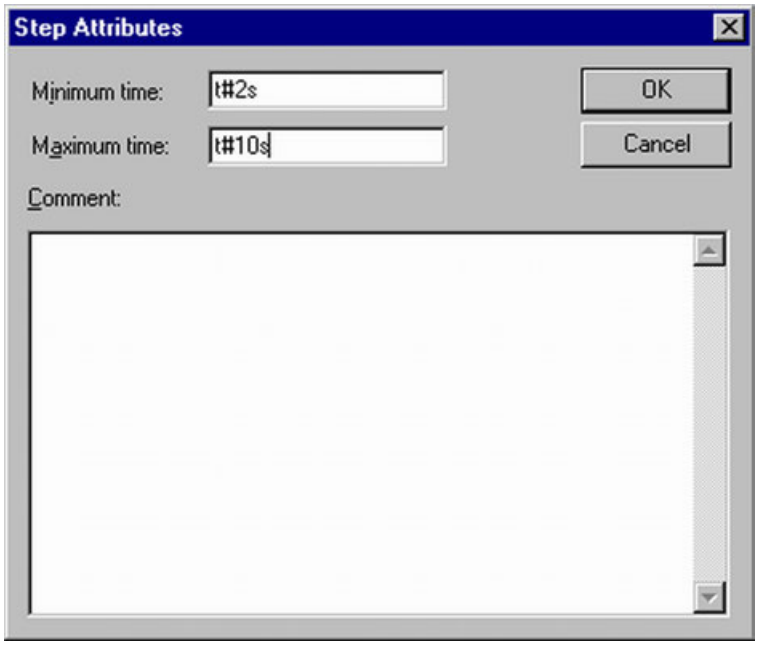
With this command you can delete the actions of the first step of the marked block or of the transitions body of the first transition ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*.

If, during a step, you implement either only the action, the entry-action, or the exit-action, then the same will be deleted by the command. Otherwise a dialog box appears, and you can select which action or actions are to be deleted.

If the cursor is located in the action of an IEC step, then only this association will be deleted ↳ *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172*. If an IEC step with an associated action is selected, then this association will be deleted ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171*. During an IEC step with several actions, a selection dialog box will appear.

'Extras' 'Step Attributes'

With this command you can open a dialog box in which you can edit the attributes for the marked step.



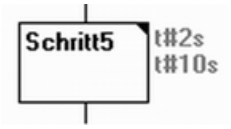
You can take advantage of three different entries in the step attribute dialog box. Under 'Minimum Time', you enter the minimum length of time that the processing of this step should take. Under the 'Maximum Time', you enter the maximum length of time that the processing of this step should take. Note that the entries are of the TIME type, so you use a TIME constant (i.e. T#3s) or a variable of the TIME type.

The time settings are also accessible in dialog 'Extras' 'Time Overview' ↳ *Chapter 1.4.1.3.11.9.21 “'Extras' 'Time Overview'” on page 335*.

Under 'Comment' you can insert a comment to the step. In the 'Sequential function chart options' dialog which you open under 'Extras' 'Options', you can then define whether comments, the time setting or nothing is displayed for the steps in the SFC editor. On the right, next to the step, either the comment or the time setting or none of both will appear.

Those attributes which are not displayed because of the options settings, can - additionally to the step name - be displayed in a tooltip, which appears when the cursor is placed on the upper resp. lower right corner of the step box *Chapter 1.4.1.3.11.9.22 “'Extras' 'Options'” on page 336*.

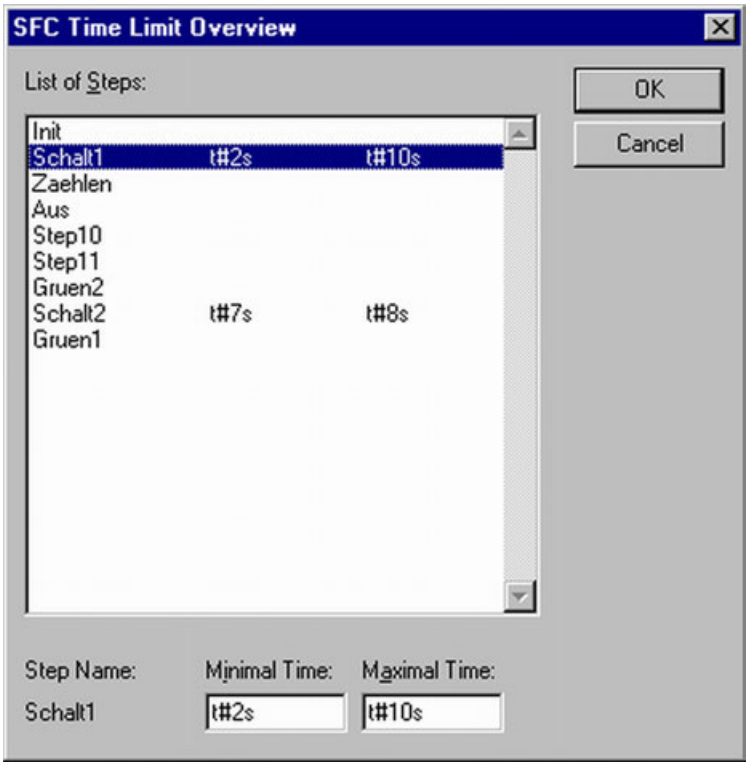
If the maximum time is exceeded, SFC flags are set which the user can query *Chapter 1.4.1.1.10.5.8 “SFC flags” on page 174*.



The example shows a step whose execution should last at least two, and at the most, ten seconds. In Online mode, there is, in addition to these two times, a display of how long the step has already been active.

'Extras' 'Time Overview'

With this command you can open a window in which you can edit the time settings of your SFC steps:



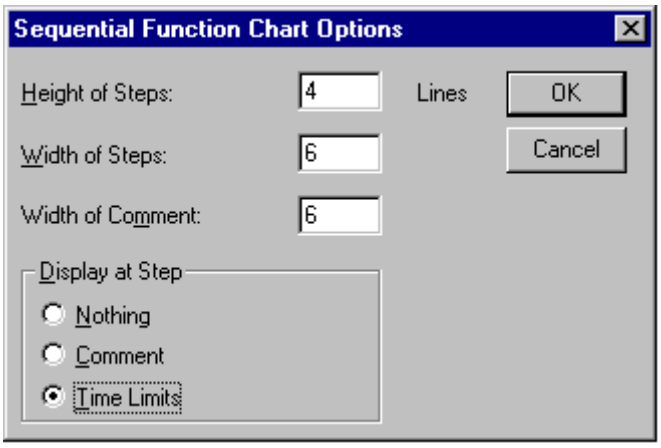
In the time boundaries overview, all steps of your SFC POU are displayed. If you have entered a time boundary for a step, then the time boundary is displayed to the right of the step (first, the lower limit, then the upper limit) *Chapter 1.4.1.3.11.9.20 “'Extras' 'Step Attributes'” on page 334*. You can also edit the time boundaries. To do so, click on the desired step in the overview. The name of the step is then shown below in the window. Go to the 'Minimum Time' or 'Maximum Time' field, and enter the desired time boundary there. If you close the window with OK, then all of the changes will be stored.

In the example, steps 2 and 6 have a time boundary. Shift1 lasts at least two, and at most, ten seconds. Shift2 lasts at least seven, and at most, eight seconds.

'Extras' 'Options'

With this command you open a dialog box in which you can set different options for your SFC POU.

Dialog Box for Sequential Function Chart Options:



Under 'Step Height', you can enter how many lines high an SFC step can be in your SFC editor. 4 is the standard setting here. Under 'Step Width', you can enter how many columns wide a step should be. 6 is the standard setting here.

You can also preset the 'Display at Step', i.e. which of the attributes, defined 'Extras' 'Step Attributes' should be displayed next to the step ↳ *Chapter 1.4.1.3.11.9.20 “'Extras' 'Step Attributes'” on page 334*. Either choose Comment, Time Limits, or Nothing:

- If "Nothing" is set, the defined comment and the time limits nevertheless can be shown in a tooltip, which appears when the cursor is placed on the lower right corner of the step box.
- If "Comment" is set, the defined comment and the time limits can be shown in a tooltip, which appears when the cursor is placed on the upper right corner of the step box.
- If "Time Limits" is set, the defined comment and the time limits nevertheless can be shown in a tooltip, which appears when the cursor is placed on the lower right corner of the step box.

'Extras' 'Associate Action'

With this command actions and Boolean variables can be associated with IEC steps ↳ *Chapter 1.4.1.1.10.5.2 “Action” on page 171* ↳ *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172*.

To the right of, and next to the IEC step, an additional divided box is attached, for the association of an action. It is preset in the left field with the qualifier "N" and the name "Action." Both presets can be changed. For this you can use the input assistant ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

Maximum nine actions can be assigned to an IEC step.

New actions for IEC steps are created in the Object Organizer for an SFC POU with the 'Project' 'Add Action' command.

'Extras' 'Use IEC-Steps'

Symbol:

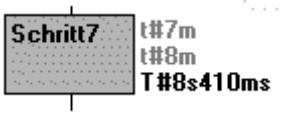
If this command is activated (denoted by a check in front of the menu item and a printed symbol in the Tool bar), then IEC steps will be inserted instead of the simplified steps upon insertion of step transitions and parallel branches ↳ *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172* ↳ *Chapter 1.4.1.1.10.5.4 “Transition / Transition condition” on page 172* ↳ *Chapter 1.4.1.1.10.5.10 “Parallel branch” on page 176*.

If this option is switched on, the Init step is set as an IEC step when you create a new SFC POU.

These settings are saved in the file "codesys.ini" and are restored when CODESYS gets started again.

Sequential function chart in online mode

With the Sequential Function Chart editor in Online mode, the currently active steps will be displayed in blue. If you have set it under 'Extras' 'Options', then the time management is depicted next to the steps ↳ *Chapter 1.4.1.3.11.9.22 “'Extras' 'Options'” on page 336*. Under the lower and upper bounds that you have set, a third time indicator will appear from which you can read how long the step has already been active.



In the figure above the step depicted has already been active 8 seconds and 410 milliseconds. The step must, however, be active for at least 7 minutes before the step will be left.

With 'Online' 'Toggle Breakpoint' a breakpoint can be set on a step, or in an action at the locations allowed by the language in use. Processing then stops prior to execution of this step or before the location of the action in the program. Steps or program locations where a breakpoint is set are marked in light blue.

If several steps are active in a parallel branch, then the active step whose action will be processed next is displayed in red.

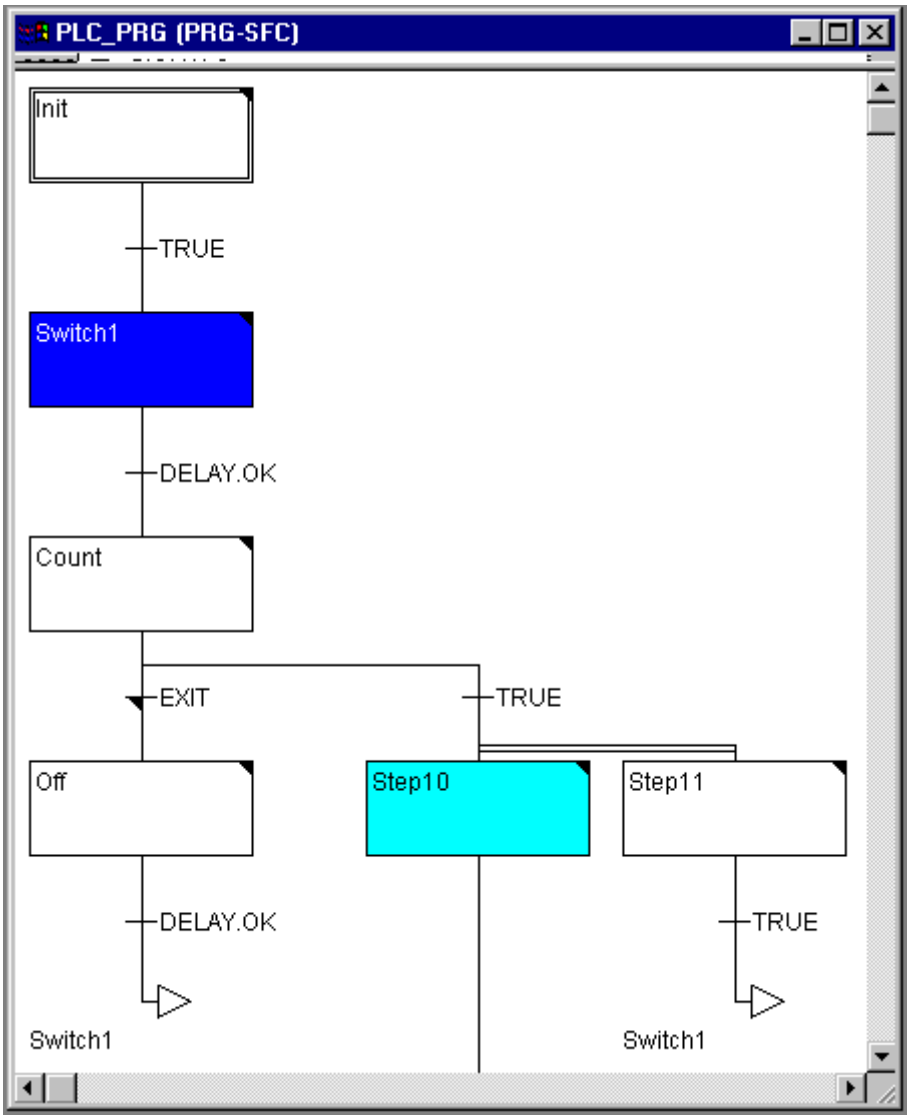
If IEC step has been used, then all active actions in Online mode will be displayed in blue ↳ *Chapter 1.4.1.1.10.5.6 “IEC step” on page 172*.

With the command 'Online' 'Step over' it is stepped always to the next step which action is executed. If the current location is:

- a step in the linear processing of a POU or a step in the rightmost parallel branch of a POU, execution returns from the SFC POU to the caller. If the POU is the main program, the next cycle begins.
- a step in a parallel branch other than the rightmost, execution jumps to the active step in the next parallel branch.
- the last breakpoint location within an action, execution jumps to the caller of the SFC.
- the last breakpoint location within an IEC action, execution jumps to the caller of the SFC.
- the last breakpoint position within an input action or output action, execution jumps to the next active step.

With 'Online' 'Step in' even actions can be stepped into. If an input, output or IEC action is to be jumped into, a breakpoint must be set there. Within the actions, all the debugging functionality of the corresponding editor is available to the user.

If you rest the mouse cursor for a short time on a variable in the declaration editor, the type, the address and the comment of the variable will be displayed in a tooltip.



If you rename a step and perform an Online Change while this step is active, the program will be stopped in undefined status !

Processing order of elements in a sequence:

- First, all Action Control Block flags in the IEC actions that are used in this sequence are reset (not, however, the flags of IEC actions that are called within actions).
- All steps are tested in the order which they assume in the sequence (top to bottom and left to right) to determine whether the requirement for execution of the output action is provided, and this is executed if that is the case.
- All steps are tested in the order which they assume in the sequence to determine whether the requirement for the input action is provided, and this is executed if that is the case.
- For all steps , the following is done in the order which they assume in the sequence:
  - If applicable, the elapsed time is copied into the corresponding step variable.
  - If applicable, any timeout is tested and the SFC error flags are serviced as required.
  - For non-IEC steps, the corresponding action is now executed.

- IEC actions that are used in the sequence are executed in alphabetical order. This is done in two passes through the list of actions. In the first pass, all the IEC actions that are deactivated in the current cycle are executed. In the second pass, all the IEC actions that are active in the current cycle are executed.
- Transitions are evaluated: If the step in the current cycle was active and the following transition returns TRUE (and if applicable the minimum active time has already elapsed), then the following step is activated.

The following must be noted concerning implementation of actions:

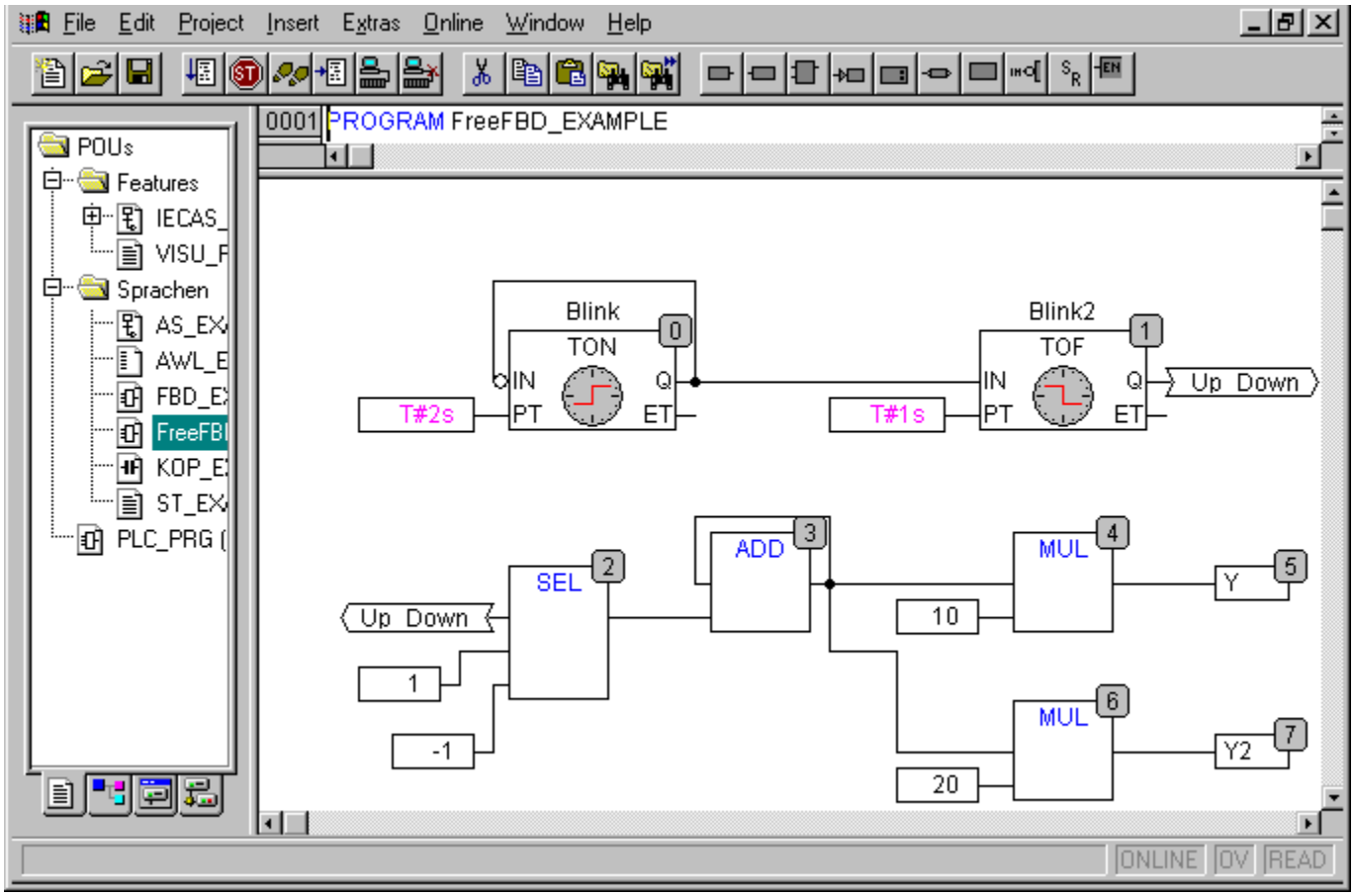
It can come about that an action is carried out several times in one cycle because it is associated with multiple sequences. (For example, an SFC could have two IEC actions A and B, which are both implemented in SFC, and which both call IEC action C; then in IEC actions A and B can both be active in the same cycle and furthermore in both actions IEC action C can be active; then C would be called twice).

If the same IEC action is used simultaneously in different levels of an SFC, this could lead to undesired effects due to the processing sequence described above. For this reason, an error message is issued in this case. It can possibly arise during processing of projects created with older versions of CODESYS.

> *In monitoring expressions (e.g. A AND B) in transitions, only the "Total value" of the transition is displayed.*

The continuous function chart editor (CFC)

Overview



No snap grid is used for the continuous function chart editor so the elements can be placed anywhere. Elements of the sequential processing list include boxes, input, output, jump, label, return and comments. The inputs and outputs of these elements can be connected by dragging a connection with the mouse. The connecting line will be drawn automatically. The shortest possible connection line is drawn taking into account existing connections. The connecting lines are automatically adjusted when the elements are moved. If the case arises where a connecting line cannot be drawn simply because of lack of space, a red line will be shown between the input and the associated output instead. This line will be converted into a connecting line just as soon as space is available.

One advantage of the continuous function chart as opposed to the usual Function Block Diagram Editor is the fact that feedback paths can be inserted directly.

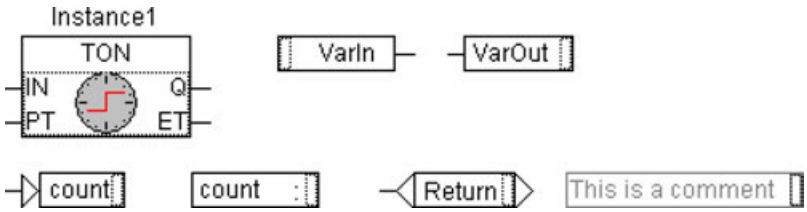
The most important commands can be found in the context menu.

Cursor positions in the CFC

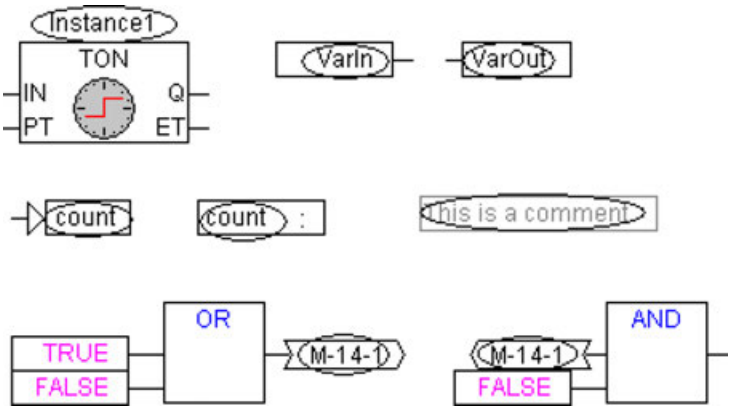
Each text is a possible cursor position. The selected text is shaded in blue and can be modified.

In all other cases the current cursor position is shown by a rectangle made up of points. The following is a list of all possible cursor positions with examples:

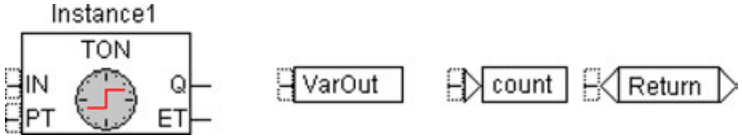
1. Trunks of the elements box, input, output, jump, label, return and comments:



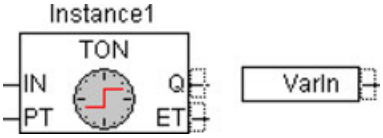
2. Text fields for the elements box, input, output, jump, label, return and comments as well as text fields for connection marker:



3. Inputs for the elements box, input, output, jump and return:



4. Outputs for the elements box and input:



'Insert' 'Box' in the CFC

Symbol: Shortcut: <Ctrl>+<B>

This command can be used to paste in operators, functions, function blocks and programs. First of all, it is always inserted an "AND" operator. This can be converted by Selection and Overwrite of the text into every other operator, into every function, into every function block and every program. The input assistance serves to select the desired block from the list of supported blocks. If the new block has another minimum number of inputs, these will be attached. If the new block has a smaller highest number of inputs, the last inputs will be deleted.

'Insert' 'Input' in CFC

Symbol: Shortcut: <Ctrl>+<E>

This command is used to insert an input. The text offered "???" can be selected and replaced by a variable or constant. The input assistance can also be used here.

'Insert' 'Output' in CFC

Symbol: Shortcut: <Ctrl>+<A>

This command is used to insert an output. The text offered "???" can be selected and replaced by a variable. The input assistance can also be used here. The value which is associated with the input of the output is allocated to this variable.

'Insert' 'Jump' in CFC

Symbol: Shortcut: <Ctrl>+<J>

This command is used to insert a jump. The text offered "???" can be selected and replaced by the jump label to which the program should jump.

The jump label is inserted using the command 'Insert 'Label'.

'Insert' 'Label' in CFC

Symbol: Shortcut: <Ctrl>+<L> This command is used to insert a label. The text offered "???" can be selected and replaced by the jump label. In Online mode a RETURN label for marking the end of POU is automatically inserted.

The jump is inserted using the command 'Insert 'Jump'.

'Insert' 'Return' in CFC

Symbol: Shortcut: <Ctrl>+<R>

This command is used to insert a RETURN command. Note that in Online mode a jump label with the name RETURN is automatically inserted in the first column and after the last element in the editor; in stepping, it is automatically jumped to before execution leaves the POU.

'Insert' 'Comment' in CFC

Symbol: Shortcut: <Ctrl> + <K>

This command is used to insert a comment. You obtain a new line within the comment with <Ctrl> + <Enter>.

'Insert' 'Input of box' in CFC

Shortcut: <Ctrl>+<U>

This command is used to insert an input at a box. The number of inputs is variable for many operators (e.g. ADD can have two or more inputs).

To increase the number of inputs for such an operator by one, the box itself must be selected.

Insert' 'In-Pin' in CFC, 'Insert' 'Out-Pin'

Symbol: 

These commands are available as soon as a macro is opened for editing. They are used for inserting in- or out-pins as in- and outputs of the macro. They differ from the normal in- and outputs of POUs by the way they are displayed and in that they have no position index.

'Extras' 'Negate' in CFC

Symbol:  Shortcut: <Ctrl> + <N>

This command is used to negate inputs, outputs, jumps or RETURN commands. The symbol for the negation is a small cross on the connection.

The input of the element block, output, jump or return is negated when it is selected.

The output of the element block or input is negated when it is selected (Cursor position 4).

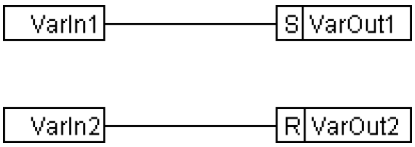
A negation can be deleted by negating again.

'Extras' 'Set/Reset' in CFC

Symbol:  Shortcut: <Ctrl> + <T>

This command can only be used for selected inputs of the element output .

The symbol for Set is S and for Reset is R.



VarOut1 is set to TRUE, if VarIn1 delivers TRUE. VarOut1 retains this value, even when VarIn1 springs back to FALSE.

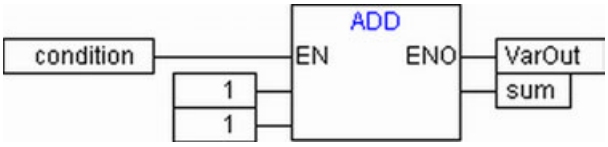
VarOut2 is set to FALSE, if VarIn2 delivers TRUE. VarOut2 retains this value, even when VarIn2 springs back to FALSE.

Multiple activation of this command causes the output to change between Set, Reset and the normal condition.

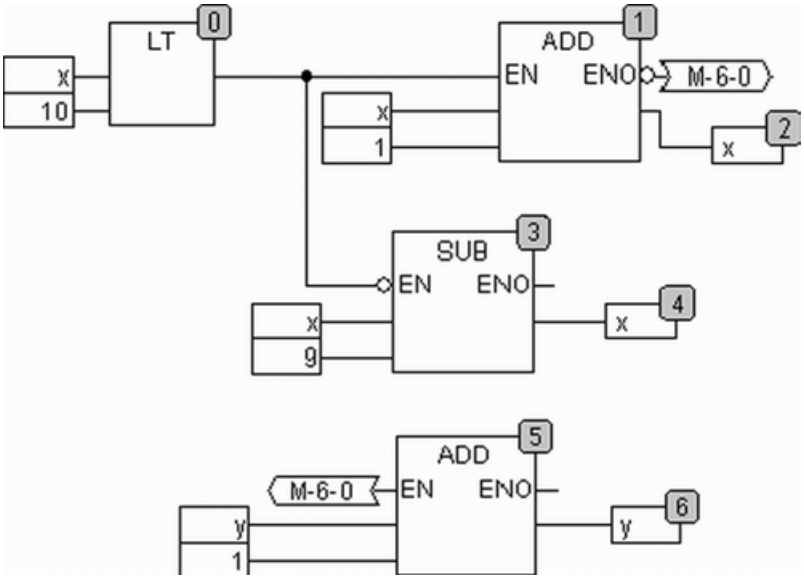
'Extras' 'EN/ENO' in CFC

Symbol:  Shortcut: <Ctrl> + <I>

This command is used to give a selected block (Cursor position 3) an additional Boolean enable input EN (Enable In) and a Boolean output ENO (Enable Out).



ADD is only executed in the example when the Boolean variable "condition" is TRUE. VarOut will also be set to TRUE after the execution of ADD. But if afterwords condition changes to FALSE, ADD will not be executed any more and thus VarOut remains TRUE! The example below shows how the value ENO can be used for further blocks:

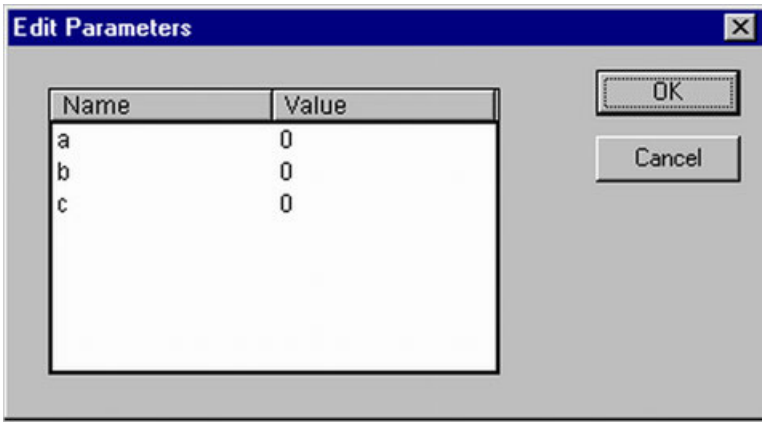


x should initialised to 1 and y initialised to 0. The numbers in the right corner of the block indicate the order in which the commands are executed.

x will be increased by one until it reaches the value 10. This causes the output of the block LT(0) to deliver the value FALSE and SUB(3) and ADD(5) will be executed. x is set back to the value 1 and y is increased by 1. LT(0) is executed again as long as x is smaller than 10. y thus count the number of times x passes though the range of values 1 to 10.

'Extras' 'Properties...' in CFC

Constant input parameters (VAR_INPUT CONSTANT) from functions ↳ *Chapter 1.4.1.2.3.34 “'Project' 'Project info'” on page 246* and function blocks ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153* are not shown directly in the continuous function chart editor. These can be shown and their value can be changed when one selects the trunk of the block in question and then selects the command 'Extras' 'Properties' or simply double clicks on the trunk. The 'Edit parameters' dialog opens:



The values of the constant input parameter (VAR_INPUT CONSTANT) can be changed. Here it is necessary to mark the parameter value in the column Value. Another mouse click or pressing on the space bar allows this to be edited. Confirmation of the change to the value is made by pressing the <Enter> key or pressing <Escape> rejects the changes. The button 'OK' stores all of the changes which were made.

This functionality and the associated declaration of variables with keyword "VAR_INPUT CONSTANT" is only of impact for the CFC editor. In the FBD editor always all INPUT variables will be displayed at a box, no matter whether declared as VAR_INPUT or VAR_INPUT CONSTANT. Also for text editors this does not make any difference.

Moving/Copying elements in CFC

One or more selected elements can be moved with the arrow keys + <Shift> key. Another possibility is to move elements using a depressed left mousekey. These elements are placed by releasing the left mousekey in as far as they do not cover other elements or exceed the foreseen size of the editor. The marked element jumps back to its initial position in such cases and a warning tone sounds.

One or more selected elements can be copied with the command 'Edit' 'Copy' and inserted with the command 'Edit' 'Paste'.

Creating connections

An input of an element can be precisely connected to the output of another element. An output of an element can be connected to the inputs of a number of other elements.

There are a number of possibilities to connect the input of an element E2 with the output of an element E1.

Place the mouse on the output of element E1, click with the left mousekey, hold the left mousekey down and drag the mouse cursor onto the input of element E2 and let the left mousekey go. A connection is made from the output of element E1 to the mouse cursor during this dragging operation with the mouse.

Place the mouse on the input of element E2, click with the left mousekey, hold the left mousekey down and drag the mouse cursor onto the output of element E1 and let the left mousekey go.

Move one of the elements E1 or E2 and place it in such a way by letting go of the left mousekey that the output of element E2 and the input of element E1 touch.

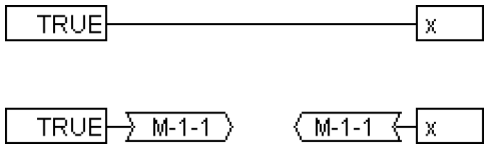
Where element E2 is a block with a free input, a connection can also be made by dragging the mouse from an output from E1 to the trunk of E2. A connection with the free input at the highest position on E2 will be created when the mousekey is released. In the case where block E2 does not have a free input but is an operator which can have an input added to it, a new input will be automatically generated.

The output and input of a block can be connected together (feedback path) by using this method. To establish a connection between two pins, click with the left mouse button on one pin, hold the button down and thus drag the connection to the desired pin, where you then release the button. If during the dragging of the connection extends outside working area of the editor, scrolling occurs automatically. For simple data types, type testing is carried out during the connection. If the types of the two pins are not compatible, the cursor changes to "Forbidden". For complex data types, no testing takes place.

'Extras' 'Connection marker'

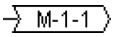
Connections can also be represented by a connector (connection marker) instead of a connecting line. Here the output and the associated input have a connector added to them which is given a unique name.

Where a connection already exists between the two elements which should now be represented by connectors, the output of the connecting line is marked and the menu point 'Extras' 'Connection marker' is selected. The following diagram shows a connection before and after the selection of this menu point.

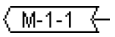


A unique name is given as standard by the program which begins with M, but which can be changed The connector name is stored as an output parameter, but can be edited both at the input and at the output.

It is important to know that the connector name is associated with a property of the output of a connection and is stored with it.

1. Edit the connector at the output: 

If the text in the connector is replaced, the new connector name is adopted by all associated connectors at the inputs. One cannot, however, select a name which already belongs to another connection marker since the uniqueness of the connector name would be violated.

2. Edit the connector at the input: 

If the text in a connector is replaced, it will also be replaced in the corresponding connection marker on the other POU. Connections in connector representations can be converted to normal connections in that one marks the output of the connections (Cursor position 4) and again selects the menu point 'Extras' 'Connection marker'.

Changing connections

A connection between the output of an element E1 and the input of an element E2 can easily be changed into a connection between the output of element E1 and the input of element E3. The mouse is clicked on the input of E2, the left mousekey is kept depressed, the mouse cursor is moved to the input of E3 and then released.

Deleting connections

There are a number of possibilities for removing the connection between the output of an element E1 and the input of an element E2:

- Select the output of element E1 and press the <Delete> key or execute the command 'Edit' 'Delete'. Several connections will be removed at the same if the output of E1 is connected to more than one of inputs.
- Select the input of element E2 and press the <Delete> key or execute the command 'Edit' 'Delete'.
- Select the input of E2 with the mouse, hold the left mousekey depressed and drag the connection from the input to E2 away. The connection is removed when the left mousekey is released in a free area of the screen.

Insert inputs/outputs on the fly

If exactly one input or output pin of an element is selected, then the corresponding input- or output- element can be directly inserted and its editor field filled with a string by entering the string at the keyboard.

Order of execution

The elements block, output, jump, return and label each possess a number indicating the order in which they are executed. In this sequential order the individual elements are evaluated at run time.

When pasting in an element the number is automatically given according to the topological sequence (from left to right and from above to below). The new element receives the number of its topological successor if the sequence has already been changed and all higher numbers are increased by one.

The number of an element remains constant when it is moved.

The sequence influences the result and must be changed in certain cases.

If the sequence is displayed, the corresponding sequential execution number is shown in the upper right hand corner of the element.

'Extras' 'Order' 'Show Order'

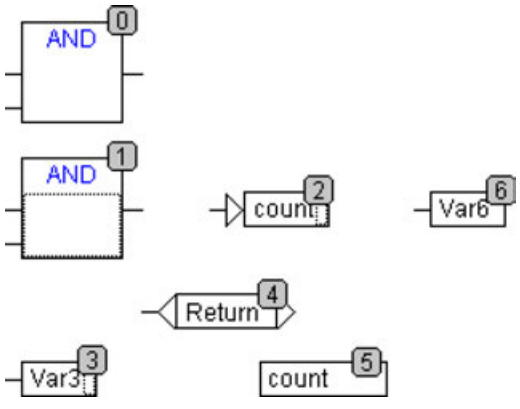
This command switches the display of the order of execution on and off. The default setting is to show it (recognised by a tick (✓) in front of the menu point).

The relevant order of execution number appears in the upper right hand corner for the elements block, output, jump, return and label.

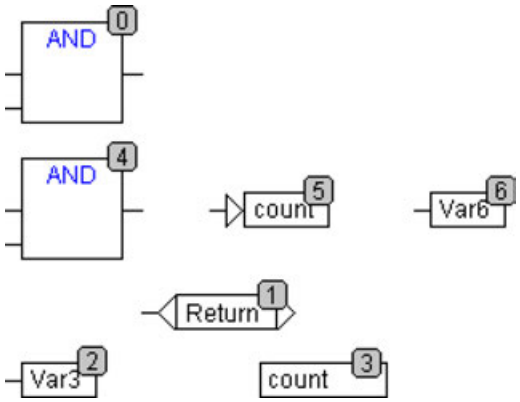
'Extras' 'Order' 'Order topologically'

Elements are ordered in a topological sequence when the execution takes place from left to right and from above to below, that is the number increases from left to right and from above to below for topologically arranged elements. The connections are not relevant, only the location of the elements is important.

All selected elements are topologically arranged when the command 'Extras' 'Order' 'Order topologically' is executed. All elements in the selection are taken out of the sequential processing list by this process. The elements are then entered into the remaining sequential processing list individually from bottom right through to upper left. Each marked element is entered into the sequential processing list before its topological successor, i.e. it is inserted before the element that in a topological sequencing would be executed after it, when all elements in the editor were sequenced according to a topological sequencing system. This will be clarified by an example.



The elements with numbers 1, 2 and 3 are selected. If the command 'Order topologically' is selected the elements are first taken out of the sequential processing list. Var3, the jump and the AND-operator are then inserted again one after the other. Var3 is placed before the label and receives the number 2. The jump is then ordered and receives the number 4 at first but this then becomes 5 after the AND is inserted. The new order of execution which arises is:



When a newly generated block is introduced it will be placed by default in front of its topological successor in the sequential processing list.

'Extras' 'Order' 'One up'

With this command all selected elements with the exception of the element which is at the beginning of the sequential processing list are moved one place forwards in the sequential processing list.

'Extras' 'Order' 'One down'

With this command all selected elements with the exception of the element which is at the end of the sequential processing list are moved one place backwards in the sequential processing list.

'Extras' 'Order' 'Start'

With this command all selected elements will be moved to the front of the sequential processing list whereby the order within the group of selected elements is maintained. The order within the group of unselected elements also remains the same.

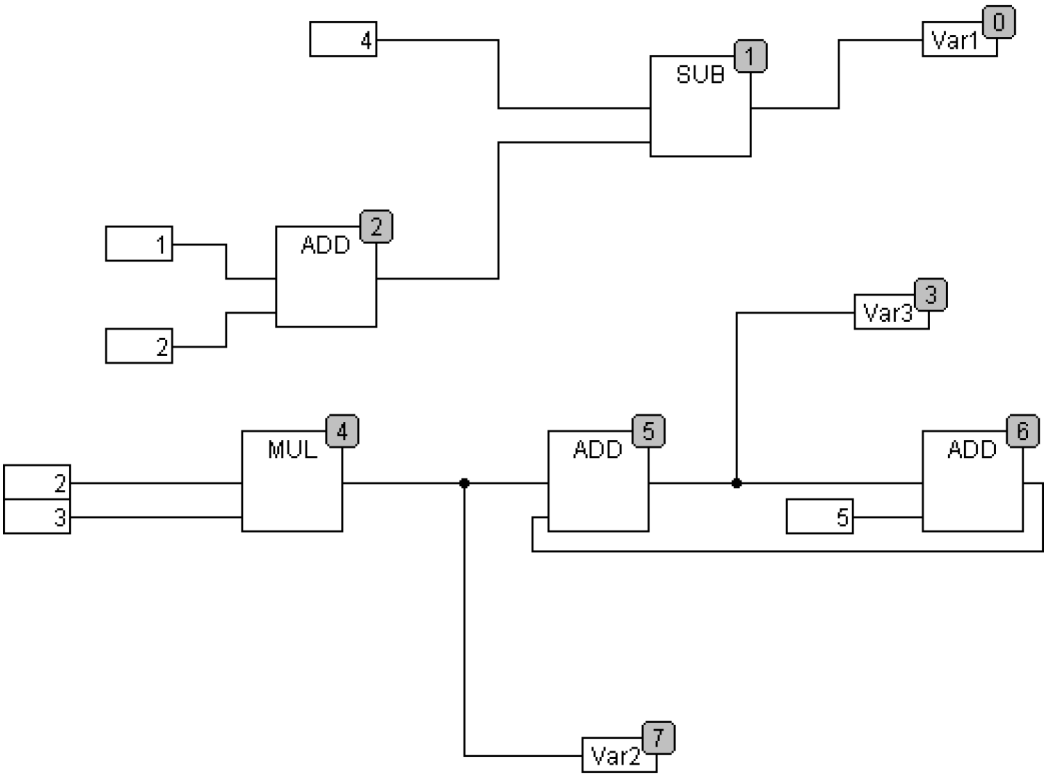
'Extras' 'Order' 'End'

With this command all selected elements will be moved to the end of the sequential processing list whereby the order within the group of selected elements is maintained. The order within the group of unselected elements also remains the same.

'Extras' 'Order' 'Order everything according to data flow'

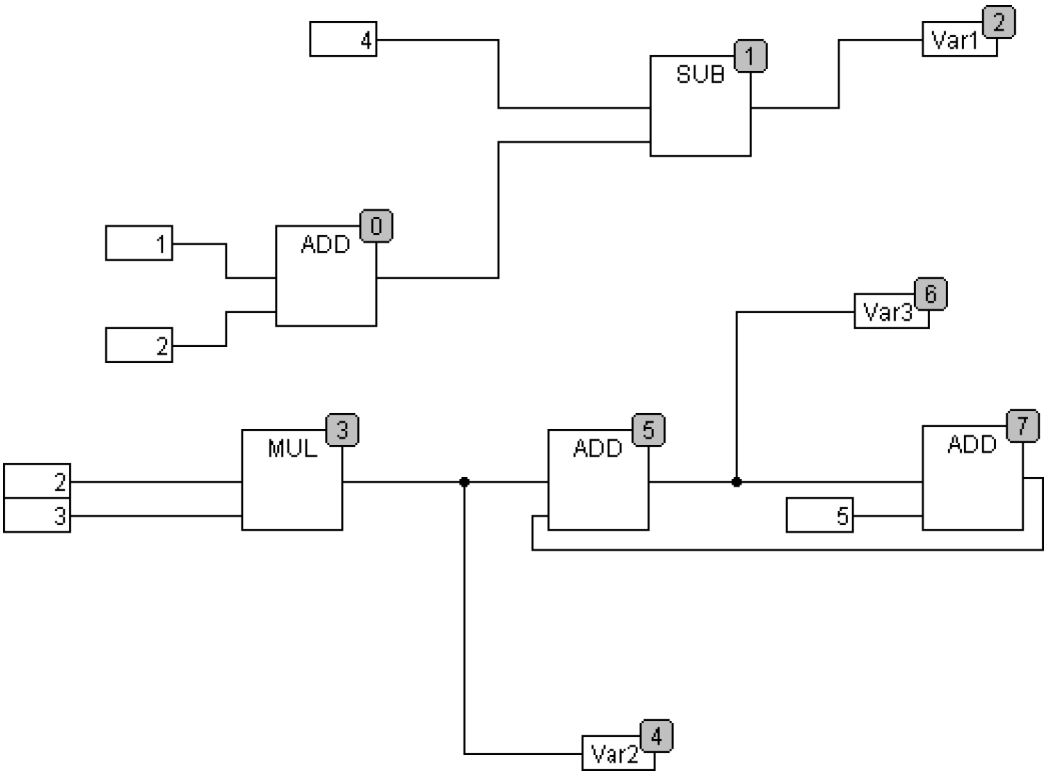
This command effects all elements. The order of execution is determined by the data flow of the elements and not by their position. The diagram below shows elements which have been ordered topographically.

Sequence before the ordering according to data flow:



The following arrangement exists after selecting the command.

Sequence after the ordering according to data flow:



When this command is selected the first thing to happen is that the elements are ordered topographically. A new sequential processing list is then created. Based on the known values of the inputs, the computer calculates which of the as yet not numbered elements can be processed next. In the above "network" the block AND, for example, could be processed immediately since the values at its inputs (1 and 2) are known. Block SUB can only then be processed since the result from ADD must be known first, etc.

Feedback paths are inserted last *Chapter 1.4.1.3.11.10.34 “Feedback paths in CFC” on page 350*.

The advantage of the data flow sequencing is that an output box which is connected to the output of a block comes immediately after it in the data flow sequencing system which by topological ordering would not always be the case. The topological ordering can deliver another result in some cases than ordering by data flow, a point which one can recognise from the above example.

'Extras' 'Create macro"

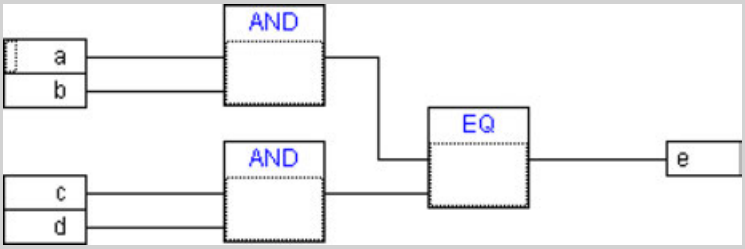
Symbol: 

With this command, several POUs that are selected at the same time can be assembled into a block, which can be named as a macro. Macros only can be reproduced by Copy/Paste, whereby each copy becomes a separate macro whose name can be chosen independently. Macros are thus not references. All connections that are cut by the creation of a macro generate in- or out-pins on the macro. Connections to inputs generate an in-pin. The default name appears next to the pin in the form In<n>. For connections to outputs, Out<n> appears. Affected connections which had connection markers prior to the creation of the macro, retain the connection marker on the PIN of the macro.

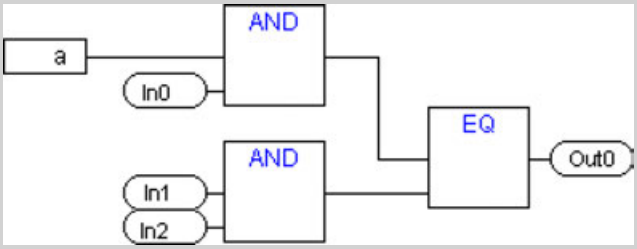
At first, a macro has the default name "MACRO". This can be changed in the Name field of the macro use. If the macro is edited, the name of the macro will be displayed in the title bar of the editor window appended to the POU name.

Example:

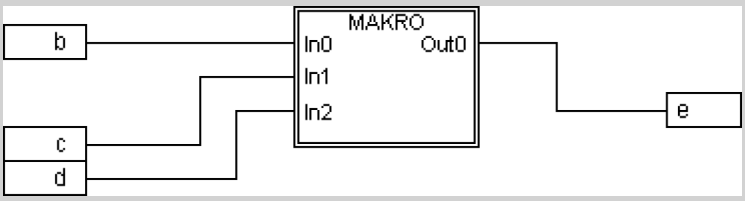
Selection:



Macro:



In the editor:



'Extras' 'Edit Macro'

Symbol: 

By this command, or by double clicking on the body of the macro, the macro is opened for editing in the editor window of the associated POU *Chapter 1.4.1.3.11.10.30 “'Extras' 'Create macro'” on page 349*. The name of the macro is displayed appended to the POU name in the title bar.

The pin boxes generated for the in- and outputs of the macro during creation can be handled like normal POU in- and outputs. They can also be moved, deleted, added, etc. They differ only in how they are displayed and have no position index. For adding you can use the buttons  (input) or  (output), which are available in the menu bar. Pin boxes have rounded corners. The text in the pin-box matches the name of the pin in the macro display.

The order of the pins in the macro box follows the order of execution of the elements of the macro. A lower order index before a higher one, higher pin before lower.

The processing order within the macro is closed, in other words the macro is processed as a block, at the position of the macro in the primary POU. Commands for manipulating the order of execution therefore operate only within the macro.

'Extras' 'Expand macro'

With this command, the selected macro is re-expanded and the elements contained in it are inserted in the POU at the macro's location *Chapter 1.4.1.3.11.10.30 “'Extras' 'Create macro'” on page 349*. The connections to the pins of the macro are again displayed as connections to the in- or outputs of the elements. If the expansion of the macro can not occur at the location of the macro box for lack of space, the macro is displaced to the right and down until enough space is available.

If the project is saved under project version number 2.1, the macros will likewise all be expanded. All macros will also be expanded before conversion into other languages.

'Extras' 'Back one macro level', 'Extras' 'Back all macro level'

Symbols: 

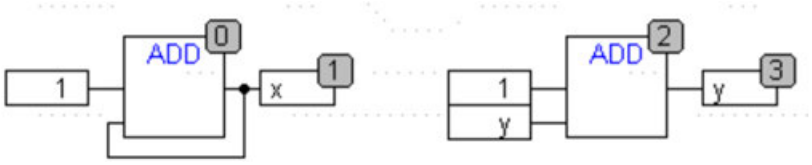
These commands are also available in the toolbar, as soon as a macro is opened for editing *Chapter 1.4.1.3.11.10.30 “'Extras' 'Create macro'” on page 349*. If macros are nested within one another, it is possible to switch to the next higher or to the highest display level.

Feedback paths in CFC

Feedback paths can only be displayed directly in the continuous function chart editor and not in the usual Function Block Diagram Editor. Here it should be observed that the output of a block always carries an internal intermediate variable. The data type of the intermediate variable results, for operators, from the largest data type of the inputs.

The data type of a constant is obtained from the smallest possible data type, that is the constant '1' adopts the data type SINT. If now an addition with feedback and the constant '1' is executed, the first input gives the data type SINT and the second is undefined because of the feedback. Thus the intermediate variable is also of the type SINT. The value of the intermediate variable is only then allocated to the output variable.

The diagram below shows an addition with feedback and an addition with a variable. The variables x and y should be of the type INT here.



There are differences between the two additions:

The variable y can be initialised with a value which is not equal to zero but this is not the case for intermediate variable for the left addition.

The intermediate variable for the left addition has the data type SINT while that on the right has the data type INT. The variables x and y have different values after the 129th call up. The variable x, although it is of the type INT, contains the value 127 because the intermediate variable has gone into overflow. The variable y contains the value 129, on the other hand.

Zoom to POU

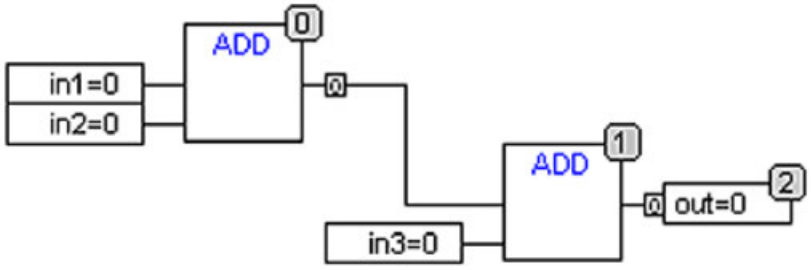
Shortcut: <Alt>+<Enter>

With this command a selected POU is loaded into its editor. The command is available in the context menu (<F2>) or in the 'Extras' menu, if the cursor is positioned on the name of a POU in a text editor or if the POU box is selected in a graphic editor.

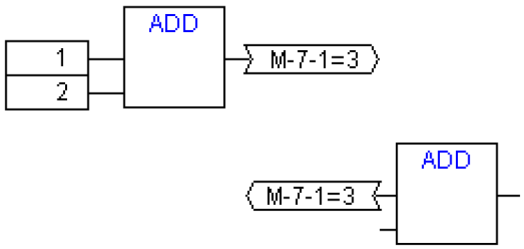
If you are dealing with a POU from a library, then the library manager is called up, and the corresponding POU is displayed.

CFC in Online mode

Monitoring: The values for inputs and outputs are displayed within the input or output boxes. Constants are not monitored. For non-boolean variables, the boxes are expanded to accommodated the values displayed. For boolean connections, the variable name as well as the connection are displayed in blue if the value is TRUE, otherwise they remain black. Internal boolean connections are also displayed Online in blue in the TRUE state, otherwise black. The value of internal non-boolean connections is displayed in a small box with rounded corners on the output pin of the connection.



Pins in macros are monitored like in- or output boxes.



Non-boolean connections with connection markers display their value within the connection marker. For boolean connections, the lines as well as the marker names are displayed in blue if the line is carrying the value TRUE, otherwise black.

Flow control: When flow control is switched on, the connections that have been traversed are marked with the color selected in the project options.

Breakpoints: Breakpoints can be set on all elements that also have a processing sequence order index ↳ *Chapter 1.4.1.1.11.3 “Breakpoint” on page 182*. The processing of the program will be halted prior to execution of the respective element, that is for POUs and outputs before the assignment of inputs, for jump labels before execution of the element with the next index. The processing sequence index of the element is used as the breakpoint position in the Breakpoint dialog.

The setting of breakpoints on a selected element is accomplished with the F9 key or via the menu item 'Breakpoint on/off' in the 'Online' or 'Extras' menu or in the editor's context menu. If a breakpoint is set on an element, then this will be erased and reversed the next time the command 'Breakpoint on/off' is executed. In addition, the breakpoint on an element can be toggled by double-clicking on it.

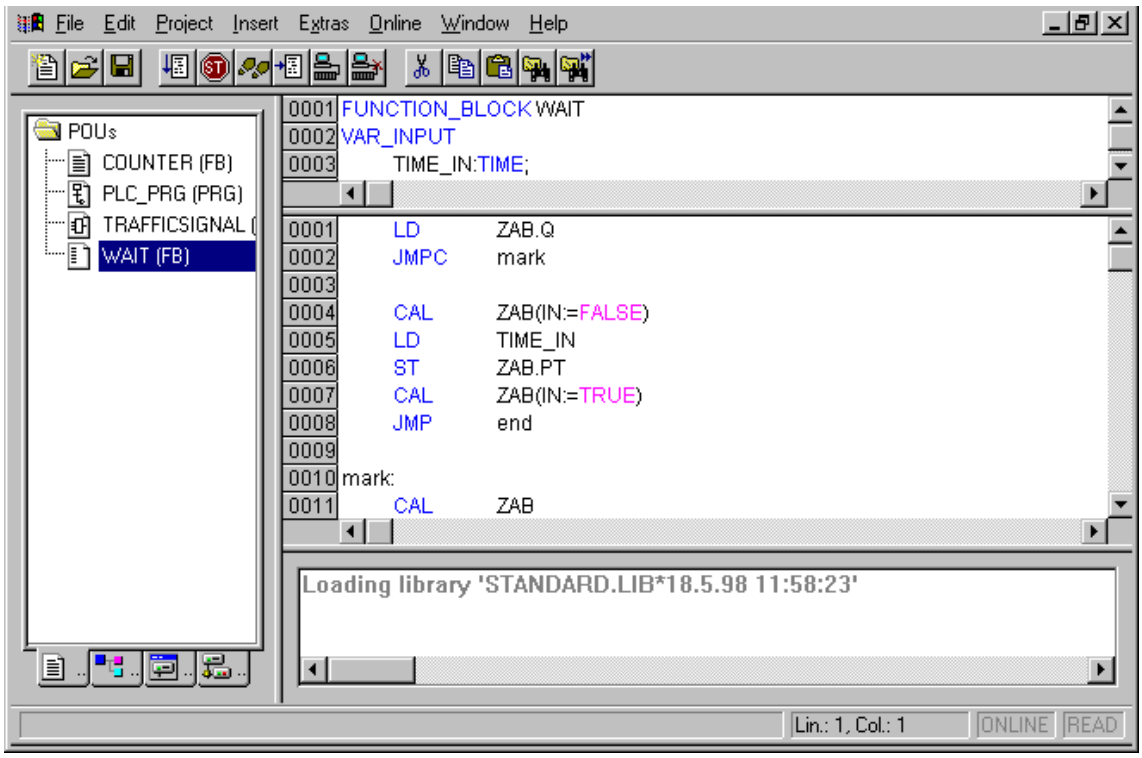
Breakpoints are displayed in the colors entered in the project options.

RETURN label: In Online mode, a jump label with the name „RETURN" is automatically generated in the first column and after the last element in the editor. This label marks the end of the POU and is jumped to when stepping just before execution leaves the POU. No RETURN marks are inserted in macros.

Stepping: When using 'Step over' the element with the next-higher order index will always be jumped to ↳ *Chapter 1.4.1.2.6.13 “'Online' 'Step over'” on page 286*. If the current element is a macro or a POU, then its implement branches when 'Step in' is in effect ↳ *Chapter 1.4.1.2.6.14 “'Online' 'Step in'” on page 286*. If a 'Step over' is executed from there, the element whose order index follows that of the macro is jumped to.

1.4.1.3.12 The text editors

Overview



The text editors used for the implementation portion (the Instruction List editor and the Structured Text editor) provide the usual Windows text editor functions.

The implementation in the text editors is supported by syntax coloring.

In Overwrite mode the status bar shows a black OV. You can switch between Overwrite mode and Insert mode by key <Ins>

The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

'Insert''Operators'

With this command all of the operators available in the current language are displayed in a dialog box.

If one of the operators is selected and the list is closed with OK, then the highlighted operator will be inserted at the present cursor position. (This is managed here just as it is in the input assistant).

'Insert' 'Operand'

With this command all variables in a dialog box are displayed. You can select whether you would like to display a list of the global, the local, or the system variables.

If one of the operands is chosen, and the dialog box is closed with OK, then the highlighted operand will be inserted at the present cursor position. (This is managed here just as it is in the input assistant).

'Insert' 'Function'

With this command all functions will be displayed in a dialog box ↳ *Chapter 1.4.1.1.9.3 “Function” on page 151*. You can choose whether to have a list displaying user-defined or standard functions.

If one of the functions is selected and the dialog box is closed with OK, then the highlighted function will be inserted at the current cursor position. (The management will proceed, as in the input selection.)

If the 'With arguments' option was selected in the dialog box, then the necessary input and output variables will also be inserted.

'Insert' 'Function Block'

With this command all function blocks are displayed in a dialog box ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*. You can choose whether to have a list displaying user-defined or standard function blocks.

If one of the function blocks is selected and the dialog box is closed with OK, then the highlighted function block will be inserted at the current cursor position. (This is managed here just as it is in the input assistant).

If the 'With arguments' option was selected in the dialog box, then the necessary input variables of the function block will also be inserted. However you are not forced to assign these parameters.

Calling POUs with output parameters in text editors

The output parameters of a called POU can be directly assigned upon being called in the text languages IL and ST.

Example: Output parameter out1 of afbinst is assigned variable a.

IL:

```
CAL afbinst(in1:=1, out1=>a)
```

ST:

```
afbinst(in1:=1, out1=>a);
```

If the POU is inserted via input assistant (<F2>) with option 'With arguments' in the implementation window of a ST or IL POU, it will automatically be displayed with all parameters in this syntax. However you are not forced to assign these parameters.

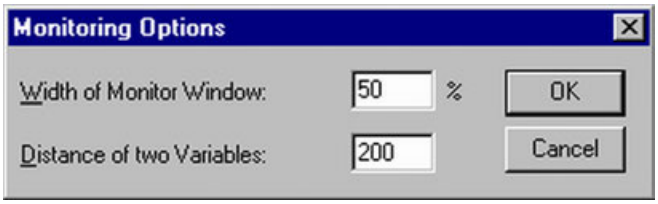
'Extras' 'Monitoring Options'

With this command you can configure your monitoring window. In the text editors, the window is divided into two halves during monitoring. The program is located in the left half. In the right half, all variables that are located in the corresponding program line are monitored.

You can specify the Monitor Window Width and which Distance two variables should have in a line. An distance declaration of 1 corresponds, in this case, to a line height in the selected font.

Regard that the width of the window halves also can be modified by drawing the divider with the mouse.

Monitoring Options Dialog Box:



Breakpoint positions

Breakpoint positions include all positions in a program at which values of variables can change or where the program flow branches off. (Exception: function calls. If necessary, a breakpoint in the function must be set here.) At the positions lying inbetween, a breakpoint would not even make sense, since nothing has been able to change in the data since the preceding breakpoint position.

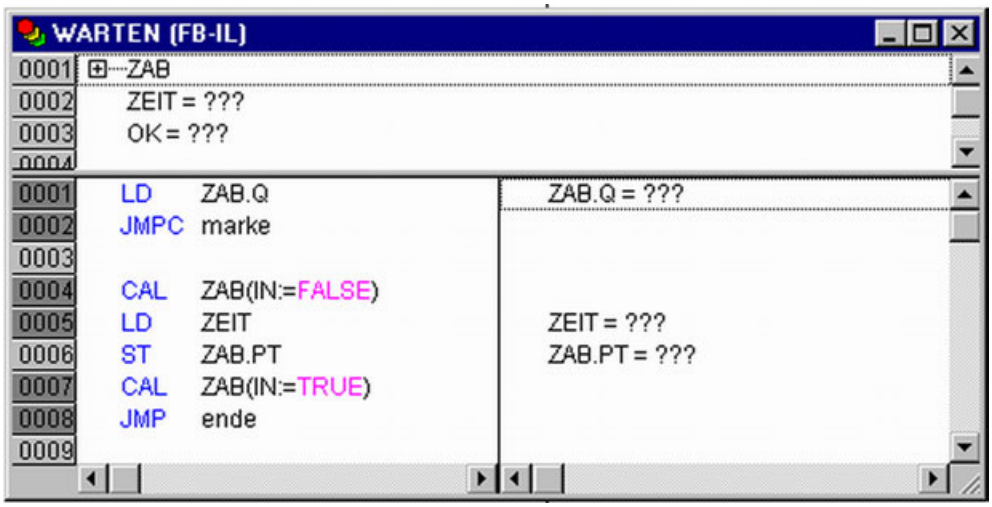
This results in the following breakpoint positions in the IL:

- At the start of the POU
- At every LD, LDN (or, in case a LD is located at a label, then at the label)
- At every JMP, JMPC, JMPCN
- At every label
- At every CAL, CALC, CALCN
- At every RET, RETC, RETCN
- At the end of the POU

Structured Text accommodates the following breakpoint positions:

- At every assignment
- At every RETURN and EXIT instruction
- in lines where conditions are being evaluated (WHILE, IF, REPEAT)
- At the end of the POU

Breakpoint positions are marked by darker line number fields (in the color which is set in the project options):



How do you set a breakpoint?

In order to set a breakpoint, click the line number field of the line where you want to set a breakpoint. If the selected field is a breakpoint position, then the color of the line numbers field will change from dark gray to light blue, and the breakpoint will be activated in the PLC.

Deleting breakpoints

Correspondingly, in order to delete a breakpoint, click on the line number field of the line with the breakpoint to be deleted.

Setting and deleting of breakpoints can also be selected via the menu ('Online' 'Toggle Breakpoint'), via the function key <F9>, or via the symbol in the tool bar ↬ *Chapter 1.4.1.2.6.11 "'Online' 'Toggle breakpoint'" on page 285*.

Line number of the text editor

The line numbers of the text editor give the number of each text line of an implementation of a POU.

In Off-line mode, a simple click on a special line number will mark the entire text line.

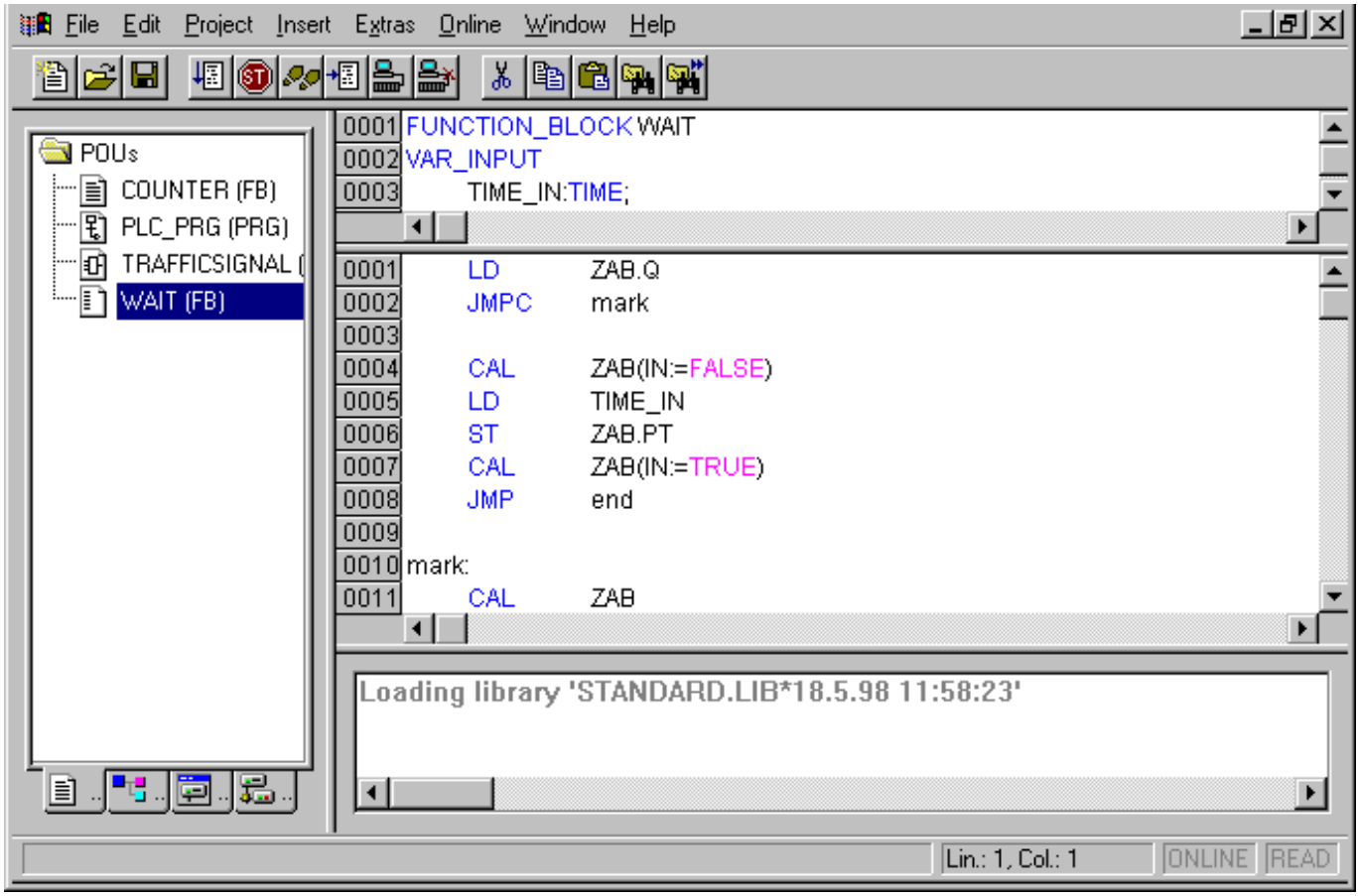
In Online mode, the background color of the line number indicates the breakpoint status of every line. The standard settings for the colors are

- dark gray: This line is a possible position for a breakpoint.
- light blue: a breakpoint has been set in this line.
- red: The program has reached this point.

In Online mode, simply clicking the mouse will change the breakpoint status of this line.

The instruction list editor

Overview



All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The Instruction List editor is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>). Multiline POU calls are also possible.

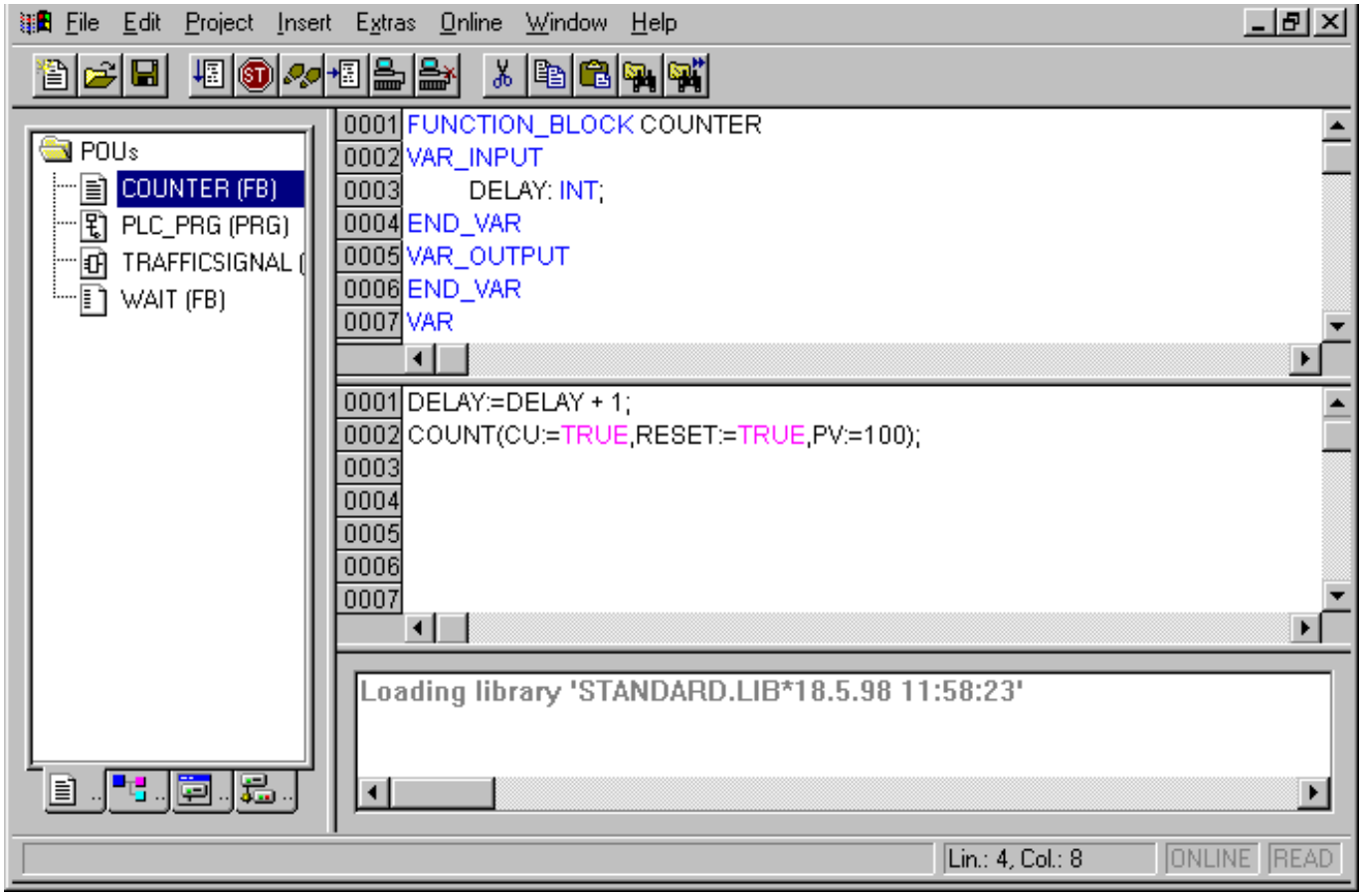
Example

```
CAL CTU_inst(
CU:=%IX10,
PV:=(
LD A
ADD 5
)
)
```

IL in online mode

With the 'Online' 'Flow control' command ↳ *Chapter 1.4.1.2.6.21 “'Online' 'Display flow control'” on page 290*, an additional field in which the accumulator contents is displayed is inserted in the IL editor on the left side of every line.

The structured text editor



All editors for POUs consist of a declaration part and a body. These are separated by a screen divider.

The editor for Structured Text is a text editor with the usual capabilities of Windows text editors. The most important commands are found in the context menu (right mouse button or <Ctrl>+<F10>).

1.4.1.4 The 'Resources' tab

1.4.1.4.1 Global variables, variable configuration, document frame

'Global Variables' folder

Objects in 'Global Variables' folder

In the Object Organizer, you will find two objects in the *“Resources”* register card in the *“Global Variables”* folder (default names of the objects in parentheses).

- Global Variables ↯ *Chapter 1.4.1.4.1.2 “Global variables” on page 357*
- Variables Configuration ↯ *Chapter 1.4.1.4.1.4.1 “Overview” on page 361*

All variables defined in these objects are recognized throughout the project. If the Global Variables folder is not opened, you can open it with a double-click *[Enter]* in the line.

Select the corresponding object. The *“'Object Open'”* command opens a window with the previously defined global variables. The editor for this works the same way as the declaration editor.

Global variables

Normal variables, constants or remanent variables that are known throughout the project can be declared as global variables.

In a project you can define a local variable which has the same name as a global variable. In this case within a POU the locally defined variable will be used.

It is not allowed to name two global variables identically. For example, you will get a compiler error if you have defined a variable "var1" in the PLC Configuration as well as in a global variables list.

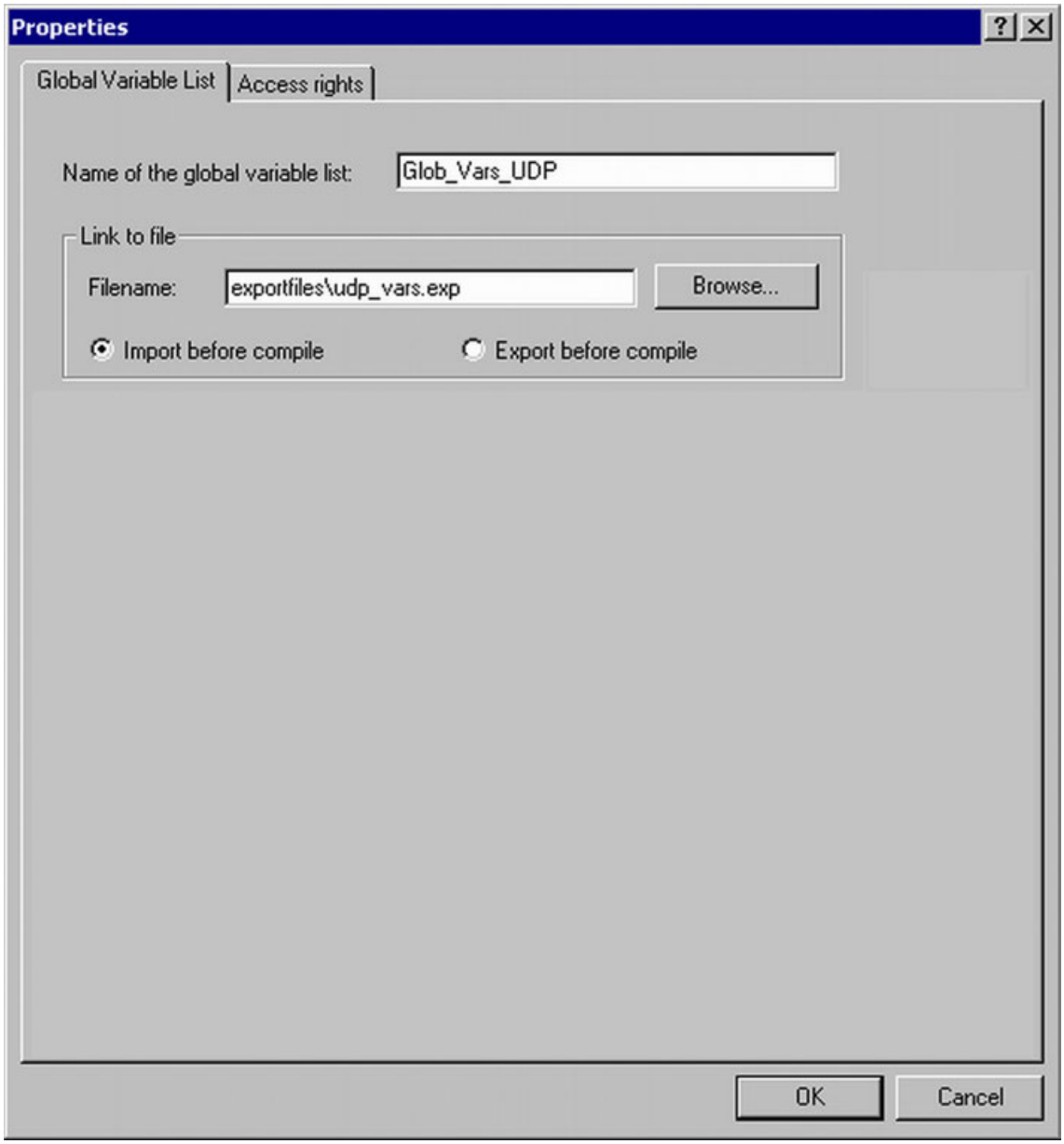
Global variable lists

Create a global variable list

To create a Global Variable List, open the register 'Resources' in the Object Organizer and select the entry 'Global Variables' or select an already existing list. Then choose the command 'Project' 'Object' 'Add'to open the dialog Global variable list ↬ *Chapter 1.4.1.2.4.6 “'Project' 'Object' ' Add'” on page 258*.

This dialog can also be opened by the command 'Project' 'Object' 'Properties' which is available if an existing Global Variable List is marked in the object organizer ↬ *Chapter 1.4.1.2.4.12 “'Project' 'Object properties'” on page 261*. It shows the configuration of this list.

Dialog to create a new Global Variable List:



Name of the global variable list: Insert a list name.

Link to file:

Filename: If you have an export file (*.exp) or a DCF file, which contains the desired variables, you can set up a link to this file. To do this, insert the path of the file in the field Filename resp. press the button Browse to get the standard dialog 'Select text file'. DCF files are converted to ICE syntax when they are read in.

Activate option Import before compile, if you wish that the variable list will be read from the external file before each compilation of the project. Activate the option Export before compile, if you want the variable list to be written to the external file before each compilation of the project.

If you close the 'Global variable list' dialog with OK, the new object is created. Global variables lists can be recognized in the Object Organizer by the symbol . With the command 'Project' 'Object' 'Properties' you can re-open the 'Global variable list' configuration dialog for the entry marked in the Object Organizer.

Several variables lists

Global variables, global network variables (VAR_GLOBAL) and variable configurations (VAR_CONFIG) must be defined in separate objects.

If you have declared a large number of global variables, and you would like to structure your global variables list better, then you can create further variables lists.

In the Object Organizer, select the Global Variables folder or one of the existing objects with global variables. Then execute the 'Project' 'Object Add' command. Give the object that appears in the dialog box a corresponding name. With this name an additional object will be created with the key word VAR_GLOBAL. If you prefer an object a variable configuration, change the corresponding key word to VAR_CONFIG.

Editing global variable lists

The editor for global variables works similar to the declaration editor. But note that you cannot edit in this editor a list, which is an figure of an linked external variable list. External variable lists only can be edited externally and they will be read at each opening and compiling of the project.

Syntax:

```
VAR_GLOBAL
 (* Variables declarations *)
END_VAR
```

Editing remanent global variables lists

If they are supported by the runtime system, remanent variables may be processed. There are two types of remanent global variables:

- Retain variables remain unchanged after an uncontrolled shutdown of the runtime system (off/on) or an 'Online' 'Reset' *Chapter 1.4.1.2.6.8 “'Online' 'Reset'” on page 284*.
- Persistent variables remain only unchanged after a program download *Chapter 1.4.1.2.6.5 “'Online' 'Download'” on page 283*.

Persistent variables are not automatically also Retain variables !

Remanent variables are additionally assigned the keyword RETAIN and/or PERSISTENT.

Syntax:

```
VAR_GLOBAL RETAIN
 (* Variables declarations *)
END_VAR
VAR_GLOBAL PERSISTENT
 (* Variables declarations *)
END_VAR
```

For the combination of retain and persistent properties both keywords are used:

```
 VAR_GLOBAL RETAIN PERSISTENT
```

or

```
 VAR_GLOBAL PERSISTENT RETAIN
```

Global constants

Global constants additionally get the keyword CONSTANT.

Syntax:

```
VAR_GLOBAL CONSTANT
 (* Variables declarations *)
END_VAR
```

Variable configuration

Overview

In function blocks it is possible to specify addresses for inputs and outputs that are not completely defined, if you put the variable definitions between the key words VAR and END_VAR. Addresses not completely defined are identified with an asterisk.

Example

```
FUNCTION_BLOCK locio
VAR
 loci AT %I*: BOOL := TRUE;
 loco AT %Q*: BOOL;
END_VAR
```

Here two local I/O-variables are defined, a local-In (%I*) and a local-Out (%Q*).

If you want to configure local I/Os for variables configuration in the Object Organizer in the Resources register card, the object Variable_Configuration will generally be available. The object then can be renamed and other objects can be created for the variables configuration.

The editor for variables configuration works like the declaration editor.

Variables for local I/O-configurations must be located between the key words VAR_CONFIG and END_VAR.

The name of such a variable consists of a complete instance path through which the individual POUs and instance names are separated from one another by periods. The declaration must contain an address whose class (input/output) corresponds to that of the incompletely specified address (%I*, %Q*) in the function block. Also the data type must agree with the declaration in the function block.

Configuration variables, whose instance path is invalid because the instance does not exist, are also denoted as errors. On the other hand, an error is also reported if no configuration exists for an instance variable. In order to receive a list of all necessary configuration variables, the "All Instance Paths" menu item in the 'Insert' menu can be used.

Example for a variable configuration

Assume that the following definition for a function block is given in a program:

```
VAR
Hugo: locio;
Otto: locio;
END_VAR
```

Then a corrected variable configuration would look this way:

```
VAR_CONFIG
PLC_PRG.Hugo.loci AT %IX1.0 : BOOL;
PLC_PRG.Hugo.loco AT %QX0.0 : BOOL;
PLC_PRG.Otto.loci AT %IX1.0 : BOOL;
PLC_PRG.Otto.loco AT %QX0.3 : BOOL;
END_VAR
```

'Insert' 'All Instance Paths'

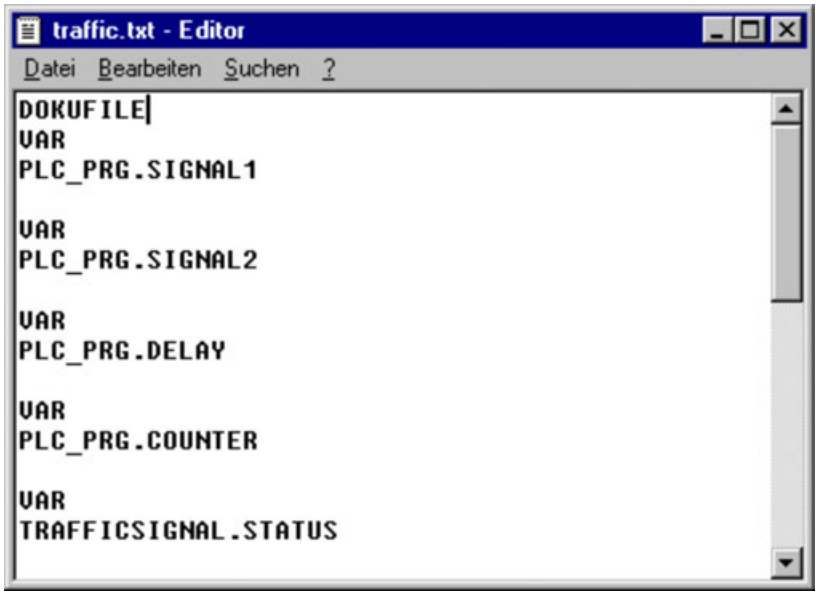
With this command a VAR_CONFIG - END_VAR-block is generated that contains all of the instance paths available in the project. Declarations already on hand do not need to be reinserted in order to contain addresses already in existence. This menu item can be found in the window for configuration of variables if the project is compiled ↳ *Chapter 1.4.1.2.3.12 "'Project' 'Rebuild all'" on page 232*.

Document frame

Overview

If a project is to receive multiple documentations, perhaps with German and English comments, or if you want to document several similar projects that use the same variable names, then you can save yourself a lot of work by creating a docuframe with the 'Extras' 'Make Docuframe File' command *Chapter 1.4.1.4.1.5.2 “'Extras' 'Make Docuframe File'” on page 362*.

The created file can be loaded into a desired text editor and can be edited. The file begins with the DOCUFILE line. Then a listing of the project variables follows in an arrangement that assigns three lines to each variable: a VAR line that shows when a new variable comes; next, a line with the name of the variable; and, finally, an empty line. You can now replace this line by using a comment to the variable. You can simply delete any variables that you are unable to document. If you want, you can create several document frames for your project.



In order to use a document frame, give the 'Extras' 'Link Docu File' command *Chapter 1.4.1.4.1.5.3 “'Extras' 'Link Docu File'” on page 362*. Now if you document the entire project, or print parts of your project, then in the program text, there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

'Extras' 'Make Docuframe File'

Use this command to create a document frame. The command is at your disposal, whenever you select an object from the global variables. A dialog box will open for saving files under a new name. In the field for the name file, the *.txt extension has already been entered. Select a desired name. Now a text file has been created in which all the variables of your project are listed.

'Extras' 'Link Docu File'

With this command you can select a document frame.

The dialog box for opening files is opened. Choose the desired document frame and press OK. Now if you document the entire project, or print parts of your project, then in the program text there will be an insertion of the comment produced in the docuframe into all of the variables. This comment only appears in the printout!

To create a document frame, use the 'Extras' 'Make Docuframe File' command *Chapter 1.4.1.4.1.5.2 “'Extras' 'Make Docuframe File'” on page 362*.

1.4.1.4.2 Alarm configuration

Overview

The alarm system allows to detect critical process states, to record them and to visualize them for the user with the aid of a visualization element. The alarm handling can be done in Automation Builder or alternatively in the PLC. For alarm handling in the PLC please see the target settings in category 'Visualization' ↳ *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*.

If supported by the target system, the dialogs for 'Alarm configuration' are available in the 'Resources' tab.

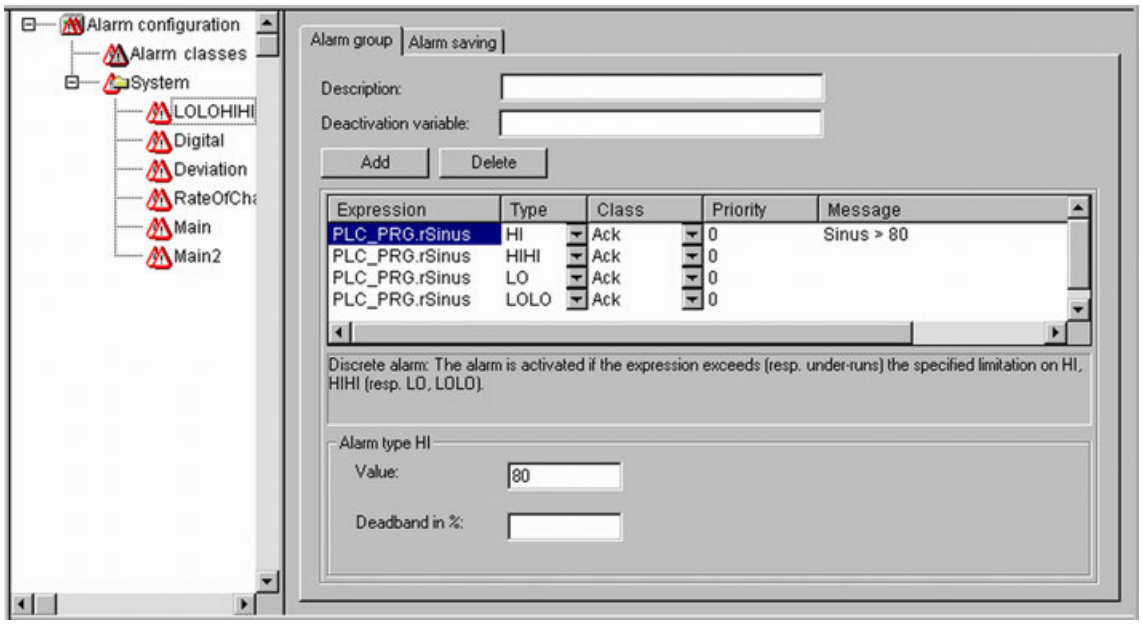
Here you define alarm classes and alarm groups. An alarm class serves for the typing of an alarm, that means it assigns certain parameters to the alarm. An alarm group serves for the concrete configuration of one or several alarms (which get assigned a certain class and further parameters) for the use in the project. Thus a class is useful for structuring the available alarms. The different alarm groups are defined by the user by inserting appropriate entries below the header 'System' in the configuration tree.

For the visualization of alarms the alarm table element is available in the visualization ↳ *Chapter 1.4.3.5.26 “Alarm table” on page 684*. Using this table the user can watch and acknowledge alarms. The language of the displayed alarm message texts might be switched dynamically.

If a history, i.e. recording of alarm events should be written to a log-file, such a file must be defined and for each alarm group the saving behavior must be defined.

When you open the 'Alarm configuration' in the Resources tab, the dialog 'Alarm configuration' opens with a bi-partited window, which concerning the mode of operation is similar to that of the PLC Configuration or Task configuration. In the left part the configuration tree is displayed, in the right part the appropriate configuration dialog will be opened.

Example of an Alarm configuration :



Open by a mouse-click on the plus sign at the entry 'Alarm configuration' the currently available configuration tree. If you are going to create a new configuration, this tree only will show the entries 'Alarm classes' and 'System'.

See also:

↳ *Chapter 1.4.1.4.2.2 “General information on alarms, terms” on page 364*

↳ *Chapter 1.4.1.4.2.3 “Alarm classes” on page 364*

↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*

↳ *Chapter 1.4.1.4.2.5 “Alarm saving” on page 369*

General information on alarms, terms

The usage of an alarm system obeys the following universal descriptions and definitions concerning alarms:

- Alarm: Generally an alarm is regarded as a special condition (expression value).
- Priority: The priority, also named "severity", of an alarm describes how important (severe) the alarm condition is. The highest priority is "0", the lowest valid priority value is "255".
- Alarm state: An expression/variable configured for the alarm control can have the following states: NORM (no alarm), INTO (alarm just has come), ACK (alarm has come and has been acknowledged by the user), OUTOF (alarm state has been terminated, alarm "has gone", but not yet acknowledged !)
- Sub-State: An alarm condition can have limits (Lo, Hi) and "extreme" limits (LoLo, HiHi). Example: The value of an expression ascends and first will transit the HI-limit, thus causing the coming of an HI-alarm. If the value continues ascending and exceeds also the HIHI-limit before the alarm gets acknowledged by the user, then the HI-alarm will get acknowledged automatically and just the HIHI-alarm remains in the alarm list (which is an internal list used for alarm administration). The HI-state in this case is named sub-state.
- Acknowledgement of alarms: The main purpose of alarms is to inform the user on alarm situations. In doing so it often is necessary to make sure that the user has noticed this information (see possible actions assigned to an alarm in the alarm class configuration). The user must acknowledge the alarm in order to get the alarm removed from the alarm list.
- Alarm Event: An alarm event must not be mixed up with an alarm condition. While an alarm condition can be valid for a longer period of time, an alarm event just describes the momentary occurrence of an change, e.g. a change from the normal state to the alarm state. In the alarm configuration for the three types of events and the corresponding alarm states the same names are used (INTO, ACK, OUTOF).

Supported features:

- Deactivation of the alarm generation for single alarms as well as for alarm groups
- Selection of the alarms which should be displayed by defining alarm groups and priorities
- Saving of all alarm events in an alarm table
- Visualization element 'Alarm table' in the visualization

See also:

Alarm configuration overview ↳ *Chapter 1.4.1.4.2.1 “Overview” on page 363*

Alarm classes ↳ *Chapter 1.4.1.4.2.3 “Alarm classes” on page 364*

Alarm groups ↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*

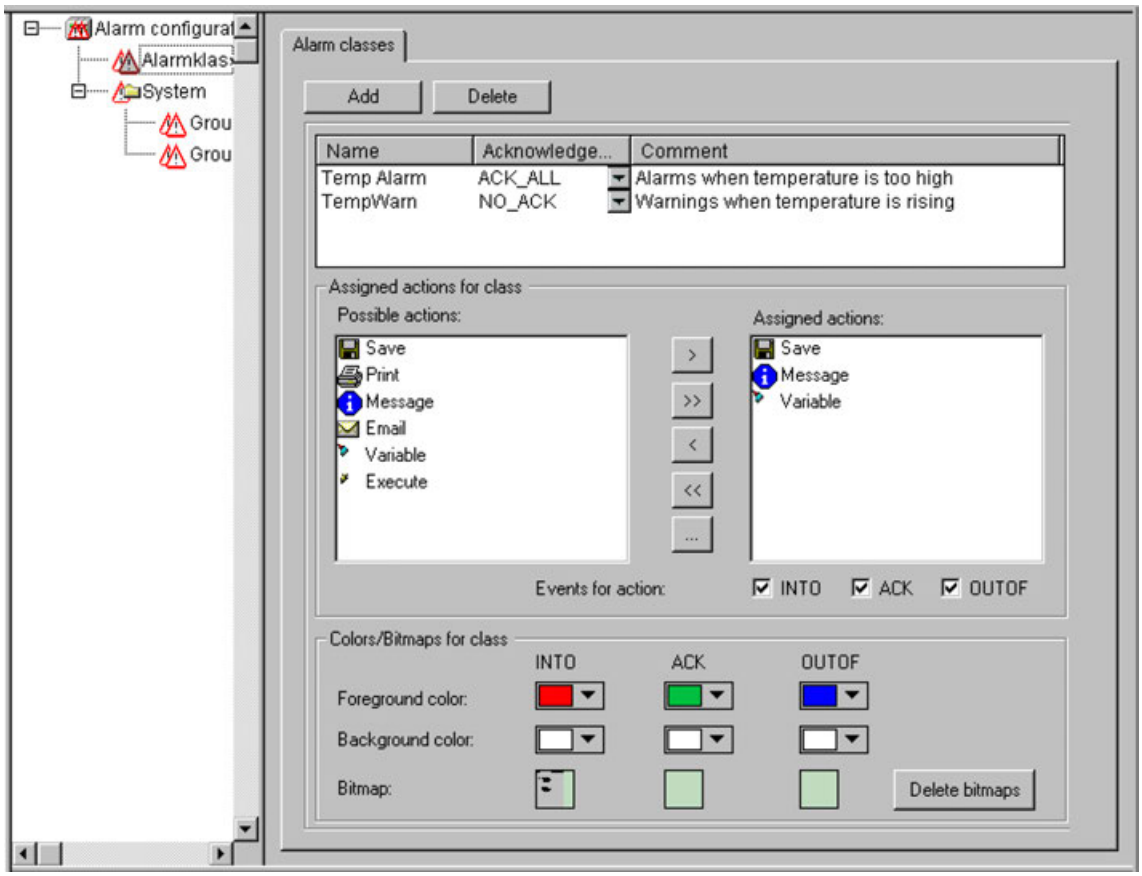
Alarm saving ↳ *Chapter 1.4.1.4.2.5 “Alarm saving” on page 369*

Alarm classes

Alarm classes are used for the general description of certain alarm criteria, such as how to handle acknowledgements (confirmation of an alarm by the user), which actions should automatically run as soon as a particular alarm state has been detected and which colors and bitmaps are to be used for a visualization of an alarm table ↳ *Chapter 1.4.1.4.2.2 “General information on alarms, terms” on page 364*↳ *Chapter 1.4.3.5.26 “Alarm table” on page 684*. Alarm classes are defined globally in the alarm configuration and are then available as a base configuration when configuring alarm groups ↳ *Chapter 1.4.1.4.2.1 “Overview” on page 363* ↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*.

Configuration of alarm classes

Select entry 'Alarm classes' in the alarm configuration tree. The configuration dialog 'Alarm classes' gets opened:



Press button 'Add' in order to create a new alarm class. Thereupon in the upper window a line will be inserted, primarily only with an entry "NOACK" (no acknowledgement) in the 'Acknowledgement' column. Define a name for the alarm class in the corresponding field in the 'Name' column (open an edit frame by a mouse-click on the field) and if necessary modify the acknowledgement type in column 'Acknowledgement'. The following acknowledgements are available:

- NO_ACK: No acknowledgement of the alarm by the user is required
- ACK_INTO: A "come" alarm condition (status "INTO", alarm occurs) must be confirmed by the user.
- ACK_OUTOF: A "gone alarm" (status "OUTOF", alarm terminated) must be confirmed by the user.
- ACK_ALL: Gone and come alarm conditions must be confirmed by the user ↳ *Chapter 1.4.1.4.2.2 “General information on alarms, terms” on page 364*.

Additionally you can enter a Comment.

Entries for further alarm classes each will be added at the end of the list.

Use button Delete to remove the currently selected entry from the list.

Assigned actions for class <class name>

Each alarm class defined in the upper window can get assigned a list of actions, which should be performed as soon as an alarm event occurs ↳ *Chapter 1.4.1.4.2.2 “General information on alarms, terms” on page 364*.

In the list of Possible actions select one and press button ">" to get it into the field Assigned actions. This field will finally contain the selection of actions assigned to the alarm class. Via button ">>" you can add all actions at a single blow. Via "<" resp.. "<<" you can remove one or all actions from the done existing selection. If an action is marked in the 'Assigned actions' list, via "..." a corresponding dialog can be opened to define the desired E-Mail settings, the printer settings, the process variable resp. the executable program and, if applicable, a message text.

The following action types (Possible actions) are supported (for a definition of a message text see below):

Action	Description	Settings to be done in the corresponding dialog
Save	The alarm event will be saved internally, in order to be given out e.g.in a log-file ↳ *Chapter 1.4.1.4.2.5 “Alarm saving” on page 369*. Please note: In this case the log-file must be defined in the configuration of the alarm groups ↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*!	The settings are done in the Alarm group definition in the Alarm saving dialog ↳ *Chapter 1.4.1.4.2.5 “Alarm saving” on page 369*
Print	A message text is sent to a printer.	Printer: Select one of the printers defined on the local system; Outputtext: Message text (see below) which should be printed out ● Please note that this function is not supported for target visualization!
Message	In the current visualization of the alarm a message window will be opened showing the defined text ↳ *Chapter 1.4.3.5.26 “Alarm table” on page 684*.	Message: Message text to be displayed in the message window ● Please note that this function is not supported for target visualization!
E-Mail	An E-Mail containing the defined message will be sent.	From: E-Mail address of sender; To: E-Mail address of recipient; Subject: any subject; Message: Message text (see below); Server: Name of the E-Mail server
Variable	A variable of the program will get the alarm status resp. a message text string.	Variable: Variable name: You can select project variables via the input assistant (<F2>): A boolean variable will indicate the alarm states NORM =0 and INTO=1, an integer variable will indicate the alarm states NORM =0, INTO =1, ACK =2, OUTOF =4; a string variable will get the message text defined in field; Message (see below)
Execute	An executable file will be started as soon as the alarm event occurs.	Executable file: name of the file to be executed (e.g. notepad.exe, you can use the "..." button to get the standard dialog for selecting a file; Parameter: appropriate parameter(s) which should be attached to the call of the exe-file

Definition of the message text

For action types 'Message', 'Print', 'Email' or 'Variable' you can define a message text which should be output in case of an Alarm Event. Line breaks at the text definitions in 'Message', 'Email' or 'Variable' can be inserted by <Ctrl>+<Enter>.

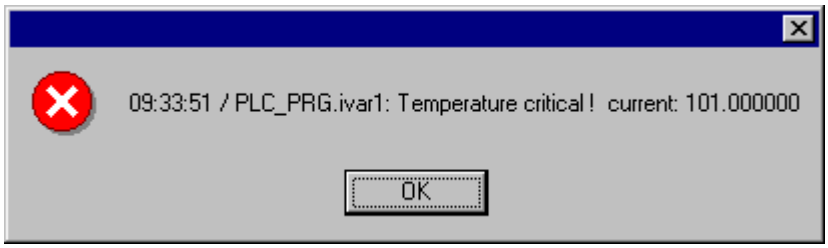
The following placeholders can be used when defining the alarm message:

MESSAGE	The message text which is defined for the particular alarm in the configuration of the alarm group will be used ↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*.
DATE	Date, when the alarm status was reached (INTO).
TIME	Time of alarm entry.
EXPRESSION	Expression (defined in alarm group) which has caused the alarm.
PRIORITY	Priority of the alarm (defined for alarm group.)
VALUE	Current value of the expression (see above).
TYPE	Alarm type (defined in alarm group)
CLASS	Alarm class (defined in alarm group)
TARGETVALUE	Target value for alarm types DEV+ and DEV- (defined in alarm group)
DEADBAND	Tolerance of the alarm (defined in alarm group)
ALLDEFAULT	Any information on the alarm will be output, like described for the line entries in a log file (History).

Example of defining an alarm message:

For a definition of a message box enter the following in the message window:

Further on when defining the alarm in the alarm group enter in column 'Message' the following: "Temperature critical !". The output of the final alarm message will be like follows:



Note for translation to other languages: The message text will also be affected in case of a change of the project language if it is included in a *.vis-file or a translation file *.tlt ↳ *Chapter 1.4.1.2.3.16 “Create translation file” on page 233*. BUT: In this case - like texts referring to a visualization it has to be set between two "#"-characters (e.g. in the example shown above : "#Temperature critical !#" and "TIME /EXPRESSION: MESSAGE #current#: VALUE", in order to get the text entered in the translation file as ALARMTEXT_ITEMs.)

A log file for action 'Save' is to be defined in the configuration of the alarm group ↳ *Chapter 1.4.1.4.2.4 “Alarm groups” on page 368*.

Alarm events for actions

For each action you define, at which alarm events it should be started ↳ *Chapter 1.4.1.4.2.2 “General information on alarms, terms” on page 364*.

Activate the desired events:

INTO The alarm occurs. Status = INTO.

ACK Acknowledgement by the user has been done. Status = ACK.

OUTOF Alarm state terminated. Status = OUTOF.

Colors/Bitmaps for class <class name>

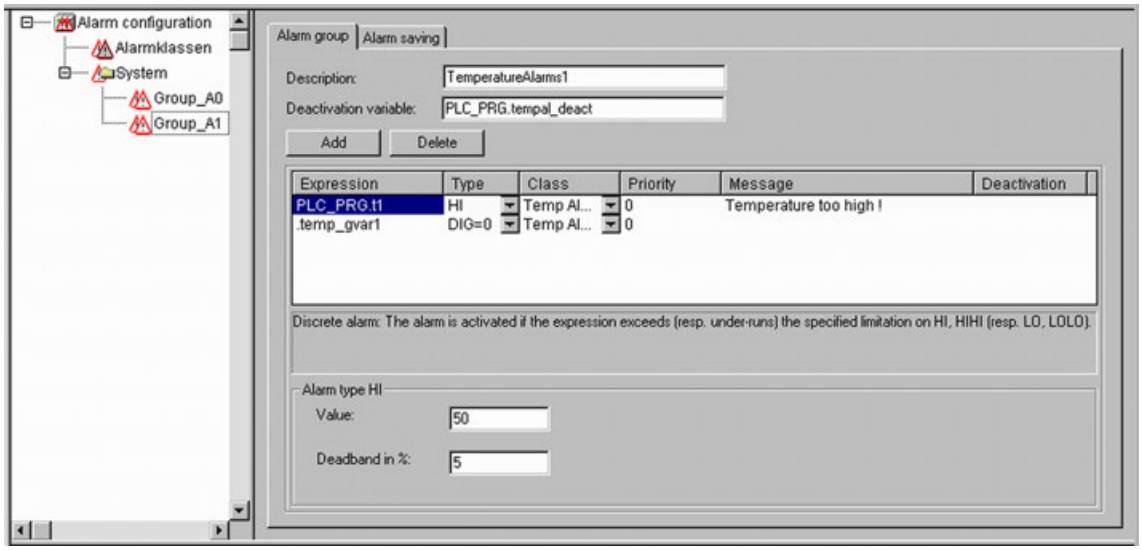
Each alarm class can get assigned own colors and bitmaps, which will be used for the differentiation of the alarms in the visualization element alarm table ↳ *Chapter 1.4.3.5.26 “Alarm table” on page 684*. Select a Foreground color and Background color for the possible events INTO, ACK and OUTOF.

The standard dialog for selecting a color will open as soon as you perform a mouse-click on the color symbol. For selecting a bitmap a mouse-click on the grey rectangle will open the standard dialog for selecting a file.

Alarm groups

Alarm groups are used for organizing the available alarms. Each alarm is definitely assigned to right one alarm group and is managed by this group. All alarms of a group can get assigned a common deactivation variable and common parameters regarding the alarm saving. Regard that even a single alarm must be configured within an alarm group.

A hierarchical structure of alarm groups can be defined via Folder elements. When a alarm group is selected in the configuration tree, automatically the dialog Alarm group will be displayed:



In the field *“Description”* you can enter a name for the alarm group.

As *“Deactivation variable”* a boolean project variable can be defined. At a rising edge on this variable the alarm creation for all alarms of the group will be deactivate, at a falling edge it will be re-activated.

Via button *[Add]* an alarm can be added to the group. A new line in the table window will be inserted and there the following parameters are to be set:

Expression: Enter here the project variable or an expression (e.g. "a + b") to which the alarm should refer. It is recommended to use the input assistant <F2> or the "Intellisense" function for a correct entry ↳ *Chapter 1.4.1.3.6 “Intellisense function” on page 295*.

Type: The alarm types listed in the following can be used. For each type regard the appropriate comment regarding the definitions to be done in the area beyond the table !

- DIG=0 Digital alarm, active as soon as the expression gets FALSE.
- DIG=1 Digital alarm, active as soon as the expression gets TRUE.
- LOLO Analog alarm, active as soon as the value of the expression falls below the Value defined for Alarm type LOLO. You can define a tolerance (Deadband). As long as the expression value is within the dead band, no alarm will be activated, even if the LOLO-value has been falling below the limit.
- LO corresponding to LOLO
- HI Analog alarm, active as soon as the expression exceeds the Value defined for Alarm type HIHI. You can define a tolerance (Deadband). As long as the expression value is within the dead band, no alarm will be activated, even if the HI value has exceeded the limit.
- HIHI corresponding to HI
- DEV- Deviation from the target value; Alarm gets active as soon as the value of the expression falls below the value defined for Alarm type DEV- plus the percentage deviation. Percentage deviation = target value * (deviation in %) / 100.
- DEV+ Deviation from the target value); Alarm gets active as soon as the value of the expression exceeds the value defined for Alarm type DEV+ plus the percentage deviation. Percentage deviation = target value * (deviation in %) / 100.
- ROC Rate of Change per time unit; Alarm gets active as soon as the expression deviates strongly from the previous value. The limit value for activating an alarm is defined by the number of value changes (Rate of changes) per second, minute or hour (units per).

Class: Choose the desired alarm class. The selection list will offer all classes which have been defined in the alarm class configuration before the last saving of the project ↳ *Chapter 1.4.1.4.2.3 “Alarm classes” on page 364*.

Priority: Here you can define a priority level 0-152. 0 is the highest priority. The priority will impinge on the sorting of the alarms within the alarm table.

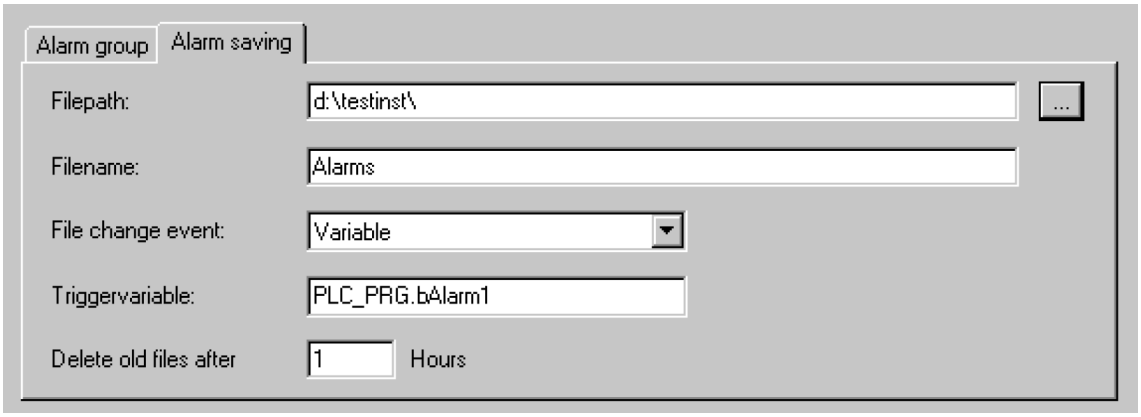
Message: Define here the text for the message box, which will appear in case of an alarm. This box must be confirmed by the user with OK, but this OK will not automatically acknowledge the alarm ! For confirming (acknowledge) the alarm the alarm table must be accessed. This is possible via the visualization element alarm table or via the date of the alarm entry in the table. This date can be read from a log file which can be created optionally. Regard that, when visualizing the alarms in an alarm table element in a visualization, the message text defined here might be overwritten by a corresponding text entry from an language file in xml-format, which is specified in the visualization for the purpose of dynamic language switching ↳ *Chapter 1.4.3.8.3 “Dynamic language switching” on page 709*.

Deactivation: Here a project variable can be entered, which at a rising edge will deactivate any creation of the alarm. Regard however, that this setting Alarm group' will be overwritten by the entry which might be found in the field 'Deactivation variable'.

Alarm saving

For each alarm group a file can be defined, in which the alarm events are stored, if (!) a 'Save' action has been assigned to the class in the alarm class configuration dialog ↳ *Chapter 1.4.1.4.2.3 “Alarm classes” on page 364*.

Select the alarm group in the configuration tree and open the dialog tab 'Alarm saving':



The following definitions are possible:

- Filepath: Directories path of the file which is defined in Filename; via button "..." you get the standard dialog for selecting a directory. If the target-specific option 'Alarmhandling on PLC' is activated, this path will be ignored an the file will be saved in the download directory of the PLC *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*.
- Filename: Name of the file which should save the alarm events (e.g. "alarmlog"). Automatically a file will be created which gets the name defined here plus an attached digit and which has the extension ".alm". The digit indicates the version of the log-file. The first file gets a "0"; each further file, which will be created according to the defined File change event, will be numbered with 1, 2 etc. (Examples: "alarmlog0.alm", "alarmlog1.alm).
- File change event: Define here the event which will cause the creation of a new file for alarm-saving. Possible entries: Never, after one Hour, after one Day, after one Week, after one Month, at a rising edge of the variable defined in field Triggervariable, when the number of records in the file as defined in Number of records gets exceeded.
- Triggervariable resp. Number of records: see above, File change event.
- Delete old files after .. Hours: Number of days since the day of creation, after which all alarm log-files except from the actual one should be deleted.

The log-file (History) contains the following entries:

See the column types and exemplary entries for two alarms.

Date/ Time in DWORD	Date	Time	Event	Expression	Alarm type	Limit	Tolerance	Current value	Class	Priority	Message
10469 63332	6.3.03	16:08: 52	INTO	PLC_ PRG. b	LO	-30	5	-31	Alarm _high	0	Tem- pera- ture !
10469 63333	6.3.03	16:08: 53	ACK	PLC_ PRG. n	HIHI	35			Warnn g	9	Rising Temp. !

Example appearance of the log-file	1046963332,6.3.03 16:08:52,INTO,PLC_PRG.ivar5,HIHI,,,, 9.00,a_class2,0, 1046963333,6.3.03 16:08:53,INTO,PLC_PRG.ivar4,ROC,2,,, 6.00,a_class2,2, 1046963333,6.3.03 16:08:53,INTO,PLC_PRG.ivar3,DEV-,,,, -6.00,a_class2,5, 1046963334,6.3.03 16:08:54,INTO,PLC_PRG.ivar2,LOLO,-35,,3, -47.00,warning,10,warning: low temperature ! 1046963334,6.3.03 16:08:54,INTO,PLC_PRG.ivar1,HI,20,,5, 47.00,a_class1,2,temperature to high ! Acknowledge !

Alarm configuration 'Extras' 'Settings'

The dialog Alarm configuration settings opens on the command 'Extras' 'Settings' in the Alarm Configuration:

Category date/ time

Here you set the formatting for the representation of the alarms in the log-file. Define the format according to the following syntax. Dashes and colons are to be set in inverted commas:

for date: dd'-'MM'-'yyyy -> e.g. "12.Jan-1993"

for time: hh':'mm':'ss (12-hours format) -> e.g. "01:10:34" or HH':'mm':'ss (24-hours format) -> e.g. "13:10:34"

Language

The usage of this dialog corresponds to that used for doing the language settings of a visualization object ↳ *Chapter 1.4.3.8 “Language switching” on page 706*.

Choose here a language file (*.vis or *.tlt) which should be used for the alarm configuration texts when the Option for the desktop is changed ↳ *Chapter 1.4.1.2.2.5 “Options for the desktop” on page 205*. Make sure that for this purpose the language file i.a. contains the translations for the text strings of the alarm configuration. See also: 'Project' 'Translate into another language' ↳ *Chapter 1.4.1.2.3.15 “'Project' 'Translate into another language'” on page 233*.

Alternatively you can define dynamic language switching by specifying a special language file in xml-format, like it is used for visualization elements. Note however that the actual settings only concern the display of the alarm messages. If the alarm configuration is visualized by an alarm table element in a visualization object, there the separate visualization language settings will be valid!

Online

Deactivate alarm evaluation in online mode: If this online setting is set, there will be no alarm handling in online mode. This means that no alarm evaluation corresponding to the current alarm configuration will be done and no alarms will be displayed. This might be desired in certain environments in order to save computing time.

1.4.1.4.3 Library manager

Overview

The library manager shows all libraries that are connected with the current project. The POUs, data types, and global variables of the libraries can be used the same way as user-defined POUs, data types, and global variables.

The library manager is opened with the 'Window' 'Library Manager' command. Information concerning included libraries is stored with the project and can be viewed in the dialog 'Informations about external library'. To open this dialog select the corresponding library name in the library manager and execute the command 'Extras' 'Properties'.

Libraries can contain pragma instructions in the declaration part, which effect that when the library is included in a project, not the complete declaration part gets displayed ↳ *Chapter 1.4.1.3.10.3 “Pragmas for controlling the display of library declaration parts” on page 312*. Thus particular variable declarations or comments can be "concealed" from the user.

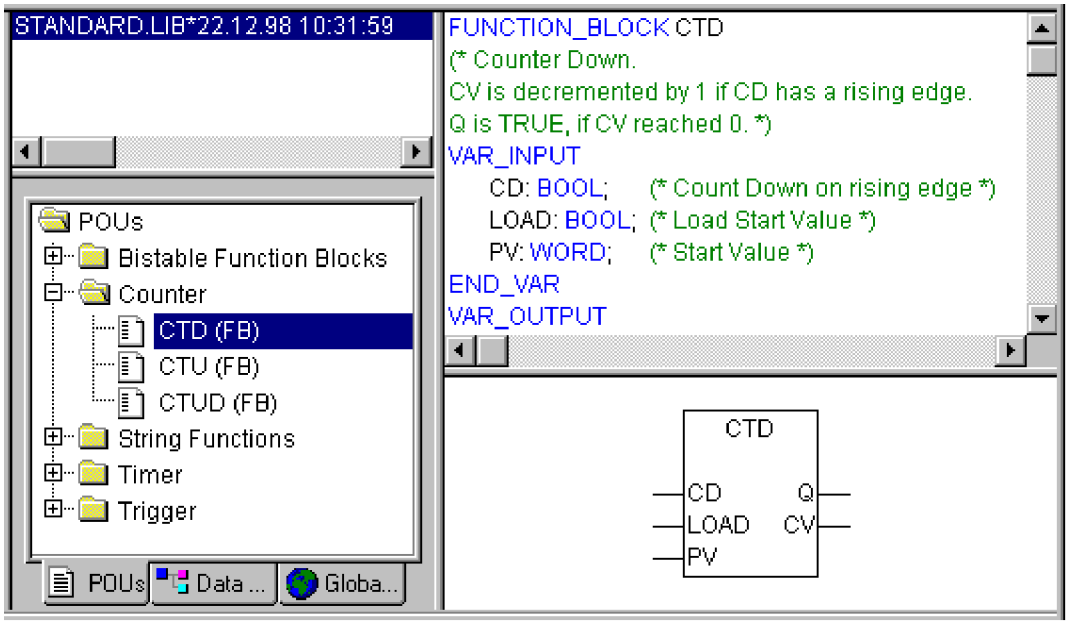
The window of the library manager is divided into areas by screen dividers. The libraries attached to the project are listed in the upper left area.

Depending on which register card has been selected, there is a listing of the POUs , Data types, Visualizations or Global variables of the library selected in the upper area. Folders are opened and closed by doubleclicking the line or pressing <Enter>.

There is a plus sign in front of closed folders, and a minus sign in front of opened folders.

If a POU is selected by clicking the mouse or selecting with the arrow keys then the declaration of the POU will appear in the upper right area of the library manager; and in the lower right is the graphic display in the form of a black box with inputs and outputs.

With data types and global variables, the declaration is displayed in the right area of the library manager.



See also:

Standard library *Chapter 1.4.1.4.3.2 “Standard library” on page 372*

User-defined libraries *Chapter 1.4.1.4.3.3 “User-defined libraries” on page 372*

'Insert' 'Additional Library' *Chapter 1.4.1.4.3.4 “'Insert' 'Additional Library'” on page 373*

Standard library

The library with "standard.lib" is always available. It contains all the functions and function blocks which are required from the IEC61131-3 as standard POUs for an IEC programming system. The difference between a standard function and an operator is that the operator is implicitly recognized by the programming system, while the standard POUs must be tied to the project (standard.lib).

The code for these POUs exists as a C-library.

User-defined libraries

If a project is to be compiled in its entity and without errors, then it can be saved in a library with the 'Save as' command in the 'File' menu. The project itself will remain unchanged. An additional file will be generated, which has the default extension ".lib". This library afterwards can be used and accessed like e.g. the standard library.

For the purpose to have available the POUs of a project in other projects, save the project as an Internal Library *.lib. This library afterwards can be inserted in other projects using the library manager.

Regard the possibility to define via pragmas to what extent the declaration part of the library should be visible in the Library Manager when the library has been included in a project.

If you have implemented POUs in other programming languages, e.g. C, and want to get them into a library, then save the project using data type External Library *.lib). You will get the library file but additionally a file with the extension "*.h". This file is structured like a C header file and contains the declarations of all POUs, data types and global variables, which are available with the library. If an external library is used in a project, then in simulation mode that implementation of the POUs will be executed; but on the target the C-written implementation will be processed.

If you want to add licensing information to a library, then press button Edit license info... and insert the appropriate settings in the dialog 'Edit Licensing Information'.

'Insert' 'Additional Library'

With this command you can attach an additional library to your project.

The command opens the dialog for opening a file. If the currently set directory does not contain the desired library, you can select another directory in field Library directory where all directories will be offered, which are defined in 'Project' 'Options' 'Directories' 'Libraries' (File type "*.lib"). Choose the desired library - multiple selection is possible - and confirm with OK. The dialog will close and the library gets inserted to the Library Manager. Now you can use the objects of the library in the project like user-defined objects.

Library paths

Regard which libraries directories are currently defined in the project options ↳ *Chapter 1.4.1.2.2.7 “Options for directories” on page 207*. If you insert a library from a directory which is not defined there, the library will be entered with the respective path.

Example: You insert library standard.lib from directory "D:\codesys\libraries\standard".

- If this directory is defined in the project options, the entry in the library manager will be: "standard.lib <date and time of file>".
- If in the project options there is just defined a directory "D:\codesys\libraries", then the entry in the library manager will be: "standard\standard.lib <date and time of file>".
- If no matching directory at all is defined in the project options, then the complete path will be entered: "D:\codesys\libraries\standard\standard.lib <date and time of file>".

When re-opening the project the libraries will be searched according to entries in the library manager. So for example, if just the library file name is entered there, the library will be searched in the libraries directories defined in the project options.

If libraries are not found when opening a project, you will be asked whether you want to change the library directory defined in the project options. If you say no, a dialog will open providing information on the libraries not found and the respective entries in the Library Manager will be displayed red-colored. In this case you can select a red entry and choose command *“Search”* from the context menu. Thus you will get the dialog for opening a file where you can browse for the missing library and reload it immediately.

Licensing

As soon as you include a library for which a license is needed and no valid license is found, you may get a message that the library is only available in demo mode or that the library is not licensed for the currently set target. You can ignore this message at that time or start appropriate actions concerning the license. An invalid license will produce an error during compile ↳ *Chapter 1.4.1.2.3.11 “'Project' 'Build'” on page 231*. In this case a doubleclick on the error message resp. <F4> will open the dialog 'License information' where you can start the appropriate actions guided by a wizard ↳ *Chapter 1.4.6.1 “Overview” on page 733*.

1.4.1.4.4 Log

Overview

The log stores in chronological order actions that occur during an Online session. For this purpose a binary log file (*.log) is set up. Afterward, the user can store excerpts from the appropriate project log in an external log.

The log window can be opened in either Offline or Online mode and can thus serve as a direct monitor online.

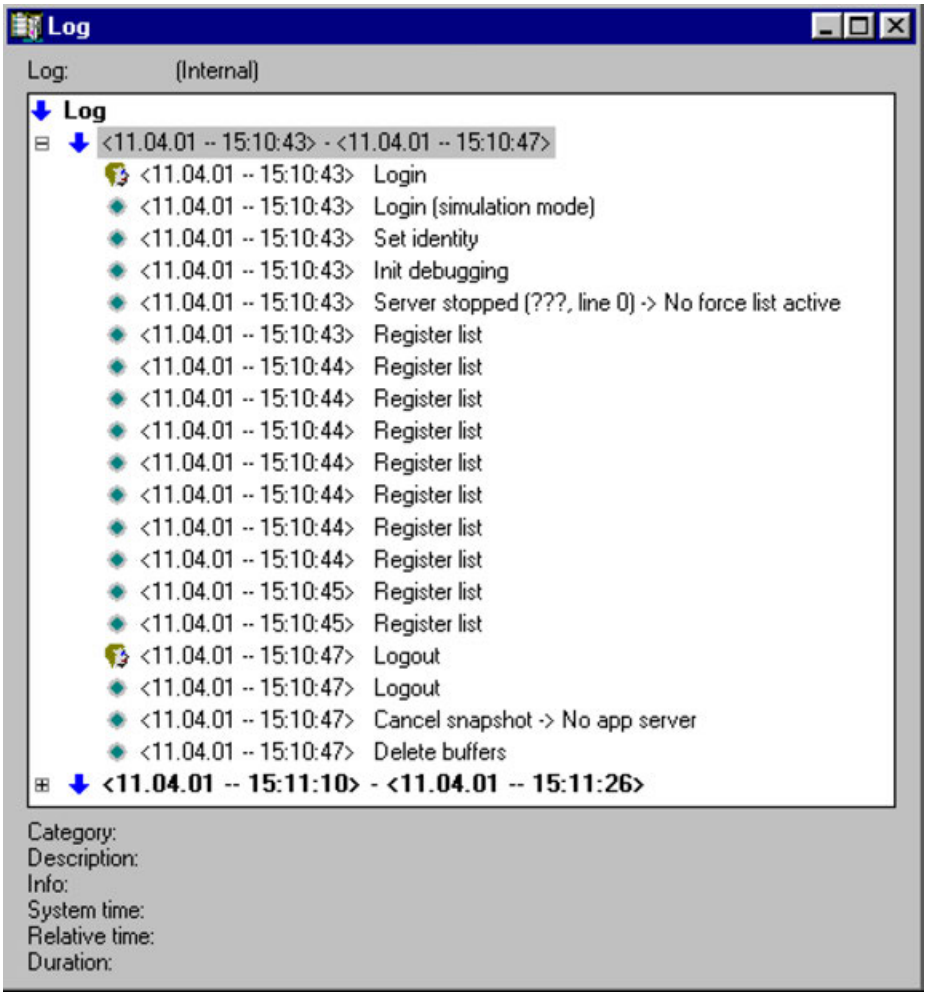
See also:

'Window' 'Log' ↳ *Chapter 1.4.1.4.4.2 “'Window' 'Log'” on page 374*

Menu Log ↳ *Chapter 1.4.1.4.4.3 “Menu log” on page 375*

'Window' 'Log'

To open, select the menu item 'Window' 'Log' or select entry 'Log' in the Resources tab.



In the log window, the file name of the currently displayed log appears after Log. If this is the log of the current project, the word "(Internal)" will be displayed.

Registered entries are displayed in the log window. The newest entry always appears at the bottom.

Only actions belonging to categories that have been activated in the 'Filter' field of the menu 'Project' 'Options' 'Log' will be displayed ↳ *Chapter 1.4.1.2.2.8 “Options for log” on page 208*.

Available information concerning the currently selected entry is displayed below the log window.

Category: The category to which the particular log entry belongs. The following four categories are possible:

- User action: The user has carried out an Online action (typically from the Online menu).
- Internal action: An internal action has been executed in the Online layer (e.g. Delete Buffers or Init debugging).
- Status change: The status of the runtime system has changed (e.g. from Running to Break, if a breakpoint is reached).
- Exception: An exception has occurred, e.g. a communication error.

Description: The type of action. User actions have the same names as their corresponding menu commands; all other actions are in English and have the same name as the corresponding `OnlineXXX()` function.

Info: This field contains a description of an error that may have occurred during an action. The field is empty if no error has occurred.

System time: The system time at which the action began, to the nearest second.

Relative time: The time measured from the beginning of the Online session, to the nearest millisecond.

Duration: Duration of the action in milliseconds.

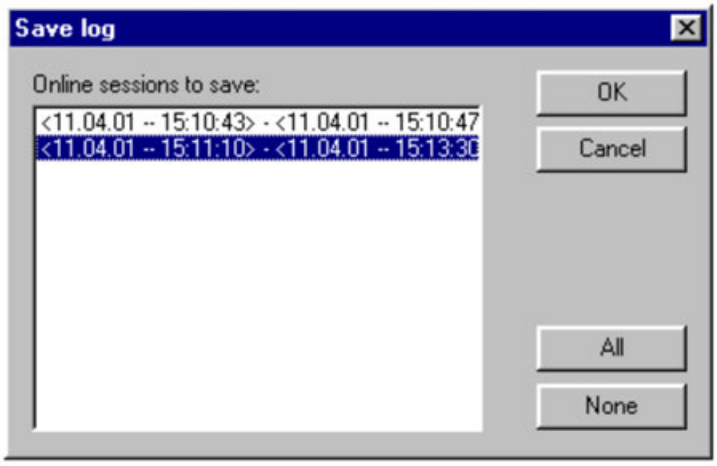
Menu log

When the log window has the input focus, the menu option Log appears in the menu bar instead of the items 'Extras' and 'Options'.

The menu includes the following items:

Load: An external log file *.log can be loaded and displayed using the standard file open dialog. The log that is present in the project will not be overwritten by the command. If the log window is closed and later opened again, or a new Online session is started then the version that is loaded will again be replaced by the project log.

Save: This menu item can only be selected if the project log is currently displayed. It allows an excerpt of the project log to be stored in an external file. For that, the following dialog will be displayed, in which the Online sessions to be stored can be selected:



After successful selection, the standard dialog for storing a file opens ('Save Log').

Display Project Log: This command can only be selected if an external log is currently displayed. It switches the display back to the project log

1.4.1.4.5 PLC browser

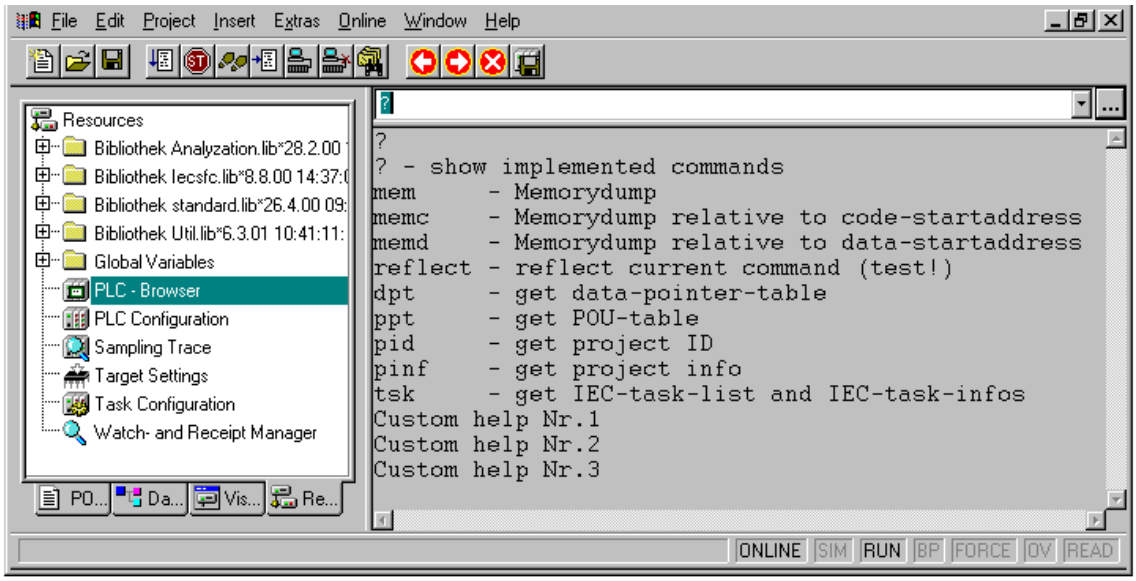
Overview

The PLC browser is a text-based control monitor (terminal). Commands for the request of specific information from the controller are entered in an entry line and sent as string to the controller. The returned response string is displayed in a result window of the browser. This functionality serves diagnostic- and debugging purposes.

↳ *Chapter 1.4.1.4.5.3 “Command entry in the PLC browser” on page 376*

General remarks concerning PLC browser operation

Select the entry PLC-Browser in the Resources tab-control. It will be available there if it is activated in the current target settings (category networkfunctionality).



The browser consists of a command entry line and a result/display window.

In a selection box the input line displays a list of all the commands entered since the start of the project (input history). They are available for re-selection until the project is closed. Only commands, which differ from those already existing, are added to the list.

The entered command is sent to the controller with *[Enter]*. If there is no Online connection, the command is displayed in the result window in the same way as it is sent to the controller, otherwise the response from the controller is shown there. If a new command is sent to the controller, the content of the result window is deleted.

Commands can be entered in the form of command strings, the use of macros is possible as well ↳ *Chapter 1.4.1.4.5.3 “Command entry in the PLC browser” on page 376* ↳ *Chapter 1.4.1.4.5.4 “Use of macros during the command entry in PLC-browser” on page 378*.

Command entry in the PLC browser

Basically the PLC browser makes available the standard commands hard-coded in the runtime system. It is concerned with functions for direct memory manipulation, for the output of project- and status functions as well as for runtime monitoring. They are described in the browser's ini-file, which is an integral part of the Target Support Package. These standard commands can be further supplemented by specialized ones, e.g. self-diagnostic functions or other status messages of the control application. The expansion of the command list must be carried out both in the customer interface in the runtime system as well as through additional entries in the Browser ini-file.

When opening the project the command list available in the PLC browser is generated based on the entries in the Browser ini-file. It can be accessed as input help using the *[...]* key in the dialog "Insert standard command" or using *[F2]*. Also the command 'Insert' 'Standard commands' can be used to get the command list. A command can be typed in manually to the command line or it can be selected from the list by a double-click on the appropriate entry.

The general command syntax is:

<KEYWORD><LEER><KEYWORD-DEPENDEND PARAMETERS>

The keyword is the command. With which parameters it can be expanded is described in the respective tooltip in the entry help window.

The command, which has been sent, is repeated in the output data window, the controller's response appears below it.

Example

Request for the project Id from the controller with the command "pid"

Entry in command line:

```
pid
```

Output in result window:

```
pid
Project-ID: 16#0025CFDA
```

A help text can be supplied for each standard command with ?<BLANK><KEYWORD>. This is similarly defined in the ini-file.

The following commands are firmly integrated in the runtime system and contained in the ini-file with the corresponding entries for entry help, tooltips and help:

Command	Description
?	The runtime system supplies a list of the available commands. The list is independent of the status of the description files of the target system.
mem	Hexdump of a memory area Syntax 1: mem <start address> <end address> Addresses can be entered decimal, hexadecimal (Prefix 16#) or as a macro.
memc	Hexdump relative to the start address of the code in the controller; like mem, the data are added to the code area start address.
memd	Hexdump relative to the data base address in the controller; like mem, the data are added to the data area start address.
reflect	Reflect current command line, for test purposes.
dpt	Read and display data-pointer table.
ppt	Read and display POU table.
pid	Read and display project Id.
pinf	Read and display project info (see 'Project' 'Project Info').
tsk	Show list of IEC-tasks incl. task infos defined in the project.
startprg	Start PLC program ('Online' 'Start').
stopprg	Stop PLC program ('Online' 'Stop').
resetprg	Reset PLC program. Only not-retentive data get initialized. ('Online' 'Reset).

Command	Description
resetprgcold	Reset PLC program cold. Retentive data also get initialized. ('Online' 'Reset (cold)').
resetprgorg	Reset PLC program original. The current application program as well as all data (incl. retentive and persistent) are deleted.
('Online' 'Reset (origin)').	
reload	Reload boot project.
getprgprop	Read and display program properties (Name, title, version author, date).
getprgstat	Read and display program status (e.g. "run", "stop", last error, flags)
filedir	File command "dir". List of files in the PLC directory.
filecopy	Copy file [from] [to]. Example: "filecopy filename.txt filename2.txt".
filerename	Rename files on PLC [old] [new]. Example: filerename oldname.txt newname.txt".
filedelete	Delete file on PLC; Example: "filedelete file.xml".
saveretain	Save retain variables. The name of the save file will be displayed afterwards.
restoreretain	Load retain variables. The name of the save file, from which the variables values are restored, will be displayed.
setpwd	Set password on controller; Syntax: setpwd <password> [level], e.g. "setpwd abcde 0" <level> can be "0" (default) just valid concerning logins from the programming system, or "1" valid for all applications
delpwd	Delete password on PLC.

Please regard:

- The first word of the command sequence entered is interpreted as keyword (<KEYWORD>).
- If the first word of the command entry is not recognized by the controller, the response 'Keyword not found' will appear in the result window.
- If a keyword is preceded by a "?<SPACE>" (e.g. "? mem"), the ini-file will be searched for the existence of a help section to this keyword. If one is available, nothing is sent to the controller, but only the help text is displayed in the output data window.

Use of macros during the command entry in PLC-browser

If a command associated with a macro is entered in the command line, this is expanded before it is sent to the controller. Then the response in the result window appears in a similarly expanded form.

The entry syntax is: <KEYWORD><macro>

<KEYWORD> is the command.

Macros

%P<NAME>	If NAME is a POU-name, the expression is expanded to <POU-Index>, otherwise there is no alteration
%V<NAME>	If NAME is a variable name, the expression is expanded to #<INDEX>:<OFFSET>, otherwise there is no alteration (this notation #<INDEX>:<OFFSET> is interpreted by the controller as a memory address)
%T<NAME>	If NAME is a variable name, the expression is expanded to <VARIABLENTYP>, otherwise there is no alteration.
%S<NAME>	If NAME is a variable name, the expression is expanded to <SIZEOF(VAR)>, otherwise there is no alteration.

The % character is ignored if the escape symbol \ (Backslash) is placed in front. The escape symbol as such is only transmitted if written \\.

Example

Entry in command line: (memory dump of the variable .testit ?)

```
mem %V.testit
```

Output in result window:

```
mem #4:52
03BAAA24 00 00 00 00 CD CD CD CD ....ÍÍÍÍ
```

Further PLC browser options

In the 'Extras' menu or in the PLC browser's toolbar there are the following commands for handling the command entry or history list:

With 'History forward' and 'History backward' you scroll backwards and forwards through the query results already carried out. The history recording is continued until you leave the project.

With 'Cancel' you break off a query which has been initiated.

With 'Save history list' you save the query results carried out up until that point in an external text file. The dialogue 'Save file as' will appear, in which you can enter a file name with the extension „.bhl" (Browser History List).

'Print last command' opens the standard dialogue to print. The current query plus the output data in the message window can be printed.

1.4.1.4.6 Sampling trace

Overview and configuration

Overview

Sample tracing will be available as an object in the resources, if it is activated in the target settings (category 'General'). It can be used to trace the progression of values for variables is traced over a certain time. These values are written in a ring buffer (trace buffer). If the memory is full, then the "oldest" values from the start of the memory will be overwritten. As a maximum, 20 variables can be traced at the same time. A maximum of 500 values can be traced per variable.

Since the size of the trace buffer in the PLC has a fixed value, in the event of very many or very wide variables (DWORD), fewer than 500 values can be traced.

Example

If 10 WORD variables are traced and if the memory in the PLC is 5000 bytes long, then, for every variable, 250 values can be traced.

In order to be able to perform a trace, open the object for a 'Sampling Trace' in the 'Resources' register card in the Object Organizer *Chapter 1.4.1.2.1.3 “Object organizer” on page 199*. Create or load an appropriate trace configuration and define the variables to be traced.

After you have created the configuration and have started the trace in the PLC with 'Start Trace', then the values of the variables will be traced. With 'Read Trace', the final traced values will be read out and displayed graphically as curves *Chapter 1.4.1.4.6.2.1 “'Extras' 'Start Trace'” on page 382* *Chapter 1.4.1.4.6.2.2 “'Extras' 'Read Trace'” on page 382.

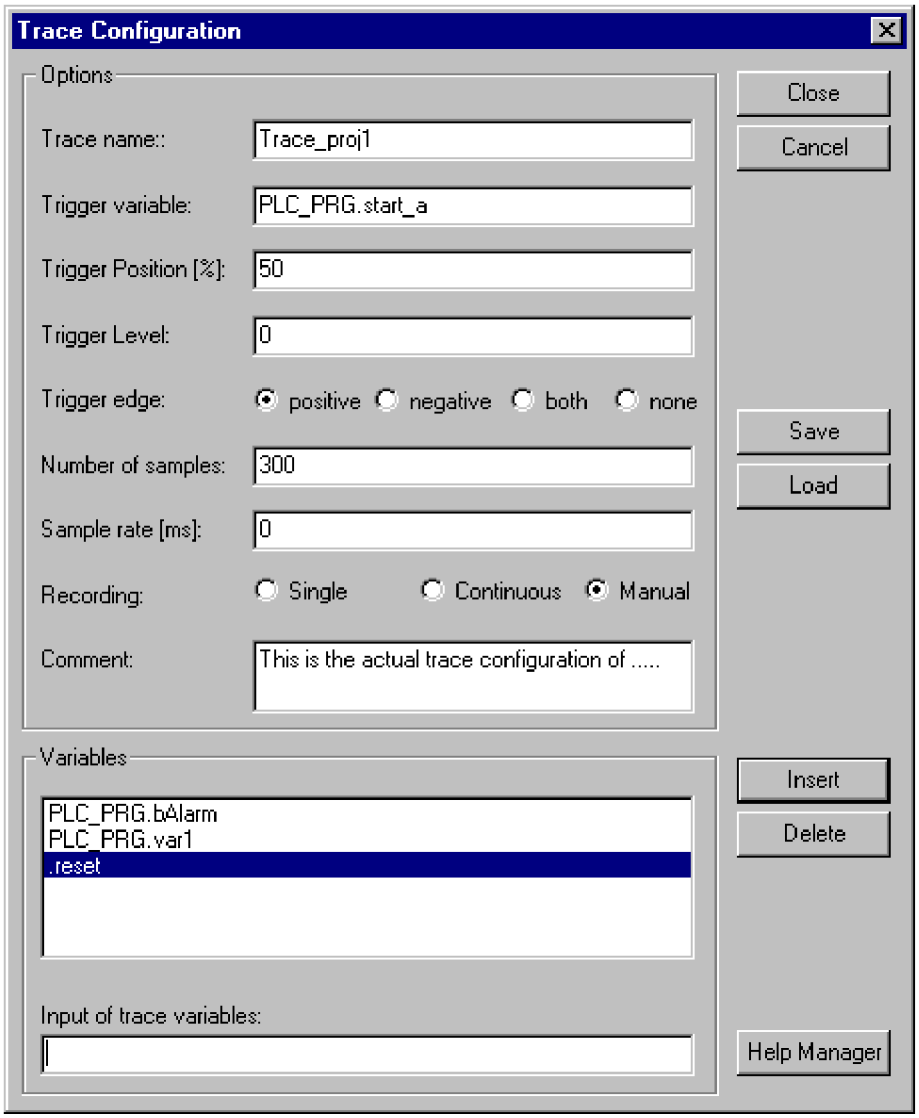
A Trace (variable values and configuration) can be saved and reloaded in project format (*.trc) or in XML format (*.mon). Just the configuration can be stored and reloaded via a *.tcf-file.

Various traces can be available in a project for getting displayed. They are listed in a selection list ('Trace') in the upper right corner of the trace window. You can select one of those to be the currently used trace configuration.

If a task configuration is used for controlling the program, the trace functionality refers to the debug task *Chapter 1.4.1.4.8.1 “Overview” on page 390.*

'Extras' 'Trace Configuration'

With this command you will be given the dialog box for entering the variables to be traced, as well as diverse trace parameters for the Sampling Trace. The dialog can also be opened by a double click in the grey area of the dialog Sampling Trace.



First define a name for the trace configuration (Trace Name). This name will be added to the selection list 'Trace' in the upper right corner of the Trace window, as soon as you have confirmed and closed the configuration dialog with OK. Optionally enter a comment.

The list of the Variables to be traced is initially empty. In order to append a variable the variable must be entered in the field under the list. Following this, you can use the Add button or the *[Enter]* to append the variable to the list. You can also use the input assistant ↬ *Chapter 1.4.1.2.5.11 "'Edit' 'Input assistant'" on page 276*. The use of enumeration variables is possible.

A variable is deleted from the list when you select the variable and then press the *[Delete]* button.

A Boolean or analog variable (also an enumeration variables) can be entered into the field Trigger Variable. The input assistance can be used here. The trigger variable describes the termination condition of the trace.

In Trigger Level you enter the level of an analog trigger variable at which the trigger event occurs. You also can use an ENUM constant here. When Triggeredgepositive is selected the trigger event occurs after an ascending edge of the Boolean trigger variable or when an analog variable has passed through the trigger level from below to above. Negative causes triggering after a descending edge or when an analog variable went from above to below. Both causes triggering for both descending and ascending edges or by a positive or negative pass, whereas none does not initiate a triggering event at all.

Trigger Position is used to set the percentage of the measured value which will be recorded before the trigger event occurs. If, for example, you enter 25 here then 25% of the measured values are shown before the triggering event and 75% afterwards and then the trace is terminated.

The field Sample Rate is used set the time period between two recordings in milliseconds resp., if supported by the target system, in microseconds. The default value "0" means one scanning procedure per cycle.

Select the mode for recalling the recorded values: With Single the Number of the defined samples are displayed one time. With Continuous the reading of the recording of the defined number of measured values is initiated anew each time. If, for example, you enter the number '35' the first display contains the first measured values 1 to 35 and the recording of the next 35 measured values (36-70) will then be automatically read, etc.Manual selection is used to read the trace recordings specifically with 'Extras' 'Read Trace' ↳ *Chapter 1.4.1.4.6.2.2 “'Extras' 'Read Trace'” on page 382*.

The recall mode functions independently of whether a trigger variable is set or not. If no trigger variable is set the trace buffer will be filled with the defined number of measured values and the buffer contents will be read and displayed on recall.

[Save] is used to store the trace configuration which has been created in a file. The standard window "File save as" is opened for this purpose.

Stored trace configurations can be retrieved using *[Load]*. The standard window "File open" is opened for this purpose.

[Save] and [Load] in the configuration dialog only relates to the configuration, not to the values of a trace recording (in contrast to the menu commands 'Save Values' and 'Load Trace' ↳ *Chapter 1.4.1.4.6.4.1 “'Save Values'” on page 385 ↳ Chapter 1.4.1.4.6.4.2 “'Load Values'” on page 385**).*

If the field Trigger Variable is empty, the trace recording will run endlessly and can be stopped by 'Extras' 'Stop Trace' ↳ *Chapter 1.4.1.4.6.2.4 “'Extras' 'Stop Trace'” on page 383*.

Selection of the variables to be displayed

The comboboxes to the right, next to the window for displaying curves trace variables defined in the trace configuration. If a variable is selected from the list, then after the trace buffer has been read the variable will be displayed in the corresponding color (Var 0 green, etc.). Variables can also be selected if curves are already displayed.

A maximum of up to eight variables can be observed simultaneously in the trace window.

Generating a trace sampling

'Extras' 'Start Trace'

Symbol:

With this command the trace configuration is transferred to the PLC and the trace sampling is started in the PLC.

'Extras' 'Read Trace'

Symbol

With this command the present trace buffer is read from the PLC, and the values of the selected variables are displayed.

'Extras' 'Auto Read'

With this command the present trace buffer is read automatically from the PLC, and the values are continuously displayed.

If the trace buffer is automatically read, then a check (ü) is located before the menu item.

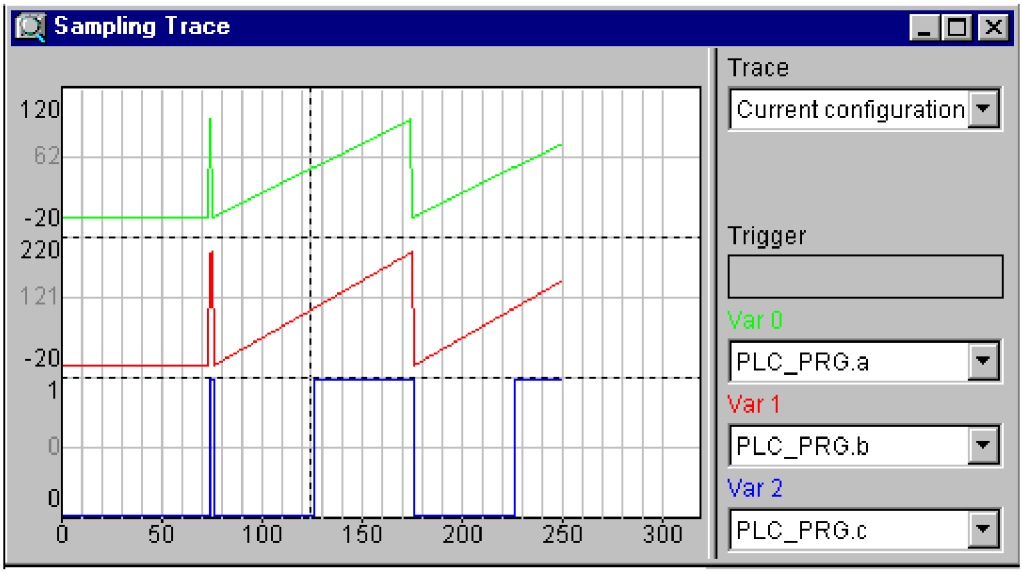
'Extras' 'Stop Trace'

Symbol:

This command stops the Sampling Trace in the PLC.

Looking at the trace sampling

Display of the sampling trace



If a trace buffer is loaded (), then the values of all variables to be displayed can be read out ('Extras' 'Read Trace' or 'Extras' 'AutoRead') and will be displayed accordingly ↳ *Chapter 1.4.1.4.6.2.1 "'Extras' 'Start Trace'" on page 382* ↳ *Chapter 1.4.1.4.6.2.2 "'Extras' 'Read Trace'" on page 382* ↳ *Chapter 1.4.1.4.6.2.3 "'Extras' 'Auto Read'" on page 382*. If no scan frequency has been set, then the X axis will be inscribed with the continuous number of the traced value. The trace buffer will be deleted as soon as the trace sampling is stopped ↳ *Chapter 1.4.1.4.6.2.4 "'Extras' 'Stop Trace'" on page 383*.

The status indicator of the trace window (first line) indicates whether the trace buffer is full and when the trace is completed.

If a value for the scan frequency was specified, then the x axis will specify the time of the traced value. The time is assigned to the "oldest" traced value. In the example, e.g., the values for the last 25 seconds are indicated.

The Y axis is inscribed with values in the appropriate data type. The scaling is laid out in a way that allows the lowest and the highest value to fit in the viewing area. In the example, Var 0 has taken on the lowest value of 6, and the highest value of 100: hence the setting of the scale at the left edge.

If the 'Extras' 'Trace Configuration' ↳ *Chapter 1.4.1.4.6.1.2 "'Extras' 'Trace Configuration'" on page 380* is met, then a vertical dotted line is displayed at the interface between the values before and after the appearance of the trigger requirement.

'Extras' 'Cursor Mode'

The easiest way to set a cursor in the monitoring area is to click there with the left mouse button. A cursor appears and can be moved by the mouse. At the top of the monitoring window the current x-position of the cursor is displayed. In the fields next to 'Var 0', 'Var 1', ..., 'Var n' the value of the respective variable is shown.

Another way is the command 'Extras' 'Cursor mode'. With this command two vertical lines will appear in the Sampling Trace. First they are laying one on the other. One of the lines can be moved to the right or to the left by the arrow keys. By pressing *[Ctrl]* + *[left]* or *[Ctrl]* + *[right]* the speed of the movement can be increased by factor 10.

If additionally *[Shift]* is pressed, the second line can be moved, showing the difference to the first one.

'Extras' 'Multi Channel'

With this command you can alternate between single-channel and multi-channel display of the Sampling Trace. In the event of a multi-channel display, there is a check (ü) in front of the menu item.

The multi-channel display has been preset. Here the display window is divided into as many as eight display curves. For each curve the maximum and the minimum value are displayed at the edge.

In a single-channel display, all curves are displayed with the same scaling factor and are superimposed. This can be useful when displaying curve abnormalities.

'Extras' 'Show grid'

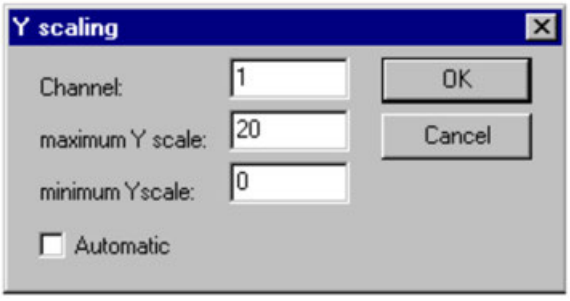
With this command you can switch on and off the grid in the graphic window. When the grid is switched on, a check symbol will appear next to the menu item.

'Extras' 'Y Scaling'

With this command you can change the preset Y scaling of a curve in the trace display. By doubleclicking on a curve you will also be given the dialog box 'Y-scaling'.

As long as option Automatic is activated, the default scaling will be used, which depends on the type of the used variable. In case of enumeration variables the enumeration values will be displayed at the scale. In order to change the scaling, deactivate option 'Automatic' and enter the number of the respective curve (Channel) and the new maximum (maximum y scale) and the new minimum value (minimum y scale) on the y axis.

The channel and the former value are preset.



'Extras' 'Stretch'

Symbol:

With this command you can stretch (zoom) the values of the Sampling Trace that are shown. The beginning position is set with the horizontal picture adjustment bar. With repeated stretches that follow one-after-another, the trace section displayed in the window will increasingly shrink in size.

This command is the counterpart to 'Extras' 'Compress' ↳ *Chapter 1.4.1.4.6.3.7 “'Extras' 'Compress'” on page 385*.

'Extras' 'Compress'

Symbol:

With this command the values shown for the Sampling Trace are compressed; i.e., after this command you can view the progression of the trace variables within a larger time frame. A multiple execution of the command is possible.

This command is the counterpart to 'Extras' 'Stretch' ↳ *Chapter 1.4.1.4.6.3.6 “'Extras' 'Stretch'” on page 384*.

'Extras' 'Save trace values'

Use the commands of this menu to save traces (configuration + values) to files resp. to reload them from files to the project. Besides that a trace can be saved in a file in ASCII format.

*Regard the alternative way of storing and reloading traces by using the commands of menu 'Extras' 'External Trace Configurations' (XML format, *.mon-Datei)* ↳ *Chapter 1.4.1.4.6.5 “'Extras' 'External Trace Configurations'” on page 386!*

'Save Values'

With this command you can save a Sampling Trace (values + configuration data). The dialog box for saving a file is opened. The file name receives the extension "*.trc".

Be aware, that here you save the traced values as well as the trace configuration, whereas Save trace in the configuration dialog only concerns the configuration data.

The saved Sampling Trace can be loaded again with 'Load Trace' ↳ *Chapter 1.4.1.4.6.4.2 “'Load Values'” on page 385*.

Regard the alternative way of saving a trace by using the commands of menu 'Extras' 'External Trace Configurations' ↳ *Chapter 1.4.1.4.6.5 “'Extras' 'External Trace Configurations'” on page 386.*

'Load Values'

With this command a saved Sampling Trace (traced values + configuration data) can be reloaded. The dialog box for opening a file is opened. Select the desired file with the "*.trc" extension.

With 'Save Values' you can save a Sampling Trace ↳ *Chapter 1.4.1.4.6.4.1 “'Save Values'” on page 385*.

'Trace in ASCII-File'

With this command you can save a Sampling Trace in an ASCII file. The dialog box for saving a file is opened. The file name receives the extension "*.txt". The values are deposited in the file according to the following scheme:

BODAS Trace

D:\BODAS\PROJECTS\TRAFFICSIGNAL.PRO

Cycle PLC_PRG.COUNTER PLC_PRG.LIGHT1

0 2 1

1 2 1

2 2 1

.....

If no frequency scan was set in the trace configuration, then the cycle is located in the first column; that means one value per cycle has been recorded at any given time. In the other respects, the entry here is for the point in time in ms at which the values of the variables have been saved since the Sampling Trace has been run.

In the subsequent columns, the corresponding values of the trace variables are saved. At any given time the values are separated from one another by a blank space.

The appertaining variable names are displayed next to one another in the third line, according to the sequence (PLC_PRG.COUNTER, PLC_PRG.LIGHT1).

'Extras' 'External Trace Configurations'

Use the commands of this menu to save or reload traces (configuration + trace values) in files resp. from files, to load a trace from the controller to the project or to set a certain trace as that which should be used in the project.

*Regard the alternative way of storing and reloading traces by using the commands of menu 'Extras' 'Save trace values' (project format, *.trc-file, ASCII)*
↳ Chapter 1.4.1.4.6.4 “'Extras' 'Save trace values'” on page 385!

'Save to file'

With this command a trace (configuration + values) can be saved in a file in XML format. For this purpose the standard dialog for saving a file opens. Automatically the file extension *.mon will be used.

A *.mon-file can be reloaded to a project with command 'Load from file' *↳ Chapter 1.4.1.4.6.5.2 “'Load from File'” on page 386*.

*Regard the alternative way of storing and reloading traces by using the commands of menu 'Extras' 'Save trace values' (Project format, *.trc-Datei, ASCII)*
↳ Chapter 1.4.1.4.6.4 “'Extras' 'Save trace values'” on page 385!

'Load from File'

With this command a trace (configuration + values), which is available in a file in XML format (*.mon, can be loaded into the project. The dialog for opening a file will open and you can browse for files with extension *.mon. The loaded trace will be displayed and added to the selection list in field 'Trace in the configuration dialog. If you want to set it as currently used project trace configuration, use command 'Set as project configuration' *↳ Chapter 1.4.1.4.6.5.5 “Set as project configuration” on page 387*.

A *.mon-file can be created by using command 'Save to file' *↳ Chapter 1.4.1.4.6.5.1 “'Save to file'” on page 386*.

Regard the alternative way of saving a trace by using the commands of menu 'Extras' 'Save trace values' *Chapter 1.4.1.4.6.4 “'Extras' 'Save trace values'” on page 385.*

'Save to Target'

With this command in online mode a trace configuration, which is available in a file in XML format (*.mon), can be loaded into the controller (target).

The standard dialog for opening a file will be available, where per default files with extension *.mon will be displayed. Regard in this concern the possibility to save trace configurations in *.mon-files (XML format) *Chapter 1.4.1.4.6.5.1 “'Save to file'” on page 386*.

'Load from Target'

With this command the trace (configuration + values) which is currently used on the controller can be loaded to the project. It will be displayed in the trace window and can be set as project configuration *Chapter 1.4.1.4.6.5.5 “Set as project configuration” on page 387*.

Set as project configuration

With this command the trace configuration which is currently selected in the list of available traces (field 'Trace' in the trace window) can be set as active configuration within the project.

The selection list besides the currently used (top position) offers all traces which have been loaded to the project by command 'Load from file' from *.mon-files (e.g. for the purpose of viewing) *Chapter 1.4.1.4.6.5.2 “'Load from File'” on page 386*.

1.4.1.4.7 Target settings

Overview

The "Target Settings" is an object of the 'Resources'. Here you define, which target shall be used for the project and how it will be configured. If a new project is started with 'Project' 'New', a dialog will open where the target, that means a predefined configuration for a target, has to be set.

Dialog 'Target Settings'

The dialog 'Target Settings' will open automatically, when a new project is created. It also can be opened by selecting the menu item 'Target Settings' in the register 'Resources' in the Object Organizer.

Choose one of the target configurations offered at Configuration.

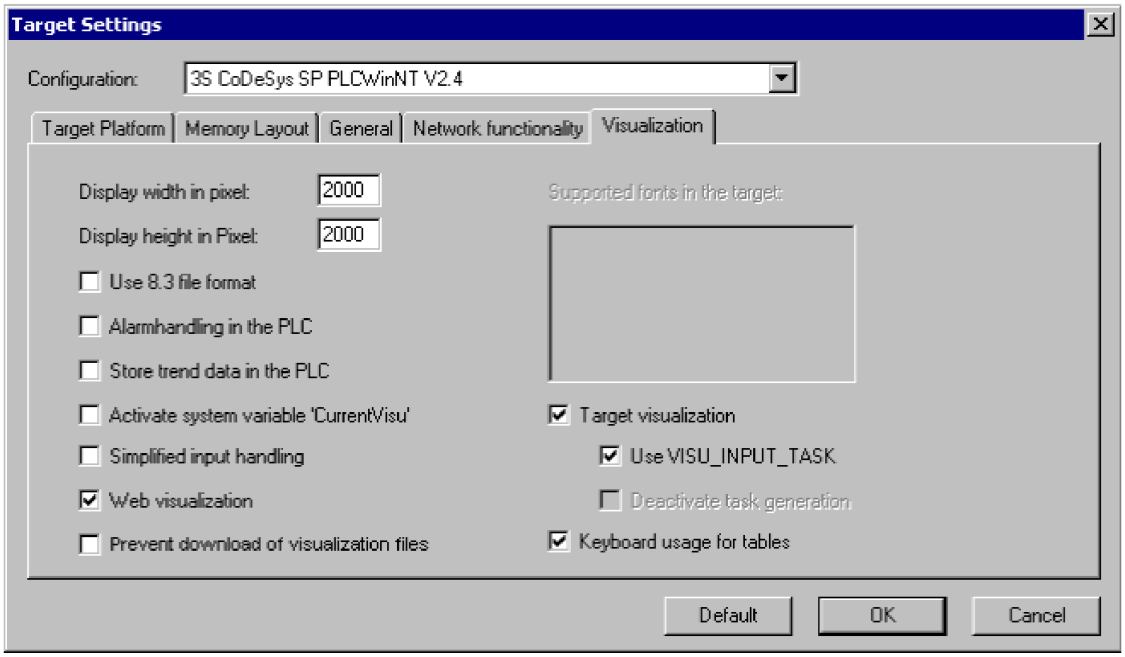
If you choose one of the installed configurations it depends on the entries in the target files, which possibilities are left to customize this configuration in the dialogs.

If a configuration is selected, which is provided with the entry "HideSettings" in the corresponding target file, you only can see the name of the configuration. Otherwise you can use the visualization dialog to modify the given configuration *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*.

Please be aware, that each modification of the predefined target configuration can cause severe changes in performance and behavior of the target!

Press <Default> if you want to reset the target settings to the standard configuration given by the target file.

Target settings in category visualization



The items described for this tab can be available for each standard target.

Dialog item	Meaning
Display width in pixel Display height in pixel	An area of the given width and height will be displayed in the editor window when editing a visualization. Thus e.g. the size of the screen on which the target visualization will run later, can be regarded when positioning the visualization elements. Use 8.3 file format The file names of the bitmaps and language files which are used in the visualization will be shortened to the 8.3-notation format and loaded to the PLC in this format.
Alarmhandling in the PLC	The task ALARM_TASK will be inserted automatically in the task configuration. It will process an implicitely created ST-code evaluating the status of the particular alarms and if applicable executing the associated actions ↬ *Chapter 1.4.1.4.2.1 “Overview” on page 363*. The ST-code needs auxiliary functions of library SysLibAlarmTrend.lib. This library will be loaded automatically. Additionally the implicitely needed libraries SysLibSockets.lib, SysLibMem.lib, SysLibTime.lib, SysLibFile.lib are loaded. These libraries must be supported by the target system. Hint: The 'Alarm handling in the PLC' can be used even if no web visualization has been activated. Even then the required ST-code will be generated.
Store trend data in the PLC	The trend handling in the PLC will be activated. The task TREND_TASK will be inserted automatically in the task configuration It will process an implicitely created ST-code for recording the trend data in a ring buffer and - if option History is activated in the trend element - for storing the values in a file system. The ST-code needs auxiliary functions of library SysLibAlarmTrend.lib. This library will be loaded automatically. Additionally the implicitely needed libraries SysLibSockets.lib, SysLibMem.lib, SysLibTime.lib, SysLibFile.lib are loaded. These libraries must be supported by the target system. Hint: 'Store Trend data....' can be used even if no web visualization has been activated. Even then the required ST-code will be generated.
Activate system variable 'CurrentVisu'	The system variable can be used for switching between visualizations ↬ *Chapter 1.4.3.11 “System variables” on page 717*.
Supported fonts in the target	List of fonts which are supported by the target system.
Simplified input handling	If activated: In 'Operation over the keyboard' the input handling in visualizations is simplified ↬ *Chapter 1.4.3.9.2 “Operation over the keyboard” on page 715*: *[Tab]* and *[Space]* are not needed to get from one input field to the next one. The selection is automatically given forward to the next field after having terminated an input by *[Return]*. An input field also can be reached via the arrow or *[Tab]* keys and then immediately an input can be entered. If not activated: *[Tab]* and *[Space]* must be used to get to the next input field and to select this field for making possible an input.
Web visualization	If activated: All visualisation objects of the project are compiled for the usage as Web-Visualization objects ↬ *Chapter 1.4.5.1 “Overview” on page 721*.

Dialog item	Meaning
Compression	If activated: The following files for the web visualization, which are to be transferred to the web server/ PLC, will be transferred in a packed format (zip-Format); otherwise in original format: ● XML visualization files ● image files (only *.bmp, because with others no compression effect) ● language files (*.xml for dynamic texts, *.tlt, *.vis) The files additionally to the existing file name get the extension ".zip". The dot in the existing name will be replaced by an underscore (example: "PLC_VISU.xml" will be renamed to "PLC_VISU_xml.zip") No compression is done for the Java-archives (minml.jar, webvisu.jar) and the main page webvisu.htm.
Prevent download of visualization files	If activated: When the project is downloaded, all files which are used in the current visualization will not be downloaded to the target system. Visualization files are only downloaded for web visualization and can be bitmaps, language files and for web visualization also XML description files. Keyboard usage for tables If this option is activated, in online mode the keyboard usage of tables in the visualization (CODESYS HMI or web visualization) is possible. Switching off this option will effect that no code is generated for the key functions.

1.4.1.4.8 Task configuration

Overview

In addition to declaring the special PLC_PRG program, you can also control the processing of your project using the task management.

A Task is a time unit in the processing of an IEC program. It is defined by a name, a priority and by a type determining which condition will trigger the start of the task. This condition can be defined by a time (cyclic, freewheeling) or by an internal or external event which will trigger the task; e.g. the rising edge of a global project variable or an interrupt event of the controller.

For each task you can specify a series of programs that will be started by the task. If the task is executed in the present cycle, then these programs will be processed for the length of one cycle.

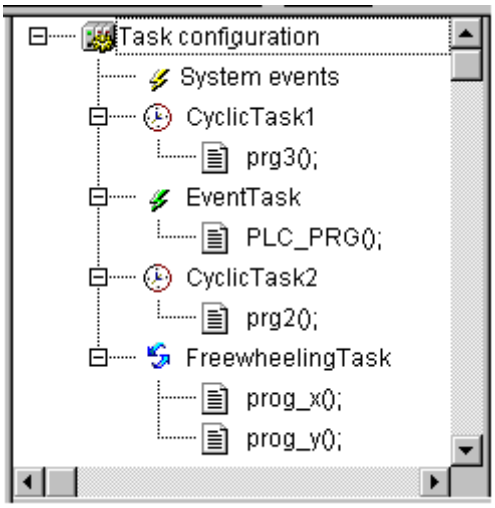
The combination of priority and condition will determine in which chronological order the tasks will be executed.

Each task can be enabled or disabled explicitly.

For each task you can configure a watch dog (time control); the possible settings depend on the target system.

Additionally there is the possibility to link System events (e.g. Start, Stop, Reset) directly with the execution of a project POU.

The Task Configuration is found as an object in the Resources tab of the Object Organizer. The Task editor is opened in a bipartited window.



In the left part of the window the tasks are represented in a configuration tree. At the topmost position you will always find the entry 'Taskconfiguration'. Below there are the entry 'System events' and the entries for the particular tasks, represented by the task name. Below each task entry the assigned program calls are inserted. Each line is preceded by an icon.

In the right part of the window a dialog will be displayed which belongs to the currently marked entry in the configuration tree. Here you can configure the tasks (Task properties), program calls (Program call) resp. define the linking of system events (System events). It depends on the target which options are available in the configuration dialogs. They are defined by a description file which is referenced in the target file. If the standard descriptions are extended by customer specific definitions, then those will be displayed in an additional tab 'Parameter' in the right part of the window.

Please do not use the same string function (see standard.lib) in several tasks, because this may cause program errors by overwriting.

Which task is being processed?

For the execution, the following rules apply:

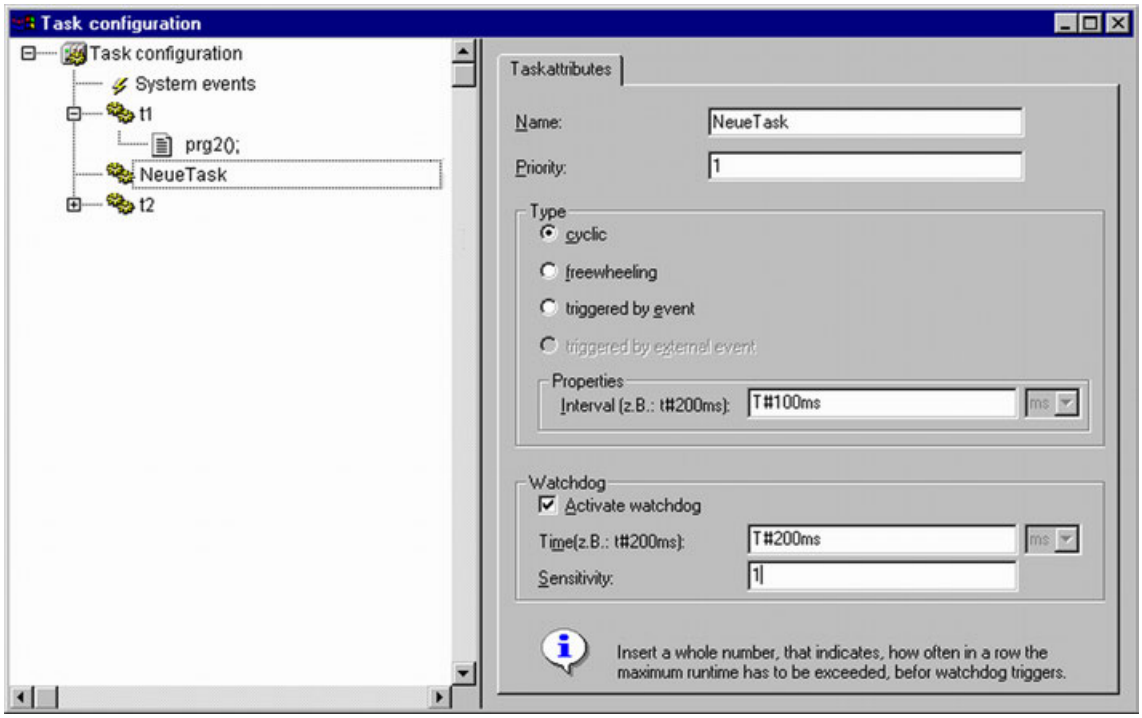
- That task is executed, whose condition has been met; i.e., if its specified time has expired, or after its condition (event) variable exhibits a rising edge.
- If several tasks have a valid requirement, then the task with the highest priority will be executed.
- If several tasks have valid conditions and equivalent priorities, then the task that has had the longest waiting time will be executed first.
- The processing of the program calls will be done according to their order (top down) in the task editor.
- Depending on the target system PLC_PRG might get processed in any case as a free-wheeling task, without being inserted in the task configuration tree.

'Insert' 'Insert Task' or 'Insert' 'Append Task'

With this command you can insert a new task into the Task Configuration. The entries each consist of a symbol and the task name.

If a task or the entry 'System events' is selected, then the 'Insert Task' command will be at your disposal. The new task will be inserted after the selected one. If the entry 'Task Configuration' is selected, then the 'Append Task' is available, and the new task will be appended to the end of the existing list. The maximum number of tasks is defined by the target system. Please regard that a certain number of tasks already might be reserved for modules of the PLC Configuration (defined in the *.cfg file).

When inserting a task, the dialog for setting the Task attributes will be opened:



Insert the desired attributes:

Name: a name for the task; with this name the task is represented in the configuration tree; the name can be edited there after a mouseclick on the entry or after pressing the <Space> key when the entry is selected.

Priority (0-31): a number between 0 and 31; 0 is the highest priority, 31 is the lowest

Types:

- cyclic (): The task will be processed cyclic according to the time definition given in the field 'Interval' (see below).
- freewheeling (): The task will be processed as soon as the program is started and at the end of one run will be automatically restarted in a continuous loop. There is no cycle time defined.
- triggered by event (): The task will be started as soon as the variable, which is defined in the Event field gets a rising edge.
- triggered by external event (): The task will be started as soon as the system event, which is defined in the Event field, occurs. It depends on the target, which events will be supported and offered in the selection list. (Not to be mixed up with system events

Properties for event triggered tasks:

- Interval (for Type 'cyclic' or 'triggered by external event' if the event requires a time entry): the period of time, after which the task should be restarted. If you enter a number, then you can choose the desired unit in the selection box behind the edit field: milliseconds [ms] or microseconds [µs]. Inputs in [ms]-format will be shown in the TIME format (e.g. "t#200ms") as soon as the window gets repainted; but you also can directly enter the value in TIME format. Inputs in [ms] will always be displayed as a pure number (e.g. "300").
- Event (for Type 'triggered by event' or 'triggered by external event'): a global variable which will trigger the start of the task as soon as a rising edge is detected. Use button ... or the input assistant <F2> to get a list of all available global variables.

Possibly the target system defines Singleton-Events. These are events, which only allow to start one single task. Whether such an event starts several tasks will be checked during compilation of the project. The check regards the data address of the event variable, not on the name.
For example: If the target system defines %MX1.1 and %IB4 as Singleton-Events, using the following variables as event variables will produce two errors (a and b as well as c and d each have the same address).

```
 VAR_GLOBAL
a AT %MX1.1: BOOL;
b AT %MX1.1: BOOL;
c AT %MB4: BOOL;
d AT %MD1: BOOL;
END_VAR
```

If there is no entry in both fields 'Interval' and 'Event', then the task interval will depend on which runtime system is used (see run time documentation); e.g. in this case for CODESYS SP NT V2.2 and higher an interval of 10 ms will be used).

Watchdog

For each task a time control (watchdog) can be configured. If the target system supports an extended watchdog configuration, possibly there are predefined upper and lower limits and a default for the watchdog time are defined, as well as a time definition in percent.

Activate watchdog: When this option is activated (☑) then regarding the currently set sensitivity (see below), the task will be terminated in error status as soon as the processing takes longer than defined in the 'Time' field (see below).

> **NOTICE!**
> Target system CODESYS SP 32-bit Full switches off the watchdog function when the flow control is active or when the execution currently is stopped at a breakpoint.

Time (e.g.: t#200ms): Watchdog time; after the expiration of this term, regarding the currently set sensitivity (see below), the watchdog will be activated unless the task has not been terminated already. Depending on the target system the time has to be entered as percent of the task interval. In this case the unit selection box is greyed and shows "%".

Sensitivity: Here you can enter in integer numbers at which overrun of the watchdog time an error should be generated. The default entry is "1", i.e. at the first overrun of the watchdog time an error occurs. Attention: If "0" is entered, the watchdog will be deactivated!

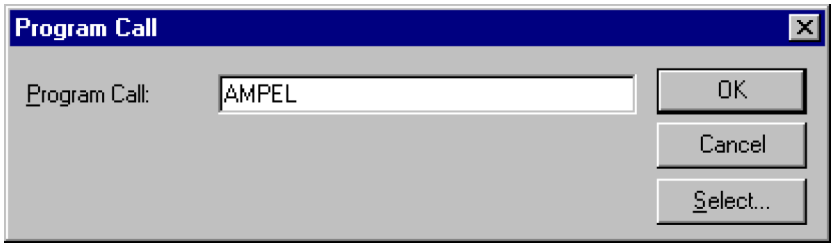
Manufacturer specific attributes:

Additionally to these standard attributes for the currently selected task manufacturer specific attributes might be displayed in a second tab "Parameters". This will be the case if it is defined in the target-specific description file for the task configuration.

'Insert' 'Insert Program Call' or 'Insert' 'Append Program Call'

With these commands you will open the dialog box for entering a program call to a task in the Task Configuration. Each entry in the task configuration tree consists of a symbol () and the program name.

With 'Insert Program Call', the new program call is inserted before the selected program call, and with 'Append Program Call' the program call is appended to the end of the existing list or program calls.



In the field 'program call' specify a valid program name out of your project or open the input assistant with the Select button to select a valid program name ↳ *Chapter 1.4.1.2.5.11 "'Edit' 'Input assistant'" on page 276*. The program name later also can be modified in the configuration tree. For this select the entry and press the <Space> key or just perform a mouseclick to open an editor field. If the selected program requires input variables, then enter these in their usual form and of the declared type (for example, prg(invar:=17)).

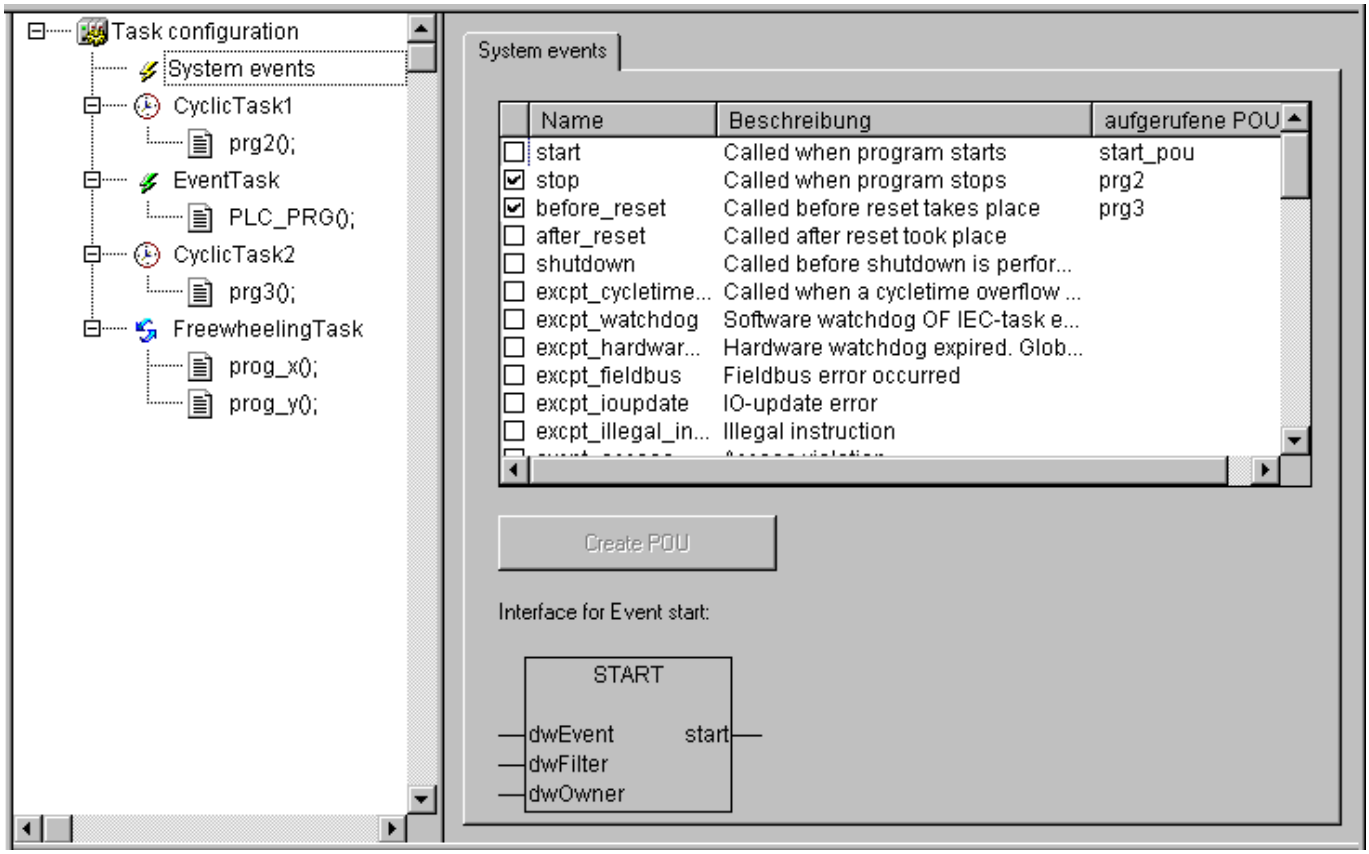
The processing of the program calls later in online mode will be done according to their order (top down) in the task editor.

Do not use the same string function in several tasks, because in this case values might be overwritten during processing of the tasks ↳ *Chapter 1.4.1.4.3.2 "Standard library" on page 372.*

System events

Instead of a "task" also a "system event" can be used to call a POU of your project. The available system events are target specific (definition in target file). The list of the standard events of the target may be extended by customer specific events. Possible events are for instance: Stop, Start, Online Change.

The assignment of system events to POUs is also done in the Task configuration editor. Use the dialog 'Events', which will be opened as soon as the entry " System-events" is selected in the task configuration tree:



Each event is represented in a line: Name and Description are displayed as defined in the target file, in the column called POU you can enter the name of the project POU which should be called and processed as soon as the event occurs.

For this use the input assistant (<F2>) or enter manually the name of an already existing POU (e.g. "PLC_PRG" or "PRG.ACT1"), or insert a name for a not yet existing POU.

In order to get a new POU (function) created in the project, press button Create POU. Hereupon the POU will be inserted in the Object Organizer. The input and output parameters which are required by the event will automatically be defined in the declaration part of the POU. Below the assignment table the currently selected event is displayed in a picture, showing the required parameters.

If you actually want the POU to be called by the event, activate the entry in the assignment table.

Activating/deactivating is done by a mouseclick on the control box.

'Extras' 'Set Debug Task'

With this command a debugging task can be set in Online mode in the task configuration ↬ *Chapter 1.4.1.4.8.3 “'Insert' 'Insert Program Call' or 'Insert' 'Append Program Call'” on page 393*. The text [DEBUG] will appear after the set task. The debugging capabilities apply, then, only to this task. In other words, the program only stops at a breakpoint if the program is gone through by the set task.

'Extras' 'Enable / disable task'

With this command the task which is currently marked in the task configuration can be disabled or re-enabled. A disabled task will not be regarded during processing of the program. In the configuration tree it is indicated by a grayed entry.

'Extras' 'Callstack'

This command is available in the Extras menu in the task configuration. If the program is stopped at a breakpoint during debugging, it can be used to show the callstack of the corresponding POU. For this purpose the debug task must be selected in the task configuration tree of ↬ *Chapter 1.4.1.4.8.5 “'Extras' 'Set Debug Task'” on page 395*. The window 'Callstack of task <task name>' will open. There you get the name of the POU and the breakpoint position (e.g. "prog_x (2)" for line 2 of POU prog_x) . Below the complete call stack is shown in backward order. If you press button 'Go To', the focus will jump to that position in the POU which is currently marked in the callstack.

1.4.1.4.9 Watch- and recipe manager

Overview

Function

In the Watch- and Recipe Manager (Resources tab of the Object Organizer) the current values of specified variables can be viewed in so-called "watch lists (Monitoring).

Further on the variables listed in a watch list can be preset with constant values and this set of values, named "Recipe" be transferred to the PLC. Also the current values of the variables of a watch list can be read from the PLC to the Watch and Recipe Manager as a preset/recipe. In this context regard the possibility to save recipes in files and to reload them to the Recipe Manager when required. See further information on the usage of recipes in: 'Creating Watch Lists, Recipes' ↬ *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

In online mode watch lists as well can be used to write and force variables ↬ *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286* ↬ *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*.

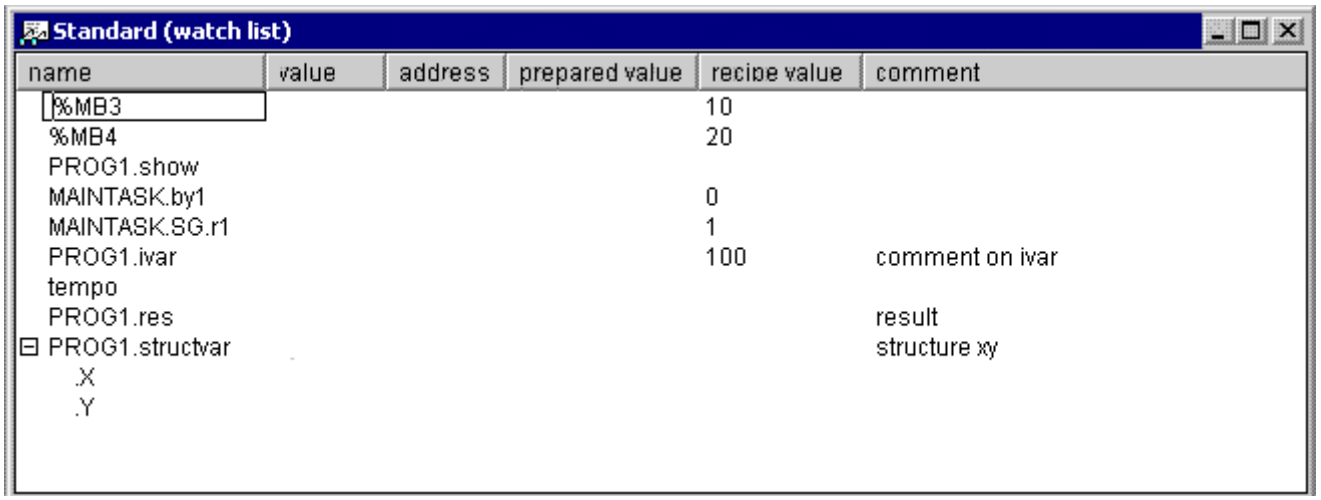
All these functions for example can be used for logging and setting of control parameters.

Editor

Tabular editor

Each watch list is viewed in a separate tabular editor window and multiple windows can be opened at the same time. In this case the available watch lists will be shown as entries in the "Resources" tab indented below the Watch and Recipe Manager. Each can be opened by a doubleclick on the entry.

The tabular editor contains columns for name, address, value, prepared value, recipe value and comment of the watch variable.



name: Here a variable identifier according to the following syntax must be entered of an address in standard format:

<POU name>.<variable name>

In case of global variables the POU name is dropped. The variable name can be multilevel. Addresses can be entered directly (e.g. "%IB0.0").

Example for a multilevel variable:	PLC_PRG.Instance1.Instance2.Structure.Component

Example for a global variable:	globalvar.component1

address, comment: As specified in the declaration of the variable.

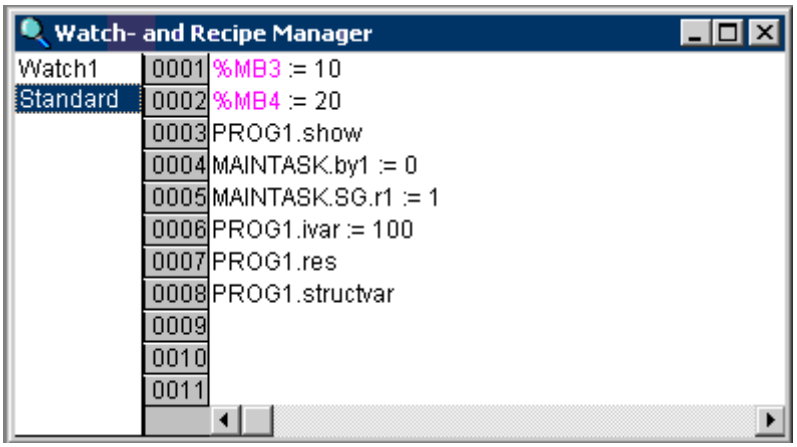
value: In online mode here the current value of the variable is displayed (Monitoring).

recipe value: Here a value can be entered, which will be transferred to the PLC when command 'Extras' 'Write Recipe' is applied on the whole watch list ↳ *Chapter 1.4.1.4.9.14 “'Extras' 'Write Recipe'” on page 401*. The recipe values of all variables of the list can be replaced by the current values from the PLC by using command 'Extras' 'Read Recipe' ↳ *Chapter 1.4.1.4.9.15 “'Extras' 'Read Recipe'” on page 402*.

In case of function block instances and structured variables a plus respectively minus sign appears in front of the name in the first column. It serves to expand resp. collapse the list of components. For function block variables the context menu is extended by the items 'Open function block' and 'Open instance'.

By a double-click on a non-editable position within the editor window, the table gets adapted to the window width and the column widths get optimized.

1-window-editor There is only one bipartite editor window, in the left part of which you find all available watch lists. For the list currently selected the right part of the window shows the associated variables. This editor view is opened via object 'Watch- and Recipe Manager' in the 'Resources' tab in the Object Organizer. The watch variables are entered line by line and a recipe value can be assigned to each by ":=".



Adding variables to watch lists is possible in offline mode by typing, or in online mode directly out of the POU editors. See 'Creating Watch Lists, Recipes' ↳ *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

A 'cross-reference list' can be called directly from a watch list, when one of the watch variables is selected. In this case the command 'Show cross references' in the Extras menu or in the context menu is available ↳ *Chapter 1.4.1.3.7 “Show cross references” on page 295*. On cross-reference list see: 'Project' 'Show cross reference' ↳ *Chapter 1.4.1.2.4.17 “'Project' 'Show cross reference'” on page 264*.

Creating watch lists, recipes

By default in each project an empty watch list "Standard" is automatically created. See in the following how further lists can be created and filled with watch variables and how recipes can be defined. Partly this depends on which (target dependant) variant of the Watch- and Recipe Manager is used:

Creating watch lists Create a new, empty list - in offline or online mode:

If entry 'Watch- and Recipe Manager' is selected in the 'Resources' tab, then in the tabular editor via command 'Object' 'Add' (menu 'Project' - 'Object', or context menu) a new, primarily empty watch list can be created ↳ *Chapter 1.4.1.4.9.1 “Overview” on page 395* ↳ *Chapter 1.4.1.2.4.6 “'Project' 'Object' ' Add'” on page 258*. In the 1-Window-Editor for this purpose the command 'New Watch List' (menu 'Insert' or context menu) is available ↳ *Chapter 1.4.1.4.9.1 “Overview” on page 395* ↳ *Chapter 1.4.1.4.9.3 “'Insert' 'New Watch List'” on page 399*. Each a dialog for entering an unique watch list name opens. After confirmation the new list will be immediately added in the Resources tree (tabular editor) resp. in the left part of the 1-Window-Editor.

Fill new list with watch variables out of the POU editors, resp. add variables to existing lists:

Only possible in online mode. It is not necessary to deactivate monitoring for this purpose ↳ *Chapter 1.4.1.4.9.13 “'Extras' 'Monitoring Active'” on page 401*.

Tabular Editor and 1-Window-Editor:

- If one or multiple variables or elements are selected in one of the POU editors, they can be directly entered in a new watch list by command 'Into new watch list' (menu 'Extras' or context menu) *Chapter 1.4.1.4.9.8 “'Extras' 'Into new watch list'” on page 400*. The new list will be added automatically as "Watch<n>" in the Resources tab below the Watch- and Recipe Manager and opened in the editor window. Renaming of the list is only possible in offline mode *Chapter 1.4.1.4.9.10 “'Extras' 'Rename Watch List'” on page 401*.
- If one or multiple variables or elements are selected in one of the POU editors, the variables can be directly added to one of the existing watch lists by command 'Add to watch list' (menu 'Extras' or context menu) *Chapter 1.4.1.4.9.7 “'Extras' 'Add to watch list'” on page 400*.
- Instead of a variable identifier an address in standard format can be entered in a watch list *Chapter 1.4.1.7.4.2 “Address” on page 441*. Command 'Insert address range' (menu 'Extras' or context menu) allows to add all addresses of a specified address range at one go *Chapter 1.4.1.4.9.9 “'Insert address range'” on page 400*.

Fill a list manually:

Possible in online and offline mode; for the syntax see: Overview, Editor *Chapter 1.4.1.4.9.1.2 “Editor” on page 396*

Tabular editor:

- If the watch list is opened, via commands 'Insert watch variable' or 'Insert' 'Attach watch variable'further variables can be added *Chapter 1.4.1.4.9.4 “'Insert' 'Insert watch variable'” on page 399* *Chapter 1.4.1.4.9.5 “'Insert' 'Attach watch variable'” on page 399*. Existing entries can be selected and removed by 'Delete watch variable' resp. <Del> *Chapter 1.4.1.4.9.6 “'Delete watch variable'” on page 400*.

1-Window-Editor:

- For entering variables in a watch list, the list must be selected in the left part of the Watch- and Recipe Manager Window. Then variables are added in the right part at the current cursor position line by line, either with the help of the input assistant <F2> (see Note on using the input assistant... below) or by typing. See Overview, Editor for the requested syntax *Chapter 1.4.1.4.9.1.2 “Editor” on page 396*. If the variables should be entered in online mode, previously the monitoring must be deactivated. Via command 'Monitoring active' in the 'Extras' or the context menu you can switch between activated and deactivated *Chapter 1.4.1.4.9.13 “'Extras' 'Monitoring Active'” on page 401*.

Note on using the input assistant in the watch- and recipe manager:

The non-structured view of the Input Assistant offers additional filter functions for selecting watch variables *Chapter 1.4.1.2.5.12 “Unstructured display” on page 276*: In the Filter input field you can enter a string and additionally specify whether this string must be found at the beginning of a variable name (Prefix), at the end of a variable name (Suffix), or at an arbitrary position within the variable name in order to get the watch variables offered for selection.

Working with recipes

The variables of a watch list can be preset with constant values. In the tabular editor this is possible via column recipe value, in the 1-Window-Editor by an assignment via ":=". Then the watch list can be used as a so-called "recipe".

If the presetting should be done automatically with the current values read from the PLC, then command 'Read Recipe' can be used.

In the 1-Window-Editor variant of the Watch- and Recipe Manager the recipe values must be assigned according to the following example:

Example: PLC_PRG.TIMER:=50

Here variable PLC_PRG.TIMER is preset with 50.

Note the following for variables of type array, structure or function block instance: The particular elements resp. instance variables must be entered explicitly in order to be able to preset them with values. Example: Let's assume that a structure STRU is defined with components a, b,c, and a structure variable struvar is declared in program PLC_PRG. In order to preset a,b,c with values, the following must be entered in the watch list:

PLC_PRG.struvar.a:=<value>

PLC_PRG.struvar.b:=<value>

PLC_PRG.struvar.c:=<value>

Correspondingly the presetting of elements of an array must be done: Example for an array variable arr_var of type ARRAY[0...6]

PLC_PRG.arr_var[0]:=<value>

PLC_PRG.arr_var[1]:=<value>

If a function block FB contains the variables x,y and an instance variable fb_inst of type fb is declared in PLC_PRG, then x and y can be preset as follows:

PLC_PRG.fb_inst.x:=<value>

PLC_PRG.fb_inst.y:=<value>

Recipes can be saved externally in a file via command *“Extras ➔ Save watch list”* and they can be loaded back to the editor via *“Extras ➔ Load watch list”*. In online mode the recipe values can be written to the variables on the PLC via *“Extras ➔ Write recipe”*.

↳ *Chapter 1.4.1.4.9.11 “'Extras' 'Save Watch List'” on page 401*

↳ *Chapter 1.4.1.4.9.12 “'Extras' 'Load Watch List'” on page 401*

↳ *Chapter 1.4.1.4.9.14 “'Extras' 'Write Recipe'” on page 401*

'Insert' 'New Watch List'

This command is only available, if the 1-Window-Editor view of the Watch- and Recipe Manager is opened ↳ *“Tabular editor” on page 396*. Then in offline as well as in online mode a new watch list can be created. A dialog for entering the watch list name will be opened for this purpose.

'Insert' 'Insert watch variable'

This command is only available in the tabular editor view of a watch list ↳ *Chapter 1.4.1.4.9.1.2 “Editor” on page 396*. It appends a new line at the end of the list and and opens an edit field in column "name" for entering a new watch variable via the input assistant (see the note in Creating Watch Lists, Recipes ↳ *“Note on using the input assistant in the watch- and recipe manager:” on page 398*) or by typing ↳ *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

'Insert' 'Attach watch variable'

This command is only available in the tabular editor view of a watch list ↳ *Chapter 1.4.1.4.9.1.2 “Editor” on page 396*. It appends a new line at the end of the list and and opens an edit field in column "name" for entering a new watch variable via the input assistant (see the note in Creating Watch Lists, Recipes ↳ *“Note on using the input assistant in the watch- and recipe manager:” on page 398*) or by typing ↳ *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

'Delete watch variable'

This command is only available in the tabular editor view of a watch list *Chapter 1.4.1.4.9.1.2 “Editor” on page 396*. It removes the currently focused line and corresponds to using the <Del> key.

'Extras' 'Add to watch list'

Short key: <Alt>+<X>+<W>

Out of the POU editors in the project you can directly add variables to a watch list.

When one or multiple variables or elements are selected in an editor, they can directly be added to one of the existing watch lists via command 'Add to watch list' ('Extras' menu or context menu). It is not necessary to deactivate the monitoring for this purpose *Chapter 1.4.1.4.9.13 “'Extras' 'Monitoring Active'” on page 401*.

For function blocks the command only will be available, if the full instance path of a variable within the POU is known.

Notes on the element selection in the POU editors:

Basically particular variables can be selected like usual for the respective editor. Additionally the following is true:

- In the FBD editor all elements on the left of a selected element will automatically be selected too. For example also the inputs a and b of an AND box will be selected when you select the AND box.
- In the Ladder editor multiple elements can be selected by keeping the <Shift> key pressed while clicking on the desired elements.
- In the ST and IL editor in the right part of the window a single variable can be selected. In the left part multiple lines can be selected in order to get all included variables added to the watch list.
- In the SFC editor you can select multiple successive elements in order to get all included variables added to the watch list.
- In the declaration parts of GVLs and POUs only particular variables can be selected.

'Extras' 'Into new watch list'

Short key: <Alt>+<X>+<N>

This command inserts the variable(s) currently selected in a POU editor to a new watch list. This list will automatically be created with name "Watch<n>", whereby n is a running number starting with 0 and used in a way that the name will be unique.

Furthermore for this command the same is true like for 'Extras' 'Add to watch list' *Chapter 1.4.1.4.9.7 “'Extras' 'Add to watch list'” on page 400*.

'Insert address range'

Instead of a variable name an address in standard format can be entered in column "name" of a watch list *Chapter 1.4.1.7.4.2 “Address” on page 441*.

The command allows to insert all addresses of a defined address range to the watch list at one go. For this purpose a dialog opens where you can specify the start address and the number of addresses. If you define for example name="%MW0" and number = "10", then the addresses %MW0 to %MW9 will be inserted in the watch list.

'Extras' 'Rename Watch List'

This command will only be available, if the 1-Window-Editor view of the Watch- and Recipe Manager is opened (target-dependent!) ↳ *"Tabular editor" on page 396*.

It opens a dialog for specifying the new watch list name.

If the tabular view is used, the watch list can be renamed in the Object Organizer via command 'Project' 'Object' 'Rename' ↳ *Chapter 1.4.1.2.4.8 "'Project' 'Object' 'Rename'" on page 259.*

'Extras' 'Save Watch List'

With this command you can save a watch list ↳ *Chapter 1.4.1.4.9.1 "Overview" on page 395*. The dialog box for saving a file is opened. The file name is preset with the name of the watch list and is given the extension "*.wtc".

The saved watch list can be loaded again with 'Extras' 'Load Watch List' ↳ *Chapter 1.4.1.4.9.12 "'Extras' 'Load Watch List'" on page 401*.

For example this allows to manage various recipes for the same set of variables, which can be loaded to the PLC as currently required ↳ *Chapter 1.4.1.4.9.2 "Creating watch lists, recipes" on page 397*.

'Extras' 'Load Watch List'

With this command you can reload a saved watch list, for example in order to reload a certain recipe to the editor, so that it can be written to the PLC ↳ *Chapter 1.4.1.4.9.2 "Creating watch lists, recipes" on page 397*. The dialog box for opening a file will be opened. Select the desired file with the "*.wtc" extension. In the dialog box that appears, you can give the watch list a new name. The file name is preset without an extension.

With 'Extras' 'Save Watch List', you can save a watch list ↳ *Chapter 1.4.1.4.9.11 "'Extras' 'Save Watch List'" on page 401*.

'Extras' 'Monitoring Active'

Short key: <Alt>+<X>+<M>

This command turns off resp. on the display of the values in the Watch- and recipe manager ↳ *Chapter 1.4.1.4.9.1 "Overview" on page 395*. If the Monitoring is activated, a check (✓) appears in front of the menu item.

'Extras' 'Write Recipe'

With this command in the online mode of the Watch- and Recipe Manager you can write the preset values () into the variables ↳ *recipe values*.

The command only concerns the values of that watch list, which is currently selected in the Watch- and Recipe Manager!

'Extras' 'Read Recipe'

With the command, in the Online Mode of the Watch- and Recipe Manager, you can replace the presetting of the variables with the present value of the variables.

Example: `PLC_PRG.Counter [:= <present value>] = <present value>`

The command only concerns the values of that watch list, which is currently selected in the Watch- and Recipe Manager!

Force and write values in the watch- and recipe manager

In online mode in each watch list you can force values and write values ↳ *Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287* ↳ *Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*.

In the tabular editor you can enter a value in column prepared value and thus prepare that value for forcing or writing on the PLC. For this purpose select the respective cell (mouseclick resp. navigation within the table via the arrow keys) and by a mouseclick or the <Enter> key open a dialog for entering the value. If they have the same data type, also the cells of multiple variables can be selected at once in order to get the same value prepared for all these variables. For multiselection of the cells just keep the <Shift> button pressed while selecting the particular cells.

In the 1-Window-Editor by a mouseclick on an specified variable value you also can open a corresponding dialog for entering the value which should be forced or written.

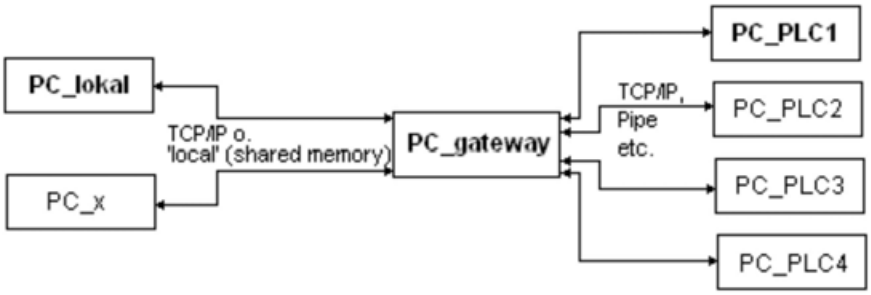
1.4.1.4.10 Workspace

This object in the 'Resources' tab provides an figure of the currently set project options. If you open it, you get the project options dialog with the know categories ↳ *Chapter 1.4.1.2.2.1 “'Project' 'Options'” on page 200*.

1.4.1.5 Principle of a gateway system

1.4.1.5.1 Overview

A gateway server can be used to allow your local PC to communicate with one or more runtime systems. The setting concerning which runtime systems can be addressed, which is specifically configured for each gateway server, and the connection to the desired gateway server, is made on the local PC. Here it is possible that both the gateway server and the runtime system(s) can run together on the local PC. If we are dealing with a gateway server which is running on another PC we must ensure that it has been started there. If you are selecting a locally installed gateway server, it automatically starts when you log onto the target runtime system. You can recognize this through the appearance of a symbol on the bottom right in the task bar. This symbol lights up as long as you are connected to the runtime system over the gateway. The menu points Info and Finish are obtained by clicking with the right mousekey on the symbol. Finish is used to switch off the gateway.



PC_local is your local PC, PC_x is another PC, which gateway addresses. PC_gateway is the PC on which the gateway server is installed, PC_PLC1 through to PC_PLC4 are PCs on which the runtime systems are running. The diagram shows the modules as separated but it is fully possible for the Gateway server and / or runtime systems to be installed together on the local PC.

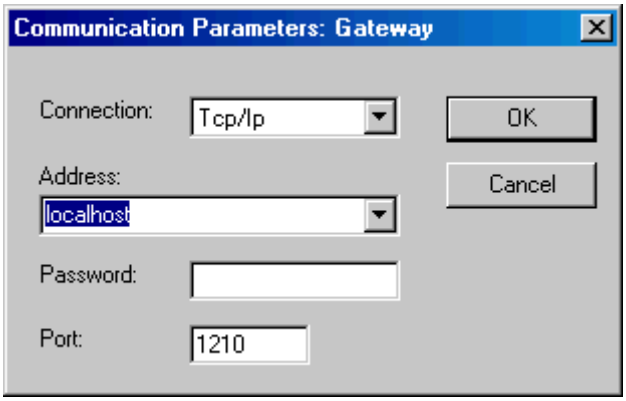
Please note that a connection to gateway is only possible over TCP/IP so make sure that your PC is configured appropriately.

The connections from gateway to the various runtime computers can, on the other hand, run over different protocols (TCP/IP, Pipe, etc.).

1.4.1.5.2 Setting up the desired gateway server and channel

Setting up the desired gateway server and channel in the communication parameters dialog

To define the connection to the desired gateway server we open the dialog 'Communication Parameters Gateway' by pressing the button Gateway.



Here you can enter and/or edit the following:

- The type of connection from your computer to the computer on which the gateway server that you want to use is running. If the gateway server is running on the local computer, connection via shared memory ("local") or via TCP/IP is possible; if connection to a different computer is needed, only TCP/IP can be used.
- The address of the computer, on which the gateway server that you want to use is running: IP address or the appropriate symbolic name such as e.g. localhost. Regard that leading zeros added to the address range number however are not allowed.

  Example: not possible: '010.107.084.050', must be entered as '10.107.84.50'). On initial setup, the standard 'localhost' is offered as the computer name (address), which means that the locally installed gateway would be accessed. The name 'localhost' is set to be identical to the local IP address 127.0.0.1 in most cases, but you may in some cases have to enter this directly into the Address field. If you want to access a gateway server on another computer, you must replace 'localhost' with its name or IP address.
- The password for the selected gateway server, if it is on a remote computer. If it is incorrectly entered, or not entered at all, an error message appears.

  Note in this connection: you can give the locally installed gateway server a password with the following procedure: click with the right mouse button on the gateway symbol in the lower right portion of the toolbar and select "Change password". A dialog comes up for changing or entering a password. If you access the gateway server locally any password that is entered will not be asked for.
- The computer’s port on which the gateway server that you wish to use is running, as a rule the correct value for the selected gateway is already given.

If the dialog is closed with OK, the corresponding entry (computer address) appears in the Channels field at the top of the 'Communication parameters' dialog, and below it the channels available on this gateway server.

Setting up the desired channel on the selected gateway server

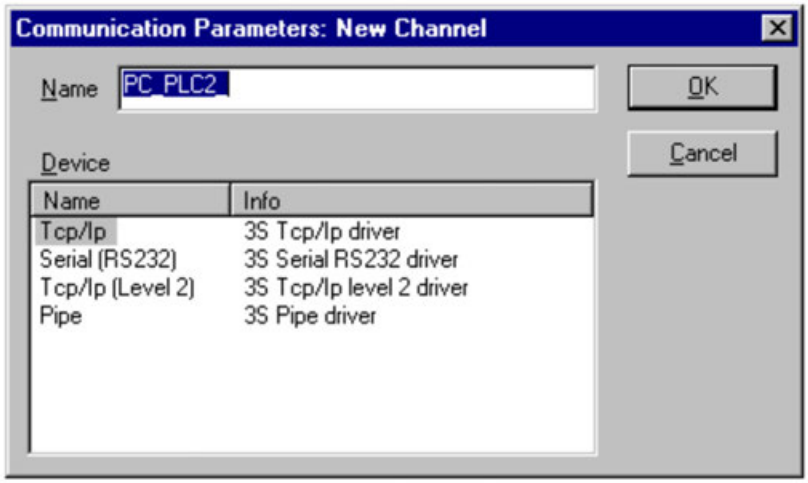
Now select one of the channels by clicking on an entry with the mouse. The corresponding parameters will then be shown in the table. If no connection can be established to the selected gateway address possibly because it has not been started or the address is incorrect "" the phrase 'not connected' appears in brackets after the address and a message 'No gateway with these settings could be found' appears. In this connection perform a quick check ↬ *Chapter 1.4.1.5.6 “Quick check in the event of unsuccessful connection attempt to the gateway” on page 407*.

Once the desired channel is set up, close the dialog using OK. The settings are saved with the project.

1.4.1.5.3 Setting up a new channel for the local gateway server

You can set up new channels for the currently connected gateway server, which are then available for establishing further connection from the server, a connection to a controller for example. The options that you have in this regard depend on the particular choice of the number of device drivers installed on your computer.

Press the New button in the 'Online' 'Communication Parameters' dialog ↬ *Chapter 1.4.1.2.6.23 “'Online' 'Communication parameters'” on page 291*. The dialog 'Communication Parameters: New Channel' comes up.



- The input field Name automatically contains the name used for the last inputted channel. If no channel has yet been defined, the current gateway name will be offered, followed by an underline character , e.g. 'localhost_'. You can edit the channel name at this point. The channel name is purely informative, it does not have to be a unique name but it is recommended to use one.
- The device drivers available on the gateway computer are listed in the table under Device. In the Name column, select by mouse click one of the available drivers; the corresponding comment, if any, appears in the Info column.

If you close the 'New Channel' dialog with OK, the newly defined channel appears in the 'Communication Parameters' dialog as a new entry in Channels at the lowest position under the minus sign. So far, it is only stored locally in the project. At this point you can edit the Value column. Now confirm the entered parameters with OK, thus leaving the 'Communication Parameters' dialog.

In order for the newly entered gateway channel and its parameters to also be known to the gateway server xy, and thus also to make it available to other computers that access this gateway xy, you must log into the runtime system. If you then re-open the 'Online' 'Communication parameters' dialog, the new channel appears in the „channel tree“, not only in its previous position but also indented under the address or name of the gateway server xy. This indicates that it is known to the network. You can now open the Communication Parameter dialog on a computer other than the local one, select gateway xy and use its new channel.

If a communication error occurs when logging in, it is possible that the interface cannot be opened (e.g. COM1 for a serial connection) possibly because it is being used by another device. It is also possible that the controller is not running.

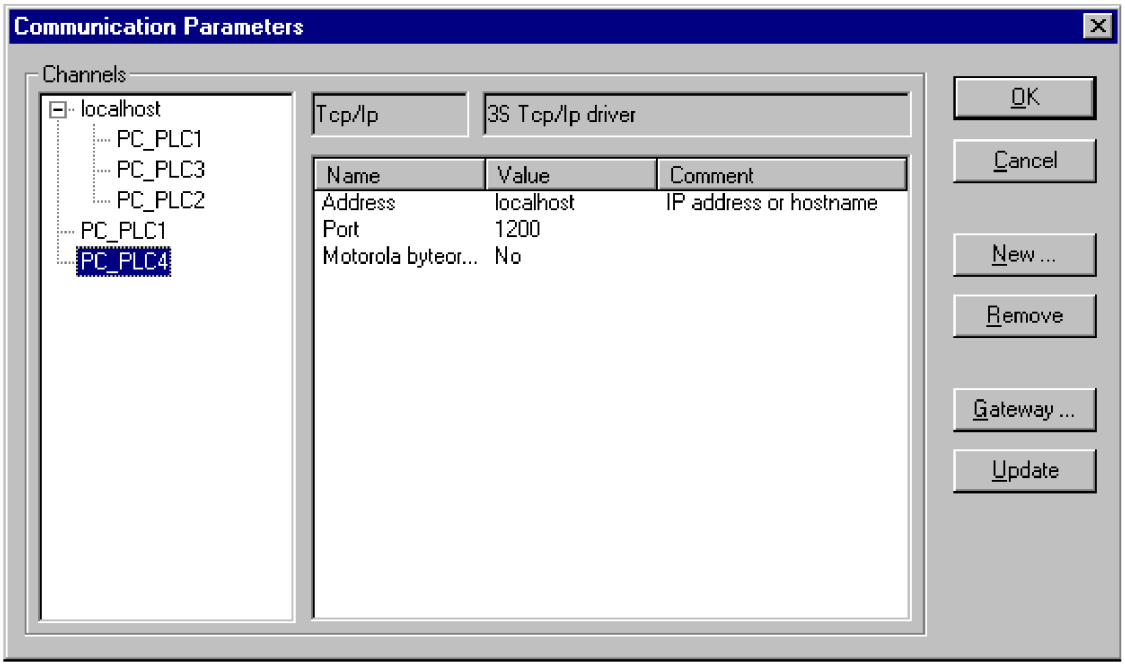
The parameters for a channel already known by the gateway server can no longer be edited in the configuration dialog. The parameter fields appear grey. You can, however, delete the connection as long as it is not active.

The deletion of a channel is not reversible. It occurs at the moment that you press on the button [Remove]!

1.4.1.5.4 What the communications parameters dialog on the local PC shows

This dialog is used to select a gateway server for the communication with a PLC. Furtheron there can be set up new channels for a gateway server which is installed on the local PC so that these channels can be used by other computers which are part of the network.

The current settings can be called up at any time using the button Update. A dialog will appear if the communications parameters have already been configured according to the example in Principle of a gateway system ↳ *Chapter 1.4.1.5.1 “Overview” on page 402*.



The Heading Channels lists two categories of connections:

- On the one hand all of the connections are shown which are installed on the currently connected gateway server called 'localhost'. Here the address or the name of this gateway is located on the upper position behind the minus sign, which in our example is running on the local computer. The appropriate address 'localhost' corresponds in the normal case to the IP address 127.0.0.1 of the local computer (PC_local). Below, indented to the right, are three addresses of runtime computers which the gateway channels are set-up to (PC_PLC1 to 3). They could have been configured both from the local PC or from the other PCs (PC_x) which are or were connected to the gateway server.
- The second category of the channels describes includes all connections to the gateway which can be set up from your local PC, over this configuration dialog for example. They create the "branch" which leads from the minus sign directly below to PC_PLC1 and PC_PLC4. These channel addresses do not necessarily have to be known yet at the gateway. For PC_PLC4 in the example described above, the configuration parameters are stored locally in the project, but they will first be known to the gateway the next time log-in to the runtime system occurs ↬ *Chapter 1.4.1.2.6.6 "'Online' 'Run'" on page 283*. This has already occurred for PC_PLC1 since the associated gateway address has appeared as an additional "sub-branch" to the "channel tree".

In the central part of the dialog one finds the designation, in each case, of the left selected channel and the associated parameter under Name, Value and Comment.

1.4.1.5.5 Tips for editing the parameters in the communications parameters dialogue

You can only edit the text fields in the column Value.

Select a text field with the mouse, and get into the editing mode by double clicking or by pressing the space bar. The text input is finished by pressing the <Enter> key.

You can use <Tabulator> or <Shift> + <Tabulator> to jump to the next or the previous switching or editing possibility.

To edit numerical values it is possible with the arrow keys or the Page Up/Down keys to change the value by one or ten units respectively. A double click with the mouse also changes the value by increasing by one unit. A typing check is installed for numerical values: <Ctrl> + <Home> or <Ctrl> + <End> deliver the lowest or the highest value respectively for the possible input values for the type of parameter in question.

1.4.1.5.6 Quick check in the event of unsuccessful connection attempt to the gateway

You should make the following checks if the connection to the selected gateway computer is not successful. (You get the message „not connected" in the Communication Parameters dialog behind the gateway server address in the field Channels):

- Has the gateway server been started (the three-color symbol appears in the bottom right portion of the toolbar) ?
- Is the IP address that you entered in the 'Gateway: Communication Parameters' dialog really that of the computer on which the gateway is running ? (use „ping" to check)
- Is the TCP/IP connection working locally? The error may possibly lie with TCP/IP.

1.4.1.6 IEC operators and additional, norm-extending functions

1.4.1.6.1 Overview

All IEC operators are supported. In contrast with the standard functions (see Appendix D of the Standard library ↬ *Chapter 1.4.1.4.3.2 “Standard library” on page 372*), these operators are recognized implicitly throughout the project.

Besides the IEC operators also the following operators are supported which are not prescribed by the standard: INDEXOF and SIZEOF (see 'Arithmetic Operators'Link auf obsoletes Modul ↬ *Chapter 1.4.1.6.2 “Arithmetic operators” on page 407*), ADR and BITADR (see 'Address Operators' ↬ *Chapter 1.4.1.6.7 “Address operators” on page 421*).

Operators are used like functions in POU.

1.4.1.6.2 Arithmetic operators

ADD

IEC operator ↬ *Chapter 1.4.1.6.1 “Overview” on page 407*: Addition of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

Two TIME variables can also be added together resulting in another time (e.g., t#45s + t#50s = t#1m35s)

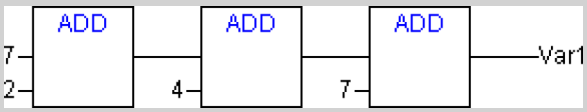
Example in IL:

```
LD  7
ADD 2,4,7
ST  Var1
```

Example in ST:

```
var1 := 7+2+4+7;
```

Example in FBD:



MUL

IEC operator ↬ *Chapter 1.4.1.6.1 “Overview” on page 407*: Multiplication of variables of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

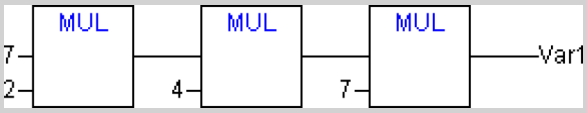
Example in IL:

```
LD 7
MUL 2,4,7
ST Var1
```

Example in ST:

```
var1 := 7*2*4*7;
```

Example in FBD:



SUB

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Subtraction of one variable from another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

A TIME variable may also be subtracted from another TIME variable resulting in third TIME type variable. Note that negative TIME values are undefined.

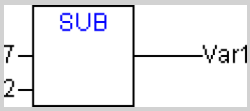
Example in IL:

```
LD  7
SUB 2
ST  Var1
```

Example in ST:

```
var1 := 7-2;
```

Example in FBD:



DIV

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL and LREAL.

Example in IL:

```
LD  8
DIV 2
ST  Var1 (* Result is 4 *)
```

Example in ST:

```
var1 := 8/2;
```

Example in FBD:

If you define functions in your project with the names CheckDivByte, CheckDivWord, CheckDivDWord and CheckDivReal, you can use them to check the value of the divisor if you use the operator DIV, for example to avoid a division by 0. The functions must have the above listed names.

NOTICE!
Note that different target systems may behave differently concerning a division by zero!

MOD

IEC operator *Chapter 1.4.1.6.1 "Overview" on page 407*: Module Division of one variable by another of the types: BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT. The result of this function will be the remainder of the division. This result will be a whole number.

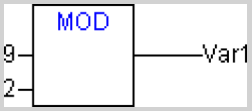
Example in IL:

```
LD 9
MOD 2
ST Var1 (* Result is 1 *)
```

Example in ST:

```
var1 := 9 MOD 2;
```

Example in FBD:

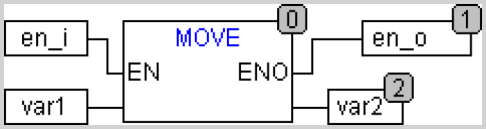


MOVE

IEC operator *Chapter 1.4.1.6.1 "Overview" on page 407*: Assignment of a variable to another variable of an appropriate type. As MOVE is available as a box in the graphic editors LD, CFC, there the (unlocking) EN/EN0 functionality can also be applied on a variable assignment. In the FBD editor this is not possible however.

Example in CFC in conjunction with the EN/EN0 function:

Only if en_i is TRUE, var1 will be assigned to var2.



Example in IL:

```
LD ivar1
MOVE
ST ivar2 (* Result: ivar2 gets assigned value of ivar1 *)
```

You get the same result with:

```
LD ivar1
ST ivar2
```

Example in ST:
```
ivar2 := MOVE(ivar1);
```
You get the same result with: ivar2 := ivar1;

INDEXOF

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: This function is not prescribed by the standard IEC61131-3.

Perform this function to find the internal index for a POU.

Example in ST:
```
var1 := INDEXOF(POU2);
```

SIZEOF

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: This function is not prescribed by the standard IEC 61131-3.

Perform this function to determine the number of bytes required by the given variable.

Example in IL:
```
arr1:ARRAY[0..4] OF INT;
Var1 INT
LD arr1
SIZEOF
ST Var1 (* Result is 10 *)
```

Example in ST:
```
var1 := SIZEOF(arr1);
```

1.4.1.6.3 Bitstring operators

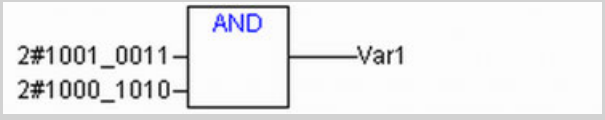
AND

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Bitwise AND of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

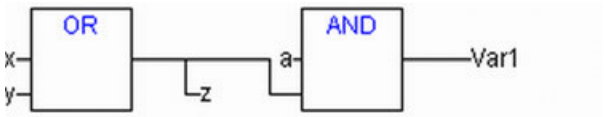
Example in IL:
```
Var1 BYTE
LD 2#1001_0011
AND 2#1000_1010
ST Var1 (* Result is 2#1000_0010 *)
```

Example in ST:
```
var1 := 2#1001_0011 AND 2#1000_1010
```

Example in FBD:



If you have a program step in the SFC like the following and if you use 68xxx generators, please note the following: The allocation of the value of the second input variable at the AND operator module to variable z will not be executed! This is due to the optimized processing in the SFC in case of value FALSE at the input variable.



OR

IEC operator ↳ *Chapter 1.4.1.6.1 "Overview" on page 407*: Bitwise OR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

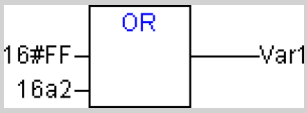
Example in IL:

```
var1 :BYTE;
LD 2#1001_0011
OR 2#1000_1010
ST  var1 (* Result is 2#1001_1011 *)
```

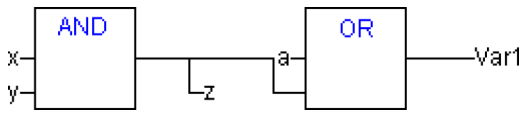
Example in ST:

```
Var1 := 2#1001_0011 OR 2#1000_1010
```

Example in FBD:



If you have a program step in the SFC like the following and if you use 68xxx generators, please note the following: The allocation of the value of the second input variable at the OR operator module to variable z will not be executed! This is due to the optimized processing in the SFC in case of value FALSE at the input variable.



XOR

IEC operator ↳ *Chapter 1.4.1.6.1 "Overview" on page 407*: Bitwise XOR of bit operands. The operands should be of the type BOOL, BYTE, WORD or DWORD.

Note the behavior of the XOR function in extended form, that means if there are more than 2 inputs. The inputs will be checked in pairs and the particular results will then be compared again in pairs (this complies with the standard, but may not be expected by the user).

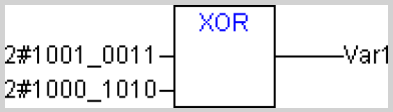
Example in IL:

```
Var1 :BYTE;
LD 2#1001_0011
XOR 2#1000_1010
ST  Var1 (* Result is 2#0001_1001 *)
```

Example in ST:

```
Var1 := 2#1001_0011 XOR 2#1000_1010
```

Example in FBD:



NOT

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Bitwise NOT of a bit operand. The operand should be of the type BOOL, BYTE, WORD or DWORD.

Example in IL:

```
Var1 :BYTE;
LD 2#1001_0011
NOT
ST  Var1 (* Result is 2#0110_1100 *)
```

Example in ST:

```
Var1 := NOT 2#1001_0011
```

Example in FBD:

1.4.1.6.4 Bit-Shift operators

SHL

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Bitwise left-shift of an operand : erg:= SHL (in, n)

in gets shifted to the left by n bits. If n > data type width, for BYTE, WORD and DWORD will be filled with zeros. But if signed data types are used, like e.g. INT, then an arithmetic shift will be executed in such cases, that means it will be filled with the value of the topmost bit.

> *The amount of bits used for the arithmetic operation is determined by the data type of the input variable! If the input variable is a constant the smallest possible data type is used. The data type of the output variable has no effect at all on the arithmetic operation.*

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

Example in IL:

```
LD  16#45
SHL 2
ST  erg_byte
```

Example in ST:

```
PROGRAM shl_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHL(in_byte,n); (* Result is 16#14 *)
erg_word:=SHL(in_word,n); (* Result is 16#0114 *)
```

Example in FBD:

SHR

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Bitwise righ-shift of an operand : erg:= SHL (in, n)

in gets shifted to the right by n bits. If n > data type width, for BYTE, WORD and DWORD will be filled with zeros. But if signed data types are used, like e.g. INT, then an arithmetic shift will be executed in such cases, that means it will be filled with the value of the topmost bit.

> *The amount of bits used for the arithmetic operation is determined by the data type of the input variable! If the input variable is a constant the smallest possible data type is used. The data type of the output variable has no effect at all on the arithmetic operation.*

See in the following example in hexadecimal notation that you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

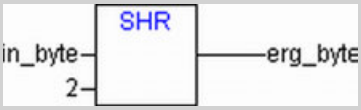
Example in IL:

```
LD  16#45
SHR 2
ST  erg_byte
```

Example in ST:

```
PROGRAM shr_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=SHR(in_byte,n); (* Result is 11 *)
erg_word:=SHR(in_word,n); (* Result is 0011 *)
```

Example in FBD:



ROL

IEC operator ↳ *Chapter 1.4.1.6.1 "Overview" on page 407*: Bitwise rotation of an operand to the left: erg:= ROL (in, n)

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the left n times while the bit that is furthest to the left will be reinserted from the right.

> *The amount of bits used for the arithmetic operation is determined by the data type of the input variable! If the input variable is a constant the smallest possible data type is used. The data type of the output variable has no effect at all on the arithmetic operation.*

In the following example in hexadecimal notation you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

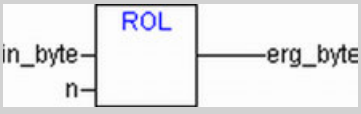
Example in IL:

```
LD  16#45
ROL 2
ST erg_byte
```

Example in ST:

```
PROGRAM rol_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROL(in_byte,n); (* Result is 16#15 *)
erg_word:=ROL(in_word,n); (* Result is 16#0114 *)
```

Example in FBD:



ROR

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Bitwise rotation of an operand to the right: erg:= ROL (in, n)

erg, in and n should be of the type BYTE, WORD or DWORD. in will be shifted one bit position to the right n times while the bit that is furthest to the right will be reinserted from the left.

The amount of bits used for the arithmetic operation is determined by the data type of the input variable! If the input variable is a constant the smallest possible data type is used. The data type of the output variable has no effect at all on the arithmetic operation.

In the following example in hexadecimal notation you get different results for erg_byte and erg_word depending on the data type of the input variable (BYTE or WORD), although the values of the input variables in_byte and in_word are the same.

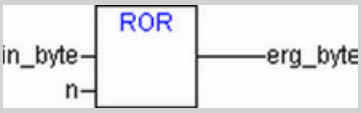
Example in IL:

```
LD 16#45
ROR 2
ST erg_byte
```

Example in ST:

```
PROGRAM ror_st
VAR
in_byte : BYTE:=16#45;
in_word : WORD:=16#45;
erg_byte : BYTE;
erg_word : WORD;
n: BYTE :=2;
END_VAR
erg_byte:=ROR(in_byte,n); (* Result is 16#51 *)
erg_word:=ROR(in_word,n); (* Result is 16#4011 *)
```

Example in FBD:



1.4.1.6.5 Selection operators

Overview

All selection operations can also be performed with variables. For purposes of clarity we will limit our examples to the following which use constants as operators.

See also:

SEL ↳ *Chapter 1.4.1.6.5.2 “SEL” on page 416*

MAX ↳ *Chapter 1.4.1.6.5.3 “MAX” on page 416*

MIN ↳ *Chapter 1.4.1.6.5.4 “MIN” on page 417*

LIMIT ↳ *Chapter 1.4.1.6.5.5 “LIMIT” on page 417*

MUX ↳ *Chapter 1.4.1.6.5.6 “MUX” on page 418*

SEL

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Binary selection.

```
OUT := SEL(G, IN0, IN1) means:
OUT := IN0 if G=FALSE;
OUT := IN1 if G=TRUE
```

IN0, IN1 and OUT can be any type of variable, G must be BOOL. The result of the selection is IN0 if G is FALSE, IN1 if G is TRUE.

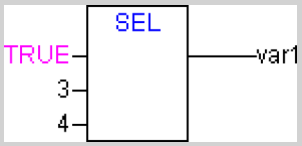
Example in IL:

```
LD  TRUE
SEL 3,4   (* IN0 = 3, IN1 =4 *)
ST  Var1  (* Result is 4 *)
LD  FALSE
SEL 3,4
ST  Var1  (* Result is 3 *)
```

Example in ST:

```
Var1:=SEL(TRUE,3,4); (* Result is 4 *)
```

Example in FBD:



An expression occurring ahead of IN1 or IN2 will not be processed if IN0 is TRUE.

MAX

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Maximum function. Returns the greater of the two values.

```
OUT := MAX(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

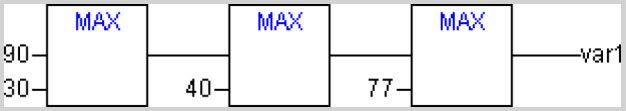
Example in IL:

```
LD 90
MAX 30
MAX 40
MAX 77
ST Var1 (* Result is 90 *)
```

Example in ST:

```
Var1:=MAX(30,40); (* Result is 40 *)
Var1:=MAX(40,MAX(90,30)); (* Result is 90 *)
```

Example in FBD:



MIN

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Minimum function. Returns the lesser of the two values.

```
OUT := MIN(IN0, IN1)
```

IN0, IN1 and OUT can be any type of variable.

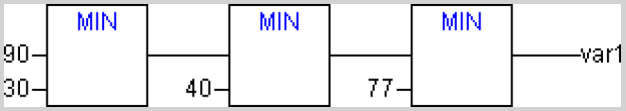
Example in IL:

```
LD 90
MIN 30
MIN 40
MIN 77
ST Var1 (* Result is 30 *)
```

Example in ST:

```
Var1:=MIN(90,30); (* Result is 30 *);
Var1:=MIN(MIN(90,30),40); (* Result is 30 *);
```

Example in FBD:



LIMIT

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Limiting

```
OUT := LIMIT(Min, IN, Max) means:
OUT := MIN (MAX (IN, Min), Max)
```

Max is the upper and Min the lower limit for the result. Should the value IN exceed the upper limit Max, LIMIT will return Max. Should IN fall below Min, the result will be Min.

IN and OUT can be any type of variable.

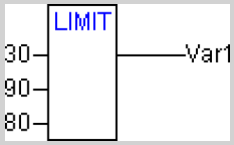
Example in IL:

```
LD 90
LIMIT 30,80
ST Var1 (* Result is 80 *)
```

Example in ST:

```
Var1:=LIMIT(30,90,80); (* Result is 80 *);
```

Example in FBD:



MUX

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Multiplexer.

```
OUT := MUX(K, IN0,...,INn) means:
OUT := INK
```

IN0, ...,INn and OUT can be any type of variable. K must be BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT or UDINT. MUX selects the Kth value from among a group of values.

Example in IL:

```
LD 0
MUX 30,40,50,60,70,80
ST Var1 (* Result is 30 *)
```

Example in ST:

```
Var1:=MUX(0,30,40,50,60,70,80); (* Result is 30 *);
```

An expression occurring ahead of an input other than INK will not be processed to save run time! Only in simulation mode all expressions will be executed.

1.4.1.6.6 Comparison operators

GT

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Greater than.

A Boolean operator which returns the value TRUE when the value of the first operand is greater than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

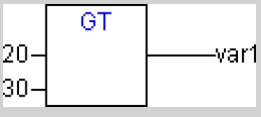
Example in IL:

```
LD 20
GE 30
ST  Var1 (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 20 > 30 > 40 > 50 > 60 > 70;
```

Example in FBD:



LT

↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Less than.

A Boolean operator that returns the value TRUE when the value of the first operand is less than that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

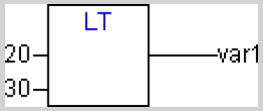
Example in IL:

```
LD 20
LT 30
ST Var1 (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 < 30;
```

Example in FBD:



LE

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Less than or equal to.

A Boolean operator that returns the value TRUE when the value of the first operand is less than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

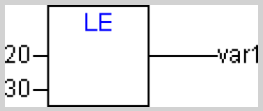
Example in IL:

```
LD 20
LE 30
ST  Var1 (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 20 <= 30;
```

Example in FBD:



GE

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Greater than or equal to.

A Boolean operator that returns the value TRUE when the value of the first operand is greater than or equal to that of the second. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

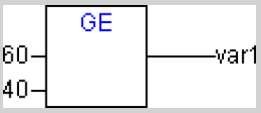
Example in IL:

```
LD 60
GE 40
ST  Var1 (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 60 >= 40;
```

Example in FBD:



EQ

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Equal to

A Boolean operator that returns the value TRUE when the operands are equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

Example in IL:

```
LD 40
EQ 40
ST Var1 (* Result is TRUE *)
```

Example in ST:

```
VAR1 := 40 = 40;
```

Example in FBD:

NE

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Not equal to

A Boolean operator that returns that value TRUE when the operands are not equal. The operands can be BOOL, BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL, LREAL, TIME, DATE, TIME_OF_DAY, DATE_AND_TIME and STRING.

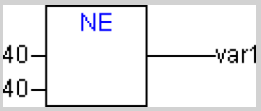
Example in IL:

```
LD 40
NE 40
ST Var1 (* Result is FALSE *)
```

Example in ST:

```
VAR1 := 40 <> 40;
```

Example in FBD:



1.4.1.6.7 Address operators

ADR

IEC operator: Address function not prescribed by the standard IEC 61131-3 ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*.

dwVar:=ADR(bVAR);

Example in IL:

```
LD bVar
ADR
ST dwVar
man_fun1
```

> **NOTICE!**
> After an online change there might be changes concerning the data on certain addresses. Please regard this in case of using pointers on addresses.

ADRINST

Operator: Address function not prescribed by the standard IEC 61131-3 ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*.

ADRINST can be used within a function block instance to return the address of this instance in a DWORD.

Examples in ST (within a function block instance):

```
dvar:=ADRINST();   (* Address of the instance is written to
variable dvar *)
fun(a:=ADRINST()); (* The instance address is given to input
parameter a of function fun *)
```

Examples in IL:

```
ADRINST
ST dvar

ADRINST
fun
```

BITADR

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Address function not prescribed by the standard IEC 61131-3.

BITADR returns the bit offset within the segment in a DWORD. Note that the offset value depends on whether the option byte addressing in the target settings is activated or not.

```
VAR
 var1 AT %IX2.3:BOOL;
 bitoffset: DWORD;
END_VAR
```

Example in ST:

```
bitoffset:=BITADR(var1); (* Result if byte addressing=TRUE: 19, if
byte addressing=FALSE: 35 *)
```

Example in IL:

```
LD Var1
BITADR
ST Var2
```

NOTICE!
After an Online Change there might be changes concerning the data on certain addresses. Please regard this in case of using pointers on addresses.

Content operator

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

Example in ST:

```
pt:POINTER TO INT;
var_int1:INT;
var_int2:INT;
pt := ADR(var_int1);
var_int2:=pt^;
```

NOTICE!
After an Online Change there might be changes concerning the data on certain addresses. Please regard this in case of using pointers on addresses.

1.4.1.6.8 Calling operators

CAL

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Calling a function block or a program

Use CAL in IL to call up a function block instance. The variables that will serve as the input variables are placed in parentheses right after the name of the function block instance.

Example: Calling up the instance Inst from a function block where input variables Par1 and Par2 are 0 and TRUE respectively.

```
CAL INST(PAR1 := 0, PAR2 := TRUE)
```

1.4.1.6.9 Type conversion

Type conversion functions

Its is forbidden to implicitly convert from a "larger" type to a "smaller" type (for example from INT to BYTE or from DINT to WORD). Special type conversions are required if one wants to do this. One can basically convert from any elementary type to any other elementary type.

Syntax:

`<elem.Typ1>_TO_<elem.Typ2>`

Please note that at ...TO_STRING conversions the string is generated left-justified. If it is defined to short, it will be cut from the right side.

See also:

- BOOL_TO conversions *Chapter 1.4.1.6.9.2 “BOOL_TO conversions” on page 423*
- TO_BOOL conversions *Chapter 1.4.1.6.9.3 “TO_BOOL conversions” on page 424*
- Conversion between integral number types *Chapter 1.4.1.6.9.4 “Conversion between integral number types” on page 425*
- REAL_TO-/ LREAL_TO conversions *Chapter 1.4.1.6.9.5 “REAL_TO-/ LREAL_TO conversions” on page 426*
- TIME_TO/TIME_OF_DAY conversions *Chapter 1.4.1.6.9.6 “TIME_TO/TIME_OF_DAY conversions” on page 426*
- DATE_TO/DT_TO conversions *Chapter 1.4.1.6.9.7 “DATE_TO/DT_TO conversions” on page 427*
- STRING_TO conversions *Chapter 1.4.1.6.9.8 “STRING_TO conversions” on page 428*

BOOL_TO conversions

Conversion from type BOOL to any other type:

For number types the result is 1, when the operand is TRUE, and 0, when the operand is FALSE.

For the STRING type the result is 'TRUE' or 'FALSE'.

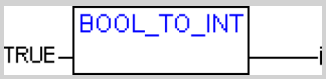
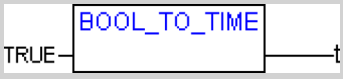
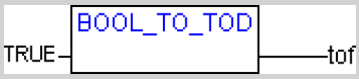
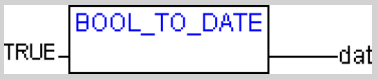
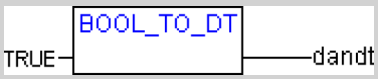
Examples in IL:

```
(*Result is 1 *)
LD TRUE
BOOL_TO_INT
ST i

(*Result is 'TRUE' *)
LD TRUE
BOOL_TO_STRING
ST str

(*Result is T#1ms *)
LD TRUE
BOOL_TO_TIME
ST t

(*Result is TOD#00:00:00.001 *)
LD TRUE
BOOL_TO_TOD
ST

(*Result is D#1970-01-01 *)
LD FALSE
BOOL_TO_DATE
ST dat

(*Result is DT#1970-01-01-00:00:01 *)
LD TRUE
BOOL_TO_DT
ST dandt
```

Examples in ST:

```
(* Result is 1 *)
i:=BOOL_TO_INT(TRUE);

(* Result is "TRUE" *)
str:=BOOL_TO_STRING(TRUE);

(* Result is T#1ms *)
t:=BOOL_TO_TIME(TRUE);

(* Result is TOD#00:00:00.001 *)
tof:=BOOL_TO_TOD(TRUE);

(* Result is D#1970 *)
dat:=BOOL_TO_DATE(FALSE);

(* Result is DT#1970-01-01-00:00:01 *)
dandt:=BOOL_TO_DT(TRUE);
```

Examples in FBD:

	`Result is 1`
	`Result is "TRUE"`
	`Result is T#1ms`
	`Result is TOD#00:00:00.001`
	`Result is D#1970-01-01`
	`Result is DT#1970-01-01-00:00:01`

TO_BOOL conversions

Conversion from another variable type to BOOL:

The result is TRUE when the operand is not equal to 0. The result is FALSE when the operand is equal to 0.

The result is true for STRING type variables when the operand is "TRUE", otherwise the result is FALSE.

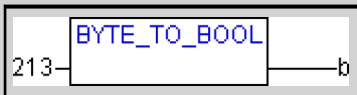
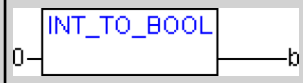
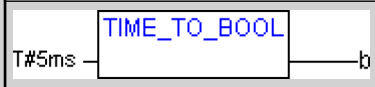
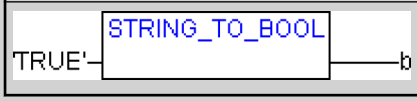
Examples in IL:

```
(*Result is TRUE *)
LD 213
BYTE_TO_BOOL
ST b

(*Result is FALSE *)
LD 0
INT_TO_BOOL
ST b

(*Result is TRUE *)
LD T#5ms
TIME_TO_BOOL
ST b

(*Result is TRUE *)
LD 'TRUE'
STRING_TO_BOOL
ST b
```

Examples in ST:

```
(* Result is TRUE *)
b := BYTE_TO_BOOL(2#11010101);

(* Result is FALSE *)
b := INT_TO_BOOL(0);

(* Result is TRUE *)
b := TIME_TO_BOOL(T#5ms);

(* Result is TRUE *)
b := STRING_TO_BOOL('TRUE');
```

Examples in FBD:

	Result is TRUE
	Result is FALSE
	Result is TRUE
	Result is TRUE

Conversion between integral number types

Conversion from an integral number type to another number type:

When you perform a type conversion from a larger to a smaller type, you risk losing some information. If the number you are converting exceeds the range limit, the first bytes for the number will be ignored.

Example in ST:

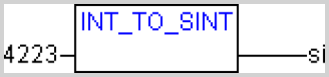
```
si := INT_TO_SINT(4223); (* Result is 127 *)
```

If you save the integer 4223 (16#107f represented hexadecimally) as a SINT variable, it will appear as 127 (16#7f represented hexadecimally).

Example in IL:

```
LD 2
INT_TO_REAL
MUL
```

Example in FBD:



REAL_TO-/ LREAL_TO conversions

Converting from the variable type REAL or LREAL to a different type:

The value will be rounded up or down to the nearest whole number and converted into the new variable type. Exceptions to this are the variable types STRING, BOOL, REAL and LREAL.

Please note at a conversion to type STRING that the total number of digits is limited to 16. If the (L)REAL-number has more digits, then the sixteenth will be rounded. If the length of the STRING is defined to short, it will be cut beginning from the right end.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

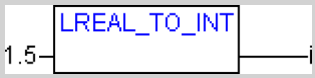
Examples in IL:

```
LD 2.7
REAL_TO_INT
GE %MW8
```

Examples in ST:

```
i := REAL_TO_INT(1.5); (* Result is 2 *)
j := REAL_TO_INT(1.4); (* Result is 1 *)
i := REAL_TO_INT(-1.5); (* Result is -2 *)
j := REAL_TO_INT(-1.4); (* Result is -1 *)
```

Examples in FBD:



TIME_TO/TIME_OF_DAY conversions

Converting from the variable type TIME or TIME_OF_DAY to a different type:

The time will be stored internally in a DWORD in milliseconds (beginning with 12:00 A.M. for the TIME_OF_DAY variable). This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

For the STRING type variable, the result is a time constant.

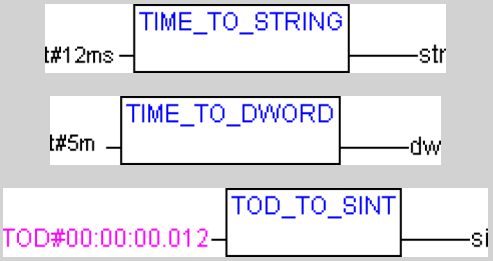
Examples in IL:

```
(*Result is 'T#12ms' *)
LD T#12ms
TIME_TO_STRING
ST str

(*Result is 300000 *)
LD T#300000ms
TIME_TO_DWORD
ST dw

(*Result is 12 *)
LD TOD#00:00:00.012
TOD_TO_SINT
ST si
```

Examples in ST:

```
(* Result is T#12ms *)
#str :=TIME_TO_STRING(T#12ms);

(* Result is 300000 *)
dw:=TIME_TO_DWORD(T#5m);

(* Result is 12 *)
si:=TOD_TO_SINT(TOD#00:00:00.012);
```

Examples in FBD:



DATE_TO/DT_TO conversions

Converting from the variable type DATE or DATE_AND_TIME to a different type:

The date will be stored internally in a DWORD in seconds since Jan. 1, 1970. This value will then be converted.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

For STRING type variables, the result is the date constant.

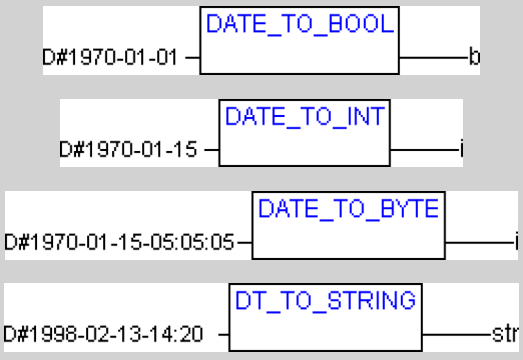
Examples in IL:

```
(* Result is FALSE *)
LD D#1970-01-01
DATE_TO_BOOL
ST b

(* Result is 29952 *)
LD D#1970-01-15
DATE_TO_INT
ST i

(* Result is 129 *)
LD DT#1970-01-15-05:05:05
DT_TO_BYTE
ST byt

(* Result is 'DT#1998-02-13-14:20' *)
LD DT#1998-02-13-14:20
DT_TO STRING
ST str
```

Examples in ST:

```
(* Result is FALSE *)
b :=DATE_TO_BOOL(D#1970-01-01);

(* Result is 29952 *)
i :=DATE_TO_INT(D#1970-01-15);

(* Result is 129 *)
byt :=DT_TO_BYTE(DT#1970-01-15-05:05:05);

(* Result is 'DT#1998-02-13-14:20' *)
str:=DT_TO_STRING(DT#1998-02-13-14:20);
```

Examples in FUP:



STRING_TO conversions

Converting from the variable type STRING to a different type:

The operand from the STRING type variable must contain a value that is valid in the target variable type, otherwise the result will be 0.

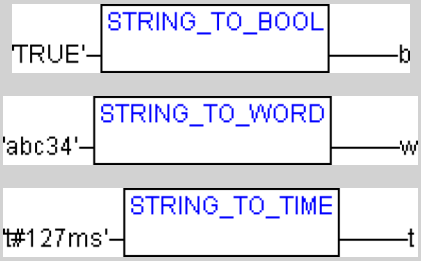
Examples in IL:

```
(* Result is TRUE *)
LD 'TRUE'
STRING_TO_BOOL
ST b

(* Result is 0 *)
LD 'abc34'
STRING_TO_WORD
ST w

(* Result is T#127ms *)
LD 't#127ms'
STRING_TO_TIME
ST t
```

Examples in ST:

```
(* Result is TRUE *)
b :=STRING_TO_BOOL('TRUE');

(* Result is 0 *)
w :=STRING_TO_WORD('abc34');

(* Result is T#127ms *)
t :=STRING_TO_TIME('T#127ms');
```

Examples in FBD:



TRUNC

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Converting from REAL to INT. The whole number portion of the value will be used.

When you perform a type conversion from a larger to a smaller type, you risk losing some information.

Example in IL:

```
LD 2.7 TRUNC GE %MW8
```

Examples in ST:

```
i:=TRUNC(1.9); (* Result is 1 *)
i:=TRUNC(-1.4); (* Result is -1 *)
```

1.4.1.6.10 Numeric operators

ABS

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the absolute value of a number. ABS(-2) equals 2.

The following type combinations for input and output variables are possible:

IN	OUT
INT	INT, REAL, WORD, DWORD, DINT
REAL	REAL
BYTE	INT, REAL, BYTE, WORD, DWORD, DINT
WORD	INT, REAL, WORD, DWORD, DINT
DWORD	REAL, DWORD, DINT
SINT	REAL
USINT	REAL
UINT	INT, REAL, WORD, DWORD, DINT, UDINT, UINT
DINT	REAL, DWORD, DINT
UDINT	REAL, DWORD, DINT, UDINT

Example in IL:

```
LD  -2
ABS
ST  i  (* Result is 2 *)
```

Example in ST:

```
i:=ABS(-2);
```

Example in FBD:

SQRT

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the square root of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT. OUT must be type REAL.

Example in IL:

```
LD  16
SQRT
ST  q  (* Result is 4 *)
```

Example in ST:

```
q:=SQRT(16);
```

Example in FBD:

LN

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the natural logarithm of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

OUT must be type REAL.

Example in IL:

```
LD 45
LN
ST  q (* Result is 3.80666 *)
```

Example in ST:

```
q:=LN(45);
```

Example in FBD:

LOG

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the logarithm of a number in base 10.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT.

OUT must be type REAL.

Example in IL:

```
LD 314.5
LOG
ST  q (* Result is 2.49762 *)
```

Example in ST:

```
q:=LOG(314.5);
```

Example in FBD:

EXP

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the exponential function.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD  2
EXP
ST  q (* Result is 7.389056099 *)
```

Example in ST:

```
q:=EXP(2);
```

Example in FBD:

SIN

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the sine of a number. The result is calculated in radians.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

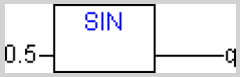
Example in IL:

```
LD  0.5
SIN
ST  q  (* Result is 0.479426 *)
```

Example in ST:

```
q:=SIN(0.5);
```

Example in FBD:



COS

IEC operator *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the cosine of a number. The result is calculated in radians.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD  0.5
COS
ST  q  (* Result is 0.877583 *)
```

Example in ST:

```
q:=COS(0.5);
```

Example in FBD:

TAN

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the tangent of a number. The value is calculated in radians.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD  0.5
TAN
ST  q  (* Result is 0.546302 *)
```

Example in ST:

```
q:=TAN(0.5);
```

Example in FBD:

ASIN

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the arc sine (inverse function of sine) of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD  0.5
ASIN
ST  q  (* Result is 0.523599 *)
```

Example in ST:

```
q:=ASIN(0.5);
```

Example in FBD:

ACOS

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the arc cosine (inverse function of cosine) of a number. The value is calculated in radians.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

Example in IL:

```
LD 0.5
ACOS
ST q  (* Result is 1.0472 *)
```

Example in ST:

```
q:=ACOS(0.5);
```

Example in FBD:

ATAN

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Returns the arc tangent (inverse function of tangent) of a number.

IN can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT. The result OUT is calculated in radians and must be type REAL.

Example in IL:

```
LD  0.5
ATAN
ST  q  (* Result is 0.463648 *)
```

Example in ST:

```
q:=ATAN(0.5);
```

Example in FBD:

EXPT

IEC operator ↳ *Chapter 1.4.1.6.1 “Overview” on page 407*: Exponentiation of a variable with another variable:

IN1 and IN2 can be type BYTE, WORD, DWORD, INT, DINT, REAL, SINT, USINT, UINT, UDINT, OUT must be type REAL.

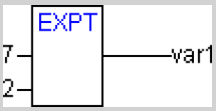
Example in IL:

```
LD  7
EXPT 2
ST  Var1 (* Result is 49 *)
```

Example in ST:

```
var1 := EXPT(7,2);
```

Example in FBD:



1.4.1.6.11 Initialization operator

INI operator

The INI operator can be used to initialize retain variables which are provided by a function block instance used in the POU.

The operator must be assigned to a boolean variable.

Syntax: <bool-Variable> := INI(<FB-instance, TRUE|FALSE)

If the second parameter of the operator is set to TRUE, all retain variables defined in the function block FB will be initialized.

Example in ST: fbinst is the instance of function block fb, in which a retain variable retvar is defined.

Declaration in POU:

```
fbinst:fb;
b:bool;
```

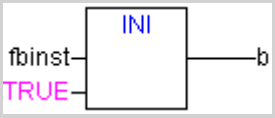
Implementation part:

```
b := INI(fbinst, TRUE);
ivar:=fbinst.retvar  (* => retvar gets initialized *)
```

Example of operator call in IL:

```
LD fbinst
INI TRUE
ST b
```

Example of operator call in FUP:



1.4.1.7 Operands

1.4.1.7.1 Overview

In constants, variables, addresses and possibly function calls can appear as operands.

1.4.1.7.2 Constants

BOOL constants

BOOL constants are the logical values TRUE and FALSE.

TIME constants

TIME constants can be declared. These are generally used to operate the timer in the standard library. A TIME constant is always made up of an initial "t" or "T" (or "time" or "TIME" spelled out) and a number sign "#".

This is followed by the actual time declaration which can include days (identified by "d"), hours (identified by "h"), minutes (identified by "m"), seconds (identified by "s") and milliseconds (identified by "ms").

Please note that the time entries must be given in this order according to length (d before h before m before s before m before ms) but you are not required to include all time increments.

Examples of correct TIME constants in a ST assignment:	TIME1 := T#14ms; TIME1 := T#100S12ms; (*The highest component may be allowed to exceed its limit*) TIME1 := t#12h34m15s; The following would be incorrect: TIME1 := t#5m68s; (*limit exceeded in a lower component*) TIME1 := 15ms; (*T# is missing*) TIME1 := t#4ms13d; (*Incorrect order of entries*)

DATE constants

These constants can be used to enter dates. A DATE constant is declared beginning with a "d", "D", "DATE" or "date" followed by "#". You can then enter any date with format Year-Month-Day.

Examples:	DATE#1996-05-06 d#1972-03-29

TIME_OF_DAY constants

Use this type of constant to store times of the day. A TIME_OF_DAY declaration begins with "tod#", "TOD#", "TIME_OF_DAY#" or "time_of_day#" followed by a time with the format: Hour:Minute:Second.

You can enter seconds as real numbers or you can enter fractions of a second.

Examples:	TIME_OF_DAY#15:36:30.123 tod#00:00:00

DATE_AND_TIME constants

Date constants and the time of day can also be combined to form so-called DATE_AND_TIME constants. DATE_AND_TIME constants begin with "dt#", "DT#", "DATE_AND_TIME#" or "date_and_time#".

Place a hyphen after the date followed by the time.

Examples:	DATE_AND_TIME#1996-05-06-15:36:30 dt#1972-03-29-00:00:00

Number constants

Number values can appear as binary numbers, octal numbers, decimal numbers and hexadecimal numbers. If an integer value is not a decimal number, you must write its base followed by the number sign (#) in front of the integer constant. The values for the numbers 10-15 in hexadecimal numbers will be represented as always by the letters A-F.

You may include the underscore character within the number.

Examples	14 (decimal number) 2#1001_0011 (dual number) 8#67 (octal number) 16#A (hexadecimal number)

These number values can be from the variable types BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, UDINT, REAL or LREAL. Implicit conversions from "larger" to "smaller" variable types are not permitted. This means that a DINT variable cannot simply be used as an INT variable. You must use the type conversion.

REAL/LREAL constants

REAL / LREAL constants can be given as decimal fractions and represented exponentially ↳ *Chapter 1.4.1.8.1.4 “REAL / LREAL” on page 443*. Use the standard American format with the decimal point to do this.

Example	7.4 instead of 7,4 1.64e+009 instead of 1,64e+009

STRING constants

A string is a sequence of characters. STRING constants are preceded and followed by single quotation marks. You may also enter blank spaces and special characters (umlauts for instance). They will be treated just like all other characters.

In character sequences, the combination of the dollar sign ($) followed by two hexadecimal numbers is interpreted as a hexadecimal representation of the eight bit character code. In addition, the combination of two characters that begin with the dollar sign are interpreted as shown below when they appear in a character sequence:

$$	Dollar signs
$'	Single quotation mark
$L or $l	Line feed
$N or $n	New line
$P or $p	Page feed
$R or $r	Line break
$T or $t	Tab

Examples	'w1Wüß?' ' Abby and Craig ' ':-)'

Typed literals

Basically, in using IEC constants, the smallest possible data type will be used. If another data type must be used, this can be achieved with the help of typed literals without the necessity of explicitly declaring the constants. For this, the constant will be provided with a prefix which determines the type.

This is written as follows: <Type>#<Literal>

<Type> specifies the desired data type; possible entries are: BOOL, SINT, USINT, BYTE, INT, UINT, WORD, DINT, UDINT, DWORD, REAL, LREAL. The type must be written in uppercase letters.

<Literal> specifies the constant. The data entered must fit within the data type specified under <Type>.

Example	var1:=DINT#34;

If the constant can not be converted to the target type without data loss, an error message is issued:

Typed literals can be used wherever normal constants can be used.

1.4.1.7.3 Variables

Overview

Variables can be declared either locally in the declaration part of a POU or in a global variable list.

In a project you can define a local variable which has the same name like a global variable. In this case within a POU the locally defined variable will be used. It is not allowed however to name two global variables identically. For example you will get a compiler error, if you have defined a variable "var1" in the PLC Configuration as well as in a global variables list.

The variable identifier may not contain any blank spaces or special characters, may not be declared more than once and cannot be the same as any of the keywords. Capitalization is not recognized which means that VAR1, Var1, and var1 are all the same variable. The underscore character is recognized in identifiers (e.g., "A_BCD" and "AB_CD" are considered two different identifiers). An identifier may not have more than one underscore character in a row. The length of the identifier, as well as the meaningful part of it, are unlimited.

Variables can be used anywhere the declared type allows for them.

You can access available variables through the input assistant.

System flags

System flags are implicitly declared variables that are different on each specific PLC. To find out which system flags are available in your system, use the command 'Insert' 'Operand'. An input assistant dialog box pops up, select the category 'System Variable'.

Accessing variables for arrays, structures and POUs

Two-dimensional array components can be accessed using the following syntax:

```
<Fieldname>[Index1, Index2]
```

Structure variables can be accessed using the following syntax:

```
<Structurename>.<Variablenname>
```

Function block and program variables can be accessed using the following syntax:

```
<Functionblockname>.<Variablename>
```

Addressing bits in variables

In integer variables individual bits can be accessed. For this, the index of the bit to be addressed is appended to the variable, separated by a dot. The bit-index can be given by any constant. Indexing is 0-based.

Bitaccess in Direct variables is not allowed Chapter 1.4.2.4.4 “The library SysLibDirect.lib” on page 570.

Example

```
a : INT;
b : BOOL;
...
a.2 := b;
```

The third bit of the variable a will be set to the value of the variable b.

If the index is greater than the bit width of the variable, the following error message is issued: Index '<n>' outside the valid range for variable '<var>'!

Bit addressing is possible with the following variable types: SINT, INT, DINT, USINT, UINT, UDINT, BYTE, WORD, DWORD.

If the variable type does not allow it, the following error message is issued: "Invalid data type '<type>' for direct indexing"

A bit access must not be assigned to a VAR_IN_OUT variable!

Bitaccess via a global constant:

If you have declared a global constant, which defines the bit-index, you can use this constant for a bitaccess.

The project option 'Replace constants' (category Build) must be activated.

See in the following examples for such a bitaccess on a variable resp. a structure variable:

Declaration in global variables list for both examples:

Variable enable defines which bit should be accessed:

```
VAR_GLOBAL CONSTANT
enable:int:=2;
END_VAR
```

Example 1, Bitaccess on an integer variable:

Declaration in POU:

```
VAR
xxx:int;
END_VAR
```

Bitaccess:

```
xxx.enable:=true;
```

The third bit in variable xxx will be set TRUE.

Example 2, Bitaccess on an integer structure component:

Declaration of structure stru1:

```
TYPE stru1 :
STRUCT
bvar:BOOL;
rvar:REAL;
wvar:WORD;
{bitaccess enable 42 'Start drive'}
END_STRUCT
END_TYPE
```

Declaration in POU:

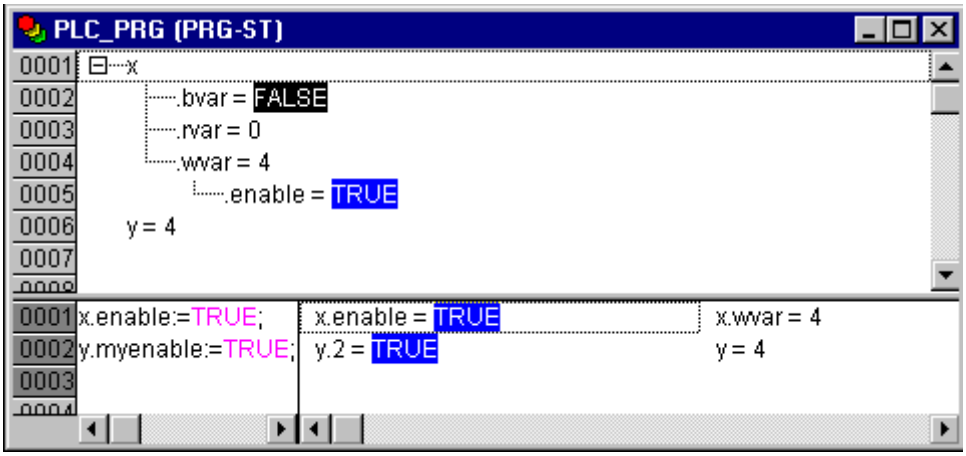
```
VAR
x:stru1;
END_VAR
```

Bitaccess:

```
x.enable:=true;
```

This will set TRUE the 42. bit in variable x. Since bvar has 8 bits and rvar has 32 bits, the bitaccess will be done on the second bit of variable wvar, which as a result will get value 4.

If a variable, which does a bitaccess on a structure variable with the aid of a global constant, should be displayed correctly in the input assistant, at monitoring in the declaration window and in the intellisense function. Please use pragma {bitaccess} which is shown in the example ↬ Chapter 1.4.1.3.10.2 “Pragma instructions for initialization, monitoring, creation of symbols, bitaccess, linking” on page 310. Then in addition you get displayed the global constant beyond the respective structure variable during monitoring in the declaration window:



1.4.1.7.4 Addresses

Addresses, note

> *Online Change might change the contents on addresses.*
>
> *Please regard this when using pointers on addresses. See Pointer and Address ↳ Chapter 1.4.1.8.2.3 “Pointer” on page 447 ↳ Chapter 1.4.1.7.4.2 “Address” on page 441.*

Address

The direct display of individual memory locations is done through the use of special character sequences. These sequences are a concatenation of the percent sign "%", a range prefix, a prefix for the size and one or more natural numbers separated by blank spaces.

The following range prefixes are supported:

I	Input
Q	Output
M	Memory location

The following size prefixes are supported:

Examples:

%QX7.5 and %Q7.5	Output bit 7.5
%IW215	Input word 215
%QB7	Output byte 7
%MD48	Double word in memory position 48 in the memory location.
%IW2.5.7.1	depending on the PLC Configuration
ivar AT %IW0 : WORD;	Example of a variable declaration including an address assignment

The current PLC Configuration for the program determines whether or not an address is valid.

Boolean values will be allocated bytewise, if no explicit single-bit address is specified.

Example: A change in the value of varbool1 AT %QW0 affects the range from QX0.0 to QX0.7.

Online Change might change the contents on addresses.

Please regard this when using Pointer on addresses *Chapter 1.4.1.8.2.3 “Pointer” on page 447.*

Memory location

You can use any supported size to access the memory location.

For example, the address %MD48 would address bytes numbers 192, 193, 194, and 195 in the memory location area (48 * 4 = 192). The number of the first byte is 0.

You can access words, bytes and even bits in the same way: the address %MX5.0 allows you to access the first bit in the fifth word (Bits are generally saved wordwise).

Online Change might change the contents on addresses.

Please regard this when using pointers on addresses.

1.4.1.7.5 Functions

In ST a function call can also appear as an operand.

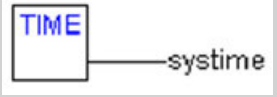
Example:	Result := Fct(7) + 3;

TIME()-Function

This function returns the time (based on milliseconds) which has been passed since the system was started. The data type is TIME.

Example in IL:	TIME ST systime (* Result e.g.: T#35m11s342ms *)

Example in ST:	systime:=TIME();

Example in FUP:	

1.4.1.8 Data types

1.4.1.8.1 Standard data types

Data types

You can use standard data types and user-defined data types when programming. Each identifier is assigned to a data type which dictates how much memory space will be reserved and what type of values it stores.

BOOL

BOOL type variables may be given the values TRUE and FALSE. 8 bits of memory space will be reserved.

See also: BOOL constants ↳ *Chapter 1.4.1.7.2.1 “BOOL constants” on page 435*

Integer data types

BYTE, WORD, DWORD, SINT, USINT, INT, UINT, DINT, and UDINT are all integer data types.

Each of the different number types covers a different range of values. The following range limitations apply to the integer data types:

Type	Lower limit	Upper limit	Memory space
BYTE	0	255	8 bit
WORD	0	65535	16 bit
DWORD	0	4294967295	32 bit
SINT	-128	127	8 bit
USINT	0	255	8 bit
INT	-32768	32767	16 bit
UINT	0	65535	16 bit
DINT	-2147483648	2147483647	32 bit
UDINT	0	4294967295	32 bit

As a result when larger types are converted to smaller types, information may be lost.

See also: Number constants ↳ *Chapter 1.4.1.7.2.6 “Number constants” on page 436*

REAL / LREAL

REAL and LREAL are so-called floating-point types. They are required to represent rational numbers. 32 bits of memory space is reserved for REAL and 64 bits for LREAL.

Valid values for REAL: 1.175494351e-38 to 3.402823466e+38

Valid values for LREAL: 2.2250738585072014e-308 to 1.7976931348623158e+308

See also: REAL/LREAL constants ↳ *Chapter 1.4.1.7.2.7 “REAL/LREAL constants” on page 437*

STRING

A STRING type variable can contain any string of characters. The size entry in the declaration determines how much memory space should be reserved for the variable. It refers to the number of characters in the string and can be placed in parentheses or square brackets. If no size specification is given, the default size of 80 characters will be used.

The string length basically is not limited, but string functions only can process strings of 1 - 255 characters!

Example of a String Declaration with 35 characters:

```
str:STRING(35):='This is a String';
```

See also: STRING constants ↬ *Chapter 1.4.1.7.2.8 “STRING constants” on page 437*

Time data types

The data types TIME, TIME_OF_DAY (abb. TOD), DATE and DATE_AND_TIME (abb. DT) are handled internally like DWORD.

Time is given in milliseconds in TIME and TOD, time in TOD begins at 12:00 A.M.

Time is given in seconds in DATE and DT beginning with January 1, 1970 at 12:00 A.M.

See in the following the time data formats used to assign values for time constants:

TIME constants: always made up of an initial "t" or "T" (or "time" or "TIME" spelled out) and a number sign "#".

This is followed by the actual time declaration which can include days (identified by "d"), hours (identified by "h"), minutes (identified by "m"), seconds (identified by "s") and milliseconds (identified by "ms"). Please note that the time entries must be given in this order according to length (d before h before m before s before m before ms) but you are not required to include all time increments.

Maximum value: 49d17h2m47s295ms (4194967295 ms)

Examples of correct TIME constants in a ST assignment:

TIME1 := T#14ms;	
TIME1 := T#100S12ms;	(*The highest component may be allowed to exceed its limit*)
TIME1 := t#12h34m15s;	

the following would be incorrect:

TIME1 := t#5m68s;	(*limit exceeded in a lower component*)
TIME1 := 15ms;	(*T# is missing*)
TIME1 := t#4ms13d;	(*Incorrect order of entries*)

DATE constants: A date constant begins with a "d", "D", "DATE" or "date" followed by "#". You can then enter any date with format Year-Month-Day. Possible values: 1970-00-00 to 2106-02-06.

Examples:

```
DATE#1996-05-06
d#1972-03-29
```

TIME_OF_DAY constants, for storing times of the day: begin with "tod#", "TOD#", "TIME_OF_DAY#" or "time_of_day#" followed by a time with the format: Hour:Minute:Second. Seconds can be entered as real numbers or you can enter fractions of a second. Possible values: 00:00:00 bis 23:59:59.999.

Examples:

```
TIME_OF_DAY#15:36:30.123
tod#00:00:00
```

DATE_AND_TIME constants, combination of date and the time of day: begin with "dt#", "DT#", "DATE_AND_TIME#" or "date_and_time#". Place a hyphen after the date followed by the time. Possible values: 1970-00-00-00:00:00 to 2106-02-06-06:28:15.

Examples:

```
DATE_AND_TIME#1996-05-06-15:36:30
dt#1972-03-29-00:00:00
```

1.4.1.8.2 Defined data types

ARRAY

One-, two-, and three-dimensional fields (arrays) are supported as elementary data types. Arrays can be defined both in the declaration part of a POU and in the global variable lists. Maximum 9 dimensions may result from nesting of arrays ("ARRAY[0..2] OF ARRAY[0..3] OF …").

Syntax:

```
<Field_Name>:ARRAY [<ll1>..<ul1>,<ll2>..<ul2>] OF <elem. Type>.
```

ll1, ll2, ll3 identify the lower limit of the field range; ul1, ul2 and ul3 identify the upper limit. The limit values must be integers and must match the DINT range of values.

Example:

```
Card_game: ARRAY [1..13, 1..4] OF INT;
```

Initializing arrays:

Example for complete initialization of an array:

```
arr1 : ARRAY [1..5] OF INT := 1,2,3,4,5;
arr2 : ARRAY [1..2,3..4] OF INT := 1,3(7); (* short for 1,7,7,7 *)
arr3 : ARRAY [1..2,2..3,3..4] OF INT := 2(0),4(4),2,3; (* short for
0,0,4,4,4,4,2,3 *)
```

Example of the initialization of an array of a structure:

```
TYPE STRUCT1
STRUCT
p1:int;
p2:int;
p3:dword;
END_STRUCT
ARRAY[1..3] OF STRUCT1:= (p1:=1,p2:=10,p3:=4723),
(p1:=2,p2:=0,p3:=299),(p1:=14,p2:=5,p3:=112);
```

Example of the partial initialization of an array:

```
arr1 : ARRAY [1..10] OF INT := 1,2;
```

Elements to which no value is pre-assigned are initialized with the default initial value of the basic type. In the example above, the elements anarray[6] to anarray[10] are therefore initialized with 0.

Accessing array components: Array components are accessed in a two-dimensional array using the following syntax:

```
<Field_Name>[Index1,Index2]
```

Example:

```
Card_game [9,2]
```

If you define a function in your project with the name CheckBounds, you can use it to check for range overflows in your project (see chapter 'How is a project structured?' ↳ *Chapter 1.4.1.1.1 “How is a project structured?” on page 145)*
↳ *Chapter 1.4.1.8.2.2 “Function CheckBounds” on page 446.*

Function CheckBounds

If you define a function in your project with the name CheckBounds, you can automatically check for out-of-range errors in arrays ↳ *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*. The name of the function is fixed and can only have this designation.

Example for the function CheckBounds:

```
FUNCTION CheckBounds : DINT
VAR_INPUT
 index, lower, upper: DINT;
END_VAR
IF  index < lower THEN
 CheckBounds := lower;
ELSIF  index > upper THEN
 CheckBounds := upper;
ELSE  CheckBounds := index;
END_IF
```

The following sample program for testing the CheckBounds function exceeds the bounds of a defined array. The CheckBounds function allows the value TRUE to be assigned, not to location A[10], but to the still valid range boundary A[7] above it. With the CheckBounds function, references outside of array boundaries can thus be corrected.

Test program for the function CheckBounds:

```
PROGRAM PLC_PRG
VAR
 a: ARRAY[0..7] OF BOOL;
 b: INT:=10;
END_VAR
a[b]:=TRUE;
```

NOTICE!
The CheckBounds-function provided by the Check.Lib library is just a sample solution! Before using that library module check whether the function is working in your sense, or implement an appropriate function directly as a POU in your project.

Pointer

Variable or function block addresses ↳ *Addresses* are saved in pointers while a program is running.

Pointer declarations have the following syntax:

```
<Identifier>: POINTER TO <Datatype/Functionblock>;
```

A pointer can point to any data type or function block even to user-defined types.

The function of the Address Operator ADR is to assign the address of a variable or function block to the pointer.

A pointer can be dereferenced by adding the content operator "^" after the pointer identifier.

A pointer is counted up bytewise ! You can get it counted up like it is usual in the C-Compiler by using the instruction p=p+SIZEOF(p^);.

Example:

```
pt:POINTER TO INT;
var_int1:INT := 5;
var_int2:INT;
pt := ADR(var_int1);
var_int2:= pt^; (* var_int2 is now 5 *)
```

Checking pointer accesses at run time

For checking pointer accesses during run time you can create check functions, which will be called automatically before each access on the address of a pointer. For this purpose the respective function must be available in the project, directly or via a library. The following functions are supported:

- Function CheckPointer: checks whether the address currently stored at the pointer is within the valid memory range,
- Function CheckPointerAligned: implicates the functionality of CheckPointer and additionally checks the memory alignment.

The functions must exactly have the mentioned names. They return the address which is used for dereferencing the pointer, thus at best that which has been passed on as the first input parameter (dwAddress in the example shown below).

See in the following example of a CheckPointerAligned function, which input parameters are processed. The parameter names are examples too. A CheckPointer function must look the same, except that there may be no parameter for the granularity of the pointer access:

`FUNCTION CheckPointerAligned : DWORD`	(* The data type of the function (return value) must be the same as used for pointers in the currently set target system; i.e. DWORD for systems using 32-bit pointers, WORD for systems using 16-bit pointers *)
`VAR_INPUT`	
`dwAddress : DWORD;`	(* Target address of the pointer; the data type must be the same as used for pointers in the currently set target system, see above: return value *)
`iSize : DINT;`	(* Size of pointer access; the data type must be integer-compatible and must cover the maximum potential data size stored at the pointer address *)
`iGran : DINT;`	(* ! not to be used in CheckPointer functions ! : Granularity of the access, e.g. "2", if INT is the smallest not-structured datatype used on the given address; the data type must be integer-compatible *)
`bWrite: BOOL;`	(*Access type: Read or Write; TRUE=read access; the data type must be BOOL *)
`END_VAR`	

If there are a CheckPointer function and a CheckPointerAligned function in the project, CheckPointerAligned will be called.

Enumeration

Enumeration is a user-defined data type that is made up of a number of string constants. These constants are referred to as enumeration values.

Enumeration values are recognized in all areas of the project even if they were declared within a POU. It is best to create your enumerations as objects in the Object Organizer under the register card Data types. They begin with the keyword TYPE and end with END_TYPE.

Syntax:

```
TYPE <Identifier>:(<Enum_0> ,<Enum_1>, ...,<Enum_n>);
END_TYPE
```

A variable of the type <Identifier> can take on one of the enumeration values and will be initialized with the first one. These values are compatible with whole numbers which means that you can perform operations with them just as you would with INT. You can assign a number x to the variable. If the enumeration values are not initialized, counting will begin with 0. When initializing, make certain the initial values are increasing. The validity of the number will be reviewed at the time it is run.

Example:

```
TYPE TRAFFIC_SIGNAL: (Red, Yellow, Green:=10); (*The initial value
for each of the colors is red 0, yellow 1, green 10 *)
END_TYPE
TRAFFIC_SIGNAL1 : TRAFFIC_SIGNAL;
TRAFFIC_SIGNAL1:=0; (* The value of the traffic signal is red*)
FOR i:= Red TO Green DO
 i := i + 1;
END_FOR;
```

The same enumeration value may not be used twice within an enumeration or within all enumerations used in the same POU.

Example:

```
TRAFFIC_SIGNAL: (red, yellow, green);
COLOR: (blue, white, red);
Error: red may not be used for both TRAFFIC_SIGNAL and COLOR.
```

Structures

Structures are created as objects in the Object Organizer under the register card Data types. They begin with the keywords TYPE and STRUCT and end with END_STRUCT and END_TYPE.

The syntax for structure declarations is as follows:

```
TYPE <Structurename>:
STRUCT
 <Declaration of Variables 1>
 .
 .
 <Declaration of Variables n>
END_STRUCT
END_TYPE
```

<Structurename> is a type that is recognized throughout the project and can be used like a standard data type.

Interlocking structures are allowed. The only restriction is that variables may not be placed at addresses (the AT declaration is not allowed!).

Example for a structure definition named Polygonline:

```
TYPE Polygonline:
STRUCT
 Start:ARRAY [1..2] OF INT;
 Point1:ARRAY [1..2] OF INT;
 Point2:ARRAY [1..2] OF INT;
 Point3:ARRAY [1..2] OF INT;
 Point4:ARRAY [1..2] OF INT;
 End:ARRAY [1..2] OF INT;
END_STRUCT
END_TYPE
```

Example for the initialization of a structure:

```
Poly_1:polygonline := ( Start:=3,3, Point1 =5,2, Point2:=7,3,
Point3:=8,5, Point4:=5,7, End := 3,5);
```

Initializations with variables are not possible. See an example of the initialization of an array of a structure under 'Arrays'.

Access on structure components:

You can gain access to structure components using the following syntax:

```
<Structure_Name>.<Componentname>
```

So for the above mentioned example of the structure 'polygonline' you can access the component 'start' by `Poly_1.Start`.

References

You can use the user-defined reference data type to create an alternative name for a variable, constant or function block.

Create your references as objects in the Object Organizer under the register card Data types. They begin with the keyword TYPE and end with END_TYPE.

Syntax:

```
TYPE <Identifier>: <Assignment term>;
END_TYPE
```

Example:

```
TYPE message:STRING[50];
END_TYPE;
```

Subrange types

A subrange type is a type whose range of values is only a subset of that of the basic type. The declaration can be carried out in the data types register, but a variable can also be directly declared with a subrange type:

Syntax for the declaration in the 'Data types' register:

```
TYPE <Name> : <Inttype> (<ug>..<og>) END_TYPE;
```

<Name>	must be a valid IEC identifier,
<Inttype>	is one of the data types SINT, USINT, INT, UINT, DINT, UDINT, BYTE, WORD, DWORD (LINT, ULINT, LWORD).
<ug>	Is a constant which must be compatible with the basic type and which sets the lower boundary of the range types. The lower boundary itself is included in this range.
<og>	Is a constant that must be compatible with the basic type, and sets the upper boundary of the range types. The upper boundary itself is included in this basic type.

Example:

```
TYPE
 SubInt : INT (-4095..4095);
END_TYPE
```

Direct declaration of a variable with a subrange type:

```
VAR
 i : INT (-4095..4095);
 ui : UINT (0..10000);
END_VAR
```

If a constant is assigned to a subrange type (in the declaration or in the implementation) that does not fall into this range (e.g. 1:=5000), an error message is issued.

In order to check for observance of range boundaries at run time, the functions CheckRangeSigned or CheckRangeUnsigned must be introduced. In these, boundary violations can be captured by the appropriate method and means (e.g. the value can be cut out or an error flag can be set.). They are implicitly called as soon as a variable is written as belonging to a subrange type constructed from either a signed or an unsigned type.

Example:

In the case of a variable belonging to a signed subrange type (like i, above), the function CheckRangeSigned is called; it could be programmed as follows to trim a value to the permissible range:

```
FUNCTION CheckRangeSigned : DINT
VAR_INPUT
 value, lower, upper: DINT;
END_VAR
IF (value < lower) THEN
 CheckRangeSigned := lower;
ELSIF(value > upper) THEN
 CheckRangeSigned := upper;
ELSE
 CheckRangeSigned := value;
END_IF
```

In calling up the function automatically, the function name CheckRangeSigned is obligatory, as is the interface specification: return value and three parameters of type DINT

When called, the function is parameterized as follows:

-
- - value: the value to be assigned to the range type
- - lower: the lower boundary of the range
- - upper: the upper boundary of the range
- - Return value: this is the value that is actually assigned to the range type

An assignment i:=10*y implicitly produces the following in the example:

```
i := CheckRangeSigned(10*y, -4095, 4095);
```

Even if y for example has the value 1000, then i still has only the value 4095 after this assignment.

The same applies to the function CheckRangeUnsigned: function name and interface must be correct.

```
FUNCTION CheckRangeUnsigned : UDINT
VAR_INPUT
 value, lower, upper: UDINT;
END_VAR
```

WARNING!
If neither of the functions CheckRangeSigned or CheckRangeUnsigned is present, no type checking of subrange types occurs during run time! The variable i could then take on any value between 32768 and 32767 at any time!

NOTICE!
If neither of the functions CheckRangeSigned or CheckRangeUnsigned is present like described above, there can result an endless loop if a subrange type is used in a FOR loop. This will happen when the range given for the FOR loop is as big or bigger than the range of the subrange type!

NOTICE!
The CheckRangeSigned-function provided with the Check.Lib library is just a sample solution! Before using the library module check whether the function is working as requested for your project, or implement an appropriate CheckRange-function directly as a POU in the project.

Example:

```
VAR
 ui : UINT (0..10000);
END_VAR

FOR ui:=0 TO 10000 DO
...
END_FOR
```

The FOR loop will never be finished, because ui cannot get bigger than 10000.

Also take care of the definition of the CheckRange functions when you define the incremental value of a FOR loop!

1.4.1.8.3 Overview operators and library elements

The table shown below shows an overview on the operators, which are available in CODESYS or in the libraries Standard.lib and Util.lib. You find there the notation for ST and IL. For IL also the supported modificators are listed.

Take note that for the 'IL operator' column: Only the line in which the operator is used will be displayed. A prerequisite is that the (first) required operand have been successfully loaded in the preceding line (e.g. LD in).

Table 13: The 'Mod. IL' column shows the possible modifiers in IL:

C	The command is only executed if the result of the preceding expression is TRUE.
N	for JMPC, CALC, RETC: The command is only executed if the result of the preceding expression is FALSE.
N	otherwise: negation of the operand (not of the accumulator)
(	Operator enclosed in brackets: only after the closing bracket is reached will the operation preceding the brackets be carried out.

Please obtain a detailed description of usage from the appropriate appendices concerning IEC operators.

Operators in CODESYS:

in ST	in AWL	Mod. AWL	Description
'			String delimiters (e.g. 'string1')
.. [ ]			Size of Array range (e.g. ARRAY[0..3] OF INT)
:			Delimiter between Operand and Type in a declaration (e.g. var1 : INT;)
;			Termination of instruction (e.g. a:=var1;)
^			Dereferenced Pointer (e.g. pointer1^)
	LD var1	N	Load value of var1 in buffer
:=	ST var1	N	Store actual result to var1
	S boolvar		Set boolean operand boolvar exactly then to TRUE, when the actual result is TRUE
	R boolvar		Set boolean operand boolvar exactly then to FALSE, when the actual result is TRUE

in ST	in AWL	Mod. AWL	Description
	JMP label	CN	Jump to label
<Program name>	CAL prog1	CN	Call program prog1
<Instance name>	CAL inst1	CN	Call function block instance inst1
<Fctname>(vx, vy,..)	<Fctname> vx, vy	CN	Call function fctname and transmit variables vx, vy
RETURN	RET	CN	Leave POU and go back to caller
	(		The value following the bracket is handled as operand, the operation before the bracket is not executed before the expression in the brackets.

in ST	in AWL	Mod. AWL	Description
	)		Now execute the operation which has been set back
AND	AND	N,(	Bitwise AND
OR	OR	N,(	Bitwise OR
XOR	XOR	N,(	Bitwise exclusive OR
NOT	NOT		Bitweise NOT
+	ADD	(	Addition
-	SUB	(	Subtraction
*	MUL	(	Multiplication
/	DIV	(	Division
>	GT	(	Greater than
>=	GE	(	Greater or equal
=	EQ	(	Equal
<>	NE	(	Not equal
<=	LE	(	Less or equal
<	LT	(	Less than
MOD(in)	MOD		Modulo Division
INDEXOF(in)	INDEXOF		Internal index of POU in1; [INT]
SIZEOF(in)	SIZEOF		Number of bytes required for the given data type of in
SHL(K,in)	SHL		Bitwise left-shift of operator in by K
SHR(K,in)	SHR		Bitwise right-shift of operator in by K
ROL(K,in)	ROL		Bitwise rotation to the left of operator in by K
ROR(K,in)	ROR		Bitwise rotation to the right of operator in by K
SEL(G,in0,in1)	SEL		Binary selection between 2 operands in0 (G is FALSE) and in1 (G is TRUE)
MAX(in0,in1)	MAX		Returns the greater of 2 values
MIN(in0,in1)	MIN		Returns the lesser of 2 values in0 and in1
LIMIT(MIN,in,Max)	LIMIT		Limits the range of values (in is set back to MIN or MAX in case of exceeding the range)

in ST	in AWL	Mod. AWL	Description
MUX(K,in0,...in_n)	MUX		Selecti the Kth value out of a group of values (in0 to In_n)
ADR(in)	ADR		Address of the operand in [DWORD]
ADRINST()	ADRINST()		Address of the function block instance from which you are calling that operator.
BITADR(in)	BITADR		Bitoffset of the operand in [DWORD]
BOOL_TO_<type>(in)	BOOL_TO_<type>		Type conversion of the boolean operand
<type>_TO_BOOL(in)	<type>_TO_BOOL		Type conversion to BOOL
INT_TO_<type>(in)	INT_TO_<type>		Type conversion of an INT Operand to another elementary type
REAL_TO_<type>(in)	REAL_TO_<type>		Type conversion of an REAL operand to another elementary type
LREAL_TO_<type>(in)	LREAL_TO_<type>		Type conversion of a LREAL operand to another elementary type
TIME_TO_<type>(in)	TIME_TO_<type>		Type conversion of a TIME operand to another elementary type
TOD_TO_<type>(in)	TOD_TO__<type>		Type conversion of a TOD operand to another elementary type
DATE_TO_<type>(in)	DATE_TO_<type>		Type conversion of a DATE operand to another elementary type
DT_TO_<type>(in)	DT_TO_<type>		Type conversion of a DT operand to another elementary type
STRING_TO_<type>(in)	STRING_TO_<type>		Type conversion of a string operand to another elementary type, in must contain valid value of desired type
TRUNC(in)	TRUNC		Conversion from REAL to INT
ABS(in)	ABS		Absolute value of operand in

in ST	in AWL	Mod. AWL	Description
SQRT(in)	SQRT		Square root of operand in
LN(in)	LN		Natural logarithm of operand in
LOG(in)	LOG		Logarithm of operand in, base 10
EXP(in)	EXP		Exponential function of operand in
SIN(in)	SIN		Sine of operand in
COS(in)	COS		Cosine of operand in
TAN(in)	TAN		Tangent of operand in
ASIN(in)	ASIN		Arc sine of operand in
ACOS(in)	ACOS		Arc cosine of operand in
ATAN(in)	ATAN		Arc tangent of operand in
EXPT(in,expt)	EXPT expt		Exponentation of operand in with expt

Elements of the Standard.lib

in ST	in AWL	Description
LEN(in)	LEN	String length of operand in
LEFT(str,size)	LEFT	Left inital string of given size of string str
RIGHT(str,size)	RIGHT	Right initial string of given size of string str
MID(str,size,pos)	MID	Partial string of str of given size at position pos
CONCAT('str1','str2')	CONCAT 'str2'	Concatenation of two subse-quent strings
INSERT('str1','str2',pos)	INSERT 'str2',p	Insert string str1 in String str2 at position pos
DELETE('str1',len,pos)	DELETE len,pos	Delete partial string (length len), start at position pos of str1
REPLACE('str1','str2',len,pos)	REPLACE 'str2',len,pos	Replace partial string of lenght len by str2, start at position pos of str1
FIND('str1','str2')	FIND 'str2'	Search for partial string str2 in str1
SR	SR	Bistable FB is set dominant
RS	RS	Bistable FB is set back
SEMA	SEMA	FB: Software Semaphor (interruptable)
R_TRIG	R_TRIG	FB: rising edge is detected
F_TRIG	F_TRIG	FB: falling edge is detected

in ST	in AWL	Description
CTU	CTU	FB: Counts upv
CTD	CTD	FB: Counts down
CTUD	CTUD	FB: Counts up and down
TP	TP	FB: trigger
TON	TON	FB: Timer On-Delay
TOF	TOF	FB: Timer Off-Delay
RTC	RTC	FB: Real-time clock

Elements of the Util.lib

Element	Description
BCD_TO_INT	Conversion of a Byte: BCD to INT format
INT_TO_BCD	Converstion of a Byte: INT to BCD format
EXTRACT(in,n)	The n-th bit of DWORD in is returned in BOOL
PACK	Up to 8 bits are packed into a byte
PUTBIT	A bit of a DWORD is set to a certain value
UNPACK	A byte is returned as single bits
DERIVATIVE	Local derivation
INTEGRAL	Integral
LIN_TRAFO	Transformation of REAL values
STATISTICS_INT	Min., Max., average values in INT format
STATISTICS_REAL	Min., Max., average in REAL format
VARIANCE	Variance
PD	PD controller
PID	PID controller
BLINK	Pulsating signal
FREQ_MEASURE	Measuring frequency of boolean input signal
GEN	Periodic functions
CHARCURVE	Linear functions
RAMP_INT	Limiting ascendance of descendance of the function beeing fed (INT)
RAMP_REAL	Limiting ascendance of descendance of the function beeing fed (REAL)
HYSTERESIS	Hysteresis
LIMITALARM	Watches whether input value exceeds limits of a defined range

1.4.1.9 Utilities

1.4.1.9.1 Command line-/Command file

Command line commands

When CODESYS is started you can add commands in the command line which will be asserted during execution of the program. These commands start with a "/". Capitalization / use of small letters is not regarded. The commands will be executed sequentially from the left to the right.

/online	Immediately after start, CODESYS tries to go online with the current project.
/run	After login, CODESYS starts the application program. Only valid in combination with /online.
/batch	CODESYS starts without user interface and returns the error code of the first error or the return value of the first command which is terminated with a warning. CODESYS will terminate immediately after the command file has been processed. The processing of the command file will be aborted as soon as the first command is processed with an error. Warnings do not terminate the processing. If neither errors nor warnings occur, the return value is S_OK.
/show ... /show hide /show icon /show max /show normal	Settings for the frame window of CODESYS can be made. The window will not be displayed, it also will not be represented in the task menu. The window will be minimized in display. The window will be maximized in display. The window will be displayed in the same status as it was during the last closing.
/out <outfile>	All messages are displayed in the message window and additionally are written in the file <outfile>.
/noinfo	No splash screen at start of CODESYS
/userlevel <group>	Definition of the user group (e.g. "/userlevel 0" for user group 0)
/password <password>	Direct input of the user group password (e.g. "/password abc")
/openfromplc	The project which is currently available on the connected target system, will be loaded.
/visudownload	If CODESYS HMI is started with a project, which does not match with the project currently available on the target system, a download will be offered. (Dialog, to be closed with YES or NO).
/notargetchange	A change of the target system only can be done via a command file ↬ *Table 31 “Select target system” on page 464*.
/targetfile <file>.trg	A target description file (*.trg) can be specified. So the already installed targets will not be regarded when CODESYS is started. The commands 'File' 'New', File' 'New from template...', 'File' 'Open...' and the list of recently opened projects will not be available. Additionally the selection list in the target settings configuration dialog will not usable.

/targetfilenosaveas <file>.trg	In addition to the effects described for the "/targetfile" option, see above, also the menu command 'File' 'Save as...' will not be available.
/cmd <cmdfile>	After starting the commands of the <cmdfile> get executed *Chapter 1.4.1.9.1.2 “Command file (cmdfile) commands” on page 459*.

Regard the following syntax for a command line:

"<Path of codesys.exe-file>" "<Path of the project>" /<command1> /<command2> ....

Example for a command line: The project ampel.pro gets opened, but no window opens. The commands included in the command file command.cmd will be executed.

```
"D:\dir1\codesys" "C:\projects\ampel.pro" /show hide /cmd command.cmd
```

Command file (cmdfile) commands

See the following tables for a list of commands which can be used in a command file (<cmdfile>). The command file you then can call by a command line *Chapter 1.4.1.9.1.1 “Command line commands” on page 457*. There is no case sensitivity. The command line will be displayed as a message in the message window and can be given out in a message file (see below) except the command is prefixed by a "@".

All signs after a semicolon (;) will be ignored (comment). Parameters containing blanks must be embraced by quotation marks. Umlauts only may be used if the command file is created in ANSI code. Keywords can be used in the command parameters. A list of the keywords you find subsequent to the following tables of command descriptions.

Table 14: Commands for controlling the subsequent commands

Command	Description
onerror continue	The subsequent commands will be executed even if an error occurs.
onerror break	The subsequent commands will not be executed any more if an error has been detected.

Table 15: Commands of the online menu

Command	Description
online login	Login with the loaded project ('Online Login')
online logout	Logout ('Online' 'Logout')
online run	Start of the application program ('Online' 'Run')
online stop	Stop application program ('Online' 'Stop')
online bootproject	Creation of a boot project. This command can be applied in offline and online mode! (See also description of command 'Online' 'Create boot project' *Chapter 1.4.1.2.6.25 “'Online' 'Create boot project'” on page 291*!)
online sourcecodedownload	Download of the source code of the project to the PLC ('Online' 'Sourcecode download')

Command	Description
online sim	Switch on of simulation mode 'Online' 'Simulation')
online sim off	Switch off of simulation mode ('Online' 'Simulation')

Table 16: Commands of the file menu

Command	Description
file new	A new project is created ('File' 'New')
file open <projectfile> possible additions:	The project <projectfile> will be loaded ('File' 'Open')
/readpwd:<readpassword>	The password for read access is given here so that no dialog asking for the password will appear when the read-protected project is opened.
/writepwd:<writepassword>	The password for full access is given here, so that no dialog asking for the password will appear when the project is openend.
file close	The current project will be closed ('File' 'Close')
file save	The current project will be stored ('File' 'Save')
file saveas <projectfile> optionally add: <type><version>	The current project will be saved with the file name <projectfile> ('File' 'Save as') Default: Project will be saved as <projectfile>.pro under the current version of CODESYS. If you want to save the project as an internal or external library or as project for an older version of CODESYS, add the respective command: Possible entries for <type>: "internallib" Save as internal library: "externallib" Save as external library: "pro" Save as project for older version: valid entries for <Version>: 15, 20, 21, 22 (product versions 1.5, 2.0, 2.1, 2.2) Example: "file save as lib_xy internallib22": The project "project xy.pro", which is created in the current version of CODESYS will be saved as "lib_xy.lib" for V2.2.
file saveas <projectfile>	The current project will be saved with the file name <projectfile> ('File' 'Save as')
file printersetup <filename>.dfr optionally add: pageperobject or pagepersubject	Define a document frame file ('File' Printer setup') and optionally define one of the print options 'New page per object' or 'New page per subobject'; these settings affect the printing of the document (project documentation, see below)
file archive <filename>.zip	The project will be archived in a zip-file with the given file name ('File' Save/Mail Archive')
file quit	CODESYS will be closed ('File' 'Exit')

Table 17: Commands of the project menu

Command	Description
project build	The project that is loaded will be incrementally compiled ('Project' 'Build')
project rebuild or project compile	The project that is loaded will be compiled in full ('Project' 'Rebuild')
project clean	Compilation information and Online Change information in the current project will be deleted ('Project' 'Clean Project')
project check	The project that is loaded will be checked ('Project' 'Check all')
project compile	The current project will be compiled by "Rebuild all" ('Project' 'Rebuild all')
project check	The current project will be checked ('Project' 'Check')
project build	The current project will be built ('Projekt' 'Build')
project import <file1> ... <fileN>	The files <file1> ... <fileN> get imported into the current project ('Project' 'Import'). Regard: Wildcards can be used, e.g. "project import C:\projects*.exp" will import all files with extension *.exp found in directory C:\projects.
project export <expfile>	The current project will be exported in the file <expfile> ('Project' 'Export')
project expmul	Each object of the current project will be exported in an own file, which gets the name of the object.
project documentation	The entire project will be printed on the default printer ('Project' 'Documentation', see also above "file printersetup")

Table 18: Commands for the control of the message file

Command	Description
out open <msgfile>	The file <msgfile> opens as message file. New messages will be appended
out close	The currently shown message file will be closed.
out clear	All messages of the currently opened message file will be deleted.

Table 19: Commands for the control of messages

Command	Description
echo on	The command lines will be displayed as messages.
echo off	The command lines will not be displayed as messages.
echo <text>	<text> will be displayed in the message window.

Table 20: Commands for the control of replace of objects respectively for the control of files for import, export, copy

Command	Description
replace yesall	Replace all (any 'query on' command will be ignored; no dialogs will open)
replace noall	Replace none (any 'query on' command will be ignored; no dialogs will open)
replace query	If a 'query on' command is set, then a dialog will open regarding the replacing of the objects even if there is a 'replace yesall' or 'replace noall' command

Table 21: Commands for the control of the default parameters of the dialogs

Command	Description
query on	Dialogs are displayed and need user input
query off ok	All dialogs respond as if the user had clicked on the 'OK' button
query off no	All dialogs respond as if the user had clicked on the 'No' button
query off cancel	All dialogs respond as if the user had clicked on the 'Cancel' button

Table 22: Command for calling command files as subprograms

Command	Description
call <parameter1> ... <parameter10>	Command files will be called as subprograms. Up to 10 parameters may be passed. In the file that is called, the parameters can be accessed with $0 - $9.
call <parameter1> ... <parameter10>	Command files are called as subroutines. Up to ten parameters can be consigned. In the subroutine called you can access the parameters using $0 - $9.

Table 23: Setting of used directories

Command	Description
dir lib <libdir>	Sets <libdir> as the library directory
dir compile <compiledir>	Sets <compiledir> as the directory for the compilation files
dir config <configdir>	Sets <configdir> as the directory for the configuration files
dir upload <uploaddir>	Sets <uploaddir> as the directory for the upload files

Remarks for settings of used directories:

Project options dialog, category 'Directories', subcategory 'General'

If several directories are defined with one of the following commands, these must be separated by a semicolon + emptyspace and the whole row of directories must be embraced by double quotation marks.

Example, two paths: `dir lib "D:\codesys\Libraries\Standard; D:\codesys\Libraries\NetVar"`

Table 24: Delaying processing of the CMDFILE

Command	Description
delay 5000	Waits 5 seconds

Table 25: Controlling the Watch and Receipt Manager

Command	Description
watchlist load <file>	Loads the Watchlist saved as <file> and opens the corresponding window ('Extras' 'Load Watchlist')
watchlist save <file>	Saves the current Watchlist as <file> ('Extras' 'Save Watchlist')
watchlist set <text>	The watchlist is set active (corresponds to selecting a watchlist in the left part of the Watch and Receipt Manager)
watchlist read	Updates the values of the Watch variables ('Extras' 'Read receipt')
watchlist write	Fills the Watch variables with the values found in the Watchlist ('Extras' 'Write receipt')

Table 26: Linking libraries

Command	Libraries
library add <library file1> <library file2> .. <library fileN>	Attaches the specified library file to the library list of the currently open project. If the file path is a relative path, the library directory entered in the project is used as the root of the path.
library delete [<library1> <library2> .. <libraryN>]	Deletes the specified libraries from the library list of the currently open project.

Table 27: Copying objects

Command	Description
object copy <source project file> <source path> <target path>	Copies objects from the specified path of the source project file to the target path of the already opened project.
If the source path is the name of an object, this will be copied. If it is a folder, all objects below this folder will be copied. In this case, the folder structure below the source folder will be duplicated.	If the target path does not yet exist, it will be created.

Table 28: Read-only access for particular objects

Command	Description
object setreadonly <TRUE|FALSE> <object type> | <object name>	Sets read-only access to a object; Define the object type and in case of object types pou, dut, gvl, vis also the name of the object.
Possible object types: pou, dut (data type), gvl (global variables list), vis (visualization), cnc (CNC object), liblist (Libraries), targetsettings, toolinstanceobject (particular Tools instance), toolmanagerobject (all instances in the Tools tree), customplconfig (PLC configuration), projectinfo (Project information), taskconfig (task configuration), trace, watchentrylist (Watch- and Recipe Manager), alarmconfig (Alarm configuration)	e.g. "object setreadonly TRUE pou plc_prg" will set the PLC_PRG to read-only access

Table 29: Entering communications parameters (gateway, device)

Command	Description
gateway local	Sets the gateway on the local computer as the current gateway.
gateway tcpip <Address> <Port>	Sets the gateway in the specified remote computer as the current gateway.
<Address>: TCP/IP address or hostname of the remote computer	<Port>: TCP/IP port of the remote gateway
Important: Only gateways that have no password set can be reached!	device guid <guid>
Sets the device with the specified GUID as the current device.	GUID must have the following format:
{01234567-0123-0123-0123-0123456789ABC}	The curly brackets and the hyphens must appear at the specified positions.
device instance <Instance name>	Sets the instance name for the current device to the name specified
device parameter <Id> <Value>	Assigns the specified value, which will then be interpreted by the device, to the parameter with the specified ID.

Table 30: System call

Command	Description
system <command>	Carries out the specified operating system command.

Table 31: Select target system

Command	Description
target <Id>	Sets the target platform for the current project. If CODESYS is getting started with command line option "/notargetchange", only by this command a target can be set.

Table 32: Query system state

Command	Description
state offline	Returns "S_OK", if currently there is no connection between programming system and target system (offline mode), otherwise "HRESULT[0x800441f0)" (online mode).
state online	Returns "S_OK", if currently there is a connection between programming system and target system (online mode), otherwise "HRESULT[0x800441f0)" (offline mode).

Table 33: Password for user group

user level	User group, the password for which is defined in the project and is given by the subsequent command "user password".
user password	Password for user group specified by the preceding command "user group".

With the password settings - when opening a project which is protected by user group passwords - you can enter the password for a certain user group. Thus, even if the display of the user input dialogs is switched off ("query off...", see above), a password-protected project can be opened via the command file. The entries for the user group and the password must be placed before the command "file open..."!

Example:

```
user level 0
user password aaa
file open "D:\codesys\projects\xxxx.pro"
query off ok
```

Table 34: Visualization settings

Command	Description
visual settings...	Corresponds to the possible settings which can be done for a visualization in 'Extras' 'Settings', category Language, or in the Target Settings, category Visualization
... language file on || off	Option 'Language file gets activated (on) or deactivated (off). In case of activation the option 'Dynamic texts' will be deactivated.
... set languagefile <path language file>	Specification of the language file to be used (.tlt or .vis). Example: "visual settings set languagefile proj1.tlt"
... dynamictexts on || off	Option 'Dynamic texts' will be deactivated (on) or deactivated (off). In case of activation the option Language file will be deactivated.
... dynamictextfiles <file path> | <file path> | ...	Specification of a list of language file paths to be used. The previous list will be deleted. Example: "visual settings D:\dynfiles\p1.xml D:\dynfiles\p2.xml"
... dynamictexthideelements on || off	Activation or deactivation of option 'Suppress elements if no text replacement has taken place'.

Command	Description
... language <language>	Specification of the language to be used; Example: "visual settings language German"
... tablekeyboardusage_web on || off	Activation or deactivation of option 'Keyboard usage for tables' in the web visualization (Target Settings).
... tablekeyboardusage_codesys on || off	Activation or deactivation of option 'Keyboard usage for tables'.
visual webvisuactivation on || off	Activation or deactivation of option web visualization.

Table 35: Commands concerning managing the project in the ENI project database

Command	Description
eni on eni off	The option 'Use source control (ENI)' will be activated or deactivated (Dialog 'Project' 'Options' 'Project source control')
eni project readonly on eni project readonly off	The option 'Read only' for the database category 'Project objects' will be activated or deactivated. (Dialog 'Project' 'Options' 'Project objects')
eni shared readonly on eni shared readonly off	The option 'Read only' for the database category 'Shared objects' will be activated or deactivated. (Dialog 'Project' 'Options' 'Shared objects')
eni set local <POUname>	The object will be assigned to category 'Local', i.e. it will not be stored in the project database. (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni set shared <POUname>	The object will be assigned to category 'Shared objects'. (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni set project <POUname>	The object will be assigned to category 'Project objects'. (Dialog 'Project' 'Object' 'Properties' 'Data base-connection')
eni <category> server <TCP/IP_Address> <Port> <Projectname> <Username> <Password>	Configures the connection to the ENI Server for the category 'Project objects'. (Dialog 'Project' 'Options' 'Project data base'); Example: `eni project server localhost 80 batchtest\project EniBatch Batch` (TCP/IP-Address = localhost, Port = 80, Project name = batchtest\project, User name = EniBatch, Password = Batch)

Command	Description
eni compile sym on eni compile sym off	The option 'Create ASCII symbol information (.sym)' for the objects of category 'Compile files' will be activated/deactivated. (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')
eni compile sdb on eni compile sdb off	The option 'Create binary symbol information (.sym)' for the objects of category 'Compile files' will be activated/deactivated. (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')
eni compile prg on eni compile prg off	The option 'Create boot project' for the objects of category 'Compile files' will be activated/deactivated. (Dialog 'Project' 'Options' 'Project source control' 'ENI settings' for 'Compile files')

In the description of the commands placeholders are used:

<category>: Replace by "project" or "shared" or "compile" depending on which of the following database categories is concerned: Project Objects, Shared Objects, Compile Files

<POUname>: Name of the object, corresponds to the object name which is used.

<Objecttype>: Replace by the shortcut, which is appended as an extension to the POU name of the object in the database, and which reflects the object type (defined by the list of object types, see ENI Administration, 'Object Types'). Example: Object "GLOBAL_1.GVL": the POU name is "GLOBAL_1", the object type is "GVL" (global variables list).

<comment>: Replace by a comment text (embraced by single quotation marks) which will be stored in the version history with the particular action.

Table 36: Commands of the menu 'Project' 'Data Base Link' for working with the database

Command	Description
eni set <category>	The object gets assigned to the named database category ('Define')
'eni set <category>set <Objecttype>:<POUname> <Objecttype>:<POUname> eni <category> getall	The objects which are listed separated by spaces will be assigned to the named database category. ('Multiple Define') Example: "`eni set project pou:as_fub pou:st_prg`" The objects (pou) as_fub and st_prg get assigned to category 'Project objects'
	The latest version of all objects of the named category will be called from the database ('Get All Latest Versions')
'eni <category>get <Objecttype>:<POUname> <Objecttype>:<POUname>	The objects of the named category, which are listed separated by spaces will be called from the database. ('Multiple Define'). ('Get latest version') Example: "`eni project get pou:as_fub gvl:global_1`" Tthe POU as_fub.pou and the global variables list global_1.gvl will be called from the database

Command	Description
eni <category> checkoutall "<comment>"	All objects of the named category will be checked out from the database. The defined comment will be stored with the check-out-action in the version history.
eni <category> checkout "<comment>" <Objecttype>:<POUname> <Objecttype>:<POUname>	All objects (Objecttype:POUname) which are listed separated by spaces will be checked out from the database. The defined comment will be stored with the check-out-action in the version history for each particular object. Example: "`eni project checkout "for working on xy" pou:as_fub gvl:global_1`" The POU as_fub and the global variables list global_1 will be checked out and the comment "for working on xy" will be stored with this action
eni <category>checkinall "<comment>"	All objects of the project, which are under source control in the project database, will be checked in. The defined comment will be stored with the check-in-action.
eni <category> checkin "<comment>" <Objecttype>:<POUname> <Objecttype>:<POUname>	All objects (Objecttype:POUname) which are listed separated by spaces will be checked in to the database. The defined comment will be stored with the check-in-action in the version history for each particular object. (see above: check out) The defined comment will be stored with the check-in-action in the version history for each particular object.

Keywords for the command parameters:

The following keywords, enclosed in "$", can be used in command parameters:

$PROJECT_NAME$	Name of the current project in CODESYS (file name without extension ".pro", e.g. "project_2.pro")
$PROJECT_PATH$	Path of the directory, where the current project file is (without indication of the drive and without a backslash at the end, e.g. "projects\sub1").
$PROJECT_DRIVE$	Drive, where the current project is (without backslash at the end, e.g. "D:")
$COMPILE_DIR$	Compile directory of the current project (with indiciation of the drive and without backslash at the end, e.g. "D:\codesys\compile")
EXE_DIR	Directory where the codesys.exe file is (with indication of the drive and without backslash at the end, e.g. D:\codesys)

Example of a command file:

A command file like shown below will open the project file ampel.pro, will then load a watch list, which was stored as w.wtc, will then start the application program and write after 1 second delay - the values of the variables into the watch list watch.wtc (which will be saved) and will finally close the project.

```
file open C:\projects\codesys_test\ampel.pro
  query off ok
  watchlist load c:\work\w.wtc
  online login
  online run
  delay 1000
  watchlist read
  watchlist save $PROJECT_DRIVE$\$PROJECT_PATH$\w_update.wtc
  online logout
  file close
```

This command file will open the project ampel.pro, will load an existing watchlist w.wtc, will start the application program, after 1 second will write the variables values to the watch list w_update.wtc, which will be saved in the directory "C:\projects\codesys_test" and then will close the project again.

A command file is called in a command line as shown here:

```
"<path of codesys.exe>" /cmd "<path of cmd file>"
```

1.4.1.9.2 Use of keyboard

Overview

If you would like to run the program using only the keyboard, you will find it necessary to use a few commands that are not found in the menu.

- The function key *[F6]* allows you to toggle back and forth within the open POU between the Declaration and the Instruction parts.
- *[Alt]* + *[F6]* allows you to move from an open object to the Object Organizer and from there to the Message window if it is open. If a Search box is open, *[Alt]* + *[F6]* allows you to switch from Object Organizer to the Search box and from the there back to the object.
- Press *[Ctrl]* + *[F6]* to move to the next open editor window, press *[Ctrl]* + *[Shift]* + *[F6]* to get to the previous
- Press *[Tab]* to move through the input fields and buttons in the dialog boxes.
- The arrow keys allow you to move through the register cards and objects within the Object Organizer and Library Manager.

All other actions can be performed using the menu commands or with the shortcuts listed after the menu commands. *[Shift]* + *[F10]* opens the context menu which contains the commands most frequently used for the selected object or for the active editor.

Key combinations

The following is an overview of all key combinations and function keys:

Function	Function Key	
General Functions		
Move between the declaration part and the instruction part of a POU	<F6>	
Context Menu	<Shift>+<F10>	
Shortcut mode for declarations	<Ctrl>+<Enter>	

Function	Function Key	
Move from a message in the Message window back to the original position in the editor	<Enter>	
Move to the next open editor window	<Ctrl>+<F6>	
Move to the previous open editor window	<Ctrl>+<Shift>+<F6>	
Open and close multi-layered variables	<Enter>	
Open and close folders	<Enter>	
Switch register cards in the Object Organizer and the Library Manager	<Arrow keys>	
Move to the next field within a dialog box	<Tab>	
Context sensitive Help	<F1>	
General Commands		
'File' 'Save'	<Ctrl>+<S>	
'File' 'Print'	<Ctrl>+<P>	
'File' 'Exit'	<Alt>+<F4>	
'Project' 'Build'	<F11>	
'Project' 'Delete Object'	<Del>	
'Project' 'Add Object'	<Ins>	
'Project' 'Rename Object'	<Spacebar>	
'Project' 'Open Object'	<Enter>	
'Edit' 'Undo'	<Ctrl>+<Z>	
'Edit' 'Redo'	<Ctrl>+<Y>	
'Edit' 'Cut'	<Ctrl>+<X> or <Shift>+<Del>	
'Edit' 'Copy'	<Ctrl>+<C>	
'Edit' 'Paste'	<Ctrl>+<V>	
'Edit' 'Delete'	<Del>	
'Edit' 'Find next'	<F3>	
'Edit' 'Input Assistant'	<F2>	
'Edit' 'Auto Declare'	<Shift>+<F2>	
'Edit' 'Next Error'	<F4>	
'Edit' 'Previous Error'	<Shift>+<F4>	
'Online' 'Log-in'	<Alt><F8>	
'Online' 'Logout'	<Ctrl>+<F8>	
'Online' 'Run'	<F5>	
'Online' 'Toggle Breakpoint'	<F9>	
'Online' 'Step over'	<F10>	
'Online' 'Step in'	<F8>	
'Online' 'Single Cycle'	<Ctrl>+<F5>	
'Online' 'Write Values'	<Ctrl>+<F7>	
'Online' 'Force Values'	<F7>	

Function	Function Key	
'Online' 'Release Force'	<Shift>+<F7>	
'Online' "Write/Force dialog'	<Ctrl><Shift>+<F7>	
'Window' 'Messages'	<Shift>+<Esc>	
FBD Editor Commands		
'Insert' 'Network (after)'	<Ctrl>+<T>	
'Insert' 'Assignment'	<Ctrl>+<A>	
'Insert' 'Jump'	<Ctrl>+<L>	
'Insert' 'Return'	<Ctrl>+<R>	
'Insert' 'Function Block'	<Ctrl>+<B>	
'Insert' 'Input'	<Ctrl>+<U>	
'Extras' 'Negate'	<Ctrl>+<N>	
'Extras' 'Zoom'	<Alt>+<Enter>	
CFC Editor Commands		
'Insert' 'POU'	<Ctrl>+<B>	
'Insert' 'Input'	<Ctrl>+<E>	
'Insert' 'Output'	<Ctrl>+<A>	
'Insert' 'Jump'	<Ctrl>+<J>	
'Insert' 'Label'	<Ctrl>+<L>	
'Insert' 'Return'	<Ctrl>+<R>	
'Insert' 'Comment'	<Ctrl>+<K>	
'Insert' 'POU input'	<Ctrl>+<U>	
'Extras' 'Negate'	<Ctrl>+<N>	
'Extras' 'Set/Reset'	<Ctrl>+<T>	
'Extras' 'Connection'	<Ctrl>+<M>	
'Extras' 'EN/ENO'	<Ctrl>+<I>	
'Extras' 'Zoom'	<Alt>+<Enter>	
LD Editor Commands		
'Insert' 'Network (after)'	<Ctrl>+<T>	
'Insert' 'Contact'	<Ctrl>+<K>	
'Insert' 'Contact (negated)'	<Ctrl>+<G>	
'Insert' 'Parallel Contact'	<Ctrl>+<R>	
'Insert' 'Parallel contact (negated)'	<Ctrl>+<D>	
'Insert' 'Function Block'	<Ctrl>+<B>	
'Insert' 'Coil'	<Ctrl>+<L>	
'Insert' ''Set' coil'	<Ctrl>+<I>	
'Insert at blocks' 'Input'	<Ctrl>+<U>	
'Insert at blocks' 'Assign'	<Ctrl>+<A>	

Function	Function Key	
'Extras' 'Negate'	<Ctrl>+<N>	
'Extras' 'Zoom'	<Alt>+<Enter>	
SFC Editor Commands		
'Insert' 'Step-Transition (before)'	<Ctrl>+<T>	
'Insert' 'Step-Transition (after)'	<Ctrl>+<E>	
'Insert' 'Alternative Branch (right)'	<Ctrl>+<A>	
'Insert' 'Parallel Branch (right)'	<Ctrl>+<L>	
'Insert' 'Jump'	<Ctrl>+<U>	
'Extras' 'Zoom Action/Transition'	<Alt>+<Enter>	
Move back to the editor from the SFC Overview	<Enter>	
Work with the PLC- resp. Task Configuration		
Open and close organization elements	<Enter>	
Place an edit control box around the name	<Spacebar>	
'Extras' 'Edit Entry'	<Enter>	
Working in the Parameter Manager Editor		
Toggle between navigation window and list editor	<F6>	
Delete a line in the list editor	<Ctrl>+<Del> <Shift>+<Del>	
Delete a field in the list editor	<Del>	

1.4.1.9.3 Reserved keywords

The following strings are reserved as keywords, i.e. they cannot be used as identifiers for variables or POUs:

ABS *Chapter 1.4.1.6.10.1 “ABS” on page 430*

ACOS *Chapter 1.4.1.6.10.10 “ACOS” on page 433*

ACTION (only used in the Export Format)

ADD *Chapter 1.4.1.6.2.1 “ADD” on page 407*

ADR *Chapter 1.4.1.6.7.1 “ADR” on page 421*

ADRINST *Chapter 1.4.1.6.7.2 “ADRINST” on page 421*

AND *Chapter 1.4.1.6.3.1 “AND” on page 410*

ANDN *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

ARRAY *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*

ASIN *Chapter 1.4.1.6.10.9 “ASIN” on page 433*

AT *Chapter 1.4.1.7.4.2 “Address” on page 441*

ATAN ↳ *Chapter 1.4.1.6.10.11 “ATAN” on page 434*
BITADR ↳ *Chapter 1.4.1.6.7.3 “BITADR” on page 421*
BOOL ↳ *Chapter 1.4.1.8.1.2 “BOOL” on page 443*
BY ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
BYTE ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
CAL ↳ *Chapter 1.4.1.6.8.1 “CAL” on page 422*
CALC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
CALCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
CASE ↳ *Chapter 1.4.1.1.10.4.9 “CASE instruction” on page 169*
CONSTANT ↳ *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*
COS ↳ *Chapter 1.4.1.6.10.7 “COS” on page 432*
DATE ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*
DINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
DIV ↳ *Chapter 1.4.1.6.2.4 “DIV” on page 408*
DO ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
DT ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*
DWORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
ELSE ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
ELSEIF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
END_ACTION (only used in the Export Format)
END_CASE ↳ *Chapter 1.4.1.1.10.4.9 “CASE instruction” on page 169*
END_FOR ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
END_FUNCTION (only used in the Export Format)
END_FUNCTION_BLOCK (only used in the Export Format)
END_IF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*
END_PROGRAM (only used in the Export Format)
END_REPEAT ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*
END_STRUCT ↳ *Chapter 1.4.1.8.2.5 “Structures” on page 449*
END_TYPE ↳ *Chapter 1.4.1.8.2.4 “Enumeration” on page 448*
END_VAR ↳ *Chapter 1.4.1.3.9.6 “Local variables” on page 301*
END_WHILE ↳ *Chapter 1.4.1.1.10.4.11 “WHILE loop” on page 170*
EQ ↳ *Chapter 1.4.1.6.6.5 “EQ” on page 420*
EXIT ↳ *Chapter 1.4.1.1.10.4.13 “EXIT instruction” on page 171*
EXP ↳ *Chapter 1.4.1.6.10.5 “EXP” on page 431*
EXPT ↳ *Chapter 1.4.1.6.10.12 “EXPT” on page 434*
FALSE ↳ *Chapter 1.4.1.8.1.2 “BOOL” on page 443*
FOR ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*
FUNCTION ↳ *Chapter 1.4.1.1.9.3 “Function” on page 151*
FUNCTION_BLOCK ↳ *Chapter 1.4.1.1.9.4 “Function block” on page 153*
GE ↳ *Chapter 1.4.1.6.6.4 “GE” on page 419*
GT ↳ *Chapter 1.4.1.6.6.1 “GT” on page 418*
IF ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*

INDEXOF ↳ *Chapter 1.4.1.6.2.7 “INDEXOF” on page 410*
INI ↳ *Chapter 1.4.1.6.11.1 “INI operator” on page 434*
INT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
JMP ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
JMPC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
JMPCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LD ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LDN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
LE ↳ *Chapter 1.4.1.6.6.3 “LE” on page 419*
LINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
LN ↳ *Chapter 1.4.1.6.10.3 “LN” on page 431*
LOG ↳ *Chapter 1.4.1.6.10.4 “LOG” on page 431*
LREAL ↳ *Chapter 1.4.1.8.1.4 “REAL / LREAL” on page 443*
LT ↳ *Chapter 1.4.1.6.6.2 “LT” on page 419*
LWORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*
MAX ↳ *Chapter 1.4.1.6.5.3 “MAX” on page 416*
MIN ↳ *Chapter 1.4.1.6.5.4 “MIN” on page 417*
MOD ↳ *Chapter 1.4.1.6.2.5 “MOD” on page 409*
MOVE ↳ *Chapter 1.4.1.6.2.6 “MOVE” on page 409*
MUL ↳ *Chapter 1.4.1.6.2.2 “MUL” on page 407*
MUX ↳ *Chapter 1.4.1.6.5.6 “MUX” on page 418*
NE ↳ *Chapter 1.4.1.6.6.6 “NE” on page 420*
NOT ↳ *Chapter 1.4.1.6.3.4 “NOT” on page 412*
OF ↳ *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*
OR ↳ *Chapter 1.4.1.6.3.2 “OR” on page 411*
ORN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
PERSISTENT ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*
POINTER ↳ *Chapter 1.4.1.8.2.3 “Pointer” on page 447*
PROGRAM ↳ *Chapter 1.4.1.1.9.7 “Program” on page 156*
R ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
READ_ONLY
READ_WRITE
REAL ↳ *Chapter 1.4.1.8.1.4 “REAL / LREAL” on page 443*
REPEAT ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*
RET ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETAIN ↳ *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*
RETC ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETCN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*
RETURN ↳ *Chapter 1.4.1.1.10.4.7 “RETURN instruction” on page 168*
ROL ↳ *Chapter 1.4.1.6.4.3 “ROL” on page 414*
ROR ↳ *Chapter 1.4.1.6.4.4 “ROR” on page 415*
S ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

SEL ↳ *Chapter 1.4.1.6.5.2 “SEL” on page 416*

SHL ↳ *Chapter 1.4.1.6.4.1 “SHL” on page 412*

SHR ↳ *Chapter 1.4.1.6.4.2 “SHR” on page 413*

SIN ↳ *Chapter 1.4.1.6.10.6 “SIN” on page 432*

SINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

SIZEOF ↳ *Chapter 1.4.1.6.2.8 “SIZEOF” on page 410*

SQRT ↳ *Chapter 1.4.1.6.10.2 “SQRT” on page 430*

ST ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

STN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

STRING ↳ *Chapter 1.4.1.8.1.5 “STRING” on page 444*

STRUCT ↳ *Chapter 1.4.1.8.2.5 “Structures” on page 449*

SUB ↳ *Chapter 1.4.1.6.2.3 “SUB” on page 408*

TAN ↳ *Chapter 1.4.1.6.10.8 “TAN” on page 433*

THEN ↳ *Chapter 1.4.1.1.10.4.8 “IF instruction” on page 168*

TIME ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*

TO ↳ *Chapter 1.4.1.1.10.4.10 “FOR loop” on page 169*

TOD ↳ *Chapter 1.4.1.8.1.6 “Time data types” on page 444*

TRUE ↳ *Chapter 1.4.1.8.1.2 “BOOL” on page 443*

TRUNC ↳ *Chapter 1.4.1.6.9.9 “TRUNC” on page 429*

TYPE ↳ *Chapter 1.4.1.8.2.4 “Enumeration” on page 448*

UDINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

UINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

ULINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

UNTIL ↳ *Chapter 1.4.1.1.10.4.12 “REPEAT loop” on page 170*

USINT ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

VAR ↳ *Chapter 1.4.1.3.9.6 “Local variables” on page 301*

VAR_ACCESS (only used very specifically, depending on the hardware)

VAR_CONFIG ↳ *Chapter 1.4.1.4.1.4.1 “Overview” on page 361*

VAR_CONSTANT ↳ *Chapter 1.4.1.3.9.8 “Constants, typed literals” on page 302*

VAR_EXTERNAL ↳ *Chapter 1.4.1.3.9.9 “External variables” on page 303*

VAR_GLOBAL ↳ *Chapter 1.4.1.4.1.3.2 “Several variables lists” on page 359*

VAR_IN_OUT ↳ *Chapter 1.4.1.3.9.5 “Input and output variables” on page 301*

VAR_INPUT ↳ *Chapter 1.4.1.3.9.3 “Input variable” on page 301*

VAR_OUTPUT ↳ *Chapter 1.4.1.3.9.4 “Output variable” on page 301*

WHILE ↳ *Chapter 1.4.1.1.10.4.11 “WHILE loop” on page 170*

WORD ↳ *Chapter 1.4.1.8.1.3 “Integer data types” on page 443*

WSTRING (IEC data type is not supported)

XOR ↳ *Chapter 1.4.1.6.3.3 “XOR” on page 411*

XORN ↳ *Chapter 1.4.1.1.10.3.2 “Modifiers and operators in IL” on page 163*

Additionally all conversion operators as listed in the 'Edit' 'Input assistant' are handled as keywords ↳ *Chapter 1.4.1.2.5.11 “'Edit' 'Input assistant'” on page 276*.

1.4.1.9.4 File types

The following file types can be created:

File extension	Example	Description	Format	Path (default)
*.pro	project01.pro	Project file	binary	project directory
*.ci	project01<number>.ci	Information on the last build (compilation) of the project → incremental compile possible; only created when project gets saved	number: coded target ID	binary
project directory	*.eci	project01<number>.eci	external Compile-Information; Subset of the ci-file in eci-format; can be read via an access-dll	number: coded target ID
PE	project directory	*.cic	project01<number>.cic	target-dependant information on the last build (compilation) of the project -> incremental compile possible; only created when project gets saved
number: coded target ID	binary	project directory	*.cit	project01<number>.cit
temporary *.ci-file; created at target change, transformed to a ci-file at next save of the project	number: coded target ID	binary	project directory	*.ri
project01<number>.ri	information on the last download, important for Online Change; created at each download	number: coded target ID	binary	project directory
*.exp	project01.exp, PLC_PRG.exp	export file ('Project' 'Export')	Export format (Text)	project directory
*.tlt	*.txt	project01.tlt	project01.txt	translation file (defined in 'Project' 'Translate in another language')
Text		*.sym	project01.sym	symbol file
Text	project directory	*.sdb	project01.sdb	symbol file
binary	project directory	*.sym_xml	project01.sym_xml	symbol file
XML	project directory	*.asd	project01.asd	save file (temporary, 'Auto save', 'Auto save before compile')
binary	project directory	*.asl	lib01.asl	save file for a library opened as project (temporary, 'Auto save', 'Auto save before compile')

File extension	Example	Description	Format	Path (default)
binary	library resp. project directory	*.bak	project01.bak	backup file for project (permanent, 'Create backup')
binary	project directory	*.prg	*.bin	default.prg
project01.prg	boot project, file name depending on target	binary	target system (created online)	project directory (created offline)
*.chk		default.chk	project01.chk	checksum for boot projekt code
binary	target system (created online)	project directory (created offline)	*.ini	codesys.ini
ini-file for various settings	Text	with codesys.exe	*.dfr	default.dfr
project01.dfr	frame file (Printer setup)	binary	with codesys.exe	*.asm
code386.asm	assembler-Listing of the created project code	Text	compile directory	*.lst
project01.lst	assembler-Listing of the created project code	Text	compile directory	
*.bpl	project01.bpl	debug-files (breakpoint-information)	Text	compile directory
	*.st	PLC_PRG.st	debug-files, implicit ST-code	Text
compile directory		*.map	project01.map	map-file; information on memory organization and variable locations
Text	compile directory		*.hex	*.h86
project01.hex (Output) resp. standard.hex (Lib)	.hex for Intel or Motorola, .h86 for Intel; compiler output or input for external library	Intel or Motorola hex-files	compile directory	resp. library directory
*.trd	projectxy0.trd	trend logging (number befor the dot is counted up, if file is full and another must be created)	Text	project directory
*.log	projectxy.log	log file (Log)	binary	project directory
*.wtc	projx_watch1.wtc	watch list (Watch- and Recipe-Manager)	Text	user defined directory
*.alm	alarmlog0.alm	alarm log-file		user defined directory resp. download-directory of the controller

File extension	Example	Description	Format	Path (default)
*.zip	projectxy.zip	project archive file; zip-file with files belonging to the project , 'File' 'Save/ Mail Archive'		user defined directory
*.trc	project01_tr1.trc	trace recording	binary	with codesys.exe
*.mon	project01_tr1.mon	trace recording	XML	with codesys.exe
*.tcf	project01_tr1.tcf	trace configuration	binary	with codesys.exe

1.4.1.10 Compiler errors and warnings

1.4.1.10.1 Remarks on compiler errors and warnings

If errors are detected during the compilation of the project, messages will be dumped in the message window. Also warnings might be displayed there. *[F4]* always allows to jump to the next message line in this window, whereby also the concerned POU will be opened. The error and warning messages are preceded by unique numbers. If a message line is selected in the message window, *[F1]* will open the corresponding online help window.

See also:

-
- *Chapter 1.4.1.2.3.11 “'Project' 'Build'” on page 231*
- *Chapter 1.4.1.2.1.6 “Message window” on page 200*
- *Chapter 1.4.1.2.5.15 “'Edit' 'Next error'” on page 278*
- *Chapter 1.4.1.2.8.1 “'Help' 'Contents and search'” on page 293*

1.4.1.10.2 Warnings

1100

"Unknown function '<name>' in library." An external library is used. Please check, whether all functions, which are defined in the .hex file, are also defined in the .lib file.

1101

"Unresolved symbol '<Symbol>'." The code generator expects a POU with the name <Symbol>. It is not defined in the project. Define a function/program with this name.

1102

"Invalid interface for symbol '<Symbol>'." The code generator expects a function with the name <Symbol> and exactly one scalar input, or a program with the name <Symbol> and no input or output.

1103

"The constant '<name>' at code address '<address>' overwrites a 16K page boundary!" A string constant exceeds the 16K page boundary. The system cannot handle this. It depends on the runtime system whether the problem could be avoided by an entry in the target file.

1200

"Task '<name>', call of '<name>' Access variables in the parameter list are not updated"

Variables, which are only used at a function block call in the task configuration, will not be listed in the cross-reference list.

1300

"File not found '<name>'"

The file, to which the global variable object is pointing, does not exist. Please check the path.

1301

"Analyze-Library not found! Code for analyzation will not be generated."

The analyze function is used, but the library analyzation.lib is missing. Add the library in the library manager.

1302

"New externally referenced functions inserted. Online Change is therefore no longer possible!"

Since the last download you have linked a library containing functions which are not yet referenced in the runtime system. For this reason you have to download the complete project.

1400

"Unknown Pragma '<Name>' is ignored!"

This pragma is not supported by the compiler. See 'Pragmas' for supported directives
↳ *Pragmas*.

1401

"The struct '<name>' does not contain any elements."

The structure does not contain any elements, but variables of this type allocate 1 Byte of memory.

1410

"'RETAIN' and 'PERSISTENT' do not have any effect in functions"

Remanent variables which are defined locally in functions are handled like normal local variables.

1411

"Variable '<name>' in the variable configuration isn't updated in any task"

The top level instance of the variable is not referenced by a call in any task. Thus it will not be copied from the process image.

Example:

```
Variable Configuration:
VAR_CONFIG
 plc_prg.aprg.ainst.in AT %IB0 : INT;
END_VAR
plc_prg:
index := INDEXOF(aprg);
```

The program aprg is referenced but not called. Thus plc_prg.aprg.ainst.in never will get the actual value of %IB0.

1412

"Unexpected token Name-Name in pragma {pragma name}"

You are using a pragma which is not written correctly or which cannot be used at this location. See *Pragmas*.

1413

"'<Name>' is not a valid key for list '<Name>'. The key will be ignored"

In the pragma a nonexistent parameter list is specified. Check the list name resp. have a look in the Parameter Manager for the currently available lists.

1414

Too many component definitions in pragma '<name>'

The Pragma contains more definitions (in square brackets) than there are elements in the corresponding array, function block or structure.

1415

'<Name>' (<Number>): The literal '<Number>' is assigned to more than one enumeration

In the declaration of enumeration <Name> the same number is assigned to more than one enumeration components (e.g. TYPE aenum (a:=1, b:=1); END_TYPE).

1500

"Expression contains no assignment. No code was generated."

The result of this expression is not used. For this reason there is no code generated for the whole expression.

1501

"String constant passed as 'VAR_IN_OUT': '<Name>' must not be overwritten!"

The constant may not be written within the POU, because there no size check is possible.

1502

"Variable '<Name>' has the same name as a POU. The POU will not be called!"	A variable is used, which has the same name like a POU.

Example:

```
PROGRAM a
...
VAR_GLOBAL
    a: INT;
END_VAR ...
a; (* Not POU a is called but variable a is loaded. *)
```

1503

"The POU '<name>' has no outputs. Box result is set to 'TRUE'."	The Output pin of a POU which has no outputs, is connected in FBD or KOP. The assignment automatically gets the value TRUE.

1504

"'<name>' ('<number>'): Statement may not be executed due to the evaluation of the logical expression"	Eventually not all branches of the logic expression will be executed.

Example:

```
IF a AND funct(TRUE) THEN ....
```

If a is FALSE then funct will not be called.

1505

"Side effect in '<Name>'! Branch is probably not executed !"	The first input of the POU is FALSE. For this reason the side branch, which may come in at the second input, will not be executed.

1506

"Variable '<name>' has the same name as a local action. The action will not be called!"	Rename the variable or the action.

1507

"Instance '<name>' has the same name as a function. The instance will not be called"	You call in ST an instance which has the same name as a function. The function will be called! Use different names.

1509

"'<name>' ('<number>'): Functions to be registered as callbacks have to start with 'Callback'."	You are using a callback function, the name of which does not start with "callback".This can cause unexpected effects on RISC resp. Motorola 68K controllers!

1510

"Operand to be rotated has no explicit type. Please use a typed literal, like 'DWORD#1'."	For the constant used in the ROL resp. ROR operation there must be a data type definition. For information on typed constants see Typed Literals ↳ *Typed Literals*.

1511

"Operand to be shifted has no explicit type. Please use a typed literal, like 'DWORD#1'."	For the constant used in the SHL resp. SHR operation there must be a data type definition. For information on typed constants see 'Typed Literals' ↳ *Typed Literals*.

1550

"Multiple calls of the POU '<Name>' in one network may lead to undesired side effects"	Check whether the multiple call of this POU is really necessary. By a multiple call unwanted value overstrikes may occur.

1600

"Open DB unclear (generated code may be erroneous)."	The original Siemens program does not tell, which POU is opened.

1700

"Input not connected."	An input box is used in CFC which has no assignment. For this no code will be generated.

1750

"Step '<Name>': the minimal time is greater than the maximal time!"	Open dialog 'Step attributes' for this step and correct the time definitions.

1751

"Caution with usage of variable '<name>'. This variable is used by implicit code and influences the behaviour of the step sequence."	We recommend to rename the variable, so that unique identifiers are used within the project and undesired side effects can be avoided.

1800

"<name>(element #<element number>): Invalid watchexpression '<name>'"	The visualization element contains an expression which cannot be monitored. Check variable name and placeholder replacements.

1801

"'<name> (number): No Input on Expression '<name>' possible"	In the configuration of the visualization object at field input a composed expression is used. Replace this by a single variable.

1802

"<Visualization object>(Element number): Bitmap '<name>' was not found"	Make sure that an external bitmap file is available in the path which is defined in the visualization configuration dialog.

1803

"'<name>'('<number>'): "The print action would not supported for web and target visualisation"	A print action is assigned to an alarm configured in the visualization. This will not be regarded in the web or target visualization.

1804

"'<name>'('<number>'): The font '<name>' is not supported by the target."	In the visualization you are using a font which is not supported by the target system. See in the target settings, category 'Visualization' for the supported fonts.

1807

"<name> (<number>): No message window for alarms for target visualization"	Note that action "message" is not supported for the target visualization!

1808

"'<name>'('<number>'): A polygon consists of to many points for target	Per default maximum 512 points are allowed, target-specifically another maximum number might be defined. By opening the configuration dialog the element will be optimized to the allowed number of points.

visualization. In case of a meter element, please open the configuration once."	
1809	
"'<name> ('<nummer>'): Invalid visualization as zoom target: '<number>'"	This visualization could not be found. Please check whether the visualization name is correct and that the visualization object is available.
1850	
"Input variable at %IB<number> is used in task '<name>' but updated in another task"	Please check which tasks are using this variable and whether the current programming is not causing undesirable effects. The update of the variable value usually is done in the task with the highest priority.
1851	
"Output variable at %IQ<number> is used in task '<name>' but updated in another task"	Please check which tasks are using this variable and whether the current programming is not causing undesirable effects. The update of the variable value usually is done in the task with the highest priority.
1852	
"CanOpen-Master might not be called cyclically in event task '<name>'! Set modul parameter UpdateTask!"	Currently the CanOpen Master is called by the named event task. If you want to get it called cyclically, specify an appropriate task via parameter UpdateTask in the PLC Configuration in dialog 'Module parameters'.
1853	
"A PDO (index: '<number>') might not be updated cyclically in event task '<name>'"	Currently the named PDO is controlled via the named event task. But if you want to get it called cyclically, you must assign an appropriate task to the PDO by shifting I/O-references to this task.
1900	
"POU '<name>' (main routine) is not available in the library"	The Start-POU (e.g. PLC_PRG) will not be available, when the project is used as library.
1901	
"Access Variables and Variable Configurations are not saved in a library!"	Access variables and variable configuration are not stored in the library.

1902

"'<Name>': is no Library for the current machine type!"	The .obj file of the library was generated for another device.

1903

"<Name>: is no valid Library"	The file does not have the format requested for the actual target.

1904

"The constant '<Name>' hides a constant of the same name in a library"	In your project you have defined a constant which has the same name as one which is defined in a linked library. The library variable will be overwritten!

1970

"Parameter manager: List '<Name>' , Column '<Name>', Value '<Name>' could not be imported!"	Check the Import-file *.prm for entries which do not match the current configuration (standard values resp. XML-description file) of the Parameter Manager.

1980

Global network variables '<Name>' '<Name>': simultaneous reading and writing may result in loss of data!"	In the configuration of the network variables list options 'Read' and 'Write' are activated (select list in the Resources tab and open dialog 'Global variables list' via command 'Properties' in the context menu). Note that this might result in data losses during communication.

1990

"No 'VAR_CONFIG' for '<name>'"	For this variable there is no address configuration available in the 'Variable Configuration' (VAR_CONFIG) ↳ *Chapter 1.4.1.4.1.4.1 “Overview” on page 361*. Open window Variable_Configuration in the Resources tab and insert the appropriate configuration (Command 'Insert' 'All Instance Paths' ↳ *Chapter 1.4.1.4.1.4.2 “'Insert' 'All Instance Paths'” on page 361*).

2500

"Task '<task name>': fno cycle time specified for cyclic task"	In the Task configuration a cyclic task has been created, for which no cycle time has been defined. Enter an appropriate time span in dialog 'Taskattributes' at "Interval".

4710

"'<name>' ('<number>'): A required expression has not been configured"	Check the configuration of the slider element in category 'Variables'. There must be defined a valid value for each of the fields 'Min.Value', 'Slider' and 'Max. Value'.

4711

"'<name>'('<number>'): The slider element is not supported by your SysLibTargetVisu."

Install and use the current version of the library SysLibTargetVisu.library.

4714

"'<name>'('<number>'): The type of the selection variable has to be a signed type compatible to INT"

Check and appropriately correct the configuration of the table element in category "Selection".

4715

"The current target does not support formatted display of time and date values"

Option "Formatted time display" is activated in the settings of a visualization ↳ *Chapter 1.4.3.6.1 "'Extras' 'Settings'" on page 700*. However this is not supported by the currently used target system.

4716

"The current target does not support simplified input handling"

In the target settings, in category 'Visualization' option 'Simplified input handling' is activated. Maybe it is not supported by the current target system, because an old version of the library SysLibTargetvisu.lib is used.

1.4.1.10.3 Errors

3100

"Code too large. Maximum size: '<number>' Byte (<number>K)"

The maximum program size is exceeded. Reduce project size.

3101

"Total data too large. Maximum size: '<number>' Byte (<number>K)"

Memory is exceeded. Reduce data usage of the application.

3110

"Error in Library '<Name>'."

The .hex file is not in INTEL Hex format.

3111

"Library '<Name>' is too large. Maximum size: 64K"

The .hex file exceeds the set maximum size.

3112

"Nonrelocatable instruction in library."	The .hex file contains an instruction which is not relocatable. The library code cannot be linked.

3113

"Library code overwrites function tables."	The ranges for code and function tables are overlapping.

3114

"Library uses more than one segment."	The tables and the code in the .hex file use more than one segment.

3115

"Unable to assign constant to VAR_IN_OUT. Incompatible data types."	The internal pointer format for string constants cannot get converted to the internal pointer format of VAR_IN_OUT, because the data are set "near" but the string constants are set " huge" or "far". If possible change these target settings.

3116

"Function tables overwrite library code or a segment boundary."	Code 166x: The external library cannot be used with the current target settings. These must be adapted resp. the library must be rebuilt with appropriate settings.

3117

"<Name> (<code>): Expression too complex. No more registers available"	The named expression is too complex to be handled by the available registers. Please try to reduce the expression by using interim variables.

3120

"Current code-segment exceeds 64K."	The currently generated code is bigger than 64K. Eventually too much initializing code is created.

3121

"POU too large."	A POU may not exceed the size of 64K.

3122

"Initialisation too large. Maximum size: 64K"	The initialisation code for a function or a structure POU may not exceed 64K.

3124

"String constant too large: '<number>' characters (maximum 253 characters)"	The given constant must be reduced in number of characters.

3130

"User-Stack too small: '<number>' DWORD needed, '<number>' DWORD available."	The nesting depth of the POU calls is too big. Enter a higher stack size in the target settings or compile build project without option 'Debug' (set in dialog 'Project' 'Options' 'Build').

3150

"Parameter <number> of function '<name>': Cannot pass the result of a IEC-function as string parameter to a C-function."	Use an intermediate variable, to which the result of the IEC function is assigned.

3160

"Can't open library file '<name>'."	A library <name> is included in the library manager for this project, but the library file does not exist at the given path.

3161

"Library '<name>' contains no code-segment"	An .obj file of a library must contain at least one C function. Insert a dummy function in the .obj file, which is not defined in the .lib file.

3162

"Could not resolve reference in Library '<name>'(Symbol '<name>', Class '<name>', Type '<name>')"	The .obj file contains an unresolvable reference to another symbol. Please check the settings of the C Compiler.

3163

"Unknown reference type in Library '<name>' (Symbol '<name>' , Class '<name>' , Type '<name>')"	The .obj file contains a reference type which is not resolvable by the code generator. Please check the settings of the C Compiler.

3200

"<name>: Boolean expression too complex"	The temporary memory of the target system is insufficient for the size of the expression. Divide up the expression into several partial expressions thereby using assignments to intermediate variables.

3201

"<name> (<network>): A network must not result in more than 512 bytes of code"	Internal jumps can not be resolved. Activate option "Use 16 bit jump offsets" in the 68k target settings.

3202

"Stack overrun with nested string/array/ structure function calls"	A nested function call CONCAT(x, f(i)) is used. This can lead to data loss. Divide up the call into two expressions.

3203

"Expression too complex (too many used address registers)."	Divide up the assignment into several expressions.

3204

"A jump exceeds 32k Bytes"	Jump distances may not be bigger than 32767 bytes.

3205

"Internal Error: Too many constant strings"	In a POU a maximum of 3000 string constants may be used.

3206

"Function block data exceeds maximal size"	A function block may produce maximum 32767 Bytes of code.

3207

"Array optimization"	The optimization of the array accesses failed because during index calculation a function has been called.

3208

"Conversion not implemented yet"	A conversion function is used, which is not implemented for the actual code generator.

3209

"Operator not implemented"	A operator is used which is not implemented for this data type and the actual code generator. MIN(string1,string2).

3210

"Function '<Name>' not found"	A function is called which is not available in the project.

3211

"Max string usage exceeded"	A variable of type string can be used in one expression 10 times at the most.

3212

"Wrong library order at POU <POU name>"	The order of libraries for this POU does not match with that in the cslib.hex file. Correct the order accordingly. (only for 68K targets, if the checking option is activated in the target file.)

3250

"Real not supported for 8 Bit Controller"	The target is currently not supported.

3251

"Date of day types are not supported for 8 Bit Controller"	The target is currently not supported.

3252

"Size of stack exceeds <number> bytes"	The target is currently not supported.

3253

"Could not find hex file: '<Name>' "	The target is currently not supported.

3254

"Call to external library function could not be resolved."	The target is currently not supported.

3255

"Pointers are not supported for 8 bit controllers."	Avoid using pointers in your program to get it running on the 8 bit system.

3260

"Function '<name>' has too many arguments: Increase the size of the argument stack in the target settings."	If possible, modify the size of the stack in dialog Target Platform in the Target Settings (Default: 40).

3400

"An error occured during import of Access variables"	The .exp file contains an incorrect access variables section.

3401

"An error occured during import of variable configuration"	The .exp file contains an incorrect configuration variables section.

3402

"An error occured during import of global variables"	The .exp file contains an incorrect global variables section.

3403

"Could not import <name>"	The section for object <name> in the .exp file is not correct.

3404

"An error occured during import of task configuration"	The section for the task configuration the .exp file is not correct.

3405

"An error occured during import of PLC configuration"	The section for the PLC configuration in the .exp file is not correct.

3406

"Two steps with the name '<name>'. Second step not imported."	The section for the SFC POU in the .exp file contains two steps with equal names. Rename one of the steps in the export file.

3407

"Predecessor step '<name>' not found"	The step <name> is missing in the .exp file.

3408

"Successor step '<name>' not found"	The step <name> is missing in the .exp file.

3409

"No successing transition for step '<´name>' "	In the .exp file a transition is missing, which requires step <name> as preceeding step.

3410

"No successing step for transition '<name>'"	In the .exp file a step is missing which requires the transition <name> as preceeding condition.

3411

"Step '<name>' not reachable from initial step"	In the .exp file the connection between step <name> and the initial step is missing.

3412

"Macro '<name>' not imported"	Check the export file.

3413

"Error during import of the CAMs."	You have imported an export file (*.exp) which contains erroneous information on a CAM. Check the export file.

3414

"Error during import of the CNC program list"	You have imported an export file (*.exp) which contains erroneous information on a CNC program. Check the export file.

3415

"Error during import of the Alarm configuration"	You have imported an export file (*.exp) which contains erroneous information on the Alarm Configuration. Check the export file.

3450

"PDO'<PDO-name>': Missing COB-Id!"	Click on the button 'Properties’ in the PLC configuration dialog for the module and enter a COB ID for the PDO <PDO Name>.

3451

"Error during load: EDS-File '<name>' could not be found, but is referenced in hardware configuration!"	The device file needed for the CAN configuration is not in the correct directory. Check the directory setting for configuration files in 'Project' 'Options' 'Directories'.

3452

"The module '<name>' couldn't be created!"	The device file for module <name> does not fit to the current configuration. It has been modified since the configuration has been set up or it is corrupted.

3453

"The channel '<name>' couldn't be created!"	The device file for channel <name> does not fit to the current configuration. It has been modified since the configuration has been set up or it is corrupted.

3454

"The address '<name>' points to an used memory!"	Option 'Check for overlapping addresses' is activated in the dialog 'Settings' of the PLC configuration and an overlap has been detected. Note that the area check is based on the size which results of the data types of the modules, not on the size which is given by the entry 'size' in the configuration file.

3455

"Error during load: GSD-File '<name>' could not be found, but is referenced in hardware configuration!"	The device file required by the PROFIBUS configuration is not in the correct directory. Check the directory setting for configuration files in 'Project' 'Options' 'Directories'.

3456

"The profibus device '<name>' couldn't be created!"	The device file for module <name> does not fit to the current configuration. It has been modified since the configuration has been set up or it is corrupted.

3457

"Error in module description!"	Please check the device file of this module.

3458

"The PLC-Configuration couldn't be created! Check the configuration files."	Check if all required configuration and device files are available in the correct path (see defined compile directory in 'Project' 'Options' /Directories).

3459

"The selected baudrate is not supported."	Change the setting in the CAN Parameter dialog. Check the specification of the baud rate given by the GSD file.

3460

3S_CanDrv.lib has the wrong version.	Make sure that the 3S_CanDrv.lib included in the project is up to date.

3461

"3S_CanOpenMaster.lib has the wrong version."	Make sure that the 3S_CanOpenMaster.lib included in the project is up to date.

3462

"3S_CanOpenDevice.lib has the wrong version."	Make sure that the 3S_CanOpenDevice.lib included in the project is up to date.

3463

"3S_CanOpenManager.lib has the wrong version."	Make sure that the 3S_CanOpenManager.lib included in the project is up to date.

3464

"3S_CanNetVar.lib has the wrong version."	Make sure that the 3S_CanNetVar.lib included in the project is up to date.

3465

"CanDevice: Sub indices have to be numerated sequentially"	In parameter lists used by the CanDevice the sub-indices must be numbered sequentially and without interruption. Check the corresponding list in the Parameter Manager.

3466

"CAN network variables: No CAN controller found in the PLC configuration"	There are network variables configured for a CAN network (Resources, Global Variables), but in the PLC Configuration there is no CAN Controller available.

3468

"CanDevice: Update task not available in the task configuration."	The update task (used for calling the CANdevice) which is defined in the Base Settings dialog of the CANdevice in the PLC Configuration, must be configured in the Task Configuraiton of the project.

3469

"The CanOpenMaster can not be called. Please assign a task manually."	Assign a task, which should call the master, via parameter UpdateTask in the Module parameters dialog in the PLC Configuration.

3470

"Invalid name in parameter UpdateTask"	Open the CanMasters Module parameter dialog in the PLC Configuration. Check parameter UpdateTask. The specified task must be available in the project. If you cannot set an appropriate task here, the device file must be checked for the corresponding value definitions for UpdateTask.

3500

"No 'VAR_CONFIG' for '<Name>'"	Insert a declaration for this variable in the global variable list which contains the 'Variable_Configuration'.

3501

"No address in 'VAR_CONFIG' for '<name>'."	Assign an address to this variable in the global variable list which contains the 'Variable_Configuration'.

3502

"Wrong data type for '<name>' in 'VAR_CONFIG"	In the global variables list which contains the 'Variable_Configuration' the variable is declared with a different data type than in the POU.

3503

"Wrong data type for '<name>' in 'VAR_CONFIG'"	In the global variables list which contains the 'Variable_Configuration' the variable is declared with a different address than in the POU.

3504

"Initial values are not supported for 'VAR_CONFIG"	A variable of the 'Variable_Configuration' is declared with address and initial value, but an initial value can only be defined for input variables without address assignment.

3505

"'<name>'is no valid instance path"	The Variable_Configuration contains a non-existent variable.

3506

"Access path expected"	In the global variable list for Access Variables the access path for a variable is not correct. Correct: <Identifier>:'<Access path>':<Type> <Access mode>.

3507

"No address specification for 'VAR_ACCESS'-variables"	The global variable list for Access Variables contains an address assignment for a variable. This is not allowed. Valid variable definition: <Identifier>:'<Access path>':<Type> <Access mode>

3550

"Duplicate definition of identifier '<name>'"	There are two tasks defined with identical names. Rename one of them.

3551

"The task '<name>' must contain at least one program call"	Insert a program call or delete the task.

3552

"Event variable '<name>' in task '<name>' not defined" There is an event variable set in the ‘Single’ field of the task properties dialog which is not declared globally in the project. Use another variable or define the variable globally.

3553

"Event variable '<name>' in task '<name>' must be of type 'BOOL'" Use a variable of type BOOL as event variable in the ‘Single’ field of the task properties dialog.

3554

"Task entry '<name>' must be a program or global function block instance" In the field 'Program call’ a function or an undefined POU is entered. Enter a valid program name.

3555

"The task entry '<name>' contains invalid parameters" In the field 'Append program call’ parameters are used which do not comply with the declaration of the program POU.

3556

"Tasks are not supported by the currently selected target" The currently defined task configuration cannot be used for the currently set target system. Change target or modify the task configuration correspondingly.

3557

"Maximum number of Tasks ('<number>') exceeded" The currently defined number of tasks exceeds the maximum number allowed for the currently set target system. Change target or modify the task configuration correspondingly. Attention: Do not edit the XML description file of the task configuration!

3558

"Priority of task '<name>' is out of valid range between '<lower limit>' and '<upper limit>'" The currently defined priority for the task is not valid for the currently set target system. Change target or modify the task configuration correspondingly.

3559

"Task '<name>': Interval-Tasks are not supported by the current target" The current task configuration contains an interval task. This is not allowed by the currently set target system. Change target or modify the task configuration correspondingly.

3560

"Task '<name>': free wheeling tasks are not supported by the current target"

The current task configuration contains a free wheeling task. This is not allowed by the currently set target system. Change target or modify the task configuration correspondingly.

3561

"Task '<name>': event tasks are not supported by the current target"

The current task configuration contains event tasks which are not supported by the currently set target system. Change target or modify the task configuration correspondingly.

3562

"Task '<name>': external event tasks are not supported by the current target"

The current task configuration contains external event tasks which are not supported by the currently set target system. Change target or modify the task configuration correspondingly.

3563

"The interval of task '<name>' is out of valid range between '<lower limit>' and '<upper limit>'"

Change the interval value in the configuration dialog for the task.

3564

"The external event '<name>' of task '<name>' is not supported by the current target"

The currently set target system does not support the external event which is defined in the task configuration for this task. Change target or modify the task configuration correspondingly.

3565

"Maximum number of event tasks ('<number>') exceeded"

The currently set target system does not allow as many event tasks as are defined at the moment. Change target or modify the task configuration correspondingly.

3566

"Maximum number of interval tasks ('<number>') exceeded"

The currently set target system does not allow as many interval tasks as defined at the moment. Change target or modify the configuration correspondingly.

3567

"Maximum number of free wheeling tasks ('<number>') exceeded"

The currently set target system does not allow as many free wheeling tasks as defined at the moment. Change target or modify the configuration correspondingly.

3568

"Maximum number of external interval tasks ('<number>') exceeded"

The currently set target system does not allow as many external interval tasks as defined at the moment. Change target or modify the configuration correspondingly.

3569

"POU '<name>' for system event '<name>' not defined"

The POU which should be called by the named system event, as defined in the task configuration, is not available in the project. Modify the task configuration correspondingly or make sure that the POU is available in the project.

3570

"The tasks '<name>' and '<name>' share the same priority"

Modify the task configuration so that each task has a different priority.

3571

"The library 'SysLibCallback' is not included in the project! System events can not be generated."

In order to create event tasks, the SysLibCallback.lib is needed. Link this library to the project in the library manager, or modify the task configuration (task attributes) so that there is no task triggered by an event.

3572

"Watchdog interval of task '<name>' is out of the valid range from <number>µs' to '<number>µs'"

In the task configuration in dialog 'Taskattributes' there is a watchdog time defined in microseconds, which is out of the valid range defined in the XML-description file ↳ *Chapter 1.4.1.4.8.1 “Overview” on page 390*.

3573

"Watchdog interval of task '<name>' is out of the valid range from <number>%' to

In the task configuration in dialog 'Taskattributes' there is a watchdog time defined in percents, which is out of the valid range defined in the XML-description file ↳ *Chapter 1.4.1.4.8.1 “Overview” on page 390*.

3574

"The event variable '<name>' respectively its direct address must not be used multiple times as an event"

A singleton event is used several times in the task configuration. For information on singleton events see the description in 'Insert' 'Insert task' or 'Insert' 'Append Task' ↳ *Chapter 1.4.1.4.8.2 “'Insert' 'Insert Task' or 'Insert' 'Append Task'” on page 391*.

3575

"Task '<name>': the cycle time has to be a multiple of '<number>' µs."	Correct the cycle time accordingly in the Taskattributes dialog for this task. The target system defines a base time and prescribes that the cycle time must be equal to or be a multiple of this base time.

3600

"Implicit variables not found!"	Use command 'Rebuild all'.

3601

"<name> is a reserved variable name"	The given variable is declared in the project, although it is reserved for the code generator. Rename the variable.

3610

" '<Name>' not supported"	The given feature is not supported by the current version of the programming system.

3611

"The given compile directory '<name>' is invalid"	There is an invalid directory given in the 'Project', 'Options', 'Directories' for the compile files.

3612

"Maximum number of POUs (<number>) exceeded! Compile is aborted."	Too many POUs and data types are used in the project. Modify the maximum number of POUs in the Target Settings / Memory Layout.

3613

"Build canceled"	The compile process was cancelled by the user.

3614

"Project must contain a POU named '<name>' (main routine) or a taskconfiguration"	Create an init POU of type Program (e.g. PLC_PRG) or set up a task configuration.

3615

"<Name> (main routine) must be of type program"	An init POU (e.g. PLC_PRG) is used in the project which is not of type Program.

3616

"Programs musn't be implemented in external libraries"	The project which should be saved as an external library contains a program. This will not be available when the library is used.

3617

"Out of memory"	Increase the virtual memory capacity of your computer.

3618

"BitAccess not supported in current code generator!"	The code generator for the currently set target system does not support bit access on variables.

3619

"Object file '<name>' and library '<name>' have different versions!"	Make sure that for the library there are available matching versions of *.lib and *.obj resp. *.hex files. These files must have the very same time stamp.

3620

"The POU '<name>' must not be present inside a library"	You want to save the project as a library of version 2.1. In this version a library may not contain a PLC_PRG object. Use a different POU name.

3621

"Cannot write compile file '<name>'"	Probably in the path which is specified for the compile file there is already a file of the same name, which is "read only". Remove that file or change the access rights.

3622

"The symbol file '<name>' could not be created"	Probably in the path which is specified for the symbol file (usually project directory) there is already a file of the same name, which is "read only". Remove that file or change the access rights.

3623

"Cannot write boot project file '<name>'"	Probably in the path which is specified for the symbol file (target specific) there is already a file of the same name, which is "read only". Remove that file or change the access rights.

3624

"Target setting <targetsetting1>=<set value> not compatible with <targetsetting2>=<set value>"	Check and correct these settings in the Target settings dialogs (Resources tab).

3700

"POU with name '<name>' is already in library '<name>'"	A POU name is used in the project, which is already used for a library POU. Rename the POU.

3701

"Name used in interface is not identical with POU Name"	Use command ‘Project’ ‘Rename object’ to rename the POU in the object organizer, or change the name of the POU in the declaration window. There the POU name has to be placed next to one of the keywords PROGRAM, FUNCTION or FUNCTIONBLOCK.

3702

"Overflow of identifier list"	Maximum 100 identifiers can be entered in one variable declaration.

3703

"Duplicate definition of identifier '<Name>'"	Take care that there is only one identifier with the given name in the declaration part of the POU.

3704

"Data recursion: "<POU 0> -> <POU 1> -> .. -> <POU 0>"	An instance of a function block is used which calls itself.

3705

"<Name>: VAR_IN_OUT in Top-Level-POU not allowed, if there is no Task-Configuration"	Create a task configuration or make sure that there are no VAR_IN_OUT variables used in PLC_PRG.

3706

"Modifier 'CONSTANT' allowed for 'VAR', 'VAR_INPUT', 'VAR_EXTERNAL' and 'VAR_GLOBAL' only"	Constants cannot be declared for this type of variable.

3720

"Address expected after 'AT'"	Add a valid address after the keyword AT or modify the keyword.

3721

"Only 'VAR' and 'VAR_GLOBAL' can be located to addresses"	Put the declaration to a VAR or VAR_GLOBAL declaration area.

3722

"Only 'BOOL' variables allowed on bit addresses"	Modify the address or modify the type of the variable to which the address is assigned.

3726

"Constants can not be laid on direct addresses"	Modify the address assignment correspondingly.

3727

"No array declaration allowed on this address"	Modify the address assignment correspondingly.

3728

"Invalid address: '<address>'"	This address is not supported by the PLC configuration. Check the PLC configuration or modify the address.

3729

"Invalid type '<name>' at address: '<Name>' "	The type of this variable cannot be placed on the given address. Example: For a target system working with ‘alignment 2’ the following declaration is not valid: var1 AT %IB1:WORD; This error message also might indicate that an array is assigned to the address of a direct variable, which is not allowed.

3740

"Invalid type: '<Name>' "	An invalid data type is used in a variable declaration.

3741

"Expecting type specification"	A keyword or an operator is used instead of a valid type identifier.

3742

"Enumeration value expected"	In the definition of the enumeration type an identifier is missing after the opening bracket or after a comma between the brackets.

3743

"Integer number expected"	Enumerations can only be initialized with numbers of type INT.

3744

"Enum constant '<name>' already defined"	Check if you have followed the rules for the definition of enumeration values: ● Within one enum definition all values have to be unique. ● Within all global enum definitions all values have to be unique. ● Within all local enum definitions all values have to be unique.

3745

"Subranges are only allowed on Integers!" Subrange types can only be defined for integer data types.

3746

"Subrange '<name>' is not compatible with Type '<name>'" One of the limits set for the range of the subrange type is out of the range which is valid for the base type.

3747

"Unknown string length: '<name>'" There is a not valid constant used for the definition of the string length.

3748

"More than three dimensions are not allowed for arrays" More than the allowed three dimensions are given in the definition of an array. If applicable use an ARRAY OF ARRAY.

3749

"Lower bound '<name>' not defined" There is an undefined constant used to define the lower limit for a subrange or array type.

3750

"Upper bound '<name>' not defined" There is an undefined constant used to define the upper limit for a subrange or array type.

3751

"Invalid string length '<number of characters>'" The defined string length exceeds the maximum value which is defined for the currently set target system.

3752

"More than 9 dimensions are not allowed for nested arrays" An array can be 1- 2- or 3-dimensional ↬ *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*. The maximum number of dimensions reached by nesting of arrays is 9, e.g. "arr: ARRAY [0..2,0..2,0..2] OF ARRAY [0..2,0..2,0..2] OF ARRAY [0..2,0..2,0..2, 0..2] OF DINT". This maximum has been exceeded in the current error case. Reduce to a maximum of 9 dimensions.

3760

"Error in inital value" Use an initial value which corresponds to the type definition. To change the declaration you can use the declaration dialog for variables (Shift/F2 or 'Edit"Autodeclare').

3761

"'VAR_IN_OUT' variables must not have an initial value."

Remove the initialisation at the declaration of the VAR_IN_OUT variable.

3780

"'VAR', 'VAR_INPUT', 'VAR_OUTPUT' or 'VAR_IN_OUT' expected"

The first line following the name of a POU must contain one of these keywords.

3781

"'END_VAR' or identifier expected"

Enter a valid identifier or a END_VAR at the beginning of the given line in the declaration window.

3782

"Unexpected end"

In the declaration editor: Add keyword END_VAR at the end of the declaration part.

In the text editor of the programming part: Add an instruction which terminates the last instruction sequence (e.g. END_IF).

This error message already might be created together with error 3703, if there are two identical declarations at the end of the declaration part.

3783

"END_STRUCT' or identifier expected"

Ensure that the type declaration is terminated correctly.

3784

"The current target doesn't support attribute <attribute name>"

The target system does not support this type of variable (e.g. RETAIN, PERSISTENT).

3800

"The global variables need too much memory. Increase the available memory in the project options."

Increase the number of segments given in the settings in dialog 'Project', 'Options', 'Build'.

3801

"The variable '<name>' is too big. (<size> byte)"

The variable uses a type which is bigger than 1 data segment. The segment size is a target specific parameter and can be modified in the target settings/memory layout.

3802

"Out of retain memory. Variable '<name>', <number> bytes."	The memory space available for retain variables is exhausted. The size of the memory area can be set target-specific in the target settings / memory layout. If retain variables are used in a function block instance, the complete instance POU will be stored in the retain memory area.

3803

"Out of global data memory. Variable '<name>', '<number>' bytes."	The memory space available for global variables is exhausted. The size of the memory area can be set target-specific in the target settings / memory layout. If you do not find the settings field in the dialog, please contact your PLC manfacturer.

3804

"The current size of the persistent data description is <'number'> bytes and exceeds the maximum of <'number'> bytes."	Reduce the persistent data to the size allowed by the target system.

3820

"'VAR_OUTPUT' and 'VAR_IN_OUT' not allowed in functions"	In a function no output or in_output variables may be defined.

3821

"At least one input required for functions"	Add at least on input parameter for the function.

3840

"Unknown global variable '<name>'!"	In the POU a VAR_EXTERNAL variable is used, for which no global variable declared.

3841

"Declaration of '<name>' do not match global declaration!"	The type given in the declaration of the VAR_EXTERNAL variable is not the same as that in the global declaration.

3850

"Declaration of an unpacked struct '<name>' inside a packed struct '<name>' is not allowed!"	This structure definition leads to a misalignment in the memory. You must change the structure definition appropriately.

3900

"Multiple underlines in identifier"	Remove multiple underlines in the identifier name.

3901

"At most 4 numerical fields allowed in addresses"	There is a direct assignment to an address which has more than four levels, e.g. %QB0.1.1.0.1.

3902

"Keywords must be uppercase"	Use capital letters for the keyword or activate option 'Autoformat' in 'Project', 'Options'.

3903

"Invalid duration constant"	The notation of the constant does not comply with the IEC61131-3 format.

3904

"Overflow in duration constant"	The value used for the time constant cannot be represented in the internal format. The maximum value which is presentable is t#49d17h2m47s295ms.

3905

"Invalid date constant"	The notation of the constant does not comply with the IEC61131-3 format.

3906

"Invalid time of day constant"	The notation of the constant does not comply with the IEC61131-3 format.

3907

"Invalid date and time constant"	The notation of the constant does not comply with the IEC61131-3 format.

3908

"Invalid string constant"	The string constant contains an invalid character.

4000

"Identifier expected"	Enter a valid identifier at this position.

4001

"Variable '<Name>' not declared"	Declare variable local or global.

4010

"Type mismatch: Cannot convert '<Name>' to '<Name>'."	Check what data type the operator expects (Browse online help for name of operator) and change the type of the variable which has caused the error, or select another variable.

4011

"Type mismatch in parameter '<Name>' of '<Name>': Cannot convert '<Name>' to '<Name>'."	The data type of the actual parameter cannot be automatically converted to that of the formal parameter. Use a type conversion or use another variable type.

4012

"Type mismatch in parameter '<Name>' of '<Name>': Cannot convert '<Name>' to '<Name>'."	A value with the invalid type <Type2> is assigned to the input variable '<Name>'. Replace the variable or constant to one of type <Type1> or use a type conversion or a constant with type-prefix.

4013

"Type mismatch in output '<Name>' of '<Name>': Cannot convert '<Name>' to '<Name>'."	A value with the invalid type <Type2> is assigned to the output variable '<Name>'. Replace the variable or constant to one of type <Type1> or use a type conversion or a constant with type-prefix.

4014

"Typed literal: Cannot convert '<name>' to '<name>'"	The type of the constant is not compatible with the type of the prefix. Example: SINT#255

4015

"Data type '<name>' illegal for direct bit access"	Direct bit addressing is only allowed for Integer- and Bitstring datatypes and not for direct variables. You are using a variable var1 of type REAL/LREAL or a constant in bit access <var1>.<bit>, or you are trying a bit access on a direct variable.

4016

"Bit index '<number>' out of range for variable of type '<name>'"	You are trying to access a bit which is not defined for the data type of the variable.

4017

"'MOD' is not defined for 'REAL'"	The operator MOD can only be used for integer and bitstring data types.

4020

"Variable with write access or direct address required for 'ST', 'STN', 'S', 'R'"	Replace the first operand by a variable with write access.

4021

"No write access to variable '<name>' allowed"	Replace the variable by a variable with write access.

4022

"Operand expected"	Add an operand behind the command.

4023

"Number expected after '+' or '-'"	Enter a digit.

4024

"Expecting <Operator 0> or <Operator 1> or ... before <Name>'"	Enter a valid operand at the named position.

4025

"Expecting ':=' or '=>' before '<Name>'"	Enter one of the operators at the named position.

4026

"'BITADR' expects a bit address or a variable on a bit address"	Use a valid bit address (e.g. %IX0.1).

4027

"Integer number or symbolic constant expected"	Enter a integer number or the identifier of a valid constant.

4028

"'INI' operator needs function block instance or data unit type instance"	Check the data type of the variable for which the INI operator is used.

4029

"Nested calls of the same function are not possible."	In non-reentrant target systems and in simulation mode a function call may not contain a call to itself as a parameter.

Example:	`fun1(a,fun1(b,c,d),e);`
	Use an intermediate table.

4030

"Expressions and constants are not allowed as operands of 'ADR'"	Replace the constant or the expression by a variable or a direct address.

4031

"'ADR' is not allowed on bits! Use 'BITADR' instead."	Use BITADR. Please note that the BITADR function does not return a physical memory address.

4032

"'<number>' operands are too few for '<name>'. At least '<number>' are needed"	Check how many operands the named operator requires and add the missing operands.

4033

"'<number>' operands are too many for '<name>'. At least '<number>' are needed"	Check how many operands the named operator requires and remove the surplus operands.

4034

"Division by 0"	You are using a division by 0 in a constant expression. If you want to cause a runtime error then use, if applicable, a variable with the value 0.

4035

"ADR must not be applied on 'VAR CONSTANT' if 'replaced constants' is activated"	An address access on constants, for which the direct values are used, is not possible. If applicable, deactivate the option 'Replace Constants' in 'Project' 'Options' 'Build'.

4040

"Label '<name>' is not defined"	Define a label with the name <LabelName> or change the name <LabelName> to that of a defined label.

4041

"Duplicate definition of label '<name>'"	The label '<Name>' has multiple definitions in the POU. Rename the label or remove one of the definitions.

4042

"No more than <number> labels in sequence are allowed"	The number of jump labels is limited to '<number>'. Insert a dummy instruction.

4043

"Format of label invalid. A label must be a name optionally followed by a colon."	The label name is not valid or the colon is missing in the definition.

4050

"POU '%s' is not defined"	Define a POU with the name '<Name>' using the command ‘Project’ ‘Add Object’, or change '<Name>' to the name of a defined POU.

4051

"'%s' is no function"	Use instead of <name> a function name which is defined in the project or in the libraries.

4052

"'<name>' must be a declared instance of FB '<name>'"	Use an instance of data type '<Name>' which is defined in the project or change the type of <Instance name> to '<Name>'.

4053

"'<name>' is no valid box or operator"	Replace '<name>' by the name of a POU or an operator defined in the project.

4054

"POU name expected as parameter of 'INDEXOF'"	The given parameter is not a valid POU name.

4060

"'VAR_IN_OUT' parameter '<name>' of '<name>' needs variable with write access as input"	Variables with write access have to be passed to VAR_IN_OUT parameters, because a VAR_IN_OUT can be modified within the POU.

4061

"'VAR_IN_OUT' parameter '<name>' of '<name>' must be used."	A VAR_IN_OUT parameter must be passed a variable with write access, because a VAR_IN_OUT can be modified within the POU.

4062

"No external access to 'VAR_IN_OUT' parameter '<name>' of '<name>'."

VAR_IN_OUT parameter may be written or read only within the POU, because they are handed over by reference.

4063

"'VAR_IN_OUT' parameter '<name>' of '<name>' must not be used with bit addresses."

A bit address is not a valid physical address. Hand over a variable or a direct non-bit address.

4064

"'VAR_IN_OUT' must not be overwritten in local action call!"

Delete the parameters set for the VAR_IN_OUT variable in the local action call.

4070

"The POU contains a too complex expression"

Decrease nesting depth by dividing up the expression into several expressions. Use intermediate variables for this purpose.

4071

"Network too complex"

Divide up the network into several networks.

4072

"Inconsistent use of an action identifier in FB type ('<name>') and instance ('<name>')."

You have defined two actions of a function block FB: e.g. a1 and a2, but in the call of one of the actions in the FBD you are using a type (string within the box, e.g. fb.a1 different to that used in the instance-name (e.g. inst.a2, above box). Correct the name correspondingly into the name of the desired action.

4100

"'^' needs a pointer type"

You are trying to dereference a variable which is not declared as a pointer.

4110

"'[<index>]' needs array variable"

[<index>] is used for a variable which is not declared as an array with ARRAY OF.

4111

"Index expression of an array must be of type 'INT'"

Use an expression of the correct type or a type conversion.

4112

"Too many indexes for array"	Check the number of indices (1, 2, or 3) for which the array is declared and remove the surplus.

4113

"Too few indexes for array"	Check the number of indices (1, 2, or 3) for which the array is declared and add the missing ones.

4114

"One of the constant indices is not within the array range"	Make sure that the used indices are within the bounds of the array.

4120

"'.' needs structure variable"	The identifier on the left hand of the dot must be a variable of type STRUCT or FUNCTION_BLOCK or the name of a FUNCTION or a PROGRAM.

4121

"'<Name>' is not a component of <object name>"	The component '<Name>' is not included in the definition of the object <object name>.

4122

"'<name>' is not an input variable of the called function block"	Check the input variables of the called function block and change ‘<name>' to one of these.

4200

"'LD' expected"	Insert at least one LD instruction after the jump label in the IL editor.

4201

"IL Operator expected"	Each IL instruction must start with an operator or a jump label.

4202

"Unexpected end of text in brackets"	Insert a closing bracket after the text.

4203

"<name> in brackets not allowed"	The operator <name> is not valid in a IL bracket expression. Not valid are: 'JMP', 'RET', 'CAL', 'LDN', 'LD', 'TIME'.

4204

"Closing bracket with no corresponding opening bracket"	Insert an opening bracket or remove the closing one.

4205

"No comma allowed after ')'"	Remove comma after closing bracket.

4206

"Label in brackets not allowed"	Shift jump label so that it is outside of the brackets.

4207

"'N' modifier requires operand of type 'BOOL', 'BYTE', 'WORD' or 'DWORD'"	The N modifier requires a data type, for which a boolean negation can be executed.

4208

"Conditional Operator requires type 'BOOL'"	Make sure that the expression gives out a boolean result or use a type conversion.

4209

"Function name not allowed here"	Replace the function call by a variable or a constant.

4210

"'CAL', 'CALC' and 'CALN' require a function block instance as operand"	Declare an instance of the function block which you want to call.

4211

"Comments are only allowed at the end of line in IL"	Shift the comment to the end of the line or to an extra line.

4212

"Accumulator is invalid before conditional statement"	The accumulator is not defined. This happens if an instruction is preceeding which does not submit a result (e.g. 'CAL').

4213

"'S' and 'R' require 'BOOL' operand"	Use a boolean variable at this place.

4250

"Another 'ST' statement or end of POU expected"	The line does not start with a valid ST instruction.

4251

"Too many parameters in function '<name>'"	There are more parameters given than are declared in the definition of the function.

4252

"Too few parameters in function '<name>'"	There are fewer parameters given than are declared in the definition of the function.

4253

"'IF' or 'ELSIF' require 'BOOL' expression as condition"	Make sure that the condition for IF or ELSIF is a boolean expression.

4254

"'WHILE' requires 'BOOL' expression as condition"	Make sure that the condition following the ‘WHILE’ is a boolean expression.

4255

"'UNTIL' requires 'BOOL' expression as condition"	Make sure that the condition following the ‘UNTIL’ is a boolean expression.

4256

"'NOT' requires 'BOOL' operand"	Make sure that the condition following the ‘NOT’ is a boolean expression.

4257

"Variable of 'FOR' statement must be of type 'INT'"	Make sure that the counter variable is of an integer or bitstring data type (e.g. DINT, DWORD).

4258

"Expression in 'FOR' statement is no variable with write access"	Replace the counter variable by a variable with write access.

4259

"Start value in 'FOR' statement is no variable with write access"	The start value in the 'FOR' instruction must be compatible to the type of the counter variable.

4260

"End value of 'FOR' statement must be of type 'INT'"	The end value in the 'FOR' instruction must be compatible to the type of the counter variable.

4261

"Increment value of 'FOR' statement must be of type 'INT'"	The incremental value in the 'FOR' instruction must be compatible to the type of the counter variable.

4262

"'EXIT' outside a loop"	Use 'EXIT' only within 'FOR', 'WHILE' or 'UNTIL' instructions.

4263

"Expecting Number, 'ELSE' or 'END_CASE'"	Within a ‘CASE' expression you can only use a number or a 'ELSE' instruction or the ending instruction 'END_CASE'.

4264

"'CASE' requires selector of an integer type"	Make sure that the selector is of an integer or bitstring data type (e.g. DINT, DWORD).

4265

"Number expected after ','"	In the enumeration of the CASE selectors there must be inserted a further selector after a comma.

4266

"At least one statement is required"	Insert an instruction, at least a semicolon.

4267

"Function block call requires function block instance"	The identifier in the function block call is no instance. Declare an instance of the desired function block or use the name of an already defined instance.

4268

"Expression expected"	Insert an expression.

4269

"'END_CASE' expected after 'ELSE'-branch"	Terminate the 'CASE' instruction after the 'ELSE' part with an 'END_CASE'.

4270

"'CASE' constant '<name>' already used"	A 'CASE' selector may only be used once within a ‘CASE' instruction.

4271

"The lower border of the range is greater than the upper border."	Modify the area bounds for the selectors so that the lower border is less than the upper border.

4272

"Expecting parameter '<name>' at place <position> in call of '<name>'!"	You can edit a function call so that the parameter names are contained, not only the parameter values. But nevertheless the position (sequence) of the parameters must be the same as in the function definition.

4273

"Parts of the 'CASE'-Range '<range>' already used in Range '<range>'"	Make sure that the areas for the selectors which are used in the CASE instruction don’t overlap.

4274

"Multiple 'ELSE' branch in 'CASE' statement"	A CASE instruction may not contain more than one 'ELSE' instruction.

4300

"Jump requires 'BOOL' as input type"	Make sure that the input for the jump or the RETURN instruction is a boolean expression.

4301

"POU '<name>' need exactly <number> inputs"	The number of inputs does not correspond to the number of VAR_INPUT and VAR_IN_OUT variables which is given in the POU definition.

4302

"POU '<name>' need exactly %d outputs"	The number of outputs does not correspond to the number of VAR_OUTPUT variables which is given in the POU definition.

4303

"'<name>' is no operator"	Replace '<name>' with a valid operator.

4320

"Non-boolean expression '<name>' used with contact" The switch signal for a contact must be a boolean expression.

4321

"Non-boolean expression '<name>' used with coil" The output variable of a coil must be of type BOOL.

4330

"Expression expected at input 'EN' of the box '<name>' " Assign an input or an expression to the input EN of POU '<name>'.

4331

"Expression expected at input '<number>' of the box '<Name>' " The input <number> of the operator POU is not assigned.

4332

"Expression expected at input '<name>' of the box '<Name>'" The input of the POU is of type VAR_IN_OUT and is not assigned.

4333

"Identifier in jump expected" The given jump mark is not a valid identifier.

4334

"Expression expected at the input of jump" Assign a boolean expression to the input of the jump. If this is TRUE, the jump will be executed.

4335

"Expression expected at the input of the return" Assign a boolean expression to the input of the RETURN instruction. If this is TRUE, the jump will be executed.

4336

"Expression expected at the input of the output" Assign a suitable expression to the output box.

4337

"Identifier for input expected" Insert a valid expression or identifier in the input box.

4338

"Box '<name>' has no inputs"	None of the inputs of the operator POU '<name>' have been assigned a valid expression.

4339

"Type mismatch at output: Cannot convert '<name>' to '<name>'	The type of the expression in the output box is not compatible with that of the expression which should be assigned to it.

4340

"Jump requires 'BOOL' as input type"	Make sure that the input for the jump is a boolean expression.

4341

"Return needs a boolean input"	Make sure that the input for the RETURN instruction is a boolean expression.

4342

"Expression expected at input 'EN' of the box '<name>'"	Assign a valid boolean expression to the EN input of the box.

4343

"Values of Constants: '<name>'"	Input '<name>' of box '<name>' is declared as VAR_INPUT CONSTANT. But to this POU box an expression has been assigned in the dialog 'Edit Parameters' which is not type compatible.

4344

"'S' and 'R' require 'BOOL' operand"	Insert a valid boolean expression after the Set or Reset instruction.

4345

"Invalid type for parameter '<name>' of '<name>': Cannot convert '<type>' to '<type>'"	An expression is assigned to input '<name>' of POU box '<name>' which is not type compatible.

4346

"Not allowed to use a constant as an output"	You can only assign an output to a variable or a direct address with write access.

4347

"'VAR_IN_OUT' parameter needs variable with write access as input"	Only variables with write access can be passed to VAR_IN_OUT parameters, because these can be modified within the POU.

4348

"Invalid program name '<name>'. A variable with the same name exists already."	You have inserted a program box in the CFC editor which has the same name as a (global) variable already existing in your project. You must rename accordingly.

4349

Input or output in POU <name> has been deleted: Check all connections to the box. This error message disappears only after the CFC was edited	Error in CFC POU.

4350

"An SFC-Action can not be accessed from outside!"	SFC actions only can be called within the SFC POU in which they are defined. But this error also will be dumped if you call an action from within a SFC POU, which is allowed, but are not using IEC steps while the iecsfc.lib is still included in your project. In this case please remove the library in the library manager and rebuild the project.

4351

"Step name is no identifier: '<name>'"	Rename the step or choose a valid identifier as step name.

4352

"Extra characters following valid step name:'<Name>'"	Remove the invalid characters in the step name.

4353

"Step name duplicated: '<Name>'"	Rename one of the steps. This error also will be generated if a step has the same name as a non-boolean variable.

4354

"Jump to undefined Step: '<Name>'"	Choose an existing step name as a target for the jump, or insert a step with name '<name>'.

4355

"A transition must not have any side effects (Assignments, FB-Calls etc.)"	A transition must be a boolean expression.

4356

"Jump without valid Step Name: '<Name>'"	Use a valid identifier as the target of the jump.

4357

"IEC-Library not found"	Check whether the library iecsfc.lib is inserted in the library manager and whether the library paths defined in ‘Project’ ‘Options’ ‘Paths’ are correct.

4358

"Action not declared: '<name>'"	Make sure that in the object organizer the action of the IEC step is inserted below the SFC POU, and that in the editor the action name is inserted in the box on the right hand of the qualifier.

4359

"Invalid Qualifier: '<name>'"	In the box on the left hand of the action name enter a qualifier for the IEC action.

4360

"Time Constant expected after qualifier '<name>'"	Enter next to the box on the left hand of the action name a time constant behind the qualifier.

4361

"'<name>' is not the name of an action"	Enter next to the box on the right hand of the qualifier the name of an action or the name of a variable which is defined in the project.

4362

"Nonboolean expression used in action: '<name>'"	Insert a boolean variable or a valid action name.

4363

"IEC-Step name already used for variable: '<Name>'"	Please rename the step or the variable.

4364

"A transition must be a boolean expression"	The result of the transition expression must be of type BOOL.

4365

"Time constant expected after qualifier '<name>'"	Open dialog 'step attributes' for the step '<name>' and enter a valid time variable or time constant.

4366

"The label of the parallel branch is no valid identifier: '<Name>'"	Enter a valid identifier next to the triangle which marks the jump label.

4367

"The label '<name>' is already used"	There is already a jump label or a step with this name. Please rename correspondingly.

4368

"Action '<name>' is used in multiple step chains, where one is containing the other!"	The action '<name>' is used in the POU as well as in one or several actions of the POU.

4369

"Exactly one network required for a transition"	There are used several FBD or LD networks for a transition. Please reduce to 1 network.

4370

"Additional lines found after correct IL-transition"	Remove the unnecessary lines at the end of the transition.

4371

"Invalid characters following valid expression: '<name>"	Remove the unnecessary characters at the end of the transition.

4372

"Step '<name>': Time limit needs type 'TIME'"	Define the time limits of the step in the step attributes by using a variable of type TIME, or by a time definition in correct format (e.g "t#200ms").

4373

"IEC-actions are only allowed with SFC-POUs"	There is an action assigned to a non-SFC-POU (see the Object Organizer), which is programmed in SFC and which contains IEC actions. Replace this action by one which contains no IEC actions.

4374

"Step expected instead of transition '<name>'"	The SFC POU is corrupt, possibly due to any export-import actions.

4375

"Transition expected instead of step '<name>'"	The SFC POU is corrupt, possibly due to any export-import actions.

4376

"Step expected after transition '<name>'"	The SFC POU is corrupt, possibly due to any export-import actions.

4377

"Transition expected after step '<name>'"	The SFC POU is corrupt, possibly due to any export-import actions.

4400

"Import / conversion of POU '<name>' contains errors resp. is not complete."	The POU cannot be converted to IEC 61131-3 completely.

4401

"S5 time constant <number> seconds is too big (max. 9990s)."	There is no valid BCD coded time in the accumulator.

4402

"Direct access only allowed on I/Os."	Make sure that you only access variables which are defined as input or output.

4403

"STEP5/7 instruction invalid or not convertible to IEC 61131-3."	Some STEP5/7 commands are not convertible to IEC 61131-3, e.g. CPU commands like MAS.

4404

"STEP5/7 operand invalid or not convertible to IEC 61131-3."	Some STEP5/7 operands are not convertible to IEC 61131-3 or an operand is missing.

4405

"Reset of a STEP5/7 timer cannot be converted into IEC 61131-3."	The corresponding IEC timer has no reset input.

4406

"STEP5/7 counter constant out of range (max. 999)."	There is no valid BCD coded counter constant in the accu.

4407

"STEP5 instruction not convertible to IEC 61131-3."	Some STEP5/7 instructions cannot be converted to IEC 61131-3, e.g. DUF.

4408

"Bit access of timer or counter words not convertible into IEC 61131-3."	Special timer/counter commands are not convertible into IEC 61131-3.

4409

"Contents of ACCU1 or ACCU2 undefined, not convertible into IEC 61131-3."	A command, which connects the both accus, cannot be converted because the accu values are not defined.

4410

"Called POU not in project."	Import the called POU.

4411

"Error in global variable list."	Please check the SEQ file.

4413

"Error in format of line in data block"	In the code which should be imported there is an erroneous date.

4414

"FB/FX name missing."	In the original S5D file the symbolic name of an (extended) POU is missing.

4415

"Instruction after block end not allowed."	A protected POU cannot get imported.

4416

"Invalid command"	The S5/S7 command cannot be disassembled.

4417

"Comment not closed"	Close the comment with "*)".

4418

"FB/FX-Name too long (max. 8 characters)"	The symbolic name of an (extended) POU is too long.

4419

"Expected format of line ""(* Name: <FB/FX-Name> *)"" "	Correct the line correspondingly.

4420

"Name of FB/FX parameter missing"	Check the POUs.

4421

"Type of FB/FX parameter invalid"	Check the POUs.

4422

"Type of FB/FX parameter missing"	Check the POUs.

4423

"Invalid FB/FX call parameter"	Check the interface of the POU.

4424

"Warning: FB/FX for call either missing or parameters invalid or has '0' parameters"	The called POU is not imported yet, is not correct, or has no parameters (in the last case you can ignore the error message).

4425

"Definition of label missing"	The target (label) of the jump is not defined.

4426

"POU does not have a valid STEP 5 block name, e.g. PB10"	Modify the POU name.

4427

"Timer type not declared"	Add a declaration of the timer in the global variables list.

4428

"Maximum number of open STEP5 brackets exceeded"	You may not use more than seven open brackets.

4429

"Error in name of formal parameter"	The parameter name may not exceed four characters.

4430

"Type of formal parameter not IEC-convertible"	In IEC 61131-3 Timer, counter and POUs cannot be converted as formal parameters.

4431

"Too many 'VAR_OUTPUT' parameters for a call in STEP5 STL"	A POU may not contain more than 16 formal parameters as outputs.

4432

"Labels within an expression are not allowed"	In IEC 61131-3 jump labels may not be inserted at any desired position.

4434

"Too many labels"	A POU may not contain more than 100 labels.

4435

"After jump / call, a new expression must start"	After jump or call a Load command LD must follow.

4436

"Bit result undefined, not convertible to IEC 61131-3."	The command which is used by VKE cannot get converted, because the value of the VKE is not known.

4437

"Type of instruction and operand are not compatible"	A bit command is used for a word operand or the other way round.

4438

"No data block opened (insert instruction C DB before)"	Insert a "A DB".

4500

"Unrecognized variable or address"	The watch variable is not declared within the project. By pressing <F2> you get the input assistant which lists the declared variables.

4501

"Extra characters following valid watch expression"	Remove the surplus signs.

4520

"Error in Pragma: Flag expected before '<Name>'!"	The pragma is not correct. Check whether ‘<Name>' is a valid flag.

4521

"Error in Pragma: Unexpected element '<Name>'!"	Check whether pragma is composed correctly.

4522

"'flag off' pragma expected!"	Pragma has not been terminated, insert a 'flag off' instruction.

4523

"Pragma {<Pragma-name>} not allowed in interface of type '<Name>'"	The Pragma cannot be used at this location, see 'Pragmas, overview' for the correct use of pragmas ↬ *Chapter 1.4.1.3.10.1 “Pragmas, overview” on page 309*.

4550

"Index out of defined range : Variable OD "number>, Line <line number>."	Ensure that the index is within the area which is defined in the target settings/networkfunctionality.

4551

"Subindex out of defined range : Variable OD "number>, Line <line number>."	Ensure that the subindex is within the area which is defined in the target settings/network functionality.

4552

"Index out of defined range : Parameter OD "number>, Line <line number>."	Ensure that the index is within the area which is defined in the target settings/network functionality.

4553

"Subindex out of defined range : Parameter OD "number>, Line <line number>."	Ensure that the subindex is within the area which is defined in the target settings/network functionality.

4554

"Variable name invalid: Variable OD <number>, Line <line number>."	Enter a valid project variable in the field 'variable'. Use the syntax <POU name>.<variable name>, or for global variables .<variable name>

4555

"Empty table-entry, input not optional: Parameter OD <number>, Line <line number>	You must make an entry in this field.

4556

"Empty table-entry, input not optional: Variable OD <number>, Line <number>"	You must make an entry in this field.

4557

"The required parameter memory is too large"	The maximum size of data which can be loaded via parameter lists of type Parameters to the controller has been exceeded. This size is defined by the target system. Information on the data size is displayed in the message window at compilation. Reduce the parameter lists size.

4558

"The required variable memory is too large"	The maximum size of data which can be loaded via parameter lists of type Variables to the controller has been exceeded. This size is defined by the target system. Information on the data size is displayed in the message window at compilation. Reduce the parameter lists size.

4560

"Invalid value: Dictionary '<Name>', column '<Name>', line '<line number>'"	Check this entry. It depends on the currently used column (attribute) definition which entries are valid for this field. This definition is given by the target-specific XML description file of the Parameter Manager, or by the standard settings which will be used if there is no description file.

4561

"Column not defined: '<Name>'"	Entries in a column of the parameter list refer to another column which is not defined. The column definitions are given by the description file (XML) of the Parameter Manager for the current target. If a description file is not available, standard settings are used.

4562

"Index/subindex used already: Dictionary '<Name>', line '<Line Number>'"	The Index/Subindex-combination must be unique throughout all parameter lists, because it can be used for parameter access. Correct the indices correspondingly.

4563

"Identifier '<Name>' used already: Dictionary '<Name>', line '<Line Number>'"	The name must be unique throughout all parameter lists, because it can be used for parameter access.

4564

"Index '<Name>' is out of range: Dictionary '<Name>', line '<Line Number>'"	Enter an index which is within the range defined in the target settings, category network functionality in field 'Index range...' for the respective list types (Variables, Parameters, Mappings).

4565

"Subindex '<Name>' is out of range: Dictionary '<Name>', line '<Line Number>'"	Enter a subindex which is within the range defined in the target settings, category network functionality in field 'SubIndex range'.

4566

"An error occurred during import of the parameter manager"	You have imported an export file which contains erroneous information on the Parameter Manager. Check the *.exp file.

4600

"Network variables: '<name>' expression is not from type bool!"	Make sure that the variable defined in the properties dialog of the network variables list at option 'Transmit on event' is of type BOOL.

4601

"Network variables '<name>': No cyclic or freewheeling task for network variable exchange found"

There is no cyclic or free-wheeling task or PLC_PRG in the project where the network variables of type CAN or UDP of the given list are used (only declaration is not sufficient!). You must take care that the variables are used in an appropriate task or in PLC_PRG. If you want to use them in several tasks, note that at data exchange the task with the highest priority will be regarded.

4602

"'<name of network variables list>': The object uses UDP port '<port number>' instead of '<port number>'"

In the Settings 'Create a global variable list' of the named network variables list a port number is used which is not the same as that which is used in the first network variables list found in the global variables folder ↳ *Chapter 1.4.1.4.1.3.1 “Create a global variable list” on page 358*. Take care that all network variables lists are using the same port!

4604

"Network variables '<name>': Base identifier has been used more than once."

The same COB-ID is used in the configuration settings of multiple network variables lists. Assign unique IDs.

4605

"Network variables '<name>': Duplicate CAN COB id."

In the configuration of a network variables list a COB-ID is used which is also specified in the CAN PLC Configuration. Assign unique IDs.

4620

Unused variables have been found in the project ↳ *Chapter 1.4.1.2.3.38 “Unused variables” on page 248*. See the description for command 'Project' 'Check' ↳ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248*.

4621

There are overlaps at the assignment of variables to memory areas via the "AT"-declaration ↳ *Chapter 1.4.1.2.3.39 “Overlapping memory areas” on page 249*. See the description of command 'Project' 'Check' ↳ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248*.

4622

IEC addresses assigned to the same memory area are referenced in more than one task. See the description of command 'Project' 'Check' ↳ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248* ↳ *Chapter 1.4.1.2.3.41 “Concurrent access” on page 249*.

4623

The project gains write access to the same memory area at more than one place. See the description of command ' ↳ *Chapter 1.4.1.2.3.37 “'Project' 'Check'” on page 248*' ↳ *Chapter 1.4.1.2.3.40 “Multiple write acces on output” on page 249*'.

4650

"AxisGroup '<Name>': Task '<Name>' does not exist."	In the PLC Configuration in the definition of the axis group (dialog 'Module parameters', column 'Value') there is a name defined for the task which is controlling the data transfer of this axis group. This task name is not known in the Task Configuration. Correct Task Configuration or PLC Configuration correspondingly.

4651

"AxisGroup '<Name>': Cycletime (dwCycle) not set."	In dialog 'Module parameters' of the axis group enter a value for the cycle time (dwCycle).

4652

"Drive '<name>': wDriveID exists already in this axis group"	In the PLC Configuration within the same axis group there is already a drive with the same Drive ID. Select the drive in the configuration tree and define an ID in configuration dialog 'Drive' which is unique within the axis group.

4656

"Drive '<name>': Scaling factor must not be 0."	Open the PLC Configuration and check the Scale entries in the configuration dialog of the drive. All fields must have entries not equal "0".

4670

"CNC program '<Name>': Global variable '<Name>' not found."	In the CNC program a global variable is used (e.g. $glob_var$), which is not defined in the project. Add the appropriate declaration or correct the assignment to the variable in the CNC program.

4671

"CNC program '<Name>': Variable '<Name>' has an incompatible type."	There is a variable assigned in an instruction of the CNC program, which is declared of a data type which is not valid in this place. Use another variable or correct the type specification.

4685

"CAM '<Name>': CAM table type unknown."	Check the data type which is specified in the CAM Editor dialog "Compile options.." for the equidistant or element optimized point table.

4686

"CAM '<Name>': CAM point exceeds datatype range."	CAM points are used which are out of the data range specified for the point table. For the current range definition see dialog 'Compile options..' in the CAM-Editor.

4700

"'<Number>' ('<Name>'): Watch expression '<Name>' is not a numeric variable."	In the configuration of the visualization a variable is used which is not a number, as required in this place (e.g. at the configuration of XOffset or Angle values etc.).

4701

"'<Name>' ('<Number>'): Watch expression '<name>' is not of type BOOL."	In the configuration of the visualization a variable is used which is not of type BOOL, as required in this place.

4702

"'<Name>' ('<Number>'): Watch expression '<name>' is not of type STRING."	The visualization contains a variable which is not of type STRING although this is required in this place (e.g. at the tooltip configuration).

4703

"'<Name>' ('<Number>'): Invalid watch expression '<Name>'"	The visualization contains an invalid variable.

4704

"'<Name>'('<Number>'): Invalid initial value in watchlist '<Name>'."	In this watchlist, used in a visualization (INTERN command in category Input), there is a erroneous initial value. Check the used list.

4705

"'<name>' ('<number>'): No valid alarm group assigned to alarm table."	Enter a valid alarm group in the configuration dialog of the alarm table (category Alarm table).

4706

"'<name>' ('<number>'): Use of alarm tables requires the target setting 'Alarmhandling in controller' to be activated."	Open the target settings in tab 'Resources', and in dialog 'Visualization' activate option 'Alarmhandling in the PLC'. Otherwise the alarm table element will not work in the target visualization which is currently activated also in the target settings.

4707

"'<name>' ('<number>'): Alarm tables are not supported by the	The target system does not support the processing of alarms (target setting 'Alarmhandling in the PLC' cannot be activated). Thus for running a target visualization (currently activated in the target settings in tab 'Visualization') the alarm table elements must be removed from the visualization.

current target. Please remove these elements from the target visualization."

4708

"'<name>' ('<number>'): Use of trends requires the target setting 'Trend recording in controller' to be activated."

Open the target settings in tab 'Resources' and in dialog 'Visualization' activate option 'Store trend data in the PLC'. Otherwise the trend element will not work in the target visualization which is currently activated also in the target settings.

4709

"'<name>' ('<number>'): Trends are not supported by the current target. Please remove these elements from the target visualization."

The target system does not support the processing of trend data (target setting 'Store trend data in the PLC' cannot be activated). Thus for running a target visualization (currently activated in the target settings in tab 'Visualization') the trend elements must be removed from the visualization.

4712

"'<Name>'('<number>'): The selected target does not support the output of passwords"

The text output in the visualization element is configured to work via dynamic texts or unicode texts. Additionally the option "Hidden" is activated in the "Input" category. This is not supported by the target visualization.

4900

"Invalid type for conversion"

You are using a type conversion which is not supported by the currently chosen code generator.

4901

"Internal error: Overflow in array access!"

The array bounds are too large for a 32-bit-variable. Reduce the array index range.

5100

"<Name> (<Number>): Expression too complex. No more registers available"

The named expression is to complex to be handled by the available registers. Please try to reduce the expression by using interim variables.

1.4.2 Libraries

1.4.2.1 Standard.library

Elements of the Standard.lib

in ST	in AWL	Description
LEN(in)	LEN	String length of operand in
LEFT(str,size)	LEFT	Left inital string of given size of string str

in ST	in AWL	Description
RIGHT(str,size)	RIGHT	Right initial string of given size of string str
MID(str,size,pos)	MID	Partial string of str of given size at position pos
CONCAT('str1','str2')	CONCAT 'str2'	Concatenation of two subsequent strings
INSERT('str1','str2',pos)	INSERT 'str2',p	Insert string str1 in String str2 at position pos
DELETE('str1',len,pos)	DELETE len,pos	Delete partial string (length len), start at position pos of str1
REPLACE('str1','str2',len,pos)	REPLACE 'str2',len,pos	Replace partial string of lenght len by str2, start at position pos of str1
FIND('str1','str2')	FIND 'str2'	Search for partial string str2 in str1
SR	SR	Bistable FB is set dominant
RS	RS	Bistable FB is set back
SEMA	SEMA	FB: Software Semaphor (interruptable)
R_TRIG	R_TRIG	FB: rising edge is detected
F_TRIG	F_TRIG	FB: falling edge is detected
CTU	CTU	FB: Counts upv
CTD	CTD	FB: Counts down
CTUD	CTUD	FB: Counts up and down
TP	TP	FB: trigger
TON	TON	FB: Timer On-Delay
TOF	TOF	FB: Timer Off-Delay
RTC	RTC	FB: Real-time clock

1.4.2.1.1 String functions

LEN

Provided by standard.lib.

Returns the length of a string. Input STR is of type STRING, the return value of the function is type INT.

Example in IL

```
LD 'SUSI'
LEN
ST VarINT1 (* Result is 4 *)
```

Example in FBD

Example in ST VarSTRING1 := LEN ('SUSI');

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

LEFT

Provided by standard.lib.

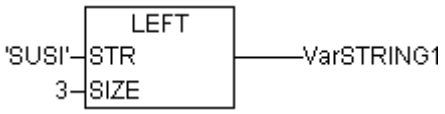
Left returns the left, initial string for a given string. Input STR is type STRING, SIZE is of type INT, the return value of the function is type STRING.

LEFT (STR, SIZE) means: Take the first SIZE character from the left in the string STR.

Example in IL

```
LD   'SUSI'
LEFT 3
ST   VarSTRING1 (* Result is 'SUS' *)
```

Example in FBD



Example in ST VarSTRING1 := LEFT ('SUSI',3);

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

RIGHT

Provided by standard.lib.

Right returns the right, initial string for a given string.

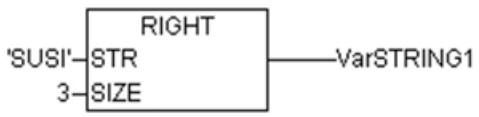
RIGHT (STR, SIZE) means: Take the first SIZE character from the right in the string STR.

Input STR is of type STRING, SIZE is of type INT, the return value of the function is of type STRING.

Example in IL

```
LD 'SUSI'
RIGHT 3
ST VarSTRING1 (* Result is 'USI' *)
```

Example in FBD



Example in ST VarSTRING1 := RIGHT ('SUSI',3);

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

MID

Provided by standard.lib.

Mid returns a partial string from within a string.

Input STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

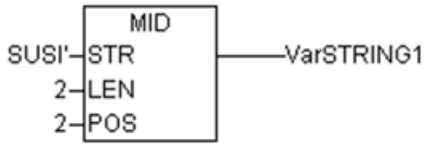
MID (STR, LEN, POS) means: Retrieve LEN characters from the STR string beginning with the character at position POS.

Example in IL LD 'SUSI'

MID 2,2

ST VarSTRING1 (* Result is 'US' *)

Example in FBD



Example in ST VarSTRING1 := MID ('SUSI',2,2);

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

CONCAT

Provided by standard.lib.

Concatenation (combination) of two strings.

The input variables STR1 and STR2 as well as the return value of the function are type STRING.

Example in IL LD 'SUSI'

CONCAT 'WILLI'

ST VarSTRING1 (* Result is 'SUSIWILLI' *)

Example in FBD

Example in ST VarSTRING1 := CONCAT ('SUSI','WILLI');

The CONCAT function does not work, if nested over more than five levels.

String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.

INSERT

Provided by standard.lib.

INSERT inserts a string into another string at a defined point.

The input variables STR1 and STR2 are type STRING, POS is type INT and the return value of the function is type STRING.

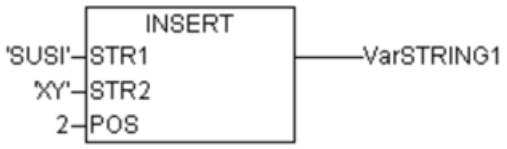
INSERT(STR1, STR2, POS) means: insert STR2 into STR1 after position POS.

Example in IL LD 'SUSI'

INSERT 'XY',2

ST VarSTRING1 (* Result is 'SUXYSI' *)

Example in FBD



Example in ST VarSTRING1 := INSERT ('SUSI','XY',2);

String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.

DELETE

Provided by standard.lib.

DELETE removes a partial string from a larger string at a defined position.

The input variable STR is type STRING, LEN and POS are type INT, the return value of the function is type STRING.

DELETE(STR, L, P) means: Delete L characters from STR beginning with the character in the P position.

Example in IL

```
LD 'SUXYSI'
DELETE 2,3
ST Var1 (* Result is 'SUSI' *)
```

Example in FBD

Example in ST

```
Var1 := DELETE ('SUXYSI',2,3);
```

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

REPLACE

Provided by standard.lib.

REPLACE replaces a partial string from a larger string with a third string.

The input variable STR1 and STR2 are type STRING, LEN and POS are type INT, the return value of the function is type STRING.

REPLACE(STR1, STR2, L, P) means: Replace L characters from STR1 with STR2 beginning with the character in the P position.

Example in IL

```
LD 'SUXYSI'
REPLACE 'K',2,2
ST VarSTRING1 (* Result is 'SKYSI' *)
```

Example in FBD

Example in ST

```
VarSTRING1 := REPLACE ('SUXYSI','K',2,2);
```

> *String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.*

FIND

Provided by standard.lib.

FIND searches for a partial string within a string.

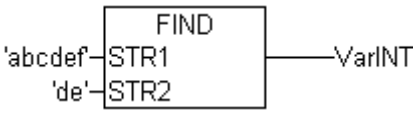
The input variable STR1 and STR2 are type STRING, the return value of the function is type STRING.

FIND(STR1, STR2) means: Find the position of the first character where STR2 appears in STR1 for the first time. If STR2 is not found in STR1, then OUT:=0.

Example in IL

```
LD   'abcdef'
FIND 'de'
ST   VarINT1 (* Result is '4' *)
```

Example in FBD



Example in ST

```
arINT1 := FIND ('abcdef','de');
```

Please note: String functions are not "thread safe": When using tasks, string functions may only be used in a single task. If the same function is used in different tasks, there is a danger of overwriting.

1.4.2.1.2 Bistable function blocks

SR

Provided by standard.lib.

Making bistable function blocks dominant:

Q1 = SR (SET1, RESET) means:

Q1 = (NOT RESET AND Q1) OR SET1

The input variables SET1 and RESET as well as the output variable Q1 are type BOOL.

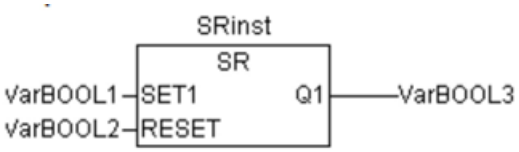
Declaration example

```
SRInst : SR ;
```

Example in IL

```
CAL SRInst(SET1 := VarBOOL1, RESET := VarBOOL2)
LD SRInst.Q1 ST VarBOOL3
```

Example in FBD



Example in ST

```
SRInst(SET1:= VarBOOL1 , RESET:=VarBOOL2 );
VarBOOL3 := SRInst.Q1 ;
```

RS

Provided by standard.lib.

Resetting bistable function blocks

Q1 = RS (SET, RESET1) means:

Q1 = NOT RESET1 AND (Q1 OR SET)

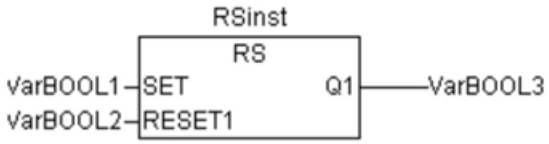
The input variables SET and RESET1 as well as the output variable Q1 are type BOOL.

Declaration example	RSInst : RS ;

Example in IL

```
CAL RSInst(SET:= VarBOOL1,RESET1:=VarBOOL2)
LD RSInst.Q1
ST VarBOOL3
```

Example in FBD



Example in ST

```
RSInst(SET:= VarBOOL1 , RESET1:=VarBOOL2 );
VarBOOL3 := RSInst.Q1 ;
```

SEMA

Provided by standard.lib.

A Software Semaphore (Interruptible)

BUSY = SEMA(CLAIM, RELEASE) means:

BUSY := X;

IF CLAIM THEN X:=TRUE;

ELSE IF RELEASE THEN BUSY := FALSE; X:= FALSE;

END_IF

X is an internal BOOL variable that is FALSE when it is initialized.

The input variables CLAIM and RELEASE as well as the output variable BUSY are type BOOL.

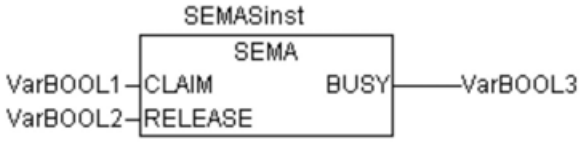
If BUSY is TRUE when SEMA is called up, this means that a value has already been assigned to SEMA (SEMA was called up with CLAIM = TRUE). If BUSY is FALSE, SEMA has not yet been called up or it has been released (called up with RELEASE = TRUE).

Declaration example	SEMAInst : SEMA ;

Example in IL

```
CAL SEMAInst(CLAIM:=VarBOOL1,RELEASE:=VarBOOL2)
LD SEMAInst.BUSY
ST VarBOOL3
```

Example in FBD



Example in ST

```
SEMAInst(CLAIM:= VarBOOL1 , RELEASE:=VarBOOL2 );
VarBOOL3 := SEMAInst.BUSY;
```

1.4.2.1.3 Trigger

R_TRIG

Provided by standard.lib.

The function block R_TRIG detects a rising edge.

```
FUNCTION_BLOCK R_TRIG
VAR_INPUT
 CLK : BOOL;
END_VAR
VAR_OUTPUT
 Q : BOOL;
END_VAR
VAR
 M : BOOL := FALSE;
END_VAR
 Q := CLK AND NOT M;
 M := CLK;
```

The output Q and the help variable M will remain FALSE as long as the input variable CLK is FALSE. As soon as CLK returns TRUE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has falling edge followed by an rising edge.

Declaration example	RTRIGInst : R_TRIG ;

Example in IL

```
CAL RTRIGInst(CLK := VarBOOL1)
LD  RTRIGInst.Q
ST  VarBOOL2
```

Example in FBD

Example in ST

```
RTRIGInst(CLK:= VarBOOL1);
```

VarBOOL2 := RTRIGInst.Q;

F_TRIG

Provided by standard.lib.

The function block F_TRIG a falling edge.

FUNCTION_BLOCK F_TRIG

VAR_INPUT

CLK: BOOL;

END_VAR

VAR_OUTPUT

Q: BOOL;

END_VAR

VAR

M: BOOL := FALSE;

END_VAR

Q := NOT CLK AND NOT M;

M := NOT CLK;

The output Q and the help variable M will remain FALSE as long as the input variable CLK returns TRUE. As soon as CLK returns FALSE, Q will first return TRUE, then M will be set to TRUE. This means each time the function is called up, Q will return FALSE until CLK has a rising followed by a falling edge.

Declaration example	FTRIGInst : F_TRIG ;

Example in IL

CAL FTRIGInst(CLK := VarBOOL1)

LD FTRIGInst.Q

ST VarBOOL2

Example in FBD

Example in ST

FTRIGInst(CLK:= VarBOOL1);

VarBOOL2 := FTRIGInst.Q;

1.4.2.1.4 Counter

CTU

Provided by standard.lib.

Function block Incrementer:

The input variables CU and RESET as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type WORD.

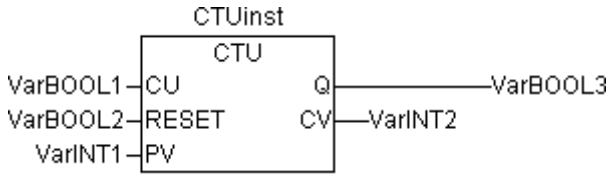
The counter variable CV will be initialized with 0 if RESET is TRUE. If CU has a rising edge from FALSE to TRUE, CV will be raised by 1.Q will return TRUE when CV is greater than or equal to the upper limit PV.

Declaration example	CTUInst : CTU ;

Example in IL

```
CAL  CTUInst(CU := VarBOOL1, RESET := VarBOOL2, PV := VarINT1)
LD   CTUInst.Q
ST   VarBOOL3
LD   CTUInst.CV
ST   VarINT2
```

Example in FBD



Example in ST

```
CTUInst(CU:= VarBOOL1, RESET:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTUInst.Q ;
VarINT2 := CTUInst.CV;
```

CTD

Provided by standard.lib.

Function block Decrementer:

The input variables CD and LOAD as well as the output variable Q are type BOOL, the input variable PV and the output variable CV are type WORD.

When LOAD_ is TRUE, the counter variable CV will be initialized with the upper limit PV. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided CV is greater than 0 (i.e., it doesn't cause the value to fall below 0).

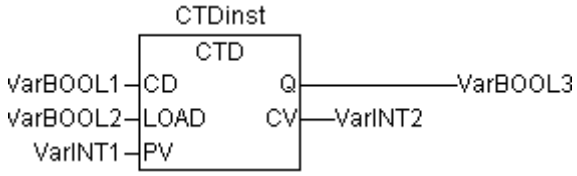
Q returns TRUE when CVis equal 0.

Declaration example	CTDInst : CTD ;

Example in IL

```
CAL  CTDInst(CD := VarBOOL1, LOAD := VarBOOL2, PV := VarINT1)
LD   CTDInst.Q
ST   VarBOOL3
LD   CTDInst.CV
ST   VarINT2
```

Example in FBD



Example in ST

```
CTDInst(CD:= VarBOOL1, LOAD:=VarBOOL2 , PV:= VarINT1);
VarBOOL3 := CTDInst.Q ;
VarINT2 := CTDInst.CV;
```

CTUD

Provided by standard.lib.

Function block Incrementer/Decrementer

The input variables CU, CD, RESET, LOAD as well as the output variables QU and QD are type BOOL, PV and CV are type WORD.

If RESET is valid, the counter variable CV will be initialized with 0. If LOAD is valid, CV will be initialized with PV.

If CU has a rising edge from FALSE to TRUE, CV will be raised by 1. If CD has a rising edge from FALSE to TRUE, CV will be lowered by 1 provided this does not cause the value to fall below 0.

QU returns TRUE when CV has become greater than or equal to PV.

QD returns TRUE when CV has become equal to 0.

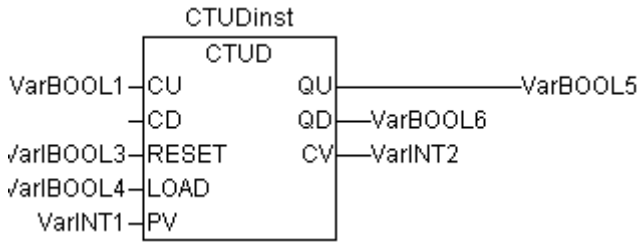
Declaration example:

```
CTUDInst : CUTD ;
```

Example in IL

```
CAL  CTUDInst(CU:=VarBOOL1, RESET:=VarBOOL3, LOAD:=VarBOOL4, PV:=VarINT1)
LD   CTUDInst.Q
ST   VarBOOL5
LD   CTUDInst.QD
ST   VarBOOL5
LD   CTUInst.CV
ST   VarINT2
```

Example in FBD



Example in ST

```
CTUDInst(CU := VarBOOL1, RESET := VarBOOL3, LOAD:=VarBOOL4 , PV:= VarINT1);
VarBOOL5 := CTUDInst.QU ;
VarBOOL6 := CTUDInst.QD ;
VarINT2 := CTUDInst.CV;
```

1.4.2.1.5 Timer

TP

Provided by standard.lib.

The function block Timer is a trigger. TP(IN, PT, Q, ET) means:

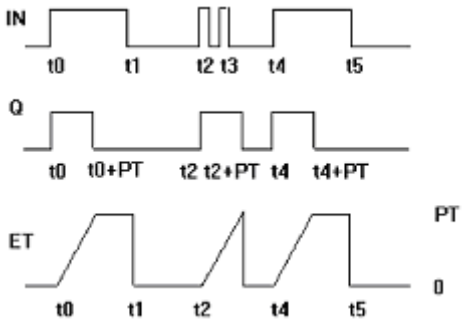
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE as from IN has got TRUE and ET is less than or equal to PT. Otherwise it is FALSE.

Q returns a signal for the time period given in PT.

Graphic Display of the TP Time Sequence



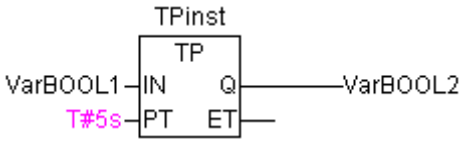
Declaration example:

```
TPInst : TP ;
```

Example in IL

```
CAL TPInst(IN := VarBOOL1, PT := T#5s)
LD  TPInst.Q
ST  VarBOOL2
```

Example in FBD



Example in ST

```
TPInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TPInst.Q;
```

TON

Provided by standard.lib.

The function block Timer On Delay implements a turn-on delay.

TON(IN, PT, Q, ET) means:

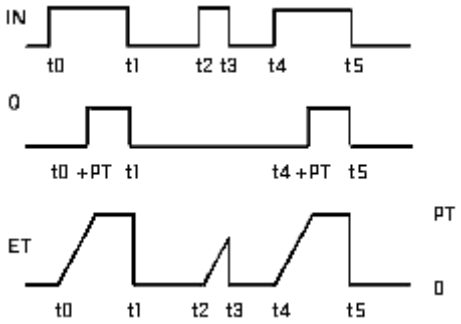
IN and PT are input variables of the BOOL and TIME types respectively. Q and ET are output variables of the BOOL and TIME types respectively. If IN is FALSE, Q is FALSE and ET is 0.

As soon as IN becomes TRUE, the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is TRUE when IN is TRUE and ET is equal to PT. Otherwise it is FALSE.

Thus, Q has a rising edge when the time indicated in PT in milliseconds has run out.

Graphic display of TON behavior over time:

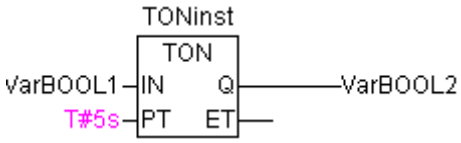


Declaration example:	TONInst : TON ;

Example in IL

```
CAL TONInst(IN := VarBOOL1, PT := T#5s)
LD  TONInst.Q
ST  VarBOOL2
```

Example in FBD



Example in ST

```
TONInst(IN := VarBOOL1, PT:= T#5s);
```

TOF

Provided by standard.lib.

The function block TOF implements a turn-off delay..

TOF(IN, PT, Q, ET) means:

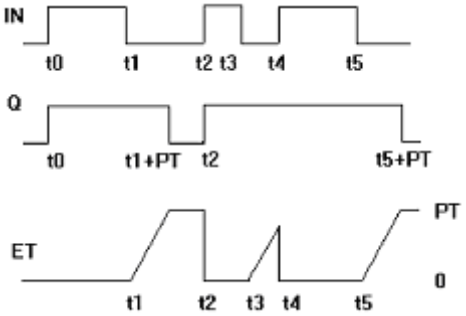
IN and PT are input variables type BOOL respectively TIME. Q and E are output variabls type BOOL respectively TIME. If IN is TRUE, the outputs are TRU respectively 0.

As soon as IN becomes FALSE, in ET the time will begin to be counted in milliseconds in ET until its value is equal to PT. It will then remain constant.

Q is FALSE when IN is FALSE und ET equal PT. Otherwise it is TRUE.

Thus, Q has a falling edge when the time indicated in PT in milliseconds has run out.

Graphic display of TOF behavior over time:

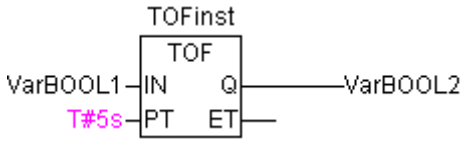


Declaration example	TOFInst : TOF ;

Example in IL

```
CAL TOFInst(IN := VarBOOL1, PT := T#5s)
LD TOFInst.Q
ST VarBOOL2
```

Example in FBD



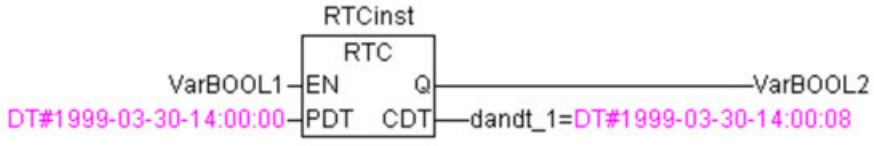
Example in ST

```
TOFInst(IN := VarBOOL1, PT:= T#5s);
VarBOOL2 :=TOFInst.Q;
```

RTC

Provided by standard.lib.

The function block run-time clock returns, starting at a given time, the current date and time.



RTC(EN, PDT, Q, CDT) means:

EN and PDT are input variables type TIME. Q and CDT are output variables type BOOL respectively DATE_AND_TIME. When EN is FALSE, the output variables Q und CDT are FALSE respectively DT#1970-01-01-00:00:00.

As soon as EN becomes TRUE, the time of PDT is set, is counted up in seconds and returned in CDT as long as EN is TRUE (see example in the picture above). As soon as EN is reset to FALSE, CDT is reset to the initial value DT#1970-01-01-00:00:00. Please note that the time in PDT is only set by a rising edge.

1.4.2.2 UTIL.library

Elements of the Util.lib

Element	Description
BCD_TO_INT	Conversion of a Byte: BCD to INT format
INT_TO_BCD	Converstion of a Byte: INT to BCD format
EXTRACT(in,n)	The n-th bit of DWORD in is returned in BOOL
PACK	Up to 8 bits are packed into a byte
PUTBIT	A bit of a DWORD is set to a certain value
UNPACK	A byte is returned as single bits
DERIVATIVE	Local derivation
INTEGRAL	Integral
LIN_TRAFO	Transformation of REAL values
STATISTICS_INT	Min., Max., average values in INT format
STATISTICS_REAL	Min., Max., average in REAL format
VARIANCE	Variance
PD	PD controller
PID	PID controller
BLINK	Pulsating signal
FREQ_MEASURE	Measuring frequency of boolean input signal
GEN	Periodic functions
CHARCURVE	Linear functions
RAMP_INT	Limiting ascendance of descendance of the function beeing fed (INT)
RAMP_REAL	Limiting ascendance of descendance of the function beeing fed (REAL)
HYSTERESIS	Hysteresis
LIMITALARM	Watches whether input value exceeds limits of a defined range

1.4.2.2.1 Overview

This library contains an additional collection of various blocks which can be used for BCD conversion, bit/byte functions, mathematical auxiliary functions, as controller, signal generators, function manipulators and for analog value processing.

As some of the functions and function blocks contain REAL variables, an accessory library named UTIL_NO_REAL exists in which these POUs are excluded.

1.4.2.2.2 BCD conversion

BCD conversion

Provided by util.lib.

A byte in the BCD format contains integers between 0 and 99. Four bits are used for each decimal place. The ten decimal place is stored in the bits 4-7. Thus the BCD format is similar to the hexadecimal presentation, with the simple difference that only values between 0 and 99 can be stored in a BCD byte, whereas a hexadecimal byte reaches from 0 to FF.

Example	The integer 51 should be converted to BCD format. 5 in binary is 0101, 1 in binary is 0001, which makes the BCD byte 01010001, which corresponds to the value $51=81.

BCD_TO_INT

Provided by util.lib.

This function converts a byte in BCD format into an INT value:

The input value of the function is type BYTE and the output is type INT.

Where a byte should be converted which is not in the BCD format the output is -1.

Examples in ST

```
i:=BCD_TO_INT(73); (* Result is 49 *)
k:=BCD_TO_INT(151); (* Result is 97 *)
l:=BCD_TO_INT(15); (* Output -1, because it is not in BCD format *)
```

INT_TO_BCD_

Provided by util.lib.

This function converts an INTEGER value into a byte in BCD format:

The input value of the function is type INT, the output is type BYTE.

The number 255 will be outputted where an INTEGER value should be converted which cannot be converted into a BCD byte.

Examples in ST

```
i:=INT_TO_BCD(49); (* Result is 73 *)
k:=BCD_TO_INT(97); (* Result is 151 *)
l:=BCD_TO_INT(100); (* Error! Output: 255 *)
```

1.4.2.2.3 Bit-/Byte functions

EXTRACT

Provided by util.lib.

Inputs to this function are a DWORD X, as well as a BYTE N. The output is a BOOL value, which contains the content of the N^{th} bit of the input X, whereby the function begins to count from the zero bit.

Examples in ST

```
FLAG:=EXTRACT(X:=81, N:=4); (* Result : TRUE, because 81 is binary 1010001, so the 4th bit is 1 *)
FLAG:=EXTRACT(X:=33, N:=0); (* Result : TRUE, because 33 is binary 100001, so the bit '0' is 1 *)
```

PACK

Provided by util.lib.

This function is capable of delivering back eight input bits B0, B1, ..., B7 from type BOOL as a BYTE.

The function block UNPACK is closely related to this function.

PUTBIT

Provided by util.lib.

The input to this function consists of a DWORD X, a BYTE N and a BOOLean value B.

PUTBIT sets the N^{th} bit from X on the value B, whereby it starts counting from the zero bit.

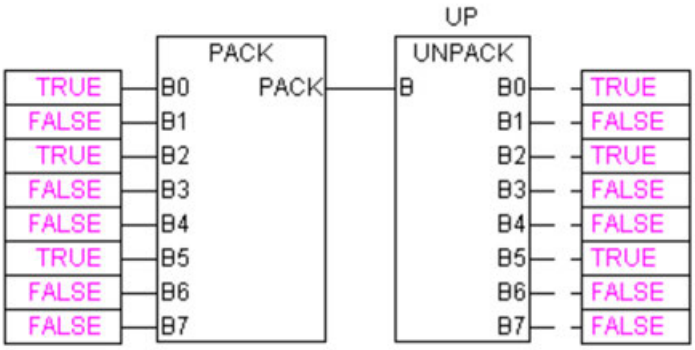
Example in ST

```
A:=38; (* binary 100110 *)
B:=PUTBIT(A,4,TRUE); (* Result : 54 = 2#110110 *)
C:=PUTBIT(A,1,FALSE); (* Result : 36 = 2#100100 *)
```

UNPACK

Provided by util.lib.

UNPACK converts the input B from type BYTE into 8 output variables B0,...,B7 of the type BOOL, and this is the opposite to PACK.



1.4.2.2.4 Mathematic auxiliary functions

DERIVATIVE

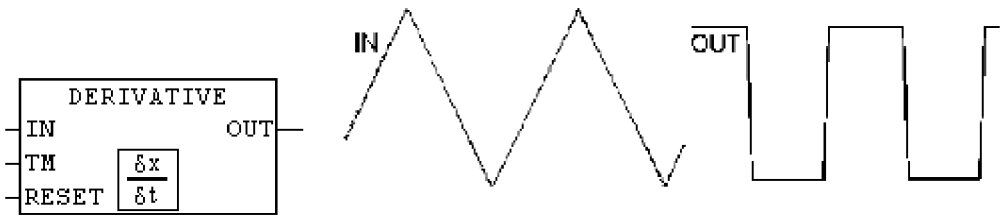
Provided by util.lib.

This function block approximately determines the local derivation.

The function value is delivered as a REAL variable by using IN. TM contains the time which has passed in ms in a DWORD and the input of RESET of the type BOOL allows the function block to start anew through the delivery of the value TRUE.

The output OUT is of the type REAL.

In order to obtain the best possible result, DERIVATIVE approximates using the last four values, in order to hold errors which are produced by inaccuracies in the input parameters as low as possible.



INTEGRAL

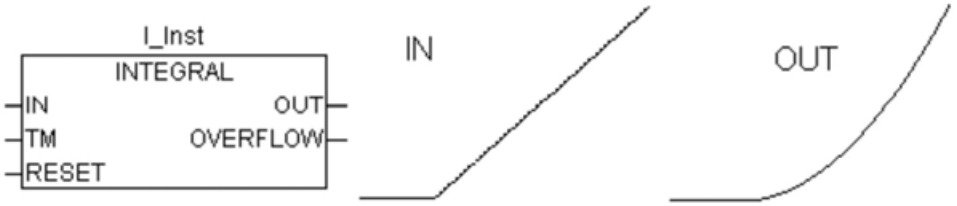
Provided by util.lib.

This function block approximately determines the integral of the function.

In an analog fashion to DERIVATIVE, the function value is delivered as a REAL variable by using IN. TM contains the time which has passed in ms in a DWORD and the input of RESET of the type BOOL allows the function block to start anew with the value TRUE.

The output OUT is of the type REAL. The integral is approximated by two step functions. The average of these is delivered as the approximated integral.

Example: Integration of a linear function:



LIN_TRAFO

This function block (util.lib. transforms a REAL-value, which lies in a range of values defined by a lower and upper limit value, to a REAL-value which lies correspondingly in another range also defined by a lower and upper limit. The following equation is basis of the conversion:

(IN - IN_MIN) : (IN_MAX - IN) = (OUT - OUT_MIN) : (OUT_MAX - OUT)

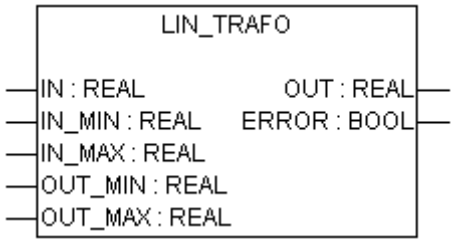


Table 37: Input variables

Variable	Data type	Description
IN	REAL	Input value
IN_MIN	REAL	Lower limit of input range of values
IN_MAX	REAL	Upper limit of input range of values
OUT_MIN	REAL	Lower limit of output value range
OUT_MAX	REAL	Upper limit of output value range

Table 38: Output variables

Variable	Data type	Description
OUT	REAL	Output value
ERROR	BOOL	Error occurred: TRUE, if IN_MIN = IN_MAX, or if IN is outside of the specified input range of values

Application example:

A temperature sensor provides Volt-values (input IN). These are to be converted to temperature values in degree celsius (output OUT). The input(Volt) range of values is defined by the limits IN_MIN=0 and IN_MAX=10. The output(degree celsius) range of values is defined by the limits OUT_MIN=-20 and OUT_MAX=40.

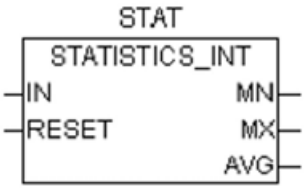
Thus for an input of 5 Volt a temperature of 10 degree celsius will result.

STATISTICS_INT

Provided by util.lib.

This function block calculates some standard statistical values:

The input IN is of the type INT. All values are initialised anew when the BOOLean input RESET is TRUE. The output MN contains the minimum, MX of the maximum value from IN. AVG describes the average, that is the expected value of IN. All three outputs are of the type INT.



STATISTICS_REAL

Provided by util.lib.

This function block corresponds to STATISTICS_INT, except that the input IN is of the type REAL like the outputs MN, MX, AVG.

VARIANCE

Provided by util.lib.

VARIANCE calculates the variance of the entered values.

The input IN is of the type REAL, RESET is of the type BOOL and the output OUT is again of the type REAL.

This block calculates the variance of the inputted values. VARIANCE can be reset with RESET=TRUE.

The standard deviation can easily be calculated as the square root of the VARIANCE.

1.4.2.2.5 Controllers

PD

The library util.lib provides the following PD controller function block:

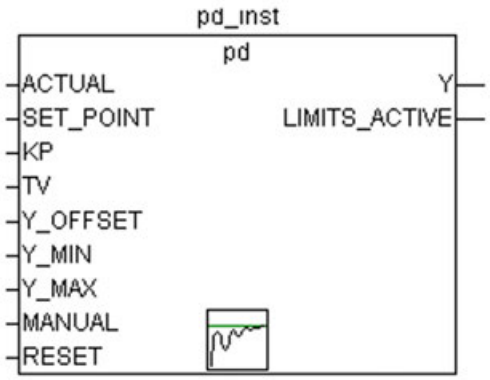


Table 39: Inputs of the function block

Variable	Data type	Description
ACTUAL	REAL	Current value of the controlled variable
SET_POINT	REAL	Desired value, command variable
KP	REAL	Coefficient of proportionality, unity gain of the P-part
TV	REAL	Derivative action time, unity gain of the D-part in seconds, e.g. "0.5" for 500 ms
Y_MANUAL	REAL	Defines output value Y in case of MANUAL = TRUE
Y_OFFSET	REAL	Offset for the manipulated variable Y
Y_MIN, Y_MAX	REAL	Lower resp. upper limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN<Y_MAX.
MANUAL	BOOL	If TRUE, manual operation will be active, i.e. the manipulated value will be defined by Y_MANUAL.
RESET	BOOL	TRUE resets the controller; during reinitialization Y = Y_OFFSET.

Table 40: Outputs of the function block

Variable	Data type	Description
Y	REAL	Manipulated value, calculated by the function block (see below)
LIMITS_ACTIVE	BOOL	TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).

Y_OFFSET, Y_MIN und Y_MAX are used for the transformation of the manipulated variable within a prescribed range.

MANUAL can be used to switch on and off manual operation. RESET serves to reset the controller.

In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error e as difference SET_POINT - ACTUAL, generates the derivation with respect to time $\delta e/\delta t$ and stores these values internally.

The output, i.e. the manipulated variable Y is calculated as follows:

$Y = KP (\Delta + TV\ \delta\Delta/\delta t) + Y\_OFFSET$ whereby Δ = SET_POINT-ACTUAL

So besides the P-part also the current change of the controller error (D-part) influences the manipulated variable.

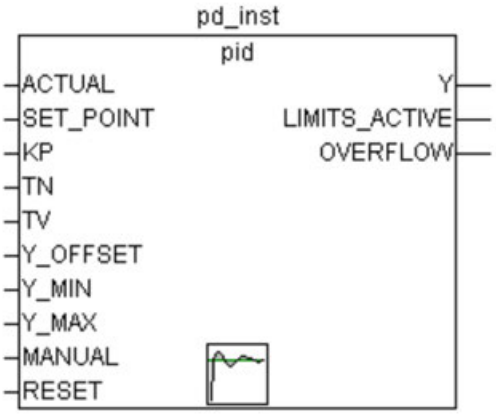
Additionally Y is limited on a range prescribed by Y_MIN and Y_MAX. If Y exceeds these limits, LIMITS_ACTIVE will get TRUE. If no limitation of the manipulated variable is desired, Y_MIN and Y_MAX have to be set to 0.

As long as MANUAL=TRUE, Y_MANUAL will be written to Y.

A P-controller can be easily created by setting TV=0.

PID

The library util.lib provides the following PID controller function block:



Unlike the PD controller, this function block contains a further REAL input TN for the readjusting time in sec (e.g. "0.5" for 500 ms).

Table 41: Inputs of the function block

Variable	Data type	Description
ACTUAL	REAL	Current value of the controlled variable
SET_POINT	REAL	Desired value, command variable
KP	REAL	Coefficient of proportionality, unity gain of the P-part
TN	REAL	Reset time, reciprocal unity gain of the I-part; given in seconds, e.g. "0.5" for 500 ms
TV	REAL	Derivative action time, unity gain of the D-part in seconds, e.g. "0.5" for 500 ms

Variable	Data type	Description
Y_MANUAL	REAL	Defines output value Y in case of MANUAL = TRUE
Y_OFFSET	REAL	Offset for the manipulated variable Y
Y_MIN, Y_MAX	REAL	Lower resp. upper limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN<Y_MAX.
MANUAL	BOOL	If TRUE, manual operation will be active, i.e. the manipulated value will be defined by Y_MANUAL.
RESET	BOOL	TRUE resets the controller; during reinitialization Y = Y_OFFSET.

Table 42: Outputs of the function block

Variable	Data type	Description
Y	REAL	Manipulated value, calculated by the function block (see below)
LIMITS_ACTIVE	BOOL	TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).
OVERFLOW	BOOL	TRUE indicates an overflow (see below)

Y_OFFSET, Y_MIN und Y_MAX serve for transformation of the manipulated variable within a prescribed range.

MANUAL can be used to switch to manual operation; RESET can be used to re-initialize the controller..

In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error e as difference from SET_POINT "“ ACTUAL, generates the derivation with respect to time $\delta e/\delta t$ and stores these values internally.

The output, i.e. the manipulated variable Y unlike the PD controller contains an additional integral part and is calculated as follows:

$Y = KP (\Delta + 1/TN \int edt + TV\ \delta\Delta /\delta t) + Y\_OFFSET$

So besides the P-part also the current change of the controller error (D-part) and the history of the controller error (I-part) influence the manipulated variable.

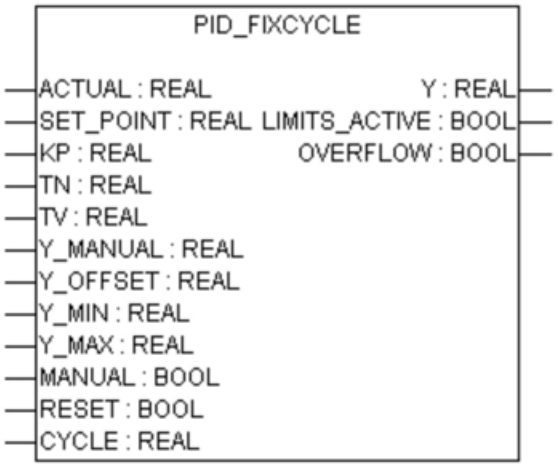
The PID controller can be easily converted to a PI-controller by setting TV=0.

Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller, if the integral of the error D becomes to great. Therefore for the sake of safety a BOOLean output called OVERFLOW is present, which in this case would have the value TRUE. This only will happen if the control system is instable due to incorrect parameterization. At the same time, the controller will be suspended and will only be activated again by re-initialization.

PID_FIXCYCLE

Provided by util.lib.

The PID_FIXCYCLE controller function block:



This function block functionally corresponds to the PID controller with the exception that the cycle time is not measured autmatically by an internal function but is set by input CYCLE (in seconds).

1.4.2.2.6 Signal generators

BLINK

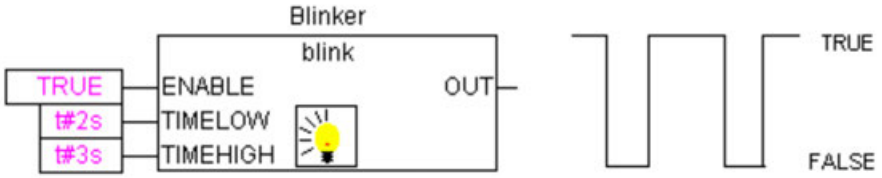
Provided by util.lib.

The function block BLINK generates a pulsating signal. The input consists of ENABLE of the type BOOL, as well as TIMELOW and TIMEHIGH of the type TIME. The output OUT is of the type BOOL.

If ENABLE is set to TRUE, BLINK begins to set the output for the time period TIMEHIGH to TRUE and afterwards to set it for the time period TIMELOW to FALSE.

When ENABLE is reset to FALSE, output OUT will not be changed, i.e. no further pulse will be generated. If you explicitly also want to get OUT FALSE when ENABLE is reset to FALSE, you might use "OUT AND ENABLE" (i.e. adding an AND box with parameter ENABLE) at the output.

Example in CFC:



FREQ_MEASURE

Provided by util.lib.

This function block measures the (average) frequency (Hz) of a boolean input signal. You can specify over how many periods it should be averaged. A period is the time between two rising edges of the input signal.

Table 43: Input variables

Variable	Data Type	Description
IN	BOOL	Input signal
PERIODS	INT	Number of periods, i.e. the time intervals between the rising edges, over which the average frequency of the input signal should be calculated. Possible values: 1 to 10
RESET	BOOL	Reset of all parameters to 0

Table 44: Output variables

Variable	Data Type	Description
OUT	REAL	Resulting frequency in [Hz]
VALID	BOOL	FALSE until the first measure has been finished, or if the period > 3*OUT (indicating something wrong with the inputs)

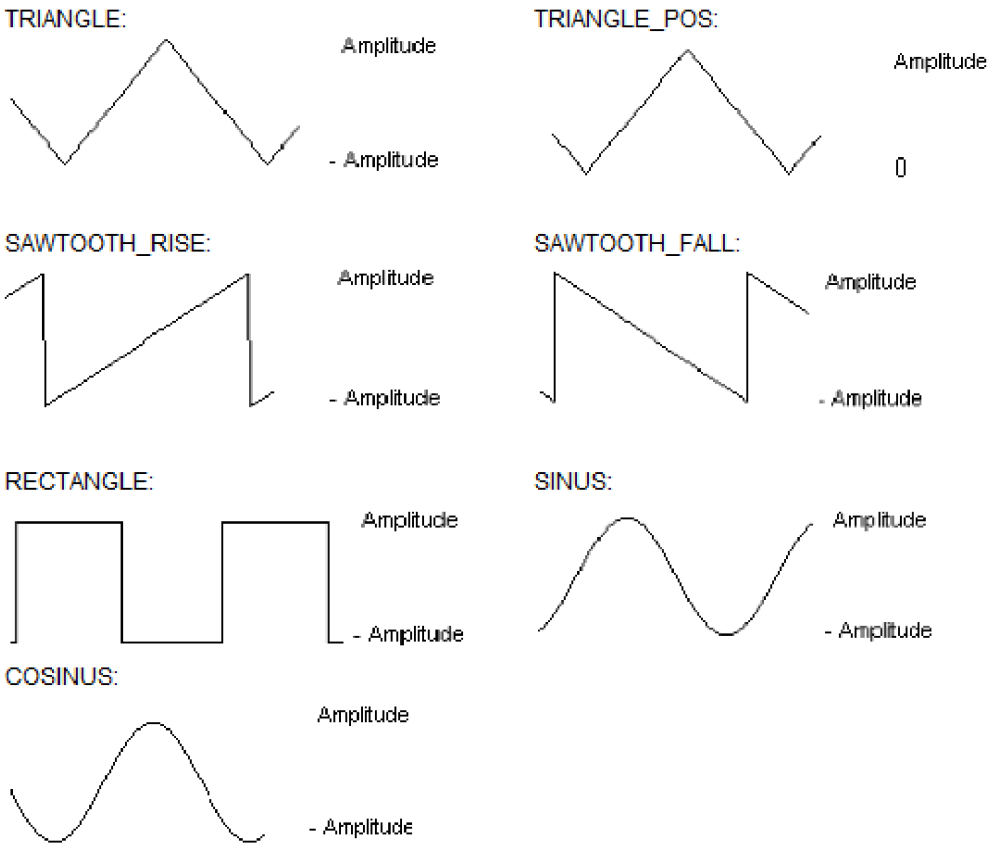
GEN

Provided by util.lib.

The function generator generates typical periodic functions:

The inputs are a composition consisting of MODE from the pre-defined counting type GEN_MODE, BASE of the type BOOL, PERIOD of the type TIME, of two INT values CYCLES and AMPLITUDE and of the BOOLean RESET input.

The MODE describes the function which should be generated, whereby the enumeration values TRIANGLE and TRIANGLE_POS deliver two triangular functions, SAWTOOTH_RISE an ascending, SAWTOOTH_FALL a descending sawtooth, RECTANGLE a rectangular signal and SINE and COSINE the sine and cosine:

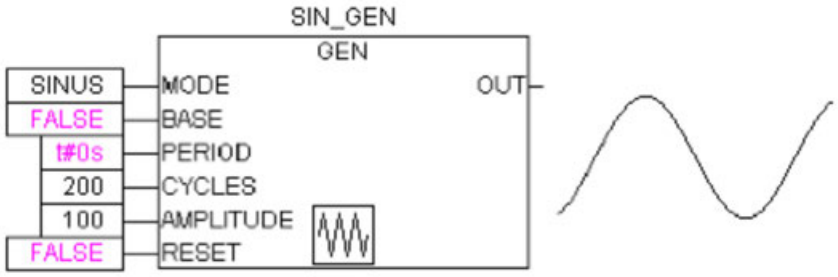


BASE defines whether the cycle period is really related to a defined time (BASE=TRUE) or whether it is related to a particular number of cycles, which means the number of calls of function block (BASE=FALSE).

PERIOD or CYCLES defines the corresponding cycle period.

AMPLITUDE defines, in a trivial way, the amplitude of the function to be generated.

The function generator is again set to 0 as soon as RESET=TRUE.

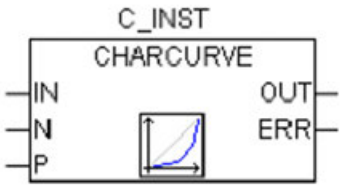


1.4.2.2.7 Function manipulators

CHARCURVE

Provided by util.lib.

This function block serves to represent values, piece by piece, on a linear function:



IN of the type INT is fed with the value to be manipulated. The BYTE N designates the number of points which defines the presentation function. This characteristic line is then generated in an ARRAY P[0..10] with P of the type POINT which is a structure based on two INT values (X and Y).

The output consists of OUT of the type INT, the manipulated value and BYTE ERR, which will indicate an error if necessary.

The points P[0]..P[N-1] in the ARRAY must be sorted according to their X values, otherwise ERR receives the value 1.

If the input IN is not between P[0].X and P[N-1].X, ERR=2 and OUT contains the corresponding limiting value P[0]. Y or P[N-1].Y.

If N lies outside of the allowed values which are between 2 and 11, then ERR=4.

Example in ST

First of all ARRAY P must be defined in the header:

```
VAR
...
CHARACTERISTIC_LINE:CHARCURVE;
KL:ARRAY[0..10] OF POINT:=(X:=0,Y:=0),(X:=250,Y:=50),
(X:=500,Y:=150),(X:=750,Y:=400),7((X:=1000,Y:=1000));
COUNTER:INT;
...
END_VAR
```

Then we supply CHARCURVE with for example a constantly increasing value:

```
COUNTER:=COUNTER+10;
CHARACTERISTIC_LINE(IN:=COUNTER,N:=5,P:=KL);
```

The subsequent tracing illustrates the effect:

RAMP_INT

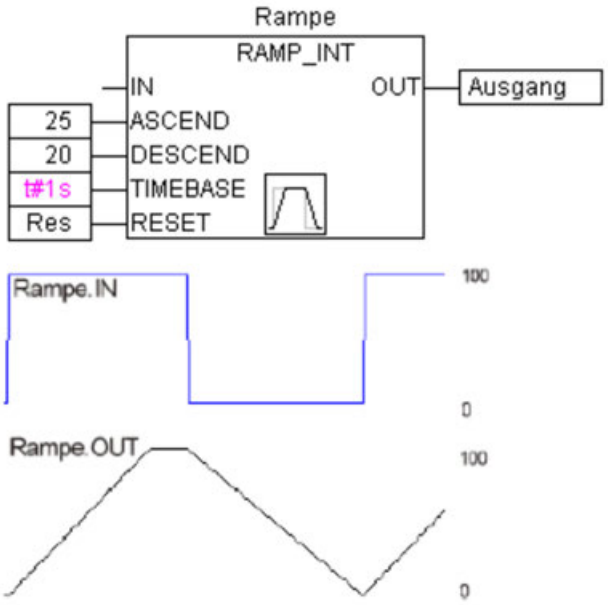
Provided by util.lib.

RAMP_INT serves to limit the ascendance or descendance of the function being fed:

The input consists on the one hand out of three INT values: IN, the function input, and ASCEND and DESCEND, the maximum increase or decrease for a given time interval, which is defined by TIMEBASE of the type TIME. Setting RESET to TRUE causes RAMP_INT to be initialised anew.

The output OUT of the type INT contains the ascend and descend limited function value.

When TIMEBASE is set to t#0s, ASCEND and DESCEND are not related to the time interval, but remain the same.



RAMP_REAL

Provided by util.lib.

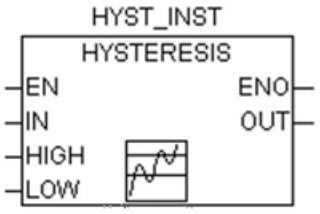
RAMP_REAL functions in the same way as RAMP_INT, with the simple difference that the inputs IN, ASCEND, DESCEND and the output OUT are of the type REAL.

1.4.2.2.8 Analog value processing

HYSTERESIS

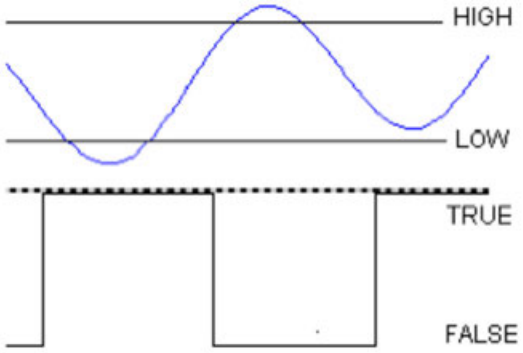
Provided by util.lib.

The input to this function block consists of three INT values IN, HIGH and LOW. The output OUT is of the type BOOL.



If IN goes below the limiting value LOW, OUT becomes TRUE. If IN goes over the upper limit HIGH, FALSE is delivered.

An illustrative example:



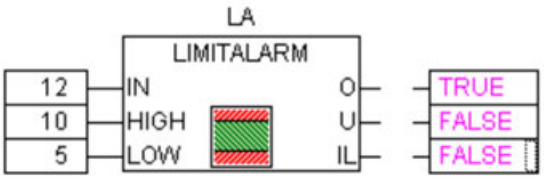
LIMITALARM

Provided by util.lib.

This function block specifies, whether the input value is within a set range and which limits it has violated if it has done so.

The input values IN, HIGH and LOW are each of the type INT, while the outputs O, U and IL are of the type BOOL. If the upper limit HIGH is exceeded by IN, O becomes TRUE, and when IN is below LOW, U becomes TRUE. IL is TRUE if IN lies between LOW and HIGH.

Result:



1.4.2.3 AnalyzationNew.library

1.4.2.3.1 Analysis of expression

This library provides modules for the analysis of expressions. If a composed expression is FALSE, those of its components can be evaluated which are adding to this result. In the SFC-Editor the flag uses this function implicitely for the analysis of expressions in transitions ↳ *Chapter 1.4.1.3.11.9.1 “Overview” on page 330* ↳ *SFCErrorAnalyzationTable*.

Example of an expression:

```
b OR NOT(y < x) OR NOT (NOT d AND e)
```

The functions: The following variables are used by all modules:

- InputExpr: BOOL, expression to be analysed
- DoAnalyze: BOOL, TRUE starts analysis
- ExpResult: BOOL, current value of the expression

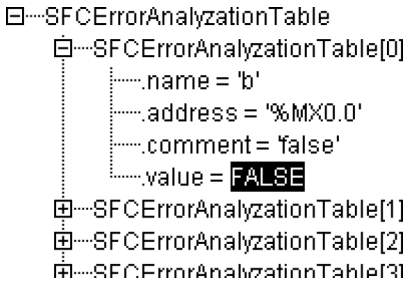
Different is the output of the result of the analyzation:

'AnalyzeExpression' returns in a string the components of the expression, which are adding to the total value FALSE. Function AppendErrorString is used for this purpose, separating the particular components in the output string by "|" characters:

OutString: STRING, Result of the analysis, Sequence of the concerned components of the expression (e.g. y < x | d)

'AnalyseExpressionTable' writes the components of the expression, which are adding to the total value FALSE, to an array. For each component the following information is provided by structure ExpressionResult: name, address, comment, (current)value:

OutTable: ARRAY [0..15] OF ExpressionResult; e.g..



'AnalyseExpressionCombined' combines the functionalities of AnalyzeExpression plus AnalyseExpressionTable

1.4.2.4 Protocol- and system libraries

1.4.2.4.1 The library SysLibCallback.lib

This library provides the functions SysCallbackRegister and SysCallbackUnregister, which serve to activate defined callback functions for runtime events.

Both functions are of type BOOL and return TRUE as soon as the required callback function successfully has been registered resp. de-registered.

The execution is synchronous.

Prototype of the callback function

The prototype of the callback function must look as follows:

```
FUNCTION Callback : DWORD
VAR_INPUT
 dwEvent:DWORD; // Event
 dwFilter:DWORD; // Filter
 dwOwner:DWORD; // Source
END_VAR
```

Attention for RISC and Motorola 68K target systems: The name of the callback function must start with "callback"!

The library functions SysCallbackRegister and SysCallbackUnregister each use the following parameters when calling the callback function which should be registered or de-registered:

Input-Variable	Data Type	Description
iPOUIndex	INT	POU Index of the callback function which should be (de)registered. The index must be accquired before with the aid of the operator INDEXOF(<function name>).
Event	RTS_EVENT	The runtime event, for which the callback function is called, is defined by a value of the enumeration RTS_EVENT, which also is contained in the library (see below).

The enumeration RTS_EVENT is defined as follows:

```
TYPE RTS_EVENT :
(
EVENT_ALL,
(* General events *)
EVENT_START,
EVENT_STOP,
EVENT_BEFORE_RESET,
EVENT_AFTER_RESET,
EVENT_SHUTDOWN,

(* Exceptions generated by run time *)
EVENT_EXCPT_CYCLETIME_OVERFLOW, (* Cycle time overflow *)
EVENT_EXCPT_WATCHDOG, (* Software watchdog OF IEC-task expired *)
EVENT_EXCPT_HARDWARE_WATCHDOG, (* Hardware watchdog expired. Global software error *)
EVENT_EXCPT_FIELDBUS, (* Fieldbus error occurred *)
EVENT_EXCPT_IOUPDATE, (* IO-update error *)

(* Exceptions generated BY system *)
EVENT_EXCPT_ILLEGAL_INSTRUCTION, (* Illegal instruction *)
EVENT_EXCPT_ACCESS_VIOLATION, (* Access violation *)
EVENT_EXCPT_PRIV_INSTRUCTION, (* Privileged instruction *)
EVENT_EXCPT_IN_PAGE_ERROR, (* Page error *)
EVENT_EXCPT_STACK_OVERFLOW, (* Stack overflow *)
EVENT_EXCPT_MISALIGNMENT, (* Datatype misalignment *)
EVENT_EXCPT_ARRAYBOUNDS, (* ARRAY bounds exceeded *)
EVENT_EXCPT_DIVIDEBYZERO, (* Division BY zero *)
EVENT_EXCPT_OVERFLOW, (* Overflow *)
EVENT_EXCPT_NONCONTINUABLE, (* Non continuable *)
EVENT_EXCPT_NO_FPU_AVAILABLE, (* FPU: No FPU available *)
```

```
EVENT_EXCPT_FPU_ERROR, (* FPU: Unspecified error *)
EVENT_EXCPT_FPU_DENORMAL_OPERAND, (* FPU: Denormal operand *)
EVENT_EXCPT_FPU_DIVIDEBYZERO, (* FPU: Division BY zero *)
EVENT_EXCPT_FPU_INVALID_OPERATION, (* FPU: Invalid operation *)
EVENT_EXCPT_FPU_OVERFLOW, (* FPU: Overflow *)
EVENT_EXCPT_FPU_STACK_CHECK, (* FPU: Stack check *)

(* IO events *)
EVENT_AFTER_READING_INPUTS,
EVENT_BEFORE_WRITING_OUTPUTS,

(* Miscellaneous events *)
EVENT_TIMER, (* Schedule tick (timer interrupt) *)
EVENT_DEBUG_LOOP, (* Debug loop at breakpoint *)

(* Online services *)
EVENT_ONLINE_SERVICES_BEGIN := 500,
EVENT_LOGIN,
EVENT_CUSTOM_SERVICES,

(* Interrupts *)
EVENT_INT_0:=1000,
EVENT_INT_1,
EVENT_INT_2,
EVENT_INT_3,
EVENT_INT_4,
EVENT_INT_5,
EVENT_INT_6,
EVENT_INT_7,
EVENT_INT_8,
EVENT_INT_9,
EVENT_INT_10,
EVENT_INT_11,
EVENT_INT_12,
EVENT_INT_13,
EVENT_INT_14,
EVENT_INT_15,
EVENT_INT_255:=1255,
EVENT_MAX
);
END_TYPE
```

1.4.2.4.2 The library SysLibCom.lib

Overview

This library supports the serial communication with a target system. If the target system provides the functionality, then the following library functions can be used to open or close a serial port and to read or write data via this port (The execution is synchronous.)

SysComOpen

This function serves to open a serial port.

The function returns a handle for the port, which can be passed on when calling other functions of the library. If the port cannot be opened, 0xFFFFFFFF will be returned as handle.

Input Variable	Data Type	Description
Port	PORTS	specifies the port which should be opened (COM1,..); Port number see below: Enumeration PORTS

Enumeration PORTS:

TYPE PORTS : (COM1:=1, COM2, COM3, COM4, COM5, COM6, COM7, COM8);

END_TYPE

SysComSetSettings

This function serves to set values like tranmission rate, stopbits, parity, function-timout, buffersize and scan-time for a serial port. The parameter value is of type POINTER TO COMSETTINGS; the structure COMSETTINGS is used.

As soon as the parameters could be set successfully, TRUE will be returned, otherwise FALSE.

Input Variable	Data Type	Description
ComSettings	POINTER TO COMSETTINGS;	Pointer to the structure COMSETTINGS; you can make use of the operator ADR (see below, example)
dwHandle	DWORD	Port handle, acquired by *Chapter 1.4.2.4.2.2 "SysComOpen" on page 564*

The structure COMSETTINGS, which is also part of the library, is defined as follows:

TYPE COMSETTINGS :

STRUCT

Port:PORTS;	Port number, see below: Enumeration PORTS
dwBaudRate:DWORD;	4800, 9600, 19200, 38400, 57600, 115200
byStopBits:BYTE;	0 = ONESTOPBIT, 1=ONE5STOPBITS, 2=TWOSTOPBITS

byParity:BYTE;	0 = NOPARITY, 1 = ODDPARITY, 2 = EVEN-PARITY
dwTimeout:DWORD;	Timeout of the interface in ms, Default = 0
dwBufferSize:DWORD;	Buffer size of the internal device buffer, Default = 0
dwScan:DWORD;	Polling time of the serial interface; should be set to 0

END_STRUCT

END_TYPE

Enumeration PORTS:

TYPE PORTS : (COM1:=1, COM2, COM3, COM4, COM5, COM6, COM7, COM8);

END_TYPE

SysComSetSettingsEx

This function of type BOOL with the parameters of type POINTER TO COMSETTINGSEX is used to set all relevant parameters of a serial communication port. Not only the parameters of the above function are set, but also the parameters for flowcontrol and character size can be set with this function. This is performed by filling them into the structure COMSETTINGSEX.

The return value of the function is true if the parameters were successfully set and false if the parameters could not be applied to the communication port. It is hardware-dependent whether the settings can be changed more often than one time after opening a port. It may be necessary to close and reopen the port before setting the parameters new.

Input Variable	Data Type	Description
ComSettingsEx	POINTER TO COMSET-TINGSEX;	Pointer to the structure COM-SETTINGSEX; Use the ADR operator to determine an address.
dwHandle	DWORD	Port handle, acquired by SysComOpen ↳ *Chapter 1.4.2.4.2.2 “SysComOpen” on page 564*

Structure COM-SETTINGSEX:

TYPE COMSETTINGSEX :

STRUCT

Size:INT;	(*The size in bytes of the structure. Use the sizeof() operator to fill in. Used for backward compatibility.*)
Port:PORTS;	(*Port number, see below: Enumeration PORTS *)
dwBaudRate:DWORD;	(* 4800, 9600, 19200, 38400, 57600, 115200 *)
byStopBits:BYTE;	(* 0 = ONESTOPBIT, 1=ONE5STOPBITS, 2=TWOSTOPBITS *)

byParity:BYTE;	(* 0 = NOPARITY, 1 = ODDPARITY, 2 = EVENPARITY *)
dwTimeout:DWORD;	(* Timeout of the port in ms, Default = 0 *)
dwBufferSize:DWORD;	(* Buffersize used by the driver, Default = 0 *)
dwScan:DWORD;	(* Poll-time of the serial driver. Should be set to 0. Only change if the documentation of the hardware-vendor tells so. *)
cByteSize : BYTE;	(*4...8: Character size in bits.*)
fOutxCtsFlow : BOOL;	(*Specifies whether the CTS (clear-to-send) signal is monitored for output flow control. If this member is TRUE and CTS is turned off, output is suspended until CTS is sent again. *)
fDtrControl : BYTE;	(*0:Disables the DTR line when the device is opened and leaves it disabled. 1:Enables the DTR line when the device is opened and leaves it on. 2:Enables DTR handshaking. *)
fDsrSensitivity : BOOL;	(*Specifies whether the communications driver is sensitive to the state of the DSR signal. If this member is TRUE, the driver ignores any bytes received, unless the DSR modem input line is high. *)
fRtsControl : BYTE;	(*0: Disables the RTS line when the device is opened and leaves it disabled. 1: Enables the RTS line when the device is opened and leaves it on. 2: Enables RTS handshaking. The driver raises the RTS line when the "type-ahead" (input) buffer is less than one-half full and lowers the RTS line when the buffer is more than three-quarters full. 3: Specifies the RTS line will be high if bytes are available for transmission. After all buffered bytes have been sent, the RTS line will be low. *)
fOutxDsrFlow : BOOL;	(*Specifies whether the DSR (data-set-ready) signal is monitored for output flow control. If this member is TRUE and DSR is turned off, output is suspended until DSR is sent again. *)

END_STRUCT

END_TYPE

Enumeration PORTS

TYPE PORTS : (COM1:=1, COM2, COM3, COM4, COM5, COM6, COM7, COM8);

END_TYPE

Example for the settings to perform a hardwarehandshake:

dwHandle: DWORD;

pt_comsettingsex:COMSETTINGSEX:=(Port:=COM1,

dwBaudRate:=38400,

byStopBits:=0,

dwTimeout:=5000,

cByteSize:=7,

byParity := 2,

fOutxCtsFlow := FALSE,

fOutxDsrFlow:=TRUE,

DtrControl := 2,

fRtsControl := 2);

Implementation:

pt_comsettingsex.Size := sizeof(pt_comsettingsex);

SysComSetSettingsEx(dwHandle := Handle, ComSettingsExt := ADR(pt_comsettingsex));

Where Handle is the returnvalue of a call to SysComOpen(COM1).

SysComClose

This function of type BOOL closes the COM port. For that purpose the port handle, which has been got by SysComOpen, must be given as input parameter. The return value will be TRUE after a successful operation, otherwise FALSE.

Input Variable	Data Type	Description
dwHandle	DWORD	Port handle, acquired by SysComOpen ↳ *Chapter 1.4.2.4.2.2 “SysComOpen” on page 564*

SysComWrite

This function of type DWORD writes the data to that port which is defined by the handle got by SysComOpen. Besides the handle also the address from which the data should be taken, the number of data which should be written and the timeout of the function must be passed on.

The function will return the number of actually written bytes.

Input Variable	Data Type	Description
dwHandle	DWORD	Port handle, acquired by SysComOpen ↳ *Chapter 1.4.2.4.2.2 “SysComOpen” on page 564*
dwBufferAddress	DWORD	Address from which the data should be taken and written to the port; you can use the ADR operator to get this address
dwBytesToWrite	DWORD	Number of bytes, which should be written
dwTimeout	DWORD	Time in [ms], after which the function will return at the latest

SysComRead

This function of type DWORD reads the data of COM-PORT. The input parameters are the port handle got by SysComOpen, the number of expected bytes and the timeout of the function. Besides that the address to which the read data should be copied, will be passed on.

The function will return the number of actually read bytes.

Input Variable	Data Type	Description
dwHandle	DWORD	Port handle, acquired by SysComOpen ↳ *Chapter 1.4.2.4.2.2 “SysComOpen” on page 564*
dwBufferAddress	DWORD	address, to which the read bytes should be copied after having been read from the port; (you can make use of the operator ADR to get this address)
dwBytesToRead	DWORD	Number of bytes, which should be read
dwTimeout	DWORD	Time in [ms], after that the function returns at the latest

SysComGetVersion2300

This function (type DWORD, always returns 0) is only used for an automatic internal version check and is not to be called explicitly in the application program.

1.4.2.4.3 The library SysLibDir.lib

Overview

If supported by the runtime system, you can use the functions of this library to handle a file directory system on the target system. Entries of the directory can be read and modified. The execution is synchronous.

SysDirCreate

This function of type BOOL can be used to create a new directory.

The return value is TRUE, if the directory has been created, or FALSE in case of error.

Input-Variable	Data type	Description
stName	STRING	Name of the directory

SysDirOpen

This function of type DWORD can be used, to open a directory in order to read the directory entries (files, subdirectories) via function SysDirRead ↳ *Chapter 1.4.2.4.3.4 “SysDirRead” on page 569*.

To close the directory, SysDirRead must be called until it returns 0.

The return value is a handle, which is required by function SysDirRead ↳ *Chapter 1.4.2.4.3.4 “SysDirRead” on page 569* as an input.

Input-Variable	Data type	Description
stDirectory	STRING	Directory name

SysDirRead

This function of type UDINT can be used to read directory entries.

Each time the function is called, one entry of the directory will be read. As long as "1" is returned, a further entry is in the directory. Thus for reading all entries the function must be called repeatedly until "0" is returned. In this case, the directory is closed and the handle is not valid any more. In every case, the function has to be called until it returns 0 to close the directory.

The information on the particular entries will be written to structure DIRECTORY_INFO.

Input-Variable	Data type	Description
hDir	DWORD	Handle of the directory; Returned by function SysDirOpen ↳ *Chapter 1.4.2.4.3.3 “SysDirOpen” on page 568*, which was called before for opening the directory.
stDirEntry	STRING	Name of an entry in the directory. Can be a file or another directory. Max. 80 characters.
pDirInfo	POINTER TO 'Structure DIRECTORY_INFO' ↳ *Chapter 1.4.2.4.3.7 “Structure DIRECTORY_INFO” on page 570*	Pointer on structure DIRECTORY_INFO, which will be filled with information on the read entry. You can enter "0" here if the information should not be read.

SysDirRemove

This function of type BOOL can be used to delete a directory.

TRUE will be returned if the directory could be removed, otherwise FALSE.

Input-Variable	Data type	Description
stName	STRING	Name of the directory to be removed

SysDirRename

This function of type BOOL can be used to rename a directory.

TRUE will be returned if the directory could be renamed, otherwise FALSE.

Input-Variable	Data type	Description
stOldName	STRING	old directory name
stNewName	STRING	new directory name

Structure DIRECTORY_INFO

This structure contains information on a directory, which is read via function SysDirRead ↳ *Chapter 1.4.2.4.3.4 “SysDirRead” on page 569*.

The components:

Variable	Data type	Description
ftTime	DIRFILETIME ↳ *Chapter 1.4.2.4.3.8 “Structure DIRFILETIME” on page 570*	Structure with information on creation date, change date, access date
dwSize	DWORD	Size of the directory entry (file)
bDirectory	BOOL	TRUE, if entry is a directory; FALSE, if the entry is a file

Structure DIRFILETIME

This structure contains data information for a directory entry. It is used by 'Structure DIRECTORY_INFO' ↳ *Chapter 1.4.2.4.3.7 “Structure DIRECTORY_INFO” on page 570*.

The components:

Variable	Data type	Description
dtCreation	DT	Date of creation
dtLastAccess	DT	Date of last access
dtLastModification	DT	Date of last modification

1.4.2.4.4 The library SysLibDirect.lib

The functions of this library serve to access variables by indices with which they are referenced in the runtime system. For detailed information see the description of the particular runtime system.

The user does not have to call a function! The functions will be called implicitely, according to the data type and the access mode of the variable, as soon as the library is linked to the project and a variable will be used in the program with the "#" like shown in the following:

iVar1 AT #MW17.4: INT;

The functions must be implemented in the runtime system as external C-functions.

The execution is synchronous.

1.4.2.4.5 The library SysLibEvent.lib

Overview

This library serves to synchronize and control the processing of two (IEC-) tasks.

A task waiting for an event can get re-activated by a second task which sets this event.

The functions are available to define, to delete and start an event resp. to set the timeout.

The execution is synchronous.

SysEventCreate

This function of type DWORD serves to create a new event and to name it. A handle will be returned, which is used by the other functions of the library to access the event.

Variable	Data Type	Description
stName	STRING	Name of the new event

SysEventDelete

This function of type BOOL deletes an event. The event is defined by the handle which was returned by the function SysEventCreate when creating the event. TRUE will be returned if as the event has been deleted successfully, otherwise FALSE.

Variable	Data Type	Description
DwHandle	DWORD	Event handle returned by SysEventCreate ↳ *Chapter 1.4.2.4.5.2 “SysEventCreate” on page 571*.

SysEventSet

This function of type DWORD is used to set an event. The event is defined by the handle which was returned by the function SysEventCreate when creating the event. TRUE will be returned if as the event has been set successfully, otherwise FALSE.

Variable	Data Type	Description
DwHandle	DWORD	Event handle returned by SysEventCreate ↳ *Chapter 1.4.2.4.5.2 “SysEventCreate” on page 571*.

SysEventWait

This function of type DWORD is used to set the timeout for an event. The event is defined by the handle which was returned by the function SysEventCreate when creating the event. TRUE will be returned if as the timeout has been set successfully, otherwise FALSE.

Variable	Data Type	Description
DwHandle	DWORD	Event handle returned by SysEventCreate *Chapter 1.4.2.4.5.2 “SysEventCreate” on page 571*.
dwTimeout	DWORD	Time in [ms], after which the function will return at the latest.

1.4.2.4.6 The library SysLibDPV1Hilscher.lib

Overview

Note: It depends on the target system, which system libraries can be used in the application program. Please see the document SysLibs_Overview.pdf.

This library supports the acyclic PROFIBUS DPV1, Class 1, Read- and Write-Services for the data transfer between master and slaves. The data are addressed within the slaves by slot and index. (Concerning this see the PB-DP standard.)

If supported by the target system the following function blocks are available:

- DPV1_Read, DPV1_ReadEx: Reading data
- DPV1_Write, DPV1_WriteEx: Writing data

Each module uses the following parameters:

Input variable	Data type	Description
ENABLE	BOOL	At a rising edge at this input the module starts processing.
Device	INT	Index of the Hilscher card, to which the request is given.
StationAddr	INT	Station address of the slave in the PROFIBUS.
Slot	INT	Data slot, for identification of the data within the slave.
Index	INT	Data index, for identification of the data within the slave.
Len	INT	Length of the data to be read/written in Bytes. Fed in the maximum length of the data buffer "buffer".
buffer	DWORD	Local address of the data. (Evaluate via using ADR().)

Output variable	Data type	Description
Ready	BOOL	The function block has finished processing.
State	V1State	Information on the state of the request (see below, Enumeration "V1State")
Size	INT	Length of the actually read/written data in case of successful execution.

Variable	Data type	Description
ENABLEOLD	BOOL	internal variable.
JobId	DWORD	internal variable.

Additionally the modules DPV1_ReadEx and DPV1_WriteEx have an output parameter Error, which tells about possibly occurred errors:

Output variable	Data type	Description
Error	ARRAY[0..7] OF BYTE;	Byte 1: Hilscher error code (see the documentation on the Hilscher PB-cards "Protocol Interface Manual PROFIBUS DP". Byte 2: Error class code. These values can be looked up in the PB standard. Bytes 3+4 (index 2+3): Slave-specific error information. These values can be looked up in the slave-specific documentation.

Enumeration V1State

This structure is used by the function blocks of the SysLibDPV1Hilscher.lib always in the output variable "state". It describes the current state of the request.

```
TYPE V1State :
(NotEnabled := 0, InvalidParam, Started, Done, DoneWithError );
END_TYPE
```

Meaning of the components:

NotEnabled := 0 - Function block currently not active

InvalidParam - Invalid input variable

Started - Function block has started execution

Done - Function block has terminated execution

DoneWithError - Function block has aborted execution with an error

1.4.2.4.7 The library SysLibFile.lib

Overview

This library supports a file system on the target computer. If the target supports the functionality, the library functions can be used to open, close, delete, rename, write to or read from files. Further functions are available for getting the file size or the date of the last access, as well as for reading or modifying the offset. The execution is synchronous.

If you are working with target system CODESYS SP RTE, please regard, that file access operations can affect the real-time behavior.

SysFileOpen

This function of type DWORD serves to open a file, which already exists or which should be created.

The return value is a file number, which will be used in the functions SysFileWrite, SysFileRead, SysFileClose as an input ('File'). In case of an error '0' will be returned resp. (watch out this exception!) '-1' by target CODESYS SP RTE.

Input-Variable	Data Type	Description
FileName	STRING	File name
Mode	STRING	Access mode: ● w write (File will be updated or created newly) ● r read (File will only be opened for reading; if the file does not exist, an error will be returned) ● rw read and write (File will be updated; if the file does not exist, an error will be returned) ● a append (File will be opened like described for 'w', but the written data will be appended at the end of the file)

SysFileWrite

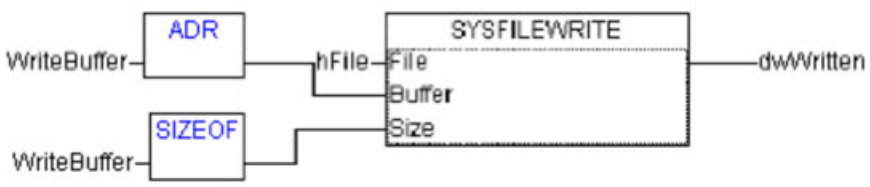
This function of type DWORD serves to write data to a file, which has been opened before by the function SysFileOpen. The return value is the number of successfully written bytes.

The return value is a file number, which will be used in the functions SysFileWrite, SysFileRead, SysFileClose as an input ('File'). In case of an error '0' will be returned resp. (watch out this exception!) '-1' by target CODESYS SP RTE.

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 "SysFileOpen" on page 573*
Buffer	DWORD	Address of the buffer (ascertainable by the ADR operator) of the file to which you want to write
Size	DWORD	Number of bytes, which you want to write to the file (ascertainable by the SIZEOF operator)

Example:

```
WriteBuffer : ARRAY[0..5] OF BYTE:=0,1,2,4,5,6;
DwWritten : DWORD;
hFile : DWORD;
```



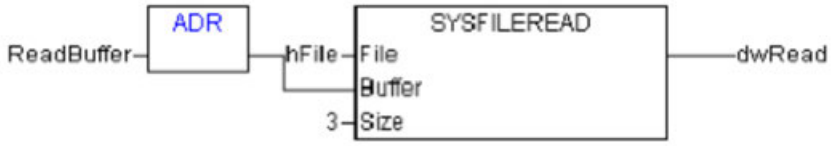
SysFileRead

This function of type DWORD serves to read a file, which has been opened before by SysFileOpen. The return value is the number of successfully read bytes.

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*
Buffer	DWORD	Address of the buffer which contains the data to be read (get this with the aid of the ADR operator)
Size	DWORD	Number of bytes to be read from the buffer

Example:

```
ReadBuffer : ReadBuffer:ARRAY[0..5] OF BYTE;
hFile : DWORD;
dwRead : DWORD;
```



SysFileClose

This function of type BOOL serves to close a file, which has been opened before by SysFileOpen. The return value is 1 (ok) or 0 (error).

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*

SysFileDelete

This function of type BOOL serves to delete a file. The return value is 1 (ok) or 0 (error).

Input-Variable	Data Type	Description
FileName	STRING	File name

SysFileCopy

This function of type UDINT serves to copy the file content to another file (different file name). It will return the number of actually copied bytes.

Input-Variable	Data Type	Description
FileDest	STRING	File to which you want to copy
FileSource	STRING	File from which you want to copy

SysFileEOF

This function of type BOOL will return 1, if the current offset is at the end of the file; it will return 0, if the end of file has not yet been reached.

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*

SysFileGetPos

This function of type DINT returns the currently set offset position in the file, which is identifed by the file number that you have got from the SysFileOpen function before ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*.

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*

SysFileGetSize

This function of type DWORD returns the size of the file (in Bytes), which is identified by the file name.

Input-Variable	Data Type	Description
FileName	DWORD	File name

SysFileGetTime

This function of type BOOL returns the creation date, the date of last access and the date of the last modification of the file (which is identified by the file name). The used format is DT. You get these data by accessing elements of the structure FILETIME.

The return value is 1(ok) or 0 (error).

Input-Variable	Data Type	Description
FileName	STRING	File name
ftFileTime	POINTER TO FILETIME	Points to the structure FILE-TIME; the operator ADR can be used for this purpose.

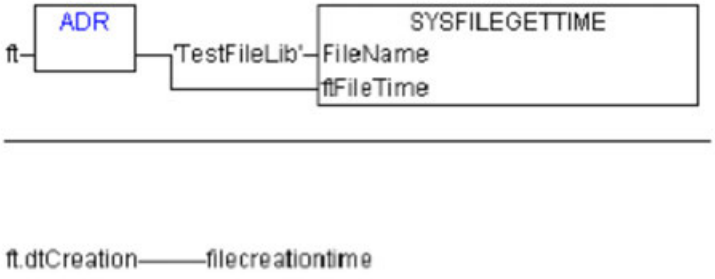
The structure FILETIME is defined as follows:

```
TYPE FILETIME
STRUCT
dtCreation:DT; (* Creation date *)
dtLastAccess:DT; (* Last access date; Attention: In VxWorks-systems possibly only day, without time! *)
dtLastModification:DT; (* Last modification date *)
END_STRUCT
END_TYPE
```

Example

For the file 'TestFile' the creation date is read:

```
Ft : FILETIME;
filecreationtime : DT;
```



SysFileRename

This function of type BOOL serves to rename a file.

It returns 1 (ok) or 0 (error).

Variable	Data Type	Description
FileOldName	STRING	Old file name
FileNewName	STRING	New file name

SysFileSetPos

This function of type BOOL serves to change the current offset (which can be read by the function SysFileGetPos ↳ *Chapter 1.4.2.4.7.9 “SysFileGetPos” on page 576*) for a file access. The file is identifed by the file number which has been got by SysFileOpen.

Input-Variable	Data Type	Description
File	DWORD	File number ↳ *Chapter 1.4.2.4.7.2 “SysFileOpen” on page 573*)
Pos	DWORD	Access offset within the file

1.4.2.4.8 The library SysLibFileAsync.lib

Overview

This library supports asynchronous file access from the IEC-application. If the target system supports the library, the following function blocks can be instanciated:

Attention: If you are working with target system CODESYS SP RTE, please regard, that file access operations can affect the real-time behavior!

The function blocks provided by this library have the following common parameters:

- Input parameter bEnable : BOOL

- Output parameter bDone : BOOL

- Output parameter bBusy : BOOL

- Output parameter bError : BOOL

- Output parameter wErrorId : WORD

All these function blocks start an action on a rising edge at the input parameter bEnable. After an action has been started, all these function blocks have to be called until the output parameter bDone is set. Now the outputs are valid.

The common parameters listed above will not explicitly be mentioned in the following descriptions of the particular function blocks.

The row I/O shows I for inputs, O for outputs.

SysFileOpenAsync

This function block is used for opening an existing file or creating a new one.

The output is hFile, a handle to the file. A file handle is an identifier for a file and is used as an input for other function blocks.

Input-Variable	Data type	Description
stFileName	STRING	File name
stMode	STRING	Mode to open the file: ● w write (file is created or overwritten) ● r read (file is only opened for reading if it exists) ● rw read and write (file is created if not existing, opened for reading if it exists) ● a append (file is opened for writing if it exists, created if it exists not, data is appended at the end always.)

SysFileCloseAsync

This function block is used to close a file. From now on the file handle is invalid and the file is free for other processes.

Variable	Datentyp	Beschreibung
hFile	DWORD	File's handle from SysFileOpenAsync ↳ *Chapter 1.4.2.4.8.2 "SysFileOpenAsync" on page 578*

SysFileWriteAsync

This function block is used for writing to a file. The file has to be opened with the function block SysFileOpenAsync ↳ *Chapter 1.4.2.4.8.2 "SysFileOpenAsync" on page 578*.

Variable	I/O	Data type	Description
hFile	E	DWORD	File identifier ↳ *Chapter 1.4.2.4.8.2 "SysFileOpenAsync" on page 578*)
pBuffer	E	DWORD	Address of the data to write (to be retrieved using ADR-operator)
dwSize	E	DWORD	Number of bytes to write (to be retrieved using SIZEOF-operator)
dwWrite	A	DWORD	Number of actually written bytes.

The data is written to the file in binary mode, means without any conversion.

SysFileReadAsync

This function block is used to read from an open file.

Variable	I/O	Data type	Description
hFile	I	DWORD	File identifier ↳ *Chapter 1.4.2.4.8.2 “SysFileOpenAsync” on page 578*)
pBuffer	I	DWORD	Address of the buffer to read to.
dwSize	I	DWORD	Number of bytes to read from the file to buffer.
dwRead	O	DWORD	Number of actually read bytes.

Parameter pBuffer must be evaluated via the ADR() operator. The files to be written are read binarily, i.e. without any conversion and copied to pBuffer.

SysFileDeleteAsync

This function block is used to delete a file.

Input-Variable	Data type	Description
stFileName	STRING	File name of the file to be deleted.

SysFileGetPosAsync

This function block retrieves the current read-/write position of the file.

Variable	Data type	Description
hFile	DWORD	File handle from SysFileOpenAsync ↳ *Chapter 1.4.2.4.8.2 “SysFileOpenAsync” on page 578*

SysFileSetPosAsync

This function block retrieves the current read-/write position of the file.

Variable	Data type	Description
hFile	DWORD	File number ↳ *Chapter 1.4.2.4.8.2 “SysFileOpenAsync” on page 578*)
dwPos	DWORD	Offset within the file which is valid for access.

SysFileEOFAsync

This function block is used to determine, whether the read-/write pointer has reached the end of the file.

Variable	I/O	Data type	Description
hFile	I	DWORD	File handle from SysFileOpenAsync
bEOF	O	DWORD	Tells if file handle is reached

SysFileGetSizeAsync

This function block is used to retrieve the size in bytes of a file.

Variable	I/O	Data type	Description
stFileName	I	STRING	File name
dwSize	O	DWORD	Size of the file in bytes

SysFileGetTimeAsync

This function block retrieves the modification times of a file.

Variable	I/O	Data type	Description
stFileName	I	STRING	File name
ftFileTime	O	POINTER TO FILETIME	Points to a FILETIMEAsync-structure; the ADR can be used

Structure FILETIMEASYNC (is included in the library):

```
TYPE FILETIMEASYNC
STRUCT
 dtCreation:DT;  (* Erstelldatum *)
 dtLastAccess:DT;  (* Datum letzter Zugriff *)
 dtLastModification:DT; (* Datum letzte Änderung *)
END_STRUCT
END_TYPE
```

SysFileCopyAsync

This function block is used to copy a file to another name/location.

Variable	I/O	Data type	Description
stFileDest	I	STRING	Name of destination file
stFileSource	I	STRING	Name of source file
dwCopied	O	DWORD	Number of copied bytes.

SysFileRenameAsync

This function block is used to rename a file.

Variable	Data type	Description
stFileOldName	STRING	Old file name
stFileNewName	STRING	New file name

SysFileCloseAllOpenAsync

With this function block one can close all currently open files, without knowing any handles or names. The system always knows these handles.

1.4.2.4.9 The library SysLibFileStream.lib

This library provides functions which correspond to ANSI C functions for file stream operations. The execution is synchronous.

The functions:

SysLibFileStream function	ANSI C Function	Data Type	Description
SysFileStreamFOpen	*fopen(char *file-name, char *mode);	DWORD	File with name file-name will be opened as stream; possible values for inputvari-able Mode: 'w' (write), 'r' (read), 'a' (append), '+', 'b', 't'
SysFileStream-Clearerr	clearerr(FILE* pFile);	DINT	internal error state of pFile will be deleted; always returns 1
SysFileStreamFClose	fclose(FILE *pFile);	DINT	all open streams will be closed (except for stdin, stdout, stderr). Returns SysFileS-treamFClose_EOF in case of error, other-wise 0.
SysFileStreamFEOF	*feof(FILE* pFile);	DINT	returns !=0, as soon as end of file in pFile is reached

SysLibFileStream function	ANSI C Function	Data Type	Description
SysFileStreamFError	ferror(FILE* pFile);	DINT	returns !=0, as soon as an error has been detected for pFile
SysFileStreamFFlush	fflush(FILE *pFile);	DINT	Characters which are still buffered internally, will be output
SysFileStreamRemove	remove(char* filename);	BOOL	File will be deleted; returns 1 for OK, 0 in case of an error
SysFileStreamRename	rename(char* filename);	BOOL	Renaming a file; returns 1 for OK, 0 in case of an error
SysFileStreamRewind	rewind(FILE* pFile);	DINT	sets file position to start and deletes internal error state; always returns 1
SysFileStreamFGetC	fgetc(FILE *pFile);	DINT	returns the next character in the stream (0--255, SYSFILESTREAM_EOF in case of an error
SysFileStreamFGetPos	*fgetpos(FILE* pFile, fpos_t * ptr);	DINT	writes current file position of pFile to ptr; fpos_t here defined as an unsigned long (32 bits)
SysFileStreamFSetPos	fsetpos(FILE* pFile, fpos_t * ptr);	DINT	sets file position of pFile according to ptr; fpos_t here is defined as unsigned long (32 bits); pFPos:DWORD; (* pointer !!*)
SysFileStreamFGetS	* fgets(char * str, int n, FILE * pFile);	POINTER TO STRING	Reads at most the next n-1 characters into the array s, (termination automatically with 0); Truncation at '\n', the '\n' will be taken over to s; Return value: s resp. 0 (at end of file or error)
SysFileStreamFPrintf_Int	fprintf(FILE* pFile, char* szFormat, intnArg);	DINT	formatted output in stream pFile; Restrictions compared to C:only 1argument of type INT/DINT etc. can be printed; szFormat should be e.g. '%d'

SysLibFileStream function	ANSI C Function	Data Type	Description
SysFileStreamF-Printf_Real	fprintf(FILE* pFile, char* szFormat, float fArg);	DINT	formatted output in stream pFile; Restrictions compared to C:only 1argument of type REAL etc. can be printed; szFormat should be e.g. '%f'
SysFileStreamF-Printf_String	fprintf(FILE* pFile, char* szFormat, char *pcArg);	DINT	formatted output in stream pFile; Restrictions compared to C:only 1argument of type STRING etc. can be printed; szFormat should be e.g. '%s'
SysFileStreamFPutC	fputc(int c, FILE *pFile);	DINT	Writing character (unsignedchar) c to stream pFile Returns c (converted to DINT) or SYSFI-LESTREAM_EOF in case of an error
SysFileStreamFPutS	fputs(char* str, FILE * pFile);	DINT	Writing string s in stream pFile Returns str (pointer to string) or SYSFI-LESTREAM_EOF in case of an error
SysFileStreamFRead	fread(void* ptr, size_t size, size_t nobj, FILE* pFile);	DWORD	nobj objects of size size will be read from pFile to ptr; Returns number of read objects
SysFileStreamFWrite	fwrite(void* ptr, size_t size, size_t nobj, FILE* pFile);	DWORD	nobj objects of size size wil lbe written from ptr to pFile; Returns number of written objects
SysFileStreamFS-canf_Int	fscanf(FILE* pFile, char* szFormat, int * pnArg);	DINT	formatted input from stream pFile; Restrictions compared to C: only 1 DINT argument can be read; szFormat should be e.g. '%d'
SysFileStreamFS-canf_String	fscanf(FILE* pFile, char* szFormat, char *pcArg);	DINT	formatted input from stream pFile; Restrictions compared to C: only 1 STRING argument can be read; szFormat should be e.g. '%s'

SysLibFileStream function	ANSI C Function	Data Type	Description
SysFileStreamFS-canf_Real	fscanf(FILE* pFile, char* szFormat, float* pfArg);	DINT	formatted input from stream pFile; Restrictions compared to C: only 1 REAL argument can be read; szFormat should be e.g. '%f'
SysFileStreamF-SeekÂ	fseek(FILE* pFile, long offset, int origin);	DINT	sets file position on offset Bytes based on origin; values for origin: SEEK_SET=Start of file, SEEK_CUR=current position; SEEK_END=End of file; 0=OK
SysFileStreamFTell	ftell(FILE* pFile);	DINT	returns current file position (based on file start) in Bytes (-1 in case of error)

1.4.2.4.10 The library SysLibGetAddress.lib

Overview

This library provides functions which - if supported by the target system - return the start address and the size of a data segment (Memory, Input, Output, Retain or Global) in a DWORD. The data segment is to be specified via its number, which is defined in the enumeration.

SysLibGetSize

This function returns the start address and the size of a data segment (Memory, Input, Output, Retain or Global) in a DWORD.

The data segment is to be specified via its number, which is defined in the enumeration ADDRESS_SEGMENTS.

Input-Variable	Data type	Description
iSegment	INT	Number of the data segment, see 'Enumeration ADDRESS_SEGMENTS' ↳ *Chapter 1.4.2.4.10.3 “Enumeration ADDRESS_SEGMENTS” on page 585*

Enumeration ADDRESS_SEGMENTS

This enumeration defines the number of a data segment (Memory, Input, Output, Retain or Global), which is needed as an input for the functions.

TYPE ADDRESS_SEGMENTS :

(

DATAID_MEMORY, (0)
DATAID_INPUT, (1)
DATAID_OUTPUT, (2)
DATAID_RETAIN, (3)
DATAID_GLOBVARS (4)
);
END_TYPE

1.4.2.4.11 The library SysLibIECTasks.lib

Overview

This library can be used to call information on the configuration of IEC tasks. To create, delete, prioritize, stop and restart a task you can use the library SysLibTasks.lib.

The execution is synchronous.

SysIECTaskGetConfig

This function of type BOOL serves to retrieve the configuration parameters of an IEC task.

The task is addressed by its name or its index, which it has got assigned in the task configuration. The structure SysIECTaskConfEntry contains all parameters which are used in the task configuration.

As soon as the task has been found, TRUE will be returned, otherwise FALSE.

Input Variable	Data Type	Description
udiTaskId	UDINT	Task Id (Index in the task configuration)
pTaskInfo	POINTER TO SYSIECTASK-CONFENTRY	Information on the task configuration (structure, see below)

Structure SysIECTaskConfEntry:

TYPE SYSIECTASKCONFENTRY :
STRUCT

byTaskNr	USINT;	(* Task Number *)
byPriority	USINT;	(* Priority, see Dialog 'Taskattributes' *)
lInterval	DINT;	(* Interval of cyclic tasks, see Dialog 'Taskattributes' *) (* (in this case ldrEvent has an invalid entry) *)
ldrEvent	LDATAREF_TYPE;	(* ldrEvent represents address of event variable (by POURef, offset and size) *)

wIndex	UINT;	(*If POU is called by task, then special wrapper _Call-Task<TaskName> (also POU) is created. *) (*Returned Index corresponds to index of this POU. *)
uiNameLen	UDINT;	(* ulNameLen has a max. length of 32 characters (max. task name length) *)
szName	STRING(80);	(* Name of the task, see Dialog 'Taskattributes' *)

END_STRUCT

END_TYPE

Structure LdataRef_Type:

TYPE LDATAREF_TYPE :

STRUCT

POURef	UINT;	POU-ID of the event variable
Offset	UDINT;	Offset for the event variable
Size	UDINT;	Size of the event variable

END_STRUCT

END_TYPE

SysIECTaskGetInfo

This function of type BOOL returns the current time values of an IEC task.

The task is identified by the task name or by the index, it has got in the task configuration.

As soon as the task has been found, TRUE will be returned, otherwise FALSE.

Input-Variable	Data Type	Description
stTaskName	STRING	Name of the task
pTaskInfo	POINTER TO SYSIECTAS-KINFO	Pointer to current data of the IEC task (structure, see below)

Structure SysIECTaskInfo:

TYPE SYSIECTASKINFO :

STRUCT

dwCount :	DWORD;	(* Number of cycles since start of task *)
dwCycleTime :	DWORD;	(* Current cycle time *)
dwCycleTimeMin :	DWORD;	(* Minimum cycle time *)
dwCycleTimeMax :	DWORD;	(* Maximum cycle time *)
dwCycleTimeAvg :	DWORD;	(* Average cycle time *)

wStatus :	WORD;	For cyclic task = 0 (PLC RUN) and for event triggered task wStatus = 1 (PLC STOP)
wMode :	WORD;	wMode = 1 (task mode running) for both tasks, although the event triggered task isn't running

END_STRUCT

END_TYPE

SysIECGetFctPointer

This auxiliary function of type DWORD returns a function pointer, which is required as input parameter for the function which is used to create a new task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629* *Chapter 1.4.2.4.26 “The library SysLibTasks.lib” on page 628*.

The function requires as an input parameter the internal index of the POU, which should be called by the task. This index can be acquired with the aid of the INDEXOF operator.

Input-Variable	Data Type	Description
wIndexOf	WORD	Internal index of the POU, which is to be called by the task.

SysIECTaskResetEvent

This auxiliary function of type BOOL resets the event variable of an event triggered IEC task.

The function has no input parameter. It is working on the current task. It returns TRUE in case of success, otherwise 0 FALSE (e.g. if the task is not an event triggered task).

The function sets the BOOLean IEC-variable, which is used as an event, to FALSE, and the internal flag of the runtime system task management to 0.

So it is achieved that a rising edge of the event variable will be regarded at the next cycle of the scheduler.

1.4.2.4.12 The library SysLibInit.lib

Overview

This library contains a function which can be used to initialize an external library, which is available as an obj-file. The execution is synchronous.

To use the functionality "Initializing of special external object libraries", include the library in the Automation Builder project. It is not necessary to call a function explicitely.

SysInitLibrary

This function of type BOOL can be used to initialize an external library. The index of the function <LibName>SetPointer will be handed over.

The function call is created automatically. Do not call the function explicitely. Include the library in the Automation Builder project.

TRUE is returned after the operation ha been finished successfully, otherwise FALSE.

Input Variable	Data Type	Description
index	DINT	Index of the function <Lib-Name>SetPointer.

1.4.2.4.13 The library SysLibInt.lib

Overview

If the target system supports this functionality you can use this library to set and remove an interrupt handler for a function. The execution is synchronous.

SysInstallHandler

This function of type BOOL sets the interrupt with a given number on the function which is identified by its address. This address can be retrieved by the SysIECGetFctPointer function (see Library SysLibIECTasks.lib) ↳ *Chapter 1.4.2.4.11.4 “SysIECGetFctPointer” on page 588*.

The return value is TRUE or FALSE depending on the success of the operation.

The interrupt handler can be removed by the SysRemoveIntHandler function.

Input Variable	Data Type	Description
iInterrupt	INT	Interrupt number
dwFctAddress	DWORD	Function pointer, retrieved with the aid of SysIEC-GetFctPointer (SysLibIEC-Tasks.lib) ↳ *Chapter 1.4.2.4.11.4 “SysIECGetFct-Pointer” on page 588*

SysRemoveHandler

This function of type BOOL removes the interrupt, which is identifed by the given number, for the function which is identified by its address. This address can be retrieved by the function SysIECGetFctPointer (Library SysLibIECTasks.lib). The interrupt handler is set by the SysInstallHandler function ↳ *Chapter 1.4.2.4.13.2 “SysInstallHandler” on page 589*.

The return value is TRUE or FALSE depending on the success of the operation.

Input Variable	Data Type	Description
iInterrupt	INT	Interrupt number
dwFctAddress	DWORD	Function pointer, retrieved with the aid of SysIEC-GetFctPointer (SysLibIEC-Tasks.lib) *Chapter 1.4.2.4.11.4 “SysIECGetFct-Pointer” on page 588*

1.4.2.4.14 The library SysLibMem.lib

Overview

This library can be used for memory management. The library functions are available to allocate, to free, to define, to compare memory locations and to copy, move or swap between different memory locations. The execution is synchronous.

SysMemAlloc

This function of type DWORD is used to dynamically allocate memory space.

The return value is either the pointer on the allocated memory location or it is 0, in case there is not as much space available as requested. This return value always should be checked, even if just a small memory area is to be allocated.

This function is not supported by CODESYS SP RTE.

Variable	Data Type	Description
dwSize	DWORD	Number of bytes to be allocated.

SysMemFree

This function of type BOOL is used to deallocate memory space.

The return value is TRUE or FALSE depending on the success of the operation.

This function is not supported by CODESYS SP RTE.

Variable	Data Type	Description
dwAddress	DWORD	Address of the memory space which is currently allocated ↳ *Chapter 1.4.2.4.14.2 “SysMemAlloc” on page 590*.
dwSize	DWORD	Number of bytes to get reallocated.

SysMemCmp

This function of type DWORD compares the content of two memory buffers of size dwCount. dwBuf1 and dwBuf2 each show the start address of the buffer.

The function returns the difference of the buffer area contents:

< 0 buf1 smaller than buf2

0 buf1 is of equal size as buf2

> 0 buf1 bigger than buf2

Variable	Data Type	Description
dwBuf1	DWORD	Address of memory buffer 1 (buf1)
dwBuf2	DWORD	Address of memory buffer 2 (buf2)
dwCount	DWORD	Number of memory bytes which should be compared

SysMemCpy

This function of type DWORD is used to copy a defined number of memory locations from one buffer to another. The function will return the pointer to the address of the destination buffer area.

The difference to SysMemMove is that you only can copy between two non-adjoining buffers.

Variable	Data Type	Description
dwDest	DWORD	Address of destination buffer
dwSrc	DWORD	Address of source buffer
dwCount	DWORD	Number of memory locations to be copied

SysMemMove

This function of type DWORD moves one memory buffer to another. The function will return the address of the destination buffer.

The difference to SysMemCpy is that this function allows to copy even memory areas which are adjoining or even overlapping.

Variable	Data Type	Description
dwDest	DWORD	Address of destination buffer
dwSrc	DWORD	Address of source buffer
dwCount	DWORD	Number of memory locations to be moved

SysMemSet

This function of type DWORD can be used to initialize a memory location with a defined value. It will return the address of the destination buffer.

Variable	Data Type	Description
dwDest	DWORD	Pointer to the address of the memory location which should be initialized
bCharacter	BYTE	Character or numeric value with which the memory location should be initialized
dwCount	DWORD	Number of memory locations in bytes

SysMemSwap

This function of type BOOL can be used to swap data.

It is used on Motorola byte order systems (PPC) to swap from Motorola to Intel byte order. On Intel systems (ARM, MIPS, SH, x86) the function has no effect. This allows to write portable libraries.

The function will return TRUE if the operation has been terminated successfully, otherwise FALSE.

The return value does not indicate whether the operation was performed or not.

Variable	Data Type	Description
dwAddress	DWORD	Address of the memory buffer to be swapped
diSwapSize	DINT	Number of locations to be swapped: 2,4,8
diSwapElements	DINT	Number of elements in the memory area to be swapped

1.4.2.4.15 The library SysLibPciCards.lib

Overview

This library is designed for access to a Pci card plugged to the Plc. If the system supports the functionality, the function SysPciGetCardInfo can be used to retrieve information about a Pci card *Chapter 1.4.2.4.15.2 “Function SysPciGetCardInfo” on page 593*.

Function SysPciGetCardInfo

This function is for retrieving information about a Pci card.

The return value is a pointer to a Structure PCI_INFO structure *Chapter 1.4.2.4.15.3 “Structure PCI_INFO” on page 593*. The structure is defined in the library too. The structure contains all the Pci configuration registers. If the specified card is not found, the function will return zero.

Input Variable	Data type	Description
usVendorId	WORD	The vendor ID of the card.
usDeviceId	WORD	The device ID of the card.
usCardIndex	WORD	The card’s index with the specified vendor ID and device ID. The index is zero-based. If there are more than one cards with this vendor and device ID, one can use the index to distinguish the cards.

Structure PCI_INFO

The structure has the following members:

Member	Data type
usVendorID	WORD
usDeviceID	WORD
usSubVendorID	WORD
usSubSystemID	WORD
ulBusNr	DWORD
SlotNr	DWORD
ulFunction	DWORD
ulBaseAddresses	ARRAY[0..5] OF DWORD
byInterrupt	BYTE
DeviceSpecific	ARRAY[0..191] OF BYTE

The structure is set up by the Pci Bios. The access is read-only, do never write to this structure.

1.4.2.4.16 The library SysLibPlcConfig.lib

Overview

This library supports the reading of the configuration data of the PLC Configuration. These data are also loaded to the controller at a download of the application and are written to structures by the runtime system. The library offers functions for getting pointers on these structures. The execution is synchronous.

Due to the fact that pointers on original structures of the runtime system are provided, the following is:

- The structure (pointer to sub-elements) may not be modified!
- If default values of parameters in the structures get modified, this will be of on effect!

CfgCCGetError

Currently not yet implemented in the runtime system. Error code always 0.

This function provides information on the errors which occur during the download of the PLC configuration.

The function returns a pointer to Structure CCLoadError ↳ *Chapter 1.4.2.4.16.7 “Structure CCLoadError” on page 595*.

CfgCCGetHeader

This function returns a pointer on the header structure of the PLC configuration ↳ *Chapter 1.4.2.4.16.8 “Structure CCHeader” on page 595*.

CfgCCGetRootModule

This function provides information on the root module of the PLC configuration. It returns a pointer to Structure CCLoadError ↳ *Chapter 1.4.2.4.16.9 “Structure CCModule” on page 595*.

CfgCCGetRootModuleByModuleId

This function provides information on the root module of the currently used PLC configuration, which is given by the module Id. The module Id is defined in the configuration file by entry "Id".

The function returns a pointer to Structure CCModule (see above, function CfgCCGetRoot-Module ↳ *Chapter 1.4.2.4.16.4 “CfgCCGetRootModule” on page 594*) ↳ *Chapter 1.4.2.4.16.9 “Structure CCModule” on page 595*.

Variable	Data Type	Description
ulModuleId	UDINT	Module id of the root module.

CfgCCGetRootModuleByNodeId

This function provides information on the root module of the currently used PLC configuration, which is given by the node Id. The node Id of the module normally results from the position of the module within the PLC Configuration.

The function returns a pointer to Structure CCModule (see above, function CfgCCGetRoot-Module ↳ *Chapter 1.4.2.4.16.4 “CfgCCGetRootModule” on page 594*) ↳ *Chapter 1.4.2.4.16.9 “Structure CCModule” on page 595*.

Variable	Data Type	Description
ulNodeId	UDINT	Node id of the root module.

Structure CCLoadError

This structure provides information on the last error during the download of the PLC configuration data. It is accessed by function CfgCCGetError ↳ *Chapter 1.4.2.4.16.2 “CfgCCGetError” on page 594*.

The components:

Variable	Data Type	Description
ulLastError	UDINT	Error code of the last error
ulAddInfo1	UDINT	According to ulLastError, the meaning changes.
ulAddInfo2	UDINT	According to ulLastError, the meaning changes.
szLastError	STRING(32)	Last error message. A possibility to make debugging easier

Structure CCHeader

This structure provides information on the header structure for the PLC configuration which has been loaded on the target system. It can be accessed by function CfgCCGetHeader ↳ *Chapter 1.4.2.4.16.3 “CfgCCGetHeader” on page 594*.

The components:

Variable	Data Type	Description
szTag	STRING(10)	zero-terminated STRING "CommConf"
cByteOrder	BYTE	The file data are in Intel ('I') or Motorola format ('M')
ulSize	UDINT	Size of the following data
lVersion	UDINT	Version number of the file

Structure CCModule

This structure provides information on the module, which e.g. can be accessd via function CfgCCGetRootModule ↳ *Chapter 1.4.2.4.16.4 “CfgCCGetRootModule” on page 594*.

The components:

Variable	Data Type	Description
ucEntryTag	BYTE	'M' = Module
ucDummy1	BYTE	
ucDummy3		
ulModuleId		Id of the module given in the configuration file *.cfg
sModuleNumber		Number of the module in the parent module (-1 if root)
usModuleTag		Describes the kind of the module (0=3S-Module, 1=DP-Master, 2=DP-Slave, 3=CAN-Master, 4=CAN-Slave, 5=DP-SingleSlave)
byDeviceDriver	BYTE	The module needs a device driver (0=FALSE, 1=TRUE)
ucDummy4	BYTE	
ucDummy5	BYTE	
ucDummy6	BYTE	
ulNodeId	UDINT	NodeId of the module
byDefinedWithStruct:	BYTE	The module was defined with a structure (0=FALSE, 1=TRUE)
ucDummy7	BYTE	
ucDummy8	BYTE	
ucDummy9	BYTE	
ulBitOffsetInput	UDINT	Offset of the modules input area
ulBitSizeInput	UDINT	Size of the modules input area in bit
ulBitOffsetOutput	UDINT	Offset of the modules output area
ulBitSizeOutput	UDINT	Size of the modules output area in bit
ulRefIdCommonDiag	UDINT	RefId of the modules common diagnosis area
ulBitOffsetCommonDiag		Offset of the modules common diagnosis area
ulBitSizeDiag	UDINT	Size of the modules diagnosis area in bit
usParameterCount	UINT	Number of parameters
usDummy	UINT	

Variable	Data Type	Description
ppccpModuleParams	POINTER TO POINTER TO ccParam	<ccParam [0..usParameter-Count]> a pointer to an array of pointers to CCMo-duleParam-structures. (Defini-tion of structure CCParam see below). Dereferencing the pointer with ppccpModuleParams^ gives you the pointer to the first parameter structure. (ppccpModuleParams+4)^ gives you the pointer to the next parameter structure. See also comment (*Read pointer to parameters *) in example project.
ulSizeOfSpecificData	UDINT	Size in bytes of the module specific data
pModuleData	POINTER TO BYTE	<MODULE_SPE-CIFIC_DATA> Here the data, according to usModuleTag is located: pModuleData is possible to be a pointer to PBSlave, CANSlave, PBMaster, PBSlave, PBSingleSlave, see definitions below.
usChannelCount:	UINT	Number of configured chan-nels
usModuleCount	UINT	Number of configured modules
In the following the Channels and Modules of this Module in the configured order are located! (DP-Slaves are ordered by the stationnumber!) This means, it is possible that another CCModule structure is inserted here.		

Variable	Data Type	Description
ppcccChannels	POINTER TO POINTER TO ccChannel;	<ccChannel [0..usChannelCount]> Definition of structure CCChannel see below * Dereferencing the pointer with ppccpChannels^ gives you the pointer to the first parameter structure. (ppccpChannels+4)^ gives you the pointer to the next parameter structure. See also comment "(*Read pointer to parameters *)" in example project.
ppccmSubModules	POINTER TO POINTER TO BYTE	<ccModule [0..usModuleCount]> Points to an array of variables of type POINTER TO ccModule. To view the contents, you have to assign the value to a variable of type "POINTER TO CCModule". Definition of structure CCModule see below. Dereferencing the pointer with ppccpSubModules^ gives you the pointer to the first parameter structure. (ppccpSubModules+4)^ gives you the pointer to the next parameter structure. See also comment "(*Read pointer to parameters *)" in example project.

Structure CCChannel

This structure provides information on the channel of a module; see usage in Structure CCModule ↳ *Chapter 1.4.2.4.16.9 “Structure CCModule” on page 595*.

Variable	Data Type	Description
ucEntryTag	BYTE	C' = Channel
ulChannelId	UDINT	Id of the channel given in the configuration file
usChannelNumber	UINT	Number of the channel in the parent module
ulRefId	UDINT	Direction of the channel (1=input, 2=output, 3=input AND output)
usChannelType	UINT	TYPE of the channel (coded as "TypeClass")
ulBitOffset	UDINT	Offset of the channel in in-/output area

Variable	Data Type	Description
usParameterCount	UINT	Number of parameters
ppccpParams	POINTER TO POINTER TO CCParam	PARAMETER[1..usParameterCount]> POINTER TO an ARRAY OF pointers TO Structure CCParam ↳ *Chapter 1.4.2.4.16.11 “Structure CCParam” on page 599*.

Structure CCParam

This structure provides information on a channel of the module; see usage in Structure CCModule and Structure CCChannel ↳ *Chapter 1.4.2.4.16.9 “Structure CCModule” on page 595* ↳ *Chapter 1.4.2.4.16.10 “Structure CCChannel” on page 598*.

Variable	Data Type	Description
ulParameterId	UDINT	Id of the parameter given in the configuration file *.cfg
usParameterNumber	UINT	Number of the parameter in the module
byReadOnly	BYTE	1=TRUE, 0=FALSE
byDummy	BYTE	
usParameterType	UINT	Type of the parameter; coded as "TypeClass"
usDummy	UINT	
ulSize	UDINT	Size of the parameter in bytes
byValue	BYTE	The memory representation of the parameter value starts with this byte. The other bytes follow immediately, if the size of the parameter value is bigger than 1.

1.4.2.4.17 The library SysLibPlcCtrl.lib

Overview

This library contains the following functions for controlling a PLC. The execution is synchronous.

- ↳ *Chapter 1.4.2.4.17.2 “SysStartPlcProgram” on page 600*
- ↳ *Chapter 1.4.2.4.17.3 “SysResetPlcProgram” on page 600*
- ↳ *Chapter 1.4.2.4.17.4 “SysStopPlcProgram” on page 600*
- ↳ *Chapter 1.4.2.4.17.5 “SysShutdownPlc” on page 600*
- ↳ *Chapter 1.4.2.4.17.6 “SysEnableScheduling” on page 601*
- ↳ *Chapter 1.4.2.4.17.7 “SysGetPlcLoad” on page 601*

Additionally there are functions to handle the retain variables:

- ↳ *Chapter 1.4.2.4.17.9 “SysRestoreRetains” on page 601*
- ↳ *Chapter 1.4.2.4.17.8 “SysSaveRetains” on page 601*

as well as a function for activating the watchdog:

- ↳ *Chapter 1.4.2.4.17.10 “SysWdgEnable” on page 602*

SysStartPlcProgram

This function of type BOOL can be used to start a PLC. It will return TRUE in case of success, otherwise FALSE.

Input-Variable	Data type	Description
bDummy	BOOL	Without function

SysResetPlcProgram

This function of type BOOL can be used to reset the PLC. The reset mode is set with the aid of the enumeration Reset_Mode. The function will return always TRUE.

This function is not synchronous, but it creates a task. The priority of the task is lower than the lowest user task.

The function may not be called in a callback, especially in no one where user tasks are created or destroyed, for example EVENT_BEFORE_RESET, EVENT_AFTER_RESET, EVENT_SHUTDOWN, EVENT_STOP.

Variable	Data type	Description
rmRESETMODE	RESET_MODE	Choose one of the enumeration values to give the desired reset command to the PLC:
0=RESET_WARM, 1=RESET_COLD, 2=RESET_HARD;	RESET_WARM corresponds with the 'Reset' command in the Online Menu, RESET_HARD corresponds with the 'Reset (original)'	

SysStopPlcProgram

This function of type BOOL can be used to stop the PLC. It will return TRUE in case of success, otherwise FALSE.

Input Variable	Data type	Description
bDummy	BOOL	Without function

SysShutdownPlc

This function of type BOOL can be used to shut down the PLC. It will return TRUE in case of success, otherwise FALSE.

Variable	Data type	Description
bDummy	BOOL	Without function.

SysEnableScheduling

This function of type DWORD can be used to enable resp. disable the scheduler for the IEC tasks in the PLC.

Variable	Data type	Description
bEnable	BOOL	Without function

SysGetPlcLoad

This function of type DWORD returns the current processor load of the IEC tasks.

Variable	Data type	Description
bDummy	BOOL	Without function

SysSaveRetains

This function of type DINT can be used to save the values of retain variables in a file. One of the following values will be returned:

1: OK

0: No program loaded

-1: The given file could not be opened

Input Variable	Data type	Description
stFileName	STRING	Name of the file where you want to save the retain variables

SysRestoreRetains

This function of type DINT can be used to to restore the values of retain variables, which have been saved in a file *Chapter 1.4.2.4.17.8 “SysSaveRetains” on page 601*. One of the following values will be returned:

1: OK

0: No program loaded

-1: The given file could not be opened

-2: The content of the file exceeds the size of the retain area

Input Variable	Data type	Description
stFileName	STRING	Name of the file which contains the retain variables *Chapter 1.4.2.4.17.8 “SysSaveRetains” on page 601*

SysWdgEnable

This function of type BOOL can be used to activate resp. deactivate the watchdog for a specified task. It will return TRUE in case of success, otherwise FALSE.

Variable	Data type	Description
bEnable	BOOL	if TRUE: The watchdog functionality gets activated if FALSE: The watchdog functionality gets deactivated
byIECTaskIndex	BYTE	Index of the IEC task, for which the watchdog should be activated/deactivated
stIECTaskName	POINTER TO STRING	Name of the IEC task, can be a pointer to zero

1.4.2.4.18 The library SysLibPorts.lib

Overview

This library can be used to communicate with external hardware devices via their port addresses; e.g. real-time clock, graphic controller etc. The port addresses can accessed reading and writing. The execution is synchronous.

SysPortIn

This function of type BYTE returns the byte value at that port address, which has been passed on by wPort.

Variable	Data Type	Description
wPort	WORD	Port address of the hardware device.

SysPortInW

This function of type WORD returns the byte value at that port address, which has been passed on by wPort.

Variable	Data Type	Description
wPort	WORD	Port address of the hardware device.

SysPortInD

This function of type DWORD returns the byte value at that port address, which has been passed on by wPort.

Variable	Data Type	Description
wPort	DWORD	Port address of the hardware device.

SysPortOut

This function of type BOOL writes the BYTE value which is passed on by byData to that port address which is passed on by wPort. The function returns TRUE if the operation has been terminated successfully, otherwise FALSE.

Variable	Data Type	Description
wPort	WORD	Port address of the hardware device.
byData	BYTE	Value to be written to the port address.

SysPortOutW

This function of type BOOL writes the WORD value which is passed on by byData to that port address which is passed on by wPort. The function returns TRUE if the operation has been terminated successfully, otherwise FALSE.

Variable	Data Type	Description
wPort	WORD	Port address of the hardware device.
byData	WORD	Value to be written to the port address.

SysPortOutD

This function of type BOOL writes the DWORD value which is passed on by byData to that port address which is passed on by wPort. The function returns TRUE if the operation has been terminated successfully, otherwise FALSE.

Variable	Data Type	Description
wPort	WORD	Port address of the hardware device.
byData	DWORD	Value to be written to the port address.

1.4.2.4.19 The library SysLibProjectInfo.lib

Overview

The functions contained in this library can be used to read the Menu 'Project' 'Project Info' respectively the function SysGetProjectID *Chapter 1.4.2.4.19.3 “Structure PROJECT_INFO” on page 604* *Chapter 1.4.2.4.19.4 “Function SysGetProjectID” on page 604.

The execution is synchronous.

Function SysGetProjectInfo

This function of type BOOL provides the components of the project info which was entered in the programming system ('Project' 'Project Info'); the structure PROJECT_INFO is used for that purpose.

TRUE is returned if the operation has been successful, otherwise FALSE.

Input Variable	Data type	Description
ProjectInfo	POINTER TO PROJECT_INFO	Pointer on the project information which is stored in the structure PROJECT_INFO *Chapter 1.4.2.4.19.3 “Structure PROJECT_INFO” on page 604* The ADR operator can be used to retrieve the offset

Structure PROJECT_INFO

The components of this structure show the project info ('Project' 'Project Info'). The structure is used by the function SysGetProjectInfo *Chapter 1.4.2.4.19.2 “Function SysGetProjectInfo” on page 604*.

Component	Data type	corresponding field in the dialog 'Project Info'
dtDate	DT	'Change date:'
stProject	STRING(255)	'File:'
stTitle	STRING(255)	'Title:'
stVersion	STRING(255)	'Version:'
stAuthor	STRING(255)	'Author'
stDescription	STRING(255)	'Description:'

Function SysGetProjectID

This function of type DWORD returns the project ID. It is used to determine whether the project in the editor and the project on the controller are identical, whether an online change can be performed, or whether they are different projects.

The project ID is also stored in the symbol file. Using this function a visualization can check if the symbol file fits to the project on the controller.

The return value contains the project ID.

The function is supported by runtime systems CSP32F 2.4.5.0 and newer.

Input Variable	Data type	Description
---	---	---

1.4.2.4.20 The library SysLibRtc.lib

The library SysLibRtc.lib

This library contains functions for accessing the real-time clock of the local system. If the target system is supporting the functionality, the real-time clock can be read and set; additionally the current hour display mode as well as the battery can be retrieved. The execution is synchronous.

In this context regard the RTC function, which is part of the standard.ibrary and which returns the running date and time referring to a given start time.

RTC is not a real real-time clock function because the start time must be set explicitly. But using RTC will save system operating time. Think about setting the start time for RTC with the aid of the SysRtcGetTime function *Chapter 1.4.2.4.20.4 “SysRtcGetTime” on page 605.*

SysRtcCheckBattery

This function of type BOOL checks the status of the battery of the computer, which is important for the exactness of the shown clock time.

The function returns 0, if the battery is not ok, otherwise 1.

Variable	Data Type	Description
bDummy	BOOL	TRUE starts the function.

SysRtcGetHourMode

This function of type BOOL can be used to read the display mode of the real-time clock of the local system.

The function returns 0 in case of 12-hour mode, it returns 1 in case of 24-hour mode .

Variable	Data Type	Description
bDummy	BOOL	TRUE starts the function.

SysRtcGetTime

This function of type DATE_AND_TIME returns the current time which is read from the PC clock.

Variable	Data Type	Description
dummy	BOOL	TRUE starts the function.

SysRtcSetTime

This function of type DATE_AND_TIME can be used to set the real-time clock of the local system. It returns 1, if the operation has been terminated successfully, otherwise 0.

Variable	Data Type	Description
ActDateAndTime	DATE_AND_TIME	Time to which the real-time clock of the computer should be set

1.4.2.4.21 The library SysLibSem.lib

Overview

This library can be used to create and use semaphores for the synchronization of tasks. The semaphores serve to avoid any concurrent access on critical data, which are used by several tasks. The target system must support this functionality. The execution is synchronous.

SysSemCreate

This function of type DWORD can be used to create a semaphore. The function returns a handle, which identifies the semaphore and which is required as input value for the other functions of the SysLibSem.lib.

Variable	Data Type	Description
bDummy	BOOL	If bDummy=TRUE, a semaphore will be created

SysSemDelete

This function of type BOOL deletes the semaphore which is identified by the handle retrieved by SysSemCreate. TRUE will be returned in case of success, otherwise FALSE.

Variable	Data Type	Description
dwHandle	DWORD	Handle of the semaphore; was returned by SysSemCreate ↳ *Chapter 1.4.2.4.21.2 “SysSemCreate” on page 606*.

SysSemEnter

This function of type BOOL must be called before a task accesses data which also are used by other tasks. Thus the data will be blocked for other tasks, which also use SysSemEnter until by function SysSemLeave *Chapter 1.4.2.4.21.5 “SysSemLeave” on page 607* the semaphore will be set free again.

The semaphore is identified by the handle which was returned by SysSemCreate. TRUE will be returned in case of success, otherwise FALSE.

Input Variable	Data Type	Description
dwHandle	DWORD	Handle of the semaphore; was returned by SysSemCreate *Chapter 1.4.2.4.21.2 “SysSemCreate” on page 606*.

SysSemLeave

This function of type BOOL must be called after an access on data, which also are used by other tasks. This is necessary to release the semaphore which has been blocked before the data access by SysSemEnter *Chapter 1.4.2.4.21.4 “SysSemEnter” on page 607*.

The semaphore is identified by the handle which was returned by SysSemCreate. TRUE will be returned in case of success, otherwise FALSE.

Input Variable	Data Type	Description
dwHandle	DWORD	Handle of the semaphore; was returned by SysSemCreate *Chapter 1.4.2.4.21.2 “SysSemCreate” on page 606*.

SysSemTry

If a semphore can be entered by SysSemTry(), TRUE will be returned. If not the function returns FALSE with the particular error code.

In contrast to SysSemEnter *Chapter 1.4.2.4.21.4 “SysSemEnter” on page 607* SysSemTry() is non-blocking.

Input Variable	Data Type	Description
dwHandle	DWORD	Handle of the semaphore; was returned by SysSemCreate *Chapter 1.4.2.4.21.2 “SysSemCreate” on page 606*.

1.4.2.4.22 The Library SysLibShm.lib

Overview

This library provides functions for accessing a memory area which is used in common by several processes resp. referencing a physical address (Shared-Memory, shortcut ShM).

The library functions can be used to open and to close the ShM and to read and write from it. The reading, writing and closing functions need the handle which is returned by the opening function.

The execution is synchronous.

SysShmOpen

This function of type DWORD opens a Shared Memory. It returns a handle for the ShM, which can be used as a pointer. The handle is required as input parameter for the other library functions.

Variable	Data Type	Description
stName	STRING	Name of the Shared Memory, can be set as desired dwPhysicalAddress DWORD
dwPhysicalAddress	DWORD	Set here one of the following: - the desired physical address of the ShM; must be valid! - 0, if the location of the ShM area can be arbitrary pdwSize DWORD
pdwSize	DWORD	Address of size of the area for the ShM - If the ShM is already existing, the actual size will be returned. - If the ShM does not yet exist, it will be created according to the given size. If you enter "0" here, the function will fail. Thus the function also can be used to check whether the ShM already has been created.

SysShmClose

This function of type BOOL closes the Shared Memory, which is identified by the handle returned by SysShmOpen. TRUE will be returned after successful operation, otherwise FALSE.

Variable	Data Type	Description
hShm	DWORD	Handle of the Shared Memory; was returned by SysShmOpen ↳ *Chapter 1.4.2.4.22.2 “SysShmOpen” on page 608*.

SysShmRead

This function of type DWORD can be used to read a defined number of bytes from a Shared Memory, starting at a certain offset. It will return the number of actually read bytes.

Variable	Data Type	Description
hShm	DWORD	Handle of the Shared Memory; was returned by SysShmOpen ↳ *Chapter 1.4.2.4.22.2 "SysShmOpen" on page 608*.
dwOffset	DWORD	Offset in the data area, where reading should start
pData	DWORD	Address of the data buffer to be read
dwSize	DWORD	Number of bytes to be read

SysShmWrite

This function of type DWORD can be used to write a defined number of bytes to a Shared Memory. It will return the number of actually written bytes.

Variable	Data Type	Description
hShm	DWORD	Handle of the Shared Memory; was returned by SysShmOpen ↳ *Chapter 1.4.2.4.22.2 "SysShmOpen" on page 608*.
dwOffset	DWORD	Offset in the data area where the writing of the data should start
pData	DWORD	Address of the data buffer to be written
dwSize	DWORD	Number of bytes to be written

1.4.2.4.23 The library SysLibSockets.lib

Overview

This library supports the access on sockets for the communication via TCP/IP and UDP.

UDP
- *For AC500 Firmware < 2.4 FreeUDP must be used.*
- *For AC500 Firmware ≥ 2.4 UDP is available.*

If the target system supports the functionality then the functions listed below are available, each calling the corresponding function of the operating system ↳ *Chapter 1.4.2.4.23.1.1 "Constraints for AC500 V2" on page 610*.

The execution is synchronous.

The behavior of the functions may differ target-specifically.

The opening/closing of sockets may take a long time if many sockets are to be opened/closed simultaneously.

Using the asynchronous functions is recommended (only AC500 V3).
Chapter 1.4.2.4.24 “The library SysLibSocketsAsync.lib” on page 625.

TCP specific:

- *Chapter 1.4.2.4.23.24.1 “SysSockRecv” on page 622*
- *Chapter 1.4.2.4.23.24.2 “SysSockSend” on page 623*

UDP specific:

- *Chapter 1.4.2.4.23.25.1 “SysSockRecvFrom” on page 623*
- *Chapter 1.4.2.4.23.25.2 “SysSockSendTo” on page 624*

Constraints for AC500 V2

The Implementation of the *SysLibSockets* library offers the opportunity to implement user specific communication protocols based on TCP/IP sockets on AC500 V2.

However, there are some restrictions for AC500 V2.

SysLibSocketsAsync
The SysLibSocketsAsync library is not supported.

SysSockIoctl

In order to minimize the influence on timing behavior, AC500 offers the user exclusively non-blocking stream sockets.

Changing such a socket to blocking is not possible for AC500 Firmware < 2.4. The function *SysSockIoctl* will return an error when trying to do so.

SysSockCreate

Opposing standard behavior, sockets are always created non-blocking by the function *SysSockCreate*.

The system offers a limited number of sockets. If the maximum number is reached, *SysSockCreate* and *SysSockAccept* return an error.

SysSockClose

The system tries to close all sockets that are created by the application, nevertheless, the user is advised to close all sockets he created using *SysSockClose*, even in case of of a reset or a download. The function *SysSockClose* might return an error, if the socket was already closed on network side. Please refer to the *SysCallbackRegister* calls in the example application.

SysSockGetLastErrorSync

The error codes returned by *SysSockGetLastErrorSync* refer to the values of the Winsock implementation by Microsoft. Thus, development of applications using *SysLibSockets* can be done on a *SoftPLC* as well.

SysSockShutdown

The function *SysSockShutdown* is not available.

A socket should only be used in the task where it was created. Avoid using sockets in different tasks.

For AC500 Firmware < 2.4, socket options can only be set before the socket is bound.

The returned values of functions *SysSockCreate* and *SysSockAccept* are socket handles. For these a value of 0 is valid, which is why those return values should be compared to SOCKET_INVALID (=-1) instead of comparison to Zero.

Restart needed

If a task that uses sockets is suspended by the system due to an error, this might lead to a leakage of system resources. This can lead to the need of restarting the system.

After a reset or a new download of the user program the PLC can not be set to run again. A power-cycle or a reboot is needed to restart the operating system.

AC500 V2 support overview

SysSockAccept	Supported for non-blocking sockets	Returns negative values, only -1 signals error
SysSockConnect	Supported for non-blocking sockets	
SysSockSelect	Supported for non-blocking sockets	
SysSockRecv (TCP only)	Supported for non-blocking sockets	
SysSockSend (TCP only)	Supported for non-blocking sockets	
SysSockBind	Supported for non-blocking sockets	
SysSockClose	Supported for non-blocking sockets	Might return failure, when socket was already closed
SysSockCreate	Supported for non-blocking sockets	Returns negative values, only -1 signals error, Sockets are non-blocking by default
SysSockListen	Supported for non-blocking sockets	Attention when Listen socket is closed all accepted sockets get closed.
SysSockShutdown	Not supported	
SysSockSendTo (UDP only)	Not supported for AC500 firmware < 2.4 (UDP)	
SysSockRecvFrom (UDP only)	Not supported for AC500 firmware < 2.4 (UDP)	
SysSockGetLastError	Supported	See above for codes
SysSockGetLastErrorSync	Supported	See above for codes
SysSockGetOption	Supported	See above for options
SysSockIoctl	Not allowed for AC500 firmware < 2.4	See above
SysSockSetOption	Supported	See Options above
SysSockGetHostByName	Not supported as requires DNS	
SysSockNtohl	Supported	
SysSockNtohs	Supported	

SysSockHtons	Supported	
SysSockHtonl	Supported	
SysSockGetHostName	Supported	Will return product name
SysSockInetAddr	Supported	
SysSockInetNtoa	Supported	
SysSockSetIPAddress	Not supported on AC500	

SysSockAccept

This function of type DINT calls the function accept of the operating system, which can accept a connection request to the socket. A new descriptor (handle) for the socket is returned. The orignal socket will be reset to the "listening" status ↳ *Chapter 1.4.2.4.23.16 “SysSockListen” on page 618*.

Variable	Data type	Description
diSocket	DINT	A descriptor identifying a socket that has been placed in a listening state with the listen function. The connection will actually be made with the socket that is returned by the SysSockListen function. The requested connection then will be made with that socket, for which the SysSockAccept function returned a handle. (corresponding parameter, e.g. in Win32:s)
pSockAddr	DWORD	Pointer on a variable of type SOCKADDR; will be filled with the address of the caller. (corresponding parameter, e.g. in Win32: addr)
piSocketAddrSize	DWORD	Pointer to a variable of type DINT. This variable has got assigned the length of the structure SockAddr (can be retrieved with the aid of the SIZEOF operator) (corresponding parameter, e.g. in Win32: addrlen)

Structure SOCKADDR:

sin_family : INT; (* Adress-family, defines address format *)

sin_port : UINT; (* Port of the connection requesting unit *)

sin_addr : UDINT; (* IP-address of the requesting unit *)

sin_zero : ARRAY [0..7] OF SINT; (* buffer *)

SysSockBind

This function of type BOOL calls the function bind of the operating system. This function will allocate a local address to the socket which was assigned before just to an address range by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614*.

Usually the "binding" will be done before functions like SysSockListen or SysSockAccept are called for a socket *Chapter 1.4.2.4.23.16 “SysSockListen” on page 618* *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pSockAddr	DWORD	Pointer on a varialbe of type SOCKADDR; *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*
diSockAddrSize	DINT	Lenght of the structure SOCKADDR (can be retrieved with the aid of the SIZEOF operator)

SysSockClose

This function of type BOOL calls the function closesocket of the operating system, in order to close a socket.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)

SysSockConnect

This function of type BOOL calls the function connect of the operating system. In case the socket has not yet been "bound" by the SysSockBind function, now automatically a local address will be assigned to it *Chapter 1.4.2.4.23.3 “SysSockBind” on page 613*. Afterwards the socket will be ready to send and /or receive data.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pSockAddr	DWORD	Pointer on a variable of type SOCKADDR; ↳ *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*
diSockAddrSize	DINT	Length of the structure SOCKADDR; (can be retrieved with the aid of the SIZEOF operator)

Note for operating system VxWorks: There are systems on which function SysSockConnect returns FALSE even in case of success. Reason: The special behavior of connect under VxWorks.

SysSockCreate

This function of type DINT calls the function socket of the operating system. A new socket will be created and assigned to a Service Provider.

The function returns the descriptor of the new socket, which is used as input parameter in other functions of the library, e.g. SysSockBind, SysSockConnect.

Variable	Data type	Description
diAddressFamily	DINT	Address family (corresponding parameter e.g. in Win32: af)
diType	DINT	One of the following two types can be used e.g. for Windows Sockets 1.1: SOCK_STREAM, SOCK_DGRAM (corresponding parameter e.g. in Win32: type)
diProtocol	DINT	Protocol, depending on the chosen address family (corresponding parameter e.g. in Win32: protocol)

SysSockGetHostByName

This function of type DWORD ruft die Funktion hostGetByName (VxWorks)resp.gethostbyname (win32)of the operating system.

In case of successful operation the function will return the host address, otherwise SOCKET_INADDR_NONE (defined in the library as a global constant).

Variable	Data type	Description
stHostName	POINTER TO STRING	Name of the host (corresponding parameter e.g. in Win32: name)

SysSockGetHostName

This function of type BOOL calls the function gethostnameof the operating system and returns the host name.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
stHostName	STRING	Host name (corresponding parameter e.g. in Win32: name)
diNameLength	DINT	Length of the host name (corresponding parameter e.g. in Win32: buflen)

SysSockGetOption

This function of type BOOL calls the function getsockoptof the operating system, in order to get the value of a particular socket option.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
diLevel	DINT	protocol specific level; possible values: SOL_SOCKET, IPPROTO_TCP (corresponding parameter e.g. in Win32: level)
diOption	DINT	Name of the option, for which you want to get the current value; see function SysSockSetOption for a list of the options ↳ *Chapter 1.4.2.4.23.21 “SysSockSetOption” on page 621* (corresponding parameter e.g. in Win32: optname)

Variable	Data type	Description
diOptionValue	DWORD	Pointer to the variable, to which the current value of the option should be written (corresponding parameter e.g. in Win32: optval)
diOptionLength	DWORD	Pointer to the size of the variable, to which the current value of the option should be written (corresponding parameter e.g. in Win32: optlen)

Note for operating system VxWorks: There are systems on which a multiple call of SysSockGetOption only at the first call returns a reasonable option value. This is especially true if an error has occurred just before.

Reason: The special behavior of getsockopt under VxWorks.

SysSockGetLastErrorSync

This function of type INT calls the function getlasterror of the operating system, which returns the error code of the last error occurred at the given socket.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)

SysSockGetLastError

This function block calls the function getlasterror of the operating system, which returns the error code of the last error occurred at the given socket.

Input Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
bEnable	BOOL	Rising edge: the function block starts the action.

Output Variable	Data type	Description
bDone	BOOL	TRUE indicates that the function block has completed processing.
bBusy	BOOL	TRUE indicates that the function block is still processing.
bError	BOOL	TRUE indicates that an error has occured.
wErrorId	WORD	Error number
dwLastError	DWORD	Return value of getlasterror of operating system.

SysSockHtons

This function of type WORD calls the function htons of the operating system, which converts a short value from host byte order to TCP/IP network order.

The function returns the converted value.

Variable	Data type	Description
wHost	WORD	Value to be converted.

SysSockInetAddr

This function of type DWORD calls the function inet_addr of the operating system, which converts a string, containing an internet address, in an address which can be used in the IN_ADDR structure.

The function returns the converted address.

Variable	Data type	Description
stIPAddr	STRING	IP address (dotted notation) (corresponding parameter e.g. in Win32: cp)

SysSockInetNtoa

This function of type BOOL calls the function inet_ntoa (Win32) resp.inet_ntoa_b (VxWorks), which converts an Internet network address in a string in Internet standard format..

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
pInAddr	INADDR	Pointer to structure INADDR, which contains the Internet address, see below (corresponding parameter e.g. in Win32: in)
stIPAddr	STRING	IP address
diIPAddrSize	DINT	Size of the IP address

Structure INADDR:

S_addr : DWORD; (* Internet-Adresse als DWORD *)

SysSockIoctl

This function of type DINT calls the function ioctl of the operating system in order to control the I/O mode of the socket.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↯ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
diCommand	DINT	Command which you want to apply on the socket. (corresponding parameter e.g. in Win32: cmd). Valid commands are: SOCKET_FIONBIO, SOCKET_FIONREAD.
piParameter	DWORD	Pointer to the command parameter (corresponding parameter e.g. in Win32: argp)

SysSockListen

This function of type BOOL calls the function listen of the operating system. This function will cause the socket to listen to connection requests and to queue them until they can be accepted by the SysSockAccept ↯ *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612* function.

In case of successful operation the function will return TRUE.

As soon as the maximum number of connection requests in the queue is exceeded the function will return FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
diMaxConnections	DINT	Maximum number of connection requests, which can be put in the input queue of the socket. (corresponding parameter e.g. in Win32: backlog)

SysSockNtohl

This function of type DWORD calls the function ntohl of the operating system, which converts a u_long value of the TCP/IP network order to the host byte order.

The function will return the value in host byte order.

Variable	Data type	Description
dwNet	DWORD	u_long value to be converted (corresponding parameter e.g. in Win32: netlong)

SysSockNtohs

Attention: Please regard the General Remarks on the library.

This function of type WORD calls the function ntohs of the operating system, which converts a u_short value from the TCP/IP network order to the host byte order.

The function will return the value in host byte order.

Variable	Data type	Description
wNet	WORD	u_short value to be converted (corresponding parameter e.g. in Win32: netshort

SysSockSelect

This function of type DINT calls the function select of the operating system to check whether one or several sockets are ready for certain communication actions. The group of sockets, to which this request should be applied, can be defined via the structure SOCKET_FD_SET.

The function will return the result of the select function.

Variable	Data type	Description
diWidth	DINT	Size of structure SOCKET_FD_SET.
fdRead	DWORD	Optionally a pointer to the structure defining the socket set for which the status of the read actions should be checked. You also can pass 0. Structure SOCKET_FD_SET see below (corresponding parameter e.g. in Win32: readfds)
fdWrite	DWORD	Optionally a pointer to the structure, defining the socket for which the status of the write actions should be checked. You also can pass 0. Structure SOCKET_FD_SET see below (corresponding parameter e.g. in Win32: writefds)
fdExcept	DWORD	Optionally a pointer to the structure, defining the socket for which the error status should be checked. You also can pass 0. Structure SOCKET_FD_SET see below (corresponding parameter e.g. in Win32: exceptfds)
ptvTimeout	DWORD	Maximum time which the SysSockSelect function will wait for an answer; Structure SOCKET_TIMEVAL, see below (corresponding parameter e.g. in Win32:timeout)

Structure SOCKET_FD_SET

fd_count:	UDINT;	(* Number of sockets *)
fd_array:	ARRAY [0..63] OF DINT;	(* Field with socket descriptors *)

Structure SOCKET_TIMEVAL:

tv_sec:	DINT;	(* seconds *)
tv_usec:	DINT;	(* microseconds *)

SysSockSetIPAddress

This function of type BOOL is only implemented for VxWorkstargets. It sets the IP address of the given card.

In case of successful operation the function will return TRUE, otherwise FALSE.

For other operating systems the function always will return FALSE.

Variable	Data type	Description
stCardName	STRING	Name of the network card
stIPAddress	STRING	IP address to be set

SysSockSetOption

This function of type BOOL calls the function getsockoptof the operating system in order to set particular socket options..

For a description of the getsockopt function please see the online help resp. documentation on the operating system.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
diLevel	DINT	Protocol specific level (corresponding parameter e.g. in Win32: level)
diOption	DINT	Name of the option: (corresponding parameter e.g. in Win32: optname) depends on the operating system
diOptionValue	DWORD	Option value; Deactivate boolean option values by setting a "0"; otherwise set the value (corresponding parameter e.g. in Win32: optval)
diOptionLength	DWORD	Length of the buffer for the option value (corresponding parameter e.g. in Win32: optlen)

SysSockShutdown

This function of type BOOL calls the function shutdown of the operating system in order to inhibit further send or receive actions. The function does not close the socket ! This must be done via SysSockClose *Chapter 1.4.2.4.23.4 “SysSockClose” on page 613*.

In case of successful operation the function will return TRUE, otherwise FALSE.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
diHow	DINT	Here you define, which type of communication actions should be inhibited (corresponding parameter e.g. in Win32: how)

SysSockHtonl

This function of type DWORD calls the function htonl of the operating system, which will convert a u_long value from host byte order to TCP/IP network order.

The function returns the converted value.

Variable	Data type	Description
dwHost	DWORD	Value to be converted.

TCP specific functions

SysSockRecv

This TCP/IP specific function of type DINT calls the function read (VxWorks) resp. recv (Win32) of the operating system in order to receive data which have been sent to the socket.

The function will return the number of read bits.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pbyBuffer	DWORD	Address of the buffer from which the data should be read (corresponding parameter e.g. in Win32: buf)

Variable	Data type	Description
diBufferSize	DINT	Size of the buffer from which the data should be read (corresponding parameter e.g. in Win32: len)
diFlags	DINT	Defines in which way the function should be called; depending on the socket options. (corresponding parameter e.g. in Win32: flags)

If the socket has been "gracefully closed" , 0 will be returned, otherwise 1.

SysSockSend

This TCP/IP specific function of type DINT calls the function send of the operating system in order to send the data which are buffered at the socket.

The function will return the number of sent bits. If the socket has been "gracefully closed" , 0 will be returned, otherwise 1.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pbyBuffer	DWORD	Address of the buffer from which the data should be send (corresponding parameter e.g. in Win32: buf)
diBufferSize	DINT	Size of the buffer from which the data should be send (corresponding parameter e.g. in Win32: len)
diFlags	DINT	Defines in which way the function should be called; depending on the socket options. (corresponding parameter e.g. in Win32: flags)

UDP specific functions

SysSockRecvFrom

This UDP specific function of type DINT calls the function recvfrom of the operating system, in order to read the data which have been sent to the socket.

The function will return the number of read bits. If the socket has been "gracefully closed" , 0 will be returned, otherwise 1.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pbyBuffer	DWORD	Address of the buffer from which the data should be received (corresponding parameter e.g. in Win32: buf)
diBufferSize	DINT	Size of the buffer from which the data should be received (corresponding parameter e.g. in Win32: len)
diFlags	DINT	Defines in which way the function should be called; depending on the socket options. (corresponding parameter e.g. in Win32: flags)
pSockAddr	DWORD	Pointer to a variable of type SOCKADDR; ↳ *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*
diSockAddrSize	DINT	Length of structure SockAddr (can be retrieved via the SIZEOF operator) (corresponding parameter e.g. in Win32: iSockAddrSize)

SysSockSendTo

This UDP specific function of type DINT calls the function send of the operating system in order to send the data which are stored at the socket.

The function will return the number of read bits. If the socket has been "gracefully closed" , 0 will be returned, otherwise 1.

Variable	Data type	Description
diSocket	DINT	Descriptor of the socket, returned by SysSockCreate ↳ *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614* (corresponding parameter e.g. in Win32: s)
pbyBuffer	DWORD	Address of the buffer from which the data should be sent (corresponding parameter e.g. in Win32: buf)

Variable	Data type	Description
diBufferSize	DINT	Size of the buffer from which the data should be sent (corresponding parameter e.g. in Win32: en)
diFlags	DINT	Defines in which way the function should be called; depending on the socket options. (corresponding parameter e.g. in Win32: flags)
pSockAddr	DWORD	Pointer to a variable or type SOCKADDR; *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*
diSockAddrSize	DINT	Length of the structure Sock-Addr (can be retrieved via the SIZEOF operator) (corresponding parameter e.g. in Win32: iSockAddrSize)

1.4.2.4.24 The library SysLibSocketsAsync.lib

This library offers the same functionality as 'The library SysLibSockets.lib', however function blocks instead of functions are used and the processing is asynchronous *Chapter 1.4.2.4.23.1 “Overview” on page 609*.

Restriction
This library is not supported for AC500 V2 devices.

- *The opening/closing of sockets may take a long time if many sockets are to be opened/closed simultaneously.*
- *Using the asynchronous functions is recommended.*

Corresponding to the parameters of the functions in SysLibSockets.lib the respective function blocks of SysLibSocketsAsync.lib have specific input parameters with identic impact.

Corresponding to the return values of the functions in SysLibSockets.lib the respective function blocks of SysLibSocketsAsync.lib have specific output parameters with identic impact.

Additionally the following input and output parameters are available in all function blocks:

Input:	bEnable	BOOL	Rising edge: the function block starts the action.
Outputs:	bDone	BOOL	TRUE indicates that the function block has completed processing.
	bBusy	BOOL	TRUE indicates that the function block is still processing.

	bError	BOOL	TRUE indicates that an error has occurred.
	wErrorId	WORD	Error number

Each function block will start the respective action as soon as a rising edge is detected at bEnable. Then it must be called cyclically until bDone=TRUE. Thereafter the outputs bError and wErrorId as well as the specific output parameters of the particular function block are in effect.

- *Chapter 1.4.2.4.23.2 “SysSockAccept” on page 612*
- *Chapter 1.4.2.4.23.3 “SysSockBind” on page 613*
- *Chapter 1.4.2.4.23.4 “SysSockClose” on page 613*
- *Chapter 1.4.2.4.23.5 “SysSockConnect” on page 613*
- *Chapter 1.4.2.4.23.6 “SysSockCreate” on page 614*
- *Chapter 1.4.2.4.23.7 “SysSockGetHostByName” on page 614*
- *Chapter 1.4.2.4.23.8 “SysSockGetHostName” on page 615*
- *Chapter 1.4.2.4.23.11 “SysSockGetLastError” on page 616*
- *Chapter 1.4.2.4.23.9 “SysSockGetOption” on page 615*
- *Chapter 1.4.2.4.23.12 “SysSockHtons” on page 617*
- *Chapter 1.4.2.4.23.12 “SysSockHtons” on page 617*
- *Chapter 1.4.2.4.23.13 “SysSockInetAddr” on page 617*
- *Chapter 1.4.2.4.23.14 “SysSockInetNtoa” on page 617*
- *Chapter 1.4.2.4.23.15 “SysSockIoctl” on page 618*
- *Chapter 1.4.2.4.23.16 “SysSockListen” on page 618*
- *Chapter 1.4.2.4.23.17 “SysSockNtohl” on page 619*
- *Chapter 1.4.2.4.23.18 “SysSockNtohs” on page 619*
- *Chapter 1.4.2.4.23.19 “SysSockSelect” on page 619*
- *Chapter 1.4.2.4.23.20 “SysSockSetIPAddress” on page 620*
- *Chapter 1.4.2.4.23.21 “SysSockSetOption” on page 621*
- *Chapter 1.4.2.4.23.22 “SysSockShutdown” on page 621*

TCP specific:

- *Chapter 1.4.2.4.23.24.1 “SysSockRecv” on page 622*
- *Chapter 1.4.2.4.23.24.2 “SysSockSend” on page 623*

UDP specific:

- *Chapter 1.4.2.4.23.25.1 “SysSockRecvFrom” on page 623*
- *Chapter 1.4.2.4.23.25.2 “SysSockSendTo” on page 624*

1.4.2.4.25 The library SysLibStr.lib

Overview

This library provides functions for string operations. If the target system is supporting the functionality, the library functions can be used to compare or copy strings or to retrieve the length of a string. The execution is synchronous.

SysStrCmp

This function of type DINT compares lexicographically two strings and returns one of the following values:

Return value < 0 String1 smaller than String2

Return value = 0 String1 = String2

Return value > 0 String1 bigger than String2

Variable	Data Type	Description
dwString1	STRING	First string (String1)
dwString2	STRING	Second string (String2)

SysStrCmpI

This function of type DINT checks whether two strings are identical and returns one of the following values:

Return value < 0 String1 smaller than String2

Return value = 0 String1 = String2

Return value > 0 String1 bigger than String2

Variable	Data Type	Description
dwString1	STRING	First string (String1)
dwString2	STRING	Second string (String2)

SysStrCmpN

This function of type DINT compares the size of two strings, whereby a defined number of characters counted from the beginning of the string will be considered. One of the following values will be returned:

Return value < 0 String1 smaller than String2

Return value = 0 String1 = String2

Return value > 0 String1 bigger than String2

Variable	Data Type	Description
sString1	STRING	First string (String1)
sString2	STRING	Second string (String2)
diChars	DINT	Number of locations for which (start counting from the beginning of the string) the size of the strings should be compared

SysStrCmpNI

This function of type DINT checks whether a defined number of characters of two strings (starting at the beginning of the string) are identical. One of the following return values will show the result:

Return value < 0 String1 smaller than String2

Return value = 0 String1 = String2

Return value > 0 String1 bigger than String2

Variable	Data Type	Description
sString1	STRING	First string (String1)
sString2	STRING	Second string (String2)
diChars	DINT	Number of characters starting at the beginning of the string, which will be checked for identical values in both strings

SysStrCpy

This function of type DWORD copies one string (Str1) to another (Str2). It will return a pointer to the target string Str2.

Variable	Data Type	Description
sString1	STRING	String, to which you want to copy (destination)
sString2	STRING	String, which should be copied (source)

SysStrLen

This function of type DINT acquires the length of a string. It will return the number of characters, excluding the "terminal NULL".

Variable	Data Type	Description
sString1	STRING	String, for which the length should be retrieved

1.4.2.4.26 The library SysLibTasks.lib

Overview

If the target system supports the functionality, then the library functions can be used to manage tasks. That means to generate, to delete, to modify priority level, to stop and restart tasks. The execution is synchronous.

These functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library SysLibSema in this case.

If you need functions to get information on the configuration of IEC-Tasks, use the library SysLibIECTasks.lib. ↳ *Chapter 1.4.2.4.11.1 “Overview” on page 586*.

Functions to be used within a task:

- ↳ *Chapter 1.4.2.4.26.9 “SysTaskSleep” on page 632*
- ↳ *Chapter 1.4.2.4.26.10 “SysTaskEnd” on page 633*
- ↳ *Chapter 1.4.2.4.26.11 “SysTaskGetCurrent” on page 633*

SysTaskCreate

This function of type UDINT creates a new task. It will return an unique Id number for the task, which is required as an input parameter for the other functions of SysLibTask.lib.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library SysLibSema in this case.

Input Variable	Data type	Description
stName	STRING	Name of the task
byPriority	BYTE	Priority of the Task; possible values : 0-255, Reserved: 0..31 for System IEC-Tasks: 32..63 Communication -Tasks: 64 and higher
udiInterval	UDINT	Task interval in milliseconds
pfFunction	DWORD	Function pointer, which must be acquired with the aid of the function SysIECGetFct-Pointer()
pArgument	DWORD	Pointer to parameters for the new task

SysTaskDestroy

This function of type BOOL can be used to delete a task. It will return TRUE, if the operation has succeeded, otherwise FALSE.

SysTask-functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case ↳ *Chapter 1.4.2.4.21.1 “Overview” on page 606.*

Input Variable	Data type	Description
udiTaskId	UDINT	Id of the task, which should be deleted; Id is returned by Sys-TaskCreate ↳ *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*

SysTaskGetInfo

This function of type BOOL returns information on a task, which is identified by the task Id.

SysTask-functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case *Chapter 1.4.2.4.21.1 “Overview” on page 606.*

Input Variable	Data type	Description
udiTaskId	UDINT	Id of the task, on which you want to get information; this Id was returned by SysTask-Create at creating the task *Chapter 1.4.2.4.26.2 “Sys-TaskCreate” on page 629*
pSysTaskInfo	POINTER TO SYSTASKINFO	Pointer on the structure Sys-TaskInfo, see below, which contains information on the task

Structure SysTaskInfo:

TYPE SYSTASKINFO :

STRUCT

dwHandle:	DWORD;	(* Operating system handle of the task *)
dwId:	DWORD;	(* Index of the task *)
dwSem:	DWORD;	(*Reserved, only use in the runtime system *)
wIECTaskNr:	WORD;	(* IEC task index in case it is an IEC task *)
stName:	STRING;	(* Name of the task *)

END_STRUCT

END_TYPE

SysTaskGetPriority

This function of type BYTE returns the priority of the task identified by the task Id.

The priority can be a value between 0 (=highest priority) and 255 (=lowest priority).

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case *Chapter 1.4.2.4.21.1 “Overview” on page 606.*

Input Variable	Data type	Description
udiTaskId	UDINT	Id of the task, of which you want to know the priority level; this Id was returned by SysTaskCreate during creation of the task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*

SysTaskSetPriority

This function of type BOOL can be used to define the priority level for a task which is identified by the task Id. TRUE will be returned in case of a successful operation, otherwise FALSE.

The priority level can be a value between 0 (=highest priority) and 255 (=lowest priority).

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case Chapter 1.4.2.4.21.1 “Overview” on page 606.

Input Variable	Data type	Description
UdiTaskId	UDINT	Id of the task, for which the priority level should be set; this Id was returned by SysTaskCreate during creation of the task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*
byPriority	BYTE	Priority ; possible values : 0 "“ 255 - Reserved for system: 0..31 - IEC-Tasks: 32..63 - Communication tasks: 64 and higher

SysTaskSuspend

This function of type BOOL can be used to stop a task during operation. The task will be identified by the task Id. (By calling the function SysTaskResume the processing can be continued later *Chapter 1.4.2.4.26.8 “SysTaskResume” on page 632*.)

TRUE will be returned in case of a successful stop of the task, otherwise FALSE.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case Chapter 1.4.2.4.21.1 “Overview” on page 606.

Input Variable	Data type	Description
udiTaskId	UDINT	Id of the task, which should be stopped; this Id was returned by SysTaskCreate during creation of the task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*

SysTaskResume

This function of type BOOL can be used to continue the processing of a task, which was stopped before by the function SysTaskSuspend *Chapter 1.4.2.4.26.7 “SysTaskSuspend” on page 631*.

TRUE will be returned in case of a successful stop of the task, otherwise FALSE.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case Chapter 1.4.2.4.21.1 “Overview” on page 606.

Input Variable	Data type	Description
udiTaskId	UDINT	Id of the task, which should continue processing; this Id was returned by SysTaskCreate during creation of the task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*

SysTaskSleep

This function of type BOOL can be used to interrupt the processing in a running task and to make it continue after a defined period of time.

TRUE will be returned, if the sleep function has been executed successfully, otherwise FALSE.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case Chapter 1.4.2.4.21.1 “Overview” on page 606.

Input Variable	Data type	Description
udiMilliseconds	UDINT	Time in milliseconds after which the stopped (sleeping) task should continue to be processed

SysTaskEnd

This function of type BOOL should be called by a task as soon its processing has been terminated. Typically this should be done immediately before the task is left.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case *Chapter 1.4.2.4.21.1 “Overview” on page 606.*

Input Variable	Data type	Description
udiExitCode	UDINT	Should be 0
udiTaskId	UDINT	Id of the task, which should be terminated; has been returned by SysTaskCreate during creation of the task *Chapter 1.4.2.4.26.2 “SysTaskCreate” on page 629*

SysTaskGetCurrent

This function of type UDINT can be called by the currently processing task in order to get returned the own task Id.

SysTask functions are not reentrant. This should be no problem in normal cases, but if in the application program a situation might occur, where several IEC-tasks create and manage additional tasks, the calls to these functions have to be synchronized. You can use the library 'The library SysLibSem.lib' in this case *Chapter 1.4.2.4.21.1 “Overview” on page 606.*

Input Variable	Data type	Description
bDummy	BOOL	TRUE starts the function

1.4.2.4.27 The library SysLibSymbols.lib

Overview

If supported by the target system (CSP32F as from Version 2.4.5.0), the function SysLibGetSymbolAddress1 provided by this library can be used to read the physical address of a symbol of an IEC project. The processing is done synchronously *Chapter 1.4.2.4.27.2 “Function SysLibGetSymbolAddress” on page 633*.

Function SysLibGetSymbolAddress

This function of type DWORD of the library 'The Library SysLibSymbols' returns the physical address for the given symbol. A variable must be specified by its full name, e.g. "PLC_PRG.a’ or "*.global_var’ *Chapter 1.4.2.4.27.1 “Overview” on page 633*.

As a precondition the symbol file must be available on the PLC and the symbol must be entered in the symbol file. The appropriate settings are to be done in the project options and in the target settings dialogs.

There is still the possibility to get the address of a variable by the ADR operator (e.g. ADR(PLC_PRG.a)). But in this case the symbol must be known already at compile time. The SysLibGetSymbolAddress function however allows to adapt the symbol name during run time ↳ *Chapter 1.4.2.4.27.2 “Function SysLibGetSymbolAddress” on page 633*.

The return value is the address of the variable or 0 in case the symbol could not be found.

Input Variable	Data Type	Description
pszSymbol	STRING	Name of the variable

1.4.2.4.28 The library SysLibTime.lib

Overview

This library provides function blocks for reading the real-time clock of the local system. The execution is synchronous.

- ↳ *Chapter 1.4.2.4.28.2 “CurTime” on page 634*
- ↳ *Chapter 1.4.2.4.28.3 “CurTimeEx” on page 634*

Used structures:

- ↳ *Chapter 1.4.2.4.28.4 “Structure SystemTimeDate” on page 635*
- ↳ *Chapter 1.4.2.4.28.5 “Structure SysTime64” on page 635*

CurTime

This function block provides the real time of the local system in microseconds; using the structure SysTime64.

VARINOUT Variable	Data type	Description
SystemTime	SysTime64	Value of the real-time clock of the local system in microseconds, see Structure SysTime64 ↳ *Chapter 1.4.2.4.28.5 “Structure SysTime64” on page 635*

CurTimeEx

This function block provides extended information on the real-time clock data on the local system.

VARINOUT Variable	Data type	Description
SystemTime	SysTime64	Value of the real-time clock in microseconds, see Structure SysTime64 ↳ *Chapter 1.4.2.4.28.5 “Structure SysTime64” on page 635*
TimeDate	SystemTimeDate	Detailed information on the value of the real-time clock, see Structure SystemTimeDate ↳ *Chapter 1.4.2.4.28.4 “Structure SystemTimeDate” on page 635*

Structure SystemTimeDate

This structure contains the following information on the real time given by the local system clock. It is used by the function block CurTimeEx ↳ *Chapter 1.4.2.4.28.3 “CurTimeEx” on page 634*.

Component	Data type	Description
dwLowMSecs	DWORD	The value of the real-time clock is returned in microseconds, using a Low DWORD plus a High DWORD, see Structure SysTime64 ↳ *Chapter 1.4.2.4.28.5 “Structure SysTime64” on page 635*
dwHighMsec	DWORD	
Year	UINT	Year, e.g. "2002"
Month	UINT	Month, e.g. "12"
Day	UINT	Day of month, e.g. "3"
Hour	UINT	Hour of the current day, e.g. "13"
Minute	UINT	Minutes of the current hour, e.g. "43"
Second	UINT	Seconds of the current minute, e.g. "15"
Milliseconds	UINT	Milliseconds of the current second, e.g. "649"
DayOfWeek	UINT	Day of the week, e.g. "2" (Sunday=0, Monday = 1...)

Structure SysTime64

This structure contains the real time of the local system in microseconds. A Low- plus a High-DWORD are used for that purpose, thus 64 bit are available. The structure is used by the function blocks CurTime and CurTimeEx ↳ *Chapter 1.4.2.4.28.2 “CurTime” on page 634* ↳ *Chapter 1.4.2.4.28.3 “CurTimeEx” on page 634*.

Component	Data type	Description
ulLow	DWORD	Low DWORD of the real-time value (microseconds)
ulHigh	DWORD	High DWORD of the real-time value (microseconds)

1.4.3 Visualization

1.4.3.1 Overview

A visualization is a graphical representation of the project variables which allows inputs to the PLC program in online mode via mouse and keypad. The integrated visualization provides graphic elements which can be arranged as desired and can be connected with project variables. Thereupon in online mode the look of the graphical elements will change depending on the variables values.

Simple example: In order to represent a fill level, which is calculated by the PLC program, draw a bar and connect it to the corresponding project variable, so that the length and color of the bar will show the current fill level value. Add a text field which will display the current value in a text string and a button for starting and stopping the program.

The properties of a single visualization element as well as of the whole visualization object will be defined in appropriate configuration dialogs and in the Object Properties dialog. There it is possible to set basic parameters by activating options as well as to define a dynamic parameterizing by entering project variables.

Additional special possibilities for configuring are given by the programmability of element properties via structure variables.

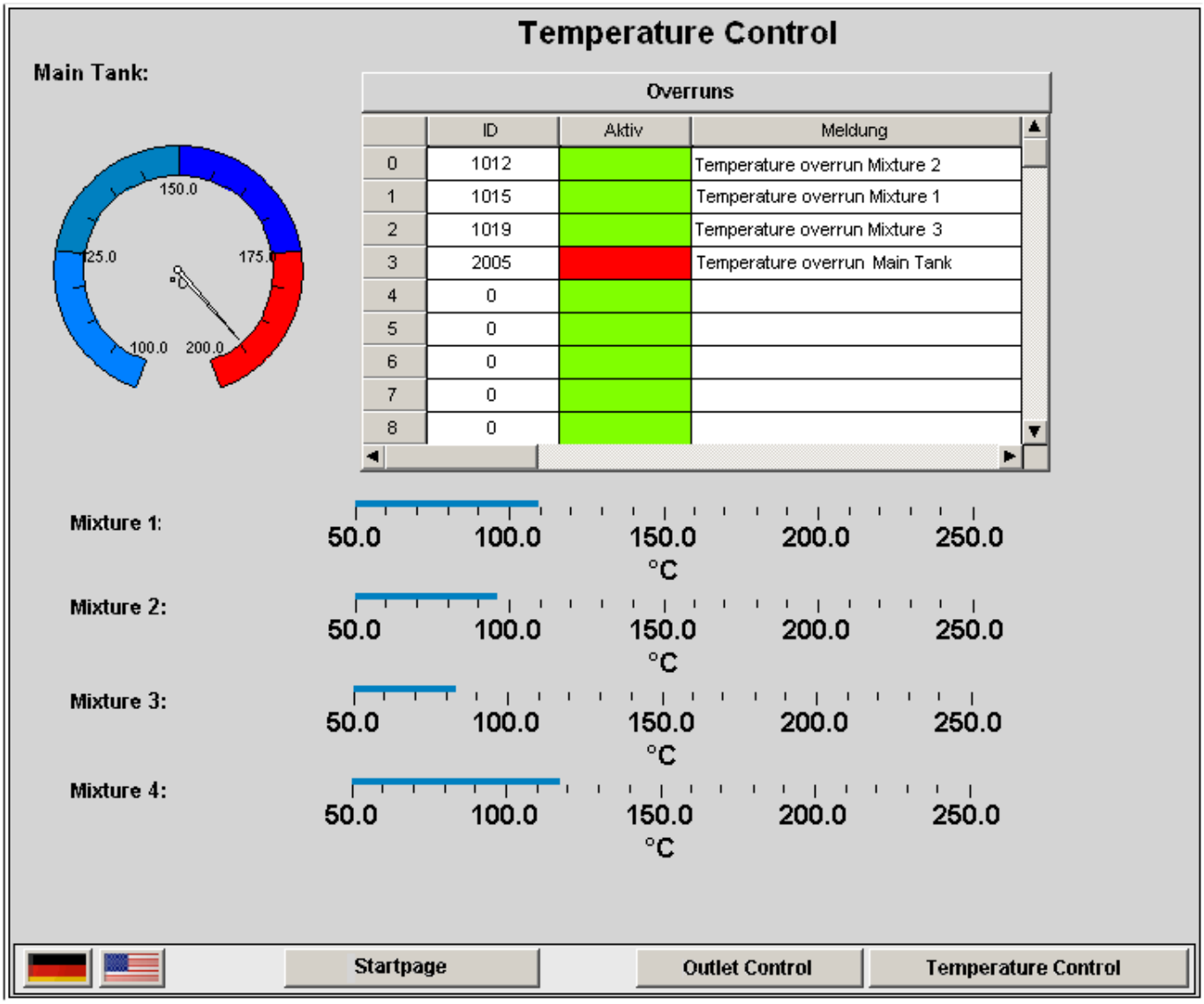
Using placeholders in the configuration dialogs may save a lot of effort in case you want to use the same visualization object several times with different configurations.

The visualization which is created in the programming system will in many cases be used as the only user interface available for controlling and watching the associated PLC program in online mode. For this purpose it must be possible to give inputs to the program solely by activating visualization elements. To reach this you can use special input possibilities during the configuration of the visualization and you have the option to define special hotkeys for each particular visualization.

A visualization can later be used in different ways:

- It can be made available on CODESYS HMI, a special runtime system for operating the visualization in full screen mode on a PLC computer.
- It can be made available as a web visualization, which allows to call and operate it via the Internet (useful for remote maintenance purposes)

Example of a visualization:



1.4.3.2 Create a new visualization

A visualization object is managed in the 'Visualization' register of the Object Organizer. It contains an arrangement of visualization elements and can get certain object properties. One or several visualization objects can be created in a Automation Builder project and might be linked with each other.

In order to create a visualization object in the Object Organizer, you must select the register card for Visualization in the Object Organizer.

Using the 'Project' 'Object Add' command, you can create a new visualization object. Open the 'New visualization' dialog, in which you can enter the name of the new visualization (see the remarks below). Once a valid entry is made, that is not a name that is already in use and no special characters used, you can close the dialog with OK. A window opens, in which you can edit the new visualization.

When the visualization object is marked in the Object Organizer, via command 'Project' 'Object' 'Properties' the properties dialog can be opened, where you can make settings concerning the usage of the object in a web visualization as well as concerning a Master layout ↳ *Chapter 1.4.1.2.4.12 “'Project' 'Object properties'” on page 261*.

When defining the name of the visualization object, please regard the following:

- A visualization named "PLC_VISU" per default automatically will be used as start visualization in a web visualization or in CODESYS HMI, if there not explicitly another visualization is configured for this.
- A visualization may not get the same name as another object within the project because this would result in problems when changing between visualizations.

If you want to use the implicit variable CurrentVisu (type STRING) for addressing the currently opened visualization object, with compiler versions < V2.3.7.0 and if the library SysLibStr.lib is not included in the project, you must use capital letters for the names of the visualization objects (e.g. PLC_VISU).

For information on implicit variables see 'System variables' ↳ *Chapter 1.4.3.11 “System variables” on page 717.*

1.4.3.3 Inserting visualization elements

A visualization element is a graphical element, which is used to fill a visualization object. The available elements are offered in the menu bar. Each element gets a separate configuration.

You can insert various geometric forms, as well as bitmaps, metafiles, buttons, various special elements and existing visualizations into your visualization. Regard the possibility of defining a special directory for visualization files in the Options for directories ↳ *Chapter 1.4.1.2.2.7 “Options for directories” on page 207*.

Go to the 'Insert' menu item and select freely from the following commands: 'Rectangle', 'Rounded Rectangle', 'Ellipse', 'Polygon', 'Polyline', 'Curve', 'Pie', 'Bitmap', 'Visualization', 'Button', 'Table', 'ActiveX-Element', 'Scrollbar', 'Meter', 'Bar Display', 'Histogram', 'Alarm table', 'Trend', 'WMF file'.

Alternatively you can use the tool bar.

If you then draw the mouse pointer to the editor window, the corresponding element symbol will be displayed at the pointer. Click on the desired starting point of your element and move the pointer with pressed left mouse key until the element has the desired dimensions.

If you want to create a polygon or a line, first click with the mouse on the position of the first corner of the polygon resp. on the starting point of the line, and then click on the further desired corner points. By doubleclicking on the last corner point you will close the polygon and it will be completely drawn respectively the line will be completed. If you want to create a curve (Bezier curves) determine the initial and two other points with mouse clicks to define the circumscribing rectangle. An arc is drawn after the third mouse click. You can then change the position of the end point of the arc by moving the mouse and can then end the process with a double click or add another arc with additional mouse clicks.

Furthermore pay attention to the status bar and the change from select and insert modes.

1.4.3.3.1 'Insert' 'Rectangle'

Symbol:

With the command you can insert a rectangle as an element into your present visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.2 'Insert' 'Rounded Rectangle'

Symbol:

With the command you can insert a rectangle with rounded corners as an element in your present visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.3 'Insert' 'Ellipse'

Symbol:

With the command you can insert a circle or an ellipse as an element in your present visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.4 'Insert' 'Polygon'

Symbol:

With the command you can insert a polygon as an element in your present visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.5 'Insert' 'Polyline'

Symbol:

With the command you can insert a line as an element into your current visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.6 'Insert' 'Curve'

Symbol:

With the command you can insert a Bezier curve as an element into your current visualization ↳ *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

1.4.3.3.7 'Insert' 'Pie'

Symbol:

Use this command to insert a Pie Segment as an element into your current visualization.

While pressing the left mouse button, bring up an area in the desired size. An oval element including a line marking the radius at the 0° position will be displayed. As long as keeping the mouse button pressed you can immediately change size and position of the element by moving the mouse. A little black square is attended to the the element, indicating the corner of a virtual rectangle surronding the element.

In order to define the start and end angles of a Pie, select the end point of the radius line on the circular arc by a mouse-click. As soon as you, keeping the mouse button pressed, move the cursor, two small rectangles will be displayed, indicating the two angle positions. As from now those can be selected and moved seperately. If you want the angle values get defined dynamically by variables, open the configuration dialog category 'Angle' and enter the desired variable names *Chapter 1.4.3.5.5 “Angle” on page 650*.

You can resize or reshape the element later by either clicking on the centre point, the cursor getting displayed as diagonally crossed arrows, and moving the mouse while keeping the mouse button pressed (or using the arrow keys). Alternatively you can select and move the corner indicating little square outside of the element. In order to move the element to another position, click inside the element to get the cursor beeing displayed as vertically crossed arrows and then move the cursor.

1.4.3.3.8 'Insert' 'Bitmap'

Symbol:

With the command you can insert a bitmap as an element in your present visualization *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

While pressing the left mouse button, bring up an area in the desired size. The standard dialog box for browsing for a file will be opened. You can use this dialog to select and enter a bitmap file from the local file system. In the configuration dialog of the inserted bitmap you then can define, whether a link to the bitmap file should be stored or the bitmap should be inserted as an element.

Alternatively you can specify a project variable to define which bitmap should be used. This allows a dynamic change of bitmaps in online mode. The variable must be entered in the configuration dialog of an already inserted bitmap element and it must contain the name of a bitmap file managed in the project-global *Chapter 1.4.3.7.1 “'Extras' 'Bitmap list'” on page 706*.

For a description on the configuration of a bitmap element please see 'Bitmap' *Chapter 1.4.3.5.19 “Bitmap” on page 669*.

1.4.3.3.9 'Insert' 'Visualization'

Symbol:

With the command you can insert an existing visualization as an element in your present visualization *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

While pressing the left mouse button, bring up an area in the desired size. A selection list of existing visualizations opens. After you have selected the desired visualization, it will be inserted in the defined area. An inserted visualization will also be named "instance".

1.4.3.3.10 'Insert' 'Button'

Symbol:

This command is used to insert a button into your current visualization *Chapter 1.4.3.3 “Inserting visualization elements” on page 638*.

Drag the element to the desired size with the left mouse button held down.

If a toggle variable is configured for the button it displays the state of this variable by visually displaying whether it is pressed or not pressed. Conversely, the variable is toggled by "pressing" the button.

Like for a "Bitmap" element also for the filling of a button element an image file can be specified (static or dynamic use). See the description of the configuration dialog for 'Insert Bitmap' ↳ *Chapter 1.4.3.3.8 “'Insert' 'Bitmap'” on page 640*.

1.4.3.3.11 'Insert' 'WMF file'

Symbol:

This command is used to insert a Windows Metafile. The standard dialog for opening a file will appear, where you can select a file (extension *.wmf). After having closed the dialog with OK the file will be inserted as an element in the visualization. Please regard, that no link to a file will be saved, like it is done when you insert a bitmap, but the elements of the metafile will be inserted as a group ↳ *Chapter 1.4.3.3.8 “'Insert' 'Bitmap'” on page 640 ↳ Chapter 1.4.3.4.8 “Grouping elements” on page 644*.

1.4.3.3.12 'Insert' 'Table'

Symbol:

Use this command to insert a Table element as an element into your current visualization. It is used to display the current values of the elements of an array.

While pressing the left mouse button, bring up an area in the desired size. Before the element gets displayed the configuration dialog 'Configure Table' will be opened ↳ *Chapter 1.4.3.5.20 “Table” on page 671*. Here you will find additionally to the standard categories Tooltip and Security the categories 'Table', 'Columns', 'Rows' and 'Selection' where you can define contents and appearance of the table.

1.4.3.3.13 'Insert' 'ActiveX-Element'

Symbol:

Use this command to insert an ActiveX Control into your current visualization. It can be used later on Windows32 systems in CODESYS HMI. While pressing the left mouse button, bring up an area in the desired size. It will be inserted as a rectangle with the inscription "Control:".

To select a certain ActiveX Control and to configure the method calls and the display open the dialog 'Configure ActiveX-Control' by a double-click on the element resp. via command 'Extras' 'Configure' ↳ *Chapter 1.4.3.5.4 “'Extras' 'Configure'” on page 649*.

1.4.3.3.14 'Insert' 'Scrollbar'

Use this command to insert a Scrollbar element in the current visualization. In the configuration of that element you can assign a variable to the element, the value of which will be changed when the user moves the scrollbar slider in online mode ↳ *Chapter 1.4.3.5.22 “Scrollbar” on page 679*. Vice versa the slider position will reflect the current value of the variable if this value is given by any other input. In the target visualization the element will work only if it is supported by the target system.

Insert the element by drawing a rectangle in desired size. The shape of the rectangle determines whether the scrollbar will be entered vertically or horizontally .

1.4.3.3.15 'Insert' 'Meter'

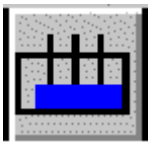
Symbol:

Use this command to insert a Meter as an element into your current visualization. It provides a scale which is defined as a sector of a circular arc, and a pointer element.

While pressing the left mouse button, bring up an area in the desired size. Before the element gets displayed the configuration dialog 'Meter' will be opened ↳ *Chapter 1.4.3.5.23 “Meter” on page 680*. Here you can define various parameters concerning the display of the element and a preview is available to check the configuration before really inserting the element by confirming the dialog.

1.4.3.3.16 'Insert' 'Bar Display'

Symbol:



Use this command to insert a Bar Display element into your current visualization. It is used to visualize the value of the assigned variable by a bar indicating the value by its length along a horizontal scale.

While pressing the left mouse button, bring up an area in the desired size. Before the element gets displayed the configuration dialog 'Configure bar display' will be opened ↳ *Chapter 1.4.3.5.24 “Bar display” on page 682*. Here you can define various parameters concerning the display of the element and a preview is available to check the configuration before really inserting the element by confirming the dialog.

1.4.3.3.17 'Insert' 'Histogram'

Symbol:

Use this command to insert a Histogram element into your current visualization. It is used to visualize the elements of an array by bars which are placed side by side each indicating the value of the element by its length.

While pressing the left mouse button, bring up an area in the desired size. Before the element gets displayed the configuration dialog 'Configure Histrogram' will be opened ↳ *Chapter 1.4.3.5.25 “Histogram” on page 683*. Here you can define various parameters concerning the display of the element and a preview is available to check the configuration before really inserting the element by confirming the dialog.

1.4.3.3.18 'Insert' 'Alarm table'

Symbol:

Use this command to insert an alarm table into your current visualization object.

While pressing the left mouse button, bring up an area in the desired size. Before the element gets displayed the configuration dialog 'Configure Alarm table' will open *Chapter 1.4.3.5.26 “Alarm table” on page 684*. Here you will find additionally to the standard categories Tooltip and Security the categories 'Alarmtable', 'Settings for sorting', 'Columns' and Settings for alarmtable' where you can define contents and appearance of the table.

An alarm table can be used to visualize the alarms, which have been defined in the Alarm configuration of the project.

1.4.3.3.19 'Insert' 'Trend'

Symbol:

Use this command to insert a trend element into your current visualization object.. While pressing the left mouse button, bring up an area in the desired size. The configuration (axes, variables, history) is done in the configuration dialog 'Trend' *Chapter 1.4.3.5.27 “Trend” on page 688*.

The trend element, also named oscilloscope element, is used to display variable values within a certain time period. It stores the data in a file on the client and displays them as a graph. As soon as a value changes, a new entry will be made in the file, showing date/time and the new value.

The trend element is drawn transparently. So you can assign any desired background (bitmap, color).

1.4.3.4 Positioning visualization elements

1.4.3.4.1 Selecting visualization elements

The selection mode is activated by default. In order to select an element, click with the mouse on the element. You can also select the first element of the elements list by pressing the <Tab> key and jump to the next by each further keystroke. If you press *[Tab]* while pressing *[Shift]*, you jump backwards in the order of the elements list.

In order to select elements, which are placed one upon the other, first select the top level element by a mouse-click. Then do further mouse-clicks while *[Ctrl]* + *[Shift]* are pressed, to reach the elements in the underlying levels .

In order to mark multiple elements, press and hold *[Shift]* and click the corresponding elements, one after another. Or while holding down the left mouse button, pull a window over the elements to be selected.

In order to select all the elements, use the 'Extras' 'Select All' *Chapter 1.4.3.4.4 “'Extras' 'Select All'” on page 644* command.

If you are in the element list (called by 'Extras' 'Element list'), you can select the concerned element in the visualization by selecting a line.

1.4.3.4.2 Changing the selection and insert mode

After the insertion of a visualization element, there is an automatic change back into the selection mode. If you want to insert an additional element the same way, you can once again select the corresponding command in the menu.

You can also quickly change between the selection mode and the insert mode by pressing *[Ctrl]* and the right mouse button simultaneously.

In the insert mode, the corresponding symbol will also appear at the mouse pointer, and the name will also be indicated in black in the status bar.

1.4.3.4.3 'Extras' 'Select'

This command is used to switch the selection mode on or off. This can also be achieved by pressing the right mouse-key while holding down the *[Ctrl]*key at the same time.

1.4.3.4.4 'Extras' 'Select All'

This command allows you to select all visualization elements within the current visualization object.

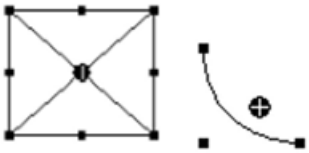
1.4.3.4.5 Copying visual elements

One or more selected elements can be inserted with the 'Edit' 'Copy'command, the *[Ctrl]* + *[C]* combination, or the corresponding copy symbol, and with 'Edit' 'Paste'.

A further possibility is to select the elements and to again click in one of these elements with *[Ctrl]* held down. If you now hold the left mouse button down, you can separate the elements thus copied from the original.

1.4.3.4.6 Modifying visualization elements

You can select an element which has already been inserted by a mouse click on the element or by pressing *[tab]*. A small black square will appear at each corner of each of the elements, (with ellipses at the corners of the surrounding rectangle). Except in the case of polygons, lines or curves further squares appear in the middle of the element edges between the corner points.



With a selected element, the turning point (balance point) is also displayed at the same time. You can then rotate the element around this point with a set motion/angle. The turning point is displayed as a small black circle with a white cross. You can drag the turning point with a pressed left mouse button.

You can change the size of the element by clicking on one of the black squares and, while keeping the left mouse button pressed, controlling the new outline.

With the selection of a polygon, you can drag each individual corner using the same technique. While doing this, if you press *[Ctrl]* then an additional corner point will be inserted at the corner point, an additional corner point will be inserted, which can be dragged by moving the mouse. By pressing *[Shift]* + *[Ctrl]*, you can remove a corner point.

1.4.3.4.7 Dragging visualization elements

One or more selected elements can be dragged by pressing the left mouse button or the arrow key.

1.4.3.4.8 Grouping elements

Elements can be grouped by selecting all desired elements and performing the command 'Extras' 'Group'. The group will behave like a single element:

- the grouped elements get a collective frame; when dragging the frame, depending on the current configuration all elements will be stretched, compressed or keep their original size; only the group can be moved to another position.
- the grouped elements get collective properties: inputs only can effect the group and not a single element. Thus the elements also get one collective configuration (category 'group'). The property 'Change color' can not be configured for a group!

To redefine a single element of a group, the grouping must be redone by the command 'Extras' 'Ungroup'. The configuration of the group will be lost in this case.

As soon as you save the project as program version 2.1 or lower, a group of visualization elements will be resolved automatically; that means that the elements of the group will be shown as single elements in the visualization.

1.4.3.4.9 'Extras' 'Send to Front'

Use this command to bring selected visualization elements to the front.

1.4.3.4.10 'Extras' 'Send to Back'

Use this command to send selected visualization elements to the back.

1.4.3.4.11 'Extras' 'Align'

Use this command to align selected visualization elements.

The following alignment options are available:

- Left: the left edge of each of the elements will be aligned to the element that is furthest to the left.
- the same is true for Right / Top / Bottom.
- Horizontal Center: each of the elements will be aligned to the average horizontal center of all elements.
- Vertical Center: each of the elements will be aligned to the average vertical center of all elements.

1.4.3.4.12 'Extras' 'Element list'

This command opens a dialog box containing a list of all visualization elements including their number, type and position. The element number will be displayed in the element in the editor view, if the corresponding option is activated in the visualization settings ↳ *Chapter 1.4.3.6.1 "'Extras' 'Settings'" on page 700*. The position is given according to the x and y position of the upper left (x1, y1) and the lower right (x2, y2) corner of the element.

When one or more items have been selected, the corresponding elements in the visualization are marked for visual control and if necessary the display will scroll to that section of the visualization that contains the elements.

Use the *[To front]* button to bring selected visualization elements to the front. Use the *[To behind]* button to move them to the back.

Below the elements list there you find depending on which element is currently selected - one of the following combinations of edit fields where you can modify size and position of the element:

- If a rectangle, rounded rectangle, ellipse, bitmap, visualization, button or a meta file is currently selected, then next to the text "Rectangle (x1, y1, x2, y2)" there are four edit fields, where the actual x/y positions are shown and can be modified.
- If a line, polygon or a curve is currently selected, a table will be available showing the actual X-Position and Y-Position of each of the black squares which mark the shape of the element, as soon as it is selected. These values can be edited here.

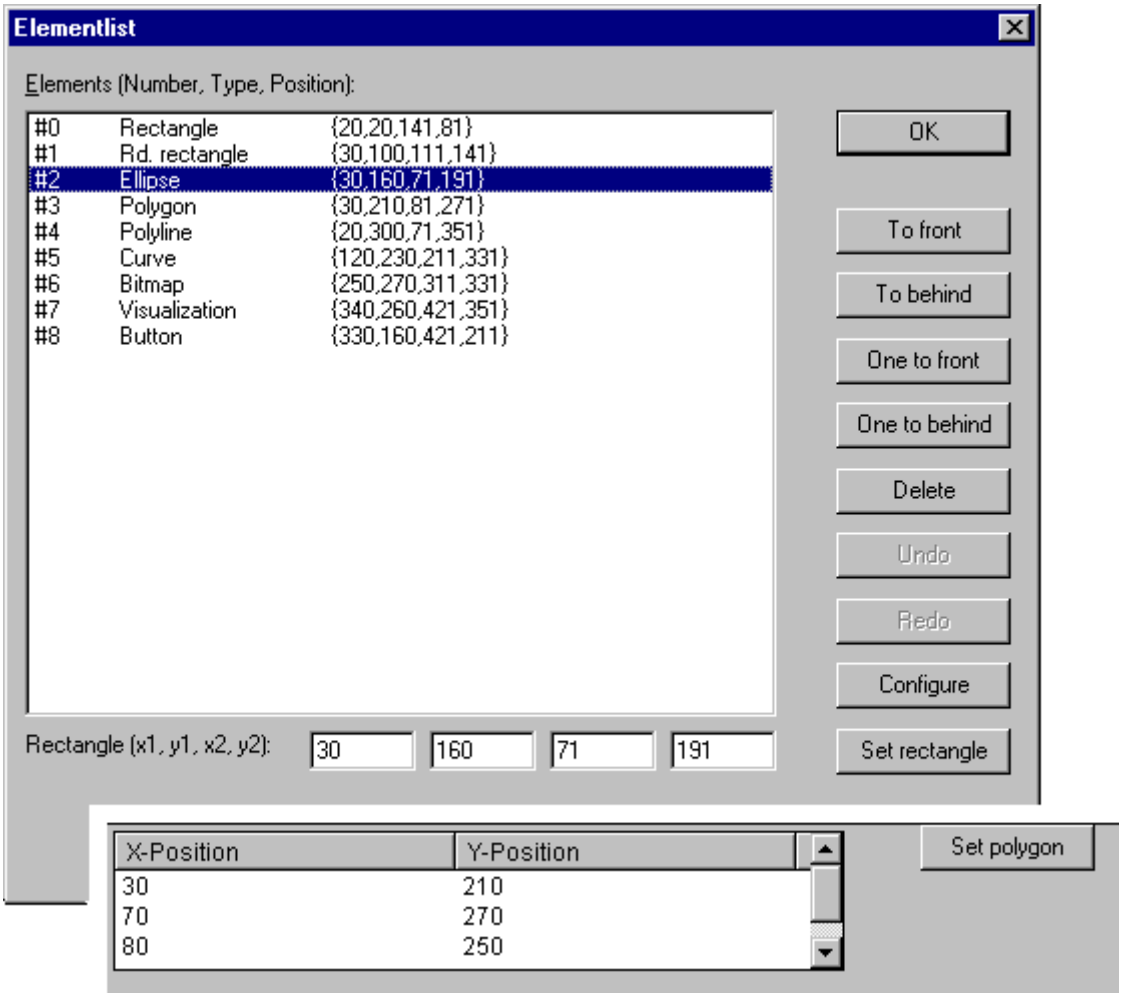
To set the modified position values in the elements list and in the visualization, press button *[Set rectangle]* (in case 1.) resp. *[Set polygon]* (in case 2.).

Use the Delete button to remove selected visualization elements.

Use the *[Undo]* and *[Redo]* buttons to undo or restore changes that have been made just as you would do with the commands 'Edit' 'Undo' and 'Edit' 'Redo' . In the dialog box, you can observe the changes that are being made.

Click on *[OK]* to close the dialog box and confirm the changes.

Use *“Configure”* to get the configuration dialog for the element.



1.4.3.4.13 Status bar in the visualization

If a visualization has the focus, the current X and Y position of the mouse cursor in pixels relative to the upper left corner of the image is displayed in the status bar. If the mouse pointer is located on an Element, or if the element is being processed, then the number of the element will be displayed. If you have selected an element to insert, then this element will also appear (for example, Rectangle).

1.4.3.5 Configuring visualization elements

1.4.3.5.1 Overview

In the configuration dialogs opened via the 'Extras' 'Configure' command the properties of an element or object are set either by activating options or dynamically by inserting project variables ↳ *Chapter 1.4.3.5.4 “'Extras' 'Configure'” on page 649*. Besides that the properties can be programmed via the components of a structure variable, which can be defined for each visualization element.

Regard the order of analysis, which will be followed in online mode:

- *The values which are given dynamically, i.e. via project variables, will overwrite the fix parameters defined for the same property.*
- *If an element property is defined by a "normal" project variable as well as by the component of a structure variable, then in online mode primarily the value of the project variable will be regarded.*

Please regard the possibility of using placeholders as well as the special input possibilities which are useful if the visualization should be used in CODESYS HMI or web visualization , that means if the visualization serves as the only user interface for a PLC program ↳ *Chapter 1.4.3.5.30 “Special input possibilities for operating versions” on page 695*.

Dialogs which include the configuration of Colors and Fonts for a visualization element, can look differently depending on the currently selected target system. Possibly instead of the standard dialog there is a dialog with restricted options. This might be reasonable for projects, which are designated for a use in target visualizations.

Also regard that in the project options a separate directory can be defined for visualization files ↳ *Chapter 1.4.1.2.2.7 “Options for directories” on page 207*.

1.4.3.5.2 Placeholder

At each location in the configuration dialog at which variables or text are entered, a placeholder can be set in place of the respective variable or text. This makes sense if the visualization object is not to be used directly in the program, but is created to be inserted in other visualization objects as an "instance". When configuring such an Instance, the placeholders can be replaced with variable names or with text (see Configuring an inserted visualization , there you also find an example for using placeholders ↳ *Chapter 1.4.3.5.28 “Visualization” on page 693*).

Any string enclosed in two dollar signs ($) is a valid placeholder (e.g. $variable1$, variable$x$). For each placeholder a „value group" can be defined as an input specification in the 'Placeholder list' dialog (called from 'Extras' 'List of placeholders'). With one of these values you can replace the placeholder when configuring an instance of the visualization object. A placeholder list will be available in the instance to do this replacements.

Example of an application of the placeholder concept

Instances of a function block can easily be displayed with the help of instances of the same visualization. For example, in configuring the visualization visu, which visualizes the variables of function block, one could begin each variable entry with the placeholder FUB (e.g. FUB.a). If an instance of visu is then used (by inserting visu in another visualization or by calling via 'Zoom to vis.'), then in the configuration of this instance the placeholder FUB can then be replaced with the name of the function block instance to be visualized. This might look like shown in the following: In the project define a function block containing the following declarations:

```
FUNCTION_BLOCK fu
VAR_INPUT
 changecol : BOOL; (* should cause a color change in the
visualization *`)
END_VAR
```

In PLC_PRG define two instances of 'fu':

```
inst1_fu : fu;
inst2_fu : fu;
```

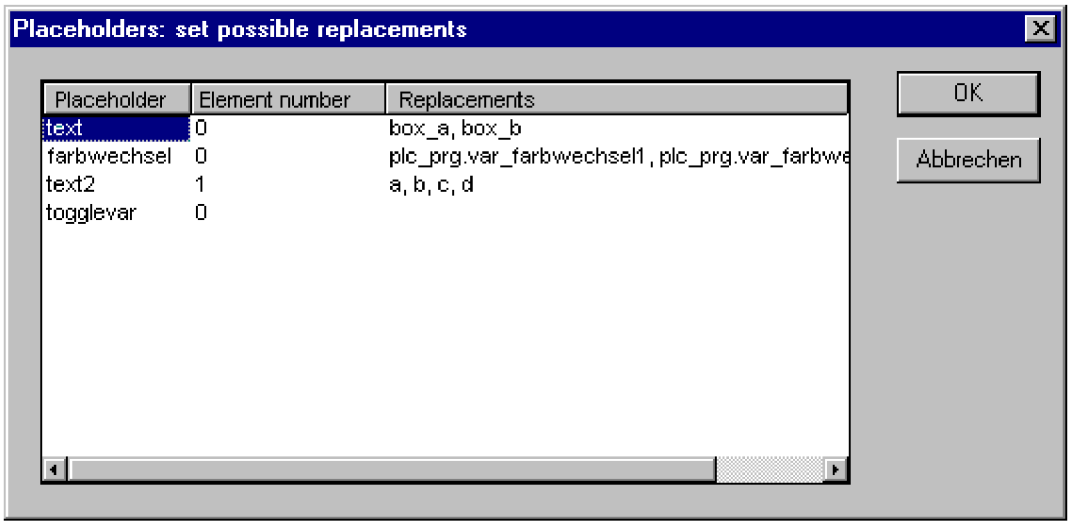
1. Create a visualization object 'visu'. Insert an element and open the configuration dialog, category 'Variables'. Enter in field 'Change color' the following: "FUB.changecol". Open category 'Input' and enter in field 'Tap Variable' "FUB.changecol". Open category 'Text' and enter"FUB - change color ".
2. Create another visualization object 'visu1'.
3. Insert visualization 'visu' twice in 'visu1' (two references of 'visu').
4. Mark the first reference of 'visu' and open the configuration dialog of category 'Visualization'. Press button 'Placeholder', so that the placeholder list will be displayed. There replace entry 'FUB' by 'PLC_PRG.inst_1'.
5. Now mark the second instance of 'visu' and (like described for the first one) replace 'FUB' by 'PLC_PRG.inst_2'.

   ⇨ In online mode, the values of the variables which are used to configure the two instances of 'fu' will be visualized in the corresponding instance of 'visu'. Of course the placeholder FUB can be used at all places in the configuration of 'visu' where variables or text strings are entered.

1.4.3.5.3 'Extras' 'List of Placeholders'

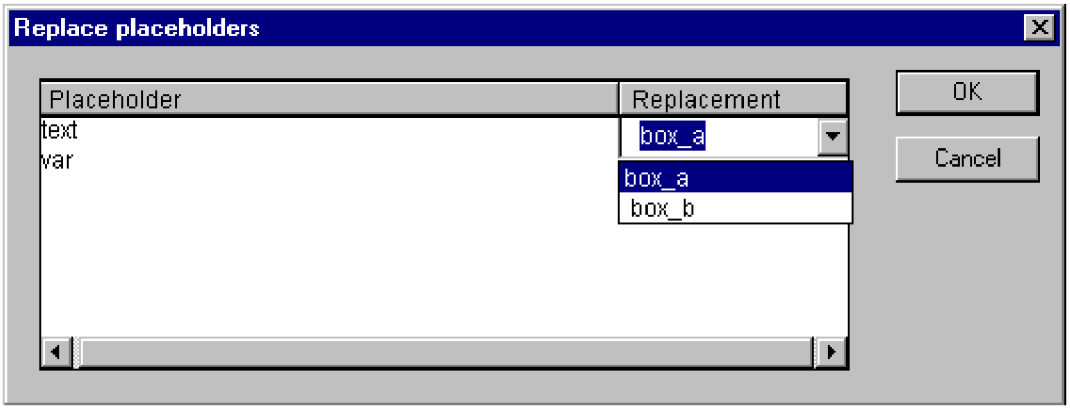
This list is used to manage placeholders and to configure them:

Primarily you use the list when configuring a visualization object, which later should be inserted, which means instanced, in other visualization(s). For this reason you will use placeholders instead of or additionally to variables and strings in the configuration dialogs. You can open the dialog 'Placeholders' by the command 'List of Placeholders' in the 'Extras' menu or in the context menu.



Column 'Placeholder' will list all placeholders, which are currently used in the configuration of the visualization object. Column 'Element number' shows the elements which contain a placeholder. In column 'Replacements' you can enter one or several strings (text, variable, expression) which you want to get available later when replacing a placeholder during the configuration of an instance of the visualization object. The elements of the selection must be entered separated by commas. If no or an impossible replacement string is specified, then the placeholder can be replaced with any desired text later during the configuration of the visualization's reference.

Later you use the list of placeholders when configuring an instance of the above mentioned visualization object, that means after this object has been inserted (as a 'reference') in another visualization by the command 'Insert' 'Visualization'. For this purpose do the following to open the dialog: Select the inserted visualization, execute command 'Extras' 'Configure' and press button 'Placeholders' in Category 'Visualization'. In this case the dialog will only contain two columns:



Column 'Placeholder' – like described above – shows all placeholders which have been defined for the primary visualization object. If additionally a selection of possible replacements had been defined, this list will now be available in column 'Replacement'. Select one of the entries to replace the placeholder in the present instance. If no replacements have been pre-defined then you can manually enter an expression or variable. For this purpose perform a mouse-click on the field in column Replacement to open an editor field.

1.4.3.5.4 'Extras' 'Configure'

With this command, the 'Configure element' dialog opens for configuring the selected visualization element ↬ *Chapter 1.4.3.4.1 “Selecting visualization elements” on page 643*. You are given the dialog box when you double-click on the element.

Select a category in the left area of the dialog box (available categories depending on element type) and fill out the requested information in the right area. This has to be done by activating options resp. by inserting the name of valid variables, whose values should define the property of the element.

There are also configuration dialogs available for a group of elements. Regard that the settings will be valid for the "element" group. If you want to configure the particular elements of the group, you have to resolve the group.

If you have defined an element property by a "static" setting as well as dynamically by a variable, then in online mode the variable will overwrite the static value (Example: "Alarm color Inside" can be defined statically in category 'Color' and additionally dynamically in category 'Colorvariables' by a variable). If the setting is controlled by a "normal" project variable as well as by a structure variable, then the value of structure variable also will be overwritten by the "normal" project variable.

Meter, Bar Display and Histogram must be re-grouped before!

At locations in the element configuration where variables are operative, the following Entries are possible:

- Variable names, for which input assistant is available.
- Expressions which are assembled from component accesses, field accesses with constant index, variables and direct addresses.
- Operators and constants, which can be combined at will with the aforementioned expressions.
- Placeholders instead of variable names or text strings.

Examples of permissible expressions	x + y 100*PLC_PRG.a TRUE NOT PLC_PRG.b 9*sin(x + 100)+cos(y+100)

Function calls are not possible. Invalid expressions result in an error message on login („Invalid Watch expression..."). Examples of invalid expressions: fun(88), a := 9, RETURN.

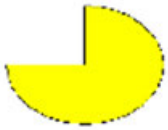
There are two possible ways in the configuration dialogs to write **global variables**: „.globvar" and „globvar" are equivalent. The style with a dot (which is that used in the Watch- and Receipt Manager) is not allowed within an assembled expression, however.

Regard also the possibility of using placeholders ↳ *Chapter 1.4.3.5.2 “Placeholder” on page 647*.

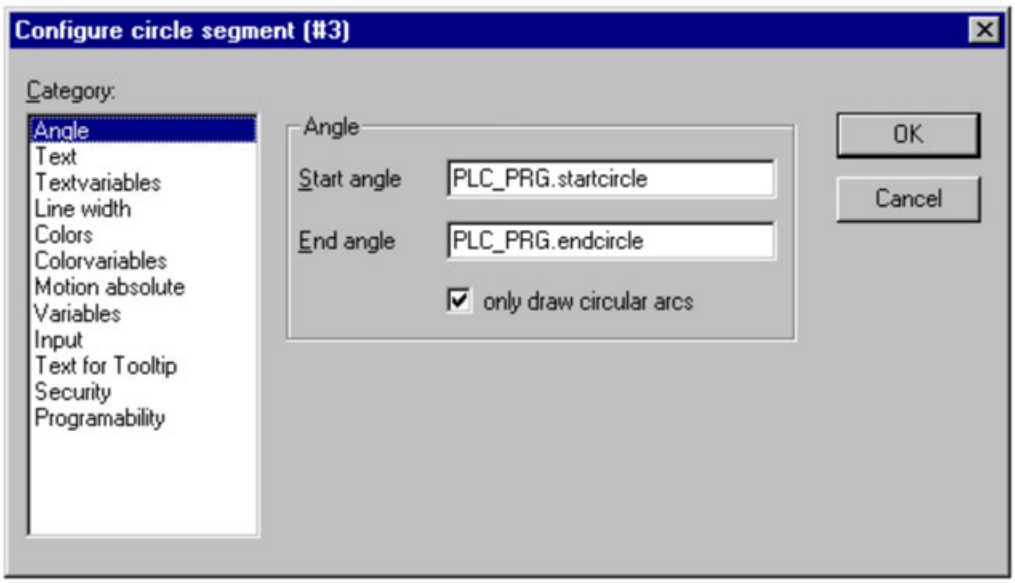
1.4.3.5.5 Angle

In the configuration dialog 'Configure Pie' in the Angle category you can each enter a value or a variable defining the start angle and the end angle of the sector element in degrees. The sector will be drawn clockwise from the start angle position to the end angle position.

Example:	Enter start angle: "90", end angle: "180"

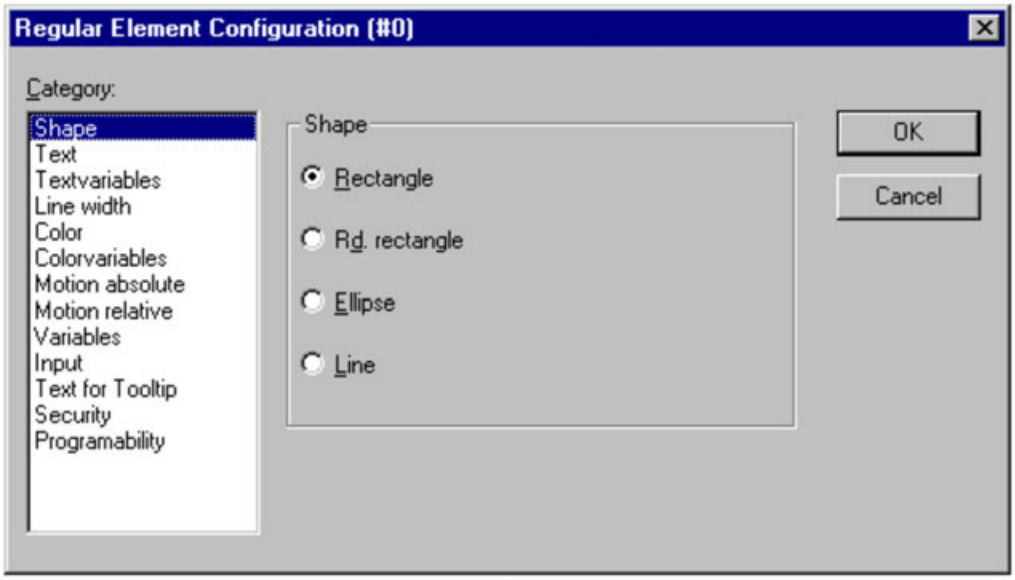


Dialog for Configuring a Pie



1.4.3.5.6 Shape

In the visualization element configuration dialog box, you can select in the 'Shape' category from among Rectangle, Rounded Rectangle, Line and Ellipse respectively Polygon, Line and Curve. The form will change into the size already set.

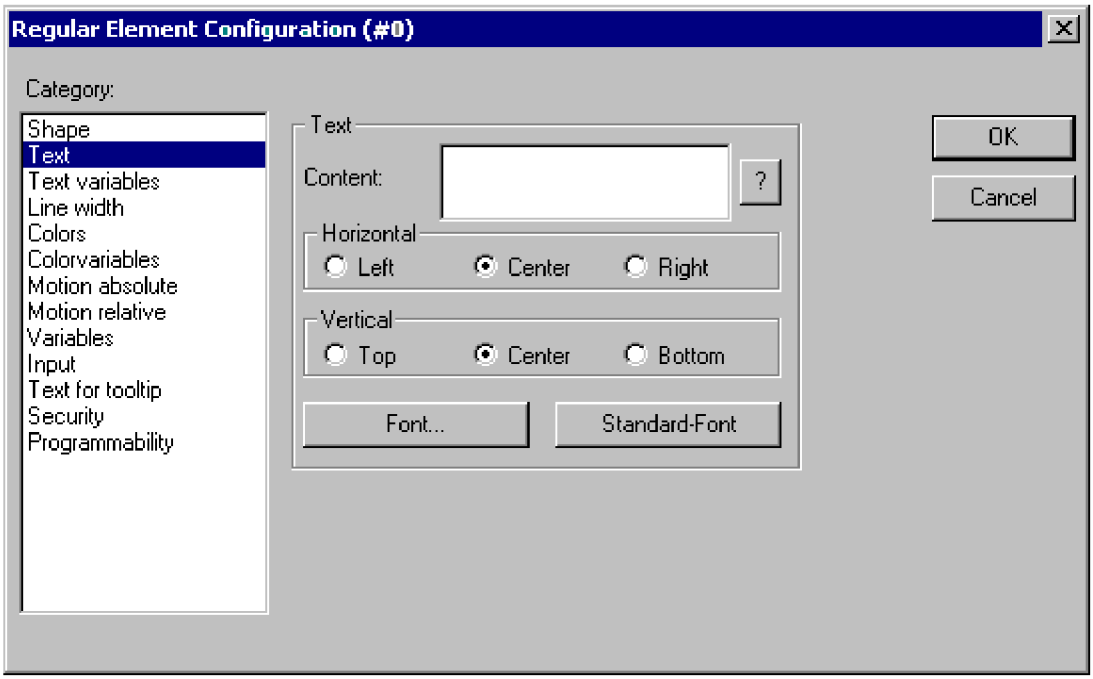


1.4.3.5.7 Text

In the dialog for configuring visualization elements, you can specify a text for the element in the Text category. This can be entered directly or/and a variable can be defined which will determine the text string. The usage of placeholders is possible ↳ *Chapter 1.4.3.5.2 “Placeholder” on page 647*. Also the default settings for font and alignment are done here.

> *As soon as text parameters are additionally provided dynamically, which means by a system or structure variable (see resp. 'Programmability'), the static definitions which are done in the currently opened dialog , will be overwritten*
> ↳ *Chapter 1.4.3.5.8 “Textvariables” on page 655!*

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.



Enter the text in the Content field. With the key combination *[Ctrl]* + *[Enter]* you can insert line breaks, with *[Ctrl]* + *[Tab]*, tab stops. Besides the input of a pure text string you can use the following formatting sequences (via button the corresponding online help page can be opened):

- If you include "%s" into the text, then this location, in Online mode, will be replaced by the value of the variable from the Textdisplay field of the Variables category. Instead of "s" you also can use other formatting strings conforming with the standard C-library function 'sprintf': See here a sample list of strings and arguments:

Character	Argument / Output as
d,i	Decimal number
o	Unsigned octal number (without leading zerol)
x	Unsigned hexadecimal number (without leading 0x)
u	Unsigned decimal number
c	Single character

Character	Argument / Output as
s	String
f	REAL-values; syntax: `%|<alignment><minimal width><accuracy>|f` Accuracy defines the number of decimal places behind the comma (Default is 6). Example see below.

- If you want to get displayed a percent sign% combined with one of the formatting strings mentioned above, you must enter "%%". For example: Enter "Rate in %%: %s" to get displayed in online mode "Rate in %: 12" (if the text display variable currently is "12").

- Be careful with letter case: capital letters, i.e. %S instead of %s not acceptable."

The value of the variable will be displayed correspondingly in online mode. You can enter any IEC-conforming format strings, which fit to the type of the used variable.

It is not checked whether the type which is used in the formatting string matches with the type of the variable which is defined in the 'Text Output' field!

Example:

Input in the 'Content' field: `Fill level %2.5f mm`

Input in the 'Textdisplay' field (variable of type REAL), e.g.: `plc_prg.fvar1`

-> Output in online mode e.g.: `Fill level 32.8999 mm`

- If you enter "%t", followed by a certain sequence of special placeholders, then this location will be replaced in Online mode by the system time. The placeholders define the display format, see the following table.

Do not insert any other characters before %t in the 'Content' field (in contrast this is allowed for e.g. "%s", see above)

%a	Abbreviated weekday name
%A	Full weekday name
%b	Abbreviated month name
%B	Full month name
%c	Date and time representation appropriate for locale
%d	Day of month as decimal number (01 "“ 31)
%H	Hour in 24-hour format (00 "“ 23)
%I	Hour in 12-hour format (01 "“ 12)
%j	Day of year as decimal number (001 "“ 366)
%m	Month as decimal number (01 "“ 12)
%M	Minute as decimal number (00 "“ 59)

%p	Current locale’s A.M./P.M. indicator for 12-hour clock
%S	Second as decimal number (00 "“ 59)
%U	Week of year as decimal number, with Sunday as first day of week (00 "“ 53)
%w	Weekday as decimal number (0 "“ 6; Sunday is 0)
%W	Week of year as decimal number, with Monday as first day of week (00 "“ 53)
%x	Date representation for current locale
%X	Time representation for current locale
%y	Year without century, as decimal number (00 "“ 99)
%Y	Year with century, as decimal number
%z, %Z	Time-zone name or abbreviation; no characters if time zone is unknown
%%	Percent sign

Examples: %t%a %b %d.%m.%y %H:%M:%S

-> Display in online mode: Wed Aug 28.08.02 16:32:45

Between the placeholders you can insert any text strings:

%Today is %d.%m.%y

-> Display in online mode: Today is 28.08.02

If a text string is to be transferred into a translation file, which will then be used in Online mode to enable switching into another national language, it must be delimited at the beginning and end by #.

Examples: "#Pump 1#" or else even "#Pump# 1".

The second case might for example in the event of multiple occurrences of the text Pump (Pump 1, Pump 2, etc.), prevent multiple appearances in the translation.

- If you include"%<PREFIX>" into the text, you can enter instead of "PREFIX" a certain string, which will serve as an identifier concerning the use of dynamic texts. The prefix will be used together with an ID number, which is to be defined in the 'Variables' category of the configuration dialog in field 'Textdisplay'. The combination references to a certain text, which is defined in a XML file available for this purpose and listing all possible dynamic texts. Thus at run time the text which is indicated by the current ID-Prefix-combination will be displayed. For further information see also the description of Language Switching ↳ *Chapter 1.4.3.8 “Language switching” on page 706*. See there also information on the usage of Unicode-Format.

The configured text will appear online in the prescribed alignment within the element: horizontally left, center or right and vertically top, center or bottom.

Via button *“Font”* the dialog for font selection will appear. Choose the desired font and confirm with OK. (The list of offered fonts depends on the target system.) The Standard-Font button can be used to assign the currently valid standard font to the element. That is defined in the visualization settings in category Display ↳ *Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*.

1.4.3.5.8 Textvariables

In category 'Textvariables' of the dialog for configuring visualization elements you can specify a variable which should dynamically set color and font of that string which is defined in category 'Text' ↳ *Chapter 1.4.3.5.7 “Text” on page 652*. At best enter the variable name with the aid of the input assistant (*[F2]*).

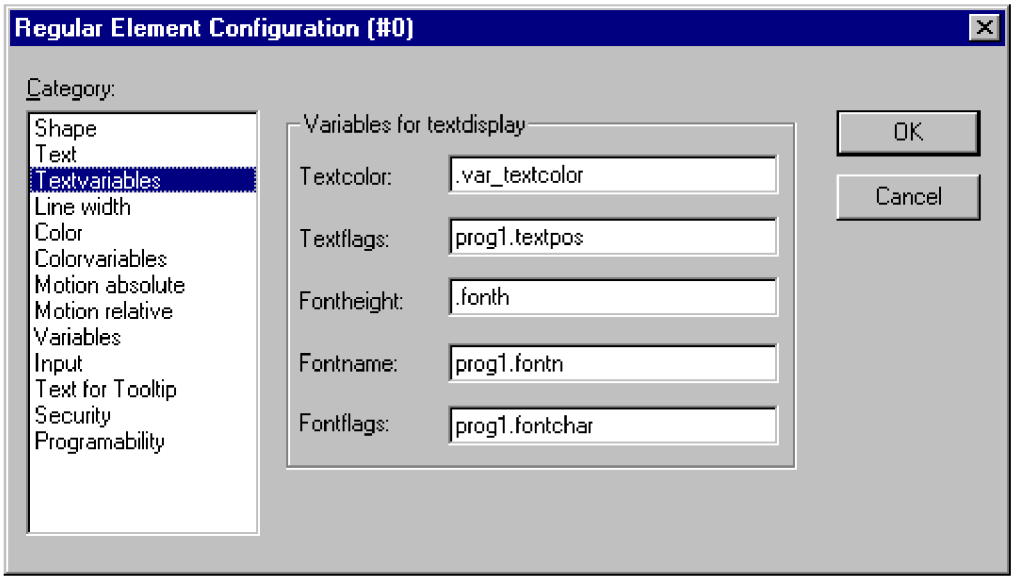
You can also use components of the structure VisualObjectType to set the text properties. For this see the description of category 'Programability' ↳ *Chapter 1.4.3.5.18 “Programmability” on page 665*; there you will find the possible values of the particular structure components and their effect.

If there are corresponding static definitions in category 'Text', these will be overwritten by the dynamic parameter values.

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.

The parameters of the dialog:

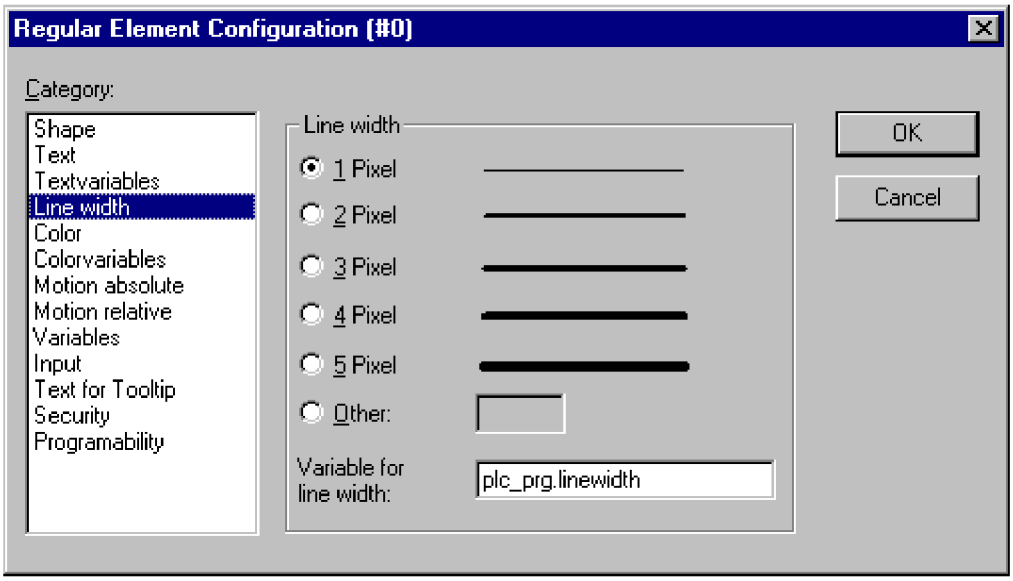
Parameter:	**Meaning:**	**Example entry of project variable:**	**Example Usage of variable in program:**	**corresponding component of structure VisualObjectType:**
Textcolor:	Text color	"plc_prg.var_textcolor"	var_textcolor=16#FF00FF -> Farbe	dwTextColor
Textflags:	Alignment (right, left, centered...)	"plc_prg.textpos"	textpos:=2 -> Text right justified	dwTextFlags
Fontheight:	Font height in Pixel	".fonth"	fonth:=16; -> Font height 16 pt	ntFontHeight
Fontname:	Font name	"vis1.fontn"	fontn:=arial; -> Arial is used	stFontName
Fontflags:	Font display (bold, underlined, italic...)	"plc_prg.fontchar"	fontchar:=2 -> Text will be displayed bold	dwFontFlags



1.4.3.5.9 Line width

In the dialog for configuring visualization elements, you can choose the line width for an element. As predefined options you find width settings from 1 to 5 pixel, additionally an other value can be entered manually (Other:), or a project variable (Variable for line width:) can be inserted. For the latter the input assistance (*[F2]*) can be used.

> *As soon as the parameter is additionally defined dynamically, i.e. by a structure variable, see category 'Programmability', the static setting will be overwritten in online mode* ↳ *Chapter 1.4.3.5.18 “Programmability” on page 665.*

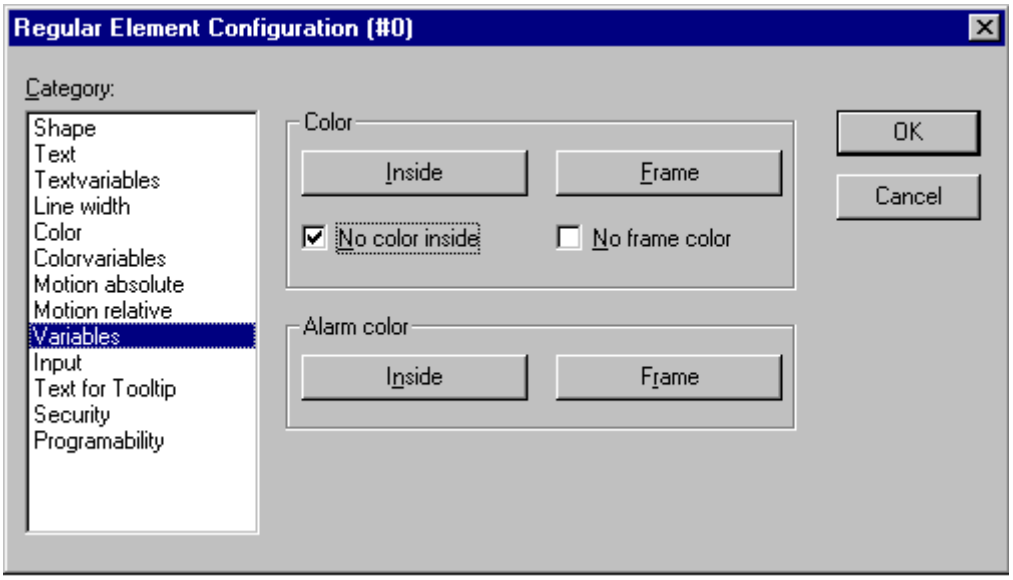


1.4.3.5.10 Colors

In the visualization element configuration dialog box, in the Color category you can select primary colors and alarm colors for the inside area and for the frame of your element. Choosing the options no color inside and no frame color you can create transparent elements.

As soon as the parameter is additionally defined dynamically by a variable, the static setting will be overwritten in online mode.

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.



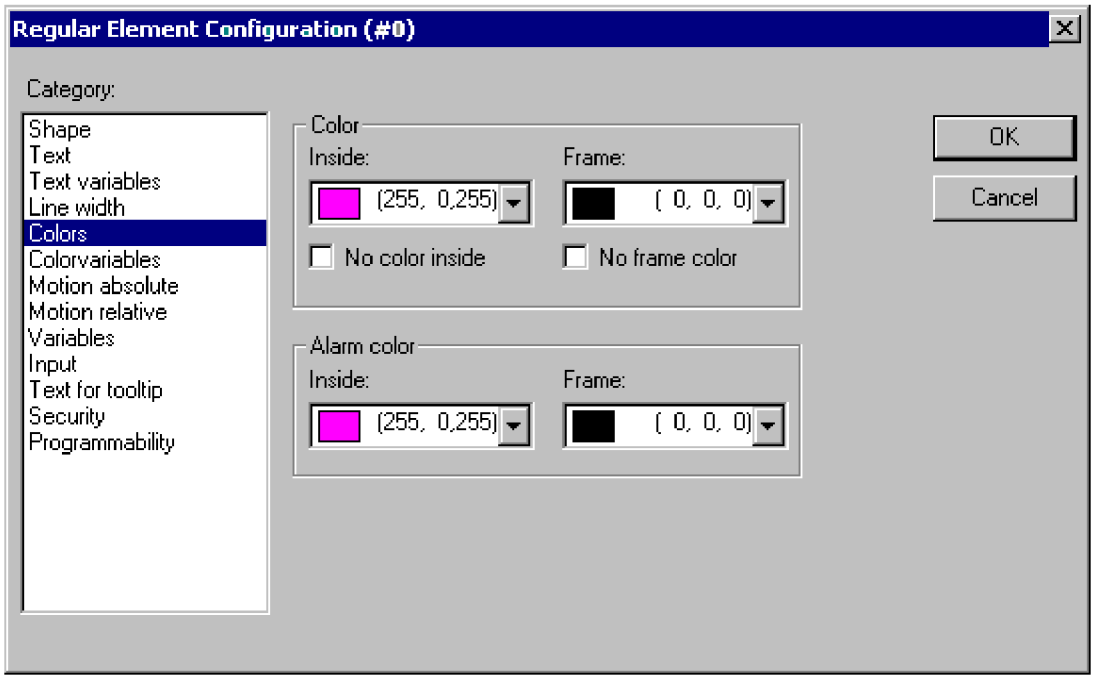
If you now enter a Boolean variable in the Variables category in the Change Color field, then the element will be displayed in the Color set, as long as the variable is FALSE. If the variable is TRUE, then the element will be displayed in its Alarm Color.

The change color function only becomes active, if the PLC is in Online Mode!

If you want to change the color of the frame, then press the Frame button, instead of the Inside button. In either case, the dialog box will open for selection of the color.

Here can to choose the desired hue from the primary colors and the user-defined colors. By pressing the Define Colors you can change the user-defined colors.

Depending on the target the dialog might offer only a restricted selection of colors. This can be reasonable for creating projects which are intended for target visualizations. In this case the colors are defined via selection lists.



1.4.3.5.11 Color variables

Here you can enter project variables (e.g. PLC_PRG.color_inside), which should determine the particular property in online mode: These property definitions also or additionally can be programmed with the aid of components of the structure VisualObjectType. Therefore see the description on the "Programmability" of a visualization element. There you will find a list of the possible values and their effects.

> *The variables, entered in the Color Variables dialog, in online mode will overwrite the static values given in the 'Color' category as well as corresponding values given by a structure variable.*

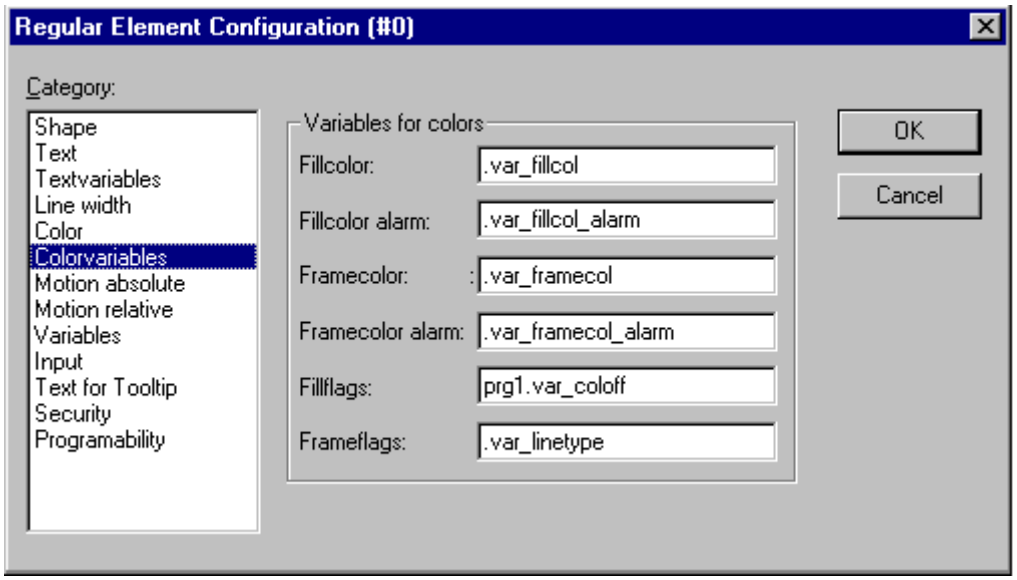
In case of multiple definition of an element property consider the specific order of precendence concerning according to which a value might be overwritten in online mode by another.

The parameters of the dialog:

Parameter	Description	Example of an entry	Example for using the variable in the program	corresponding component of structure VisualObjectType
Fillcolor:	fill color	"plc_prg.var_fillcol"	var_var_fillcol:=16#FF00FF -> fill color pink	dwFillColor
Fillcolor alarm:	fill color if the 'Change color' variable is TRUE (↳ *Chapter 1.4.3.5.10 “Colors” on page 656*)	"plc_prg.var_fillcol_a"	var_fillcol_a:=16#FF00FF -> alarm fill color pink	dwFillColorAlarm

Parameter	Description	Example of an entry	Example for using the variable in the program	corresponding component of structure VisualObjectType
Framecolor:	frame color	"plc_prg.var_framecol"	var_framecol::=16#FF00FF -> frame color pink	dwFrameColor
Framecolor alarm:	frame color if the 'Change color' variable is TRUE (*Chapter 1.4.3.5.10 "Colors" on page 656*)	"plc_prg.var_framecol"	var_framecol:=16#FF00FF -> alarm frame color farbe pink	dwFrameColorAlarm
Fillflags:	The current inside color configuration can be activated (FALSE) resp. deactivated (TRUE)	"plc_prg.var_col_off"	var_col_off:=1 -> the color definition for the fill color will not be regarded, that for the frame remains valid	dwFillFlags
Frameflags:	Display of the frame (solid, dotted etc.)	"plc_prg.var_linetype"	var_linetype:=2; -> frame will be displayed as dotted line	dwFrameFlags

Dialog Box for Configuring Visualization Elements (Category Colorvariables):



1.4.3.5.12 Motion absolute

In the visualization element configuration dialog box, in the Motion absolute category, X- or Y-Offset fields variables can be entered. These variables can shift the element in the X or the Y direction, depending on the respective variable value. A variable in the Scale field will change the size of the element linear to its current value. This value, which is used as scaling factor, will be divided by 1000 implicitely, so that it is not necessary to use REAL-variables in order to get a reduction of the element. The scaling always will refer to the balance point.

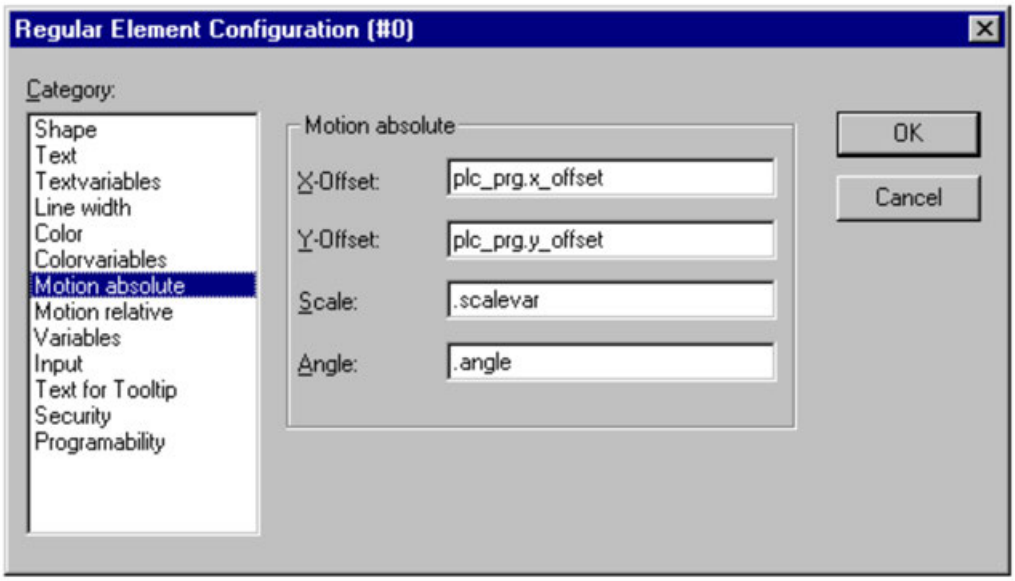
A variable in the Angle field causes the element to turn on its turning point, depending on the value of the variable. (Positive Value = Mathematic Positive = Clockwise). The value is evaluated in degrees. With polygons, every point rotates; in other words, the polygon turns. With all other elements, the element rotates, in such a way, that the upper edge always remains on top.

The turning point appears after a single click on the element, and is displayed as a small black circle with a white cross. You can drag the turning point with a pressed left mouse button.

> *In online mode the variables which are set in the 'Motion absolute' dialog will override the values of structure components which additionally might be used to define the same property ('Programability').*

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.

Visualization Element Configuration Dialog Box (Motion Absolute Category)



1.4.3.5.13 Motion relative

In the dialog for configuring visualization elements in the Motion Relative category, you can assign variables to the individual element edges. Depending on the values of the variables, the corresponding element edges are then moved. The easiest way to enter variables into the fields is to use the Input Assistant (*[F2]*).

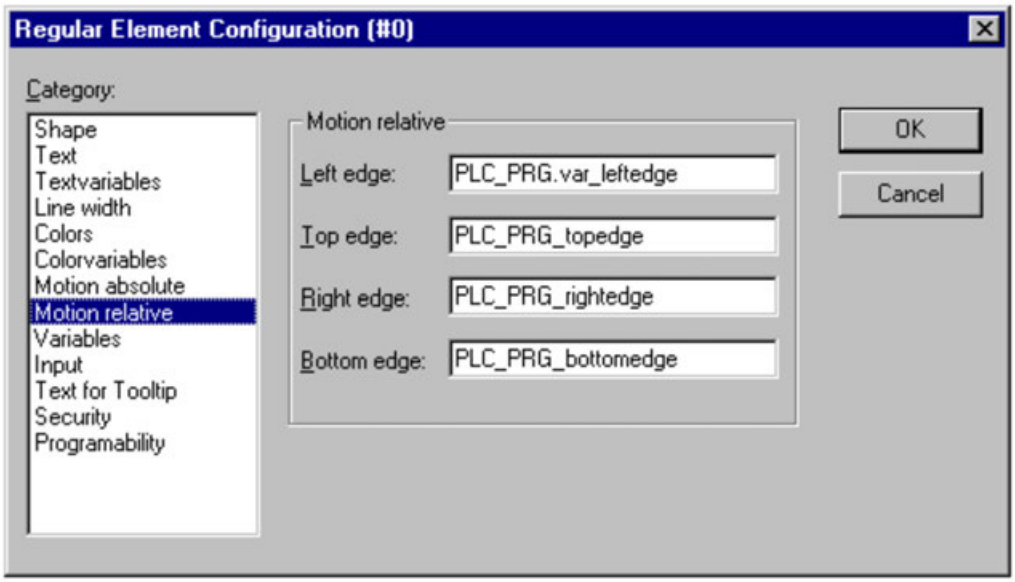
The four entries indicate the four sides of your element. The base position of the corners is always at zero. A new value in the variables, in the corresponding column, shifts the boundary in pixels around this value. Therefore, the variables that are entered ought to be INT variables.

> *Positive values shift the horizontal edges downward, or, the vertical edges, to the right!*

> *In online mode the variables which are set in the 'Motion absolute' dialog will override the values of structure components which additionally might be used to define the same property ('Programmability').*

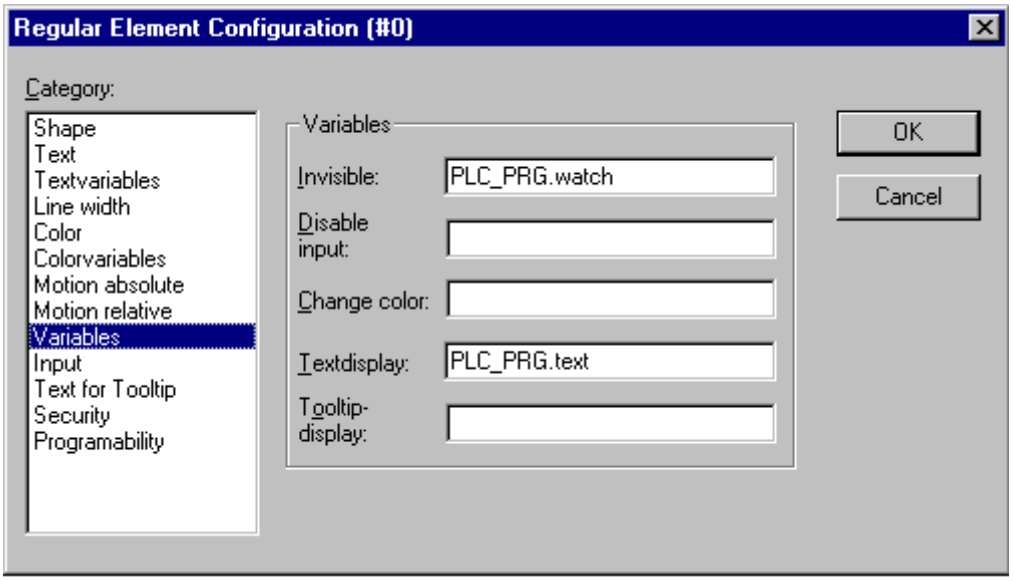
In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.

Dialog Box for Configuration of Visualization Elements (Motion Relative Category)



1.4.3.5.14 Variables

Visualization Element Configuration Dialog Box (Variables Category):



You can enter the variables that describe the status of the visualization elements in the Variable category within the dialog box for configuring visualization elements. The simplest way to enter variables in the fields is to use the Input Assistant.

> *In online mode the variables which are set in the 'Motion absolute' dialog will override the values of structure components which additionally might be used to define the same property ('Programmability').*

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.

You can enter Boolean variables in the Invisible and Change color fields. The values in the fields determine their actions. If the variable of the Invisible field contains the value FALSE, the visualization element will be visible. If the variable contains the value TRUE, the element will be invisible.

Disable input: If the variable entered here is TRUE, all settings of category 'Input' will be ignored.

Change color: If the variable which is defined in this field, has the value FALSE, the visualization element will be displayed in its default color. If the variable is TRUE, the element will be displayed in its alarm color.

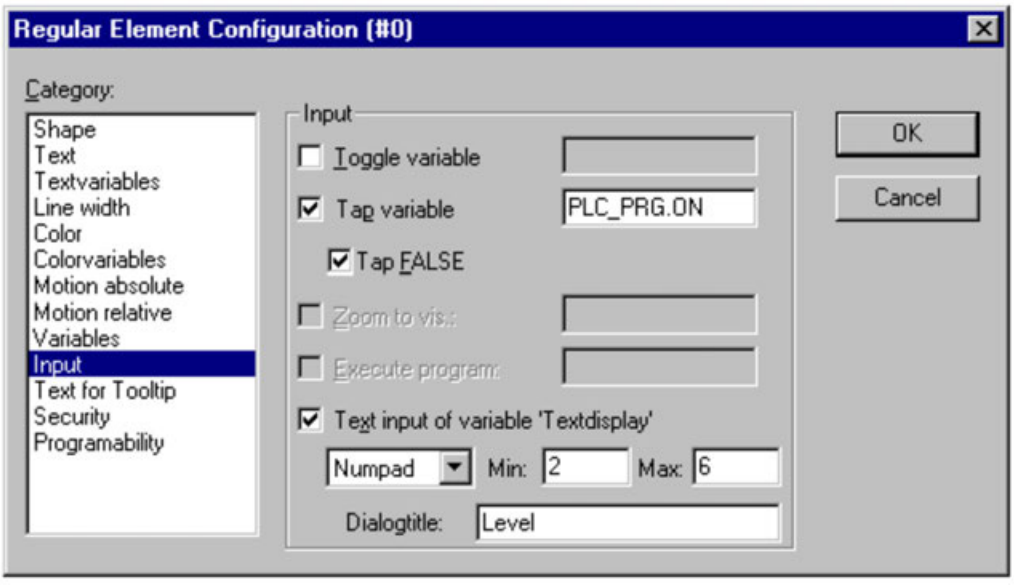
Textdisplay:

- If you have inserted a "%s" in the Content field of the Text category or if you have included "%s" in the textstring, then the value of the variable which is defined in 'Textdisplay' will be displayed in online mode in the visualization object. "%s" will be replaced by the value.
- If you have inserted resp. included a "%<PREFIX>" in the Content field of the Text category ("PREFIX" must be a sequence of letters), then the variable resp. the numeric value which is entered here in 'Textdisplay' will be interpreted as an ID, which in combination with the prefix serves as a reference on a text, which is described in a XML file. This text will be displayed in online mode instead of "%<PREFIX>" in the visualization object. Thus a dynamic modification of the text display is possible.
- If you want to edit the value of the variable in Online mode using the keyboard, you can do this via the 'Text input of variable' 'Textdisplay' in the Input category.

Tooltip-display: Here you can enter a string or a variable of type STRING, which defines the ID for a dynamic tooltip text. In combination with a prefix to be defined in category 'ToolTip' uniquely a certain text in a dynamic textlist can be called to be displayed as tooltip ↬ *Chapter 1.4.3.5.16 "Text for ToolTip" on page 664*. Thus dynamic text switching resp. language switching is also possible for tooltip texts.

1.4.3.5.15 Input

Dialog for configuring the visualization elements (Category Input):



Toggle variable: If this option is activated, in online mode you will toggle the value of the variables which are located in the input field by each mouse click on the visualization element. You can obtain input assistance for data entry via *[F2]*. The value of the Boolean variable changes with each mouse click from TRUE to FALSE and then back to TRUE again at the next mouse click, etc.

Tap Variable: If this option is activated, in online mode you can switch the value of the Boolean variable which is located in the input field, between TRUE and FALSE. Place the mouse cursor on the element, press the mouse-key and hold it depressed. If option Tap FALSE is activated, the value is set to FALSE as soon as the mouse key is pressed, otherwise it is set to TRUE at this moment. The variable changes back to its initial value as soon as you release the mouse key.

Zoom to Vis...: If this option is activated, you can enter in the edit field the name of a visualization object of the same project to which you want to jump by a mouse-click on the element in online mode. In this case always first the window of the target visualization will be opened before that of the current one will be closed.

The following entries are allowed:

- The name of a visualization object of the current project (see Object Organizer)
- If a visualization reference that contains placeholders is to be jumped to, the placeholders can be directly replaced by variable names or text when called up ↬ *Chapter 1.4.3.5.2 “Placeholder” on page 647*. For this pupose, conform to the following syntax:
  <Visuname>(<Placeholder1>:=<Text1>, <Placeholder2>:=<Text2>,..., <Placeholder n>:=<Textn>).
  Example:
  Calling the visualization visu1, whereby the placeholders var_ref1 and var_ref2 used in visu1 are replaced by the variables PLC_PRG.var1 and PROG.var1 respectively:
  visu1(var_ref1:=PLC_PRG.var1, var_ref2:=PROG.var1)
- If a program variable of the type STRING (e.g. PLC_PRG.xxx) has been entered instead of a visualization object, then this variable can be used to define the name of the visualization object (e.g. ‚visu1') which the system should change to when a mouse click occurs (e.g. xxx:= ‚visu1).
- If you issue the command „ZOOMTOCALLER" in the Zoom to vis. field, a backward jump into the calling visualization is achieved in Online mode by a mouse click on the element, if such a constellation was configured.

The implicit variable CurrentVisu (type STRING, for implicit (system) variables see here) describes the name of the currently opened visualization object. For example it can be used in the application to control which visualization should be opened resp. to see which is the currently opened.

Note: with compiler version < V2.3.7.0, and if the library SysLibStr.lib is not included in the project, this will only work if the names of the visualization objects are defined in capital letters.

Example: CurrentVisu:='PLC_VISU';

Execute program: If this option is activated you can enter ASSIGN- or special "INTERN"-commands in the input field, which will be executed in online mode as soon as you perform a mouse-click on the element. Press button "..." to get the dialog Configure programs where you can select the desired commands (Add) and arrange them in the desired order (Before, After).

This feature especially is important if the visualization will be the only operating interface of a system (pure operating version).

Text input of variable 'Textdisplay': If this option is activated, in online mode you can enter text in an edit field in this visualization element. This value upon pressing *[Enter]* will be written to the variable that appears in the textdisplay field of the 'Variables' category.

If option Hidden is activated, the text will be replaced by asterisks ("***") in the online display of the visualization element.

Select in the scroll box which kind of input should be possible later in online mode

Text: An edit field will open, where you can enter the value.

Numpad resp. Keypad: A window will open showing an image of the numeric resp. alphabetic keypad, where you can enter a value by activating the appropriate key elements. This might be useful if the visualization must be operatable via a touch screen. The range of valid input values can be restricted by defining a minimum and a maximum value in the edit fields 'Min' and 'Max'.

> *In case of using target visualization regard the possibility to get information on user inputs via mouse-clicks by special interface functions and to use that directly in the project.*

In case of multiple definition of an element property consider the specific order of precedence concerning according to which a value might be overwritten in online mode by another.

1.4.3.5.16 Text for ToolTip

The dialog Text for Tooltip offers an input field for text which appears in a text field as soon as the mouse cursor is passed over the object in online mode. The text can be formatted with line breaks by using the key combination *[Ctrl] + [Enter]*.

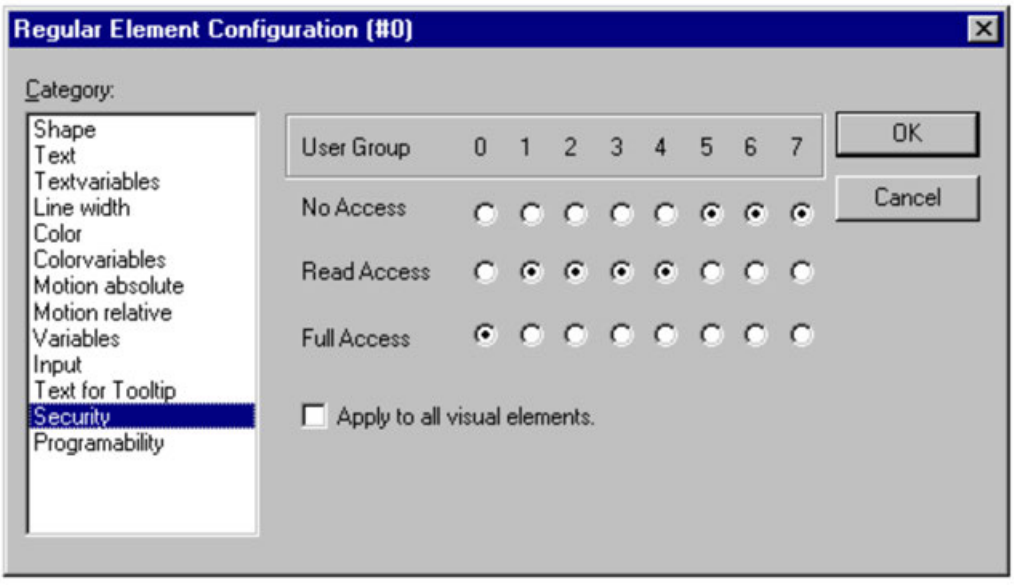
Dynamic language switching

If in field 'Content' you enter "%<PREFIX>" instead of a fix text string, "PREFIX" can be any string which serves as an identifier for the usage of dynamic texts. The prefix will be used in combination with an ID, which is to be specified in category 'Variables' in field 'Tooltip-display'. The combination prefix-ID references uniquely a certain text which is defined in a special language file in xml-format. Thus at run time always that text will be displayed which corresponds to the currently valid combination of prefix and ID. For further information see the description of the dialog 'Settings', category 'Language', and the help pages on Language Switching in the Visualization ↳ *Chapter 1.4.3.8 “Language switching” on page 706*.

Concerning the usage of Unicode format please also see the pages on Language Switching in the Visualization.

1.4.3.5.17 Security

It might be useful that different user groups get different operating possibilities and display of a visualization. This can be reached by assigning different access rights concerning particular visualization elements. You can do this for the eight user groups ↳ *Chapter 1.4.1.2.4.12 “'Project' 'Object properties'” on page 261* ↳ *Chapter 1.4.1.2.3.43 “'Project' 'Passwords for user groups'” on page 250*. The access rights can be assigned by activating the appropriate option in the configuration dialog 'Access rights' for a visualization element:



Access right	Effect in online mode
No Access	Element will not be visible
Read Access	Element will be visible but not operatable (no inputs allowed)
Full Access	Element is not visible and not operatable

If you want to assign the access rights also to all other elements of the visualization object, activate option 'Apply to all visual elements'.

Please regard, that the access rights which are set for the visualization object in the 'Project''Object' 'Properties' dialog, are independent on those of the particular visualization elements!

1.4.3.5.18 Programmability

Programability The properties of an visualization element can not only be defined by a static setting or by a "normal" project variable, but also by the components of a structure variable, which is exclusively used for programming visualization elements.

For this purpose the structure VisualObjectType is available in the library SysLibVisu.lib. Its components can be used to define most of the element properties.

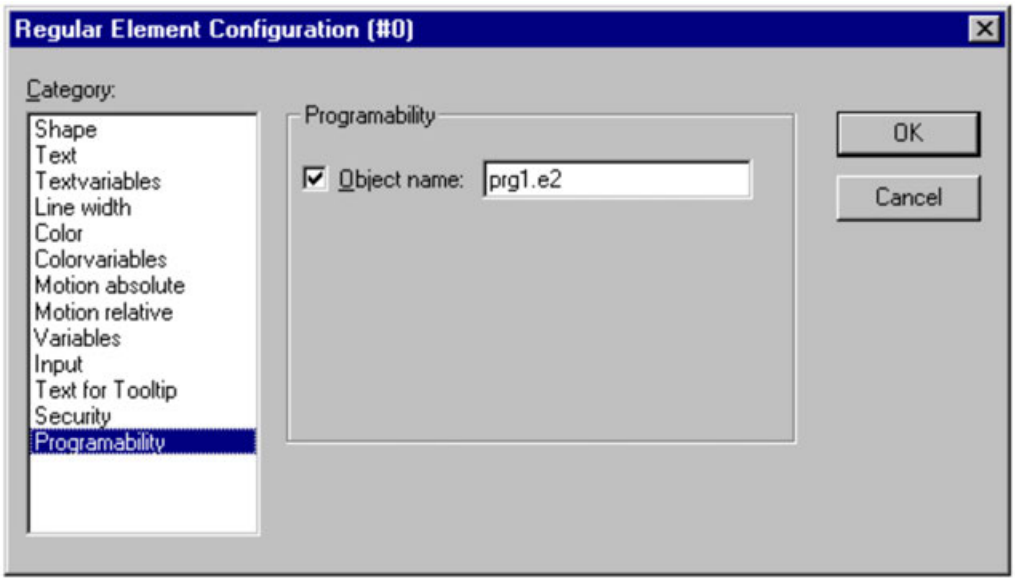
In case of multiple definition of a element property the value of the "normal" project variables will overwrite that of the structure variable and both will overwrite a static definition.

In order to configure the element properties by using a structure variable, do the following:

Open the configuration dialog, category 'Programmability' and enter a new, unique (!) variable name in the field 'Object Name'. For this purpose you must activate the option by a mouse-click in the checkbox. The variable automatically will be declared with type 'VisualObjectType', a structure which is contained in the library 'SysLibVisu.Lib'. The declaration is done implicitely and not visible for the user. Make sure that the library is included in the library manager.

After the next compile the newly assigned structure variable will be available in the project. (Hint: Activate the Intellisense functionality 'List components' in the project options, category Editor, in order to get the structure components in a selection list as soon as the variable name followed by a dot is entered).

Example: If you have defined a Object Name "visu1_line" for a visualization element, then you can program the line width of this element by e.g. "visu1_line.nLIneWidth:=4".



The structure VisualObject-Type

The following table will show you all components of the structure and references to the corresponding items in the different categories of the configuration dialog:

At the beginning of the component name the data type is integrated:

- n INT
- dw DWORD
- b BOOL
- st STRING

Component (+Data type)	Effect	Example (the Object Name "vis1" has been defined for the element)	corresponding entries in configuration dialog:
nXOffset : INT;	Shift element in X-direction	vis1.nXOffset:=val2; (element is set to position X=val2)	- Cat. Motion absolute: X-Offset
nYOffset : INT;	Shift element in Y-direction	vis1.nYOffset:=22; (element is set to position Y=val2)	- Cat. Motion absolute: Y-Offset
nScale : INT;	Change of the size	vis1.nScale:=plc_prg.scale_var; (element size changes linear with change of value of plc_prg.scale_var)	- Cat. Motion absolute: Scaling
nAngle : INT;	Rotating element around its center	vis1.anglevar:=15; (element rotates clockwise by 15)	- Cat. Motion absolute: angle
bInvisible : BOOL;	Element is visible / invisible	vis1.visible:=TRUE; (element is invisible)	- Cat. Color: No color inside + No frame color - Cat. Colorvariables: Fillcolor + Framecolor

Component (+Data type)	Effect	Example (the Object Name "vis1" has been defined for the element)	corresponding entries in configuration dialog:
stTextDisplay : STRING;	Text is displayed in element	vis1.TextDisplay:='ON / OFF'; element is inscribed with this text	- Cat. Text: entry at 'Content'
bToggleColor : BOOL;	color change when toggling between TRUE and FALSE	vis1.bToggleColor:=alarm_var; As soon as alarm_var gets TRUE, the element gets the color defined via the components dwFillColorAlarm, dwFrameColorAlarm resp. via the settings in category 'Colorvariables' or 'Color'.	- Cat. Input: Toggle variable + - Cat. Variables: Change color
bInputDisabled: BOOL;	if FALSE: Inputs in category 'Input' are ignored	vis1.bInputDisabled:=FALSE; (no input is possible for this elementt)	- Cat. Variables: 'Disable Input'
stTooltipDisplay:STRING;	Text of the tooltip	vis1.stTooltipDisplay:='Switch for .....';	- Cat. Text for Tooltip: Entry in 'Content:'
dwTextFlags:DWORD;	Text position: 1 left justified 2 right justified: 4 centered horizontally 8 top 10 bottom 20 centered vertically Note: Always set a horizontal and a vertical position (addition of values)!	vis1.dwTextFlags:=24; (Text will be placed in the center of the element (4 + 20)	- Cat. Text: Horizontal and Vertical options - Cat. Textvariables: Textflags
dwTextColor : DWORD;	Text color (definition of colors see subsequent to this table)	vis1.dwTextColor := 16#00FF0000; (Text is blue-colored)	- Cat. Text: Font | Color - Cat. Textvariables: Textcolor
nFontHeight : INT;	Font height in Pixel should be in range 10-96	vis1.nFontHeight:=16; (Text height is 16 pt)	- Cat. Text: Font | Grad - Cat. Textvariables: Font heigth

Component (+Data type)	Effect	Example (the Object Name "vis1" has been defined for the element)	corresponding entries in configuration dialog:
dwFontFlags : DWORD;	Font display. Available flags: 1 italic 2 fett 4 underlined 8 canceled + combinations by adding values	vis1.dwFont-Flags:=10; (Text is displayed blue and canceled)	- Cat. Text: Schrift | Schriftschnitt - Cat. Textvariables: Fontflags
stFontName : STRING;	Change font	vis1.stFontName:=’Arial’; (Arial is used)	- Cat. Text: Schrift | Schriftart - Cat. Textvariables: Fontname
nLineWidth : INT;	Line width of the frame (pixels)	vis1.nLWidth:=3; (Frame width is 3 Pixels)	- Cat. Line width
dwFillColor : DWORD;	Fill color (definition of colors see subsequent to this table)	vis1.dwFill-Color":=16#00FF0000; (Element ist im "Normalzustand" blau)	- Cat. Color: Color | Inside - Cat. Colorvariables: Inside
dwFillColorAlarm : DWORD;	Fill color as soon as bToggleColor gets TRUE, see above) (definition of colors see subsequent to this table)	vis1.dwFillColorAlarm:=16#00808080; (as soon as Variable togglevar gets TRUE, the element will be displayed grey-colored)	- Cat. Color: Alarm color | Inside - Cat. Colorvariables: Inside Alarm
dwFrameColor: DWORD;	Frame color (definition of colors see subsequent to this table)	vis1.dwFrame-Color:=16#00FF0000; (Frame is blue-colored)	- Cat. Color: Color | Frame - Cat. Colorvariables: Frame
dwFrameColorAlarm: DWORD;	Fill color as soon as bFrameColor gets TRUE, see above (definition of colors see subsequent to this table)	vis1.dwFrameColorAlarm:=16#00808080; (as soon as Variable vis1.bToggle-Color gets TRUE, the frame will be displayed grey-colored)	- Cat. Color: Alarm color | Frame - Cat. Colorvariables: Frame Alarm

Component (+Data type)	Effect	Example (the Object Name "vis1" has been defined for the element)	corresponding entries in configuration dialog:
dwFillFlags: DWORD;	Color, as defined by the color variables, can be displayed or ignored 0 = show color >0 = ignore setting	vis1.dwFillFlags:=1; (element gets invisible)	- Cat. Color: No color inside + No frame color - Cat. Colorvariables: Fillflags
dwFrameFlags: DWORD;	Display of frame: 0 full 1 dashed (---) 2 dotted ( ) 3 dash-point (_._._) 4 dash-point-point (_.._..) 8 blind out line	vis1.FrameFlags:=1; (Frame will be displayed as dashed line)	- Cat. Colorvariables: Frameflags

Defining color values

Example: e1.dwFillColor := 16#00FF00FF;

A color is entered as a hex number which is composed of the Blue/Green/Red (RGB) components. The first two zeros after "16#" should be set to in each case, to fill the DWORD size. For each color value 256 (0-255) colors are available.

- FF Blue component
- 00 Green component
- FF Red component

Example for a blinking visualization element

Define a global variable 'blink1' of type VisualObjectType in the configuration of a rectangle. In a program of function block the value of a component of the structure can be modified.

```
PROGRAM PLC_PRG
VAR
n:INT:=0;
bMod:BOOL:=TRUE;
END_VAR
(* Blinking element *)
n:=n+1;
bMod:= (n MOD 20) > 10;
IF bMod THEN
blinker.nFillColor := 16#00808080; (* Grau *)
ELSE blinker.nFillColor := 16#00FF0000; (* Blau *)
END_IF
```

1.4.3.5.19 Bitmap

You can do the following settings in category 'Bitmap' in the visualization element configuration dialog box:

Specify, which image file should be used. This specification can be static or dynamic. If there are entries both in the 'Bitmap' field and in the 'Bitmap Variable' field, then that in the 'Bitmap' field will be ignored:

Bitmap: Static definition: You can enter the path of an image file available in the local file system. Via button ... the standard dialog for browsing for a file will be opened, where you can select the desired file.

Bitmap variable: Dynamic definition: A project variable of type STRING can be entered here, containing the name of the image file currently to be used. This allows dynamic change of images in online mode, but only works with image files which are entered in the project-global "bitmap list" ↯ *Chapter 1.4.3.7.1 “'Extras' 'Bitmap list'” on page 706*. The string-variable always only must get assigned the file name, even if there is a full path specified in the bitmap list.

See in the following example 'stBitmap' as string-variable, which is got assigned various image files that are part of the global bitmap list.

```
CASE nId OF
0: stBitmap := 'background.bmp';
1: stBitmap := 'deutest.bmp';
2: stBitmap := 'alarm.bmp';
END_CASE
```

If stBitmap e.g. is declared in object PLC_PRG, it can be entered here in the configuration dialog in field 'Bitmap variable' like follows: "PLC_PRG.stBitmap".

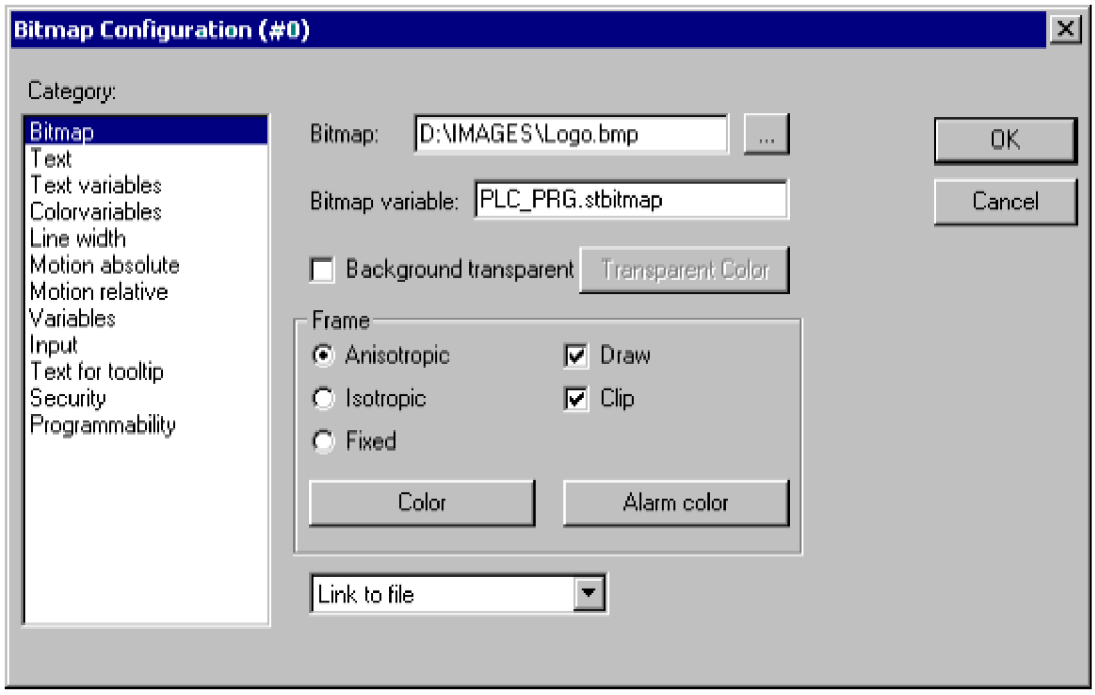
The following entries affect the frame of the bitmap:

By selecting Anisotropic, Isotropic or Fixed you specify how the bitmap should react to changes in the size of the frame. Anisotropic means that the bitmap remains the same size as the frame which allows you to change the height and width of the bitmap independently. Isotropic means that the bitmap retains the same proportions even if the overall size is changed (i.e., the relationship between height and width is maintained). If Fixed is selected, the original size of the bitmap will be maintained regardless of the size of the frame.

If the Clip option is selected together with the Fixed setting, only that portion of the bitmap that is contained within the frame will be displayed.

If you select the Draw option, the frame will be displayed in the color selected in the Color and Alarm color buttons in the color dialog boxes (standard or target-specific options). The alarm color will only be used if the variable in the Change Color field in the Variable category is TRUE.

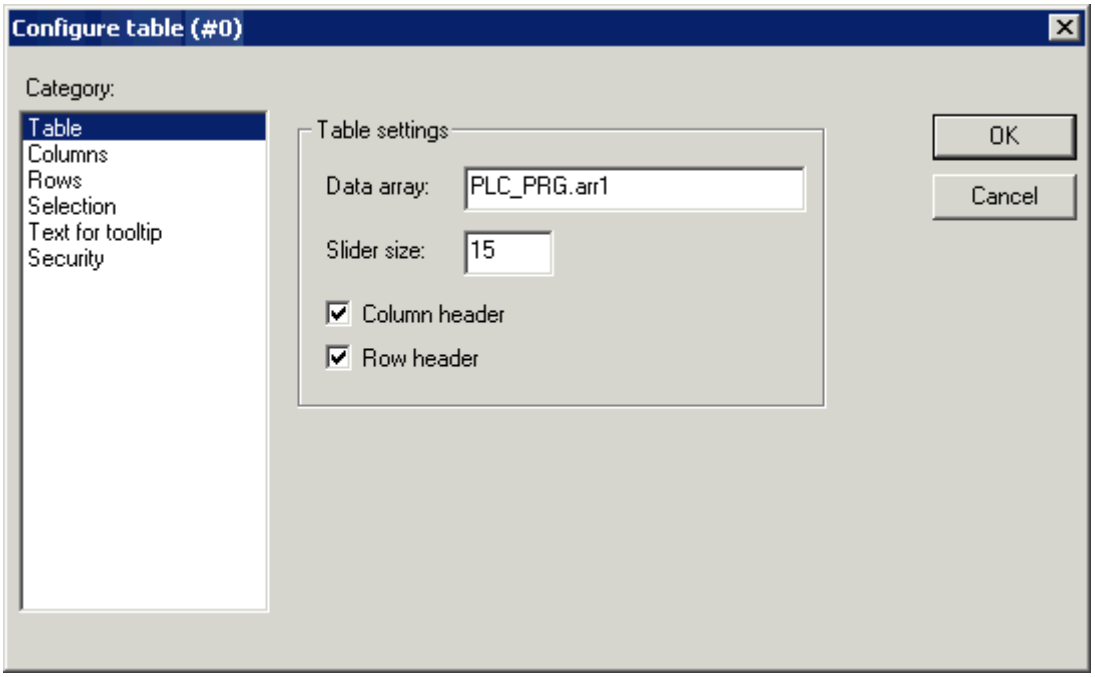
In the selection list in the lower part of the dialog you can define whether the bitmap should be inserted in the project (Embed) or whether just a link to an external bitmap-file (path as entered above in the 'Bitmap' field) should be created (Remember the link). It is reasonable to keep the bitmap file in the project directory, because then you can enter a relative path. Otherwise you would enter an absolute path and this might cause problems in case you want to transfer the project to another working environment.



1.4.3.5.20 Table

As soon as a table is inserted for the purpose of visualization of an array, the dialog Configure Table will be opened. Besides the categories 'Tooltip' and 'Security' which are also available for other visualization elements, the following categories will be available for configuring display and contents of the table.

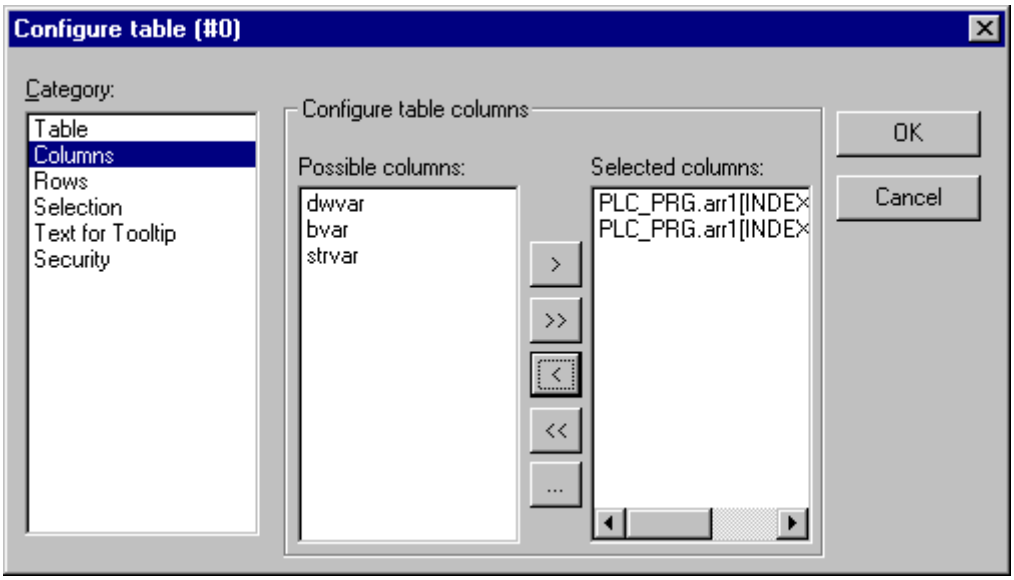
Category table:



Do the following table settings:

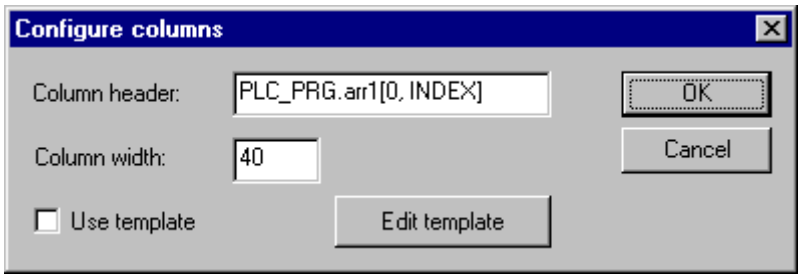
- Data array: Insert the name of an array which should be visualized in the table. It is recommended to use the input assistant (<F2>) resp. the Intellisense function.
- Slider size: Insert here the desired height of the slider which will appear at the bottom of the table element if the display of the array columns exceeds the element width.
- Column header, Line header: Activate these options if you want to get displayed the titles in the table. The line title reflects the array index (first column of the table), the column title can be defined in category 'Columns'.

Category columns:



Here you define the table elements. In the left window you get al list of all elements, which are handled in the array per index. In case of an array of a structure these would be the structure components.

Using the arrow button > you can transfer a selected component from the left window to the right window where you define the set of elements to be displayed in the table. Pressing button >> all elements will be transferred at a single blow. In the same manner you can remove elements from an already defined set (<, <<). In order to modify the default settings concerning the display of the table column for one of the elements, perform a double-click on the desired entry in the right part of the window, or press button '...' to open the dialog 'Configure columns':



Editing the column header and the column width:

Initially the edit field Column header will contain an automatically created title (e.g. "PLC_PRG.arr1[INDEX].iNo" in case of an array of structure for the column representing the structure component "iNo") which you can change. Further on the Column width (number of characters) can be set.

Editing configuration parameters for all elements of a column:

By default the table fields are displayed as simple rectangles and the entries are not editable. If you however activate button Edit template for the currently marked column, you can modify the parameters of the fields of this column, e.g. the line width, text input etc. The template affects all fields of the current column and can be edited via the known configuration dialog for a visualization element.

If you want to configure only one or several particular fields of the column, you can use the following placeholders determining the desired row and column: $ROWCONST$, $COLCONST$, INDEX. (INDEX has the same effect as $ROWCONST$).

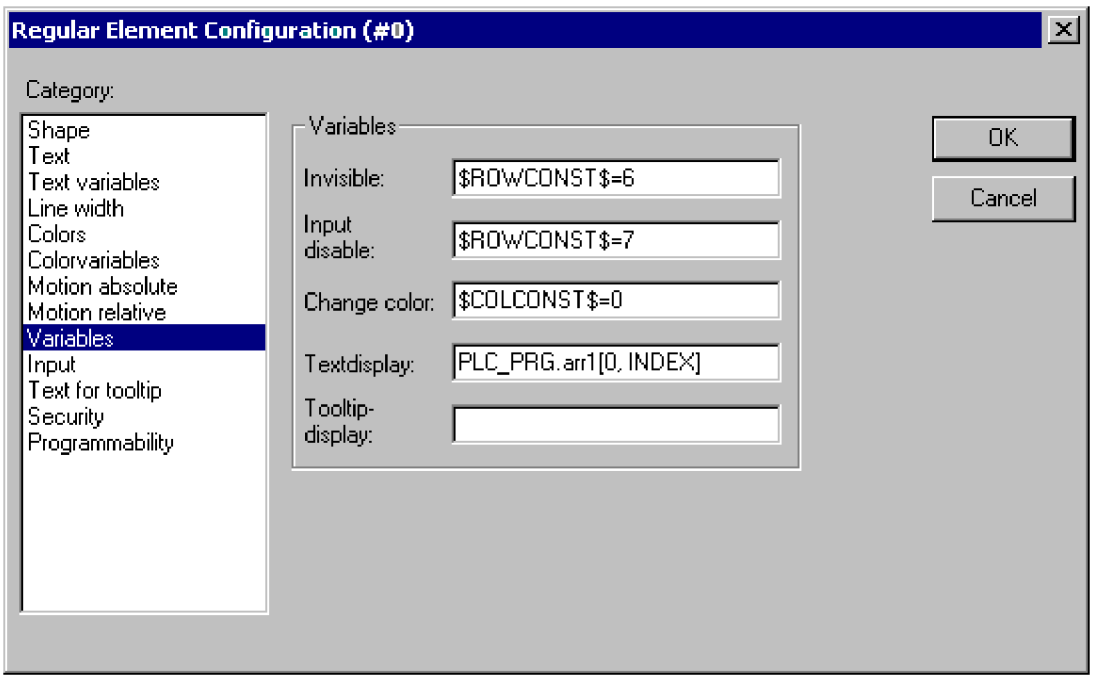
Examples for the use of placeholders in column-templates:

Example 1:

You visualize an array "arr1 [0..2] of BOOL" (table with 1 column) and you want, that in online mode by a mouse-click on a table cell the cell gets red-colored and the corresponding array element will be toggled and vice versa. To reach this activate 'Use template' in the configuration dialog for the column and define the template as follows: Category 'Input', Action 'Toggle variable': "PLC_PRG.arr1[INDEX]. Category 'Colors': Alarm color red. Category 'Variables', Action 'Change color': "PLC_PRG.arr1[INDEX].

Example 2:

The following template configuration has been created for a table column representing Index "0" of an array:



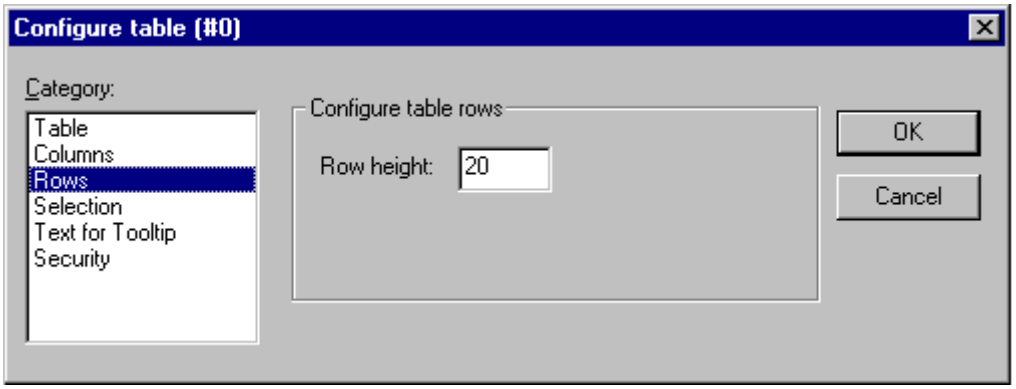
Meaning: For the concerned table column...

- the field in line 6 (line number, index) is invisible.
- in the field in line 7 no edit field can be opened.
- in all fields of column 0 the color will change to alarm color. The other fields of the column remain in base color.
- always entered automatically, but of course changeable, is the textdisplay configuration, which combined with the "%s" entry in category "Text" effects that the corresponding variable value will be displayed in the table field.

The placeholder entries can be connected with "AND" resp. "OR"; Example: "$ROWCONST$=1 OR $ROWCONST$=3" makes that both fields will get the respective setting.

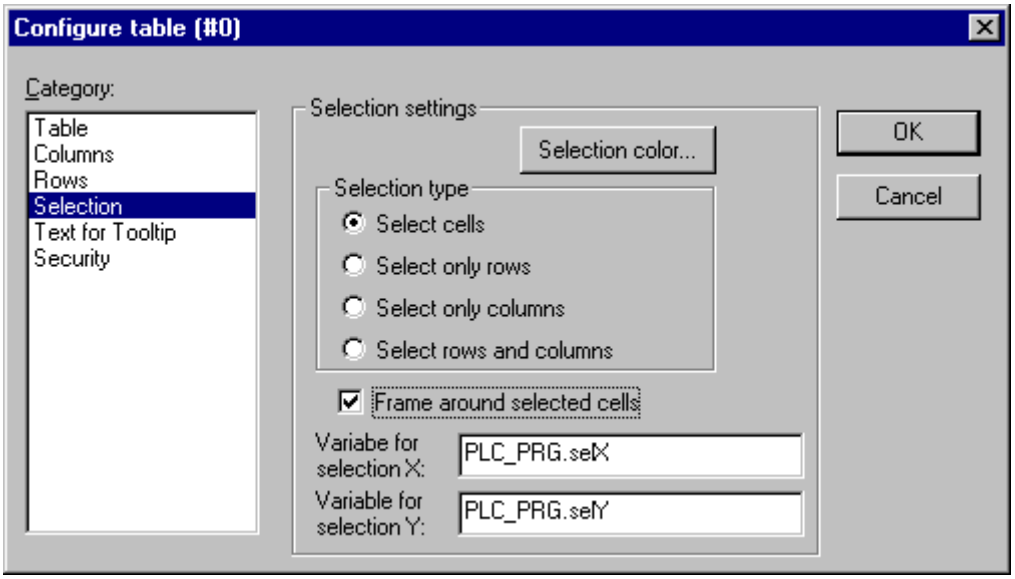
The use of the currently configured template can be activated or deactivated: Option Use template.

Category rows:



Row height: Insert the desired height in number of pixels.

Category selection:



Here you can set the following parameters concerning the selection behavior within the table:

- Selection color: Press this button to define a color for selected cells. You get the standard dialog for choosing a color or a color selection list.
- Selection type: Define which part of the table will be selected when you perform a mouse-click on one of the table fields in online mode:
  - Select single cells: Only the cell will be selected.
  - Select only rows: The whole line will be selected.
  - Select only columns: The whole column will be selected.
  - Select rows and columns: The whole column and line will be selected.
- Frame around selected cells: A selected cell gets surrounded by a frame.
- Variable for selection X, Variable for selection Y: Here you each can enter a project variable, which will indicate the X- resp. Y-Index of the selected table cell.

Example: Create a table element visualizing the array of a structure

```
TYPE strucTab :
STRUCT
iNo: INT; bDigi : BOOL; sText:STRING; byDummy: BYTE;
END_STRUCT END_TYPE
```

In PLC_PRG define the following array:

```
arr1:ARRAY [1..5] OF strucTab;
```

and the following variables:

```
selX:INT;
```

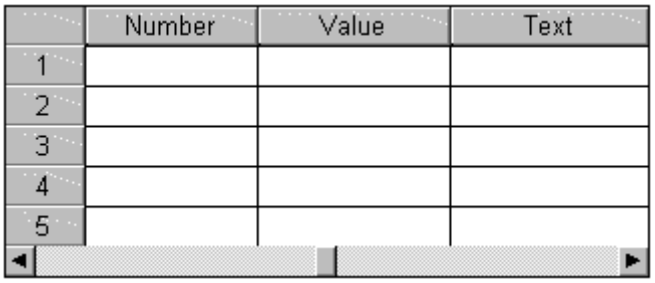
```
selY:INT;
```

Create a visualization object and insert a table element. Configure like follows:

Cat. Table: data array: "PLC_PRG.arr1"

Cat. Columns: Transfer the components iNo, bDigi, sText to the right window - In the right window perform a double-click on the first entry (PLC_PRG.arr1[INDEX].iNo) and in the dialog which will open, replace the default title by "Number". Confirm with OK and also define new column titles for the other two entries (e.g. "Value" and "Text").

Cat. 'Selection': Enter here at 'Variable Selection X': "PLC_PRG.selX" and at Variable Selection "Y: PLC_PRG.selY". Activate option 'Frame around selected cells'. Press button 'Selection color' and choose color 'yellow'. Close the configuration dialog with OK. The table element now should be displayed as shown in the following:



	Number	Value	Text
1			
2			
3			
4			
5			

At the left border the numbers of the array index, at the top the titles of the selected structure components. You can modify the column widths by placing the cursor on the separator between two columns and moving the mouse as soon as the cursor appears as a horizontal double-arrow. .

In online mode the current values of the array elements will be displayed in the table cells. As soon as you select a table cell by a mouse-click, it will get yellow-colored and surrounded by a frame. Example:

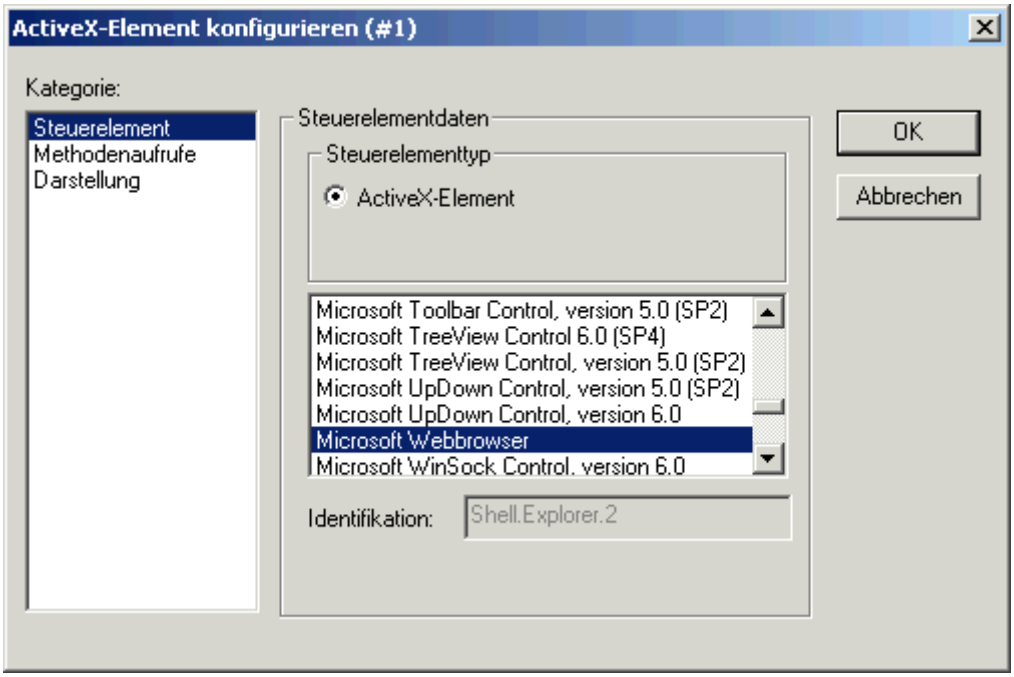
	Number	Value	Text
1	0	TRUE	text1
2	33	TRUE	text2
3	55	FALSE	abc
4	0	FALSE	
5	0	TRUE	

1.4.3.5.21 ActiveX element

The ActiveX element serves for displaying a passive ActiveX Control within a visualization. The element is usable on Windows32 based systems in CODESYS HMI.

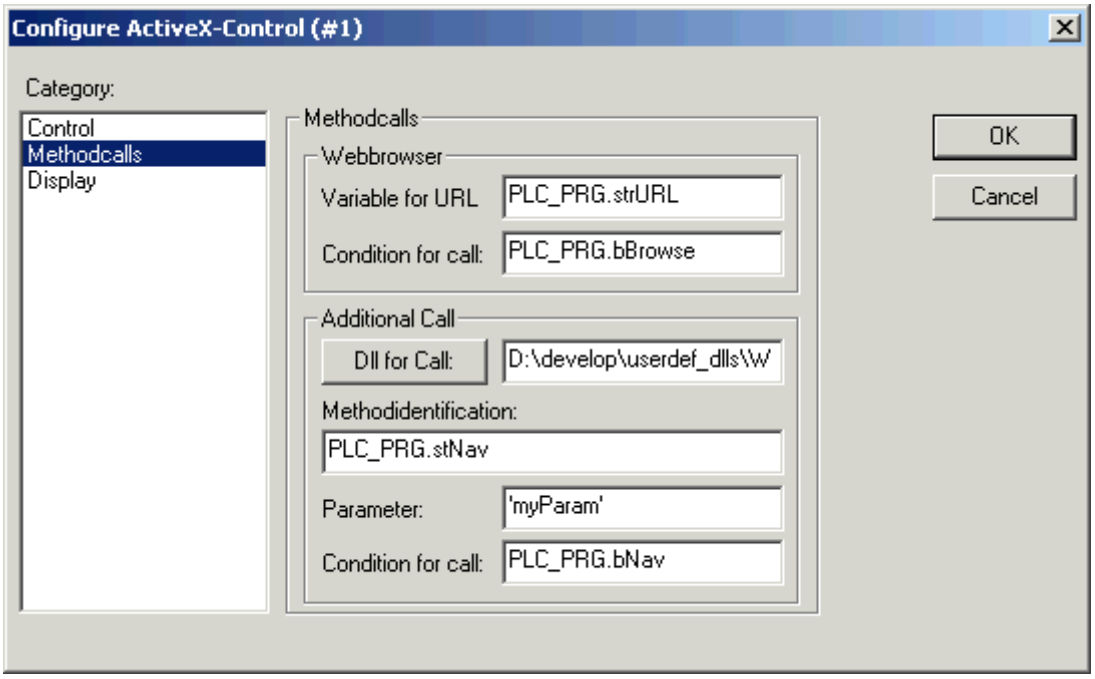
The configuration dialog is opened by a double-click on the Inserted element and offers three sub-dialogs, for selecting the control type, for defining method calls and for the configuration of the display:

Control:



In this dialog you can mark the desired ActiveX Control in the selection list offering all ActiveX Controls which are registered on your computer.

Methodcalls:



Here you configure the method calls for the chosen ActiveX Control:

Web browser:

These input fields only can be edited if you are configuring a control element which supports the 'IWebBrowser' interface (e.g. Internet Explorer or Mozilla Browser). In this case the method Navigate can be called (other method calls must be controlled via a user defined DLL, see below 'Additional Call').

Enter an URL as a parameter value in field Variable for URL (Input as a string between inverted commas) resp. a project variable of type STRING defining an URL. The browser will be called as soon as the variable entered in field Condition for call gets TRUE (rising edge). If no call condition is defined here, in the target visualization the browser will be called in each cycle of the visualization task!

Additional Call:

Via a user defined Windows-Dll you can define method calls for the ActiveX Control in order to control the behavior of the control at a call. For this purpose you must enter the DLL in the field at DLL for Call. If you press the button you get the File Open dialog to browse for a DLL. If the DLL is in the visualization files directory which is specified in the project options, just the path relative to this directory will be entered, otherwise the complete file path.

> *If the DLL is to be used on a runtime system with a target visualization, it must explicitly be copied there. When the Control is called in the target visualization, only the file name contained in the path will be regarded.*

The DLL is called as soon as the variable defined below in Condition for call gets TRUE (rising edge). If no condition is specified, in the target visualization it will be called in each cycle of the visualization task!

Regard the following when creating an appropriate DLL:

The DLL must export a method "ExecuteActiveXCall" with this function prototype :

```
void ExecuteActiveXCall(IUnknown* pUnk, char* pszId, char* pszParam,
char* pszReturnBuffer, int nReturnBufferSize, DWORD* pdwReturnFlag);
```

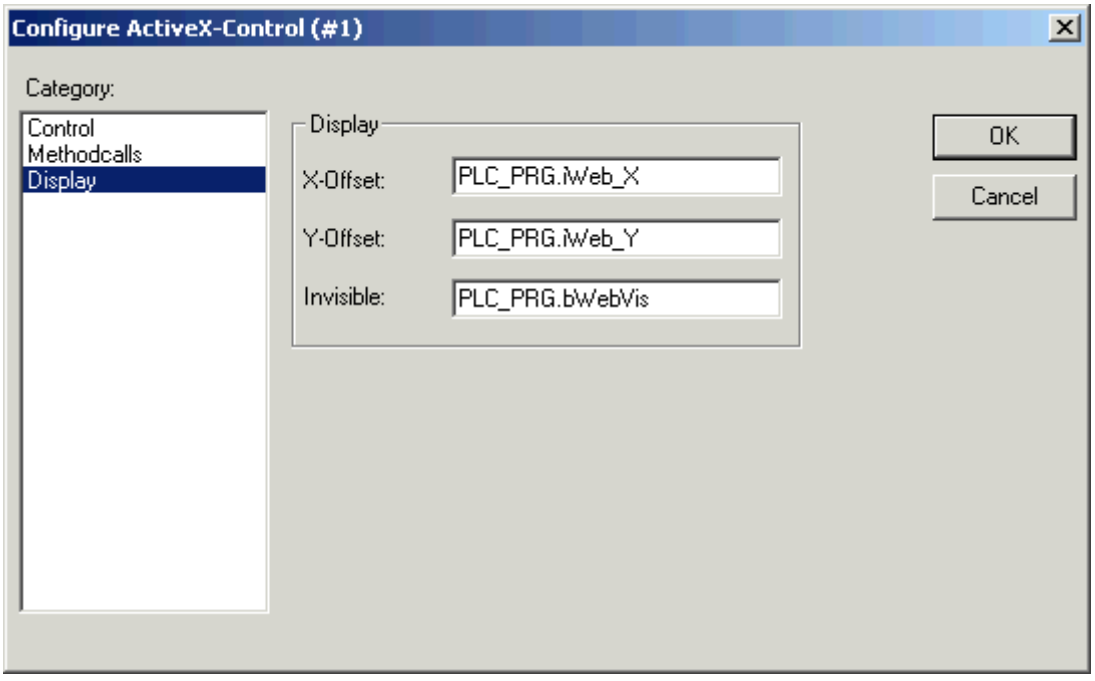
The function will be called with the following parameters which can be defined in the configuration dialog:

- pszId : string resp. string variable specified in field Methodidentification
- pszParam : value specified in field Parameter

The parameter pUnk allows a query of further Com(ActiveX-)interfaces. With these interfaces you can call any Method on your ActiveX-Control with any parameters packed in a string!

The parameters pszReturnBuffer, nReturnBufferSize and pdwReturnFlag currently are not used.

Display:



In this dialog you can specify the variables defining the position with X-Offset, Y-Offset (see ↳ *Chapter 1.4.3.5.12 “Motion absolute” on page 659*) and visibility of the control element Invisible (see ↳ *Chapter 1.4.1.7.3.1 “Overview” on page 438*).

Example of a DLL source file: This example DLL will only call the methods GoBack or GoForward of the control if it supports the 'IWebBrowser' interface.

The method is chosen by the parameter pszId.

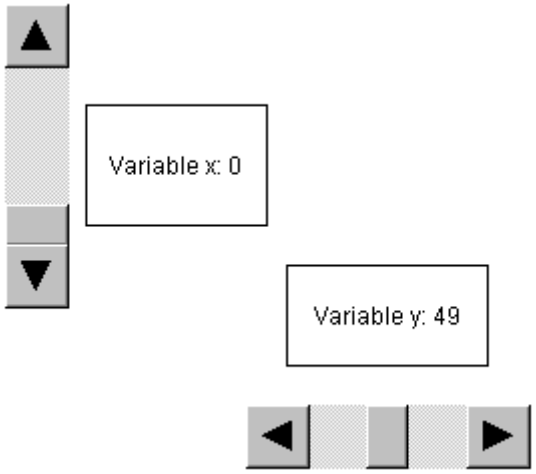
```
#include "stdafx.h"
#include <unknwn.h>
#include <exdisp.h>
BOOL APIENTRY DllMain( HANDLE hModule,
           DWORD  ul_reason_for_call,
            LPVOID lpReserved
)
{
    return TRUE;
}
extern "C" __declspec (dllexport) void ExecuteActiveXCall(IUnknown*
pUnk, char* pszId, char* pszParam,
  char* pszReturnBuffer, int nReturnBufferSize, DWORD* pdwReturnFlag)
{
if (strcmp(pszId, "IWebBrowser|GoBack") == 0)
{
IUnknown* pNewUnk;
IWebBrowser* pwb;
pUnk->QueryInterface(IID_IWebBrowser, (void**) &pNewUnk);
pwb = (IWebBrowser*) pNewUnk;
if (pwb)
{
pwb->GoBack();
pwb->Release();
}
}
else if (strcmp(pszId, "IWebBrowser|GoForward") == 0)
{
IUnknown* pNewUnk;
IWebBrowser* pwb;
pUnk->QueryInterface(IID_IWebBrowser, (void**) &pNewUnk);
pwb = (IWebBrowser*) pNewUnk;
if (pwb)
{
pwb->GoForward();
pwb->Release();
```

```
}
}
}
```

1.4.3.5.22 Scrollbar

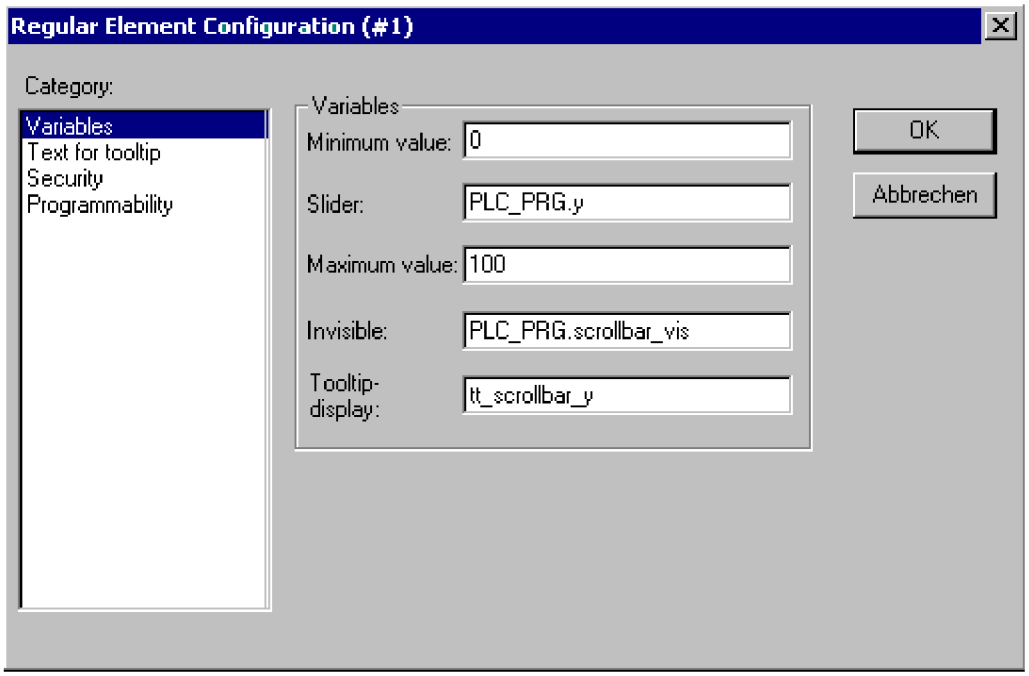
The scrollbar element can be used to modify the value of a variable (or vice versa). The slider position will correspond to the value of an assigned variable within a defined range of values.

The user can move the slider in online mode by clicking on the slider and then moving it by moving the mouse - or alternatively by clicking on one of the arrow symbols on the scrollbar, whereby each mouse-click will move the slider (and thus the variable value) by 1 in the respective direction: If the slider is moved to the right resp. up, the value will be increased, if the slider is moved to the left resp. down, the value will be decreased.



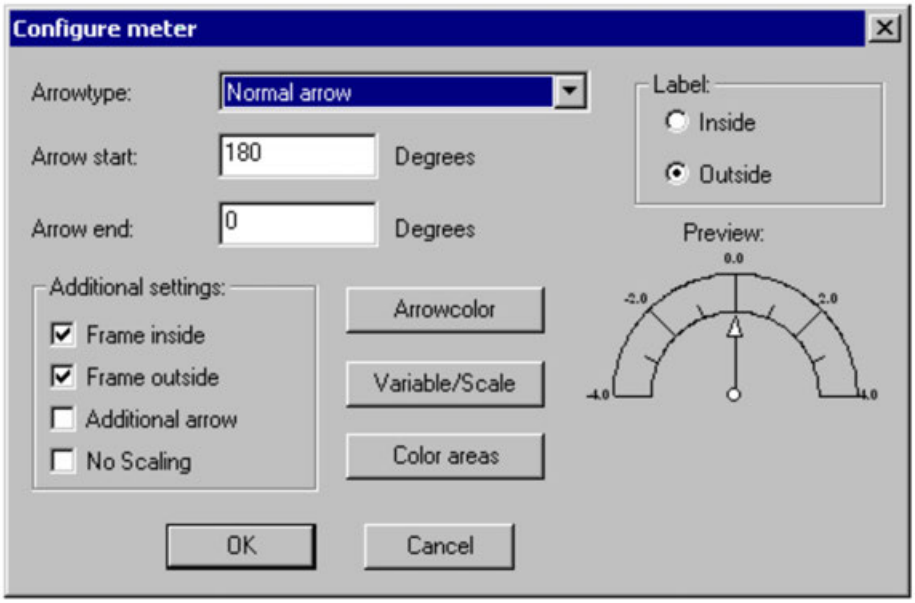
A vertical or horizontal display of the scrollbar can be reached by a respective shaping of the element (see 'Insert' 'Scrollbar').

The configuration dialog gets opened by a double-click on the element.



- Slider: Enter here the path of the project variable, the value of which should correspond to the slider position between the given Minimum value and Maximum value (e.g. "PLC_PRG.ivar"). So, when the slider gets moved in online mode, the value of the variable will change. Vice versa, if the variable gets modified by any other input the slider would get moved correspondingly.
- Minimum value, Maximum value: Boundaries for the range of values which can be displayed for the variable assigned to the slider. In horizontal scrollbars the minimum value corresponds to the left-most position of the slider, in vertically scrollbars to the bottom-most position. The values can be entered directly or via a project variable (e.g. "0", "200", "PLC_PRG.minvar").
- Invisible: The visibility of the element in online mode can be controlled dynamically, if here an appropriate boolean variable is entered, e.g. "PLC_PRG.bScrollbar_vis". A static definition is also possible ("TRUE", "FALSE" resp. "0", "1"). Default: visible.
- Tooltip-display: If "dynamic texts" should be used for the tooltip text, here the ID of the prefix-ID-combination must be entered, which uniquely references the desired text in a specific language file (xml-format). You can enter the ID directly (string, e.g. "tt_scrollbar1") or via a project variable of string format which gives the ID (e.g. "PLC_PRG.tt_ID"). Concerning this see the help pages on Tooltip and generally on Language Switching.

1.4.3.5.23 Meter



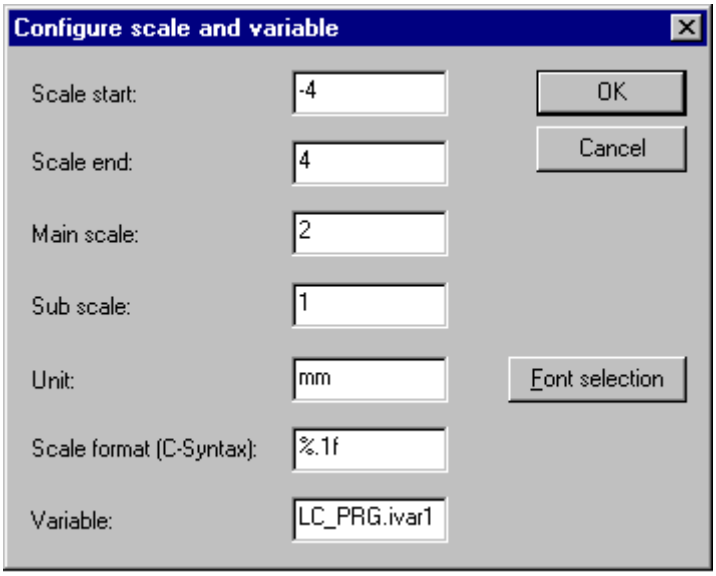
This dialog will open automatically as soon as you insert a Meter into a visualization object. A Preview is part of the dialog, immediately showing how the element will look as a result of the currently set parameters:

- Arrowtype: Define the type of the arrow which will point at the current value on the Meter. Possible types: Normal arrow, Thin arrow, Wide arrow, Thin needle.
- Arrow start, Arrow end: Here you define the start and the end positions of the scale on a virtual circular arc in ° Degrees (angle). (Example: a Start angle of 180° and an End angle of 0° will define a upturned semicircle).
- Arrow color: This button opens the standard dialog or a target-specific color selection list for choosing a color. Define the color of the pointer.
- Variable/Scale: This button opens the dialog *"Dialog 'Configure scale and variable'" on page 681*.
- Color areas: This button opens the dialog *"Dialog 'Configure color areas'" on page 681*: Here you can define a separate color for each partition of the scale.

Additional settings:

- Frame inside, Frame outside: If one or both of these options is/are activated, an inner or outer frame will be added to the scale arc.
- Additional arrow: In addition to the main pointer a little arrow will indicate the current value directly on the scale.
- No Scaling: If this option is activated, the inserted element cannot get resized.

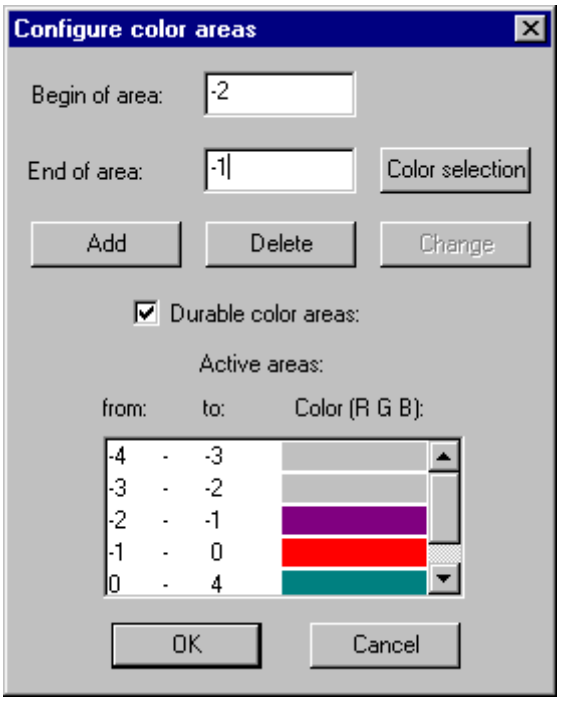
Dialog 'Configure scale and variable'



- Scale start, Scale end: lowest and highest value on the scale, e.g. "-4" and "4".
- Main scale: Define which intervals on the scale should be marked "with all", that means which should get a scale pitch and a label. If you insert e.g. "2", each second integer value will be indicated.
- Sub scale: In addtion to the main scale (Label + long pitch lines) here you can define a sub-scale which will be displayed as short pitch lines without any labels.
- Unit: Define here the scale unit, e.g. "cm" or "sec". The unit is indicated by a label at the origin of the pointer.
- Scale format (C-Syntax): According to the C-syntax you can define the display format of the scale labels; see the description concerning Category 'Text'. Example: If you insert "%1.1f" the scale values will be indicated by a floating point number with one decimal place before and one after the comma (e.g. "12.0")
- Variable: Here you can define a variable which is assigned to the pointer position. (e.g. "PLC_PRG.posvar")
- Font selection: This button will open the standard dialog for defining the font used in the Meter element (the selection list can be target-specific).

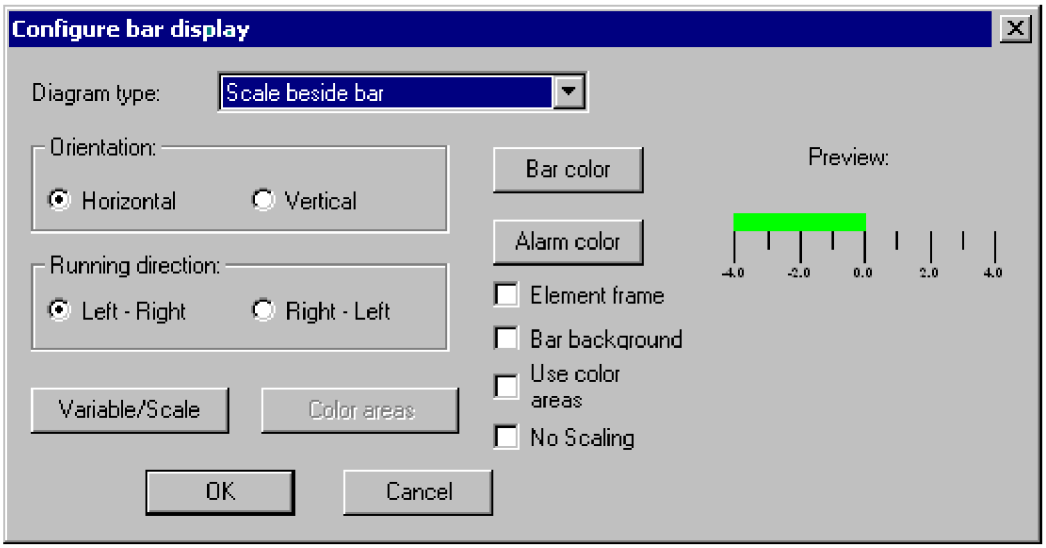
Dialog 'Configure color areas'

Dialog for the configuration of color areas for a Meter:



- Begin of area, End of area: Insert here the start and end values of the scale partition which should get the color defined in the following:
- Color selection: This button opens the standard dialog for choosing a color or a target-specific selection list. Confirm your selection with OK, which will close the dialog, and press button Add, whereupon the color and the assigned partition of the scale will be added to the window 'Active areas'. In order to remove an already defined area, select the entry and press Delete.
  If the option Durable color areas is activated, the defined color ranges will be displayed permanently, otherwise in online mode just that partition of the scale will be colored which contains the current value of the respective value.
- Label: Depending on which of the options is activated (inside or outside), the scale labels are placed at the inside or the outside of the circular arc of the scale.

1.4.3.5.24 Bar display

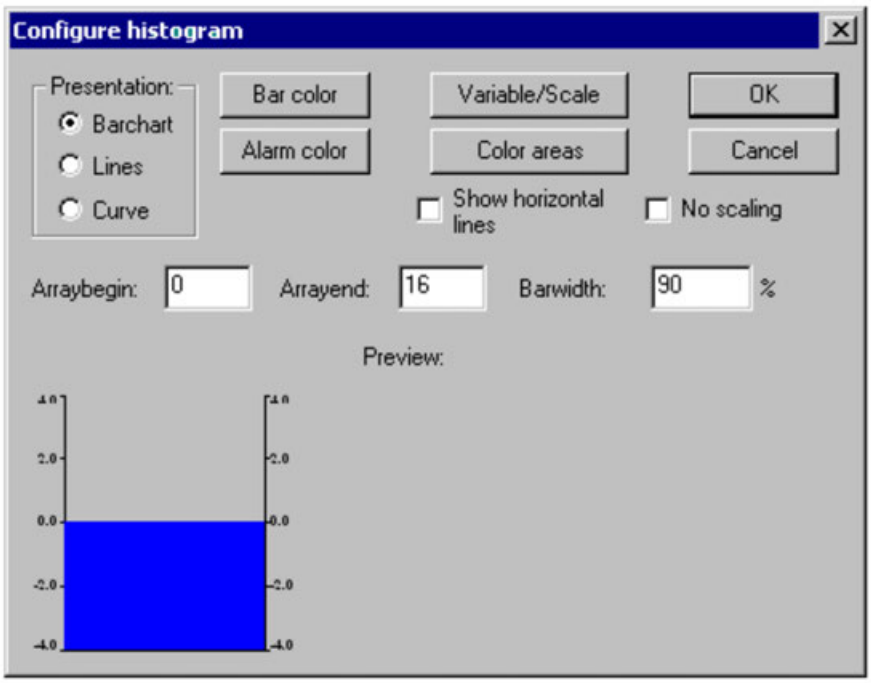


This dialog will be opened as soon as you insert a Bar Display element into a visualization object. A Preview is part of the dialog, immediately showing how the element will look as a result of the currently set parameters:

- Diagram type: Choose one of the options: 'Scale beside bar', 'Scale inside bar' and 'Bar inside scale'.
- Orientation: Define one of the options: Horizontal or Vertical bar.
- Running direction: Choose whether the bar should be elongated corresponding to a growing value of the assigned variable in Left "“ Right or in Right "“ Left direction.
- Bar color: This button opens the standard dialog for choosing a color. Define a color for the bar in normal state (no alarm). If option 'Use color areas' (see below) is activated, no entries are possible.
- Alarm color: This button opens the dialog 'Configure alarm', where you define at which value the bar will be displayed in alarm color and which is the alarm color: Insert the desired limit value in the edit field and activate one of the Conditions greater than or lower than, in order to define whether values higher or lower than the limit value should set off an alarm. Press button *“Alarm color”* to open the standard dialog or a target-specific color selection list for choosing the alarm color. Close both dialogs with OK in order to confirm the settings and to return to the main dialog for configuring the bar display. If the option 'Use color ranges' (see below) is activated, no entries are possible.
- Variable/Scale: This button opens the dialog 'Configure scale and variable', which corresponds to that used for the Meter element ↬ *Chapter 1.4.3.5.23 “Meter” on page 680*.
- Element frame: If this option is activated a frame will enclose the bar display.
- Bar background: If this option is activated, the whole display range will be indicated by a black bar in the background of the current values' bar, otherwise only the current values' bar will be displayed.
- Use color areas: If this option is activated, any settings defined in the dialogs for 'Bar color' and 'Alarm color' (see above) will not be valid. In this case the color area definitions will be used, which have been made in the dialog 'Configure color areas'. This dialog can be opened by pressing button 'Color areas' (see below)
- Color areas: This button opens the dialog 'Configure color areas' where you can define a separate color for each partition of the scale. These definitions will only be valid if the option 'Use color areas' (see above) is activated. Use the dialog as described for the Meter element ↬ *Chapter 1.4.3.5.23 “Meter” on page 680*.
- No Scaling: If this option is activated, the inserted element cannot get resized.

1.4.3.5.25 Histogram

A histogram element can be used to visualize an array. The values of the array elements will be represented by bars or lines side by side, indicating the current values of the element by their height.

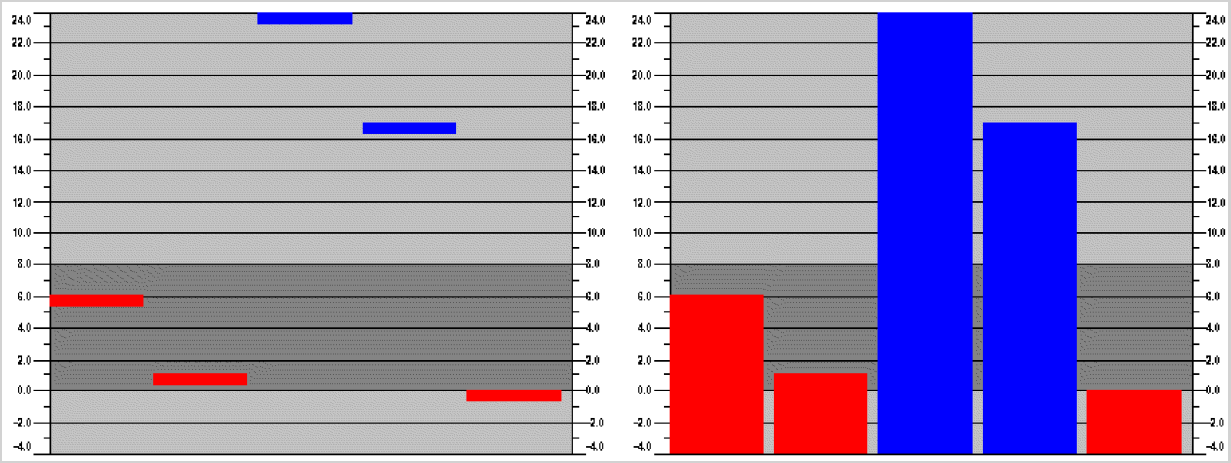


The configuration dialog will be opened as soon as you insert a histogram element into a visualization object. A preview is part of the dialog, immediately showing how the element will look as a result of the currently set parameters:

- Presentation: Activate one of the options Barchart or Lines.
- Show horizontal lines: If this option is activated, horizontal lines spanning the diagram will additionally display the scale gradation.
- No Scaling: If this option is activated, the inserted element cannot get resized.
- Alarm color: This button opens the dialog 'Configure alarm', where you define at which value the bar will be displayed in alarm color and which is the alarm color: Insert the desired threshold value in the edit field and activate one of the Conditions greater than or less than, in order to define whether values higher or lower than the limit value should set off an alarm. Press button *“Alarm color”* to open the standard dialog or a target-specific color selection list for choosing the alarm color. Close both dialogs with OK in order to confirm the settings and to return to the main dialog for configuring the histogram.
- Variable/Scale: This button opens the dialog 'Configure scale and variable', which can be filled like described for the Meter element ↳ *Chapter 1.4.3.5.23 “Meter” on page 680*.
- Color areas: This button opens the dialog 'Configure color areas': Here you can define a separate color for each partition of the scale. See the description of the Meter where the same dialog is available ↳ *Chapter 1.4.3.5.23 “Meter” on page 680*.
- Bar color: This button opens the standard dialog for choosing a color. Define a color for the bar in normal state (no alarm).

  Define which range of the array should be displayed:
  - Arraybegin: First array element to be displayed (Index).
  - Arrayend: Last array element to be displayed (Index).
- Barwidth: Define the width of the bars in percent by the total width available for one bar.

Example: See in the following picture an example of the online display (bars resp. lines) of a histogram which represents an array arr1 [0..4] of INT. The arraybegin was set to "0", the arrayend to "4", the scale start to "-4", the scale end to "24", the main graduation was set to "2", the sub-graduation to "1" and the scale range 0 "“ 8 has got assigned another color (dark grey) than the rest of the scale range. Furtheron the bars should be displayed alarm-colored (blue) as soon as the value of the corresponding array element exceeds "8". You see the array elements arr1[2] and arr1[3] currently being in alarm state:

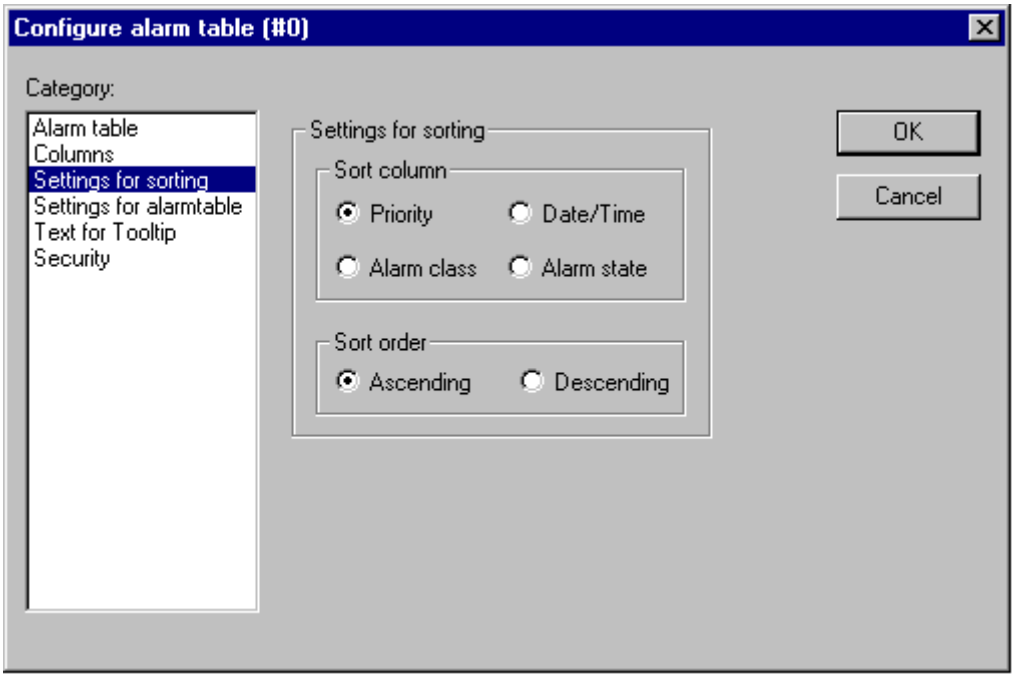


1.4.3.5.26 Alarm table

The element 'Alarm table' is used to visualize alarms, which must be defined before, see 'Alarm configuration, Overview' ↳ *Chapter 1.4.1.4.2.1 “Overview” on page 363*.

As soon as the element gets inserted in the visualization object, the dialog 'Configure alarm table'. Besides the known categories for configuration of tooltip and security the following settings concerning display and selection in the table can be made.

Category alarm table

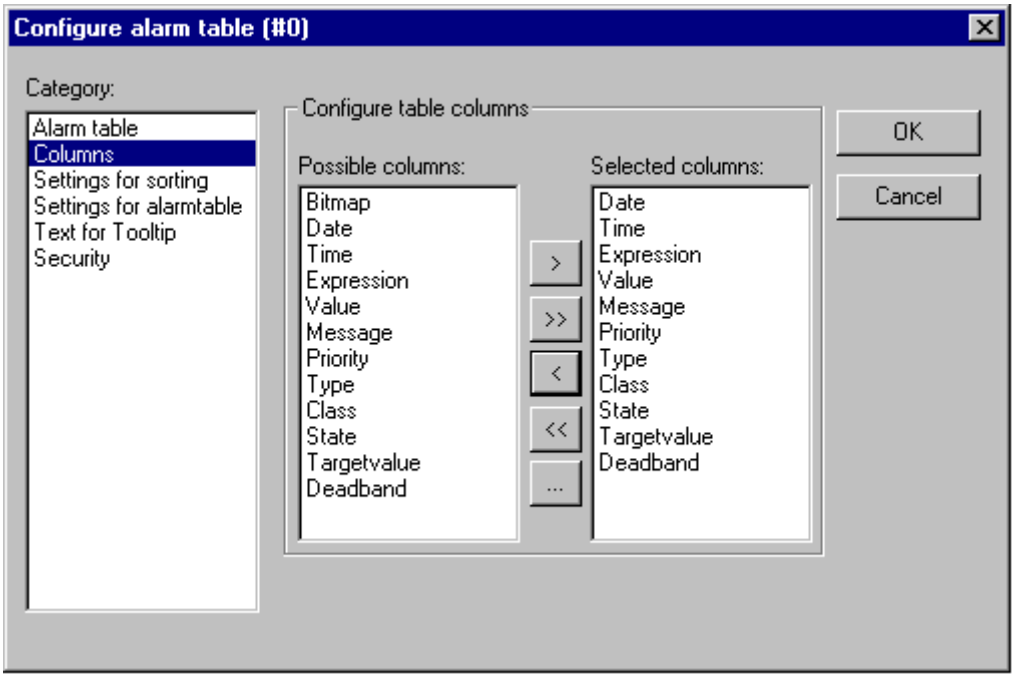


Define what you want to get displayed in the alarm table:

- Change alarm group: Press this button to get the selection tree of the alarm configuration, which offers all alarm groups currently defined. Choose the desired group (which even may contain just one alarm).

  Note: The name of the alarm group, as it is displayed here (e.g. "System/Alarmgroup1") can be used as PREFIX in a XML file for dynamic texts in order to get dynamic language switching for the message texts in the alarm table; the associated ID in this must be the number of that line in the Alarm group configuration table, that defines the respective message text.
- Priority: Define the priority for which you want to get displayed all alarms. Permissible range: 0 to 255.
- Alarm classes: Mark a class which you want to get displayed and press button Add to add the class to the list in the window 'Alarm classes'. Do this for all required classes. In order to remove a marked entry from the alarm classes window press button Delete.
- Activate options Column heading resp. Row heading, if the headings should be displayed in the alarm table.

Category settings for sorting

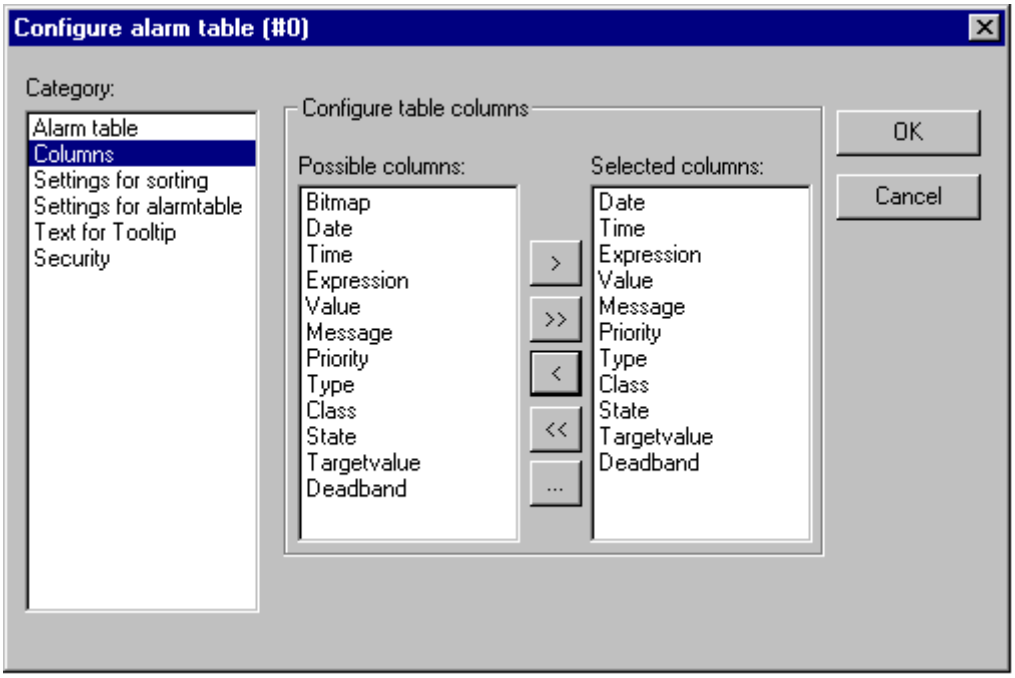


Define here according to which criteria the alarm table should be sorted:

- Sort column: Sorting according to Priority, Alarm class, Date/Time or Alarm state
- Sort order: Ascending or Descending;
  Example: Ascending according to priority means that the table will start with alarms of priority 0 (if available), followed by higher numbered priorities.

> *When using in a target visualization this settings are not regarded in the display of the alarm history. There the sorting always is descending according to the date.*

Category columns

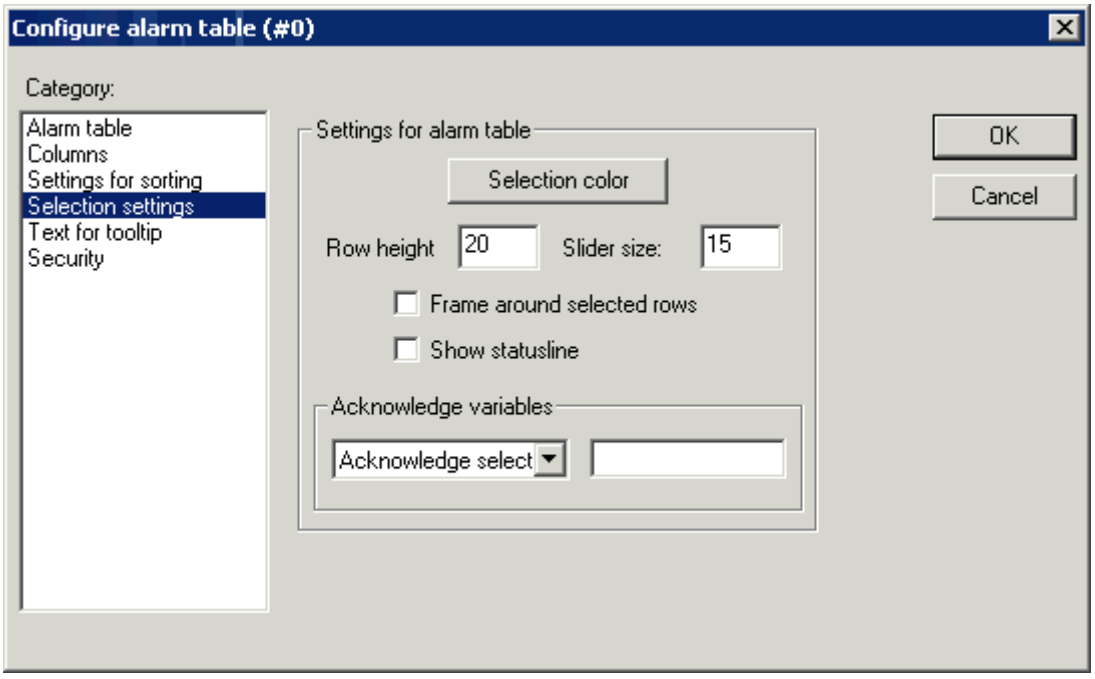


Define here, which of the columns (alarm parameters) should be displayed in the alarm table: The parameters are defined -- except for date and time (alarm is coming) and alarm state in the configuration of the alarm groups: Bitmap, Date, Time, Expression, Value, Message, Priority, Type, Class, State, Target value (for alarm types DEV+ and DEV-), Deadband.

Using the buttons ">", ">>", you can take single resp. all parameters from the left to the right window. The selection defined in the right window will be displayed in the alarm table. Using the buttons "<" resp. "<<" entries can be removed from the selection.

For each column you can open the dialog 'Configure columns' by a double-click on the entry in the right window. In this dialog Column header and Column width can be defined.

Category selection settings for alarm table



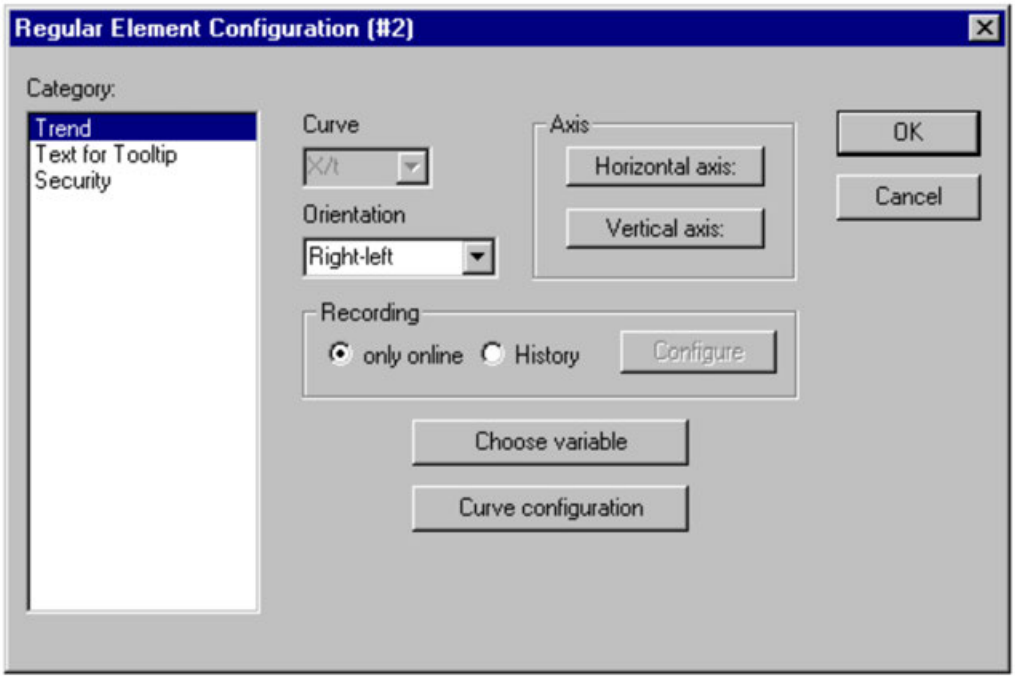
Define here some settings for the display for the chosen table fields:

- Selection color: This button opens the standard dialog or a target-specific selection list for choosing a color. Define the color in which selected fields should be displayed.
- Row height: Height of the table rows in Pixel.
- Slider size: Height of the slider (Pixel) at the bottom of the table.
- Frame around selected rows: If this option is activated, selected table rows will get a frame.
- Show statusline: If this option is activated, below the alarm table a status bar will be displayed providing the following buttons for the operation in online mode:
  - Acknowledge: All alarm entries marked in the alarm table get acknowledged.
  - Acknowledge all: All alarm entries listed in the alarm table get acknowledged.
  - History: If this button is pressed, instead of the current status of the alarms the table will show a complete list of all events which have occurred up to now (all transitions between any alarm stati). In this list no acknowledgement is possible ! Any new events will be added currently.

    If you have defined a record file, also there you will find this history for all alarm classes, for which the action 'Save' has been activated.
  - Start: cancels Stop (see below)
  - Stop: the current update of the list with newly occurring events will be stopped until it is restarted by pressing button 'Start'.
- Acknowledge variables: This option is only available as long as you have not chosen option 'Show statusline' (see above). If it is activated, the functions described above for the status string buttons can get controlled by variables. In order to define these variables, choose a function from the selection list and enter a project variable in the assigned edit field. Thus for example the acknowledgement of all alarms in online mode can be done by a rising edge of the assigned variable.

The web visualization might be configured in a way, that in online mode a tooltip will display the full string of a text entry, which is only partially visible in the alarm table.

1.4.3.5.27 Trend



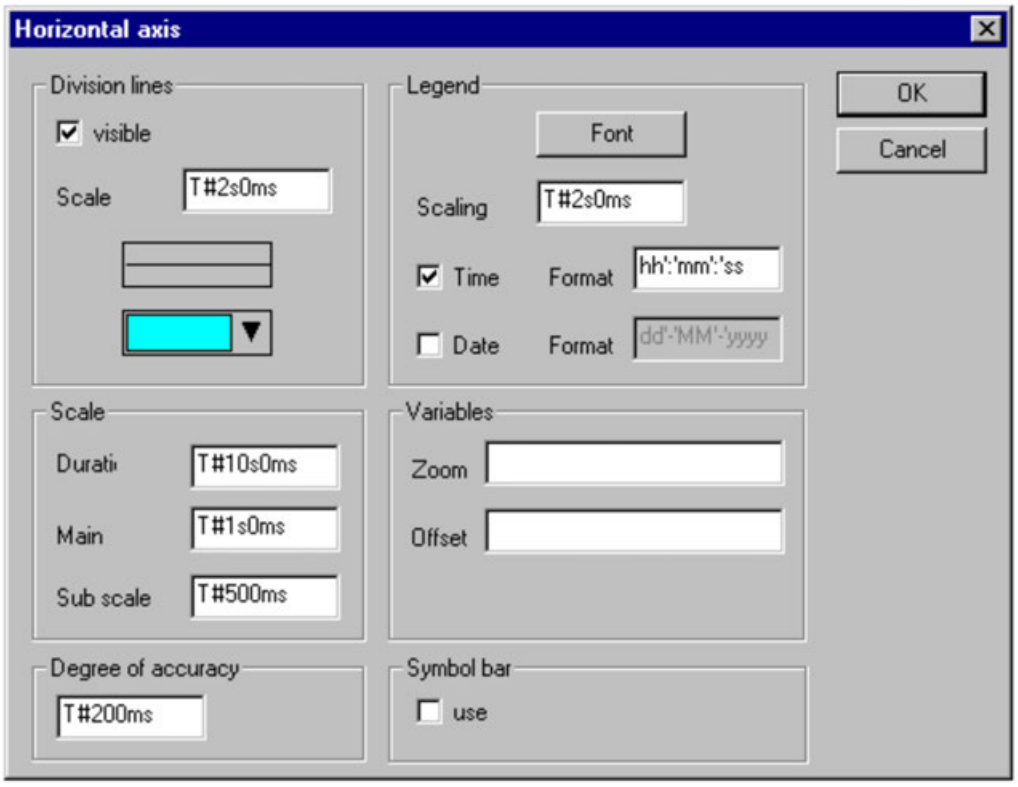
The Trend element can be used to log the time dependent behavior of variable values in the online mode. The online presentation is done in a diagram, in case of logging to a text file each of the values is written to a separate line.

In the dialog for configuring visualization elements in category 'Trend' you can do the following settings :

- Curve: X/t, horizontal axis = time axis, vertical axis = scale of values
- Orientation: Left-right or Right.-left: The latest value will be displayed on the left/right side;
- Axis:
  - Horizontal Axis *“Horizontal axis” on page 689*
  - Vertical Axis *“Vertical axis” on page 690*
- Recording: Define here whether the trend should be recorded 'only online', i.e. the time dependent behavior of the variable values will be displayed using the chosen range of the scale, or whether the record should be saved to history file, which can be configured after pressing the button History. The dialog corresponds to that which is used for the configuration of the alarm log file.

  In the log file for each time of measurement a separate line is written which contains the name and the values of all regarded variables. Each line starts with a unique identifier in DWORD format, which is built from the date of measuring.
- Choose variable *“Choose variable” on page 690*
- Curve configuration *“Curve configuration” on page 691*

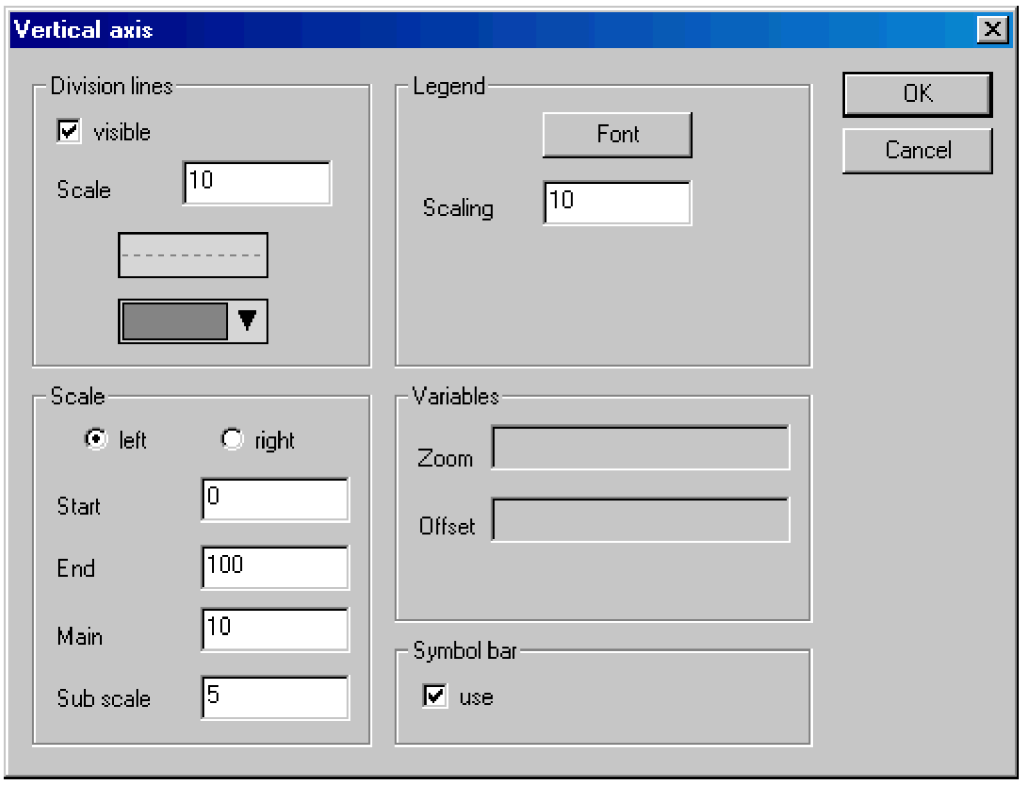
Horizontal axis



- Division lines: Activate option 'visible', if vertical division lines should be displayed which are elongating the scale marks. In this case define the 'Scale': The given number defines the interval between the division lines on the horizontal axis. Type (normal ___, dashed _ _ _, dotted ....., dashdotted _ . _ .) and color of the lines can be defined in dialogs which will open when you perform a mouse-click on the corresponding rectangle showing the currently set line type resp. color.
- Scale: The shown range of the scale is determined by the entry for Duration. If here e.g. " T#20s0ms" is defined, the scale will display a period of 20 seconds. The Main division and the Sub scale division, which will be displayed by the means of long and short marks are to be defined according to the same syntax.
- Degree of accuracy: Define here (in the standard format for dates, e.g. T#5ms) the interval for displaying the current values of the variables.

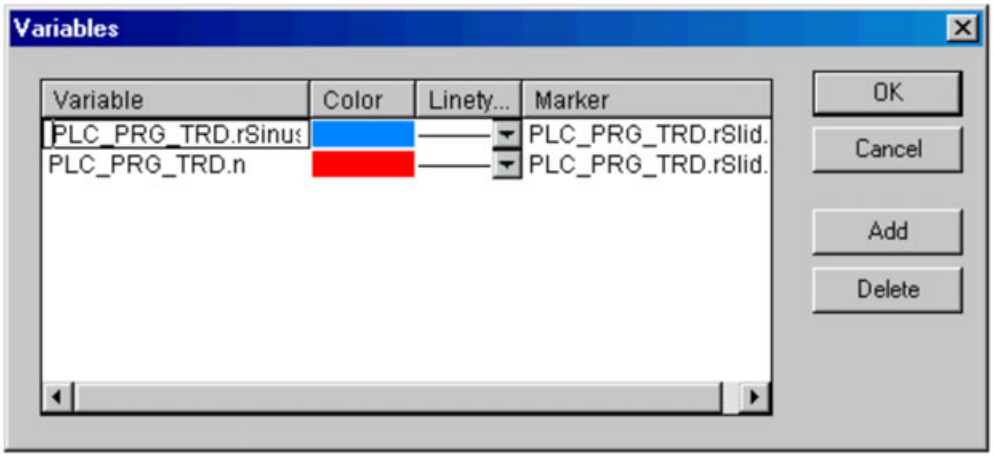
- Legend: Here you define the display of the legend. Via button Font the standard or a target-specific restricted dialog for setting the font will be opened. At Scaling define the distances between the particular letterings on the scale (e.g. T#4ms, if the scale markings should get a lettering each 4 milliseconds). The lettering will contain the Time and/or Date, depending on which options are activated. The desired format each can be defined in the field 'Format'. Regard that you can either set the 12-hour format ("hh") or the 24-hour format ("HH").
- Variables: Here you can define project variables, which contain the zoom values resp. offset values for the horizontal scale. For example the offset of the display range of the horizontal axis will be set to "10" as soon as the variable assigned here gets value 10.
- Symbol bar: If option use is activated, at the bottom of the element a horizontal symbol bar will be added, providing buttons for scrolling and zooming in online mode. The simple arrow buttons will move the displayed range along the time axis step by step, the double arrow buttons will shift it to the end resp. start of the record. The zoom buttons allow a zooming of the horizontal scale step by step. To get a possibility to restore the original settings concerning zoom and offset, define the vertical symbol bar to get the 'home' symbol.

Vertical axis



- Dialog for configuration of the vertical axis in the trend element
- Division lines: corresponding to the horizontal axis (see above)
- Scale: Define whether the scale should be displayed at the left or right border of the trend diagram. Choose the Start value (lower end) and End value (upper end) of the scale as well as the Main and Sub scale divisions (longer and shorter markings will be displayed in the here defined distances).
- Legend: Font and divisions; see above, horizontal axis
- Variables: see above, horizontal axis
- Symbol bar: see above, horizontal axis, additionally there is a "home" button for restoring the standard settings concerning zooming and offset of the axes.

Choose variable Choose variable: Press this button to get the dialog Variables, where you can configure the variables for which the trend record should be done and how they should be displayed:



Enter a project variable in column Variable (mouse-click on the field will open an edit frame). It is recommended to use the input assistant <F2> or Intellisense function.

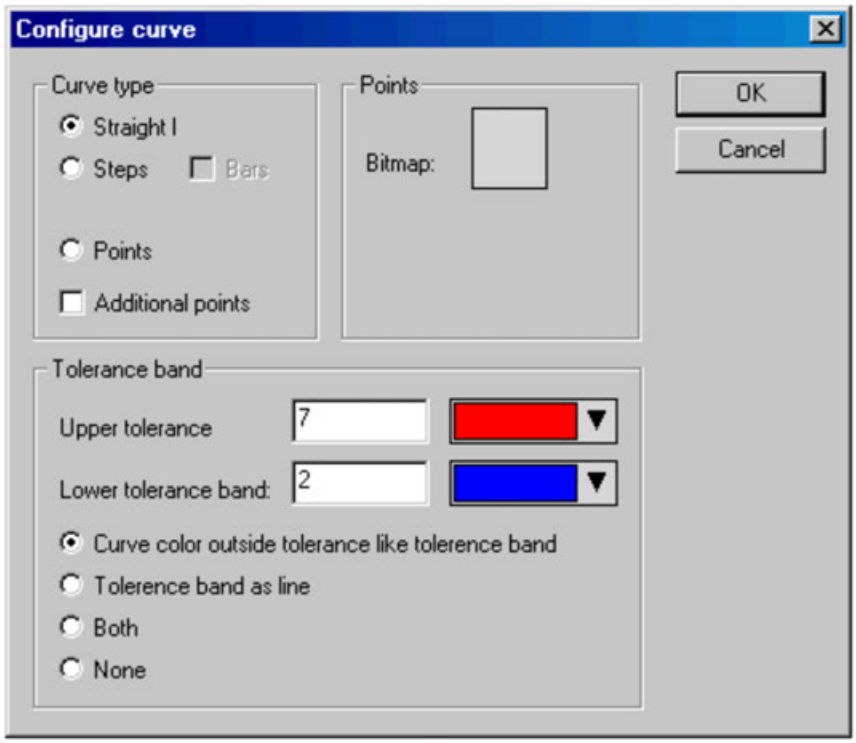
Color and Line type for the display of the variable in the record you can define by a mouse-click on the corresponding field in column Color (the standard dialog or a target-specific color selection list) resp. by selecting a line type in the corresponding field of column Line type (normal ___, dashed _ _ _, dotted ....., dashdotted _ . _ .).

In column Marker you can define a variable, which will provide the currently recorded value when you use the marker function in online mode. The marker will be displayed as a little grey triangle in the upper left corner of the diagram. If you click on the triangle and keep the mouse-button pressed, you can shift a vertical marker line along the horizontal time axis. The variable defined as 'marker' then will read the corresponding value from the record curve of the associated project variable.

Do the settings for all variables you want to record. Via button Add a further line will be added at the end of the list. A line can be deleted by button Delete.

Curve configuration

Curve configuration: This button opens the dialog Curve configuration. Here some settings concerning the trend curves can be done:



Curve type: Select one of the options Straight line, Steps or Points. For the first two types the display of Additional points can be defined. For displaying a point a bitmap can be defined, otherwise a filled rectangle (same color as curve) will be used as point symbol. Press the rectangle next to Bitmap to get the standard dialog for selecting a bitmap file. Via Delete the currently set bitmap can be removed from the configuration.

Tolerance band: You can define an upper and lower limit value on the vertical axis to be displayed as a tolerance band. For each band a color (Press the color rectangle to get the default resp. target-specific selection dialog) can be defined. If the bands should be displayed in online mode, activate option Tolerance band as line. If you want the curve to get displayed in the color defined for the respective band as soon as exceeding the tolerance value, activate Curve color outside tolerance like tolerance band. Activate Both or None if you want to activate both or none of the above described display options at a time.

Example: Display of a trend element in online mode

Declaration in program PLC_PRG:

```
VAR
n: INT;
rSinus:REAL;
rValue:REAL;
rSlider1:REAL; (*for marker function*)
rSlider2:REAL; (*for marker function*)
END_VAR
```

Programmteil von PLC_PRG:

```
n:=n+1;
rValue := rValue + 0.01;
rSinus:=SIN(rValue)*50 + 50;
IF n>100 THEN
n:=0;
END_IF
```

Configuration of a trend element in a visualization:

Orientation Left-Right, History activated

Horizontal axis: Division lines: T#2s, Duration: T#10s, Main: T#1s, Sub scale: T#500ms, Degree if accuracy: T#200ms, Legend: Time Format ('HH':'mm':'ss'), Scaling T#2s. Symbol bar activated.

Vertical axis: Division lines visible, Scale: 10, dotted, grey; Scale left, Start: 0, End: 100, Main: 10, Sub scale: 5; Legend: 10; Symbol bar activated.

Variables:

1. Variable PLC_PRG.rsinus, blue line, Marker: PLC_PRG_TRD.rSlider1;

2. Variable PLC_PRG.n, red line, Marker: PLC_PRG_TRD.rSlider2

Curve configuration: Straight line, no tolerance band

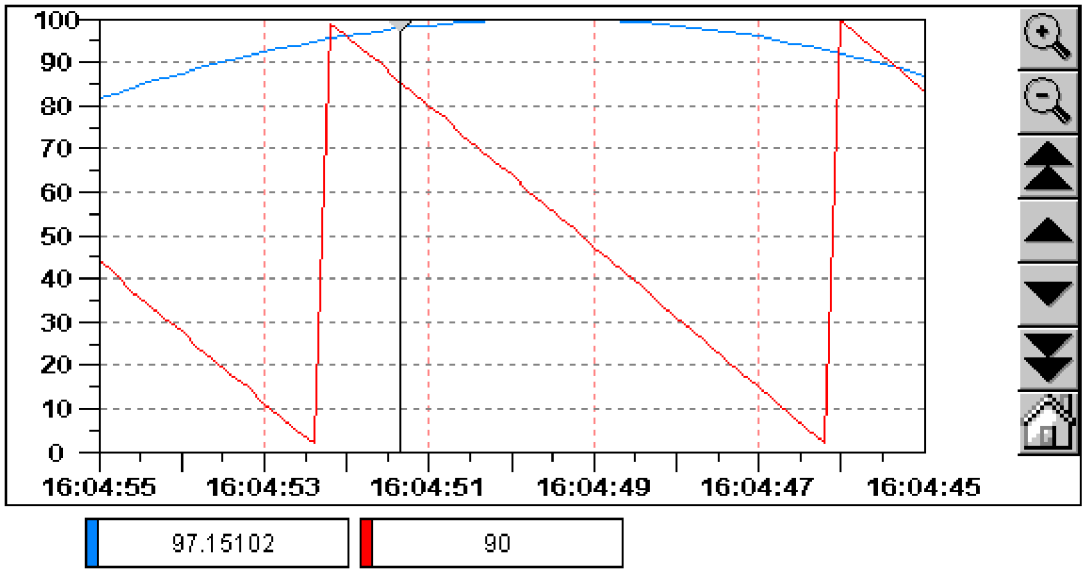
Configuration of two display fields for the current record values provided by the marker variables:

Rectangle element 1: Category Text: insert "%s" in the Content field; Category Variables: insert in field Textdisplay: PLC_PRG.rSlider1

Rectangle element 2: Category Text: insert "%s" in the Content field; Category Variables: insert in field Textdisplay: PLC_PRG.rSlider2

(additionally insert a rectangle element at the left border of the rectangle elements 1 and 2, showing the curve color of the corresponding record variable)

Result in online mode after login and start of the program:



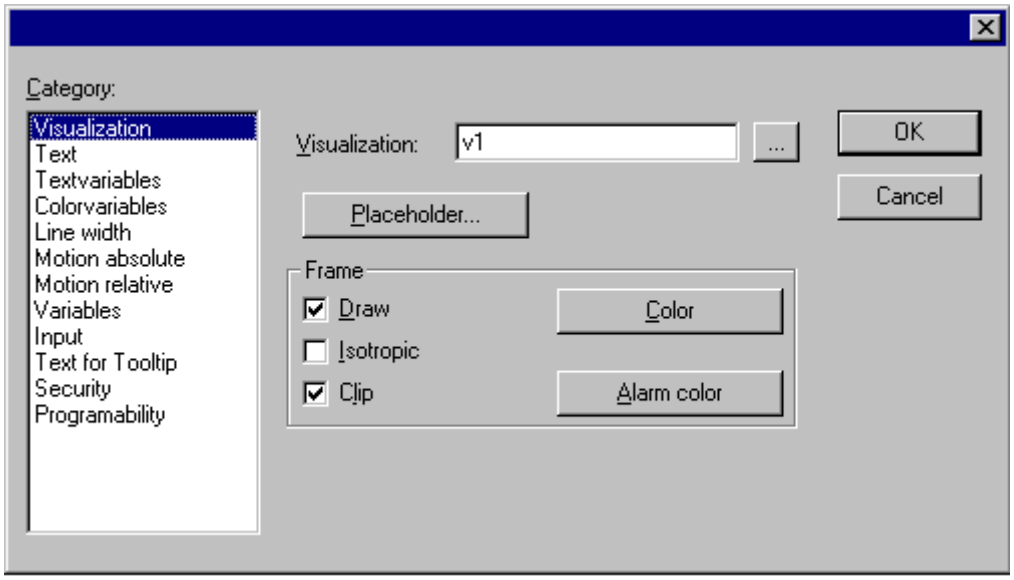
The record is running from the left to the right; the latest value is shown on the leftmost position; every 200 milliseconds the current value will be added to the display. The arrow buttons in the symbol bars allow shifting the displayed time range. Using the simple arrow buttons you can shift step by step, using the double arrows you get to the end resp. start of the record.

For example: if you go to the start of the record by pressing the double arrows pointing to the left, you get a still display of the former values. If you then move the marker (grey triangle in the upper left corner) along the time axis, you can read the exact values of each of the both recorded variables for each time in the rectangle elements below the diagram.

1.4.3.5.28 Visualization

When you insert a visualization as an element in another visualization, you are creating a "reference" of the visualization.

The configuration of this reference can be done in the Visualization category within the visualization element configuration dialog box.



Enter the object namefor the visualization, which should be inserted, in the Visualization field. Use the ... button to open a dialog box containing the visualizations available in this project. Any visualization may be used with the exception of the current one.

The possible settings concerning the visualization frame are the same as described for a bitmap (see above).

The *[Placeholder]* button leads to the 'Replace placeholder' dialog. It lists in the 'Placeholder' column all the placeholders which had been inserted in the configuration dialogs of the "mother"-visualization and offers in the 'Replacements' column the possibility of replacing these for the current reference with a definite value. Which replacements are possible in a given case depends on whether a value group was predefined in the 'Extras' 'Placeholder list' dialog in the "mother"-visualization. If this is the case, it will be displayed in a combo box for selection. If nothing was pre-defined, double clicking on the corresponding field in the Replacements column opens an editing field which can be filled in as desired.

A further possibility for replacing placeholders in references occurs directly when you define the call of a visualization by an entry into the **Zoom to vis**. option field in the configuration dialog ('Input' category).

No control of the chronological sequence of replacements is possible! Therefore no placeholders should be replaced with text that also contains placeholders!

When using placeholders it is no longer possible to check for invalid entries in the configuration of the visualization element immediately upon compilation of the project. Hence the appropriate error messages are first issued in Online mode (...Invalid Watch expression..).

Example of an application of the placeholder concept

Instances of a function block can easily be displayed with the help of references of the same visualization. For example, in configuring the visualization visu, which visualizes the variables of function block, one could begin each variable entry with the placeholder FUB (e.g. FUB.a). If a reference from visu is then used (by inserting visu in another visualization or by calling via 'Zoom to vis.'), then in the configuration of this reference the placeholder FUB can then be replaced with the name of the function block instance to be visualized.

This might look like shown in the following:

In the project define a function block containing the following declarations:

```
FUNCTION_BLOCK fu

VAR_INPUT

changecol : BOOL; (* should cause a color change in the visualization
*`)

END_VAR
```

In PLC_PRG define two instances of 'fu':

```
inst1_fu : fu;

inst2_fu : fu;
```

Create a visualization object 'visu'. Insert an element and open the configuration dialog, category 'Variables'. Enter in field 'Change color' the following: "FUB.changecol". Open category 'Input' and enter in field 'Tap Variable' "FUB.changecol". Open category 'Text' and enter"FUB - change color ". Define an alarm color in category 'Colors'.

Create another visualization object 'visu1'.

Insert visualization 'visu' twice in 'visu1' (two references of 'visu').

Mark the first reference of 'visu' and open the configuration dialog of category 'Visualization'. Press button 'Placeholder', so that the placeholder list will be displayed. There replace entry 'FUB' by 'PLC_PRG.inst1_fu'.

Now mark the second reference of 'visu' and (like described for the first one) replace 'FUB' by 'PLC_PRG.inst2_fu'.

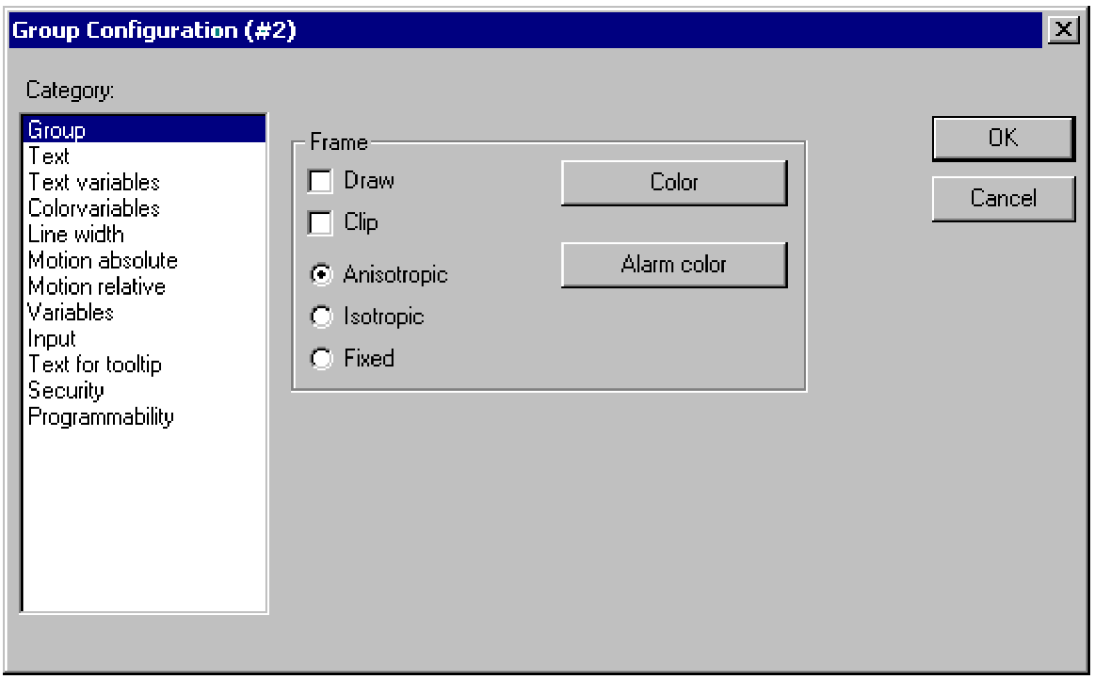
Now in online mode the values of the variables which are used to configure the two instances of 'fu' will be visualized in the corresponding reference of 'visu'.

Of course the placeholder FUB can be used at all places in the configuration of 'visu' where variables or text strings are entered.

> *Attention: Online behavior of a visualization reference: If you insert a visualization and then select and configure this reference, it will be regarded as a single object and in online mode will react to inputs correspondingly to its configuration. In contrast: if you do not configurate the reference, then in online mode its particular visualization elements will react exactly like those of the original visualization.*

1.4.3.5.29 Group

The configuration dialog for a group of visualization elements offers the same options Anisotropic, Isotropc,Fixed, Draw, Clip, Color and Alarm color in field 'Frame' as that for a Bitmap. For example regard the possibility to keep the size of the single elements of the group even when the frame gets stretched or compressed.



1.4.3.5.30 Special input possibilities for operating versions

The visualization can target specifically be used with CODESYS HMI or as web visualization as a mere operating interface. Then no menus and status and tool bars will be available to the user and no possibility to modify the code.

Thus, when a visualization is created with CODESYS for the purpose of being used as a 'operating version' the principal control and monitoring functions in a project must be assigned to visualization elements thus making them accessible via mouse click or keyboard in Online mode.

See in the following some special input possibilities to configure visualization elements for the purpose of being used in CODESYS HMI. They are available in the configuration dialog for a visualization element:

Enter internal commands in the field Execute program in the category Input according to the following syntax (The dialog 'Configure Programs' is available for this purpose):

INTERN <COMMAND> [PARAMETER]*

The following table shows the available internal commands. Some of them expect to receive several parameters, which are then entered separated by spaces. Optional parameters are enclosed in square brackets. For those commands which require that a Watch list be specified, a placeholder can be used instead of the direct name. If you enter several commands for one element, these are separated by commas.

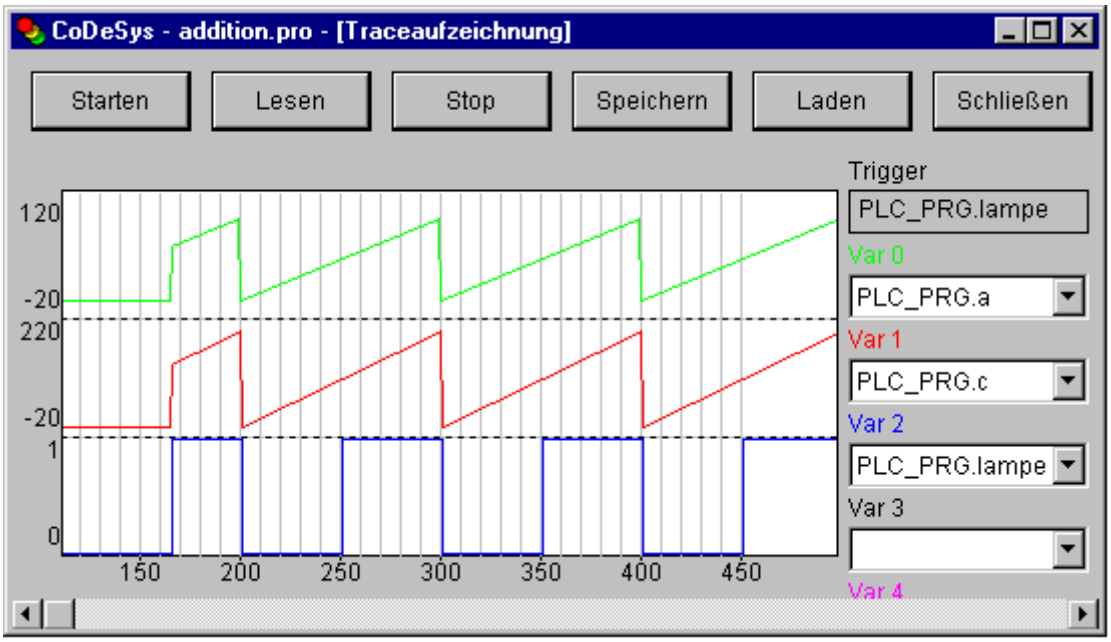
Command	**The equivalent in the programming version of CODESYS HMI**	**Explanation**
ASSIGN <Variable>:=<Expression>	Assignment	A variable or expression gets assigned to another variable. Example: INTERN ASSIGN PLC_PRG.ivar1:=PROG1.ivar +12;
<Path executable program> [Path of the file to be opened] 2)	Program call	The program will be executed. Example: C:\programms\notepad.exe text.txt
LANGUAGEDIALOG2)	'Extras' 'Settings' ↯ *Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*	The dialog for visualization settings which includes the category language gets opened.
LANGUAGE <language identifier as used in the currently set language file *.vis, *.tlt or *.txt> Attention: For visualizations it is recommended to use the *.vis language file.	'Extras' 'Settings' ↯ *Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*, Language	The desired language is set without using the dialog for visualization settings.
LANGUAGE DEFAULT <language identifier as used in the currently set language file>	'Extras' 'Settings' ↯ *Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*, Language	For dynamic texts the default language will be used, which is defined in the currently included XML file.
DELAY <delay time in milliseconds>[1) 2)]		The next command will not be executed before this time has elapsed. E.g. a delay of 500 ms is necessary between DEFINERECEIPT, READRECEIPT and SAVEWATCH.
DEFINERECEIPT <name of watch list>	Select watch lists	A watch list is selected from the receipt manager which enters your name (name) when the command is given. The variables in this watch list are registered and displayed.

Command	The equivalent in the programming version of CODESYS HMI	Explanation
READRECEIPT <name of watch list>	'Read receipt'	In the defined watch list the pre-definition of the variables will be replaced by the current values. Regard: The watch list must be defined before via DEFINERECEIPT and a delay of 500 ms must be inserted (see above: command DELAY).
WRITERECEIPT <name of watch list>	'Write receipts'	The name of a watch list of the receipt manager is expected. The receipt of this watch list will be written. A previous execution of DEFINERECEIPT is not necessary.
SAVEWATCH	'Save watch list'	The receipt will be read into the current watch list which will be stored in a file. Important: call a previous DEFINERECEIPT to define the current receipt and insert a delay of 500 ms (see above: command DELAY).
LOADWATCH	'Load watch list'+ 'Write receipt'	The standard window 'File open' appears, from which a previously stored receipt can be selected. This receipt will be immediately written into the controller system.
CHANGEUSERLEVEL	-	A dialog for setting the user group level will open. The eight user group levels are offered for selection.
CHANGEPASSWORD	cp. 'Project' 'User Group Passwords...'	A dialog for changing the user group password will appear.
SAVEPROJECT[1) 2)]	'File' 'Save'	The project will be saved.
EXITPROGRAM[1) 2)]	'File' 'Close'	The program will be exited.
PRINT[1) 2)]	'File' 'Print'	The current visualization will be printed out online.
HELP <name of help file>[1) 2)]	Call of a help file	Depending on which language is set for the visualization, a help file will be called which is entered for that language in the codesys.ini-file *Chapter 1.4.3.6.1 "'Extras' 'Settings'" on page 700*.

Command	The equivalent in the programming version of CODESYS HMI	Explanation
TRACE[1) 2)]	Resources, Sampling Trace	The window for trace recording (Sampling Trace) will be opened. The menu commands Trace Start, Read, Stop, Save, Load which are available in the full version are available in this window.
CAM[1) 2)]	Resources, CAMs	If there is a CAMs definition available in the project (Resources), the CAM-Editor will be opened. As soon as the editor will be closed again, it will be returned to the visualization.
CNC[1) 2)]	Resources, CNC program list	f there is a CNC program list available in the project (Resources), the CNC editor will be opened. As soon as the CNC editor will be closed again, it will be returned to the visualization.
[1)] not supported for target visualization [2)] not supported for web visualization		
Only for usage in a web visualization: The following commands can be used to give the write access in online mode on a visualization to a certain client. This is of interest if multiple clients at the same time might modify data on the PLC *Chapter 1.4.3.9.4 “Access protection for multi-client operations” on page 716*.		
REQUESTWRITEACCESS	Request for write access	
RELEASEWRITEACCESS	Deallocation of the write access which has been requested before	
GLOBALRELEASEWRITEACCESS	Global deallocation of the write access	
INTERN LINK <URL>	The web visualization will switch over within the browser to the defined URL (Unified resource location)	
INTERN LINK <HTTP file path>	The defined file will be opened.	

Command	The equivalent in the programming version of CODESYS HMI	Explanation
INTERN LINK mailto:<email address>	The entry mask for sending an EMail to the defined address will be opened; e.g. " INTERN LINK mailto:s.sdfjksk@compa-nyxy.com"	
INTERN CONNECT_TO <PLC name>|<Start-Visu>	The PLC (target) will be switched; Preconditions: The web server must be configured appropri-ately with the connection parameters for the respective target systems and a matching ini-file for the PLCHandler must be available. PLC name: Name of the target PLC, as defined in the PLCHandler ini-file. Start-Visu: Name of the desired start visualization page. The web server will establish the connection to the respective PLC automatically. Example: "INTERN CONNECT_TO PLC1|PLC_VISU"	

Dialog for the trace recording in the operating version:



1.4.3.6 Configuring visualization objects

Besides the configuration of the individual visualization elements also the visualization object as a whole can get configured. This is possible concerning the settings for frame, language, grid, background, placeholders etc. as well as the assignment of special hotkey definitions (keyboard usage), which should be valid for exactly one visualization object. These settings are done in the visualization editor via commands of the 'Extras' menu.

Besides that outside of the editor you can do some settings in the Properties dialog of the visualization object, which concern the usage as web visualization or as Master layout.

Regard that in the project options a separate directory can be defined for visualization files
↳ *Chapter 1.4.1.2.2.7 “Options for directories” on page 207*.

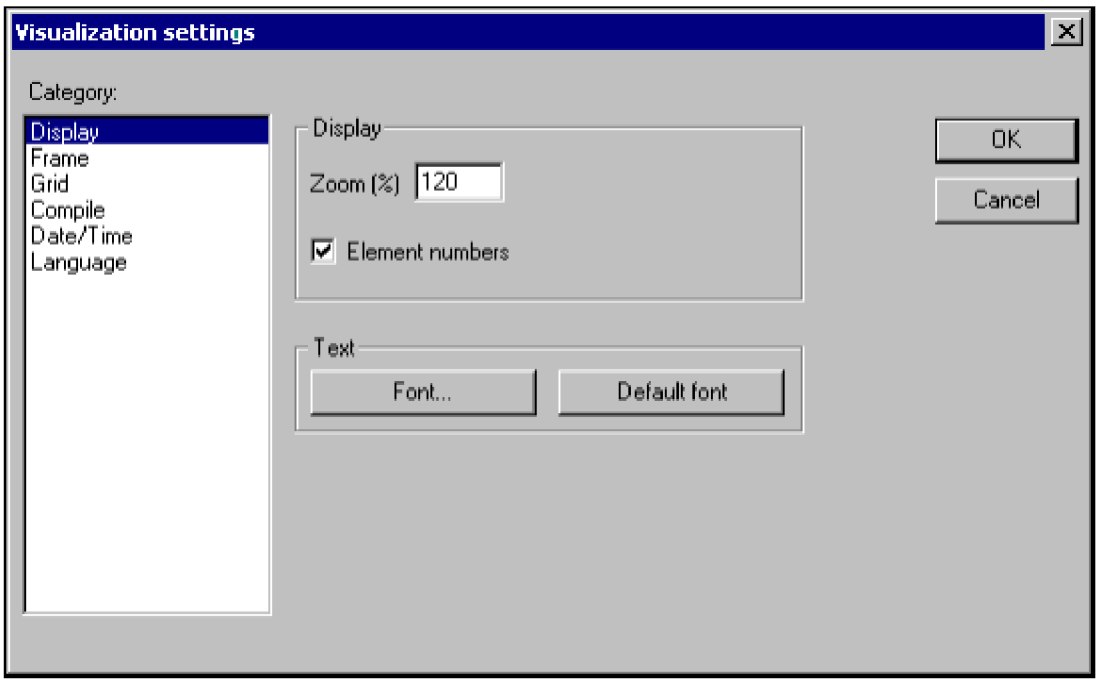
Image files can be used for the background (static) of a visualization in "bitmap" elements (static or dynamic use).

1.4.3.6.1 'Extras' 'Settings'

When this command is used, a dialog box will open in which you can make certain settings that affect the display and language in the visualization as well as the check of the visualization variables.

> *The categories Display, Frame and Language also can be edited in the online mode.*

- Category Display : Enter a zoom factor into the field Zoom of between 10 and 500 % in order to increase or decrease the size of the visualization display. If option 'Element numbers' is activated, in each visualization element its number will be displayed, via which it is managed in the Element List.
  In the 'Text' area via button *[Font]* the standard dialog for selecting a font can be used for defining the font for the current visualization. This font will be automatically applied to all visualization elements, which do not have got a different font explicitly assigned via the element configuration (category 'Text'). Via button *[Standard-Font]* the project font, which is defined in the Project Options, can be set as Visualization font, which also will only effect those elements without individual font definitions. Even those elements however can get assigned the currently valid standard font by the *[Standard-Font]* button in their element configuration dialog.



- Category Frame: If **Auto-scrolling** is selected, the visible portion of the visualization window will move automatically when you reach the edge while drawing or moving a visualization element. If *"Best fit in Onlinemode"* is selected, the entire visualization including all elements will be shown in the window in Online mode regardless of the size of the window. When *"Include Background Bitmap"* is selected, the background bitmap will be fitted into the window as well, otherwise only the elements will be considered.
- Category Grid: Define here whether the grid points are visible in the offline mode, whereby the spacing between the visible points is at least 10 even if the entered size is smaller than that. In this case the grid points only appear with a spacing which is a multiple of the entered size. Selecting *"Active"* causes the elements to be placed on the snap grid points when they are drawn and moved. The spacing of the grid points is set in the field Size.

- Category Compile: Per default the used variables get not checked for validity before going online with a project. If you want this check to be done already during a build run of the project (command 'Project ' 'Build' resp. 'Rebuild all'), then activate option *“Check visualization variable on compile”*. Invalid variables will be announced by a warning "... invalid watch expression..." in the message window.
- Category Time/Date: Here you can define in which format the date or time data outputs in a visualization should be displayed.
  If option *“Formatted time display”* is activated, outputs controlled by a variable of the corresponding time or date datatype (e.g. timevar:TIME;) will be displayed in that format which is defined in the respective edit field here in the dialog. If the option is not activated or if for a datatype no formatting is defined here, the output will be displayed in that format which is used when assigning values to date and time constants (e.g. "t#12h34m15s"), see 'Time data types' *Chapter 1.4.1.8.1.6 “Time data types” on page 444*.

During compilation as target visualization a warning will be displayed, if the functionality is not supported by the target system.

Settings for the following datatypes are possible: Format for TIME/TOD, Format for DATE, Format for DT.

You can use the formattings listed in the table stated below. Upper and lower case must be regarded. Empty spaces inserted in a formatting definition will be displayed in the output string at the same position. Additional characters, which should not be interpreted as format definition, must be embraced by single quotation marks.

Example: A project variable "timevar" of data type TIME is defined, which is configured to control the text output of a visualization element. In the Time/Date settings of this visualization, in the edit field at "Format for TIME/TOD" the data hh':'mm':'ss tt is entered. Output by the respective visualization element in online mode (e.g. if timevar has the value "t#12h34m15s"): 12:24:15 PM.

h	Hours as number, with no leading zero for single-digit hours; 12-hour clock
hh	Hours as number, with leading zero for single-digit hours; 12-hour clock
H	Hours as number, with no leading zero for single-digit hours; 24-hour clock
HH	Hours as number, with leading zero for single-digit; 24-hour clock
m	Minutes as number, with no leading zero for single-digit minutes
mm	Minutes as number, with leading zero for single-digit minutes
s	Seconds as number, with no leading zero for single-digit seconds
ss	Seconds as number, with leading zero for single-digit seconds
ms	Milliseconds as number, with no leading zero for single-digit microseconds
t	One-character time marker string; "A" (ante meridiem) for the time between 00:00 and 11:59, "P" (post meridiem) for the time between 12:00 and 23:59
tt	Two-character time marker string: "AM" (ante meridiem) for the time between 00:00 and 11:59, "PM" (post meridiem) for the time between 12:00 and 23:59
d	Day of the month as number, with no leading zero for single-digit days
dd	Day of the month as number, with leading zero for single-digit days
M	Month as number, with no leading zero for single-digit days
MM	Month as number, with leading zero for single-digit days
y	Year as last two digits, with no leading zero for years less than 10 (example: year 2007 is displayed as "7")

yy	Year as last two digits, with leading zero for years less than 10 (example: year 2007 is displayed as "07")
yyyy	Year represented by full four digits (example: year 2007 is displayed as "2007")

- Category Language: Here you can specify in which national language the text that you assigned to an element in the Text and Text for Tooltip options should be displayed. Additionally the option 'Dynamic Texts' allows a dynamic change of the displayed text.

The text display changes only in Online mode!

1.4.3.6.2 'Extras' 'Select Background Bitmap'

Use this command to open the dialog box for selecting files. Select a file with the extension "*.bmp". The selected bitmap will then appear as the background in your visualization.

The bitmap can be removed with the command 'Extras' 'Clear Background Bitmap'.

1.4.3.6.3 'Extras' 'Clear Background Bitmap'

Use this command to remove the bitmap as the background for the current visualization.

You can use the command 'Extras' 'Select Background Bitmap' to select a bitmap for the current visualization ↳ *Chapter 1.4.3.6.2 “'Extras' 'Select Background Bitmap'” on page 702*.

1.4.3.6.4 'Extras' 'Keyboard usage'

The use of hotkeys can optimize the pure keyboard operation of a visualization.

In the configuration of a visualization object you can define hotkeys which will cause actions like visualization elements do. For example you could define that "“ if visualization 'xy' is active "“ in online mode the hotkey <Strg><F2> will stop the program, which also will happen as soon as element 'z' of visu 'xy' gets an input (by mouse-click or via touch screen).

Anyway per default the keys <Tabulator> <Space> <Enter> will work in that way that in online mode each element of a visualization can be selected and activated.

The dialog 'Keyboard usage: set possible keystrokes' can be called in the menu 'Extras' or in the context menu:

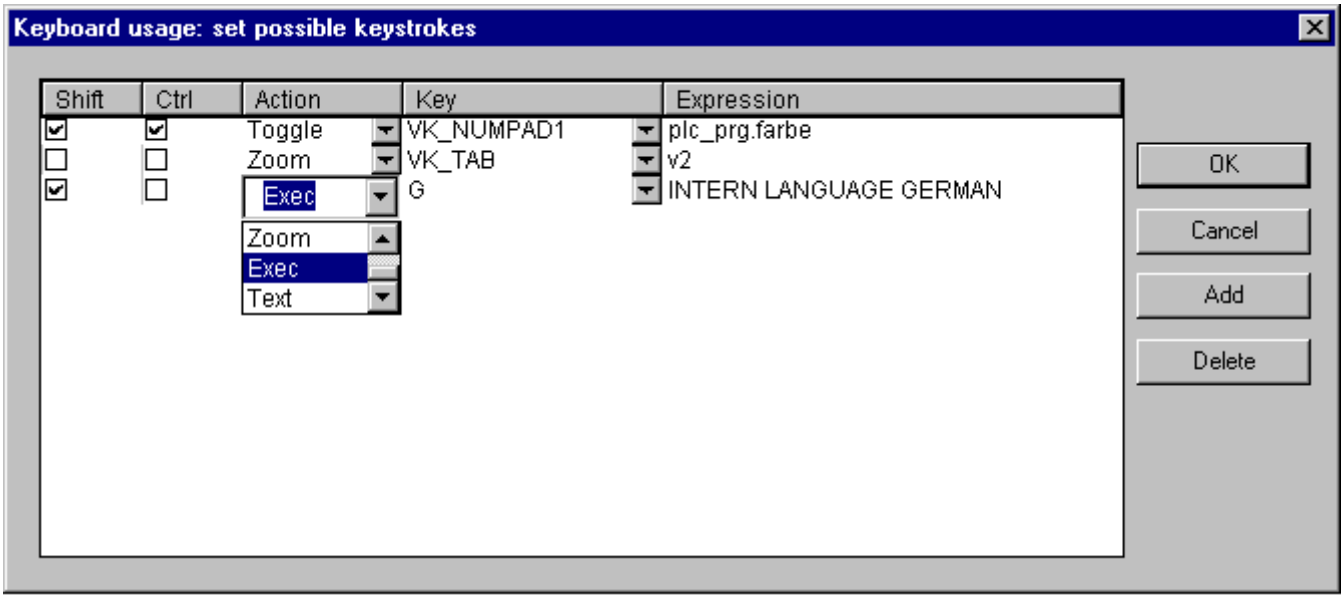


Fig. 11: Dialog 'Keyboard usage: set possible keystrokes'

In column Key a selection list offers the following keys to which an action can get assigned:

VK_TAB	Tab-Key
VK_RETURN	Enter-Key
VK_SPACE	Space-Key
VK_ESCAPE	Esc-Key
VK_INSERT	Insert-Key
VK_DELETE	Delete-Key
VK_HOME	Pos1-Key
VK_END	End-Key
VK_PRIOR	Bild-Key (previous)
VK_NEXT	Bild-Key (next)
VK_LEFT	Arrow-Key (to the left)
VK_RIGHT	Arrow-Key (to the right)
VK_UP	Arrow-Key (up)
VK_DOWN	Arrow-Key (down)
VK_F1-VK_F12	Function keys F1 to F12
0-9	Keys 0 to 9
A-Z	Keys A to Z
VK_NUMPAD0 - VK_NUMPAD9	Keys 0 to 9 of the numeric keypad
VK_MULTIPLY	Key* of the numeric keypad
VK_ADD	Key+ of the numeric keypad
VK_SUBTRACT	Key- of the numeric keypad
VK_DIVIDE	Key÷ of the numeric keypad

In the columns **Shift** and **Ctrl** you can add the *[Shift]*- and/or the *[Ctrl]*-key to the already chosen key, so that a key combination will result.

See the possible key combinations for the particular visualization variants ↬ *Chapter 1.4.3.12 “Possible key combinations for the particular visualization variants” on page 718*.

In column **Action** you define what should happen as soon as the key (combination) will be pressed. Select the desired action from the list and insert an appropriate expression. See in the following the available actions and valid expressions, corresponding to those which can be set in the configuration dialog of category 'Input':

Action	Meaning	Expression
Toggle	Toggle variable	Variable, e.g."plc_prg.tvar"
Tap true	Tap variable (set to TRUE)	Program variable, e.g. "plc_prg.svar"
Tap false	Tap variable (set to FALSE)	Program variable, e.g. "plc_prg.xvar"
Zoom	Zoom to Vis.	Name of the visualization object to which you want to jump, e.g. "Visu1"
Exec	Execute program	Name of the executable file, e.g. "notepad C:\help.txt" (notepad will start and open the file help.txt)
Text	Text input of variable 'Textdisplay'	Number of the element for which the text input is to be configured, e.g. "#2". Display of element numbers can be switched on in 'Extras' 'Settings'; also see 'Elementlist...'.

In column Expression you must enter "“ depending on the type of action "“ either a variable name, a INTERN-command, a visualization name of a text string, exactly like you would do in the configuration dialog of category 'Input' for the corresponding visualization element.

Use button *[Add]* to add another empty line at the end of the table. Use the *[Delete]* button to remove the line where the cursor is positioned currently. *[OK]* resp. *[Cancel]* will save resp. not save the done settings and close the dialog.

The keyboard usage can be configured separately for each visualization object. Thus the same key (combination) can start different actions in different visualization.

Example

The following key configurations have been done for the visualizations VIS_1 and VIS_2:

Table 45: VIS_1:

Shift	Ctrl	Action	Key	Expression
x		Toggle	A	PLC_PRG.automatic
	x	Zoom	Z	VIS_2

Table 46: VIS_2:

Shift	Ctrl	Action	Key	Expression
		Exec	E	INTERN LANGUAGE DEUTSCH
	x	Zoom	Z	PLC_VISU

If you now go online and set the focus to VIS_1, then pressing *[Shift]* + *[A]* will cause that variable PLC_PRG.automatic will be toggled. *[Ctrl]* + *[Z]* will cause a jump from Visu1 to VIS_2.

If VIS_2 is the active window, pressing key *[E]* will cause that the language within the visualization will switch to German. *[Ctrl]*+ *[Z]* here will cause a jump to visualization PLC_VISU.

1.4.3.6.5 Master layout

The usage of a "Master layout" in visualizations for example could be used to provide a dialog in various visualizations without the need to explicitly insert it in each of them. Whether and when the dialog will be displayed in online mode could be controlled via a variable which is defined in the Master layout configuration for the visibility of the dialog elements .

Use as master layout

If a visualization is defined as "Master layout", it will be inserted automatically in all other visualizations (if those are not explicitly excluded, see below) and will be available there with all its functions during online mode. It always will be inserted on the front level, if you however it always want to get it in the background, then activate option 'in background' in the properties dialog of the visualization element (see below). A master layout cannot be edited any longer in the visualization where it has been inserted. Modifications of the configuration only can be done in the master layout visualization itself.

Appointing a visualization to be the master layout is done in the Properties dialog, which can be opened for an object currently selected in the Object Organizer via command 'Project' 'Objectproperties' ↳ *Chapter 1.4.1.2.4.12 “'Project' 'Object properties'” on page 261*. Option Master layout must be activated for this purpose. If prior to this another visualization has been defined to be a master layout, that one will automatically be re-defined to be a "normal" visualization (option Visualization in the Properties dialog.

Visualization without master layout

Also in the Properties dialog of a visualization object you can define that it should be used as a Visualization without master layout.

1.4.3.6.6 Use as web visualization

If a project is created for a web visualization, for each visualization object you can define whether it should be used for this purpose or not ↳ *Chapter 1.4.5.1 “Overview” on page 721*.

There to select the visualization object in the Object Organizer and open the Properties dialog ↳ *Chapter 1.4.1.2.4.12 “'Project' 'Object properties'” on page 261*. If in the Target Settings the options web visualization are activated, in the Properties dialog the corresponding options use as web visualization automatically also will be activated. In order to explicitly exclude the object from the use in a web visualization deactivate the appropriate option.

1.4.3.7 Images in visualization

Image files can be used in a visualization object for the background as well as in visualization elements of type "Bitmap". The following formats are supported:

CODESYS HMI:	*.bitmap, *.jpg, *.tif
Web visualization:	*.bmp, *.jpg

In "Bitmap" elements a dynamic change of images can be reached by specifying the image file name by a project variable instead of using a definite file reference.

↳ *Chapter 1.4.3.5.19 “Bitmap” on page 669*

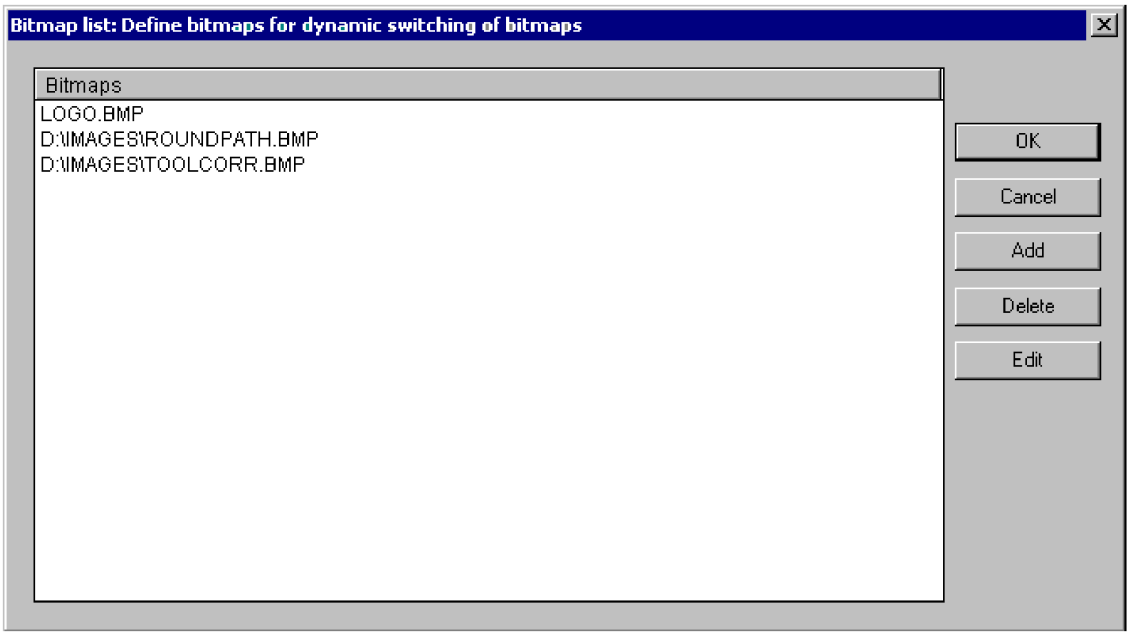
↳ *Chapter 1.4.3.7.1 “'Extras' 'Bitmap list'” on page 706*

The background image of a visualization can only be defined by a static file reference.

↳ *Chapter 1.4.3.6.2 “'Extras' 'Select Background Bitmap'” on page 702*

1.4.3.7.1 'Extras' 'Bitmap list'

In the 'Extras' menu which is available when the Visualization editor is active, command 'Bitmap list' opens dialog 'Bitmap list:...". Here you can enter image files, which then can be used in all visualization objects via a project variable defining the content of a "Bitmap" element. This allows a dynamic change of images in online mode - alternatively to the static use of image files, which for this purpose are not to be part of the bitmap list.



In order to enter an image file in the bitmap list, use button *[Add]*. The standard dialog for browsing for a file will be opened, where you can select an image file (*.bmp, *.tif, *.jpg, regard the supported file formats). If the chosen file is in the project directory, only the file name will be added to the list, otherwise the full path.

Via button *[Delete]* the currently selected entry (click by the right mouse-button) can be deleted.

In order to modify an entry, either edit it directly (click by the left mouse-button on the entry opens the edit frame), of use button *[Edit]* to open the standard dialog for browsing for a file. For the latter the entry first must be selected by a click with the right mouse-button.

[OK] saves the current list.

Each file name specified in the list now can be assigned to a variable of type STRING, which is entered in the bitmap configuration of a visualization element in order to define the image to be used ↳ *Chapter 1.4.3.5.19 “Bitmap” on page 669*.

1.4.3.8 Language switching

1.4.3.8.1 Overview

The language switching for texts, tooltips and alarm messages in a visualization can be done via static texts or via dynamic texts, which must be provided by a file ↳ *Chapter 1.4.3.8.2 “Static language switching” on page 707* ↳ *Chapter 1.4.3.8.3 “Dynamic language switching” on page 709*. Unicode format is only possible with dynamic texts.

How can a language switching be done:

In the configuration dialog ‘Visualization settings’ in the selection list below 'Language' you can choose one of the languages defined in the currently used language file, which should be used as '(start) language' in online mode; for the example shown below: german and english.

A language switch in online mode is done via an input on a visualization element. For this purpose the internal commands "INTERN LANGUAGE <language>" and "INTERN LANGUAGEDIALOG" are available, which can be used in the configuration dialog in category 'input' ↳ *Chapter 1.4.3.5.30 “Special input possibilities for operating versions” on page 695* ↳ *Chapter 1.4.3.5.15 “Input” on page 662*.

Example: Insert a button element which can be used to switch the visualization texts to German. For this purpose, label the element with ‘German’, in configuration category ‘Input’ activate option ‘Execute program’ and define a command "INTERN LANGUAGE <language>". "language" is to be replaced by the language shortcut used in the language file, thus for the vis-file example shown under 'Static Language Switching' ↳ *Chapter 1.4.3.8.2 “Static language switching” on page 707*: "INTERN LANGUAGE german". If the button will be operated in online mode the visualization texts will be displayed according to the entries which are available for "german" in the language file.

Unicode format: Unicode format is only possible with dynamic texts. An appropriate entry must be available in the XML language file. Additionally the following preconditions must be complied:

1. In the configuration of the visualization object a unicode-enabled font must be specified.

   Attention: Currently for tooltip texts this font must be specified in the codesys.ini file ("FaceTooltip=<font>") and is not read from the XML language file!

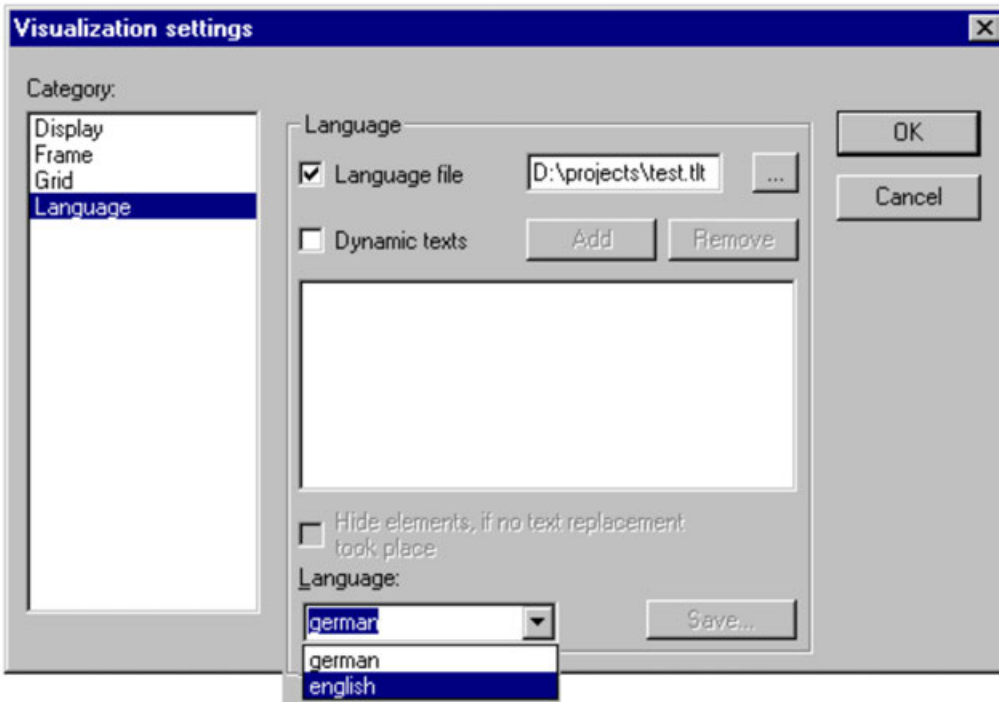
2. In order to get work the unicode font in the target visualization, it must be supported by the target system.
3. In order to get work the unicode font in the web visualization, the configuration entry TOOLTIPFONT in webvisu.htm must be set with a unicode-enabled font.

1.4.3.8.2 Static language switching

For a static switch of the language a language file (*.vis, *.tlt, *.txt) file can be used (for how to create see below). The difference to the dynamic language switching is that the language cannot be defined by a project variable during run time.

*For visualizations it is recommended to use the *.vis language file, because *.tlt- or *.txt-translation files only work for visualizations in CODESYS or CODESYS HMI and also in those not for the Meter, Bar display and Histogram elements.*

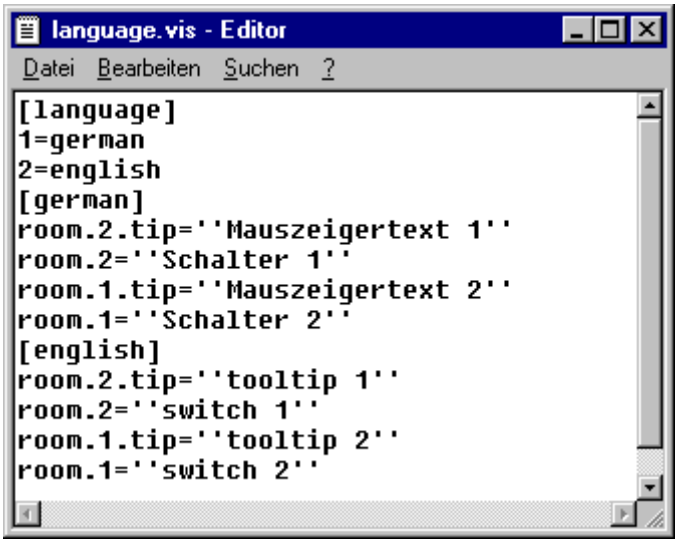
In dialog ‘Visualization settings’ you configure, which language file should be used with the project. In order to choose a translation (*.tlt, *.txt) or a pure visualization language file (*.vis), which contains the texts in the various languages, activate the Language file option and in the input field next to it enter the appropriate file path. Via the button you get the standard dialog for opening a file.



Regarding creating a translation file *.tlt, see 'Project' 'Translate into another language'
↳ *Chapter 1.4.1.2.3.15 “'Project' 'Translate into another language'” on page 233*.

For creating a special language file *.vis for the visualization, perform the following steps:

1. Open likewise the Settings Visualization dialog, Language category.
2. Choose option language file. In the associate input field enter where you want to store the file. The extension is .vis. If a language file with the extension .vis is already present, it will be offered to you here.
3. In the input field next to Language you fill in a keyword for the language which is currently used in the visualization, i.e."german" (or "D"). Then press the button *[Save]*.
4. A file with the extension .vis will be created, which now can be edited by a normal text editor. For example you can open the file by notepad. Example file:



5. You get a list of the text variables for the language currently used in the visualization. It includes a reference to the title of this list, for example "1=german" as reference to the title [german]. You can extend the list by copying all lines, then replacing the German by English text and setting a new title [english]. Beyond the line 1=german you accordingly have to add 2=english.

   To view the visualization in one of the prepared languages, open the dialog *“Language”* again. In the option field beyond *“Language”* now you can choose between german and english (for the example described above).

1.4.3.8.3 Dynamic language switching

Dynamic texts allow switching between different, always language-assigned text versions for a visualization element (texts, texts for tooltips, message texts in alarm tables). The difference to static text is that the definite text selection also can be done via a variable used in the application.

In the configuration of the element a Prefix-ID-combination is entered, which is assigned to a text in a XML files (also named "textlist" in the following). The ID can be defined by a project variable.

Example of application: The ID represents an error number, as Prefix e.g. "Error" is used. The language file provides via the Prefix-ID-combination an appropriate error message, which (depending on the currently set language) will be displayed in this language.

Please regard:

- The language files for dynamic texts can be created in Unicode (UTF-16) or ANSI (ISO-8859-1), e.g. " <?xml version='1.0' encoding='UTF-16'?>". For the preconditions for the usage of unicode-able fonts please see static language switching *Chapter 1.4.3.8.2 “Static language switching” on page 707.*

- For the target visualization the start language, the directory for the XML file to be used and a list of XML files can be defined by the target system. This allows to modify these parameters later without the need of creating a new boot project. Thus in an easy way existing textlists can be modified (start language, texts) resp. new languages can be added. If the target system is providing such a configuration, the textlists which are defined for the visualization, will not be regarded in online mode! If no target-specific configuration is available for the language switching, then after a modification of the textlists a project download must be done.

1.4.3.8.4 Configuration of dynamic language switching

You can control dynamically which text (also alarm message text in an alarm table) will be displayed in a visualization element resp. as tooltip of an element in online mode by using Prefix-ID-combinations, each pointing to another text defined in a XML file.

For this purpose a XML file describing the text assignments must be linked to the project in the configuration of the visualization (*Chapter 1.4.3.6.1 “'Extras' 'Settings'” on page 700*). The XML file must have a certain format. A language code is added to the particular texts, thus later you not only can switch between different text contents but also language switching is supported.

In the configuration of a visualization element, for which the text display should be switched dynamically, Prefix and ID are entered (see below) whereby the ID can be provided by a project variable. A default language can be defined via an INTERN command (LANGUAGE DEFAULT).

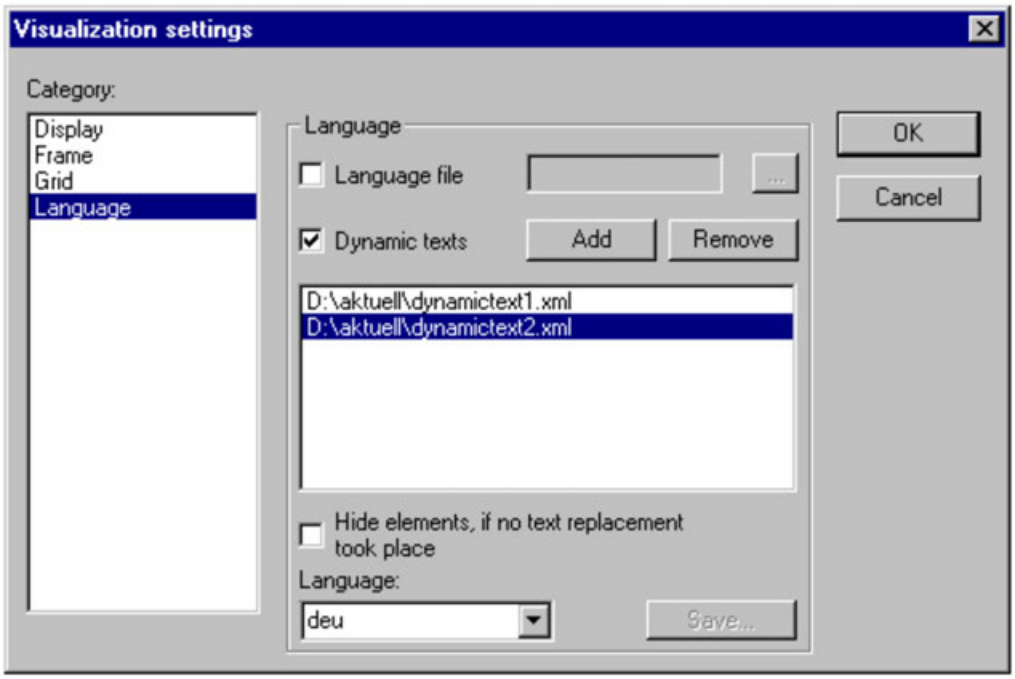
Set the following entries in the different configuration dialogs of a visualization:

1. Link the XML file(s) and choose the (start) language: Dialog 'Settings' category Language:

   Activate option *“Dynamic texts”* and press *[Add]*, in order to link one or several XML files, which are available on your system, to the project. The selected files will be listed in the window below the button. Press *[Delete]* if you want to remove a selected file from the list. If you want to get displayed just those visualization elements, for which a dynamic text replacement is done, then activate option *“Hide elements, if no text replacement took place”*.

   Selecting one of the language identifiers offered in the selection list at field *“Language”* will cause the display of those text versions (for the corresponding prefix-ID-combination) which are marked with that language identifier in the XML file.

   ⇨ Configuration dialog Settings, category Language, for dynamic texts:



2. Specify the ID (as used in the XML file) in configuration dialog 'Variables' in field 'Textdisplay' resp. 'Tooltip-display':

   Enter a value (number) resp. a project variable which should define the ID of a text (as used in the XML file).

   In case of message texts of an alarm table, the ID must match the respective line number in the table .

3. Define the text format in configuration dialog 'Text' resp. 'Text for tooltip':

   In the Content field, insert a placeholder "%<PREFIX>" at that position of the text, where you want to get displayed a dynamic text in online mode. Instead of "PREFIX" you can enter any string matching with a PREFIX-definition used in the XML textlist. See the description for the 'Text' configuration dialog.

For each prefix-ID-combination, which is found in a linked XML file, the assigned text will be displayed in the visualization element in online mode. If no appropriate entry is found, no replacement will be done.

1.4.3.8.5 XML file for dynamic texts

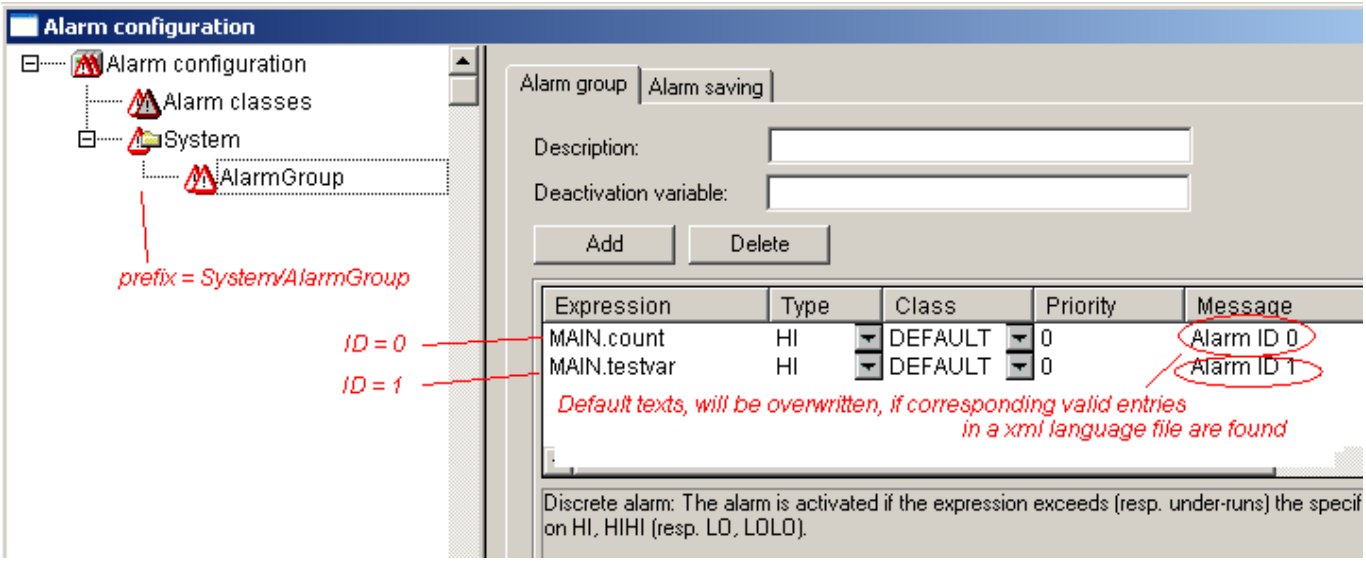
For a description how to use dynamic texts in the visualization see

'Dynamic Language Switching' *Chapter 1.4.3.8.3 “Dynamic language switching” on page 709*

'Configuration of Dynamic Language Switching' *Chapter 1.4.3.8.4 “Configuration of dynamic language switching” on page 709*

The basic file must be available in XML format (<file name>.xml). In this file texts are assigned to identifiers(which are a combination of a prefix and a ID). In case of normal texts and tootip texts these combinations can be entered in the configuration of a visualization element, in order to get displayed the respective text in online mode. In case of an alarm table the prefix and ID are given by the "path" of the alarm group in the alarm configuration (prefix) and the line number of the text definition in the alarm group configuration table (ID).

Example picture showing prefix and ID in an alarm configuration:



In the header section of the file a default language and a default font assigned to a language can be defined. See an example of an description file at the end of this page.

The descriptions in the XML file are enclosed by tag <dynamic-text> and <\dynamic_text> which have to be entered at the beginning resp. end of the file.

The language files for dynamic texts (as of program version V2.3.4.0) can be created in Unicode (UTF-16) or ANSI (ISO-8859-1). This is to be defined via the encoding syntax at the beginning of the XML file, e.g. " <?xml version='1.0' encoding='UTF-16'?>". For the preconditions for the usage of unicode-able fonts, see 'Language Switching' *Chapter 1.4.3.8 “Language switching” on page 706*.

To facilitate the creation of the language file the standard installation provides some Excel macros.

Primary formats of the XML file, which do not use the <dynamic_text>\<\dynamic_text> tags or the header section, will be supported further on.

Structure of the file:

The header section starts with <header> and is closed with <\header>. If you want to define a default language, use entry <default-language>. A default font which is assigned to a certain language, can be defined via entry <default-font>. These entries are optional. If they are missing, the dynamic text in the visualization will be displayed according to the local configuration settings of the visualization.

<header>	
<default-language><language></default-language>	Default language; that means that if there is no text entry available for the currently set language, that text will be used which is found within the same text entry for the default language. If also for the default language no text is found, "<PREFIX> <ID>" will be displayed. If multiple XML files are used, thus providing multiple headers, only that header section will be regarded, which is read at last. So it is reasonable to use only one header section ! The language token must correspond which one of that used in the text entries (see below). Note: In online mode the default language can be set explicitly via a visualization element configured with command INTERN LANGUAGE DEFAULT in category Input, Execute program.
<default-font>	Default font for <language>: The given font (e.g. "Arial" will automatically be used for all elements, which display dynamic texts in <language>. The language token must correspond which one of that used in the text entries (see below). For the preconditions for the usage of unicode-able fonts please see 'Language Switching' ↳ *Chapter 1.4.3.8 “Language switching” on page 706*.
<language><language></language>	
<font-name><font></font-name>	
</default-font>	
<default-font>	Further default fonts for other languages
<language>.....	
....	
</default-font>	
</header>	

The list of the assignments of the Prefix-ID combinations to texts must start with <text list> and end with </text list>. The particular text entries each start with <text prefix> and end with <\text>.

A text entry which is assigned to a Prefix-ID-combination must contain the following lines:

<text prefix>= "<PREFIX> id="<ID>"	"PREFIX" corresponds to the <PREFIX> used in the visualization element configuration (category Text or Tooltip); in case of an message text used in an alarm table the path of the *alarm group* within the alarm configuration starting at node "System" must be specified, e.g. "System/Alarmgroup_xy" "ID" corresponds to the entry in category 'Variables', Textdisplay or Tooltip-display; in case of an of an message text used in an alarm table the number of the line which defines that text in the 'Alarm groups' must be specified ↳ *Chapter 1.4.1.4.2.4 "Alarm groups" on page 368*.
<language> <!CDATA[<TEXT>]]</language>	Use any string as an 'language' identifier (e.g. "english"). This identifier then will be displayed in the 'Settings' dialog, category Language of the visualization element in the selection list at 'Language'; instead of "TEXT" insert any text which then will be displayed instead of the above defined ID-prefix-combination in the visualization element.
</text>	

For each prefix-ID-combination at least for 1 language a text entry must be available. E.g. see in the file example shown below: <deutsch> indicates the start of the german version of a text, </deutsch> terminates the text.

Dynamic texts on the one hand can serve to display Texts in different languages, but of course on the other hand they also can be used to change the content of a text (same language) display dynamically.

Example: You want to have two visualization elements, one for visualizing the current machine identification, the other for visualizing an error message according to a currently given error number:

1. Define in PLC_PRG the following variables: ivar of type INT, defining the current machine identification; errnum of type INT defining the current error number.
2. Configure a visualization element for displaying the current machine identification:

   - In category 'Text'' in the text field enter: "%<Maschine>"

   - In category 'Variables' at Textdisplay" enter: "PLC_PRG.ivar"
3. Configure another visualization element for displaying the error message for the currently occurred error:

   - In category 'Text' in the text field enter: "%<Error>"

   - In category 'Variables' at 'Textdisplay' enter: "PLC_PRG.errnum"
4. Create a XML file, e.g. with name dynamictextsample.xml, according to the syntax described above, which should look as followed for the current example: Sample XML file for dynamic texts ↳ *"Sample XML file for dynamic texts" on page 714*
5. In the visualization open dialog 'Settings', category Language: Activate option 'Dynamic Texts'; Add file dynamictextsample.xml, now available on your computer, to the file list.
6. Go online with the project.
7. In the visualization settings set language to "deutsch". Set PLC_PRG.ivar to "1" and PLC_PRG.errnum to "4711". Now in the visualization elements the following texts should be displayed: "Vorschub" resp. "Fehler an Position 4711". The texts will be displayed in Arial 13.
8. Set PLC_PRG.ivar to "2" and PLC_PRG.errnum to "2000". The texts will change to "Beschleunigung" and "Das ist ein Fehlertext über mehrere Zeilen".

9. In the visualization settings change the language to "english". Now the following texts will be displayed:

   "Acceleration" and "This is a error text over more than one line".

Sample XML file for dynamic texts

```
<dynamic-text>
    <header>
    <default-language>deutsch</default-language>
    <default-font>
      <language>english</language>
      <font-name>Arial</font-name>
      <font-color>0,0,0</font-color>
      <font-height>-13</font-height>
      <font-weight>700</font-weight>
      <font-italic>false</font-italic>
      <font-underline>false</font-underline>
      <font-strike-out>false</font-strike-out>
      <font-char-set>0</font-char-set>
    </default-font>
    </header>
  <text-list>
    <text prefix="ERROR" id="4711">
      <english>Error at position 4711</english>
    </text>
    <text prefix="ERROR" id="815">
      <english>Error at position 815</english>
    </text>
    <text prefix="ERROR" id="2000">
        <english>
          <![CDATA[
          This is a error text over more than
          one line
          ]]>
        </english>
    </text>
    <text prefix="MASCHINE" id="1">
        <english>
            <![CDATA[ Feed rate
          ]]>
        </english>
    </text>
    <text prefix="MASCHINE" id="2">
       <english>
           <![CDATA[ Acceleration
        ]]>
       </english>
    </text>
  </text-list>
</dynamic-text>
```

1.4.3.8.6 Calling up language-dependent online help via a visualization element

The calling of a different Help file with a visualization element can be tied in with the language currently entered for the visualization. For this purpose, the command INTERN HELP must be entered for this element in the configuration dialog in category 'Input' at the location 'Execute program', and a [Visu-Helpfiles] section must be present in the codesys.ini-file. Below this, the corresponding help files must be assigned to the languages available for selection in the visualization.

Example	[Visu-Helpfiles] German=C:\PROGRAMME\HELP\<hilfedatei_german>.chm English=C:\PROGRAMME\HELP\<hilfedatei_english>.chm

1.4.3.9 Visualization in online mode

1.4.3.9.1 Overview

Regard the following items concerning a visualization in online mode:

- Order of evaluation:
  - Dynamically defined element properties (by variables) will overwrite the (static) base settings defined by options in the configuration dialogs.
  - If an element property is defined by a "normal" project variable as well as by the component of a structure variable (programmability), then primarily the value of the project variable will be regarded.
  - A visualization can be configured in that way that in online mode it can be operated solely by inputs via keyboard. This is an important feature especially for using the visualization with CODESYS HMI, as web visualization.
- Before a download of the project regard the current setting of option 'Prevent download of visualization files' in the target settings. This concerns all files which are used in the current visualization. Visualization files are only downloaded for web visualization and can be bitmaps, language files and for web visualization also XML description files.
- The configuration settings for Display, Frame and Language can also be edited in online mode.
- As long as a visualization "reference" is not configured explicitly, the particular elements of the reference in online mode will react on inputs like those of the original visualization ("mother" of the references).
- When you switch the language ('Extras' 'Settings') this will only effect the display in online mode.
- A visualization can be printed in online mode.
- If a visualization is used as target visualization, information on user entries via mouse-clicks can be scanned with the help of special interface functions and thus be used in the project.

You can find descriptions on the online usage of some visualization elements, like e.g. Trend or Table, in the help page on the configuration of the respective element.

1.4.3.9.2 Operation over the keyboard

In order to get independent from the mouse or a touch screen, it is useful to configure a visualization in a way that allows pure keyboard operation:

Per default the following key (combinations) will work in online mode anyway (no special configuration necessary):

- Pressing the *[Tabulator]* key selects the first element in the element list for which an input is configured. Each subsequent pressing of the key moves one to the next element in the list. Within tables you will jump to the next table field. Pressing the key while keeping the *[Shift]* key depressed selects the previous element. Depending on the target a simplified input handling may be possible.
- The *[arrow]* keys can be used to change from a selected element to a neighbouring one in any direction.
- The *[Space bar]* is used to execute an activity on the selected visualization element. If the element is one which has a text output variable or if it is a table field, a text input field will be opened which displays the text contents of the variable resp. the field. Pressing the *[Enter]* key writes in this value.

Additional key (combinations) for the online operation can be defined in the configuration dialog 'Keyboard usage'. There also the keys *[Tab]*, *[Space]* and *[Enter]* can get assigned another functions than the above described standards.

The individual elements of references behave in Online mode identically to the corresponding ones in the visualization that is referenced. They will therefore react the same way as individual elements to inputs and operation by mouse and keyboard; the display of tooltips in references is also element-dependent. When processing the element list, as for instance when jumping from one input element to the next using the tabulator, the processing of all individual elements of a reference proceeds from the location of the reference in the element list before jumping to the next element of the list.

Operation over the keyboard in online mode is of greatest significance, if the visualization should be used with CODESYS HMI or web visualization. In the web visualization a specific setting in webvisu.htm allows to keep effective the operation over the keyboard even if currently an entry field is opened.

1.4.3.9.3 'File' 'Print'

'File' 'Print'is used to print out the contents of the visualization window in online mode. Visualizations which stretch over the border of the window can lead to inconsistencies particularly when there are moving elements in the visualization.

1.4.3.9.4 Access protection for multi-client operations

Due to the fact that it is possible to connect multiple visualization clients to one PLC (target and web visualization), multiple clients might access data in the PLC at the same time. If this is not desired, an access protection can be defined by certain INTERn commands, so that always only one client can write the data on the PLC.

The following INTERN commands are available for restricting the write access to one client:

REQUESTWRITEACCESS	The client requests write access on the visualization. Using the system variable CurrentWriteAccessClientId the client then can check whether it has got write access. This implicit variable stores the identification (ID) of the client which currently has write access. Each client has an unique ID "CurrentClientID".Thus the following expression can be used to allow a certain input on a visualization: CurrentWriteAccessClientId = CurrentClientId Example: The expression could be used in the configuration of an element to define its visibility : If the expression is TRUE, i.e. the client has write access, then the element is visible, i.e. allows input. The web server will check at each write request, whether the requesting client currently has write access.
RELEASEWRITEACCESS	The client deallocates its write access.
GLOBALRELEASEWRITEACCESS	The write access is deallocated client-independently.

Automatic deallocation of write access: If during monitoring it is detected that the client which has currently write access is not connected to the PLC any longer, variable CurrentWriteAccessClientId will be set to "-1", which means, that no client at all has write access. This also can be done via the application.

1.4.3.10 Visualizations in libraries

Visualizations can also be stored in libraries and thus be made available to projects in the form of library POUs. They can be inserted as references or they can be called up via the command „Zoom to vis." in the input configuration of another visualization which is part of the project.

Visualizations used in a project must have unique names. It can be problematic if for instance a visualization from a library is called or referenced which has the same name as one present in the project. Because, in processing references or visualization calls in the program, first the visualizations in the project, and only thereafter the ones in the loaded libraries will be implemented.

1.4.3.11 System variables

Implicit variables The following implicitely created system variables can be used to program a visualization:

Implicitely generated variable	Data type	Function	Currently usable in HMI	Currently usable in Simulation	Currently usable in Target Visu	Currently usable in Web-Visu
CurrentVisu	String[40]	Name of the currently opened visualization. If the name gets changed, a change to another visualization will be done. Note: With compiler versions < V2.3.7.0 the name string MUST be defined in capital letters. With compiler versions as from V2.3.7.0 the string can be defined in small letters IF the library SysLibStr.lib is included in the project. Depending on the target system this variable can be activated/deactivated in the Target Settings dialog, category Visualization.	x	x	x	x
CurrentCaller	String[40]	Name of the previously opened visualization. Is used for the ZOOMTOCALLER functionality. Only set and modified in TargetVisu.	-	-	x	-
CurrentLanguage	String[40]	Currently set language, available in the language file. The language. Only set and modified in TargetVisu.	-	-	x	-
CurrentUserLevel *	INT	Currently set user level 0..7.	x	x	x	x

Implicitely generated variable	Data type	Function	Currently usable in HMI	Currently usable in Simulation	Currently usable in Target Visu	Currently usable in Web-Visu
CurrentPasswords[0 .. 7] *	ARRAY [0..7] of String[20]	All passwords which are defined in "Usergroup passwords".	x	x	x	x
CurrentWriteAccessClientId	DWORD	ID of the visualization client which in a multi-client operation currently has the write access on the PLC data	-	-	x	x
CurrentClientId	DWORD	ID of the current visualization client	-	-	x	x

Implicit variables as remanent variables:

The implicit variables of a target visualization can be declared as remanent variables *Chapter 1.4.1.3.9.7 “Remanent variables” on page 302*:

For this purpose the variables must be declared explicitly as global variables. This declaration MUST be done in the topmost (alphabetic order) global variables list in folder 'Global Variables' in the Resources tab *Chapter 1.4.1.4.1.2 “Global variables” on page 357*. If the declaration is placed in another global variable list, a compile error will occur.

Example:

```
VAR_GLOBAL RETAIN
 VisuDoExecuteUserlevelInit : BOOL := TRUE;
 CurrentUserLevel : INT := 0;
 CurrentPasswords : ARRAY[0..7] OF STRING[20] := 'a','b','c','d','e','f','g','h';
END_VAR
```

* Regard for the variables CurrentUserLevel, CurrentPasswords[...]: Those must be of the same type (normal, RETAIN, PERSISTENT...)! If they are defined as remanent variables, additionally a variable "VisuDoExecuteUserlevelInit" of type BOOL must be declared as a RETAIN variable in the global variables list, initialized with TRUE (" VisuDoExecuteUserlevelInit : BOOL := TRUE;").

1.4.3.12 Possible key combinations for the particular visualization variants

The table below shows all possible key combinations which are supported for keyboard usage in the particular visualization variants.

The following abbreviations are used:

- C for CODESYS / CODESYS HMI
- TV for target visualization
- WV for web visualization

If an abbreviation is available in a column, the referring visualization variant supports the corresponding key combination.

Table 47: Comments for the particular lines

	no modifier	Shift	Ctrl	Shift+Ctrl	Comment
VK_TAB	C	C	C	C	K4
VK_RETURN	C / TV	C / TV	C / TV	C / TV	
VK_SPACE	C / WV	C / WV	C / WV	C / WV	K4; K5
VK_ESCAPE	C / TV / WV	C / TV / WV			K3
VK_INSERT	C / TV	C / TV	C / TV	C / TV	
VK_DELETE	C / TV	C / TV	C / TV	C / TV	
VK_HOME	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_END	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_PRIOR	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_NEXT	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_LEFT	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_RIGHT	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_UP	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_DOWN	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F1	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	K1
F2	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F3	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F4	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F5	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F6	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F7	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F8	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F9	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F10	C / TV	C / TV	C / TV	C / TV	K2
F11	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F12	C / WV	C / WV	C / WV	C / WV	
0	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
1	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
2	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
3	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
4	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
5	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
6	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
7	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
8	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
9	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
A	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
B	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
C	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
D	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	

	no modifier	Shift	Ctrl	Shift+Ctrl	Comment
E	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
F	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
G	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
H	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
I	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
J	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
K	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
L	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
M	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
N	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
O	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
P	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
Q	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
R	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
S	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
T	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
U	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
V	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
W	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
X	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
Y	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
Z	C / TV / WV	C / TV / WV	C / TV / WV	C / TV / WV	
VK_NUMPAD 0	C / TV / WV	C / TV / WV			
VK_NUMPAD 1	C / TV / WV	C / TV / WV			

K1: Additionally, the online help will be displayed

K2: In the web visualization and the IExplorer the File Menu will be focused

K3: Ctrl/Esc opens the start menu, Shift/Ctrl/Esc opens the Taskmanager

K4: Tab and Space have a different function in the target visualization

K5: Shift/Space opens the applet configuration

1.4.4 HMI

1.4.4.1 Overview

CODESYS HMI is a runtime system for the execution of visualizations which have been created with CODESYS. If a PLC program contains appropriate visualizations, those will be displayed in fullscreen mode after having started CODESYS HMI and the user can operate the control and watch functions, which are contained in the program by mouse-clicks and keyboard. This is possible, even if the Automation Builder project has got a read protection. The user has no possibility to edit the program, menues and function bars are not available, it is a pure operating version.

Thus the main control and watch functions of the project must be assigned to visualization elements which can be operated in online mode. For this purpose special input possibilities for CODESYS HMI are available in the configuration dialog of a visualization element ↳ *Chapter 1.4.3.5.30 “Special input possibilities for operating versions” on page 695*.

1.4.4.2 Installation, start and operation

Installation

CODESYS HMI can be installed with the standard Automation Builder setup. If no valid license is available, a time-limited demo version is available.

Start and operation

CODESYS HMI (*CoDeSysHMI.exe*) is started by a **command line**:

In each case at least the desired Automation Builder project has to be given in the command line. If no further parameters are set there, CODESYS HMI automatically will start with a visualization POU named PLC_VISU (if existant in the project) and on that target or simulation mode, which was set when the project had been saved last.

Additionally as well the known command line and command file commands as the following special parameters can be used:

- **"/simulation" resp. "/target"**
  Per default it will be started in that mode which was set when the project was saved last. Using the parameter "/simulation" resp. "/target" in the command line it can be set explicitely whether the project should run in simulation mode or on the target.
- **/visu <visualization POU>**
  If the project contains a visualization POU named PLC_VISU, it will start automatically with this one. If another POU should be the entrance, it has to be set in the command line with "/visu <name of visualization POU>".
- **/visudownload**
  Overrides the download lock: If the user tries to log in with a project, which is different to that on the PLC, per default a download of the new project (dialog) can be initiated. If however there is an entry "visudownload=0" in the *codesys.ini* file, then no download is possible. This lock can be overridden by the command line parameter "/visudownload".
- **/visucompactload**
  This parameter can be used for optimization at starting a project for which no download is required. If a download gets necessary anyway, e.g. effected by parameter "/visudownload", "/visucompactload" will be ignored.

Example for a command line

```
D:\PROGRAMME\CoDeSysHMI /simulation D:\PROJECTS\PROJECT.PRO /visu
overview
```

The project *project.pro* will start in simulation mode and with the visualization POU 'overview'.

Paths containing spaces must be bordered by quotation marks (").

The project will start in full screen mode with the entrance POU.

CODESYS HMI can be operated corresponding to the functions of the visualization elements via keyboard and mouse. If there is no visualization element configured with a corresponding function, CODESYS HMI at any time can be terminated by pressing *[Alt]* + *[F4]*.

1.4.5 Web visualization

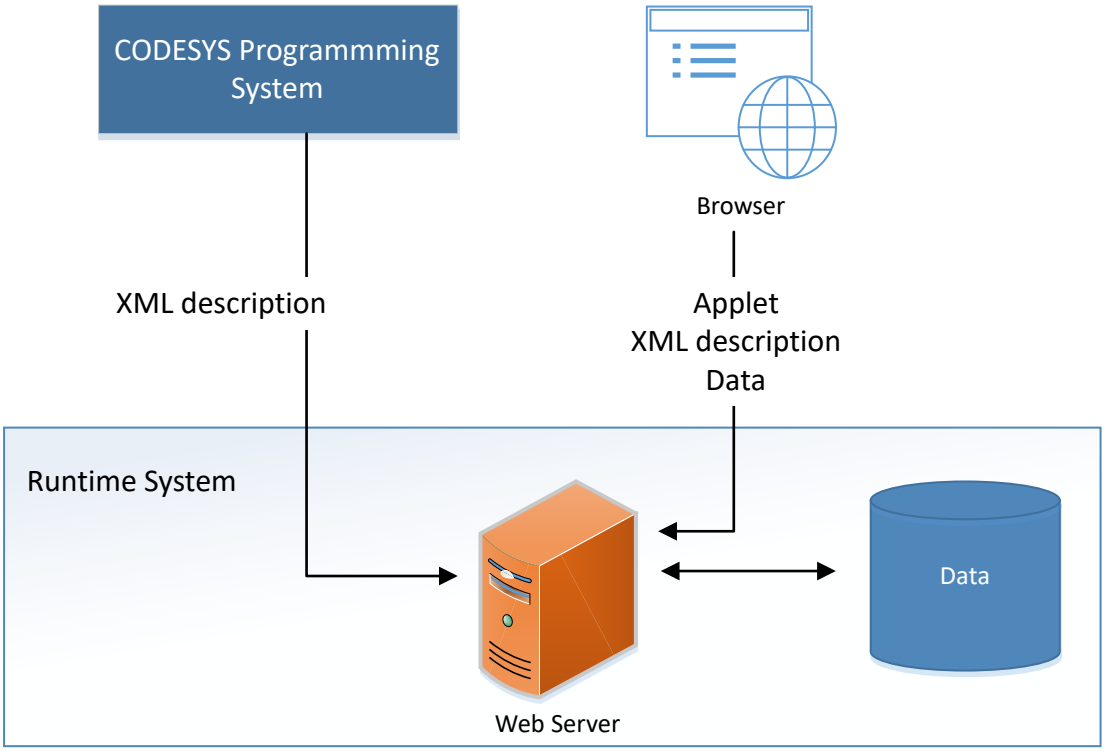
1.4.5.1 Overview

The web visualization is a target specific application of a visualization.

CODESYS can create XML descriptions of the visualization objects and download them to the PLC. There a web server will provide the PLC data in XML format too and thus can create a continuously updated visualization which can be opened in the web browser of any computer which is connected via Internet, independently from the target platform (e.g. useful for remote maintenance purposes).

Regard the Restrictions and Special Features when programming for a web visualization.

The web server can connect dynamically to several controllers if configured appropriately. Visualization elements can be configured in a way that they will effect a connection change to another target system.



Before a download regard the current setting of option 'Prevent download of visualization files' in the target settings. This concerns all files which are in the directory for visualization files!

1.4.5.2 Preconditions

In order to make a visualization created in CODESYS available as web visualization, the following preconditions must be fulfilled:

- the target system must support the functionality; that means that in the target settings the option 'web visualization' must be activated. If defined in the target file, this can be done by the user in the 'General' target settings.
- A correctly configured web server must be started. (see below)
- In order to get the web visualization displayed on a computer, a Web browser (IE-Explorer oder Netscape) is required.

- Operating system: Windows NT/2000, Windows CE, Linux, RTE
- Regard when using a HTTP-Proxy-Server: If the communication within the Internet browser is done via a HTTP-Proxy-Server, the monitoring of the visualization data might cause problems. If "USEFIXSOCKETCONNECTION" in WebVisu.htm is set TRUE, the telegram format used for the process data communication between applet and web server will not match the HTTP standard. In this case the HTTP Proxy will not forward the telegrams to the web server.

  A workaround is to configure a SOCKS-Proxy:.Clients behind a firewall which want to establish a connection to an external server, instead have to connect to a SOCKS-Proxy. This Proxy-Server checks whether the client is authorized to communicate with an external server and forwards the requests transparently to the server.

  The SOCKS protocol is an Internet-Proxy protocol, allowing client-server applications to transparently use the services of a firewall. SOCKS is an abbreviation for "SOCKets".

Before downloading the project regard the current setting of option 'Prevent download of visualization files' in the target settings. This concerns all files which are used in the current visualization. Those can be bitmaps, language files and for web visualization also XML description files.

1.4.5.3 Editing the WebVisu.htm file

During installation of Automation Builder the file WebVisu.htm (base-HTML-page for the web visualization) gets copied to the subdirectory "visu" in the installation directory. This is the basic html-page for the web visualization. Before it gets downloaded to the target system together with the web visualization project, it can be edited in a text editor in order to change the behavior of the web visualization:

Webvisu.htm per default contains the following:

```
<HTML>

<HEAD>

   <TITLE>Applet HTML Page</TITLE>

</HEAD>

<BODY>

<APPLET CODEBASE=. CODE=webvisu/WebVisu.class
archive="webvisu.jar,minml.jar" name="WebVisu" width="1600"
height="1200">

<param name="STARTVISU" value="PLC_VISU">

<param name="UPDATETIME" value="100">

<param name="USECURRENTVISU" value="FALSE">

</APPLET>

</BODY>

</HTML>
```

The following parameters are of meaning for an adaption of the behavior of the web visualization:

width, height	Definition of the size of the screen. Regard the possibility to make visible this size already during creating a visualization (Target Settings: Display width/height in pixel).
STARTVISU	Definition of the start POU (Default: PLC_VISU)
UPDATETIME	Definition of the monitoring interval (ms)
USECURRENTVISU	Definition whether an automatic change to another visualization will be done, as soon as the system variable 'CurrentVisu' is changed by the PLC program. For information on implicit (system) variables in the visualization see 'Implicit Variables' *Chapter 1.4.3.11 “System variables” on page 717*).

Optionally the file can be extended by the following entries in the APPLET definition part:

<param name="USEFIXSOCKETCONNECTION" value="FALSE or TRUE">	If this parameter is TRUE, a fix socket connection will be used for monitoring; if it is FALSE or if the entry is missing at all, for each monitoring request a new socket will be used. Default: FALSE. Attention: If parameter USEURLCONNECTION (see below) is used, USEFIXSOCKETCONNECTION may not be set TRUE.
<param name="FORCEDLOAD" value="Comma-separated list of visualization names">	The visualizations specified here will be loaded already when the web visualization is loaded, not just when they are opened for the first time. Thus time is saved at later changes of visualizations, because then the data not have to be transferred first by the web server. Examples: <param name="FORCEDLOAD" value="VISU_1, VISU_2, VISU_3 "> <param name="FORCEDLOAD" value="TREND">
<param name="COMPRESSEDFILES" value="FALSE or TRUE">	The files to be transferred for the web visualization to the web server can be provided in a packed format ("<filename>_<extension original format>.zip"). See 'Target settings in category visualization' *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*. If parameter "COMPRESSEDFILES" is set "TRUE", the web visualization first will try to load from the web server all needed files which have got appended the extension ".zip"and to unzip those. If the request for a zip-file fails, it will be tried to load the original file. If parameter "COMPRESSEDFILES" is not available or is configured with value FALSE, any available zip-files will be ignored.

<param name="USEURLCONNECTION" value="URL">	If this parameter is available, the communication will be done via the specified URL-connection. Per default a simple socket connection is used. Attention: If parameter USEFIXSOCKETCONNECTION (see above) is set TRUE, USEFIXSOCKETCONNECTION may not be used additionally. Example: <param name="USEURLCONNECTION" value="http://192.168.100.19:8080/webvisu.htm"
<param name= "PLCSTATEINTERVAL" value="Time interval for cyclic status requests">	The status of the PLC will be requested according to this time interval (milliseconds). Default: 2000. See also: Status check, AutoReload, File error ini.xml.
<param name="FORCEDLOAD" value="comma-separated list of visualization names">	The visualizations specified here will be loaded already when the web visualization is loaded, not just when they are opened for the first time. Thus time is saved at later changes of visualizations, because then the data not have to be transferred first by the web server. Examples: <param name="FORCEDLOAD" value="VISU_1, VISU_2, VISU_3 "> <param name="FORCEDLOAD" value="TREND">
<param name="COMPRESSEDFILES" value="FALSE or TRUE">	The files to be transferred for the web visualization to the web server can be provided in a packed format ("<filename>_<extension original format>.zip"). See 'Target settings in category visualization' *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*. If parameter "COMPRESSEDFILES" is set TRUE, the web visualization first will try to load from the web server all needed files which have got appended the extension ".zip"and to unzip those. If the request for a zip-file fails, it will be tried to load the original file. If parameter "COMPRESSEDFILES" is not available or is configured with value FALSE, any available zip-files will be ignored.
<param name="SELECTION" value="Line width|RED|GREEN|BLUE">	Here the line width and color for the display of the current selection can be defined. Syntax: LINEWIDTH|RED|GREEN|BLUE; Example: "4|0|0|255"
<param name="ERROR_SENSITIVITY" value="Number of trials to get a file transferred">	This parameter defines, how many trials will be done to get a visualization file transferred from the web server, before an applet error will appear.

<param name="KEYPADINDIALOGS" value="FALSE or TRUE">	If a touch screen is used for working with the web visualization, this parameter should be set TRUE in order to get an input possibility in any case for each dialog; if applicable via numpad/keypad.
<param name="KEYBOARDUSAGEFROM-DIALOGS" value="FALSE or TRUE">	If this parameter is set TRUE, the keyboard usage is always active, even if a modal dialog - like e.g. the numpad - is currently opened.
<param name="WRITEACCESSLOCK" value="FALSE or TRUE">	This parameter only should be set TRUE, if the web server supports multi-client processing and if an access lock for various clients is desired. Concerning access control in multi-client operation please see 'Access protection for multi-client operations' *Chapter 1.4.3.9.4 “Access protection for multi-client operations” on page 716*.
<param name="DEFAULTENCODING" value="FALSE or TRUE">	If this parameter is set TRUE and the language switching is done via ASCII language files, the default encoding - currently set in the system - will be used for the interpretation of the language file.
<param name="ENCODINGSTRING" value="encoding string">	If the default encoding of the system is not set as desired, you can define here the desired encoding by entering the appropriate string. Examples for encoding strings: German: "ISO-8859-1" Russian: "ISO-8859-5" Japanese: "MS932"
<param name="PLCSTATEINTERVAL" value="Zykluszeit">	Cycle time in milliseconds, according to which the web client will check the PLC status. It will be checked whether the PLC is in Start or Stop status and whether a download has been done.
<param name="ALARMUPDATEBLOCKSIZE" value="Number of alarm states to be updated per cycle">	This parameter can be set in order to change the update of the alarm states. Due to the fact that not all alarm states can be updated within one cycle, it might be useful to exactly defined the number of alarms which should be updated per cycle. This number can be specified as a numeric value.
<param name="SUPPORTTOOLTIPSINA-LARMTABLE" value="TRUE oder FALSE">	If this parameter is set TRUE, the tooltip functionality in the alarm table will be activated. This means: If any text entry in the alarm table cannot be displayed completely, a tooltip will be available showing the complete text string as soon as the mouse pointer is moved on the respective table cell.
<param name="TOOLTIPFONT" value="Schrifttyp|Schriftgröße">	This parameter serves to define the font for all tooltips. Syntax: Font|Size; Example: "Arial|11".
<param name="FILEOPENSAVEDIALOG-FONT" value="Schrifttyp|Schriftgröße ">	This parameter serves to define the font for the File-Open-dialog. Syntax: Font|Size; Example: "Arial|11".

<param name="ALARMTABLEFONT" value="NAME|HEIGHT|WEIGHT|CHARSET|ITALIC|HORZ_ALIGN|VERT_ALIGN">	This parameter serves to define the font for the alarm table. Syntax: NAME|HEIGHT|WEIGHT|CHARSET|ITALIC|HORZ_ALIGN|VERT_ALIGN Example: "Arial|11|0|0|false|left|center".
<param name="USECURRENTLANGUAGE" value="TRUE oder FALSE">	If this parameter is set TRUE, the current language setting always will be synchronized between web and target visualization (via implicit variable CurrentLanguage); i.e. at a language switch caused by an input in one of the both visualization types each in the other type the language will be switched too. The visualization currently is not included in this match.

1.4.5.4 Status check, auto-reload, file error_ini.xml

Status changes of the PLC or the visualization project are notified as messages in an error_ini.xml file. Usually in this case communication errors, start/stop of the PLC, breakpoint states, new download etc. will be reported in message boxes. The ini-file however also might define error messages like "page cannot be displayed", "too many variables to be monitored" or the like. Also as a reaction on a download or online change of the project an automatic reload of the web visualization might be defined.

The time interval for a cyclic status check can be defined in the file WebVisu.htm (PLCSTATEINTERVAL).

A communication failure during the monitoring of the web visualization will not be dumped as an error message, but will be displayed in the status line of the web browser.

1.4.5.5 Preparing a web visualization

- Create the visualization(s) for your PLC program as usual in CODESYS. If you want a certain visualization to be called as starting object, then name it 'PLC_VISU'. It will be loaded automatically as soon as the visualization is called via the Internet. Regard the effects of the currently active target settings, category Visualization.
- For visualization objects, which should not be part of the web version of the visualization, deactivate option web visualization' in the dialog 'Object' 'Properties' in category 'Visualization'.
- Regard the possibility to use visualization elements for the purpose of switching between several controllers, to which the web server then will connect automatically.
- If needed, modify the basic html-page ↯ *Chapter 1.4.5.3 “Editing the WebVisu.htm file” on page 723*. For example instead of PLC_VISU another visualization can be defined as start page of the web visualization.
- Perform command 'Project' 'Clean all', then 'Project' 'Build'. Before downloading the project regard the current setting of option in the target settings ↯ *Chapter 1.4.1.4.7.3 “Target settings in category visualization” on page 388*. This concerns all files which are used in the current visualization. Those can be bitmaps, language files and for web visualization also XML description files.
- Log in to the target system ('Project' 'Online' 'Login') and start the project on the PLC.

1.4.5.6 Configuration and start of the web server

- The web server must be available as an appropriate executable file (*.exe) for the used target system. It also can be installed and started as a sevice. The following command line parameters can be used for this purpose when webserver.exe is started:
  - i = web server gets installed as a service
  - u = web server service gets uninstalled
  - s = web server service gets started
  - e = web server service gets terminated
- The configuration of the server can be done by a configuration file or - restricted usable - by parameters added in the command line when calling the server-exe. The definitions given by a configuration file will overwrite those of the command line.

  The possible parameters:

webserver-port-nr	Port, on which the web server expects requests of the client (web browser)	Default: 80
target-port-nr	Port of the runtime system	Default: 1200
target-ip-address	IP address of the runtime system	Default: localhost
use-file-upload-dir	If this flag is set to TRUE, additionally the file-upload-directory (see below, file-upload-dir) must be defined, where the *.xml, *.bmp, *.jpg, etc. files of the web visualization are stored on the target system and from where they are uploaded to the browser.	Default:false
file-upload-dir	Directory for the web visualization files	Default:""
use-intel-byte-order	Type of byte-order (true or false): if "false, the data will get swapped (Motorola Byte-Order).	Default: true
The following entries refer to the use of a description file for the PLCHandler in order to be able to use the MultiPLC-functionality. If these entries are missing, automatically the TCP/IP-connection to the above specified target system will be used.		
plc-description-file	Path of the ini-file for the PLCHandler, describing the communication parameters for all PLCs (controllers) to which the web server then can connect via Gateway/PLCHandler. (Per default the ini-file is in the same directory as web server.exe). If the visualization (web client) via INTERN-command CONNECT_TO tells to which PLC a connection should be established, this connection will be built automatically by the server and will be maintained until the next change.	Default: ""

plc-entries	Section containing the entries <plc-entry> for the various PLCs	Default: ""
plc-entry	Entry for a PLC, contains PLC name and directory	Default: ""
plc-name	Name of the PLC, as defined in the PLCHandler.ini-file	Default: ""
plc-directory	Directory of the PLC files; can be defined absolutely or relatively to the file-upload directory (see above). Example: For the example configuration shown below a definition ".\FD" for the plc-directory results in the following location: "C:\Programme\codesysV23\FD"	Default: ""

If you define a "file-upload" directory, where the controller can store the visualization download files, then the visualization files will be automatically updated at each download. The advantage of this upload-directory is that the controller is not involved and thus not strained. The web server gets the files directly from the directory and by this the data transfer is much quicker. This is especially of impact in case of a big amount of data.

A configuration file for the web server must be available in XML-format and it must be named "webserver_conf.xml". It must be found in the directory where the webserver-exe is. If no configuration file is available, the above mentioned default settings will be used (if not changed by parameters added to the call of the server in a command line.)

Example configuration in the webserver_conf.xml file

```
<webserver-configuration>
    <webserver-port-nr> 8080 </webserver-port-nr>
    <target-port-nr> 1200 </target-port-nr>
    <target-ip-address> localhost </target-ip-address>
    <use-file-upload-dir> true </use-file-upload-dir>
    <file-upload-dir> C:\Programme\CoDeSysV23\ </file-upload-dir>
    <use-intel-byte-order> true </use-intel-byte-order>
    <plc-description-file> PlcHandler.ini </plc-description-file>
    <plc-entries>
        <plc-entry>
            <plc-name> MASTER </plc-name>
            <plc-directory> .\MASTER </plc-directory>
        </plc-entry>
        <plc-entry>
            <plc-name> FD </plc-name>
            <plc-directory> .\FD </plc-directory>
        </plc-entry>
        <plc-entry>
            <plc-name> DL </plc-name>
            <plc-directory> .\DL </plc-directory>
        </plc-entry>
     </plc-entries>
</webserver-configuration>
```

A call in a command line must use the following syntax:

```
WebServer [webserver-Port-nr] [target-port-nr] [target-IP-address] |
[file-upload-dir]
```

Only these parameters can be used in a command line. The others described above, concerning byte-order and multiPLC-functionality must come via the configuration file.

Thus a call corresponding to the above shown configuration example would look like follows, whereby the settings for <plc-description-file> and <plc-entries> must be available in a configuration file:

```
> webserver 8080 1200 localhost c:\Programme\codesysV23
```

The parameter settings in the command line are without any effect, if there is a valid configuration file available.

1.4.5.7 Calling a web visualization via internet

Insert the following address in the browser:

```
http://<IP-Adresse of the web server>:<Port of the web server>/
webvisu.htm
```

Example: `http://localhost:8080/webvisu.htm`

WebVisu is the default HTML-file. It contains a <applet> tag which will start the WebVisu-Applet so that the desired start page of the visualization will be displayed. Now you can start to operate the visualization.

If the browser accepts cookies (if necessary, activate this option) the current language setting for dynamic texts will be stored and at the next call of the web visualization will be re-used automatically. If the stored language is not available in the newly opened visualization, the default language of this visualization will be used. If no default language is defined, the visualization texts will not be translated until the language gets switched explicitly.

1.4.5.8 Restrictions and special features

Table 48: INTERN commands

PRINT	Printout of the current visualization is not supported for web visualization.
Execute external program	This command for the execution of an external program is not supported for web visualization.
LANGUAGEDIALOG	The command for calling the configuration dialog containing the category 'Language' is not supported for web visualization.
EXITPROGRAM	The command for exiting the program is not supported for web visualization. Can be realized via command INTERN LINK.
TRACE	The command for opening the Sampling Trace window is not supported for web visualization. This function will be taken by the Trend element.
SAVEPROJECT	The command for saving the project is not supported for web visualization.

↳ *Chapter 1.4.3.5.30 “Special input possibilities for operating versions” on page 695*

Table 49: Accessing variables

Dynamic indexing within an array ↳ *Chapter 1.4.1.8.2.1 “ARRAY” on page 445*	"Array1[Index].a" is not possible, however "Array1[10].a" is possible.
Replacing placeholders containing an expression	Placeholder: abc + 5 Replacement: PLC_PRG.n + 500 Should result in PLC_PRG.n + 500 + 5, this however is not possible in the web visualization.
Pointer variables ↳ *Chapter 1.4.1.8.2.3 “Pointer” on page 447*	Pointer variables like PLC_PRG.pdw2^ cannot be monitored.

Table 50: Others

Transparent bitmaps	Transparent bitmaps currently are not supported.

Table 51: Alarm handling

Actions	Action 'Print' currently is not supported.
Settings for sorting (history)	The sorting within the alarm table, displayed via button 'History', always is according to date. The settings as done in the configuration of the alarm table element are not regarded.
IEC operators ↳ *Chapter 1.4.1.6 “IEC operators and additional, norm-extending functions” on page 407*	The following IEC operators currently are not supported by the web visualization: Arithmetic operators: ● MOVE ● INDEXOF ● SIZEOF Bitstring operators: ● XOR Bitshift operators: ● SHL ● SHR ● ROL ● ROR Selection operators: ● LIMIT ● MUX Comparison operators: ● EQ ● NE All address operators All calling operators All conversion operators Numeric operators: ● SQRT ● LN ● LOG ● EXP ● ASIN ● ACOS ● ATAN ● EXPT

The following features are only suported by the web visualization:

Table 52: INTERN commands

INTERN LINK	Via "INTERN LINK <URL>" a change to another web page can be defined. Example: "INTERN LINK http://www.3s-software.com" "INTERN LINK <webaddress file>" opens a PDF file, which must be available on the server. Example: "INTERN LINK http://localhost:8080/test.pdf " "INTERN LINK <Email address>" opens a window for sending an Email. Example: "INTERN LINK mailto:support@3s-software.com"
INTERN CONNECT_TO <PLC-Name>|<Start-Visu>	The target PLC can be changed if the web server is configured appropriately with the connection parameters for multiple PLCs (see 'Configuration and start of the web server' *Chapter 1.4.5.6 “Configuration and start of the web server” on page 728*). PLC-Name: Name of the PLC, as defined in the ini file of the PLCHandler. Start-Visu: Name of the visualization which should be displayed at start of the web visualization. As soon as the web server gets the request for a PLC-change, it will automatically establish the connection to the respective PLC.

Chapter 1.4.3.5.30 “Special input possibilities for operating versions” on page 695

Language for dynamic texts

If the browser accepts cookies (if necessary, activate this option) the current language setting for dynamic texts will be stored and at the next call of the web visualization will be re-used automatically. See the note in 'Calling a web-visualization via internet' *Chapter 1.4.5.7 “Calling a web visualization via internet” on page 730*.

Using a HTTP-proxy-server

If a HTTP-Proxy-Server is used, additionally a SOCKS-Proxy-Server might be necessary in order to avoid monitoring problems. See 'Preconditions' *Chapter 1.4.5.2 “Preconditions” on page 722*.

1.4.6 License manager

1.4.6.1 Overview

The license manager

The *License Manager* is available to handle the licenses for modules, as well as licenses for modules for which an appropriate license information file is provided, on your computer. You can create a project and provide it as a licensed library. The Licensing Manager will be installed automatically with any module, which requires a license *Chapter 1.4.6.2 “Creating a licensed library” on page 734*.

1.4.6.2 Creating a licensed library

A project can be saved as a library. If you want to create a licensed library you have to add the appropriate license information. For this perform the command *“File ➔ Save as”*, choose data type 'Internal Library' or 'External Library' and press button *[Edit license info]*↳ *Chapter 1.4.1.2.3.6 “'File' 'Save as'” on page 224*.

In the dialog *“Edit Licensing Information”* enter the information described below. The license information will be added to the *“Project Info”*↳ *Chapter 1.4.1.2.3.34 “'Project' 'Project info'” on page 246*.

When later on the library will be included in a project, the license information can be checked up in the object properties dialog of the library in the*“library manager”* ↳ *Chapter 1.4.1.4.3.1 “Overview” on page 371*.

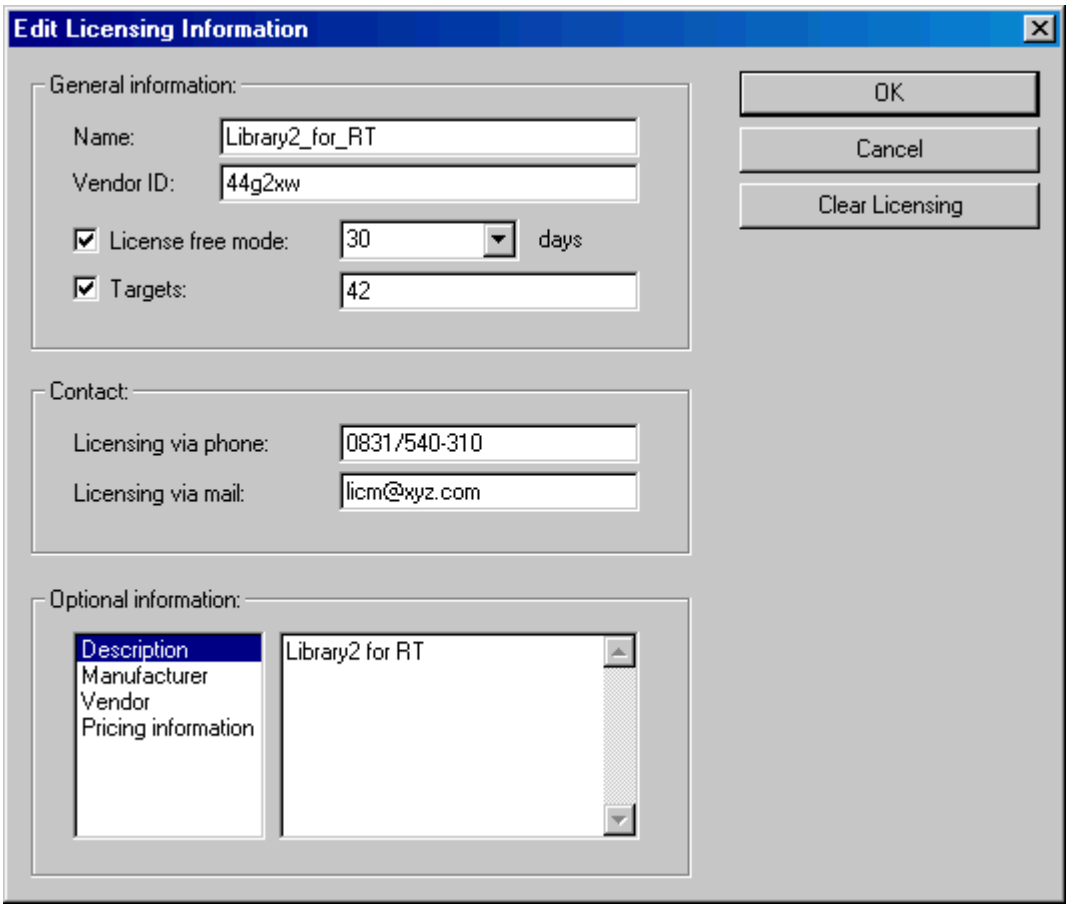


Fig. 12: Edit licensing information

Name	Enter a name for the library module which is used to represent it in the *Licensing Manager*. This input is mandatory.
Vendor-ID	A manufacturer identifier, depending on the manufacturer specific licensing management tool.
Targets	Enter here the target ID(s) of the target system(s) for which the license should be valid. You can enter multiple IDs separated by a semicolon or as a range. Example: "12;15-19;21"; herewith the IDs 12,15,16,17,18,19,21 are entered.
Licensing via phone / mail	Insert here the phone number resp. email address of the license provider. These inputs are mandatory.
Optional information	In the right window you can enter a text referring to the item currently marked in the left window: *“Description”*, *“Manufacturer”*, *“Vendor”*, *“Pricing information”*

It is reasonable to protect a library, which has been provided with licensing information, by a password↳ Chapter 1.4.1.2.2.10 “Passwords” on page 211. If you are going to save the project without password you will be pointed to that by a message box.

The licensing information of a library is stored internally with the library and will be registered on the computer automatically as soon as the library is included in a project. The license information of modules which are not provided must be provided in a separate description file in compatible XML format, which can be read by the Licensing Manager.

1.5 Libraries and solutions

1.5.1 Information on libraries

System libraries

When upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically. Older library versions will be removed as coexistence of a new library version and an older library version is not possible. Check the available library version in the Library Manager.

Usually, when upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically and older library versions are removed.

As an exception, for the CANopen device CM598-CN both library versions are available in the Library Manager due to compatibility reasons. However, coexistence of a new library version and an older library version is not possible. In order to avoid compile errors remove the older library version.

↳ *Chapter 1.2.17 “Converting an AC500 V2 project to an AC500 V3 project” on page 67*

Customer libraries

Target change from AC500 V2 to AC500 V3

After a target change from AC500 V2 to AC500 V3 the customer libraries have to be converted manually using the Library Converter . For further information see ↳ *Chapter 1.6.5.1.6 “Later change-over of a target system” on page 5782*.

Some Standard CODESYS libraries are automatically converted during the target change.

1.5.2 Reference to CODESYS (V2)

Note that CODESYS libraries are used. For further information see ↳ *Chapter 1.4.1 “Development system” on page 145*.

1.5.3 Error messages of the AC500 V2 function block libraries

1.5.3.1 0000hex...0FFFhex - telegram error

DEC	HEX	Error description
0	0000	No error
1	0001	COM_MOD_MAST: Error message from slave ILLEGAL FUNCTION ETH_MOD_MAST:Error message from slave ILLEGAL FUNCTION

DEC	HEX	Error description
2	0002	COM_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS ETH_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS
3	0003	COM_MOD_MAST: Error message from slave ILLEGAL DATA VALUE ETH_MOD_MAST: Error message from slave ILLEGAL DATA VALUE
4	0004	COM_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE ETH_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE
5	0005	COM_MOD_MAST: Error message from slave ACKNOWLEDGE ETH_MOD_MAST: Error message from slave ACKNOWLEDGE
6	0006	COM_MOD_MAST: Error message from slave SLAVE DEVICE BUSY ETH_MOD_MAST: Error message from slave SLAVE DEVICE BUSY
8	0008	COM_MOD_MAST: Error message from slave MEMORY PARITY ERROR ETH_MOD_MAST: Error message from slave MEMORY PARITY ERROR
9	0009	COM_MOD_MAST: Error message from slave SEE SLAVE DESCRIPTION ETH_MOD_MAST: Error message from slave SEE SLAVE DESCRIPTION
10	000A	COM_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE ETH_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE
11	000B	COM_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND ETH_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND
4095	0FFF	FLASH_READ, FLASH_WRITE FLASH_DEL while ERR=FALSE: Block execution is in process

1.5.3.2 1000hex...1FFFhex - device error

DEC	HEX	Error description
4097	1001	Device does not exist
4098	1002	Command not supported by the device. The function is not supported by the device firmware/hardware. Block library newer than the device firmware. FC..: No high-speed counter available at the given module.
4100	1004	Error operating mode. FC..: Operating mode "0" -> No counter set in the PLC configuration.
4101	1005	Invalid status FLASH_READ: Block is not written yet FLASH_WRITE: Block was already written RETAIN..: No program loaded Library PROFINET IO: PNIO_WRITE, PNIO_READ: Internal error. Restart the PLC.
4117	1015	Format error SD..: File cannot be read because of an invalid format. Data could not be read or not be read completely.

DEC	HEX	Error description
4119	1017	Incorrect length PERSISTENT..: Data have an incorrect length RETAIN..: Data have an incorrect length
4120	1018	Checksum error
4122	101A	FC..: Internal error (e.g. no CS31 Adr parameter found, wrong size of CS31 Adr) HA..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in configuration found)
4123	101B	Device access error Flash..: Resources are not available HA..: Remote CPU failure: Other CPU is off or out of order PERSISTENT..: Data could not be copied, access error or no data do exist RETAIN..: Data could not be copied, access error or no data do exist SD..: Access to the memory card is not possible (e.g. memory exhausted, file already opened, etc.
4124	101C	Incorrect number PERSISTENT..: Because of the current CPU parameters, data are loaded only partly
4127	101F	Access protection SD..: Memory card is write protected
4128	1020	Error when opening SD..: Error when opening a file stored on the memory card
4129	1021	Not found FC..: CS31 Bus Module not found EtherCAT Modul not found HA_CS31_CONTROL: Own CI590-CS31-HA slave failure (missing module) HA_CS31_DIAG: One or more CI590-CS31-HA slave(s) is/are inactive HA_CS31_DIAG_VIA_CM574-RS: One or more CI590 are inactive SD..: The searched sector could not be found in the file TASK_INFO: Unknown task
4130	1022	End reached SD..: Section end or end of file reached PERSISTENT..: Because of the current CPU parameters, data are not loaded
4131	1023	Reading error FLASH..: Reading error in data segment: Incorrect checksum
4132	1024	Writing error FLASH..: Block cannot be programmed SD..: File could not be deleted or written

DEC	HEX	Error description
4137	1029	FC..: Wrong configuration (e.g. no CM574-RS configured on specified slot) HA..: Remote CI590-CS31-HA slave failure (missing module)
8191	1FFF	Not ready Flash..: The command is already executed by an other instance SD..: Command cannot be executed. Another instance is already active.

1.5.3.3 2000hex...2FFFhex - interface error

DEC	HEX	Error description
8193	2001	Invalid interface, Communication Module number or slot ID COM: Interface is not configured in the "free mode" HA: Wrong COM number at input COM ETH-MOD-MAST: Invalid interface or slot ID
8194	2002	Command not supported by the interface. The function is not supported by the device firmware. Block library newer than the device firmware.
8195	2003	Invalid interface or Communication Module type. Block is not suitable for this type.
8197	2005	HA..: CS31 Bus failure
8198	2006	Fault on both local and remote CPU.
8211	2013	Timeout COM_MOD_MAST: Slave did not respond within the specified time HA..: No Ethernet link, Error in DPRAM communication between CM574-RS and AC500 CPUs. ECAT_COE_READ, ECAT_COE_Write: Response timeout occurs
8212	2014	Framing error (incorrect transmission rate, number of stop bits and/or bits per character)
8213	2015	Parity error HA: Remote CS31 slave sync error. Difference in digital/ analog output buffer of PLC A and B
8214	2016	Idle error COM..: Character timeout occurred
8215	2017	Invalid length COM_MOD_MAST: Invalid data length received COM_REC: Received more data than expected
8216	2018	Checksum error
8217	2019	Handshake error

DEC	HEX	Error description
8218	201A	Service failed COM_REC: Unknown error message of the interface COM_SET_PROT: Interface hardware not accessible. Initialization already failed during system start-up. COM_MOD_SLV_SET_ADDR..: Internal error (e.g. wrong return value of internal function, null pointer received) CS31..: Internal error (e.g. CS31 communication failed) FC..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in config found) HA..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in configuration found)
8219	201B	Access error HA..: Remote CS31 Bus failure: Other CPUs' CS31 Bus is out of order IO..: Module number does not exist
8220	201C	Incorrect number (quantity) COM_SET_PROT: Invalid protocol index. Index is not supported by the device firmware. COM_MOD_SLV_SET_ADDR..: Invalid protocol index. HA..: Numbers of CI590-CS31-HA devices configured in line A and line B are different IO..: Invalid module number
8223	201F	Access denied COM..: Access to the interface is not possible at present. Automation Builder, OPC or another program is logged in via the interface.
8226	2022	COM_MOD_SLV_SET_ADDR..: Wrong configuration (No protocols configured). FC..: Wrong configuration (e.g. no Automation Builder configuration, no CS31 configured, no CS31 modules found). HA..: Overflow of HA_DATA reference table
8233	2029	● HA_CS31_CONTROL: – CI590-CS31-HA slave configuration not complete – CI590-CS31-HA slaves in Bus1 and Bus2 are mix-wired – Remote CI590-CS31-HA Failure – CS31 Master Cross Wired / No CS31 configuration / No submodules on CS31 bus ● HA_CS31_DIAG/ HA_CS31_DIAG_VIA_CM574-RS – CI590-CS31-HA slaves in line A and line B are mix-wired.
8234	202A	HA: configuration error: ● Wrong CM574-RS communication ● CM574-RS COM interface is not configured for shared communication ● CM574-RS not added to Communication Module slot ● Wrong configuration of DIAG blocks

1.5.3.4 3000hex...3FFFhex - protocol error

DEC	HEX	Error description
12289	3001	Unknown protocol or protocol not configured. MqttClient: The Network Connection has been made but the MQTT service is unavailable on the specified port.
12290	3002	Command not supported by the protocol. The function is not supported by the device firmware. Block library newer than the device firmware.
12291	3003	Another protocol is configured. FC_DC551: The selected interface (COMx) is not set to the CS31 protocol. COM_MOD_SLV_SET_ADDR..: The selected interface (COMx) is not set to the Multi protocol. COM_MOD_SLV_SET_ADDR..: Wrong protocol index (No Modbus protocol on specified index). HA: No CS31 protocol at COM. BACnet B-ASC: Multiple function block instances, one instance allowed only.
12292	3004	Operating mode error COM_MOD_MAST: Invalid operating mode (master/slave).
12293	3005	Protocol status error Fieldbus Communication Module .._SYS_DIAG: Master is not in the OPERATE state. ETH_SMTP_EMAIL_SEND: Internal error (ulHandle wrong). BACnet B-ASC: No device object instantiated.
12307	3013	IEC..: ACTCON timeout ETH_SMTP_EMAIL_SEND: Server timeout MqttClient: The timeout value for the communication has been exceeded.
12308	3014	IEC..: NACK received
12309	3015	ETH_SMTP_EMAIL_SEND: Syntax error in mail address
12310	3016	IEC..: Timeout
12311	3017	Incorrect length ARC..: Buffer is full CAN2..: Total length of all messages too high IEC..: Queue overrun ETH_UDP..: Buffer is full. MqttClient: Received topic or payload is too long.
12313	3019	IEC..: ACTERM timeout Wait answer ETH_SMTP_EMAIL_SEND: Could not connect to server. Not reachable or does not answer.

DEC	HEX	Error description
12314	301A	Execution failed COM_SET_PROT: Initialization of the protocol failed IEC..: Send failed due to queue deleted COM_MOD_SLV_SET_ADDR..: Internal error (e.g. null pointer received, wrong return value of internal function) ETH_SMTP_EMAIL_SEND: Internal error. Mail not sent. (e.g. out of resources) ETH_ICMP_PING: Target Host did not answer the ping echo request before the specified timeout MqttClient: MQTT broker did not answer the ping. MQTT client has passed the KeepAlive or MQTT broker is unreachable.
12315	301B	Access error ARC..: Buffer does not exist / is not specified CAN2..: Buffer does not exist / is not specified ETH_UDP..: Buffer does not exist / is not specified IEC..: Protocol access error IO..: There is no module in the selected slot
12316	301C	Wrong number IO..: Invalid module number > max.
12319	301F	Access denied COM..: Access to the interface is not possible at present. Automation Builder, OPC or another program is logged in via the interface. MqttClient: The Client identifier is correct UTF-8 but not allowed by the Server.
12320	3020	Error when opening ARC..: Error during protocol initialization. Protocol not yet ready. CAN2..: Error during protocol initialization. Protocol not yet ready. IEC..: Data type mismatch for this PIN when data arrived. ETH_UDP..: Error during protocol initialization. Protocol not yet ready. ETH_SMTP_EMAIL_SEND: ● File for attachment cannot be opened ● Server not ready (wrong port configured?) MqttClient: The Server does not support the level of the MQTT protocol requested by the Client.
12321	3021	ETH_SMTP_EMAIL_SEND: File for attachment not found CPU_PROD_ENTRY_READ: Entry not found

DEC	HEX	Error description
12325	3025	Address error COM_MOD_MAST: Receive telegram does not contain the expected register address. ETH_MOD_MAST: Cannot connect to server (IP address) OR response telegram contains wrong UNIT_ID (must be the same as in request). ETH_SMTP_EMAIL_SEND: Internal error (Could not bind sockets). ETH_UDP_REC: Wrong INDEX for connection. ETH_UDP_SEND: Wrong INDEX for connection or IP Destination does not contain a valid IP address. MqttClient: Connection refused, maybe the IP address is malformed.
12326	3026	Function error COM_MOD_MAST: The received FCT does not correspond to the sent FCT.
12327	3027	Invalid value COM_MOD_MAST: Receive telegram contains an unexpected value. MqttClient: Network error on MqttSubscribe/MqttUnsubscribe. Maybe the topic is not valid.
12328	3028	ETH_SMTP_EMAIL_SEND: Out of sockets. Mail not sent (resource starvation). ETH_ICMP_PING: Could not create internal Task (not enough resources). BACnet B-ASC: Protocol task start failed.
12329	3029	HA..: CI590-CS31-HA slave configuration is not complete.
12331	302B	Unspecified error MqttClient: Internal library returned an unspecified error.
12333	302D	No connection, sending not possible due to no connection, either closed or not established yet MqttClient: No connection to an MQTT Broker.
12540	30FC	ETH_SMTP_EMAIL_SEND: Mailbox unavailable or not found on target server. SMTP error by server.
12788	31F4	ETH_SMTP_EMAIL_SEND: Syntax Error. SMTP error by server.
12789	31F5	Syntax error in parameters or arguments.
12790	31F6	Command not implemented. SMTP error by server.
12791	31F7	The SMTP server has encountered a bad sequence of commands, or it requires an authentication.
12823	3217	ETH_SMTP_EMAIL_SEND: Wrong user login and/or password. Check configuration. SMTP Error by server. MqttClient: Authentication failed: Bad username, password OR client id.
12838	3226	ETH_SMTP_EMAIL_SEND
12839	3227	ETH_SMTP_EMAIL_SEND: Mailbox unavailable or not found on target server. SMTP Error by server.
12840	3228	ETH_SMTP_EMAIL_SEND: Exceeded mailbox storage on target server. SMTP Error by server.
12841	3229	ETH_SMTP_EMAIL_SEND: Mailbox name not allowed. SMTP Error by server.
12848	3230	MqttClient: Error on TLS handshake.

DEC	HEX	Error description
12849	3231	MqttClient: Server certificate not valid. Check if PLC date has been set correctly.
12850	3232	MqttClient: Server certificate format is not formatted as PEM.
12851	3233	MqttClient: Server certificate has expired.
12852	3234	MqttClient: Client certificate not valid. Check if PLC date has been set correctly.
12853	3235	MqttClient: Client certificate or client key format is not formatted as PEM.
12854	3236	MqttClient: Client certificate has expired.
16383	3FFF	Not ready. Resources currently not available. COM_MOD_MAST: Transmission is not possible at the moment. Another instance of the function block is already transmitting. COM_SEND: Transmission is not possible at the moment. Another instance of the function block is already transmitting. BACnet B-ASC: No memory for BACnet objects.

1.5.3.5 4000hex...4FFFhex - block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.3.6 5000hex...5FFFhex - request error

DEC	HEX	Error description
20482	5002	The request is not supported (e.g. in the simulation mode of the Automation Builder).
20485	5005	Invalid internal state -> function block probably called in invalid manner/ sequence
20499	5013	Timeout error ECAT_COE_READ, ECAT_COE_Write: Request timeout occurs
20503	5017	Incorrect length ARC..: Invalid data length CAN2..: Invalid DLC in message ETH_UDP..: Invalid data length ECAT_COE_READ, ECAT_COE_Write: Request/ Response length invalid
20504	5018	Traceability data is corrupted.

DEC	HEX	Error description
20507	501B	Access error COM_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges. ETH_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges.
20508	501C	Invalid number (quantity) CAN2A_SEND: Invalid number of messages at input NUM COM_MOD_MAST: Invalid number of data at NB (0 or more than permitted). ETH_MOD_MAST: Invalid number of data at NB (0 or more than permitted).20515
20513	5021	IO_PROD_ENTRY_READ: I/O Module not found: input MODULE invalid
20515	5023	Failed to read traceability data from I/O Module.
20517	5025	Address error ARC..: Invalid IP address CAN2..: Invalid identifier in message COM_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_UDP..: Invalid IP address.
20518	5026	Function error COM_MOD_MAST: Invalid function code FCT. ETH_MOD_MAST: Invalid function code FCT.
20735	50FF	DIAG..: Simulation mode
24575	5FFF	Operation pending / busy -> wait and try again later

1.5.3.7 6000hex...6FFFhex - communication module errors

DEC	HEX	Error description
24577	6001	EtherCAT: Invalid command received PNIO: Creating a TLR-timer-packet in RPC task failed due to insufficient memory
24578	6002	EtherCAT: No link exists OR the watchdog expired CAN..: Service was rejected by the node with SDO abortion. Index/subindex not valid or no access to the specified node. DNM..: Resource not available or invalid class ID DPM../DPV1..: Resource not available. Free buffer memory in slave is not sufficient for the requested service. PNIO: Generic RPC-error code or not enough memory
24579	6003	EtherCAT: Error during reading the bus configuration OR the requested watchdog time is too small. DPM../DPV1..: Requested service (e.g. DPV1) is not active in the slave.

DEC	HEX	Error description
24580	6004	Communication Module does not support DPRAM message communication function blocks OR There is no receive function block active in IEC project of Communication Module. EtherCAT: Error during processing the bus configuration OR The requested watchdog time is too large.
24581	6005	EtherCAT: Existing bus does not match configured bus OR Error during reset (resetting watchdog).
24582	6006	EtherCAT: Not all slaves are available OR Error during reset (cleanup dynamic resources) DPM../DPV1..: Slave address not configured
24583	6007	EtherCAT: Error during reset (stopping the master) OR master is in critical error state, reset required.
24584	6008	EtherCAT: Error during reset (deinitializing the master) OR error activating the watchdog. DNM..: Service not available in module. Read/write function not supported by the selected class.
24585	6009	EtherCAT: Error during reset (cleanup the dynamic resources) OR size of configured input data is larger than cyclic DPM input data size. DNM..: Attribute invalid or not supported DPM../DPV1..: No data received from slave
24586	600A	EtherCAT: Master is in critical error state. Reset required OR size of configured output data is larger than DPM data output size.
24587	600B	EtherCAT: The requested bus cycle time is invalid DNM..: Request is already in progress
24588	600C	EtherCAT: Invalid parameter for broken slave behavior. DNM..: Conflict of the object status
24589	600D	EtherCAT: Master is in wrong internal state
24590	600E	EtherCAT: The watchdog expired DNM..: Attribute cannot be set or writing is not permitted
24591	600F	EtherCAT: Invalid SlaveID was used for CoE DNM..: Permission check faulty or access denied
24592	6010	EtherCAT: No available resources for CoE Transfer DNM..: Status conflict. Device prohibits execution
24593	6011	EtherCAT: Internal error during CoE usage CAN..: No response from the selected node DNM..: No response from the selected device DPM../DPV1..: No response received from slave.
24594	6012	EtherCAT: Invalid index on slave requested DPM../DPV1..: Master not in logical "token ring". PNIO: Internal error inside Communication Module's firmware
24595	6013	EtherCAT: Invalid bus communication state for CoE-usage CAN..: Selected node is not ready for operation DNM..: Not enough receive data

DEC	HEX	Error description
24596	6014	EtherCAT: Frame with CoE data is lost CAN..: Local resources are not available. Requested bus parameters are not available. Communication Module is not configured. DNM..: Local resources are not available. Requested bus parameters are not available. Communication Module is not configured. PNIO: Internal error inside Communication Module's firmware
24597	6015	EtherCAT: Timeout during CoE service CAN..: Parameter error DNM..: Parameter error
24598	6016	EtherCAT: Slave is not addressable (not on bus or power down) DNM..: Object does not exist
24599	6017	EtherCAT: Invalid list type requested Other Communication Modules: Received data length too big. Internal buffer too small.
24600	6018	EtherCAT: Data in slave response is to large for confirmation packet. PNIO: Internal error inside Communication Module's firmware
24601	6019	EtherCAT: Invalid access mask selected (during GetEntryDesc) DPM../DPV1..: Unexpected reaction of slave or reaction not in accordance with standard.
24602	601A	EtherCAT: Slave Working Counter error during CoE service PNIO: Another request is already running ETH2 not supported on AC500 CPU with Ethernet
24603	601B	EtherCAT: The service is already in use
24604	601C	EtherCAT: Command is not usable in this communication state
24605	601D	EtherCAT: Distributed Clocks must be activated for this command
24606	601E	EtherCAT: The scan is already running. It cannot be started twice at the same time
24607	601F	EtherCAT: Timeout during bus scan, but at least one link is established
24608	6020	EtherCAT: The bus scan was not started before or it is not finished yet
24609	6021	EtherCAT: The requested slave is invalid
24610	6022	EtherCAT: Internal error during CoE usage
24612	6024	PNIO: Internal error inside Communication Module's firmware
24615	6027	EtherCAT: Internal error of SDO protocol
24616	6028	EtherCAT: Internal error of SDO protocol
24617	6029	EtherCAT: Internal error of SDO protocol
24618	602A	EtherCAT: Internal error of SDO protocol
24619	602B	EtherCAT: Internal error of SDO protocol
24620	602C	EtherCAT: Internal error of SDO protocol
24621	602D	EtherCAT: Not enough memory
24622	602E	EtherCAT: Selected object could not be accessed
24623	602F	EtherCAT: Selected object is write only

DEC	HEX	Error description
24624	6030	EtherCAT: Selected object is read only CAN..: Function timeout DNM..: Device not configured PNIO: Internal error inside Communication Module's firmware
24625	6031	EtherCAT: Selected object does not exist PNIO: Internal error inside Communication Module's firmware
24626	6032	EtherCAT: PDO mapping failed DNM..: Format error in the received data ETH_UDP..: TCP/UDP task not available or IP task not ready. PNIO: Internal error inside Communication Module's firmware
24627	6033	EtherCAT: Selceted object is to large to be mapped to PDO CAN..: Maximum buffer size of the receive data exceeded ETH_UDP..: Internal task with configuration data not available PNIO: The ALPMR protocol-machine corresponding to the index in request packet is invalid
24628	6034	EtherCAT: General parameter error occured CAN..: Function not available. Code unknown. DNM..: Code unknown ETH_MOD..: Invalid parameter for "ServerConnection" ETH_UDP..: No MAC address available PNIO: The ALPMR protocol-machine is invalid for the current request
24629	6035	EtherCAT: Internal device error occured CAN..: Unknown area. Buffer exceeded. DNM..: Overflow of buffer length ETH_MOD..: Invalid parameter for "Task Timeout" ETH_UDP..: Waiting for warm start performed by the application
24630	6036	EtherCAT: Hardware error occured CAN..: Unknown function in HOST message or function still active DNM..: Other service still active ETH_MOD..: Invalid parameter for "OBM Timeout" ETH_UDP..: Unknown flag in start parameters DPM../DPV1..: Slave denied access to the requested data
24631	6037	EtherCAT: Invalid data type CAN..: Parameter error DNM..: Parameter error or MAC ID beyond the valid range ETH_MOD..: Invalid parameter for "Mode" ETH_UDP..: Invalid IP address in start parameters PNIO: The index of ALPMR protocol-machine is invalid
24632	6038	EtherCAT: Invalid data type ETH_MOD..: Invalid parameter for "Send Timeout" ETH_UDP..: Invalid subnet mask in start parameters

DEC	HEX	Error description
24633	6039	EtherCAT: Invalid data type CAN..: Sequence error DNM..: Sequence error or one MAC ID was multiple used in one network ETH_MOD..: Invalid parameter for "Connect Timeout" ETH_UDP..: Invalid gateway IP in start parameters
24634	603A	EtherCAT: Invalid sub-index ETH_MOD..: Invalid parameter for "Close Timeout".
24635	603B	EtherCAT: Invalid parameter value CAN..: Data error DNM..: Data error ETH_MOD..: Invalid parameter for "Swab" ETH_UDP..: Unknown device type
24636	603C	EtherCAT: Invalid parameter value CAN..: Node address configured twice DNM..: Display of total number of data sets incorrect ETH_MOD..: TCP task not ready ETH_UDP..: Access to IP address in the specified source failed
24637	603D	EtherCAT: Invalid parameter value CAN..: ADD table incorrect DNM..: ADD table incorrect ETH_MOD..: PLC task not ready ETH_UDP..: Initialization of the driver layer failed
24638	603E	EtherCAT: Invalid parameter value CAN..: Total length of the node parameters incorrect DNM..: Size of the I/O configuration table incorrect ETH_MOD..: Error during initialization ETH_UDP..: No source specified for IP address (BOOTP, DHCP, IP address parameter)
24639	603F	EtherCAT: Internal error of SDO protocol CAN..: Transmission type unknown DNM..: I/O configuration does not match with the ADD table
24640	6040	EtherCAT: Internal device error occured CAN..: Length of the PDO-cfg file too big DNM..: Parameter size incorrct or channel/handler already in use PNIO: Internal error inside Communication Module's firmware
24641	6041	EtherCAT: Internal device error occured CAN..: Unknown transmission rate DNM..: Number of defined inputs in the ADD table does not match with the I/O configuration PNIO: Internal error inside Communication Module's firmware

DEC	HEX	Error description
24642	6042	EtherCAT: Internal device error occured CAN..: COB-ID SYNC beyond the valid range DNM..: Number of defined outputs in the ADD table does not match with the I/O
24643	6043	EtherCAT: Internal device error occured CAN..: Value of the synchronization timer invalid DNM..: Unknown data type in the I/O configuration
24644	6044	EtherCAT: Unknown SDO protocol error CAN..: Input offset of the PDOs too big DNM..: Defined data type of an I/O module does not match with the defined data size
24645	6045	CAN..: Output offset of the PDOs too big DNM..: The configured output address of an I/O module is not within the permitted address range of 3584 bytes
24646	6046	CAN..: Inconsistency between the PDO and the ADD table DNM..: The configured input address of an I/O module is not within the permitted address range of 3584 bytes
24647	6047	CAN..: Length of the ADD table inconsistent DNM..: Unknown connection type
24648	6048	CAN..: Total data length inconsistent DNM..: Several identical connections defined
24649	6049	CAN..: COB-ID Emergency beyond the permitted range DNM..: The configured value of the "Exp_Packet_Rate" of a connection is smaller than the value of the "Prod_Inhibit_Time"
24650	604A	CAN..: COM-ID Node Guard beyond the permitted range DNM..: Inconsistent parameter field "DNM_PRED_MSTSL_CFG_DATA"
24651	604B	CAN..: Configured PDO length greater than 8 DNM..: Device could not perform "Duplicate_MAC-ID check". Incorrect transmission rate or no connection to the device possible.
24652	604C	CAN..: Number of defined objects in SDO data too big DNM..: Value of "usRecFragTimer" beyond the permitted range
24657	6051	PNIO: The current bus state is OFF and no frames can be sent
24659	6053	PNIO: The state of APMS protocol-machine is invalid for the current request
24660	6054	PNIO: APMS was not able to get an Edd_Frame buffer for sending a packet
24661	6055	PNIO: An error occurred while APMS was trying to send an Edd_Frame
24662	6056	PNIO: Device not reachable (DEV_NAME is not projected)
24663	6057	PNIO: Insufficient memory for APMS_send_req_Date() to allocate a timer-indication packet
24672	6060	PNIO: The acyclic service failed. The I/O module answered with an error code. See output STATUS (EtherCAT status) for details.
24679	6067	PNIO: The maximum amount of data supported by this service is exceeded.
24686	606E	ETH_UDP..: Timeout has occurred

DEC	HEX	Error description
24687	606F	ETH_MOD..: Unknown send or receive telegram ETH_UDP..: Invalid timeout parameter
24688	6070	ETH_MOD..: TCP responds with an error ETH_UDP..: Invalid socket
24689	6071	ETH_MOD..: No corresponding socket found ETH_UDP..: Command cannot be executed in the current socket state
24690	6072	ETH_MOD..: Command with invalid value
24691	6073	ETH_MOD..: TCP task status error ETH_UDP..: No access to target IP address
24692	6074	ETH_UDP..: Invalid option parameter
24693	6075	ETH_MOD..: No free socket found ETH_UDP..: Invalid command parameter
24694	6076	ETH_MOD..: TCP command is directed to an unknown socket ETH_UDP..: Invalid IP address or no access to address HA..: Wrong IP address configured
24695	6077	ETH_MOD..: Time for a client job is over ETH_UDP..: Invalid port number or port not available
24696	6078	ETH_MOD..: Socket has been closed unexpectedly ETH_UDP..: Connection closed
24697	6079	ETH_MOD..: Not-Ready flag has been set by the user ETH_UDP..: Connection reset
24698	607A	ETH_MOD..: OMB task cannot open socket ETH_UDP..: Invalid protocol
24699	607B	ETH_MOD..: Watchdog event in PLC task, only in I/O mode ETH_UDP..: No socket available
24700	607C	ETH_MOD..: TCP task in configuration state ETH_UDP..: Invalid MAC address
24701	607D	ETH_MOD..: PLC task not initialized
24702	607E	ETH_MOD..: Server socket was closed without response from the device
24703	607F	ETH_MOD_MAST.
24705	6081	DPM../DPV1..: DPV1 not in "OPEN" state
24706	6082	ETH_UDP..: Invalid mode parameter DPM../DPV1..: Invalid parameters received from slave. Communication stopped.
24707	6083	ETH_UDP..: Maximum data length exceeded or ARP cache full DPM../DPV1..: Service still active. Parallel operation is not possible
24708	6084	ETH_UDP..: Maximum number of messages exceeded DPM../DPV1..: Data length too high for the reserved buffer
24709	6085	ETH_UDP..: Maximum number of IP multicast groups exceeded DPM../DPV1..: Wrong parameter
24710	6086	ETH_UDP..: ARP input not found in ARP cache

DEC	HEX	Error description
24725	6095	ETH_UDP..: Invalid message response received
24727	6097	ETH_MOD..: Invalid message length ETH_UDP..: Invalid message length
24728	6098	CAN..: Unknown message command DNM..: Unknown message command ETH_MOD..: Unknown message command ETH_UDP..: Unknown message command
24730	609A	DPM../DPV1..: Invalid message command
24732	609C	ETH_UDP..: Sequence error during transmission in Sequence Message Mode
24734	609E	ETH_UDP..: Command cannot be executed or command is currently executed
24736	60A0	ETH_MOD..: Error in telegram header
24737	60A1	CAN..: Node address beyond the permitted range DNM..: Device address beyond the permitted range ETH_MOD..: Invalid address detected in the telegram DPM../DPV1..: Invalid slave address
24738	60A2	CAN..: Invalid address range DNM..: Invalid address range
24739	60A3	CAN..: Data buffer overflow DNM..: Data buffer overflow ETH_MOD..: Invalid data address
24741	60A5	CAN..: Incorrect data counter DNM..: Incorrect data counter ETH_MOD..: Invalid data counter
24742	60A6	CAN..: Unknown data type DNM..: Unknown data type
24743	60A7	CAN..: Unknown function DNM..: Unknown function ETH_MOD..: OBM task received an error in the response of the TCP task
24776	60C8	CAN..: Communication Module is not configured DNM..: Communication Module is not configured ETH_UDP..: Task not initialized
24778	60CA	ETH_MOD..: OBM task does not have a segment from RCS
24779	60CB	ETH_MOD..: Unknown or invalid sender specified with the command
24786	60D2	ETH_UDP..: No configuration data available
24788	60D4	ETH_UDP..: Error while reading the configuration data
24789	60D5	ETH_UDP..: Error while creating the diagnosis structure
24794	60DA	ETH_UDP..: Not enough memory available
24832	6100	PNIO: Generic RPC-error code. See output STATUS (PROFINET-status) for details.
24847	610F	ETH_MOD_MAST: Wrong MBAP header received

DEC	HEX	Error description
24848	6110	ETH_MOD..: Invalid Unit Identifier received
24850	6112	ETH_MOD_MAST: Invalid MBAP header Length value
25088	6200	PNIO: Internal error inside Communication Module's firmware
25089	6201	PNIO: Internal error inside Communication Module's firmware
26117	6605	PNIO: Internal error inside Communication Module's firmware
26118	6606	PNIO: Internal error inside Communication Module's firmware
26119	6607	PNIO: Internal error inside Communication Module's firmware

Abbreviations	
RPC	Remote Procedure Call
CMCTL	Controller Context Management
APMS	Acyclic Protocol-Machine sender
APMR	Acyclic Protocol-Machine receiver

1.5.3.8 7000hex...7FFFhex - product libraries

Table 53: Drives library

Dec	Hex	Error description
28672	7000	Any activity was NOT completed within an appropiate TIME
28673	7001	Read or write parameter could not be completed
28674	7002	A parameter at the function block is out of the possible range. This does not refer to the parameter range which is allowed for the drive but just to the 32-bit integer which is used for internal calculation
28675	7003	The field bus connection is faulty
28677	7005	Wrong PPO type
28678	7006	Wrong or no adapter type could be detected
28679	7007	Drive type does not match to function block
28680	7008	Function aborted
28681	7009	Error while reading scaling parameter for REF1
28682	700A	Wrong parameter number at read/write parameter
28683	700B	COM interface differs from others on same LINE_TOKEN variable
28684	700C	Profile type error
28685	700D	Function block Read_Parameters has been executed with error
28686	700E	Function block Write_Parameters has been executed with error
28687	700F	No connection to communication block, or error in communication block
28688	7010	Error at reading the drives communication profile value
28689	7011	Drive communication profile can not be used with this function block
28690	7012	PROFINET or PROFIBUS write packet size exceed 240 byte data limit

Table 54: Solar library

Dec	Hex	Error description
28928	7100	ALARM: Limit exceeded. System has moved more than POS_DEG_LIMIT distance without any moving command.
28929	7101	ALARM: Timeout. The system has not reached POS_DEG_LIMIT within the tolerance time (3 * t POS_TIME_LIMIT [ms]).
28930	7102	ALARM: Limit at wrong side reached. System has reached POS_DEG_LIMIT distance in the opposite DIR to actual movement order.
28931	7103	ALARM: Low Limit exceeded. Tracker position is less than VIRTUAL_LIMIT_MIN.
28932	7104	High Limit exceeded. Tracker position is more than VIRTUAL_LIMIT_MAX.
28933	7105	Warning: Interlocking. Try to move BACKWARD while STOP_BWD input is set.
28934	7106	Warning: Interlocking. Try to move FORWARD while STOP_FWD input is set.
28935	7107	Warning: Interlocking. Both STOP_BWD and STOP_FWD input are set.

Table 55: Data logger library

Dec	Hex	Error description
29184	7200	zLOG_ERROR_NO_TYPE_SPECIFIED
29185	7201	zLOG_ERROR_LENGTH_BOOL_EXCEEDED
29186	7202	zLOG_ERROR_LENGTH_BYTE_EXCEEDED
29187	7203	zLOG_ERROR_LENGTH_INT_EXCEEDED
29188	7204	zLOG_ERROR_LENGTH_WORD_EXCEEDED
29189	7205	zLOG_ERROR_LENGTH_DINT_EXCEEDED
29190	7206	zLOG_ERROR_LENGTH_DWORD_EXCEEDED
29191	7207	zLOG_ERROR_LENGTH_REAL_EXCEEDED
29195	720B	zLOG_ERROR_OPEN_ERR
29196	720C	zLOG_ERROR_GETPOS_ERR
29197	720D	zLOG_ERROR_SETPOS_ERR
29198	720E	zLOG_ERROR_WRITE_ERR
29199	720F	zLOG_ERROR_READ_ERR
29200	7210	zLOG_ERROR_CLOSE_ERR
29201	7211	zLOG_ERROR_GETSIZE_ERR
29202	7212	zLOG_ERROR_FLUSH_ERR
29205	7215	zLOG_ERROR_MAX_NUMBER_OF_FILES_EXCEEDED
29206	7216	zLOG_ERROR_MAX_NUMBER_OF_DATASETS_PRO_FILE
29207	7217	EXCEEDED zLOG_ERROR_FILE_WRITE_FAILED
29208	7218	zLOG_ERROR_FILE_READ_FAILED
29210	721A	zLOG_ERROR_CREATE_DIRECTORY
29211	721B	zLOG_ERROR_FILE_MOVE
29212	721C	zLOG_ERROR_NO_CSV_FOUND_IN_DATASET
29213	721D	zLOG_ERROR_DELETE_ACTUAL_FILE
29214	721E	zLOG_ERROR_FORMAT_DISK1
29215	721F	zLOG_ERROR_FORMAT_DISK2

Dec	Hex	Error description
29216	7220	zLOG_ERROR_DELETE_ACTUAL_FILES_NO_FILE_FOUND
29217	7221	zLOG_ERROR_DELETE_SAVE_FILES_NO_FILE_FOUND
29218	7222	zLOG_ERROR_ILLEGAL_MODE
29219	7223	zLOG_ERROR_ILLEGAL_DESIGNATION_ON_DISK1
29220	7224	zLOG_ERROR_ILLEGAL_DESIGNATION_ON_DISK2
29221	7225	zLOG_ERROR_FORMAT_NOT_SUPPORTED
29222	7226	zLOG_ERROR_MODE3_AND_DISK2_EXTENTION_TRUE_NOT_ALLOWED
29230	722E	zLOG_ERROR_29230_TIMEOUT_DIRECTORY_CREATE_FILE_MOVE
29231	722F	zLOG_ERROR_RESERVE
29232	7230	zLOG_ERROR_TIMEOUT_DISK_FORMAT
29233	7231	zLOG_ERROR_TIMEOUT_FILE_DELETE
29234	7232	zLOG_ERROR_TIMEOUT_WRITE_DATASET_IN_FILE
29235	7233	zLOG_ERROR_TIMEOUT_READ_DATASET_IN_CASE_OF_CONNECT
29236	7234	zLOG_ERROR_TIMEOUT_WRITE_DATASET_IN_CASE_OF_CONNECT
29237	7235	zLOG_ERROR_TIMEOUT_READ_DATASET_IN_CASE_OF_NOT_CONNECT
29238	7236	zLOG_ERROR_TIMEOUT_IEC60870_COMMUNICATION

Table 56: Temperature control library

Dec	Hex	Error description
29312	7280	Fault (TuneFault): "Tuning failed": outputs disabled, see Tune_Status Remedy: Check for tune set point and currrent temperatur
29313	7281	Fault (TC_Fault_1): "Bad Thermocouple reading": outputs disabled. Remedy: Bad Thermocouple reading: outputs disabled
29314	7282	Fault (TC_Fault_2): "Plausibility check not passed": outputs disabled. Remedy: Check for inverted connection of Thermocouple.
29315	7283	Fault (HighHighTempFalt): "HighHigh temperature alarm": outputs disabled. Remedy ● Check for defective cooling device. ● Check for correct setting of HIGHHIGH_TEMP and SET_TEMP.
29316	7284	Fault (LowLowTempFault): "LowLow temperature alarm": outputs disabled. Remedy ● Check for defective heating device. ● Check for correct setting of LOWLOW_TEMP and SET_TEMP.
29319	7287	High temperature alarm
29320	7288	Low temperature alarm
29321	7289	High deviation alarm
29330	7292	Zone_size and zone_index are not assigned
29335	7297	No group function block call possible: internal Group function call for TECT_TEMP_CONTROL or TECT_PWM8 failed.
29336	7298	TimeOut of one operation state.
29337	7299	Read data failed: no correct format of data items.

Dec	Hex	Error description
29322	728A	Low deviation alarm
29323	728B	Fault: (NoHighHighLowLow): "No plausible HighHigh or LowLow limit is defined" Remedy: Check the HighHighTemp and low low temp.
29324	728C	Fault (WrongLimits): "No plausible wrong limits defined" Remedy: Check the defined limits.
29325	728D	Heat and Cool are not enabled: Monitoring only.
29326	728E	Faut (NoAutoTune): "AutoTune cannot be started" Remedy: Process in Manual mode or Heat is not enabled
29327	728F	Fault (NoPIDProcess): "PID process cannot be started" Remedy: Process in Manual mode or no KS, TU, TG parameters are accepted.
29328	7290	Fault (Improper setting of HigHigh and High): HighHigh and High limits are not set correctly. Values are not taken by process/ADR_ZONEDATA. Remedy: Check parameter settings.
29329	7291	Fault (Improper setting of LowLow and Low): LowLow and Low limits are not set correct. Values are not taken by process/ADR_ZONEDATA. Remedy: Check parameter settings.
29330	7292	Fault (No assignment for zone index and size): Zone_size and zone_index are not assigned. Remedy: Check parameter settings.
29335	7297	Fault (No Group function possible):no Group function block call possible: internal Group function call for TECT_TEMP_CONTROL or TECT_PWM8 failed.
29336	7298	TimeOut of one operation state.
29337	7299	Fault (Reading failed):Read data failed: no correct format of data items. Remedy: Check parameter settings.
29338	729A	Fault (Incomplete log saving): Restart of the EN is required since last log saving is not complete
29339	729B	Fault (Less zone data available in RECIPE file than requested):Mismatch in number of zones declared and zone data in recipe. Remedy: Check the zone data in CSV file and Num_of_zones declared in RECIPE.

Table 57: CMS/FM502 library

Dec	Hex	Error description
29440	7300	Invalid file format.
29445	7305	Number of channels > 1, not supported by this version of library.
29440	7300	Invalid RIFF Chunk ID
29441	7301	Invalid WAV Chunk ID
29442	7302	Invalid FMT Chunk ID
29443	7303	Invalid DATA Chunk ID
29444	7304	Invalid LABL Chunk ID
29445	7305	Number of channels >1 , not supported by this version of library.
29504	7340	No Channel activated
29505	7341	Internal Error (Start)

Dec	Hex	Error description
29506	7342	Internal Error (Poll)
29507	7343	Internal Error (Finish)
29508	7344	Not possible because of measurement running
29519	734F	No Measurement possible. Device in Failure Loop.
29520	7350	Comunication Timeout to FM502
29521	7351	Internal File Error
29522	7352	Plausibility Check of Configuration wrong
29569	7381	Error in memory allocation to the input signal, due to insufficient memory available.
29571	7383	Wrong intermediate variable value
29572	7384	Error during reading wave file - check wave file for format compatibility.
29573	7385	Could not open indicated wave file.

1.5.4 Standard function block libraries AC500

1.5.4.1 ARCNET library

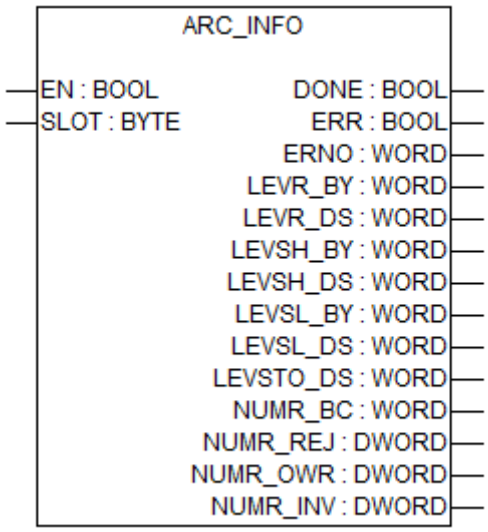
Library file name: **ARCNET_AC500_Vx.lib**

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

The use of the function blocks presupposes that the ARCNET Communication Module is configured in the ARCNET data exchange mode.

1.5.4.1.1 Function blocks

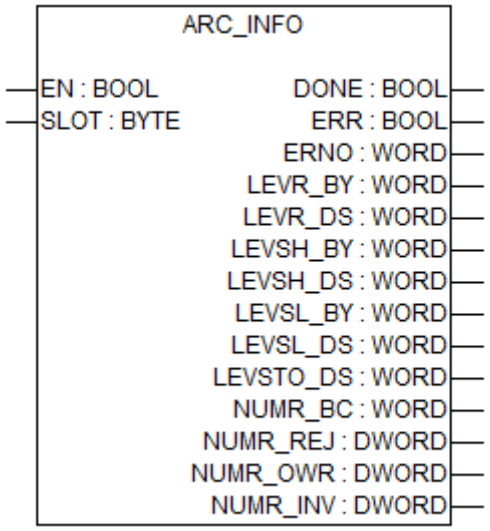
ARC_INFO



Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	General

Using the ARC_INFO function block, various status information about the ARCNET processing can be read. This information can only be read if in the "ARCNET data exchange".

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

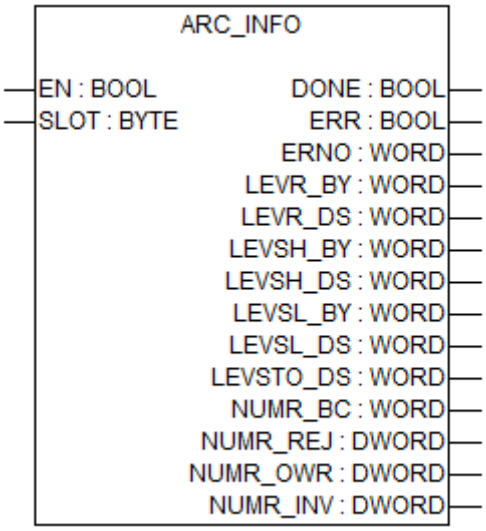
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEVR_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVSH_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_BY displays the filling level of the high priority send buffer in bytes.

LEVSH_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_DS displays the filling level of the high priority send buffer in data sets.

LEVSL_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_BY displays the filling level of the low priority send buffer in bytes.

LEVSL_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_DS displays the filling level of the low priority send buffer in data sets.

LEVSTO_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSTO_DS displays the filling level of the timeout buffer in data sets.

NUMR_BC

Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_BC outputs the number of broadcasts (data packets to all stations) which were received by this station.

NUMR_REJ

Data type	Default value	Range	Unit
DWORD	-	-	-

At output NUMR_REJ, the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the PLC configuration of the ARCNET processing.

NUMR_OWR

Data type	Default value	Range	Unit
DWORD	-	-	-

At output NUMR_OWR, the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the PLC configuration of the ARCNET processing.

NUMR_INV

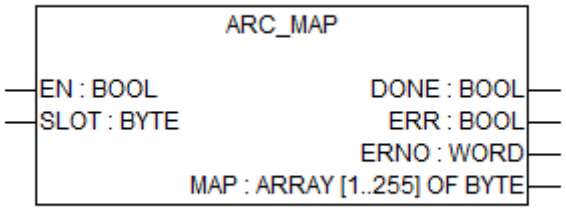
Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in ST

```
Info  (EN := Info_EN,
       SLOT := Info_SLOT;

Info_DONE := Info.DONE;
Info_ERR := Info.ERR;
Info_ERNO := Info.ERNO;
Info_LEVR_BY := Info.LEVR_BY;
Info_LEVR_DS := Info.LEVR_DS;
Info_LEVSH_BY := Info.LEVSH_BY;
Info_LEVSH_DS := Info.LEVSH_DS;
Info_LEVSL_BY := Info.LEVSL_BY;
Info_LEVSL_DS := Info.LEVSL_DS;
Info_LEVSTO_DS := Info.LEVSTO_DS;

Info_NUMR_BC := Info.NUMR_BC;
Info_NUMR_REJ := Info.NUMR_REJ;
Info_NUMR_OWR := Info.NUMR_OWR;
Info_NUMR_INV := Info.NUMR_INV;
```

ARC_MAP

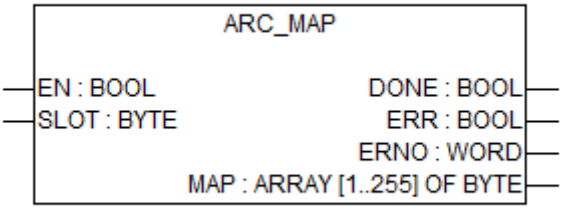


Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.3
Type	Function block without historical values
Group	General

The function block ARC_MAP provides information about the ARCNET network map. The output MAP shows the status of every node. If the node is active, a "1" is shown at the corresponding output array. (Each possible node has an own byte at the array; e. g. nodes 2 and 3 are active -> array of Byte [2] = 1 and array of byte [3] = 1, the others have the value 0).

Every node of the ARCNET map is checked by the RTS for about 210 ms. So the complete cycle time for the whole map takes about 255*210 ms (If the map consists of 255 nodes. If the network consists of 5 nodes it takes about 1s).

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

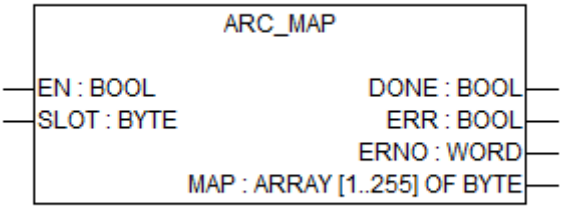
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

MAP

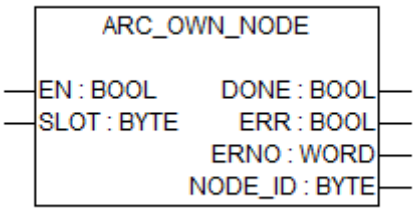
Data type	Default value	Range	Unit
ARRAY	-	-	-

As long as EN = TRUE and at least 2 nodes are active (a network consists of min 2 nodes), output array [1..255] of BYTE shows the status information of the network map.

Function call in ST

```
MAP (EN := MAP_EN,
SLOT := MAP_SLOT);
MAP_DONE := MAP.DONE;
MAP_ERR := MAP.ERR;
MAP_ERNO := MAP.ERNO;
MAP_MAP := MAP.MAP;
```

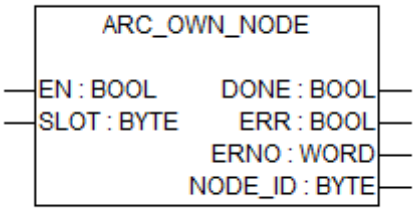
ARC_OWN_NODE



Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block without historical values
Group	General

Prior to setting into operation an AC500 Communication Module, it must be configured using the system configuration of the Control Builder. One of the parameters of an ARCNET Communication Module is its own NODE_ID. Using the function block ARC_NODE_ID, the latest configured NODE ID of the device at SLOT can be read. If no ARCNET Communication Module is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

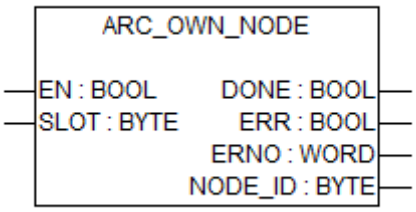
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE_ID

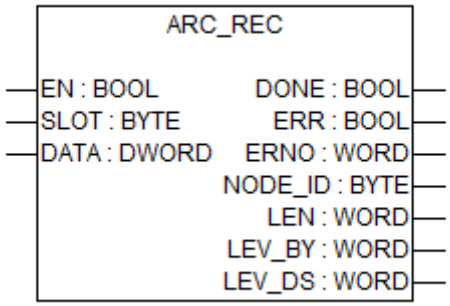
Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

Function call in ST

```
ARC_OWN_NODE(EN := OwnNode_EN, SLOT := OwnNode_SLOT);

OwnNode_DONE := OwnNode.DONE;
OwnNode_ERR := OwnNode.ERR;
OwnNode_ERNO := OwnNode.ERNO;
  OwnNode_IP_ADR := OwnNode.NODE_ID;
```

ARC_REC



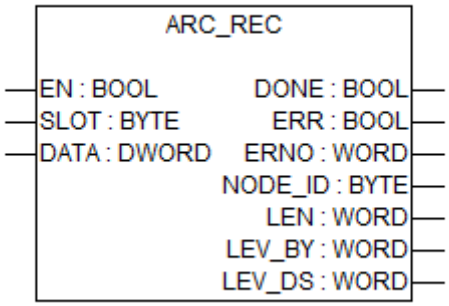
Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	Data

The operating system reads the received ARCNET data packets from the ARCNET Communication Module and stores them in the receive buffer. The buffer size is determined using the ↳ *Chapter 1.6.5.2.3.6 "Parameters of PM5x1-ARCNET (onboard ARCNET)" on page 5846* of the CPU. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ARC_REC function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The NODE ID of the sending device and the data packet length are supplied at the outputs NODE_ID and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

Before the ARC_REC function block can read data packets from the receive buffer, the setting "ARCNET data exchange" must have been selected in the ↳ *Chapter 1.6.5.2.3.6 "Parameters of PM5x1-ARCNET (onboard ARCNET)" on page 5846* of the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

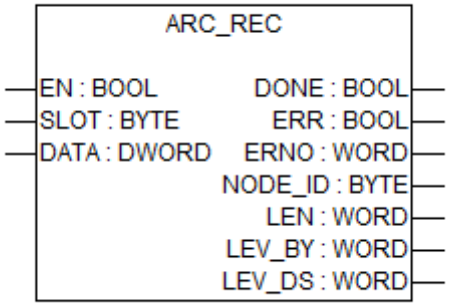
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

LEN

Data type	Default value	Range	Unit
WORD	-	-	byte

Output LEN displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

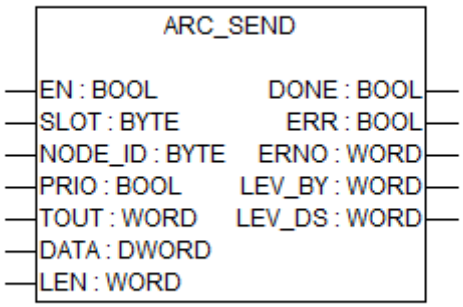
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
REC  (EN := Rec_EN,
     SLOT := Rec_SLOT,
     DATA := ADR(Rec_DATA));


REC_DONE := Rec.DONE;
REC_ERR := Rec.ERR;
REC_ERNO := Rec.ERNO;
REC_NODE_ID := Rec.NODE_ID;
REC_LEN := Rec.LEN;
REC_LEVR_BY := Rec.LEV_BY;
REC_LEVR_DS := Rec.LEV_DS;
```

ARC_SEND

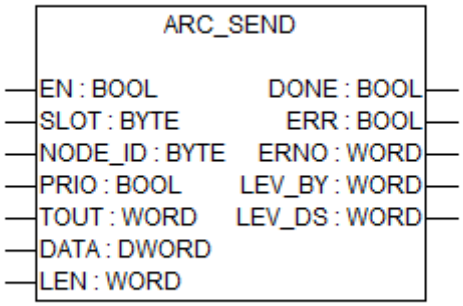


Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	Data

The function block ARC_SEND is used to transmit data packets via the ARCNET Communication Module. The specified packages are stored in the transmit buffer which is selected by input PRIO. From there, the operating system hands over the data packets to the ARCNET Communication Module in order to transmit them to the target address specified at input NODE_ID. The transmit buffer size is determined using the PLC Configuration of the Control Builder. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data packets has been stored in the transmit buffer or that an error occurred during Function Block processing. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

Before the ARC_SEND function block can store data packets in the receive buffer, the setting "ARCNET data exchange" must have been selected in the *Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846* of the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

PRIO

Data type	Default value	Range	Unit
BOOL	-	-	-

Input PRIO is used to specify the transmit priority of the data packet.

The following applies:

PRIO = FALSE

The specified data packet has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

PRIO = TRUE

The specified data packet has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT

Data type	Default value	Range	Unit
WORD	200ms	-	-

Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If a data packet cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data packet (see also the PLC configuration of the Control Builder) are stored to the timeout buffer and can then be read using the function block ARC_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0 means: Default value for TOUT, given by PLC configuration, is used.

Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:

Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

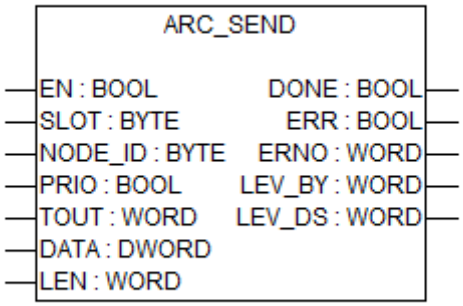
CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

LEN

Data type	Default value	Range	Unit
WORD	-	-	byte

Output LEN displays the length of the received data package in bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

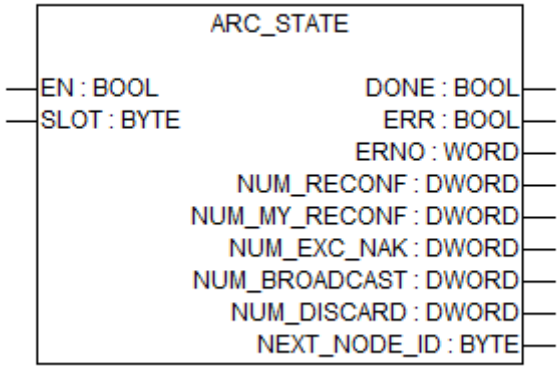
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
Send (EN := Send_EN,
     SLOT := Send_SLOT,
     NODE_ID := Send_NODE_ID,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     DATA := ADR(Send_DATA),
     LEN := Send_LEN);


Send_DONE := Send.DONE;
Send_ERR := Send.ERR;
Send_ERNO := Send.ERNO;
Send_LEV_BY := Send.LEV_BY;
Send_LEV_DS := Send.LEV_DS;
```

ARC_STATE

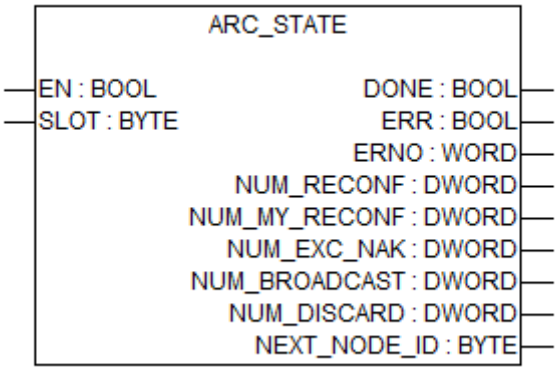


Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.3
Type	Function block without historical values
Group	General

The function block ARC_STATE is used to get general status information about the ARCNET network. With this FB you get

- the number of reconfigurations (whole number and caused by this device),
- the number of excessive NAKs
- the number of broadcasts
- the number of discards
- the number of the next node ID.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

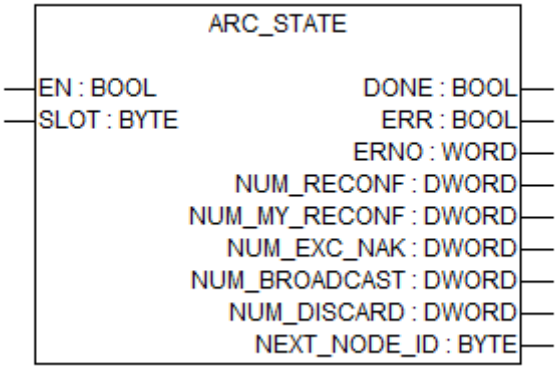
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_RECONF

Data type	Default value	Range	Unit
DWORD	-	-	-

Total number of reconfigurations

NUM_MY_RECONF

Data type	Default value	Range	Unit
DWORD	-	-	-

Number of reconfigurations caused by this device

NUM_EXC_NAK

Data type	Default value	Range	Unit
DWORD	-	-	-

Total number of excessive NAKs.

NUM_BROAD-CAST

Data type	Default value	Range	Unit
DWORD	-	-	-

NUM_DISCARD

Data type	Default value	Range	Unit
DWORD	-	-	-

Total number of receipts discarded due to no matching destination handle found.

NEXT_NODE_ID

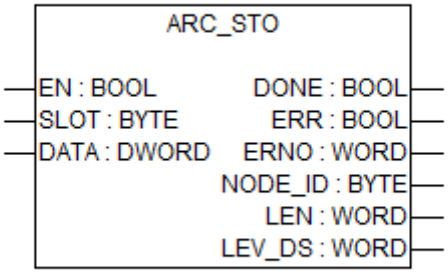
Data type	Default value	Range	Unit
BYTE	-	-	-

Next node ID.

Function call in ST

```
STATE (EN := STATE_EN,
SLOT := STATE_SLOT);


STATE_DONE := STATE.DONE;
STATE_ERR := STATE.ERR;
STATE_ERNO := STATE.ERNO;
STATE_NUM_RECONF := STATE.NUM_RECONF;
STATE_NUM_MY_RECONF:= STATE.NUM_MY_RECONF;
STATE_NUM_EXC_NAK := STATE.NUM_EXC_NAK;
STATE_NUM_BROADCAST:= STATE.NUM_BROADCAST;
STATE_NUM_DISCARD:= STATE.NUM_DISCARD;
STATE_NEXT_NODE_ID:= STATE.NEXT_NODE_ID;
```

ARC_STO



Parameter	Value
Included in library	ARCNET_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	Data

During the transmission of a data packet, the success of the transmission is monitored by an adjustable timeout period. When this time is exceeded, distinctive information of the data packet is stored in the timeout buffer.

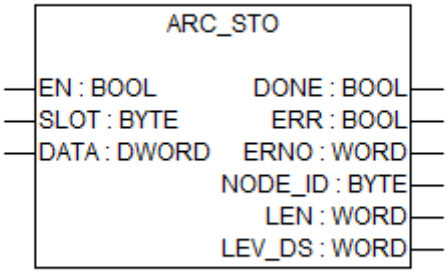
These are:

- the NODE ID of the receiver (4 bytes)
- the length of the involved data packet
- header data of the involved data set.

The length of the timeout buffer as well as the number of user data to be stored can be set for the ARCNET processing in the PLC configuration of the Control Builder. The buffer is constructed as a circular buffer (FIFO). If the buffer is full, the oldest entry in the buffer is overwritten. If the ARC_STO function block is enabled by EN = TRUE, it checks whether a data packet is stored in the buffer and provides the user with the above mentioned information as of the variable given at input DATA. At the outputs NODE_ID and LEN, the NODE ID and the original length of the telegram, which could not be sent, are available.

Before the ARC_STO function block can be used, the setting "ARCNET data exchange" has to be selected. Additionally, the input TOUT of the ARC_SEND function block must be >0.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

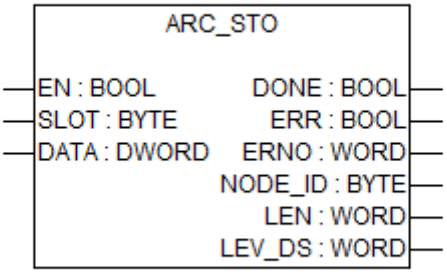
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

LEN

Data type	Default value	Range	Unit
WORD	-	-	byte

Output LEN displays the length of the received data package in bytes.

LEV_DS

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
Sto  (EN := Sto_EN,
     SLOT := Sto_SLOT,
     DATA := ADR(Sto_DATA));


Sto_DONE := Sto.DONE;
Sto_ERR := Sto.ERR;
Sto_ERNO := Sto.ERNO;
Sto_NODE_ID := Sto.NODE_ID;
Sto_LEN := Sto.LEN;
Sto_LEV_DS := Sto.LEV_DS;
```

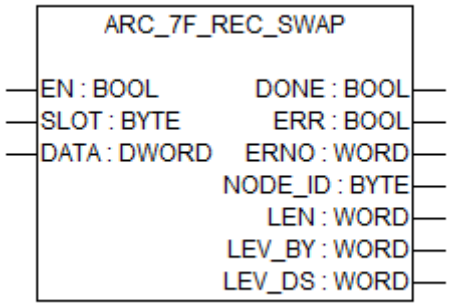
1.5.4.2 Extended ARCNET library

Library file name: **ARCNET_Ext_AC500_Vx.lib**

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

The use of the function blocks presupposes that the ARCNET Communication Module is configured in the ARCNET data exchange mode.

1.5.4.2.1 ARC_7F_REC_SWAP



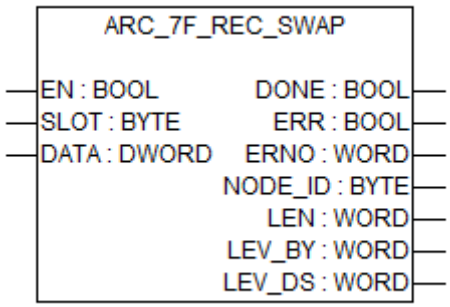
Parameter	Value
Included in library	ARCNETExt_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	Data

The operating system reads the received ARCNET data packets from the ARCNET Communication Module and stores them in the receive buffer. The buffer size is determined using the *Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846* of the CPU. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ARC_7F_REC_SWAP function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The NODE ID of the sending device and the data packet length are supplied at the outputs NODE_ID and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

Before the ARC_REC function block can read data packets from the receive buffer, the setting "ARCNET data exchange" must have been selected in the *Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846* of the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

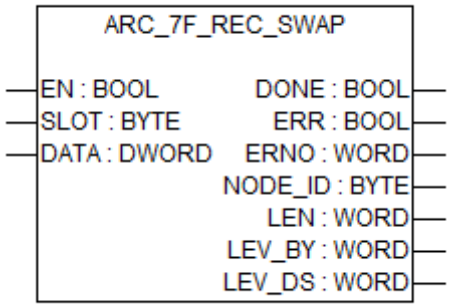
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

LEN

Data type	Default value	Range	Unit
WORD	-	-	byte

Output LEN displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

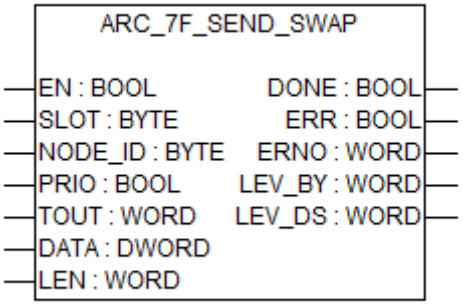
One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

1.5.4.2.2 ARC_7F_SEND_SWAP

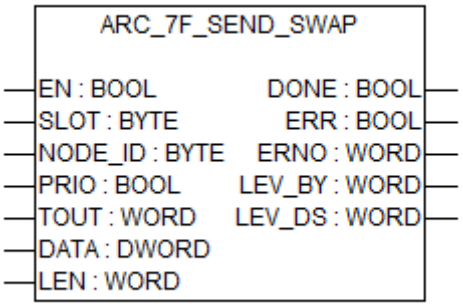


Parameter	Value
Included in library	ASCII_AC500_V12.lib
Available as of firmware	V1.2
Type	Function block with historical values
Group	Data

The function block ARC_7F_SEND_SWAP is used to transmit data packets via the ARCNET Communication Module. The specified packages are stored in the transmit buffer which is selected by input PRIO. From there, the operating system hands over the data packet to the ARCNET Communication Module in order to transmit them to the target address specified at input NODE_ID. The transmit buffer size is determined using the PLC Configuration ↳ *Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846* of the Automation Builder. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data packet has been stored in the transmit buffer or that an error occurred during Function Block processing. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

Before the ARC_SEND function block can store data packets in the receive buffer, the setting "ARCNET data exchange" must have been selected in the *Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846* of the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

Output NODE_ID displays the own NODE ID of the Communication Module.

PRIO

Data type	Default value	Range	Unit
BOOL	-	-	-

Input PRIO is used to specify the transmit priority of the data packet.

The following applies:

PRIO = FALSE

The specified data packet has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

PRIO = TRUE

The specified data packet has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT

Data type	Default value	Range	Unit
WORD	200ms	-	-

Using input TOUT, the timeout period can be specified. If TOUT <> 0, the ARCNET data exchange is automatically performed with receive acknowledgement. If a data packet cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data packet (see also the PLC configuration of the Control Builder) are stored to the timeout buffer and can then be read using the function block ARC_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0 means: Default value for TOUT, given by PLC configuration, is used.

Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:

Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. DATA must be the address of a variable of the type ARRAY or STRUCT.

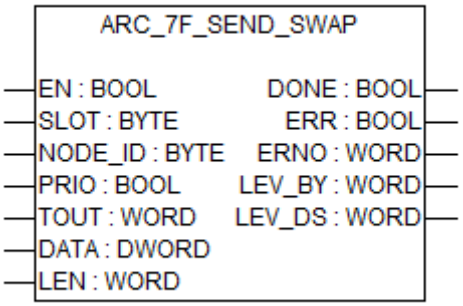
CAUTION: Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

LEN

Data type	Default value	Range	Unit
WORD	-	-	byte

Output LEN displays the length of the received data package in bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

1.5.4.3 ASCII communication library

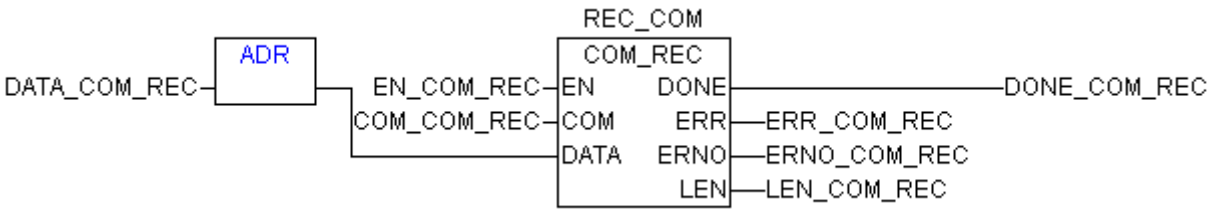
Library file name: **ASCII_AC500_Vx.lib**

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

There is no particular function block available for setting the communication interfaces in transmitting and receiving direction. This is done in the configuration within CODESYS.

1.5.4.3.1 Function blocks

COM_REC



Parameter	Value
Included in library	ASCII_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	ASCII communication

The function block COM_REC is used for receiving data via a serial interface in "free mode". The number of COM_REC function blocks within a project as well as their distribution (i.e. assignment) to different user tasks is not restricted. However, it has to be observed that the function blocks are mutually interlocked, i.e. it must be ensured that only one function block is active at the same time. To avoid the loss of telegram parts and to prevent that telegrams are evaluated incorrectly or not at all, a change of activity between two COM_REC function blocks should only occur if the function block to be deactivated has signalized the termination of the receive process by setting DONE = TRUE and if the received telegram has been evaluated. It is essential to ensure that all function block instances are inactive prior to initiating an activity change. Thus, it is strongly recommended to use only one COM_REC function block within a project. This way, any responsibility conflicts can be avoided.

With a FALSE -> TRUE edge at function block input EN, the function block checks the input values. If they are valid, the function block reads the receiving buffer of the corresponding COMx interface one-time.

By specifying the length of the memory address for the received data, the format of the telegram is not restricted in any way. The length of the received data block is limited to a maximum of 256 bytes and is output at LEN.

During project planning it has to be observed that enough free memory space is reserved starting at address DATA for storing the received data (e.g. ARRAY [1..256] OF BYTE).

If a valid received telegram is available in the memory area starting at DATA, this is always indicated by DONE = TRUE.

The inputs can neither be duplicated nor negated/inverted.

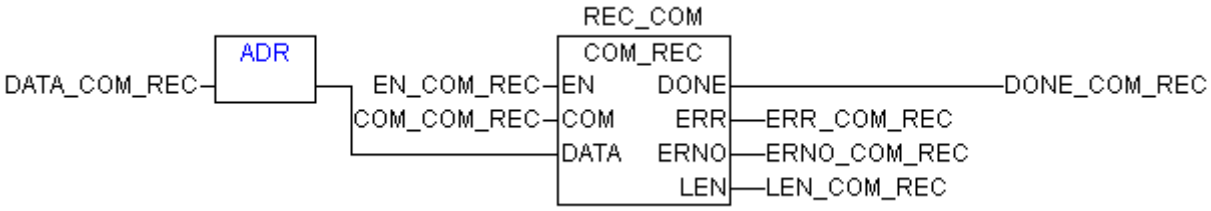
Receive error

Possible receive errors are detected by the function block and indicated by ERR = TRUE. In this case, an error number is output at ERNO. The function block recognizes overflow, parity and framing errors. In this case, the communication parameters (transmission rate, char length, no. of stop bits, parity) of the communication partners have to be checked.

Times

There is no particular function block available for setting the times in transmitting and receiving direction. This is done in AC500 controller configuration.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At the COM input, the number of the serial interface is specified.

COM = 1: COM1

COM = 2: COM2

DATA

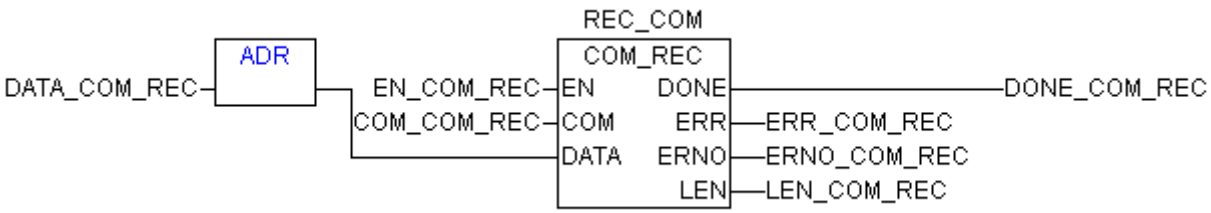
Data type	Default value	Range	Unit
DWORD	-	-	-

At input DATA, the start address for storing the received data is specified using an ADR operator. Received data can be stored in the operand area as well as in variables. Some peculiarities have to be observed when receiving binary values.

When using IEC bit operands as storage address, only operands are allowed which end with ".0" (e.g. %QX62.0 allowed, %QX62.1 forbidden).

When storing a received telegram within the IEC bit operand area, it has to be observed that a received byte describes 8 bit operands. However, if a received telegram is stored to a Boolean variable, this variable is considered as FALSE if the byte has the value 0 and TRUE for all other values of the byte.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEN

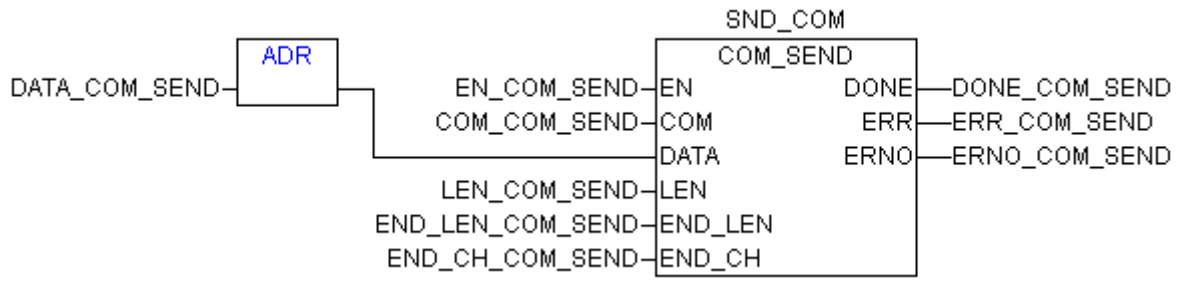
Data type	Default value	Range	Unit
WORD	-	-	-

LEN outputs the length of the received data (in bytes) including end character (if contained). LEN is only valid if DONE = TRUE. LEN has always to be considered together with output ERR.

Function call in ST

```
REC_COM
(EN := EN_COM_REC,
COM := COM_COM_REC,
DATA := ADR(DATA_COM_REC));
DONE_COM_REC := REC_COM.DONE;
ERR_COM_REC := REC_COM.ERR;
ERNO_COM_REC := REC_COM.ERNO;
LEN_COM_REC := REC_COM.LEN;
```

COM_SEND



Parameter	Value
Included in library	ASCII_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	ASCII communication

The function block COM_SEND is used for sending data via a serial interface. The number of COM_SEND function blocks in a project as well as their usage in (i.e. assignment to) different user tasks is not restricted. Transmission is triggered by a FALSE > TRUE edge at input EN. By specifying the length of the memory address for the data to be transmitted, the format of the telegram is not restricted in any way.

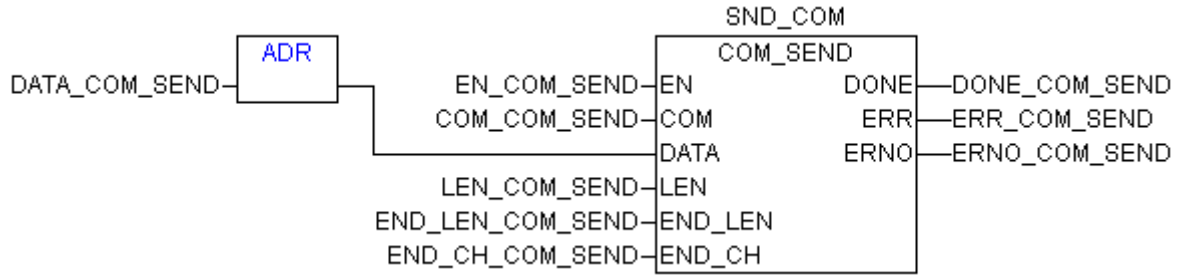
The length of the data block to be transmitted is not limited. It is recommended to write only data blocks up to a maximal size of 256 bytes into the transmit buffer, since data can only be transmitted if enough free memory space is available for the transmit buffer. If required, longer telegrams can be generated by using several COM_SEND function blocks following each other immediately without considering their individual DONE outputs.

The inputs can neither be duplicated nor negated/inverted.

Telegram length

The maximum length of a transmit telegram can be controlled by evaluating the output DONE. DONE = TRUE indicates an empty transmit buffer. If a transmission is only triggered if DONE = TRUE, one single telegram will be sent per COM_SEND. Theoretically, endless data streams can be generated by ignoring DONE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At the COM input, the number of the serial interface is specified.

COM = 1: COM1

COM = 2: COM2

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

At input DATA, the start address for storing the received data is specified using an ADR operator. Received data can be stored in the operand area as well as in variables. Some peculiarities have to be observed when receiving binary values.

When using IEC bit operands as storage address, only operands are allowed which end with ".0" (e.g. %QX62.0 allowed, %QX62.1 forbidden).

When storing a received telegram within the IEC bit operand area, it has to be observed that a received byte describes 8 bit operands. However, if a received telegram is stored to a Boolean variable, this variable is considered as FALSE if the byte has the value 0 and TRUE for all other values of the byte.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

LEN outputs the length of the received data (in bytes) including end character (if contained). LEN is only valid if DONE = TRUE. LEN has always to be considered together with output ERR.

END_LEN

Data type	Default value	Range	Unit
BYTE	-	-	-

Number of end characters to be attached: 0, 1, 2 means none, one or two.

The inputs LEN and END_LEN have to be smaller than 256! Otherwise, an error will occur.

END_CH

Data type	Default value	Range	Unit
WORD	-	-	-

At input END_CH, the value of the telegram end character(s) must be specified which has (have) to be attached to the actual transmit data. END_CH is considered if input END_LEN of the COM_SEND function block is not 0. END_CH is applied with a FALSE > TRUE edge at EN. If required, the end character can vary within each telegram.

If, for example, a CR/LF shall be attached to a telegram, the value of END_CH is as follows: 16#0D0A and END_LEN: 2.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
SEND_COM
(EN := EN_COM_SEND,
COM := COM_COM_SEND,
DATA := ADR(DATA_COM_SEND),
LEN := LEN_COM_SEND,
END_CH := END_CH_COM_SEND,
END_LEN := END_LEN_COM_SEND);

DONE_COM_SEND := SEND_COM.DONE;

ERR_COM_SEND := SEND_COM.ERR;
ERNO_COM_SEND := SEND_COM.ERNO;
```

1.5.4.4 CAA_File library

Library file name: **CAA_File.lib**

The IEC61131 library CAA_FILE provides function blocks for accessing directories and files. File access is only supported in binary mode.

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

For usage of CAA_File Library, CPU firmware V2.1 or higher is needed.

CAA_File.lib requires the following additional libraries:

- CAA_ASYNCMAN.lib
- CAA_CALLBACK.lib
- CAA_TICK.lib
- CAA_TICKUTIL.lib
- CAA_TYPES.lib

Most input values of the function blocks are stored in local internal variables. This does not apply for the contents of memory structures where a pointer is passed on as an input e.g. pBuffer of FILE_Read or FILE_Write.

Differing from Standard ABB Libraries CAA function blocks set EITHER xDone (i.e. function block finished without error) OR xError (i.e. function block finished with error)!

1.5.4.4.1 Characteristics

CAA_FILENAME which is described in CAA_Types.lib should be usable for library CAA_File.lib in the following way:

- Separators in paths in CAA_FILENAME is the back-slash ("\").
- Only absolute paths are allowed to use for file operation: drivename\directory\file.ext
- File and directory names have to be in standard DOS 8.3 notation without \ / : * ? "< > |
- Directory and file names may not contain whitespaces and are not case-sensitive.

Available drives for the use with this library are:

- Userdisk
- Memory card
- Flash disk – only on PM592-ETH (PLC with internal mass storage)
- SRAM Disk – only PM57x, PM58x, PM59x

Restrictions for the use of the drives are:

- Maximum file names: 8.3 filenames, e.g.: abcdefgh.123
- Maximum total path length: 255 characters
- Maximum file size: 4 GB -1 Byte
- Maximum number of simultaneous user handles (directories + files): 6 (PM55x) or 12 (PM57x,PM58x,PM59x)
- Maximum number of files in the root directory is limited (depends on memory location).

Behavior of CAA_File on reset

A reset operation is executed if any user calls any online reset command or downloads a new application into the PLC. All file handles which are opened during execution of IEC application using FILE_Open and FILE_DirOpen will be closed after application reset. This assures that all handles opened by the user application will be closed and returned to the system. Please note that the application is not reset in case of a stop or an online change operation, so any file handle in IEC application is kept opened.

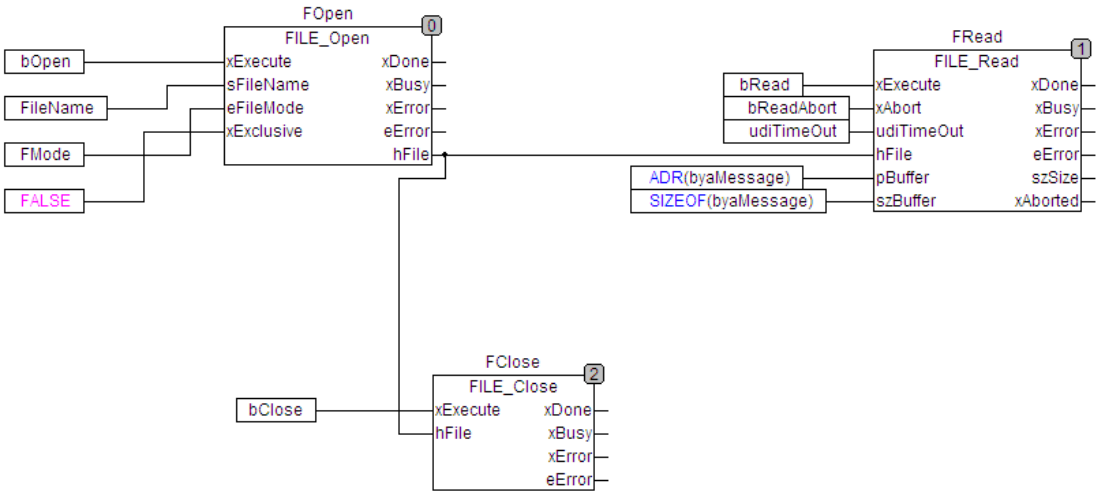
If a CAA_File function block is in BUSY state at the moment when reset command is executed the behavior of this function block is the following:

- If the request to file system which is executed by the function block is not yet processed by file system, the function block will be aborted. This means that the file operation will not be executed at all.
- If the file operation is already started and the corresponding function block does not support an abort command, the file operation will be finished and after that the corresponding file handle will be closed.
- If the file operation is already started and the corresponding function block supports an abort command (i.e. FILE_Write, FILE_Read, FILE_Copy) the file operation is aborted and after that the corresponding file handle is closed. Please refer to the description of abort for each function block for further details.

Usage of CAA_File function blocks and online changes

There are 2 function blocks in CAA_File.lib which use ADR operator to pass parameter from IEC application to function block. During design and maintenance of IEC application which uses these function block it is necessary to consider behavior of these function blocks during online changes.

Example of a simple program:

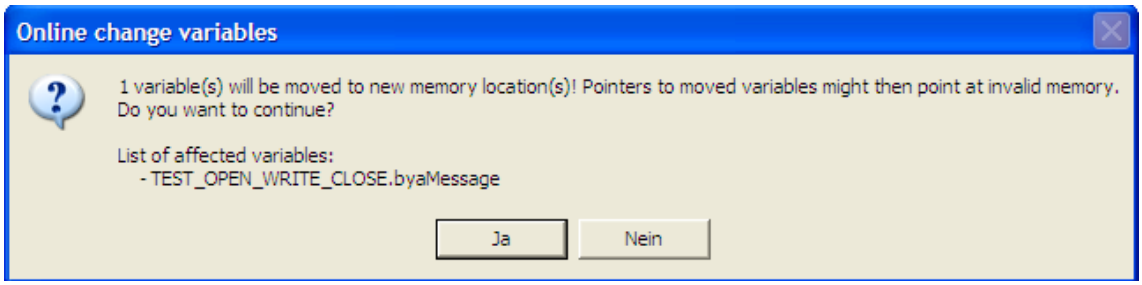


As known during online change variables in IEC application could be relocated to different memory position. In the example, variable byaMessage could be relocated as well. All file operations which are invoked by CAA_File function block are executed in separate low priority task and pointer to read buffer (byaMessage in the example) is saved in context of this task during all file operation.

It means that if operation FILE_Write is not finished before online change some data could be lost or damaged, because pointer to buffer which is used for execution of file operation is not valid anymore.

The following recommendation could be done how to use FILE_Write and FILE_Read function block to avoid any problems after online change:

- Global variable or variable which is declared inside POU Program is only relocated if size of variable is changed. Any variable which is declared in POU function block is relocated even if data structure of function block is changed, e.g. any new variable is added to function block, any variable is deleted etc. So it is recommended to use for read/write buffer global variables and keep their size constant during online change or use variables which are declared in function block, but keep the structure of function block data unalterable.
- If any variable in application is relocated during online change the following message appears



Please check if any variable in your project which is used as write or read buffer does not present in this list. If variable is nevertheless here please check that xBusy output of appropriate function block is set to FALSE. If so, then online change is absolutely safe, otherwise you may meet the problems which are described above and it is recommended to avoid online change and postpone it for a while, till xBusy will not be set to TRUE.

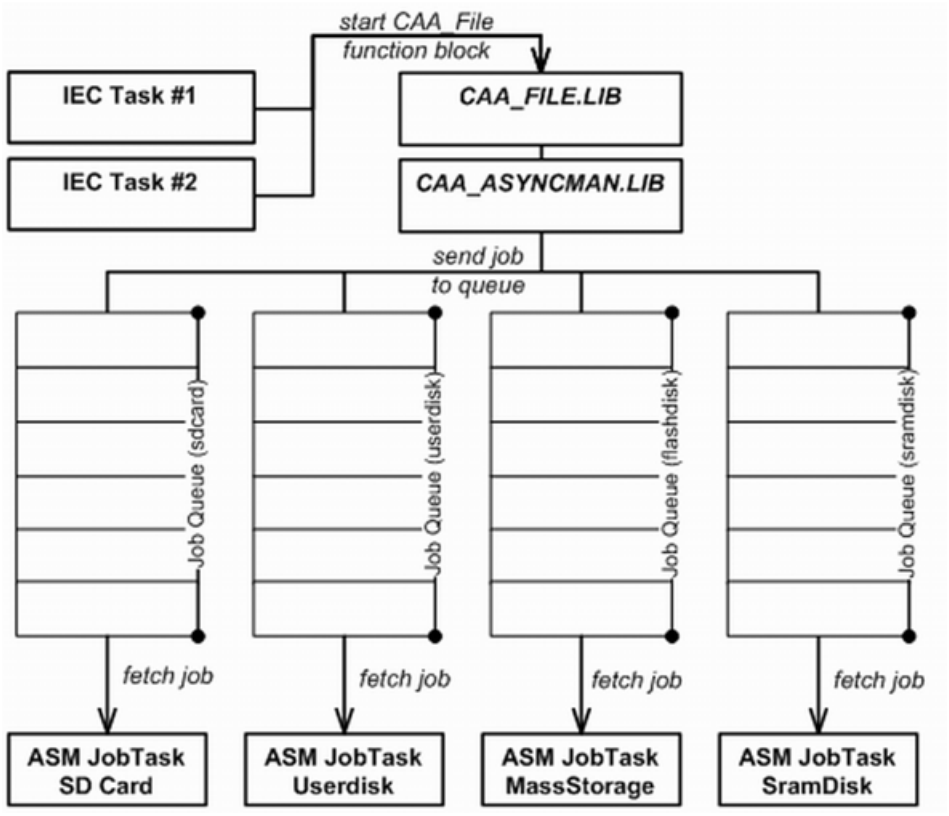
Implementation notes for CAA_File.lib

- Ensure that the file/directory handle is returned to the system (with FILE_Close/FILE_DirCLose) before discarding the local variable holding it.
- You cannot open a file with write access simultaneously.
- Only a successfully call to FILE_Close or FILE_Flush updates a file's representation (i.e. size & content) on a device completely and definitely.
- All outputs of all function blocks will be invalidated, if xExecute is set to FALSE, thus they have to be read previously.

- Differing from Standard ABB libraries, CAA function blocks set EITHER xDone (i.e. FB finished without error) OR xError (i.e. FB finished with error).
- The path separator is a single backslash („\“), not a double backslash („\\“) or a forward slash („/“).
- Both pointer and data contents handed over to read/write function blocks have to be stable during operation!
- CAA_File.lib works in binary mode, thus it reads and writes bytewise. There is no automatic null-termination of the data of any kind, i.e. everything is a byte array: You hand over the starting address of the array and the number of bytes you want to have read/written from/to the array with the given starting address. The interpretation of the read data is up to the user application.
- Files and directory handles are only auto-closed in case of a application download or reset. A stop or online change of the IEC application does NOT close any file handles.

Notes on CAA_AsyncMan .lib

CAA_File.lib requires the additional library CAA_AsyncMan.lib. This library provides the means to handle all CAA_File.lib function blocks completely asynchronous. For each call to a CAA_File function block a so called "job" will be created. One task handles the job requests sent by the IEC application for each file I/O module in the AC500 PLC. These tasks run with a low priority, so the available file throughput depends on the other tasks running on the PLC - i.e. If the PLC is already completely busy doing communication, handling I/O Modules or calculation IEC tasks, the file operations might not be carried out. If a file I/O module is currently busy with an ongoing job request, new requests are queued and executed in a FIFO (“first in, first out”) manner.



Thus to understand the AC500’s file access system behavior the user has to keep in mind that:

- Every file operation is done asynchronously in a separate worker task.
- There is one worker task per I/O module (e.g. userdisk, memory card,…).
- The worker tasks have a low priority and thus might be blocked by other PLC processes.
- Each worker task has a FIFO-queue for the incoming jobs (i.e. triggered function blocks) for its I/O module.
- Most jobs cannot be interrupted or aborted once they are started (exceptions see function block descriptions).

These aspects result in the danger of priority inversion:

If a low priority task already triggered a file access job and another task with higher priority tries to access the same I/O module, the task with higher priority still will have to wait for the task with low priority to complete its job, i.e. the task priorities are inverted.

CAA_AsyncMan.lib can handle a maximum of 20 active function blocks in parallel. In this context, "active" means function block instances triggered via a rising edge and not yet reset by a falling edge on their input xExecute. If this limit is exceeded, additional triggered function blocks will return with undefined errors (e.g. "Invalid error 5802").

1.5.4.4.2 Function blocks

Error messages

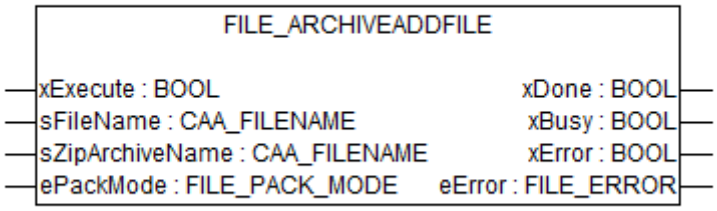
If a function block returns an "invalid" (i.e. internal) error code which is not listed here, check for a possible version mismatch between PLC firmware and the CAA File Libraries. If the function block is supported definitely by the firmware version in the PLC, the IEC application might have too many active function blocks in parallel (maximum number is 20).

Function blocks of CAA_File.lib can return the following error codes:

Value	Error Name	Description
0	FILE_NO_ERROR	No error
5101	FILE_TIME_OUT	Time limit exceeded
5102	FILE_ABORT	Order has been aborted by activating input xAbort
5103	FILE_HANDLE_INVALID	Invalid handle
5104	FILE_NOT_EXIST	Directory or file does not exist
5105	FILE_EXIST	Directory or file already exists
5106	FILE_NO_MORE_ENTRIES	No further entries are available
5107	FILE_NOT_EMPTY	File or directory is not empty
5108	FILE_READ_ONLY_CAA	Drive, file or directory is write-protected
5109	FILE_WRONG_PARAMETER	Wrong parameter(s) at function block
5110	FILE_ERROR_UNKNOWN	Unknown error
5111	FILE_WRITE_INCOMPLETE	Not all data has been written
5112	FILE_NOT_IMPLEMENTED	Function not supported
5113	FILE_NO_RESSOURCES	No file handles or user tasks available (too many files open or tasks accessing file system)
5114	FILE_NO_SPACE	Volume is full
5115	FILE_NO_DEVICE	Cannot open disk or no valid disk present
5151	FILE_CANT_BE_OPENED	File/directory could not be accessed to execute operation
5152	FILE_ERROR_OTHER	Internal or not further specified error
5153	FILE_SOURCE_CANT_BE_OPENED	Source file could not be opened to execute required operation
5154	FILE_DEST_CANT_BE_OPENED	Destination file could not be opened to execute required operation
5155	FILE_CANT_WRITE	Data could not be written to destination file

Value	Error Name	Description
5156	FILE_CANT_READ	Data could not be read from source file
5157	FILE_CONFIRM_FORMAT_OPERATION	Formatting operation has not been confirmed at xEnable before activating the function block with xExecute.

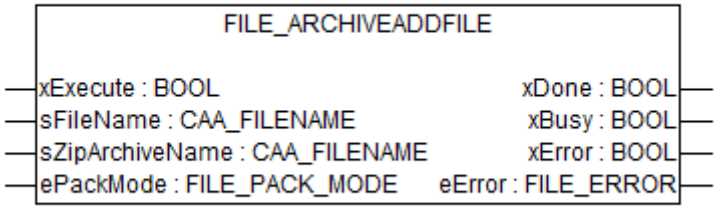
FILE_ArchiveAddFile



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveAddFile adds a file to an archive. If the archive does not exist a new one is created. The file is always added to the archive in uppercase.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

sZipArchive-Name

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sZipArchiveName specifies archive name to unpack.

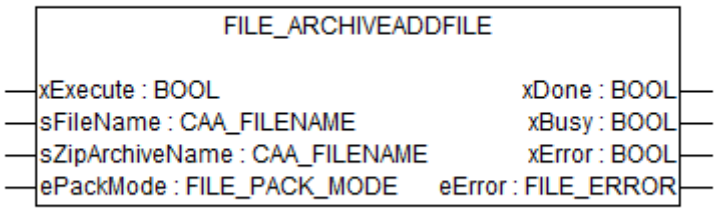
ePackMode

Data type	Default value	Range	Unit
FILE_PACK_MODE	-	-	-

Input ePackMode specifies the mode how a file is packed into archive.

FILE_MODE	Description
FILE_PACK_OVERWRITE	If file in archive already exists it is allowed to overwrite it.
FILE_PACK_WITHOUT_PATH	If this mode is defined then the file will be packed only with its name, otherwise absolute file name is saved in the archive.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

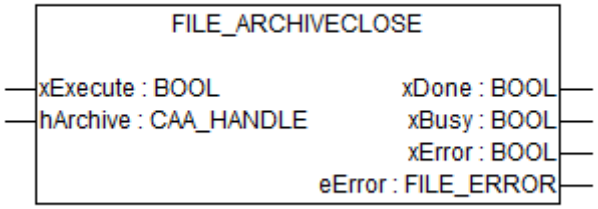
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

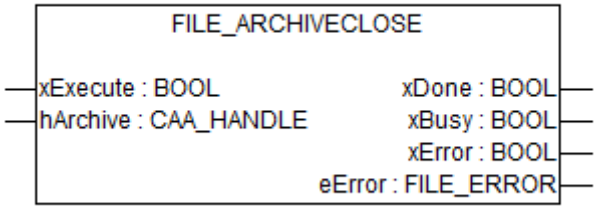
FILE_ArchiveClose



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveClose closes archive. After this operation archive is again available again for packing operation (FILE_ArchiveAddFile).

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

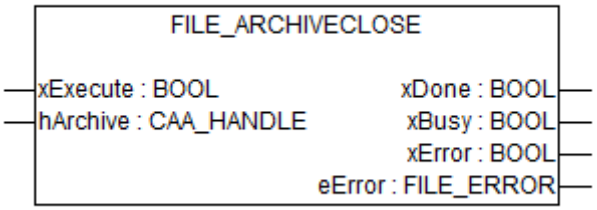
The function block is activated via a positive edge at this input.

hArchive

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

Input hArchive specifies archive handle which has to be closed.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

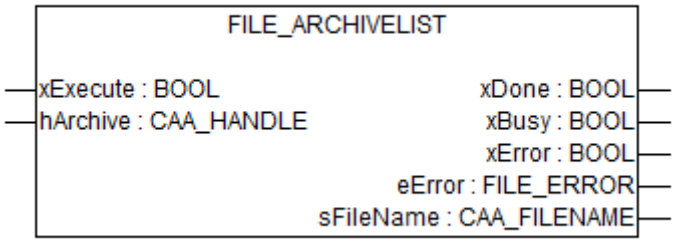
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

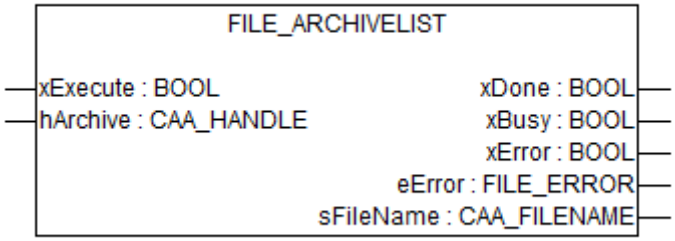
FILE_ArchiveList



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveList returns current file entry in archive. If the function block cannot find any further entries the error message FILE_NO_MORE_ENTRIES is generated. The functionality is similar to the function block FILE_DirList but instead of directory archive is accessed.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

hArchive

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

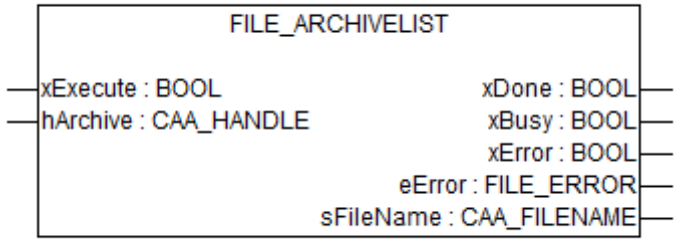
Input hArchive specifies archive handle which has to be closed.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

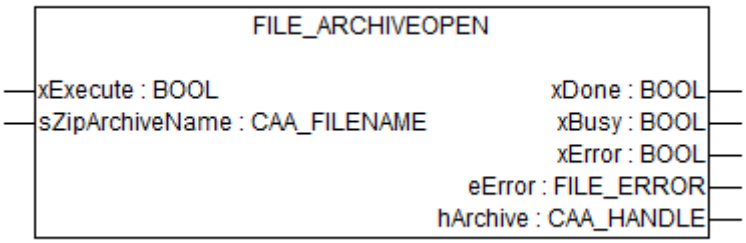
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

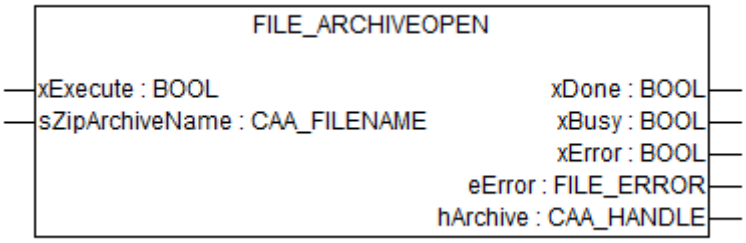
FILE_ArchiveOpen



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveOpen opens an archive and returns the archive handle. Afterwards it is possible to iterate over all files in archive. If archive is opened using this function block all unpack operations (FILE_ArchiveUnpackFile, FILE_ArchiveUnpackFile) could be executed on this archive. Packing operation (FILE_ArchiveAddFile) is not allowed. Archive must be closed first to execute packing operation.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

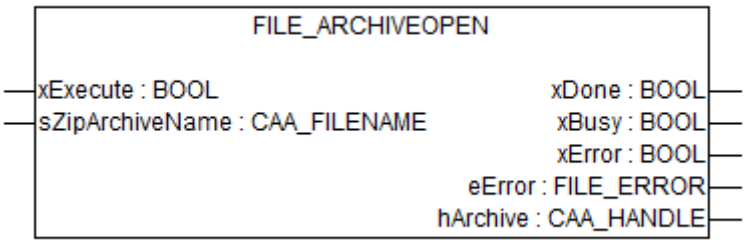
The function block is activated via a positive edge at this input.

sZipArchive-Name

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sZipArchiveName specifies archive name to unpack.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

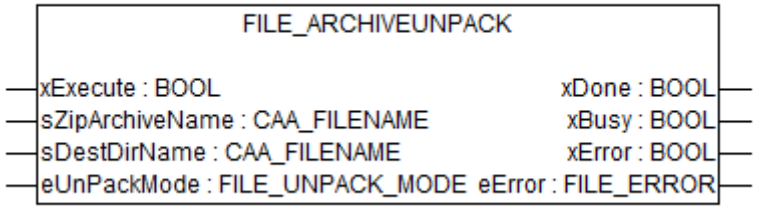
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

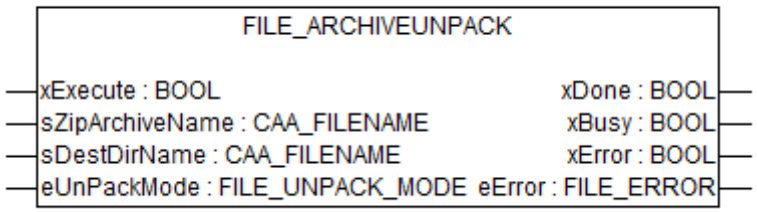
FILE_ArchiveUnpack



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveUnpack unpacks an archive. Function block iterates over all files in archive and unpacks each one. If it is not possible to unpack file error is generated but function block tries to unpack other files in archive. Function block returns error which corresponds to the last failed sub-operation. Only files which are stored with names with 8.3 format could be unpacked.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sZipArchive-Name

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sZipArchiveName specifies archive name to unpack.

sDestDirName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sDestDirName specifies name of destination directory, where archive has to be unpacked.

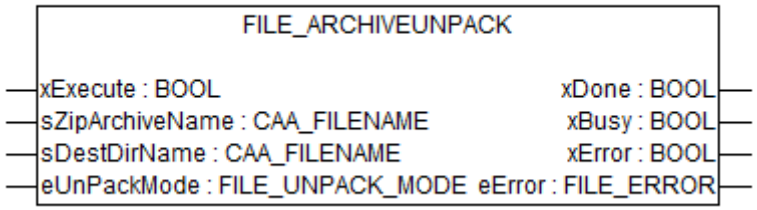
eUnpackMode

Data type	Default value	Range	Unit
FILE_UNPACK_MODE	-	-	-

Input eUnpackMode specifies the mode for archive unpacking operation.

FILE_MODE	Description
FILE_UNPACK_USE_ABSOLUTE_PATH_SAVED_IN_ARCHIVE	If this mode is defined then path to directory is saved in archive. That means sDestDirName is ignored and files are extracted in saved archive directory path, otherwise files are extracted to directory which is specified using sDestDirName. Destination drive name has to be always available in the specified path, i.e. path has to be absolute. If mode is defined but path for any file is not available in archive then this file is not extracted.
FILE_UNPACK_OVERWRITE	If any file in destination directory already exists it is allowed to overwrite it. Input is set to FALSE when only new files are added to destination directory.
FILE_UNPACK_CREATE_DIRS	If destination directory does not exist it will be automatically created. If mode is not set and destination directory does not exist function block returns error. Destination directory could be defined in archive itself if FILE_UNPACK_USE_ABSOLUTE_PATH_SAVED_IN_ARCHIVE is set.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

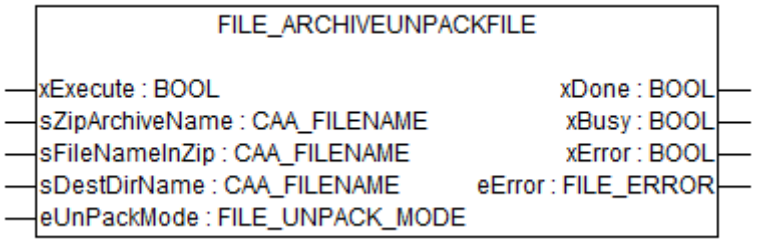
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

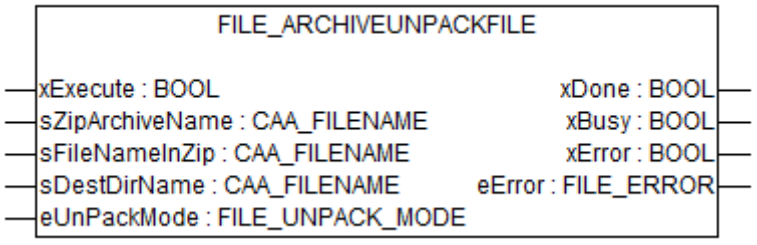
FILE_ArchiveUnpackFile



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Archive Services

FILE_ArchiveUnpackFile unpacks a specific file from an archive. The function block finds the file with name specified by end user and unpacks it to the destination directory sDestDirName. Only files which are stored with names with 8.3 format can be unpacked.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sZipArchive-Name

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sZipArchiveName specifies archive name to unpack.

sDestDirName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sDestDirName specifies name of destination directory, where archive has to be unpacked.

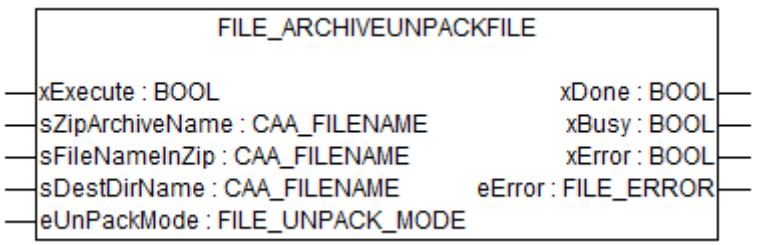
eUnpackMode

Data type	Default value	Range	Unit
FILE_UNPACK_MODE	-	-	-

Input eUnpackMode specifies the mode for archive unpacking operation.

FILE_MODE	Description
FILE_UNPACK_USE_ABSOLUTE_PATH_SAVED_IN_ARCHIVE	If this mode is defined then path to directory is saved in archive. That means sDestDirName is ignored and files are extracted in saved archive directory path, otherwise files are extracted to directory which is specified using sDestDirName. Destination drive name has to be always available in the specified path, i.e. path has to be absolute. If mode is defined but path for any file is not available in archive then this file is not extracted.
FILE_UNPACK_OVERWRITE	If any file in destination directory already exists it is allowed to overwrite it. Input is set to FALSE when only new files are added to destination directory.
FILE_UNPACK_CREATE_DIRS	If destination directory does not exist it will be automatically created. If mode is not set and destination directory does not exist function block returns error. Destination directory could be defined in archive itself if FILE_UNPACK_USE_ABSOLUTE_PATH_SAVED_IN_ARCHIVE is set.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

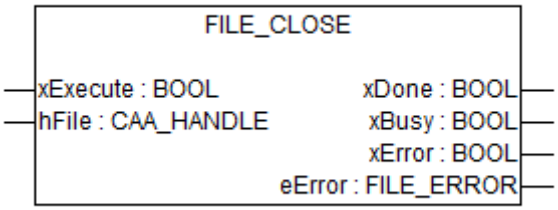
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

FILE_Close

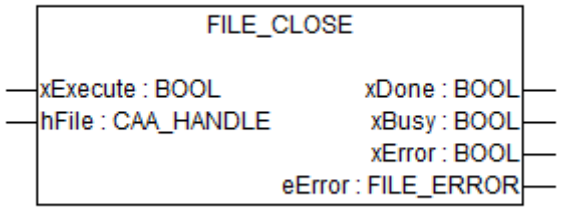


Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Close terminates the file access, i.e. closes the file. If an operation (FILE_Read, FILE_Write, FILE_GetPos, FILE_SetPos) which uses the file handle to be closed is ongoing, it will be aborted before closing file.

Without a correct FILE_Close written data can be lost!

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

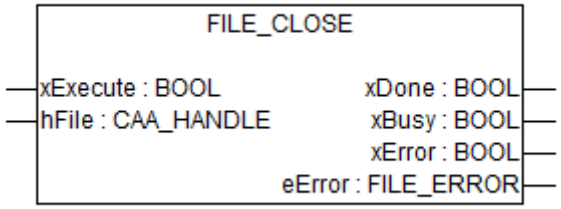
The function block is activated via a positive edge at this input.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

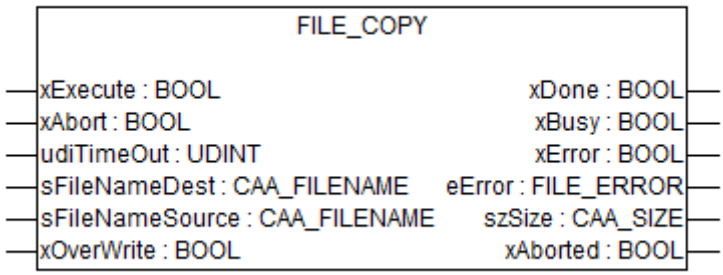
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

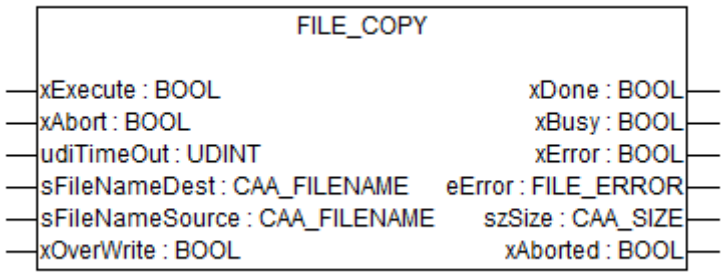
FILE_Copy



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Copy copies a file. The target file is created or - if already existing - overwritten. During the operation a temporary file with the extension .tmp is created. If the copying is aborted by the user the .tmp file will remain on the device.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xAbort

Data type	Default value	Range	Unit
BOOL	-	-	-

Input xAbort stops the Function Block's processing without waiting for it to finish.

udiTimeOut

Data type	Default value	Range	Unit
UDINT	-	-	-

Input udiTimeOut defines the time out in microseconds for the function block to wait on the operation to be finished. A value of 0 means no time out.

sFileNameDest

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Absolute Path of destination.

sFileName-Source

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

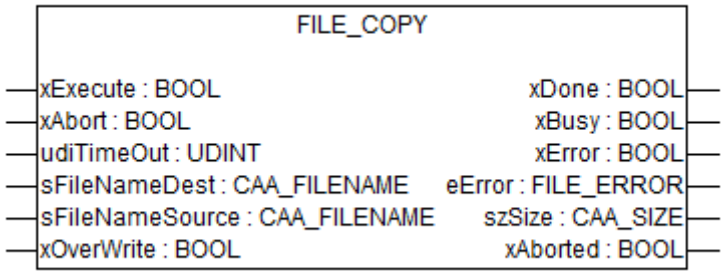
Absolute Path of source.

xOverWrite

Data type	Default value	Range	Unit
BOOL	-	-	-

If input xOverwrite is set to TRUE, the target file will be overwritten, if it exists. Otherwise an error is raised.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

szSize

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

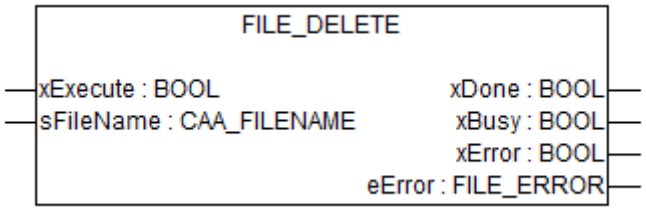
Output szSize is set to the number of bytes copied.

xAborted

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xAborted is set if the function block has been aborted.

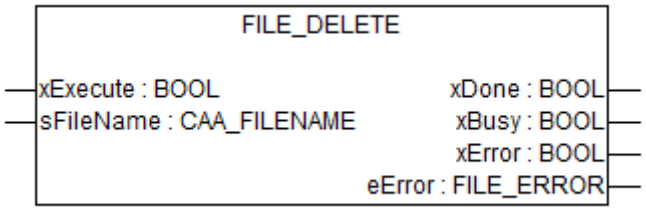
FILE_Delete



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Delete deletes a file. This is not possible if the file is open via FILE_Open.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

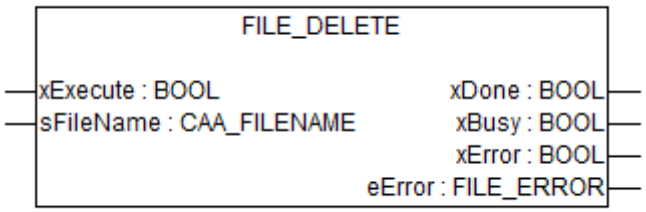
The function block is activated via a positive edge at this input.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

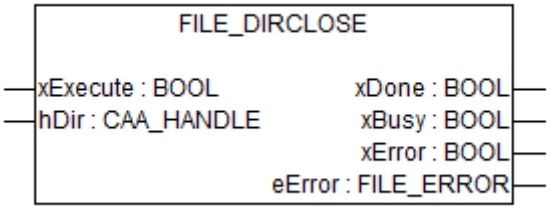
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

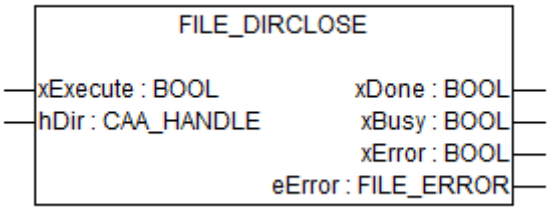
FILE_DirClose



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirClose closes the access on the specified directory.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

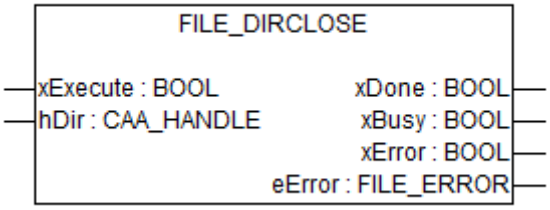
The function block is activated via a positive edge at this input.

hDir

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

Input hDir sets the handle of the directory to close.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

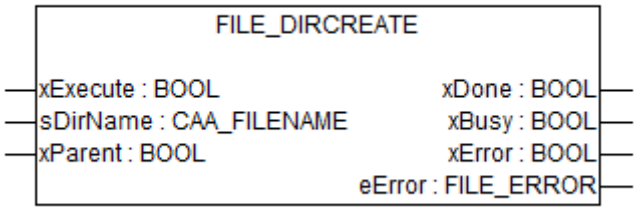
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

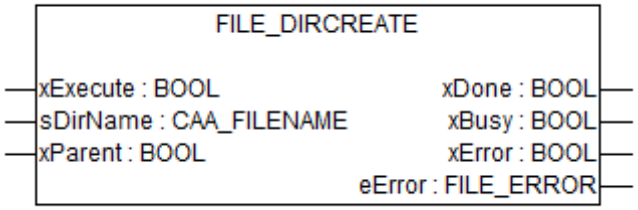
FILE_DirCreate



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirCreate creates a directory. If the directory already exists, an error is generated.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sDirName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

At input sDirName the absolute path and directory name of the new directory has to be specified.

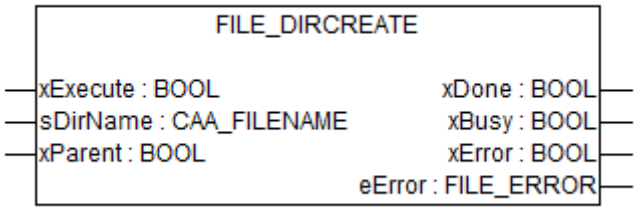
xParent

Data type	Default value	Range	Unit
BOOL	-	-	-

If input xParent = TRUE the full path specified at sDirName is created if it is not existing.

This input is not supported on the AC500 PLCs.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

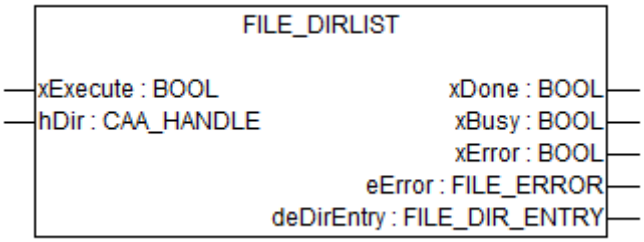
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

FILE_DirList

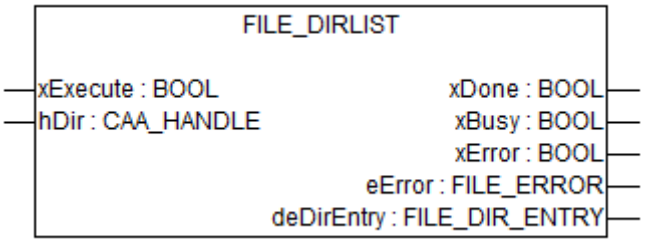


Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirList reads directory or file properties. The function block needs a valid handle which identifies the directory as a transfer parameter. The information is written to structure deDirEntry of type FILE_DIR_ENTRY. If the function block cannot find any further entries the error message FILE_NO_MORE_ENTRIES is generated and the entries in the structure deDirEntry are deleted.

If FILE_DirList is called for an empty volume (e.g. just formatted), an empty file is returned which represents the volume entry of the device. Calling the function block again delivers an error return with code FILE_NO_MORE_ENTRIES.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

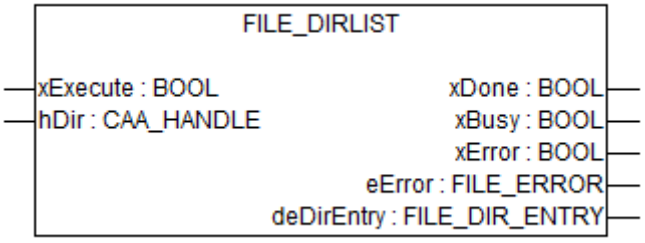
The function block is activated via a positive edge at this input.

hDir

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

Input hDir sets the handle of the directory to close.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

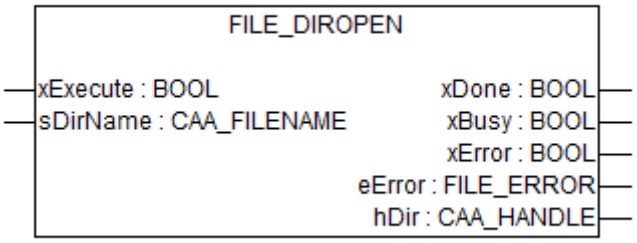
deDirEntry

Data type	Default value	Range	Unit
FILE_DIR_ENTRY	-	-	-

Output deDirEntry is filled with the information on the read directory or file entry:

FILE_DIR_ENTRY		
Element	**Type**	**Description**
sEntry	CAA_FILENAME	Name of the directory or file
szSize	CAA_SIZE	File size
xDirectory	BOOL	Directory or file: TRUE => Directory FALSE => File
xExclusive	BOOL	Access mode on file: TRUE => exclusive access on the file FALSE => multiple access on this file possible
dtLastModification	DT	Date and time of last modification

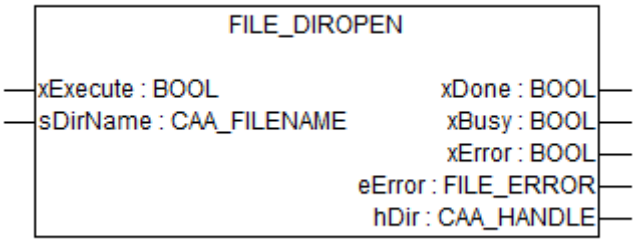
FILE_DirOpen



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirOpen opens a directory which entries (files and sub-directories) should be read with the help of the function block FILE_DirList. The return value is a handle.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

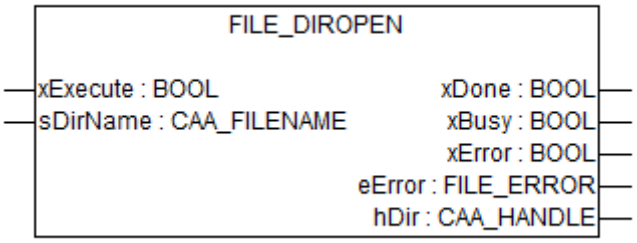
The function block is activated via a positive edge at this input.

sDirName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

At input sDirName the absolute path and directory name of the new directory has to be specified.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

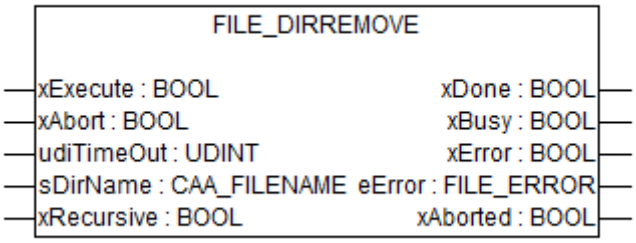
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

hDir

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

Input hDir sets the handle of the directory to close.

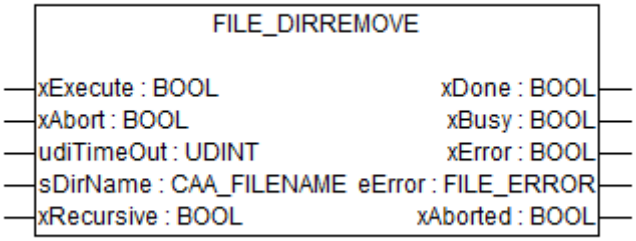
FILE_DirRemove



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirRemove deletes a directory. The directory cannot be deleted if it is opened via FILE_DirOpen or if it is not empty.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xAbort

Data type	Default value	Range	Unit
BOOL	-	-	-

Input xAbort stops the Function Block's processing without waiting for it to finish.

udiTimeOut

Data type	Default value	Range	Unit
UDINT	-	-	-

Input udiTimeOut defines the time out in microseconds for the function block to wait on the operation to be finished. A value of 0 means no time out.

sDirName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

At input sDirName the absolute path and directory name of the new directory has to be specified.

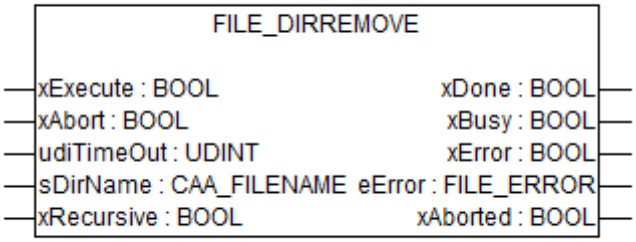
xRecursive

Data type	Default value	Range	Unit
BOOL	-	-	-

If Input xRecursive is set all files and sub-directories within the directory specified at sDirName will be deleted.

This input is not supported on the AC500 PLCs.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

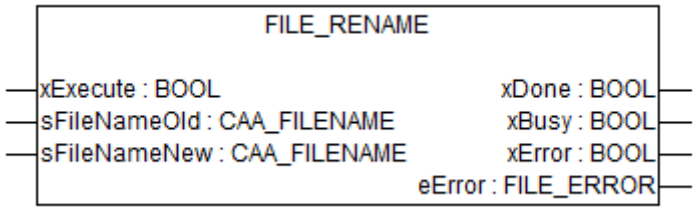
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

xAborted

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xAborted is set if the function block has been aborted.

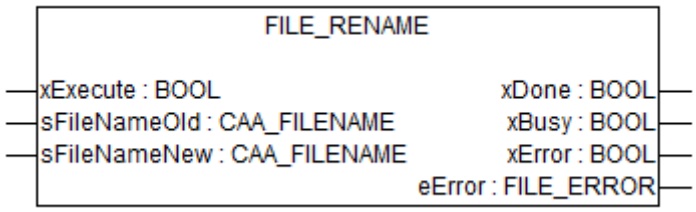
FILE_Rename



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Rename modifies the file name. This does not work if the file is open via FILE_Open. The function block cannot be used to move files or directories!

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sFileNameOld

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

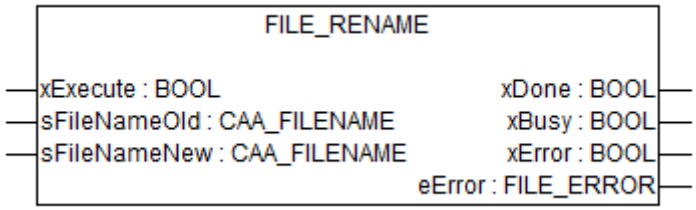
Absolute Path with current file name.

sFileNameNew

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Absolute Path with new file name.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

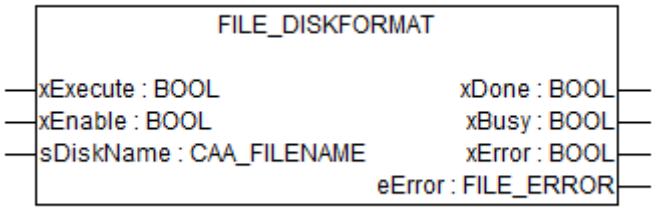
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

FILE_DiskFormat



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.3
Type	Function block with historical values
Group	Storage Device Services

This function block formats a storage device, i.e. erasing all content on it and writing a new, clean file system to it.

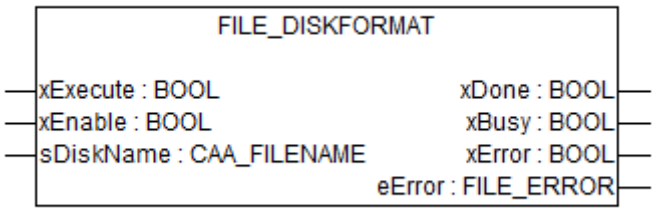
CAUTION!
The format protection (see *Chapter 1.6.5.4.3 “AC500-specific PLC browser commands” on page 6222*) is automatically unlocked and all data on the device will be lost!

The function block "FILE_DiskFormat" can format the "sramdisk", "userdisk" and "flashdisk".

It is not possible to format the "sdcard".

See also AC500 CPU Storage Devices for hints on memory location maintenance.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xEnable

Data type	Default value	Range	Unit
BOOL	-	-	-

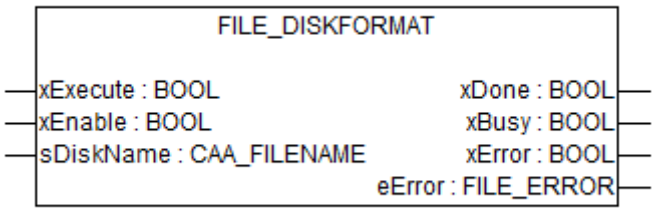
This input must be TRUE before the function block may be triggered via xExecute (i.e. "are you sure"-flag).

sDiskName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Name of storage device to format (e.g. "sramdisk" or "userdisk\").

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

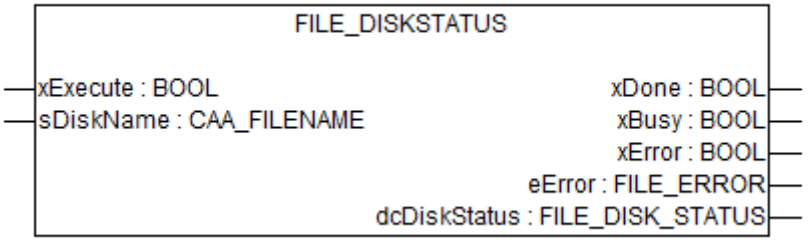
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

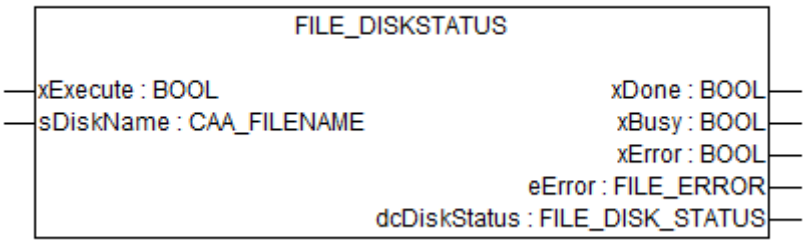
FILE_DiskStatus



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.3
Type	Function block with historical values
Group	Storage Device Services

This function block retrieves information about a storage device.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sDiskName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

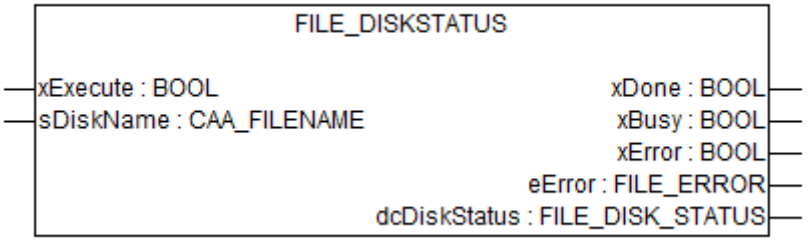
Name of storage device to format (e.g. "sramdisk" or "userdisk\").

sDiskName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Name of storage device to format (e.g. "sramdisk" or "userdisk\").

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

dcDiskStatus

Data type	Default value	Range	Unit
FILE_DISK_STATUS	-	-	-

Output dcDiskStatus contains all available information about the storage device:

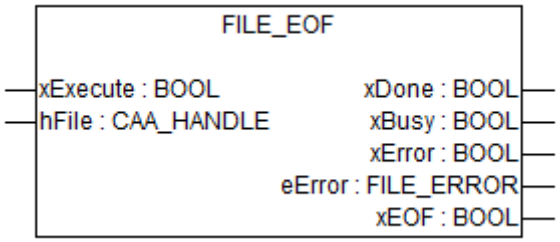
Free, used and total available space on the device, block size, cluster size, total cluster count and the device condition. The meaning of the decive conditions DC_GREEN, DC_YELLOW, DC_RED is storage device specific as shown in the following table:

state	userdisk	flashdisk	SRAM disk	memory card
GREEN	userdisk ready	flashdisk ready	SRAM disk ready	memory card ready
YELLOW	-	threshold exceeded	-	-
RED	userdisk not ready	flashdisk is read only	SRAM disk not ready	memory card not ready

To optimize the usage of a storage device try to fit your read/write operation's sizes and file sizes with the device's block and/or cluster sizes.

See also AC500 CPU Storage Devices for hints on memory location maintenance.

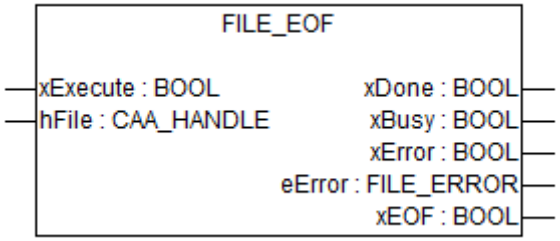
FILE_EOF



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_EOF sets xEOF to TRUE if the current offset is equal to the end of the file. If the end of the file has not yet been reached, FALSE is returned.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

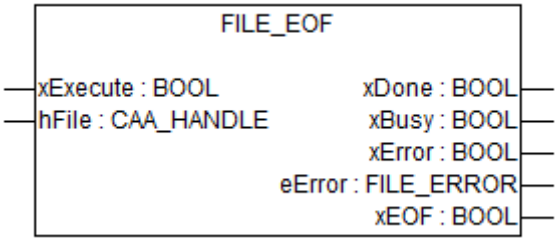
The function block is activated via a positive edge at this input.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

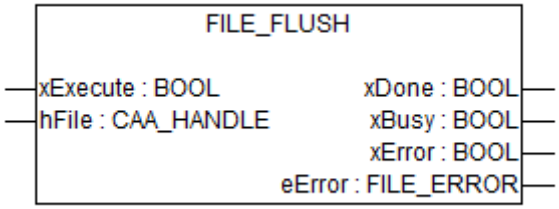
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

xEOF

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xEOF is set if the current position of the file stream pointer is the end of the file.

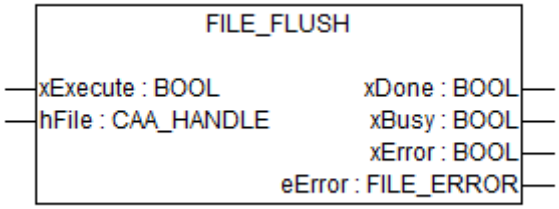
FILE_Flush



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Flush flushes data from the file system cache to the disk for the file with given file handle hFile. Calling this Function Block guarantees that data will be stored on the disk.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

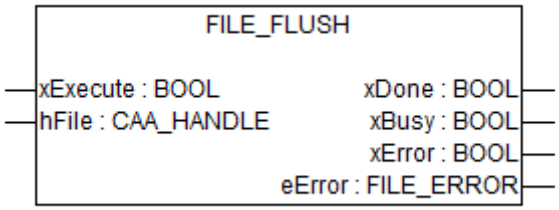
The function block is activated via a positive edge at this input.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

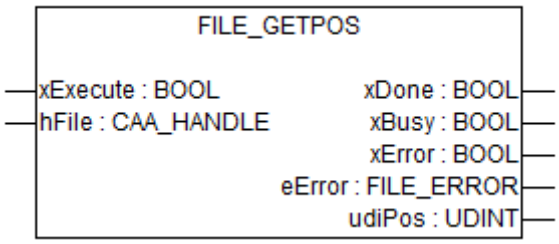
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

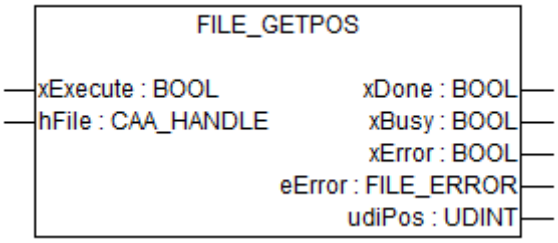
FILE_GetPos



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_GetPos returns the offset position of the file stream pointer currently set in the file. The file must be opened via FILE_Open.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

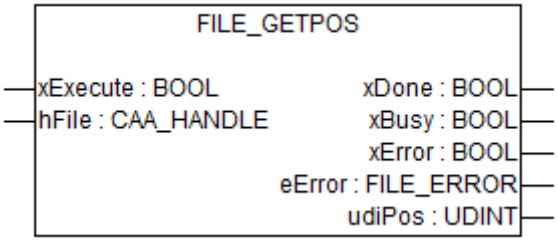
The function block is activated via a positive edge at this input.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

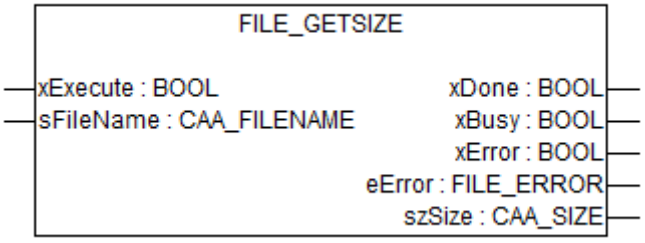
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

udiPos

Data type	Default value	Range	Unit
UDINT	-	-	-

Output udiPos is set to the current offset position i.e. number of bytes from the start of the file.

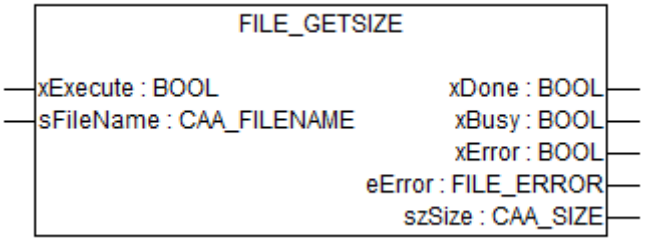
FILE_GetSize



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_GetSize returns the size of the file specified by sFileName. If the file is still open, it must be flushed or closed to get the actual size.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

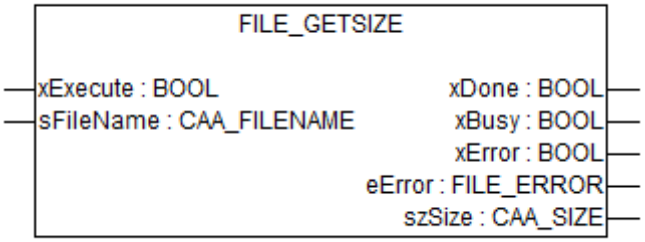
The function block is activated via a positive edge at this input.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

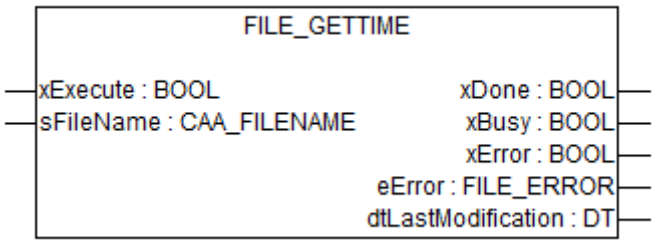
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

szSize

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

Output szSize is set to the number of bytes copied.

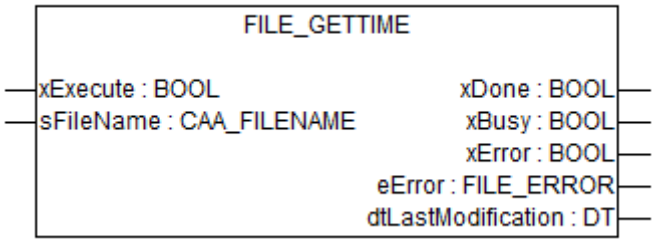
FILE_GetTime



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_GetTime returns date and time of the last modification of the file specified by sFileName.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

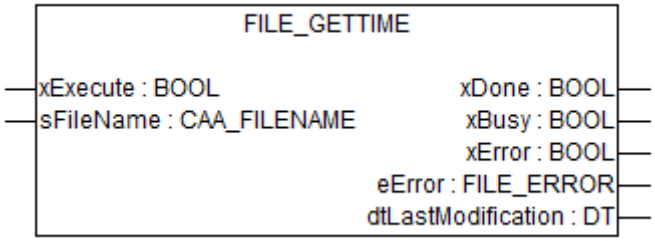
The function block is activated via a positive edge at this input.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

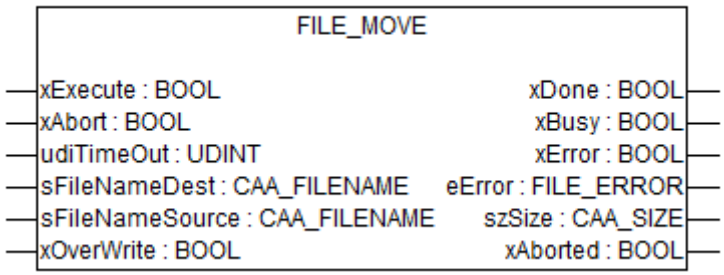
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

dtLastModification

Data type	Default value	Range	Unit
DT	-	-	-

Output dtLastModification is set to the date and time of the last file modification (e.g. dt#2006-05-08-00:00:00).

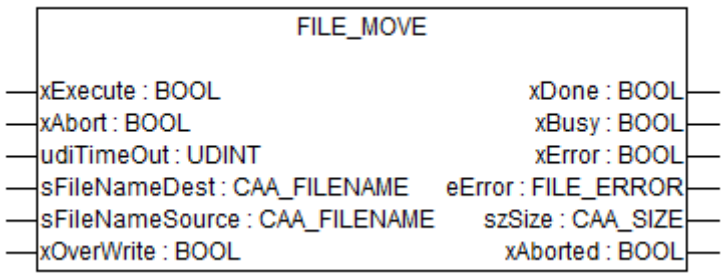
FILE_Move



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.3
Type	Function block with historical values
Group	File

This function block moves a file. The target file is created or - if already existent - overwritten. During the operation a temporary file with the extension .tmp is created. If the moving is aborted by the user the .tmp-file will remain on the device.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xAbort

Data type	Default value	Range	Unit
BOOL	-	-	-

Input xAbort stops the Function Block's processing without waiting for it to finish.

udiTimeOut

Data type	Default value	Range	Unit
UDINT	-	-	-

Input udiTimeOut defines the time out in microseconds for the function block to wait on the operation to be finished. A value of 0 means no time out.

sFileNameDest

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Absolute Path of destination.

sFileName-Source

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

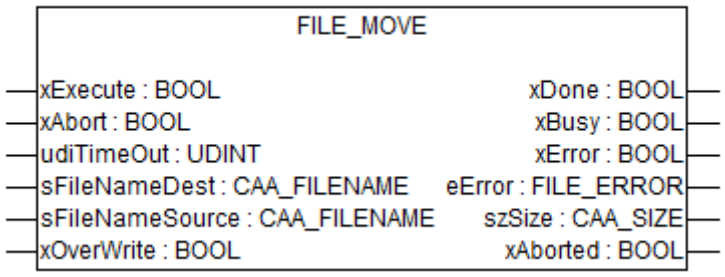
Absolute Path of source.

xOverWrite

Data type	Default value	Range	Unit
BOOL	-	-	-

If input xOverwrite is set to TRUE, the target file will be overwritten, if it exists. Otherwise an error is raised.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

szSize

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

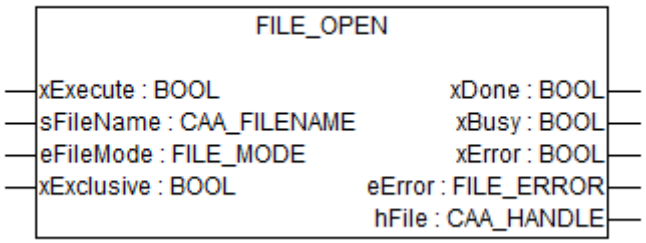
Output szSize is set to the number of bytes copied.

xAborted

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xAborted is set if the function block has been aborted.

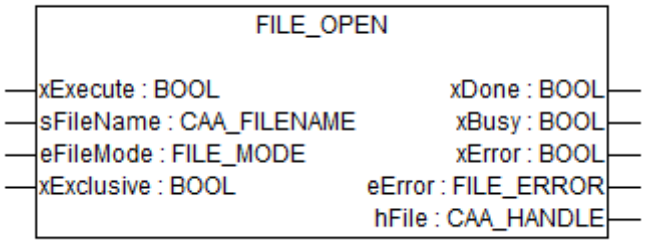
FILE_Open



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Open retrieves a file handle from the drive's file system for further file operations. This function block opens an already existing file or creates a new one depending on the set FILE_MODE at input eFileMode. The return value is a file handle which can be used as an input hFile in the function blocks FILE_Read, FILE_Write, FILE_GetPos, FILE_SetPos, FILE_EOF and FILE_Close.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sFileName

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

Input sFileName specifies an absolute file name which has to be added to archive. The file name appears in archive in uppercase.

eFileMode

Data type	Default value	Range	Unit
FILE_MODE	-	-	-

Input eFileMode determines the opening mode for the file.

"File lock" describes, if a file can be opened again, while it is still opened in one of the modi. "File size" defines how the file size is treated by the open-operation. "File pointer" describes the initial position of the pointer into the file after the open operation.

FILE_MODE	Description
FILE_MWRITE	"write only" - write data to a file. File lock: Cannot be reopened in any other mode. File size: File is createdor existing file content is deleted. File pointer: At beginning of file.
FILE_MREAD	"read only" - read data from a file. File lock: Can be reopened in read-mode. File size: Unchanged, no writing possible. File pointer: At beginning of file.

FILE_MODE	Description
FILE_MRDWR	"read and write" - read and write data to a file. File lock: Cannot be reopened in any other mode. File size: File is created or existing file content is deleted. File pointer: At beginning of file.
FILE_MAPPD	"append to a file" - read and write to a file. File lock: File can be reopened in read mode. File size: File is created (if not existing), existing file content kept. File pointer: At end of file.

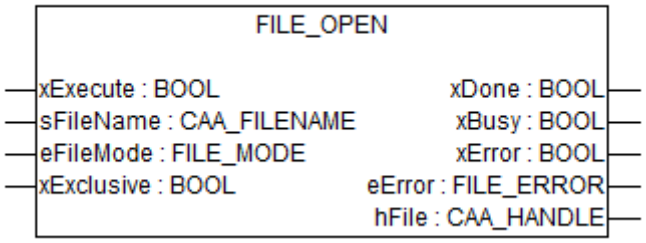
xEclusive

Data type	Default value	Range	Unit
BOOL	-	-	-

Additional input for file access mode.

This input is not supported on the AC500 PLCs.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

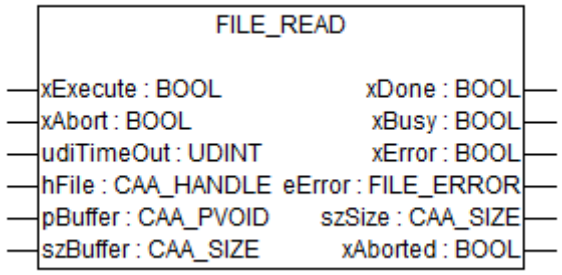
Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

FILE_Read

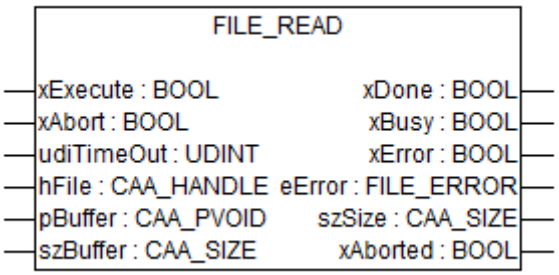


Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Read reads the file which was previously opened via FILE_Open. If less characters can be read than specified in szBuffer, the function block returns an active xDone and indicates the current number of characters in szSize. The size of the target memory structure for the bytes to be read and the number of bytes to be read will not be checked.

The stability of the pointer on the data structures and their contents must be guaranteed if online change is executed. If pointer to data structure is changed during online changes operation could lead to damage of application data.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xAbort

Data type	Default value	Range	Unit
BOOL	-	-	-

Input xAbort stops the Function Block's processing without waiting for it to finish.

udiTimeOut

Data type	Default value	Range	Unit
UDINT	-	-	-

Input udiTimeOut defines the time out in microseconds for the function block to wait on the operation to be finished. A value of 0 means no time out.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

pBuffer

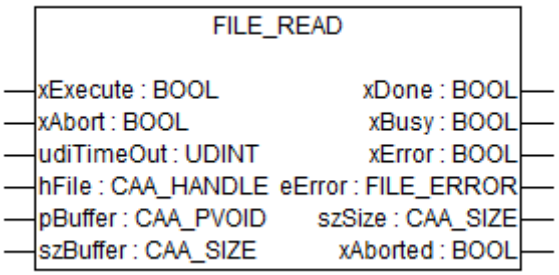
Data type	Default value	Range	Unit
CAA_PVOID	-	-	-

Target address of the buffer where read data will be copied to; can be retrieved via operator ADR.

szBuffer

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

szSize

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

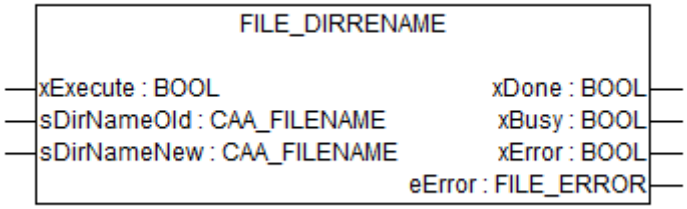
Output szSize is set to the number of bytes copied.

xAborted

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xAborted is set if the function block has been aborted.

FILE_DirRename

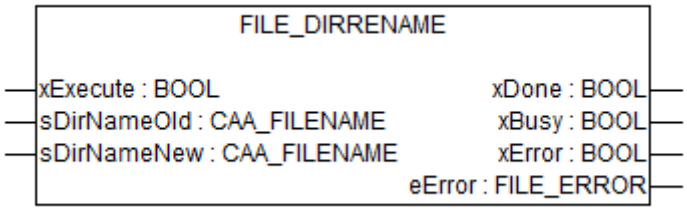


Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	Directory

FILE_DirRename modifies the directory name. This does not work if the directory is opened via FILE_DirOpen. The function block cannot be used to move directories!

CAA_DirRename only supported on the flash disk!

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

sDirNameOld

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

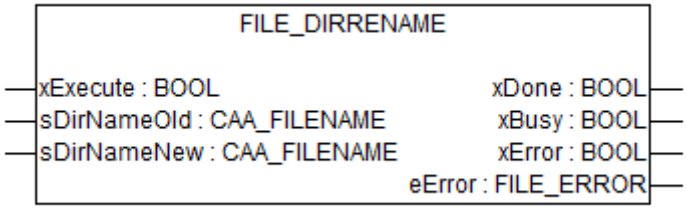
At input sDirNameOld the absolute path with current directory name is set.

sDirNameNew

Data type	Default value	Range	Unit
CAA_FILENAME	-	-	-

At input sDirNameNew the absolute path with new directory name is set.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

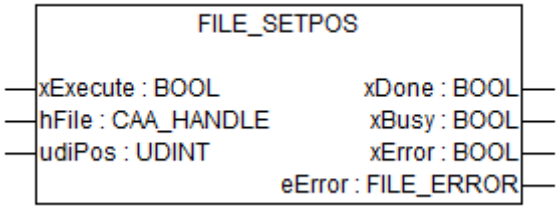
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

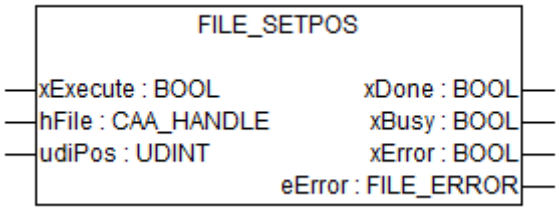
FILE_SetPos



Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_SetPos sets the offset position in bytes of the file stream in the file counting from the beginning of the file. The file must be opened via FILE_Open. If the new position would place the file stream pointer outside the file, an error is returned and the position is not changed.

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

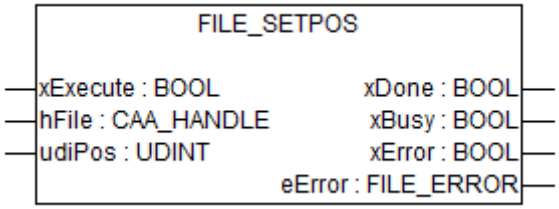
File handle retrieved with CAA_Open.

udiPos

Data type	Default value	Range	Unit
UDINT	-	-	-

Output udiPos is set to the current offset position i.e. number of bytes from the start of the file.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

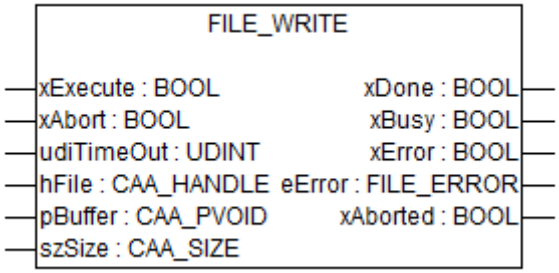
Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

FILE_Write



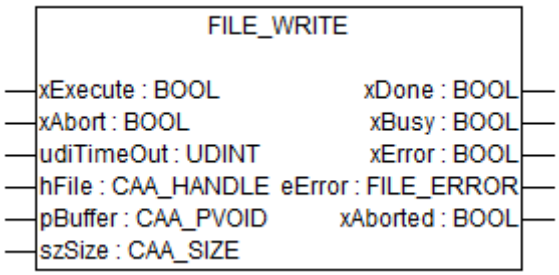
Parameter	Value
Included in library	CAA_File.lib
Available as of firmware	V2.1
Type	Function block with historical values
Group	File

FILE_Write writes data into the file which was previously opened via FILE_Open. The contents of the memory area indicated by pointer pBuffer should not be modified during the write action! The size of the structure of the memory containing the bytes to be written as well as the number of bytes to be written will not be checked. The stability of the pointer on the data structures and their contents must be guaranteed if online change is executed. If pointer to data structure is changed during online changes operation could lead to file damage.

For performance reasons the file system buffers written data. In effect only flushing or closing a file updates its representation on the device completely.

Without a correct FILE_Close written data will be lost!

Input description



xExecute

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated via a positive edge at this input.

xAbort

Data type	Default value	Range	Unit
BOOL	-	-	-

Input xAbort stops the Function Block's processing without waiting for it to finish.

udiTimeOut

Data type	Default value	Range	Unit
UDINT	-	-	-

Input udiTimeOut defines the time out in microseconds for the function block to wait on the operation to be finished. A value of 0 means no time out.

hFile

Data type	Default value	Range	Unit
CAA_HANDLE	-	-	-

File handle retrieved with CAA_Open.

pBuffer

Data type	Default value	Range	Unit
CAA_PVOID	-	-	-

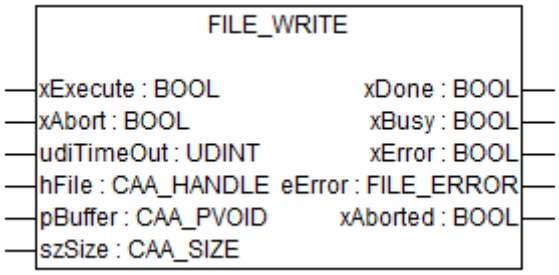
Target address of the buffer where read data will be copied to; can be retrieved via operator ADR.

szSize

Data type	Default value	Range	Unit
CAA_SIZE	-	-	-

Output szSize is set to the number of bytes copied.

Output description



xDone

Data type	Default value	Range	Unit
BOOL	-	-	-

If Output xDone changes to TRUE, the function block has finished its processing without an error and all other outputs become valid.

xBusy

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xBusy signals if a function block has been triggered and is TRUE as long it is processing.

xError

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xError is set if the function block finished and an error occurred during the Function Block processing. If xError is TRUE the error code can be read from output eError.

eError

Data type	Default value	Range	Unit
FILE_ERROR	-	-	-

Output eError is set if an error occurred during the function block processing. For error codes, see 'Error Messages of the CAA_File function blocks' ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*.

xAborted

Data type	Default value	Range	Unit
BOOL	-	-	-

Output xAborted is set if the function block has been aborted.

1.5.4.4.3 Data types

Data type	Base type	Description	Note
CAA_FILENAME	STRING(256)	Used for file names and paths	
FILE_ERROR	INT	Error codes of CAA_File.lib function blocks	See Error messages ↳ *Chapter 1.5.4.4.2.1 “Error messages” on page 793*
FILE_PACK_MODE	DWORD	Options for packing data to an archive	FILE_PACK_WITHOUT_PATH FILE_PACK_OVERWRITE
CAA_HANDLE	DWORD	File / directory handle for function block operations	-
FILE_UNPACK_MODE	DWORD	Options for unpacking data from an archive	FILE_UNPACK_USE_ABSOLUTE_PATH_SAVED_IN_ARCHIVE FILE_UNPACK_OVERWRITE FILE_UNPACK_CREATE_DIRS
CAA_SIZE	UDINT	File size type	In Bytes
FILE_DIR_ENTRY	STRUCT	Structure with information about directory entry	For details, see function block FILE_DirList ↳ *Chapter 1.5.4.4.2.13 “FILE_DirList” on page 815*
FILE_DISK_STATUS	STRUCT	Structure with information about status	For details, see function block FILE_DiskStatus ↳ *Chapter 1.5.4.4.2.18 “FILE_DiskStatus” on page 826*
FILE_MODE	ENUM	Mode for opening a file with function block FILE_OPEN ↳ *Chapter 1.5.4.4.2.25 “FILE_Open” on page 839*	FILE_MWRITE FILE_MREAD FILE_MRDWR FILE_MAPPD

Data type	Base type	Description	Note
FILE_ATTRIB	ENUM	File attribute	FILE_ARCHIVE FILE_HIDDEN FILE_NORMAL FILE_READONLY
FILE_DISK_CONDITION	ENUM	State of storage device	For details, see function block FILE_DiskStatus *Chapter 1.5.4.4.2.18 “FILE_DiskStatus” on page 826*

1.5.4.5 Camswitch library

Library file name: **CAMSWITCH_AC500_Vx.lib**

System requirements

Following libraries are needed:

- CODESYS 2.3.9.3 or later
- SysExt_AC500_V10.lib or later
- SysLibTime.lib Version 2.4.0.6 or later

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The MC function blocks represent additional functionality. They are not required for normal EtherCAT operation.

The MC function blocks functionalities are used exclusively by CI511-ETHCAT and CI512-ETHCAT.

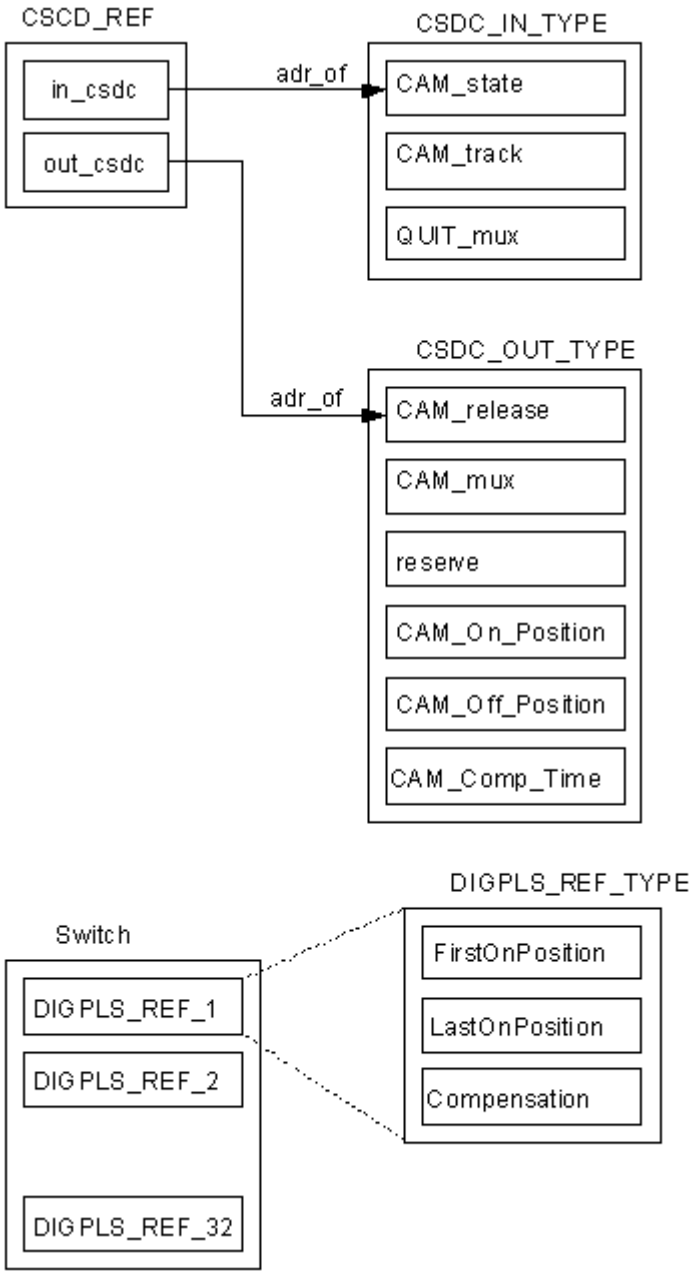
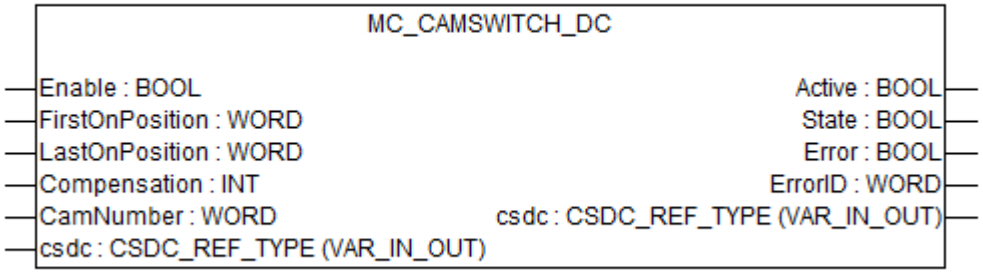


Fig. 13: Relation of the data types for the cam switch

1.5.4.5.1 Function blocks

MC_CamSwitch_DC



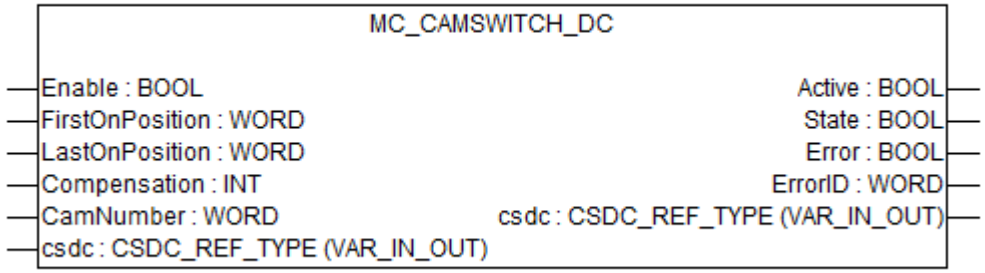
Parameter	Value
Included in library	CAMSWITCH_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Group/Subgroup

Input description

MC_CamSwitch_DC provides the possibility to switch a cam on/off at projected positions. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the switch-off-position are individually changeable online. Further more a compensation time in both directions can be added to the switch position. Compensation is applied to both switch-on-position and switch-off-position.

The actual state of the cam can also be monitored with the output “state”.



Enable

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

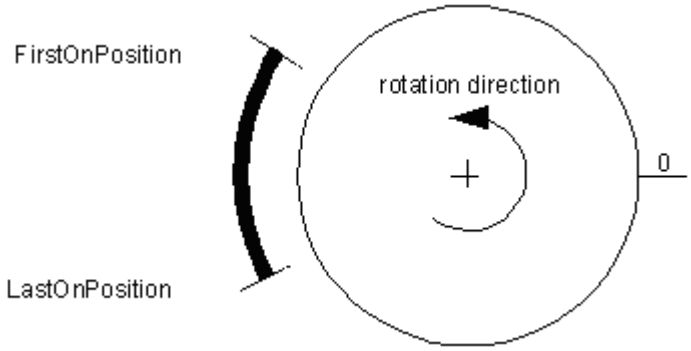
LastOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

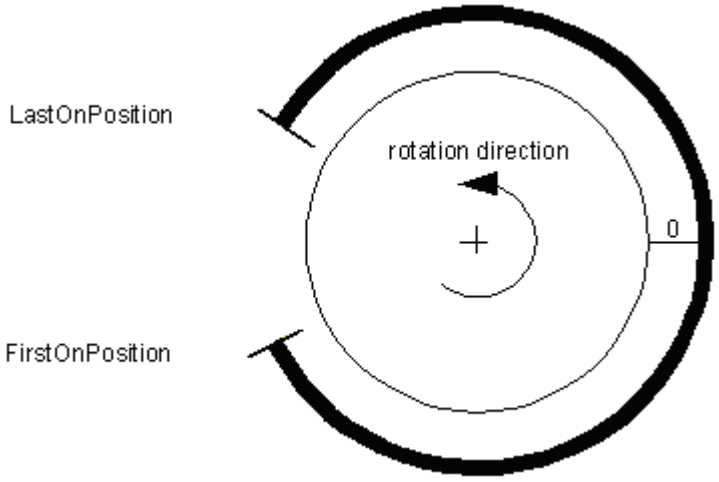
The value of LastOnPosition is automatically truncated to the resolution of the circle without notice. For example, all LastOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

LastOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

If FirstOnPosition is smaller than LastOnPosition, the cam is switched on between the values of FirstOnPosition and LastOnPosition of the same turn.



If FirstOnPosition is greater than LastOnPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and LastOnPosition of the next turn.



If FirstOnPosition is equal to LastOnPosition, the cam is switched on for one interrupt cycle.

Compensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	-	-

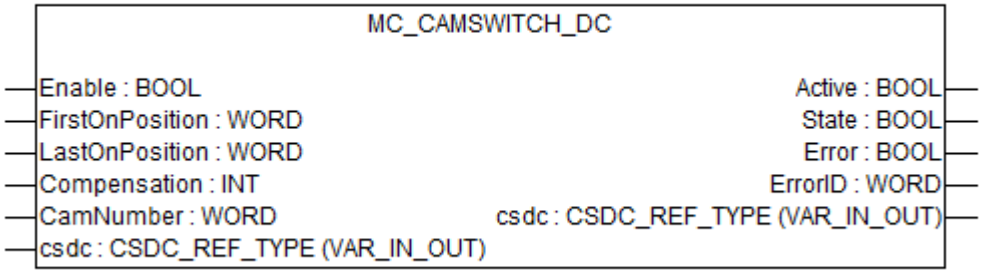
Number of the cam; the parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type	Default value	Range	Unit
DWORD	-	-	-

See data type CSDC_REF_TYPE.

Ouput description



Active

Data type	Default value	Range	Unit
BOOL	-	-	-

Parameters (FirstOnPosition, LastOnPosition, Compensation) are transferred during active is TRUE.

State

Data type	Default value	Range	Unit
DWORD	-	-	bit

The 32 bits of state correspond to the states of 32 cams of a CI51x-ETHCAT Module. If a bit is set, the corresponding cam is on, otherwise the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	-	-

The output ERR=TRUE indicates that the communication to the device is not working properly.

ErrorID

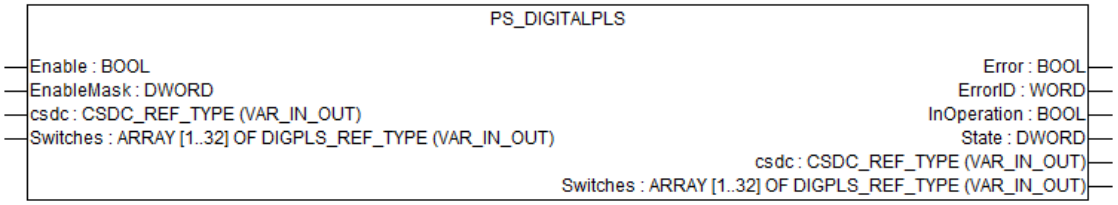
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

Function call in ST

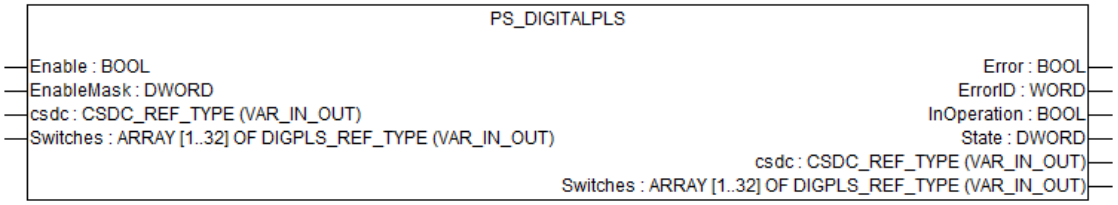
```
cs1.in_csdc  := ADR(CI512_INPUTS_CAM_STATE);
cs1.out_csdc := ADR(CI512_OUTPUTS_CAM_RELEASE);


CAM1 (
cscd              := cs1,
FirstOnPosition   := CAM1_FirstOnPosition,
LastOnPosition    := CAM1_LastOnPosition,
Compensation      := CAM1_Compensation,
CamNumber         := CAM1_CamNumber);


CAM1_Active      := CamSwitch.active;
CAM1_State       := CamSwitch.state;
CAM1_Error       := CamSwitch.error;
CAM1_ErrorID     := CamSwitch.error_id;
```

PS_DigitalPLS



Parameter	Value
Included in library	CAMSWITCH_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Group/Subgroup

Input description



PS_DigitalPLS provides the possibility to control all the cams on an ABB CI51x-ECAT.

The switch-on and switch-off-position of each cam and the compensation time of each cam can be set individually via the input Switches. The position of the reference axis is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on position and the switch-off-position can be changed individually online. Furthermore, a compensation time in both directions can be added to the switch position. Compensation is applied to both switch-on-position and switch-off-position.

Individual cam can be disabled by clearing the corresponding bit in the EnableMask.

The actual state of the cams can also be monitored with the output “State”.

Enable

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

EnableMask

Data type	Default value	Range	Unit
DWORD	-	-	-

The 32 bits of EnableMask correspond to the 32 cams of a CI51x-ETHCAT device. Setting a bit will enable the corresponding cam.

csdc

Data type	Default value	Range	Unit
DWORD	-	-	-

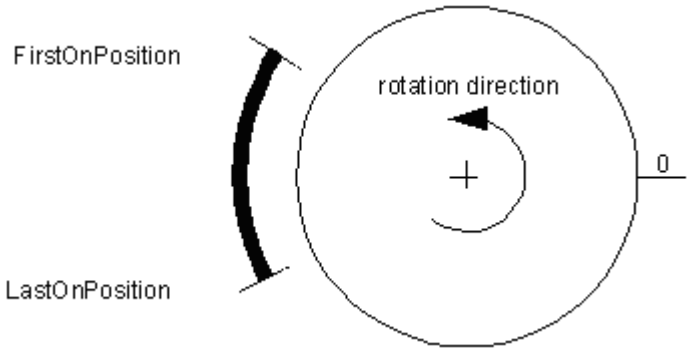
See data type CSDC_REF_TYPE.

Switches OF DIGPLS_REF_TYPE

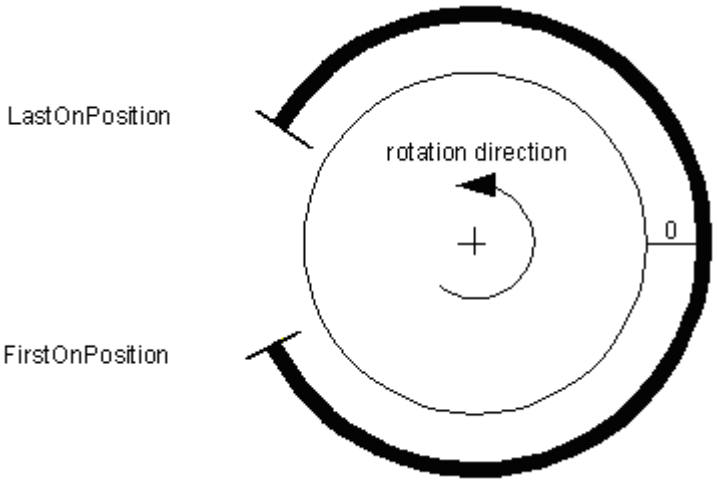
Data type	Default value	Range	Unit
ARRAY	-	1...32	-

See data type DIGPLS_REF_TYPE.

If FirstOnPosition is smaller than LastOnPosition, the cam is switched on between the values of FirstOnPosition and LastOnPosition of the same turn.



If FirstOnPosition is greater than LastOnPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and LastOnPosition of the next turn.

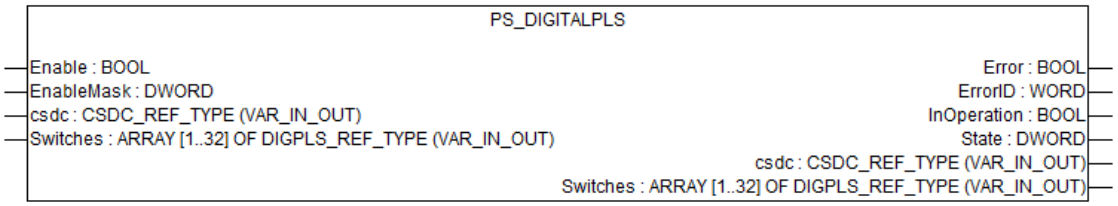


If FirstOnPosition is equal to LastOnPosition, the cam is switched on for one interrupt cycle.

The POU PS_DigitalPLS automatically truncates FirstOnPosition and LastOnPosition to the resolution of the circle without notice. For example, all FirstOnPosition/LastOnPosition which are greater than 36000 will be limited to 36000 (given that the default resolution of CI51x-ETHCAT cam is used).

The parameters FirstOnPosition, LastOnPosition and Compensation can be changed at any time. They will be immediately valid after sending to the CI51x-ETHCAT via the POU.

Output description



Error

Data type	Default value	Range	Unit
BOOL	-	-	-

The output ERR=TRUE indicates that the communication to the device is not working properly.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

InOperation

Data type	Default value	Range	Unit
BOOL	-	-	-

TRUE if the POU is running.

State

Data type	Default value	Range	Unit
DWORD	-	-	bit

The 32 bits of state correspond to the states of 32 cams of a CI51x-ETHCAT Module. If a bit is set, the corresponding cam is on, otherwise the cam is off.

Function call in ST

```
cs1.in_csdc := ADR(CI512_INPUTS_CAM_STATE);
cs1.out_csdc := ADR(CI512_OUTPUTS_CAM_RELEASE);


PLS1 (
Enable       := PLS1_Enable,
EnableMask   := PLS1_EnableMask,
cscd         := cs1,
Switches     := PLS1_Switches);



PLS1_Error := PLS1.Error;
PLS1_ErrorID := PLS1.ErrorID;
PLS1_InOperation := PLS1.InOperation;
PLS1_State := PLS1.State;
```

1.5.4.5.2 Data types

CSDC_REF_TYPE

The library contains the data type CSDC_REF_TYPE to facilitate access to the process data image of the I/O module, on which the cams are located. This data type is declared as follows:

```
TYPE CSDC_REF_TYPE :
STRUCT
number   : INT;
name     : STRING;
in_csdc  : POINTER TO CSDC_IN;
out_csdc : POINTER TO CSDC_OUT;
END_STRUCT
END_TYPE
```

number

Data type	Default value	Range	Unit
INT	-	-	-

Can optionally be used to identify the EtherCAT slave.

name

Data type	Default value	Range	Unit
INT	-	-	-

Can optionally be used to identify the EtherCAT slave.

in_csdc

Data type	Default value	Range	Unit
DWORD	-	-	-

in_csdc is the address of the first element of the input data. For the CI51x-ETHCAT it shall be initialized with the address of the parameter CAM_state (see type CSDC_IN_TYPE ↳ *Chapter 1.5.4.5.2.2 “CSDC_IN_TYPE” on page 861*).

out_csdc

Data type	Default value	Range	Unit
DWORD	-	-	-

out_csdc is the address of the first element of the output data. For the CI51x-ETHCAT it shall be initialized with the address of the parameter CAM_release (see type CSDC_OUT_TYPE ↳ *Chapter 1.5.4.5.2.3 “CSDC_OUT_TYPE” on page 861*).

CSDC_IN_TYPE

The library contains the data type CSDC_IN_TYPE, which contains all the parameters needed to realize the cam switch function. POU accesses to the required inputs in the process data image of the I/O module are performed with these parameters.

The application program shall not access the elements of this structure, so no detail descriptions of the structure elements are given.

Function call in ST

```
TYPE CSDC_IN_TYPE :
(*declaration of I/O-structures for CS-device*)
STRUCT
CAM_state : DWORD;
CAM_track : WORD;
QUIT_mux : BYTE;
END_STRUCT
END_TYPE
```

CSDC_OUT_TYPE

The library contains the data type CSDC_OUT_TYPE, which contains all the parameters needed to realize the cam switch function. Accesses to the required outputs in the process data image of the I/O module are performed with these parameters.

Application program shall not access the elements of this structure, so no detail descriptions of the structure elements are given.

```
TYPE CSDC_OUT_TYPE :
STRUCT
CAM_release      : DWORD;
CAM_mux          : BYTE;
reserve          : BYTE;
CAM_On_Position  : WORD;
CAM_Off_Position : WORD;
CAM_Comp_Time    : WORD;
END_STRUCT
END_TYPE
```

DIGPLS_REF_TYPE

Data type DIGPLS_REF_TYPE describes a cam.

LastOnPosition

Data type	Default value	Range	Unit
REAL	-	-	µs

Position of the referenced axis where the cam is switched off.

Compensation

Data type	Default value	Range	Unit
INT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

Function call in ST

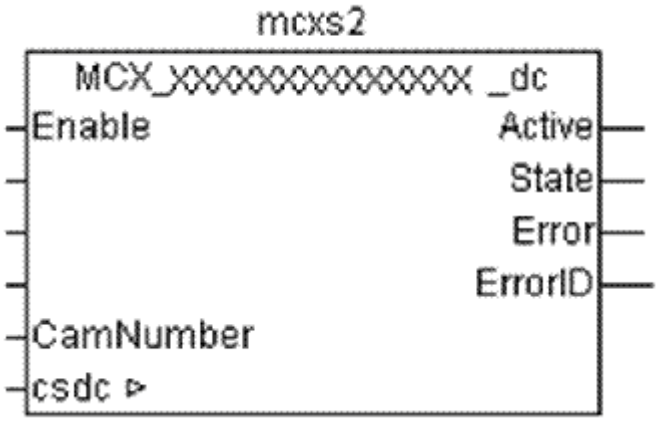
```
TYPE DIGPLS_REF_TYPE :
STRUCT
FirstOnPosition : REAL;
LastOnPosition  : REAL
Compensation    : INT;
END_STRUCT
END_TYPE
```

1.5.4.6 Extended camswitch library

Library file name: **MCX_AC500_Vx.lib**

All function blocks follow the same basic rules which are as follows:

- The “Enable” enables the cam switch. It will be switched off immediately with Enable = FALSE and the respective output will be switched off.
- Every function block has to be assigned to a certain cam switch by specifying the number at input CamNumber. CamNumber runs from 1..32.
- Every function block has to be assigned to a certain device. This is done by “csdc” which holds addresses of the devices process image
- The “active” output indicates that the specific function block communicates with the device. A complete pulse ensures the data has been transferred
- The “state” output indicates that the cam switch is “on” which means the output of the cam is on.



All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

System requirements

Following system requirements have to be fulfilled:

- CODESYS 2.3.9.3 or later
- SysExt_AC500_V10.lib or later
- SysLibTime.lib Version 2.4.0.6 or later

Restrictions

- Compensation time
  - The time is given in a resolution of µs, but the best achievable resolution is the cycle-time of the device (200 µs with 32 cams)
  - For times larger then 32767, the resolution will be increased internally. The user gives still a us-time in a DINT-value
  - Times which are larger then 524 ms will be transferred with a resolution of 512 µs. A compensation time up to 8 s could be reached.
- Best achievable resolution is cycle-time of the device (80 µs / 100 µs / 200 µs).

1.5.4.6.1 Architecture

Identification of binary inputs and outputs

The device has 8 inputs, 8 combined input/outputs (dc) and 8 outputs.

The cam controller holds up to 32 cams which could be assigned to the 8 outputs and 8 DCs. Several cams might be assigned to the same output. The BinaryReference and BinaryShift are assigned to the 8 digital inputs or to the 8 DCs. The inputs and outputs are numbered from 0 to 15, where 0..7 is a DC, the function decides if it is to be used as input or outputs, numbers 8..15 are inputs and outputs and it depends on the function if for example the input 8 or output 8 is used.

	DC (number)		Input (number)		Output (number)
1.0	0	2.0	8	3.0	8
1.1	1	2.1	9	3.1	9
1.2	2	2.2	10	3.2	10

	DC (number)		Input (number)		Output (number)
1.3	3	2.3	11	3.3	11
1.4	4	2.4	12	3.4	12
1.5	5	2.5	13	3.5	13
1.6	6	2.6	14	3.6	14
1.7	7	2.7	15	3.7	15
1.8		2.8			
1.9		2.9			

Architecture and possible composition

The following view shows the general architecture and possible assignments of different cams.

The general rules are:

- Output-cams are cams which are able to directly switch an output. They may be used as stand-alone Cam-Switch or may be combined with a trigger-cam. In this case, not just a position but the respective trigger event will release the cam.
- The output-cams are:
  - CamSwitchSimple (Type 0)
  - PulseSwitch (Type 1)
  - CamSwitchTimed (Type 2)
  - CamSwitchComfort (Type 3)
  - CamSwitchMulti (Type 6)
  - CamSwitchTimeTime (Type 7) (no stand-alone)
  - CamSwitchMultiTime (Type 9)
- The trigger-cams are
  - CamShift (Type 4)
  - Different blocks refer to cams with different types. The type is defined by the parameterization of the cam.
  - The output-cams are assigned to a binary output (by parameterization).
  - The trigger-cams are assigned to a binary input (by parameterization).
  - Trigger-cams are connected to output-cams by programming of the function block in PLC program.
  - The BinaryReference is assigned to a binary input (by parameterization).
  - The Puls-Switch is assigned to a binary output (by parameterization).
  - The logic is assigned to a cam and modifies the output which belongs to this cam.
  - BinaryShift (Type 5), refer to picture.

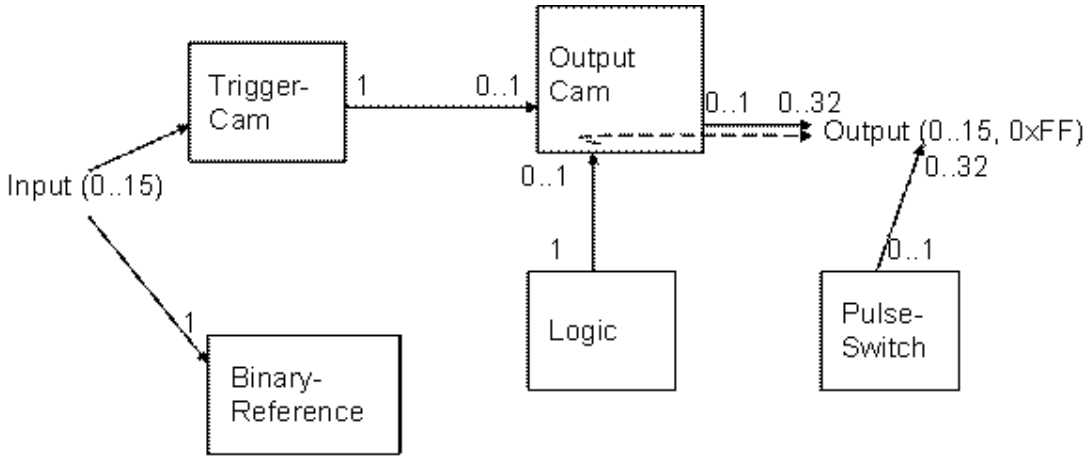


Fig. 14: BinaryShift (Type 5)

1.5.4.6.2 Controller

The device is based on the CI512-ETH and offers an cam controller functionality with the same basic functions. The configuration concerning EtherCAT communication and the usage from AC500 PLC is basically the same. The common features are

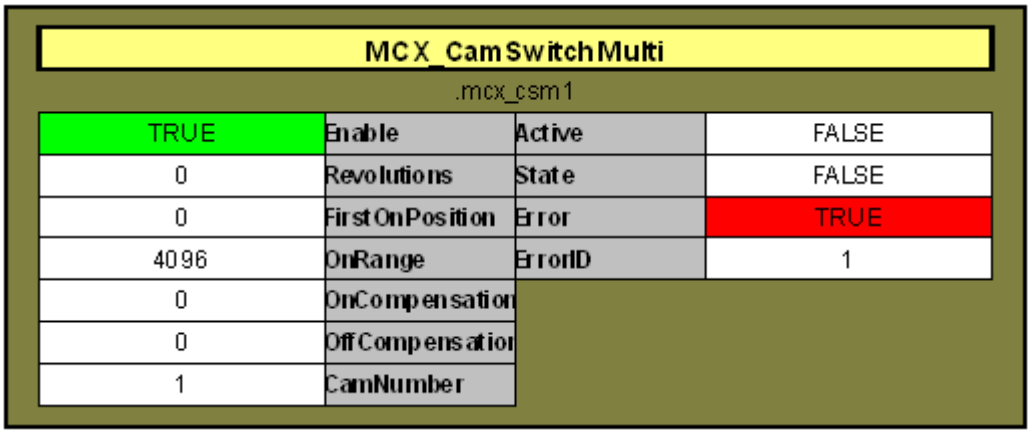
- Up to 32 cam switches with free assignment to the binary outputs
- Dynamic compensation for delay times
- Scan time from 80 µs to 200 µs

A number of additional features is available. The new features are accessible by defining a type for every cam (Parameterization) and by using a dedicated function block which relates to the cam and to the type defined. The respective functionality is explained in detail in the function block's documentation. The features of the different cam-types include:

- Different compensation times for on- and off- switching
- MXC_CamSwitchComfort_dc, MCX_CamSwitchMulti_dc
- Cams which work for a configurable numbers of revolutions
- MCX_CamSwitchMulti_dc, MCX_CamSwitchMultiTimed_dc
- Cams which are switched on for a certain time instead of certain position range.
- MCX_CamSwitchTimed_dc, MCX_CamSwitchMultiTimed_dc
- Cams which are triggered by binary inputs, by a combination of input and Cam or by combination of input and time.
- Trigger
- MCX_CamShift_dc, MCX_BinaryShift_dc
- Triggered
- MCX_CamSwitchSimple_dc, MCX_CamSwitchComfort_dc, MCX_CamSwitchTimed_dc, MCX_CamSwitchMulti_dc, MCX_camSwitchTimeTimed_dc, MCX_CamSwitchMulti-Timed_dc
- A cam which produces a certain frequency according to the velocity
- MCX_PulseSwitch_dc
- A cam which delivers information about binary inputs according to the position
- MCX_BinaryReference_dc
- A certain logic function which might be connected to the cam.
- MCX_CamLogic_dc

1.5.4.6.3 Visualization

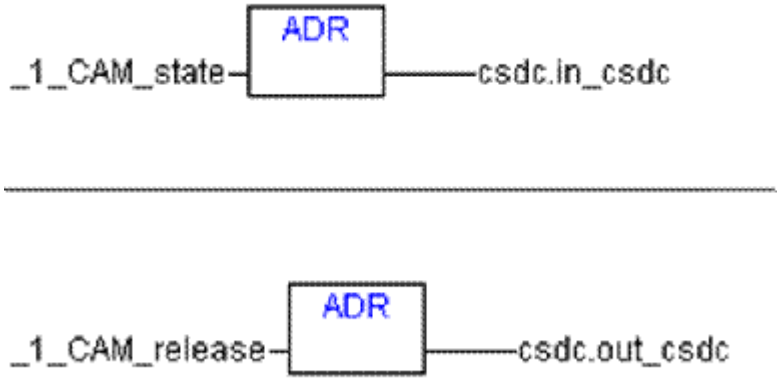
For each of the MCX… function block, the library provides an integrated visualization element. In the configuration of these visualization elements, the placeholder has to be configured with the name of the function block's instance. The visualization looks for example as follows:



The values for all inputs and outputs and as well the name of the instance are shown.

1.5.4.6.4 How to create cscd

1. Create a variable of type CSDCX_REF_TYPE
2. Write the address of the 1. input (process image) of the device to csdc.in_csdc
3. Write the address of the 1. output (process image) of the device to csdc.out_csdc



1.5.4.6.5 How to configure a cam

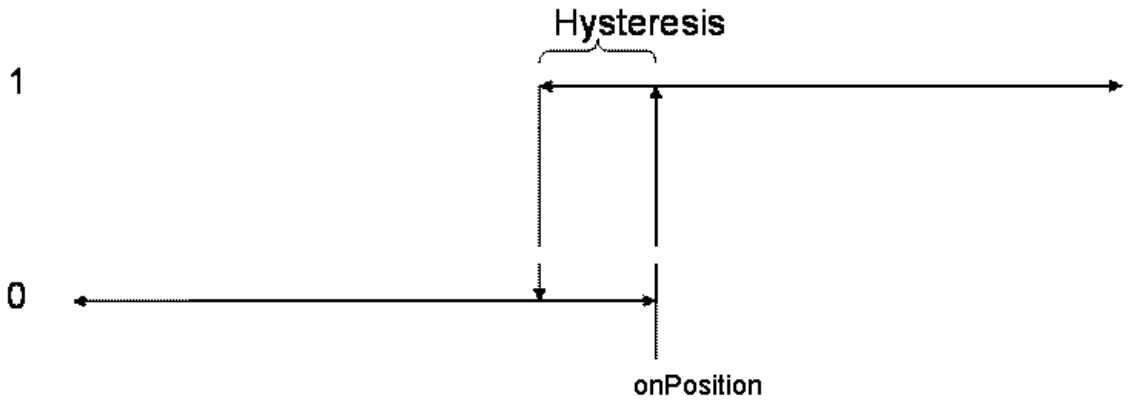
Every cam has 2 main parameters which need to be configured. This is done by the respective bus configuration tool. The parameters will be transferred by SDO during start up. (X runs from 00 to 31)

- camToTrackX 0..15
- connects the cam switch to an input or output, (input or output is determined by the type of cam switch)
- camTypeX 0..9
- selects the type of cam switch

All other data for a single cam switch is transferred by the respective function block

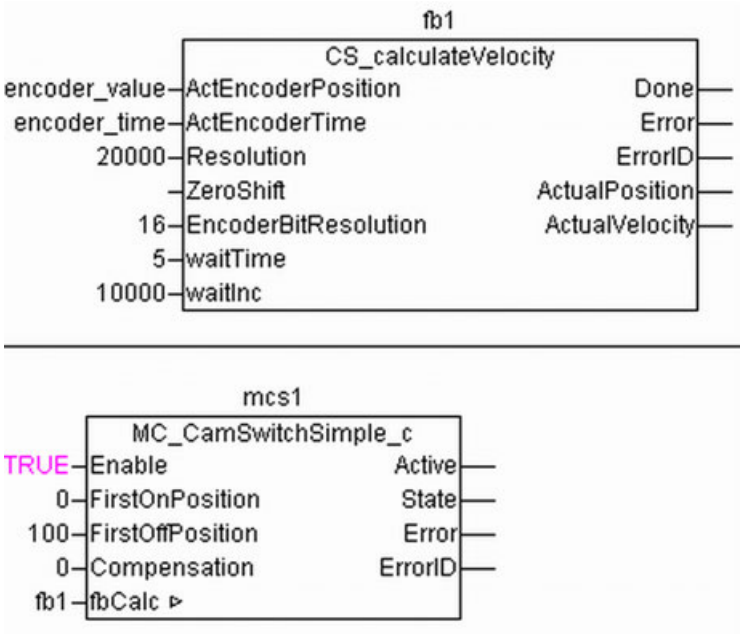
1.5.4.6.6 Hysteresis

- The PulseSwitch has an individually defined hysteresis per cam, applied to the On- and Off-Position.
- All other Cams have a common hysteresis, defined as parameter, which is applied to the „OnPosition”. The hysteresis would be just in use when the encoder would turn backwards or on standstill. Then it will prevent a jitter on the output.



1.5.4.6.7 Function blocks

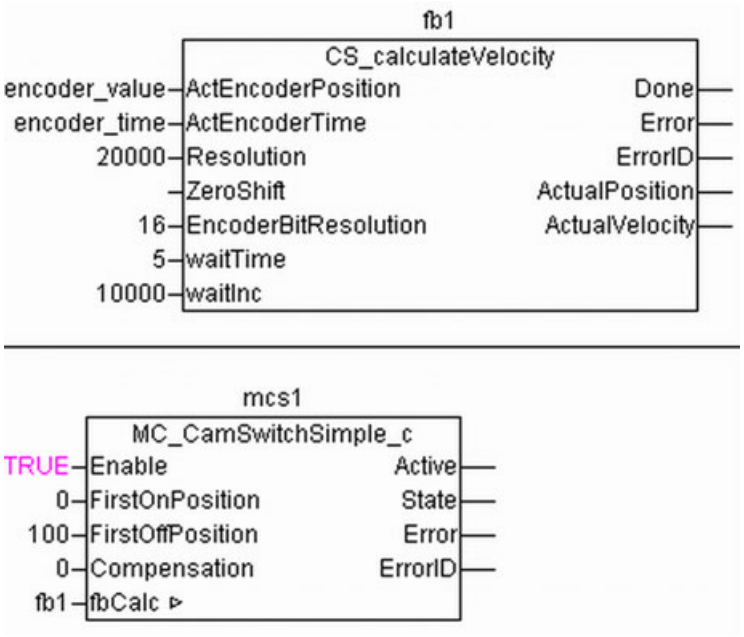
CS_calculateVelocity



Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

CS_CalculateVelocity is necessary to execute some pre-calculations which are needed for the MC_CamSwitchSimple_c. It needs the actual position and the time related to this position to calculate the velocity. In addition, the position and velocity are transferred to the units which are used from MC_CamSwitchSimple_c.



ActEncoderPosition

Data type	Default value	Range	Unit
DWORD	-	-	-

At input ActEncoderPosition, the position value of the encoder has to be given. It mus be a binary value.

ActEncoderTime

Data type	Default value	Range	Unit
DWORD	-	-	-

At input ActEncoderTime, the time value related to the ActEncoderPosition has to be given. The resolution is nanoseconds.

Resolution

Data type	Default value	Range	Unit
WORD	-	-	-

When the position should be used as it is, the value for resolution should be set to 0. Otherwise, resolution gives the value which is used to display 1 revolution.

For example, when to encoder provides 10 bits per revolution, the encoder value (ActEncoder-Position) would run from 0..1023.

- With resolution=360, the value would be converted to a 1° resolution and the positions for MC_CamSwitchSimple_c could run from 0..259.
- With resolution=0, the positions for MC_CamSwitchSimple_c could run from 0..1023.

ZeroShift

Data type	Default value	Range	Unit
DWORD	-	-	-

At input Zero Shift, an offset to adjust the value of ActEncoderPosition, can be specified.

EncoderBitResolution

Data type	Default value	Range	Unit
WORD	-	-	-

Number of bits which hold the relevant position for 1 revolution (8..32)

This parameter allows to adjust the calculation to different encoder resolutions. The encoder might deliver a single turn or multi turn value. The only requirement is to have a binary value which means 1 revolution should match 2^n increments. n is then the value for EncoderBitResolution.

waitTime

Data type	Default value	Range	Unit
WORD	-	-	-

At input waitTime the minimum time to wait between 2 calculations of velocity [µs] has to be specified.

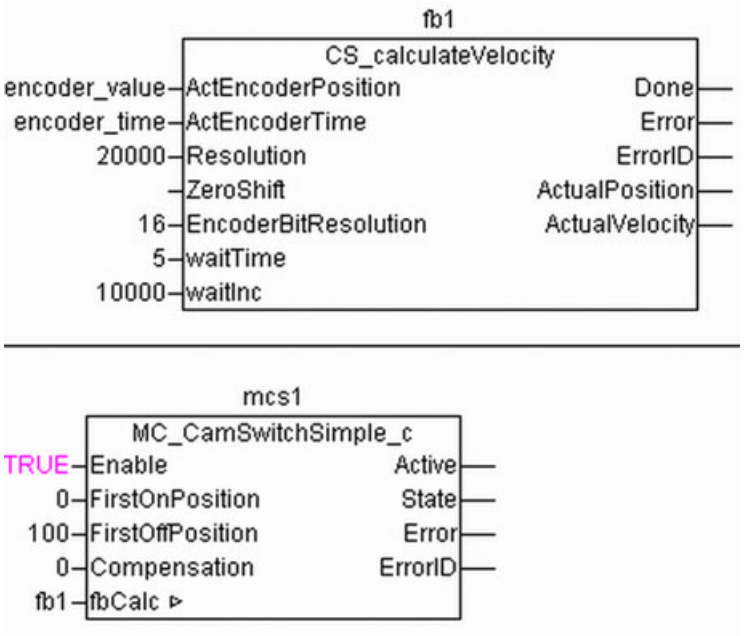
waitInc

Data type	Default value	Range	Unit
WORD	-	-	-

At input waitInc the minimum distance to wait between 2 calculations of velocity has to be specified.

When the velocity is low or the time between 2 measurements is very short, the velocity calculation with velocity = position/time will be imprecise. Therefore it is possible to give a minimum position distance (waitInc) and a minimum time distance (waitTime) which should be exceeded to perform a new calculation. The calculation will be executed when at least 1 of the 2 limits was exceeded. In addition, the velocity will be a average of 4 calculations.

Output description



Done

Data type	Default value	Range	Unit
BOOL	-	-	-

Function block has been executed at least once.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

ActualPosition

Data type	Default value	Range	Unit
DWORD	-	-	-

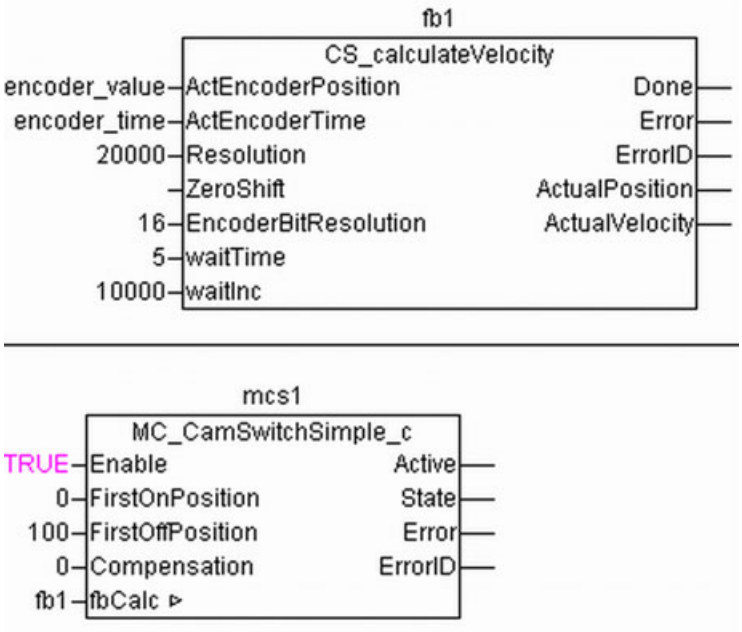
Actual position based on resolution.

ActualVelocity

Data type	Default value	Range	Unit
DINT	-	-	-

Actual velocity based on resolution and normalised with 0x10000.

MCX_CamSwitchSimple_c

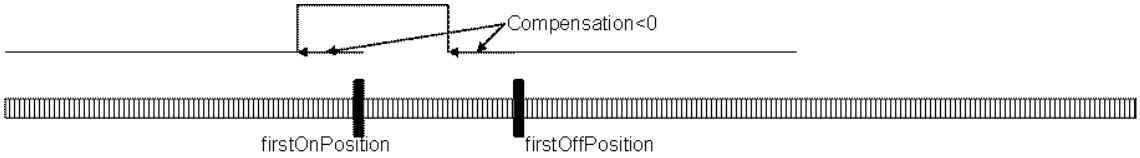


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MC_CamSwitchSimple_c provides the possibility to switch a cam on/off at projected positions. The position is given by a function block of type CS_calculateVelocity. This function block has to be executed every cycle and by the same task as the MC_CamSwitchSimple_c. MC_CamSwitchSimple_c could be used several times, there are no special hardware requirements.

The switch-on-position and the switch-off-position are individually changeable online. Further more a compensation time in both directions can be added to the switch position. Compensation is applied to both switch-on-position and switch-off-position.

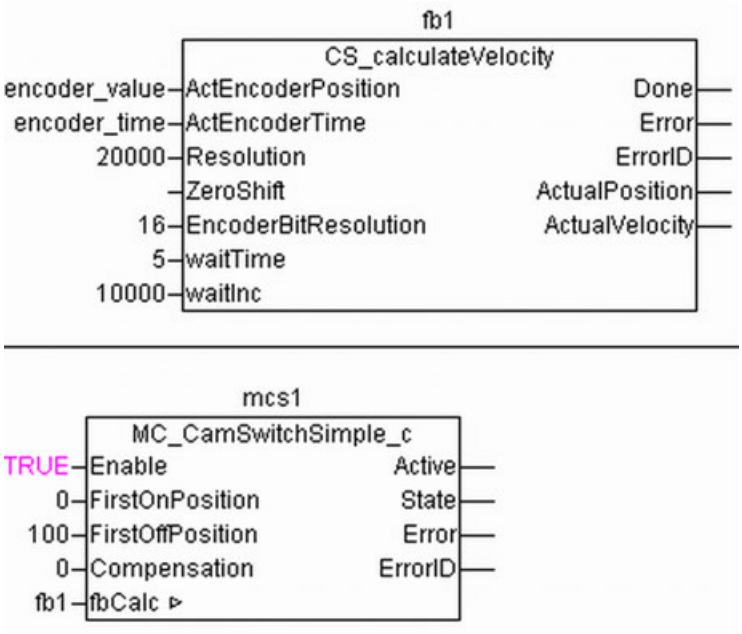


The compensation time makes the switch a “dynamic” switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position has been reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

```
Distance = velocity * compensation;
```

The switch will be switched on at “FirstOnPosition + distance” and switched off at “FirstOffPosition + distance”. While switched on, no new calculation of distance will be done.

The actual state of the cam can also be monitored with the output “state”.



Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

FirstOffPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOffPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOffPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOffPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

- *If FirstOnPosition is smaller than FirstOffPosition, the cam is switched on between the values of FirstOnPosition and FirstOffPosition of the same turn.*
- *If FirstOnPosition is greater than FirstOffPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and FirstOffPosition of the next turn.*
- *If FirstOnPosition is equal to FirstOffPosition, the cam is switched on for one device cycle.*

Compensation

Data type	Default value	Range	Unit
DINT	-	-	µs

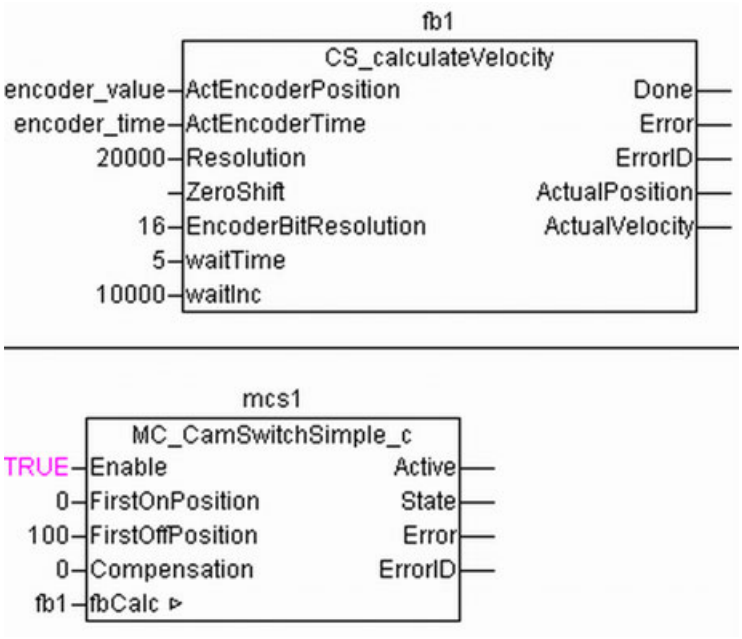
The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

fbCalc CS_CalculateVelocity

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

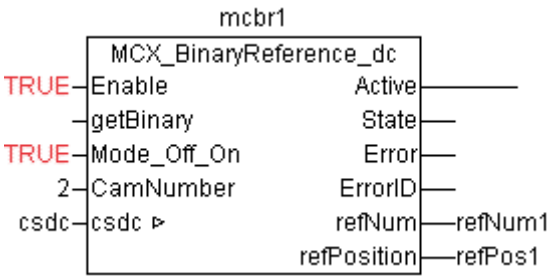
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

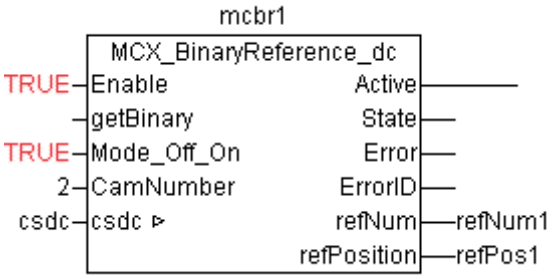
MCX_BinaryReference_DC



Read a Captured Position

Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	Camswitch Extended
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description



MCX_BinaryReference_dc provides the possibility to read the encoder position related to an edge on a binary input. The master position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The values for getBinary and Mode_On_Off are individually changeable online.

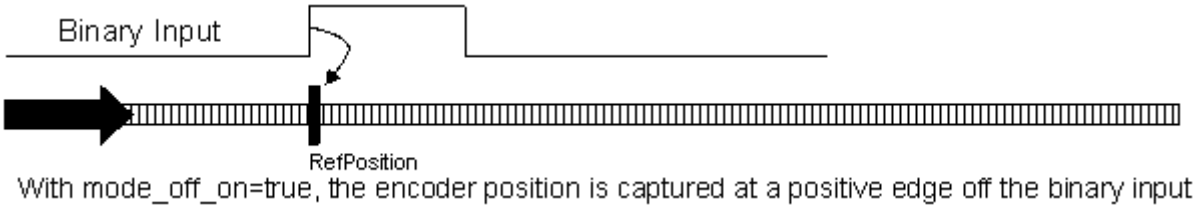
The function block is related to a binary input. It saves the actual encoder position as refPosition when an edge of the binary input has been received. Accuracy would be best when the cam is configured at a low number. The output refNum is counted up for each trigger, so this would also provide an information when more edges are triggered then could be transferred to the PLC.

- Mode_Off_On=FALSE
- a negative edge is used as trigger
- Mode_Off_On=TRUE
- a positive edge is used as trigger

The refPosition can be

- getBinary=TRUE
- the binary value as directly received from the encoder
- getBinary=FALSE
- the angle-value (position is converted by the angle resolution)

The values are transferred multiplexed, so it may take some time to receive them.



The actual state of the binary input can also be monitored with the output "state".

Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

getBinary

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

getBinary selects if a binary value or angle value should be triggered.

Mode_Off_On

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Mode_off_on selects if a positive or negative edge will be used as trigger.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

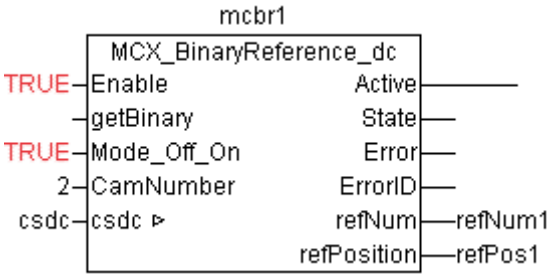
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

refNum

Data type	Default value	Range	Unit
WORD	-	0...255	-

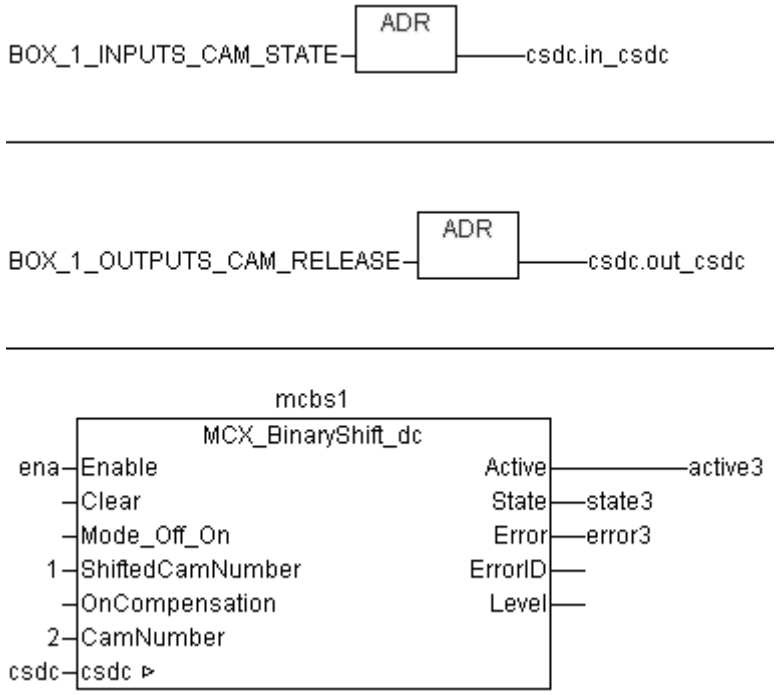
refNum returns the number of different positions which already have been captured. It counts from 0..255 and starts at 0 again. A change of refNum indicates that a new position has been captured. When the change is >1, it indicates how many positions had been captured since the last call of the function block.

refPosition

Data type	Default value	Range	Unit
DWORD	-	-	-

Output refPosition returns the captured position.

MCX_BinaryShift_DC



Create a Trigger to a Cam by a Binary Input

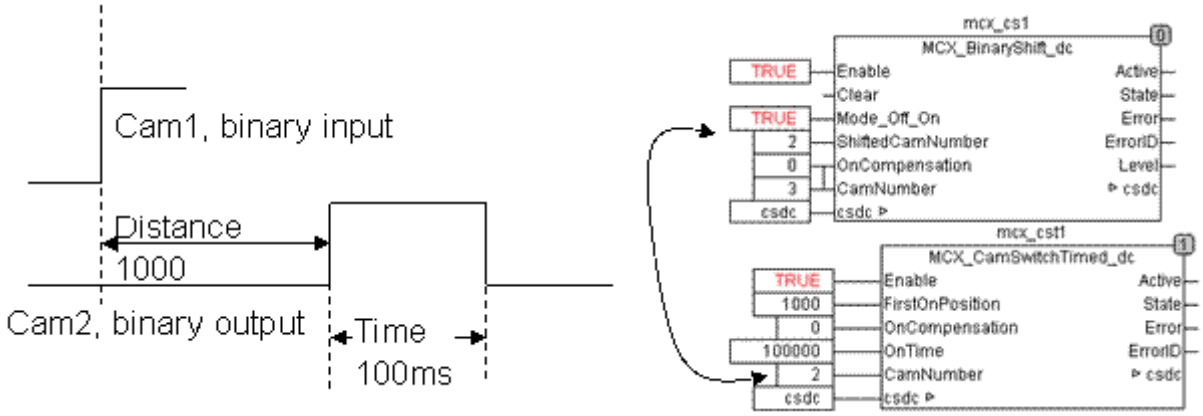
Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	Camswitch Extended
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_BinaryShift_dc provides the possibility to start a cam switch related to a binary input. The BinaryShift observes a binary input. When a positive (Mode_Off_On=TRUE) or negative (Mode_Off_On=FALSE) edge is received, the associated cam is started. It is then shifted by a certain distance or time. The following characteristics apply to the BinaryShift:

- Used to create a trigger to some kind of "žShiftedCam"
- Is edge triggered
- "ShiftedCam" might be
- Common, comfort, timed, multi, multi-timed
- Time-timed (just valid with shift)

The values for Mode_Off_On and OnCompensation are individually changeable online.



The example shows how a BinaryShift and a CamSwitchTimed would work together:

- Mode_Off_On=TRUE, therefore a positive edge is evaluated. The CamSwitchTimed is triggered in relation to the received edge.
- After the positive edge on a binary input was received, the connected CamSwitchTimed would be triggered. The output "level" will be increased by 1
- When the encoder moved a distance of 1000 (FirstOnPosition of CamSwitchTimed), the output of the cam will be switched on. It will stay on for 100 ms. The output "level" of the BinaryShift will be decreased by 1.
- When additional positive edges are received outside before the distance of 1000 was moved, the respective actions are queued. This is possible for up to 15 actions.

The actual state of the cam can also be monitored with the output "state". It will show the cam, not the binary input.

Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

Clear

Data type	Default value	Range	Unit
BOOL	-	-	-

Input Clear deletes the already triggered, but not yet executed cam switches.

Mode_Off_On

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Mode_off_on selects if a positive or negative edge will be used as trigger.

ShiftedCam-Number

Data type	Default value	Range	Unit
WORD	-	-	-

Input ShiftedCamNumber selects the number of the cam which has to be triggered, runs from 1..32.

OnCompensa-tion

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

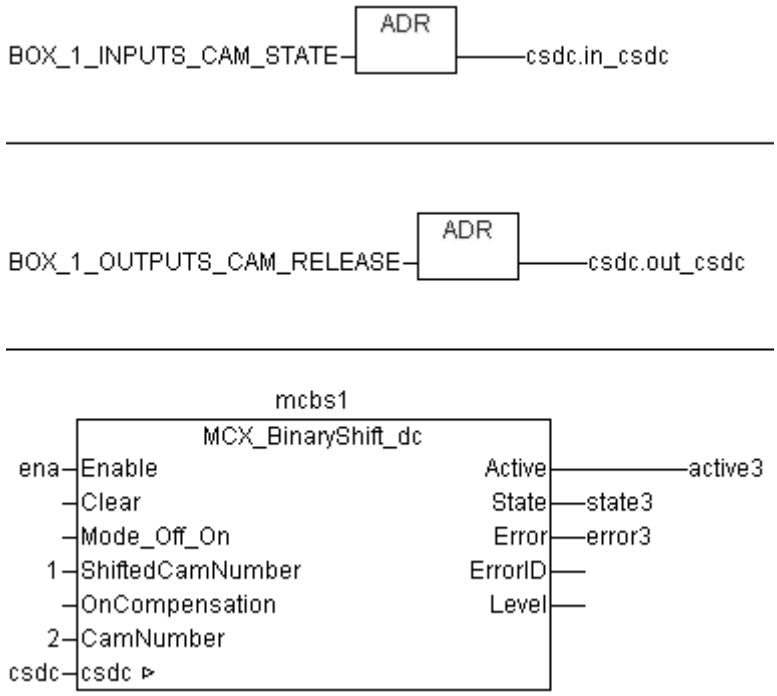
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

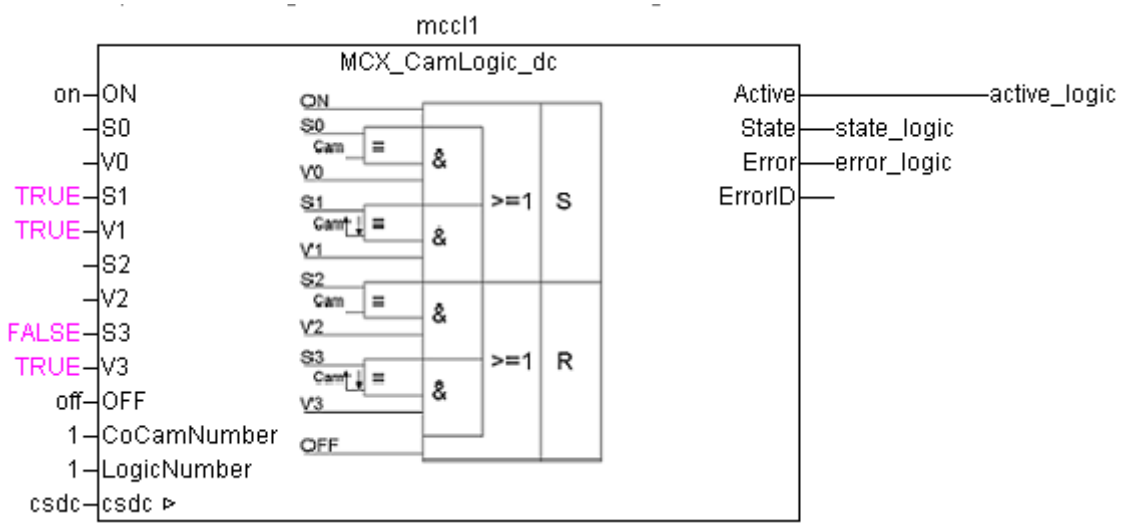
ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

Level

Data type	Default value	Range	Unit
BYTE	-	-	-

Output Level return the number of trigger actions which are queued.

MCX_CamLogic_DC



Interconnection to a Cam.

Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	Camswitch Extended
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

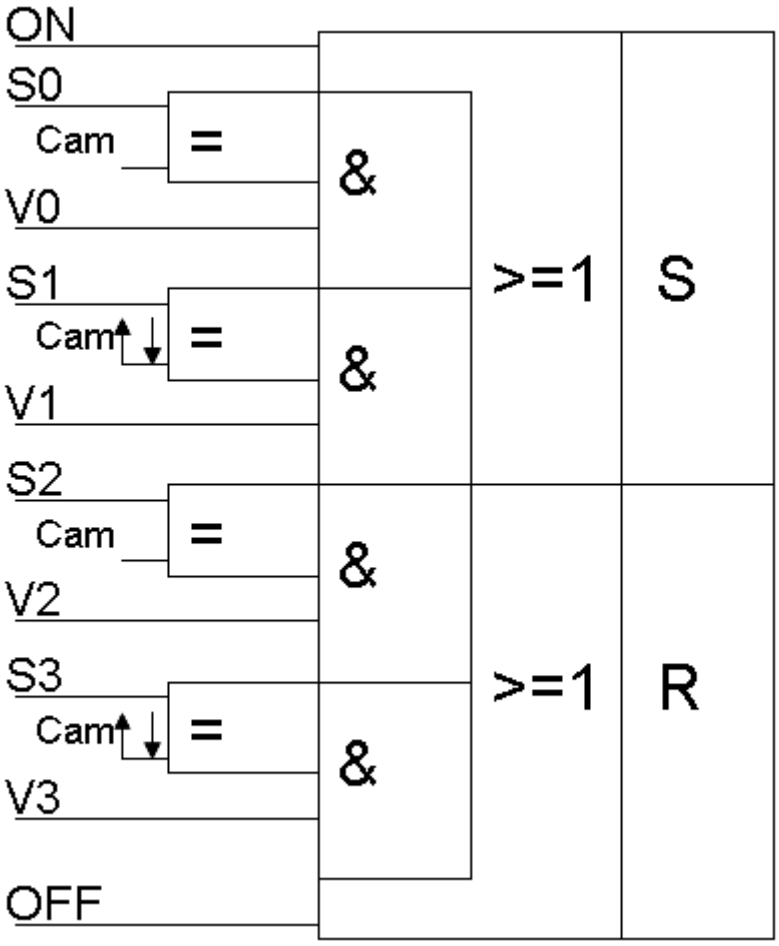
Input description

MCX_CamLogic_dc provides the possibility to add some logical (boolean) interconnection to a cam result. Usually, the binary output is switched on when the cam is switched on. Several cams could be configured to use the same binary output. This will result in an OR-connection of the cam-states to switch on the binary output. The cam state will correspond to the binary output. When other logic operations are necessary, the MCX_CamLogic_dc could be used. It will allow to:

- Switch on/off the output corresponding to the cam state
- Switch on/off the output corresponding to the inverted cam state
- Switch on/off the output corresponding to a positve/negative edge of the cam state
- Directly set or reset the logic. This will subsequently set or reset of the binary output.

Up to 16 MCX_CamLogic_dc are available. It is not necessary to have a configuration to use them, but just to use the number 1..16 to select the respective logic. The logic is connected to the cam by the input CoCamNumber.

The boolean operation which is executed matches the following diagram:



- ON=TRUE will directly set the output
- OFF=TRUE will directly reset the output
- With V0=TRUE, the cam state is enabled to set the output
- With S0=TRUE, the positive state will set the output
- With S0=FALSE, the negative state will set the output
- With V1=TRUE, the cam edge is enabled to set the output
- With S1=TRUE, the positive edge will set the output
- With S0=FALSE, the negative edge will set the output
- The reset part works accordingly
- When set and reset are active, the reset dominates

When several cams are configured to use the same output, they either all of them should have a logic block or none of them should. In case they all have a logic block, all SET-results will be connected by boolean OR and all RESET results will be connected by boolean OR. Afterwards, the operation will be executed with the OR results.

The inputs V0, V1, V2, V3, ON and OFF are transferred immediately and will be effective immediately, while the S0, S1, S2 and S3 are transferred multiplexed with other parameters (from other function block).

The actual state of the logic can also be monitored with the output "state". It will show the logics result, not the binary output.

ON

Data type	Default value	Range	Unit
BOOL	-	-	-

Input ON activates the set path of MCX_CamLogic_dc and set the binary output subsequently.

S0, S1, S2, S3 (select)

Data type	Default value	Range	Unit
BOOL	-	-	-

Inputs V0, V1, V2, V3 select the condition to enable the respective path.

V0, V1, V2, V3 (select)

Data type	Default value	Range	Unit
BOOL	-	-	-

Inputs S0, S1, S2, S3 select positive (TRUE) or negative (FALSE) signal for the respective path.

OFF

Data type	Default value	Range	Unit
BOOL	-	-	-

Input OFF activates the reset path of MCX_CamLogic_dc and reset the binary output subsequently.

CoCamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

LogicNumber

Data type	Default value	Range	Unit
WORD	-	1...16	-

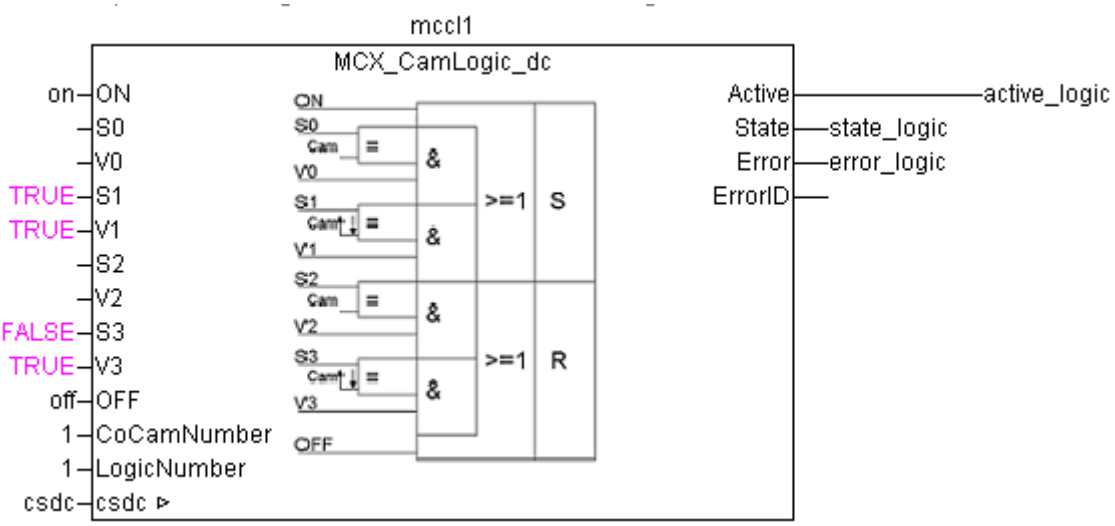
Number of the logic block: Each number should just be used once. An additional configuration is not required.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

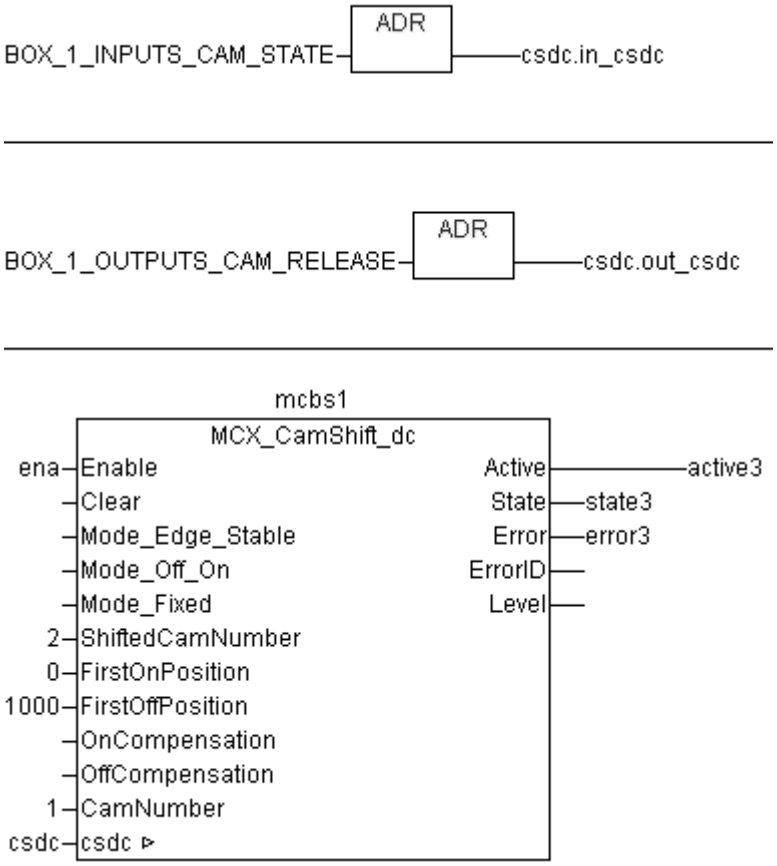
ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

MCX_CamShift_DC

Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0

Parameter	Value
Type	Function block with historical values
Group	Group/Subgroup
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

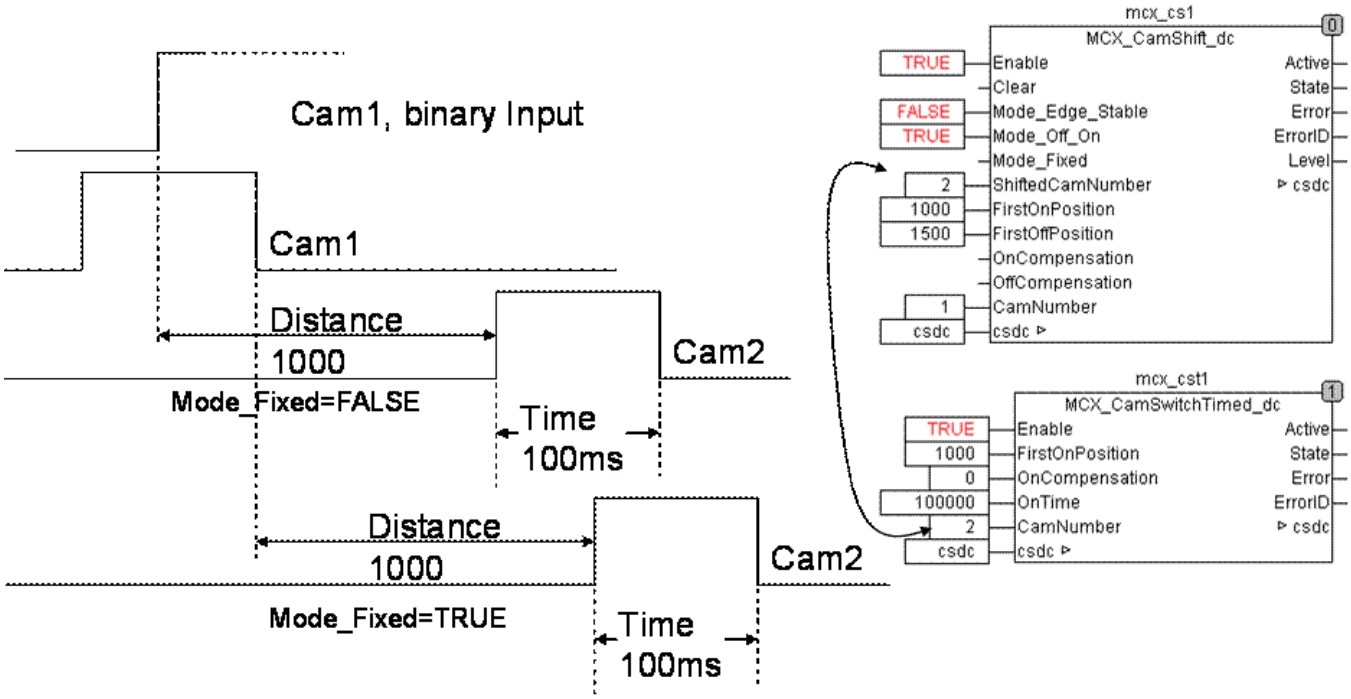


Input description

MCX_CamShift_dc provides the possibility to start a cam switch related to a "supervisor" cam in relation with a binary input. The CamShift observes a binary input during the length of the cam. Depending on the result, the associated cam is started. It is then "shifted" by a certain distance or time. The following characteristics apply to the CamShift:

- Used to create a trigger to some kind of "ShiftedCam"
- Is edge or level triggered
- "ShiftedCam" might be
- Common, comfort, timed, multi, multi-timed
- Time-timed (just valid with shift)

The values for Mode_Edge_Stable, Mode_Off_On, Mode_Fixed, FirstOnPosition, FirstOffPosition, OnCompensation and OffCompensation are individually changeable online.

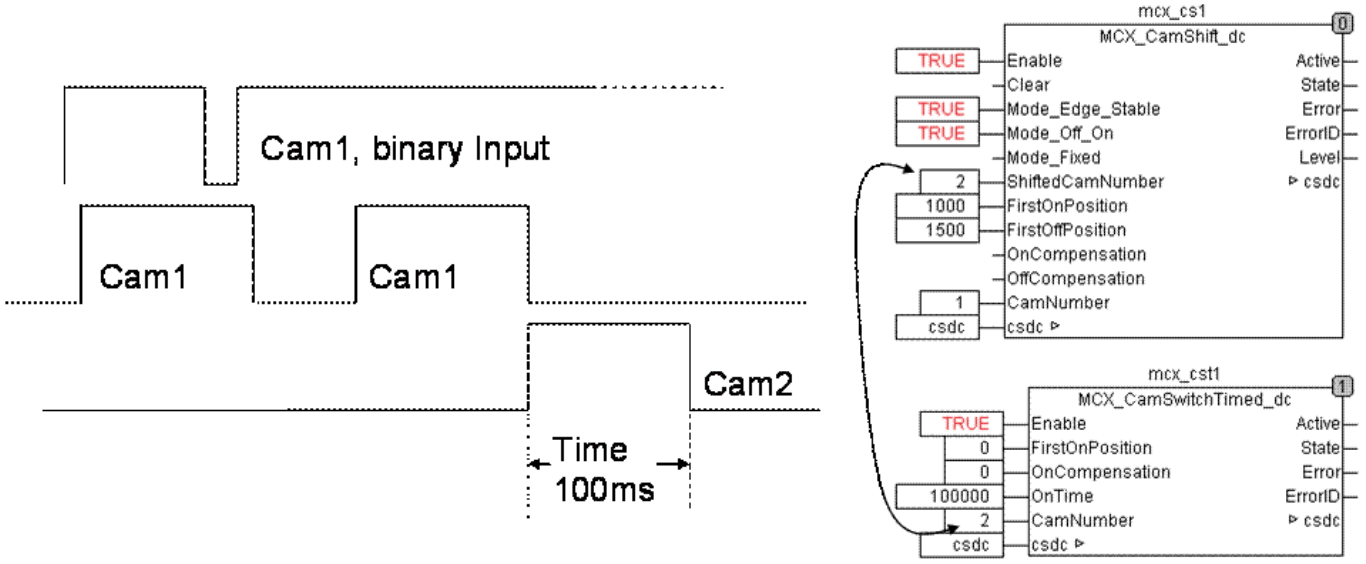


The example shows how a Cam and a CamSwitchTimed would work together:

- Mode_Edge_stable=FALSE, therefore a edge is evaluated
- Mode_Off_On=TRUE, therefore a positive signal is evaluated
- Two examples are shown, with Mode_Fixed=FALSE first: The CamswitchTimed is triggered in relation to the received edge.

With Mode_Fixed=TRUE, it is related to the end of Cam1.

- After the positive edge on a binary input was received, the connected CamSwitchTimed would be triggered. The output "level" will be increased by 1. It is not necessary to receive an edge during the length of the cam, a high signal right from the beginning would do the same.
- When the encoder moved a distance of 1000 (FirstOnPosition), the output of the cam will be switched on. It will stay on for 100 ms. The output "level" of the CamShift will be decreased by 1.
- When additional positive edges are received outside the length of Cam1, these are ignored. When Cam1 will be switched on before the distance of 1000 was moved, the respective actions are queued. This is possible for up to 15 actions.



The 2nd example shows the result when Mode_Edge_Stable=TRUE is used (Mode_Fixed in this case is "don't care"). The binary input has to be high during the length of Cam1, otherwise the ShiftedCam will not be started.

The actual state of the cam can also be monitored with the output "state". It will show the Cam, not the binary input.

Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

Clear

Data type	Default value	Range	Unit
BOOL	-	-	-

Input Clear deletes the already triggered, but not yet executed cam switches.

Mode_Edge_Stable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Mode_Edge_Stable selects whether an edge signal or a stable signal is to be used as trigger (FALSE= edge).

If mode_edge_stable=TRUE, the binary signal must be stable during the range of the trigger-cam, otherwise the shifted cam will not be released.

Mode_Off_On

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Mode_off_on selects if a positive or negative edge will be used as trigger.

Mode_Fixed

Data type	Default value	Range	Unit
BOOL	-	-	-

Mode_Fixed selects if an action is triggered in relation to the binary signal or to the end of the cam.

When Mode_Fixed=TRUE, the end of the cam is the reference point. When Mode_Fixed=FALSE, the binary signal is the reference point The shifted cam is started with its own FirstOnPosition as a distance from the reference point. Mode_Fixed=FALSE is only valid with Mode_Edge_Stable=FALSE.

Mode_Edge_Stable	Mode_Fixed	Reference Point
FALSE	FALSE	Binary signal
FALSE	TRUE	Cam
TRUE	Don't care	Cam

ShiftedCam-Number

Data type	Default value	Range	Unit
WORD	-	-	-

Input ShiftedCamNumber selects the number of the cam which has to be triggered, runs from 1..32.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

FirstOffPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOffPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOffPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOffPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

- *If FirstOnPosition is smaller than FirstOffPosition, the cam is switched on between the values of FirstOnPosition and FirstOffPosition of the same turn.*
- *If FirstOnPosition is greater than FirstOffPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and FirstOffPosition of the next turn.*
- *If FirstOnPosition is equal to FirstOffPosition, the cam is switched on for one device cycle.*

OnCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

OffCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with OffCompensation:

- If OffCompensation is positive, the switch action will be delayed by the value of OffCompensation.
- If OffCompensation is negative, the switch action will be brought forward by the value of OffCompensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

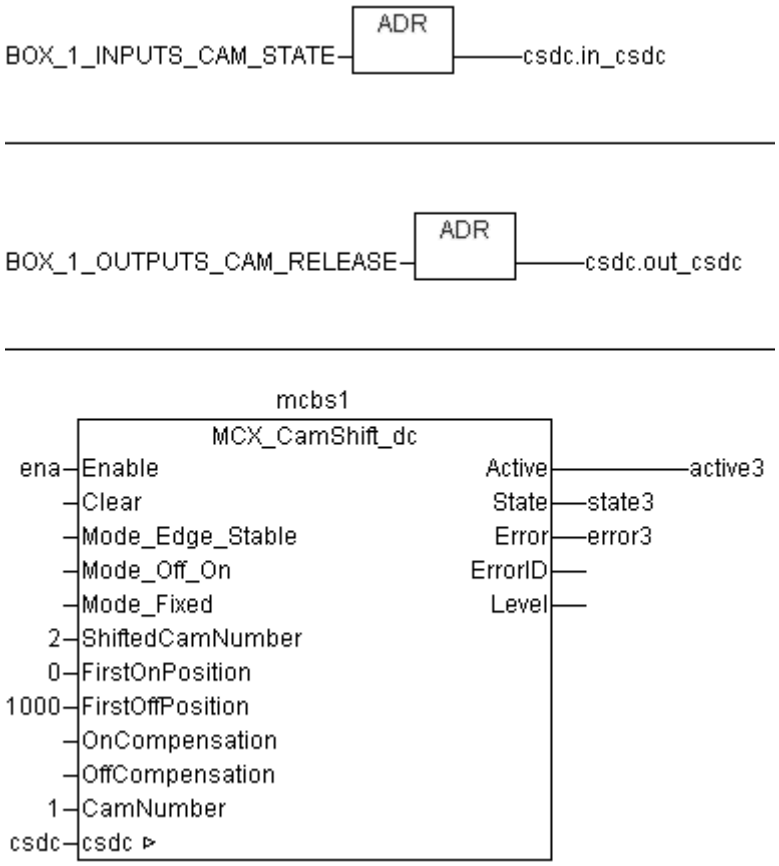
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

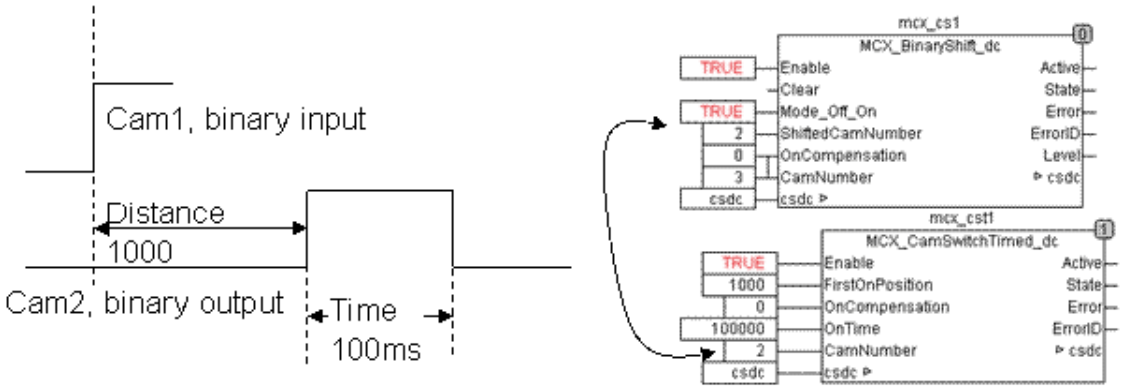
ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

Level

Data type	Default value	Range	Unit
BYTE	-	-	-

Output Level return the number of trigger actions which are queued.

MCX_CamSwitchComfort_DC

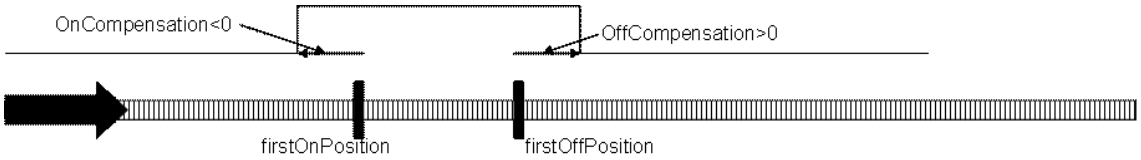


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	Group/Subgroup
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_CamSwitchComfort_dc provides the possibility to switch a cam on/off at projected positions. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the switch-off-position are individually changeable online. Further more a compensation time in both directions can be added to the switch position.



The compensation time makes the switch a "dynamic" switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position has been reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

```
OnDistance = velocity * OnCompensation;
OffDistance = velocity * OffCompensation;
```

The switch will be switched on at "FirstOnPosition + distance" and switched off at "FirstOffPosition + distance". While switched on, no new calculation of distance will be done.

On different times for OnCompensation and OffCompensation, the time to be switched on will be increased or decreased. For example, with a positive OnCompensation and a negative OffCompensation, the time to be switched on will be smaller. The minimum time will be a single device cycle (200 µs), the maximum will be a full revolution with a single cycle off.

The actual state of the cam can also be monitored with the output "state".

Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

FirstOffPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOffPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOffPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOffPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

- *If FirstOnPosition is smaller than FirstOffPosition, the cam is switched on between the values of FirstOnPosition and FirstOffPosition of the same turn.*
- *If FirstOnPosition is greater than FirstOffPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and FirstOffPosition of the next turn.*
- *If FirstOnPosition is equal to FirstOffPosition, the cam is switched on for one device cycle.*

OnCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

OffCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with OffCompensation:

- If OffCompensation is positive, the switch action will be delayed by the value of OffCompensation.
- If OffCompensation is negative, the switch action will be brought forward by the value of OffCompensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

Output description

active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

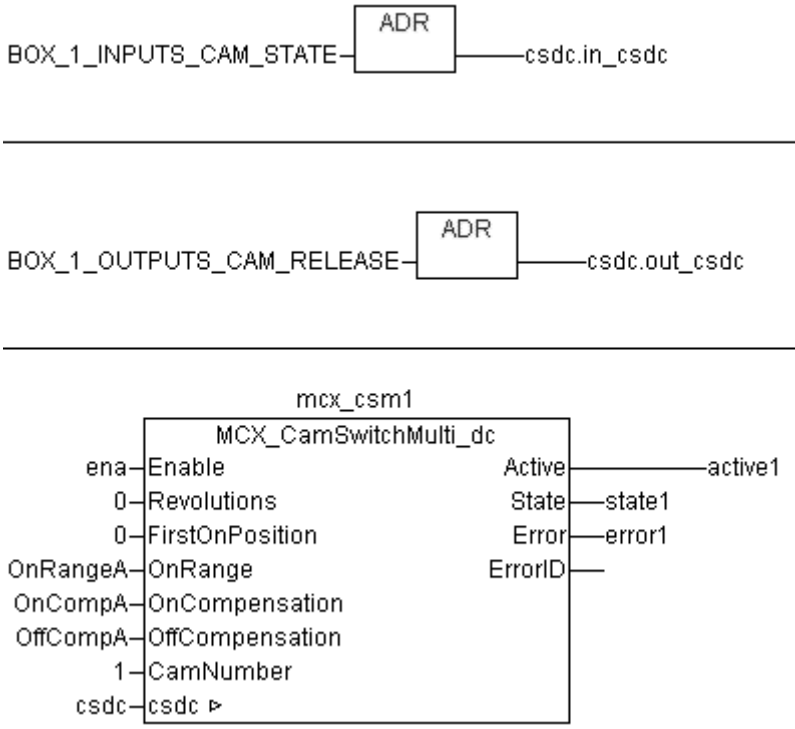
ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

MCX_CamSwitchMulti_DC



Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_CamSwitchMulti_dc provides the possibility to switch a cam on/off at projected positions. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the on-range are individually changeable online. Further more a different compensation time in both directions can be added to the switch position.

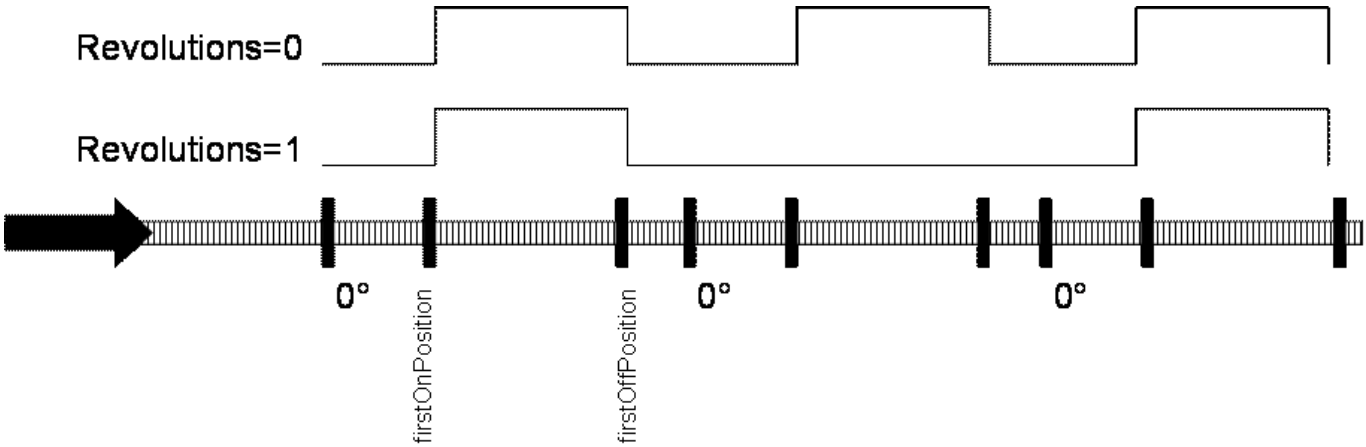
The compensation time makes the switch a "dynamic" switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position has been reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

OnDistance = velocity * OnCompensation;

OffDistance = velocity * OffCompensation;

The switch will be switched on at "FirstOnPosition + OnDistance" and switched off at "FirstOff-Position + OnRange + OffDistance". While switched on, no new calculation of distance will be done.

CamSwitchMulti matches the CamSwitchComfort with an additional information about revolutions. It might switch a cam a distance of several revolutions ahead. The OnDistance and OffDistance might as well cover a distance of several revolutions. OnRange must not be larger then 1 revolution. With revolutions=0, the behavior will be the same as MCX_CamSwitchComfort_dc.

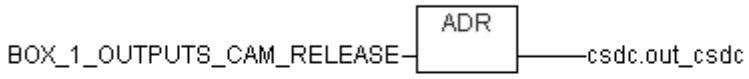


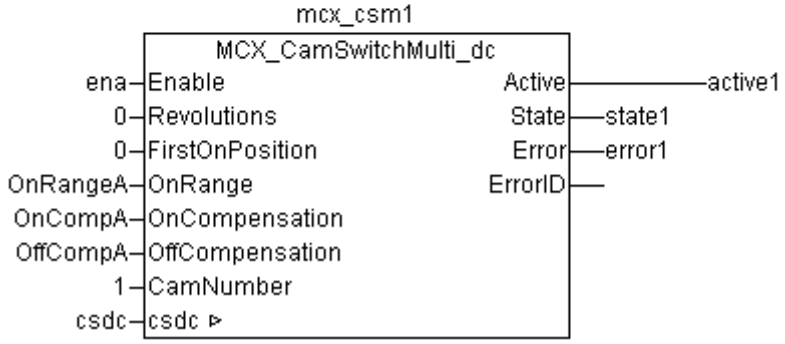
"Revolutions" inside the device start from counting up 0 at power on, therefore, even with a single-turn encoder, the "multi" could be used. The revolutions do not match the revolutions counting of a connected multi-turn encoder, it has just a relative value.

Connected with a "…shift", (BinaryShift or CamShift) it will trigger a cam several revolutions ahead.

On different values for OnCompensation and OffCompensation, the time to be switched on will be increased or decreased. For example, with OnCompensation >0 and OffCompensation <0, the Cam will be switched on later and switched off earlier, so the angle to be switched on is decreased. The minimum time will be a single device cycle (200 µs), the maximum will be a number of revolutions according to the input "revolution+1", with a single cycle off.

The actual state of the cam can also be monitored with the output "state".





Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

OnCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

OnRange

Data type	Default value	Range	Unit
WORD	-	-	-

The value of OnRange is automatically truncated to the resolution of the circle without notice. For example, all OnRange greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

OnRange can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

OffCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with OffCompensation:

- If OffCompensation is positive, the switch action will be delayed by the value of OffCompensation.
- If OffCompensation is negative, the switch action will be brought forward by the value of OffCompensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

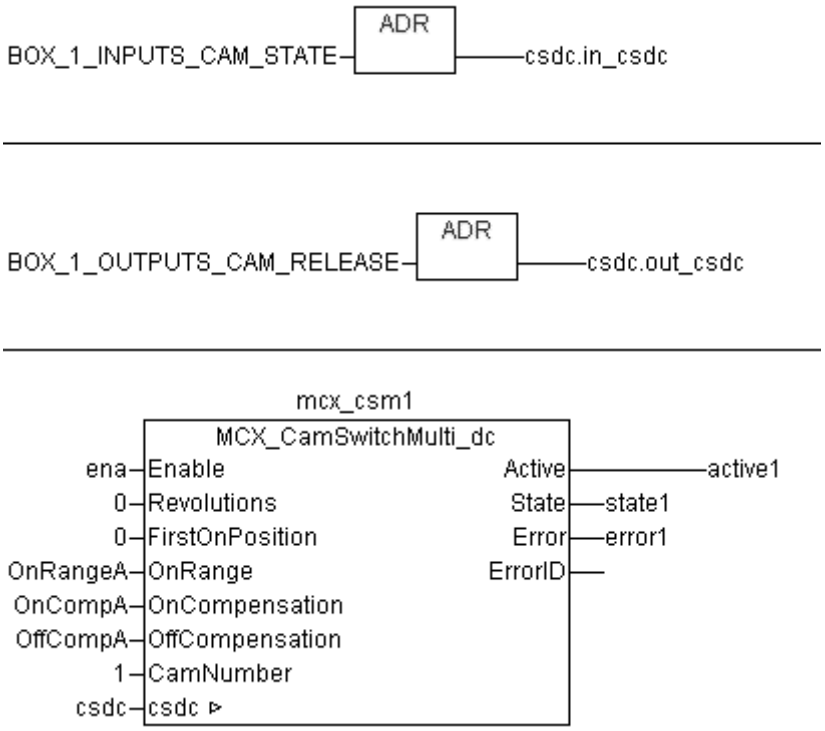
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

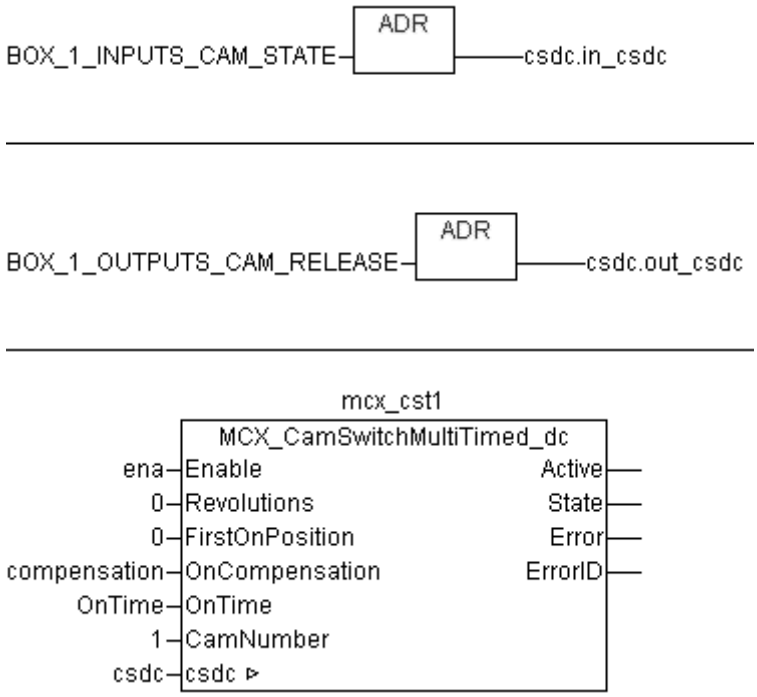
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

MCX_CamSwitchMultiTimed_DC

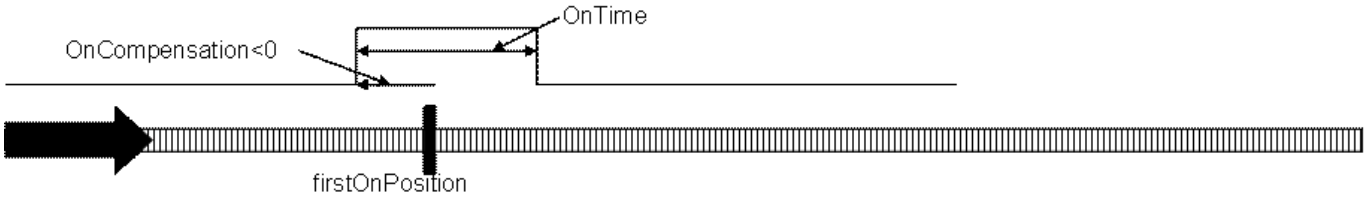


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_CamSwitchMultiTimed_dc provides the possibility to switch a cam on a projected position and off after a projected time. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the time are individually changeable online. Further more a compensation time in both directions can be added to the switch-on-position.



The compensation time makes the switch a "dynamic" switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position was reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

Distance = velocity * compensation;

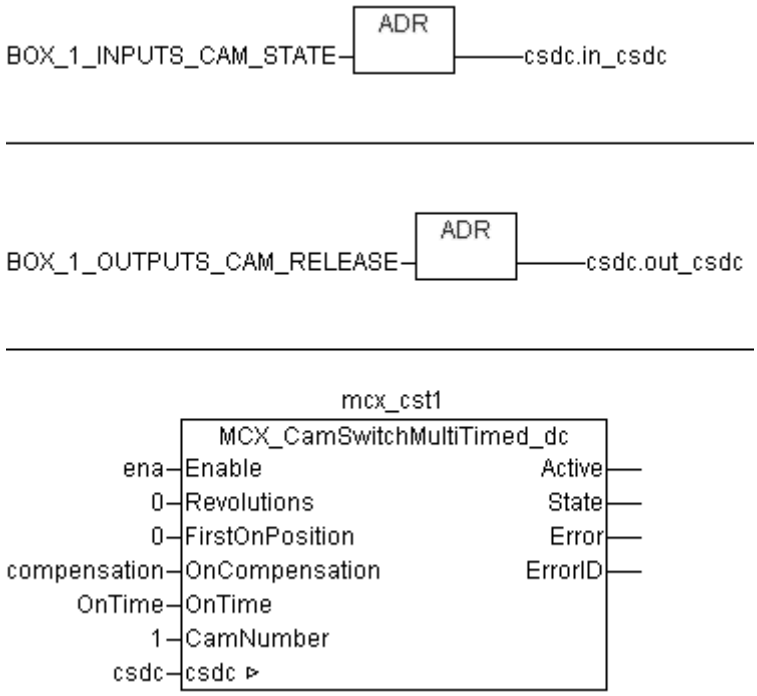
The switch will be switched on at "FirstOnPosition + distance" and switched after the time OnTime elapsed. The time is given in us. The accuracy of time is 200 µs.

CamSwitchMultiTimed matches the CamSwitchTimed with an additional information about revolutions. With revolutions=0, the behavior will be as MCX_CamSwitchTimed_dc. With revolutions > 0, the switching will leave out the respective number of revolutions.

"Revolutions" inside the device start from counting up 0 at power on, therefore, even with a single-turn encoder, the "multi" could be used. The revolutions do not match the revolutions counting of a connected multi-turn encoder, it has just a relative value.

Connected with a "…shift", (BinaryShift or CamShift) it will trigger a cam several revolutions ahead.

The actual state of the cam can also be monitored with the output "state".



Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

OnCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

OnTime

Data type	Default value	Range	Unit
DWORD	-	0...10 s	µs

OnTime gives the duration to switch on the cam. Time will be evaluated with an accuracy of 200 µs (microseconds).

OnTime can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

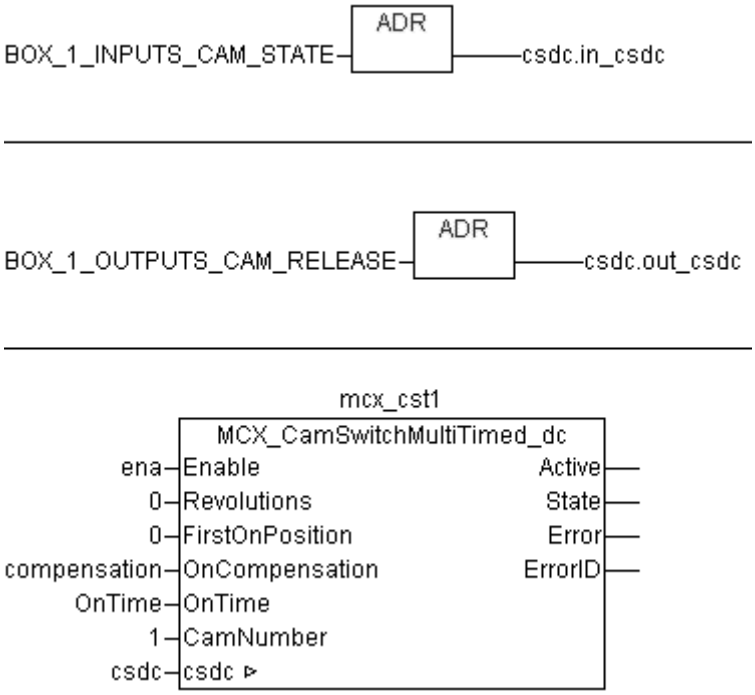
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

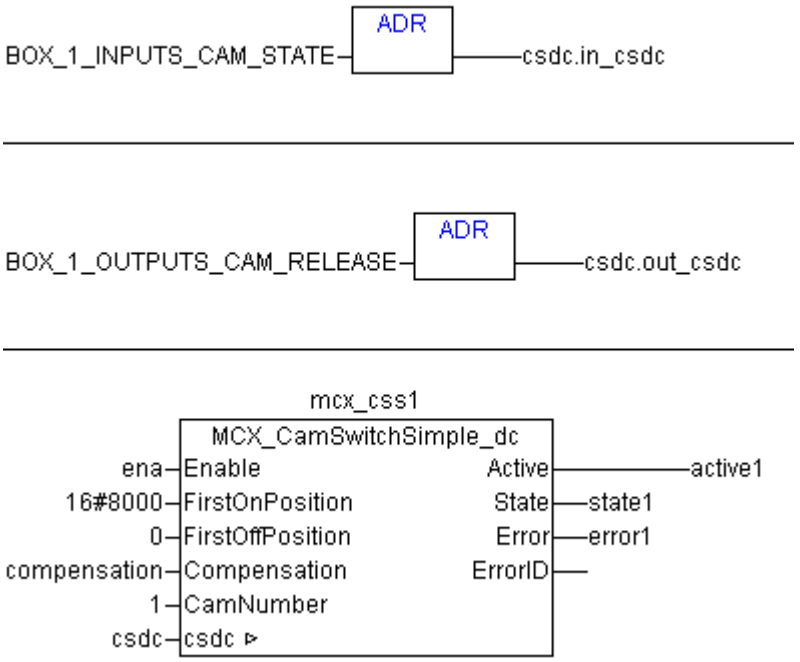
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

MCX_CamSwitchSimple_DC

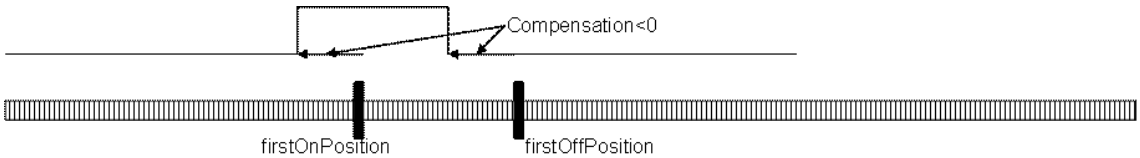


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_CamSwitchSimple_dc provides the possibility to switch a cam on/off at projected positions. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the switch-off-position are individually changeable online. Further more a compensation time in both directions can be added to the switch position. Compensation is applied to both switch-on-position and switch-off-position.

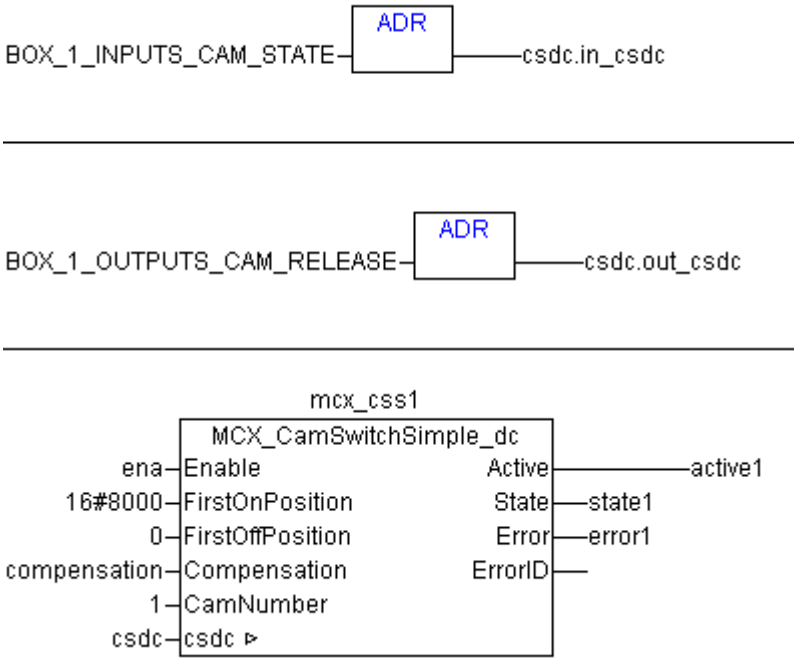


The compensation time makes the switch a "dynamic" switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position has been reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

```
Distance = velocity * compensation;
```

The switch will be switched on at "FirstOnPosition + distance" and switched off at "FirstOffPosition + distance". While switched on, no new calculation of distance will be done.

The actual state of the cam can also be monitored with the output "state".



Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

FirstOffPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOffPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOffPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOffPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

- *If FirstOnPosition is smaller than FirstOffPosition, the cam is switched on between the values of FirstOnPosition and FirstOffPosition of the same turn.*
- *If FirstOnPosition is greater than FirstOffPosition, the cam is switched on between the values of FirstOnPosition of the actual turn and FirstOffPosition of the next turn.*
- *If FirstOnPosition is equal to FirstOffPosition, the cam is switched on for one device cycle.*

Compensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

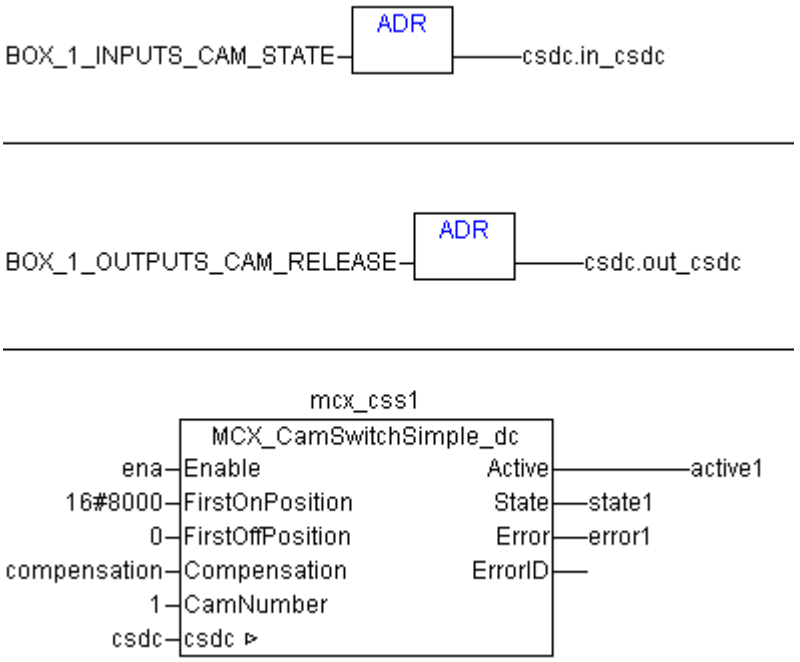
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

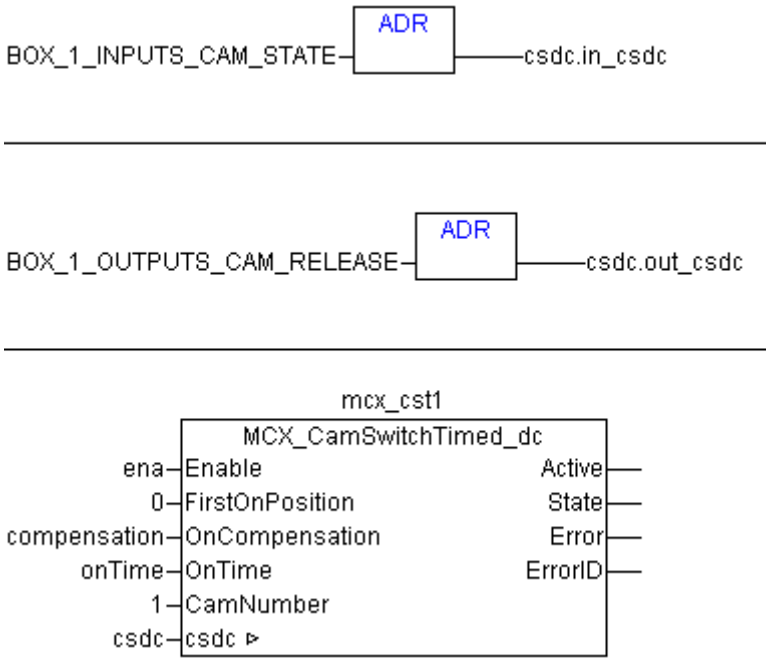
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

MCX_CamSwitchTimed_DC

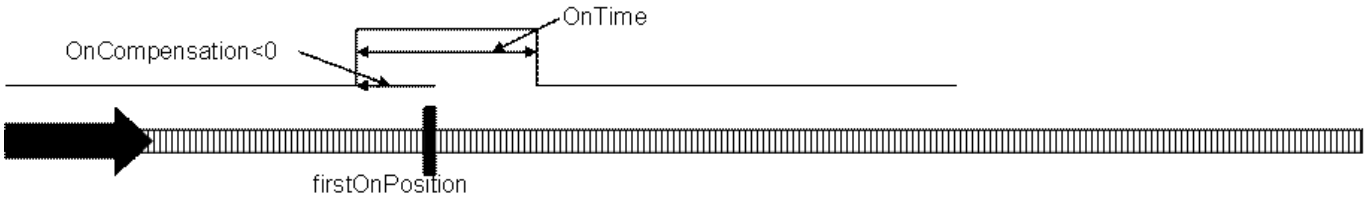


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_CamSwitchTimed_dc provides the possibility to switch a cam on/off at projected positions. The position is given by an encoder, which must be available, if the cam switch functionality is to be used.

The switch-on-position and the switch-off-position are individually changeable online. Further more a compensation time in both directions can be added to the switch position. Compensation is applied to both switch-on-position and switch-off-position.

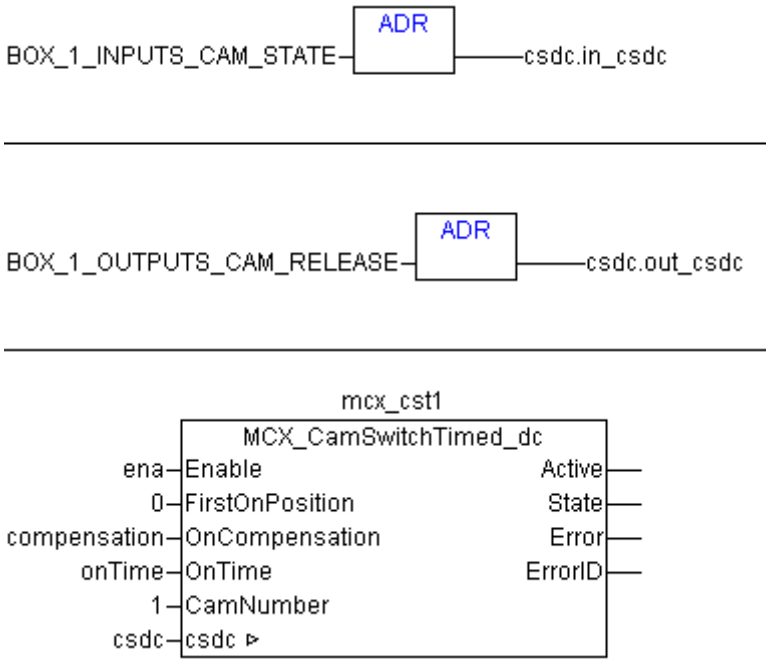


The compensation time makes the switch a "dynamic" switch. The position where to switch on and off is calculated according to the actual velocity, in a way that the output occurs a certain time before (negative compensation) or after (positive compensation) the position has been reached. The time is given at the input compensation, the distance is calculated. The accuracy of time to be reached depends on the stability of velocity, which means just on a constant velocity, the time will be correct.

Distance = velocity * compensation;

The switch will be switched on at "FirstOnPosition + distance" and switched off at "FirstOffPosition + distance". While switched on, no new calculation of distance will be done.

The actual state of the cam can also be monitored with the output "state".



Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

FirstOnPosition

Data type	Default value	Range	Unit
WORD	-	-	-

The value of FirstOnPosition is automatically truncated to the resolution of the circle without notice. For example, all FirstOnPosition greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

FirstOnPosition can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

OnCompensation

Data type	Default value	Range	Unit
DINT	-	-	µs

The switch action is rescheduled with Compensation:

- If Compensation is positive, the switch action will be delayed by the value of Compensation.
- If Compensation is negative, the switch action will be brought forward by the value of Compensation.

OnTime

Data type	Default value	Range	Unit
DWORD	-	0...10 s	µs

OnTime gives the duration to switch on the cam. Time will be evaluated with an accuracy of 200 µs (microseconds).

OnTime can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

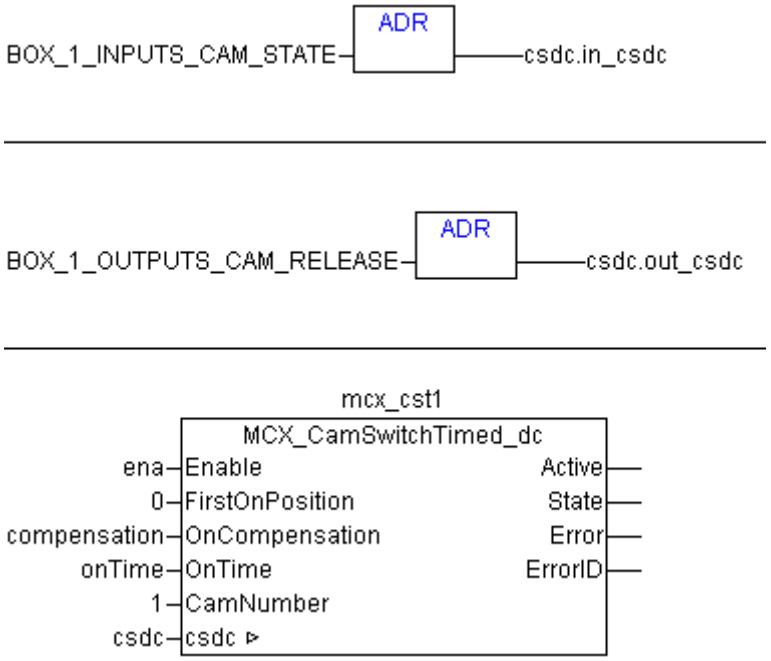
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

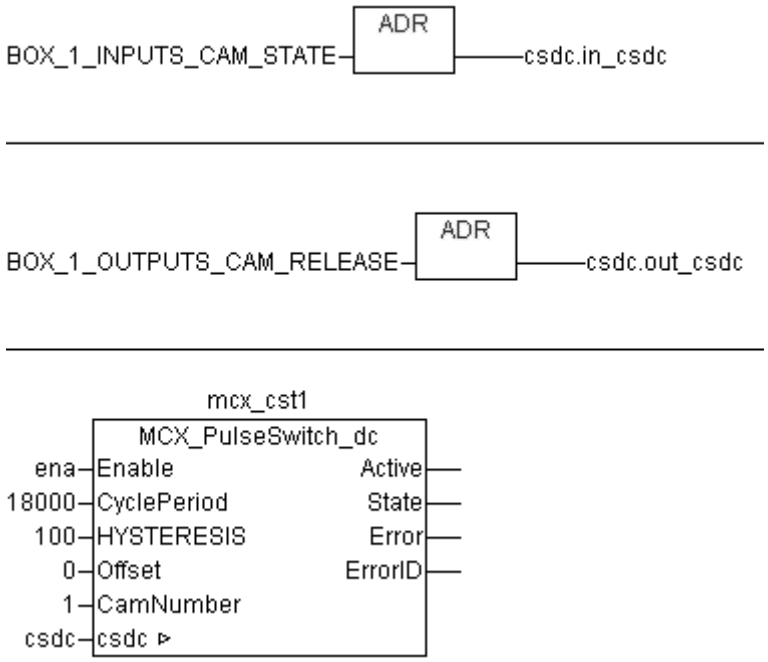
Data type	Default value	Range	Unit
WORD	-	-	-

ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

MCX_PulseSwitch_DC

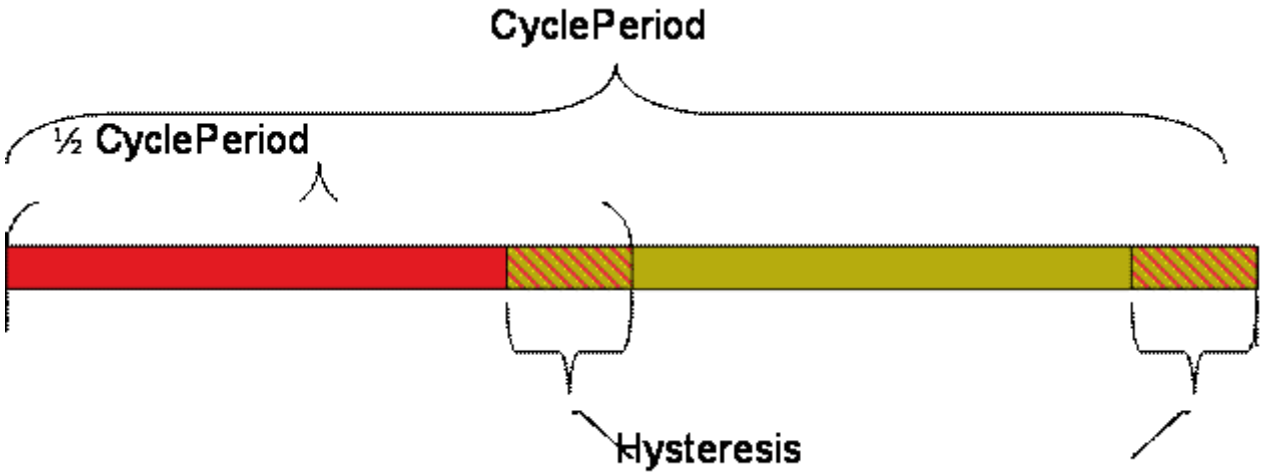


Parameter	Value
Included in library	MCX_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Available for CI511-ETHCAT and CI512-ETHCAT	With device index C0 and above

Input description

MCX_PulseSwitch_dc provides the possibility to generate a pulse train according to the master position; the frequency is proportional to the velocity of the master. The master position is given by an encoder, which must be available, if the cam switch functionality is to be used.

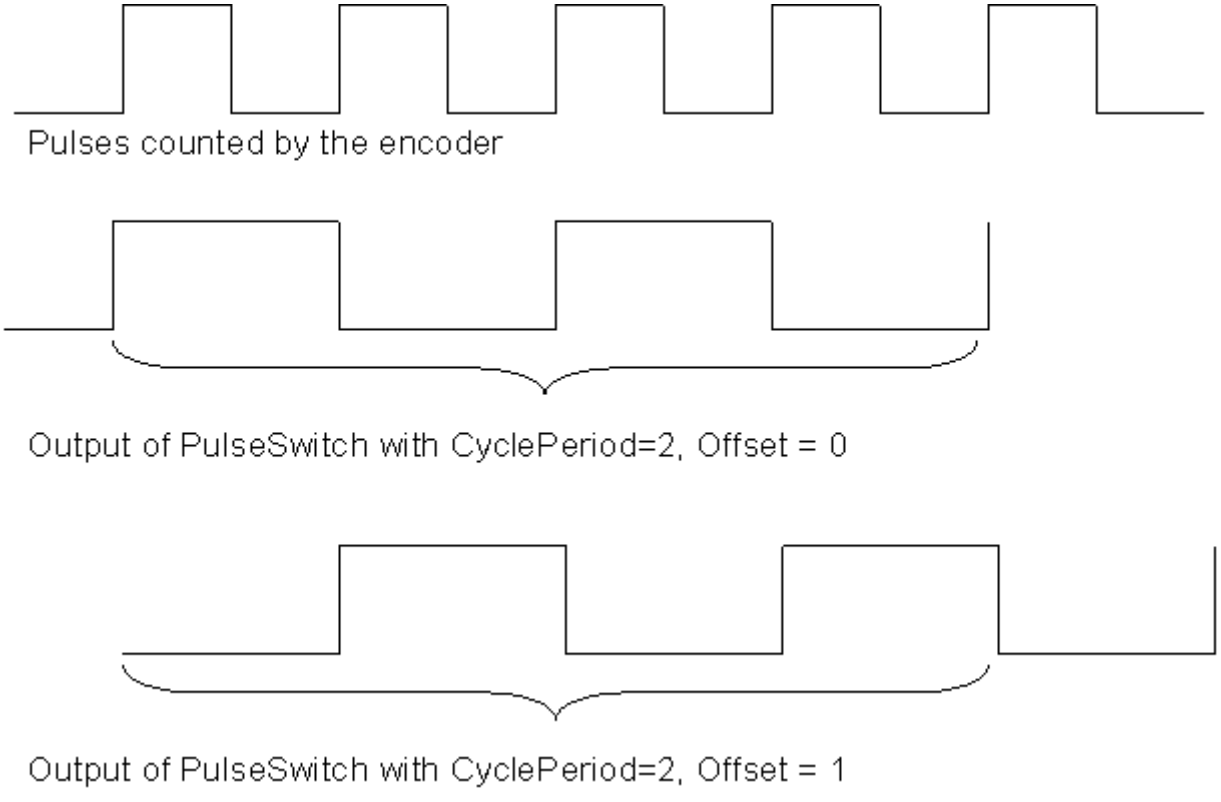
The values for CyclePeriod, Hysteresis and Offset are angle-values. They are individually changeable online.



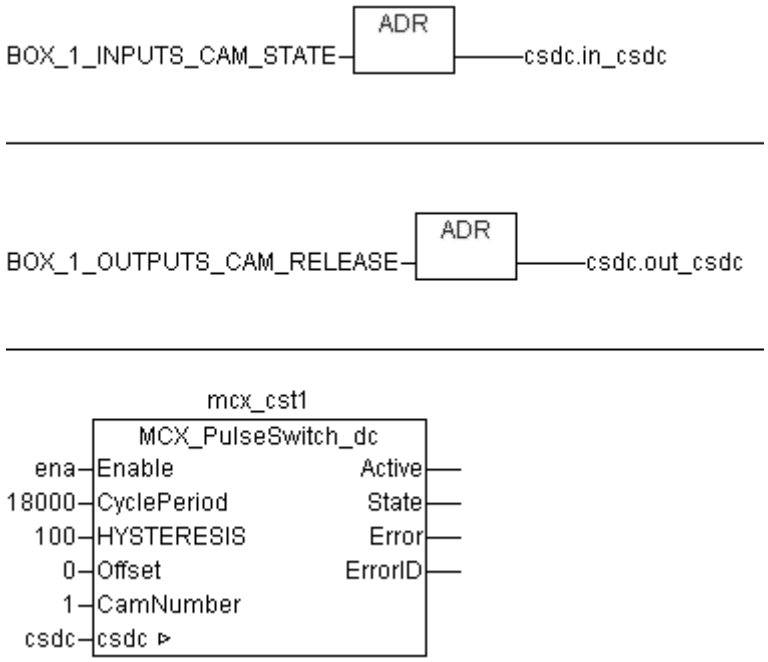
CyclePeriod is a fraction of 1 revolution (at maximum 1 revolution).The PulseSwitch will be switched on when the position is between CyclePeriod/2 and CyclePeriod " Hysteresis (shown green). The PulseSwitch will be switched off when the position is between 0 and CyclePeriod/2 " Hysteresis (shown red). It will not be changed otherwise, this prevents a toggling of the output in case the encoder stopped close to the switching position.

For example, when 1 revolution is 360° and CyclePeriod is 10°, it will be switched on for 5° and switched off for 5°, in total 36 pulses will be generated per revolution.

Several MCX_PulseSwitche_dc may be used to generate coordinated outputs, delayed by offset to each other.



The actual state of the cam can also be monitored with the output "state".



Enable

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The function block is activated with Enable TRUE. With Enable=FALSE, the respective cam will be switched off immediately.

CyclePeriod

Data type	Default value	Range	Unit
WORD	-	-	-

Number of encoder pulses for one pulsewith-period. The value of CyclePeriod is automatically truncated to the resolution of the circle without notice. For example, all CyclePeriod greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

CyclePeriod can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

Hysteresis

Data type	Default value	Range	Unit
WORD	-	-	-

The value of Hysteresis is automatically truncated to the resolution of the circle without notice. For example, all Hysteresis greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

Hysteresis can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

Offset

Data type	Default value	Range	Unit
WORD	-	-	-

The value of Offset is automatically truncated to the resolution of the circle without notice. For example, all Offset greater than 36000 will be limited to 36000 (given if the default resolution of CI51x-ETHCAT cam is used).

Offset can be changed at any time. It will be immediately valid after sending to the CI51x-ETHCAT via the POU.

CamNumber

Data type	Default value	Range	Unit
WORD	-	1...32	-

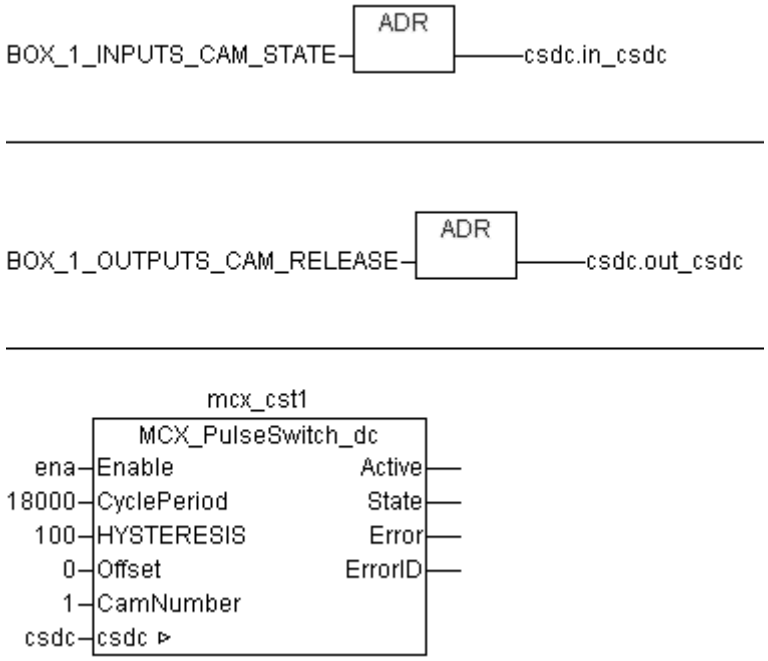
Number of the cam: this number runs from 1..32. The parameter cam_To_Track[0..31] must be used to map the cam to the available digital outputs. A configuration tool must be used for setting the cam_To_Track.

csdc

Data type: CSDCX_REF_TYPE

The 2 elements in_csdc and out_csdc of csdc must be initialized with the beginning address of the input data area and the output data area to facilitate proper function of the cam.

Output description



active

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Parameters (getBinary, Mode_On_Off) are transferred during active is TRUE.

state

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The parameter State shows the actual cam state. If State is TRUE, the cam is on. If State is FALSE, the cam is off.

Error

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

The output ERR signals any fault detected during the processing of the function block. This output always has to be checked in conjunction with the DONE output. If DONE is TRUE and ERR is TRUE, a processing fault was detected. The value of the ERNO output provides the according error number.

ErrorID

Data type	Default value	Range	Unit
WORD	-	-	-

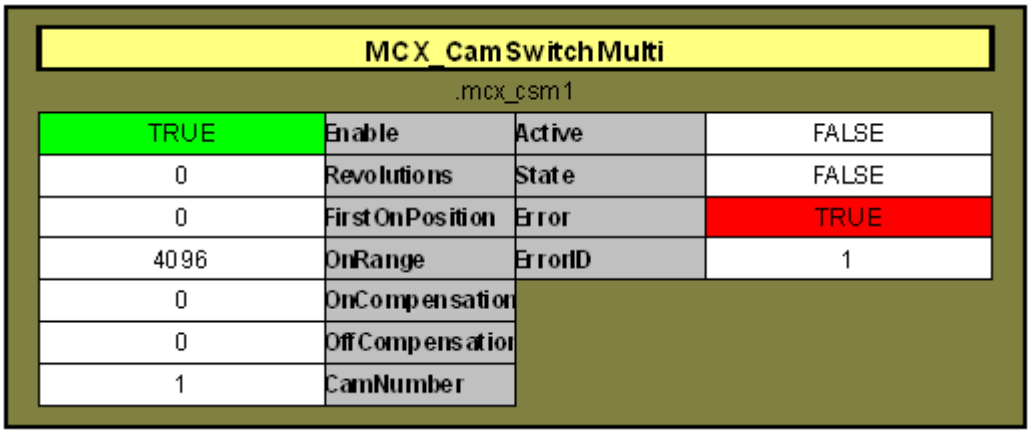
ErrorID is 1 if communication an error occurred.

ErrorID is 2 if a wrong parameter value is applied to the function block.

ErrorID is 3 if a wrong cam type is configured in the device CI51x-ETHCAT.

1.5.4.6.8 Visualization

For each of the MCX… function block, the library provides an integrated visualization element. In the configuration of these visualization elements, the placeholder has to be configured with the name of the function block's instance. The visualization looks for example as follows:



The values for all inputs and outputs and as well the name of the instance are shown.

1.5.4.7 CANopen library

Library file name: **CANopen_AC500_Vx.lib**

The library CANopen_AC500_Vx.lib is intended to be used with the CANopen master communication module.

Beside the cyclic IO communication, which is configured in the CANopen master configuration and controlled by the runtime system, additional features are implemented in the CANopen master. This library provides the possibility to use these features within a PLC application. With the contained function blocks the following functionality can be realized:

- sending and receiving of raw CAN telegrams (11bit and 29bit identifiers)
- reading of diagnosis information
- controlling of the NMT state machine of a CANopen node
- reading and writing of service data objects (SDOs)
- resetting of internal error counters
- reading of handshake error counters of synchronous IO update
- deactivating or activating of a CANopen node.

Additional system libraries are required by this library:

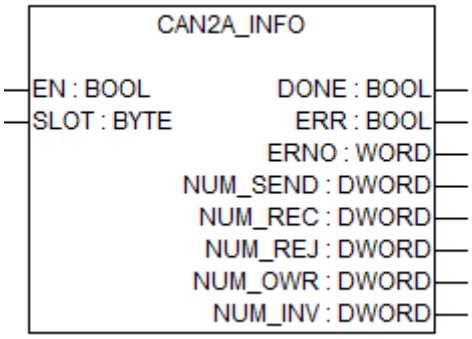
- CANopen_CME_AC500_V25.lib
- CANopen_CMN_AC500_V25.lib
- CMN_AC500_V24.lib
- SysExt_AC500_V10.lib

The CANopen library and the required system libraries will be automatically added to the PLC application when a CANopen master communication module is configured in the PLC configuration.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

1.5.4.7.1 Function blocks

CAN2A_INFO

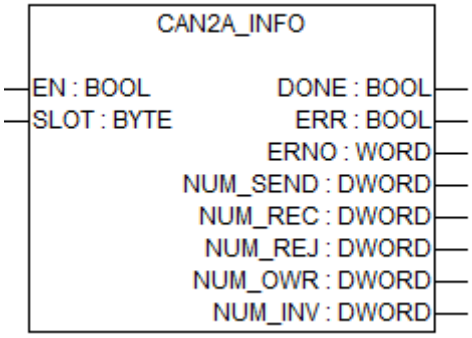


Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CAN2A_INFO outputs information concerning the status of the CAN 2.0A communication.

Input description

Using the function block CAN2A_INFO, various status information about the CAN 2.0A communication can be read.



The function block CAN2A_INFO outputs information concerning the status of the CAN 2.0A communication.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

EN

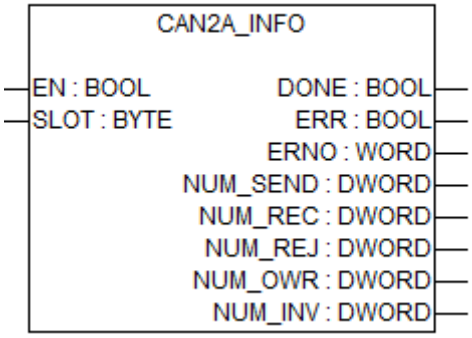
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



The function block CAN2A_INFO outputs information concerning the status of the CAN 2.0A communication.

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SEND

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_SEND displays the number of transmitted CAN 2.0B telegrams. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the number of received CAN 2.0B telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REJ displays the number of CAN 2.0B telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_OWR displays the number of CAN 2.0B telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_INV

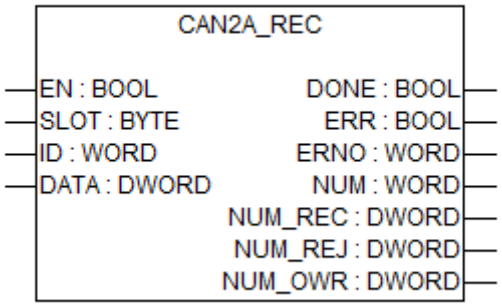
Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_INV displays the number of received CAN 2.0B telegrams that could not be assigned to any buffer defined in the controller configuration. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

Function call in ST

```
Info2a (EN      := Info2a_EN,
        SLOT    := Info2a_SLOT);


Info2a_DONE     := Info2a.DONE;
Info2a_ERR      := Info2a.ERR;
Info2a_ERNO     := Info2a.ERNO;
Info2a_NUM_SEND := Info2a.NUM_SEND;
Info2a_NUM_REC  := Info2a.NUM_REC;
Info2a_NUM_REJ  := Info2a.NUM_REJ;
Info2a_NUM_OWR  := Info2a.NUM_OWR;
Info2a_NUM_INV  := Info2a.NUM_INV;
```

CAN2A_REC



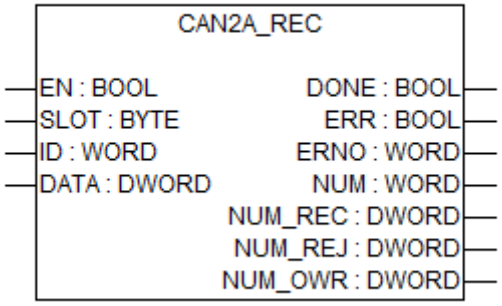
Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

With CAN2A_REC, CAN telegrams with 11 bit identifiers (base frame format or CAN2.0A) can be received.

Each identifier to be received needs a configured receive buffer.

Identifiers without configured receive buffer won´t be considered.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

ID

Data type	Default value	Range	Unit
WORD	-	-	-

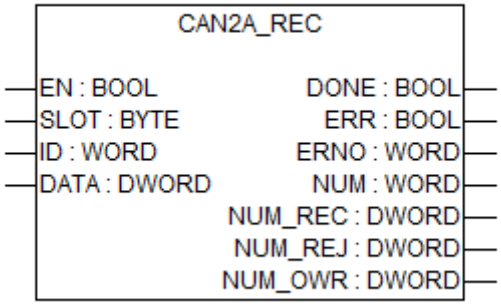
Input ID is used to specify the identifier of the CAN 2.0B telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the function block outputs.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

The input DATA is used to provide the pointer to data structure of the object which is to be written.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM

Data type	Default value	Range	Unit
WORD	-	-	-

Output NUM displays the number of CAN 2.0A telegrams not yet read from the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the number of received CAN 2.0B telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REJ displays the number of CAN 2.0B telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR

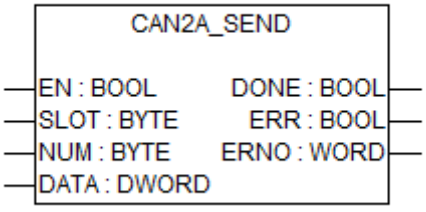
Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_OWR displays the number of CAN 2.0B telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

Function call in ST

```
Rec2a (EN   := Rec2a_EN,
       SLOT := Rec2a_SLOT,
       ID   := Rec2a_ID,
       DATA := ADR(Rec2a_DATA));


Rec2a_DONE     := Rec2a.DONE;
Rec2a_ERR      := Rec2a.ERR;
Rec2a_ERNO     := Rec2a.ERNO;
Rec2a_NUM      := Rec2a.NUM;
Rec2a_NUM_REC  := Rec2a.NUM_REC;
Rec2a_NUM_REJ  := Rec2a.NUM_REJ;
Rec2a_NUM_OWR  := Rec2a.NUM_OWR;
```

CAN2A_SEND



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

Using CAN2A_SEND, CAN telegrams with 11 bit identifiers according to CAN 2.0A can be transmitted.

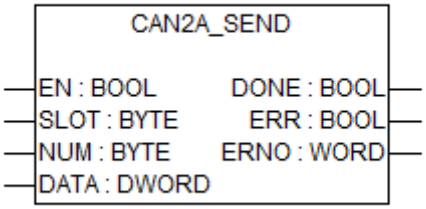
Every time a FALSE->TRUE edge is applied to input EN, CAN2A_SEND reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE and ERR = FALSE. A possibly occurring error is indicated by output ERR = TRUE.

The function block is able to transmit several telegrams to the Communication Module within one event. The Communication Module in turn transmits these telegrams sequentially via the bus.

When the function block is used with the Communication Module CM578-CAN the total length is 254 bytes. Thus, the maximum number of simultaneously transmitted telegrams depends on the sum of the individual telegram lengths. If all telegrams to be transmitted do not contain any other data than the 2 header bytes (identifier, RTR and DLC; Data Length Code DLC = 0), up to 127 telegrams can be transmitted at the same time (2 x 127 = 254). However, if all telegrams contain the maximum 8 bytes of data, only up to 25 telegrams can be transmitted to the Communication Module simultaneously ((2 + 8) x 25 = 250).

When the function block is used with the Communication Module CM598-CN the number of frames which can be sent at once is limited to 16.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NUM

Data type	Default value	Range	Unit
BYTE	-	-	-

Input NUM is used to specify the number of valid telegrams to be transmitted and stored starting at address DATA.

When the function block is used with the Communication Module CM578-CN, the upper limit of input NUM depends on the total length of all telegrams. The total length is calculated by the function block from the data length codes (DLC) of the individual telegrams and must not exceed 254 bytes. Otherwise an error message is generated. In such cases, the number of telegrams to be transmitted has to be chosen correspondingly that the total length does not exceed 254.

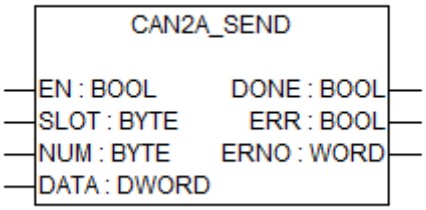
When the function block is used with the Communication Module CM598-CN the upper limit of input NUM is 16.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

The input DATA is used to provide the pointer to data structure of the object which is to be written.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

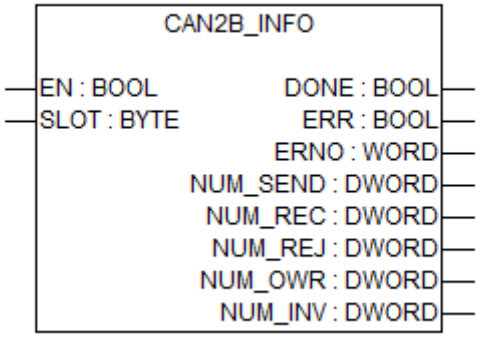
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
Send2a (  EN   := Send2a_EN,
          SLOT := Send2a_SLOT,
          NUM  := Send2a_NUM,
          DATA := ADR(Send2a_DATA));


Send2a_DONE    := Send2a.DONE;
Send2a_ERR     := Send2a.ERR;
Send2a_ERNO    := Send2a.ERNO;
```

CAN2B_INFO

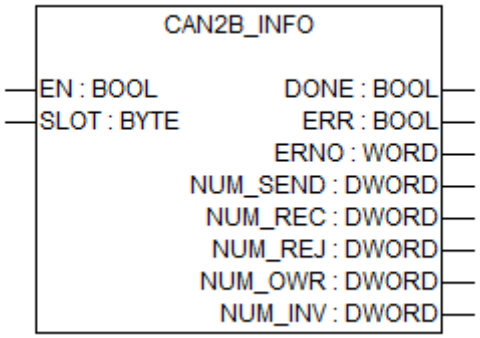


Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CAN2B_INFO outputs information concerning the status of the CAN 2.0B communication.

Input description

Using CAN2B_INFO, various status information about the CAN 2.0B communication can be read.



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

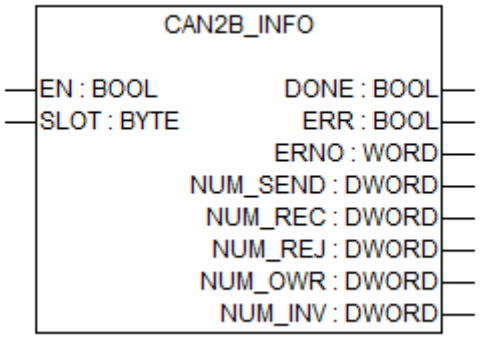
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SEND

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_SEND displays the number of transmitted CAN 2.0B telegrams. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the number of received CAN 2.0B telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REJ displays the number of CAN 2.0B telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_OWR displays the number of CAN 2.0B telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_INV

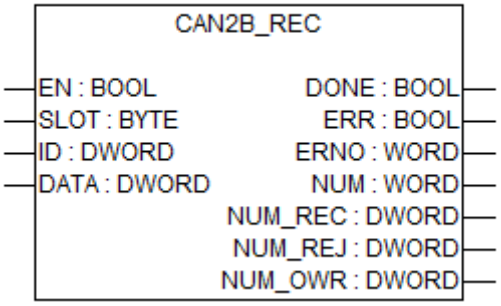
Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_INV displays the number of received CAN 2.0B telegrams that could not be assigned to any buffer defined in the controller configuration. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

Function call in ST

```
Info2b ( EN      := Info2b_EN,
         SLOT    := Info2b_SLOT);


Info2b_DONE     := Info2b.DONE;
Info2b_ERR      := Info2b.ERR;
Info2b_ERNO     := Info2b.ERNO;
Info2b_NUM_SEND := Info2b.NUM_SEND;
Info2b_NUM_REC  := Info2b.NUM_REC;
Info2b_NUM_REJ  := Info2b.NUM_REJ;
Info2b_NUM_OWR  := Info2b.NUM_OWR;
Info2b_NUM_INV  := Info2b.NUM_INV;
```

CAN2B_REC



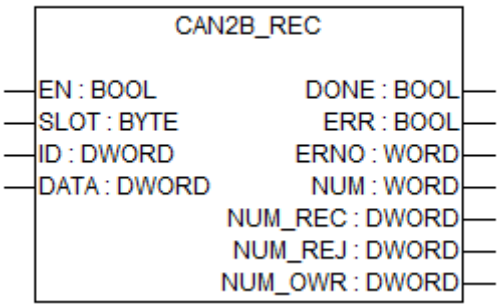
Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

With CAN2B_REC, CAN telegrams with 29 bit identifiers (extended frame format or CAN2.0B) can be received.

In the configuration the "extended" format" must be activated and the ID filter has to be parameterized accordingly. In addition each identifier to be received needs a configured receive buffer.

Identifiers without configured receive buffer won´t be considered.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

ID

Data type	Default value	Range	Unit
WORD	-	-	-

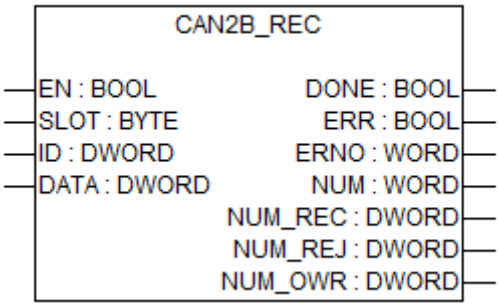
Input ID is used to specify the identifier of the CAN 2.0B telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the function block outputs.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

The input DATA is used to provide the pointer to data structure of the object which is to be written.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM

Data type	Default value	Range	Unit
WORD	-	-	-

Output NUM displays the number of CAN 2.0A telegrams not yet read from the buffer of the selected identifier. Stopping the PLC using the keypad or the online functions "STOP", "Reset" or "Reset (cold)" does not influence the output value.

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the number of received CAN 2.0B telegrams, regardless of whether a corresponding buffer has been set via the PLC configuration or not. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept.

NUM_REJ

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REJ displays the number of CAN 2.0B telegrams rejected due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

NUM_OWR

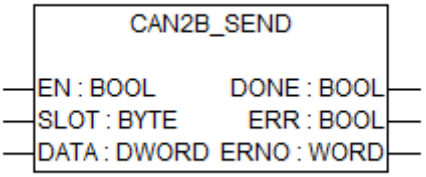
Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_OWR displays the number of CAN 2.0B telegrams overwritten by a new incoming telegram due to a full receive buffer. The value can be reset to 0 using the online functions "Reset" or "Reset (cold)". Stopping the PLC via the keypad or the online function "STOP" does not influence the output. In this case, the values are kept. Whether incoming telegrams are generally discarded in case of a full receive buffer or the oldest entry stored in the buffer is always overwritten by a new telegram can be set using the controller configuration.

Function call in ST

```
Rec2b (   EN   := Rec2b_EN,
          SLOT := Rec2b_SLOT,
          ID   := Rec2b_ID,
          DATA := ADR(Rec2b_DATA));


Rec2b_DONE     := Rec2b.DONE;
Rec2b_ERR      := Rec2b.ERR;
Rec2b_ERNO     := Rec2b.ERNO;
Rec2b_NUM      := Rec2b.NUM;
Rec2b_NUM_REC  := Rec2b.NUM_REC;
Rec2b_NUM_REJ  := Rec2b.NUM_REJ;
Rec2b_NUM_OWR  := Rec2b.NUM_OWR;
```

CAN2B_SEND



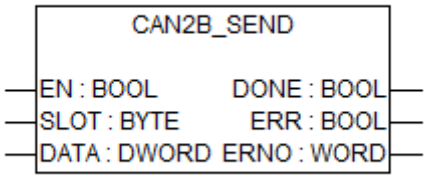
Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5:3
Type	Function block with historical values

Using CAN2B_SEND, CAN telegrams with 29 bit identifiers according to CAN 2.0B can be transmitted.

Every time a FALSE->TRUE edge is applied to input EN, CAN2B_SEND reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE and ERR = FALSE. A possibly occurring error is indicated by output ERR = TRUE.

Only one CAN 2.0B can be sent with each transmission.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

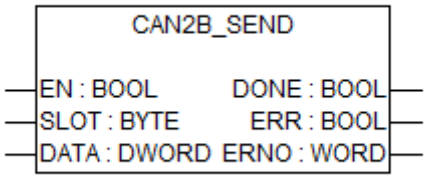
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

The input DATA is used to provide the pointer to data structure of the object which is to be written.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

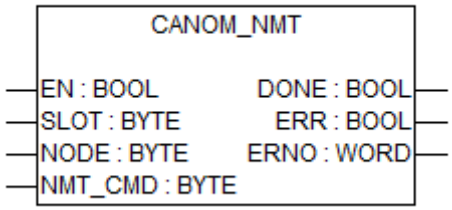
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
Send2b (  EN   := Send2b_EN,
          SLOT := Send2b_SLOT,
          DATA := ADR(Send2b_DATA));


Send2b_DONE    := Send2b.DONE;
Send2b_ERR     := Send2b.ERR;
Send2b_ERNO    := Send2b.ERNO;
```

CANOM_NMT



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

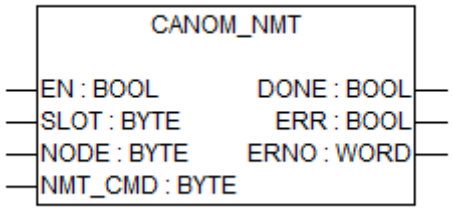
CANOM_NMT can be used to control the NMT state machine of one specific or all slaves.

Input description

CANOM_NMT can be used to control the NMT state machine of one specific or all slaves.

Every time a FALSE->TRUE edge is applied to input EN, CANOM_NMT reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

Normally, control of the slave operating states is performed by the automatic control of the CANopen Communication Module used as NMT master. However, for special applications it can be required to change the state of a specific slave 'manually'. This functionality can be achieved using CANOM_NMT.



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

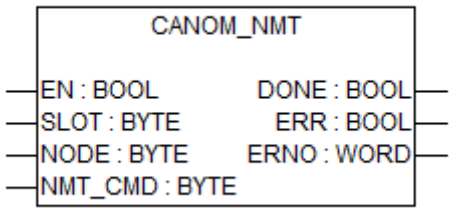
NMT_CMD

Data type	Default value	Range	Unit
BYTE	-	-	-

At input NMT_CMD, the NMT command to be sent is specified. The following NMT commands are defined:

NMT command	Constant	Meaning
1	CANOM_NMT_CMD_START	Start remote node (slave)
2	CANOM_NMT_CMD_STOP	Stop remote node (slave)
128	CANOM_NMT_CMD_ENTER_PREOP	Enter pre-operational: Set slave to pre-operational mode
129	CANOM_NMT_CMD_RESET_NODE	Reset node (slave)
130	CANOM_NMT_CMD__RESET_COM	Reset communication

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

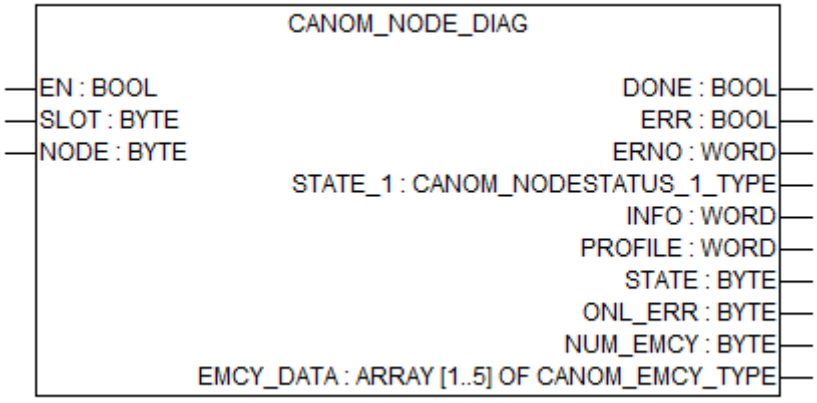
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
Nmt( EN       := Nmt_EN,
     SLOT     := Nmt_SLOT,
     NODE     := Nmt_NODE,
     NMT_CMD := Nmt_NMT_CMD);


Nmt_DONE      := Nmt.DONE;
Nmt_ERR       := Nmt.ERR;
Nmt_ERNO      := Nmt.ERNO;
```

CANOM_NODE_DIAG



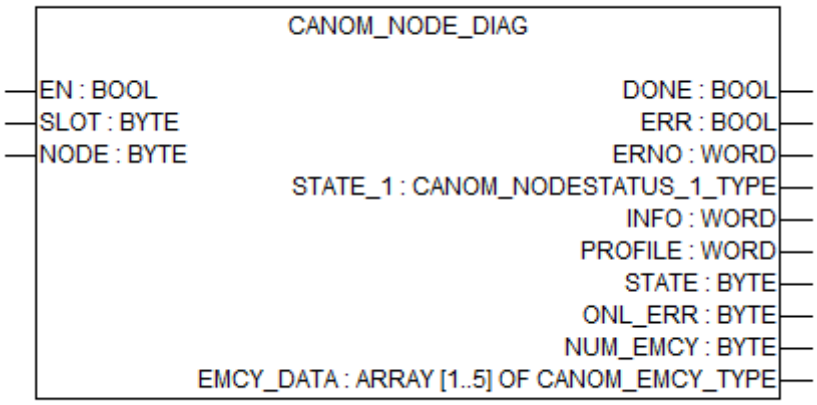
Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

Using the function block CANOM_NODE_DIAG, diagnosis data of the individual slaves can be requested.

Every time a FALSE->TRUE edge is applied at input EN, CANOM_NODE_DIAG reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The function block CANOM_NODE_DIAG is not supported by the Communication Module CM598-CAN. For the Communication Module CM598-CAN should be used the function block CANOM_NODE_DIAG_EXT.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

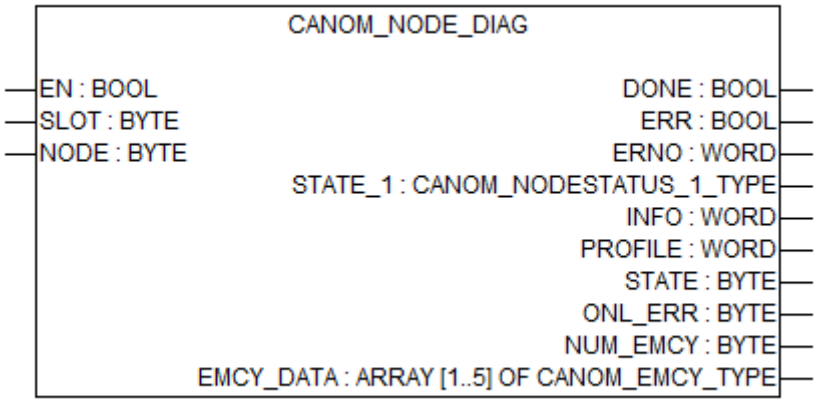
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE_1

Data type	Default value	Range	Unit
CANOM_NODES-TATUS_1_TYPE	-	-	-

The output STATE_1 provides diagnosis flags of the specified device.

INFO

Data type	Default value	Range	Unit
WORD	-	-	-

Output INFO provides additional information according to the CiA specification from the object 16#1000.

PROFILE

Data type	Default value	Range	Unit
WORD	-	-	-

Output PROFILE provides the profile number according to the CiA specification from the object 16#1000.

STATE

Data type	Default value	Range	Unit
BYTE	-	-	-

Output STATE provides the current operating condition of the relevant slave. STATE is only valid if DONE = TRUE and ERR = FALSE.

The following table describes the possible values of STATE and their meanings as specified in the CANopen specification.

STATE	Description
1	Disconnected
2	Connecting
3	Preparing
4	Prepared
5	Operational
127	Pre-Operational

ONL_ERR

Data type	Default value	Range	Unit
BYTE	-	-	-

ONL_ERR outputs a value describing possibly existing communication errors between the master Communication Module and the slave. ONL_ERR is only valid if DONE = TRUE and ERR = FALSE.

The error identifiers of ONL_ERR correspond to the IDs of output CANOM_ERR.EVENT of the function block CANOM_STATE. They are described in the table provided in the CANOM_STATE Function Block description.

NUM_EMCY

Data type	Default value	Range	Unit
BYTE	0	0 ... 5	-

NUM_EMCY provides the number of emergency messages contained in EMCY_DATA.

EMCY_DATA

Data type	Default value	Range	Unit
ARRAY OF CANOM_EMCY_TYPE	-	1 ... 5	-

EMCY_DATA outputs the up to 5 buffered emergency messages of the slave. The number of valid messages is output by NUM_EMCY.

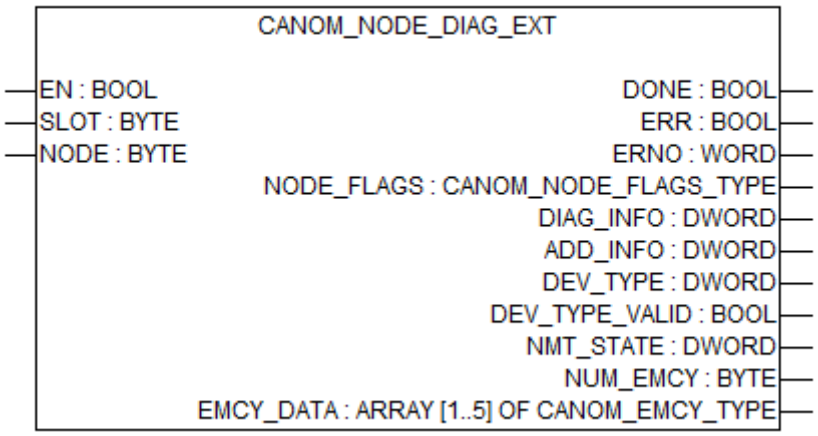
EMCY_DATA is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type CANOM_EMCY_TYPE is defined in the CANopen library.

Function call in ST

```
NodeDiag(  EN       := NodeDiag_EN,
           SLOT     := NodeDiag_SLOT,
           NODE     := NodeDiag_NODE;


NodeDiag_DONE       := NodeDiag.DONE;
NodeDiag_ERR        := NodeDiag.ERR;
NodeDiag_ERNO       := NodeDiag.ERNO;
NodeDiag_STATE_1    := NodeDiag.STATE_1;
NodeDiag_INFO       := NodeDiag.INFO;
NodeDiag_PROFILE    := NodeDiag.PROFILE;
NodeDiag_STATE      := NodeDiag.STATE;
NodeDiag_ONL_ERR    := NodeDiag.ONL_ERR;
NodeDiag_NUM_EMCY   := NodeDiag.NUM_EMCY;
NodeDiag_EMCY_DATA  := NodeDiag.EMCY_DATA;
```

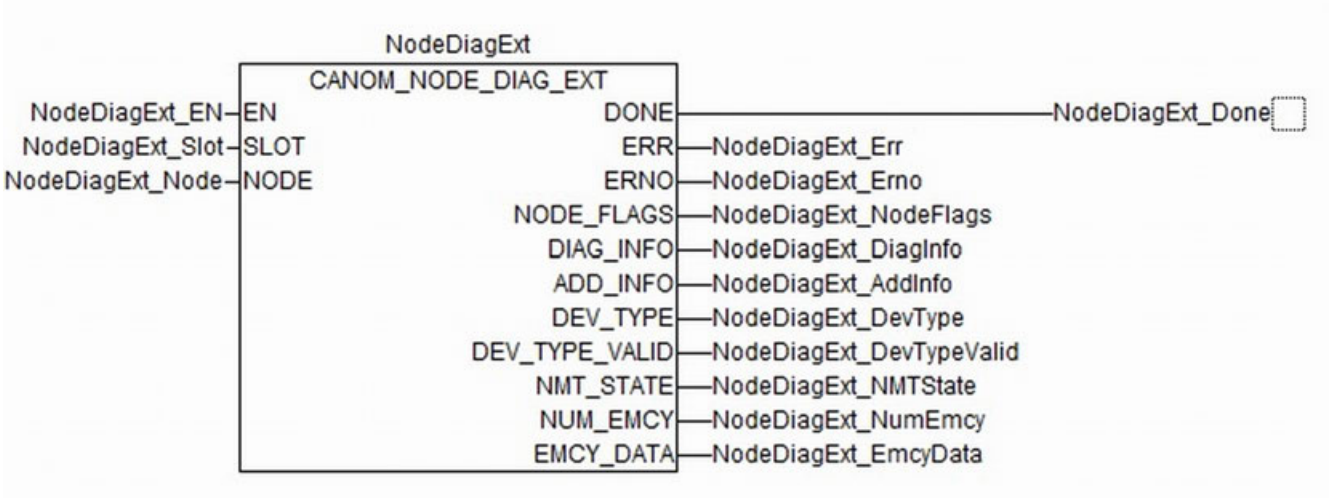
CANOM_NODE_DIAG_EXT



The function block CANOM_NODE_DIAG_EXT reads the diagnosis data of a slave.

Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

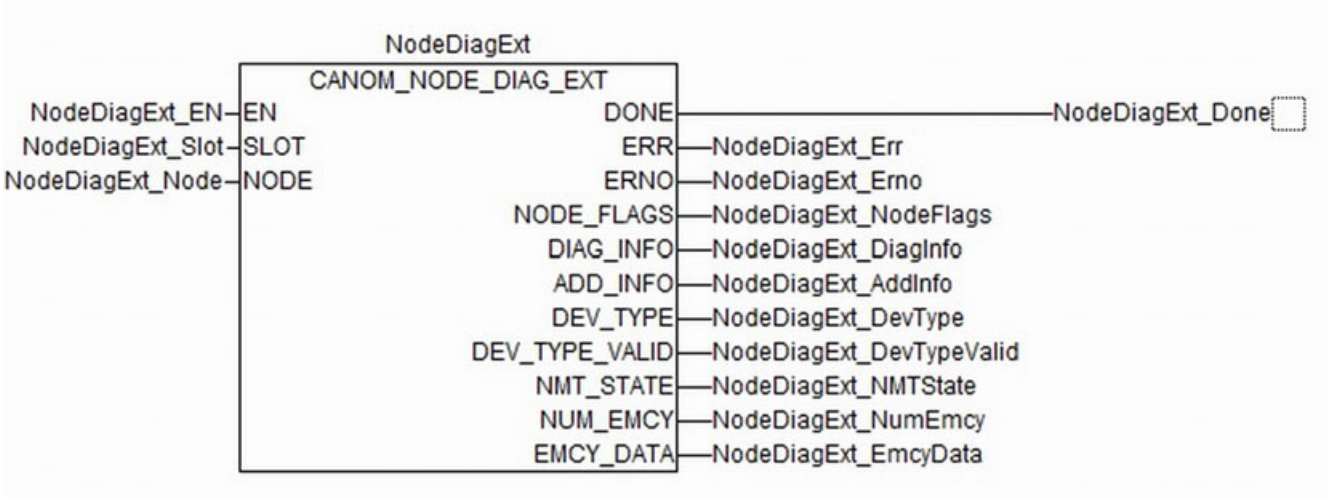
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE_FLAGS

Data type	Default value	Range	Unit
CANOM_NODE_FLAGS_TYPE	-	-	-

Output NODE_FLAGS provides diagnosis flags of the specified device.

DIAG_INFO

Data type	Default value	Range	Unit
DWORD	0	-	-

Output DIAG_INFO provides the last diagnosis info of the specific device.

The output can have the following values:

Value	Description
16#40420053	State of Node was not handled
16#C042004B	Connection to Node is lost
16#C042004D	Error of heartbeat protocol
16#C042004C	Error of guarding protocol
16#C042004A	Node is in an unexpected state
16#4042004E	Heartbeat protocol was started for the node
16#C0420027	Timeout exceeded during SDO transfer
16#C0420028	Error occurred during SDO transfer
16#C0420054	The device type of node is not equal to configured value
16#C042004F	Unexpected bootup message from the node was received
16#40420055	Emergency message from node was received

ADD_INFO

Data type	Default value	Range	Unit
DWORD	0	-	-

Output ADD_INFO provides additional information.

DEV_TYPE

Data type	Default value	Range	Unit
DWORD	0	-	-

Output DEV_TYPE provides the device type according to the CiA specification from the object 16#1000.

NMT_STATE

Data type	Default value	Range	Unit
DWORD	0	-	-

Output NMT_STATE provides the current NMT state of the specified CANopen device. The output can have the following values:

Value	Constant	Description
16#00000000	CANOM_NODE_NMT_STATE_UNKNOWN	Node NMT State is unknown
16#00000001	CANOM_NODE_NMT_STATE_INITIALISING	Node NMT State INITIALISING
16#00000002	CANOM_NODE_NMT_STATE_STOPPED	Node NMT State STOPPED
16#00000003	CANOM_NODE_NMT_STATE_OPERATIONAL	Node NMT State OPERATIONAL
16#00000004	CANOM_NODE_NMT_STATE_PRE_OPERATIONAL	Node NMT State PRE_OPERATIONAL
16#00000005	CANOM_NODE_NMT_STATE_RESET_APPLICATION	Node NMT State RESET_APPLICATION
16#00000006	CANOM_NODE_NMT_STATE_RESET_COMM	Node NMT State RESET_COMM

NUM_EMCY

Data type	Default value	Range	Unit
BYTE	0	0 ... 5	-

NUM_EMCY provides the number of emergency messages contained in EMCY_DATA.

EMCY_DATA

Data type	Default value	Range	Unit
ARRAY OF CANOM_EMCY_TYPE	-	1 ... 5	-

EMCY_DATA outputs the up to 5 buffered emergency messages of the slave. The number of valid messages is output by NUM_EMCY.

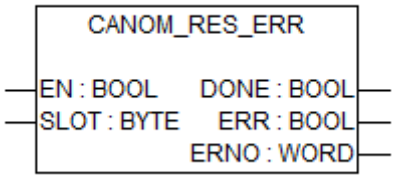
EMCY_DATA is only valid if DONE = TRUE and ERR = FALSE.

The structure of the type CANOM_EMCY_TYPE is defined in the CANopen library.

Function call in ST

```
NodeDiagExt(  EN       := NodeDiagExt_EN,
              SLOT     := NodeDiagExt_SLOT,
              NODE     := NodeDiagExt_NODE;


NodeDiagExt_DONE      := NodeDiagExt.DONE;
NodeDiagExt_ERR       := NodeDiagExt.ERR;
NodeDiagExt_ERNO      := NodeDiagExt.ERNO;
NodeDiagExt_NodeFlags := NodeDiagExt.NodeFlags;
NodeDiagExt_DiagInfo  := NodeDiagExt.DIAG_INFO;
NodeDiagExt_AddInfo   := NodeDiagExt.ADD_INFO;
NodeDiagExt_DevType   := NodeDiagExt.DEV_TYPE;
NodeDiagExt_ONL_ERR   := NodeDiagExt.DEV_TYPE_VALID;
NodeDiagExt_NMTState  := NodeDiagExt.NMT_STATE;
NodeDiagExt_NUM_EMCY  := NodeDiagExt.NUM_EMCY;
NodeDiagExt_EMCY_DATA := NodeDiagExt.EMCY_DATA;
```

CANOM_RES_ERR



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CANOM_RES_ERR can be used to reset various internal error indications and counters of the Communication Module.

> *The function block CANOM_RES_ERR is not supported by the Communication Module CM598-CAN.*

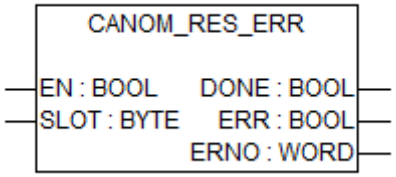
Input description

Using the function block CANOM_RES_ERR it is possible to reset the following Communication Module internal error indications and error counters output via the function block CANOM_STAT:

- STATE_BITS.EVENT
- STATE_BITS.TIMEOUT
- BUS_ERR
- BUS_OFF
- TOUT_ERR
- LOST_REC

For explanations of the error indications please refer to the description of the function block CANOM_STATE *Chapter 1.5.4.7.1.14 “CANOM_STATE ” on page 952*.

The reset is initiated by a FALSE->TRUE edge at input EN.



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

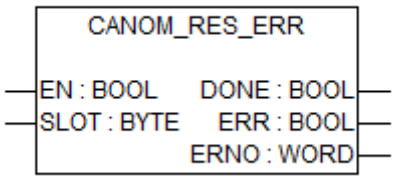
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

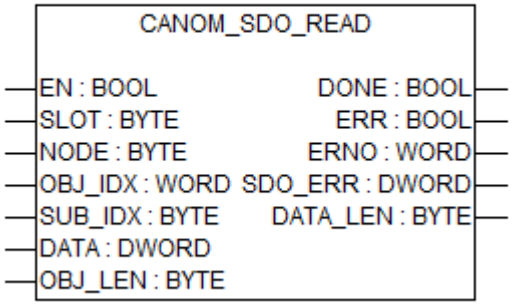
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ResErr( EN    := ResErr_EN,
        SLOT  := ResErr_SLOT);


ResErr_DONE   := ResErr.DONE;
ResErr_ERR    := ResErr.ERR;
ResErr_ERNO   := ResErr.ERNO;
```

CANOM_SDO_READ



The function block CANOM_SDO_READ can be used to read individual service data objects (SDOs) from a slave.

In the library CANopen_AC500_V25.lib the function block CANOM_SDO_READ was extended with the input OBJ_LEN.

Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

Using the function block CANOM_SDO_READ it is possible to read service data objects (SDOs) from a slave.

Every time a FALSE->TRUE edge is applied to input EN, CANOM_SDO_READ reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

The CANopen object model is defined in the communication profile and device profile specifications (see also System Technology of the CANopen Communication Modules). Furthermore, the device-specific objects are explained in the corresponding device description of the slave.

Input description

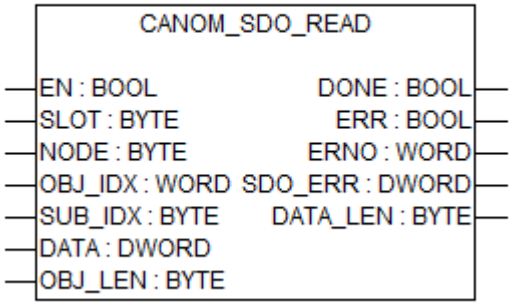


Fig. 15: CANOM_SDO_READ

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

OBJ_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input OBJ_IDX is used to specify the object index of the object which is to be written.

SUB_IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

Input SUB_IDX is used to specify the sub index of the object which is to be written.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

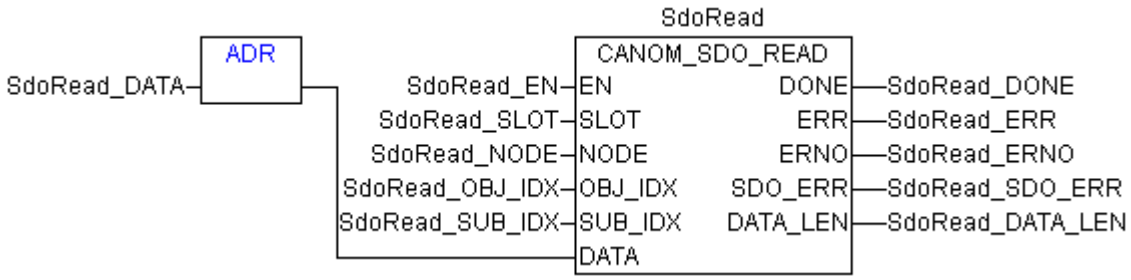
The input DATA is used to provide the pointer to data structure of the object which is to be written.

OBJ_LEN

Data type	Default value	Range	Unit
BYTE	0	0 ... 247	Number of Bytes

Input OBJ_LEN is used to specify the length of the object to be read from the CANopen slave.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

SDO_ERR

Data type	Default value	Range	Unit
DWORD	-	-	-

If the slave answered the SDO message with an abort message, the transmitted error code is output at SDO_ERR. The value output at SDO_ERR is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6003hex (24579dec).

DATA_LEN

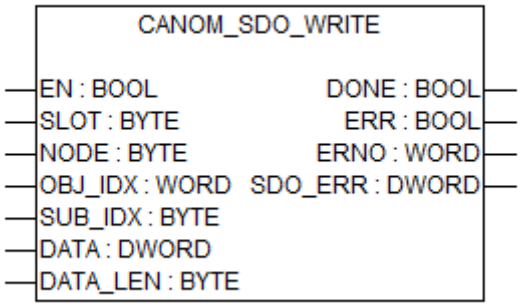
Data type	Default value	Range	Unit
BYTE	-	-	-

Input DATA_LEN is used to specify the size of the data structure which is provided at Input DATA.

Function call in ST

```
SdoRead ( EN       := SdoRead_EN,
          SLOT     := SdoRead_SLOT,
          NODE     := SdoRead_NODE,
          OBJ_IDX  := SdoRead_OBJ_IDX
          SUB_IDX  := SdoRead_SUB_IDX
          DATA     := ADR(SdoRead_DATA);


SdoRead_DONE       := SdoRead.DONE;
SdoRead_ERR        := SdoRead.ERR;
SdoRead_ERNO       := SdoRead.ERNO;
SdoRead_SDO_ERR    := SdoRead.SDO_ERR;
SdoRead_DATA_LEN   := SdoRead.DATA_LEN;
```

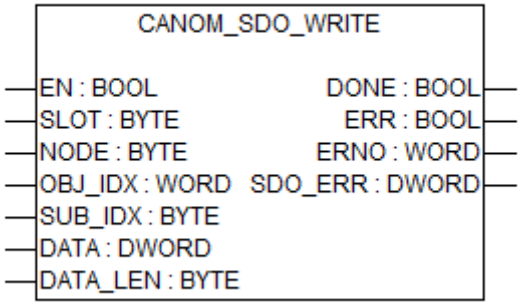
CANOM_SDO_WRITE



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CANOM_SDO_WRITE provides write access to the Object Directory of a CANopen slave using the SDO protocol.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

OBJ_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input OBJ_IDX is used to specify the object index of the object which is to be written.

SUB_IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

Input SUB_IDX is used to specify the sub index of the object which is to be written.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

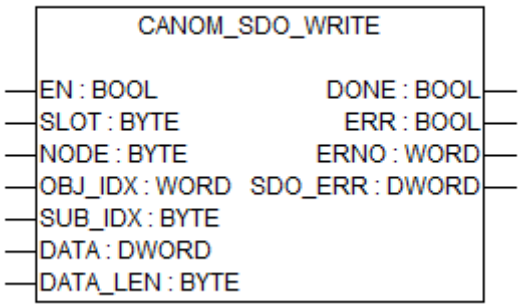
The input DATA is used to provide the pointer to data structure of the object which is to be written.

DATA_LEN

Data type	Default value	Range	Unit
BYTE	-	-	-

Input DATA_LEN is used to specify the size of the data structure which is provided at Input DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

SDO_ERR

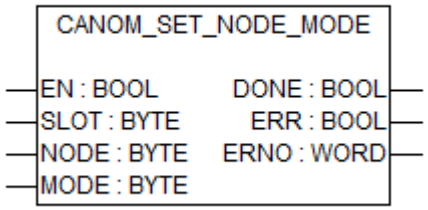
Data type	Default value	Range	Unit
DWORD	-	-	-

If the slave answered the SDO message with an abort message, the transmitted error code is output at SDO_ERR. The value output at SDO_ERR is only valid if DONE = TRUE, ERR = TRUE and ERNO = 6003hex (24579dec).

Function call in ST

```
SdoWrite (EN       := SdoWrite_EN,
          SLOT     := SdoWrite_SLOT,
          NODE     := SdoWrite_NODE,
          OBJ_IDX  := SdoWrite_OBJ_IDX,
          SUB_IDX  := SdoWrite_SUB_IDX,
          DATA     := ADR(SdoWrite_DATA,
          DATA_LEN := ADR(SdoWrite_DATA_LEN);


SdoWrite_DONE       := SdoWrite.DONE;
SdoWrite_ERR        := SdoWrite.ERR;
SdoWrite_ERNO       := SdoWrite.ERNO;
SdoWrite_SDO_ERR    := SdoWrite.SDO_ERR;
```

CANOM_SET_NODE_MODE



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

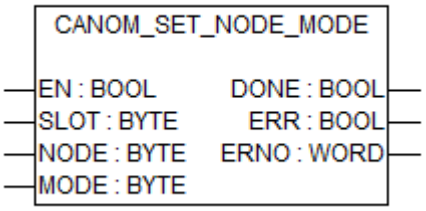
Using the function block CANOM_SET_NODE_MODE, one specific CANopen node can be deactivated or activated.

Every time a FALSE -> TRUE edge is applied to input EN, CANOM_SET_NODE_MODE reads the data at its inputs and sends a corresponding request message to the Communication Module.

Further FALSE -> TRUE edges at input EN are ignored until the processing of the active requests is completed. The completion of the request processing is indicated by DONE = TRUE.

All configured CANopen nodes are activated by default and are controlled by the CANopen Communication Module used as NMT master. However, for special applications it can be required to deactivate or activate one specific CANopen node 'manually'. This functionality can be achieved using CANOM_SET_NODE_MODE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

NODE

Data type	Default value	Range	Unit
BYTE	-	1...127	-

The Input NODE specifies the node ID of the slave.

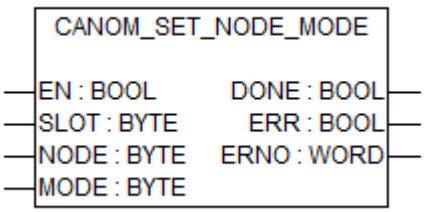
MODE

Data type	Default value	Range	Unit
BYTE	-	-	-

At input MODE, the mode of the CANopen node is specified. The following modes are defined:

NMT com-mand	Constant	Meaning
1	CANOM_NODE_MODE_ACTIVATE	Activate remote node (slave)
2	CANOM_NODE_MODE_DEACTIVATE	Deactivate remote node (slave)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

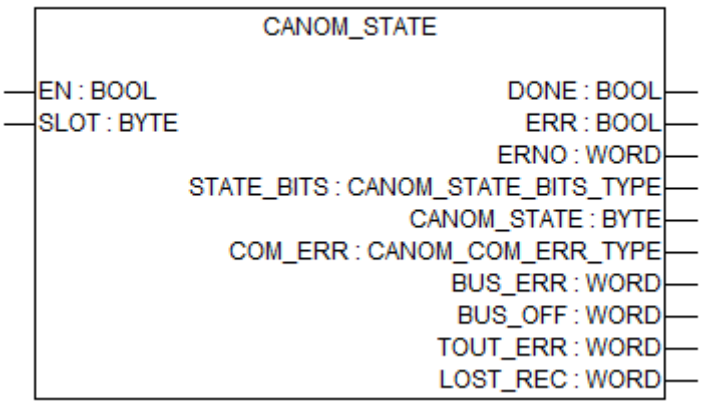
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
SetNodeMode(     EN    := SetNodeMode_EN,
                          SLOT    := SetNodeMode_SLOT,
                          NODE    := SetNodeMode_NODE,
                          MODE    := SetNodeMode_MODE);

SetNodeMode_DONE    := SetNodeMode.DONE;
SetNodeMode_ERR     := SetNodeMode.ERR;
SetNodeMode_ERNO    := SetNodeMode.ERNO;
```

CANOM_STATE

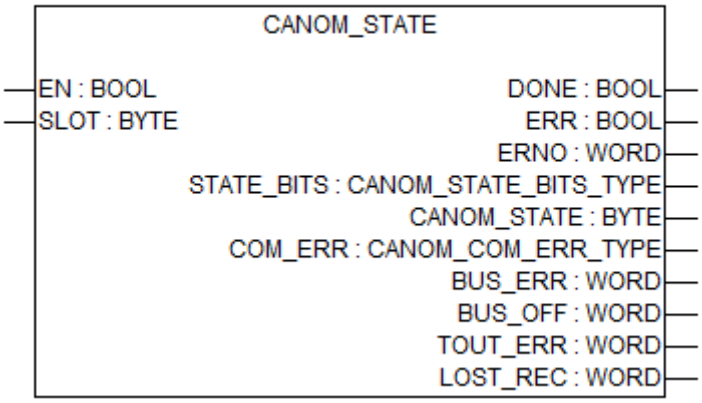


Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CANOM_STATE outputs the status of a CANopen Communication Module. The outputs provide information about the communication status and error events.

CANOM_STATE is active if input EN = TRUE. If the function block is active and if no errors occurred during Function Block processing, the current values are permanently displayed at the outputs.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

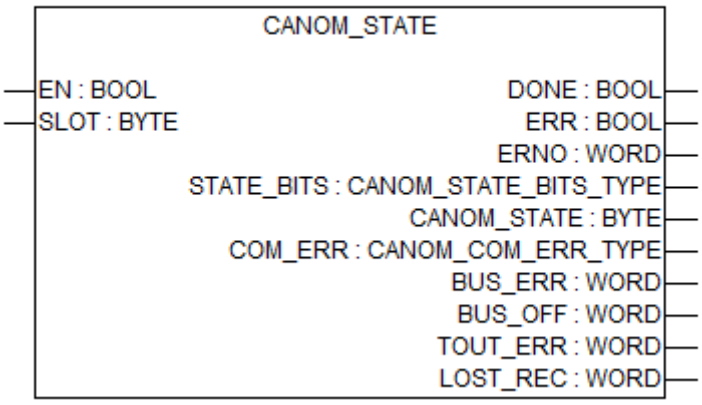
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE_BITS

Data type	Default value	Range	Unit
CANOM_STATE_BITS_TYPE	-	-	-

The Output STATE_BITS provides error or diagnosis flags of the CANopen Communication Module.

Some of them can be reset using the function block CANOM_RES_ERR *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

CANOM_STATE

Data type	Default value	Range	Unit
BYTE	-	-	-

CANOM_STATE outputs the general communication state of the CANopen master. The following states are defined:

State		Meaning
Dec	**Hex**	
0	00	OFFLINE
64	40	STOP
128	80	CLEAR
192	C0	OPERATE

CANOM_STATE = OFFLINE

If CANOM_STATE is set to OFFLINE, the CANopen Communication Module performs an initialization. After the initialization phase is completed, the Communication Module changes to STOP state.

CANOM_STATE = STOP

If CANOM_STATE has the value STOP, the Communication Module is completely initialized. In this state the Communication Module is ready to receive configuration data. There is no data exchange with the slaves. The Communication Module has this state if no user program is running.

CANOM_STATE = CLEAR

If the user program is started, the Communication Module changes from STOP to CLEAR and starts to establish the connections defined during configuration. When the setup has been completed successfully, the Communication Module moves to OPERATE state. If an error occurs during parameterization, the Communication Module changes back to STOP state.

CANOM_STATE = OPERATE

Normally, the Communication Module is in OPERATE state while a user program is running. In this state the master exchanges I/O data with the slaves. If an error occurs during this process and if ‘Auto Clear Mode’ has been selected during configuration, the Communication Module changes back to CLEAR state and tries to establish the connections again. If ‘Auto Clear Mode’ has not been selected during configuration, the Communication Module remains in OPERATE state in case of an error. If the user program is stopped, the Communication Module also changes back to STOP state.

CANOM_STATE is only valid, if EN = TRUE and ERR = FALSE.

COM_ERR

Data type	Default value	Range	Unit
CANOM_COM_ERR_TYPE	-	-	-

The Output COM_ERR provides detailed information to the corresponding output STATE_BITS.

The element Address contains the node address of the faulty device and the element Event the error code.

If several errors occur simultaneously, the output contains the error of the device with the lowest address.

COM_ERR can be reset using the function block CANOM_RES_ERR *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

BUS_ERR

Data type	Default value	Range	Unit
WORD	-	-	-

BUS_ERR outputs the number of occurred bus errors. A bus error occurs if the internal error frame counter exceeds a specific value. BUS_ERR is only valid if EN = TRUE and ERR = FALSE.

BUS_ERR can be reset using the function block CANOM_RES_ERR *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

BUS_OFF

Data type	Default value	Range	Unit
WORD	-	-	-

BUS_OFF outputs how often the Communication Module has been excluded from bus activities. An exclusion from the bus activities is performed, if an overflow of the internal error frame counter occurs. The Communication Module is automatically re-initialized after each overflow. BUS_OFF is only valid, if EN = TRUE and ERR = FALSE.

BUS_OFF can be reset using the function block CANOM_RES_ERR *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

TOUT_ERR

Data type	Default value	Range	Unit
WORD	-	-	-

TOUT_ERR outputs the number of telegrams that could not be transmitted successfully. The transmission of a telegram is considered as failed, if it could not be transmitted within 20 ms, for instance because the communication partner could not be contacted via the bus. TOUT_ERR is only valid, if EN = TRUE and ERR = FALSE.

TOUT_ERR can be reset using the function block CANOM_RES_ERR.

LOST_REC

Data type	Default value	Range	Unit
WORD	-	-	-

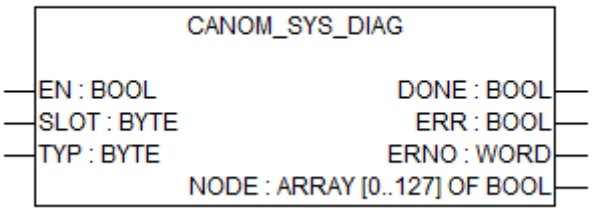
LOST_REC outputs the number of received telegrams that were rejected because they could not be processed successfully due to a CAN chip overload. LOST_REC is only valid, if EN = TRUE and ERR = FALSE.

LOST_REC can be reset using the function block CANOM_RES_ERR *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

Function call in ST

```
State ( EN         := State_EN,
        SLOT       := State_SLOT);


State_DONE         := State.DONE;
State_ERR          := State.ERR;
State_ERNO         := State.ERNO;
State_STATE_BITS   := State.STATE_BITS;
State_CANOM_STATE  := State.CANOM_STATE;
State_CANOM_ERR    := State.CANOM_ERR;
State_BUS_ERR      := State.BUS_ERR;
State_BUS_OFF      := State.BUS_OFF;
State_TOUT_ERR     := State.TOUT_ERR;
State_LOST_ERR     := State.LOST_ERR;
```

CANOM_SYS_DIAG



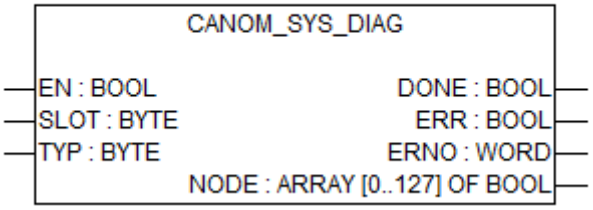
Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

The function block CANOM_SYS_DIAG outputs a bitfield as a state survey of all slaves (nodes) at output NODE. Three different survey types can be selected via input TYP.

The function block CANOM_SYS_DIAG outputs different status surveys of all slaves. Three survey types can be selected:

- configuration survey
- operational survey
- diagnosis survey

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

TYP

Data type	Default value	Range	Unit
BYTE	-	-	-

The input TYP is used to select the type of status survey displayed at output NODE.

TYP = 1 Configuration survey

Output NODE displays which slaves are successfully connected to the master (TRUE). Please note that the master only establishes connections to slaves that have been announced to the master during the definition of the configuration data.

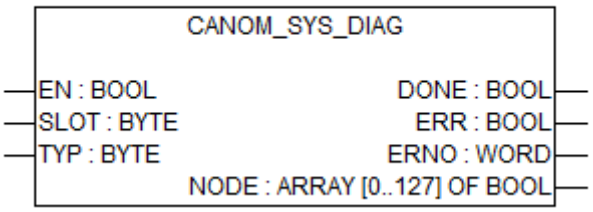
TYP = 2 Operational survey

Output NODE displays which slaves are error-free and in operation. A slave can only be announced as operational if it has been configured in the master. The operational survey can only be requested if the Communication Module is in OPERATE state.

TYP = 3 Diagnosis survey

Output NODE indicates which slaves report a diagnosis. The diagnosis survey can only be requested if the Communication Module is in OPERATE state.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE

Data type	Default value	Range	Unit
ARRAY OF BOOL	-	0 ... 127	-

NODE outputs the status survey as a bitfield. Each individual bit within this field represents one slave. The index number corresponds to the slave's node address. If a bit is set to TRUE, the state selected using TYP applies to the corresponding slave.

If e.g. TYP = 1 is selected and NODE[2] = TRUE, the slave with this node address was successfully configured by the master and is currently in operation. If NODE[2] = FALSE, the configuration of the specific slave has not yet been completed or the slave is not part of the master's configuration data.

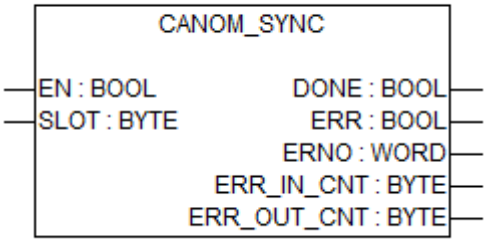
If TYP = 3, NODE[2] = TRUE for example means that the slave with the node address 2 has received an emergency message or that the diagnosis indication of the slave has changed. In this case, a diagnosis description can be requested using the function block CANOM_NODE_DIAG *Chapter 1.5.4.7.1.8 “CANOM_NODE_DIAG” on page 933*.

The at "NODE" output bitfield is only valid, if DONE = TRUE and ERR = FALSE.

Function call in ST

```
SysDiag( EN    := SysDiag_EN,
         SLOT  := SysDiag_SLOT,
         TYP   := SysDiag_TYP);


SysDiag_DONE   := SysDiag.DONE;
SysDiag_ERR    := SysDiag.ERR;
SysDiag_ERNO   := SysDiag.ERNO;
SysDiag_NODE   := SysDiag.NODE;
```

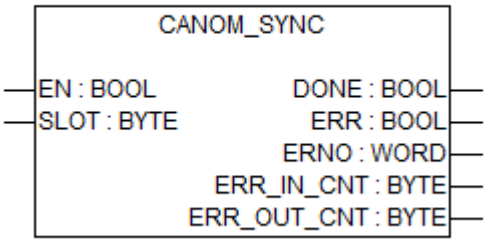
CANOM_SYNC



Parameter	Value
Included in library	CANopen_AC500_V25.lib
Available as of firmware	V2.5.3
Type	Function block with historical values

CANOM_SYNC is used to read the handshakes error counters of synchronous IO update mode. The counter ERR_IN_CNT will be set by the CANopen Master, when the IEC task was not started within the current bus cycle. And when the IEC task was not finished within the current bus cycle it will set the counter ERR_OUT_CNT.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

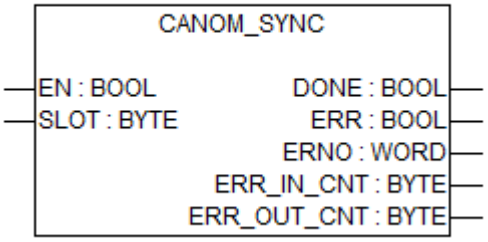
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERR_IN_CNT

Data type	Default value	Range	Unit
BYTE	-	0 ... 255	-

At Output ERR_IN_CNT the number of input handshake errors are provided.

ERR_OUT_CNT

Data type	Default value	Range	Unit
BYTE	-	0 ... 255	-

At Output ERR_OUT_CNT the number of output handshake errors are provided.

Function call in ST

```
CANSync (EN  :=xCANSync_EN,
         SLOT  :=byCANSync_SLOT);

xCANSync_DONE          := CANSync.DONE;
xCANSync_ERR           := CANSync.ERR;
byCANSync_ERNO         := CANSync.ERNO;
byCANSync_ERR_IN_CNT   := CANSync.ERR_IN_CNT;
byCANSync_ERR_OUT_CNT  := CANSync.ERR_OUT_CNT;
```

1.5.4.7.2 Structures group CAN

Group: CAN	
CAN2A_MESSAGE_TYPE	Telegram structure according to CAN 2.0A
CAN2B_MESSAGE_TYPE	Telegram structure according to CAN 2.0B

CAN2A_MESSAGE_TYPE

The library contains one general data type CAN2A_MESSAGE_TYPE to describe a CAN 2.0 A telegram with an 11 bit identifier. This data type is declared as follows:

```
TYPE CAN2A_MESSAGE_TYPE:
STRUCT
    ID: WORD;
    RTR: BOOL;
    DLC: BYTE;
    DATA: ARRAY [1 ... 8] OF BYTE;
END_STRUCT
END_TYPE
```

ID

Data type	Default value	Range	Unit
WORD	-	0 ... 2047	-

ID contains the general 11 bit identifier.

RTR

Data type	Default value	Range	Unit
BOOL	-	-	-

RTR (remote transmission request) contains the RTR bit in the telegram header.

DLC

Data type	Default value	Range	Unit
BYTE	-	-	-

DLC (data length code) contains the data length code in the telegram header and specifies the valid length in bytes for the user data following in DATA. Valid values for DLC are 0 to 8.

DATA

DATA (ARRAY[1..8] OF BYTE) contains the telegram data (if available). A CAN telegram can contain 0 to 8 bytes of data. The actual data length of a telegram is described by the data length code (DLC) contained in the telegram header. In DATA, only the first bytes are valid as specified by the DLC.

CAN2B_MESSAGE_TYPE

The library contains one general data type CAN2B_MESSAGE_TYPE to describe a CAN 2.0 B telegram with a 29 bit identifier. This data type is declared as follows:

```
TYPE CAN2B_MESSAGE_TYPE:
STRUCT
    ID: DWORD;
    RTR: BOOL;
    DLC: BYTE;
    DATA: ARRAY [1..8] OF BYTE;
END_STRUCT
END_TYPE
```

ID

Data type	Default value	Range	Unit
DWORD	-	-	-

ID contains the general 29 bit identifier, its range of values reaches from 0 to 536870911 (16#0 to 16# 1FFFFFFF).

RTR

Data type	Default value	Range	Unit
BOOL	-	-	-

RTR (remote transmission request) contains the RTR bit in the telegram header.

DLC

Data type	Default value	Range	Unit
BYTE	-	-	-

DLC (data length code) contains the data length code in the telegram header and specifies the valid length in bytes for the user data following in DATA. Valid values for DLC are 0 to 8.

DATA

DATA (ARRAY[1..8] OF BYTE) contains the telegram data (if available). A CAN telegram can contain 0 to 8 bytes of data. The actual data length of a telegram is described by the data length code (DLC) contained in the telegram header. In DATA, only the first bytes are valid as specified by the DLC.

1.5.4.7.3 Structures Group CANopen

Group: CANopen	
CANOM_COM_ERR_TYPE	Communication error
CANOM_EMCY_TYPE	Emergency telegram
CANOM_NODESTATUS_1_TYPE	Node diagnosis
CANOM_STATE_BITS_TYPE	Bits for Communication Module state description
CANOM_NODE_FLAGS_TYPE	Diagnosis flags of a CANopen Device

CANOM_COM_ERR_TYPE

Using CANOM_ERR, it is possible to locate communication errors more precisely. The output CANOM_ERR is represented as a structure of the type CANOM_COM_ERR_TYPE. In the CANopen library this data type is declared as follows:

```
TYPE CANOM_COM_ERR_TYPE:
STRUCT
   ADDRESS: BYTE;
   EVENT:   BYTE;
END_STRUCT
END_TYPE
```

The output CANOM_ERR is not supported by the Communication Module CM598-CAN.

ADDRESS

In case of an error, ADDRESS contains the node address of the faulty device. If ADDRESS has the value 255, the error is located in the Communication Module itself.

Data type	Default value	Range	Unit
BYTE	-	-	-

EVENT

In case of an error, EVENT contains the error causing event. The event refers to a single node address (ADDRESS <> 255) or the Communication Module itself (ADDRESS = 255).

ADDRESS <> 255 Error at subscribers with node address ADDRESS

Event	Description	Error source	Cause / Remedy
30	Slave monitoring timeout	Slave	Check connection and node address of the slave
31	Slave aborted operational mode	Slave	Reset slave
32	Sequence error in node guarding protocol	Slave	Reset slave
33	No response to configured remote frame PDO	Slave	Check whether slave remote frames are supported
34	No slave response during configuration	Slave	Check whether the slave is connected and ready for operation
35	Actual device profile of the slave differs from the configured device profile	Configuration	Check the profile supported by the slave
36	Actual device type of the slave differs from the configured device type	Configuration	Check the services supported by the slave
37	Unknown SDO response received	Slave	Slave does not meet the CiA protocol specifications
38	Length indicator of a received SDO response is not equal to 8	Slave	Slave does not meet the CiA protocol specifications
39	Slave is not processed. It is in the STOP state.	Configuration/ Communication Module	Deactivate the 'Auto Clear Mode'

ADDRESS = 255 Communication Module error

Event	Description	Error source	Cause / Remedy
52	Unknown process data handshake mode	Configuration	Contact customer support
56	Invalid data transfer rate	Configuration	Contact customer support
60	Node address configured twice	Configuration / other device	Check node addresses of all devices specified in configuration data
210	No configuration data	Configuration / Communication Module	Load configuration data into Communication Module

Event	Description	Error source	Cause / Remedy
212	Error while reading database	Configuration / Communication Module	Load configuration data into Communication Module again, contact customer support
220	Watchdog error	Controller	Contact customer support

Data type	Default value	Range	Unit
BYTE	-	-	-

CANOM_EMCY_TYPE

The data type CANOM_EMCY_TYPE corresponds to the format of the emergency telegram described in the CANopen communication profile and is defined as follows:

```
TYPE CANOM_EMCY_TYPE:
STRUCT
    ERROR_CODE: WORD;
    ERROR_REG:  BYTE;
    ERROR_DATA: ARRAY[1...5] OF BYTE;
END_STRUCT
END_TYPE
```

ERROR_CODE

Data type	Default value	Range	Unit
WORD	-	-	-

For the emergency object the emergency error codes described in the following table are defined in the CANopen communication profile.

Emergency error code		Description / Error cause
Decimal	Hexadecimal	
00000...00255	0000...00FF	Error on reset or no error
04096...04351	1000...10FF	General error
08192...08447	2000...20FF	Current error
08448...08703	2100...21FF	- error on the device input side
08704...08959	2200...22FF	- error inside the device
08960...09215	2300...23FF	- error on the device output side
12288...12543	3000...30FF	Voltage error
12544...12799	3100...31FF	- supply voltage error
12800...13055	3200...32FF	- error inside the device
13056...13311	3300...33FF	- error on the device output side
16384...16639	4000...40FF	Temperature error

Emergency error code		Description / Error cause
Decimal	**Hexadecimal**	
16640...16895	4100...41FF	- ambient temperature
16896...17151	4200...42FF	- temperature inside the device
20480...20735	5000...50FF	Hardware error in the device
24576...24831	6000...60FF	Software error in the device
24832...25087	6100...61FF	- device-internal software
25088...25343	6200...62FF	- application software
25344...25599	6300...63FF	Data
28672...28927	7000...70FF	Error in additional modules
32768...33023	8000...80FF	- monitoring
33024...33279	8100...81FF	- communication
36864...37119	9000...90FF	External error
61440...61695	F000...F0FF	Error of additional functions
65280...65535	FF00...FFFF	Device-specific errors

ERROR_REG

Data type	Default value	Range	Unit
BYTE	-	-	-

The variable ERROR_REG displays the error register value (object 1001 hex) of the slave. This value is transmitted by the slave as a part of the emergency message.

ERROR_DATA

Data type	Default value	Range	Unit
ARRAY OF BYTE	-	1 ... 5	-

If applicable, ERROR_DATA is used to output manufacturer-specific error information transmitted by the slave as part of the emergency message. For detailed information about the meaning of these data please refer to the particular device documentation.

CANOM_NODESTATUS_1_TYPE

The structure CANOM_NODESTATUS_1_TYPE is declared as follows in the CANopen library:

```
TYPE CANOM_NODESTATUS_1_TYPE:
STRUCT
    NO_RESPONSE: BOOL;
    EMCY_OVF:    BOOL;
    PRM_FAULT:   BOOL;
    GUARD_ACT:   BOOL;
    reserved1:   BOOL;
    reserved2:   BOOL;
    reserved3:   BOOL;
    DEACTIVATED: BOOL;
END_STRUCT
END_TYPE
```

NO_RESPONSE

Data type	Default value	Range	Unit
BOOL	-	-	-

If this bit is set, the slave with the node number specified at function block input NODE does not respond to the master requests. Normally NO_RESPONSE should be set to FALSE.

EMCY_OVF

Data type	Default value	Range	Unit
BOOL	-	-	-

This bit is set by the Communication Module, if more emergency messages were received from the called slave than the buffer can store (refer to function block outputs NUM_EMCY and EMCY_DATA).

PRM_FAULT

Data type	Default value	Range	Unit
BOOL	-	-	-

This bit is set, if the nominal slave configuration defined in the master differs from the actual slave configuration.

GUARD_ACT

Data type	Default value	Range	Unit
BOOL	-	-	-

This bit is set by the Communication Module, if the node guarding protocol for this slave is active. This is only a status indication. The active node guarding protocol between the master and the slave is not synonymous with a node guarding error.

reserved1

Data type	Default value	Range	Unit
BOOL	-	-	-

reserved2

Data type	Default value	Range	Unit
BOOL	-	-	-

reserved3

Data type	Default value	Range	Unit
BOOL	-	-	-

DEACTIVATED

Data type	Default value	Range	Unit
BOOL	-	-	-

This bit is set to TRUE, if the slave defined in the configuration data of the master is deactivated and not processed.

CANOM_STATE_BITS_TYPE

The structure STATE_BITS consists of six boolean variables displaying different communication states. In the CANopen library, the data type CANOM_STATE_BITS_TYPE is defined as follows:

```
TYPE CANOM_STATE_BITS_TYPE:
STRUCT
   CTRL:      BOOL;
   AUTO_CLR:  BOOL;
   NO_EXCH:   BOOL;
   FATAL:     BOOL;
   EVENT:     BOOL;
   reserved1: BOOL;
   TIMEOUT:   BOOL;
   reserved2: BOOL;
END_STRUCT
END_TYPE
```

CTRL

Data type	Default value	Range	Unit
BOOL	-	-	-

If this bit is TRUE, a parameter setting error occurred. During normal operation, CTRL should be FALSE. If this is not the case, the parameter and configuration data have to be checked.

AUTO_CLR

If AUTO_CLR is set to TRUE, the Communication Module stopped data exchange with all slaves due to communication errors and changed back to CLEAR state (see also CANOM_STATE).

Data type	Default value	Range	Unit
BOOL	-	-	-

NO_EXCH

This bit is set to TRUE, if no exchange of process data can be performed with one or several slaves. The error can be caused by the configuration data or the slaves themselves.

Data type	Default value	Range	Unit
BOOL	-	-	-

FATAL

If FATAL is set to TRUE, no communication via CANopen is possible due to a fatal internal error.

Data type	Default value	Range	Unit
BOOL	-	-	-

EVENT

EVENT is set to TRUE, if the Communication Module detects transmission errors. The number of occurring transmission errors is displayed at the corresponding outputs BUS_ERR and BUS_OFF. If the EVENT bit is set to TRUE, reset is only possible via the function block CANOM_RES_ERR ↳ *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

Data type	Default value	Range	Unit
BOOL	-	-	-

reserved1

This bit is reserved and currently not in use.

Data type	Default value	Range	Unit
BOOL	-	-	-

TIMEOUT

If TIMEOUT is set to TRUE, transmission of at least one telegram failed. The transmission of this telegram was aborted and its content is lost. TIMEOUT = TRUE is an indication that the communication partner could not be contacted via the bus. The number of failed transmissions is displayed at output TOUT_ERR. If the TIMEOUT bit is set to TRUE, reset is only possible using the function block CANOM_RES_ERR ↳ *Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941*.

Data type	Default value	Range	Unit
BOOL	-	-	-

reserved2

This bit is reserved and currently not in use.

Data type	Default value	Range	Unit
BOOL	-	-	-

CANOM_NODE_FLAGS_TYPE

The data type CANOM_NODE_FLAGS_TYPE has the following structure:

```
TYPE CANOM_NODE_FLAGS_TYPE :
STRUCT
    xSDO_TIMEOUT:       BOOL:=False;
    xSDO_ERROR:         BOOL:=False;
    CFG_DEFAULT:        BOOL:=False;
    xHARTBEAT_STARTED:  BOOL:=False;
    xGUARD_ERROR:       BOOL:=False;
    xCON_LOST:          BOOL:=False;
    xHARTBEAT_ERROR:    BOOL:=False;
    xUNEXPECTED_STATE:  BOOL:=False;
    xEMCY_RECEIVED:     BOOL:=False;
    xEMCY_BUFF_OVER:    BOOL:=False;
    xBOOTUP:            BOOL:=False;
    xUNEXPECTED_BOOTUP: BOOL:=False;
```

```
    xINVALID_PARAMETER: BOOL:=False;
    xSTATE_NOT_HANDLED: BOOL:=False;
    xDEACTIVATED:       BOOL:=False;
END STRUCT
END TYPE
```

xSDO_TIMEOUT Timeout during SDO transfer occured.

Data type	Default value	Range	Unit
BOOL	-	-	-

xSDO_ERROR: Error during SDO transfer occured.

Data type	Default value	Range	Unit
BOOL	-	-	-

CFG_DEFAULT Configuration default occured.

Data type	Default value	Range	Unit
BOOL	-	-	-

xHART-BEAT_STARTED Hartbeat protocol is started.

Data type	Default value	Range	Unit
BOOL	-	-	-

xGUARD_ERRO R A guarding message has been lost

Data type	Default value	Range	Unit
BOOL	-	-	-

xCON_LOST Node guarding has been lost.

Data type	Default value	Range	Unit
BOOL	-	-	-

xHART-BEAT_ERROR Error in hartbeat protocol recognized.

Data type	Default value	Range	Unit
BOOL	-	-	-

xUNEX-PECTED_STATE Node is in unexpected NMT state

Data type	Default value	Range	Unit
BOOL	-	-	-

xEMCY_RECEIVED Emergency telegram received.

Data type	Default value	Range	Unit
BOOL	-	-	-

xEMCY_BUFF_OVER Emergency buffer overflow occured.

Data type	Default value	Range	Unit
BOOL	-	-	-

xBOOTUP Expected boot-up message from node received.

Data type	Default value	Range	Unit
BOOL	-	-	-

xUNEX-PECTED_BOOTUP Unexpected boot-up message from node received.

Data type	Default value	Range	Unit
BOOL	-	-	-

xIN-VALID_PARAM-ETER Parameter set of node is invalid.

Data type	Default value	Range	Unit
BOOL	-	-	-

xSTATE_NOT_HANDLED At least one state has been omitted during the initialization sequence of the node.

Data type	Default value	Range	Unit
BOOL	-	-	-

xDEACTIVATED Node is deactivated and not handled by the master.

Data type	Default value	Range	Unit
BOOL	-	-	-

1.5.4.8 CD522 library

Library file name: **CD522_AC500_Vx.lib**

The CD522 library contains all function blocks necessary for using the function module CD522.

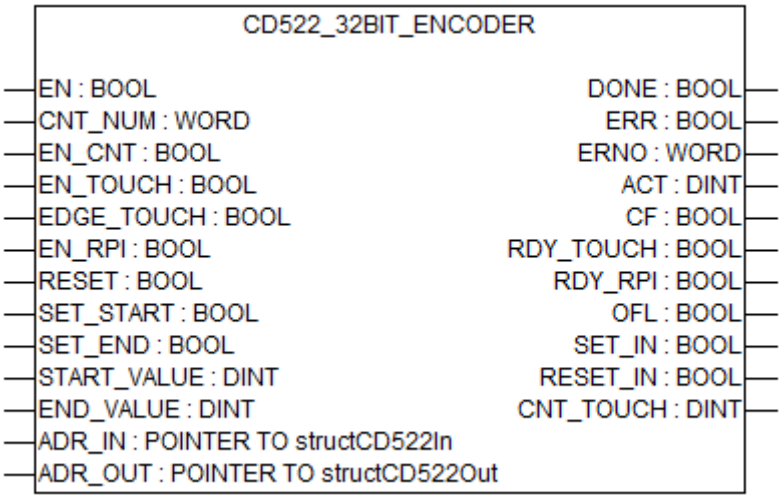
Once a CD522 has been added to the configuration, the library is automatically included with the next compilation of the project.

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The function blocks are available in AC500 control systems with a runtime system of version V1.0.2 or higher and S500 I/O modules (DC551) with firmware version V1.11 or higher.

1.5.4.8.1 Function blocks

CD522_32BIT_ENCODER



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

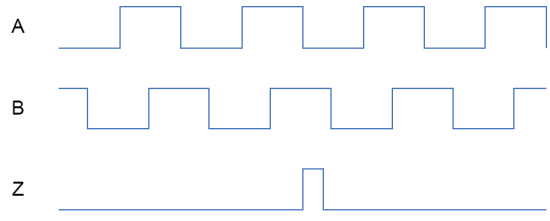
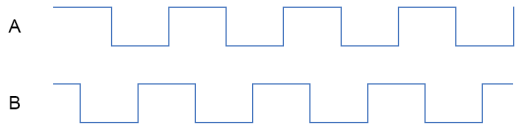
The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function encoder of module CD522, different operating modes are available. The function block CD522_32BIT_ENCODER should be used with one of these operating modes:

Operating Mode 11	"Incremental encoder"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x1 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 12	"Incremental encoder X2"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x2 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 13	"Incremental encoder X4"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x4 count, with possibility of touch/catch value, RPI function, set and reset actions.	

The module CD522 provides 2 encoder functions for relative positioning with 3 signals. 2 signals are used for rotation discrimination and pulse count, identified by A0 and B0 for counter0 and A1 and B1 for counter1. The third one is used in multi-turn encoder to count the number of rotation (mechanical zero), identified by Z0 for counter0 and Z1 for counter1.

The rotation is identified with a shift angle (90°) between A and B signal. In the module CD522, the clockwise rotation is identified with A signal in advance to B (see figure below).

Clockwise rotation - A signal ahead from B signal	**Counter-clockwise rotation - A signal late from B signal**
	
Clockwise rotation in CD522 module	Counter-clockwise rotation in CD522 module

Depending on which kind of operating mode is specified, the counting procedure will be x1, x2 or x4 count. Basically the x1 counting mode is used (mode 11). The encoder module discriminates the rotating way and count one pulse for each rising edge of the A signal.

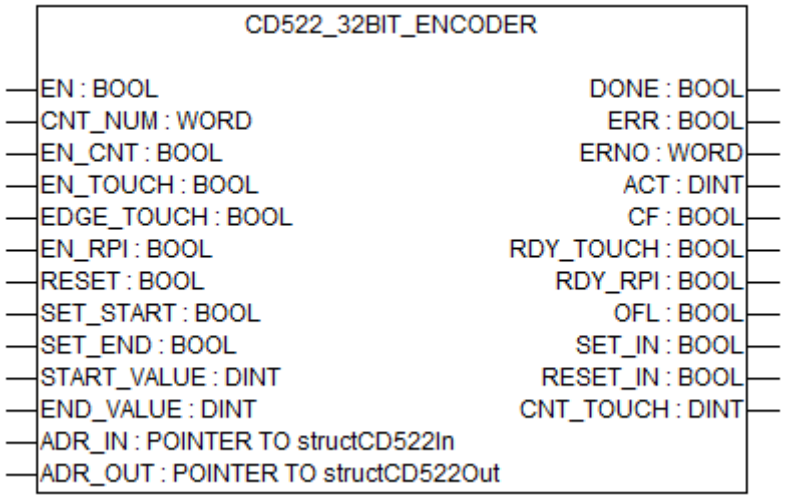
With clockwise rotation, function block CD522_32BIT_ENCODER counts downwards.

With counter-clockwise rotation, function block CD522_32BIT_ENCODER counts upwards.

In order to increase resolution, the x2 counting mode can be specified (mode 12). The encoder module counts one pulse on each rising edge of A signal and one pulse on each rising edge of B signal.

The resolution could be multiplied by 4, using the x4 counting mode (mode 13). The encoder module counts a pulse on both rising and falling edge of A signal and B signal.

Input description



EN_TOUCH The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated integer (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH (see figure below).

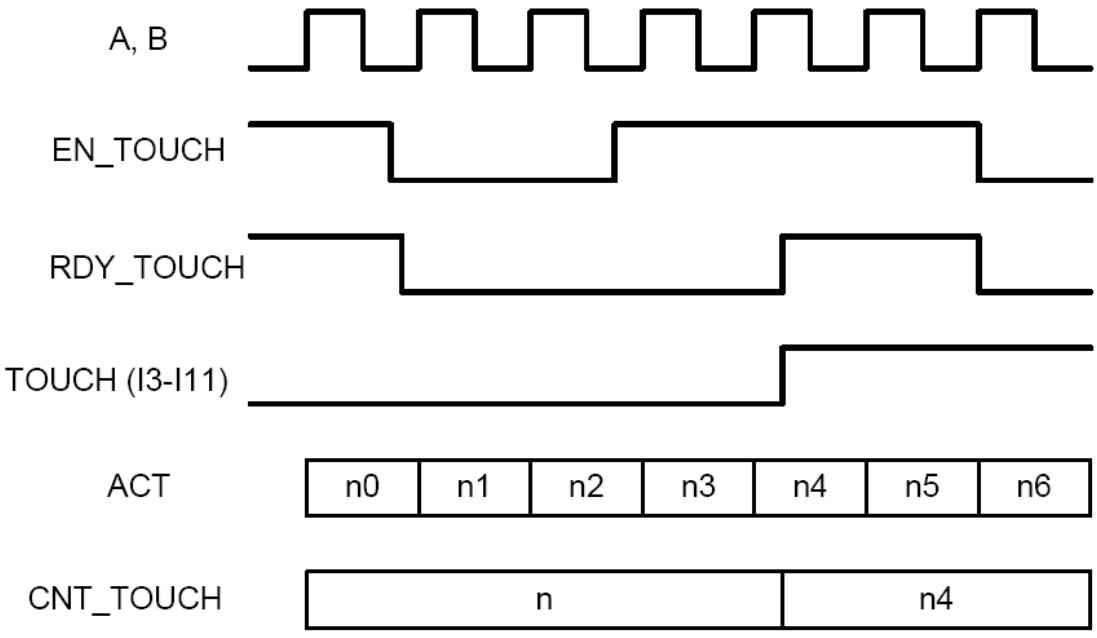


Fig. 16: Procedure and associated counting value with A signal

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

RPI procedure The RPI (Reference Point Initialization) is used to synchronize the counter value with the mechanical zero reference based on signal Z.

RPI procedure is enabled with control bit (EN_RPI). If this control bit is set, the module checks for the Z signal. When the signal appears, the set value is copied in the current counter value and RDY_RPI is set (see figure below).

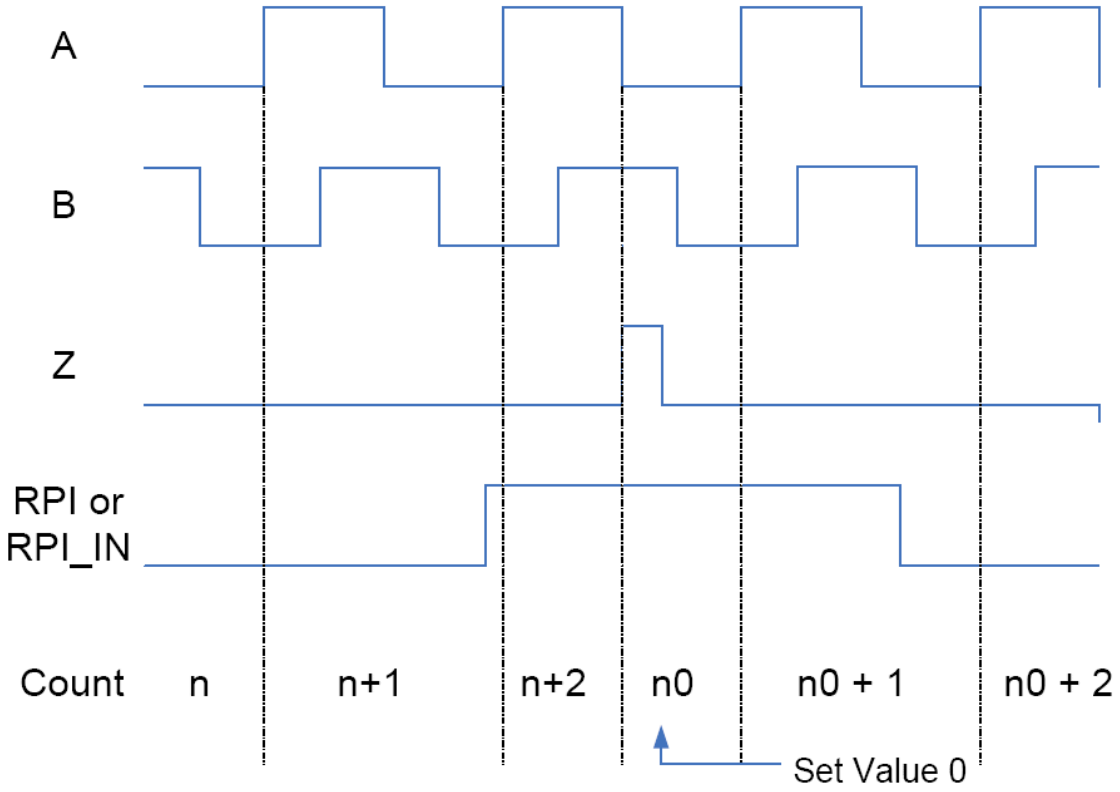


Fig. 17: RPI operation

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_CNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

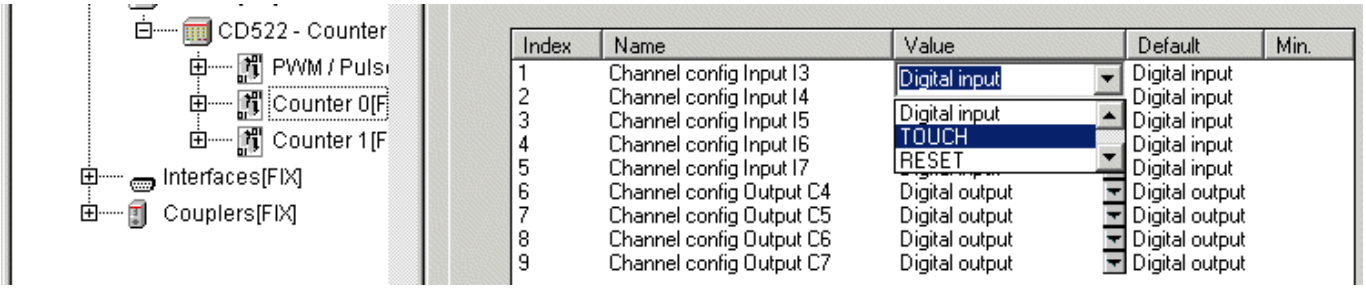
Input EN_CNT corresponds to bit 0 in "control byte".

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input I3 (for counters 0-A and 0-B)) or I11 (for counters 1-A and 1-B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.



The next measurement is again initiated by a rising edge at input EN_TOUCH.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Input EN_TOUCH corresponds to bit 6 in "control byte".

Only one function may be enabled at a time, either the RPI (reference point indicator) or TOUCH (touch trigger measurement). If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a RPI function will have a higher priority than TOUCH.

EDGE_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of input I3 (for counter 0) or I11 (for counter 1).

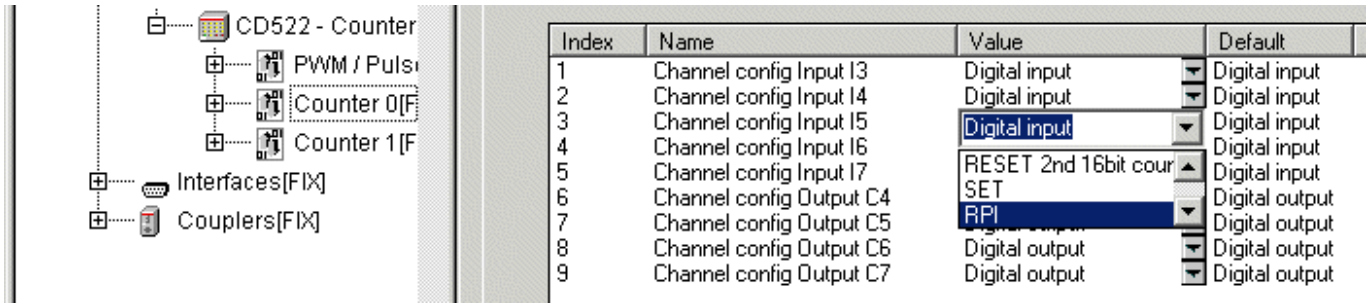
If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of input I3 (for counter 0) or I11 (for counter 1).

Input EDGE_TOUCH corresponds to bit 7 in "control byte".

EN_RPI

Data type	Default value	Range	Unit
BOOL	-	-	-

A rising edge at input EN_RPI enables a reference point initiator measurement. If input EN_RPI = TRUE, a rising edge at inputs I3, I4, I5, I6 or I7 (for counter 0) or I11, I12, I13 (for counter 1) validates the counter value capture and the counter reset during the capture.



Input EN_RPI corresponds to bit 4 in "control byte".

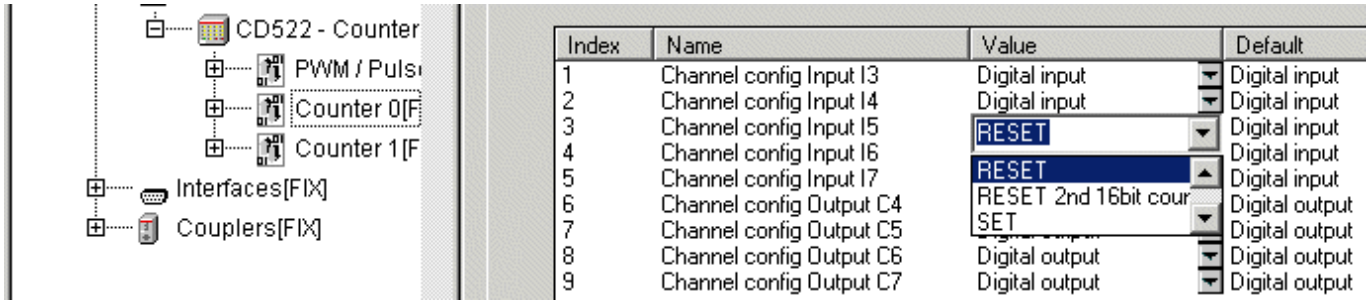
Only one function may be enabled at a time, either the RPI (reference point indicator) or TOUCH (touch trigger measurement). If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a RPI function will have a higher priority than TOUCH.

RESET

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input RESET=TRUE, the counter takes the values 0 to transfer it to output ACT. As long as input RESET=TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input I3, I4, I5, I6 or I7 (for counter 0) or I11, I12, I13, I14 or I15 (for counter 1) causes the function block to reset the value at output ACT.



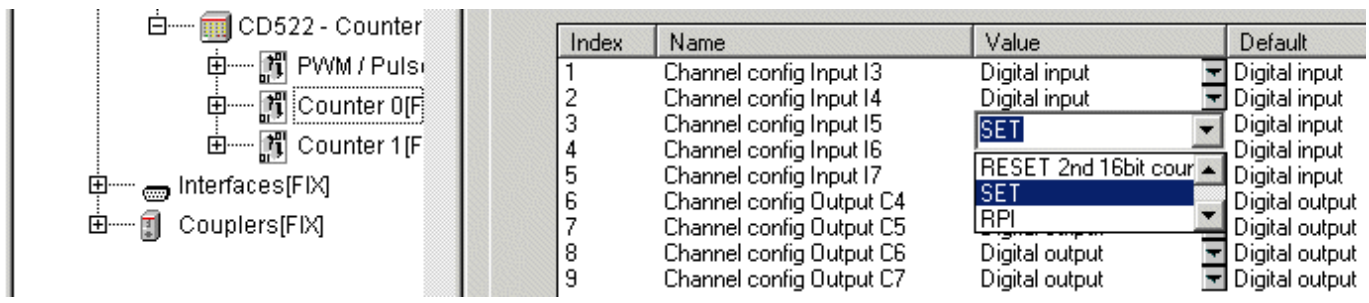
Input EN_RESET corresponds to bit 2 in "control byte".

SET_START

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input I3, I4, I5,I6 or I7 (for counter 0) or I11, I12, I13,I14 or I15 (for counter 1) causes the function block to store the START_VALUE value and to display this value at output ACT.



Input EN_SET corresponds to bit 1 in "control byte".

SET_END

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

START_VALUE

Data type	Default value	Range	Unit
DINT	-	-	-

The counter can be set to a start value. This value must be applied to the input START_VALUE.

If input SET=TRUE, counter takes this value.

Input START_VALUE corresponds to counter settings (DWORD) of CD522.

END_VALUE

Data type	Default value	Range	Unit
DINT	-	-	-

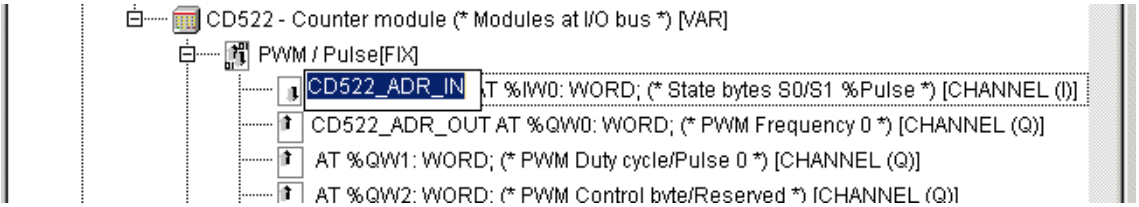
If the counter reaches the planned input END_VALUE, the binary output CF is set to TRUE and the value is stored.

Input END_VALUE corresponds to counter settings (DWORD) of CD522.

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

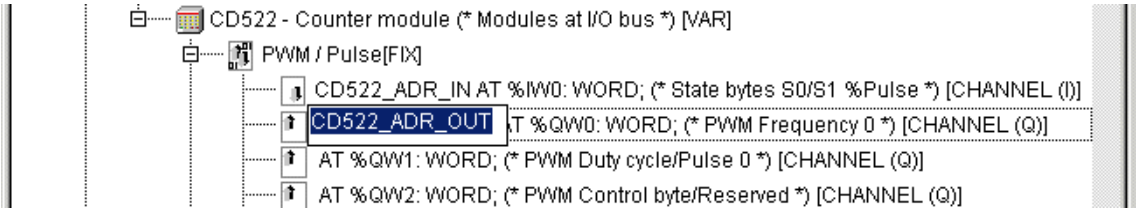
Example (for counter 0):



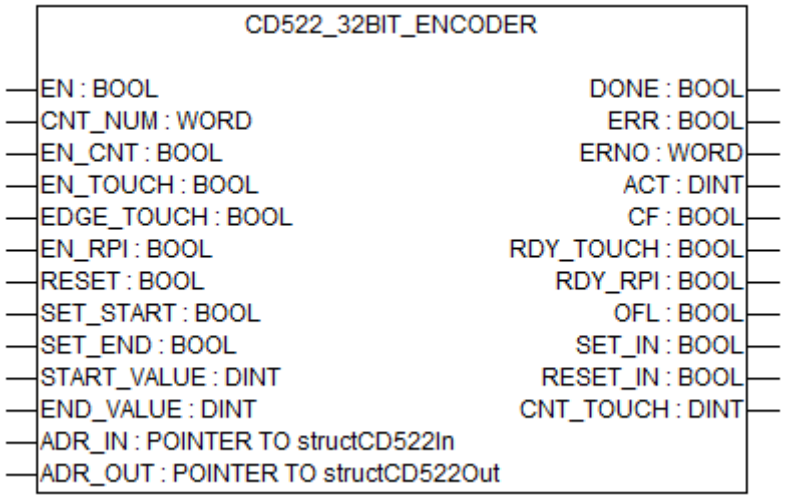
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT

Data type	Default value	Range	Unit
DINT	-	-	-

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

Output ACT corresponds to input DWORD in "32bit counter".

CF

Data type	Default value	Range	Unit
BOOL	-	-	-

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

This information is validated when input B is set to TRUE.

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The output RDY_TOUCH is set to TRUE when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Output RDY_TOUCH corresponds to input bit 2 in "state byte".

RDY_RPI

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RDY_RPI is set to TRUE when the RPI operation is done. If input EN_RPI is set to FALSE, the output RDY_RPI is set to FALSE.

Output RDY_RPI corresponds to input bit 6 in state byte.

OFL

Data type	Default value	Range	Unit
BOOL	-	-	-

The overflow is specified at the output OFL.

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648. Any exceeding or falling below of this value (depending to up and down use) will set OFL to TRUE.

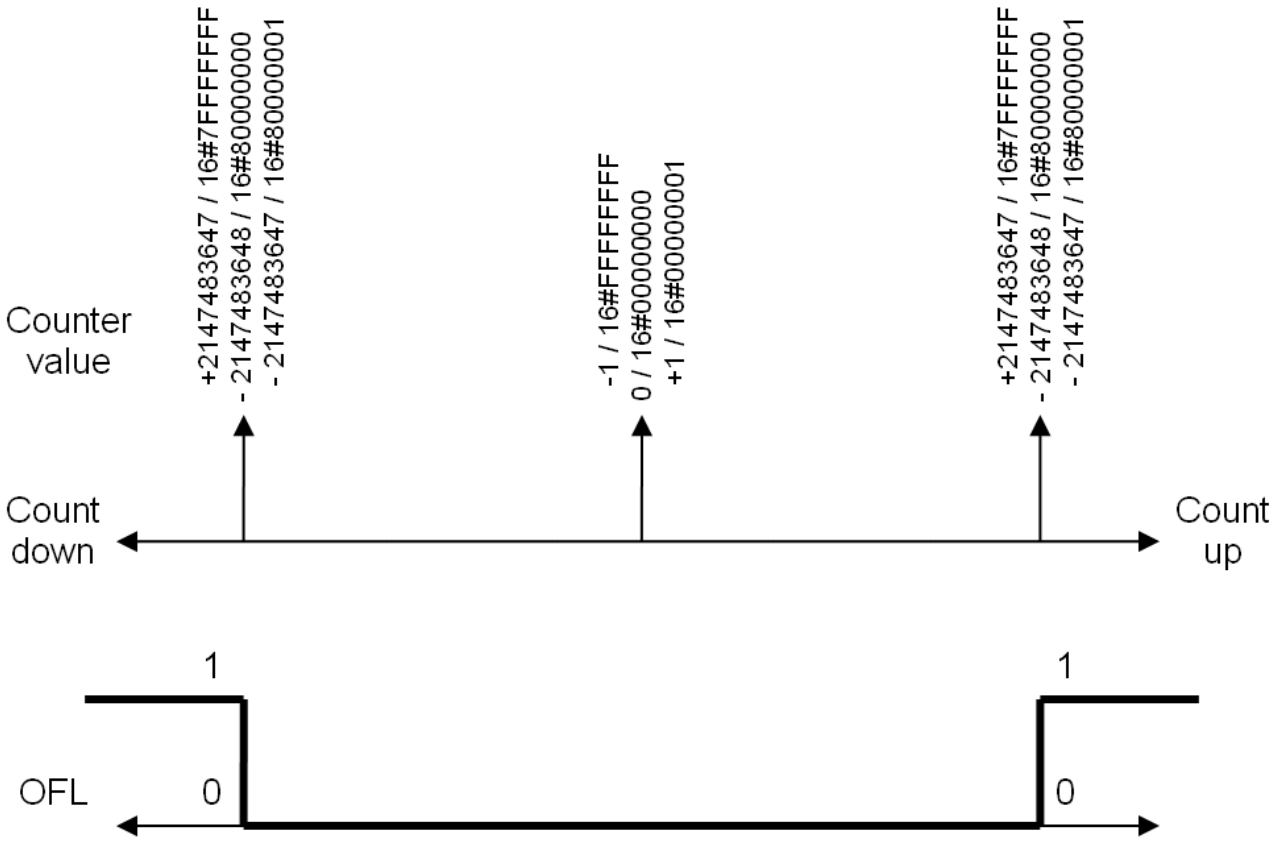


Fig. 18: Detection for output OFL

Output OFL corresponds to input bit 3 in state byte.

SET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output SET_IN is set to TRUE if one of the inputs is configured as SET input.

Output SET_IN corresponds to input bit 4 in "state byte".

RESET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RESET_IN is set to TRUE if one of the inputs is configured as RESET input.

Output RESET_IN corresponds to input bit 5 in "state byte".

CNT_TOUCH

Data type	Default value	Range	Unit
DINT	0	-	-

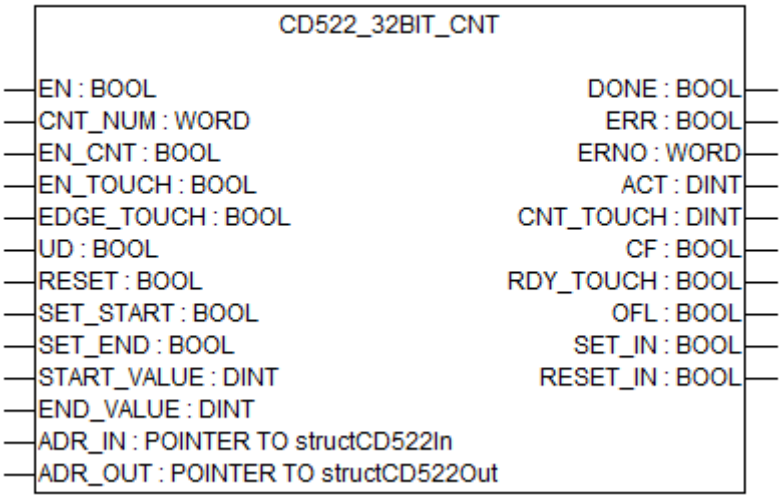
The output CNT_TOUCH displays the result of the catch/touch trigger measurement.

Output CNT_TOUCH corresponds to input DWORD in "TOUCH counter value".

Function call in ST

```
CD522_32BITENCODER (EN          :=CD522_32BITENCODER_EN,
                    EN_CNT      :=CD522_32BITENCODER_EN_CNT,
                    CNT_NUM     :=CD522_32BITENCODER_CNT_NUM,
                    EN_TOUCH    :=CD522_32BITENCODER_EN_TOUCH,
                    EDGE_TOUCH  :=CD522_32BITENCODER_EDGE_TOUCH,
                    EN_RPI      :=CD522_32BITENCODER_EN_RPI,
                    RESET       :=CD522_32BITENCODER_RESET,
                    SET_START   :=CD522_32BITENCODER_SET_START,
                    SET_END     := CD522_32BITENCONDER_SET_END,
                    START_VALUE :=CD522_32BITENCODER_START_VALUE,
                    END_VALUE   :=CD522_32BITENCODER_EN_VALUE,
                    ADR_IN      := ADR(CD522_32BITENCODER_ADR_IN),
                    ADR_OUT     := ADR(CD522_32BITENCODER_ADR_OUT));

CD522_32BITENCODER_DONE      :=CD522_32BITENCODER.DONE;
CD522_32BITENCODER_ERR       :=CD522_32BITENCODER.ERR;
CD522_32BITENCODER_ERNO      :=CD522_32BITENCODER.ERNO;
CD522_32BITENCODER_ACT       :=CD522_32BITENCODER.ACT;
CD522_32BITENCODER_CF        :=CD522_32BITENCODER.CF
CD522_32BITENCODER_RDY_TOUCH := CD522_32BITENCODER.RDY_TOUCH;
CD522_32BITENCODER_RDY_RPI   := CD522_32BITENCODER.RDY_RPI;
CD522_32BITENCODER_OFL       :=CD522_32BITENCODER.OFL;
CD522_32BITENCODER_SET_IN    :=CD522_32BITENCODER.SET_IN;
CD522_32BITENCODER_RESET_IN  :=CD522_32BITENCODER.RESET_IN;
CD522_32BITENCODER_CNT_TOUCH :=CD522_32BITENCODER.CNT_TOUCH;
```

CD522_32BIT_CNT



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2

Parameter	Value
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

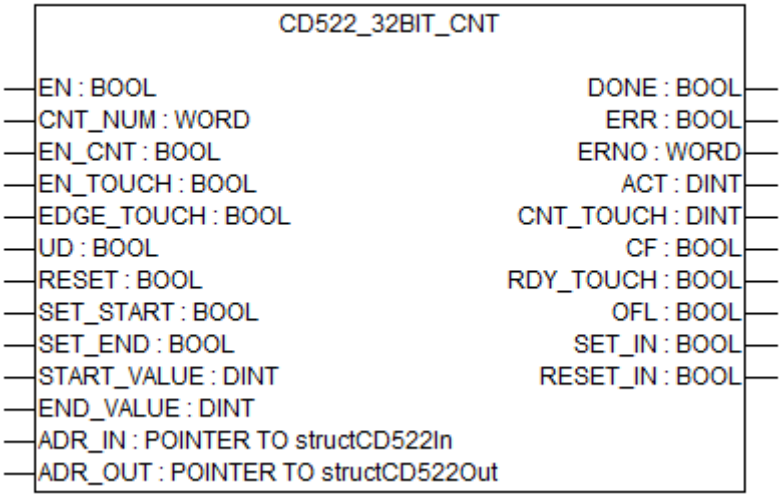
The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_32BIT_CNT should be used with one of these operating modes:

Operating Mode 1	"Up/Down counter (A)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use one up/down 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 2	"Up/Down counter with release input (B)"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 5	"Up/Down dynamic set (B)/rising edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical.	
Operating Mode 6	"Up/Down dynamic set (B)/falling edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical.	

The module CD522 provides 2 Up/Down 32-bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

Input description



EN_TOUCH

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated integer (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH (see figure below).

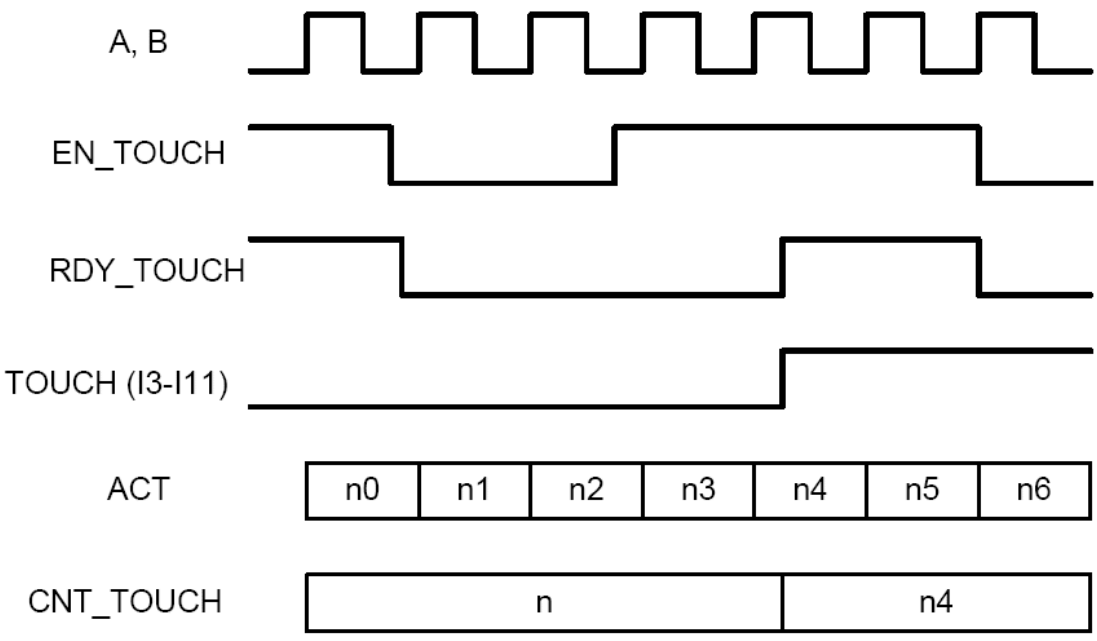


Fig. 19: Procedure and associated counting value with A signal

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_COUNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

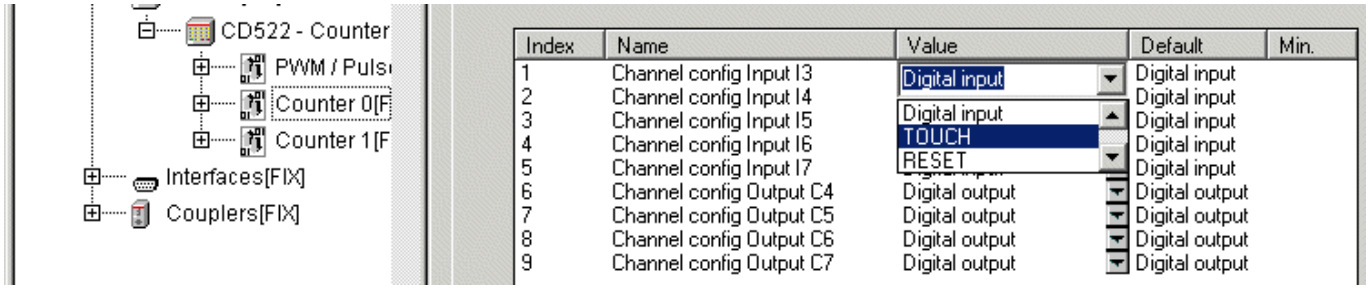
Input EN_CNT corresponds to bit 0 in "control byte".

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input I3 (for counters 0-A and 0-B)) or I11 (for counters 1-A and 1-B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.



The next measurement is again initiated by a rising edge at input EN_TOUCH.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Input EN_TOUCH corresponds to bit 6 in "control byte".

EDGE_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

Input EDGE_TOUCH corresponds to bit 7 in "control byte".

UD

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD, the counting selection is set for up/down counting mode:

UD=FALSE: count up

UD=TRUE: count down

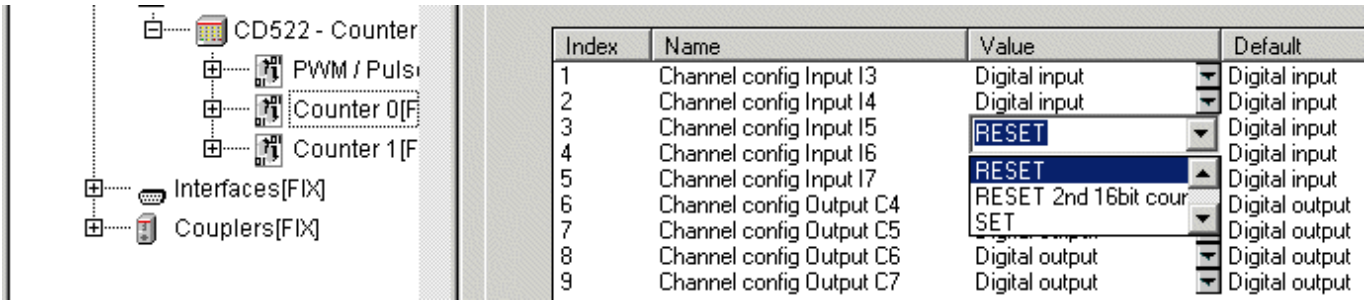
Input UD corresponds to output bit 5 in "control byte".

RESET

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input RESET=TRUE, the counter takes the values 0 to transfer it to output ACT. As long as input RESET=TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input I3, I4, I5, I6 or I7 (for counter 0) or I11, I12, I13, I14 or I15 (for counter 1) causes the function block to reset the value at output ACT.



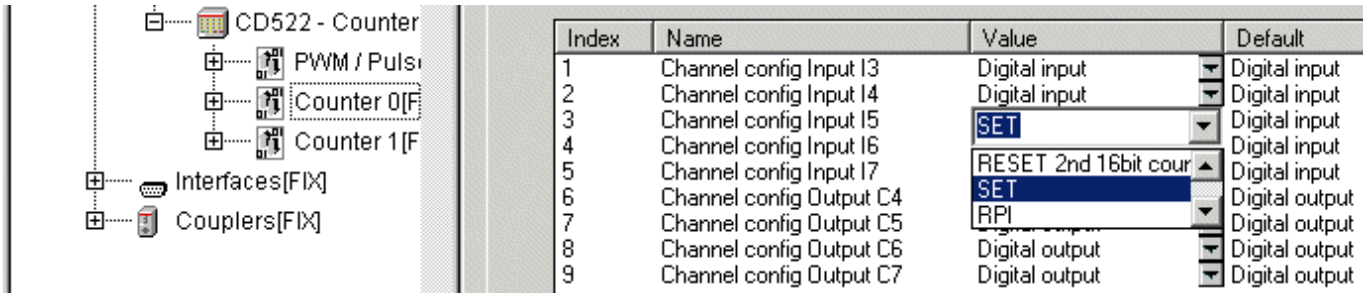
Input EN_RESET corresponds to bit 2 in "control byte".

SET_START

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input I3, I4, I5,I6 or I7 (for counter 0) or I11, I12, I13,I14 or I15 (for counter 1) causes the function block to store the START_VALUE value and to display this value at output ACT.



Input EN_SET corresponds to bit 1 in "control byte".

SET_END

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

START_VALUE

Data type	Default value	Range	Unit
DINT	-	-	-

The counter can be set to a start value. This value must be applied to the input START_VALUE.

If input SET=TRUE, counter takes this value.

Input START_VALUE corresponds to counter settings (DWORD) of CD522.

END_VALUE

Data type	Default value	Range	Unit
DINT	-	-	-

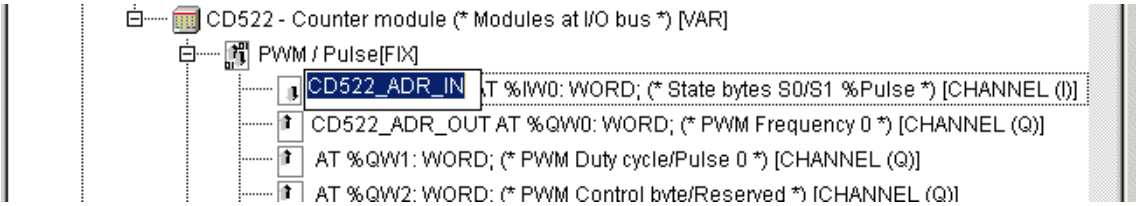
If the counter reaches the planned input END_VALUE, the binary output CF is set to TRUE and the value is stored.

Input END_VALUE corresponds to counter settings (DWORD) of CD522.

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

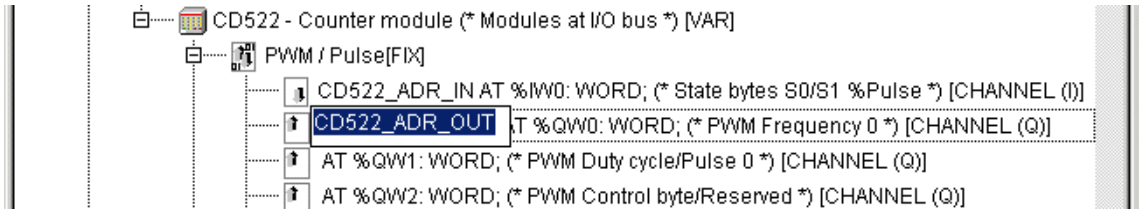
Example (for counter 0):



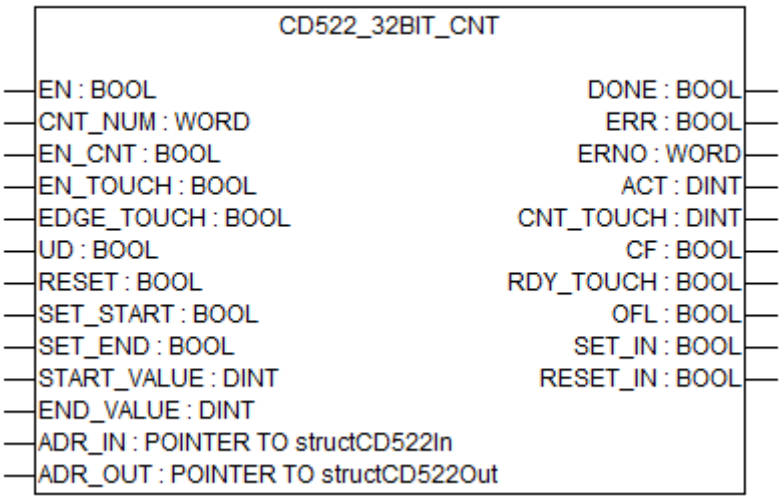
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT

Data type	Default value	Range	Unit
INT	0	-	-

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

Output ACT corresponds to input low word in "32bit counter".

CNT_TOUCH

Data type	Default value	Range	Unit
DINT	0	-	-

The output CNT_TOUCH displays the result of the catch/touch trigger measurement.

Output CNT_TOUCH corresponds to input DWORD in "TOUCH counter value".

CF

Data type	Default value	Range	Unit
BOOL	-	-	-

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

This information is validated when input B is set to TRUE.

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The output RDY_TOUCH is set to TRUE when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Output RDY_TOUCH corresponds to input bit 2 in "state byte".

OFL

Data type	Default value	Range	Unit
BOOL	-	-	-

The overflow is specified at the output OFL.

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648. Any exceeding or falling below of this value (depending to up and down use) will set OFL to TRUE.

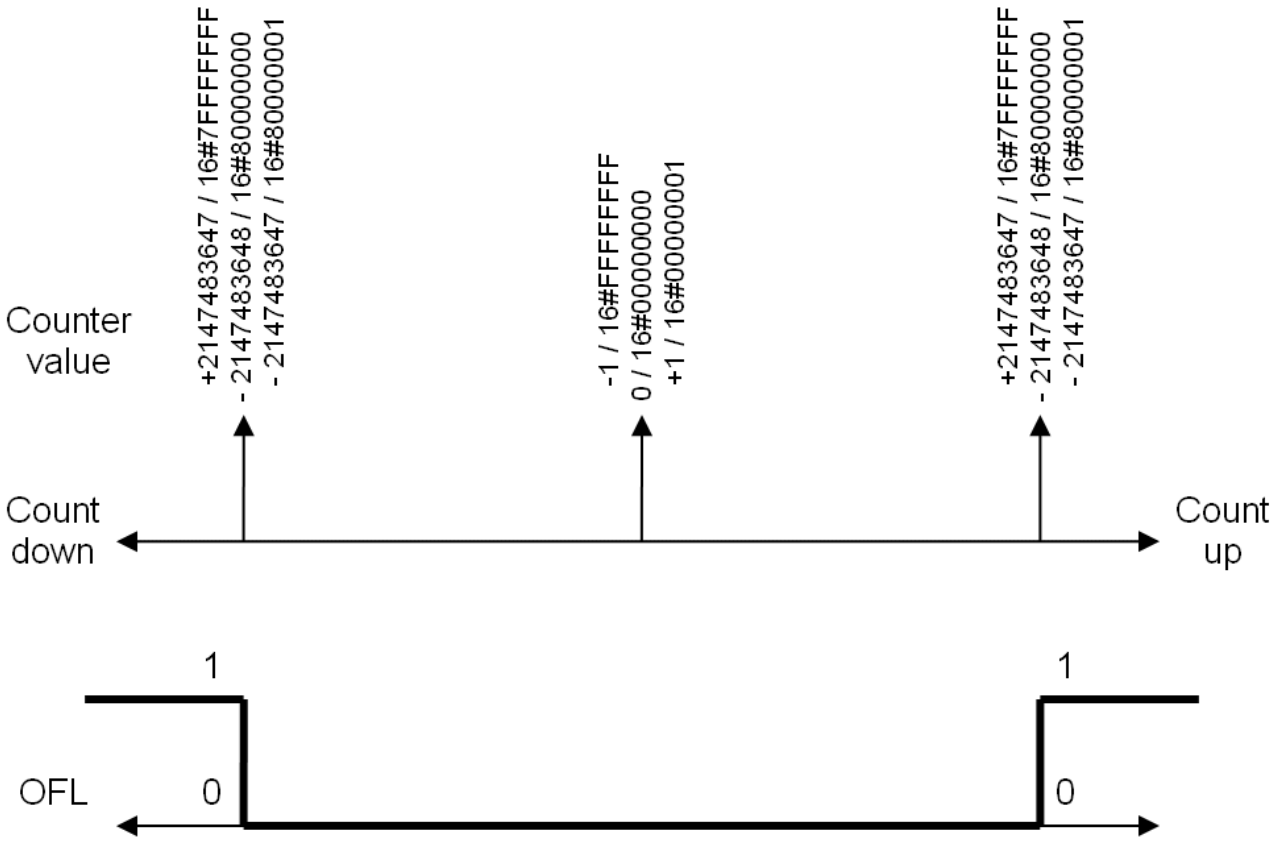


Fig. 20: Detection for output OFL

Output OFL corresponds to input bit 3 in state byte.

SET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output SET_IN is set to TRUE if one of the inputs is configured as SET input.

Output SET_IN corresponds to input bit 4 in "state byte".

RESET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RESET_IN is set to TRUE if one of the inputs is configured as RESET input.

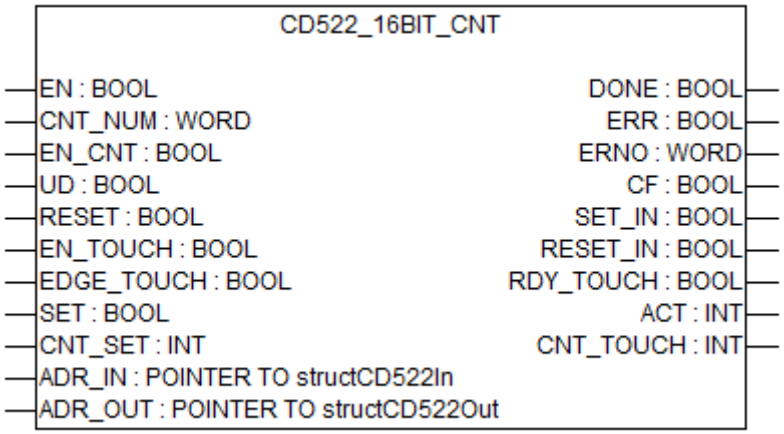
Output RESET_IN corresponds to input bit 5 in "state byte".

Function call in ST

```
CD522_32BITCNT (EN          := CD522_32BITCNT_EN,
                CNT_NUM     := CD522_32BITCNT_CNT_NUM,
                EN_CNT      := CD522_32BITCNT_EN_CNT,
                EN_TOUCH    := CD522_32BITCNT_EN_TOUCH,
                EDGE_TOUCH  := CD522_32BITCNT_EDGE_TOUCH,
                UD          := CD522_32BITCNT_UD,
                RESET       := CD522_32BITCNT_RESET,
                SET_START   := CD522_32BITCNT_SET_START,
                SET_END     := CD522_32BITCNT_SET_END,
```

```
                    START_VALUE := CD522_32BITCNT_START_VALUE,
                    END_VALUE   := CD522_32BITCNT_END_VALUE,
                    ADR_IN      := ADR(CD522_32BITCNT_ADR_IN),
                    ADR_OUT     := ADR(CD522_32BITCNT_ADR_OUT));

CD522_32BITCNT_DONE      :=CD522_32BITCNT.DONE;
CD522_32BITCNT_ERR       :=CD522_32BITCNT.ERR;
CD522_32BITCNT_ERNO      :=CD522_32BITCNT.ERNO;
CD522_32BITCNT_ACT       :=CD522_32BITCNT.ACT;
CD522_32BITCNT_CNT_TOUCH :=CD522_32BITCNT.CNT_TOUCH;
CD522_32BITCNT_CF        :=CD522_32BITCNT.CF;
CD522_32BITCNT_RDY_TOUCH :=CD522_32BITCNT.RDY_TOUCH;
CD522_32BITCNT_OFL       :=CD522_32BITCNT.OFL;
CD522_32BITCNT_SET_IN    :=CD522_32BITCNT.SET_IN;
CD522_32BITCNT_RESET_IN  :=CD522_32BITCNT.RESET_IN;
```

CD522_16BIT_CNT



The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

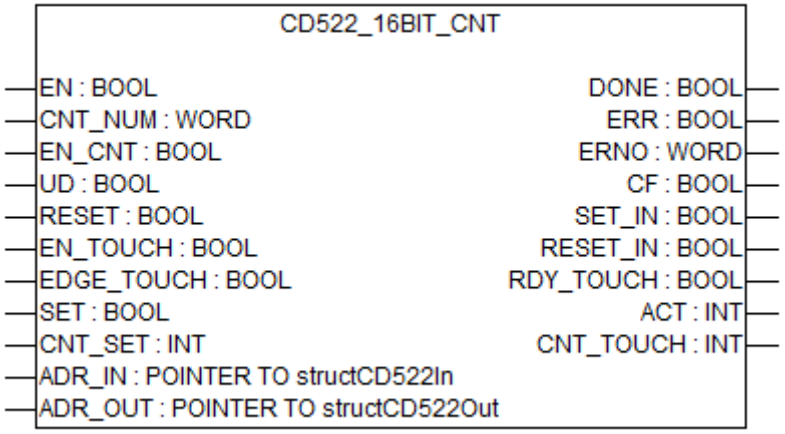
In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_CNT should be used with one of these operating modes:

Operating Mode 8	**"Up/Down with release (B), 0 cross detection"**
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 16 bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	

The module CD522 provides 2 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

Input description



EN_TOUCH

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated integer (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH (see figure below).

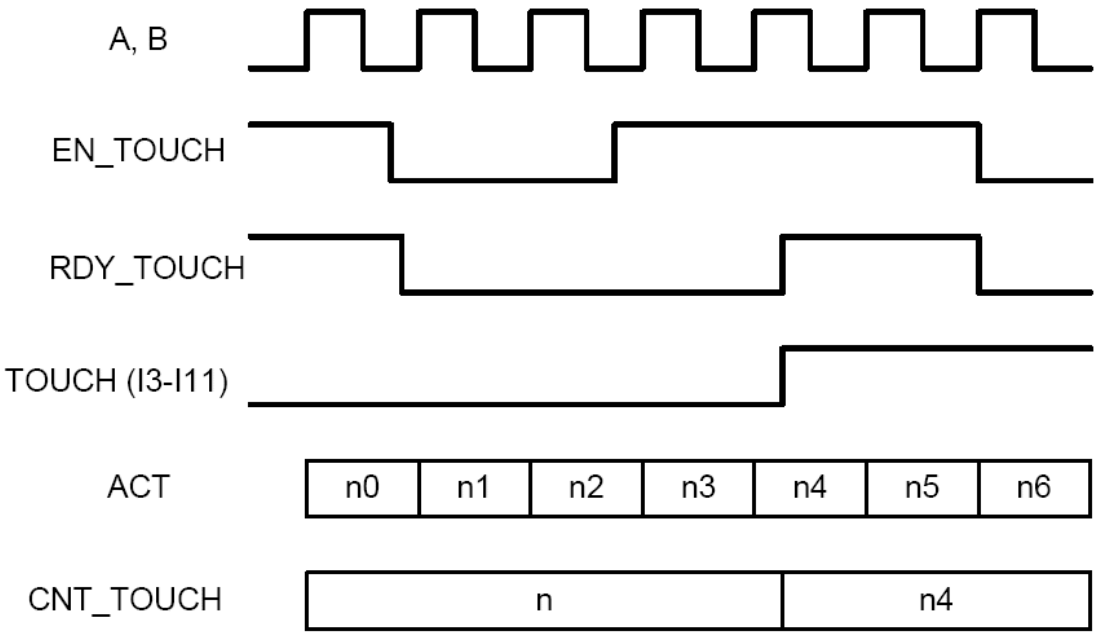


Fig. 21: Procedure and associated counting value with A signal

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_COUNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

Input EN_CNT corresponds to bit 0 in "control byte".

UD

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD, the counting selection is set for up/down counting mode:

UD=FALSE: count up

UD=TRUE: count down

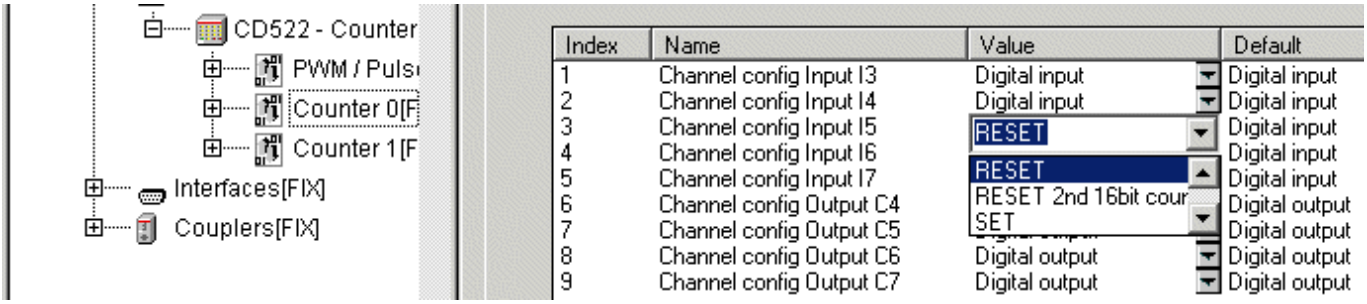
Input UD corresponds to output bit 5 in "control byte".

RESET

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input RESET=TRUE, the counter takes the values 0 to transfer it to output ACT. As long as input RESET=TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input I3, I4, I5, I6 or I7 (for counter 0) or I11, I12, I13, I14 or I15 (for counter 1) causes the function block to reset the value at output ACT.



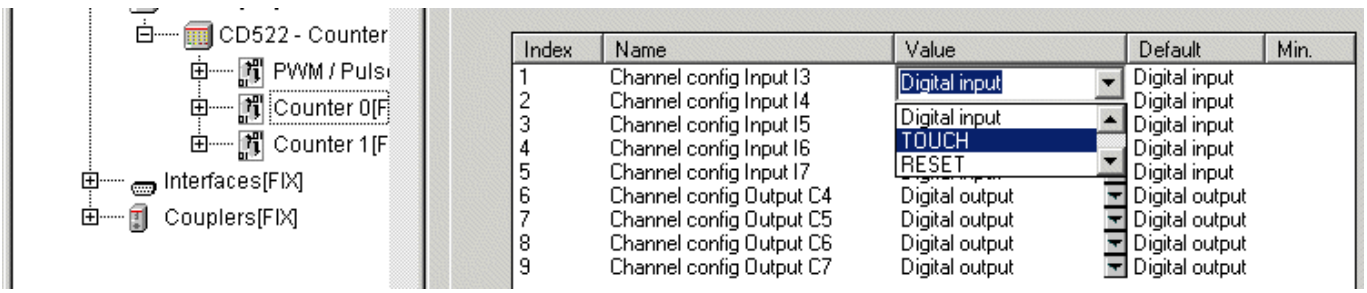
Input EN_RESET corresponds to bit 2 in "control byte".

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input I3 (for counters 0-A and 0-B)) or I11 (for counters 1-A and 1-B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.



The next measurement is again initiated by a rising edge at input EN_TOUCH.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Input EN_TOUCH corresponds to bit 6 in "control byte".

EDGE_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

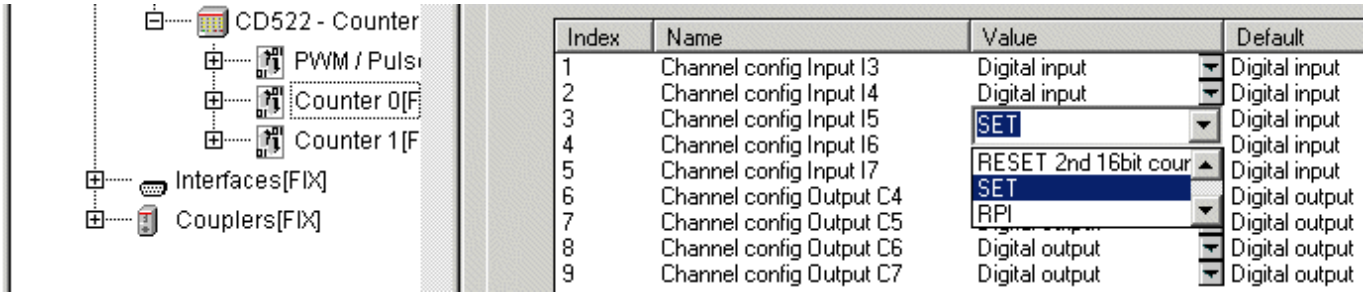
Input EDGE_TOUCH corresponds to bit 7 in "control byte".

SET

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input I3, I4, I5,I6 or I7 (for counter 0) or I11, I12, I13,I14 or I15 (for counter 1) causes the function block to store the START_VALUE value and to display this value at output ACT.



Input EN_SET corresponds to bit 1 in "control byte".

CNT_SET

Data type	Default value	Range	Unit
INT	-	-	-

The counter can be set to a start value. This value must be applied to the input CNT_SET.

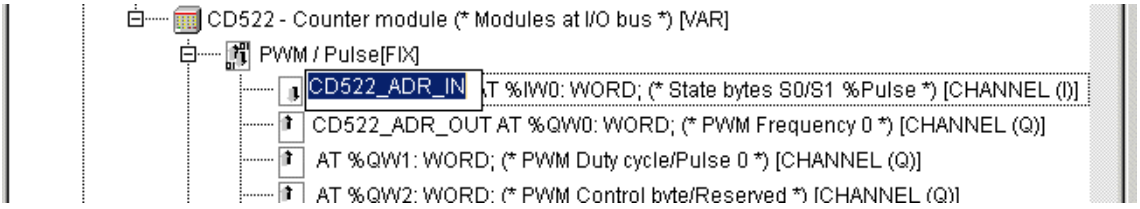
If input SET=TRUE, counter takes this value.

Input START_VALUE corresponds to output low word in "counter settings".

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

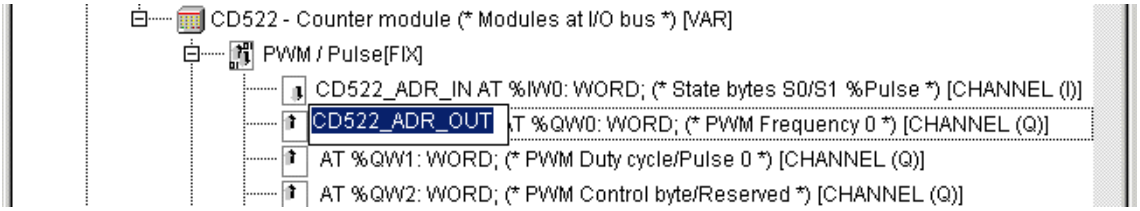
Example (for counter 0):



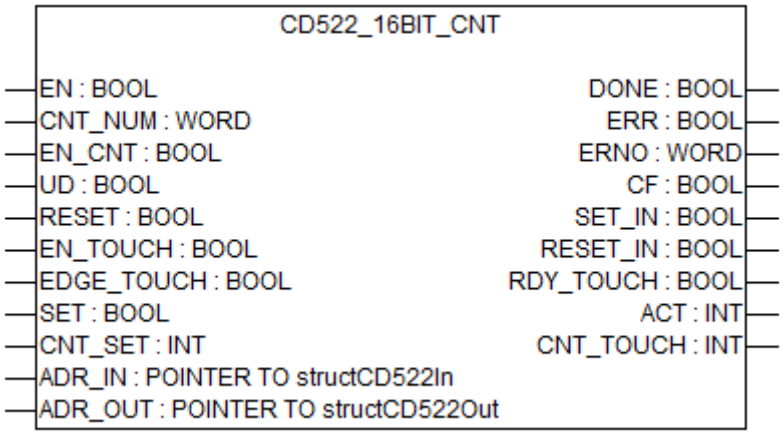
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

CF

Data type	Default value	Range	Unit
BOOL	-	-	-

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

This information is validated when input B is set to TRUE.

SET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output SET_IN is set to TRUE if one of the inputs is configured as SET input.

Output SET_IN corresponds to input bit 4 in "state byte".

RESET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RESET_IN is set to TRUE if one of the inputs is configured as RESET input.

Output RESET_IN corresponds to input bit 5 in "state byte".

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The output RDY_TOUCH is set to TRUE when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Output RDY_TOUCH corresponds to input bit 2 in "state byte".

ACT

Data type	Default value	Range	Unit
INT	0	-	-

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

Output ACT corresponds to input low word in "32bit counter".

CNT_TOUCH

Data type	Default value	Range	Unit
INT	-	-	-

The output CNT_TOUCH displays the result of the catch/touch trigger measurement.

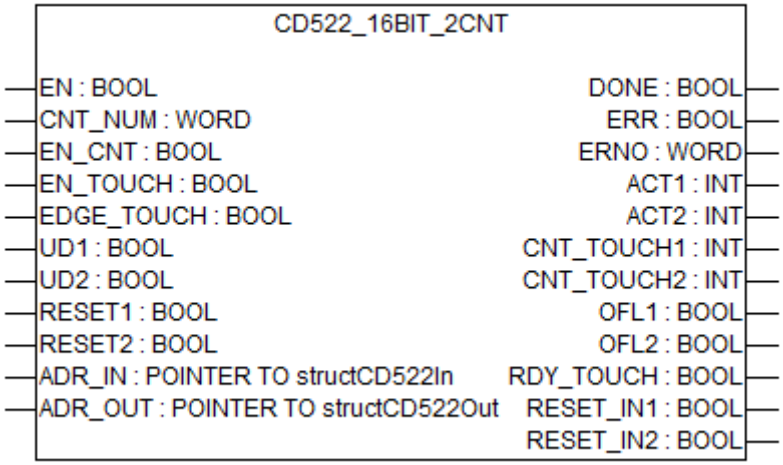
Output CNT_TOUCH corresponds to input low word in "TOUCH counter value".

Function call in ST

```
CD522_16BITCNT (EN:=CD522_16BITCNT_EN,
     CNT_NUM:=CD522_16BITCNT_CNT_NUM,
     EN_CNT:=CD522_16BITCNT_EN_CNT,
     UD:=CD522_16BITCNT_UD,
     RESET:= CD522_16BITCNT_RESET,
     EN_TOUCH:=CD522_16BITCNT_EN_TOUCH,
     EDGE_TOUCH:=CD522_16BITCNT_EDGE_TOUCH,
     SET:= CD522_16BITCNT_SET,
     CNT_SET:= CD522_16BITCNT_CNT_SET,
     ADR_IN:= ADR(CD522_16BITCNT_ADR_IN),
     ADR_OUT:= ADR(CD522_16BITCNT_ADR_OUT));
```

```
CD522_16BITCNT_DONE := CD522_16BITCNT.DONE;
CD522_16BITCNT_ERR := CD522_16BITCNT.ERR;
CD522_16BITCNT_ERNO := CD522_16BITCNT.ERNO;
CD522_16BITCNT_CF := CD522_16BITCNT.CF;
CD522_16BITCNT_SET_IN := CD522_16BITCNT.SET_IN;
CD522_16BITCNT_RESET_IN := CD522_16BITCNT.RESET_IN;
CD522_16BITCNT_RDY_TOUCH := CD522_16BITCNT.RDY_TOUCH;
CD522_16BITCNT_ACT := CD522_16BITCNT.ACT;
CD522_16BITCNT_CNT_TOUCH := CD522_16BITCNT.CNT_TOUCH;
```

CD522_16BIT_2CNT



The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

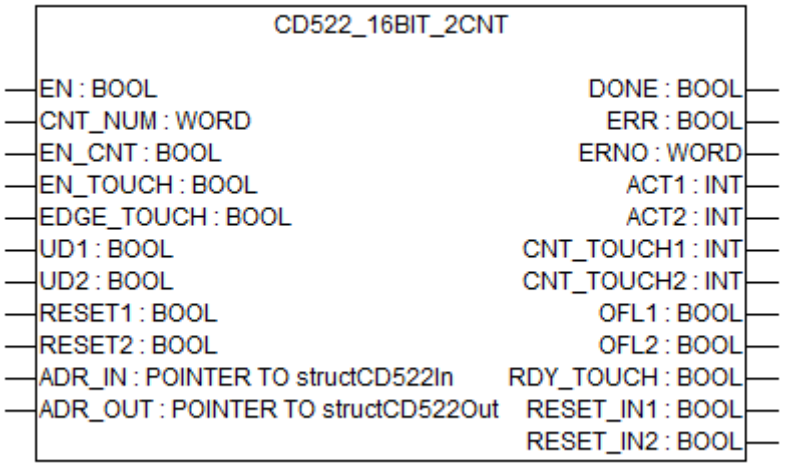
In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_2CNT should be used with one of these operating modes:

Operating Mode 3	"Up/Down counters (A,B)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use 2 Up/Down 16 bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	
Operating Mode 4	"Up/Down (A, B on falling edges)"
Should be specified in PLC Configuration, parameter mode counter in order to use two Up/Down 16 bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and overflow flag.	

The module CD522 provides 4 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 and B0 for counter A and A1 and B1 for counter B.

Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

Input description



EN_TOUCH

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated integer (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH (see figure below).

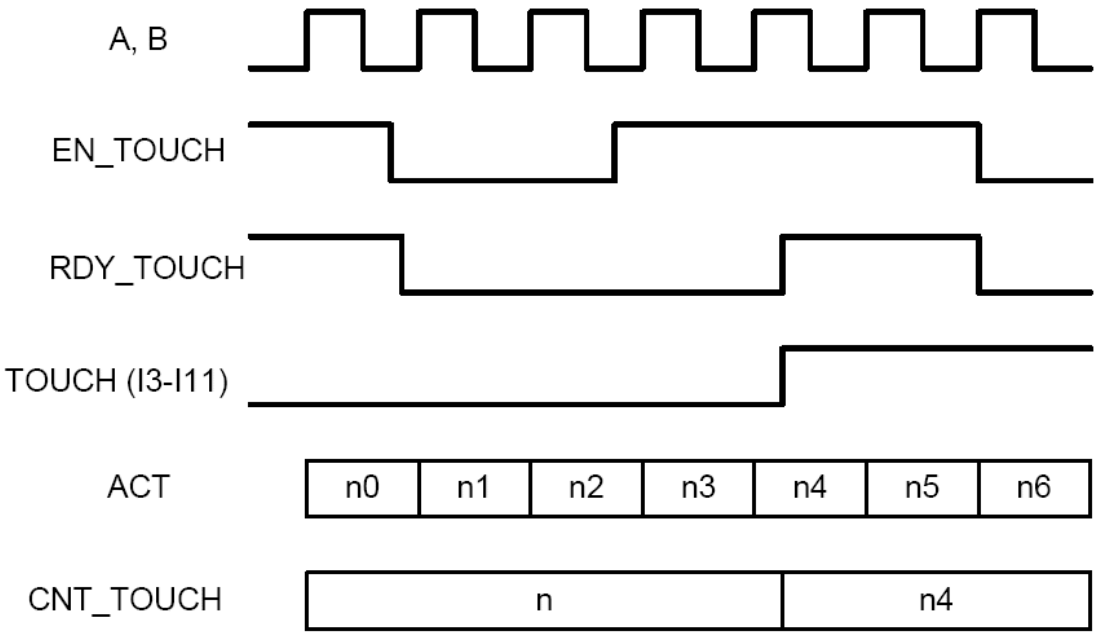


Fig. 22: Procedure and associated counting value with A signal

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_CNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

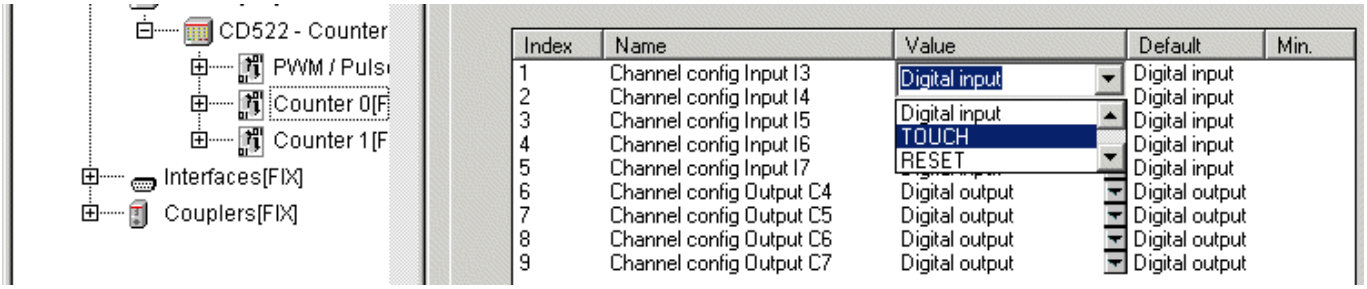
Input EN_CNT corresponds to bit 0 in "control byte".

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input I3 (for counters 0-A and 0-B)) or I11 (for counters 1-A and 1-B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.



The next measurement is again initiated by a rising edge at input EN_TOUCH.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Input EN_TOUCH corresponds to bit 6 in "control byte".

EDGE_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of input I3 (for counters 0-A and 0-B) or I11 (for counters 1-A and 1-B).

Input EDGE_TOUCH corresponds to bit 7 in "control byte".

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting selection is set for up/down counting mode for counter A:

UD1=FALSE: count up

UD1=TRUE: count down

Input UD1 corresponds to output bit 3 in control byte.

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting selection is set for up/down counting mode for counter B:

UD2=FALSE: count up

UD2=TRUE: count down

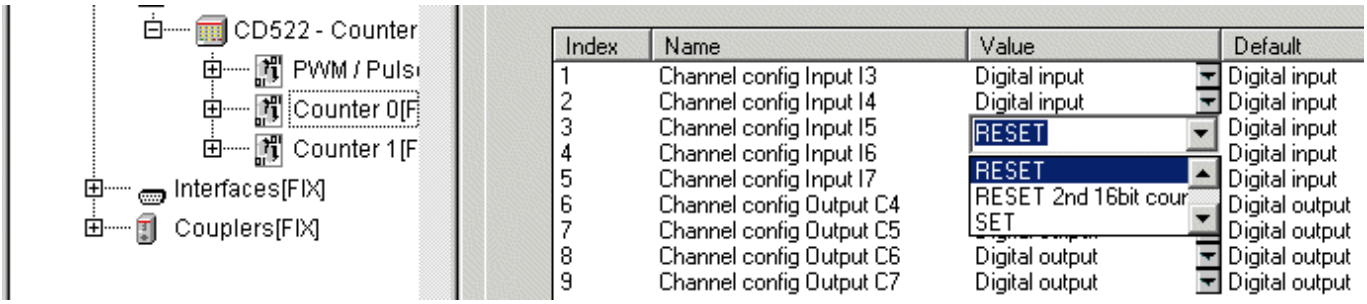
Input UD2 corresponds to output bit 5 in control byte.

RESET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input RESET1=TRUE, the counter A takes the values 0 to transfer it to output ACT1. As long as input RESET1=TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input I3, I4, I5, I6 or I7 (for counter 0-A) or I11, I12, I13, I14 or I15 (for counter 1-A) causes the function block to reset the value at output ACT1.



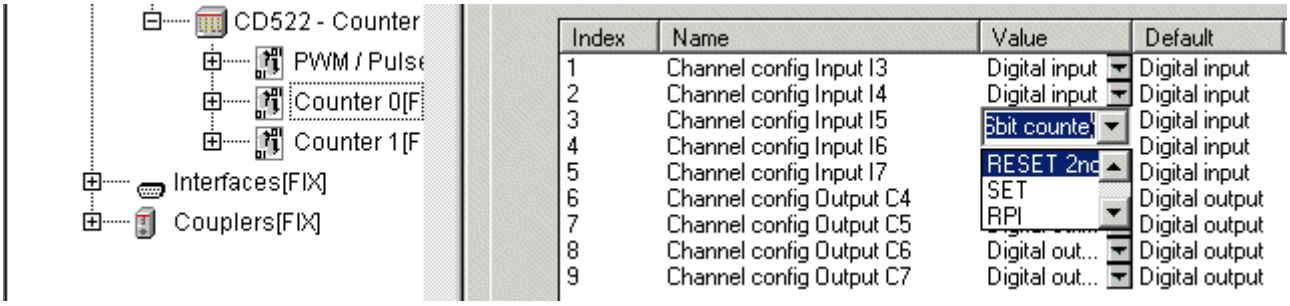
Input EN_RESET1 corresponds to bit 2 in "control byte".

RESET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input RESET2=TRUE, the counter takes B the values 0 to transfer it to output ACT2. As long as input RESET2=TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input I3, I4, I5, I6 or I7 (for counter 0-B) or I11, I12, I13, I14 or I15 (for counter 1-B) causes the function block to reset the value at output ACT2.

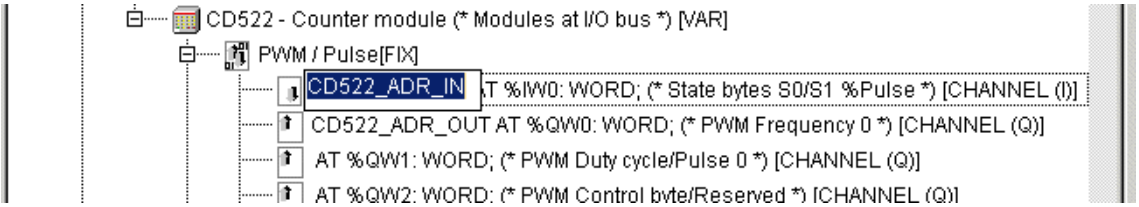


Input EN_RESET2 corresponds to bit 3 in "control byte".

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

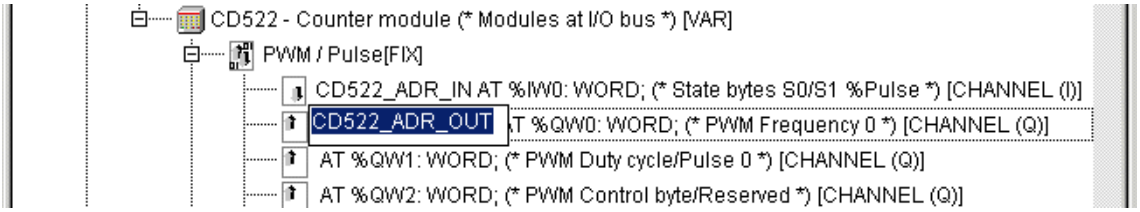
Example (for counter 0):



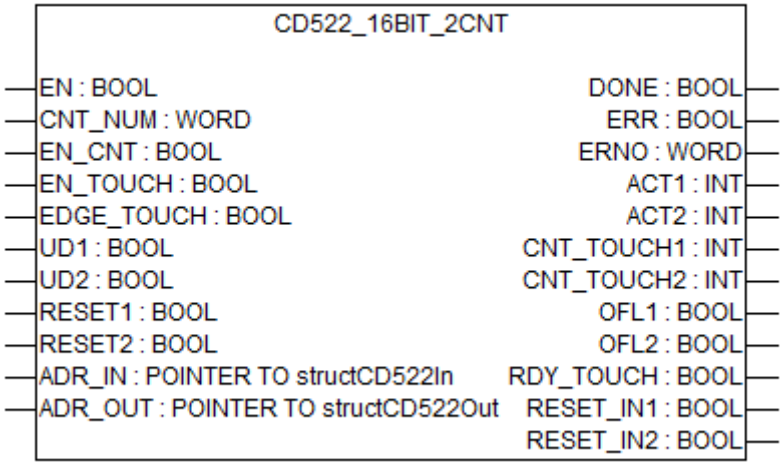
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT1

Data type	Default value	Range	Unit
INT	-	-	-

The current counter value (actual value) from counter A can be retrieved at any time using the output ACT1 of the function block.

Output ACT1 corresponds to input high word in 32bit counter.

ACT2

Data type	Default value	Range	Unit
INT	-	-	-

The current counter value (actual value) from counter B can be retrieved at any time using the output ACT2 of the function block.

Output ACT2 corresponds to input low word in 32bit counter.

CNT_TOUCH1

Data type	Default value	Range	Unit
INT	-	-	-

The output CNT_TOUCH1 displays the result of the catch/touch trigger measurement for counter A.

Output CNT_TOUCH1 corresponds to input high word in TOUCH counter value.

CNT_TOUCH2

Data type	Default value	Range	Unit
INT	-	-	-

The output CNT_TOUCH2 displays the result of the catch/touch trigger measurement for counter B.

Output CNT_TOUCH2 corresponds to input low word in TOUCH counter value.

OFL2

Data type	Default value	Range	Unit
BOOL	-	-	-

The overflow from counter A is specified at the output OFL1.

The counter operates as infinite counter. It is set to TRUE when an overflow occurs, i. e. the counter value ACT1 goes up to value 16#FFFF= -1. Any exceeding or falling below of this value (depending to up use and down use) will set OFL1=TRUE. The output OFL1 is reset when the configuration is changed and if counter value ACT1 is set or reset.

Output OFL1 corresponds to input bit 3 in state byte.

OFL2

Data type	Default value	Range	Unit
BOOL	-	-	-

The overflow from counter B is specified at the output OFL2.

The counter operates as infinite counter. It is set to TRUE when an overflow occurs, i. e. the counter value ACT2 goes up to value 16#FFFF= -1. Any exceeding or falling below of this value (depending to up use and down use) will set OFL2=TRUE. The output OFL2 is reset when the configuration is changed and if counter value ACT2 is set or reset.

Output OFL2 corresponds to input bit 4 in state byte.

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The output RDY_TOUCH is set to TRUE when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Output RDY_TOUCH corresponds to input bit 2 in "state byte".

RESET_IN1

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RESET_IN1 is set to TRUE if one of the inputs is configured as RESET input for counter A.

Output RESET_IN1 corresponds to input bit 5 in "state byte".

RESET_IN2

Data type	Default value	Range	Unit
BOOL	-	-	-

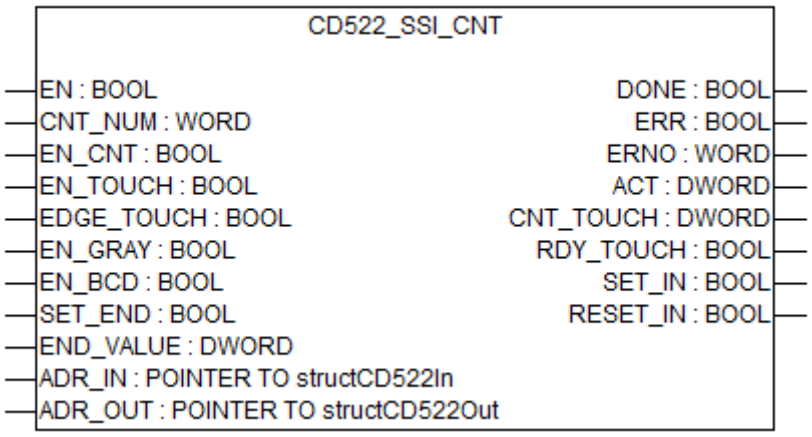
The output RESET_IN2 is set to TRUE if one of the inputs is configured as RESET input for counter B.

Output RESET_IN2 corresponds to input bit 7 in "state byte".

Function call in ST

```
CD522_16BIT2CNT   (EN:=CD522_16BIT2CNT_EN,
         CNT_NUM:= CD522_16BIT2CNT_CNT_NUM,
         EN_CNT:=CD522_16BIT2CNT_EN_CNT,
         EN_TOUCH:=CD522_16BIT2CNT_EN_TOUCH,
         EDGE_TOUCH:=CD522_16BIT2CNT_EDGE_TOUCH,
         UD1:=CD522_16BIT2CNT_UD1,
         UD2:= CD522_16BIT2CNT_UD2,
         RESET1:=CD522_16BIT2CNT_RESET1,
         RESET2:=CD522_16BIT2CNT_RESET_IN1,
         ADR_IN:= ADR(CD522_16BIT2CNT_ADR_IN),
         ADR_OUT:= ADR(CD522_16BIT2CNT_ADR_OUT));


CD522_16BIT2CNT_DONE               := CD522_16BIT2CNT.DONE;
CD522_16BIT2CNT_ERR                := CD522_16BIT2CNT.ERR;
CD522_16BIT2CNT_ERNO               := CD522_16BIT2CNT.ERNO;
CD522_16BIT2CNT_ACT1               := CD522_16BIT2CNT.ACT1;
CD522_16BIT2CNT_ACT2               := CD522_16BIT2CNT.ACT2;
CD522_16BIT2CNT_CNT_TOUCH1         := CD522_16BIT2CNT.CNT_TOUCH1;
CD522_16BIT2CNT_CNT_TOUCH2         := CD522_16BIT2CNT.CNT_TOUCH2;
CD522_16BIT2CNT_OFL1               := CD522_16BIT2CNT.OFL1;
CD522_16BIT2CNT_OFL2               := CD522_16BIT2CNT.OFL2;
CD522_16BIT2CNT_ RDY_TOUCH         := CD522_16BIT2CNT.RDY_TOUCH;
CD522_16BIT2CNT_RESET_IN1          := CD522_16BIT2CNT.RESET_IN1;
CD522_16BIT2CNT_RESET_IN2          := CD522_16BIT2CNT.RESET_IN2;
```

CD522_SSI_CNT



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Available from CD522 firmware	V2.1
Type	Function block with historical values

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_SSI_CNT should be used with one of these operating modes:

Operating Mode 14	"SSI, absolute encoder"
Should be specified in PLC Configuration; parameter mode counter in order to use absolute encoder with SSI interface.	

The module CD522 provides 2 SSI absolute encoder functions. There is an interface for absolute angle and linear encoders (displacement measurement systems).

It allows the transmission of absolute position information through a serial data transfer.

The transmission is based on synchronous serial communication. The device sends a clock signal to the encoder and synchronously,the encoder returns the positioning data from the most significant to the less significant bit.

The synchronization for a new data stream is based on time without clock pulse. This quiet time depends on the encoder.

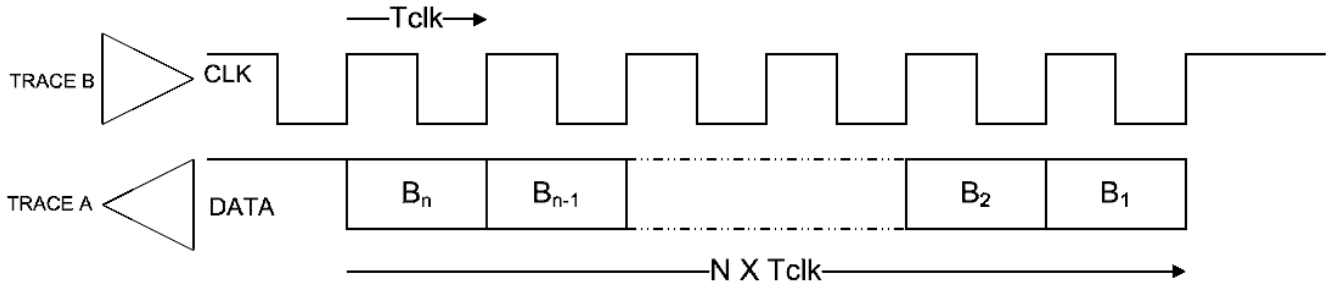
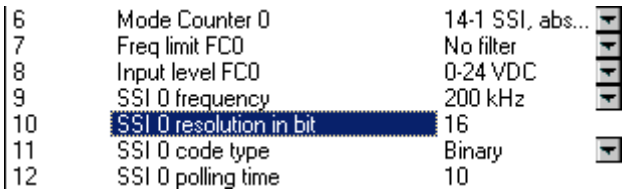


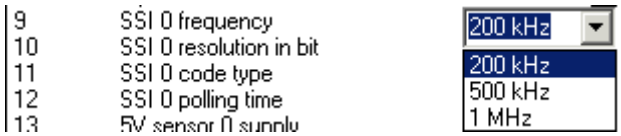
Fig. 23: Chronogram with data organization with the clock pulse

The resolution of the encoder device (see technical data from manufacturer) can be set with configuration, number of bits, etc. in PLC configuration, parameter SSI resolution in bit:



The trace B of module CD522 is switched as output signal (differential). On the rising edge of the signal, the sensor shifts a new value, starting from the most significant bit.

The clock frequency can be set to 200 kHz, 500 kHz, and 1 MHz Should be specified in PLC Configuration, parameter SSI frequency:



NOTICE!

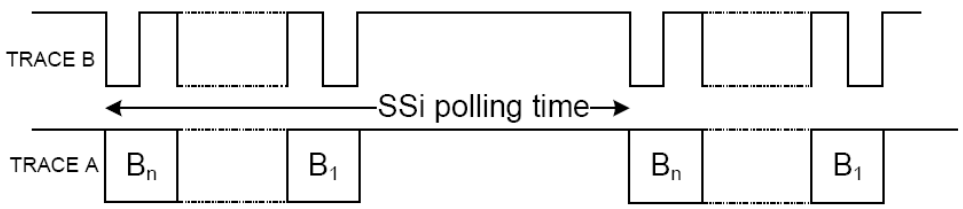
Risk of malfunctions!

The clock frequency is only an approximately value.

Do not use the clock frequency for any other purposes, e.g. time measurements.

SSI polling time definition

The complete read sequence is launched regularly by the module CD522. The interval between each sequence can be set from 1 ms to 255 ms in PLC Configuration, parameter SSI polling time:

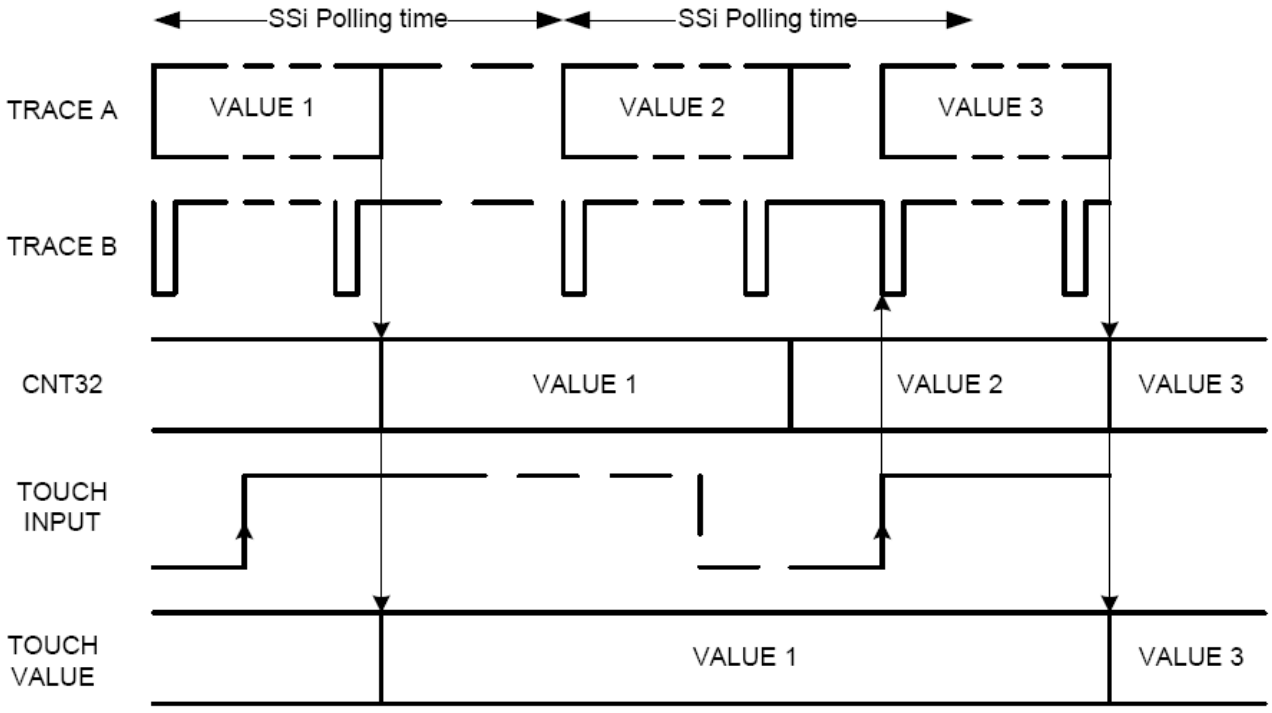


SSI and touch/ catch operation

Touch operation is valid with SSI sensor. The goal of touch operation is to synchronize sensors with the same hardware signal. In the SSI mode the management is different depending on the reading procedure is running or not.

If the reading procedure has already started while the touch signal becomes active, the reading procedure finishes normally and the last read value is stored in the touch register.

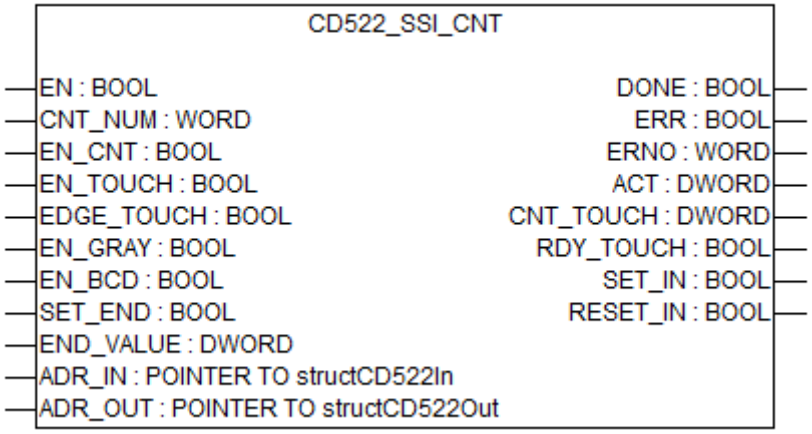
If the reading procedure has not started, the encoder is in the interval between 2 measurements. The reading procedure is started one time more and the result of the last reading is stored in the touch register.



SSI and Touch/Catch Operation: see EN_TOUCH *"EN_TOUCH" on page 1009*.

Overflow Operation: see OFL *"OFL" on page 989*.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_CNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

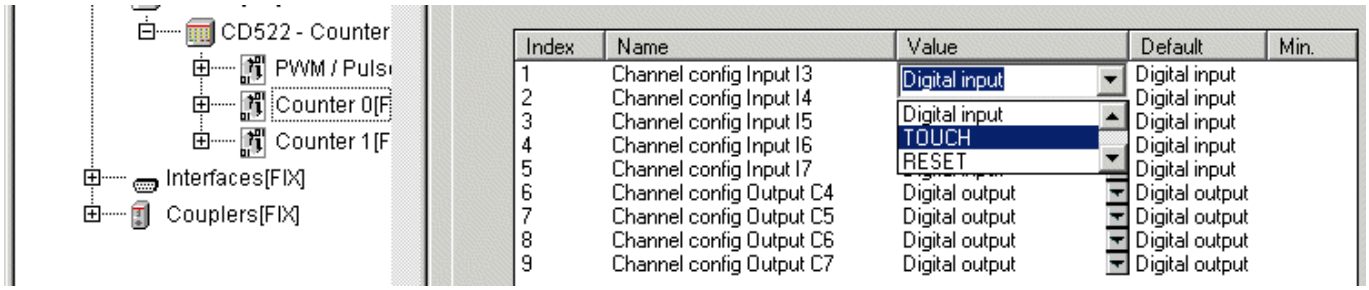
Input EN_CNT corresponds to bit 0 in "control byte".

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (NTOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status NTOUCH is set to TRUE.

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input I3 (for counters 0-A and 0-B)) or I11 (for counters 1-A and 1-B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.



The next measurement is again initiated by a rising edge at input EN_TOUCH.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Input EN_TOUCH corresponds to bit 6 in "control byte".

EDGE_TOUCH

Data type	Default value	Range	Unit
BOOL	-	-	-

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of input I3 (for counter 0) or I11 (for counter 1).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of input I3 (for counter 0) or I11 (for counter 1).

Input EDGE_TOUCH corresponds to bit 7 in "control byte".

EN_GRAY

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_GRAY=TRUE, the actual value will be calculated from GRAY code.

EN_BCD

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_BCD=TRUE, the actual value will be calculated from BCD code.

SET_END

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

END_VALUE

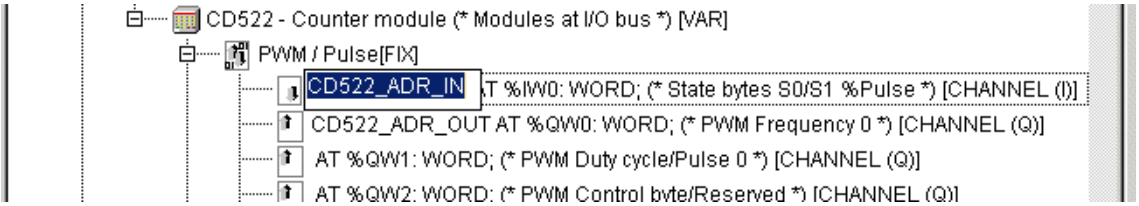
Data type	Default value	Range	Unit
DWORD	-	-	-

If the counter reaches the planned input END_VALUE (read while SET_END is set to TRUE), the outputs configured as ENDV is set to TRUE.

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

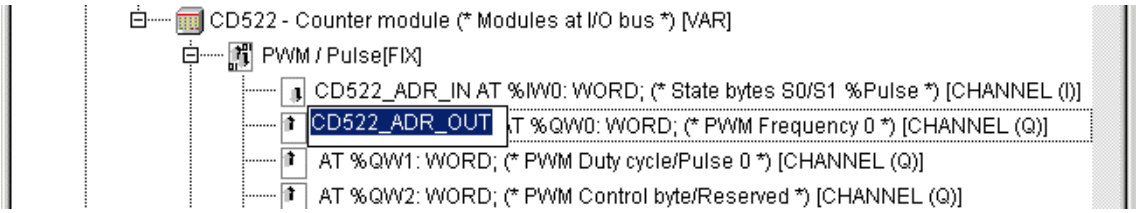
Example (for counter 0):



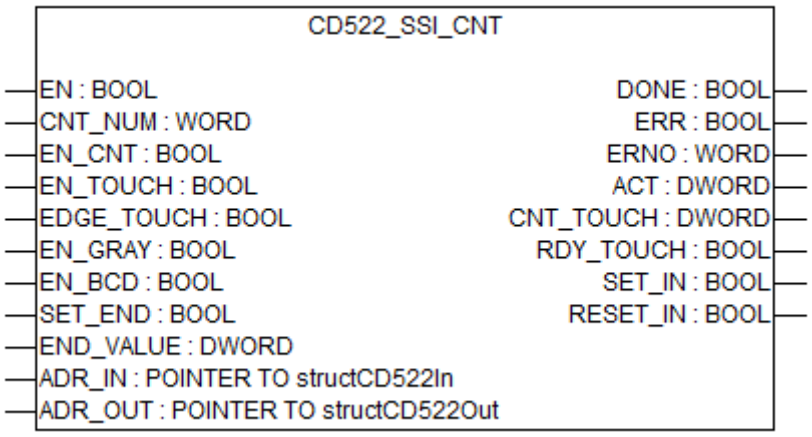
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT

Data type	Default value	Range	Unit
INT	0	-	-

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

Output ACT corresponds to input low word in "32bit counter".

CNT_TOUCH

Data type	Default value	Range	Unit
DWORD	-	-	-

The output CNT_TOUCH displays the result of the catch/touch trigger measurement.

Output CNT_TOUCH corresponds to input DWORD in "TOUCH counter value".

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The output RDY_TOUCH is set to TRUE when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

Output RDY_TOUCH corresponds to input bit 2 in "state byte".

SET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

The output SET_IN is set to TRUE if one of the inputs is configured as SET input.

Output SET_IN corresponds to input bit 4 in "state byte".

RESET_IN

Data type	Default value	Range	Unit
BOOL	-	-	-

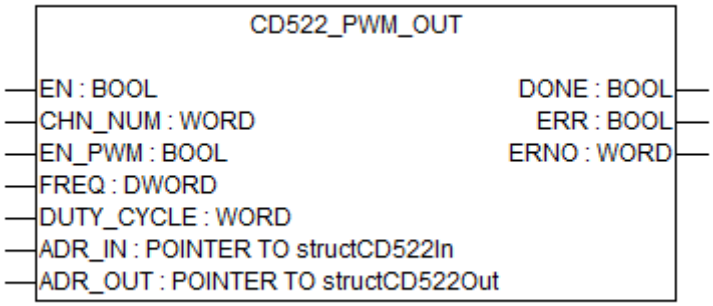
The output RESET_IN is set to TRUE if one of the inputs is configured as RESET input.

Output RESET_IN corresponds to input bit 5 in "state byte".

Function call in ST

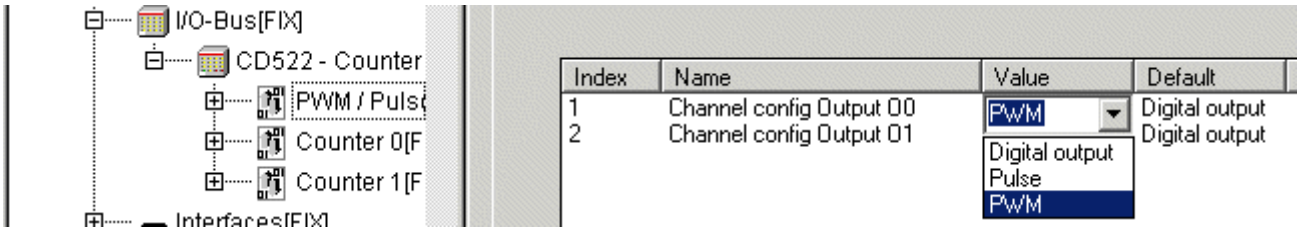
```
CD522SSICNT (EN          := CD522CNTSSI_EN,
             CNT_NUM     := CD522SSICNT_CNT_NUM,
             EN_CNT      := CD522COUNTSSI_EN_CNT,
             EN_TOUCH    := CD522CNTSSI_EN_TOUCH,
             EDGE_TOUCH  := CD522CNTSSI_EDGE_TOUCH,
             EN_GRAY     := CD522CNTSSI_EN_GRAY,
             EN_BCD      := CD522CNTSSI_EN_BCD,
             SET_END     := CD522CNTSSI_SET_END,
             END_VALUE   := CD522CNTSSI_END_VALUE,
             ADR_IN      := ADR(CD522CNTSSI_ADR_IN),
             ADR_OUT     := ADR(CD522CNTSSI_ADR_OUT));


CD522SSICNT_DONE         := CD522SSICNT.DONE;
CD522SSICNT_ERR          := CD522SSICNT.ERR;
CD522SSICNT_ERNO         := CD522SSICNT.ERNO;
CD522SSICNT_ACT          := CD522SSICNT.ACT;
CD522SSICNT_CNT_TOUCH    := CD522SSICNT.CNT_TOUCH;
CD522SSICNT_RDY_TOUCH    := CD522SSICNT.RDY_TOUCH;
CD522SSICNT_SET_IN       := CD522SSICNT.SET_IN;
CD522SSICNT_RESET_IN     := CD522SSICNT.RESET_IN;
```

CD522_PWM_OUT



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Available from CD522 firmware	V2.1
Type	Function block with historical values

The module CD522 can be used to control one output pulsing signal (Max= 100 KHz) with an adjustable duty cycle (ON/OFF ratio, max=100%). The PWM operating mode is configured in Automation Builder.



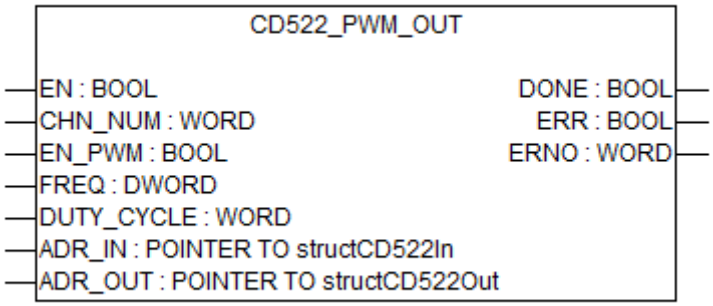
After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_PWM_OUT should be used to control with input EN_PWM, configure the frequency with input FREQ and the input duty cycle DUTY_CYCLE of PWM outputs (pulse-width modulator).

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CHN_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CHN_NUM contains the channel number managed by the function block:

- CHN_NUM = 0: the output O0 is managed
- CHN_NUM = 1: the output O1 is managed

EN_PWM

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_PWM = TRUE, the pulse-width modulator is enabled. If EN_PWM = FALSE, no pulse-width modulation is performed.

Input EN_PWM corresponds to output bit 7 in "control byte".

FREQ

Data type	Default value	Range	Unit
DWORD	-	1 ... 100	KHz

Input FREQ is used to specify the frequency of output. The frequency can be set from 1 to 100 KHz.

The duty cycle is fixed (50 %).

The frequency value is defined with a double word (DWORD), but the internal communication with CD522 is realized with WORD type. For frequencies greater than 65535 Hz (word limitation) an additional bit is internally used as multiplier. Thus, for such frequencies the resolution becomes 10 Hz.

The additional multiplier bit is the bit 0 of "control byte". If bit 0 of "control byte"=TRUE, the frequency multiplier x10 is enabled.

DUTY_CYCLE

Data type	Default value	Range	Unit
WORD	-	1 ... 100	KHz

The input DUTY_CYCLE is used to specify the percentage of time set to TRUE. The duty cycle is from 0.0 to 100.0 (0 % to 100 %); the value is written without dot. That means, for example, for 75.8 % the value written will be 758. The max. value authorized will be 1000 to specify a duty cycle = 100 %.

On fast outputs O0 and O1, the brightness of the yellow LED depends on the value of duty cycle specified (from 0 to 100 %).

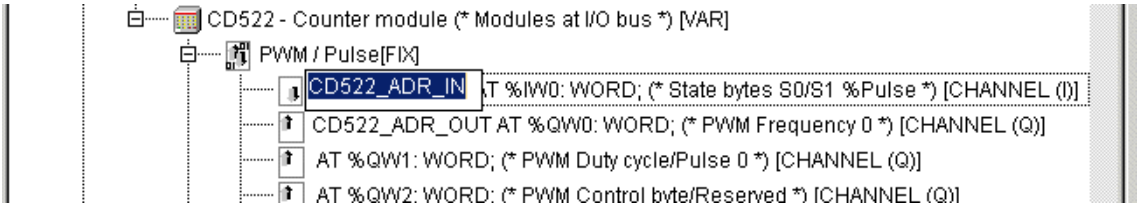
If the written value is greater than 1000 or less than 0, an error code will be displayed in ERR and ERNO outputs and the red error LED CH-ERR1 will flash. The last entered valid value will be used or the value '0' if no value has been entered before.

Input DUTY_CYCLE corresponds to output word "PWM Duty cycle/pulse".

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

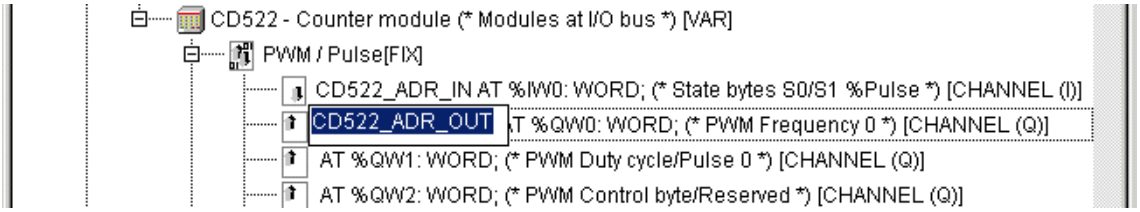
Example (for counter 0):



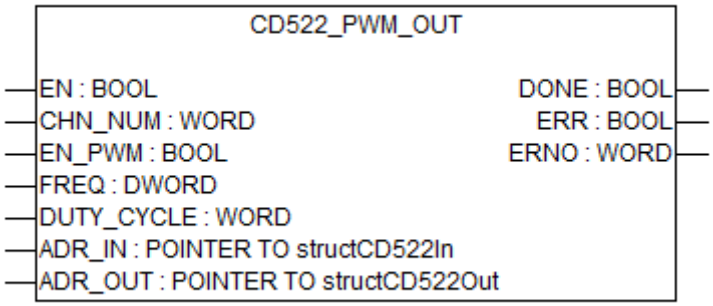
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

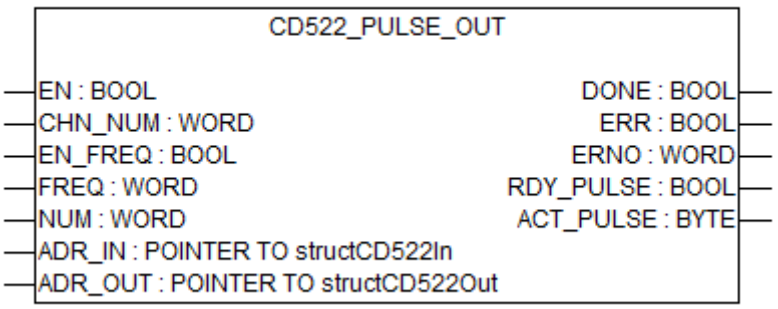
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

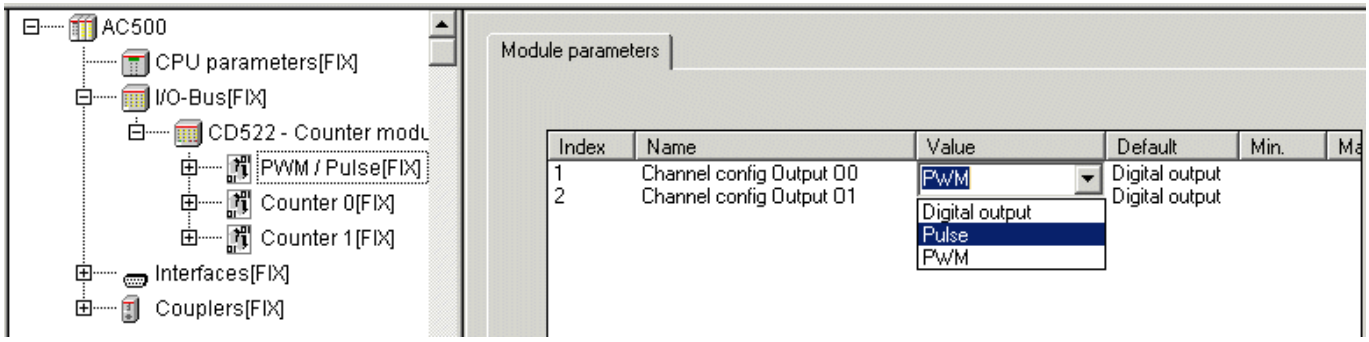
```
CD522PWMOUT    (EN          := CD522PWMOUT_EN,
                CHN_NUM     := CD522PWMOUT_CHN_NUM,
                FREQ        := CD522PWMOUT_FREQ,
                DUTY_CYCLE  := CD522PWMOUT_DUTY_CYCLE,
                ADR_IN      := ADR(CD522PWMOUT_ADR_IN),
                ADR_OUT     := ADR(CD522PWMOUT_ADR_OUT));


CD522PWMOUT_DONE := CD522PWMOUT.DONE;
CD522PWMOUT_ERR  := CD522PWMOUT.ERR;
CD522PWMOUT_ERNO := CD522PWMOUT.ERNO;
```

CD522_PULSE_OUT



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Available from CD522 firmware	V2.1
Type	Function block with historical values

The module CD522 can be used to control one output pulses signals with a fixed duty cycle (ON/OFF ratio 50 %) and number of pulses sent with a fixed frequency (can be modified) .The PULSE operating mode is configured in PLC Configuration using module parameters:



After that, it is activated during the initialization phase (power-on, cold start, warm start).

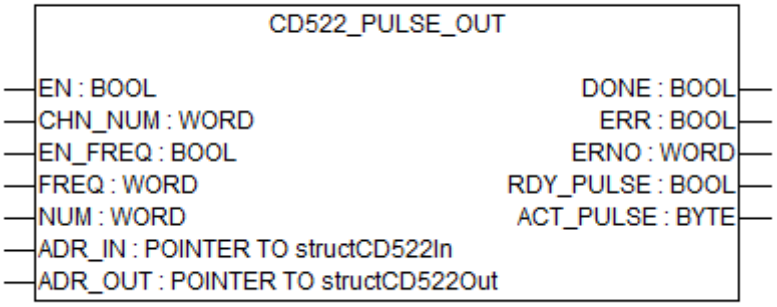
The data exchange from and to the user program is performed by using input and output operands. These necessary operands are created and reserved automatically, when one CD522 module is added into the I/O bus configuration.

The module CD522 provides two independents outputs used in PULSE mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_PULSE_OUT should be used to control the pulse output, with input EN_FREQ, configure the frequency with input FREQ and the number of pulses with input NUM. The number of pulses sent can be displayed in percentage (from 0 % to 100%).

On the fast outputs O0 or O1, the brightness of yellow LED depends on the number of pulse emitted (from 0 and 100%), When the value 100% is obtained, the yellow LED status is off.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CHN_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CHN_NUM contains the channel number managed by the function block:

- CHN_NUM = 0: the output O0 is managed
- CHN_NUM = 1: the output O1 is managed

EN_FREQ

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_FREQ=TRUE, the frequency output is enabled. If EN_FREQ = FALSE, no frequency output is performed.

Input EN_FREQ corresponds to output bit 7 in "control byte".

FREQ

Data type	Default value	Range	Unit
WORD	-	0 ... 15	Khz

The input FREQ is used to specify the frequency of output pulse.

The input FREQ corresponds to output word "PWM Frequency".

NUM

Data type	Default value	Range	Unit
WORD	-	-	-

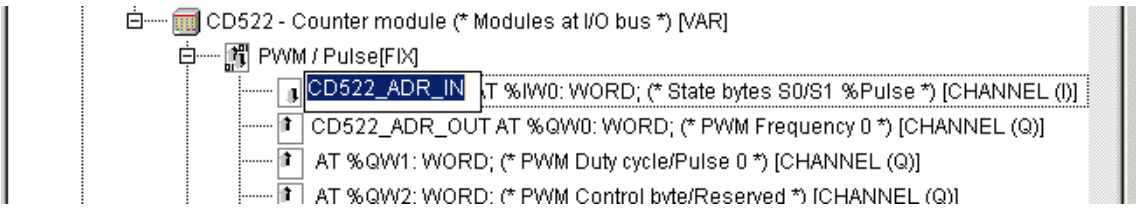
The input NUM is used to specify the number of pulses to be sent. The number is from 1 to 65535 pulses.

The input NUM corresponds to output word "PWM Frequency".

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

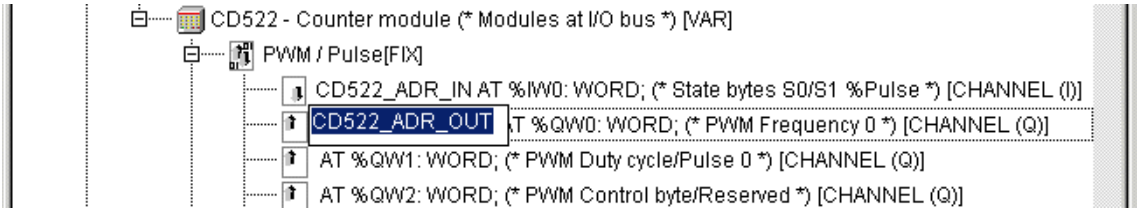
Example (for counter 0):



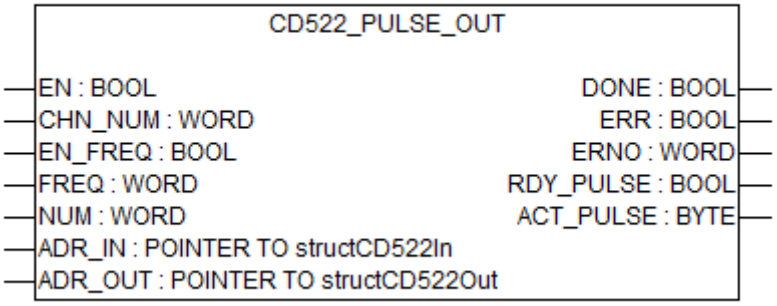
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY_PULSE

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RDY_PULSE is set to TRUE when all pulses have been sent.

ACT_PULSE

Data type	Default value	Range	Unit
BYTE	-	-	%

The output ACT_PULSE return the percentage of pulses sent, from 0 to 100 %.

100 % is obtained only when the totality of pulses has been sent.

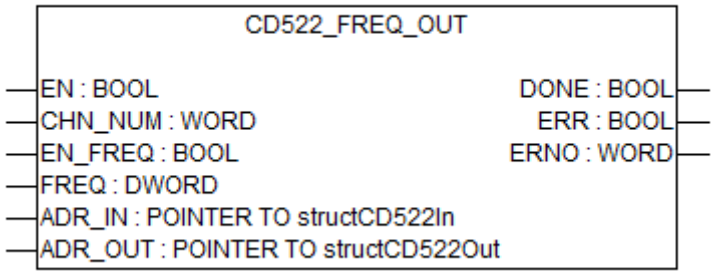
On the fast outputs O0 or O1, the brightness of the yellow LED depends on the number of pulse emitted (from 0 to 100 %), When the value 100 % is obtained, the yellow LED status is off.

Output ACT_PULSE corresponds to input byte in "state byte Sx %pulse".

Function call in ST

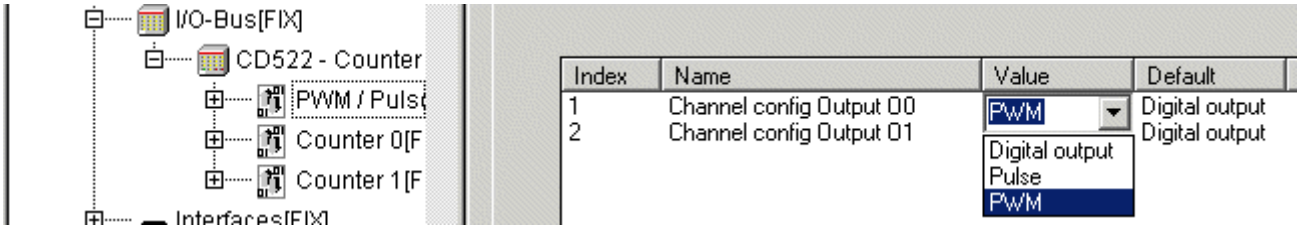
```
CD522PULSEOUT     (EN      := CD522PULSEOUT_EN,
                   CHN_NUM := CD522PULSEOUT_CHN_NUM,
                   EN_FREQ := CD522PULSEOUT_EN_FREQ,
                   FREQ    := CD522PULSEOUT_FREQ,
                   NUM     := CD522PULSEOUT_NUM,
                   ADR_IN  := ADR(CD522PULSEOUT_ADR_IN),
                   ADR_OUT := ADR(CD522PULSEOUT_ADR_OUT));


CD522PULSEOUT_DONE      := CD522PULSEOUT.DONE;
CD522PULSEOUT_ERR       := CD522PULSEOUT.ERR;
CD522PULSEOUT_ERNO      := CD522PULSEOUT.ERNO;
CD522PULSEOUT_RDY_PULSE := CD522PULSEOUT.RDY_PULSE;
CD522PULSEOUT_ACT_PULSE := CD522PULSEOUT.ACT_PULSE;
```

CD522_FREQ_OUT



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

The module CD522 can be used to control one output pulses signals with an fixed duty cycle (ON/OFF ratio 50 %).The PWM operating mode is configured in PLC Configuration using module parameters:



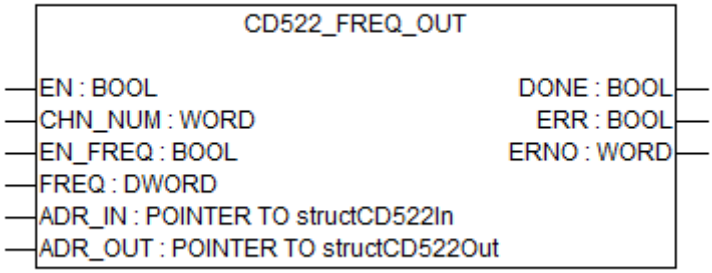
After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_FREQ_OUT should be used to control with input EN_FREQ and configure the frequency with input FREQ of frequency outputs (1 kHz to 100 kHz).

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CHN_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CHN_NUM contains the channel number managed by the function block:

- CHN_NUM = 0: the output O0 is managed
- CHN_NUM = 1: the output O1 is managed

EN_FREQ

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_FREQ=TRUE, the frequency output is enabled. If EN_FREQ = FALSE, no frequency output is performed.

Input EN_FREQ corresponds to output bit 7 in "control byte".

FREQ

Data type	Default value	Range	Unit
DWORD	-	1 ... 100	KHz

Input FREQ is used to specify the frequency of output. The frequency can be set from 1 to 100 KHz.

The duty cycle is fixed (50 %).

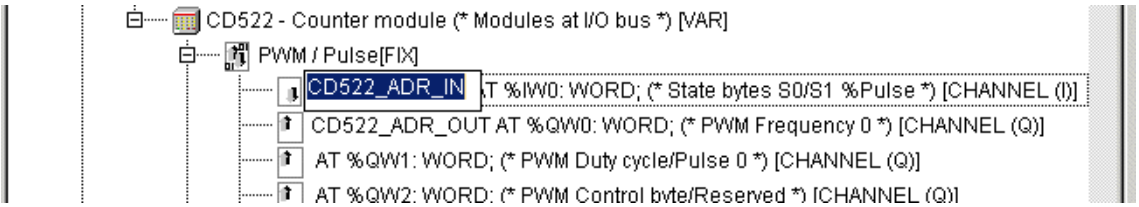
The frequency value is defined with a double word (DWORD), but the internal communication with CD522 is realized with WORD type. For frequencies greater than 65535 Hz (word limitation) an additional bit is internally used as multiplier. Thus, for such frequencies the resolution becomes 10 Hz.

The additional multiplier bit is the bit 0 of "control byte". If bit 0 of "control byte"=TRUE, the frequency multiplier x10 is enabled.

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

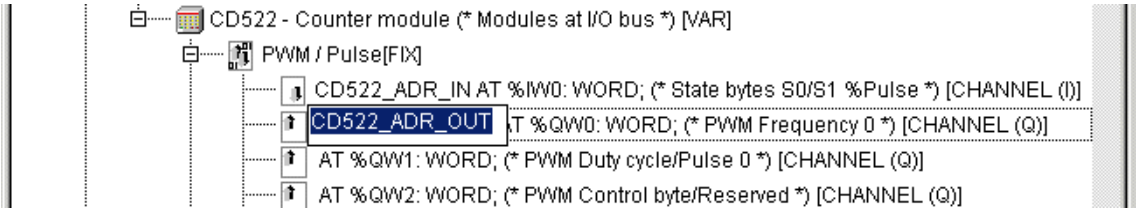
Example (for counter 0):



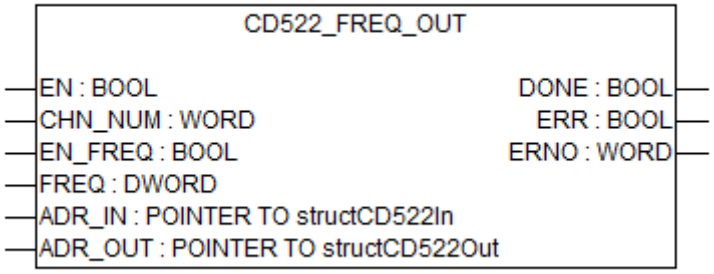
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

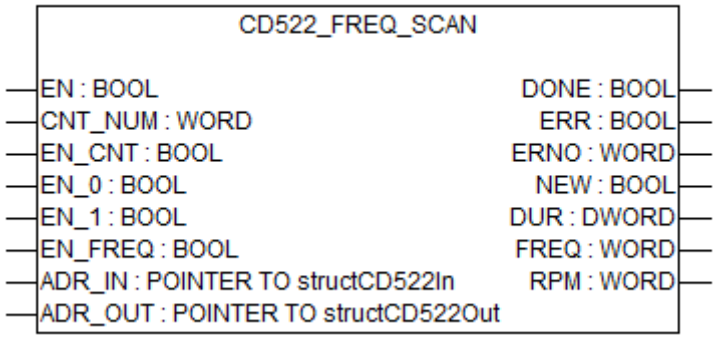
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
CD522FREQOUT        (EN:= CD522FREQOUT_EN,
     CHN_NUM:= CD522FREQOUT_CHN_NUM,
     EN_FREQ:= CD522FREQOUT_EN_FREQ,
     FREQ:= CD522FREQOUT_FREQ,
     ADR_IN:= ADR(CD522FREQOUT_ADR_IN),
     ADR_OUT:= ADR(CD522FREQOUT_ADR_OUT));


CD522FREQOUT_DONE := CD522FREQOUT.DONE;
CD522FREQOUT_ERR := CD522FREQOUT.ERR;
CD522FREQOUT_ERNO := CD522FREQOUT.ERNO;
```

CD522_FREQ_SCAN



Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Available from CD522 firmware	V2.1
Type	Function block with historical values

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_FREQ_SCAN should be used with one of these operating modes:

Operating Mode 15	"Time frequency meter (Z)"
Should be specified in PLC Configuration with the parameter mode counter.	

The module CD522 provides 2 channels (Z0 and Z1) which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs. Both have the same specification and can work separately.

The function block CD522_FREQ_SCAN should be used to control with input EN_CNT, configure the capture on falling edge with input EN_0 or rising edge with input EN_1 of signal, and the specification of the mode of the measurement (time, frequency and Rpm) with input EN_FREQ.

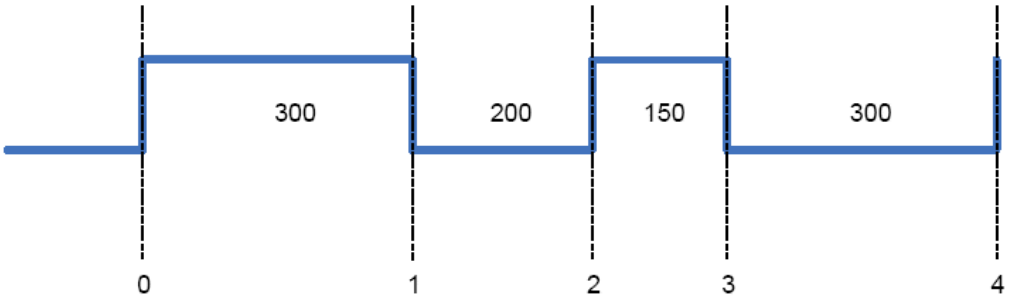
The table shows values measured according to configuration input parameters and this example of timing.

NOTICE!

Risk of malfunctions!

Never use the time measurement (bit EN_FREQ=FALSE) mode if the CD522 is connected to a CS31 communication interface module, e. g. CI592.

Depending on the input parameters of function block, the result of time measurement can be measured in time in µs, frequency in Hz or speed of rotation in rotation per minute.



EN_0	EN_1	EN_FREQ	Type	1	2	3	4
FALSE	FALSE	TRUE	No measurement	0	0	0	0
FALSE	TRUE	TRUE	Between 2 falling edges			350	
TRUE	FALSE	TRUE	Between 2 rising edges		500		450
TRUE	TRUE	TRUE	Between any 2 edges	300	200	150	300
FALSE	FALSE	FALSE	No measurement	0	0	0	0
FALSE	TRUE	FALSE	Between the rising edge and the subsequent falling edge	300		150	

EN_0	EN_1	EN_FREQ	Type	1	2	3	4
TRUE	FALSE	FALSE	Between the falling edge and the subsequent rising edge		200		300
TRUE	TRUE	FALSE	Between any 2 edges	300	200	150	300

Input description

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

CNT_NUM

Data type	Default value	Range	Unit
WORD	-	-	-

CNT_NUM contains the counter number in the module:

- CNT_NUM = 0 is related to input A0, B0, Z0
- CNT_NUM = 1 is related to input A1, B1, Z1

EN_CNT

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

Input EN_CNT corresponds to bit 0 in "control byte".

EN_0

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_0=TRUE, the time frequency measurement will be captured on the falling edge of signal.

Input EN_0 corresponds to output bit 1 in control byte.

EN_1

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If EN_1=TRUE, the time frequency measurement will be capture on rising edge of signal.

Input EN_1 corresponds to output bit 2 in control byte.

EN_FREQ

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If EN_FREQ=FALSE, the time frequency measurement will be specified in time mode and displayed on output DUR (in µs).

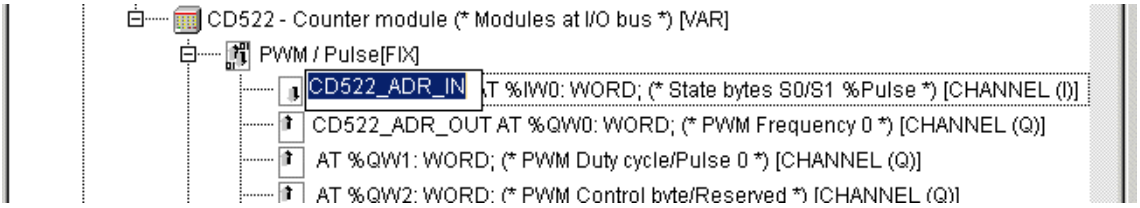
If EN_FREQ= TRUE, the time frequency measurement will be specified in frequency and rpm modes and displayed on output FREQ (in Hz) and RPM (in rotation per minute).

Input EN_FREQ corresponds to output bit 3 in control byte.

ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

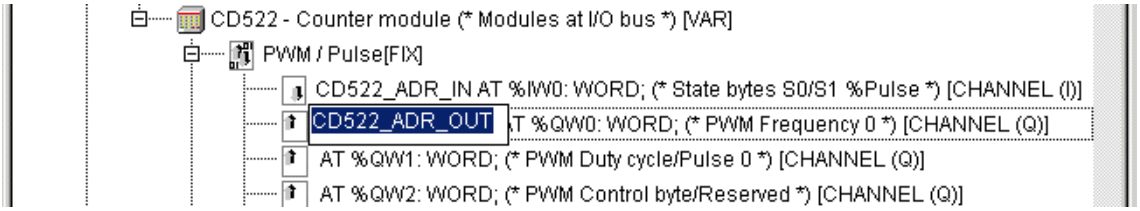
Example (for counter 0):



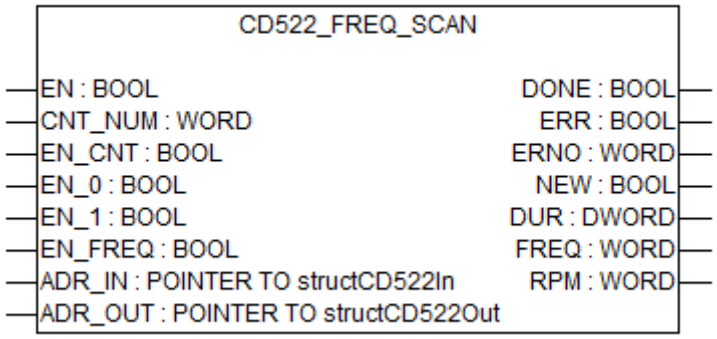
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NEW

Data type	Default value	Range	Unit
BOOL	-	-	-

If output NEW=TRUE, a new timing value is available.

Output NEW corresponds to input bit 6 in state byte.

DUR

Data type	Default value	Range	Unit
DWORD	-	-	-

The output DUR is used for display the result of timing measurement. If the input EN_FREQ=FALSE, measured time is in µs.

Output DUR corresponds to input word Counter low word.

FREQ

Data type	Default value	Range	Unit
WORD	-	-	-

The output FREQ is used for display the result of time measurement. If the input EN_FREQ=TRUE, measured frequency is in Hz.

Output FREQ corresponds to input word Counter high word.

RPM

Data type	Default value	Range	Unit
WORD	-	-	-

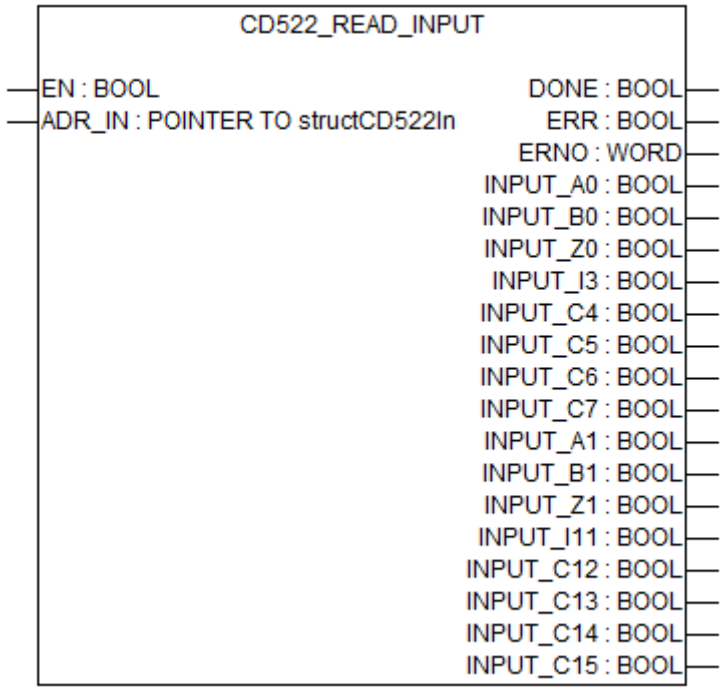
The output RPM is used for display the result of timing measurement. If the input EN_FREQ= TRUE, measured speed of rotation is in rpm (rotations per minute).

Output RPM corresponds to input word Counter low word.

Function call in ST

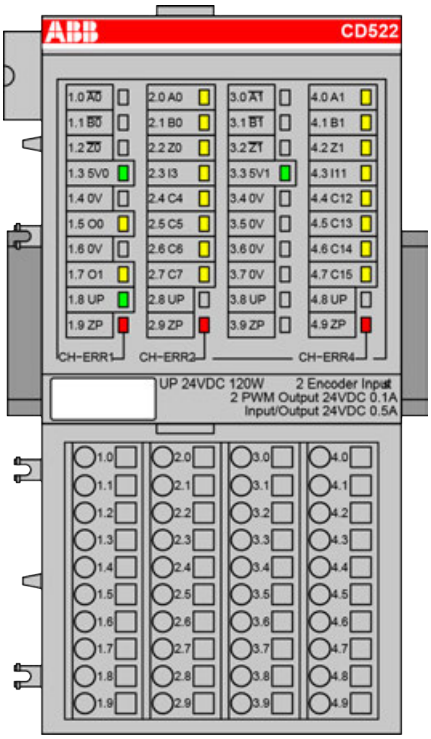
```
CD522FREQSCAN  (EN:=CD522FREQSCAN_EN,
  CNT_NUM:= CD522FREQSCAN_CNT_NUM,
  EN_CNT:= CD522FREQSCAN_EN_CNT,
  EN_0:=CD522FREQSCAN_EN_0,
  EN_1:=CD522FREQSCAN_EN_1,
  EN_FREQ:=CD522FREQSCAN_EN_FREQ,
  ADR_IN:= ADR(CD522FREQSCAN_ADR_IN),
  ADR_OUT:= ADR(CD522FREQSCAN_ADR_OUT));


CD522FREQSCAN_DONE     := CD522FREQSCAN.DONE;
CD522FREQSCAN_ERR      := CD522FREQSCAN.ERR;
CD522FREQSCAN_ERNO     := CD522FREQSCAN.ERNO;
CD522FREQSCAN_NEW      := CD522FREQSCAN.NEW;
CD522FREQSCAN_DUR      := CD522FREQSCAN.DUR;
CD522FREQSCAN_FREQ     := CD522FREQSCAN.FREQ;
CD522FREQSCAN_RPM      := CD522FREQSCAN.RPM;
```

CD522_READ_INPUT

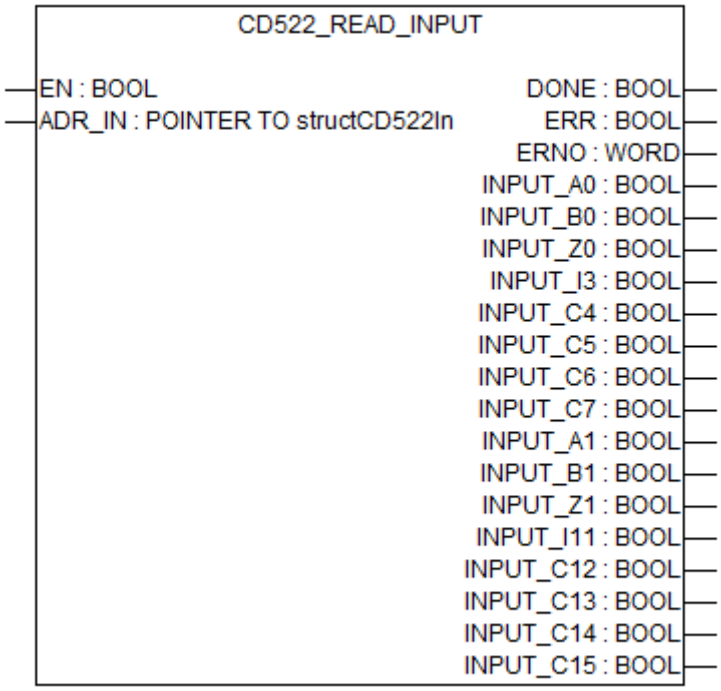


Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

The function block is used to read binary inputs from CD522. The output parameter INPUT_xx of the function block refers to the respective input channel xx on CD522.



Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

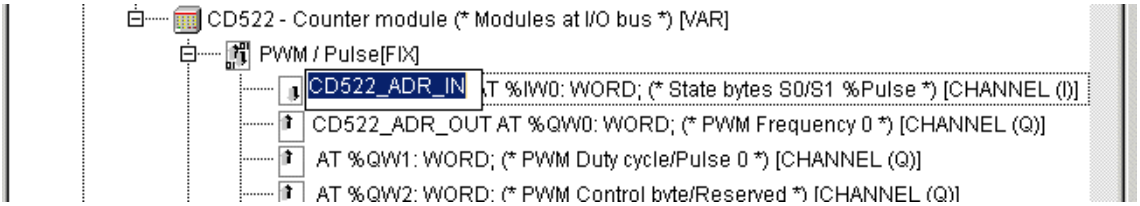
In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

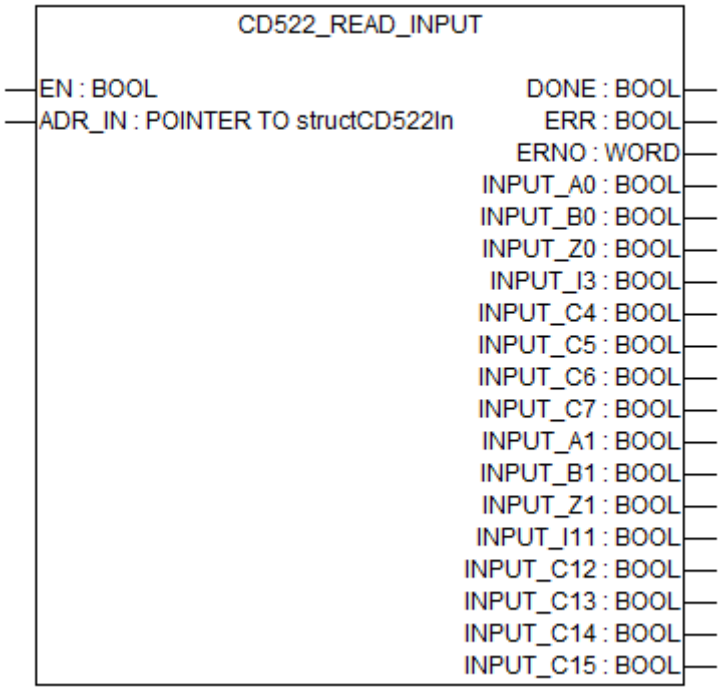
ADR_IN

At ADR_IN input (POINTER TO structCD522counterin), the address of the first input data from the structure of counter input of CD522 should be connected. The use of an ADR operator is needed. If input ADR_IN is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

INPUT

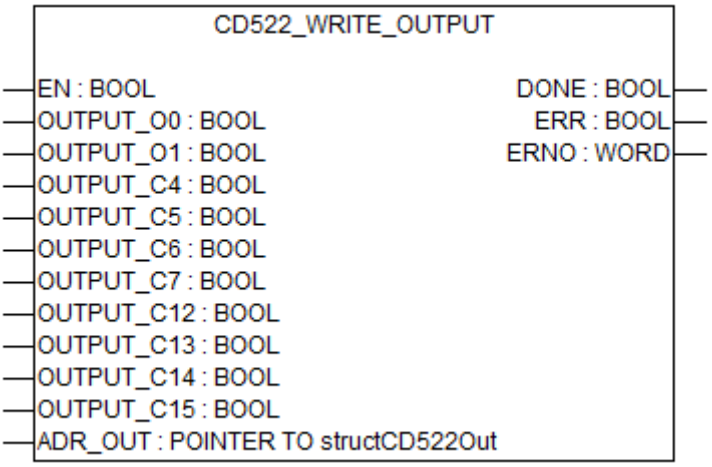
Data type	Default value	Range	Unit
BOOL	-	-	-

MUSS HIER WAS STEHEN???????

Function call in ST

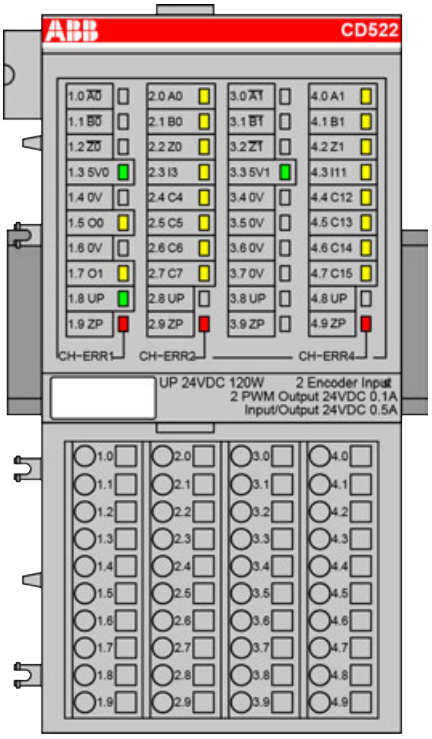
```
CD522READINPUT   (EN:=CD522_READINPUT_EN,

         ADR_IN:= ADR(CD522_READINPUT_ADR_IN));


CD522_READINPUT_DONE                := CD522_READINPUT.DONE;
CD522_READINPUT_ERR                 := CD522_READINPUT.ERR;
CD522_READINPUT_ERNO                := CD522_READINPUT.ERNO;
CD522_READINPUT_INPUT_A0           := CD522_READINPUT.INPUT_A0
CD522_READINPUT_INPUT_B0           := CD522_READINPUT.INPUT_B0
CD522_READINPUT_INPUT_Z0           := CD522_READINPUT.INPUT_Z0
CD522_READINPUT_INPUT_I3           := CD522_READINPUT.INPUT_I3
CD522_READINPUT_INPUT_C4           := CD522_READINPUT.INPUT_C4
CD522_READINPUT_INPUT_C5           := CD522_READINPUT.INPUT_C5
CD522_READINPUT_INPUT_C6           := CD522_READINPUT.INPUT_C6
CD522_READINPUT_INPUT_C7           := CD522_READINPUT.INPUT_C7
CD522_READINPUT_INPUT_A1           := CD522_READINPUT.INPUT_A1
CD522_READINPUT_INPUT_B1           := CD522_READINPUT.INPUT_B1
CD522_READINPUT_INPUT_Z1           := CD522_READINPUT.INPUT_Z1
CD522_READINPUT_INPUT_I11           := CD522_READINPUT.INPUT_I11
CD522_READINPUT_INPUT_C12           := CD522_READINPUT.INPUT_C12
CD522_READINPUT_INPUT_C13           := CD522_READINPUT.INPUT_C13
CD522_READINPUT_INPUT_C14           := CD522_READINPUT.INPUT_C14
CD522_READINPUT_INPUT_C15           := CD522_READINPUT.INPUT_C15
```

CD522_WRITE_OUTPUT

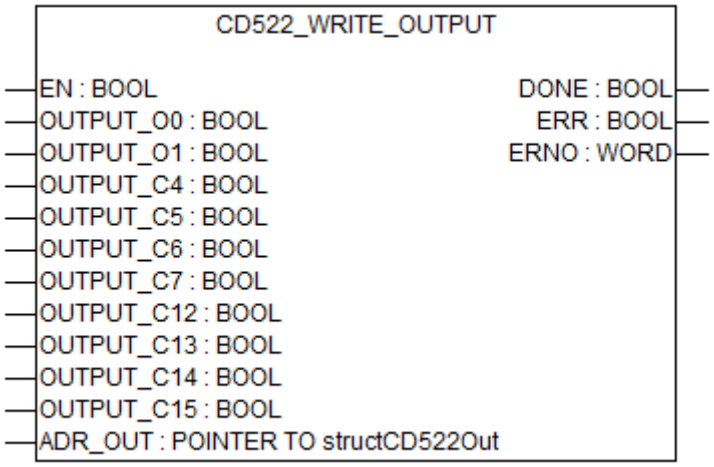


Parameter	Value
Included in library	CD522_AC500_V13.lib
Available from CD522 firmware	V2.1
Available from runtime system:	V1.0.2
Available from S500 I/O modules (DC551) firmware	V1.11
Type	Function block with historical values

The function block is used to write binary outputs from CD522. The input parameter OUTPUT_xx of the function block refers to the respective input channel xx on CD522.



Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

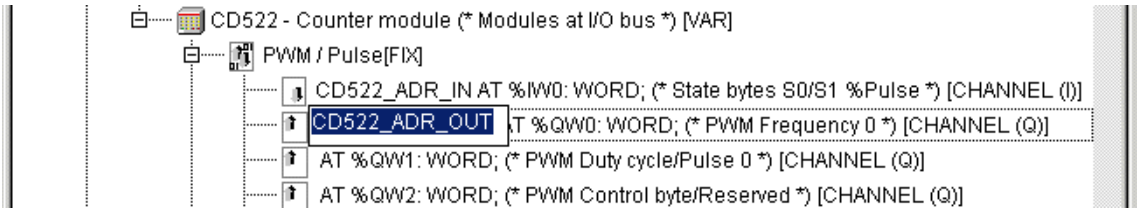
OUTPUT

Data type	Default value	Range	Unit
BOOL	-	-	-

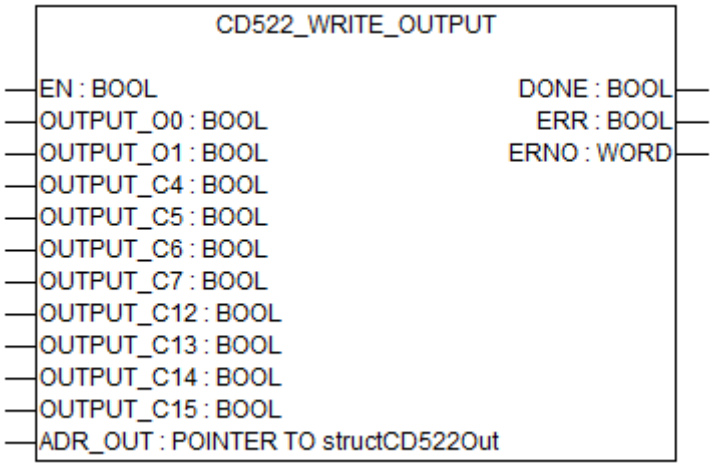
ADR_OUT

At ADR_OUT input (POINTER TO structCD522counterout), the address of the first output data from the structure of counter output of CD522 should be connected. The use of an ADR operator is needed. If input ADR_OUT is not connected, then the outputs DONE=FALSE and ERR= TRUE.

Example (for counter 0):



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
CD522WRITEOUTPUT   (EN:=CD522_WRITEOUTPUT_EN,
         OUTPUT_O0:= CD522_WRITEOUTPUT_OUTPUT_O0,
         OUTPUT_O1:= CD522_WRITEOUTPUT_OUTPUT_O1,
         OUTPUT_C4:= CD522_WRITEOUTPUT_OUTPUT_C4,
         OUTPUT_C5:= CD522_WRITEOUTPUT_OUTPUT_C5,
         OUTPUT_C6:= CD522_WRITEOUTPUT_OUTPUT_C6,
         OUTPUT_C7:= CD522_WRITEOUTPUT_OUTPUT_C7,
         OUTPUT_C12:= CD522_WRITEOUTPUT_OUTPUT_C12,
         OUTPUT_C13:= CD522_WRITEOUTPUT_OUTPUT_C13,
         OUTPUT_C14:= CD522_WRITEOUTPUT_OUTPUT_C14,
         OUTPUT_C15:= CD522_WRITEOUTPUT_OUTPUT_C15,
         ADR_OUT:= ADR(CD522_WRITEOUTPUT_ADR_OUT));
```

```
CD522_WRITEOUTPUT_DONE                  := CD522_WRITEOUTPUT.DONE;
CD522_WRITEOUTPUT_ERR                   := CD522_WRITEOUTPUT.ERR;
CD522_WRITEOUTPUT_ERNO                  := CD522_WRITEOUTPUT.ERNO;
```

1.5.4.9 Counter library

Library file name: **Counter_AC500_Vx.lib**

The Counter Library contains function blocks that simplify the use of the fast counters of the S500 I/O modules on the I/O bus of the CPU and the fast counters inside CS31 communication interface modules respectively.

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The function blocks are available in AC500 control systems with runtime system version V2.0.0 or higher and S500 I/O modules with firmware version V1.3 or higher.

1.5.4.9.1 Function blocks

CNT_DC551

Parameter	Value
Included in library	Counter_AC500_V20.lib
Available as of runtime system	V2.0.0
Available as of CS31 com-munucation interface module firmware	V1.3, V2.0
Remark	V1.3 at DC551-CS31, V2.0 at CI590-CS31-HA and CI592-CS31
Type	Function block with historical values

Function block CNT_DC551 is used to control the fast counter of the CS31 communucation interface modules (e. g. DC551-CS31, CI590-CS31-HA and CI592-CS31).

The operating modes of the fast counters are described in *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

To activate the fast counter in CS31 communucation interface modules, the address switch of the CS31 communucation interface modules must be set to a value in the range between 70 and 99. Then, the actual module address is the set address minus 70, i.e. it is in a range between 0 and 29.

Data exchange between CPU and high-speed counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1

- 4 words analog inputs for 2 actual value double words of counters 1 and 2

- 2 bytes digital outputs for control bytes of counters 1 and 2

- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

Thus, 1 CS31 communucation interface module with activated fast counter (without S500 expansion modules) occupies 2 CS31 software modules:

Module 1: digital module with digital inputs (3 bytes) and digital outputs (4 bytes)

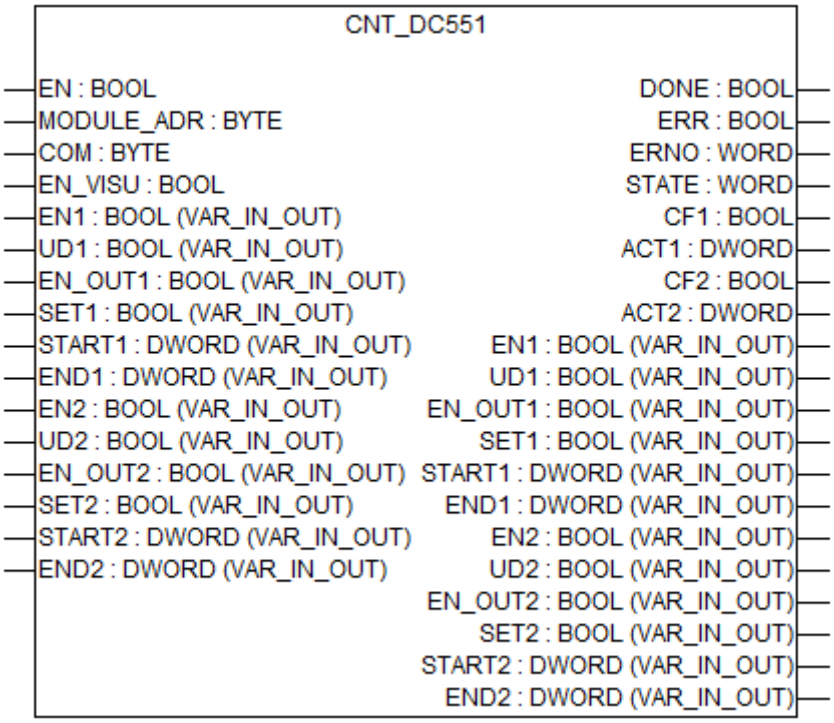
Module 2: analog module with analog inputs (8 words) and analog outputs (4 words)

The address of the modules corresponds to the address set at the DC551 minus 70.

CAUTION!
The high-speed counters of the S500 communication interface modules are not available if they are connected to CS31 communication interface modules. They are only available after connecting the modules to the I/O Bus of the AC500 CPU.

The function block CNT_DC551 has an integrated visualization visuCNT_DC551 that can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

MODULE_ADR

Data type	Default value	Range	Unit
BYTE	-	0 ... 29	-

At input MODULE_ADR, the address set at the CS31 communucation interface module is entered.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At input COM, the number of the serial interface is specified to which the CS31 communucation interface module is connected.

Valid values are 0 and 1 for COM1 and 2 for COM2 (only CM574-RS).

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except SLOT, COM, MODULE_ADR, EN and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed ↳ *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

EN1

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up

UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

Only for fast counter operating modes 1 and 2:

If EN_OUT1 = FALSE, the related digital output DO acts as an output indicating "end value reached" of the fast counter.

If EN_OUT1 = TRUE, the related digital output DO can be used as normal digital output.

The value change of EN_OUT1 is only effective when EN = TRUE.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always over-written by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the CS31 communucation interface module.

END1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the CS31 communucation interface module.

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up

UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT2 is reserved. A variable with the value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always over-written by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the CS31 communucation interface module.

END2

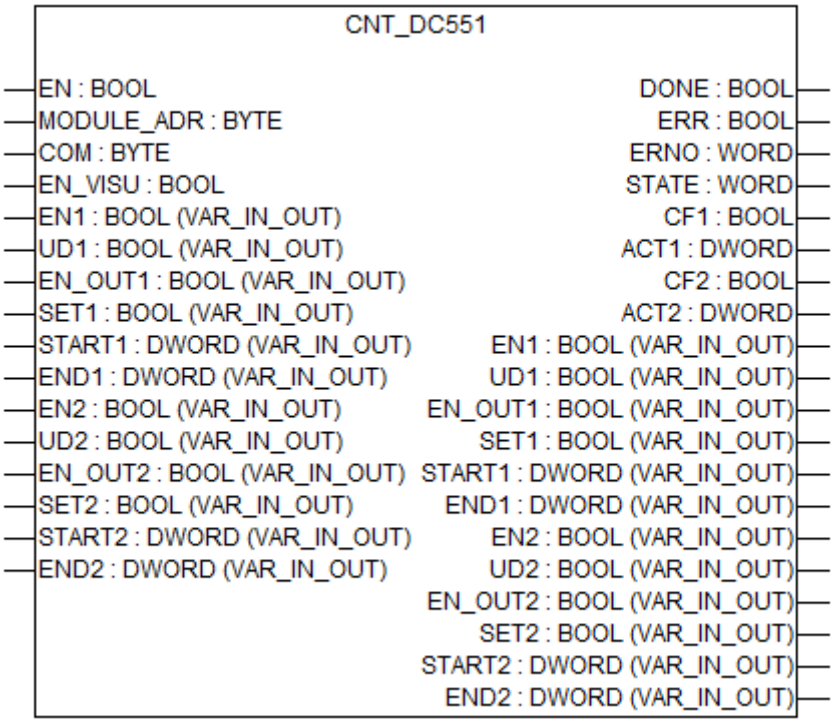
Data type	Default value	Range	Unit
DWORD	-	-	-

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the CS31 communucation interface module.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
WORD	-	-	-

Output STATE indicates the current mode of fast counter. The mode of fast counter (0-10) can be configured in the channel parameter of the PLC configuration (please refer to fast counter mode in system technology chapter 9.3).

CF1

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2

Data type	Default value	Range	Unit
DWORD	-	-	-

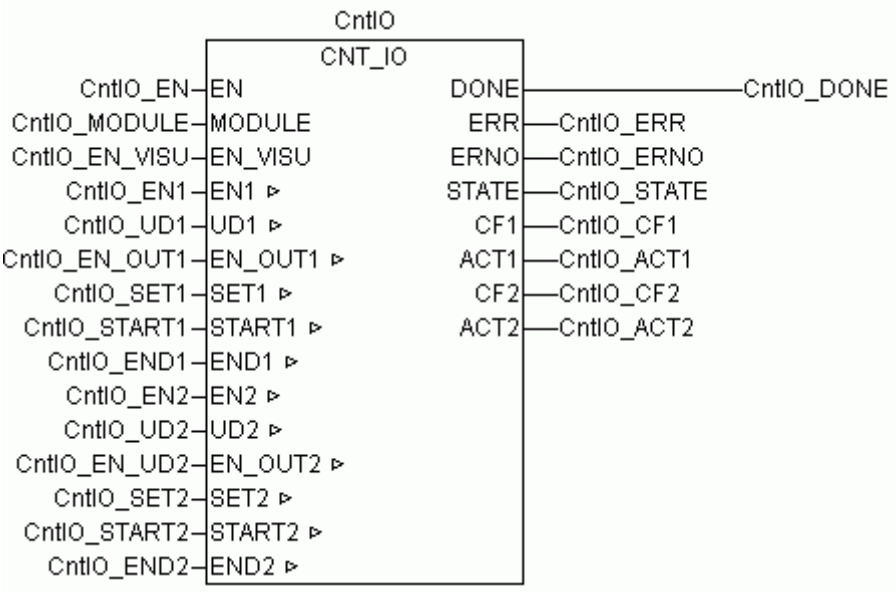
At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in ST

```
CntDC551 (EN          := CntDC551_EN
          MODULE_ADR := CntDC551_MODULE_ADR,
          COM         := CntDC551_COM,
          EN_VISU     := CntDC551_EN_VISU,
          EN1         := CntDC551_EN1,
          UD1         := CntDC551_UD1,
          EN_OUT1     := CntDC551_EN_OUT1,
          SET1        := CntDC551_SET1,
          START1      := CntDC551_START1,
          END1        := CntDC551_END1,
          EN2         := CntDC551_EN2,
          UD2         := CntDC551_UD2,
          EN_OUT2     := CntDC551_EN_OUT2,
          SET2        := CntDC551_SET2,
          START2      := CntDC551_START2,
          END2        := CntDC551_END2);


CntDC551_DONE      := CntDC551.DONE;
CntDC551_ERR       := CntDC551.ERR;
CntDC551_ERNO      := CntDC551.ERNO;
CntDC551_CF1       := CntDC551.CF1;
CntDC551_ACT1      := CntDC551.ACT1;
CntDC551_CF2       := CntDC551.CF2;
CntDC551_Act2      := CntCS31.ACT2;
```

CNT_IO



Function block CNT_IO is used to control the fast counter of the digital S500 I/O Devices.

The fast counter which is integrated in CPUs PM55x and PM56x (onboard I/Os) can only be managed by function block ONB_IO_CNT ↳ Chapter 1.5.4.25.1.1 “ONB_IO_CNT” on page 1734 (and not by CNT_IO).

Parameter	Value
Included in library	Counter_AC500_V20.lib
Available as of runtime system	V2.0
Available as of S500 I/O module firmware:	V1.3
Type	Function block with historical values

Function block CNT_IO is used to control the fast counter of the digital S500 I/O Modules.

The operating modes of the fast counters are described in ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

To activate the fast counter of a digital S500 I/O module, the parameter "High-speed counter" of the I/O module must be set to the desired counting mode in the control system configuration.

Data exchange between CPU and high-speed counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1

- 4 words analog inputs for 2 actual value double words of counters 1 and 2

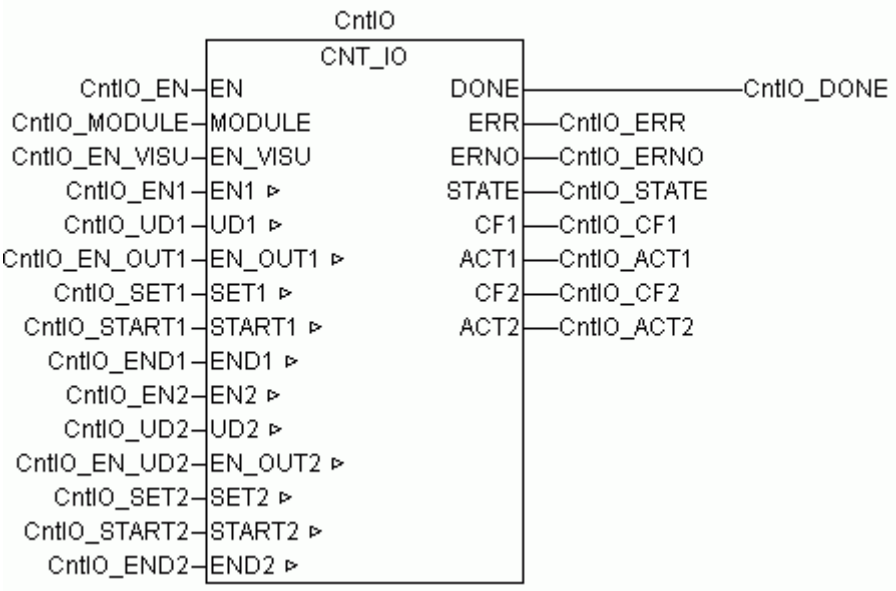
- 2 bytes digital outputs for control bytes of counters 1 and 2

- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

CAUTION!
The high-speed counters of the digital S500 I/O devices are only available when connected to the I/O bus of the AC500 CPUs. If connected to the CS31 bus (e. g. with DC551-CS31 or CI590-CS31-HA) no fast counters are available.

The function block CNT_IO has an integrated visualization visuCNT_IO that can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

Module

Data type	Default value	Range	Unit
BYTE	-	1 ... 10	-

At input MODULE, the module number of the digital S500 I/O module on the I/O bus of the AC500 CPU is specified. The first module with number 1 is the module directly right to the CPU.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except SLOT, COM, MODULE_ADR, EN and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

EN1

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up

UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

Only for fast counter operating modes 1 and 2:

If EN_OUT1 = FALSE, the related digital output DO acts as an output indicating "end value reached" of the fast counter.

If EN_OUT1 = TRUE, the related digital output DO can be used as normal digital output.

The value change of EN_OUT1 is only effective when EN = TRUE.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always over-written by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the CS31 communucation interface module.

END1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the CS31 communucation interface module.

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up

UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT2 is reserved. A variable with the value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always overwritten by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the CS31 communucation interface module.

END2

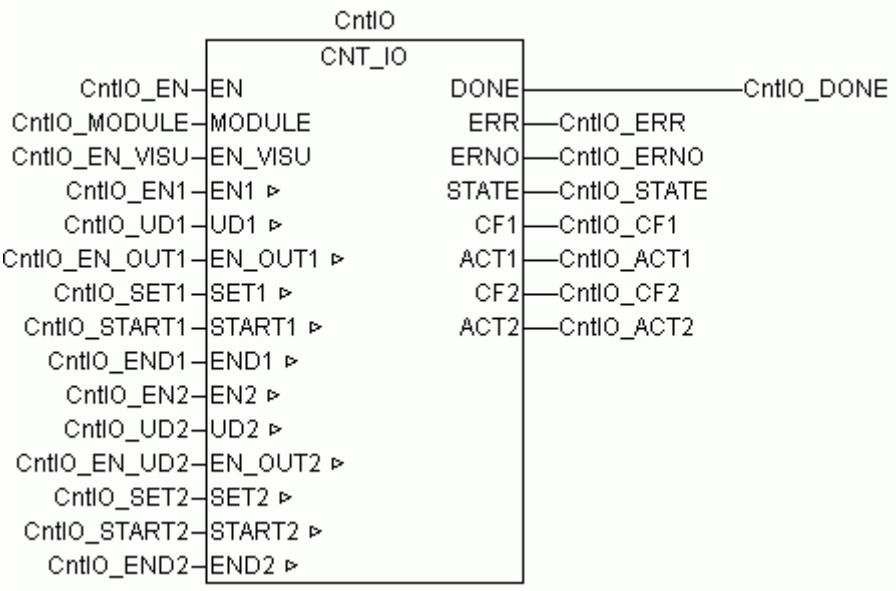
Data type	Default value	Range	Unit
DWORD	-	-	-

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the CS31 communucation interface module.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
WORD	-	-	-

Output STATE indicates the current mode of fast counter. The mode of fast counter (0-10) can be configured in the channel parameter of the PLC configuration (please refer to fast counter mode in system technology chapter 9.3).

CF1

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2

Data type	Default value	Range	Unit
DWORD	-	-	-

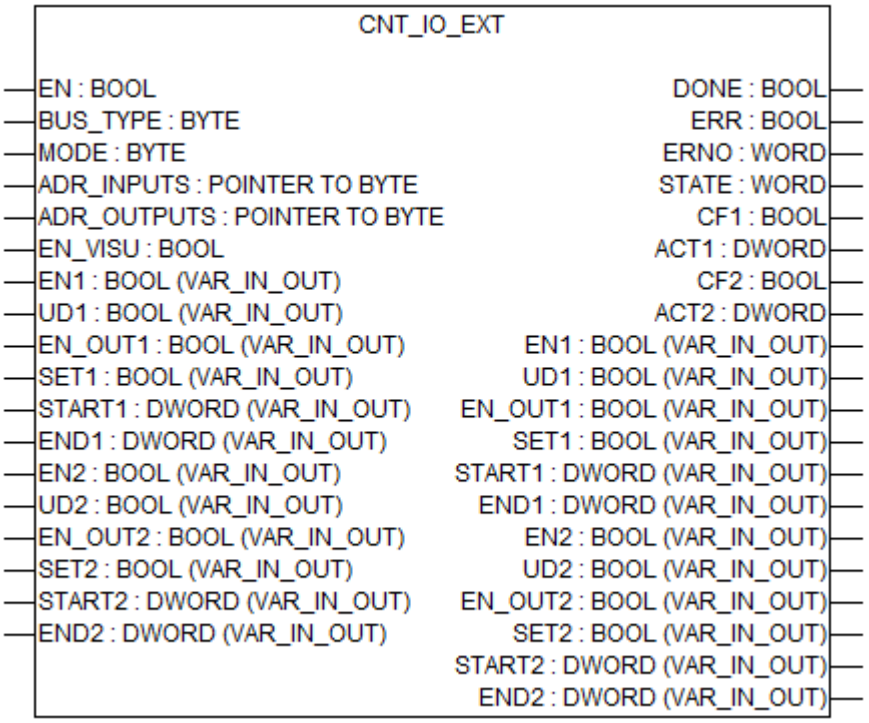
At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in ST

```
CntIO (EN        := CntIO_EN
       MODULE    := CntIO_MODULE,
       EN_VISU   := CntIO_EN_VISU,
       EN1       := CntIO_EN1,
       UD1       := CntIO_UD1,
       EN_OUT1   := CntIO_EN_OUT1,
       SET1      := CntIO_SET1,
       START1    := CntIO_START1,
       END1      := CntIO_END1,
       EN2       := CntIO_EN2,
       UD2       := CntIO_UD2,
       EN_OUT2   := CntIO_EN_OUT2,
       SET2      := CntIO_SET2,
       START2    := CntIO_START2,
       END2      := CntIO_END2);


CntIO_DONE       := CntIO.DONE;
CntIO_ERR        := CntIO.ERR;
CntIO_ERNO       := CntIO.ERNO;
CntIO_CF1        := CntIO.CF1;
CntIO_ACT1       := CntIO.ACT1;
CntIO_CF2        := CntIO.CF2;
CntIO_Act2       := CntIO.ACT2;
```

CNT_IO_EXT



The fast counter which is integrated in CPUs PM55x and PM56x (onboard I/Os) can only be managed by function block ONB_IO_CNT ↯ Chapter 1.5.4.25.1.1 “ONB_IO_CNT” on page 1734 (and not by CNT_IO_EXT).

Parameter	Value
Included in library	Counter_AC500_V20.lib
Available as of runtime system	V2.0
Available as of S500 I/O module firmware:	V1.3
Type	Function block with historical values

Function block CNT_IO_EXT is used to control the fast counter of the digital S500 I/O Modules.

The operating modes of the fast counters are described in ↯ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

To activate the fast counter of a digital S500 I/O module, the parameter "fast counter" of the I/O module must be set to the desired counting mode in the control system configuration.

Data exchange between CPU and fast counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1

- 4 words analog inputs for 2 actual value double words of counters 1 and 2

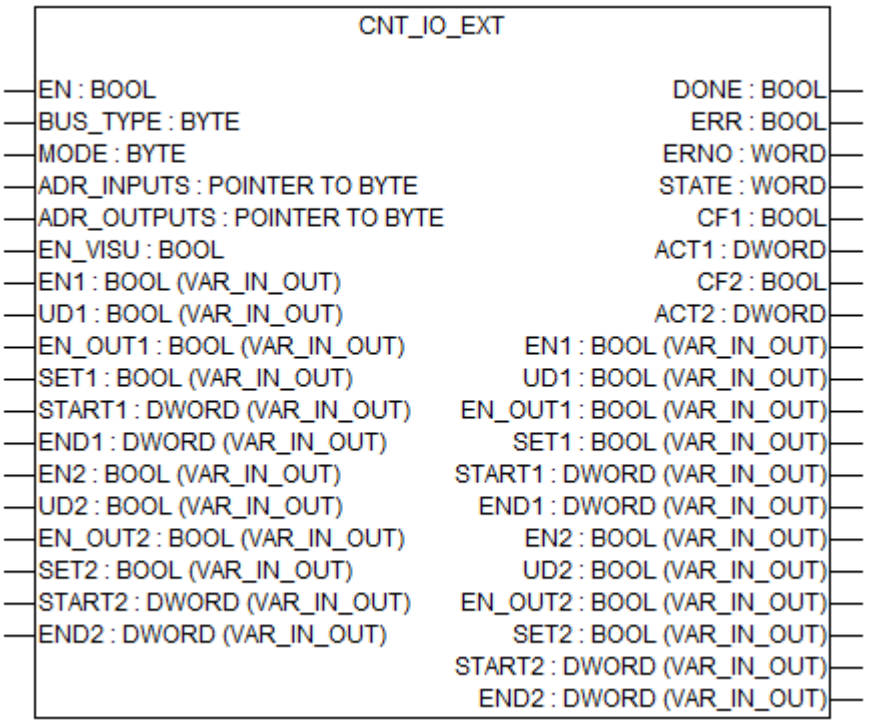
- 2 bytes digital outputs for control bytes of counters 1 and 2

- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

CAUTION!
The high-speed counters of the digital S500 I/O devices are only available when connected to the I/O bus of the AC500 CPUs. If connected to the CS31 bus (e. g. with DC551-CS31 or CI590-CS31-HA) no fast counters are available.

The function block CNT_IO_EXT has an integrated visualization visuCNT_IO_EXT that can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

BUS_TYPE

Data type	Default value	Range	Unit
BYTE	-	1 ... 10	-

At input BUS_TYPE, the user can selected between counters on I/O bus and CAN bus. The value 0 is default and means I/O bus. The value 1 means CAN bus.

MODE

Data type	Default value	Range	Unit
BYTE	0	0 ... 10	-

At input MODE the operating mode of the fast counter must be assigned. Value 0: no counter active. Value 1…10 specifies special counter modes. The operating modes of the fast counters are described in detail in a separate chapter *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

ADR_INPUTS

Data type	Default value	Range	Unit
POINTER_TO_BYTE	0	0 … 4294967295	-

Pointer to inputs of the fast counter. Use the ADR operator and the symbolic name of the counters input data in the module I/O-mapping. Example: ADR(CntInData)

ADR_OUTPUTS

Data type	Default value	Range	Unit
POINTER_TO_BYTE	0	0 … 4294967295	-

Pointer to outputs of the fast counter. Use the ADR operator and the symbolic name of the counters output data in the module I/O-mapping. Example: ADR(CntOutData)

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except SLOT, COM, MODULE_ADR, EN and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

EN1

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up

UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

Only for fast counter operating modes 1 and 2:

If EN_OUT1 = FALSE, the related digital output DO acts as an output indicating "end value reached" of the fast counter.

If EN_OUT1 = TRUE, the related digital output DO can be used as normal digital output.

The value change of EN_OUT1 is only effective when EN = TRUE.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always overwritten by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the CS31 communucation interface module.

END1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the CS31 communucation interface module.

EN2

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If input EN2 = TRUE, pulse counting of counter 2 is enabled. No pulse counting, if EN2 = FALSE.

Input EN2 corresponds to bit 1 in control byte 1

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up

UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT2 is reserved. A variable with the value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always over-written by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the CS31 communucation interface module.

END2

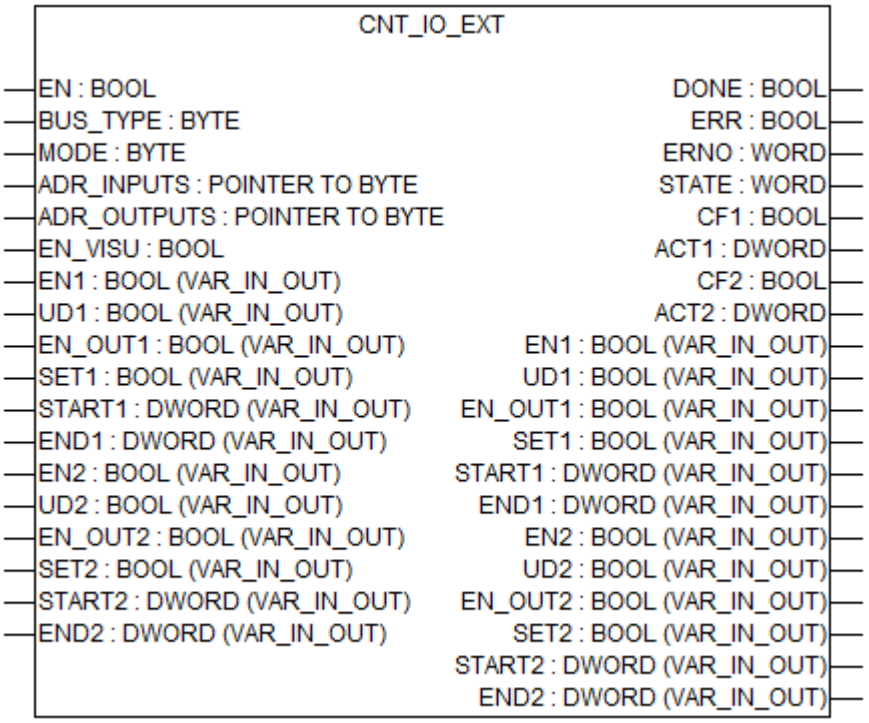
Data type	Default value	Range	Unit
DWORD	-	-	-

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the CS31 communucation interface module.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
WORD	-	-	-

Output STATE indicates the current mode of fast counter. The mode of fast counter (0-10) can be configured in the channel parameter of the PLC configuration (please refer to fast counter mode in system technology chapter 9.3).

CF1

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT2, the actual value = counter reading of counter 2 is output as double word.

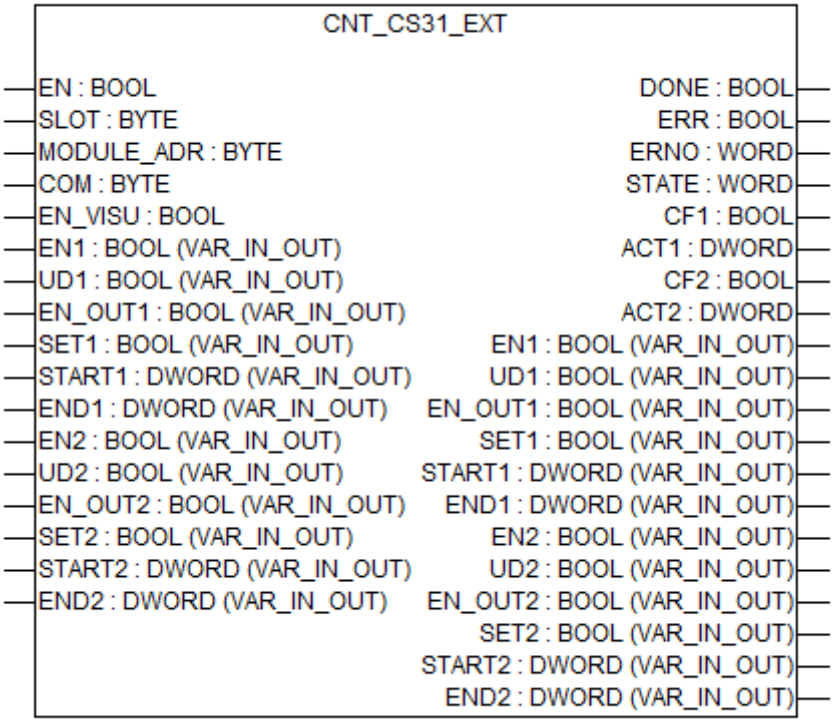
Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in ST

```
CntIoExt (EN         := CntIO_EN
       BUS_TYPE    := CntIoExt_BUS_TYPE,
       MODE        := CntIoExt_MODE,
       ADR_INPUTS  := CntIoExt_ADR_INPUTS,
       ADR_OUTPUTS := CntIoExt_ADR_OUTPUTS
       EN_VISU     := CntIoExt_EN_VISU,
       EN1         := CntIoExt_EN1,
       UD1         := CntIoExt_UD1,
       EN_OUT1     := CntIoExt_EN_OUT1,
       SET1        := CntIoExt_SET1,
       START1      := CntIoExt_START1,
       END1        := CntIoExt_END1,
       EN2         := CntIoExt_EN2,
       UD2         := CntIoExt_UD2,
       EN_OUT2     := CntIoExt_EN_OUT2,
       SET2        := CntIoExt_SET2,
       START2      := CntIoExt_START2,
       END2        := CntIoExt_END2);
```

```
Cnt_IoExt_DONE      := Cnt_IoExt.DONE;
Cnt_IoExt_ERR       := Cnt_IoExt.ERR;
Cnt_IoExt_ERNO      := Cnt_IoExt.ERNO;
Cnt_IoExt_CF1       := Cnt_IoExt.CF1;
Cnt_IoExt_ACT1      := Cnt_IoExt.ACT1;
Cnt_IoExt_CF2       := Cnt_IoExt.CF2;
Cnt_IoExt_Act2      := Cnt_IoExt.ACT2;
```

CNT_CS31_EXT



Parameter	Value
Included in library	Counter_AC500_V20.lib
Available as of runtime system	V2.0.0
Available as of CS31 com-munucation interface module firmware	V1.3, V2.0
Remark	V1.3 at DC551-CS31, V2.0 at CI590-CS31-HA and CI592-CS31
Type	Function block with historical values

Function block CNT_CS31_EXT is used to control the fast counter in CS31 communication interface modules (e. g. DC551-CS31, CI590-CS31-HA and CI592-CS31).

Function block CNT_CS31_EXT is derived from CNT_DC551 and works very similar to it.

As of firmware version 2.0 it is possible to configure and access the serial ports of a CM574-RS through the configuration of the AC500 CPU. Because CM574-RS can be installed into 1 of the 4 possible Communication Module slots, this function block has an additional input (SLOT) to give the position where the CM574 is installed. If COM1 of an AC500 CPU should be used, or if the user program which is using this function block is located in the CM574-RS, this input must have the value 0.

All other functions are the same as in CNT_DC551.

This function block can be used only with a configuration done by ABB Control Builder Plus.

The operating modes of the fast counters are described in *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

To activate the fast counter in CS31 communication interface modules, the address switch of the CS31 communication interface modules must be set to a value in the range between 70 and 99. Then, the actual module address is the set address minus 70, i.e. it is in a range between 0 and 29.

Data exchange between CPU and high-speed counter is done using input/output data. The following is required for the counter:

- 2 bytes digital inputs for status bytes 0 and 1

- 4 words analog inputs for 2 actual value double words of counters 1 and 2

- 2 bytes digital outputs for control bytes of counters 1 and 2

- 8 words analog outputs for 4 start/end value double words of counters 1 and 2

Thus, 1 CS31 communication interface module with activated fast counter (without S500 communication interface modules) occupies 2 CS31 software modules:

Module 1: digital module with digital inputs (3 bytes) and digital outputs (4 bytes)

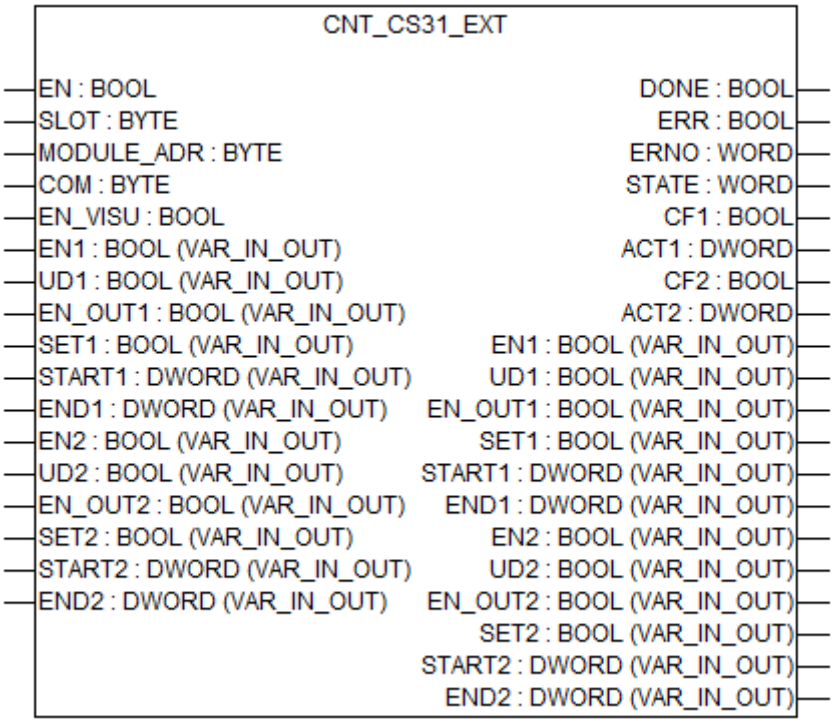
Module 2: analog module with analog inputs (8 words) and analog outputs (4 words)

The address of the modules corresponds to the address set at the DC551 minus 70.

CAUTION!
The high-speed counters of the S500 communication interface modules are not available if they are connected to CS31 communication interface modules. They are only available after connecting the modules to the I/O Bus of the AC500 CPU.

The function block CNT_DC551 has an integrated visualization visuCNT_CS31 that can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable pulse counting for input CH, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

When calling the function block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and a corresponding error is displayed at output ERR/ERNO.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

MODULE_ADR

Data type	Default value	Range	Unit
BYTE	-	0 ... 29	-

At input MODULE_ADR, the address set at the CS31 communucation interface module is entered.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At input COM, the number of the serial interface is specified to which the CS31 communucation interface module is connected.

Valid values are 0 and 1 for COM1 and 2 for COM2 (only CM574-RS).

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except SLOT, COM, MODULE_ADR, EN and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed ↳ *Chapter 1.5.4.9.2 “Visualization” on page 1064*.

EN1

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up

UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT1 is used to select the output control mode for the operating modes with direct output activation (modes 1 and 2).

Only for fast counter operating modes 1 and 2:

If EN_OUT1 = FALSE, the related digital output DO acts as an output indicating "end value reached" of the fast counter.

If EN_OUT1 = TRUE, the related digital output DO can be used as normal digital output.

The value change of EN_OUT1 is only effective when EN = TRUE.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always overwritten by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input START1 corresponds to the analog outputs 0 (bit 16..31) and 1 (bit 0..15) of the CS31 communucation interface module.

END1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

Input END1 corresponds to the analog outputs 2 (bit 16..31) and 3 (bit 0..15) of the CS31 communucation interface module.

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up

UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT2 is reserved. A variable with the value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always overwritten by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input START2 corresponds to the analog outputs 4 (bit 16..31) and 5 (bit 0..15) of the CS31 communucation interface module.

END2

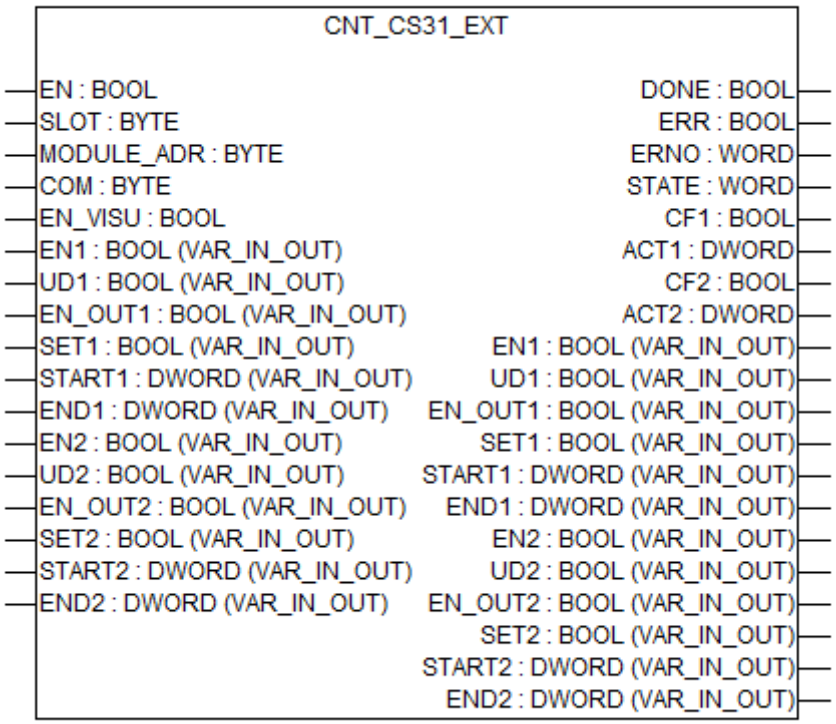
Data type	Default value	Range	Unit
DWORD	-	-	-

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Input END2 corresponds to the analog outputs 6 (bit 16..31) and 7 (bit 0..15) of the CS31 communucation interface module.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
WORD	-	-	-

Output STATE indicates the current mode of fast counter. The mode of fast counter (0-10) can be configured in the channel parameter of the PLC configuration (please refer to fast counter mode in system technology chapter 9.3).

CF1

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

Output ACT1 corresponds to the analog inputs 0 (bit 16..31) and 1 (bit 0..15).

CF2

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Output ACT2 corresponds to the analog inputs 2 (bit 16..31) and 3 (bit 0..15).

Function call in ST

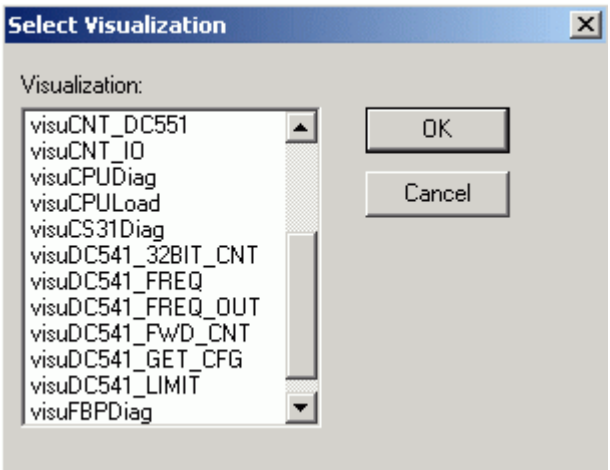
```
CntCS31ext (EN          := CntCS31ext_EN,
SLOT        := CntCS31ext_SLOT,
            MODULE_ADR := CntCS31ext_MODULE_ADR,
            COM         := CntCS31ext_COM,
            EN_VISU     := CntCS31ext_EN_VISU,
            EN1         := CntCS31ext_EN1,
            UD1         := CntCS31ext_UD1,
            EN_OUT1     := CntCS31ext_EN_OUT1,
            SET1        := CntCS31ext_SET1,
            START1      := CntCS31ext_START1,
            END1        := CntCS31ext_END1,
            EN2         := CntCS31ext_EN2,
            UD2         := CntCS31ext_UD2,
            EN_OUT2     := CntCS31ext_EN_OUT2,
            SET2        := CntCS31ext_SET2,
            START2      := CntCS31ext_START2,
            END2        := CntCS31ext_END2);


CntCS31ext_DONE      := CntCS31ext.DONE;
CntCS31ext_ERR       := CntCS31ext.ERR;
CntCS31ext_ERNO      := CntCS31ext.ERNO;
CntCS31ext_CF1       := CntCS31ext.CF1;
CntCS31ext_ACT1      := CntCS31ext.ACT1;
CntCS31ext_CF2       := CntCS31ext.CF2;
CntCS31ext_Act2      := CntCS31ext.ACT2;
```

1.5.4.9.2 Visualization

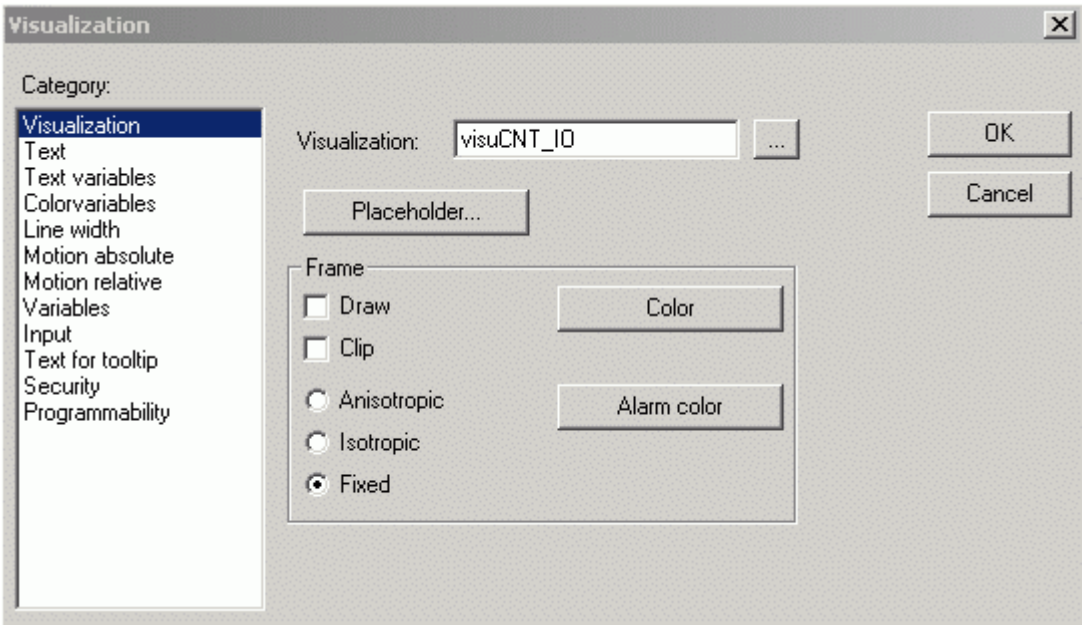
The visualization can be used to display the function block outputs. If the input EN_VISU of a function block is TRUE, it is also possible to control the inputs from the visualization. In order to allow the control of function block inputs from the program as well as from the visualization, these inputs are declared as VAR_IN_OUT and therefore have to be provided with variables accordingly. These inputs must not be provided with direct constants.

Proceed as follows to integrate the visualization into a project:

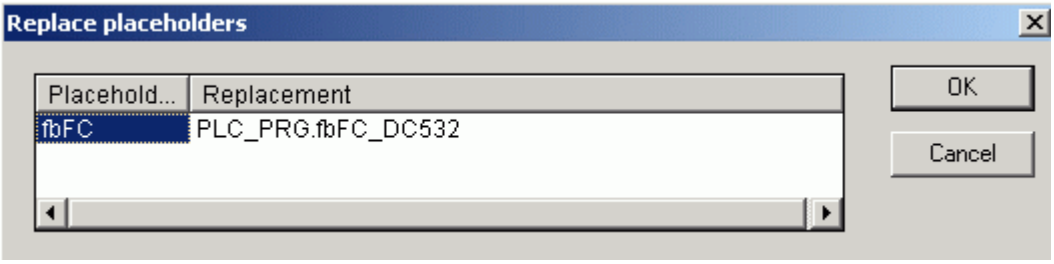
- Create a new visualization using Visualizations / Insert Object (e.g. visuTestCntIO).
- Insert a visualization using 
- In the appearing dialog, select the corresponding visualization for the function block:



- Then, the visualization integrated just now has to be configured. Highlight the visualization with a left mouse click. Then click the right mouse button and select the function "Configure..." from the context menu.
- The configuration of the visualization is done in the appearing dialog. It is recommended to set the frame to "Fixed". This way, the original width-to-height ratio and font size are kept.



- Now the visualization has to be linked to the function block instance. This is done in a dialog, opened after clicking the button "Placeholder".
- In the "Replacement" column of this dialog, the function block instance can either be entered directly or selected by pressing <F2>.



- By clicking <OK> the dialogs are closed. After this, the inserted visualization has to be adapted to the correct size.

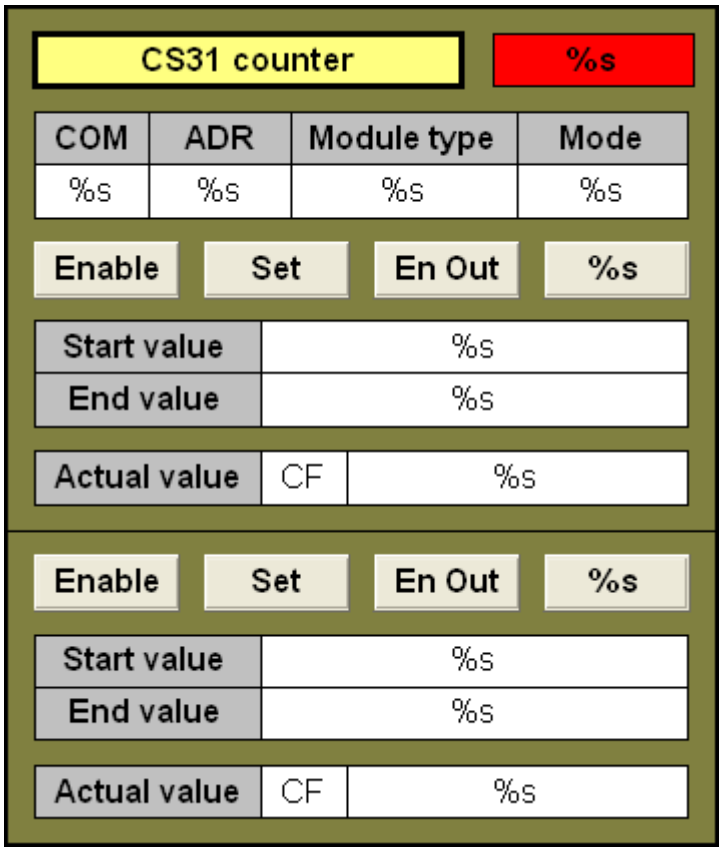


Fig. 24: Offline mode.

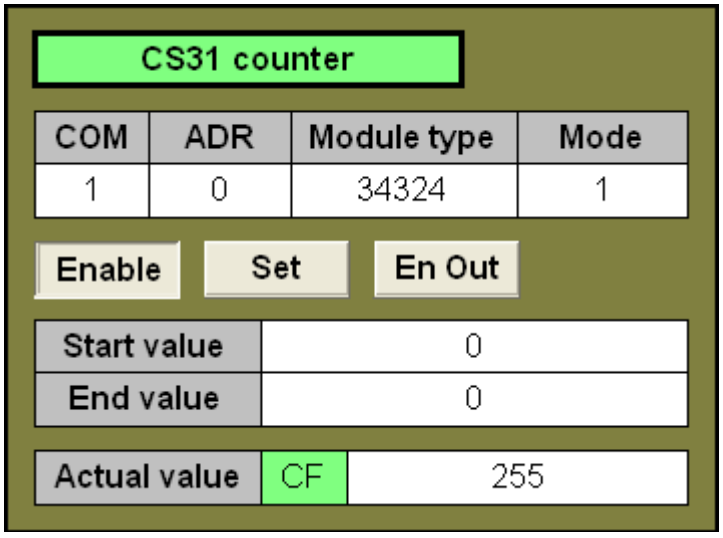
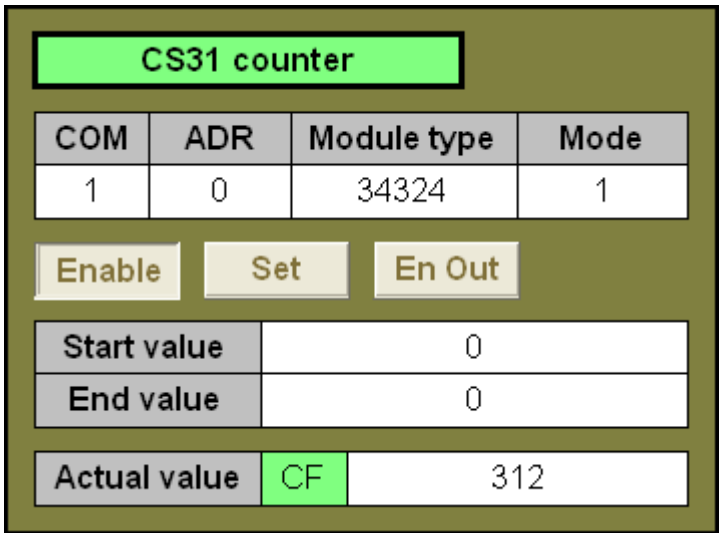


Fig. 25: Online mode, example for operating mode 1.

If an error occurred during function block processing, the error number is displayed in the top right until EN becomes FALSE.

If input EN_VISU = FALSE, the inputs cannot be modified using the visualization. The corresponding control elements are then displayed in gray:



1.5.4.10 CS31 library

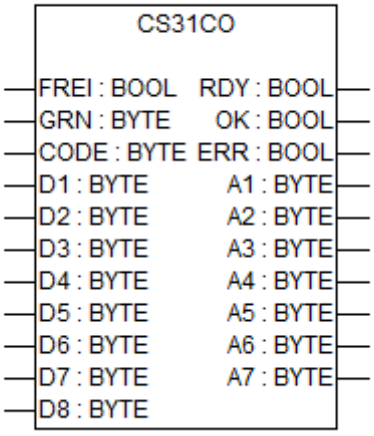
Library file name: **CS31_AC500_Vx.lib**

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

> *The function blocks of the CS31 library can only be called, if the serial interface COM1 is configured as "CS31 bus master".*

1.5.4.10.1 Function blocks

CS31CO



The function block is used to configure the AC31 remote modules. The function block can both send configuration parameters to the remote modules and also scan their currently set configuration.

Parameter	Value
Included in library	CS31_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	Group/Subgroup

The function block is used to configure the AC31 remote modules. The function block can both send configuration parameters to the remote modules and also scan their currently set configuration.

Apart from configuration of the AC31 remote modules, the function block can also process further jobs *“A1...A7” on page 1070*.

Enable for processing a job once is triggered by a FALSE/TRUE edge at input FREI.

The required job identification is specified at input CODE.

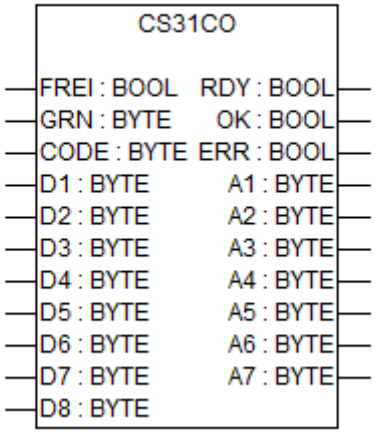
The parameters required for the job are planned at the inputs D1 ... D8.

Status messages are signalized at the outputs RDY, OK and ERR.

The response data of the job are available at the outputs A1 ... A7.

It may take several PLC cycles to process the job.

Input description



FREI

Data type	Default value	Range	Unit
BOOL	-	-	-

Processing of the function block is controlled via input FREI.

FREI = FALSE:

All function block outputs are set to the value "FALSE". However, this is not valid, if a job is currently being processed, i. e. processing of a job which is currently being processed, is not affected by FREI = FALSE.

FREI = FALSE/TRUE edge:

Processing of the job is enabled. Input FREI is no longer evaluated during processing of the job.

FREI = TRUE:

The function block is not processed, i. e. it no longer changes its outputs. However, this is not valid, if a job is currently being processed.

GRN

Data type	Default value	Range	Unit
BYTE	-	0 ... 63	-

Group number with which the remote module is addressed by the PLC program.

Example:

On binary input E 12,08, "12" is the group number and "08" is the channel number.

CODE

Data type	Default value	Range	Unit
BYTE	-	-	-

The identification of the job to be executed is specified at input CODE ↳ *"A1...A7" on page 1070*.

D1...D8

Data type	Default value	Range	Unit
BYTE	-	-	-

The parameters required for the job are preset at the inputs D1 ... D8. The number of parameters depends on the job to be executed. There are also jobs requiring no parameters ↳ *"A1...A7" on page 1070*.

RDY

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RDY indicates that processing of the job currently being processed is completed. This output does not indicate whether processing of the job was successful or not. The output RDY has therefore always to be considered together with the output OK.

RDY = TRUE and OK = TRUE:

Processing of the job is completed without errors. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = TRUE and OK = FALSE:

During processing of the job an error has been detected. ERR is set to TRUE. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = FALSE

Processing of an enabled job has not yet been completed (job is still running) or output RDY has been reset with FREI = FALSE.

OK

Data type	Default value	Range	Unit
BOOL	-	-	-

Output OK indicates whether the job has been handled successfully or whether an error has been detected during processing. In case of an error, OK ist set to FALSE and ERR is set to TRUE. The output OK is not valid until the job has been completed, i. e. if RDY = TRUE.

The following applies:

If RDY = TRUE and

OK = TRUE: The job has been processed successfully.

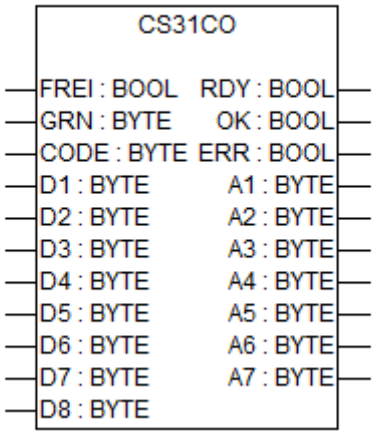
OK = FALSE: During processing of the job an error has been detected.

ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error has occurred during processing of the function block. This output always has to be considered together with the outputs RDY and OK. If RDY is TRUE, OK is FALSE and ERR is TRUE, an error occurred.

Output description



A1...A7

Data type	Default value	Range	Unit
BOOL	-	-	-

After completion of job processing, the response is available at the outputs A1 ... A7. The number of response parameters depends on the job performed is described below.

List of jobs

Processing a job consists of:

- transferring the job
- supplying the OK response or not-OK response

The OK response is described in connection with the corresponding job.

The not-OK response of the individual jobs always looks as follows:

The following basically applies for the not-OK response:

RDY: TRUE

OK: FALSE

ERR:TRUE

A1...A7: 0

Updating of the maximum number of remote modules detected

The input word EW 07,15 contains, amongst other things, the maximum number of remote modules detected in the past. The actual number of remote modules which exist at the moment may be less. This command is used to update this value. The modules which exist are counted and the value is stored. The user can inquire this value in the PLC program (EW 07,15, bit 8...15).

-	Job GRN: CODE: D1...D8:	255 (Master PLC with bus) 132 Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the open-circuit monitoring of an input to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 32 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Inquiring the open-circuit monitoring of an output to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 33 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Deactivating or activating the open-circuit monitoring of an input

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 224. Open-circuit monitoring ON 160. Open-circuit monitoring OFF Channel number Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Deactivating or activating the open-circuit monitoring of an output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 225. Open-circuit monitoring ON 161. Open-circuit monitoring OFF Channel number Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring a channel to determine whether it is configured as input or input/output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 34 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 34. Input 35. Input/output 0

Configuration of a channel as input or input/output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 162. Input 163. Input/output Channel number Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the input delay of a channel

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 38 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms 0

Setting the input delay of a channel

-	Job GRN: CODE: D1: D2:	Group number 0...63 166 Channel number Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Acknowledging errors on remote module

This command can be used to reset the error messages registered on the selected remote module. A reset is possible only if the cause of the error is no longer operative.

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 232 Lowest channel number on the module: 0. Lowest channel number on the module is 0 (<7) 8. Lowest channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Wort: odd number (1, 3, 5) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Acknowledging errors on remote module and resetting configuration values to default setting

In addition to the job "Acknowledging errors on remote module", all configurable settings are reset to the default setting.

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 233 First channel number on the module: 0. First channel number on the module is 0 (<7) 8. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Wort: odd number (1, 3, 5) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the configuration of an analog input

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 42 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 50. Input 0...20 mA 49. Input 4...20 mA 0

Inquiring the configuration of an analog output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 43 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 50. Output 0...20 mA 49. Output 4...20 mA 51. Output +10 V 0

Configuration of an analog input

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 170 Channel number 50. Input 0...20 mA 49. Input 4...20 mA Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Configuration of an analog output

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 171 Channel number 50. Output 0...20 mA 49. Output 4...20 mA 51. Output +10 V Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the bus configuration

The bus interface of the Master PLC has a list which stores specific data of the remote modules. The remote modules are numbered in this list in the order in which they can be found on the CS31 bus. The internal number of the modules must be specified with this command. The response to this command is the group number stored under this number and status information on the corresponding module.

-	Job GRN: CODE: D1: D2...D8:	Not evaluated 80 Number from the module list (1...31) Not used
-	OK response RDY: OK: A1: A2: A3: A4...A7:	TRUE TRUE Status of the remote module: Bits 0...3: Number of process data bytes (binary module) or words (word module), which the module sends to the master. Bits 4...7: Number of process data bytes (binary module) or words (word module), which the master sends to the module Group number Bit 0: 0. Lowest channel number <7 1. Lowest channel number >7 Bit 1: 0. Binary module 1. Word module 0

Read 1 ... 6 bytes

-	Job	Group number 0...63
	GRN:	49. Read 1 byte
	CODE:	50. Read 2 bytes
	D1:	51. Read 3 bytes
	D2:	52. Read 4 bytes
	D3:	53. Read 5 bytes
	D4:	54. Read 6 bytes
	D5...D8:	First channel number on the module:
		0. First channel number on the module is 0 (<7)
		1. First channel number on the module is 8 (>7)
		Module type:
		0. Binary input
		1. Analog input
		2. Binary output
		3. Analog output
		4. Binary input/output
		5. Analog input/output
		Note:
		Bit: even number (0, 2, 4)
		Word: odd number (1, 3, 5)
		Byte start address (Low Byte)
		Byte start address (High Byte)
		Not used
-	OK response	TRUE
	RDY:	TRUE
	OK:	Value of the 1st byte
	A1:	Value of the 2nd byte or 0
	A2:	Value of the 3rd byte or 0
	A3:	Value of the 4th byte or 0
	A4.	Value of the 5th byte or 0
	A5:	Value of the 6th byte or 0
	A6:	0
	A7:	

Read 1 bit from 1 byte

-	Job GRN: CODE: D1: D2: D3: D4: D5: D6...D8:	Group number 0...63 63 First channel number of the module: 0. First channel number of the module is 0 (<7) 1. First channel number of the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Bit position within the byte 0...7 Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE Bit value (0 or 1) 0

Write 1...4 bytes

-	Job GRN: CODE: D1: D2: D3: D4: D5: D6: D7: D8:	Group number 0...63 65. Write 1 byte 66. Write 2 bytes 67. Write 3 bytes 68. Write 4 bytes First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Value of the 1st byte Value of the 2nd byte or not used Value of the 3rd byte or not used Value of the 4th byte or not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

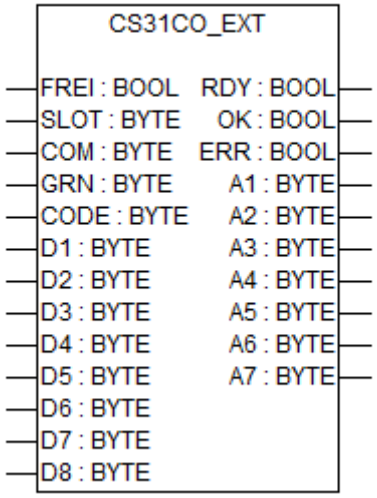
Write 1 bit of 1 byte

-	Job GRN: CODE: D1: D2: D3: D4: D5: D6: D7...D8:	Group number 0...63 79 First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Bit position within the byte 0...7 Bit value (0 or 1) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Function call in ST

```
CS31CO1(FREI := CSCO_FREI,
GRN := CSCO_GRN,CODE := CSCO_CODE,
D1 := CSCO_D1, D2 := CSCO_D2, D3 := CSCO_D3,
D4 := CSCO_D4, D5 := CSCO_D5, D6 := CSCO_D6,
D7 := CSCO_D7, D8 := CSCO_D8);


CSCO_OK:=CS31CO1.OK;
CSCO_ERR:=CS31CO1.ERR;
CSCO_A1:=CS31CO1.A1;
CSCO_A2:=CS31CO1.A2;
CSCO_A3:=CS31CO1.A3;
CSCO_A4:=CS31CO1.A4;
CSCO_A5:=CS31CO1.A5;
CSCO_A6:=CS31CO1.A6;
CSCO_A7:=CS31CO1.A7;
CSCO_RDY:=CS31CO1.RDY;
```

CS31CO_EXT



Function block CS31CO_EXT is used to configure the AC31 Modules. The function block can send configuration parameters to the AC31 Modules and scan their currently set configuration.

Parameter	Value
Included in library	CS31_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Group/Subgroup

Function block CS31CO_EXT is used to configure the AC31 Modules. The function block can send configuration parameters to the AC31 Modules and scan their currently set configuration.

Apart from configuration of the AC31 remote modules, the function block can also process further jobs *“A1...A7” on page 1070*.

Enable for processing a job once is triggered by a FALSE/TRUE edge at input FREI.

The communication port the CS31 Master is connected to is specified at input COM.

The required job identification is specified at input CODE.

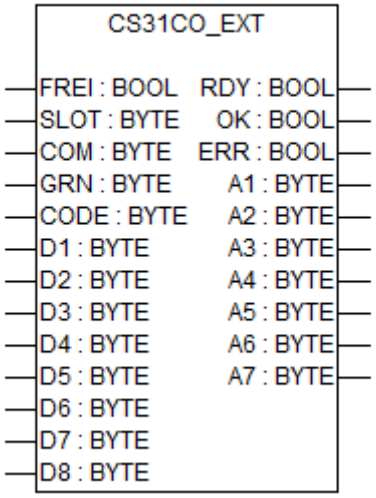
The parameters required for the job are planned at the inputs D1 ... D8.

Status messages are signalized at the outputs RDY, OK and ERR.

The response data of the job are available at the outputs A1 ... A7.

It may take several PLC cycles to process the job.

Input description



FREI

Data type	Default value	Range	Unit
BOOL	-	-	-

Processing of the function block is controlled via input FREI.

FREI = FALSE:

All function block outputs are set to the value "FALSE". However, this is not valid, if a job is currently being processed, i. e. processing of a job which is currently being processed, is not affected by FREI = FALSE.

FREI = FALSE/TRUE edge:

Processing of the job is enabled. Input FREI is no longer evaluated during processing of the job.

FREI = TRUE:

The function block is not processed, i. e. it no longer changes its outputs. However, this is not valid, if a job is currently being processed.

SLOT

Data type	Default value	Range	Unit
BYTE	-	-	-

Reserved for future extensions.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

Communication Port to which the CS31 Bus Module is connected.

GRN

Data type	Default value	Range	Unit
BYTE	-	0 ... 63	-

Group number with which the remote module is addressed by the PLC program.

Example:

On binary input E 12,08, "12" is the group number and "08" is the channel number.

CODE

Data type	Default value	Range	Unit
BYTE	-	-	-

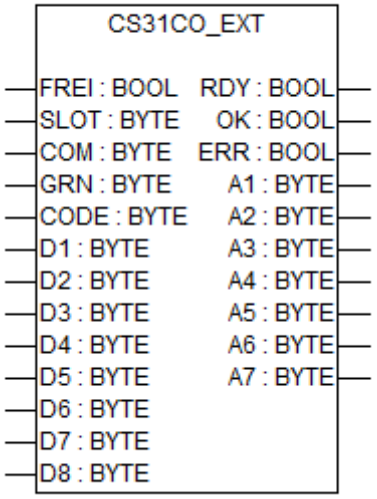
The identification of the job to be executed is specified at input CODE ↳ *"A1...A7" on page 1085*.

D1...D8

Data type	Default value	Range	Unit
BYTE	-	-	-

The parameters required for the job are preset at the inputs D1 ... D8. The number of parameters depends on the job to be executed. There are also jobs requiring no parameters ↳ *"A1...A7" on page 1085*.

Output description



RDY

Data type	Default value	Range	Unit
BOOL	-	-	-

The output RDY indicates that processing of the job currently being processed is completed. This output does not indicate whether processing of the job was successful or not. The output RDY has therefore always to be considered together with the output OK.

RDY = TRUE and OK = TRUE:

Processing of the job is completed without errors. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = TRUE and OK = FALSE:

During processing of the job an error has been detected. ERR is set to TRUE. A new job can be started with a FALSE/TRUE edge at input FREI.

RDY = FALSE

Processing of an enabled job has not yet been completed (job is still running) or output RDY has been reset with FREI = FALSE.

OK

Data type	Default value	Range	Unit
BOOL	-	-	-

Output OK indicates whether the job has been handled successfully or whether an error has been detected during processing. In case of an error, OK ist set to FALSE and ERR is set to TRUE. The output OK is not valid until the job has been completed, i. e. if RDY = TRUE.

The following applies:

If RDY = TRUE and

OK = TRUE: The job has been processed successfully.

OK = FALSE: During processing of the job an error has been detected.

ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error has occurred during processing of the function block. This output always has to be considered together with the outputs RDY and OK. If RDY is TRUE, OK is FALSE and ERR is TRUE, an error occurred.

A1...A7

Data type	Default value	Range	Unit
BOOL	-	-	-

After completion of job processing, the response is available at the outputs A1 ... A7. The number of response parameters depends on the job performed is described below.

List of jobs

Processing a job consists of:

- transferring the job
- supplying the OK response or not-OK response

The OK response is described in connection with the corresponding job.

The not-OK response of the individual jobs always looks as follows:

The following basically applies for the not-OK response:

RDY: TRUE

OK: FALSE

ERR:TRUE

A1...A7: 0

Updating of the maximum number of remote modules detected

The input word EW 07,15 contains, amongst other things, the maximum number of remote modules detected in the past. The actual number of remote modules which exist at the moment may be less. This command is used to update this value. The modules which exist are counted and the value is stored. The user can inquire this value in the PLC program (EW 07,15, bit 8...15).

-	Job GRN: CODE: D1...D8:	255 (Master PLC with bus) 132 Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the open-circuit monitoring of an input to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 32 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Inquiring the open-circuit monitoring of an output to determine whether it is activated or deactivated

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 33 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 47. Open-circuit monitoring ON 32. Open-circuit monitoring OFF 0

Deactivating or activating the open-circuit monitoring of an input

-	Job	Group number 0...63
	GRN:	224. Open-circuit monitoring ON
	CODE:	160. Open-circuit monitoring OFF
	D1:	Channel number
	D2...D8:	Not used
-	OK response RDY:	TRUE
	OK:	TRUE
	A1...A7:	0

Deactivating or activating the open-circuit monitoring of an output

-	Job	Group number 0...63
	GRN:	225. Open-circuit monitoring ON
	CODE:	161. Open-circuit monitoring OFF
	D1:	Channel number
	D2...D8:	Not used
-	OK response	TRUE
	RDY:	TRUE
	OK:	0
	A1...A7:	

Inquiring a channel to determine whether it is configured as input or input/output

-	Job	Group number 0...63
	GRN:	34
	CODE:	Channel number
	D1:	Not used
	D2...D8:	
-	OK response	TRUE
	RDY:	TRUE
	OK:	34. Input
	A1:	35. Input/output
	A2...A7:	0

Configuration of a channel as input or input/output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 162. Input 163. Input/output Channel number Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the input delay of a channel

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 38 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms 0

Setting the input delay of a channel

-	Job GRN: CODE: D1: D2:	Group number 0...63 166 Channel number Input delay 2. 2 ms 4. 4 ms : : 30. 30 ms 32. 32 ms
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Acknowledging errors on remote module

This command can be used to reset the error messages registered on the selected remote module. A reset is possible only if the cause of the error is no longer operative.

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 232 Lowest channel number on the module: 0. Lowest channel number on the module is 0 (<7) 8. Lowest channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Wort: odd number (1, 3, 5) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Acknowledging errors on remote module and resetting configuration values to default setting

In addition to the job "Acknowledging errors on remote module", all configurable settings are reset to the default setting.

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 233 First channel number on the module: 0. First channel number on the module is 0 (<7) 8. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Wort: odd number (1, 3, 5) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the configuration of an analog input

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 42 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 50. Input 0...20 mA 49. Input 4...20 mA 0

Inquiring the configuration of an analog output

-	Job GRN: CODE: D1: D2...D8:	Group number 0...63 43 Channel number Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE 50. Output 0...20 mA 49. Output 4...20 mA 51. Output +10 V 0

Configuration of an analog input

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 170 Channel number 50. Input 0...20 mA 49. Input 4...20 mA Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Configuration of an analog output

-	Job GRN: CODE: D1: D2: D3...D8:	Group number 0...63 171 Channel number 50. Output 0...20 mA 49. Output 4...20 mA 51. Output +10 V Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Inquiring the bus configuration

The bus interface of the Master PLC has a list which stores specific data of the remote modules. The remote modules are numbered in this list in the order in which they can be found on the CS31 bus. The internal number of the modules must be specified with this command. The response to this command is the group number stored under this number and status information on the corresponding module.

-	Job GRN: CODE: D1: D2...D8:	Not evaluated 80 Number from the module list (1...31) Not used
-	OK response RDY: OK: A1: A2: A3: A4...A7:	TRUE TRUE Status of the remote module: Bits 0...3: Number of process data bytes (binary module) or words (word module), which the module sends to the master. Bits 4...7: Number of process data bytes (binary module) or words (word module), which the master sends to the module Group number Bit 0: 0. Lowest channel number <7 1. Lowest channel number >7 Bit 1: 0. Binary module 1. Word module 0

Read 1 ... 6 bytes

-	Job	Group number 0...63
	GRN:	49. Read 1 byte
	CODE:	50. Read 2 bytes
	D1:	51. Read 3 bytes
	D2:	52. Read 4 bytes
	D3:	53. Read 5 bytes
	D4:	54. Read 6 bytes
	D5...D8:	First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Not used
-	OK response	TRUE
	RDY:	TRUE
	OK:	Value of the 1st byte
	A1:	Value of the 2nd byte or 0
	A2:	Value of the 3rd byte or 0
	A3:	Value of the 4th byte or 0
	A4.	Value of the 5th byte or 0
	A5:	Value of the 6th byte or 0
	A6:	0
	A7:	

Read 1 bit from 1 byte

-	Job GRN: CODE: D1: D2: D3: D4: D5: D6...D8:	Group number 0...63 63 First channel number of the module: 0. First channel number of the module is 0 (<7) 1. First channel number of the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Bit position within the byte 0...7 Not used
-	OK response RDY: OK: A1: A2...A7:	TRUE TRUE Bit value (0 or 1) 0

Write 1...4 bytes

-	Job	Group number 0...63
	GRN:	65. Write 1 byte
	CODE:	66. Write 2 bytes
	D1:	67. Write 3 bytes
	D2:	68. Write 4 bytes
	D3:	First channel number on the module:
	D4:	0. First channel number on the module is 0 (<7)
	D5:	1. First channel number on the module is 8 (>7)
	D6:	Module type:
	D7:	0. Binary input
	D8:	1. Analog input
		2. Binary output
		3. Analog output
		4. Binary input/output
		5. Analog input/output
		Note:
		Bit: even number (0, 2, 4)
		Word: odd number (1, 3, 5)
		Byte start address (Low Byte)
		Byte start address (High Byte)
		Value of the 1st byte
		Value of the 2nd byte or not used
		Value of the 3rd byte or not used
		Value of the 4th byte or not used
-	OK response	TRUE
	RDY:	TRUE
	OK:	0
	A1...A7:	

Write 1 bit of 1 byte

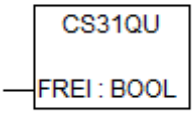
-	Job GRN: CODE: D1: D2: D3: D4: D5: D6: D7...D8:	Group number 0...63 79 First channel number on the module: 0. First channel number on the module is 0 (<7) 1. First channel number on the module is 8 (>7) Module type: 0. Binary input 1. Analog input 2. Binary output 3. Analog output 4. Binary input/output 5. Analog input/output Note: Bit: even number (0, 2, 4) Word: odd number (1, 3, 5) Byte start address (Low Byte) Byte start address (High Byte) Bit position within the byte 0...7 Bit value (0 or 1) Not used
-	OK response RDY: OK: A1...A7:	TRUE TRUE 0

Function call in ST

```
CS31COEXT   (FREI := CS31COEXT_FREI,
             GRN  := CS31COEXT_GRN,
             CODE := CS31COEXT_CODE,
             D1   := CS31COEXT_D1,
             D2   := CS31COEXT_D2,
             D3   := CS31COEXT_D3,
             D4   := CS31COEXT_D4,
             D5   := CS31COEXT_D5,
             D6   := CS31COEXT_D6,
             D7   := CS31COEXT_D7,
             D8   := CS31COEXT_D8);


CS31COEXT_OK   :=CS31COEXT.OK;
CS31COEXT_ERR  :=CS31COEXT.ERR;
CS31COEXT_A1   :=CS31COEXT.A1;
CS31COEXT_A2   :=CS31COEXT.A2;
CS31COEXT_A3   :=CS31COEXT.A3;
```

```
CS31COEXT_A4   :=CS31COEXT.A4;
CS31COEXT_A5   :=CS31COEXT.A5;
CS31COEXT_A6   :=CS31COEXT.A6;
CS31COEXT_A7   :=CS31COEXT.A7;
CS31COEXT_RDY  :=CS31COEXT.RDY;
```

CS31QU



This function block allows to acknowledge automatically error messages of AC31 remote modules. Error messages are stored on the AC31 remote modules until they are acknowledged. Even if the error has been removed, the error message is still pending on the module until acknowledgement and is also signalized to the PLC until the message is acknowledged.

Processing of the function block is enabled with a TRUE signal at input FREI, and the function block then acknowledges AC31 errors continuously.

It may take several PLC cycles to acknowledge an error on an AC31 module.

If the function block is enabled, it constantly checks whether an AC31 error of class 3 or 4 has occurred and acknowledges this error.

An AC31 error of class 3 has occurred:

The function block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,13 is reset and LED FK3 is deactivated.

Example of a FK3 error:

A remote module is disconnected from the CS31 bus.

An AC31 error of class 4 has occurred:

The function block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,14 is reset.

Example of a FK4 error:

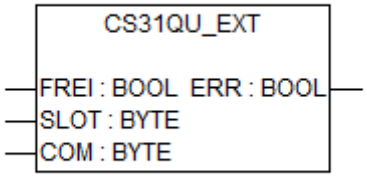
A remote module signalizes an open circuit.

Parameter	Value
Included in library	CS31_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	Group/Subgroup

Function call in ST

```
CS31QU1(FREI := CSQU_FREI);
```

CS31QU_EXT



Function block CS31QU_EXT allows to acknowledge automatically error messages of AC31 Modules.

Parameter	Value
Included in library	CS31_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Group/Subgroup

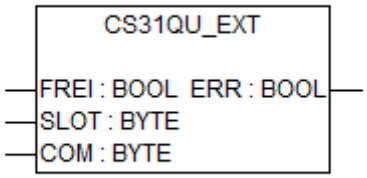
This function block allows to acknowledge automatically error messages of AC31 remote modules. Error messages are stored on the AC31 remote modules until they are acknowledged. Even if the error has been removed, the error message is still pending on the module until acknowledgement and is also signalized to the PLC until the message is acknowledged.

Processing of the function block is enabled with a TRUE signal at input FREI, and the function block then acknowledges AC31 errors continuously.

It may take several PLC cycles to acknowledge an error on an AC31 module.

If the function block is enabled, it constantly checks whether an AC31 error of class 3 or 4 has occurred and acknowledges this error.

Input description



FREI

Data type	Default value	Range	Unit
BOOL	-	-	-

Processing of the function block is controlled via input FREI.

FREI = FALSE:

All function block outputs are set to the value "FALSE". However, this is not valid, if a job is currently being processed, i. e. processing of a job which is currently being processed, is not affected by FREI = FALSE.

FREI = FALSE/TRUE edge:

Processing of the job is enabled. Input FREI is no longer evaluated during processing of the job.

FREI = TRUE:

The function block is not processed, i. e. it no longer changes its outputs. However, this is not valid, if a job is currently being processed.

SLOT

Data type	Default value	Range	Unit
BYTE	-	-	-

Reserved for future extensions.

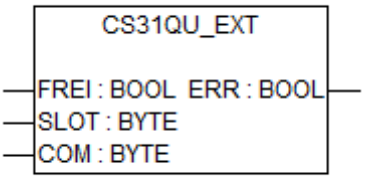
COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At input COM, the number of the serial interface of the CS31 Master (AC500 or CM574) is specified.

Valid values: 0 and 1 for COM1, 2 for COM2 (CM574-RS only)

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error occurred during Function Block processing.

An AC31 error of class 3 has occurred:

The function block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,13 is reset and LED FK3 is deactivated.

Example of a FK3 error:

A remote module is disconnected from the CS31 bus.

An AC31 error of class 4 has occurred:

The function block acknowledges the error on the AC31 remote module which signalizes the error and also clears the error message on the PLC, i.e. the error flag M 255,14 is reset.

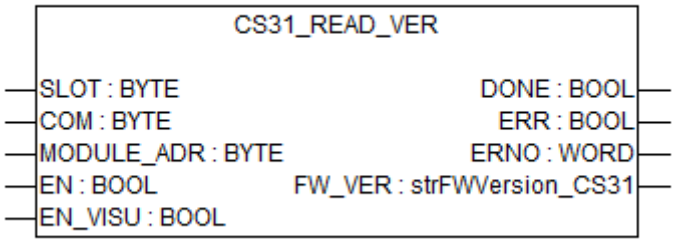
Example of a FK4 error:

A remote module signalizes an open circuit.

Function call in ST

```
CS31QUEXT      (FREI := CS31QUEXT_FREI,
                SLOT := CS31QUEXT_SLOT,
                COM  := CS31QUEXT_COM);


CS31QUEXT_ERR        := CS31QUEXT.ERR;
```

CS31_READ_VER



Function block CS31_READ_VER is used to read out the firmware version of CS31 Bus Modules.

Parameter	Value
Included in library	CS31_AC500_V20.lib
Available as of firmware	V2.0.0
Type	Function block with historical values
Group	Group/Subgroup

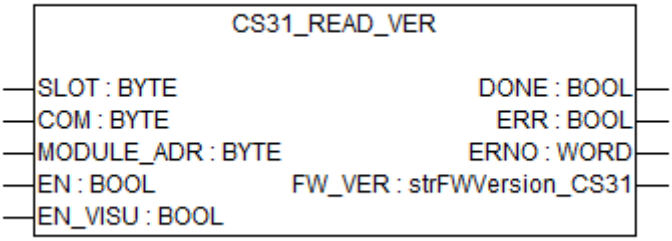
Function block CS31_READ_VER is used to readout the firmware version of CS31 communication interface module and of the Expansion Modules connected to them. The result is returned in a structure of the type strFWVersion_CS31.

The structure strFWVersion_CS31 is composed as follows:

TYPE strFWVersion_CS31 : STRUCT			
	abyFW_BM:	ARRAY[0..4] OF BYTE;	(* 5 bytes: FW version of CS31 Bus Module*)
	abyFW_Ext_1:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 1*)
	abyFW_Ext_2:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 2*)
	abyFW_Ext_3:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 3*)
	abyFW_Ext_4:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 4*)
	abyFW_Ext_5:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 5*)

	abyFW_Ext_6:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 6*)
	abyFW_Ext_7:	ARRAY[0..3] OF BYTE;	(* 4 bytes: FW version of Expansion Module 7*)
END_STRUCT END_TYPE			

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	-	-	-

Reserved for future extensions.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

Communication Port to which the CS31 Bus Module is connected.

MODULE_ADR

Data type	Default value	Range	Unit
BYTE	-	0 ... 99	-

Address of the CS31 Bus Module from which the firmware versions should be readout.

EN

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated with a value transition from FALSE->TRUE at the variable EN. Otherwise it remains disabled.

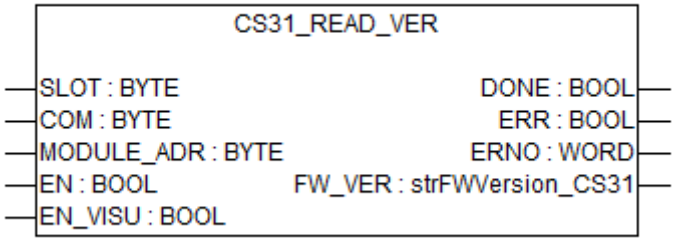
If the function block is activated the specified input values are used and the output values are available at the function block outputs.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except EN and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled. The actual values are always displayed.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

FW_VER

Data type	Default value	Range	Unit
strFWVersion_CS31	-	-	-

In the structure of this output value the firmware versions of the CS31 communication interface module and its expansion modules is stored.

Function call in ST

```
CS31ReadVer (SLOT       := CS31ReadVer_SLOT,
             COM        := CS31ReadVer_COM,
             MODULE_ADR := CS31ReadVer_MODULE_ADR,
             EN         := CS31ReadVer_EN,
             EN_VISU    := CS31ReadVer_EN_VISU);


CS31ReadVer_DONE   := CS31ReadVer.DONE;
CS31ReadVer_ERR    := CS31ReadVer.ERR;
CS31ReadVer_ERNO   := CS31ReadVer.ERNO;
CS31ReadVer_FW_VER := CS31ReadVer.FW_VER;
```

1.5.4.11 DC541 library

1.5.4.11.1 DC541

Library file name: **DC541_AC500_Vx.lib**

This library contains all function blocks necessary for using DC541. The configuration of DC541 is done in PLC configuration.

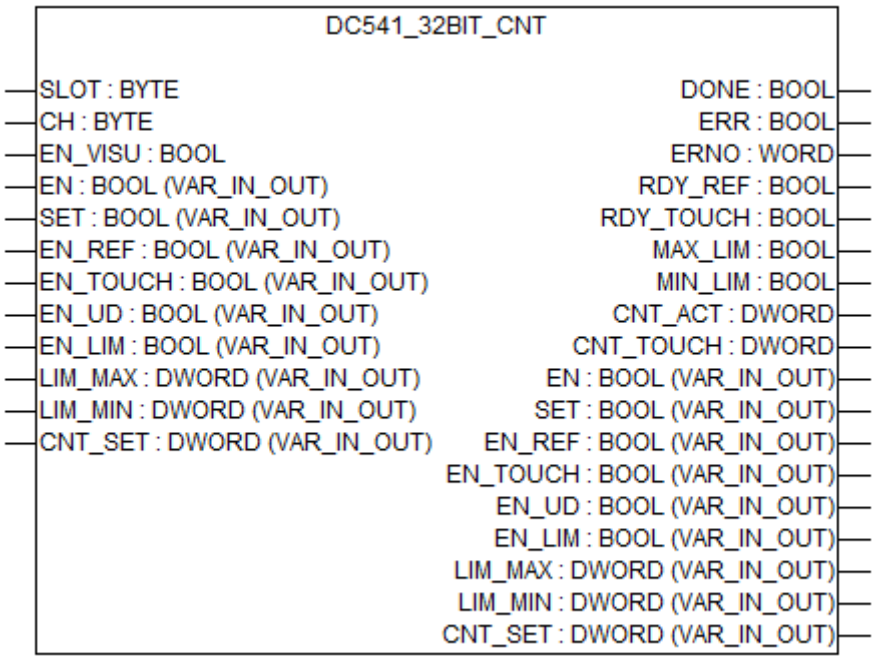
Once DC541 has been added to the PLC configuration, the library is automatically included with the next compilation of the project.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

The function blocks are available in AC500 control systems with a runtime system of version V1.1.2 or later.

Function blocks

DC541_32BIT_CNT



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3
Available as of firmware DC541	V1.1

Parameter	Value
Type	Function block with historical values
Group	Counters

The 32-bit counter is a count up/count down counter with a directional discriminator. The counter can be used in two counting modes:

- EN_UD = FALSE: Encoder mode
  Connection of an incremental transmitter (track A / track B, offset by 90°)
  It is possible to count signals up to approx. 60 kHz. This corresponds to a motor with a rotational speed of 3.600 rpm and a transmitter with 1.000 pulses per rotation. Pulse multiplication (x2 or x4) is not used.
- EN_UD = TRUE: Up / down mode
  Up-/down counter
  It is possible to count signals up to approx. 60 kHz. Count-up for signals on channel C1, count-down for signals on channel C0.

The counter always uses the channels C0...C3 of the DC541:

- C0: Track A of the incremental transmitter.
- C1: Track B of the incremental transmitter.
- C2 and C3: Reference cam or touch trigger.

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected using input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32 bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, a corresponding error message is displayed at the outputs ERR/ERNO. In this case, the values for LIM_MIN and LIM_MAX are not forwarded to the DC541. The difference between LIM_MAX and LIM_MIN has to be at least twice the number (frequency) of counting pulses per DC541 cycle.

Example

Number of counting pulses (frequency) = 40 kHz = 40000 increments/s = 40 increments/ms

Cycle time of the DC541 = 100 µs

LIM_MIN = 0

Number of counting pulses per DC541 cycle: 40 increments/ms = 4 increments/100 µs

LIM_MAX > 8

Using input SET, the counter is set to the value CNT_SET. This value is kept as long as input SET = TRUE.

If the reference point approach is enabled at input EN_REF, the counter is set to the value of input CNT_SET when a rising edge occurs on channel C2 or C3.

Using input EN_TOUCH, a touch trigger measurement is enabled. This means: With the rising edge on channel C2 or C3, the counting value is stored and displayed at output CNT_TOUCH. The validity of CNT_TOUCH is indicated by output RDY_TOUCH. This functionality can be used to determine the counter value with regard to an external event. The results are increment accurate.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a corresponding error message is displayed at the outputs ERR/ERNO.

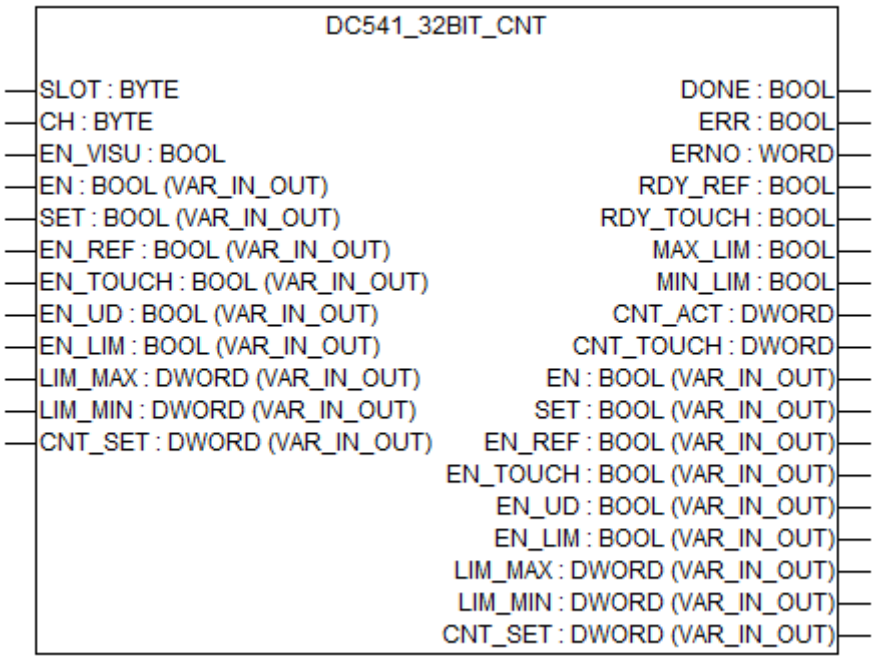
To initiate a new reference point approach or touch trigger measurement, a positive edge at the corresponding enabling input is necessary.

If the zero track of an incremental transmitter is wired to channel C2 or C3, no touch trigger measurement may be performed in the region of the reference cam!

The device DC541 must be configured as counting device (counter mode).

This function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE ↳ *Chapter 1.5.4.11.1.2 “Visualizations” on page 1154*.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	0 for input C0 only	-

This input is used to select the input for the counter. The device occupies the inputs C0...C3. If an invalid value is entered at input CH or if the selected channel is not configured as a 32-bit counter (32BIT_CNT), the function block is aborted.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

SET

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

At a rising edge at this input, the counter is set to the value of input CNT_SET. Counting is enabled after reset process to SET = TRUE.

EN_REF

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

A rising edge at EN_REF input (enable reference) enables the reference point approach. If EN_REF = TRUE, a rising edge at input C2 or C3 causes the actual value of the counter CNT_ACT to be set to the value of input CNT_SET.

The next measurement is again initiated by a rising edge at input EN_REF.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, the function block is aborted.

EN_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

A rising edge at EN_TOUCH input (enable touch trigger) enables a touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at input C2 or C3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

The next measurement is again initiated by a rising edge at input EN_TOUCH.

Only one function may be enabled at a time, either the reference point approach or the touch trigger measurement. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, the function block is aborted.

EN_UD

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The input EN_UD (enable up/down) is used to select the counting mode of the 32-bit counter.

EN_UD = FALSE: Connection of an incremental transmitter. Track A/B offset by 90°.

EN_UD = TRUE: Up/down counter. C1: count up, C0: count down.

EN_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Input EN_LIM (enable limit) is used to set the operating mode for the counter:

EN_LIM = FALSE: Infinite counter (endless mode).

EN_LIM = TRUE: Limiting counter (limit mode).

Switching between the operating modes can be done during running operation. And it is possible to change the upper limit value LIM_MAX and the lower limit value LIM_MIN during running operation. LIM_MAX has to be higher than LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

In case of a rising edge at input EN_LIM (transition from endless mode to limit mode), the present counter value can change depending on the values of LIM_MIN and LIM_MAX.

Present counter value	Changed counter value
ACT_CNT < LIM_MIN	ACT_CNT := LIM_MIN
LIM_MIN ≤ ACT_CNT ≤ LIM_MAX	ACT_CNT := ACT_CNT (no change)
ACT_CNT > LIM_MAX	ACT_CNT := LIM_MIN

LIM_MAX

Data type	Default value	Range	Unit
DWORD	-	-	-

LIM_MAX (limit maximum) is used to set the upper limit value for the counter. LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

LIM_MIN

Data type	Default value	Range	Unit
DWORD	-	-	-

LIM_MIN (limit minimum) is used to set the lower limit value for the counter. The upper limit value LIM_MAX has to be higher than LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

CNT_SET

Data type	Default value	Range	Unit
DWORD	-	-	-

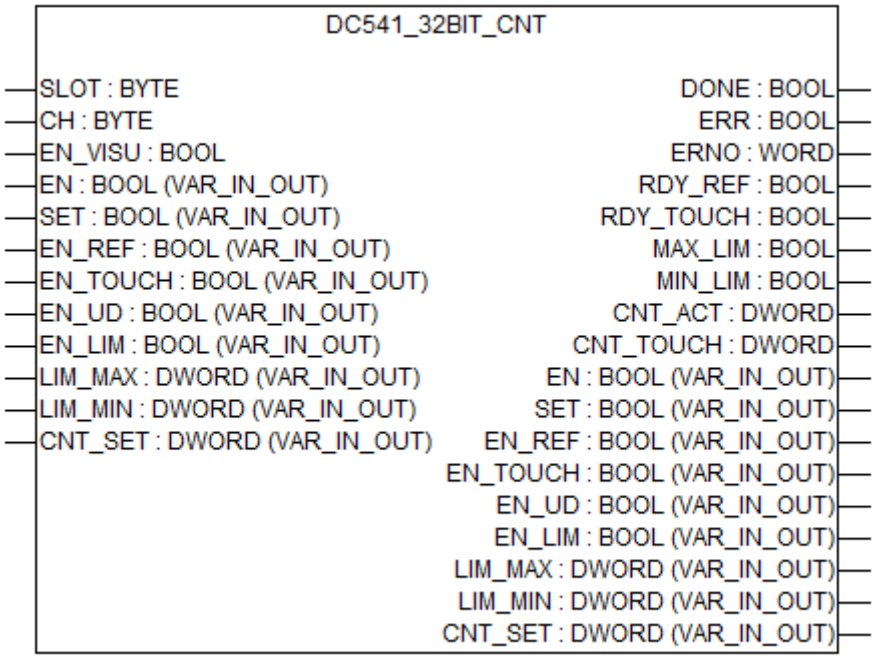
CNT_SET (counter set) is used to adjust the counter set value.

If input SET = TRUE, the counter is set to the value of input CNT_SET and remains at this value as long as input SET = TRUE. Counting is enabled again with the occurrence of the falling edge at input SET.

In case of a reference point approch (EN_REF = TRUE), the rising edge on channel C2 causes the actual value of the counter (ACT_CNT) to be adjusted to the set value CNT_SET.

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE. The function block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 58: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

RDY_REF

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

RDY_REF (ready reference) displays the ready message of the reference point approach.

RDY_TOUCH

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

RDY_TOUCH (ready message touch trigger) displays the ready message of the touch trigger measurement.

MAX_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

In the infinite counter operating mode (endless mode), MAX_LIM indicates whether the actual counter value is higher than the value set at input LIM_MAX.

In the limiting counter operating mode (limit mode), output MAX_LIM is FALSE.

MIN_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

In the infinite counter operating mode (endless mode), output MIN_LIM indicates whether the actual counter value is lower than the value set at input LIM_MIN.

In the limiting counter operating mode (limit mode), output MIN_LIM is FALSE.

CNT_ACT

Data type	Default value	Range	Unit
DWORD	-	-	-

CNT_ACT (counter actual) displays the actual counter value of the count up counter.

CNT_TOUCH

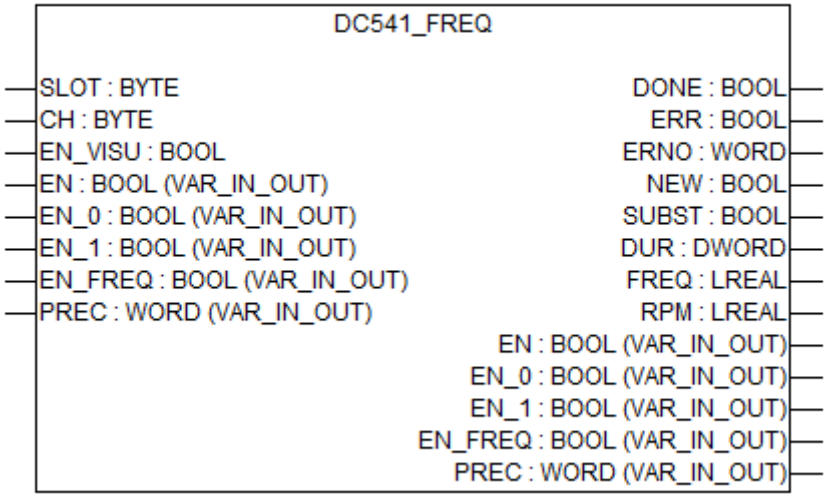
Data type	Default value	Range	Unit
DWORD	-	-	-

Output CNT_TOUCH (counter touch trigger) displays the result of the touch trigger measurement, i.e. the actual counter value at the moment of the occurrence of the rising edge at input C2 after initiation of a touch trigger measurement by a rising edge at input EN_TOUCH.

Function call in ST

```
DC541Cnt32Bit (SLOT := DC541Cnt32Bit_SLOT,
      CH      := DC541Cnt32Bit_CH,
      EN_VISU := DC541Cnt32Bit_EN_VISU,
      EN      := DC541Cnt32Bit_EN,
      SET     := DC541Cnt32Bit_SET,
      EN_REF  := DC541Cnt32Bit_EN_REF,
      EN_TOUCH:= DC541Cnt32Bit_EN_TOUCH,
      EN_UD   := DC541Cnt32Bit_EN_UD,
      EN_LIM  := DC541Cnt32Bit_EN_LIM,
      LIM_MAX := DC541Cnt32Bit_LIM_MAX,
      LIM_MIN := DC541Cnt32Bit_LIM_MIN,
      CNT_SET := DC541Cnt32Bit_CNT_SET);



DC541Cnt32Bit_DONE      := DC541Cnt32Bit.DONE;
DC541Cnt32Bit_ERR       := DC541Cnt32Bit.ERR;
DC541Cnt32Bit_ERNO      := DC541Cnt32Bit.ERNO;
DC541Cnt32Bit_RDY_REF   := DC541Cnt32Bit.RDY_REF;
DC541Cnt32Bit_RDY_TOUCH := DC541Cnt32Bit.RDY_TOUCH;
DC541Cnt32Bit_MAX_LIM   := DC541Cnt32Bit.MAX_LIM;
DC541Cnt32Bit_MIN_LIM   := DC541Cnt32Bit.MIN_LIM;
DC541Cnt32Bit_CNT_ACT   := DC541Cnt32Bit.CNT_ACT;
DC541Cnt32Bit_CNT_TOUCH := DC541Cnt32Bit.CNT_TOUCH;
```

DC541_FREQ



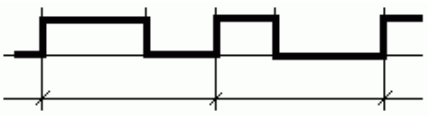
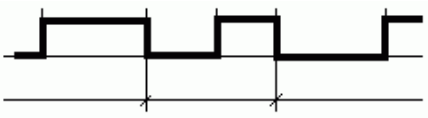
Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.2
Type	Function block with historical values
Group	Counters

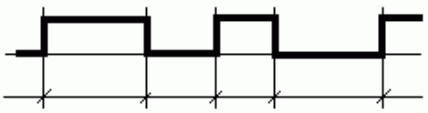
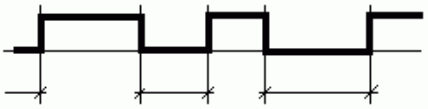
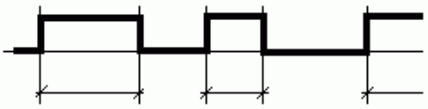
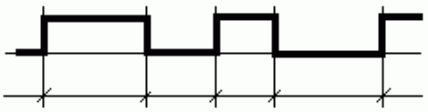
The function block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

It is able to measure frequencies from 0 to 2000 Hz (2 kHz). In order to obtain a precise measurement of frequencies > 50 Hz, a correspondingly high accuracy setting has to be chosen. It is recommended to use an accuracy of PREC = 1000, i.e. 0.001.

This function block has to be called cyclically, one time per second at least.

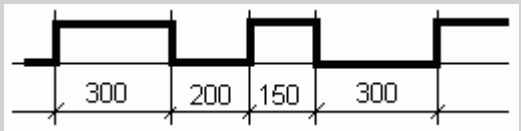
The inputs EN_0, EN_1 and EN_FREQ are used to determine the edges to be measured. If input EN_FREQ = TRUE, the frequency and the rotational speed are calculated in addition to the time measurement.

EN_0	EN_1	EN_FREQ	Edges measured	FREQ/RPM
FALSE	FALSE	TRUE	No measurement is performed.	yes
FALSE	TRUE	TRUE	Measurement of time between two rising edges. 	yes
TRUE	FALSE	TRUE	Measurement of time between two falling edges. 	yes

EN_0	EN_1	EN_FREQ	Edges measured	FREQ/RPM
TRUE	TRUE	TRUE	Measurement of time between any two edges. 	yes
FALSE	FALSE	FALSE	No measurement is performed.	no
FALSE	TRUE	FALSE	Measurement of time between the falling edge and the subsequent rising edge. 	no
TRUE	FALSE	FALSE	Measurement of time between the rising edge and the subsequent falling edge. 	no
TRUE	TRUE	FALSE	Measurement of time between any two edges. 	no

Example

Different time measurement results depending on the values applied to the inputs EN_0, EN_1 and EN_FREQ.



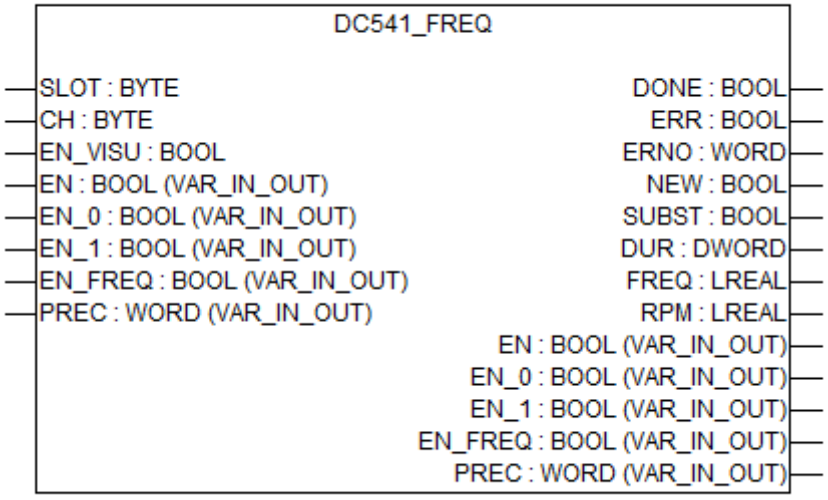
EN_0	EN_1	EN_FREQ	Time measurement (duration DUR) [µs]			
			1	2	3	4
FALSE	FALSE	TRUE	0	0	0	0
FALSE	TRUE	TRUE	-	500	-	450
TRUE	FALSE	TRUE	-	-	350	-
TRUE	TRUE	TRUE	300	200	150	300
FALSE	FALSE	FALSE	0	0	0	0
FALSE	TRUE	FALSE	300	-	150	-
TRUE	FALSE	FALSE	-	200	-	300
TRUE	TRUE	FALSE	300	200	150	300

The output NEW indicates that new measurement results are available.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency measurement.

This function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE *Chapter 1.5.4.11.1.2 “Visualizations” on page 1154*.

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	0-7 for the inputs C0-C7	-

CH is used to select the channel for time and frequency measurement. If an invalid value is specified at input CH or if the selected channel is not configured as frequency measurement, the function flock is aborted with DONE = ERR = TRUE and an error is displayed.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

EN_0 (enable 0)

Data type	Default value	Range	Unit
BOOL	-	-	-

This input is used to determine whether falling edges are considered for measurement:

EN_0 = TRUE: Measurement considers falling edges.

EN_0 = FALSE: Measurement does not consider falling edges.

EN_1

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

If EN_1=TRUE, the time frequency measurement will be capture on rising edge of signal.

Input EN_1 corresponds to output bit 2 in control byte.

EN_FREQ

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

EN_FREQ = FALSE: Measurement of time between positive or negative edges

EN_FREQ = TRUE: Measurement of time between edges and calculation of frequency and speed of rotation

PREC

Data type	Default value	Range	Unit
WORD	-	-	-

PREC (precision) input is used to specify the demanded accuracy of the measurement.

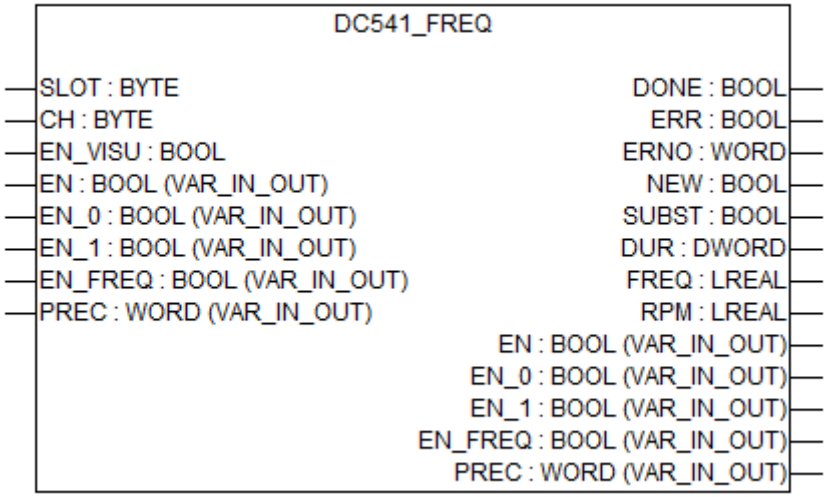
PREC = 10 corresponds to an accuracy of 0.1 (one tenth)

PREC = 100 corresponds to an accuracy of 0.01 (one hundredth)

PREC = 1000 corresponds to an accuracy of 0.001 (one thousandth)

Depending on the cycle time and the demanded accuracy, the DC541 extends its measuring time and counts multiple pulses. This setting is only used in the frequency measurement operating mode (EN_FREQ = TRUE). Then, the displayed measurement values correspond to the measured average value. This reduces jitter and latency effects resulting in improved accuracy.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 59: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

NEW

Data type	Default value	Range	Unit
BOOL	-	-	-

Output indicates that new measurement results are available.

NEW = TRUE: New measurement results are available.

NEW = FALSE: No new measurement results are available.

SUBST

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

SUBST (substitute) output indicates whether the output values are based on new measurement values or calculated using substitute values. Without the substitute value, the function block would still display a short time and a high speed of rotation in case of a decrease of the frequency, since no new measurement is initiated without a new signal edge. If SUBST = TRUE, the extended distance between the edges is extrapolated. PREC > 0 is an assumption for the calculation using a substitute value.

SUBST = FALSE: New measurement values.

SUBST = TRUE: Calculation using substitute values.

DUR

Data type	Default value	Range	Unit
DWORD	-	-	µs

DUR (duration) output displays the measured time.

FREQ (frequency)

Data type	Default value	Range	Unit
LREAL	-	-	Hz

If input EN_FREQ = TRUE, output FREQ displays the frequency calculated from the measured time.

If input EN_FREQ = FALSE, output FREQ = 0.

RPM

Data type	Default value	Range	Unit
LREAL	-	-	rpm

If input EN_FREQ = TRUE, output RPM (revolutions per minute) displays the speed of rotation calculated from the measured time.

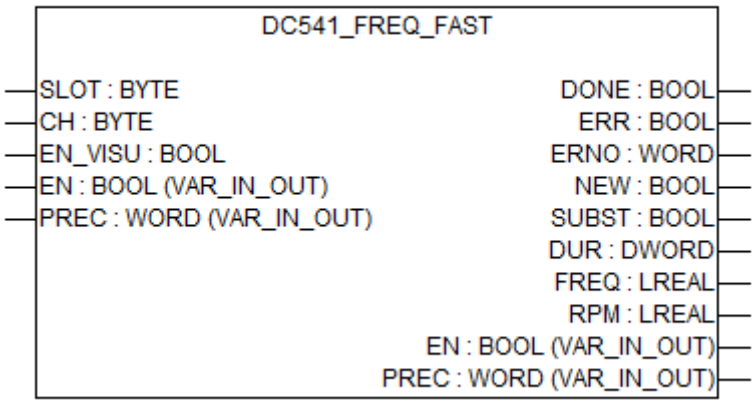
If input EN_FREQ = FALSE, output RPM =0.

Function call in ST

```
DC541Freq(SLOT      := DC541Freq_SLOT,
          CH        := DC541Freq_CH,
          EN_VISU   := DC541Freq_EN_VISU,
          EN        := DC541Freq_EN,
          EN_0      := DC541Freq_EN_0,
          EN_1      := DC541Freq_EN_1,
          EN_FREQ   := DC541Freq_EN_FREQ,
          PREC      := DC541Freq_PREC);


DC541Freq_DONE      := DC541Freq.DONE;
```

```
DC541Freq_ERR       := DC541Freq.ERR;
DC541Freq_ERNO      := DC541Freq.ERNO;
DC541Freq_NEW       := DC541Freq.NEW;
DC541Freq_SUBST     := DC541Freq.SUBST;
DC541Freq_DUR       := DC541Freq.DUR;
DC541Freq_FREQ      := DC541Freq.FREQ;
DC541Freq_RPM       := DC541Freq.RPM;
```

DC541_FREQ_FAST



Parameter	Value
Included in library	DC541_AC500_V12.lib
Available as of firmware	V1.2, DC541 firmware: V1.3
Type	Function block with historical values
Group	Counters

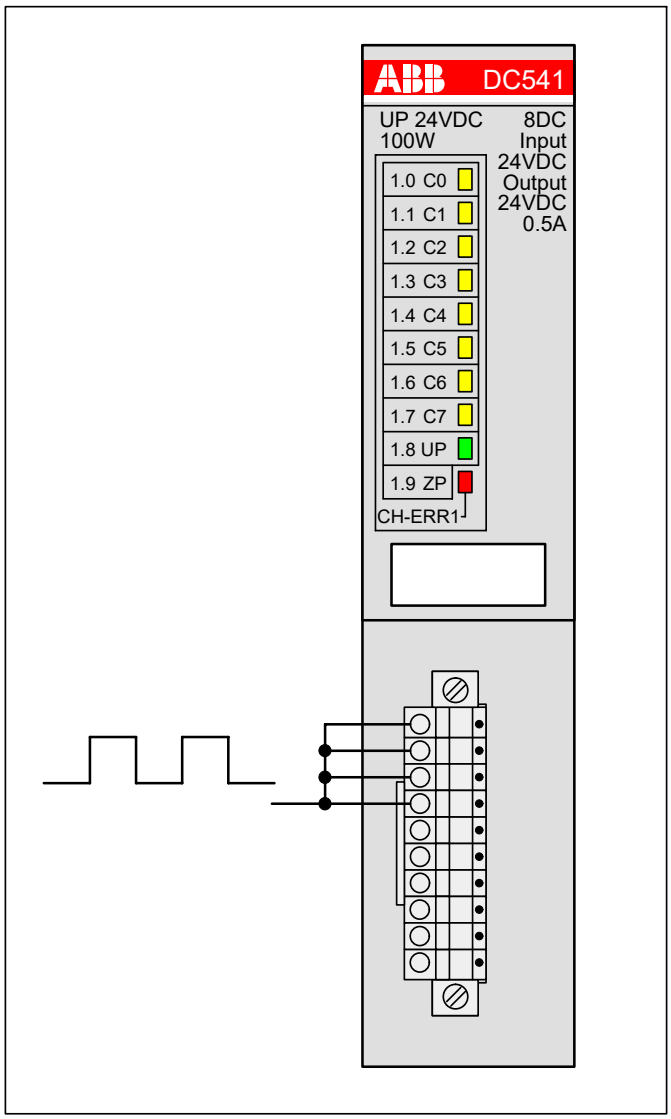
The function block DC541_FREQ_FAST is used to measure times, frequencies and rotational speeds with a resolution of 1 µs.

It is able to measure frequencies from 0-50000 Hz (50 kHz).

The function block works like DC541_FREQ, but it reaches a resolution of 1us ↳ *Chapter 1.5.4.11.1.1.2 “DC541_FREQ” on page 1111*.

Restrictions:

- The function block is only allowed once per DC541 and only for channel 0.
- It occupies the resources of the first 4 inputs, which cannot be used for other purposes.
- The inputs C0..C3 must be connected in parallel.



This function block has to be called cyclically, one time per second at least.

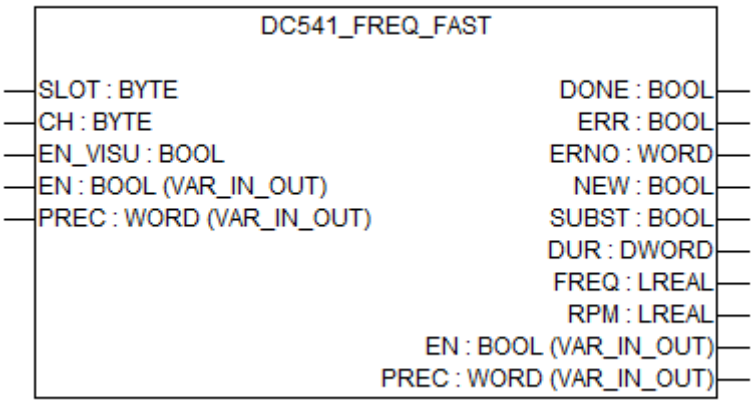
It always measures the time and frequency from one positive edge to the next positive edge.

The output NEW indicates that new measurement results are available.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency measurement.

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE ↳ *Chapter 1.5.4.11.1.2 “Visualizations” on page 1154*.

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	0-7 for the inputs C0-C7	-

CH is used to select the channel for time and frequency measurement. If an invalid value is specified at input CH or if the selected channel is not configured as frequency measurement, the function flock is aborted with DONE = ERR = TRUE and an error is displayed.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

PREC

Data type	Default value	Range	Unit
WORD	-	-	-

PREC (precision) input is used to specify the demanded accuracy of the measurement.

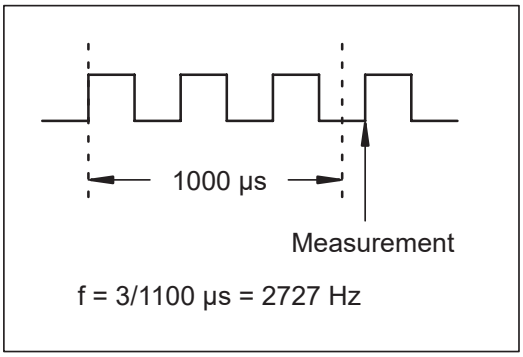
PREC = 10 corresponds to an accuracy of 0.1 (one tenth)

PREC = 100 corresponds to an accuracy of 0.01 (one hundredth)

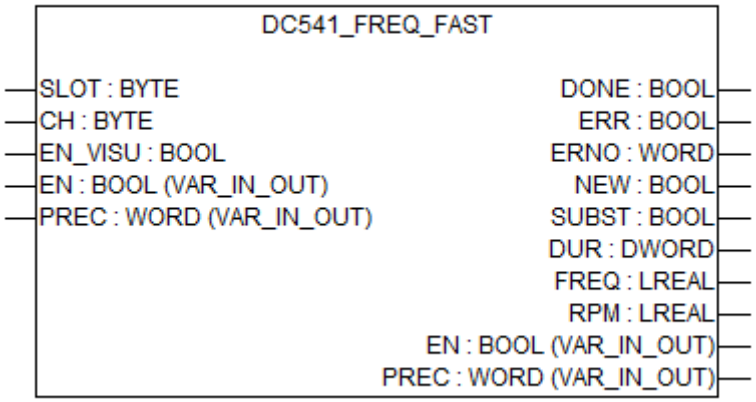
PREC = 1000 corresponds to an accuracy of 0.001 (one thousandth)

Depending on the cycle time and the demanded accuracy, the DC541 extends its measuring time and counts multiple pulses. This setting is only used in the frequency measurement operating mode (EN_FREQ = TRUE). Then, the displayed measurement values correspond to the measured average value. This reduces jitter and latency effects resulting in improved accuracy.

As the resolution for measuring is 1 µs, the precision corresponds directly to the measuring time in µs. For times < 100 µs, the device will just calculate the values as fast as possible, which will be the cycle time of the device, around 100 µs. For higher precision, the device will use the pulses per measuring time. It will measure the pulses and the time, with the time on a 1µs resolution.



Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 60: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

NEW

Data type	Default value	Range	Unit
BOOL	-	-	-

Output indicates that new measurement results are available.

NEW = TRUE: New measurement results are available.

NEW = FALSE: No new measurement results are available.

SUBST

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

SUBST (substitute) output indicates whether the output values are based on new measurement values or calculated using substitute values. Without the substitute value, the function block would still display a short time and a high speed of rotation in case of a decrease of the frequency, since no new measurement is initiated without a new signal edge. If SUBST = TRUE, the extended distance between the edges is extrapolated. PREC > 0 is an assumption for the calculation using a substitute value.

SUBST = FALSE: New measurement values.

SUBST = TRUE: Calculation using substitute values.

DUR

Data type	Default value	Range	Unit
DWORD	-	-	µs

DUR (duration) output displays the measured time.

FREQ (frequency)

Data type	Default value	Range	Unit
LREAL	-	-	Hz

If input EN_FREQ = TRUE, output FREQ displays the frequency calculated from the measured time.

If input EN_FREQ = FALSE, output FREQ = 0.

RPM

Data type	Default value	Range	Unit
LREAL	-	-	rpm

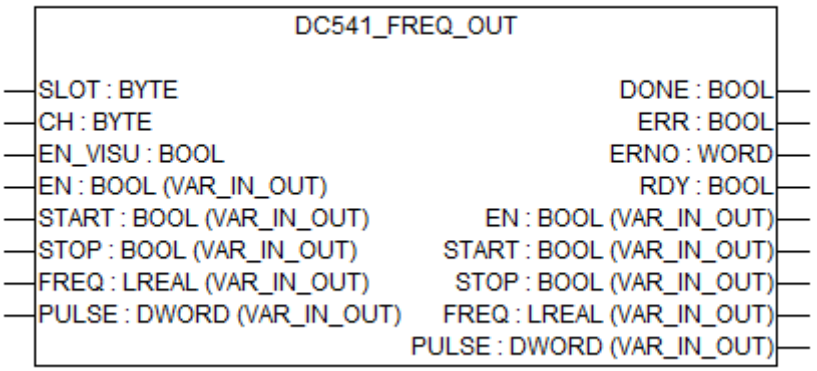
If input EN_FREQ = TRUE, output RPM (revolutions per minute) displays the speed of rotation calculated from the measured time.

If input EN_FREQ = FALSE, output RPM =0.

Function call in ST

```
DC541FreqFast(SLOT      := DC541FreqFast_SLOT,
              CH        := DC541FreqFast_CH,
              EN_VISU   := DC541FreqFast_EN_VISU,
              EN        := DC541FreqFast_EN,
              EN_0      := DC541FreqFast_EN_0,
              EN_1      := DC541FreqFast_EN_1,
              EN_FREQ   := DC541FreqFast_EN_FREQ,
              PREC      := DC541FreqFast_PREC);


DC541FreqFast_DONE      := DC541FreqFast.DONE;
DC541FreqFast_ERR       := DC541FreqFast.ERR;
DC541FreqFast_ERNO      := DC541FreqFast.ERNO;
DC541FreqFast_NEW       := DC541FreqFast.NEW;
DC541FreqFast_SUBST     := DC541FreqFast.SUBST;
DC541FreqFast_DUR       := DC541FreqFast.DUR;
DC541FreqFast_FREQ      := DC541FreqFast.FREQ;
DC541FreqFast_RPM       := DC541FreqFast.RPM;
```

DC541_FREQ_OUT



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.2
Type	Function block with historical values
Group	Counters

The function block DC541_FREQ_OUT is used to output pulses with a fixed frequency on one channel of the device DC541. It is able to output pulses with a frequency between 0.2 and 2.5 kHz. The pulse jitter depends on the cycle time of the DC541. The pulse length is always a multiple of the cycle time of the DC541.

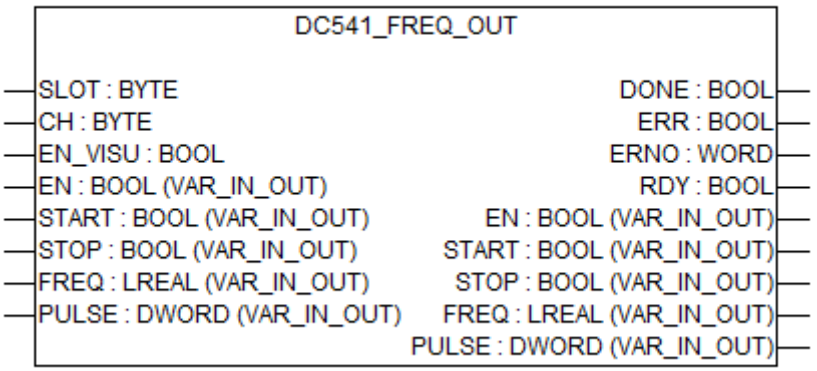
In case of a presetting of PULSE = 0, the output of pulses is infinite. The pulse output is started with a positive edge at input START. The output is aborted if START = FALSE. A positive edge at input STOP interrupts the pulse output. The output is continued if STOP = FALSE.

If input PULSE > 0, the function block outputs the number of pulses specified at input PULSE with the frequency specified at input FREQ on the channel specified at input CH. After the function block has output the number of pulses specified at PULSE, the output RDY becomes TRUE.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency output.

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	0-7 for the inputs C0-C7	-

CH is used to select the channel for time and frequency measurement. If an invalid value is specified at input CH or if the selected channel is not configured as frequency measurement, the function flock is aborted with DONE = ERR = TRUE and an error is displayed.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

START

Data type	Default value	Range	Unit
BOOL	-	-	-

A positive edge at input START starts the frequency output. The frequency output is aborted if START = FALSE.

STOP

Data type	Default value	Range	Unit
BOOL	-	-	-

A positive edge at input STOP interrupts the frequency output. If a given number of pulses has to be output, the counter continues counting if STOP = FALSE. The STOP is not synchronized with the frequency.

FREQ

Data type	Default value	Range	Unit
LREAL	-	0.2-2500.0	Hz

FREQ (frequency) input is used to preset the frequency to be output.

PULSE

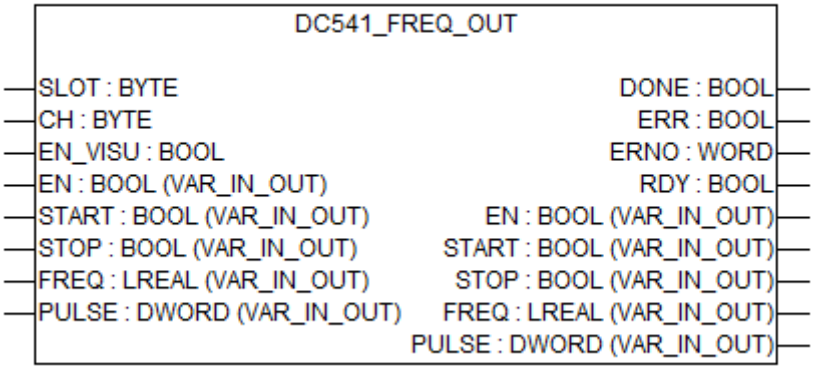
Data type	Default value	Range	Unit
DWORD	-	-	-

Input PULSE is used to determine whether frequency output shall be performed endless or only for the specified number of pulses.

PULSE = 0: Endless output.

PULSE > 0: Output of the specified number of pulses.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 61: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

RDY (ready)

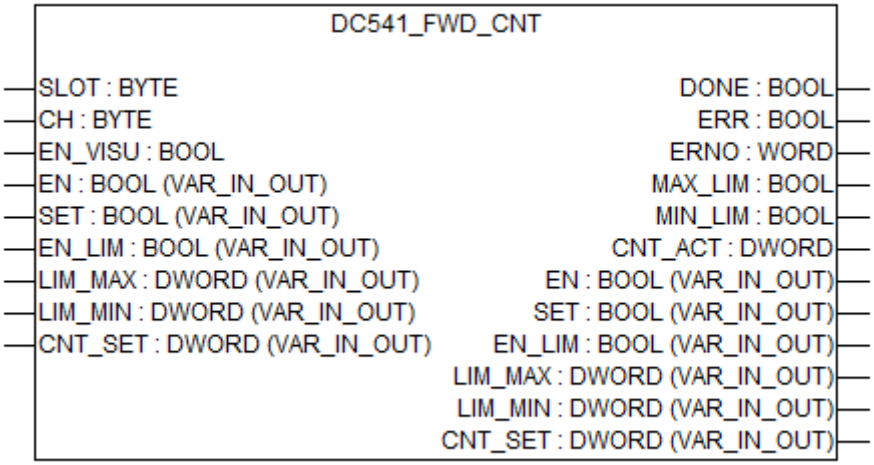
Data type	Default value	Range	Unit
BOOL	-	-	-

If PULSE > 0, output RDY becomes TRUE after the specified number of pulses has been output.

Function call in ST

```
DC541FreqOut(SLOT      := DC541FreqOut_SLOT,
             CH        := DC541FreqOut_CH,
             EN_VISU   := DC541FreqOut_EN_VISU,
             EN        := DC541FreqOut_EN,
             START     := DC541FreqOut_START,
             STOP      := DC541FreqOut_STOP,
             FREQ      := DC541FreqOut_FREQ,
             PULSE     := DC541FreqOut_PULSE);


DC541FreqOut_DONE      := DC541FreqOut.DONE;
DC541FreqOut_ERR       := DC541FreqOut.ERR;
DC541FreqOut_ERNO      := DC541FreqOut.ERNO;
DC541FreqOut_RDY       := DC541FreqOut.RDY;
```

DC541_FWD_CNT



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.1
Type	Function block with historical values
Group	Counters

The function block DC541_FWD_CNT provides a 32-bit count up counter which is able to count a maximum frequency of 50 kHz at the inputs C0 and C1 or 5 kHz at the inputs C2-C7. In the DC541, the counter is implemented as a 16 bit counter. The actual counter value ACT_CNT is bulit inside the function block by adding the counter differences that occur within the individual cycles. In order not to loose any counting pulses, the function block has to be called cyclically.

- Channel 0-1: 50 kHz max. -> 32767 / 50 = 655 ms
- Channel 2-7: 5 kHz max. -> 32767 / 5 = 6550 ms

Using the counter e.g. in a 100 ms task will prevent any loss of counting pulses.

Operating modes

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32-bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

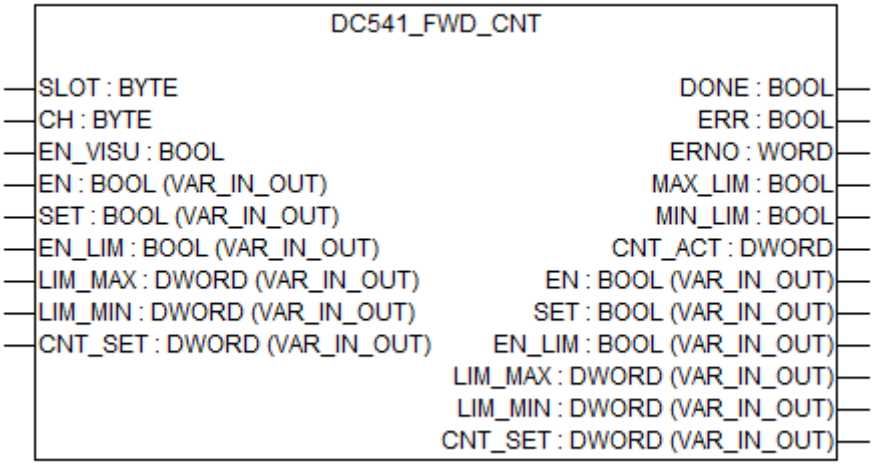
If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

The device DC541 must be configured as counting device (counter mode).

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	0-7 for the inputs C0-C7	-

CH is used to select the channel for time and frequency measurement. If an invalid value is entered at input CH or if the selected channel is not configured as 32-bit count up counter (FWD_CNT), the function flock is aborted with DONE = ERR = TRUE and an error is displayed.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

SET

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

At a rising edge at this input, the counter is set to the value of input CNT_SET. Counting is enabled after reset process to SET = TRUE.

EN_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Input EN_LIM (enable limit) is used to set the operating mode for the counter:

EN_LIM = FALSE: Infinite counter (endless mode).

EN_LIM = TRUE: Limiting counter (limit mode).

Switching between the operating modes can be done during running operation. And it is possible to change the upper limit value LIM_MAX and the lower limit value LIM_MIN during running operation. LIM_MAX has to be higher than LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

In case of a rising edge at input EN_LIM (transition from endless mode to limit mode), the present counter value can change depending on the values of LIM_MIN and LIM_MAX.

Present counter value	Changed counter value
ACT_CNT < LIM_MIN	ACT_CNT := LIM_MIN
LIM_MIN ≤ ACT_CNT ≤ LIM_MAX	ACT_CNT := ACT_CNT (no change)
ACT_CNT > LIM_MAX	ACT_CNT := LIM_MIN

LIM_MAX

Data type	Default value	Range	Unit
DWORD	-	-	-

LIM_MAX (limit maximum) is used to set the upper limit value for the counter. LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

LIM_MIN

Data type	Default value	Range	Unit
DWORD	-	-	-

LIM_MIN (limit minimum) is used to set the lower limit value for the counter. The upper limit value LIM_MAX has to be higher than LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

CNT_SET

Data type	Default value	Range	Unit
DWORD	-	-	-

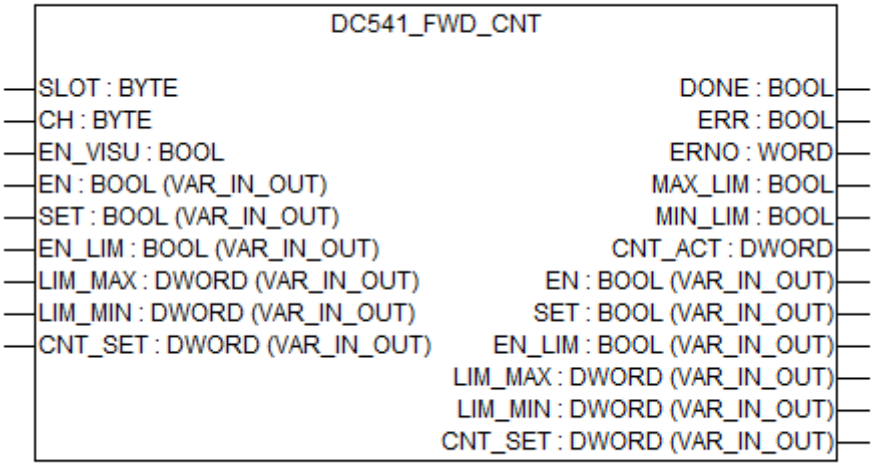
CNT_SET (counter set) is used to adjust the counter set value.

If input SET = TRUE, the counter is set to the value of input CNT_SET and remains at this value as long as input SET = TRUE. Counting is enabled again with the occurrence of the falling edge at input SET.

In case of a reference point approch (EN_REF = TRUE), the rising edge on channel C2 causes the actual value of the counter (ACT_CNT) to be adjusted to the set value CNT_SET.

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE. The function block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 62: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it

DEC	HEX	Error description
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

MAX_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

In the infinite counter operating mode (endless mode), MAX_LIM indicates whether the actual counter value is higher than the value set at input LIM_MAX.

In the limiting counter operating mode (limit mode), output MAX_LIM is FALSE.

MIN_LIM

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

In the infinite counter operating mode (endless mode), output MIN_LIM indicates whether the actual counter value is lower than the value set at input LIM_MIN.

In the limiting counter operating mode (limit mode), output MIN_LIM is FALSE.

CNT_ACT

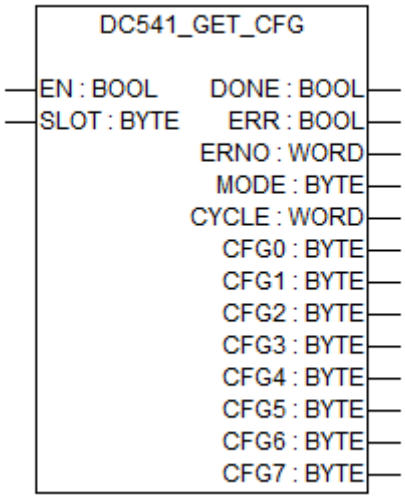
Data type	Default value	Range	Unit
DWORD	-	-	-

CNT_ACT (counter actual) displays the actual counter value of the count up counter.

Function call in ST

```
DC541FwdCnt(SLOT      := DC541FwdCnt_SLOT,
            CH        := DC541FwdCnt_CH,
            EN_VISU   := DC541FwdCnt_EN_VISU,
            EN        := DC541FwdCnt_EN,
            SET       := DC541FwdCnt_SET,
            EN_LIM    := DC541FwdCnt_EN_LIM,
            LIM_MAX   := DC541FwdCnt_LIM_MAX,
            LIM_MIN   := DC541FwdCnt_LIM_MIN,
            CNT_SET   := DC541FwdCnt_CNT_SET);


DC541FwdCnt_DONE      := DC541FwdCnt.DONE;
DC541FwdCnt_ERR       := DC541FwdCnt.ERR;
DC541FwdCnt_ERNO      := DC541FwdCnt.ERNO;
DC541FwdCnt_RDY_REF   := DC541FwdCnt.RDY_REF;
DC541FwdCnt_RDY_TOUCH := DC541FwdCnt.RDY_TOUCH;
DC541FwdCnt_MAX_LIM   := DC541FwdCnt.MAX_LIM;
DC541FwdCnt_MIN_LIM   := DC541FwdCnt.MIN_LIM;
DC541FwdCnt_CNT_ACT   := DC541FwdCnt.CNT_ACT;
```

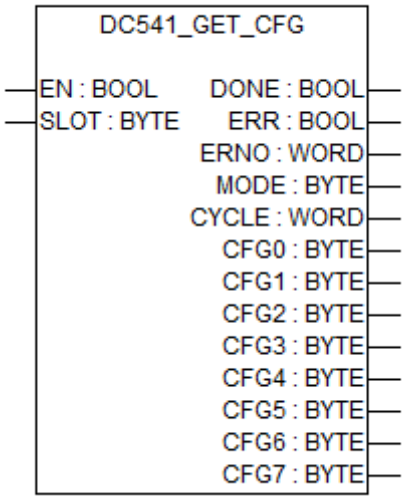
DC541_GET_CFG



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.1
Type	Function block with historical values
Group	Diagnosis

The function block DC541_GET_CFG is used to poll the actual configuration of the device DC541. The information is displayed as long as input EN is TRUE. The displayed information includes the operating mode and the cycle time of the device as well as the configuration of the channels C0...C7.

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE. The function block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

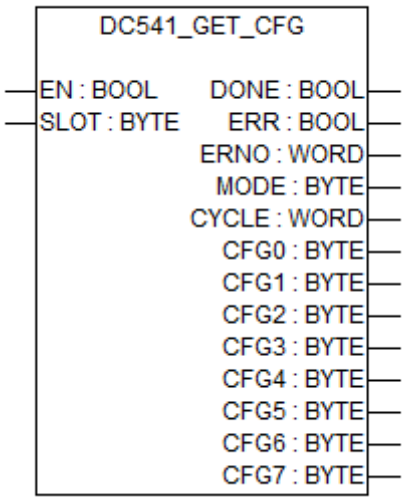
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 63: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

MODE

Data type	Default value	Range	Unit
BYTE	-	-	-

Output displays the actual operating mode of the device.

Parameter	Value	Operating mode
DC541_MODE_IO	= 16#11 = 17	IO device (IO mode)
DC541_MODE_COUNT	= 16#22 = 34	Counting device (counter mode)

CYCLE

Data type	Default value	Range	Unit
WORD	-	-	µs

CYCLE (cycle time) output displays the cycle time of the device. The cycle time is set during the device configuration and can have the following values depending on the channel configuration:

Parameter		Description	Value
IO device			50 µs
Counting device		1-2 functions	50 µs
		3-4 functions	100 µs
		5-8 functions	200 µs
"Functions"			
	PWM	Pulse-width modulator	
	FREQ	Time and frequency measurement	
	FREQ_OUT	Frequency output	
	32BIT_CNT	32-bit counter	

Parameter		Description	Value
	FWD_CNT	32-bit count up counter	
	LIMIT	Limit value monitoring for the 32-bit counter	

CFG0...CFG7

Data type	Default value	Range	Unit
BYTE	-	-	-

The outputs display the current channel configuration of the channels C0...C7.

Table 64: MODE = DC541_MODE_IO = 16#11 = 17

CFGx	Channel	Function
0	0...7	Normal input
1	0...7	Normal output
255	0...7	Interrupt input

Table 65: MODE = DC541_MODE_COUNT = 16#22 = 34

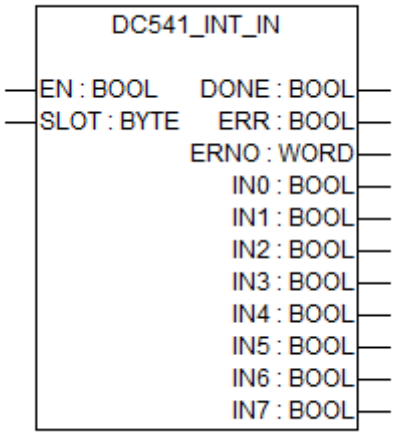
CFGx	Channel	Function
0	0...7	Normal input
1	0...7	Normal output
2	0...7	Pulse-width modulator (PWM)
3	0...1	50 kHz count up counter (FWD_CNT)
3	2...7	5 kHz count up counter (FWD_CNT)
4	0...7	Time and frequency measurement (FREQ)
5	4...7	Limit values for 32-bit counter (LIMIT)
6	0	Up/down 32-bit counter (uses channels 0...3). The channels CFG1...CFG3 can be set as desired.
7	0...7	Frequency output

Function call in ST

```
DC541GetCfg (EN   := DC541GetCfg_EN,
             SLOT := DC541GetCfg_SLOT);


DC541GetCfg_DONE  := DC541GetCfg.DONE;
DC541GetCfg_ERR   := DC541GetCfg.ERR;
DC541GetCfg_ERNO  := DC541GetCfg.ERNO;
DC541GetCfg_MODE  := DC541GetCfg.DIAG;
DC541GetCfg_CYCLE := DC541GetCfg.CYCLE;
DC541GetCfg_CFG0  := DC541GetCfg.CFG0;
DC541GetCfg_CFG1  := DC541GetCfg.CFG1;
DC541GetCfg_CFG2  := DC541GetCfg.CFG2;
```

```
DC541GetCfg_CFG3  := DC541GetCfg.CFG3;
DC541GetCfg_CFG4  := DC541GetCfg.CFG4;
DC541GetCfg_CFG5  := DC541GetCfg.CFG5;
DC541GetCfg_CFG6  := DC541GetCfg.CFG6;
DC541GetCfg_CFG7  := DC541GetCfg.CFG7;
```

DC541_INT_IN

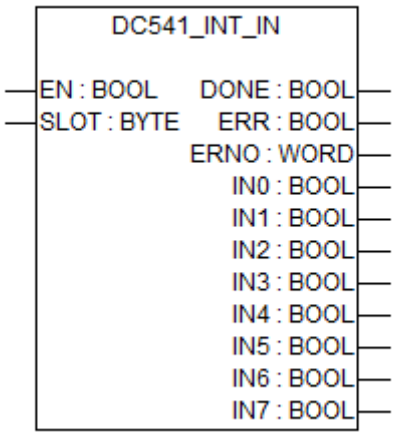


Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.2, DC541 firmware: V1.0
Type	Function block with historical values
Group	IO

Using the function block DC541_INT_IN, the interrupt program can be polled which inputs (C0...C7) triggered interrupts since the last function block calling. The corresponding outputs IN0...IN1 are set to TRUE.

The function block DC541_INT_IN can only be used if the module is configured as I/O module.

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN has to be continuously set to TRUE.

When the function block is called for the first time, input SLOT is checked for validity and the corresponding device is checked for correct configuration in the operating mode "IO mode". If this is not the case, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE.

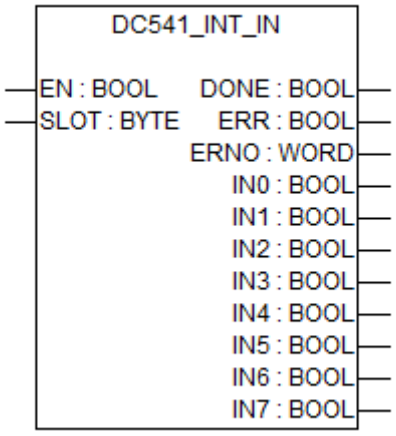
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 66: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

IN0...IN7

Data type	Default value	Range	Unit
BOOL	-	-	-

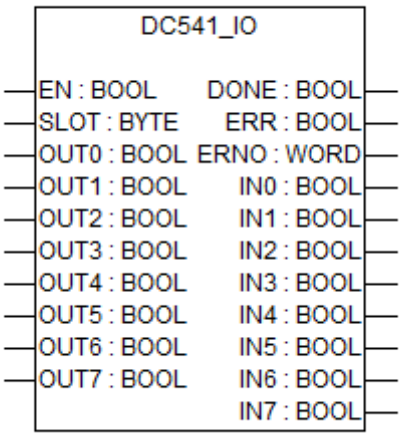
The outputs display which inputs (C0...C7) triggered an interrupt since the last function block calling.

Function call in ST

```
DC541IntIn(EN   := DC541IntIn_EN,
           SLOT := DC541IntIn_SLOT);


DC541IntIn_DONE := DC541IntIn.DONE;
DC541IntIn_ERR  := DC541IntIn.ERR;
DC541IntIn_ERNO := DC541IntIn.ERNO;
DC541IntIn_IN0  := DC541IntIn.IN0;
DC541IntIn_IN1  := DC541IntIn.IN1;
DC541IntIn_IN2  := DC541IntIn.IN2;
```

```
DC541IntIn_IN3  := DC541IntIn.IN3;
DC541IntIn_IN4  := DC541IntIn.IN4;
DC541IntIn_IN5  := DC541IntIn.IN5;
DC541IntIn_IN6  := DC541IntIn.IN6;
DC541IntIn_IN7  := DC541IntIn.IN7;
```

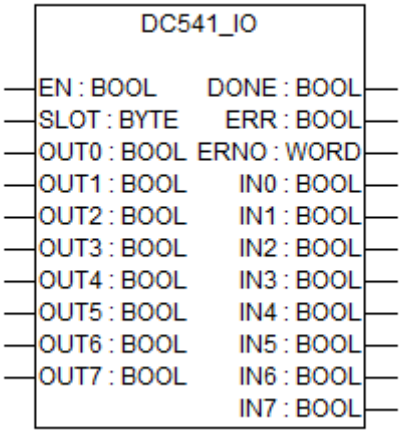
DC541_IO



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.2, DC541 firmware: V1.0
Type	Function block with historical values
Group	IO

The function block DC541_IO is used to read the "normal" inputs and to write the outputs of the DC541. Reading and writing the inputs and outputs is performed as long as input EN is TRUE. This function block can be used for both operating modes, i.e. when the device is configured as I/O module and when it is configured as counting device (counter mode).

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE. The function block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

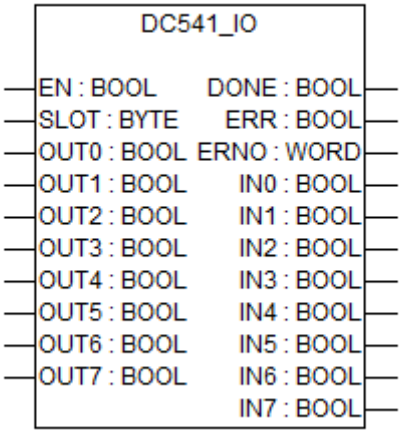
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

OUT0...OUT7

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The inputs are used to specify the set status of the outputs C0...C7. Only channels that are configured as outputs are currently written.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 67: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

IN0...IN7 (input C0...C7)

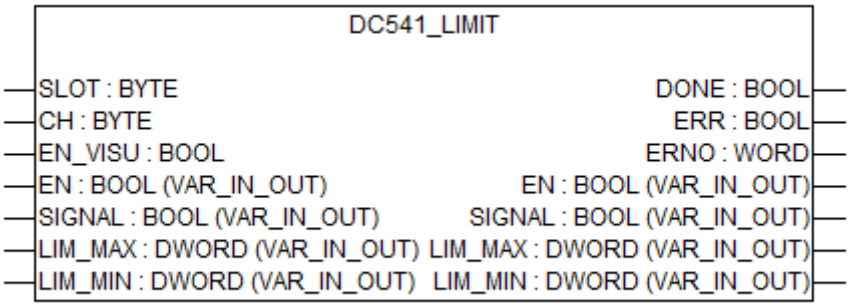
Data type	Default value	Range	Unit
BOOL	-	-	-

The outputs display the current status of the channels C0...C7.

Function call in ST

```
DC541Io (EN   := DC541Io_EN,
         SLOT := DC541Io_SLOT,
         OUT0 := DC541Io_OUT0,
         OUT1 := DC541Io_OUT1,
         OUT2 := DC541Io_OUT2,
         OUT3 := DC541Io_OUT3,
         OUT4 := DC541Io_OUT4,
         OUT5 := DC541Io_OUT5,
         OUT6 := DC541Io_OUT6,
         OUT7 := DC541Io_OUT7);


DC541Io_DONE := DC541Io.DONE;
DC541Io_ERR  := DC541Io.ERR;
DC541Io_ERNO := DC541Io.ERNO;
DC541Io_IN0  := DC541Io.IN0;
DC541Io_IN1  := DC541Io.IN1;
DC541Io_IN2  := DC541Io.IN2;
DC541Io_IN3  := DC541Io.IN3;
DC541Io_IN4  := DC541Io.IN4;
DC541Io_IN5  := DC541Io.IN5;
DC541Io_IN6  := DC541Io.IN6;
DC541Io_IN7  := DC541Io.IN7;
```

DC541_LIMIT



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.1
Type	Function block with historical values
Group	Counters

The function block DC541_LIMIT is used for limit value monitoring of the 32-bit counter. The function block can be used to directly display various counting values of the 32-bit counter (DC541_32BIT_CNT) via binary outputs. Using the input SIGNAL you can determine whether the corresponding output is switched to FALSE or TRUE.

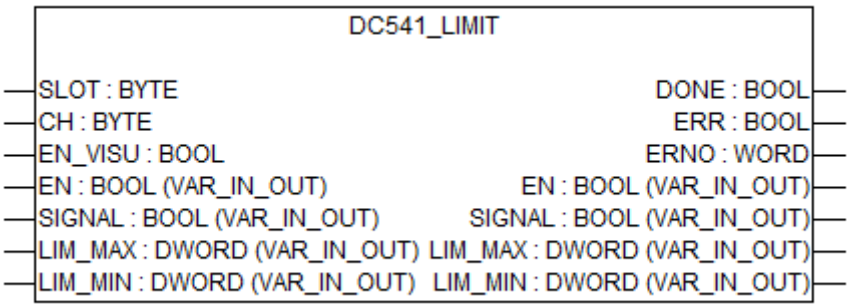
The time resolution of the function block is < 100 µs, i.e. the result is increment accurate up to a frequency of 10 kHz.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If the lower limit value LIM_MIN is higher than the upper limit value LIM_MAX, an error message is displayed.

The device DC541 must be configured as counting device (counter mode) and channel C0 as 32-bit counter.

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU is TRUE ↬ *Chapter 1.5.4.11.1.2 “Visualizations” on page 1154*.

Input description



SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	4-7 for the outputs C4-C7	-

Input CH is used to select the channel for the limit value monitoring. If an invalid value is specified at input CH or if the selected channel is not configured as Limit Channel 0 or if channel C0 is not configured as 32-bit counter, the function block is aborted with DONE = ERR = TRUE and an error is displayed.

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

If EN_VISU input (enable input in visualization) is TRUE, it is possible to control the function block inputs (except SLOT, CH and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

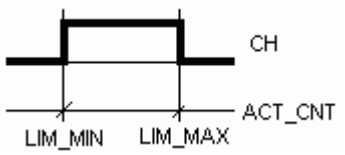
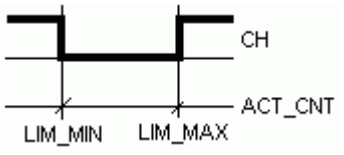
In order to enable pulse counting for input CH, input EN has to be continuously TRUE. The function block is not processed if input EN = FALSE.

When the function block is called for the first time, the inputs are checked for validity and plausibility and the corresponding device is checked for correct configuration in the operating mode "counting mode". If this is not the case, processing is aborted.

SIGNAL

Data type	Default value	Range	Unit
BOOL	-	-	-

Using the input SIGNAL you can determine whether the corresponding output CH is switched to TRUE or FALSE.

Counter value of the 32-bit counter DC541_32BIT_CNT / ACT_CNT	Output CH when SIGNAL = TRUE	Output CH when SIGNAL = FALSE
ACT_CNT < LIM_MIN	FALSE	TRUE
LIM_MIN ≤ ACT_CNT ≤ LIM_MAX	TRUE	FALSE
ACT_CNT > LIM_MAX	FALSE	TRUE
		

If SIGNAL is TRUE, the output CH is TRUE, if the counter value of the 32-bit counter is within the range given by the limit values LIM_MIN and LIM_MAX. If the counter value is out of this range, output CH is FALSE.

If SIGNAL is FALSE, the output CH is FALSE, if the counter value of the 32-bit counter is within the range given by the limit values LIM_MIN and LIM_MAX. If the counter value is out of this range, output CH is TRUE.

LIM_MAX

Data type	Default value	Range	Unit
DWORD	-	-	-

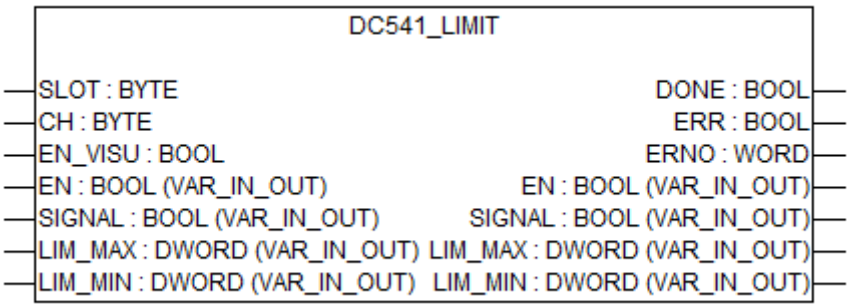
LIM_MAX (limit maximum) is used to set the upper limit value for the counter. LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

LIM_MIN

Data type	Default value	Range	Unit
DWORD	-	-	-

LIM_MIN (limit minimum) is used to set the lower limit value for the counter. The upper limit value LIM_MAX has to be higher than LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

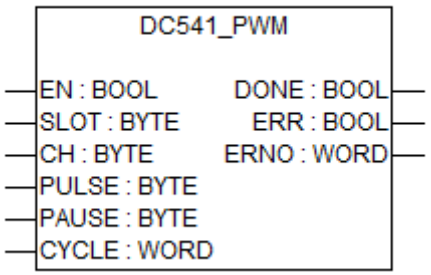
Table 68: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

Function call in ST

```
DC541Limit (SLOT    := DC541Limit_SLOT,
            CH      := DC541Limit_CH,
            EN_VISU := DC541Limit_EN_VISU,
            EN      := DC541Limit_EN,
            SIGNAL  := DC541Limit_SIGNAL,
            LIM_MAX := DC541Limit_LIM_MAX,
            LIM_MIN := DC541Limit_LIM_MIN);


DC541Limit_DONE  := DC541Limit.DONE;
DC541Limit_ERR   := DC541Limit.ERR;
DC541Limit_ERNO  := DC541Limit.ERNO;
```

DC541_PWM



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.1
Type	Function block with historical values
Group	Counters

The function block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio. The on and off times are adjusted as 8 bit numbers.

The minimum switching time is specified at input CYCLE, i.e. if an output has been switched to FALSE or TRUE by the PWM, this output remains in this state for at least this time (CYCLE µs).

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the function block ↯ *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132* (output CYCLE).

Examples

Table 69: Cycle time of DC541 = 50 µs

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
1	2	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	10 x TRUE / 20 x FALSE / 10 x TRUE / 20 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	90 x TRUE / 60 x FALSE / 90 x TRUE / 60 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE

Table 70: Cycle time of DC541 = 100 µs

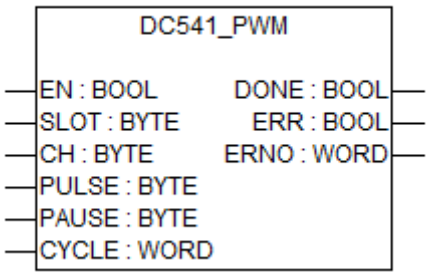
PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
1	2	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE
4	8	500	5 x TRUE / 10 x FALSE / 5 x TRUE / 10 x FALSE / … i.e. 500 µs = TRUE and 1000 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	45 x TRUE / 30 x FALSE / 45 x TRUE / 30 x FALSE / … i.e. 4500 µs = TRUE and 3000 µs = FALSE

Table 71: Cycle time of DC541 = 200 µs

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
1	2	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE /... i.e. 600 µs = TRUE, 1200 µs = FALSE
4	8	500	3 x TRUE / 6 x FALSE / 3 x TRUE / 6 x FALSE / ... i.e. 600 µs = TRUE, 1200 µs = FALSE (as in example 1, i.e. ratio 1 : 2)
3	2	3000	22 x TRUE / 15 x FALSE / 23 x TRUE / 15 x FALSE / … i.e. first 4400 µs = TRUE and 3000 µs = FALSE and then 4600 µs = TRUE and 3000 µs = FALSE

The device DC541 must be configured as counting device (counter mode).

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN has to be continuously set to TRUE.

When the function block is called for the first time, input SLOT is checked for validity and the corresponding device is checked for correct configuration in the operating mode "IO mode". If this is not the case, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type	Default value	Range	Unit
BYTE	-	-	-

Input is used to select the output to be pulsed.

PULSE

Data type	Default value	Range	Unit
BYTE	-	-	-

Input is used to specify the ratio for the TRUE signal (on time ratio).

PAUSE

Data type	Default value	Range	Unit
BYTE	-	-	-

Input is used to specify the ratio for the FALSE signal (off time ratio).

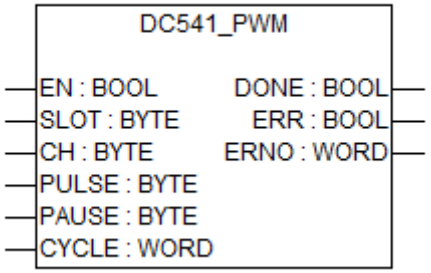
CYCLE

Data type	Default value	Range	Unit
WORD	-	-	µs

Input CYCLE is used to specify the minimum switching time of the output. The output remains in the TRUE or FALSE state for at least the time specified at CYCLE.

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the function block DC541_GET_CFG, output CYCLE *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132*.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

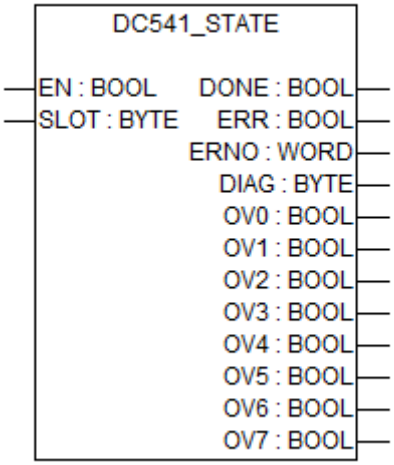
Table 72: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
	400x	Wrong value at function block input number x
16385...16399	4001...400F	x = 1...F
24586	600A	Lifesign error from DC541

Function call in ST

```
DC541Pwm(EN     := DC541Pwm_EN,
         SLOT   := DC541Pwm_SLOT,
         CH     := DC541Pwm_CH,
         PULSE  := DC541Pwm_PULSE,
         PAUSE  := DC541Pwm_PAUSE,
         CYCLE  := DC541Pwm_CYCLE);


DC541Pwm_DONE   := DC541Pwm.DONE;
DC541Pwm_ERR    := DC541Pwm.ERR;
DC541Pwm_ERNO   := DC541Pwm.ERNO;
```

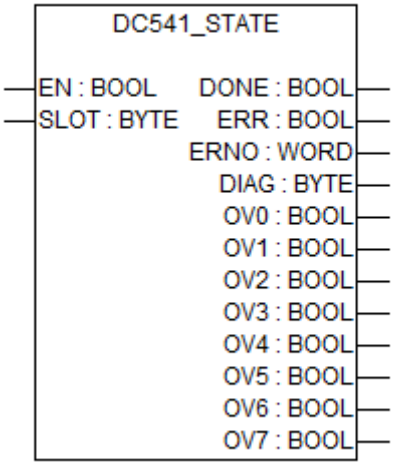
DC541_STATE



Parameter	Value
Included in library	DC541_AC500_V11.lib
Available as of firmware	V1.1.3, DC541 firmware: V1.1
Type	Function block with historical values
Group	Diagnosis

The function block DC541_STATE is used to poll the status of the device DC541. The information is displayed as long as input EN is TRUE. The displayed information includes the device diagnosis and the output status of the channels C0...C7.

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

A FALSE -> TRUE edge at input EN activates the status polling. If the value at input SLOT is not valid, processing is aborted and an error is displayed. The function block outputs are updated as long as input EN = TRUE. The function block processing has been completed successfully, if output DONE changes to TRUE. During the processing of a request, state changes at input EN are recognized but not evaluated.

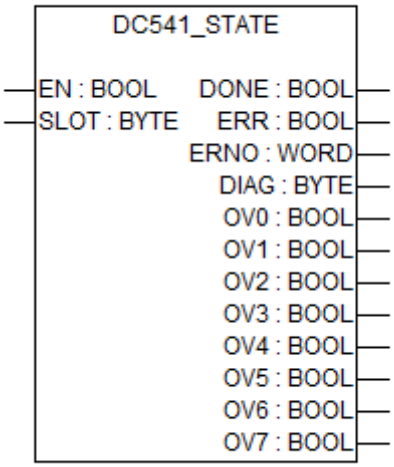
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 73: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

DIAG

Data type	Default value	Range	Unit
BYTE	-	0 16#01 = 1 -> Watchdog error	-

DIAG (diagnosis) output displays the diagnosis message of the DC541. If DIAG is 0, no diagnosis message is available. The following messages are possible:

OV0...OV7

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The outputs OV0...OV7 (overload C0...C7) display the current output status of the channels C0...C7. If the respective bit is TRUE, the corresponding output cannot be switched e.g. due to an overload or a short circuit.

Function call in ST

```
DC541State(EN   := DC541State_EN,
           SLOT := DC541State_SLOT);


DC541State_DONE := DC541State.DONE;
DC541State_ERR  := DC541State.ERR;
DC541State_ERNO := DC541State.ERNO;
DC541State_DIAG := DC541State.DIAG;
DC541State_OV0  := DC541State.OV0;
DC541State_OV1  := DC541State.OV1;
DC541State_OV2  := DC541State.OV2;
DC541State_OV3  := DC541State.OV3;
DC541State_OV4  := DC541State.OV4;
DC541State_OV5  := DC541State.OV5;
DC541State_OV6  := DC541State.OV6;
DC541State_OV7  := DC541State.OV7;
```

Visualizations

The visualization can be used to display the function block outputs. If the input EN_VISU of a function block is TRUE, it is also possible to control the inputs from the visualization. In order to allow the control of function block inputs from the program as well as from the visualization, these inputs are declared as VAR_IN_OUT and consequently have to be provided with variables. These inputs must not be provided with direct constants.

Example visualization visuDC541_32BIT_CNT

Integrated visualization of function block DC541_32BIT_CNT

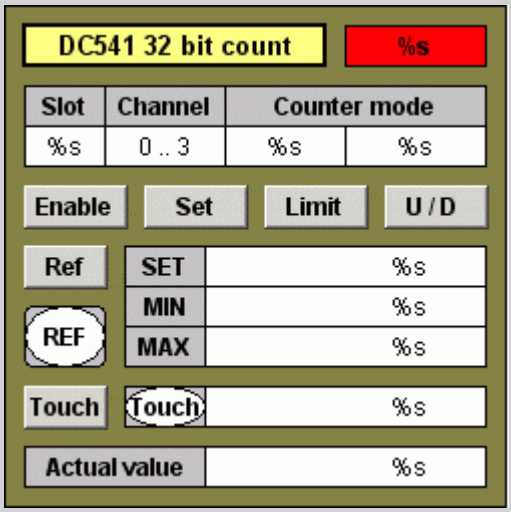


Fig. 26: Offline mode

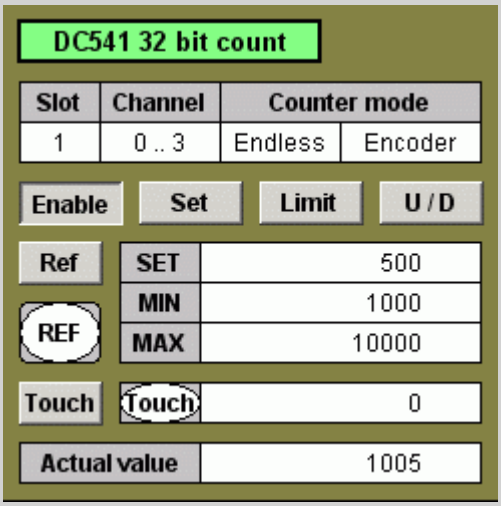


Fig. 27: Online mode, EN_VISU = TRUE

If an error occurred during function block processing, the error number is displayed in the top right until EN = FALSE.

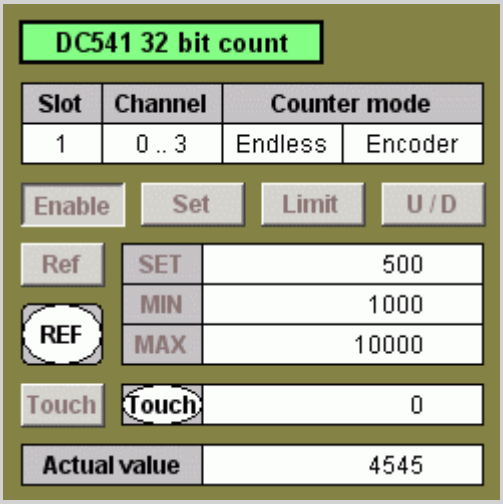
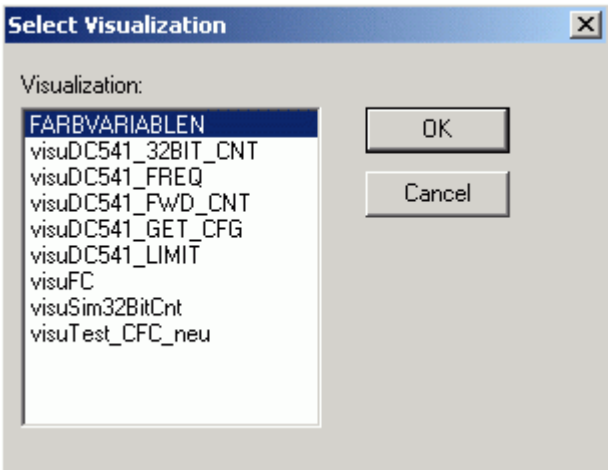


Fig. 28: Online mode, EN_VISU = FALSE. The inputs cannot be modified using the visualization.

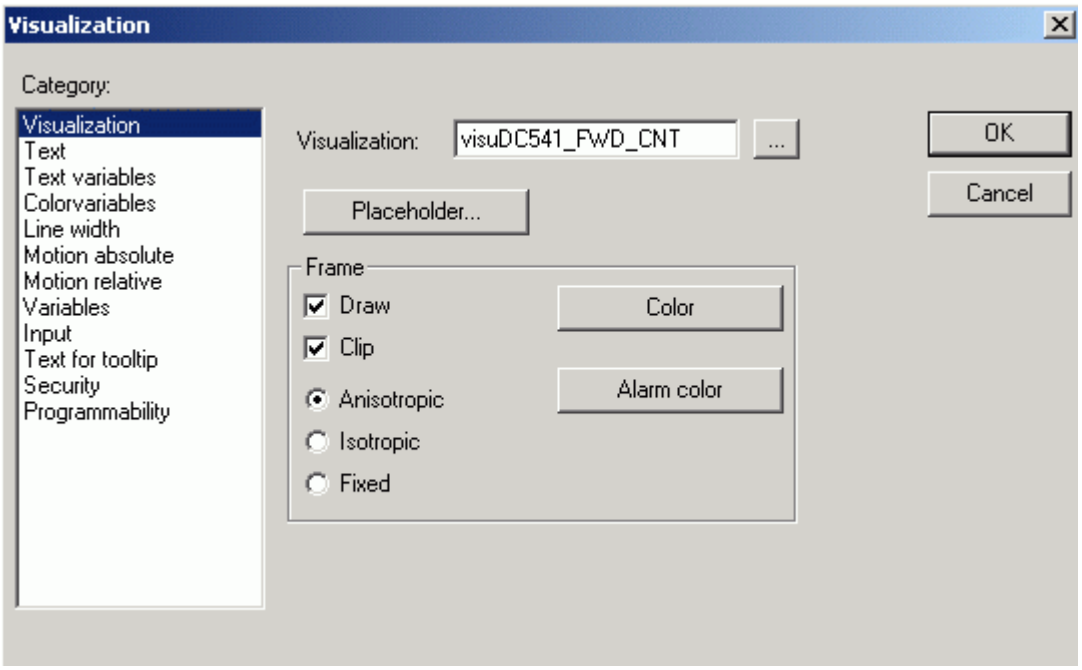
Integrate a visualization into a project

1. In the **Visualizations** tab, add an object.
2. Click on .
3. Select the corresponding visualization for the function block.



Configure the visualization

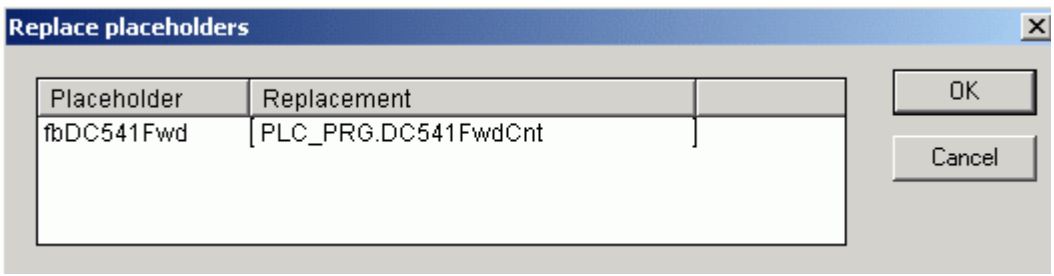
1. Right-click on the visualization and select **Configure**.



2. Under **Frame**, we recommend to select *Fixed* to maintain the original width-to-height ratio and font size.

Link the visualization to the function block instance

3. Click the **Placeholder** button.
4. In column **Replacement**, enter the function block instance directly or press F2 to open the input assistant.



5. Click **OK** to close all dialogs.
6. Adapt the visualization to the correct size.

1.5.4.11.2 DC541 dampener library

Library file name: **DC541_DAMPENER_AC500_V13.lib**

This library contains all function blocks necessary for using DC541 in DAMPENER-mode.

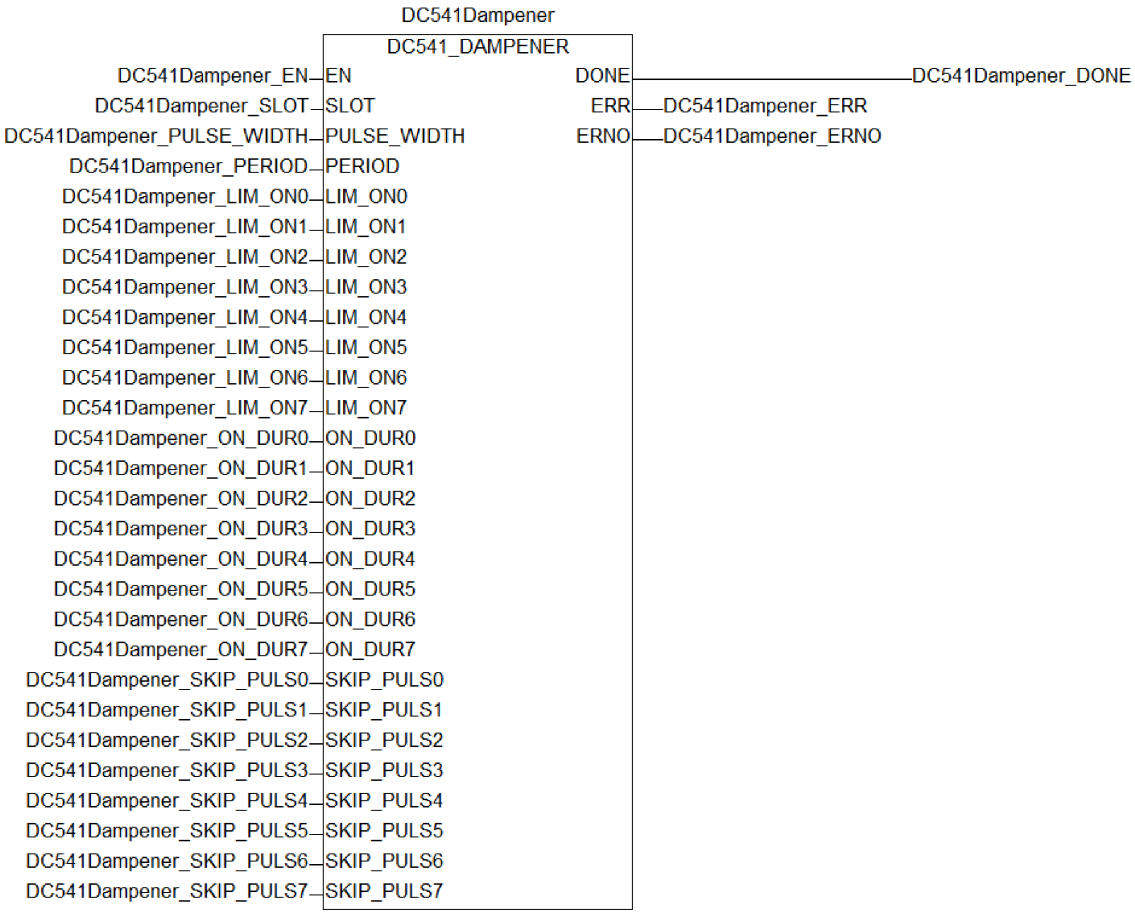
The configuration of DC541 is done in PLC configuration.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

The function blocks are available in AC500 control systems with a runtime system of version V1.3 or later.

Function blocks

DC541_DAMPENER



Parameter	Value
Included in library	DC541_DAMPENER_AC500_V13.lib
Available as of firmware	V1.12
Type	Function block with historical values
Available as of PLC runtime system	V1.3

The function block DC541_DAMPENER transfers all values to control the DAMPENER functionality for DC541 device.

The Dampener-Mode is a modified PWM-Mode. If the device is configured as DAMPENER, all 8 outputs are controlled by the DAMPENER functionality.

All 8 outputs are switched on/off with the same frequency, but use a different time to switch on a a different duration for being on. These values are specified by function block DC541_DAMPENER with input parameters LIM_MINx and ON_DURx.

The following rule has to be applied:

- LIM_MINx < PERIOD
- ON_DURx < PERIOD

All input parameters can be continuously modified, the values are accepted immediately. The following values are specified:

- Length for a single pulse PULSE_WIDTH in microseconds from 100 to 65535
- Number of pulses for a complete period: PERIOD 100 to 65535, every pulse as a PULSE_WITDH length

The resulting frequency is then calculated as:

f = 1/ (PULSE_WIDTH*PERIOD) with a range from 1/ (100*100µs) = 100 Hz to 1/ (65535*65535µs) = 0.00023 Hz

The device uses an internal counter with a clock PULSE_WIDTH µs which counts from 0 to PERIOD.

When the condition LIM_MINx = counter is reached, the output is switched on for ON_DURx pulses.

In addition, it is possible to leave out up to 255 periods for a certain output by using SKIP_PULSx input parameter.

The following example shows the different result with SKIP_PULS=0 and SKIP_PULS=1 and otherwise identical values.

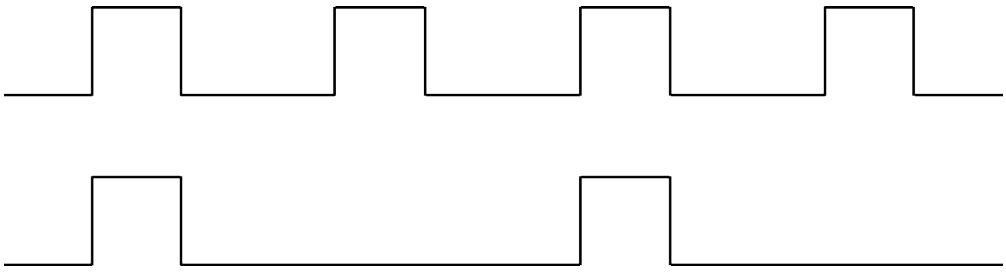
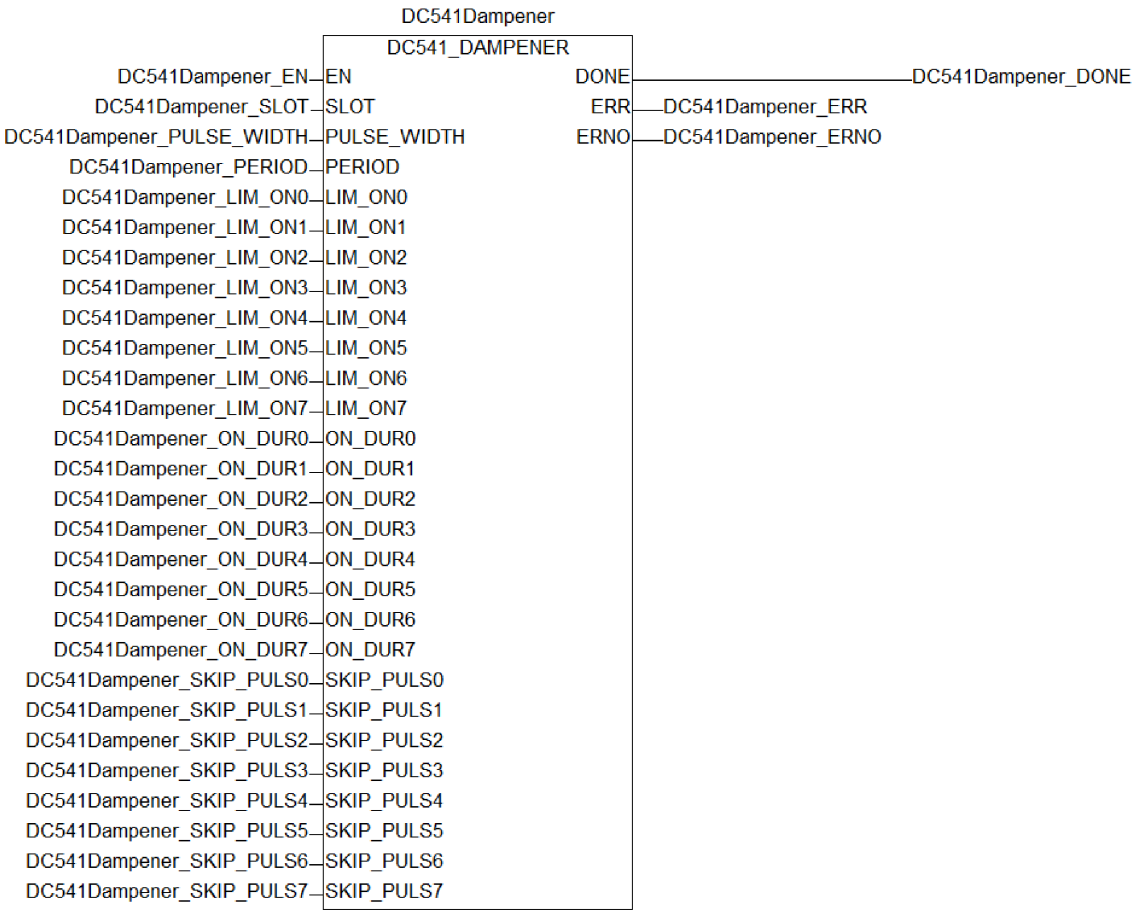


Fig. 29: SKIP_PULSE

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	-	-	-

This input is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

PULSE_WIDTH

Data type	Default value	Range	Unit
WORD	-	100 ... 65535	µs

The input PULSE_WIDTH gives the time in microseconds which is used to increase the internal counter by 1. Also, this gives the minimal time available for switching on a binary output. All other times are a multiple of this time

PERIOD

Data type	Default value	Range	Unit
WORD	-	100 to 65535	PULSE_WITDH length

Period duration [in number of pulses], i.e. internal counter value is being set to 0 when this value is reached.

LIM_MIN

Data type	Default value	Range	Unit
WORD	-	-	-

Lower limit for switching on, the respective binary output is switched on when internal counter reaches this value.

ON_DUR

Data type	Default value	Range	Unit
WORD	-	-	-

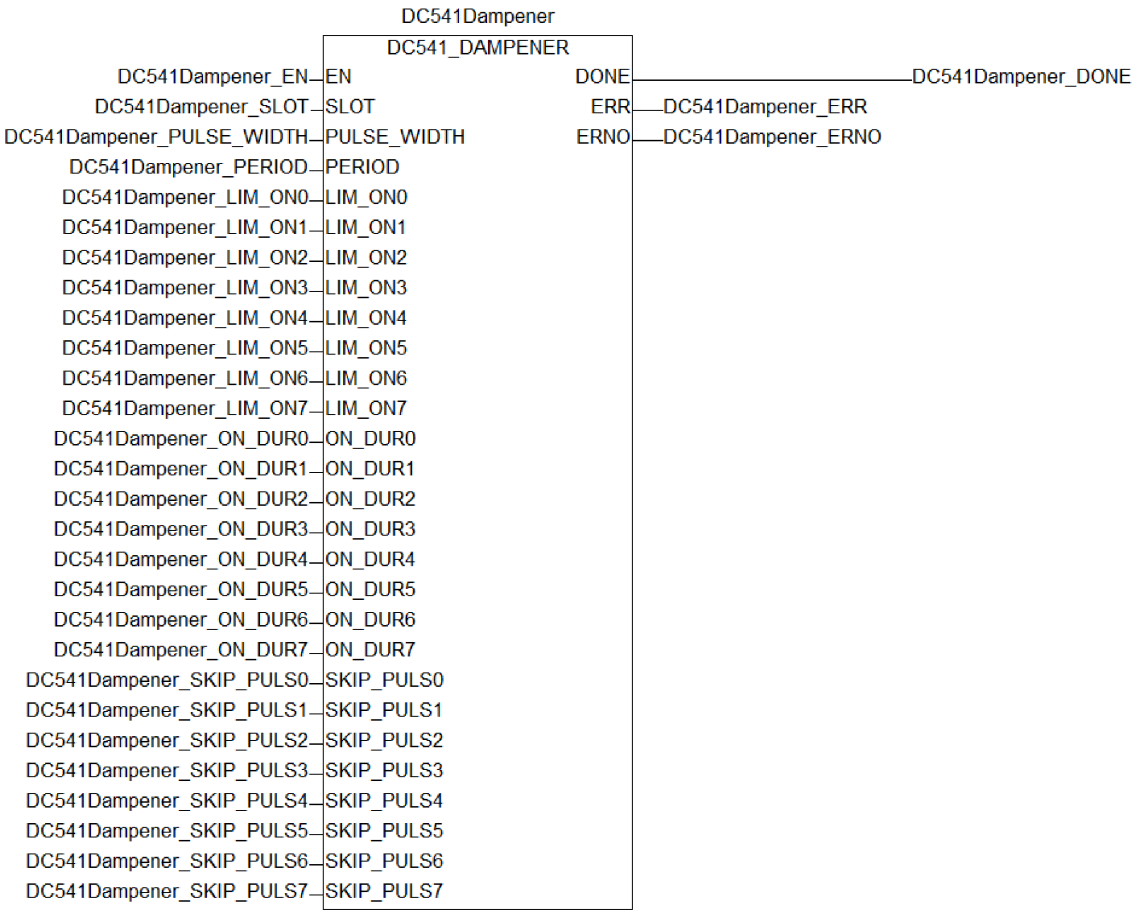
Duration [in number of pulses] for switching on the respective binary output.

SKIP_PULSE

Data type	Default value	Range	Unit
WORD	-	-	-

An optional number of periods to skip switching on the respective binary output

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO (error number)

Data type	Default value	Range	Unit
WORD	0	≥ 0	-

Output provides an error identifier if an invalid value was applied to an input. ERNO always has to be considered together with the output ERR. The value output at ERNO is only valid if ERR is TRUE.

Table 74: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

1.5.4.11.3 DC541 PWM library

Library file name: **DC541_PWM_AC500_V22.lib**

This library contains all function blocks necessary for using DC541 in PWM-mode.

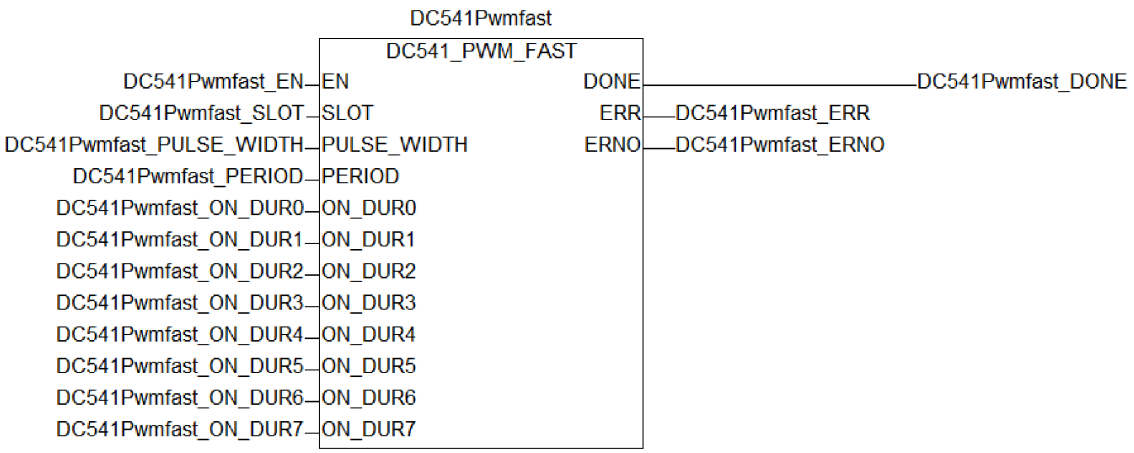
The configuration of DC541 is done in PLC configuration.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

The function blocks are available in AC500 control systems with a runtime system of version V2.2 or later.

Function blocks

DC541_PWM_FAST



Parameter	Value
Included in library	DC541_PWM_AC500_V22.lib
Available as of firmware	V1.12
Type	Function block with historical values
Available as of runtime system	V2.2

The block DC541_DAMPENER transfers all values to control the PWM functionality for DC541 device.

If the device is configured in PWM-mode, all 8 outputs are controlled by PWM functionality (Pulse-width Modulation).

All 8 outputs are switched on with the same frequency, at the same start time, but with different duration.

The duration is adjusted with the block DC541_PWM_FAST with values for ON_DURx.

The following rule has to be applied:

- ON_DURx < PERIOD

All input parameters can be continuously modified, the values are accepted immediately.

The following values are specified:

- Minimal length for a single pulse PULSE_WIDTH in microseconds from 40 to 32767
- Number of pulses for a complete period: PERIOD 2 to 255

The resulting frequency is then calculated as:

f = 1/ (PULSE_WIDTH*PERIOD) with a range from 1/ /(2*4040 to 32767µs) = 12.5 kHz to 1/(255*32767µs) = 0.12 Hz

The device uses an internal counter with a clock PULSE_WIDTH µs which counts from 0 to PERIOD.

When the condition ON_DURx = counter is reached, the respective output is switched on, off otherwise.

All outputs with ON_DURx>0 are switched on at the same.

The following sequence would be achieved with:

- PULSE_WIDTH= 50, PERIOD=10, ON_DUR0=6 (SIGNAL0) and ON_DUR1=4 (SIGNAL1) or
- PULSE_WIDTH= 100, PERIOD=5, ON_DUR0=3 and ON_DUR1=2

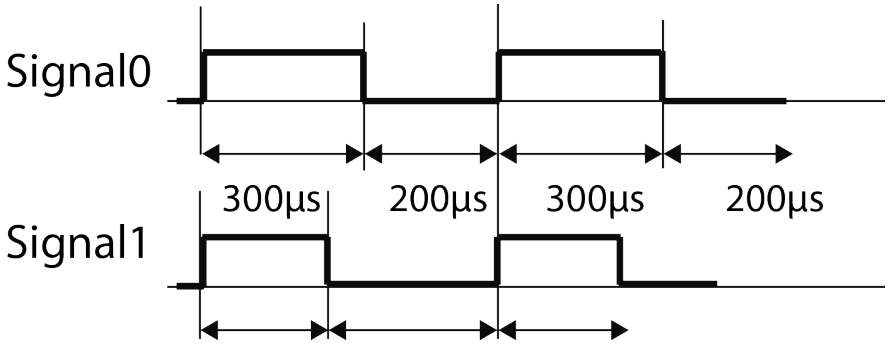
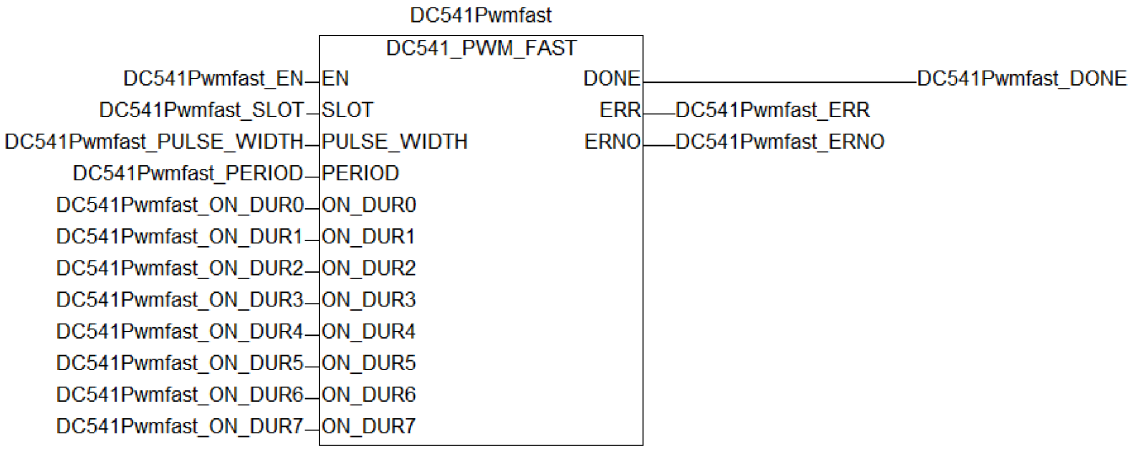


Fig. 30: SIGNAL_PWM

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	-	-	-

This input is used to select the slot (module number) of the DC541.

The slots are numbered consecutively from right to left. Slot 1 is the first slot on the left of the CPU.

PULSE_WIDTH

Data type	Default value	Range	Unit
WORD	-	100 ... 65535	µs

The input PULSE_WIDTH gives the time in microseconds which is used to increase the internal counter by 1. Also, this gives the minimal time available for switching on a binary output. All other times are a multiple of this time

PERIOD

Data type	Default value	Range	Unit
WORD	-	100 to 65535	PULSE_WITDH length

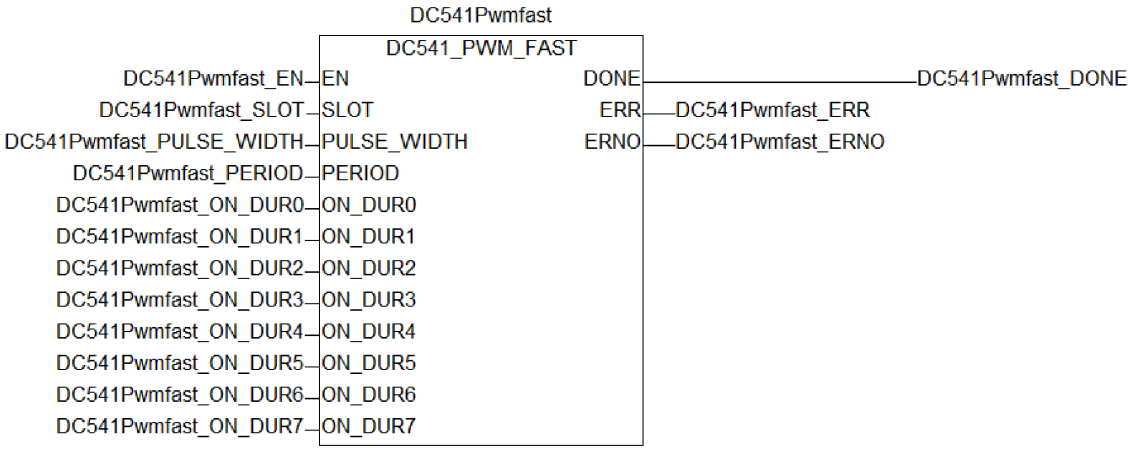
Period duration [in number of pulses], i.e. internal counter value is being set to 0 when this value is reached.

ON_DUR

Data type	Default value	Range	Unit
WORD	-	-	-

Duration [in number of pulses] for switching on the respective binary output.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE.

Table 75: DC541 error numbers

DEC	HEX	Error description
8195	2003	Unknown/wrong communication module at communication module slot
12315	301B	No access to channel, other function block might be using it
12316	301C	Illegal channel number
12329	3029	Device is not configured
16385...16399	400x 4001...400F	Wrong value at function block input number x x = 1...F
24586	600A	Lifesign error from DC541

1.5.4.12 Diagnosis library

Library file name: **Diag_AC500_V20.lib**

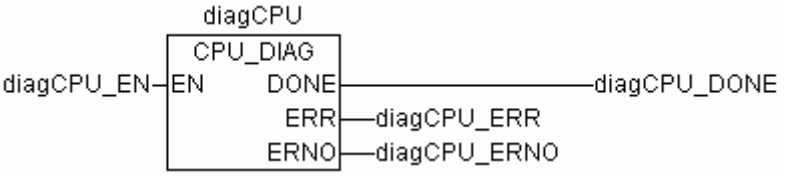
The function blocks of the diagnosis library enable the direct access on the data of the AC500 diagnosis system.

The library is not loaded automatically into an AC500 project.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

1.5.4.12.1 Function blocks

CPU_DIAG



Parameter	Value
Included in library	Diag_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Function block with historical values
Group	-

Using the function block CPU_DIAG, the entries of the AC500 diagnosis system can be read and displayed using the integrated visualization as plain text.

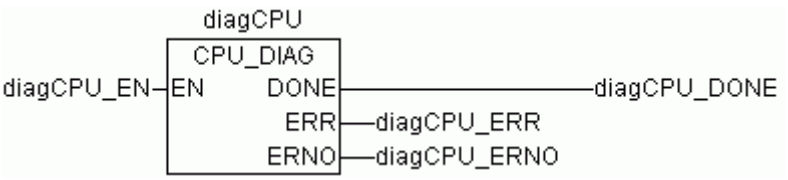
The instance diagCPU of the function block CPU_DIAG is declared in the global variables list GL_AC500_Diagnosis. The integrated visualization of the function block accesses the variables of this instance.

In order to include the function block into an AC500 project, it is only necessary to call the function block instance. Example in ST:

diagCPU(EN := TRUE);

By means of the visualization Visu_CPU_Diag, navigation within the diagnosis buffer is possible and upcoming diagnosis entries can be acknowledged.

Input description

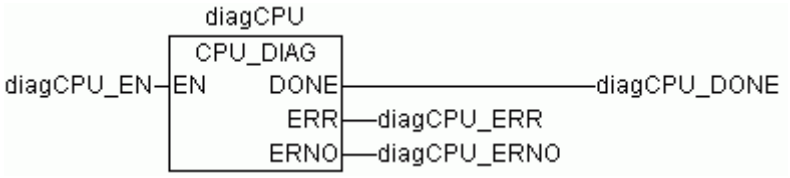


EN

Data type	Default value	Range	Unit
BOOL			

EN = TRUE enables the processing of the function block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

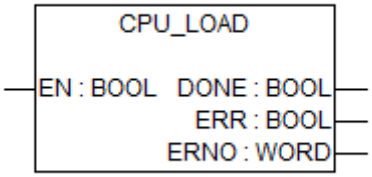
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
diagCPU(EN := diagCPU_EN);


diagCPU_DONE := diagCPU.DONE;
diagCPU_ERR  := diagCPU.ERR;
diagCPU_ERNO := diagCPU.ERNO;


or:


diagCPU(EN := TRUE);
```

CPU_LOAD



Parameter	Value
Included in library	Diag_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Function block with historical values
Group	-

The function block CPU_LOAD outputs the CPU capacitiy utilization in [%].

The following is displayed:

current load

minimum load

maximum load

average (avg) load

The instance CPU of the function block CPU_LOAD is declared in the global variables list GL_AC500_Diagnosis. The integrated visualization of the function block accesses the variables of this instance.

In order to include the function block into an AC500 project, it is only necessary to call the function block instance. Example in ST:

CPU(EN := TRUE);

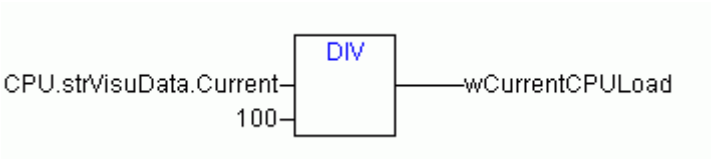
The values are entered as 0.01% values into the internal structure strVisuData of the type strCPU_LOAD in the instance CPU of the function block CPU_LOAD. To obtain the values in [%], they have to be divided by 100. The structure is composed as follows:

TYPE strCPU_LOAD :			
STRUCT			
	Current	: WORD;	(* current value *)
	Avg	: WORD;	(* average value *)
	Minimum	: WORD;	(* minimum value *)
	Maximum	: WORD;	(* maximum value *)
	Counter	: DWORD;	(* counter *)
	Conf	: WORD;	(* bit 0 = 1 (1) --> Update CPU_load *)
			(* bit 1 = 1 (2) --> Heap check *)
END_STRUCT			
END_TYPE			

If the user program should have access to these values, this is done as follows:

CPU.strVisuData.Variable.

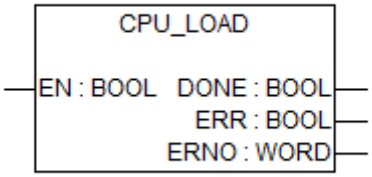
In the following example, the value of the current CPU load is read and assigned to the variable wCurrentCPULoad:



Using the visualization Visu_CPU_Load, the values are represented graphically *Chapter 1.5.4.12.2 “Visualizations” on page 1182*.

The same values can also be polled using the PLC browser command "cpuload".

Input description

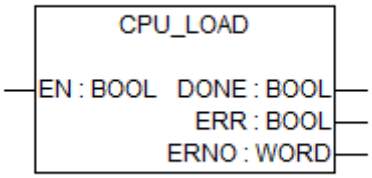


EN

Data type	Default value	Range	Unit
BOOL			

EN = TRUE enables the processing of the function block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

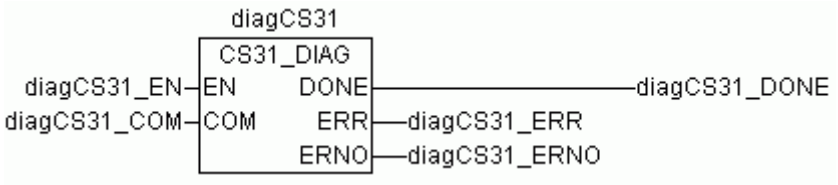
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
CPU(EN := CPU_EN);


CPU_DONE := CPU.DONE;
CPU_ERR  := CPU.ERR;
CPU_ERNO := CPU.ERNO;


or:


CPU(EN := TRUE);
```

CS31_DIAG



Parameter	Value
Included in library	Diag_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Function block with historical values
Group	-

The function block CS31_DIAG provides the diagnosis data of the CS31 bus to the user.

The instance diagCS31 of the function block CS31_DIAG is declared in the global variables list GL_AC500_Diagnosis. The integrated visualization of the function block accesses the variables of this instance.

In order to include the function block into an AC500 project, it is only necessary to call the function block instance. Example in ST:

diagCS31(EN := TRUE, COM := 1);

The read values are written into internal structures of the instance diagCS31.

Structure	TYPE	Assignment
strVisuData1	strCS31_DiagBus	Diagnosis data of the CS31 master
strVisuData	strCS31_DiagModule	Diagnosis data of all 31 CS31 software modules ARRAY[0..31] OF strCS31_DiagOneModule
CS31Mod	strCS31_DiagOneModule	Diagnosis data of one CS31 software module

The structure strCS31_DiagBus is composed as follows:

```
TYPE strCS31_DiagBus :
(* STRUCT info about CS31 bus *)
STRUCT
    iCS31BusState :     DINT;   (* Bit 0=1 waiting for slave
                                   Bit 1=1 initialization
                                   Bit 4=1 I/O data exchange
                                   := 19 all modules at bus *)
    iNumberModule :     DINT;   (* current no. of CS31 software modules on the CS31 bus *)
    iMaxNumberModule :  DINT;   (* max. number of modules at CS31 bus since power ON *)
    iErrorSum :         DINT;   (* sum error counter *)
    ulCycleCount :      DWORD;  (* counter of CS31 bus cycles *)
    byStateDiag :       BYTE;   (* general bus diagnosis: 0=ok, 1=no module on the bus *)
END_STRUCT
END_TYPE
```

The structure strCS31_DiagOneModule is composed as follows:

```
TYPE strCS31_DiagOneModule :
STRUCT
    dwScheduleCycle :   DWORD;  (* internal time *)
    byPhysicalAdress :  BYTE;   (* module address x 2 *)
    byCommState :       BYTE;   (* internal communication status:
    bySlaveType :       BYTE;   (* Slave type:
                                   00 = digital input
                                   01 = analog input
                                   02 = digital output
                                   03 = analog output
                                   04 = digital input/output
                                   05 = analog input/output
                                   06 = special module
```

	byInputBytes :	BYTE;	(* reported number of input bytes *)
	byOutputBytes :	BYTE;	(* reported number of output bytes *)
	byInputBytesConf :	BYTE;	(* configured number of input bytes *)
	byOutputBytesConf :	BYTE;	(* configured number of output bytes *)
	byModuleConf :	BYTE;	(* 1-PLC config, 3- ignore module *)
	byStateDiag :	BYTE;	(* Diagnostic Condition: each bit indicates a diagnosis type)
			byStateDiag = 0 : no diagnosis Bit 0 (0x01, 1) : a device that was present on the bus has been disconnected from the bus. Bit 1 (0x02, 2) : A configured device or module could not be found on the bus Bit 2 (0x04, 4) : A device or module that is not configured, was found on the bus Bit 3 (0x08, 8) : It was noted a difference between a configured and reported I / O peripheral. Bit 4 (0x10, 16) : A CS31 device (no device of the S500 series) has a new error message reported (short circuit or internal fault) Bit 5 (0x20, 32) : A CS31 device (no device of the S500 series) has reported a new error message Bit 6 (0x40, 64) : A CS31 device of the S500 series has reported a new error message. Bit 7 (0x80, 128) : <not used>
	byd1 :	BYTE;	(* internal variable *)
	usErrorCounter :	WORD;	(* Error counter for the module *)
	byDiagErr :	BYTE;	(* internal error number *)
	byDiagChannnel :	BYTE;	(* Channel in which the error has been has occurred *)
	byd2 :	BYTE;	(* internal variable *)
	byd3 :	BYTE;	(* internal variable *)

	pConfig :	DWORD;	(* internal variable *)
	END_STRUCT END_TYPE		

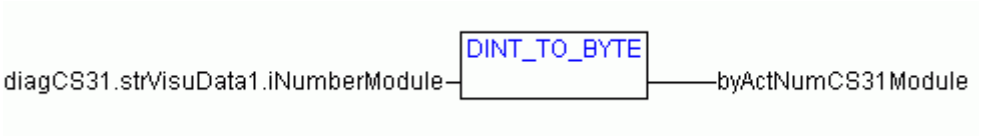
The structure strCS31_DiagModule contains an ARRAY[0..31], i.e. with 32 entries of the structure strCS31_DiagOneModule.

This way, the structures strVisuData1 and strVisuData provide access to all diagnosis data of the CS31 bus.

If the user program should have access to these values, this is done as follows:

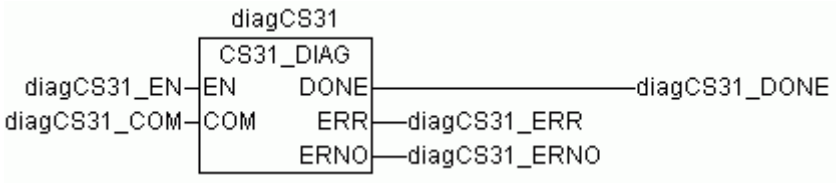
diagCS31.strVisuData1.Variable.

In the following example, the current number of CS31 modules is read and assigned to the variable byActNumCS31Module:



The most important data are available in the visualization Visu_CS31_Diag of the function block ↳ *Chapter 1.5.4.12.2 “Visualizations” on page 1182*.

Input description



EN

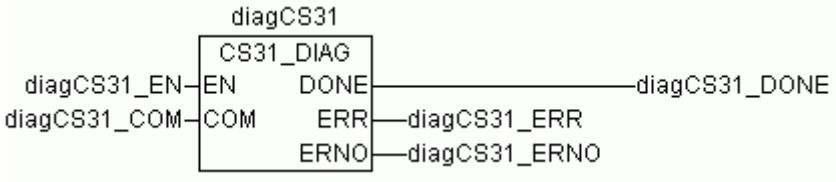
Data type	Default value	Range	Unit
BOOL			

EN = TRUE enables the processing of the function block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

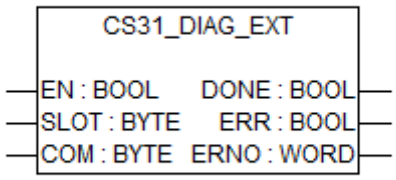
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
diagCS31(EN := diagCS31_EN, COM := diagCS31_COM);


diagCS31_DONE := diagCS31.DONE;
diagCS31_ERR  := diagCS31.ERR;
diagCS31_ERNO := diagCS31.ERNO;


or:


diagCS31(EN := TRUE, COM := 1);
```

CS31_DIAG_EXT



Parameter	Value
Included in library	Diag_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	-

Function block CS31_DIAG provides the diagnosis data of the CS31 Bus to the user.

The read values are written into internal structures of the instance.

Structure	TYPE	Assignment
strVisuData1	strCS31_DiagBus	Diagnosis data of the CS31 master
strVisuData	strCS31_DiagModule	Diagnosis data of all 31 CS31 software modules ARRAY[0..31] OF strCS31_DiagOneModule
CS31Mod	strCS31_DiagOneModule	Diagnosis data of one CS31 software module

The structure strCS31_DiagBus is composed as follows:

TYPE strCS31_DiagBus : (* STRUCT info about CS31 bus *) STRUCT			
	iCS31BusState :	DINT;	(* Bit 0=1 waiting for slave Bit 1=1 initialization Bit 4=1 I/O data exchange := 19 all modules at bus *)
	iNumberModule :	DINT;	(* current no. of CS31 software modules on the CS31 bus *)
	iMaxNumberModule :	DINT;	(* max. number of modules at CS31 bus since power ON *)
	iErrorSum :	DINT;	(* sum error counter *)
	ulCycleCount :	DWORD;	(* counter of CS31 bus cycles *)
	byStateDiag :	BYTE;	(* general bus diagnosis: 0=ok, 1=no module on the bus *)
END_STRUCT END_TYPE			

The structure strCS31_DiagOneModule is composed as follows:

TYPE strCS31_DiagOneModule : STRUCT			
	dwScheduleCycle :	DWORD;	(* internal time *)
	byPhysicalAdress :	BYTE;	(* module address x 2 *)
	byStateDiag :	BYTE;	(* communication status:

				00 = OK Bit 0=1 (1) module disconnected again Bit 1=1 (2) module not available on bus Bit 2=1 (4) unconfigured module on bus Bit 3=1 (8) difference in I/O range Bit 4=1 (16) E4 – error message Bit 5=1 (32) Bit 6=1 (64) *)
	bySlaveType :	BYTE;	(* Slave type:	
				00 = digital input 01 = analog input 02 = digital output 03 = analog output 04 = digital input/output 05 = analog input/output 06 = special module
	byInputBytes :	BYTE;	(* number of reported input bytes *)	
	byOutputBytes :	BYTE;	(* number of reported output bytes *)	
	byInputBytesConf :	BYTE;	(* number of input bytes in configuration *)	
	byOutputBytesConf :	BYTE;	(* number of output bytes in configuration *)	
	byModuleConf :	BYTE;	(* 1-PLC config, 3- ignore module *)	
	byStateDiag :	BYTE;	(* internal flag *)	
	byd1 :	BYTE;	(* internal flag *)	
	usErrorCounter :	WORD;	(* single-error counter *)	
	byDiagErr :	BYTE;	(* internal error number *)	
	byDiagChannnel :	BYTE;	(* internal error channel *)	
	byd2 :	BYTE;	(* internal flag *)	
	byd3 :	BYTE;	(* internal flag *)	
	pConfig :	DWORD;	(* internal flag *)	
	END_STRUCT END_TYPE			

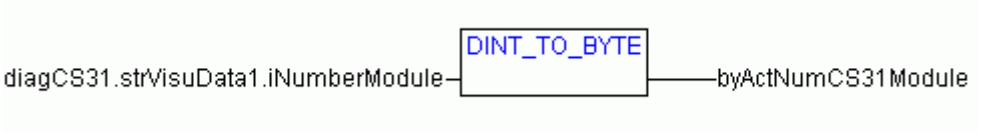
The structure strCS31_DiagModule contains an ARRAY[0..31], i.e. with 32 entries of the structure strCS31_DiagOneModule.

This way, the structures strVisuData1 and strVisuData provide access to all diagnosis data of the CS31 bus.

If the user program should have access to these values, this is done as follows:

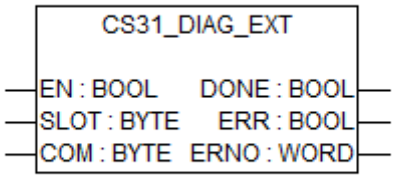
diagCS31ext.strVisuData1.Variable.

In the following example, the current number of CS31 modules is read and assigned to the variable byActNumCS31Module:



The most important data are available in the visualization Visu_CS31_Diag_EXT of the function block *Chapter 1.5.4.12.2 “Visualizations” on page 1182*.

Input description



EN

Data type	Default value	Range	Unit
BOOL			

EN = TRUE enables the processing of the function block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

SLOT

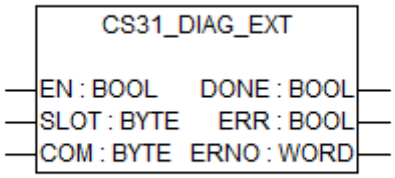
Data type	Default value	Range	Unit
BYTE	-	-	-

Reserved for future extensions.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

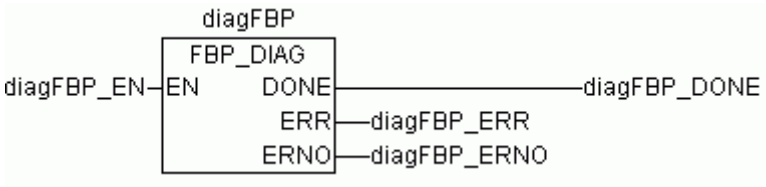
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
diagCS31EXT  (EN   := diagCS31EXT_EN,
              SLOT := diagCS31EXT_SLOT,
              COM  :=diagCS31EXT_COM);


diagCS31EXT_DONE   := diagCS31EXT.DONE;
diagCS31EXT_ERR    := diagCS31EXT.ERR;
diagCS31EXT_ERNO   := diagCS31EXT.ERNO;


or:


diagCS31EXT (EN   := TRUE,
             SLOT := 0,
             COM  := 1);
```

FBP_DIAG



Parameter	Value
Included in library	Diag_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Function block with historical values
Group	-

The function block FBP_DIAG is used to output the diagnosis data of the FBP slave interface.

The instance diagFBP of the function block DIAG_FBP is declared in the global variables list GL_AC500_Diagnosis. The integrated visualization of the function block accesses the variables of this instance.

In order to include the function block into an AC500 project, it is only necessary to call the function block instance. Example in ST:

diagFBP(EN := TRUE);

The read values are written into internal structures of the instance diagFBP.

Structure	**TYPE**	**Assignment**
strVisuData	strFBP_Info	Diagnosis data of the FBP slave interface
strVisuData1	strFBP_Statistics	Statistic data of the FBP slave interface

The structure strFBP_Info is composed as follows:

TYPE strFBP_Info : STRUCT			
	pbyBinInputs :	POINTER TO BYTE;	(* internal variable *)
	pbyBinOutputs :	POINTER TO BYTE;	(* internal variable *)
	pbyAnaInputs :	POINTER TO BYTE;	(* internal variable *)
	pbyAnaOutputs :	POINTER TO BYTE;	(* internal variable *)
	pParameter :	POINTER TO BYTE;	(* internal variable *)
	apsModule :	ARRAY[1..8] OF POINTER TO strFBP_ModuleInfo;	(* submodule info *)
	usNumModules :	WORD;	(* number of modules *)
	usSizeParameters :	WORD;	(* parameter length in bytes *)
	usNumParameters :	WORD;	(* number of parameters *)
	usNumBinaryInputs :	WORD;	(* number of digital input bytes *)
	usSizeBinaryInputs :	WORD;	(* length of digital input bytes *)
	usNumBinaryOutputs :	WORD;	(* number of digital output bytes *)
	usSizeBinaryOutputs :	WORD;	(* length of digital output bytes *)
	usNumAnalogueInputs :	WORD;	(* number of analog input words *)
	usSizeAnalogueInputs :	WORD;	(* length of analog inputs in bytes *)
	usNumAnalogueOutputs :	WORD;	(* number of analog output words *)
	usSizeAnalogueOutputs :	WORD;	(* length of analog outputs in bytes *)
	ulBaudRate :	DWORD;	(* current baud rate *)

	byBusAddress :	BYTE;	(* bus address reported to FBP *)
	bySelectedTelegramType :	BYTE;	(* telegram type (acc. to FBP spec) *)
	bySelectedBaudRate :	BYTE;	(* set baud rate (acc. to FBP spec) *)
	byPrmFormat :	BYTE;	(* parameter format (acc. to FBP spec) *)
END_STRUCT END_TYPE			

The structure strFBP_Statistics is composed as follows:

TYPE strFBP_Statistics : STRUCT			
	ulNumInitRequestsSent :	DWORD;	(* no. of Inits sent to FBP since configuration *)
	ulNumRec :	DWORD;	(* no. of telegrams received from FBP since config *)
	ulNumSend :	DWORD;	(* no. of telegrams sent to FBP since config *)
	ulNumRecErrors :	DWORD;	(* number of faulty telegrams since config *)
	ulNumTimeouts :	DWORD;	(* number of timeouts since configuration *)
	ulNumChecksumErrors :	DWORD;	(* number of telegrams with checksum error *)
	ulNumInvalidRecs :	DWORD;	(* number of wrong/unknown telegrams *)
END_STRUCT END_TYPE			

If the user program should have access to these values, this is done as follows:

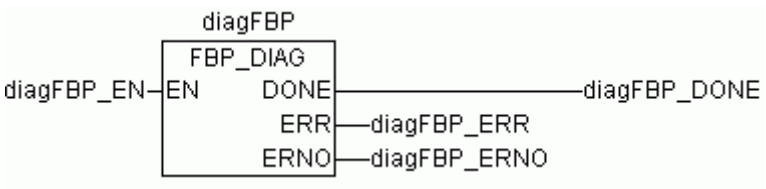
diagFBP.strVisuData.Variable.

In the following example, the FPB slave address is read and assigned to the variable byFBPAddress:

diagFBP.strVisuData.byBusAddress----------byFBPAddress

Using the visualization Visu_FBP_Diag, the values are represented graphically ↳ *Chapter 1.5.4.12.2 “Visualizations” on page 1182*.

Input description

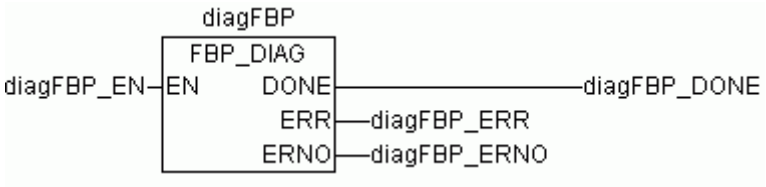


EN

Data type	Default value	Range	Unit
BOOL			

EN = TRUE enables the processing of the function block. With a FALSE -> TRUE edge at input EN, the last five entries are read from the diagnosis buffer of the CPU.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

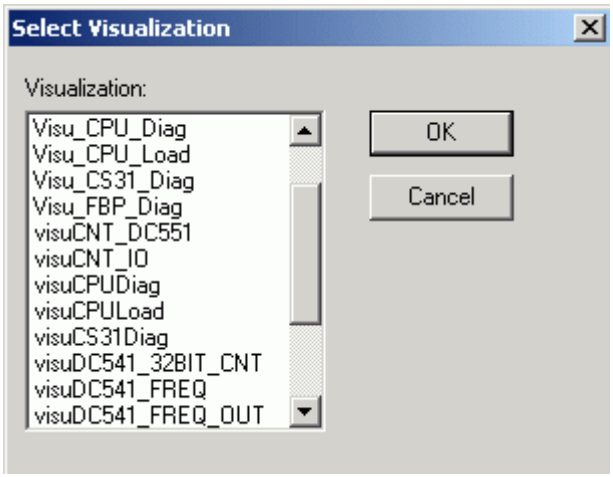
```
diagFBP(EN := diagFBP_EN);


diagFBP_DONE := diagFBP.DONE;
diagFBP_ERR  := diagFBP.ERR;
diagFBP_ERNO := diagFBP.ERNO;


or:


diagFBP(EN := TRUE);
```

1.5.4.12.2 Visualizations

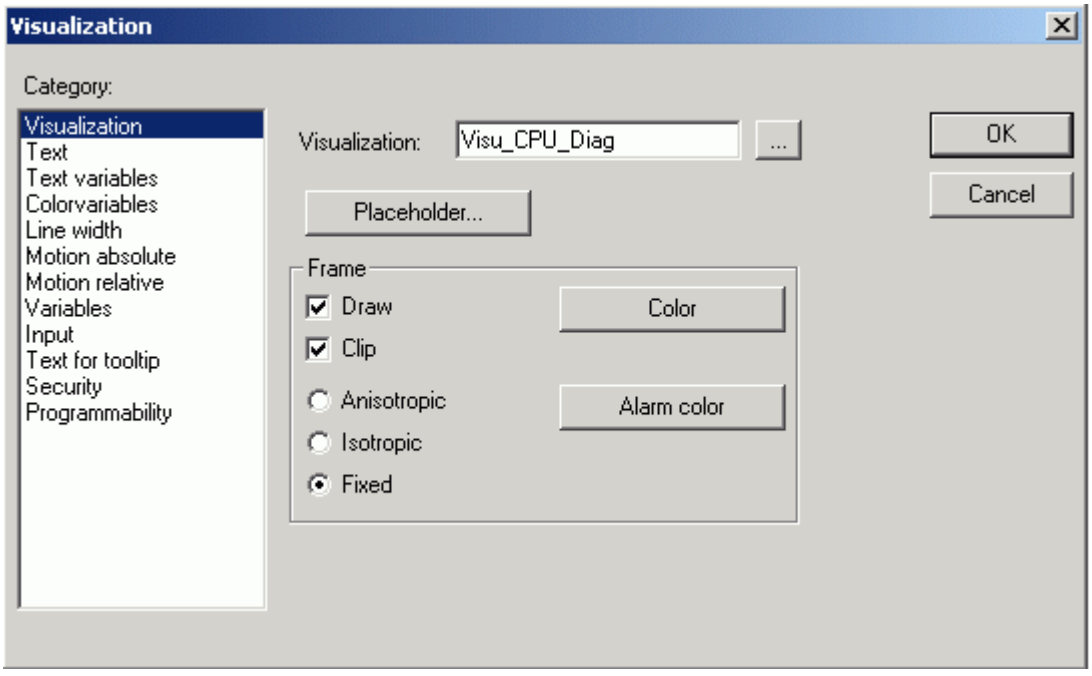
Group:	
Visu_CPU_Diag	Visualization of the CPU diagnosis
Visu_CPU_Load	Visualization of the CPU load
Visu_CS31_Diag	Visualization of the CS31 diagnosis
Visu_FBP_Diag	Visualization of the FBP slave interface diagnosis

Proceed as follows to integrate the visualization into a project:

- Create a new visualization using Visualizations / Insert Object (e.g. visuCPUDiag).
- Insert a visualization using 
- In the appearing dialog, select the corresponding visualization for the function block: e. g. Visu_CPU_Diag



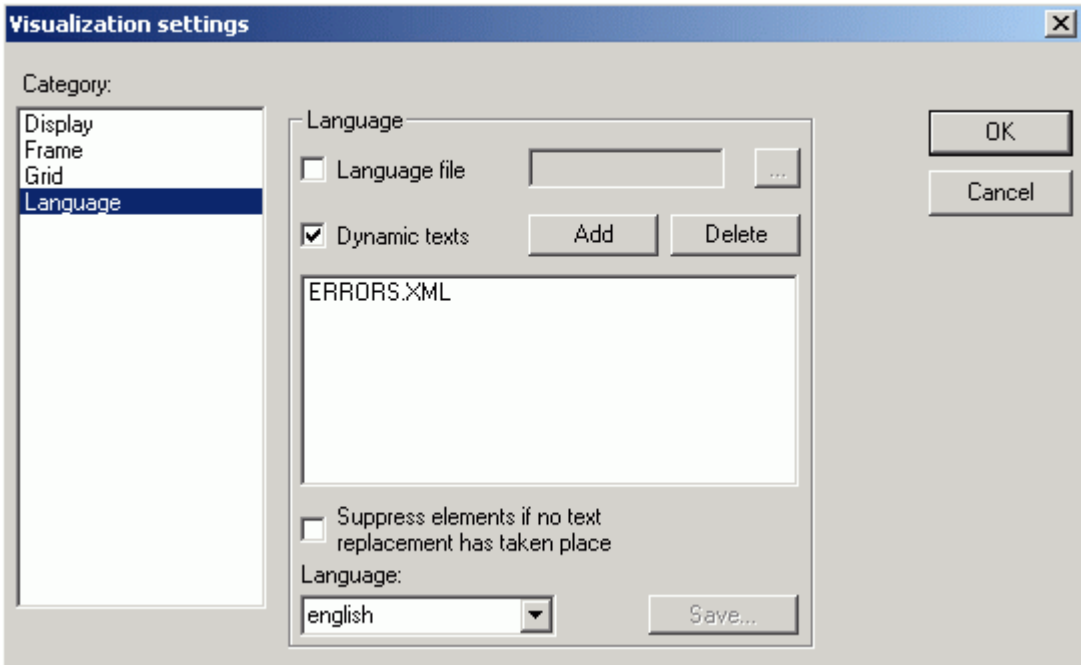
- Then, the visualization integrated just now has to be configured. Highlight the visualization with a left mouse click. Then click the right mouse button and select the function "Configure..." from the context menu.
- The configuration of the visualization is done in the appearing dialog. It is recommended to set the frame to "Fixed". This way, the original width-to-height ratio and font size are kept.



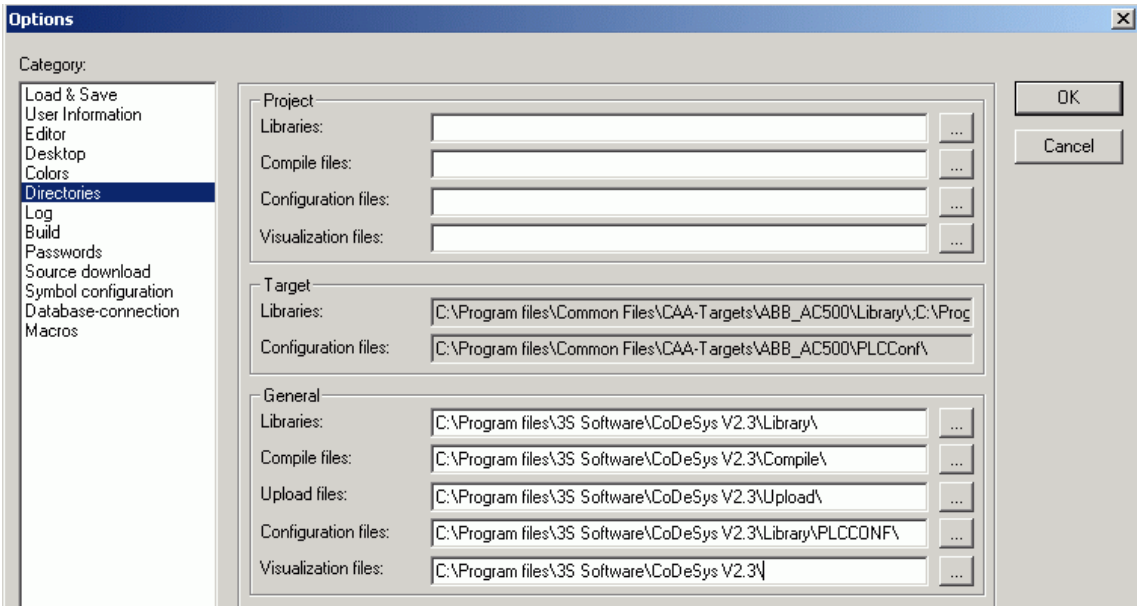
- By clicking on <OK> the dialogs are closed. After this, the inserted visualization has to be adapted to the correct size.

In the visualization of the function block CPU_DIAG, the diagnosis messages shall be output as language-dependent plain text. Proceed as follows to include the corresponding text file "Errors.xml":

- Switch to the visualization editor.
- Select the menu item "Extras"/"Settings". In the appearing dialog, select the category "Language", check "Dynamic texts" and enter "ERRORS.XML" as file name. Now, the desired language can be selected in the field "Language".



- Then, the path has to be entered in order to find the language file "ERRORS.XML". Select "Project"/"Options"/"Directories", go to "General"/"Visualization files" and enter the path CoDeSys.exe. In case of a default installation, this is:
  C:\Program Files\3S Software\CoDeSys V2.3 or C:\Programme\3S Software\CoDeSys V2.3.
  During installation of Automation Builder, the file ERRORS.XML is copied to this directory.



The following little example illustrates how to integrate diagnosis function blocks into a project and how to operate them via a central visualization.

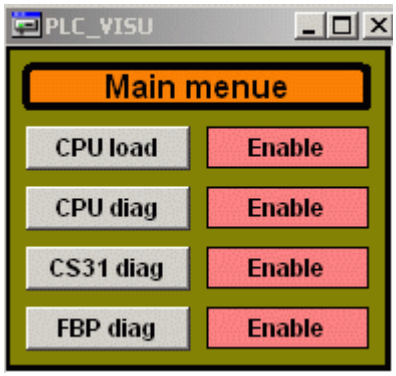
In the same way as described for the function block CPU_DIAG, the visualizations for the function block CPU_LOAD (visuCPULoad), CS31_DIAG (visuCS31Diag) and FBP_DIAG (visuFBP-Diag) have to be created.

By means of the visualization PLC_VISU, the individual visualizations can be launched and enabled or disabled. For that purpose, the program "PLC_Diagnosis()" is included into the AC500 project:

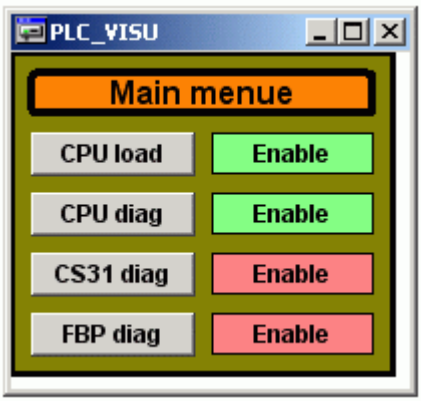
The program contains the following declarations and calls:

```
PROGRAM PLC_Diagnosis


VAR


bCPUEnable : BOOL := TRUE;
bCS31Enable : BOOL := FALSE;
bFBPEnable : BOOL := FALSE;
bCPUDiagEnable : BOOL := TRUE;


END_VAR


(* Call AC500 diagnosis *)
CPU(EN := bCPUEnable);
diagCPU(EN := bCPUDiagEnable);
diagCS31(EN := bCS31Enable, COM := 1);
diagFBP(EN := bFBPEnable);
```

In offline mode, the related visualization PLC_VISU looks as follows:



After enabling the CPU diagnosis and CPU load, PLC_VISU is displayed as follows in online mode:

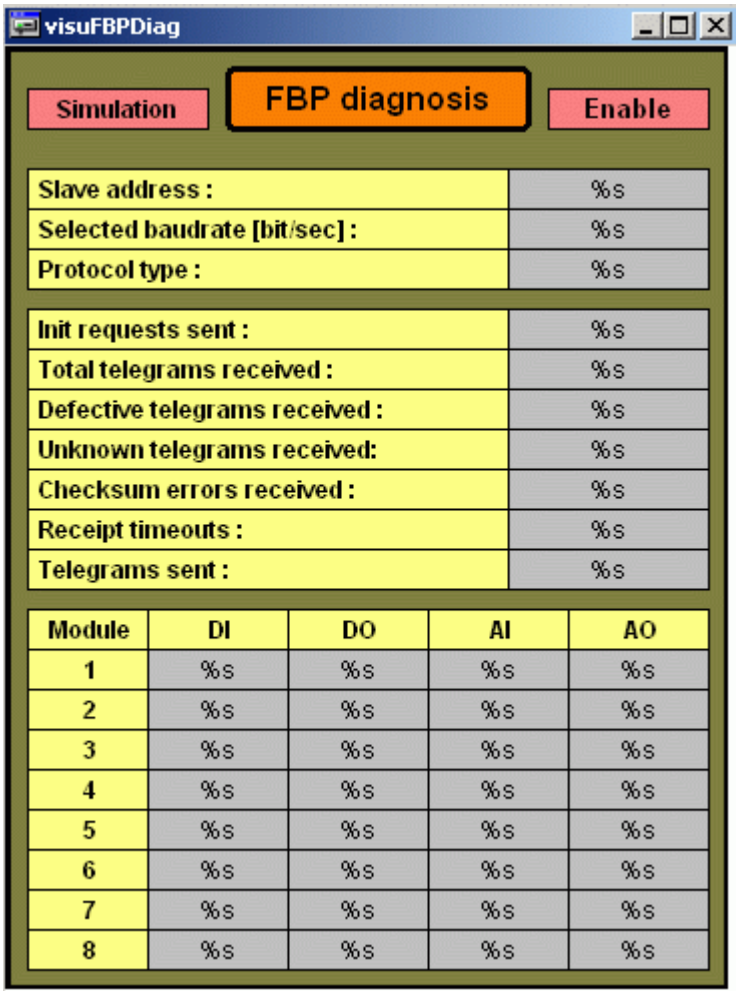


Using the buttons on the left side, the corresponding visualization is called. Clicking the "Enable" buttons enables the processing of the related function blocks.

The program "PLC_Diagnosis()" and the visualization PLC_VISU are contained in the export file AC500_PLC_Diagnosis.exp and can be inserted into an AC500 project via Project/Import.

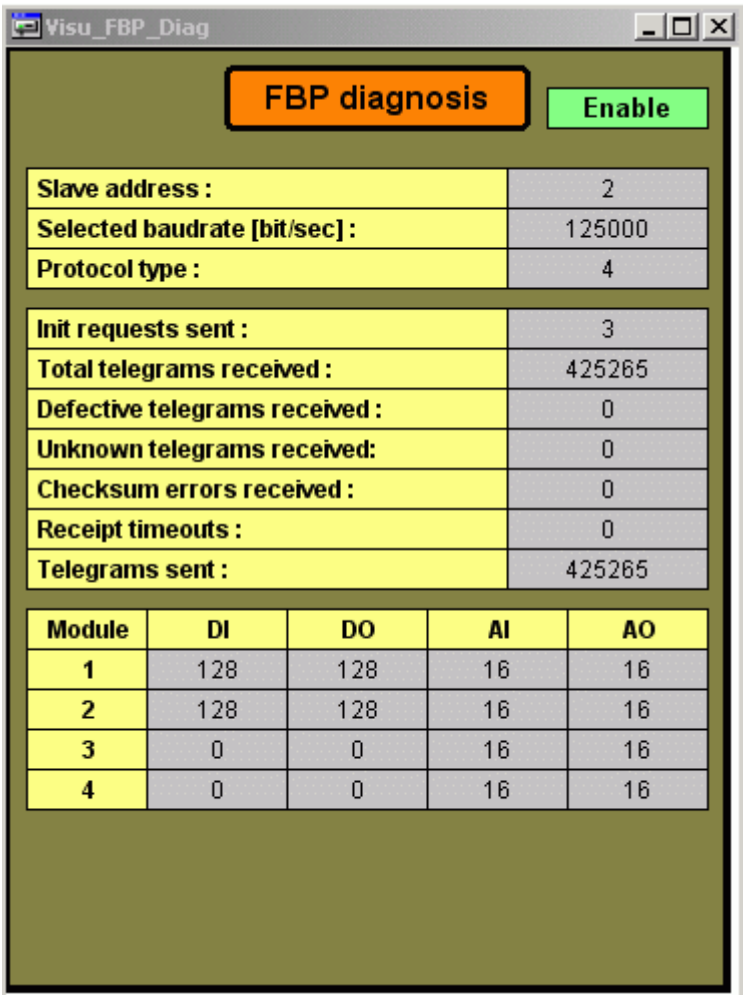
Integrated visualization of function block FBP_DIAG

The visualization Visu_FBP_Diag is part of the function block FBP_DIAG:



If input EN = TRUE, the values for the FBP diagnosis are read cyclically and entered into the structures.

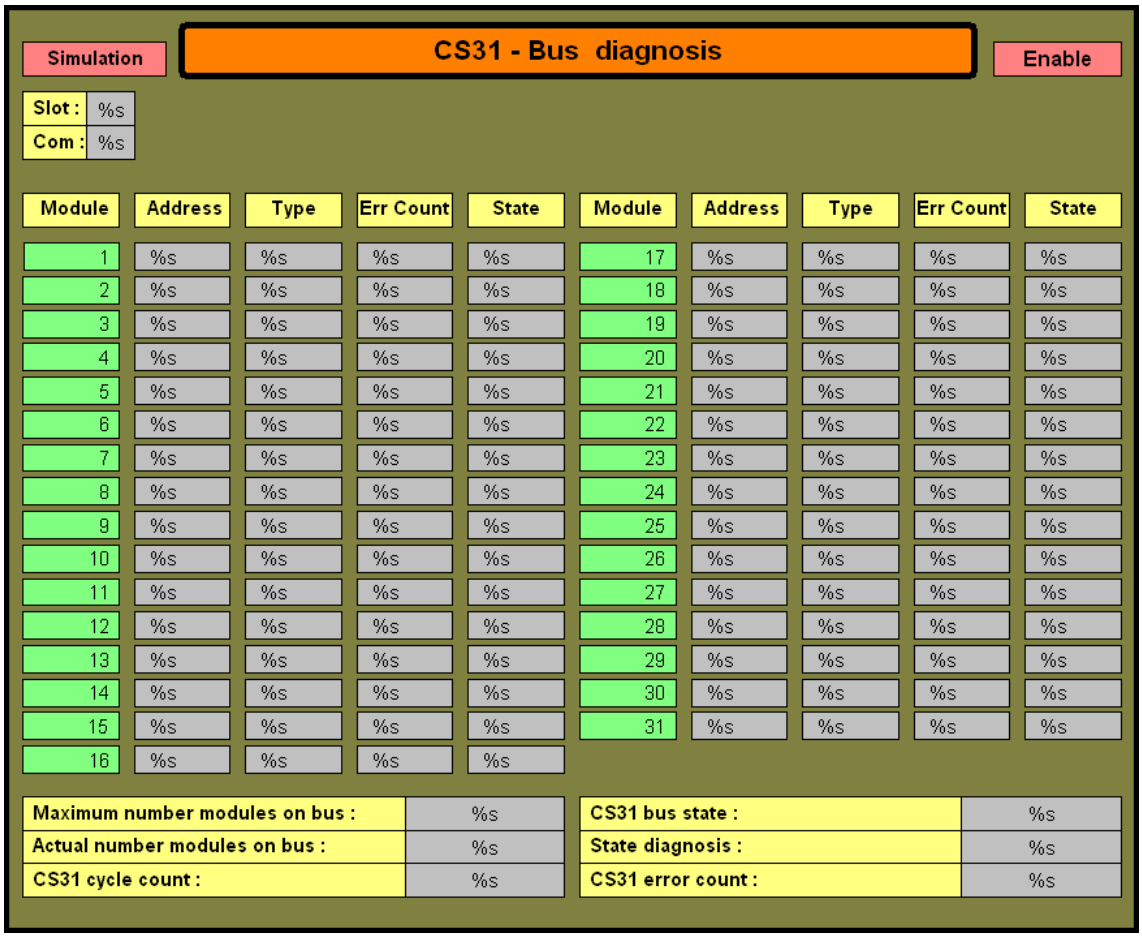
In online mode, the visualization appears, for example, as follows:



The visualization displays the number of modules that are connected to the FBP slave interface in the PLC configuration.

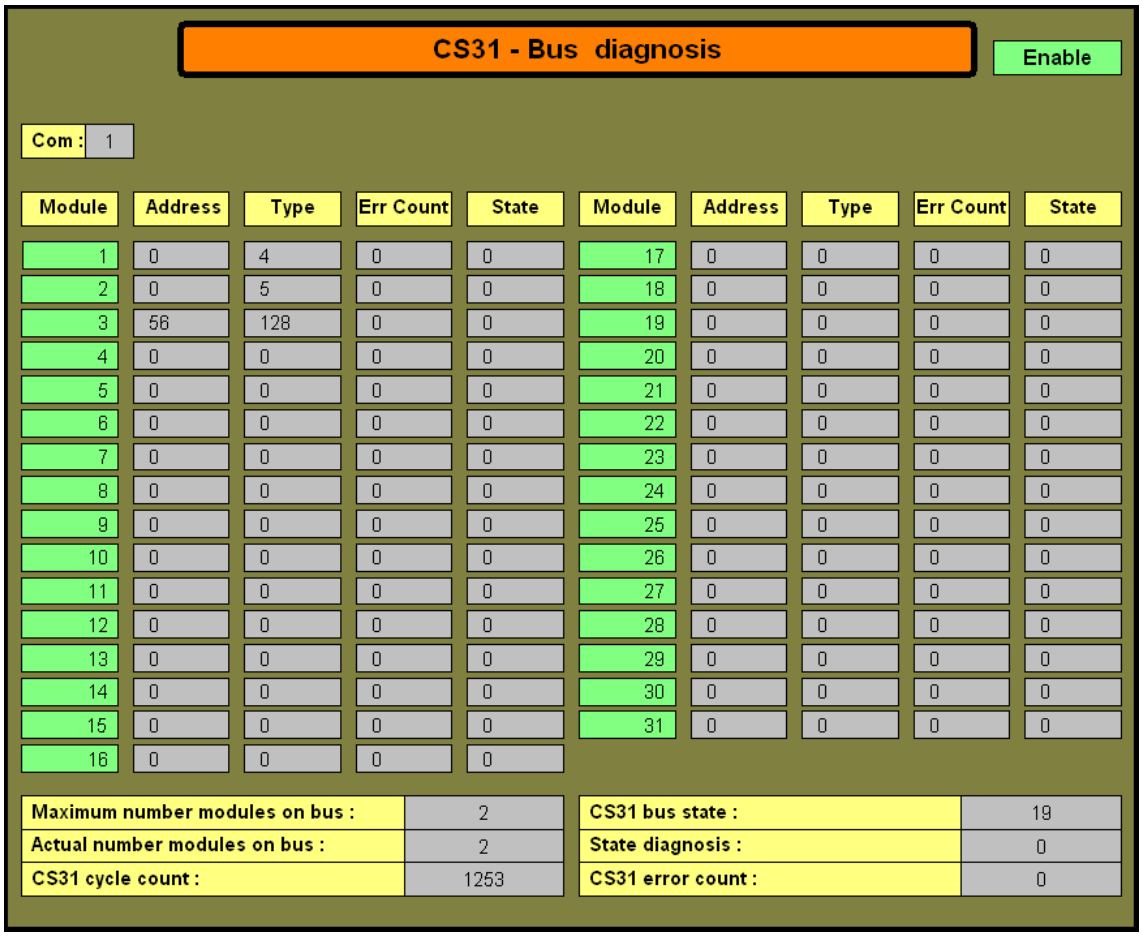
Integrated visualization of function block CS31_DIAG_EXT

The visualization Visu_CS31_Diag_EXT is part of the function block CS31_DIAG_EXT:



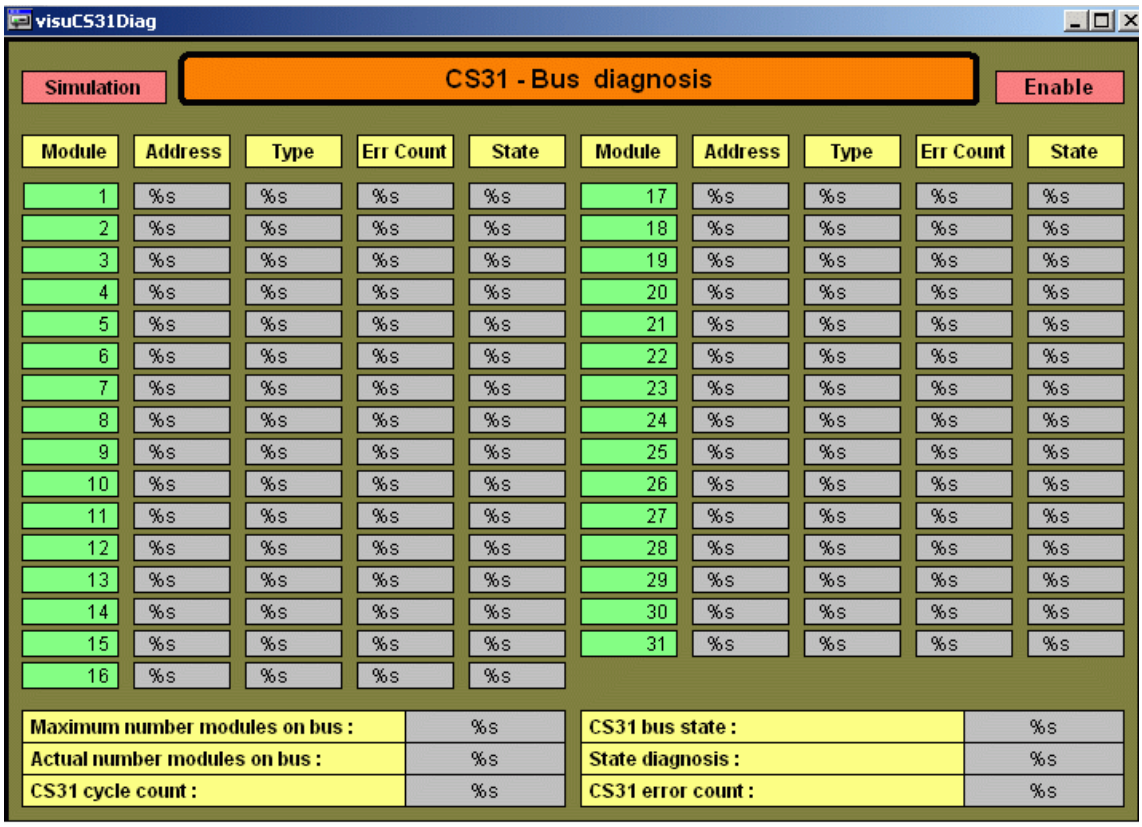
If input EN = TRUE, the values for the CS31 diagnosis are read cyclically and entered into the structures.

In online mode, the visualization appears, for example, as follows:



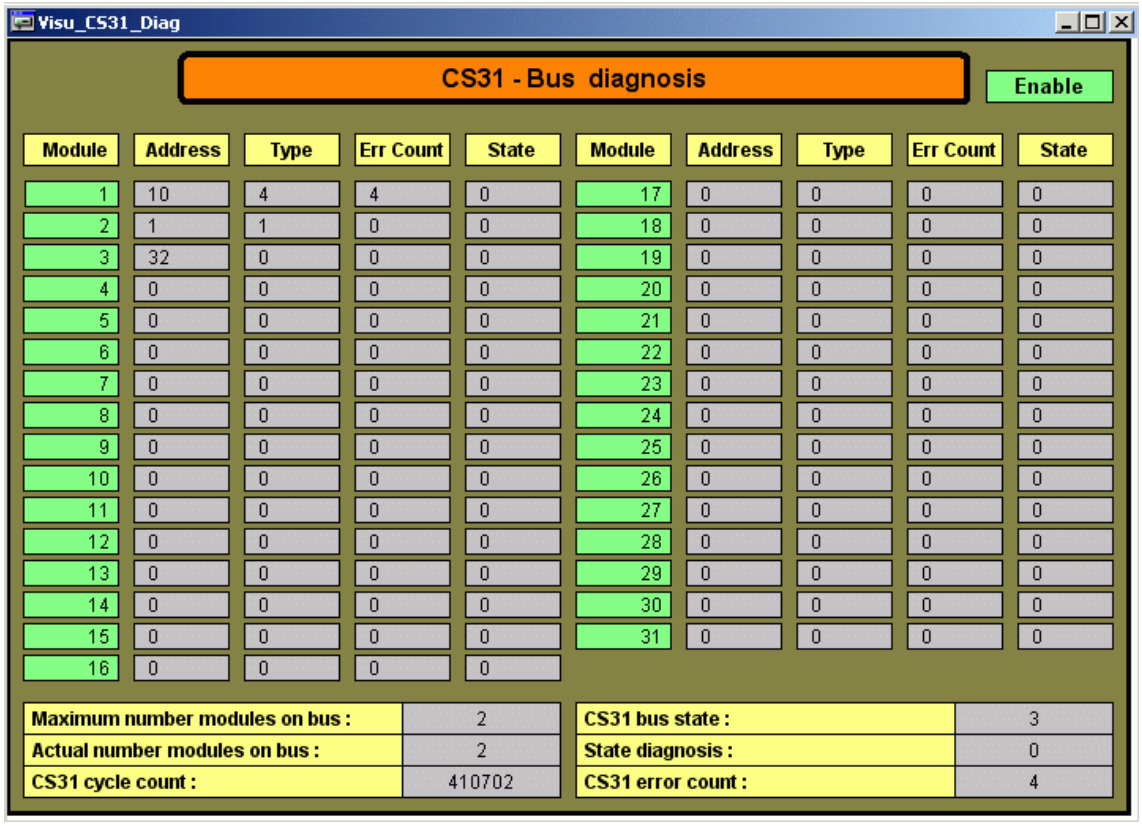
Integrated visualization of function block CS31_DIAG

The visualization Visu_CS31_Diag is part of the function block CS31_DIAG:



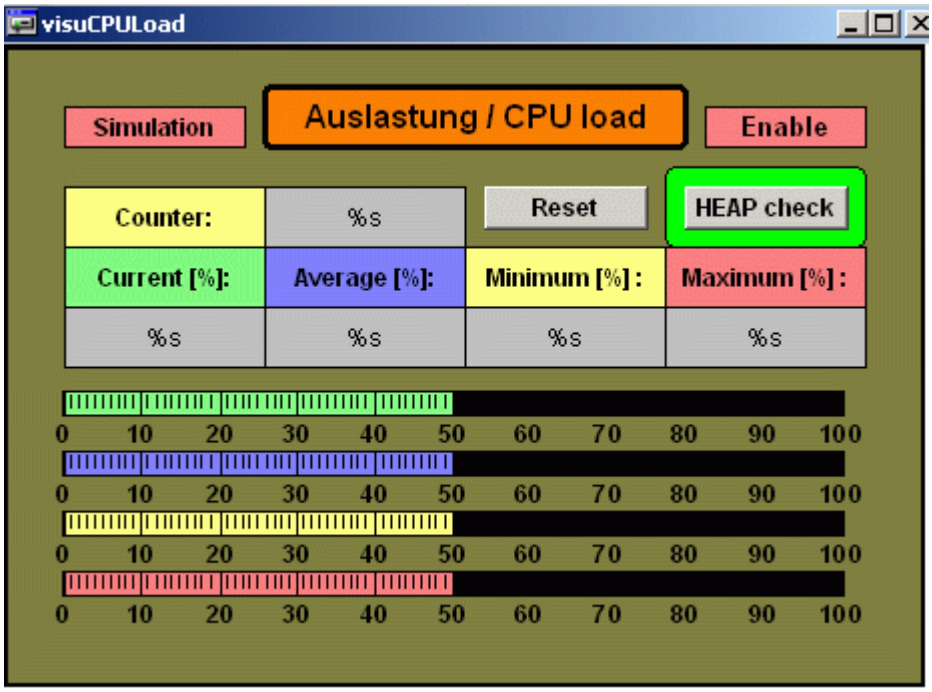
If input EN = TRUE, the values for the CS31 diagnosis are read cyclically and entered into the structures.

In online mode, the visualization appears, for example, as follows:



Integrated visualization of function block CPU_LOAD

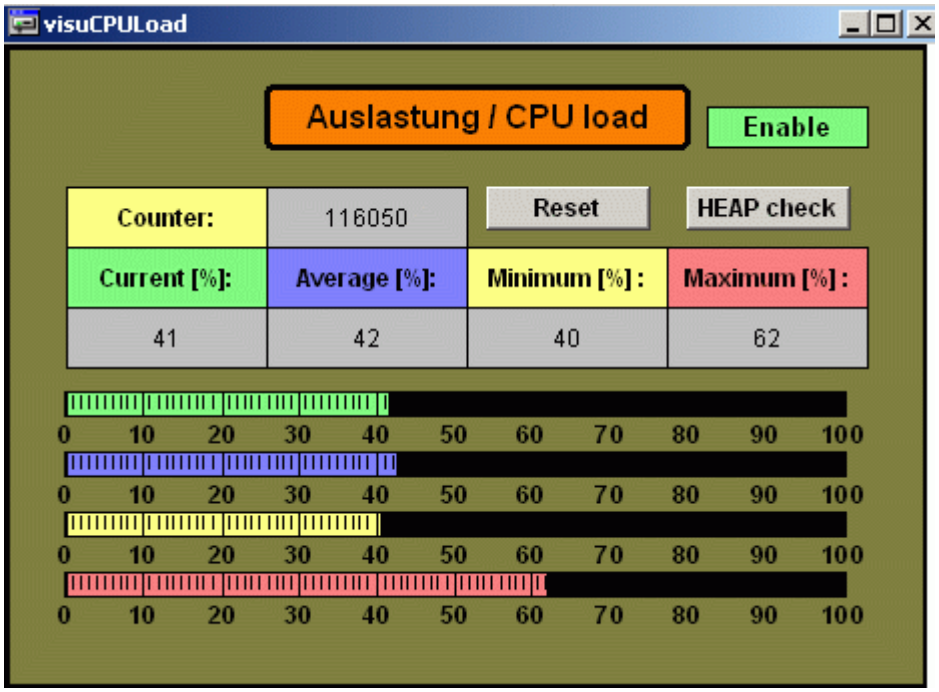
The visualization Visu_CPU_Load is part of the function block CPU_LOAD:



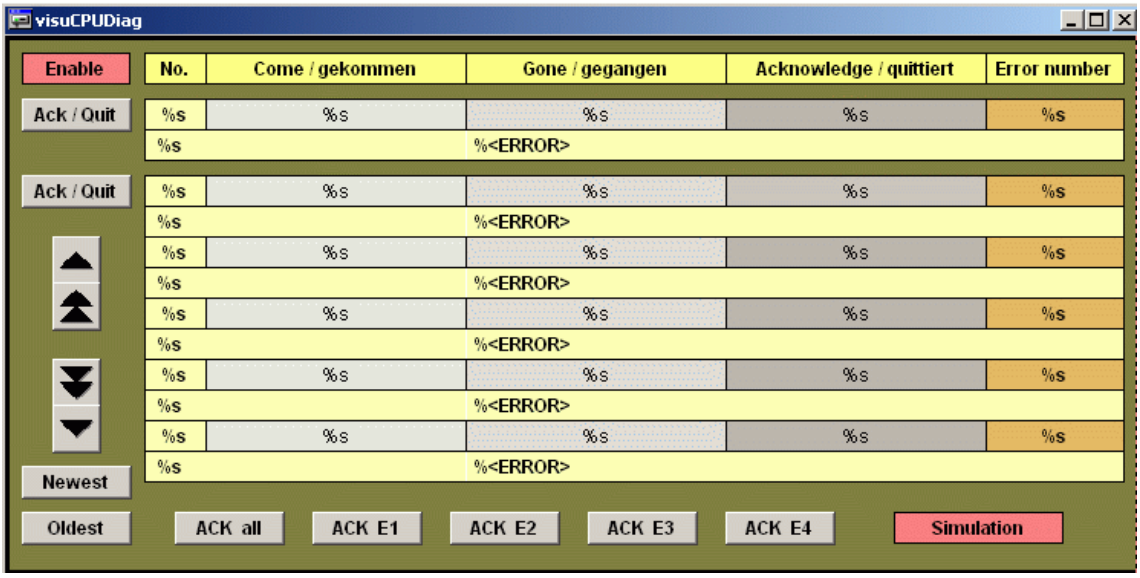
If input EN = TRUE, the CPU load values in [%] are cyclically output in a numerical and graphical (bar display) representation. Pressing the button <Reset> resets the values.

The button <HEAP check> checks the internal HEAP memory of the CPU. While the HEAP check is activated, no load values are calculated and output.

In online mode, the visualization appears, for example, as follows:



Integrated visualization of function block CPU_DIAG



This section describes how to operate the visualization.

The following is output for each diagnosis entry:

- current error number "No."
- time stamps for errror Come / Gone / Acknowledge
- "Error number"
- short text and error text taken from "ERRORS.XML"

In the upper area, the last diagnosis entry is displayed. This entry can be acknowledged by pressing the button <Ack / Quit>.

In the middle, the last 5 diagnosis entries are displayed after a FALSE/TRUE edge. The buttons on the left allow to navigate within the diagnosis buffer:

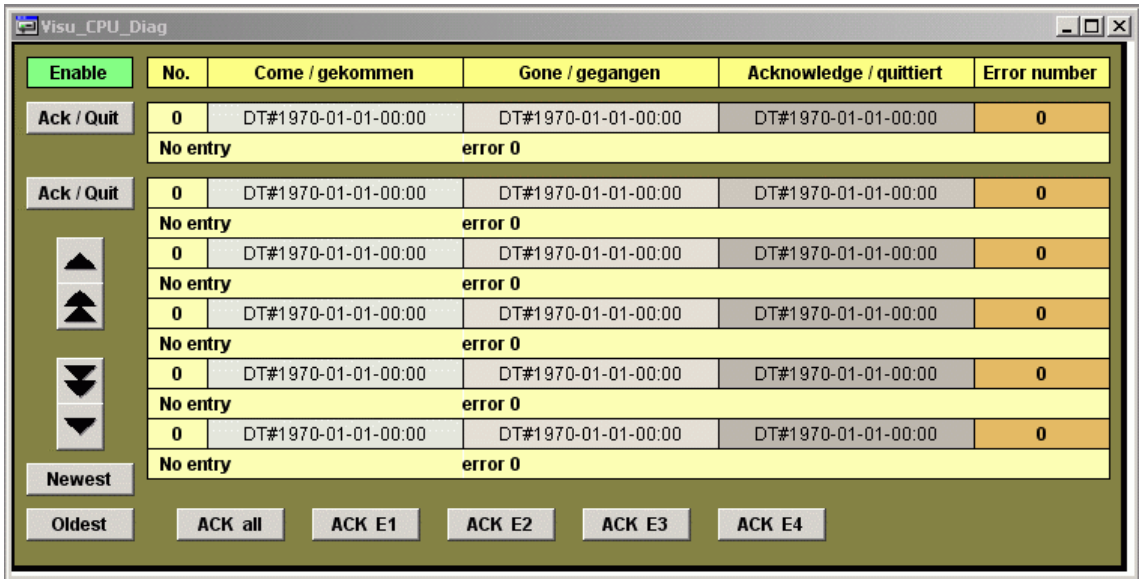
- <Ack / Quit>	- acknowledges the diagnosis entry displayed next to the button
- arrow up	- next entry
- arrow down	- previous entry
- double arrow up	- 5 entries forward
- double arrow down	- 5 entries back
- Newest	- 5 entries back starting with the latest entry
- Oldest	- 5 entries forward starting with the oldest entry

Using the buttons below the diagnosis entries, errors can be acknowledged:

- <ACK all>	- acknowledges all errors
- <ACK E1>	- acknowledges E1 errors (fatal errors)
- <ACK E2>	- acknowledges E2 errors (severe errors)
- <ACK E3>	- acknowledges E3 errors (minor error)
- <ACK E4>	- acknowledges E4 errors (warnings)

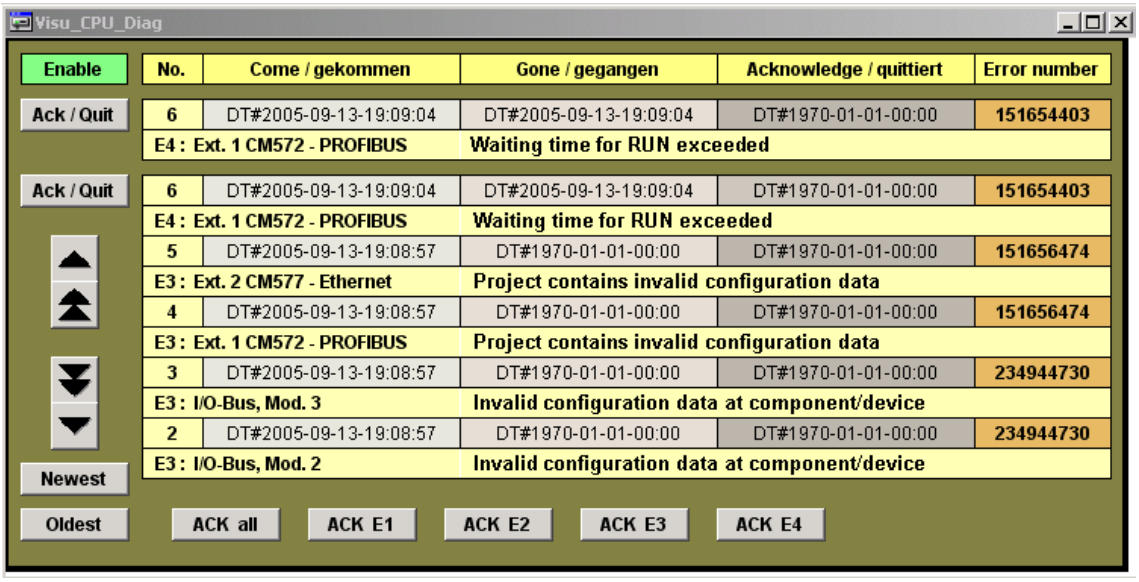
When acknowlegding errors, the function blocks DIAG_ACK and DIAG_ACK_ALL of the library SysInt_AC500_V10.lib are called.

If no diagnosis entries exist, the visualization in online mode looks as follows:



Remark: The PLC real-time clock (RTC) is not set in the example.

If errors exist, the visualization could, for example, appear as follows:



1.5.4.12.3 Structures

Group: Type_CPU_Diagnosis	
AC500_Diag_Entry	Structure of an AC500 diagnosis entry
strCPU_LOAD	Structure of the CPU load

Group: Type_CS31_Diagnosis	
strCS31_DiagBus	Diagnosis structure of CS31 master
strCS31_DiagModule	Diagnosis structure of all modules
strCS31_DiagOneModule	Diagnosis structure of a slave at the CS31 bus

Group: Type_FBP_Diagnosis	
strFBP_Info	Diagnosis structure of the FBP slave interface
strFBP_ModuleInfo	Diagnosis structure of a module of the FBP slave interface
strFBP_Statistics	Diagnosis structure with statistic values of the FBP slave interface

1.5.4.12.4 Global variables lists

Variables list: GL_AC500_Diagnosis		
Variable	**Type**	**Description**
CPU	CPU_LOAD	Instance of the function block CPU load
diagCPU	CPU_DIAG	Instance of the function block CPU diagnosis
diagCS31	CS31_DIAG	Instance of the function block CS31 diagnosis
diagFBP	FBP_DIAG	Instance of the function block FBP slave diagnosis

Variables list: GL_Diag_Constant		
Variable	**Type**	**Description**
wERNO_SIMULA-TION_MODE	WORD	16#50FF = 20735 dec, error number for simulation mode

1.5.4.13 Ethernet library

Library file name: **Ethernet_AC500_Vx.lib**

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

Operating the controller as open Modbus on TCP/IP subscriber can be performed simultaneously with other protocols. When operated in this mode, the Ethernet Communication Module is able to execute the functionality of several servers or several clients at the same time. Mixed operation is also possible.

In order to operate the controller as open Modbus on TCP/IP server (slave), only the Communication Module has to be configured. An additional use of the open Modbus on TCP/IP function blocks in the user program is not necessary.

To operate the controller as open Modbus on TCP/IP client (master), the Communication Module also has to be configured. In this case, one or more ETH_MODMAST function blocks have to be configured in the user program additionally.

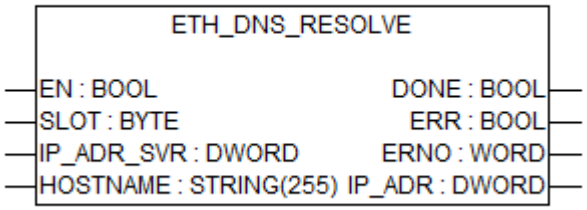
The ETH_MODMAST function block can be optionally operated in server mode as well as in client mode or in mixed operation.

Table 76: Reserved ports

Port		Reserved for
DEC	**HEX**	
32768	8000	Ethernet UDP/IP data exchange with AC31 header (ETH_UDP_xxx function blocks)
1200	04B0	TCP/IP gateway access
502	01F6	Open Modbus on TCP/IP

1.5.4.13.1 Function blocks

ETH_DNS_RESOLVE

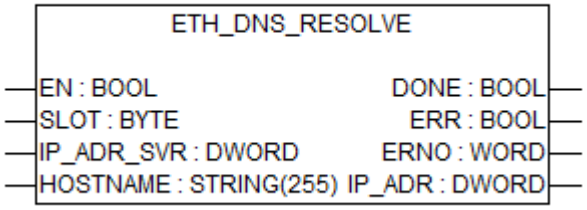


The function block ETH_DNS_RESOLVE returns an IP address of a user-specified hostname via a given DNS server.

The input SLOT is not considered by this function block.

Available as of runtime system:	V2.1	Remark:
Included in library:	Ethernet_AC500_V10.lib	Only supported with onboard Ethernet
Type	Function block with historical values	

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

IP_ADR_SVR

Data type	Default value	Range	Unit
DWORD	-	-	-

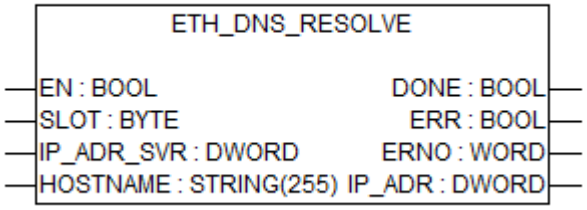
At input IP_ADR_SVR, the IP address of the DNS server to be used for resolving the hostname is specified.

HOSTNAME

Data type	Default value	Range	Unit
STRING	Empty string	-	-

At input HOSTNAME, the domain of the host to be resolved to an IP address is specified. This name is sent ot the DNS server, which answers with an IP address.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output IP_ADR displays the resolved IP address of the hostname provided by the input HOST-NAME. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

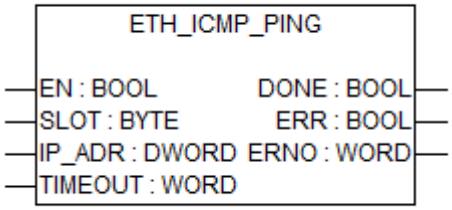
IP_ADR (dec) 3222214658

Function call in ST

```
DnsResolve    (EN         := DnsResolve_EN,
               SLOT       := DnsResolve_SLOT,
               IP_ADR_SVR := DnsResolve_IP_ADR_SVR,
               HOSTNAME   := DnsResolve_HOSTNAME);
```

```
DnsResolve_DONE   := DnsResolve.DONE;
DnsResolve_ERR    := DnsResolve.ERR;
DnsResolve_ERNO   := DnsResolve.ERNO;
DnsResolve_IP_ADR := DnsResolve.IP_ADR;
```

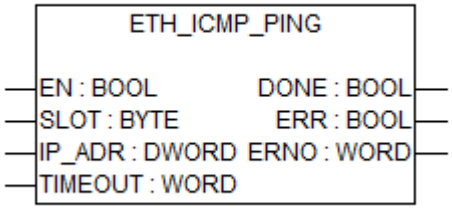
ETH_ICMP_PING



This function block only works on configured onboard Ethernet Modules, starting from CPU firmware V2.1. If no onboard Ethernet Communication Module is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.1	Remark:
Included in library:	Ethernet_AC500_V10.lib	Only supported on onboard Ethernet communication modules
Type	Function block with historical values	

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

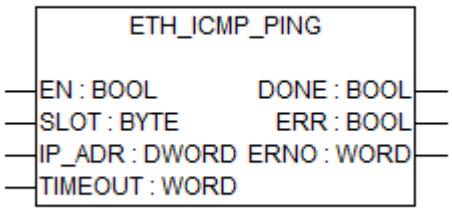
At input IP_ADR, the IP address of the host to be pinged is specified.

TIMEOUT

Data type	Default value	Range	Unit
WORD	5.000	-	ms

At input TIMEOUT, the time to wait for an ICMP echo reply from the host is specified. After this time has run out, an error will be set as output.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

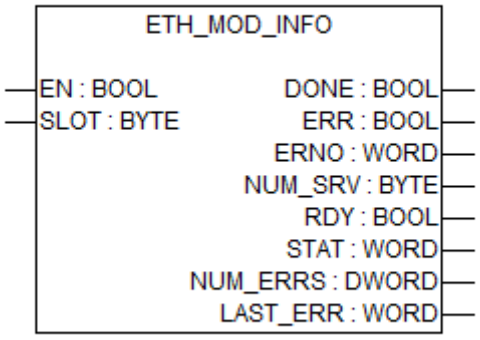
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
Ping
 (EN        := Ping_EN,
 SLOT       := Ping_SLOT,
 IP_ADR     := Ping_IP_ADR
 TIMEOUT   := Ping_TIMEOUT)


Ping_DONE  := Ping.DONE;
Ping_ERR   := Ping.ERR;
Ping_ERNO  := Ping.ERNO;
```

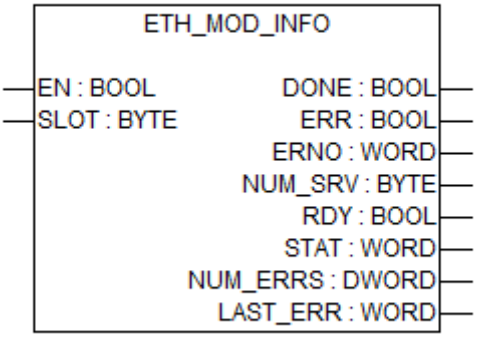
ETH_MOD_INFO



The function block ETH_MOD_INFO reads the status information of the open Modbus on TCP/IP processing. It can be used for pure server (slave) or client (master) operation of the controller as well as for mixed operation.

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

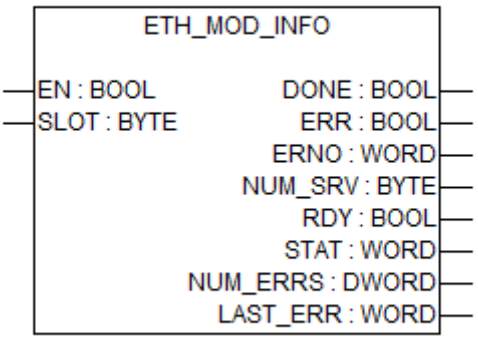
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SRV

Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_SRV (number of servers) output indicates the number of parallel server channels configured with Automation Builder software. NUM_SRV is only valid if DONE = TRUE and ERR = FALSE.

RDY

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

RDY (ready) output indicates the readiness for operation of the open Modbus on TCP/IP processing. If RDY = TRUE, the server processing as well as the client processing are ready for operation. RDY is only valid if DONE = TRUE and ERR = FALSE.

STAT

Output STAT displays the current operating state of the open Modbus on TCP/IP processing. STAT is only valid if DONE = TRUE and ERR = FALSE.

STAT		Description
Decimal	Hexadecimal	
0	00	Processing not initialized
1	01	Processing initialized and running
2	02	Processing initialization in progress
3	03	Initialization error
4	04	Processing initialized and waiting for TCP task

NUM_ERRS

Data type	Default value	Range	Unit
DWORD	-	-	-

The total number of errors detected by the Communication Module since last power up or reset.

LAST_ERR

Data type	Default value	Range	Unit
WORD	-	-	-

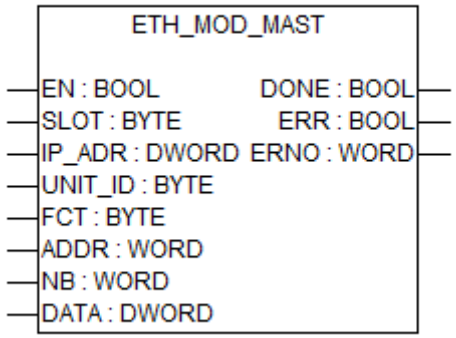
LAST_ERR outputs the last error occurred on the Communication Module. The error message encoding at output LAST_ERR applies to all Ethernet function blocks and is explained at the beginning of the library descriptions.

The outputs gives the value which has been received directly from the Communication Module. To find this error code inside the list of error messages, the value 0x6000 has to be added to the error value.

Function call in ST

```
ModInfo (EN := ModInfo_EN,
     SLOT := ModInfo_SLOT);


ModInfo_DONE := ModInfo.DONE;
ModInfo_ERR := ModInfo.ERR;
ModInfo_ERNO := ModInfo.ERNO;
ModInfo_NUM_SRV := ModInfo.NUM_SRV;
ModInfo_RDY := ModInfo.RDY;
ModInfo_STAT := ModInfo.STAT;
ModInfo_NUM_ERRS := ModInfo.NUM_ERRS;
ModInfo_LAST_ERR := ModInfo.LAST_ERR;
```

ETH_MOD_MAST



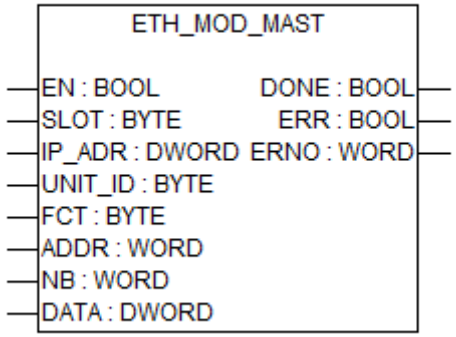
The ETH_MOD_MAST function block implements the open Modbus on TCP/IP client functionality for the Ethernet Communication Module specified at input SLOT. Depending on the configuration of the Communication Module, several ETH_MOD_MAST function blocks can be used in parallel. Prior to the use of ETH_MOD_MAST the Communication Module has to be configured.

With each FALSE > TRUE edge at input EN, the function block ETH_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and then sends this telegram to the slave.

> *Only a single instance per slave can be used with this function block.*

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

At IP_ADR, the IP address of the server has to be specified to which the telegram should be sent. Each byte in IP_ADR represents one octet of the address.

Example	IP address 192.15.24.2 IP_ADR (hex) 16#C00F1802 IP_ADR (dec) 3222214658

UNIT_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

At input UNIT_ID, the address of the Modbus slave must be specified which is connected serially to the Modbus server defined by IP_ADR. If no further slaves are connected, this input is not used.

FCT

Data type	Default value	Range	Unit
BYTE	-	-	-

The function code of the request telegram is specified at input FCT.

01 or 02	Read n bits
03 or 04	Read n words
05	Write one bit (encoded in one word)
06	Write one word
07	Read 8 bit
15	Write n bits (encoded in one byte)
16	Write n words
22	Mask write
23	Read/write multiple words in one telegram

The max. telegram length for onboard Ethernet CPUs with function code 3/4 is 96 words.

The max. telegram length for onboard Ethernet CPUs with function code 15 is 1536 bits.

ADDR

Data type	Default value	Range	Unit
WORD	-	-	-

The operand/register address in the slave from which data should be read or written is specified at input ADDR.

The access to operands of AC500 devices in Modbus slave mode is defined via the Modbus cross-reference list. Only operands that are listed in the cross-reference list may be used (↯ *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*).

Only operands that are listed in the Modbus address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

NB

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input NB (number), the number of data to be written or read is specified. The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB. The following restrictions apply to the length:

FCT		NBmax	
Dec	**Hex**	**Serial**	**Modbus on TCP/IP**
01 or 02	01 or 02	2000 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
03 or 04	03 or 04	125 words / 62 double words	125 words / 62 double words
05	05	1 bit	1 bit
06	06	1 word	1 word
07	07	8 bits	8 bits
15	0F	1968 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
16	10	123 words / 61 double words	123 words / 61 double words
22	16	Write: 1 word	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 123 words / 61 double words	Read: 125 words / 62 double words Write: 123 words / 61 double words

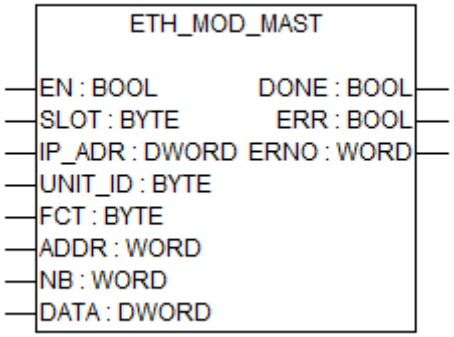
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

At input DATA, the address of the first operand in the master is specified, from which data are copied/written to the slave or to which the data read by the slave should be stored. For this purpose it is necessary that the operand type (e.g. bit) matches the selected function (e.g. FCT 1, read n bits).

If using Modbus function codes 22 or 23, the according data structures COM_MOD_FCT22_TYPE ↳ *Chapter 1.5.4.22.2.1 “COM_MOD_FCT22_TYPE” on page 1702* or COM_MOD_FCT23_TYPE ↳ *Chapter 1.5.4.22.2.2 “COM_MOD_FCT23_TYPE” on page 1703* must be defined and applied to DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	-	-

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error occurred during Function Block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO

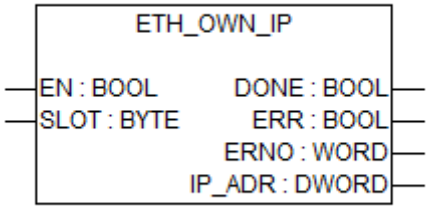
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ModMast (EN := ModMast_EN,
         SLOT := ModMast_SLOT,
         IP_ADR := ModMast_IP_ADR,
         UNIT_ID := ModMast_UNIT_ID,
         FCT := ModMast_FCT,
         ADDR := ModMast_ADDR,
         NB := ModMast_NB,
         DATA := ADR(ModMast_DATA));


ModMast_DONE := ModMast.DONE;
ModMast_ERR := ModMast.ERR;


ModMast_ERNO := ModMast.ERNO;
```

ETH_OWN_IP

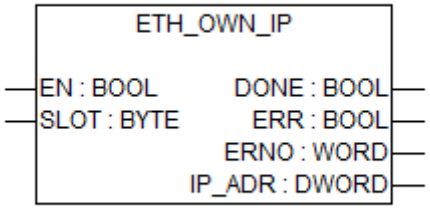


The function block ETH_OWN_IP outputs the IP address of the Communication Module installed at slot SLOT. If no Ethernet Communication Module is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

> *If using this function block in a loop (e.g. while), a sleep command must be inserted.*

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

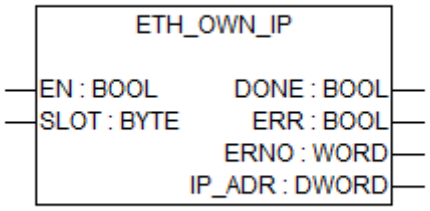
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

IP_ADR displays the IP address which should be set at the CPU / Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

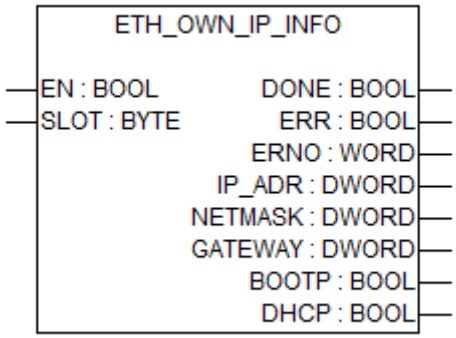
IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

Function call in ST

```
OwnIp(EN := OwnIp_EN,
      SLOT := OwnIp_SLOT);


OwnIp_DONE := OwnIp.DONE;
OwnIp_ERR := OwnIp.ERR;
OwnIp_ERNO := OwnIp.ERNO;
OwnIp_IP_ADR := OwnIp.IP_ADR;
```

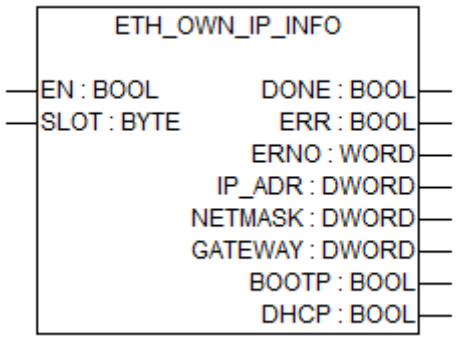
ETH_OWN_IP_INFO



The function block ETH_OWN_IP_INFO outputs the IP setup of the communication module installed at slot SLOT.If no Ethernet communication module is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V1.3
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

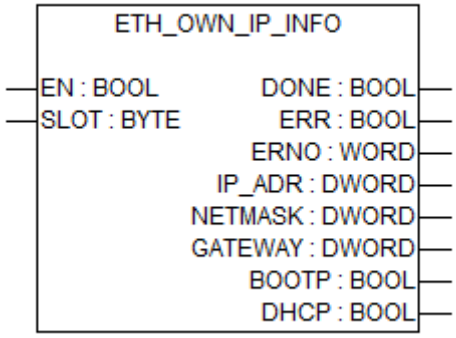
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

IP_ADR displays the IP address which should be set at the CPU / Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

NETMASK

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output NETMASK displays the own netmask. Each byte in NETMASK represents one octet of the address.

Example

Netmask 255.255.255.0

NETMASK (hex) 16#FFFFFF00

NETMASK (dec) 4294967040

GATEWAY

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output GATEWAY displays the IP address of the gateway. Each byte in GATEWAY represents one octet of the address.

Example

IP address 192.168.0.1

IP_ADR (hex) 16#C0A80001

IP_ADR (dec) 3232235521

BOOTP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output BOOTP indicates whether the BOOTP service is activated for the Communication Module.

DHCP

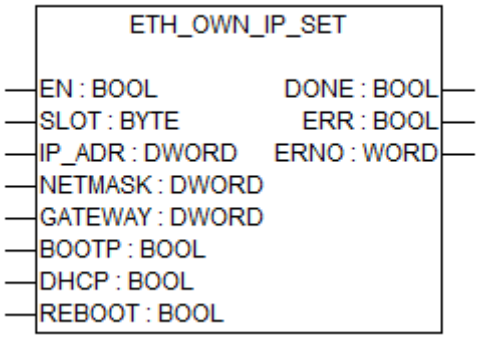
Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DHCP indicates whether the DHCP service is activated for the Communication Module.

Function call in ST

```
IpInfo(EN := IpInfo_EN,
       SLOT := IpInfo_SLOT);


IpInfo_DONE := IpInfo.DONE;
IpInfo_ERR := IpInfo.ERR;
IpInfo_ERNO := IpInfo.ERNO;
IpInfo_IP_ADR := IpInfo.IP_ADR;
IpInfo_NETMASK := IpInfo.NETMASK;
IpInfo_GATEWAY := IpInfo.GATEWAY;
IpInfo_BOOTP := IpInfo.BOOTP;
IpInfo_DHCP := IpInfo.DHCP;
```

ETH_OWN_IP_SET



Function block ETH_OWN_IP_SET is used to set

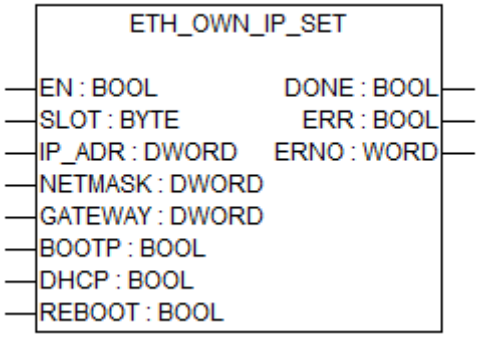
- IP address
- Subnet mask
- Gateway and
- some other parameters

at the CPU / Communication Module installed at slot SLOT.

> *The new IP configuration data will be available after the next reboot of the CPU / Communication module. Use input REBOOT=TRUE to restart the CPU / Communication Module immediately after changing the IP configuration data.*

Available as of runtime system:	V2.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

IP_ADR displays the IP address which should be set at the CPU / Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

NETMASK

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input NETMASK, the subnet mask which should be set at the CPU / Communication Module is specified. Each byte in NETMASK represents one octet of the address.

Example	Netmask 255.255.255.0 NETMASK (hex) 16#FFFFFF00 NETMASK (dec) 4294967040

GATEWAY

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input GATEWAY, the default gateway which should be set at the CPU / Communication Module is specified. Each byte in GATEWAY represents one octet of the address.

Example	IP address 192.168.0.1 IP_ADR (hex) 16#C0A80001 IP_ADR (dec) 3232235521

BOOTP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output BOOTP indicates whether the BOOTP service is activated for the Communication Module.

DHCP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

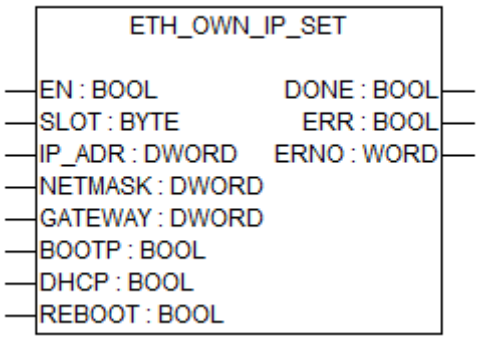
Output DHCP indicates whether the DHCP service is activated for the Communication Module.

REBOOT

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

A TRUE signal at input REBOOT restarts the CPU / Communication Module after changing the IP settings.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

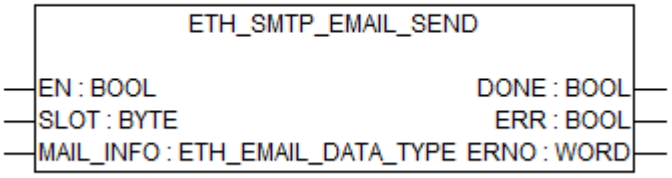
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
IPSet      (EN        := IPSet_EN,
            SLOT      := IPSet_SLOT,
            IP_ADR    := IPSet_IP_ADR,
            NETMASK   := IPSet_NETMASK,
            GATEWAY   := IPSet_GATEWAY,
            BOOTP     := IPSet_BOOTP,
            DHCP      := IPSet_DHCP,
            REBOOT    := IPSet_REBOOT);


IPSet_DONE := IPSet.DONE;
IPSet_ERR  := IPSet.ERR;
IPSet_ERNO := IPSet.ERNO;
```

ETH_SMTP_EMAIL_SEND

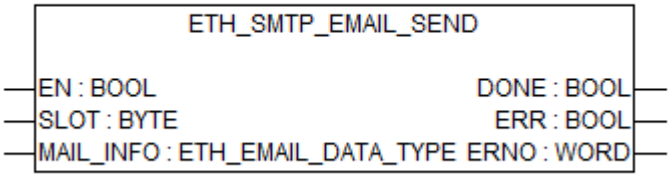


The function block ETH_SMTP_EMAIL_SEND uses a user-specified SMTP server to deliver an email to given mailboxes.

This function block only works on configured onboard Ethernet Modules. If no onboard Ethernet Communication Module is installed at SLOT, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.1	Remark:
Included in library:	Ethernet_AC500_V10.lib	Only supported with onboard Ethernet
Type	Function block with historical values	

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

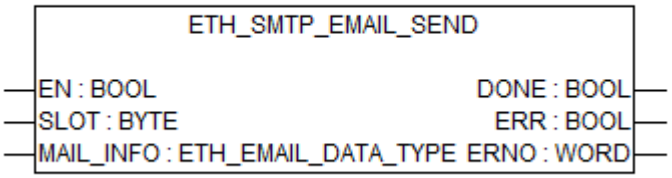
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

MAIL_INFO

Data type: ETH_EMAIL_DATA_TYPE (slot)

Input MAIL_INFO provides all necessary data to send an email via the data structure ETH_EMAIL_DATA_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

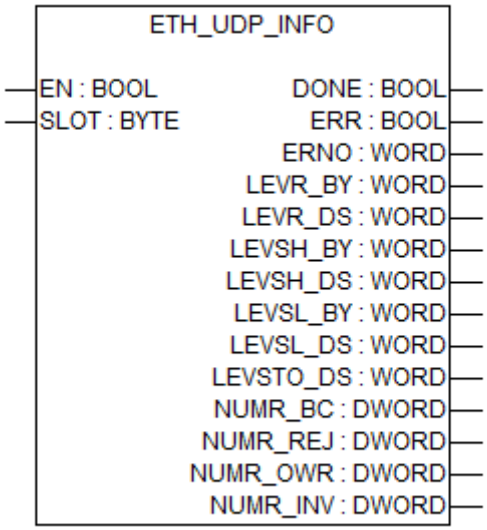
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

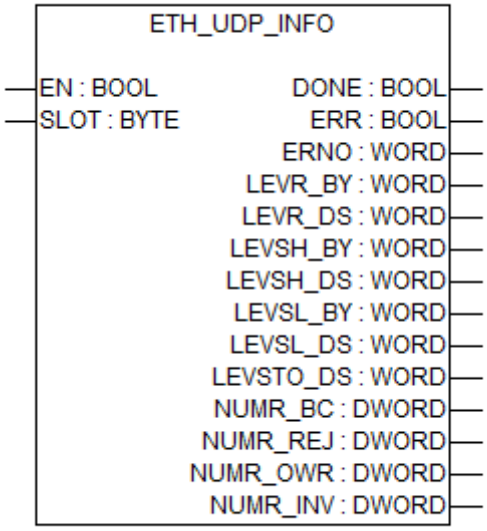
ETH_UDP_INFO



The function block ETH_UDP_INFO reads the status information of the UDP/IP processing.

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

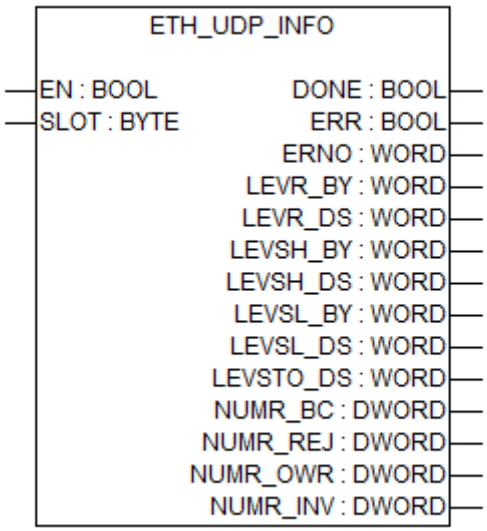
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEVR_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVSH_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_BY displays the filling level of the high priority send buffer in bytes.

LEVSH_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_DS displays the filling level of the high priority send buffer in data sets.

LEVSL_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_BY displays the filling level of the low priority send buffer in bytes.

LEVSL_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_DS displays the filling level of the low priority send buffer in data sets.

LEVSTO_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSTO_DS displays the filling level of the timeout buffer in data sets.

NUMR_BC

Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_BC outputs the number of broadcasts (data packets to all stations) which were received by this station.

NUMR_REJ

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_REJ (number of receipts rejected), the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_OWR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_OWR (number of receipts overwritten), the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_INV

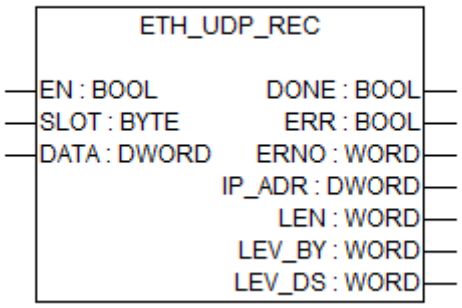
Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in ST

```
Info  (EN := Info_EN,
       SLOT := Info_SLOT)


Info_DONE := Info.DONE;
Info_ERR := Info.ERR;
Info_ERNO := Info.ERNO;
Info_LEVR_BY := Info.LEVR_BY;
Info_LEVR_DS := Info.LEVR_DS;
Info_LEVSH_BY := Info.LEVSH_BY;
Info_LEVSH_DS := Info.LEVSH_DS;
Info_LEVSL_BY := Info.LEVSL_BY;
Info_LEVSL_DS := Info.LEVSL_DS;

Info_LEVSTO_DS := Info.LEVSTO_DS;


Info_NUMR_BC := Info.NUMR_BC;
Info_NUMR_REJ := Info.NUMR_REJ;
Info_NUMR_OWR := Info.NUMR_OWR;
Info_NUMR_INV := Info.NUMR_INV;
```

ETH_UDP_REC

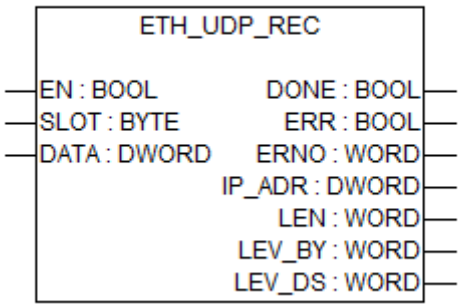


The operating system reads the received UDP/IP data packets from the Ethernet Communication Module and stores them in the receive buffer. The buffer size is determined using the PLC configuration. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ETH_UDP_REC function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The address of the sending device and the data packet length are supplied at the outputs IP_ADR and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

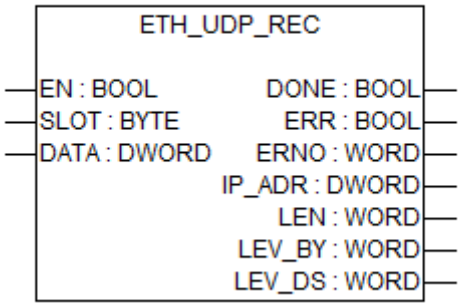
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output IP_ADR displays the IP address of the sending device which transmitted the received data package. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEN (length) displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

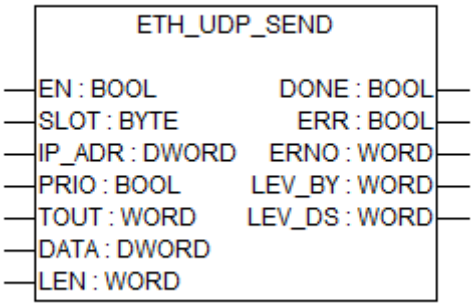
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
REC  (EN := Rec_EN,
     SLOT := Rec_SLOT,
     DATA := ADR(Rec_DATA));


REC_DONE := Rec.DONE;
REC_ERR := Rec.ERR;
REC_ERNO := Rec.ERNO;
REC_IP_ADR := Rec.IP_ADR;
REC_LEN := Rec.LEN;
REC_LEVR_BY := Rec.LEV_BY;
REC_LEVR_DS := Rec.LEV_DS;
```

ETH_UDP_SEND

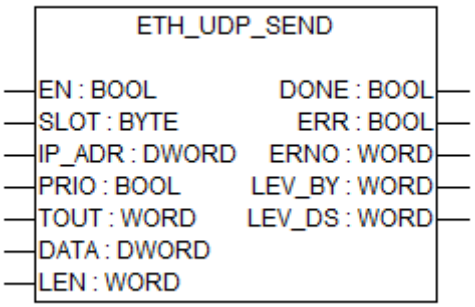


The function block ETH_UDP_SEND is used to transmit data packets via the UDP/IP protocol of the Ethernet Communication Module. The specified packages are stored in the transmit buffer selected by input PRIO. From there, the operating system hands over the data packets to the Ethernet Communication Module in order to transmit them to the target address specified at input IP_ADR. The transmit buffer size is defined by the PLC configuration. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data packet has been stored in the transmit buffer or that an error occurred during Function Block processing. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

The function block ETH_UDP_SEND cannot store data packets to the transmit buffer until the Ethernet UDP/IP processing is set in the controller configuration (link to Controller configuration of UDP/IP processing).

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

At this input, the IP address of the recipient is specified. Each byte in IP_ADR represents one octet of the address.

Example: IP address 192.15.24.2,

IP_ADR (hex) 16#C00F1802,

IP_ADR (dec) 3222214658

PRIO

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Input PRIO (priority) is used to specify the transmit priority of the data packet.

The following applies:

If PRIO = FALSE the specified data packet has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

If PRIO = TRUE the specified data packet has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT

Data type	Default value	Range	Unit
WORD	-	-	-

Using input TOUT (timeout), the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If a data packet cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data packet (see *Chapter 1.6.5.3.7.1 “Contents of the UDP protocol configuration” on page 6185*) are stored to the timeout buffer and can then be read using the function block ETH_UDP_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0:

Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:

Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

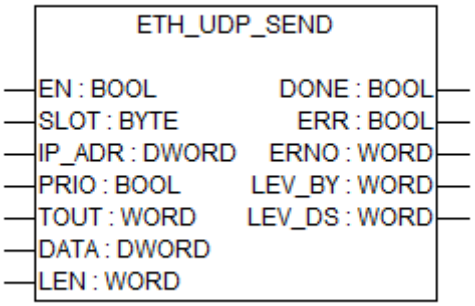
At input DATA, the address of the variable is specified the data of which are transmitted as user data in this package. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

LEN

Data type	Default value	Range	Unit
WORD	-	1...1464	-

At input LEN (length), the number of user data bytes is specified for the specified package.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY (level in bytes) displays the filling level (in bytes) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the IP address of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS

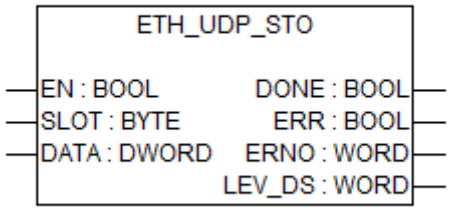
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level (in data records) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
Send (EN := Send_EN,
     SLOT := Send_SLOT,
     IP_ADR := Send_IP_ADR,
     PRIO := Send_PRIO,
     TOUT := Send_TOUT,
     DATA := ADR(Send_DATA),
     LEN := Send_LEN);


Send_DONE := Send.DONE;
Send_ERR := Send.ERR;
Send_ERNO := Send.ERNO;
Send_LEV_BY := Send.LEV_BY;
Send_LEV_DS := Send.LEV_DS;
```

ETH_UDP_STO



The ETH_UDP_STO function block reads lost data packets from the timeout data buffer and stores the user data to the specified memory area.

During the transmission of a data packet, the success of the transmission is monitored by an adjustable timeout period. When this time is exceeded, distinctive information of the data packet are stored in the timeout buffer.

These are:

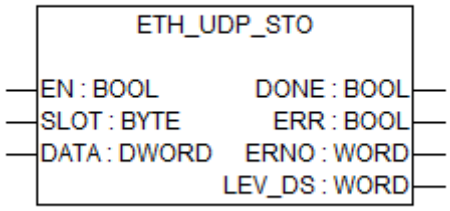
- the IP address of the receiver (4 bytes)
- header data of the data set (the number is specified with the controller configuration of the UDP/IP processing).

The buffer length can as well be set using the controller configuration of the UDP/IP processing. The buffer is constructed as a circular buffer (FIFO). If the buffer is full, the oldest entry in the buffer is overwritten. When a rising edge occurs at input EN, the ETH_UDP_STO function block verifies whether a data packet is stored in the buffer and makes the information mentioned above available for the user (starting at the variable specified at input DATA).

The function block ETH_UDP_STO cannot be used until the Ethernet UDP/IP processing is set in the controller configuration (link to Controller configuration of UDP/IP processing) Additionally, input TOUT of the transmit function block ETH_UDP_SEND must be <> 0.

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

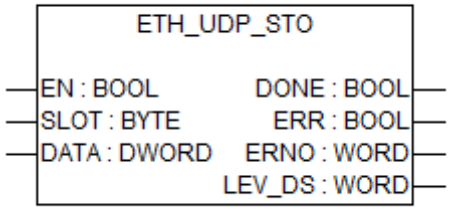
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_DS

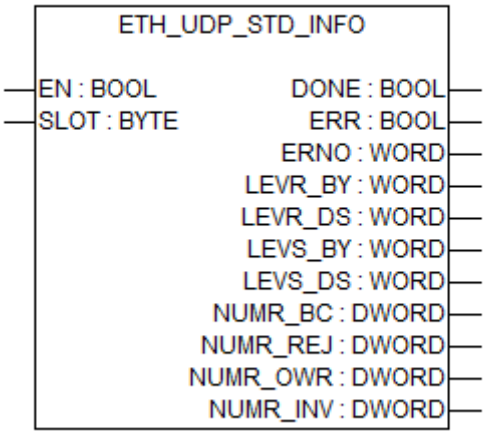
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
Sto  (EN := Sto_EN,
     SLOT := Sto_SLOT,
     DATA := ADR(Sto_DATA));


Sto_DONE := Sto.DONE;
Sto_ERR := Sto.ERR;
Sto_ERNO := Sto.ERNO;
Sto_LEV_DS := Sto.LEV_DS;
```

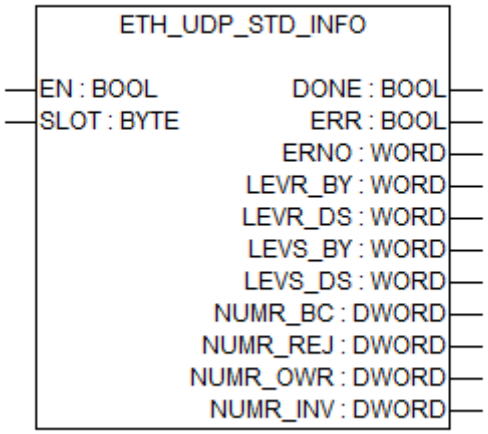
ETH_UDP_STD_INFO



The function block ETH_UDP_STD_INFO reads the status information of the UDP/IP processing.

Available as of runtime system:	V1.3
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

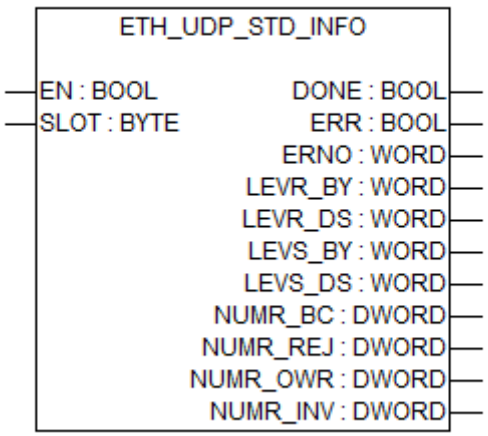
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEVR_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVS_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVS_BY (level of all send buffers in bytes) displays the filling level of all send buffers in bytes.

LEVS_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVS_DS (level of all send buffers in data sets) displays the filling level of all send buffers in data sets.

NUMR_BC

Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_BC outputs the number of broadcasts (data packets to all stations) which were received by this station.

NUMR_REJ

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_REJ (number of receipts rejected), the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_OWR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_OWR (number of receipts overwritten), the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_INV

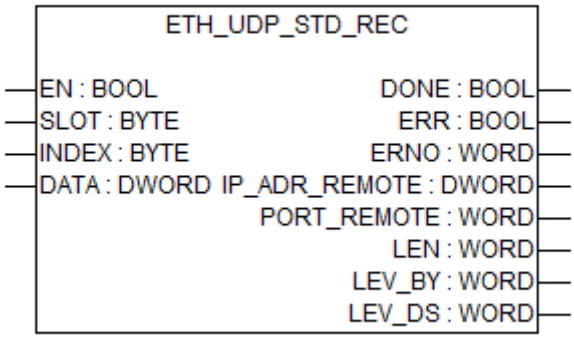
Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in ST

```
Info  (EN := Info_EN,
       SLOT := Info_SLOT)


Info_DONE := Info.DONE;
Info_ERR := Info.ERR;
Info_ERNO := Info.ERNO;
Info_LEVR_BY := Info.LEVR_BY;
Info_LEVR_DS := Info.LEVR_DS;
Info_LEVSH_BY := Info.LEVSH_BY;
Info_LEVSH_DS := Info.LEVSH_DS;
Info_LEVSL_BY := Info.LEVSL_BY;
Info_LEVSL_DS := Info.LEVSL_DS;

Info_LEVSTO_DS := Info.LEVSTO_DS;


Info_NUMR_BC := Info.NUMR_BC;
Info_NUMR_REJ := Info.NUMR_REJ;
Info_NUMR_OWR := Info.NUMR_OWR;
Info_NUMR_INV := Info.NUMR_INV;
```

ETH_UDP_STD_REC



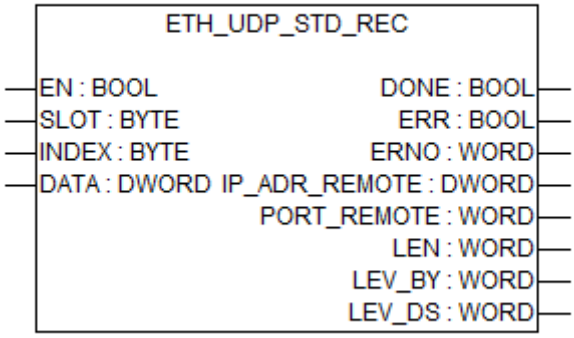
The ETH_UDP_STD_REC function block reads the next data record from the UDP/IP receive buffer and stores the user data to the configured memory area.

The operating system reads the received UDP/IP data packets from the Ethernet Communication Module and stores them in the receive buffer. The buffer size is determined using the PLC configuration. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ETH_UDP_STD_REC function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The address of the sending device and the data packet length are supplied at the outputs IP_ADR_REMOTE, PORT_REMOTE and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during function block processing, the error is indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEV_BY) and data records (LEV_DS).

Available as of runtime system:	V1.3
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

INDEX

Data type	Default value	Range	Unit
BYTE	-	-	-

Index of the standard UDP/IP connection corresponding to the PLC configuration, starting with 0. The function block only works on the connection that was selected by this input, and with the parameters set in the PLC configuration, e.g. port, buffer size, behavior in case of buffer overflow etc.

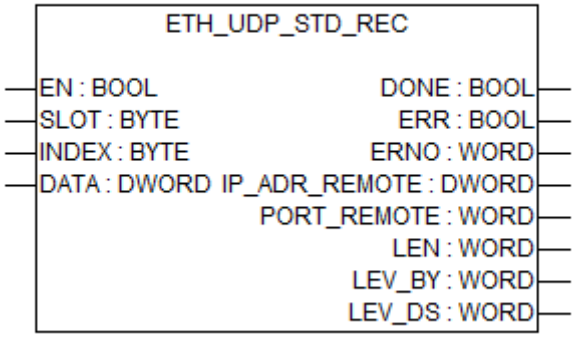
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

Output IP_ADR_REM (IP address remote) displays the IP address of the sending device which transmitted the received data package. Each byte in IP_ADR_REM represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

PORT_ ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PORT_ADR_REM (remote port address) displays the UDP remote port address of the sending device which transmitted the received data package.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEN (length) displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

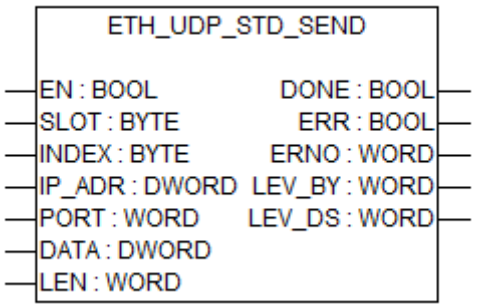
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
REC  (EN := Rec_EN,
     SLOT := Rec_SLOT,
     INDEX := Rec_INDEX,
     DATA := ADR(Rec_DATA));


REC_DONE := Rec.DONE;
REC_ERR := Rec.ERR;
REC_ERNO := Rec.ERNO;
REC_IP_ADR_REMOTE := Rec.IP_ADR_REMOTE;
REC_PORT_REMOTE := Rec.PORT_REMOTE;
REC_LEN := Rec.LEN;
REC_LEVR_BY := Rec.LEV_BY;
REC_LEVR_DS := Rec.LEV_DS;
```

ETH_UDP_STD_SEND

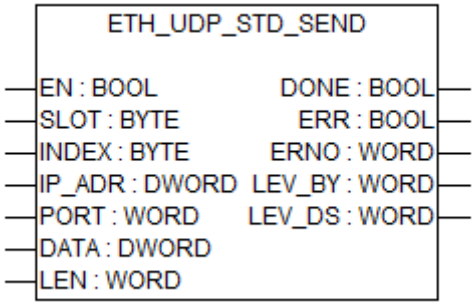


The function block ETH_UDP_STD_SEND is used to transmit data packets via the standard UDP/IP protocol of the Ethernet Communication Module. The specified packages are stored in the transmit buffer. From there, the operating system hands over the data packets to the Ethernet Communication Module in order to transmit them to the target address specified at input IP_ADR. The transmit buffer size is defined by the PLC configuration. Output DONE indicates that the specified data packet has been stored in the transmit buffer or that an error occurred during Function Block processing. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

The function block ETH_UDP_STD_SEND cannot store data packets to the transmit buffer until the Ethernet UDP/IP processing is set in the controller configuration (see also the PLC configuration of the standard UDP/IP processing).

Available as of runtime system:	V1.3
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

INDEX

Data type	Default value	Range	Unit
BYTE	-	-	-

Index of the standard UDP/IP connection corresponding to the PLC configuration, starting with 0. The function block only works on the connection that was selected by this input, and with the parameters set in the PLC configuration, e.g. port, buffer size, behavior in case of buffer overflow etc.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

At this input, the IP address of the recipient is specified. Each byte in IP_ADR represents one octet of the address.

Example: IP address 192.15.24.2,

IP_ADR (hex) 16#C00F1802,

IP_ADR (dec) 3222214658

PORT

Data type	Default value	Range	Unit
WORD	-	-	-

The target port of the receiver is specified here (port to send to).

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

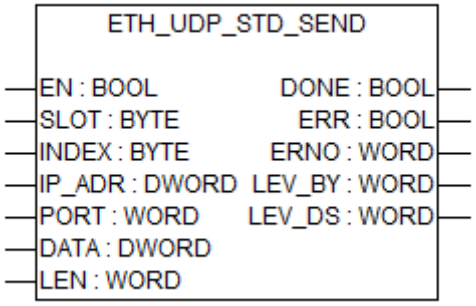
At input DATA, the address of the variable is specified the data of which are transmitted as user data in this package. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

LEN

Data type	Default value	Range	Unit
WORD	-	1...1464	-

At input LEN (length), the number of user data bytes is specified for the specified package.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY (level in bytes) displays the filling level (in bytes) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the IP address of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS

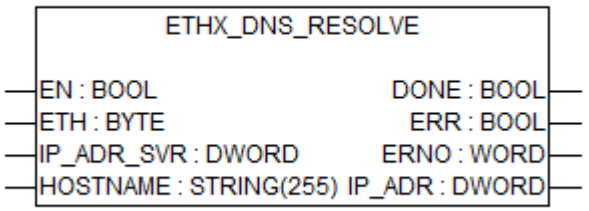
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level (in data records) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
Send (EN := Send_EN,
     SLOT := Send_SLOT,
    INDEX := Send_INDEX,
     IP_ADR := Send_IP_ADR,
     PORT := Send_PORT,
     DATA := ADR(Send_DATA),
     LEN := Send_LEN);


Send_DONE := Send.DONE;
Send_ERR := Send.ERR;
Send_ERNO := Send.ERNO;
Send_LEV_BY := Send.LEV_BY;
Send_LEV_DS := Send.LEV_DS;
```

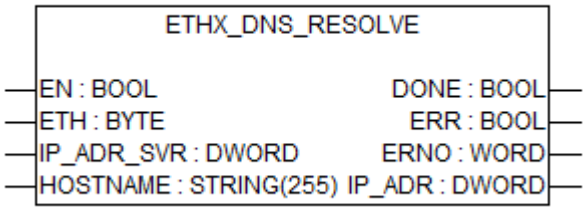
ETHx_DNS_RESOLVE



The function block ETHx_DNS_RESOLVE returns an IP address of a user-specified hostname via a given DNS server. If no Ethernet device is installed at ETH, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

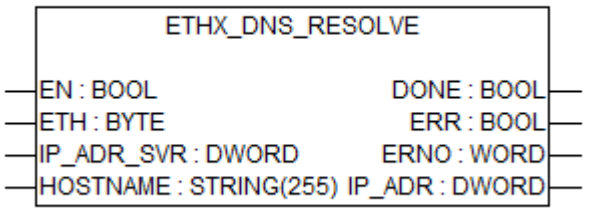
*) The Communication Modules are serially numbered from right to left, starting with module number 1.

HOSTNAME

Data type	Default value	Range	Unit
STRING	Empty string	-	-

At input HOSTNAME, the domain of the host to be resolved to an IP address is specified. This name is sent ot the DNS server, which answers with an IP address.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output IP_ADR displays the resolved IP address of the hostname provided by the input HOST-NAME. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

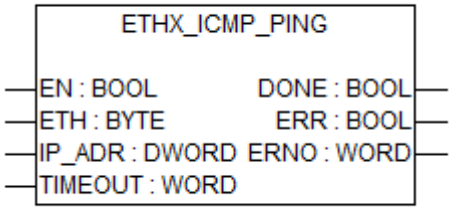
IP_ADR (dec) 3222214658

Function call in ST

```
DNSResolve    (EN          := DNSResolve_EN,
               ETH         := DNSResolve_ETH,
               IP_ADR_SVR  := DNSResolve_IP_ADR_SVR,
               HOSTNAME    := DNSResolve_HOSTNAME);
```

```
DNSResolve_DONE   := DnsResolve.DONE;
DNSResolve_ERR    := DnsResolve.ERR;
DNSResolve_ERNO   := DnsResolve.ERNO;
DNSResolve_IP_ADR := DnsResolve.IP_ADR;
```

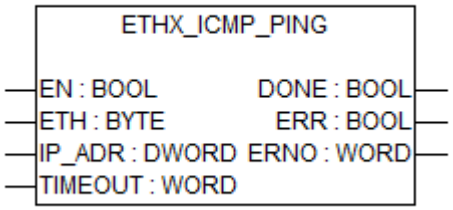
ETHx_ICMP_PING



The function block ETHx_ICMP_PING returns whether a given host in the network answers to a ping. If no Ethernet device is installed at ETH, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

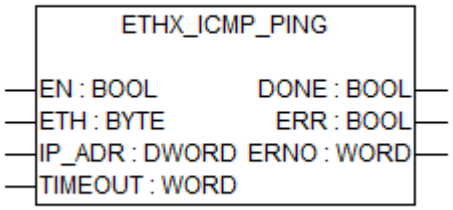
At input IP_ADR, the IP address of the host to be pinged is specified.

TIMEOUT

Data type	Default value	Range	Unit
WORD	5.000	-	ms

At input TIMEOUT, the time to wait for an ICMP echo reply from the host is specified. After this time has run out, an error will be set as output.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

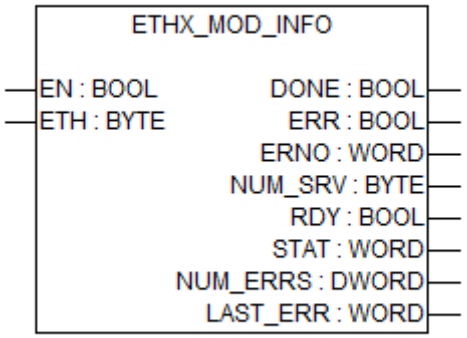
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ICMPPing (EN         := ICMPPing_EN,
          ETH        := ICMPPing_ETH,
          IP_ADR     := ICMPPing_IP_ADR
          TIMEOUT    := ICMPPing_TIMEOUT)


ICMPPing_DONE  := ICMPPing.DONE;
ICMPPing_ERR   := ICMPPing.ERR;
ICMPPing_ERNO  := ICMPPing.ERNO;
```

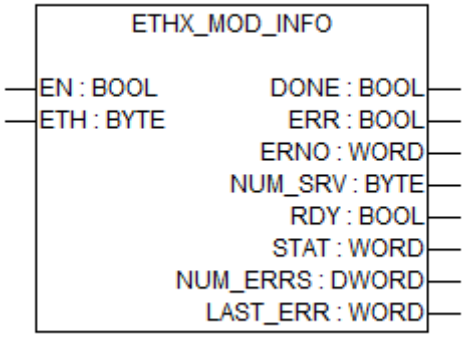
ETHx_MOD_INFO



The function block ETHx_MOD_INFO reads the status information of the open Modbus on TCP/IP processing. It can be used for pure server (slave) or client (master) operation of the controller as well as for mixed operation.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

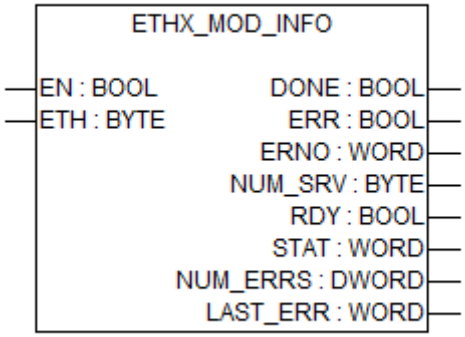
ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SRV

Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_SRV (number of servers) output indicates the number of parallel server channels configured with Automation Builder software. NUM_SRV is only valid if DONE = TRUE and ERR = FALSE.

RDY

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

RDY (ready) output indicates the readiness for operation of the open Modbus on TCP/IP processing. If RDY = TRUE, the server processing as well as the client processing are ready for operation. RDY is only valid if DONE = TRUE and ERR = FALSE.

STAT

Output STAT displays the current operating state of the open Modbus on TCP/IP processing. STAT is only valid if DONE = TRUE and ERR = FALSE.

STAT		Description
Decimal	**Hexadecimal**	
0	00	Processing not initialized
1	01	Processing initialized and running
2	02	Processing initialization in progress
3	03	Initialization error
4	04	Processing initialized and waiting for TCP task

NUM_ERRS

Data type	Default value	Range	Unit
DWORD	-	-	-

The total number of errors detected by the Communication Module since last power up or reset.

LAST_ERR

Data type	Default value	Range	Unit
WORD	-	-	-

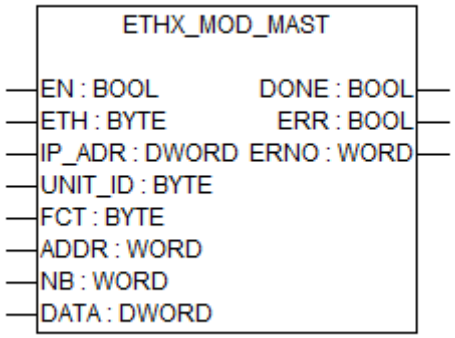
LAST_ERR outputs the last error occurred on the Communication Module. The error message encoding at output LAST_ERR applies to all Ethernet function blocks and is explained at the beginning of the library descriptions.

The outputs gives the value which has been received directly from the Communication Module. To find this error code inside the list of error messages, the value 0x6000 has to be added to the error value.

Function call in ST

```
MODInfo (EN  := MODInfo_EN,
         ETH := MODInfo_ETH);


MODInfo_DONE     := MODInfo.DONE;
MODInfo_ERR      := MODInfo.ERR;
MODInfo_ERNO     := MODInfo.ERNO;
MODInfo_NUM_SRV  := MODInfo.NUM_SRV;
MODInfo_RDY      := MODInfo.RDY;
MODInfo_STAT     := MODInfo.STAT;
MODInfo_NUM_ERRS := MODInfo.NUM_ERRS;
MODInfo_LAST_ERR := MODInfo.LAST_ERR;
```

ETHx_MOD_MAST



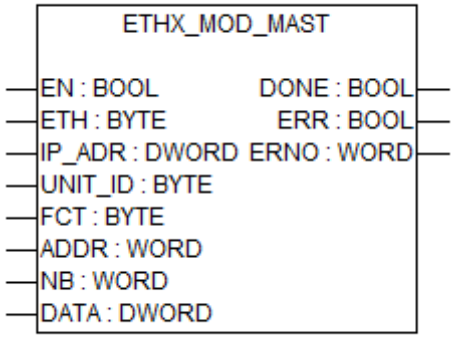
The ETHx_MOD_MAST function block implements the open Modbus on TCP/IP client functionality for the Ethernet interface specified at input ETH. Depending on the configuration of the Communication Module, several ETHx_MOD_MAST function blocks can be used in parallel. Prior to the use of ETHx_MOD_MAST for an Ethernet interface it has to be configured accordingly using Automation Builder.

With each FALSE > TRUE edge at input EN, the function block ETHx_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and then sends this telegram to the slave.

Only a single instance per slave can be used with this function block!

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

At IP_ADR, the IP address of the server has to be specified to which the telegram should be sent. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

UNIT_ID

Data type	Default value	Range	Unit
BYTE	-	-	-

At input UNIT_ID, the address of the Modbus slave must be specified which is connected serially to the Modbus server defined by IP_ADR. If no further slaves are connected, this input is not used.

FCT

Data type	Default value	Range	Unit
BYTE	-	-	-

The function code of the request telegram is specified at input FCT.

01 or 02	Read n bits
03 or 04	Read n words
05	Write one bit (encoded in one word)
06	Write one word

07	Read 8 bit
15	Write n bits (encoded in one byte)
16	Write n words
22	Mask write
23	Read/write multiple words in one telegram

The max. telegram length for onboard Ethernet CPUs with function code 3/4 is 96 words.

The max. telegram length for onboard Ethernet CPUs with function code 15 is 1536 bits.

ADDR

Data type	Default value	Range	Unit
WORD	-	-	-

The operand/register address in the slave from which data should be read or written is specified at input ADDR.

The access to operands of AC500 devices in Modbus slave mode is defined via the Modbus cross-reference list. Only operands that are listed in the cross-reference list may be used (↳ *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*).

Only operands that are listed in the Modbus address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

NB

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input NB (number), the number of data to be written or read is specified. The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB. The following restrictions apply to the length:

FCT		NBmax	
Dec	**Hex**	**Serial**	**Modbus on TCP/IP**
01 or 02	01 or 02	2000 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
03 or 04	03 or 04	125 words / 62 double words	125 words / 62 double words
05	05	1 bit	1 bit
06	06	1 word	1 word
07	07	8 bits	8 bits

FCT		NBmax	
Dec	**Hex**	**Serial**	**Modbus on TCP/IP**
15	0F	1968 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
16	10	123 words / 61 double words	123 words / 61 double words
22	16	Write: 1 word	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 123 words / 61 double words	Read: 125 words / 62 double words Write: 123 words / 61 double words

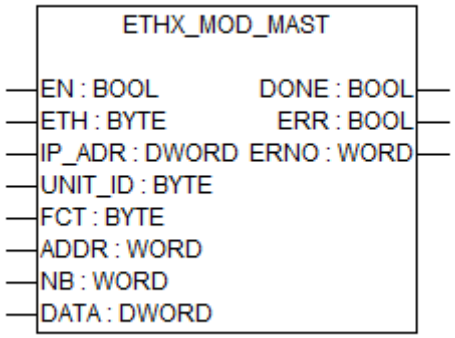
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

At input DATA, the address of the first operand in the master is specified, from which data are copied/written to the slave or to which the data read by the slave should be stored. For this purpose it is necessary that the operand type (e.g. bit) matches the selected function (e.g. FCT 1, read n bits).

If using Modbus function codes 22 or 23, the according data structures COM_MOD_FCT22_TYPE *Chapter 1.5.4.22.2.1 “COM_MOD_FCT22_TYPE” on page 1702* or COM_MOD_FCT23_TYPE *Chapter 1.5.4.22.2.2 “COM_MOD_FCT23_TYPE” on page 1703* must be defined and applied to DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	-	-

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error occurred during Function Block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO

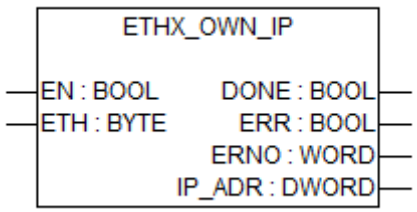
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
MODMast (EN       := MODMast_EN,
         ETH      := MODMast_ETH,
         IP_ADR   := MODMast_IP_ADR,
         UNIT_ID  := MODMast_UNIT_ID,
         FCT      := MODMast_FCT,
         ADDR     := MODMast_ADDR,
         NB       := MODMast_NB,
         DATA     := ADR(ModMast_DATA));


MODMast_DONE := MODMast.DONE;
MODMast_ERR  := MODMast.ERR;


MODMast_ERNO := MODMast.ERNO;
```

ETHx_OWN_IP

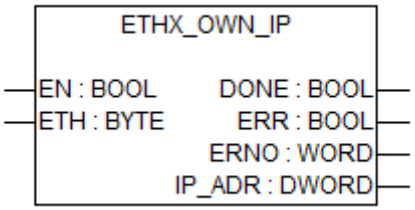


The function block ETHx_OWN_IP outputs the IP address of the Communication Module installed at input ETH.

Prior to commissioning an Ethernet interface, it has to be configured using Automation Builder software. The IP address is one of the parameters of an Ethernet interface. Using the function block ETHx_OWN_IP, the configured IP address of the device at input ETH can be read. If no Ethernet device is installed at ETH, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

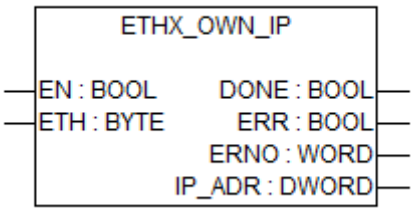
ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

Output IP_ADR displays the own IP address of the Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

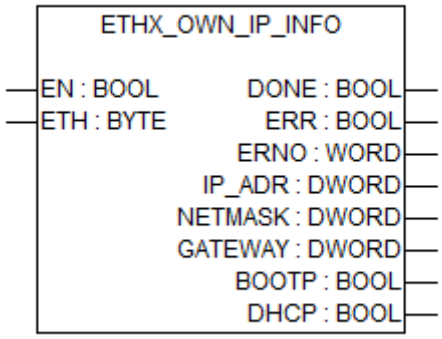
IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

Function call in ST

```
OwnIp(EN  := OwnIp_EN,
      ETH := OwnIp_ETH);


OwnIp_DONE   := OwnIp.DONE;
OwnIp_ERR    := OwnIp.ERR;
OwnIp_ERNO   := OwnIp.ERNO;
OwnIp_IP_ADR := OwnIp.IP_ADR;
```

ETHx_OWN_IP_INFO

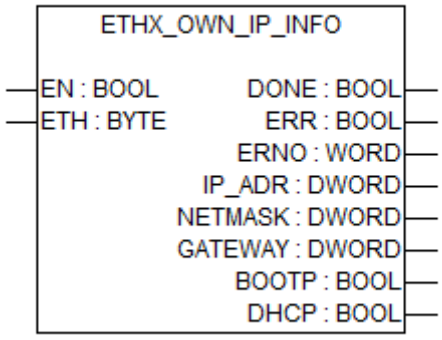


The function block ETHx_OWN_IP_INFO outputs the IP setup of the Communication Module installed at interface specified by input ETH.

Prior to commissioning an Ethernet interface, it has to be configured using Automation Builder software. Using the function block ETHx_OWN_IP_INFO, information about the IP setup of the device at ETH can be read. If no Ethernet interface is installed at input ETH, the corresponding error is generated and output at ERR and ERNO.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

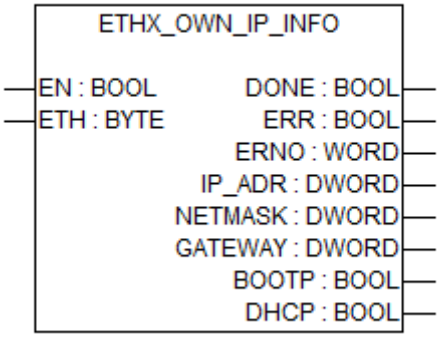
ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

IP_ADR displays the IP address which should be set at the CPU / Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

NETMASK

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output NETMASK displays the own netmask. Each byte in NETMASK represents one octet of the address.

Example

Netmask 255.255.255.0

NETMASK (hex) 16#FFFFFF00

NETMASK (dec) 4294967040

GATEWAY

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output GATEWAY displays the IP address of the gateway. Each byte in GATEWAY represents one octet of the address.

Example

IP address 192.168.0.1

IP_ADR (hex) 16#C0A80001

IP_ADR (dec) 3232235521

BOOTP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output BOOTP indicates whether the BOOTP service is activated for the Communication Module.

DHCP

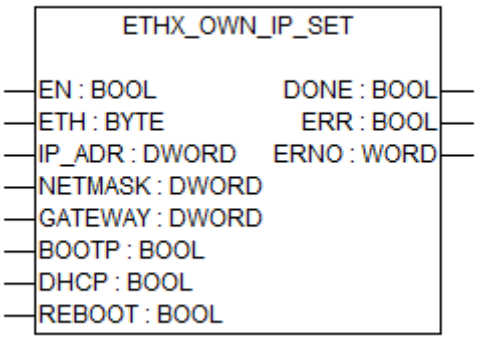
Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DHCP indicates whether the DHCP service is activated for the Communication Module.

Function call in ST

```
OwnIPInfo(EN  := OwnIPInfo_EN,
          ETH := OwnIPInfo_ETH);


OwnIPInfo_DONE    := OwnIPInfo.DONE;
OwnIPInfo_ERR     := OwnIPInfo.ERR;
OwnIPInfo_ERNO    := OwnIPInfo.ERNO;
OwnIPInfo_IP_ADR  := OwnIPInfo.IP_ADR;
OwnIPInfo_NETMASK := OwnIPInfo.NETMASK;
OwnIPInfo_GATEWAY := OwnIPInfo.GATEWAY;
OwnIPInfo_BOOTP   := OwnIPInfo.BOOTP;
OwnIPInfo_DHCP    := OwnIPInfo.DHCP;
```

ETHx_OWN_IP_SET



Function block ETH_OWN_IP_SET is used to set

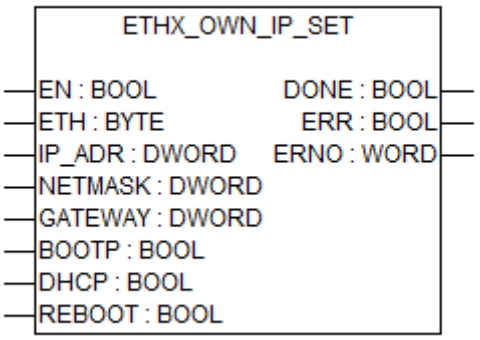
- IP address
- Subnet mask
- Gateway and
- some other parameters

at the Ethernet interface installed at input ETH.

The new IP configuration data will be available after the next reboot of the CPU / Communication module. Use input REBOOT=TRUE to restart the CPU / Communication Module immediately after changing the IP configuration data.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

IP_ADR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

IP_ADR displays the IP address which should be set at the CPU / Communication Module. Each byte in IP_ADR represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

NETMASK

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input NETMASK, the subnet mask which should be set at the CPU / Communication Module is specified. Each byte in NETMASK represents one octet of the address.

Example

Netmask 255.255.255.0

NETMASK (hex) 16#FFFFFF00

NETMASK (dec) 4294967040

GATEWAY

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input GATEWAY, the default gateway which should be set at the CPU / Communication Module is specified. Each byte in GATEWAY represents one octet of the address.

Example

IP address 192.168.0.1

IP_ADR (hex) 16#C0A80001

IP_ADR (dec) 3232235521

GATEWAY

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

Output GATEWAY displays the IP address of the gateway. Each byte in GATEWAY represents one octet of the address.

Example

IP address 192.168.0.1

IP_ADR (hex) 16#C0A80001

IP_ADR (dec) 3232235521

BOOTP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output BOOTP indicates whether the BOOTP service is activated for the Communication Module.

DHCP

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

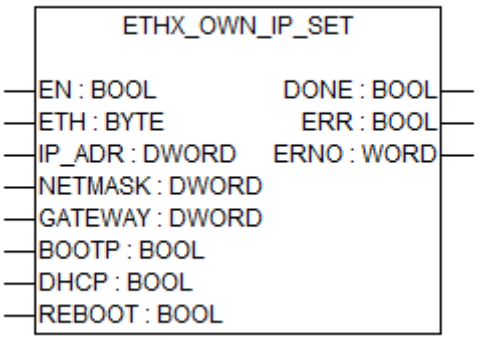
Output DHCP indicates whether the DHCP service is activated for the Communication Module.

REBOOT

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

A TRUE signal at input REBOOT restarts the CPU / Communication Module after changing the IP settings.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

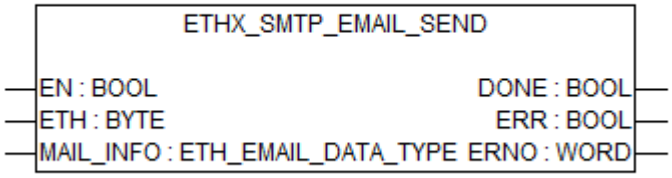
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
OwnIPSet (EN      := OwnIPSet_EN,
          ETH     := OwnIPSet_ETH,
          IP_ADR  := OwnIPSet_IP_ADR,
          NETMASK := OwnIPSet_NETMASK,
          GATEWAY := OwnIPSet_GATEWAY,
          BOOTP   := OwnIPSet_BOOTP,
          DHCP    := OwnIPSet_DHCP,
          REBOOT  := OwnIPSet_REBOOT);


OwnIPSet_DONE := OwnIPSet.DONE;
OwnIPSet_ERR  := OwnIPSet.ERR;
OwnIPSet_ERNO := OwnIPSet.ERNO;
```

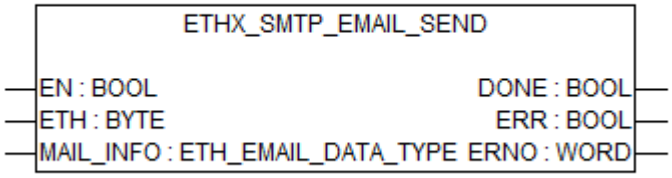
ETHx_SMTP_EMAIL_SEND



The function block ETHx_SMTP_EMAIL_SEND uses a user-specified SMTP server to deliver an email to given mailboxes.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

MAIL_INFO

Data type: ETH_EMAIL_DATA_TYPE (slot)

Input MAIL_INFO provides all necessary data to send an email via the data structure ETH_EMAIL_DATA_TYPE.

ETH BYTE (Ethernet index number)

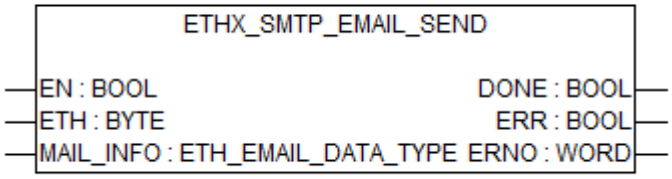
At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)

21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

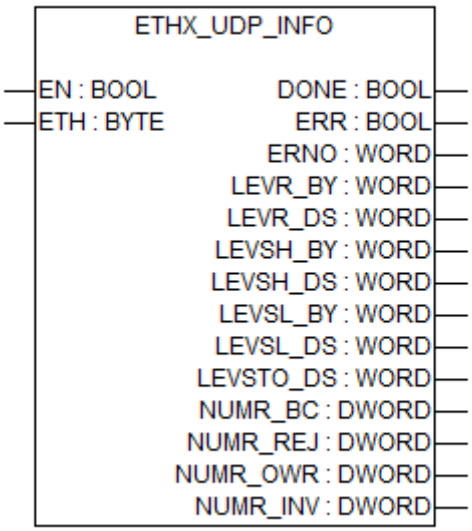
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
SMTPEmailSend (EN        := SMTPEmailSend_EN,
               ETH       := SMTPEmailSend_ETH,
               MAIL_INFO := SMTPEmailSend_MAIL_INFO)


SMTPEmailSend_DONE  := SMTPEmailSend.DONE;
SMTPEmailSend_ERR   := SMTPEmailSend.ERR;
SMTPEmailSend_ERNO  := SMTPEmailSend.ERNO;
```

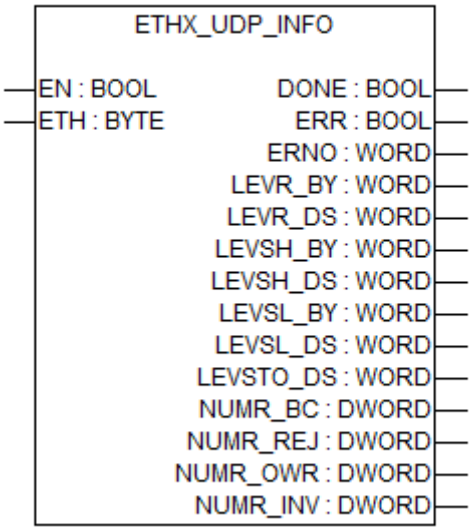
ETHx_UDP_INFO



The function block ETHx_UDP_INFO reads the status information of the UDP/IP processing.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

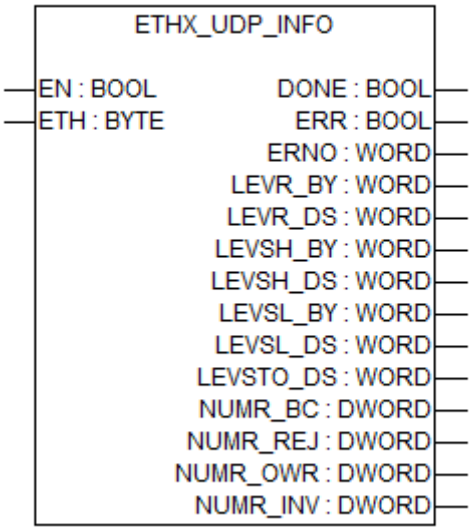
ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEVR_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVSH_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_BY displays the filling level of the high priority send buffer in bytes.

LEVSH_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSH_DS displays the filling level of the high priority send buffer in data sets.

LEVSL_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_BY displays the filling level of the low priority send buffer in bytes.

LEVSL_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSL_DS displays the filling level of the low priority send buffer in data sets.

LEVSTO_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVSTO_DS displays the filling level of the timeout buffer in data sets.

NUMR_BC

Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_BC outputs the number of broadcasts (data packets to all stations) which were received by this station.

NUMR_REJ

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_REJ (number of receipts rejected), the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_INV

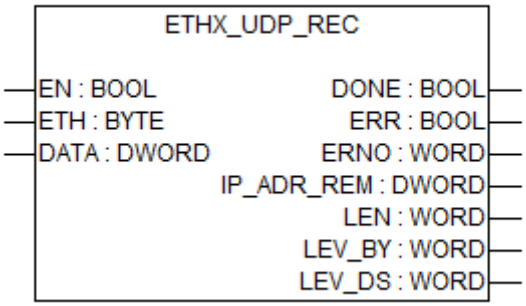
Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in ST

```
UDPInfo (EN  := UDPInfo_EN,
         ETH := UDPInfo_ETH)


UDPInfo_DONE      := UDPInfo.DONE;
UDPInfo_ERR       := UDPInfo.ERR;
UDPInfo_ERNO      := UDPInfo.ERNO;
UDPInfo_LEVR_BY   := UDPInfo.LEVR_BY;
UDPInfo_LEVR_DS   := UDPInfo.LEVR_DS;
UDPInfo_LEVSH_BY  := UDPInfo.LEVSH_BY;
UDPInfo_LEVSH_DS  := UDPInfo.LEVSH_DS;
UDPInfo_LEVSL_BY  := UDPInfo.LEVSL_BY;
UDPInfo_LEVSL_DS  := UDPInfo.LEVSL_DS;
UDPInfo_LEVSTO_DS := UDPInfo.LEVSTO_DS;
UDPInfo_NUMR_BC   := UDPInfo.NUMR_BC;
UDPInfo_NUMR_REJ  := UDPInfo.NUMR_REJ;
UDPInfo_NUMR_OWR  := UDPInfo.NUMR_OWR;
UDPInfo_NUMR_INV  := UDPInfo.NUMR_INV;
```

ETHx_UDP_REC



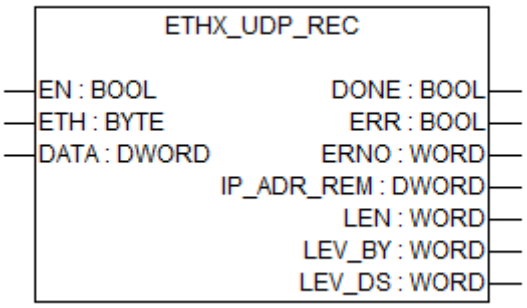
The ETHx_UDP_REC function block reads the next data record from the UDP/IP receive buffer and stores the user data to the configured memory area.

The operating system reads the received UDP/IP data packets from the Ethernet interface and stores them in the receive buffer. The buffer size is determined using the PLC configuration. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ETHx_UDP_REC function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The address of the sending device and the data packet length are supplied at the outputs IP_ADR_REM and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEVR_BY) and data records (LEVR_DS).

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

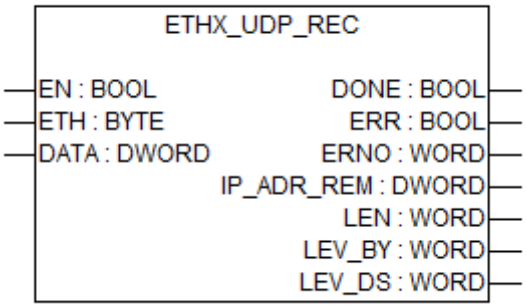
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

Output IP_ADR_REM (IP address remote) displays the IP address of the sending device which transmitted the received data package. Each byte in IP_ADR_REM represents one octet of the address.

Example	IP address 192.15.24.2 IP_ADR (hex) 16#C00F1802 IP_ADR (dec) 3222214658

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEN (length) displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

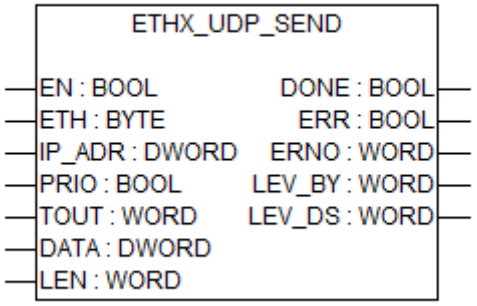
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
UDPRec (EN   := UDPRec_EN,
        ETH  := UDPRec_ETH,
        DATA := ADR(UDPRec_DATA));


UDPRec_DONE       := UDPRec.DONE;
UDPRec_ERR        := UDPRec.ERR;
UDPRec_ERNO       := UDPRec.ERNO;
UDPRec_IP_ADR_REM := UDPRec.IP_ADR_REM;
UDPRec_LEN        := UDPRec.LEN;
UDPRec_LEVR_BY    := UDPRec.LEV_BY;
UDPRec_LEVR_DS    := UDPRec.LEV_DS;
```

ETHx_UDP_SEND

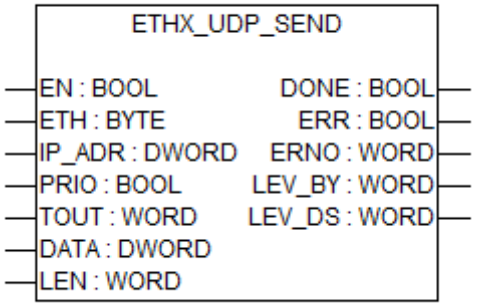


The function block ETHx_UDP_SEND is used to transmit data packets via the UDP/IP protocol of the Ethernet interface. The specified packages are stored in the transmit buffer selected by input PRIO. From there, the operating system hands over the data packets to the Ethernet interface in order to transmit them to the target address specified at input IP_ADR. The transmit buffer size is defined by the PLC configuration. Using input TOUT, the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If TOUT = 0, no acknowledgement is expected. Output DONE indicates that the specified data packet has been stored in the transmit buffer or that an error occurred during Function Block processing. If an error was detected during Function Block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

The function block ETHx_UDP_SEND cannot store data packets to the transmit buffer until the Ethernet UDP/IP processing is set in the controller configuration.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

PRIO

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Input PRIO (priority) is used to specify the transmit priority of the data packet.

The following applies:

If PRIO = FALSE the specified data packet has low priority. Thus, it is stored in the low priority transmit buffer. All outputs refer to this buffer.

If PRIO = TRUE the specified data packet has high priority. Thus, it is stored in the high priority transmit buffer. All outputs refer to this buffer.

TOUT

Data type	Default value	Range	Unit
WORD	-	-	-

Using input TOUT (timeout), the timeout period can be specified. If TOUT <> 0, the UDP/IP data exchange is automatically performed with receive acknowledgement. If a data packet cannot be transmitted within this period (no acknowledge telegram is received), transmission is aborted and the package is lost.

In this case, some distinctive bytes of the data packet (see *Chapter 1.6.5.3.7.1 “Contents of the UDP protocol configuration” on page 6185*) are stored to the timeout buffer and can then be read using the function block ETH_UDP_STO.

If TOUT = 0, no acknowledgement is expected.

The following applies:

TOUT = 0:

Data exchange without receive acknowledgement. No data are written to the timeout buffer.

TOUT <> 0:

Data exchange with receive acknowledgement. Each transmitted data record is acknowledged by the recipient. If no acknowledge telegram is received within the set timeout period (in ms), the data are written to the timeout buffer.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

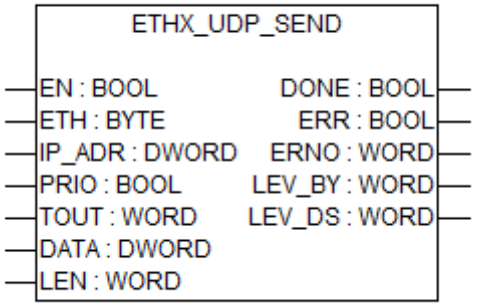
Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

LEN

Data type	Default value	Range	Unit
WORD	-	1...1464	-

At input LEN (length), the number of user data bytes is specified for the specified package.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY (level in bytes) displays the filling level (in bytes) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the IP address of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS

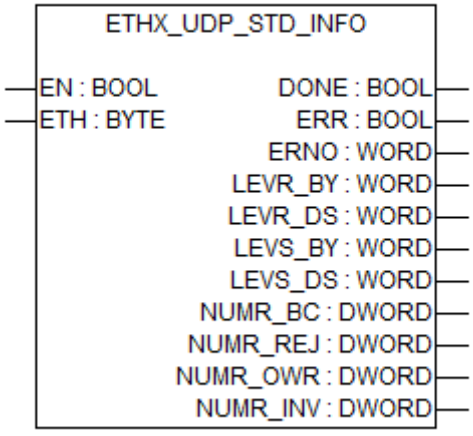
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level (in data records) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
UDPSend (EN     := UDPSend_EN,
         ETH    := UDPSend_ETH,
         IP_ADR := UDPSend_IP_ADR,
         PRIO   := UDPSend_PRIO,
         TOUT   := UDPSend_TOUT,
         DATA   := ADR(UDPSend_DATA),
         LEN    := UDPSend_LEN);


UDPSend_DONE   := UDPSend.DONE;
UDPSend_ERR    := UDPSend.ERR;
UDPSend_ERNO   := UDPSend.ERNO;
UDPSend_LEV_BY := UDPSend.LEV_BY;
UDPSend_LEV_DS := UDPSend.LEV_DS;
```

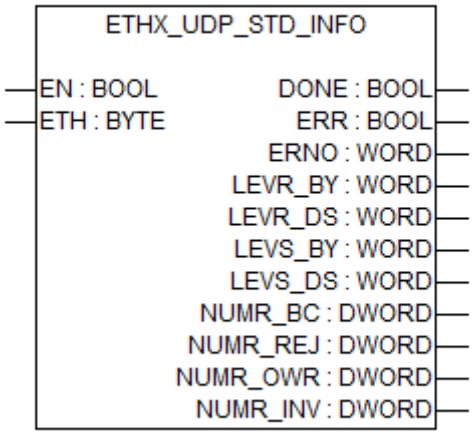
ETHx_UDP_STD_INFO



The function block ETHx_UDP_STD_INFO reads the status information of the UDP/IP processing.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

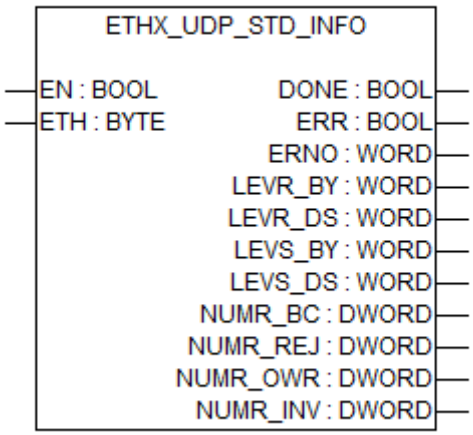
ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEVR_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_BY displays the filling level of the receive buffer in bytes.

LEVR_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVR_DS displays the filling level of the receive buffer in data sets.

LEVS_BY

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVS_BY (level of all send buffers in bytes) displays the filling level of all send buffers in bytes.

LEVS_DS

Data type	Default value	Range	Unit
WORD	-	-	-

As long as EN = TRUE, output LEVS_DS (level of all send buffers in data sets) displays the filling level of all send buffers in data sets.

NUMR_BC

Data type	Default value	Range	Unit
DWORD	-	-	-

NUMR_BC outputs the number of broadcasts (data packets to all stations) which were received by this station.

NUMR_REJ

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_REJ (number of receipts rejected), the number of data sets is displayed which were discarded during reception due to a full receive buffer. Data sets are only discarded if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_OWR

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At output NUMR_OWR (number of receipts overwritten), the number of data sets is displayed which were overwritten during reception due to a full receive buffer. Data sets are only overwritten in the receive buffer, if this is set accordingly within the configuration of the UDP/IP processing.

NUMR_INV

Data type	Default value	Range	Unit
DWORD	-	-	-

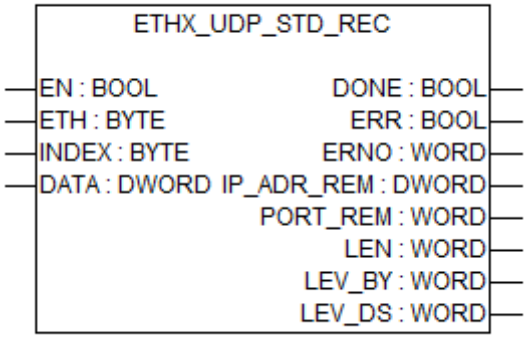
NUMR_INV outputs the number of telegrams which were received faulty by this station.

Function call in ST

```
UDPStd Info (EN  := UDPStdInfo_EN,
             ETH := UDPStdInfo_ETH);


UDPStdInfo_DONE     := UDPStdInfo.DONE;
UDPStdInfo_ERR      := UDPStdInfo.ERR;
UDPStdInfo_ERNO     := UDPStdInfo.ERNO;
UDPStdInfo_LEVR_BY  := UDPStdInfo.LEVR_BY;
UDPStdInfo_LEVR_DS  := UDPStdInfo.LEVR_DS;
UDPStdInfo_LEVS_BY  := UDPStdInfo.LEVS_BY;
```

```
UDPStdInfo_LEVS_DS  := UDPStdInfo.LEVS_DS;
UDPStdInfo_NUMR_BC  := UDPStdInfo.NUMR_BC;
UDPStdInfo_NUMR_REJ := UDPStdInfo.NUMR_REJ;
UDPStdInfo_NUMR_OWR := UDPStdInfo.NUMR_OWR;
UDPStdInfo_NUMR_INV := UDPStdInfo.NUMR_INV;
```

ETHx_UDP_STD_REC



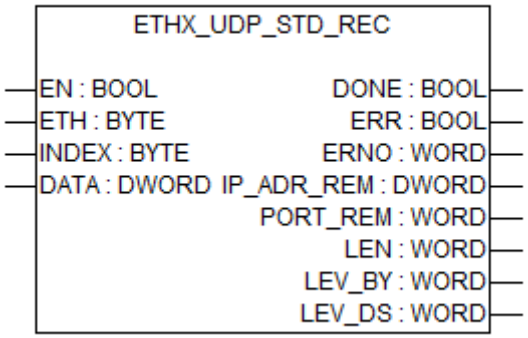
The ETHx_UDP_STD_REC function block reads the next data record from the UDP/IP receive buffer and stores the user data to the configured memory area.

The operating system reads the received UDP/IP data packets from the Ethernet Communication Module and stores them in the receive buffer. The buffer size is determined using the PLC configuration. The data packets are stored with variable lengths. For example, a data packet consisting of 16 bytes of user data occupies exactly 22 bytes in the receive buffer (4 bytes for the IP address of the sending device, 2 bytes for the packet length and 16 bytes of user data).

Using the ETHx_UDP_STD_REC function block, exactly one data packet is read. The user data are stored in the configured memory area (DATA). The address of the sending device and the data packet length are supplied at the outputs IP_ADR_REM, PORT_REM and LEN. DONE = TRUE and ERR = FALSE indicate that the reading process was successful. If an error was detected during Function Block processing, the error is indicated at the outputs ERR and ERNO. Furthermore, the function block provides information about the receive buffer filling level displayed in bytes (LEV_BY) and data records (LEV_DS).

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

INDEX

Data type	Default value	Range	Unit
BYTE	-	-	-

Index of the standard UDP/IP connection corresponding to the PLC configuration, starting with 0. The function block only works on the connection that was selected by this input, and with the parameters set in the PLC configuration, e.g. port, buffer size, behavior in case of buffer overflow etc.

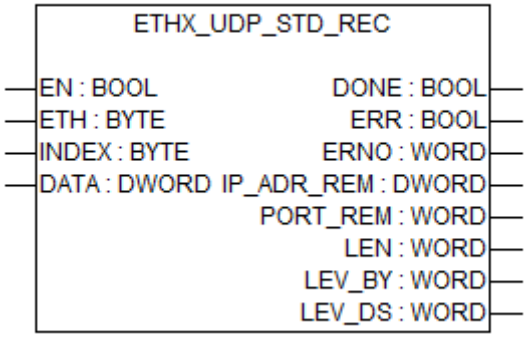
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

> *Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IP_ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

Output IP_ADR_REM (IP address remote) displays the IP address of the sending device which transmitted the received data package. Each byte in IP_ADR_REM represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

PORT_ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PORT_ADR_REM (remote port address) displays the UDP remote port address of the sending device which transmitted the received data package.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEN (length) displays the length of the received data package in bytes.

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY displays the filling level of the receive buffer in bytes. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 6 bytes in the receive buffer (4 bytes for the NODE ID of the sending device, 2 bytes for the specification of the length).

LEV_DS

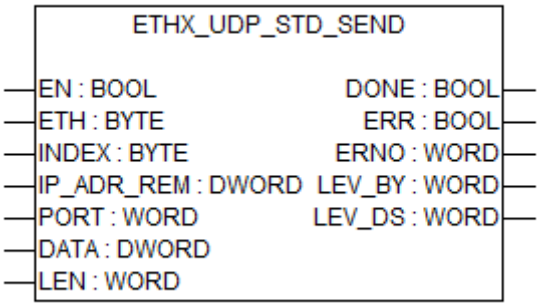
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
UDPStdRec (EN    := UDPStdRec_EN,
           ETH   := UDPStdRec_ETH,
           INDEX := UDPStdRec_INDEX,
           DATA  := ADR(UDPStdRec_DATA));


UDPStdRec_DONE       := UDPStdRec.DONE;
UDPStdRec_ERR        := UDPStdRec.ERR;
UDPStdRec_ERNO       := UDPStdRec.ERNO;
UDPStdRec_IP_ADR_REM := UDPStdRec.IP_ADR_REM;
UDPStdRec_PORT_REM   := UDPStdRec.PORT_REM;
UDPStdRec_LEN        := UDPStdRec.LEN;
UDPStdRec_LEVR_BY    := UDPStdRec.LEV_BY;
UDPStdRec_LEVR_DS    := UDPStdRec.LEV_DS;
```

ETHx_UDP_STD_SEND

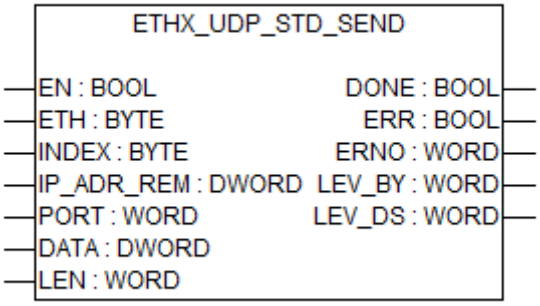


The function block ETHx_UDP_STD_SEND is used to transmit data packets via the standard UDP/IP protocol of the Ethernet interface. The specified packages are stored in the transmit buffer. From there, the operating system hands over the data packets to the Ethernet interface in order to transmit them to the target address specified at input IP_ADR_REM. The transmit buffer size is defined by the PLC configuration. Output DONE indicates that the specified data packet has been stored in the transmit buffer or that an error occurred during function block processing. If an error was detected during function block processing, the error is additionally indicated at the outputs ERR and ERNO. In case of an error, the data packet has to be transmitted again.

The function block ETHx_UDP_STD_SEND cannot store data packets to the transmit buffer until the Ethernet UDP/IP processing is set in the controller configuration (see also the PLC configuration of the standard UDP/IP processing).

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

INDEX

Data type	Default value	Range	Unit
BYTE	-	-	-

Index of the standard UDP/IP connection corresponding to the PLC configuration, starting with 0. The function block only works on the connection that was selected by this input, and with the parameters set in the PLC configuration, e.g. port, buffer size, behavior in case of buffer overflow etc.

IP_ADR_REM

Data type	Default value	Range	Unit
DWORD	-	-	-

At this input, the IP address of the recipient is specified. Each byte in IP_ADR_REM (IP address remote) represents one octet of the address.

Example

IP address 192.15.24.2

IP_ADR (hex) 16#C00F1802

IP_ADR (dec) 3222214658

PORT

Data type	Default value	Range	Unit
WORD	-	-	-

The target port of the receiver is specified here (port to send to).

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

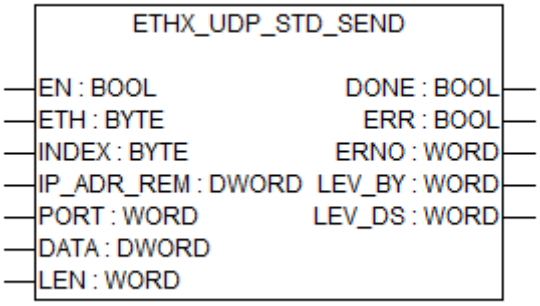
> *Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.*

LEN

Data type	Default value	Range	Unit
WORD	-	1...1464	-

At input LEN (length), the number of user data bytes is specified for the specified package.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_BY

Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_BY (level in bytes) displays the filling level (in bytes) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

One data packet occupies output LEN + 8 bytes in the transmit buffer (4 bytes for the IP address of the recipient, 2 bytes for the specification of the length and 2 bytes for the timeout period).

LEV_DS

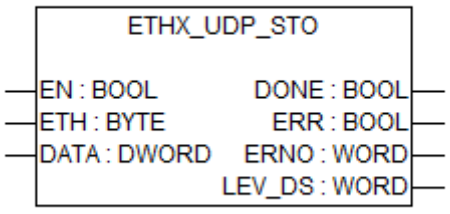
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level (in data records) of the transmit buffer. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
UDPStdSend (EN     := UDPStdSend_EN,
            ETH    := UDPStdSend_ETH,
            INDEX  := UDPStdSend_INDEX,
            IP_ADR := UDPStdSend_IP_ADR,
         PORT   := UDPStdSend_PORT,
            DATA   := ADR(UDPStdSend_DATA),
            LEN    := UDPStdSend_LEN);


UDPStdSend_DONE   := UDPStdSend.DONE;
UDPStdSend_ERR    := UDPStdSend.ERR;
UDPStdSend_ERNO   := UDPStdSend.ERNO;
UDPStdSend_LEV_BY := UDPStdSend.LEV_BY;
```

ETHx_UDP_STO



The ETHx_UDP_STO function block reads lost data packets from the timeout data buffer and stores the user data to the specified memory area.

During the transmission of a data packet, the success of the transmission is monitored by an adjustable timeout period. When this time is exceeded, distinctive information of the data packet are stored in the timeout buffer.

These are:

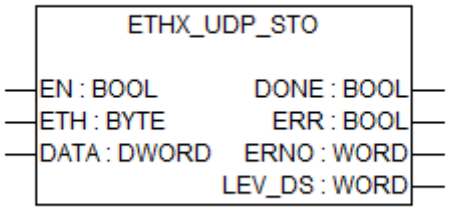
- the IP address of the receiver (4 bytes)
- header data of the data set (the number is specified with the controller configuration of the UDP/IP processing).

The buffer length can as well be set using the controller configuration of the UDP/IP processing. The buffer is constructed as a circular buffer (FIFO). If the buffer is full, the oldest entry in the buffer is overwritten. When a rising edge occurs at input EN, the ETHx_UDP_STO function block verifies whether a data packet is stored in the buffer and makes the information mentioned above available for the user (starting at the variable specified at input DATA).

The function block ETHx_UDP_STO cannot be used until the Ethernet UDP/IP processing is set in the controller configuration. Additionally, input TOUT of the transmit function block ETHx_UDP_SEND must be <> 0.

Available as of runtime system:	V2.4
Included in library:	Ethernet_AC500_V10.lib
Type	Function block with historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

ETH BYTE (Ethernet index number)

At input ETH, the Ethernet index number is specified. The assignment is as follows:

1	Ethernet interface at onboard Ethernet 1
2	Ethernet interface at onboard Ethernet 2
11	Ethernet interface at Communication Module slot 1 *)
21	Ethernet interface at Communication Module slot 2 *)
31	Ethernet interface at Communication Module slot 3 *)
41	Ethernet interface at Communication Module slot 4 *)
51	Ethernet interface at Communication Module slot 5 *)
61	Ethernet interface at Communication Module slot 6 *)

*) The Communication Modules are serially numbered from right to left, starting with module number 1.

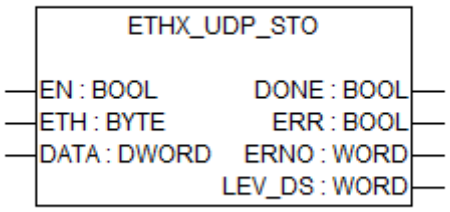
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable to which the user data shall be copied. The address specified at DATA has to belong to a variable of the type ARRAY or STRUCT.

Set the variable size to the maximum expected amount of data in order to avoid overlapping of memory areas.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

LEV_DS

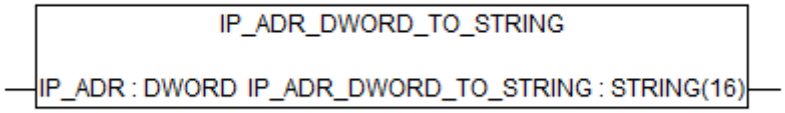
Data type	Default value	Range	Unit
WORD	-	-	-

Output LEV_DS (level in data sets) displays the filling level of the receive buffer in data records. The displayed value is updated as long as EN is TRUE and applies to the input values read with the rising edge at input EN.

Function call in ST

```
UDPStdSto (EN   := UDPStdSto_EN,
           ETH  := UDPStdSto_ETH,
           DATA := ADR(UDPStdSto_DATA));


UDPStdSto_DONE   := UDPStdSto.DONE;
UDPStdSto_ERR    := UDPStdSto.ERR;
UDPStdSto_ERNO   := UDPStdSto.ERNO;
UDPStdSto_LEV_DS := UDPStdSto.LEV_DS;
```

IP_ADR_DWORD_TO_STRING



The function block IP_ADR_DWORD_TO_STRING converts an IP address given in the DWORD format into an IP address in the STRING format.

The IP address of an Ethernet device can be represented in different ways. One of the best known formats of an IP address is the STRING format, composed of four numbers between 0 and 255, separated by dots (e.g. 192.15.24.2). The function block IP_ADR_DWORD_TO_STRING converts an IP address given in the DWORD format into an IP address in the STRING format.

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function

Input description

IP_ADR

Data type	Default value	Range	Unit
DWORD	-	-	-

At input IP_ADR, the IP address of the host to be pinged is specified.

Output description

STRING(16) The output of the function block IP_ADR_DWORD_TO_STRING outputs the converted IP address in STRING(16) format.

Example: The following value is applied at function block input IP_ADR:

IP_ADR_WORD: (hex) 16#C00F1802 or

The converted value is displayed at the function block output:

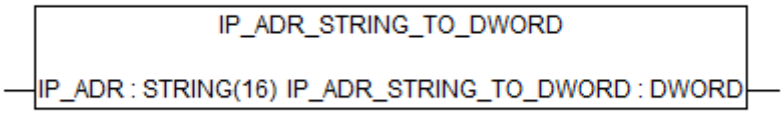
IP_ADR_WORD: (dec) 3222214658

IP_ADR_STRING: '192.15.24.2'

Function call in ST

```
IP_ADR_STRING := IP_ADR_DWORD_TO_STRING(IP_ADR_DWORD);
```

IP_ADR_STRING_TO_DWORD



The function block IP_ADR_STRING_TO_DWORD converts an IP address given in the STRING format into an IP address in the DWORD format.

The IP address of an Ethernet device can be represented in different ways. One of the best known formats of an IP address is the STRING format, composed of four numbers between 0 and 255, separated by dots (e.g. 192.15.24.2). The function block IP_ADR_STRING_TO_DWORD converts an IP address given in the STRING format into an IP address in the DWORD format.

Available as of runtime system:	V1.0
Included in library:	Ethernet_AC500_V10.lib
Type	Function

Input description

IP_ADR At input IP_ADR, the IP address in STRING(16) format is specified.

Output description

(Output) The output of the function block IP_ADR_STRING_TO_DWORD outputs the converted IP address in DWORD format.

Example: The following value is applied at function block input IP_ADR:

IP_ADR_STRING: '192.15.24.2'

The converted value is displayed at the function block output:

IP_ADR_WORD: (hex) 16#C00F1802 or

IP_ADR_WORD: (dec) 3222214658

Function call in ST

```
IP_ADR_DWORD := IP_ADR_STRING_TO_DWORD(IP_ADR_STRING);
```

1.5.4.13.2 Structures

ETH_EMAIL_DATA_TYPE

This structure is used to supply the function block ETH_SMTP_EMAIL_SEND with all necessary data to send an email via the SMTP server configured in the PLC.

The addresses and textual content of the email need to be created and defined by the user. Only references (POINTER TO STRING) are used in the structure. The contents of this structure and the memory pointed to may not be changed during operation of the function block ETH_SMTP_EMAIL_SEND.

At least one valid TO address is required for the function block ETH_SMTP_EMAIL_SEND to work. Even if using CC and/or BCC at last one TO address must be defined.

Type and the content need to be created and defined by the user.

Available as of runtime system:	V2.1	Remark:
Included in library:	Ethernet_AC500_V10.lib	

Visible variable	Type	Default value	Description
psTOAddr	POINTER TO STRING(255)	0	Pointer to a string containing TO address(es) separated by semicolon
psCCAddr	POINTER TO STRING(255)	0	Pointer to a string containing CC address(es) separated by semicolon
psBCCAddr	POINTER TO STRING(255)	0	Pointer to a string containing BCC address(es) separated by semicolon
apsBody	ARRAY[0..19] OF POINTER TO STRING(255)	0	Up to 20 pointers containing text lines for the email body
atsFiles	ARRAY[0..9] OF ETH_EMAIL_FILE_REF_TYPE	0	Up to 10 file attachments to the email via the ETH_EMAIL_FILE_REF_TYPE structure
byPrio	BYTE	0	Priority 0=normal, 1=very high, 2=high, 3=normal, 4=low, 5=very low
sSubject	STRING(255)	’’	The subject of the email

ETH_EMAIL_FILE_REF_TYPE

This structure is used to supply the function block ETH_SMTP_EMAIL_SEND with a file attachment. The ETH_EMAIL_DATA_TYPE structure holds an array of 10 ETH_EMAIL_FILE_REF_TYPE structures, so up to 10 files can be attached to each email.

The file name of an attachment can be different from the file name in the PLC's filesystem. The variable psNameInMail only defines the name of the file inside the email, while psFilePath must contain the full path including name of the file to be attached.

Type and the content need to be created and defined by the user.

Available as of runtime system:	V2.1	Remark:
Included in library:	Ethernet_AC500_V10.lib	

Visible variable	Type	Default value	Description
psNameInMail	POINTER TO STRING(255)	0	Pointer to a string containing a name to be used for the file in the email. (Can differ from the actual file name in the PLC.)
psFilePath	POINTER TO STRING(255)	0	Pointer to a string containing the full path to the file to be attached including the file name

ETH_MOD_FCT22_TYPE

This structure is used to handle the Modbus function 22 with the function block ETH_MOD_MAST ↳ *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202*. The structure has to be used at the DATA input with an ADR(instance_of_struct) operator.

Modbus function 22 could only be used with CPUs with onboard Ethernet.

Type and the content need to be created and defined by the user.

Available as of runtime system:	V2.2	Remark:
Included in library:	Ethernet_AC500_V10.lib	

Visible variable	Type	Default value	Description
wAND_Mask	WORD	0	AND mask
wOR_Mask	WORD	0	OR mask

ETH_MOD_FCT23_TYPE

This structure is used to handle the Modbus function 23 with the function block ETH_MOD_MAST *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202*. The structure has to be used at the DATA input with an ADR(instance_of_struct) operator.

Modbus function 23 could only be used with CPUs with onboard Ethernet.

Type and the content need to be created and defined by the user.

Available as of runtime system:	V2.2	Remark:
Included in library:	Ethernet_AC500_V10.lib	

Visible variable	Type	Default value	Description
pByDataWrite	POINTER TO BYTE	0	Pointer to buffer containing data to write. see FCT 16 input DATA
pByDataRead	POINTER TO BYTE	0	Pointer to buffer to store data to read. see FCT 03 input DATA
wDataAddressRead	WORD	0	Address of the data to be read. see FCT 16 input ADDR
wNumDataUnitsRead	WORD	0	Number of data units to be read. see FCT 16 input NB

1.5.4.14 EtherCAT library

Library file name: **EtherCAT_AC500_V13.lib**

The library EtherCAT_AC500_V13.lib is intended to be used with the EtherCAT master communication module. Beside the cyclic IO communication, which is configured in the EtherCAT master configuration and controlled by the runtime system, additional features are implemented in the EtherCAT master. This library provides the possibility to use these features within a PLC application. With the contained function blocks the following functionality can be realized:

- reading of diagnosis information
- reading and writing of service data objects (SDOs)
- reading of handshake error counters of synchronous IO update

Additional system libraries are required by this library:

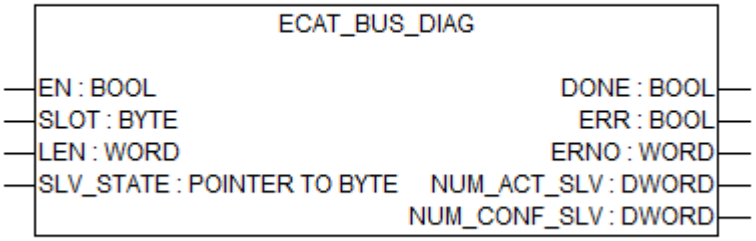
- SysExt_AC500_V10.lib
- SysLibTime.lib

The EtherCAT library and the required system libraries will be automatically added to the PLC application when an EtherCAT master communication module is configured in the PLC configuration.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

1.5.4.14.1 Function blocks

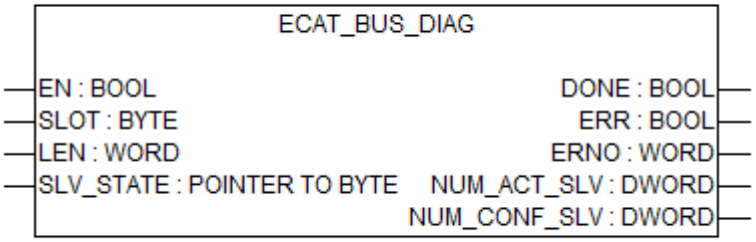
ECAT_BUS_DIAG



Parameter	Value
Included in library	EtherCAT_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	DIAG

The function block ECAT_BUS_DIAG is used to read the status and to return status information for each slave as well as the number of configured slaves and the number of active slaves of an EtherCAT Master.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

LEN

Data type	Default value	Range	Unit
UINT	0	1 ... 65535	byte

LEN (length) tells the function block how large the buffer in SLV_STATE is. For each slave 4 bytes are needed.

SLV_STATE

Data type	Default value	Range	Unit
POINTER TO ARRAY OF BYTE	0	-	-

The address of the buffer is required in order to write the state of each slave ↳ *"STATE" on page 1310*. The size of the buffer is directly related to the number of slaves. A byte for each slave is needed. In case the buffer is too small, an error will be returned. If the value of LEN does not reflect correctly the size of the buffer, this might be overrun with unpredictable consequences.

Example

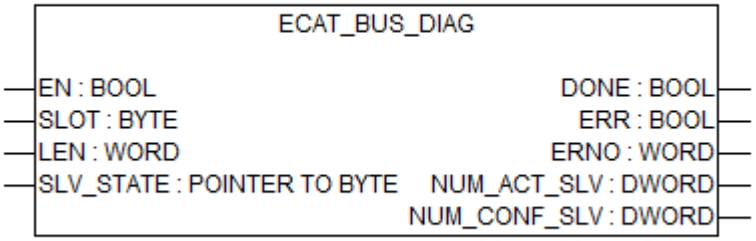
A master is configured with 20 slaves.

LEN and SLV_STATE should be configured as follows.

ECATBusDiag_LEN := 20;

ECATBusDiag_SLV_STATE: ARRAY [1..20] of BYTE;

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	-	-

Output indicates the processing state of the function block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_ACT_SLV

Data type	Default value	Range	Unit
DWORD	0	0-256	-

The variable NUM_ACT_SLV (number of active slaves) contains the number of active slaves. Active slaves exchange cyclic data with the master.

NUM_CONF_SLV

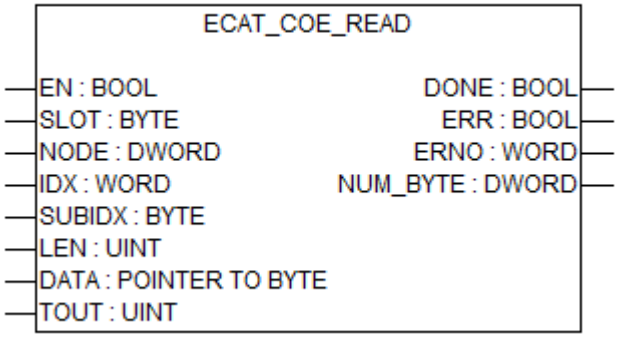
Data type	Default value	Range	Unit
DWORD	0	0-256	-

NUM_CONF_SLV (number of configured slaves) contains the number of configured slaves evaluated in the configuration file.

Function call in ST

```
ECAT_BUS_DIAG(
EN       := ECATBusDiag_EN,
SLOT     := ECATBusDiag_SLOT,
LEN      := ECATBusDiag_LEN,
SLV_STATE:=ADR(ECATBusDiag_SLV_STATE);


ECATBusDiag_DONE        := ECAT_BUS_DIAG.DONE;
ECATBusDiag_ERR         := ECAT_BUS_DIAG.ERR;
ECATBusDiag_ERNO        := ECAT_BUS_DIAG.ERNO;
ECATBusDiag_NUM_ACT_SLV := ECAT_BUS_DIAG.NUM_ACT_SLV;
ECATBusDiag_NUM_CONF_SLV:= ECAT_BUS_DIAG.NUM_CONF_SLV;
```

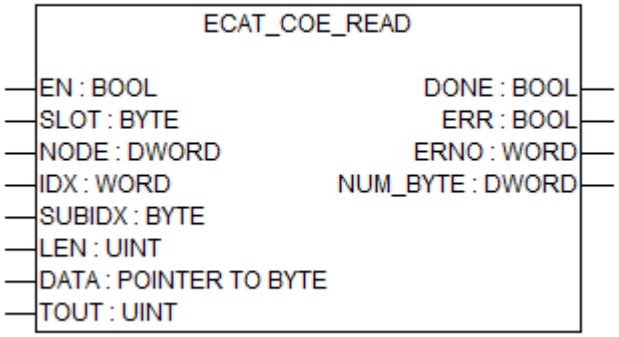
ECAT_COE_READ



Parameter	Value
Included in library	EtherCAT_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	CoE

The function block ECAT_COE_READ is used to read SD objects (SDOs) from a slave.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

IDX (index)

Data type	Default value	Range	Unit
WORD	0	1 ... 65535 (depending on the EtherCAT slave)	-

Index of the object which should be read/written.

SUBIDX

Data type	Default value	Range	Unit
BYTE	0	1 ... 255 (depending on the EtherCAT slave)	-

Subindex of the object, which should be read/written.

LEN

Data type	Default value	Range	Unit
UINT	0	1 ... 65535	byte

LEN (length) tells the function block how large the buffer in SLV_STATE is. For each slave 4 bytes are needed.

DATA

Data type	Default value	Range	Unit
POINTER TO ARRAY OF BYTE	0	-	-

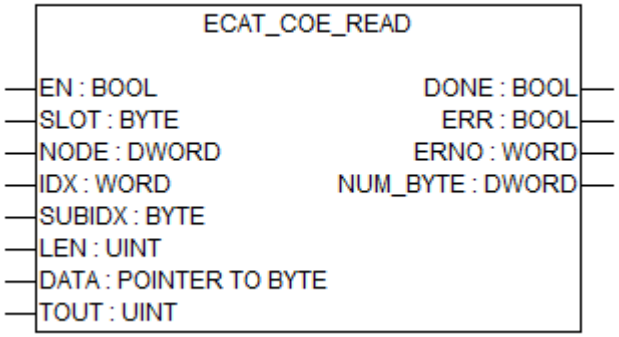
Address of the data buffer where the value of the object (Index: subindex) should be read. The needed buffer depends on the queried object.

TOUT

Data type	Default value	Range	Unit
UINT	0	0-65535	ms

Timeout constraint for the function block. If this is exceeded, the function block returns with ERR set to TRUE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_BYTE

Data type	Default value	Range	Unit
DWORD	0	depending on the EtherCAT slave	-

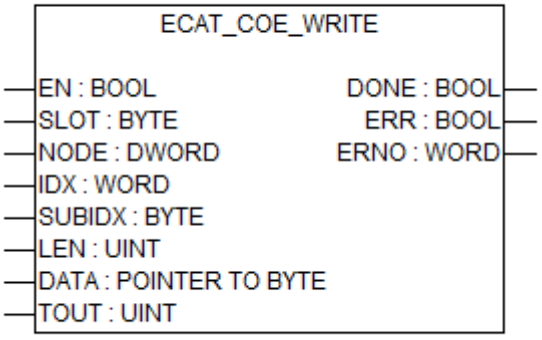
Output returns how many bytes were written into the buffer.

Function call in ST

```
ECAT_COE_READ(
EN    := ECATCoeRead_EN,
SLOT  := ECATCoeRead_SLOT,
LEN   := ECATCoeRead_LEN,
NODE  :=ECATCoeRead_NODE,
IDX   :=ECATCoeRead_IDX,
SUBIDX:=ECATCoeRead_SUBIDX,
LEN   :=ECATCoeRead_LEN,
```

```
DATA   :=ADR(ECATCoeRead_DATA),
TOUT   :=ECATCoeRead_TOUT);


ECATCoeRead_DONE     := ECAT_COE_READ.DONE;
ECATCoeRead_ERR      := ECAT_COE_READ.ERR;
ECATCoeRead_ERNO     := ECAT_COE_READ.ERNO;
ECATCoeRead_NUM_BYTE:= ECAT_COE_READ.NUM_BYTE;
```

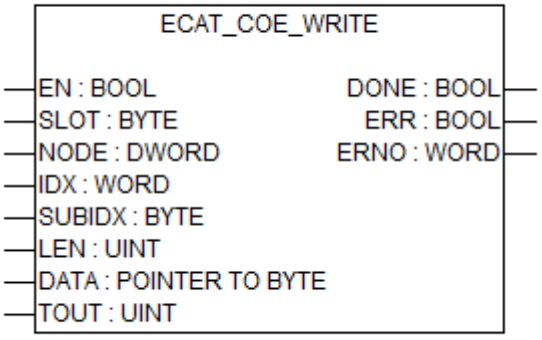
ECAT_COE_WRITE



Parameter	Value
Included in library	EtherCAT_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	CoE

ECAT_COE_WRITE writes a value to an SD Object in a slave. This object is identified with the pair Index:Subindex.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

IDX (index)

Data type	Default value	Range	Unit
WORD	0	1 ... 65535 (depending on the EtherCAT slave)	-

Index of the object which should be read/written.

SUBIDX

Data type	Default value	Range	Unit
BYTE	0	1 ... 255 (depending on the EtherCAT slave)	-

Subindex of the object, which should be read/written.

LEN

Data type	Default value	Range	Unit
UINT	0	1 ... 65535	byte

LEN (length) tells the function block how large the buffer in SLV_STATE is. For each slave 4 bytes are needed.

DATA

Data type	Default value	Range	Unit
POINTER TO ARRAY OF BYTE	0	-	-

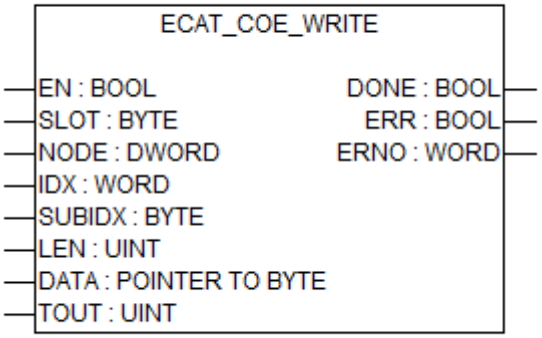
Address of the data buffer where the value of the object (Index: subindex) should be written. The needed buffer depends on the queried object.

TOUT

Data type	Default value	Range	Unit
UINT	0	0-65535	ms

Timeout constraint for the function block. If this is exceeded, the function block returns with ERR set to TRUE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

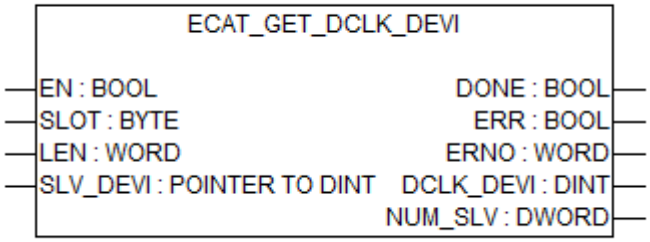
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ECAT_COE_WRITE(
EN              := ECATCoeWrite_EN,
SLOT            := ECATCoeWrite_SLOT,
NODE            := ECATCoeWrite_NODE,
IDX             := ECATCoeWrite_IDX,
SUBIDX          := ECATCoeWrite_SUBIDX,
```

```
LEN             := ECATCoeWrite_LEN,
DATA            := ADR(ECATCoeWrite_DATA),
TOUT            := ECATCoeWrite_TOUT);


ECATCoeWrite_DONE     := ECAT_COE_WRITE.DONE;
ECATCoeWrite_ERR      := ECAT_COE_WRITE.ERR;
ECATCoeWrite_ERNO     := ECAT_COE_WRITE.ERNO;
```

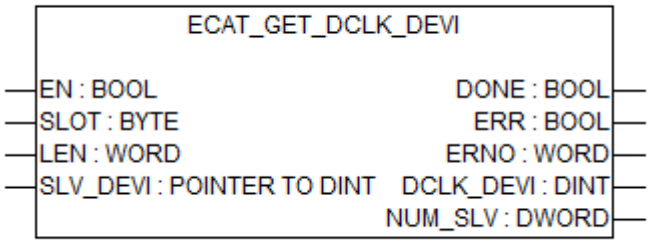
ECAT_GET_DCLK_DEVI



Parameter	Value
Included in library	EtherCAT_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	DIAG

ECAT_GET_DCLK_DEVI returns the distributed clock deviation for the EtherCAT master and for each slave individually.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

LEN

Data type	Default value	Range	Unit
UINT	0	1 ... 65535	byte

LEN (length) tells the function block how large the buffer in SLV_STATE is. For each slave 4 bytes are needed.

SLV_DEVI

Data type	Default value	Range	Unit
POINTER TO ARRAY OF DINT	0	-	-

The Input SLV_DEVI (slave deviation) provides the address of an Arrary of DINT where the distributed clock deviation of each slave is written to.

Each Element of the array has the range -2147483647... 2147483647 in unit ns.

The required size of the array is calculated as number of configured EtherCAT slaves multiplied with the size of DINT (4 bytes).

The size of the assigned array is provided at the Input LEN. If Input LEN is too small, an error will be returned.

If the value of LEN does not match with the assigned array, it can be overwritten. This will cause unexpected behavior.

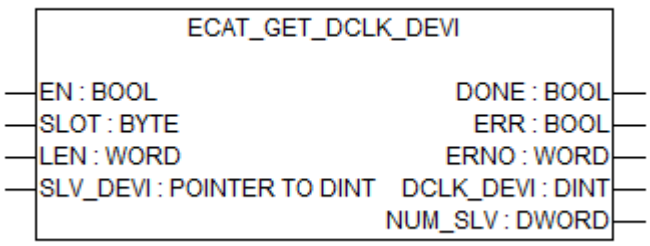
Example

A system with a master and 5 slaves is configured.

ECAT_DCLK_DEVI_LEN := 20;

ECAT_DCLK_DEVI_SLV_DEVI: ARRAY [1..5] of DINT;

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DCLK_DEVI

Data type	Default value	Range	Unit
DINT	0	-2147483647...2147483647	ns

DCLK_DEVI (distributed clock deviation) shows the broadcast system time difference.

NUM_SLV

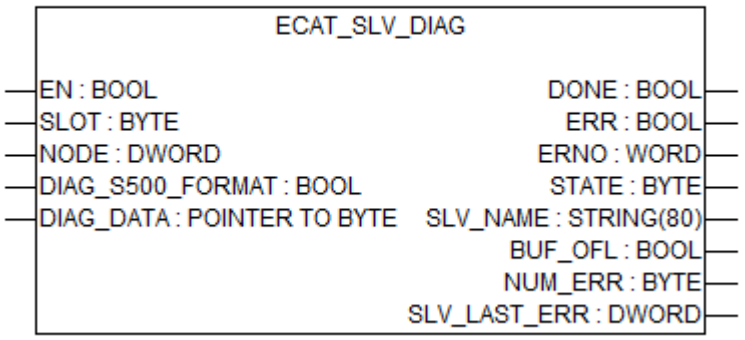
Data type	Default value	Range	Unit
DWORD	0	1-255	-

NUM_SLV shows the number of slaves for which the master has an deviation. In case the function block reports an insufficient buffer, the value of NUM_SLV has to be compared with the provided LEN value.

Function call in ST

```
ECAT_GET_DCLK_DEVI(
EN       := ECATGetDclkDevi_EN,
SLOT     := ECATGetDclkDevi_SLOT,
LEN      := ECATGetDclkDevi_LEN,
SLV_DEVI:=ADR(ECATGetDclkDevi_SLV_DEVI);


ECATGetDclkDevi_DONE     := ECAT_GET_DCLK_DEVI.DONE;
ECATGetDclkDevi_ERR      := ECAT_GET_DCLK_DEVI.ERR;
ECATGetDclkDevi_ERNO     := ECAT_GET_DCLK_DEVI.ERNO;
ECATGetDclkDevi_DCLK_DEVI:= ECAT_GET_DCLK_DEVI.DCLK_DEVI;
ECATGetDclkDevi_NUM_SLV  := ECAT_GET_DCLK_DEVI.NUM_SLV;
```

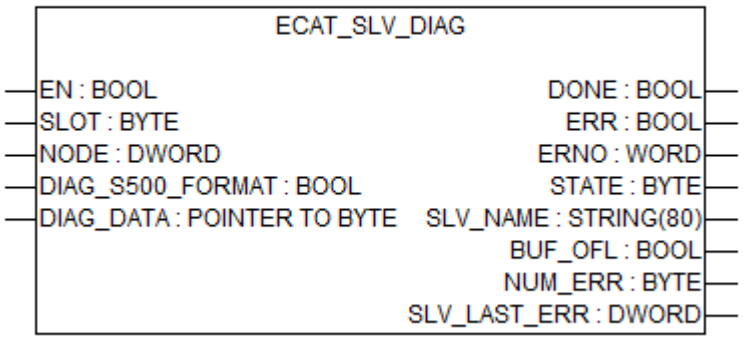
ECAT_SLV_DIAG



Available as of runtime system	V1.3.0
Included in library	EtherCAT_AC500_V13.lib
Function block Type	Function block with historical values

With the function block ECAT_SLV_DIAG status and diagnosis information of an EtherCAT slave can be read.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

DIAG_S500_FORMAT

Data type	Default value	Range	Unit
BOOL	FALSE	FALSE, TRUE	-

The Input DIAG_S500_FORMAT specifies the format of the diagnosis data which is provided in the buffer DIAG_DATA.

When DIAG_S500_FORMAT is TRUE the diagnosis data is in S500 Format. Otherwise it is in emergency data structure.

DIAG_DATA

Data type	Default value	Range	Unit
POINTER TO BYTE	0	-	-

The Input DIAG_DATA provides the pointer to buffer where the received diagnosis data is copied to.

The size of the provided buffer depends on the data structure of the diagnosis data which is specified with the Input DIAG_S500_FORMAT.

When the Input DIAG_S500_FORMAT is TRUE the buffer must have a size of at least 25 bytes.

The diagnosis data has the data type Array [1 ... 5] OF DIAGDATA.

```
TYPE DIAG_DATA :(*Error format for S500 slaves*) STRUCT
ERR_CLASS     : BYTE        :=0;   (*error class*)
SLAVE_NR      : BYTE        :=0;   (*slave number*)
MODULE_NR     : BYTE        :=0;   (*module number*)
CHANNEL_NR    : BYTE        :=0;   (*channel number*)
ERROR_NR      : BYTE        :=0;   (*error number*)
END_STRUCT
END_TYPE
```

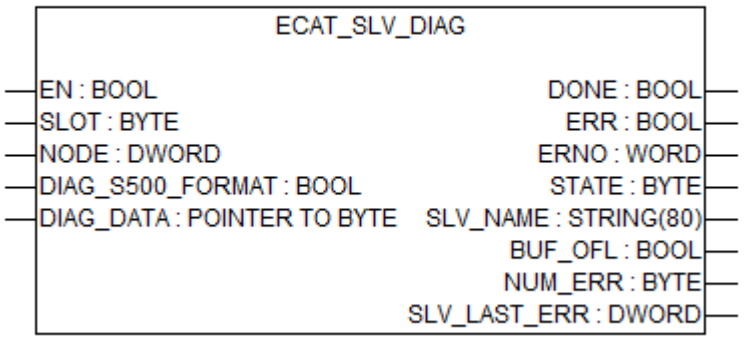
When the Input DIAG_S500_FORMAT is FALSE the buffer must have a size of at least 40 bytes.

The diagnosis data has the data type Array [1..5] OF ETHERCAT_MASTER_SLV_EMERGENCY_T.

```
TYPE ETHERCAT_MASTER_SLV_EMERGENCY_T: STRUCT
usErrorCode:    WORD               :=0;(*error code*)
bErrorRegister: BYTE               :=0;(*error register*)
abErrorData   : ARRAY [0..4]of BYTE :=5(0);(*error data*)
END_STRUCT
END_TYPE
```

The buffer should not be smaller than the sizes given for each format. Otherwise it will be overwritten and causes unexpected behavior.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Table 77: Current slave state

Data type	Default value	Range	Unit
BYTE	16#FF	01: INIT 02: PREOP 04: SAFEOP 08: OPERATIONAL 254: unknown 255: no response	-

SLV_NAME

Data type	Default value	Range	Unit
STRING	empty string	-	-

Output SLV_NAME provides the current slave name.

BUF_OFL

Data type	Default value	Range	Unit
BOOL	FALSE	FALSE, TRUE	-

Output BUF_OFL (buffer overflow) indicated whether there was overflow of the internal diagnosis buffer. In this case emergency messages were lost.

NUM_ERR

Data type	Default value	Range	Unit
BYTE	0	0 ... 5	-

Output NUM_ERR provides the number of elements (Diagnosis messages) contained within the buffer DIAG_DATA.

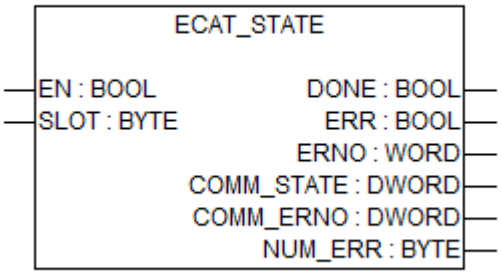
Example 1 NUM_ERR returns the value of 3 and DIAG_S500_FORMAT was TRUE. This means that the first 3 elements of DIAG_DATA contain relevant information.

Example 2 A NUM_ERR of 4 and DIAG_S500_FORMAT was FALSE means that DIAG_DATA has 32 relevant bytes (8 per error).

Function call in ST

```
ECAT_SLV_DIAG(
EN                  := ECATSlvDiag_EN,
SLOT                := ECATSlvDiag_SLOT,
SLV                 := ECATSlvDiag_SLV,
DIAG_S500_FORMAT    := ECATSlvDiag_DIAG_S500_FORMAT,
DIAG_DATA           :=ADR(ECATSlvDiag_DIAG_DATA));


ECATSlvDiag_DONE        := ECAT_SLV_DIAG.DONE;
ECATSlvDiag_ERR         := ECAT_SLV_DIAG.ERR;
ECATSlvDiag_ERNO        := ECAT_SLV_DIAG.ERNO;
ECATSlvDiag_SLV_NAME    := ECAT_SLV_DIAG.SLV_NAME;
ECATSlvDiag_BUF_OFL     := ECAT_SLV_DIAG.BUF_OFL;
ECATSlvDiag_NUM_ERR     := ECAT_SLV_DIAG.NUM_ERR;
```

ECAT_STATE

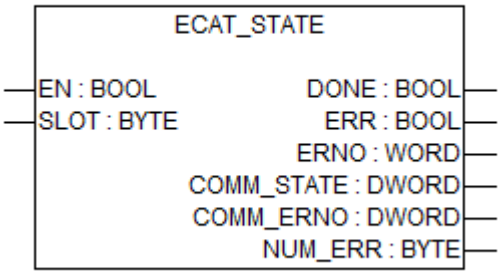


Available as of runtime system	V1.3.0
Included in library	EtherCAT_AC500_V13.lib
Function block Type	Function block with historical values

The function block ECAT_STATE is used to read the state and possible errors of an EtherCAT fieldbus.

The function block ECAT_STATE returns the status of an EtherCAT Communication Module. The function block also delivers - when present - a communication error and the number of errors since power-up.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

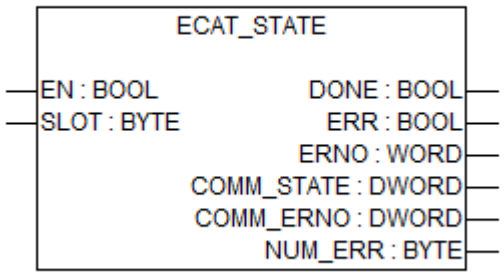
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

COMM_STATE

Data type	Default value	Range	Unit
DWORD	0	0...4	-

COMM_STATE returns the state of the field bus related to chosen master. The possible codes for COMM_STATE are:

```
UNKNOWN     0x00000000
OFFLINE     0x00000001
STOP        0x00000002
IDLE        0x00000003
OPERATE     0x00000004
```

COMM_ERNO

Data type	Default value	Range	Unit
DWORD	0	-	-

COMM_ERNO returns an error, if present, related to the communication channel. The possible codes for COMM_ERNO are following:

```
SUCCESS                                    0x00000000
WATCHDOG TIMEOUT                                      0xC000000C
INITIALIZATION FAULT                                  0xC0000100
DATABASE ACCESS FAILED                                0xC0000101
(General) CONFIGURATION FAULT                         0xC0000120
INCONSISTENT DATA SET                                 0xC0000121
DATA SET MISMATCH                                     0xC0000122
INSUFFICIENT LICENSE                                  0xC0000123
PARAMETER ERROR                                       0xC0000124
INVALID NETWORK ADDRESS                               0xC0000125
(General) NETWORK FAULT                               0xC0000140
CONNECTION CLOSED                                     0xC0000141
CONNECTION TIMED OUT                                  0xC0000142
LONELY NETWORK                                        0xC0000143
DUPLICATE NODE                                        0xC0000144
CABLE DISCONNECT                                      0xC0000145
TLR_E_ETHERCAT_MASTER                                 0xC0650001
TLR_E_ETHERCAT_MASTER_NO_LINK                         0xC0650002
TLR_E_ETHERCAT_MASTER_ERROR_READING_BUSCONFIG         0xC0650003
TLR_E_ETHERCAT_MASTER_ERROR_PARSING_BUSCONFIG         0xC0650004
TLR_E_ETHERCAT_MASTER_ERROR_BUSSCAN_FAILED            0xC0650005
TLR_E_ETHERCAT_MASTER_NOT_ALL_SLAVES_AVAIL            0xC0650006
TLR_E_ETHERCAT_MASTER_STOPMASTER_ERROR                0xC0650007
TLR_E_ETHERCAT_MASTER_DEINITMASTER_ERROR              0xC0650008
TLR_E_ETHERCAT_MASTER_CLEANUP_ERROR                   0xC0650009
TLR_E_ETHERCAT_MASTER_CRITIAL_ERROR_STATE             0xC065000A
TLR_E_ETHERCAT_MASTER_INVALID_BUSCYCLETIME            0xC065000B
TLR_E_ETHERCAT_MASTER_INVALID_BROKEN_SLV_BEH_PARA     0xC065000C
TLR_E_ETHERCAT_MASTER_WRONG_INTERNAL_STATE            0xC065000D
TLR_E_ETHERCAT_MASTER_WATCHDOG_TIMEOUT_EXPIRED        0xC065000E
TLR_E_ETHERCAT_MASTER_COE_INVALID_SLAVEID             0xC065000F
TLR_E_ETHERCAT_MASTER_COE_NO_RESOURCE                 0xC0650010
TLR_E_ETHERCAT_MASTER_COE_INTERNAL_ERROR              0xC0650011
TLR_E_ETHERCAT_MASTER_COE_INVALID_INDEX               0xC0650012
TLR_E_ETHERCAT_MASTER_COE_INVALID_COMM_STATE          0xC0650013
TLR_E_ETHERCAT_MASTER_COE_FRAME_LOST                  0xC0650014
TLR_E_ETHERCAT_MASTER_COE_TIMEOUT                     0xC0650015
TLR_E_ETHERCAT_MASTER_COE_SLAVE_NOT_ADDRESSABLE       0xC0650016
TLR_E_ETHERCAT_MASTER_COE_INVALID_LIST_TYPE           0xC0650017
TLR_E_ETHERCAT_MASTER_COE_SLAVE_RESPONSE_TOO_BIG      0xC0650018
TLR_E_ETHERCAT_MASTER_COE_INVALID_ACCESSBITMASK       0xC0650019
TLR_E_ETHERCAT_MASTER_COE_WKC_ERROR                   0xC065001A
TLR_E_ETHERCAT_MASTER_SERVICE_IN_USE                  0xC065001B
TLR_E_ETHERCAT_MASTER_INVALID_COMMUNICATION_STATE     0xC065001C
TLR_E_ETHERCAT_MASTER_DC_NOT_ACTIVATED                0xC065001D
TLR_E_ETHERCAT_MASTER_BUS_SCAN_CURRENTLY_RUNNING      0xC065001E
TLR_E_ETHERCAT_MASTER_BUS_SCAN_TIMEOUT                0xC065001F
TLR_E_ETHERCAT_MASTER_BUS_SCAN_NOT_READY_YET          0xC0650020
TLR_E_ETHERCAT_MASTER_BUS_SCAN_INVALID_SLAVE          0xC0650021
```

NUM_ERRS

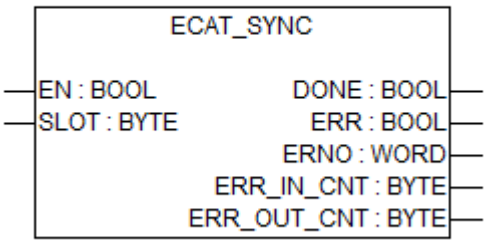
Data type	Default value	Range	Unit
BYTE	0	0 ... 255	-

NUM_ERRS contains the number of errors since the power-up of the Communication Module.

Function call in ST

```
ECAT_STATE(
EN   := ECATState_EN,
SLOT := ECATState_SLOT);


ECATState_DONE           := ECAT_STATE.DONE;
ECATState_ERR            := ECAT_STATE.ERR;
ECATState_ERNO           := ECAT_STATE.ERNO;
ECATState_COMM_STATE     := ECAT_STATE.COMM_STATE;
ECATState_COMM_ERNO      := ECAT_STATE.COMM_ERNO;
ECATState_NUM_ERR     := ECAT_STATE.NUM_ERR;
```

ECAT_SYNC



The function block ECAT_SYNC is used to synchronize an IEC task with the EtherCAT bus cycle.

With the function block ECAT_SYNC the error counters ERR_IN_CNT and ERR_OUT_CNT can be read.

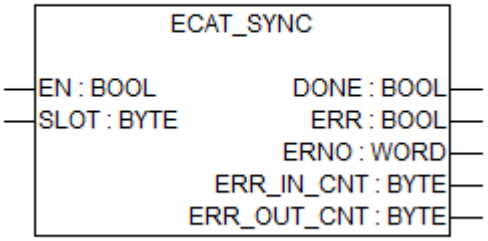
The error counter ERR_IN_CNT is set, if the IEC task was not started within the current bus cycle.

The error counter ERR_OUT_CNT is set, If the task was not finished within the current bus cycle.

The function block ECAT_SYNC requires Communication Module firmware version 2.4.11 or higher.

The function block ECAT_SYNC requires CPU firmware version 2.1.x or higher.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

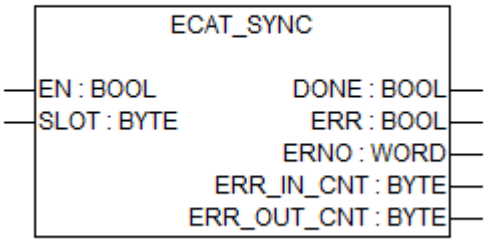
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERR_IN_CNT

Data type	Default value	Range	Unit
BYTE	0	0 ... 255	-

The variable ERR_IN_CNT (error input counter) contains the number of input errors.

The error counter ERR_IN_CNT will be incremented, if the synchronized IEC task was not started within the current bus cycle.

ERR_OUT_CNT

Data type	Default value	Range	Unit
BYTE	0	0 ... 255	-

The variable ERR_OUT_CNT (error output counter) contains the number of output errors.

The error counter ERR_OUT_CNT will be incremented, if the started task was not finished within the current bus cycle.

Function call in ST

```
ECAT_SYNC(
EN   := ECATSYNC_EN,
SLOT := ECATSYNC_SLOT);


ECATSYNC_DONE           := ECAT_SYNC.DONE;
ECATSYNC_ERR            := ECAT_SYNC.ERR;
ECATSYNC_ERNO           := ECAT_SYNC.ERNO;
ECATSYNC_ERR_IN_CNT     := ECAT_SYNC.ERR_IN_CNT;
ECATSYNC_ERR_OUT_CNT    := ECAT_SYNC.ERR_OUT_CNT;
```

1.5.4.15 Extended EtherCAT library

Library file name: **Ethercat_ext_AC500_Vx.lib**

This library is used to control the EtherCAT network and the connected slave devices. It is also used for diagnostics of the EtherCAT network and the connected slave devices.

Preconditions

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The following libraries are required to use the function blocks of the EtherCAT library:

- *CMN_AC500_V24.lib (V 1.0.0)*
- *SysInt_AC500_V10.lib (V 1.4.3)*

1.5.4.15.1 Function blocks

ECAT_SLV_SET_STATE

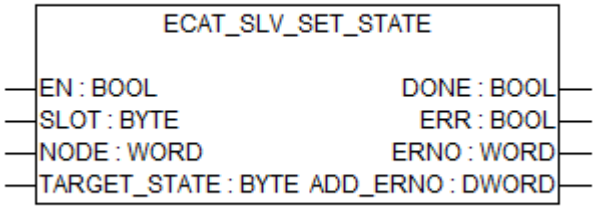


Table 78: General information

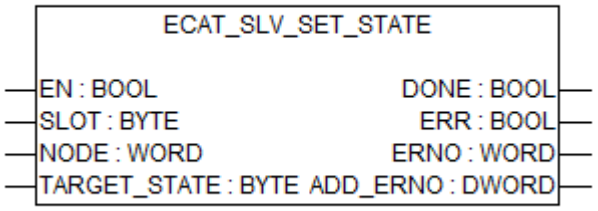
Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

ECAT_SLV_SET_STATE can be used to change state of the “EtherCAT State Machine” of an EtherCAT slave. The direct state transitions are:

- INIT -> PREOP
- PREOP -> SAFEOP
- SAFEOP -> OP
- OP -> SAFEOP
- OP -> PREOP
- OP -> INIT
- SAFEOP -> PREOP
- SAFEOP -> INIT
- PREOP -> INIT

This POU allows direct state changes and state changes over intermediate states, e.g. for a change from “init” to “Operational”.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

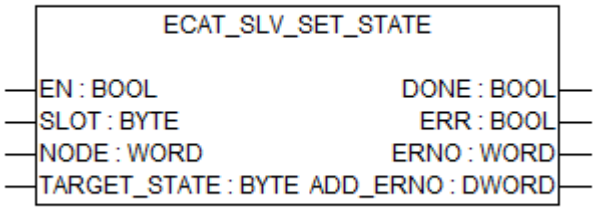
TARGET_STATE

Data type	Default value	Range	Unit
BYTE	0	1, 2, 4, 8	-

Possible target states of the slave:

- 1: INIT
- 2: PREOPERATIONAL
- 4: SAFEOPERATIONAL
- 8: OPERATIONAL

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

ECAT_SLV_GET_STATE

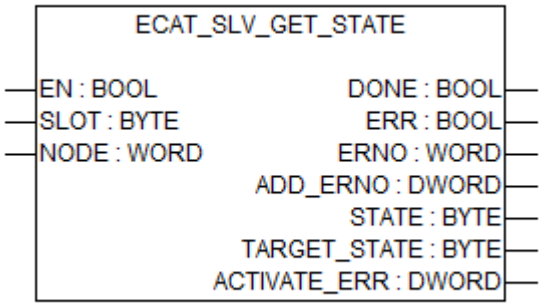
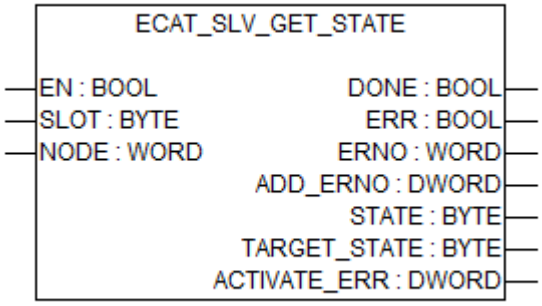


Table 79: General information

Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

The function block ECAT_SLV_GET_STATE can be used to change state of the “EtherCAT State Machine” of an EtherCAT slave. If exists, the error of the last state change is also read with this function block.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

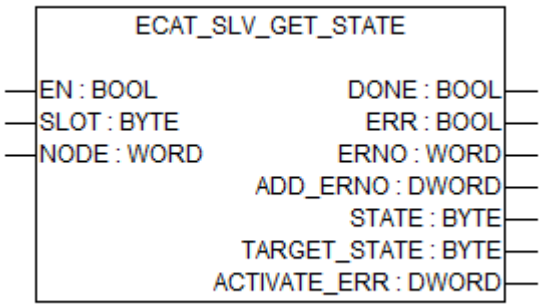
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

STATE

Data type	Default value	Range	Unit
BYTE	0	1, 2, 4, 8	-

Output STATE provides the current state of the selected slave. The following values are possible:

- 1: INIT
- 2: PREOPERATIONAL
- 4: SAFEOPERATIONAL
- 8: OPERATIONAL

TARGET_STATE

Data type	Default value	Range	Unit
BYTE	0	1, 2, 4, 8	-

Output TARGET_STATE provides the target_value of the last target in ECAT_SLV_SET_STATE. The following target values are possible:

- 1: INIT
- 2: PREOPERATIONAL
- 4: SAFEOPERATIONAL
- 8: OPERATIONAL

ACTIVATE_ERR

Data type	Default value	Range	Unit
DWORD	0	-	-

ACTIVATE_ERR shows the activation error reported by the slave.

ECAT_BUS_SET_STATE

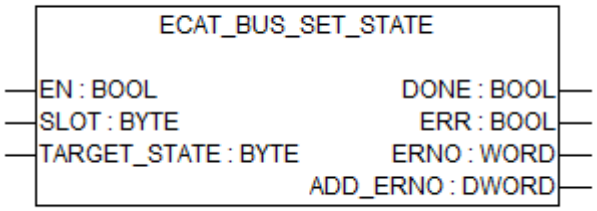
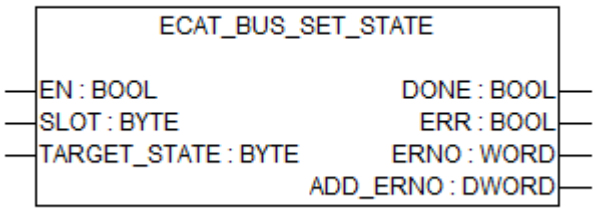


Table 80: General information

Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

ECAT_BUS_SET_STATE can be used to change the state of all connected and configured EtherCAT slaves. All transition between the four states “Init”, “Pre-Operational”, “Safe Operational” and “Operational” are possible.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

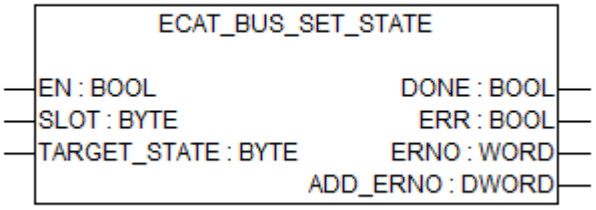
TARGET_STATE (target state)

Data type: BYTE, default value: 0, range: 1, 2, 4, 8

Target State of the bus. The following target states are possible:

- 1: INIT
- 2: PREOPERATIONAL
- 4: SAFEOPERATIONAL
- 8: OPERATIONAL

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

ECAT_START_COM

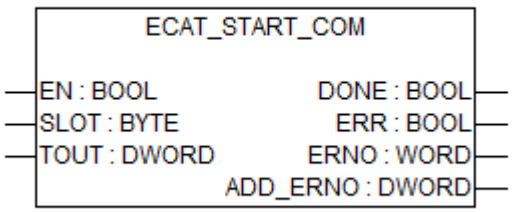
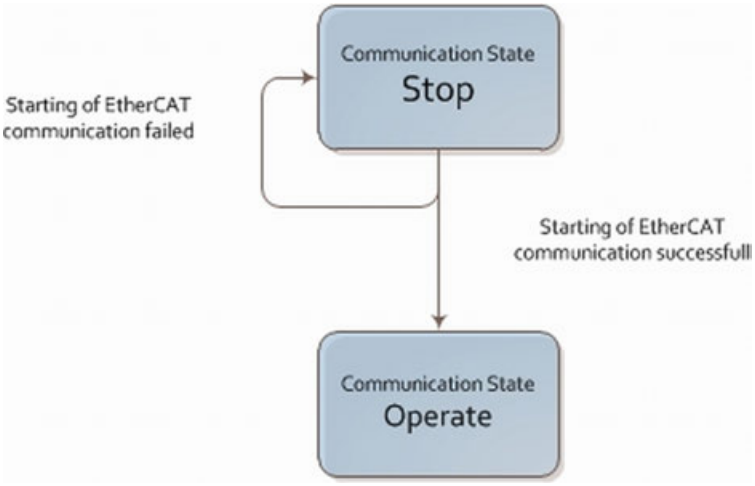


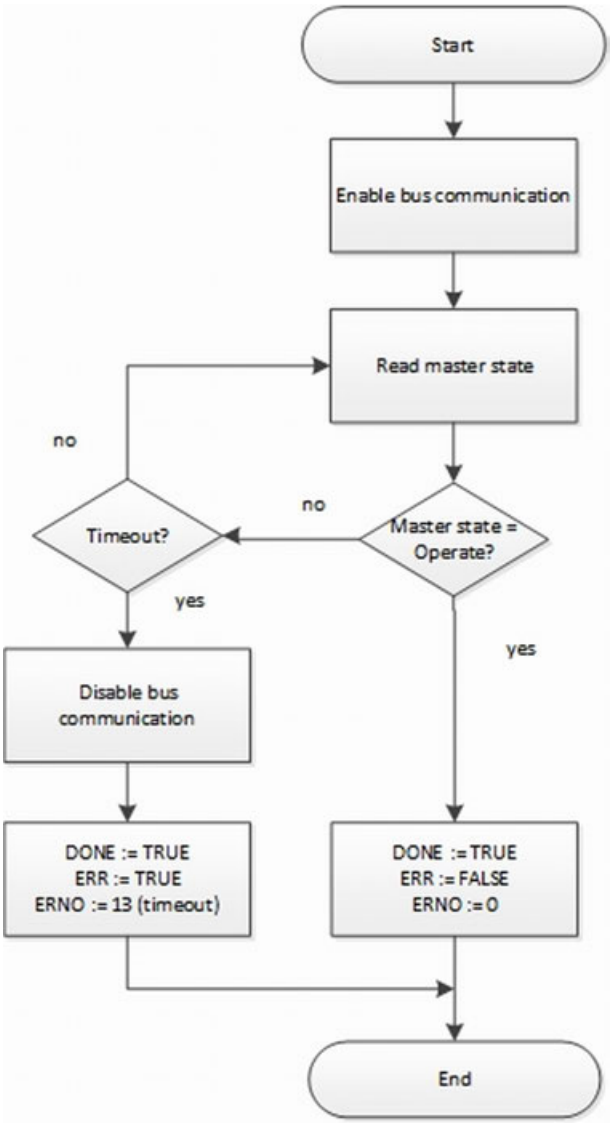
Table 81: General information

Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

ECAT_START_COM can be used to start communication of the EtherCAT master. That means the communication state of the master is changed from Stop to Operate. If execution of the function block fails the master stays in the communication state Stop.



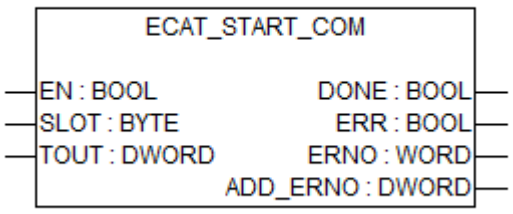
The following figure demonstrates execution of ECAT_START_COM:



With execution start the bus communication is enabled. Communication state of the Master is checked and it’s waited until the communication state is 'Operate'.

Execution of the function block was successful if the master was changed to the communication state 'Operate'. If the master is not in this state within the specified timeout, execution of the function block is failed and the bus communication is disabled again.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

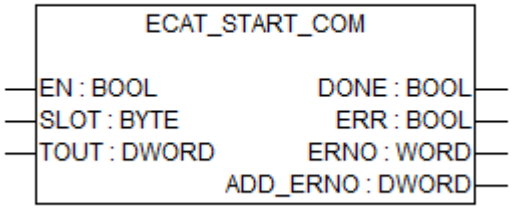
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

ECAT_STOP_COM

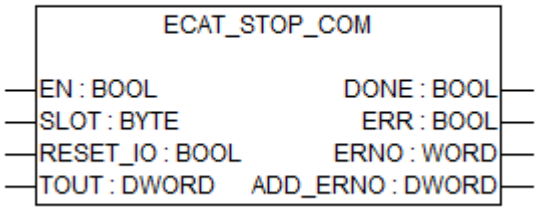
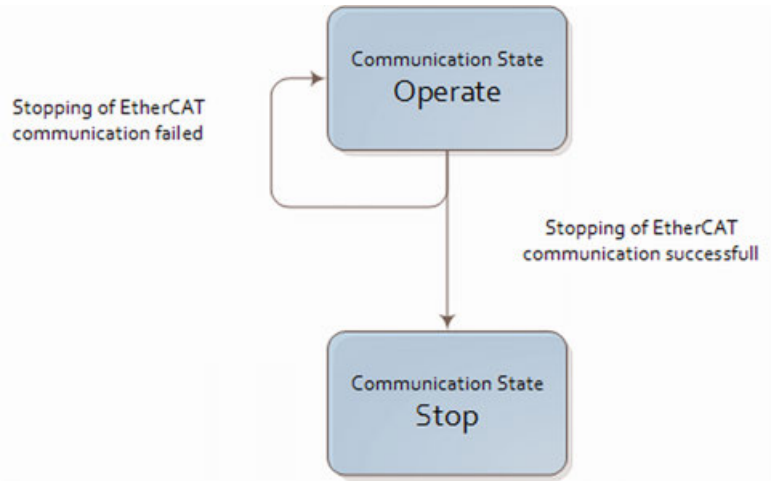


Table 82: General information

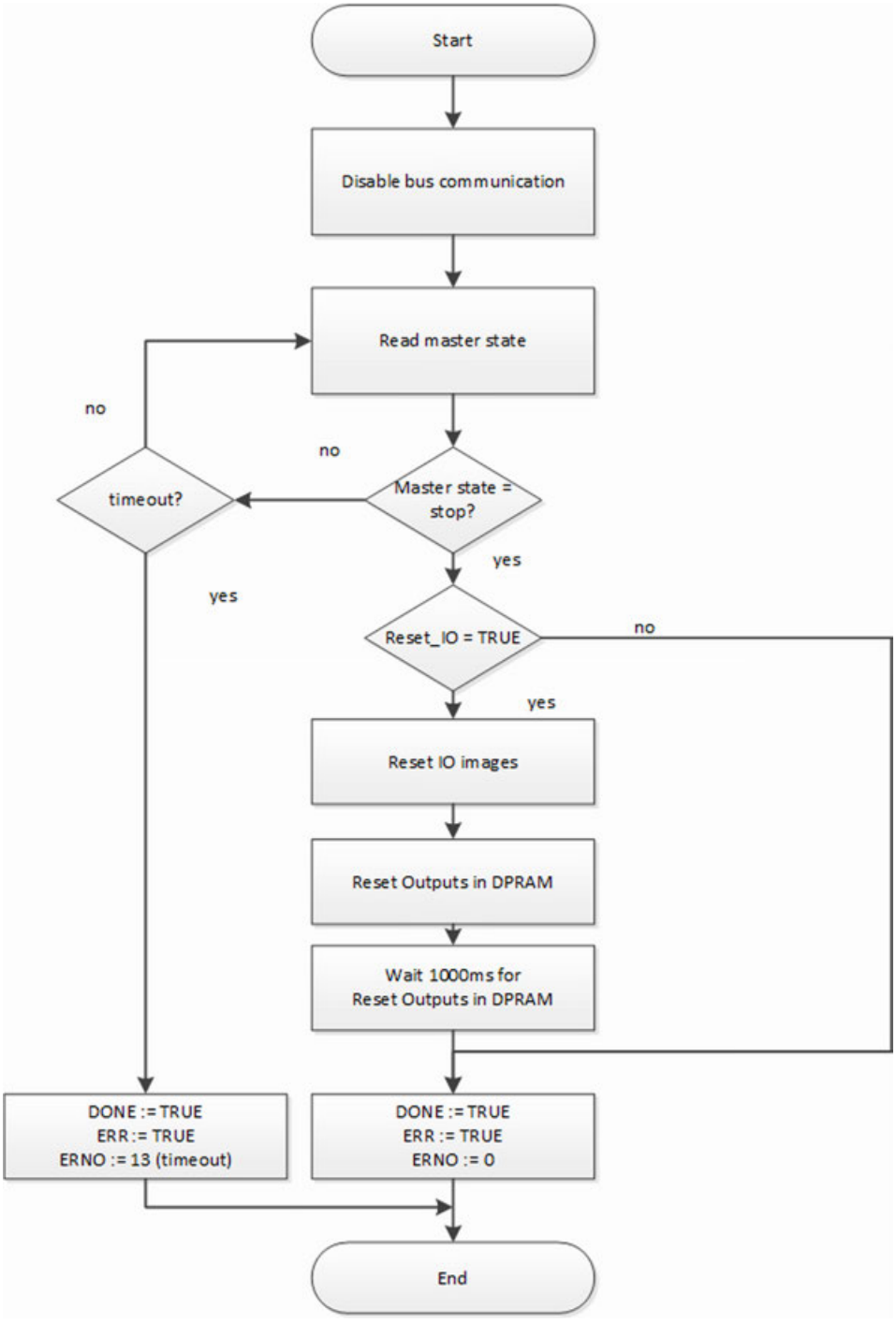
Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

ECAT_STOP_COM can be used to start communication of the EtherCAT master. That means the communication state of the master is changed from 'Operate' to 'Stop'. If execution of the function block fails the master stays in the communication state 'Operate'.



The user is responsible for resetting EtherCAT related part of the PLC application when the bus communication is restarted. At least the internal variables of the sync task have to be reset.

The following figure demonstrates execution of ECAT_STOP_COM:

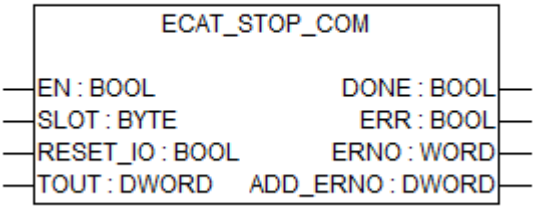


With execution start the bus communication is disabled. Communication state of the Master is checked and it's waited until the communication state is 'Stop'. If the master is in communication state 'Stop' and the input RESET_IO is TRUE the IOs will be reset.

Execution of the function block was successful if the master was changed to the communication state 'Stop' and no error occurred during IO reset. If the master is not in the communication state 'Stop' within the specified timeout, the execution of the function block is failed.

> *The IO reset function of ECAT_STOP_COM is not synchronized. That means there is no notification when the reset is finished. ECAT_STOP_COM has an internal waiting time of 1000 ms to ensure the IOs have been reset. If this time is not sufficient it has to be waited an additional time before the communication is re-started with the FB ECAT_START_COM.*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

RESET_IO

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

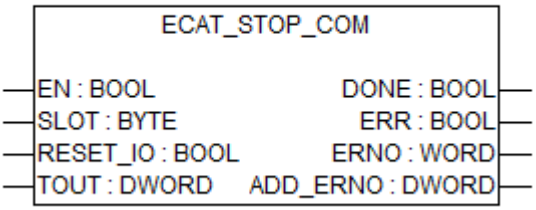
IO values will be reset if set to true.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

ECAT_SOE_READ

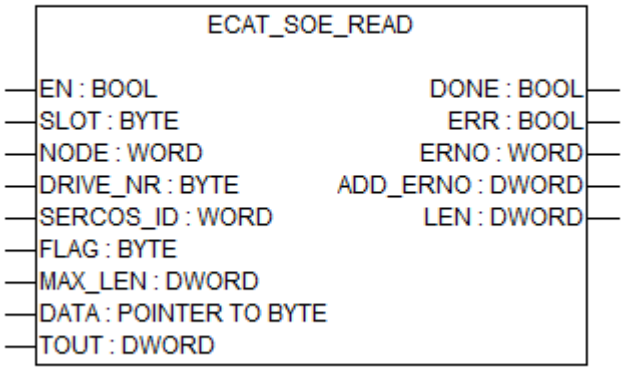
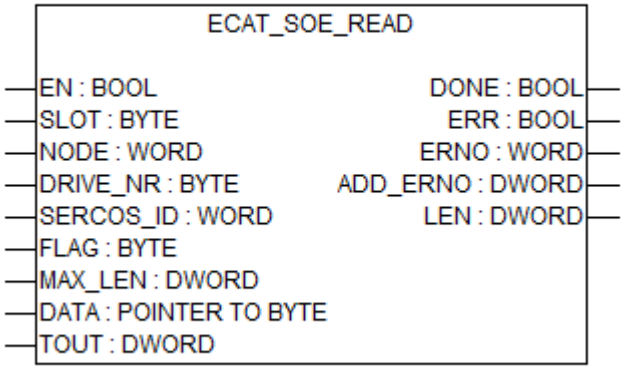


Table 83: General information

Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

ECAT_SOE_READ can be used to read SERCOS II objects.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

DRIVE_NR

Data type	Default value	Range	Unit
BYTE	0	-	-

Drive number, starting with 0.

SERCOS_ID

Data type	Default value	Range	Unit
WORD	0	-	-

SERCOS identification number of the object to be read.

FLAG

Data type	Default value	Range	Unit
BYTE	0	-	-

The following values are allowed:

- 0x01: SOE_ELEMENT_FLAGS_DATASTATE
- 0x02: SOE_ELEMENT_FLAGS_NAME
- 0x04: SOE_ELEMENT_FLAGS_ATTRIBUTE
- 0x08: SOE_ELEMENT_FLAGS_UNIT
- 0x10: SOE_ELEMENT_FLAGS_MIN
- 0x20: SOE_ELEMENT_FLAGS_MAX
- 0x40: SOE_ELEMENT_FLAGS_VALUE

MAX_LEN

Data type	Default value	Range	Unit
DWORD	0	-	-

Maximal length of the data array.

DATA

Data type	Default value	Range	Unit
POINTER TO ARRAY OF BYTE	-	-	-

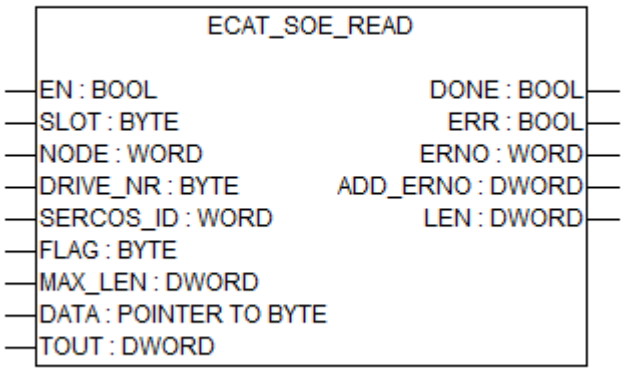
Pointer to the receive Buffer.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

LEN

Data type	Default value	Range	Unit
DWORD	0	-	-

Length of the received data in bytes.

ECAT_SOE_WRITE

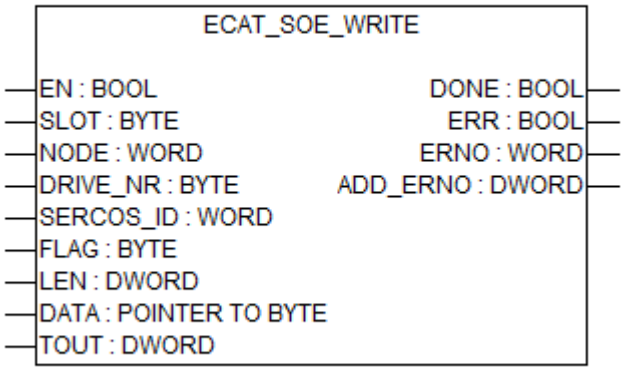


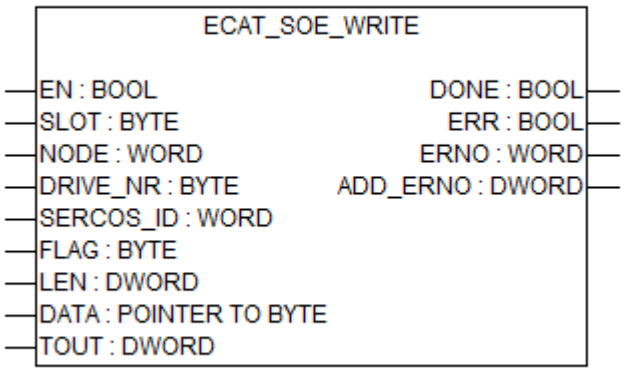
Table 84: General information

Available as of runtime system	V2.4.3 and above
Available as of AB version	V1.2 and above
Available as of CM579-EthCAT version	V4.0.0 and above
Included in library	EtherCAT_EXT_AC500_V25.lib
Type	Function block with historical values.

The function block ECAT_SOE_WRITE can be used to write SERCOS II – objects.

It depends on the specific EtherCAT device which objects or attributes of objects are writeable.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
DWORD	0	1 ... 4294967295	-

NODE represents the EtherCAT address of the slave.

DRIVE_NR

Data type	Default value	Range	Unit
BYTE	0	-	-

Drive number, starting with 0.

SERCOS_ID

Data type	Default value	Range	Unit
WORD	0	-	-

SERCOS identification number of the object to be write.

FLAG

Data type	Default value	Range	Unit
BYTE	0	-	-

The following values are allowed:

- 0x01: SOE_ELEMENT_FLAGS_DATASTATE
- 0x02: SOE_ELEMENT_FLAGS_NAME
- 0x04: SOE_ELEMENT_FLAGS_ATTRIBUTE
- 0x08: SOE_ELEMENT_FLAGS_UNIT
- 0x10: SOE_ELEMENT_FLAGS_MIN
- 0x20: SOE_ELEMENT_FLAGS_MAX
- 0x40: SOE_ELEMENT_FLAGS_VALUE

LEN

Data type	Default value	Range	Unit
DWORD	0	0…1024	-

Length of the data array.

DATA

Data type	Default value	Range	Unit
POINTER TO ARRAY OF BYTE	-	-	-

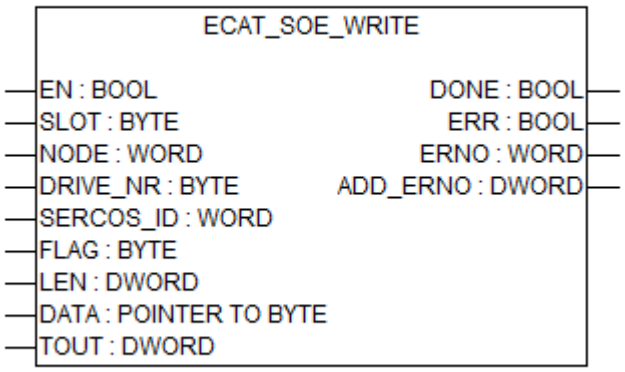
Pointer to the receive Buffer.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

1.5.4.15.2 Global variables

Communication states

Table 85: Communication states of the EtherCAT master

ECAT_MASTER_COM_STATE_UNKNOWN:	DWORD:=16#00000000;	Communication State "UNKNOWN" of the EtherCAT master
ECAT_MASTER_COM_STATE_NOT_CONFIGURED:	DWORD:=16#00000001;	Communication State "NOT CONFIGURED" of the EtherCAT master
ECAT_MASTER_COM_STATE_STOP:	DWORD:=16#00000002;	Communication State "STOP" of the EtherCAT master
ECAT_MASTER_COM_STATE_IDLE:	DWORD:=16#00000003;	Communication State "IDLE" of the EtherCAT master
ECAT_MASTER_COM_STATE_OPERATE:	DWORD:=16#00000004;	Communication State "OPERATE" of the EtherCAT master

Flags for SERCOS

Table 86: Flags for the attribute selection of the SERCOS objects

ECAT_SOE_ELEMENT_FLAGS_DATASTATE:	BYTE := 16#01;	State of the selected Sercos ID
ECAT_SOE_ELEMENT_FLAGS_NAME:	BYTE := 16#02;	Name of the selected Sercos ID
ECAT_SOE_ELEMENT_FLAGS_ATTRIBUTE:	BYTE := 16#04;	Attribute of the selected Sercos ID
ECAT_SOE_ELEMENT_FLAGS_UNIT:	BYTE := 16#08;	Unit of the selected Sercos ID

ECAT_SOE_ELE-MENT_FLAGS_MIN:	BYTE := 16#10;	Minimum value of the selected Sercos ID
ECAT_SOE_ELE-MENT_FLAGS_MAX:	BYTE := 16#20;	Maximum value of the selected Sercos ID
ECAT_SOE_ELE-MENT_FLAGS_VALUE:	BYTE := 16#40;	Current value of the selected Sercos ID

EtherCAT state machine

Table 87: State values

ECAT_STATE_MACHINE_STATE_INIT:	BYTE := 16#01;	INIT state in the EtherCAT state machine
ECAT_STATE_MACHINE_STATE_PREOP:	BYTE := 16#02;	PREOPERATIONAL state in the EtherCAT state machine
ECAT_STATE_MACHINE_STATE_SAFEOP:	BYTE := 16#04;	SAFEOPERATIONAL state in the EtherCAT state machine
ECAT_STATE_MACHINE_STATE_OP:	BYTE := 16#08;	OPERATIONAL state in the EtherCAT state machine

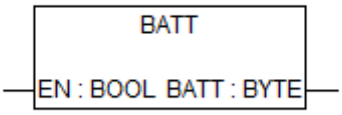
1.5.4.16 External System library

Library file name: **SysExt_AC500_Vx.lib**

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

1.5.4.16.1 Function blocks

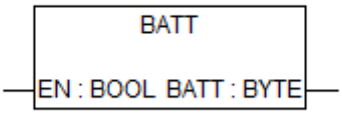
BATT



Parameter	Value
Included in library	SysExt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function
Group	Battery

Using the function block BATT the charge state of the battery can be requested.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the function block is active, the charge state of the battery is available at the function block output.

Output description

(Output)

Data type	Default value	Range	Unit
BYTE	-	-	-

The output of the function block BATT outputs the charge state of the battery. The following values are possible at present:

0: Battery empty

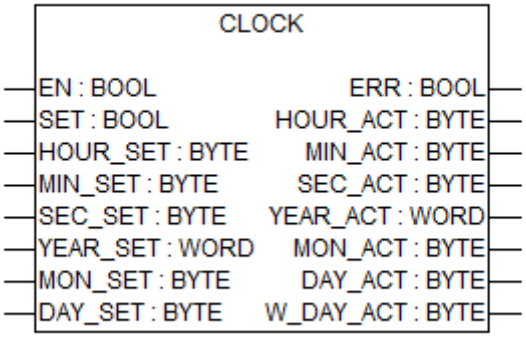
20: Remaining battery charge below 20 %

100: Battery charge OK

Function call in ST

```
BATT_LOAD := BATT(BATT_EN);
```

CLOCK

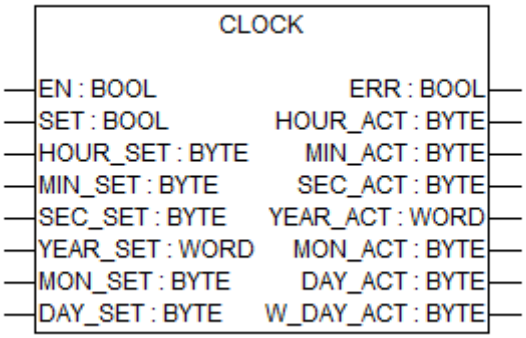


Parameter	Value
Included in library	SysExt_AC500_V10.lib
Available as of firmware	V1.0

Parameter	Value
Type	Function block with historical values
Group	Real-time clock

This function block allows to set and display the current time and date. The clock is set by means of the set inputs for the time and date. The values available at the set inputs are read in with the occurrence of a FALSE/TRUE edge at input SET. As long as a TRUE signal is present at the EN input, the current date and time are displayed at the function block outputs.

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the function block is active, the current time and date are available at the outputs.

SET

Data type	Default value	Range	Unit
BOOL	-	-	-

With an occurring FALSE/TRUE edge at input SET, the clock is set to the values available at the time and date inputs.

HOUR_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for the hours. The clock works in 24 hours mode, i.e. it changes from 23:59:59 h to 0:0:0 h.

Valid range of values: 0...23.

MIN_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for the minutes.

Valid range of values: 0...59.

SEC_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for the seconds.

Valid range of values: 0...59.

YEAR_SET

Data type	Default value	Range	Unit
WORD	-	-	-

Set input for the year. Four-digit input, e.g. 2005.

MON_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for the month.

Valid range of values: 1...12.

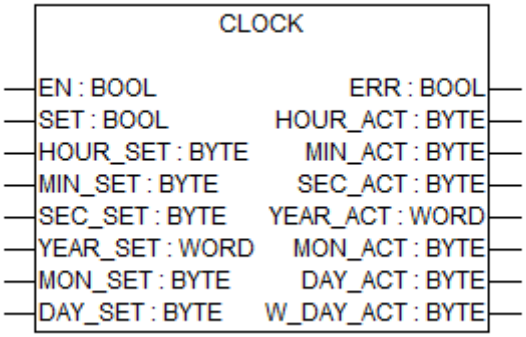
DAY_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for the day (which day of the month).

Valid range of values: 1...31.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error occurred during Function Block processing.

HOUR_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the hours.

MIN_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the minutes.

SEC_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the seconds.

YEAR_ACT

Data type	Default value	Range	Unit
WORD	-	-	-

Current value for the year.

MON_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the month.

DAY_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the day.

W_DAY_ACT

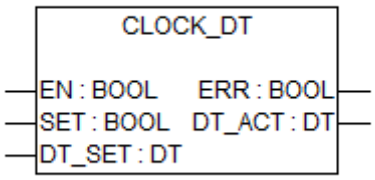
Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for the number of the day of week. 0 = Sunday, 1 = Monday, .., 6 = Saturday.

Function call in ST

```
Clock  (EN := Clock_EN,
       SET := Clock_SET,
       HOUR_SET := Clock_HOUR_SET,
       MIN_SET := Clock_MIN_SET,
       SEC_SET := Clock_SEC_SET,
       YEAR_SET := Clock_YEAR_SET,
       MON_SET := Clock_MON_SET,
       DAY_SET := Clock_DAY_SET);


Clock_ERR := Clock.ERR;
Clock_HOUR_ACT := Clock.HOUR_ACT;
Clock_MIN_ACT := Clock.MIN_ACT;
Clock_SEC_ACT := Clock.SEC_ACT;
Clock_YEAR_ACT := Clock.YEAR_ACT;
Clock_MON_ACT := Clock.MON_ACT;
Clock_DAY_ACT := Clock.DAY_ACT;
Clock_W_DAY_ACT := Clock.W_DAY_ACT;
```

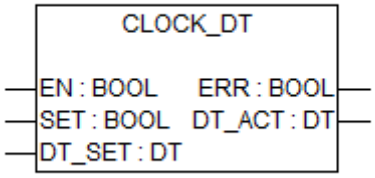
CLOCK_DT



Parameter	Value
Included in library	SysExt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	Real-time clock

This function block allows to set and display the current time and date in the standardized "DT" format. Setting of the clock is performed via set input "DT_SET". The value available at the set input is read in with the occurrence of a FALSE/TRUE edge at input SET. As long as a TRUE signal is present at the EN input, the current time and date are displayed in the standardized "DT" format at function block output "DT_ACT".

Input description



EN

Data type	Default value	Range	Unit
BOOL	-	-	-

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the function block is active, the current time and date are available at the outputs.

SET

Data type	Default value	Range	Unit
BOOL	-	-	-

With an occurring FALSE/TRUE edge at input SET, the clock is set to the values available at the time and date inputs.

DT_SET

Data type	Default value	Range	Unit
BYTE	-	-	-

Set input for time and date in standardized "DT" format. The input for a "DT" value always has to start with the preceding designation "DT#", followed by the date, a hyphen and the time.

Valid range of values: DT#1970-01-01-00:00:00 to DT#2106-02-06-06:28:15

Output description

ERR

Data type	Default value	Range	Unit
BOOL	-	-	-

Output ERR indicates whether an error occurred during Function Block processing.

DT_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

Current value for date and time in standardized "DT" format.

Function call in ST

```
Clock  (EN := ClockDt_EN,
       SET := ClockDt_SET,
       DT_SET := ClockDt_DT_SET);


ClockDt_ERR := ClockDt.ERR;
ClockDt_DT_ACT := ClockDt.DT_ACT;
```

1.5.4.17 FlexConf library

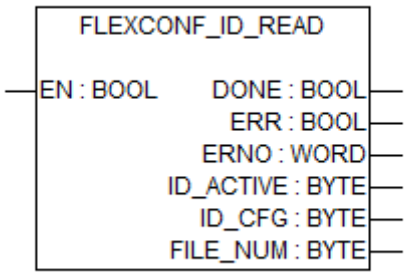
Library file name: **FlexConf_AC500_Vx.lib**

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

Only one instance of the function blocks FLEXCONF_ID_READ and FLEXCONF_ID_WRITE are allowed.

1.5.4.17.1 Function blocks

FLEXCONF_ID_READ

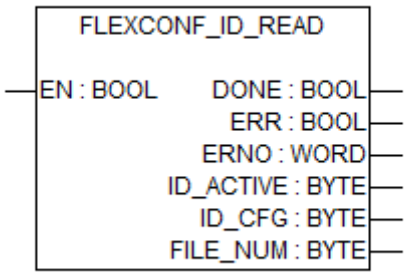


Parameter	Value
Included in library	FlexConf_AC500_V24.lib
Available as of firmware	V2.4
Type	Function block with historical values
Group	-

The function block FLEXCONF_ID_READ is used for reading the current settings of a flexible configuration. The following parameters can be read:

- Current value of the FlexConfID
- Value of the FlexConfID in the configuration data
- Numbers of alternative configuration files of the currently loaded project

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

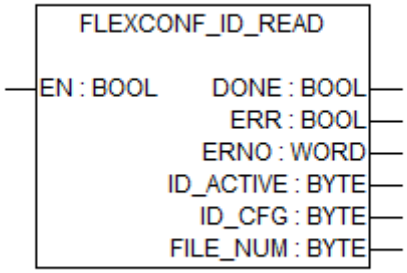
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ID_ACTIVE

Data type	Default value	Range	Unit
BYTE	-	-	-

Output ID_ACTIVE returns the current value of the FlexConfID which has been determined during booting of the PLC by reading from the configuration data file in the PLC's flash memory. This value has been used to determine the configuration file of the current executed application.

ID_CFG

Data type	Default value	Range	Unit
BYTE	-	-	-

Output ID_CFG returns the value of the FlexConfID which is stored in the configuration data file in the PLC's flash memory. This value is applied when the PLC is rebooted.

FILE_NUM

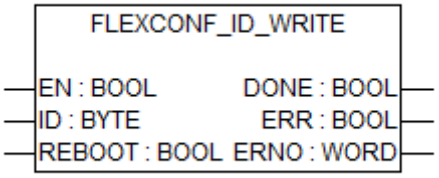
Data type	Default value	Range	Unit
BYTE	-	-	-

Ouput FILE_NUM returns the number of alternative configuration files of the currently executed application.

Function call in ST

```
FLEXCONF_ID_READ (EN := FLEXCONF_ID_READ_EN);


FLEXCONF_ID_READ_DONE         := FLEXCONF_ID_READ.DONE;
FLEXCONF_ID_READ_ERR          := FLEXCONF_ID_READ.ERR;
FLEXCONF_ID_READ_ERNO         := FLEXCONF_ID_READ.ERNO;
FLEXCONF_ID_READ_ID_ACTIVE    := FLEXCONF_ID_READ.ID_ACTIVE;
FLEXCONF_ID_READ_ID_CFG       := FLEXCONF_ID_READ.ID_CFG;
FLEXCONF_ID_READ_ID_FILE_NUM := FLEXCONF_ID_READ.FILE_NUM;
```

FLEXCONF_ID_WRITE

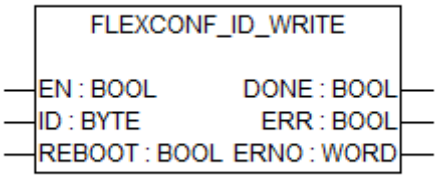


Parameter	Value
Included in library	FlexConf_AC500_V24.lib
Available as of firmware	V2.4
Type	Function block with historical values
Group	-

The function block FLEXCONF_ID_WRITE is used for writing the value of the FlexConfID to the configuration data file in the PLC's flash memory. Optionally the PLC can be rebooted after the value of the FlexConfID has been written.

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN. If the function block is active, the current values are available at the outputs.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ID

Data type	Default value	Range	Unit
BYTE	-	-	-

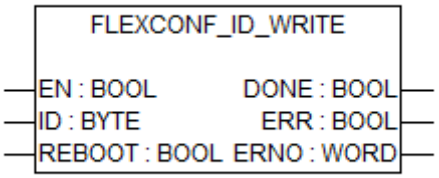
At input ID, the value of the FlexConfID which should be written to the configuration data has to be has to be specified.

REBOOT

Data type	Default value	Range	Unit
BOOL	-	-	-

If this input is set to TRUE, the PLC will be rebooted after the FlexConfID has been written. If this input is set to FALSE, the PLC will not be rebooted. In this case, the currently configuration stays active until the CPU will be rebooted manually.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
FLEXCONF_ID_WRITE (EN     := FLEXCONF_ID_WRITE_EN,
                   ID     := FLEXCONF_ID_WRITE_ID,
                   REBOOT := FLEXCONF_ID_WRITE_REBOOT);


FLEXCONF_ID_WRITE_DONE := FLEXCONF_ID_WRITE.DONE;
FLEXCONF_ID_WRITE_ERR  := FLEXCONF_ID_WRITE.ERR;
FLEXCONF_ID_WRITE_ERNO := FLEXCONF_ID_WRITE.ERNO;
```

1.5.4.18 IEC60870 library

Library file name: **IEC60870_AC500_Vx.lib**

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The library can only be executed with CPUs with onboard Ethernet and at least CPU firmware V_2_0_0.

1.5.4.18.1 Information

The function blocks of the IEC60870 library allow link-ups to be made between AC500 CPUs with onboard Ethernet and external systems. The link-up takes place via the internal Ethernet interface of the CPU. The telecontrol protocol according to IEC60870-5 is used.

The CPU can work as both control station and substation. In control direction setpoints and commands are set. In monitoring direction the control station sends status values, real values and discrete values to the substation. Via general inquiry, the substation requests the control station to send all status values, real values and discrete values. Otherwise, these values are sent by the control station on a change-driven basis, cyclically or when triggered by an application. Status values, real values and discrete values may contain timestamps. These are filled in with the time of the process station when sent. The CPU can time-synchronize the telecontrol link.

A module accepts the configuration of the physical interface (link layer) and the general protocol parts (application layer), see PLC configuration in Control Builder Plus help.

Send and receive blocks are available for data exchange. These blocks exist for the IEC60870-5 data types

- setpoint value
- command value
- double command value
- status value
- double status value
- real value and discrete value.

The inputs/outputs of the send and receive blocks are combined with the signals to be communicated.

Data flow control

Each send or receive block can only process one data message. Ideally, new data are available at each user task run-through or new data can be sent.

If the OV output (send block only) indicates TRUE, the block computes more quickly than the data can be sent. If the receive block is not computed quickly enough and has thus not collected all the data.

Alternatively, this block sends either cyclically or when the input value is changed. Ideally, the topical data can be sent via the telecontrol link in connection with every user task run-through.

Data integrity of the IEC60870-5 protocol

With the IEC60870-5 protocol, a distinction is made between data transmission in the monitoring direction (status values, real values, discrete values) and in the control direction (commands and setpoints).

All data transmissions are acknowledged from the link communication level by the receiver. This acknowledgement is not sent to the sender of the data in every telecontrol link.

For data transmission in the control direction, additional acknowledgement (e.g. actterm) is possible. These acknowledgements are not sent by every telecontrol link either. For safe data transmission, it is necessary, in such cases, to configure data readback. The receiver then sends the data received back to the sender via the corresponding send blocks.

Information in the monitoring direction is acknowledged by the receiver on the lowest communication level (link level) when received. This acknowledgement is generated by the telecontrol head itself with some telecontrol heads. In the event of overload/overrun, a data message may be lost. For data in the control direction, so-called actterm acknowledgement can be used. This additional acknowledgement is sent back to the sender when the data have been executed in the process. If data are to be sent in the monitoring direction with guaranteed transmission, it is necessary to read back the sent value via another variable and, after observing a monitoring time, resend in the event of an error.

Data transmission function blocks

Send blocks

On the basis of the communication protocol, it is sensible to restrict the data types at one send block to one type. Therefore, there are five types of send blocks:

- send of status values
- commands
- real values
- setpoints
- discrete values.

These types are mapped to the IEC1131 data types BOOL, REAL and DINT.

Operating modes of the send blocks

The send blocks know three operating modes to send their data

- caused by request pin (SEND),
- send in connection with a change of data (AUTO), or
- cyclic send of data (CYCLE).

Send via request pin

The SEND signal is evaluated on the rising edge. The RDY signal remains applied for one computation cycle. If a rising edge is generated again at the SEND signal although no acknowledgement has yet been received from the receiver, the OV pin is set in order to indicate that an overrun has taken place. The evaluation of the receive acknowledgement is carried out before the evaluation of whether transmission is to take place. This means, assuming that there is an appropriately fast telecontrol link, that in connection with change-driven and cyclic transmission, a transmission job can be sent in connection with every computation of the block. In connection with send via the request pin it is possible to send only in connection with every second computation (send takes place only with a rising edge).

Change-driven send of data

Data are always sent when the value of the input variables changes. When changes take place, there is an internal simulation that the SEND pin changed from 0 to 1.

In order to prevent unnecessarily frequent send in the event of mild fluctuations in the input value, a threshold value can be configured for real values and setpoints. The input value is not sent until it differs positively or negatively from the value last sent by more than the threshold value.

If the input value changes again although no acknowledgement has yet been received from the receiver, the OV pin is set in exactly the same way as in connection with send via the Request pin. If an error occurs during send, the job is automatically retried until the value has been sent without error.

Cyclic send

The data are automatically sent after the expiry of a configurable cycle time (SCANDOWN). This cycle time is indicated in multiples of the task cycle time in which the block is computed. In this operating mode, an overrun error can occur if the transmission is faster than the response time of the receiver. For setpoints, it is necessary to ensure that an acknowledgement is generated by the receiver which is not sent until the setpoint is accepted. The send block is not ready for transmission again until after this acknowledgement has been received.

Timestamps

Timestamps for data types with time associated to the data are generated when a data point is being given to the underlying protocol. So for example on the rising edge of the SEND Pin. For values where an explicit timestamp different from the actual time is required, a new group of function blocks has been created all ending with the extension _ET.

All timestamps are only send at all if the correct IEC60870 data type with timestamps has been configured.

Receive blocks

In receive direction, the jobs enter the device module via the interface. The device module selects the correct receive block using the telecontrol address. To this end, during installation the receive blocks pass their parameterised telecontrol addresses to the device module, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

The device module stores the data received and the receive blocks make the data available at their output pins in connection with the next computation of the user task.

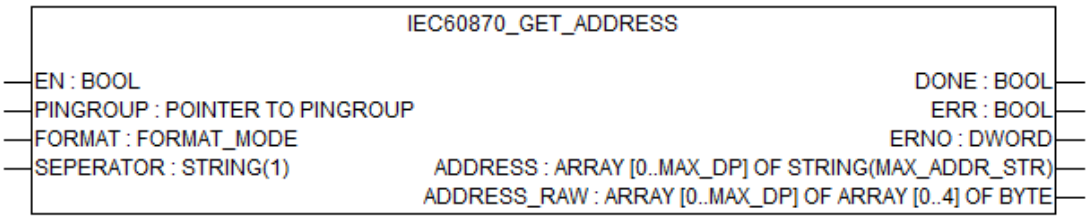
Interrogation command

The AC500 will answer interrogation command only if some general requirements are met:

For the command C_IC_NA_1 (interrogation command) the common address is either the broadcast address 255 255 or the defined station address and the qualifier of interrogation is <20> station interrogation. The answer will encode the defined station address. See ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

1.5.4.18.2 Function blocks

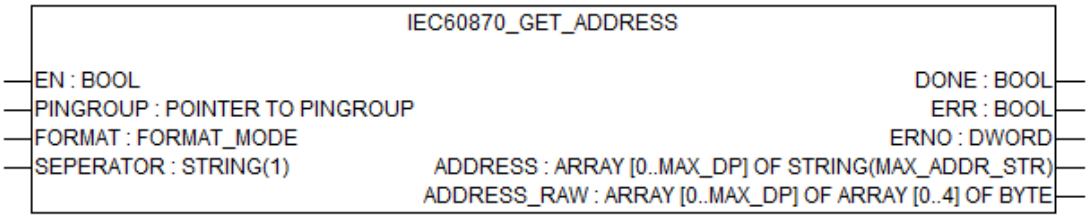
IEC60870_GET_ADDRESS



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.2.x
Type	Function block with historical values
Group	General

The function block returns the address (GADU1, GADU2, IAD1, IAD2, IAD3) of the given PINGROUP in string representation.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

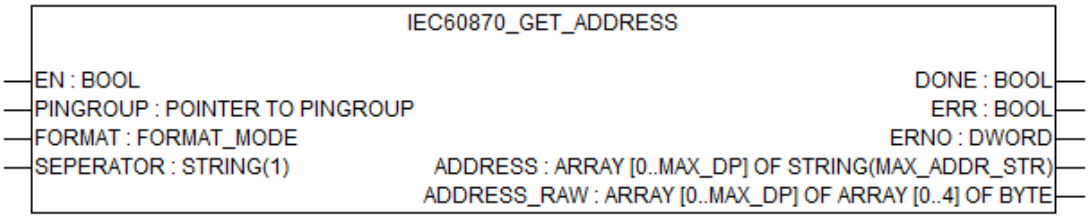
PINGROUP Pointer to a structure representing a pin group. Normally generated by the Control Builder Plus and stored under the global variable list "Command (CONSTANT)".

FORMAT Describes the desired output format of the address. The different formats can be accessed through the FORMAT_MODE (ENUM). Following values are defined:

DEC_ONE:	Decimal value of the 40 bits representing the address (default)
HEX_ASC:	Hexadecimal value of each byte in ascending order: 0xGADU1<SEP>0xGADU2<SEP>0xIAD1<SEP>0xIAD2<SEP>0xIAD3
HEX_DESC:	Hexadecimal value of each byte in descending order: 0xGADU2<SEP>0xGADU1<SEP>0xIAD3<SEP>0xIAD2<SEP>0xIAD1
DEC_ASC:	Decimal value of each byte in ascending order: GADU1<SEP>GADU2<SEP>IAD1<SEP>IAD2<SEP>IAD3
DEC_DESC:	Decimal value of each byte in descending order: GADU2<SEP>GADU1<SEP>IAD3<SEP>IAD2<SEP>IAD1

SEPARATOR One character that is used to divide the single bytes of the address from each other. See the above description of FORMAT where <SEP> represents this one character. Default is a blank.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

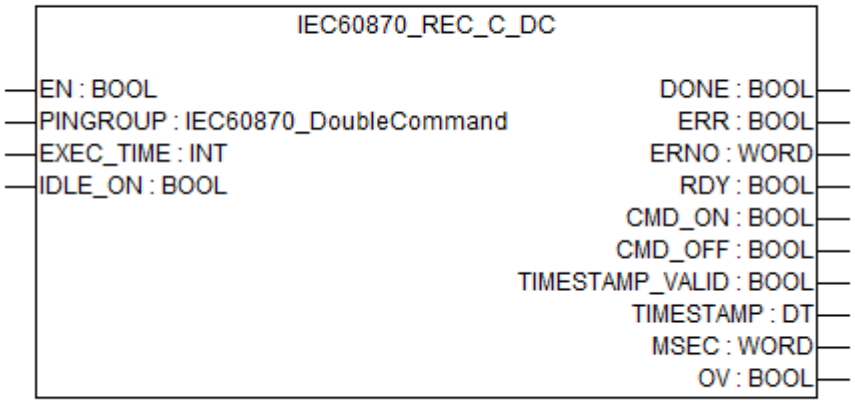
ADDRESS

Contains the string representations of the addresses for the given PINGROUP depending on the selected FORMAT (ARRAY[0..MAX_DP] OF STRING(MAX_ADDR_STR)). Each array index belongs to one data point. If only one data point exists only the first index [0] is set and all the others contain empty strings.

Function call in ST

```
IEC60870_SEND_M_EI_NA_1
                    EN                                          := TRUE,
                    PINGROUP                 := ADR(Measured_value),
                    FORMAT                        := FORMAT_HEX_ASC,
                    SEPARATOR             := ';'
);

GET_ADDR_DONE                    := GET_ADDR.DONE;
GET_ADDR_ERR                        := GET_ADDR.ERR;
GET_ADDR_ERNO                    := GET_ADDR.ERNO;
GET_ADDR_ADDRESS        := GET_ADDR.ADDRESS;
```

IEC60870_REC_C_DC



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

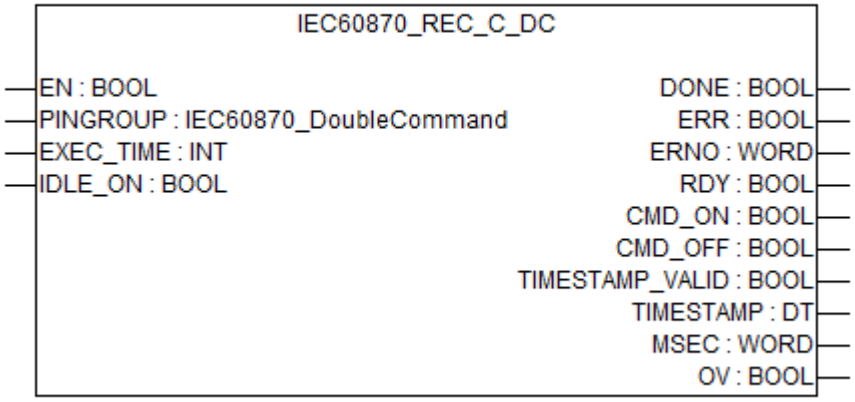
Reception of data messages of the data type according to IEC 60870-5 C_DC_xx message, sent by e.g. SEND_C_DC function block.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

EXEC_TIME INT

Data type	Default value	Range	Unit
INT	-	-	-

Input EXEC_TIME (execution time) sets the execution time in cycles for a command.

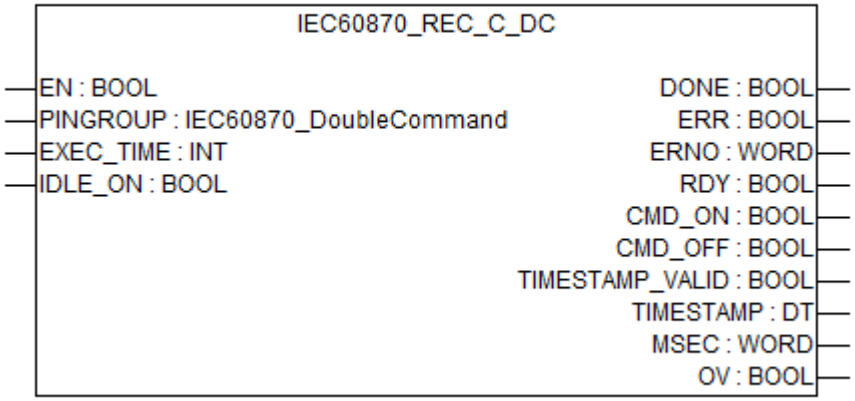
EXEC_TIME = 0: Unlimited

IDLE_ON

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Value for ON; if IDLE_ON => OFF the function block is OFF.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command was received if the output RDY = TRUE.

CMD_ON BOOL (commandon) The received command depending on input IDLE_ON.

The output always has to be considered together with output RDY.

CMD_OFF BOOL (commandoff) The received command depending on input IDLE_ON.

The output always has to be considered together with output RDY.

TIMESTAMP_VALID BOOL (timestampvalid) Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (timestamp) If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds) Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

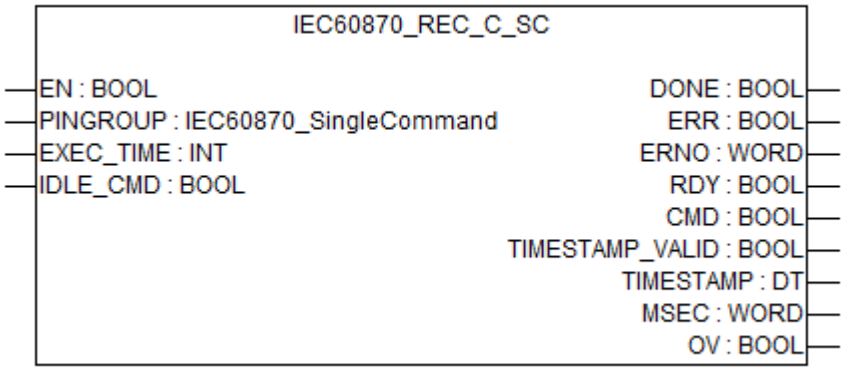
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC   (EN := REC_EN,
PINGROUP := REC_PINGROUP,
EXEC_TIME := REC_EXEC_TIME,
IDLE_ON := REC_IDLE_ON);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
REC_CMD_ON          := REC.CMD_ON;
REC_CMD_OFF         := REC.CMD_OFF;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_OV              := REC.OV;
```

IEC60870_REC_C_SC



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Group/Subgroup

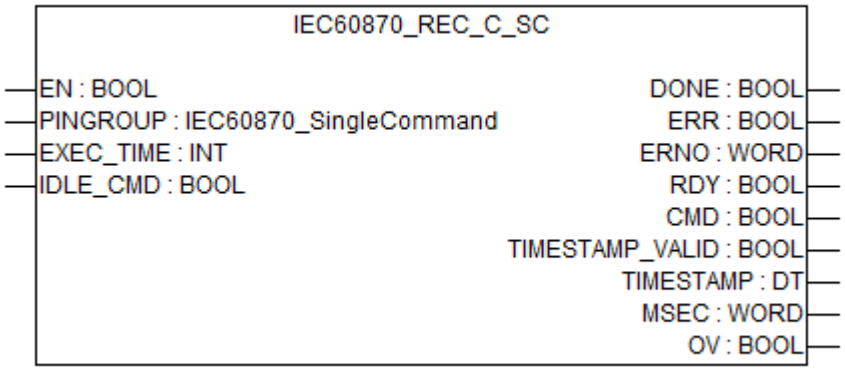
Reception of data messages of the data type according to IEC 60870-5 (C_SC_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SingleCommand (pingroup)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pingroup of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pingroup can only be assigned to one defined function block! It is not possible to use the pingroup twice (e.g. Send and Rec function blocks)!

EXEC_TIME INT (execution time)

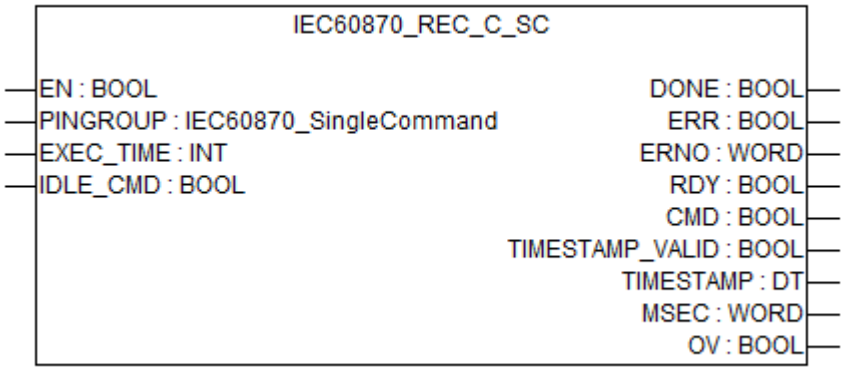
Input EXEC_TIME sets the execution time in cycles for a command.

EXEC_TIME = 0: Unlimited

IDLE_CMD BOOL (idle on)

Input for positive (IDLE_CMD = TRUE) or negative (IDLE_CMD = FALSE) logic.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command was received if the output RDY = TRUE.

CMD BOOL (command) The received command depending on input IDLE_CMD.

The output always has to be considered together with output RDY.

TIMESTAMP_VALID BOOL (timestampvalid) Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (timestamp) If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds) Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

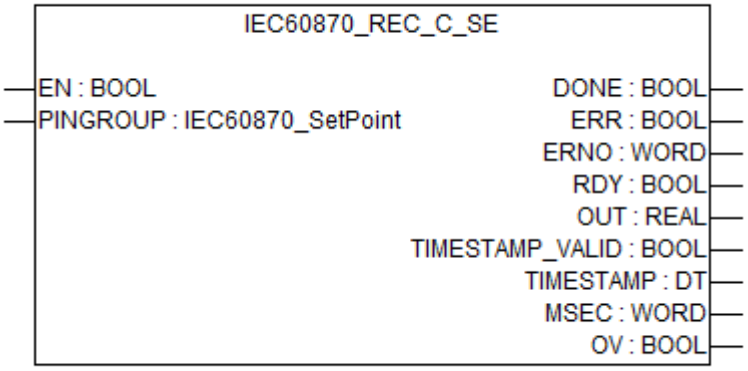
The output always has to be considered together with output RDY.

Function call in ST

```
REC   (EN       := REC_EN,
PINGROUP := REC_PINGROUP,
EXEC_TIME := REC_EXEC_TIME,
IDLE_CMD := REC_IDLE_CMD);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
```

```
REC_CMD             := REC.CMD;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_OV              := REC.OV;
```

IEC60870_REC_C_SE



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

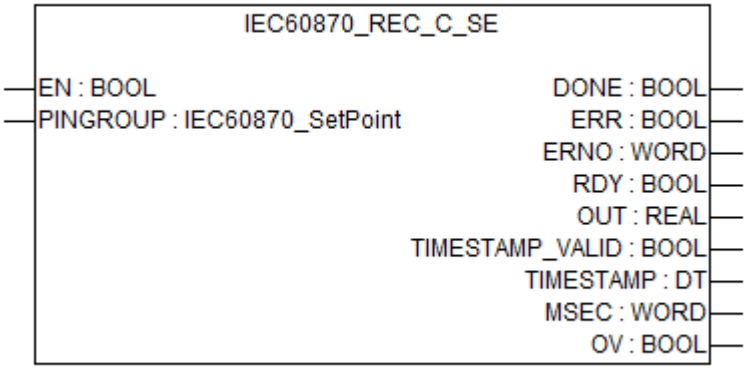
Reception of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SetPoint (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

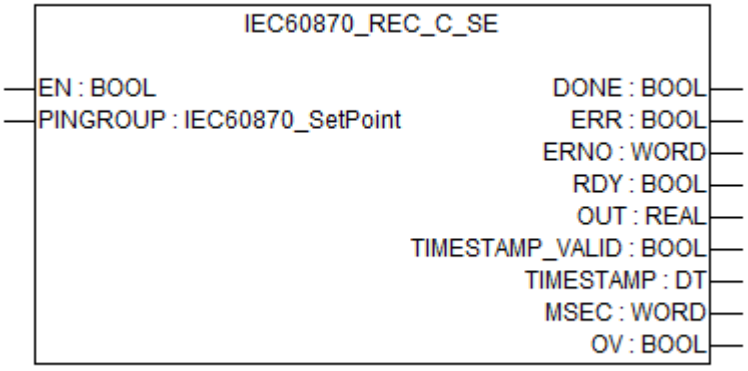
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

OUT REAL (output)

The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid)

Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp)

If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds) Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

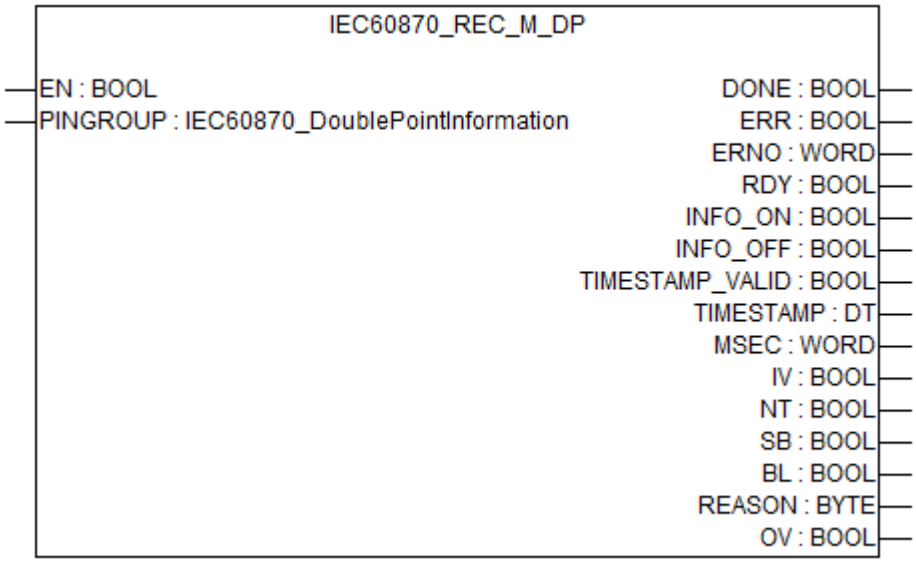
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN       := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
REC_OUT             := REC.OUT;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_OV              := REC.OV;
```

IEC60870_REC_M_DP



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Group/Subgroup

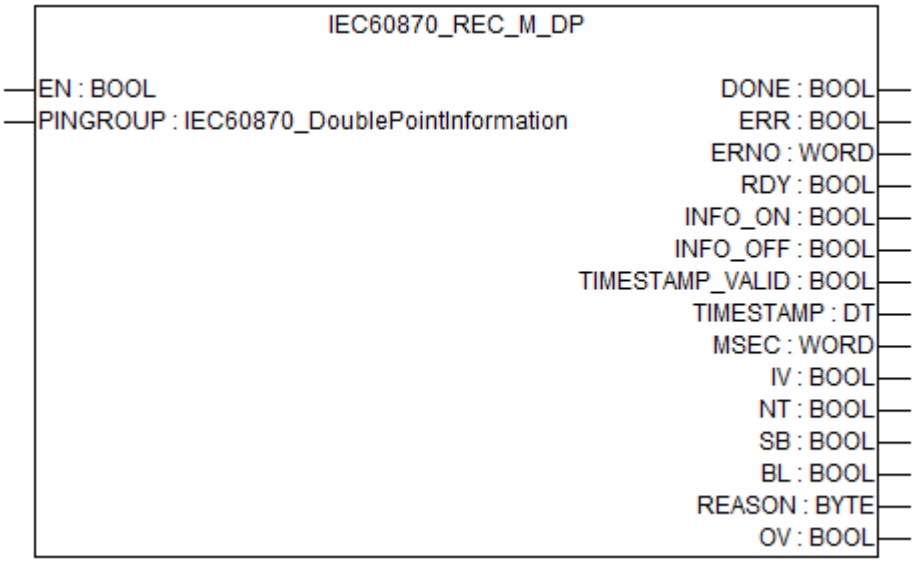
Reception of data messages of the data type according to IEC 60870-5 (M_DP_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_DoublePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

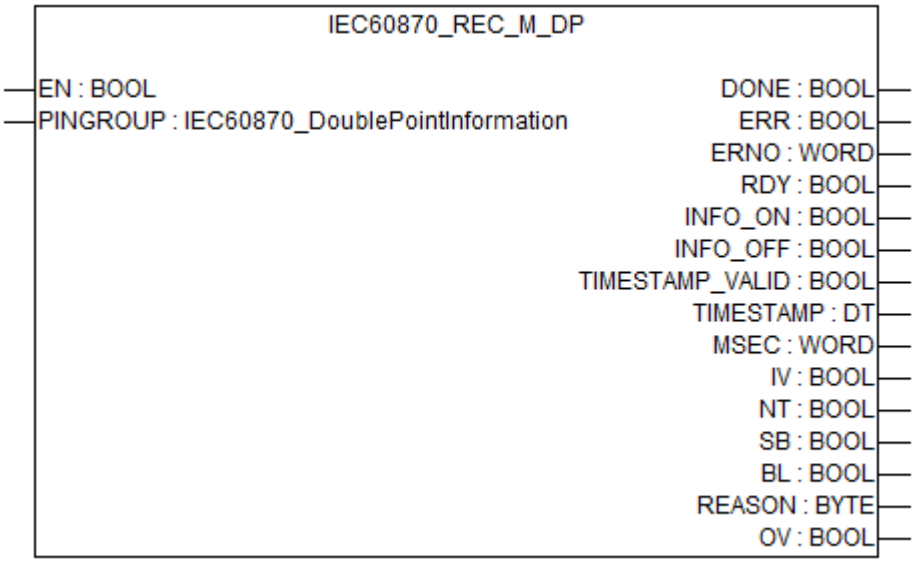
The pin group of the data type IEC 60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

INFO_ON BOOL (infoon)

The received ON information, if output INFO_ON = TRUE.

The output always has to be considered together with output RDY.

INFO_OFF BOOL (infooff)

The received OFF information, if output INFO_OFF = TRUE.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid)

Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp)

If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds)

Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

IV BOOL (invalid)

Output IV displays the status/information of the data. If IV = TRUE, the data is invalid.

The output always has to be considered together with output RDY.

NT BOOL (not topical)

Output NT displays the status/information of the data. If NT = TRUE, the data is not topical/ actual.

The output always has to be considered together with output RDY.

SB BOOL (substituted)

Output SB displays the status/information of the data. If SB = TRUE, the data is substituted.

The output always has to be considered together with output RDY.

BL BOOL (blocked)

Output BL displays the status/information of the data. If BL = TRUE, the data is blocked.

The output always has to be considered together with output RDY.

REASON BYTE (reason)

Output REASON displays the cause of the transmission.

The output always has to be considered together with output RDY.

Reasons for transmission:

1	Periodic/cyclic
2	Background interrogation
3	Spontaneous
4	Initialised
5	Requested
6	Activation
7	Confirmation of activation
8	Deactivation
9	Confirmation of deactivation
10	Termination of activation
11	Return information, caused by a remote command
12	Return information, caused by a local command
20	Interrogated by general inquiry

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

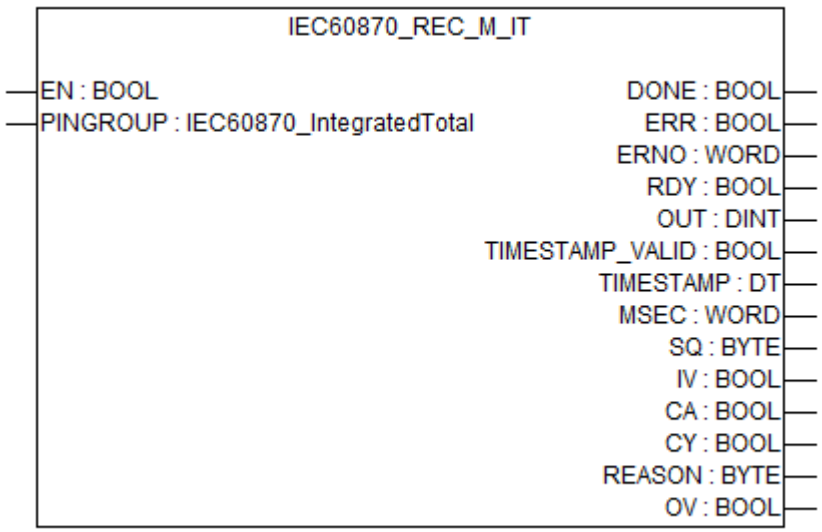
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN        := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE             := REC.DONE;
REC_ERR              := REC.ERR;
REC_ERNO             := REC.ERNO;
REC_RDY              := REC.RDY;
REC_INFO_ON          := REC.INFO_ON;
REC_INFO_OFF         := REC.INFO_FF;
REC_TIMESTAMP_VALID  := REC.TIMESTAMP_VALID;
REC_TIMESTAMP        := REC.TIMESTAMP;
REC_MSEC             := REC.MSEC;
REC_IV               := REC.IV;
REC_NT               := REC.NT;
REC_SB               := REC.SB;
REC_BL               := REC.BL;
REC_REASON           := REC.REASON;
REC_OV               := REC.OV;
```

IEC60870_REC_M_IT



Parameter	Value
Included in library	library.lib IEC60870_AC500_V20.lib
Available as of firmware	Vx.x.x V2.0
Type	Function block with historical values
Group	Data

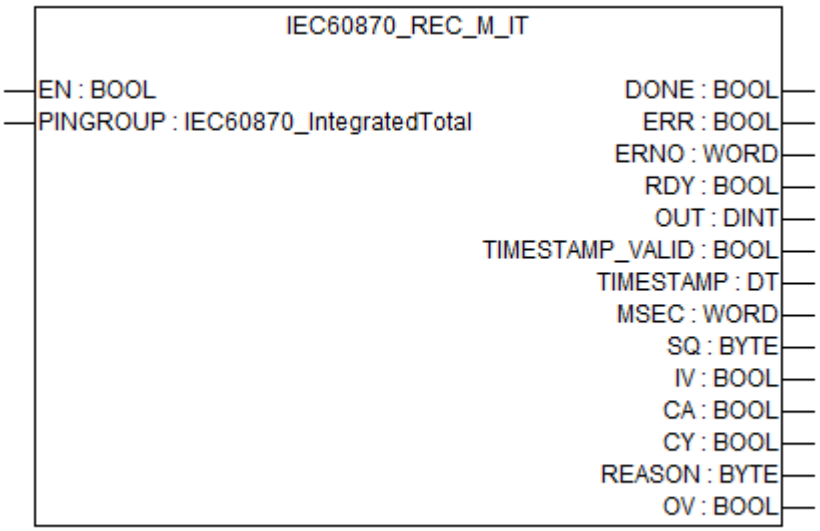
Reception of data messages of the data type according to IEC 60870-5 (M_DP_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

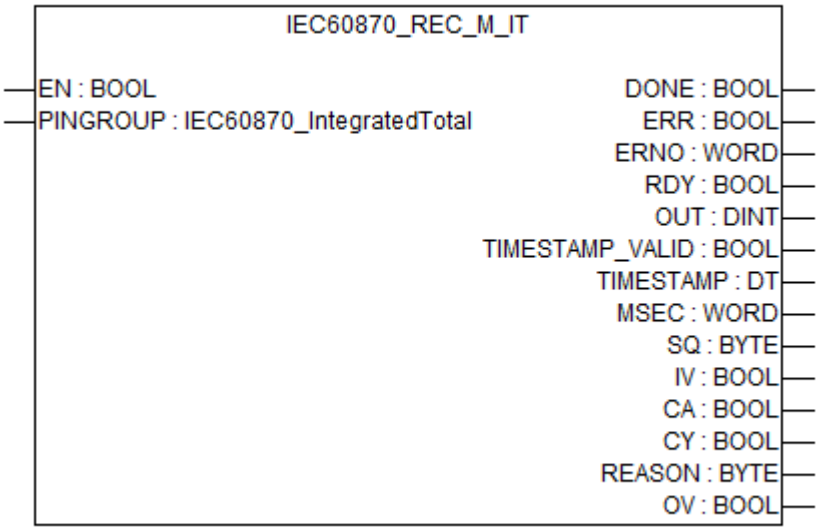
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OUT REAL (output)

The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid)

Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp)

If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds)

Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

SQ BYTE (sequence)

Output SQ displays the sequence number.

The output always has to be considered together with output RDY.

IV BOOL (invalid)

Output IV displays the status/information of the data. If IV = TRUE, the data is invalid.

The output always has to be considered together with output RDY.

CA BOOL (counter adjusted)

Output CA displays the status/information of the data. If CA = TRUE, the counter has been preset since the last reading.

The output always has to be considered together with output RDY.

CY BOOL (carry)

Output CY displays the status/informations of the data. If CY = TRUE, the transmission divided into associated measurement periods.

The output always has to be considered together with output RDY.

REASON BYTE (reason)

Output REASON displays the cause of the transmission.

The output always has to be considered together with output RDY.

Reasons for transmission:

1	Periodic/cyclic
2	Background interrogation
3	Spontaneous
4	Initialised

5	Requested
6	Activation
7	Confirmation of activation
8	Deactivation
9	Confirmation of deactivation
10	Termination of activation
11	Return information, caused by a remote command
12	Return information, caused by a local command
20	Interrogated by general inquiry

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

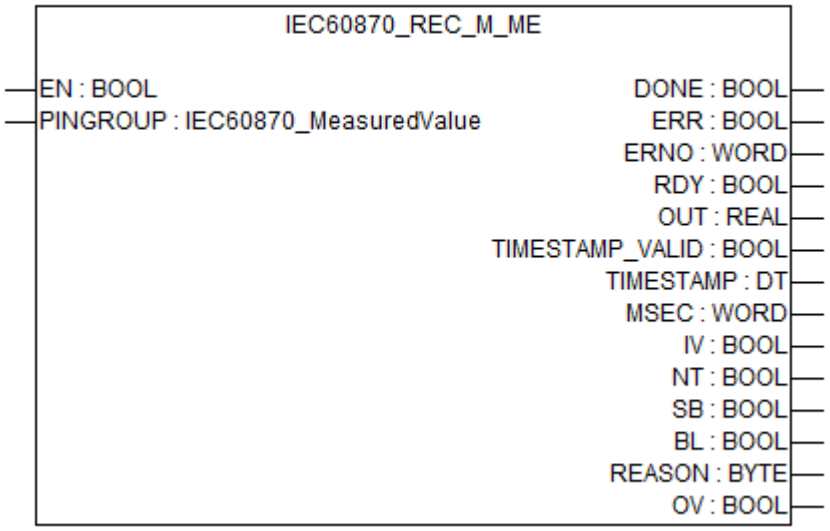
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC   (EN       := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
REC_OUT             := REC.OUT;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_SQ              := REC.SQ;
REC_IV              := REC.IV;
REC_CA              := REC.CA;
REC_CY              := REC.CY;
REC_REASON          := REC.REASON;
REC_OV              := REC.OV;
```

IEC60870_REC_M_ME



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

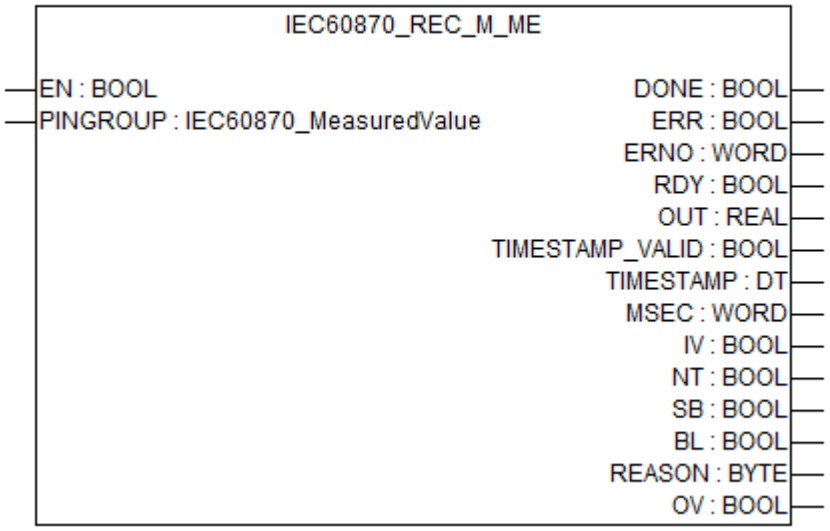
Reception of data messages of the data type according to IEC 60870-5 (M_DP_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

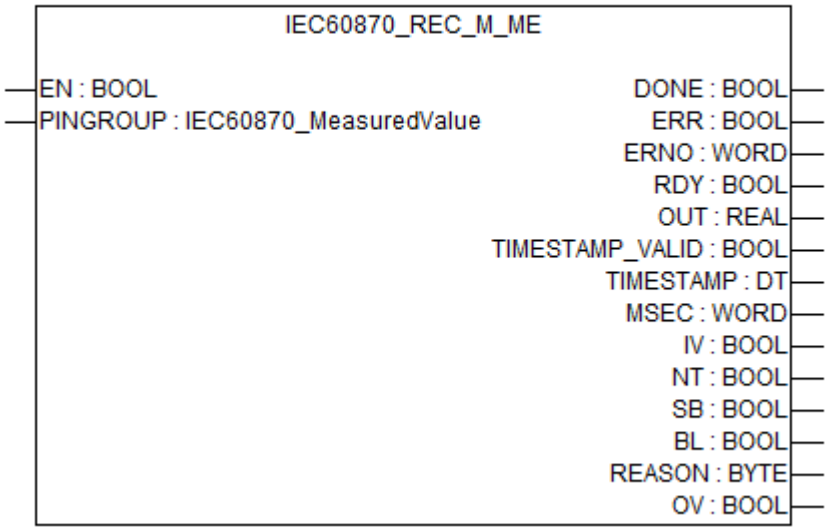
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OUT REAL (output)

The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid)

Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp)

If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds)

Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

IV BOOL (invalid)

Output IV displays the status/information of the data. If IV = TRUE, the data is invalid.

The output always has to be considered together with output RDY.

NT BOOL (not topical)

Output NT displays the status/information of the data. If NT = TRUE, the data is not topical/ actual.

The output always has to be considered together with output RDY.

SB BOOL (substituted)

Output SB displays the status/information of the data. If SB = TRUE, the data is substituted.

The output always has to be considered together with output RDY.

BL BOOL (blocked)

Output BL displays the status/information of the data. If BL = TRUE, the data is blocked.

The output always has to be considered together with output RDY.

REASON BYTE (reason)

Output REASON displays the cause of the transmission.

The output always has to be considered together with output RDY.

Reasons for transmission:

1	Periodic/cyclic
2	Background interrogation
3	Spontaneous
4	Initialised
5	Requested
6	Activation
7	Confirmation of activation
8	Deactivation
9	Confirmation of deactivation
10	Termination of activation
11	Return information, caused by a remote command
12	Return information, caused by a local command
20	Interrogated by general inquiry

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

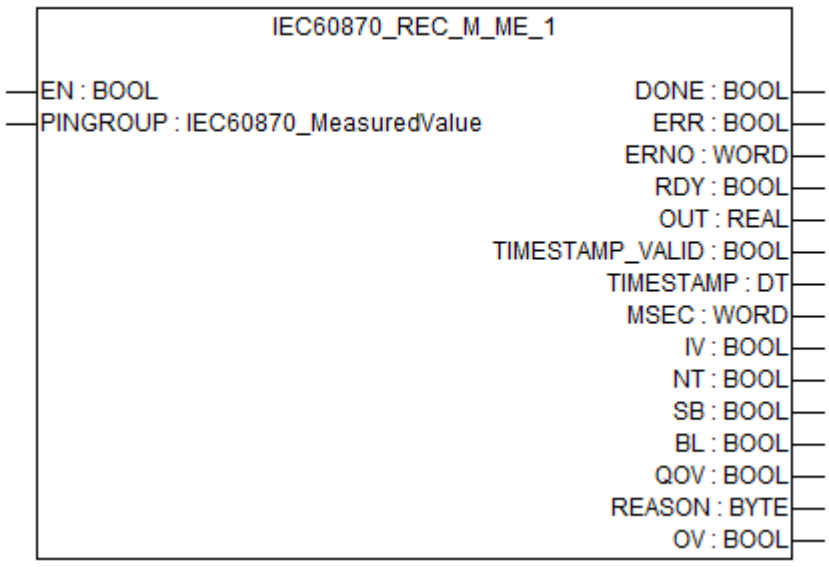
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN       := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
REC_OUT             := REC.OUT;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_IV              := REC.IV;
REC_NT              := REC.NT;
REC_SB              := REC.SB;
REC_BL              := REC.BL;
REC_REASON          := REC.REASON;
REC_OV              := REC.OV;
```

IEC60870_REC_M_ME_1



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

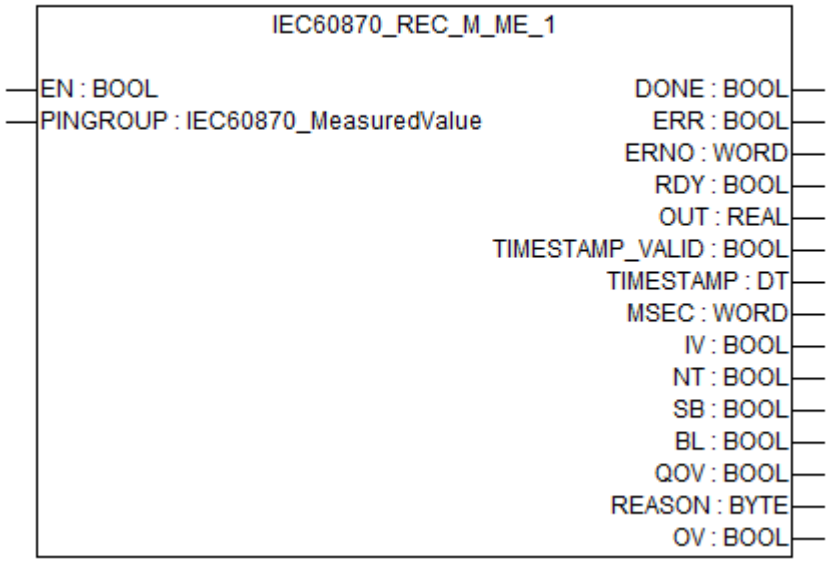
Reception of data messages of the data type according to IEC 60870-5 (M_DP_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

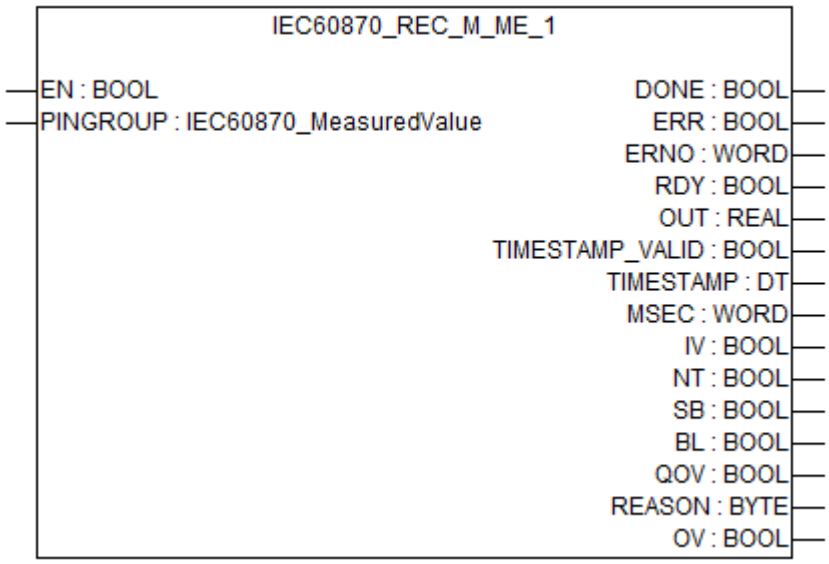
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OUT REAL (output)

The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid)

Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp)

If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds)

Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

IV BOOL (invalid)

Output IV displays the status/information of the data. If IV = TRUE, the data is invalid.

The output always has to be considered together with output RDY.

NT BOOL (not topical)

Output NT displays the status/information of the data. If NT = TRUE, the data is not topical/ actual.

The output always has to be considered together with output RDY.

SB BOOL (substituted)

Output SB displays the status/information of the data. If SB = TRUE, the data is substituted.

The output always has to be considered together with output RDY.

BL BOOL (blocked)

Output BL displays the status/information of the data. If BL = TRUE, the data is blocked.

The output always has to be considered together with output RDY.

QOV BOOL (overflow)

Output QOV displays the status/information of the data. If QOV = TRUE, the data signals an overflow.

The output always has to be considered together with output RDY.

REASON BYTE (reason)

Output REASON displays the cause of the transmission.

The output always has to be considered together with output RDY.

Reasons for transmission:

1	Periodic/cyclic
2	Background interrogation
3	Spontaneous
4	Initialised
5	Requested
6	Activation
7	Confirmation of activation
8	Deactivation
9	Confirmation of deactivation
10	Termination of activation
11	Return information, caused by a remote command

12	Return information, caused by a local command
20	Interrogated by general inquiry

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

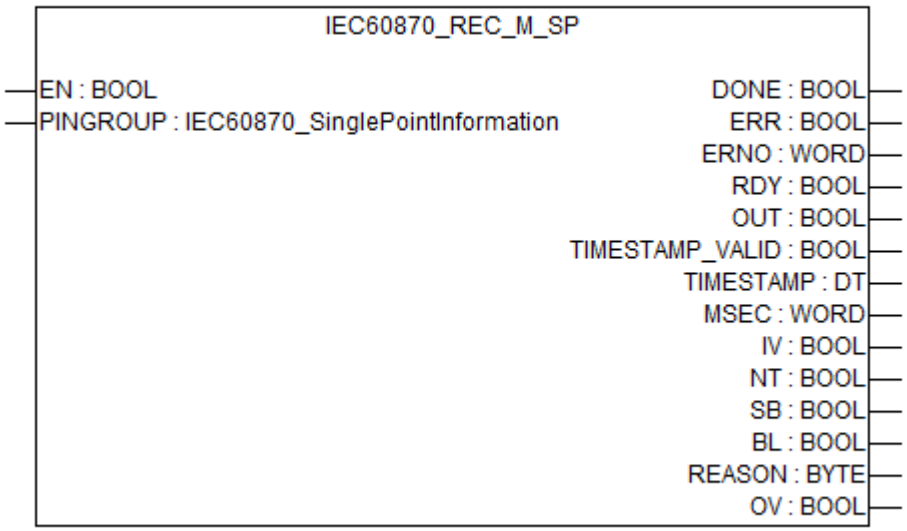
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN       := REC_EN,
        PINGROUP := REC_PINGROUP);


REC_DONE            := REC.DONE;
REC_ERR             := REC.ERR;
REC_ERNO            := REC.ERNO;
REC_RDY             := REC.RDY;
REC_OUT             := REC.OUT;
REC_TIMESTAMP_VALID := REC.TIMESTAMP_VALID;
REC_TIMESTAMP       := REC.TIMESTAMP;
REC_MSEC            := REC.MSEC;
REC_IV              := REC.IV;
REC_NT              := REC.NT;
REC_SB              := REC.SB;
REC_BL              := REC.BL;
REC_QOV             := REC.QOV;
REC_REASON          := REC.REASON;
REC_OV              := REC.OV;
```

IEC60870_REC_M_SP



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

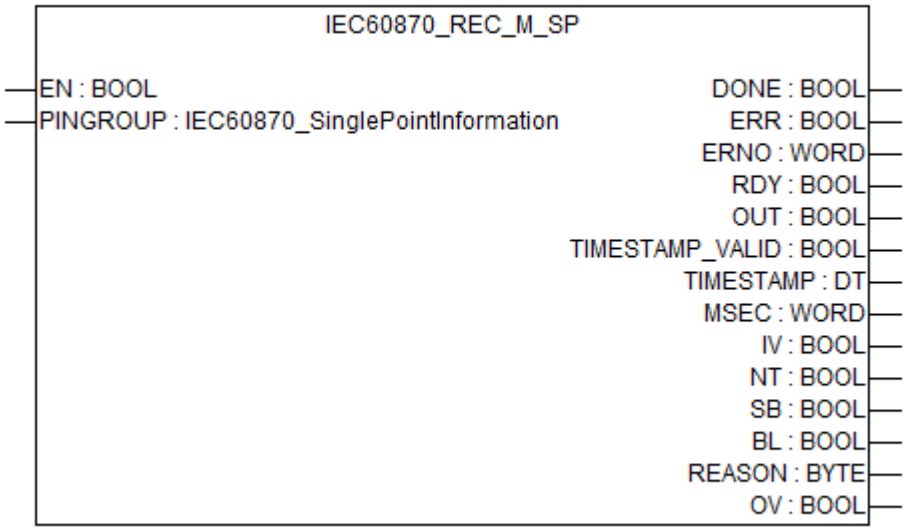
Reception of data messages of the data type according to IEC 60870-5 (M_DP_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SinglePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

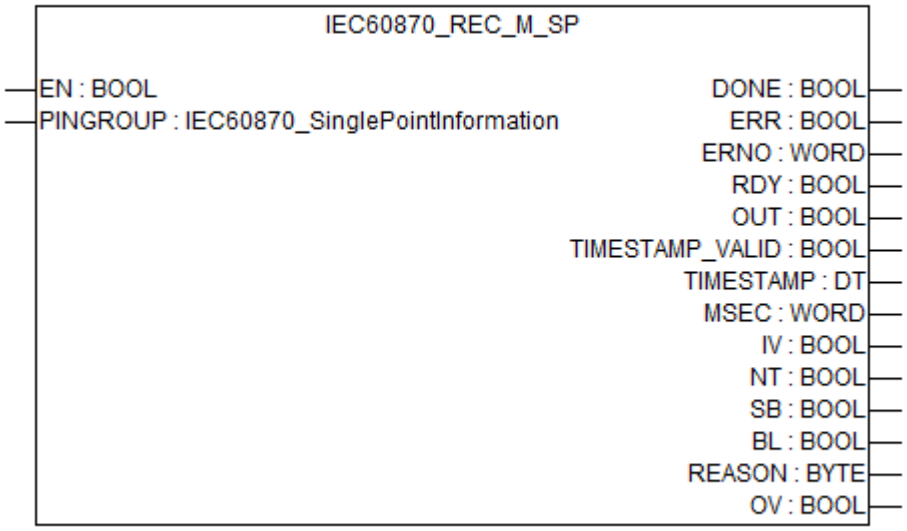
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OUT REAL (output) The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

TIME-STAMP_VALID BOOL (time-stampvalid) Valid timestamp was received when the output is TRUE.

The output always has to be considered together with output RDY.

TIMESTAMP DT (time-stamp) If data with timestamp is received, the timestamp is given at this output.

The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

MSEC WORD (milliseconds) Value of the received timestamp in milliseconds. The value is given if a valid timestamp was received and the output VALID_TIMESTAMP = TRUE.

IV BOOL (invalid) Output IV displays the status/information of the data. If IV = TRUE, the data is invalid.

The output always has to be considered together with output RDY.

NT BOOL (not topical) Output NT displays the status/information of the data. If NT = TRUE, the data is not topical/ actual.

The output always has to be considered together with output RDY.

SB BOOL (substituted) Output SB displays the status/information of the data. If SB = TRUE, the data is substituted.

The output always has to be considered together with output RDY.

BL BOOL (blocked) Output BL displays the status/information of the data. If BL = TRUE, the data is blocked.

The output always has to be considered together with output RDY.

REASON BYTE (reason) Output REASON displays the cause of the transmission.

The output always has to be considered together with output RDY.

Reasons for transmission:

1	Periodic/cyclic
2	Background interrogation
3	Spontaneous
4	Initialised
5	Requested
6	Activation
7	Confirmation of activation

8	Deactivation
9	Confirmation of deactivation
10	Termination of activation
11	Return information, caused by a remote command
12	Return information, caused by a local command
20	Interrogated by general inquiry

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

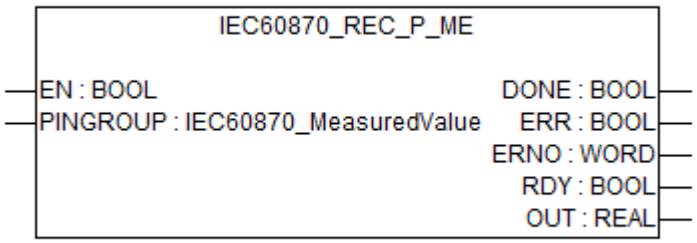
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN        := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE             := REC.DONE;
REC_ERR              := REC.ERR;
REC_ERNO             := REC.ERNO;
REC_RDY              := REC.RDY;
REC_OUT              := REC.OUT;
REC_TIMESTAMP_VALID  := REC.TIMESTAMP_VALID;
REC_TIMESTAMP        := REC.TIMESTAMP;
REC_MSEC             := REC.MSEC;
REC_IV               := REC.IV;
REC_NT               := REC.NT;
REC_SB               := REC.SB;
REC_BL               := REC.BL;
REC_REASON           := REC.REASON;
REC_OV               := REC.OV;
```

IEC60870_REC_P_ME



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0

Parameter	Value
Type	Function block with historical values
Group	Data

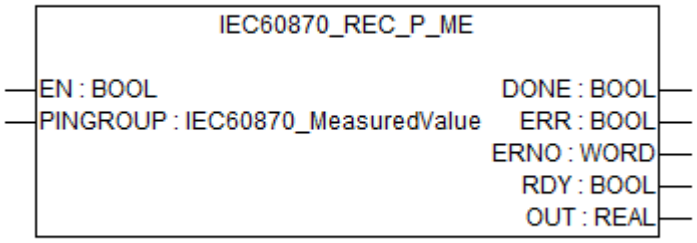
Reception of data messages of the data type according to IEC 60870-5 (M_IT_xx message).

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The data received are available at the corresponding output pins.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

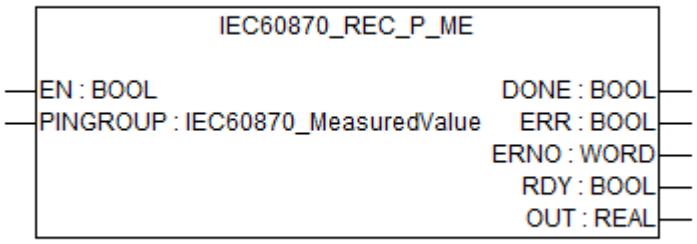
The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OUT REAL (output)

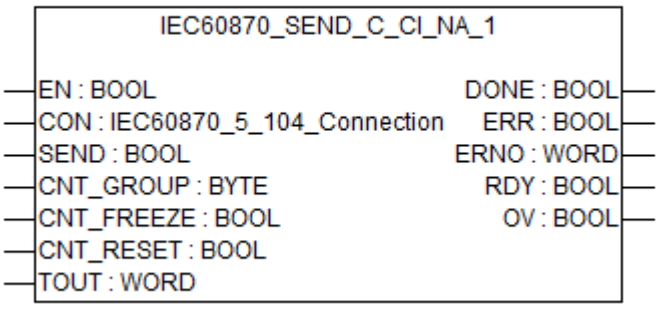
The output OUT gives the received value as a floating point number.

The output always has to be considered together with output RDY.

Function call in ST

```
REC    (EN        := REC_EN,
PINGROUP := REC_PINGROUP);


REC_DONE             := REC.DONE;
REC_ERR              := REC.ERR;
REC_ERNO             := REC.ERNO;
REC_RDY              := REC.RDY;
REC_OUT              := REC.OUT;
```

IEC60870_SEND_C_CI_NA_1



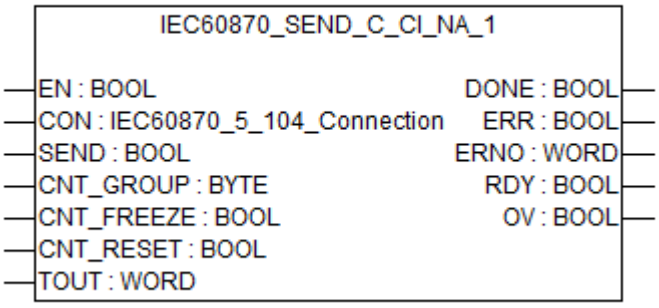
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

Start a send request on a rising edge.

CNT_GROUP BYTE (counter group)

Counter group (1...4) to poll; 5 means all groups.

CNT_FREEZE BOOL (counter freeze)

Freeze the counter.

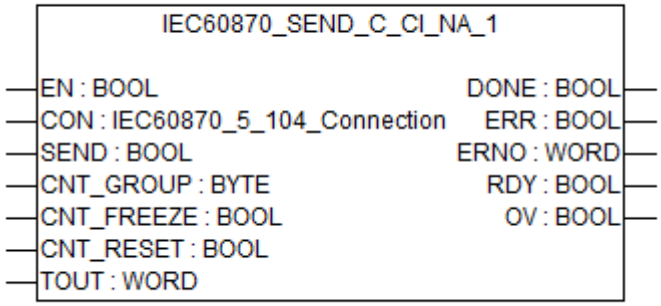
CNT_RESET BOOL (counter reset)

Reset the counter.

TOUT WORD (timeout)

Timeout in ms until ACTERM is expected; 0 = infinite.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

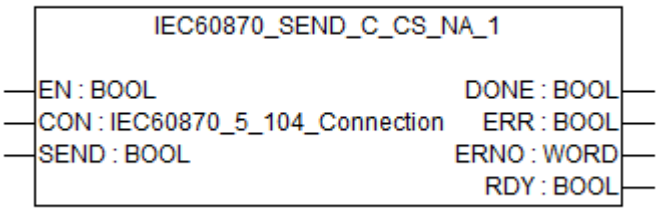
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN          := SEND_EN,
CON         := SEND_CON,
SEND        := SEND_SEND,
CNT_GROUP   := SEND_CNT_GROUP,
CNT_FREEZE  := SEND_SNT_FREEZE,
CNT_RESET   := SEND_CNT_RESET,
TOUT        := SEND_TOUT);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
SEND_OV               := SEND.OV;
```

IEC60870_SEND_C_CS_NA_1



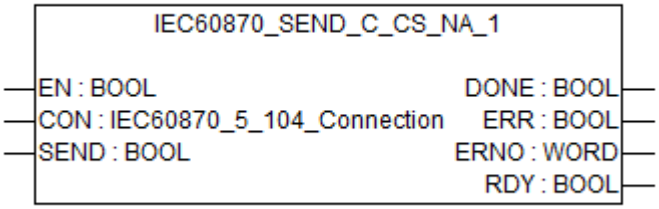
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

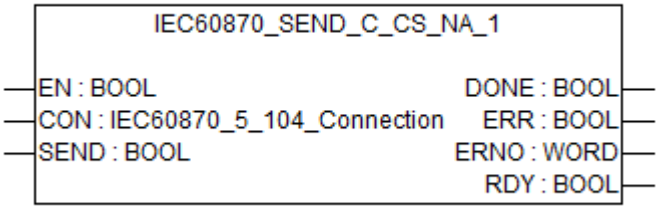
The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

Start a send request on a rising edge.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

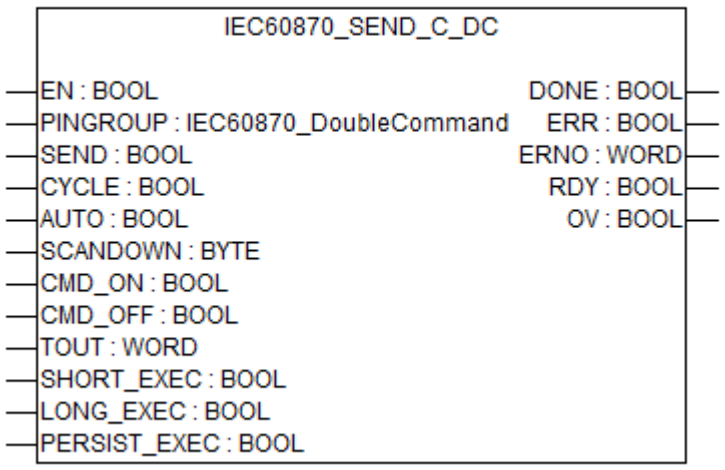
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

Function call in ST

```
SEND    (EN          := SEND_EN,
CON         := SEND_CON,
SEND        := SEND_SEND);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
```

IEC60870_SEND_C_DC



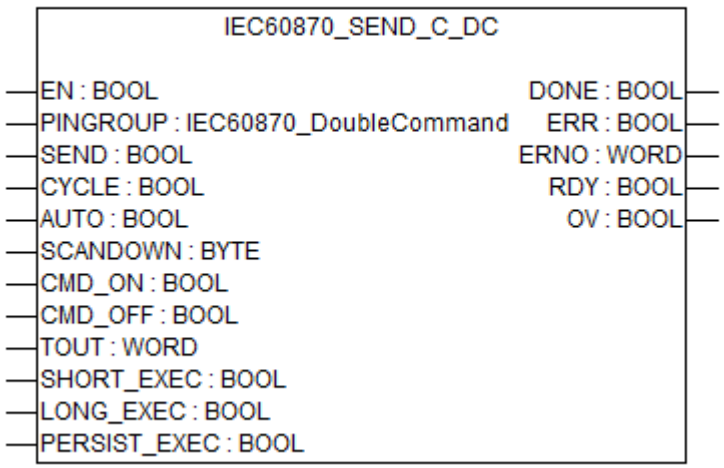
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_DoubleCommand (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec, see *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*).

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

CMD_ON BOOL (commandon) Input value Bit 1.

CMD_OFF BOOL (commandoff) Input value Bit 2.

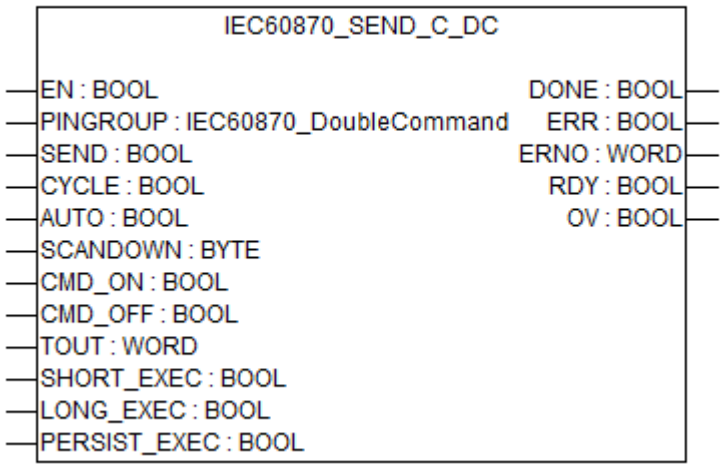
TOUT WORD (timeout) Timeout in ms until ACTERM is expected; 0 = infinite.

SHORT_EXEC BOOL (shortexecution) If input SHORT_EXEC = TRUE, short execution is enabled on controlled station.

LONG_EXEC BOOL (longexecution) If input LONG_EXEC = TRUE, long execution is enabled on controlled station.

PERSIST_EXEC BOOL (persistent execution) If input PERSISTENT_EXEC = TRUE, persistent execution is enabled on controlled station.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

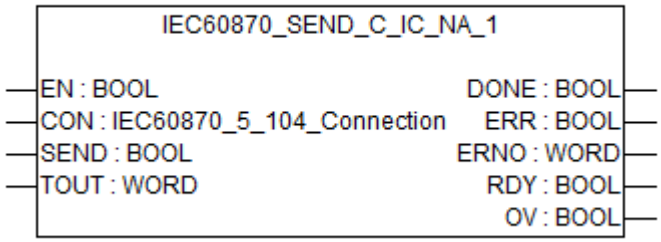
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
PINGROUP     := SEND_PINGROUP,
EXEC_TIME    := SEND_EXEC_TIME,
SEND         := SEND_SEND,
CYCLE        := SEND_CYCLE,
AUTO         := SEND_AUTO,
SCANDOWN     := SEND_SCANDOWN,
CMD_ON       := SEND_CMD_ON,
CMD_OFF      := SEND_CMD_OFF,
TOUT         := SEND_TOUT,
SHORT_EXEC   := SEND_SHORT_EXEC,
LONG_EXEC    := SEND_LONG_EXEC,
PERSIST_EXEC := SEND_PERSIST_EXEC);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_C_IC_NA_1



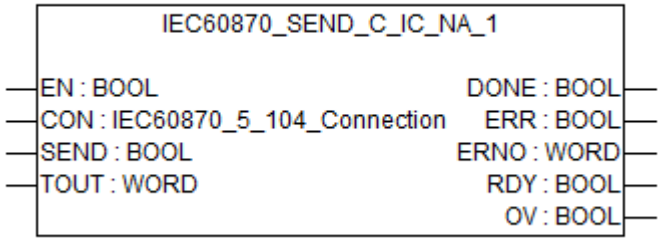
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5 protocol.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

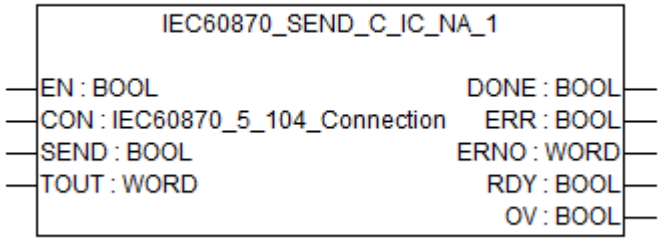
SEND BOOL (send)

Start a send request on a rising edge.

TOUT WORD (timeout)

Timeout in ms until ACTERM is expected; 0 = infinite.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

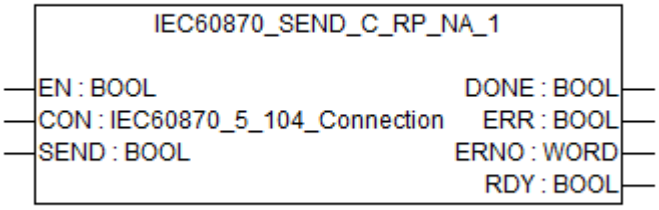
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN          := SEND_EN,
CON         := SEND_CON,
SEND        := SEND_SEND,
TOUT        := SEND_TOUT);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
SEND_OV               := SEND.OV;
```

IEC60870_SEND_C_RP_NA_1



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0

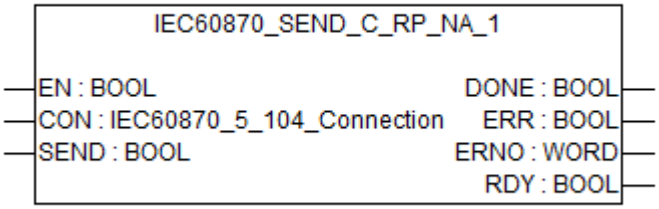
Parameter	Value
Type	Function block with historical values
Group	System_Information

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

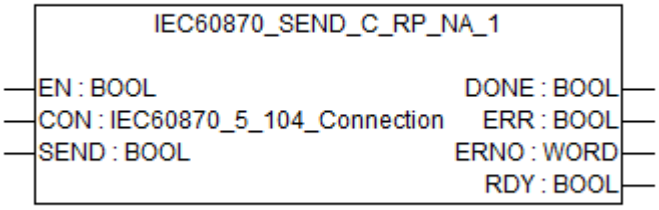
The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

Start a send request on a rising edge.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

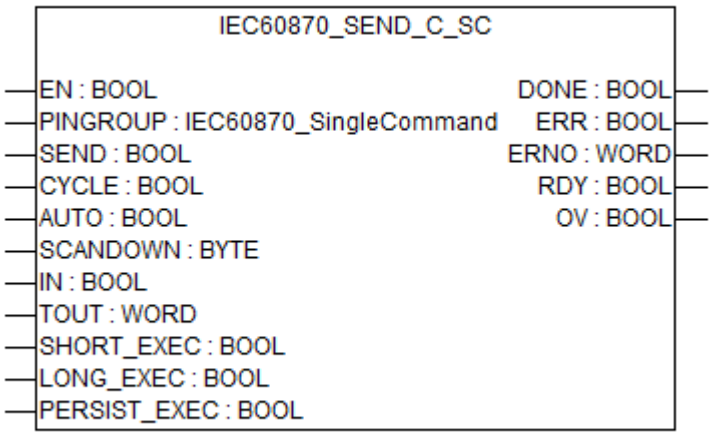
RDY (ready) BOOL

A command has been received if RDY = TRUE.

Function call in ST

```
SEND   (EN          := SEND_EN,
     CON         := SEND_CON,
SEND        := SEND_SEND);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
```

IEC60870_SEND_C_SC



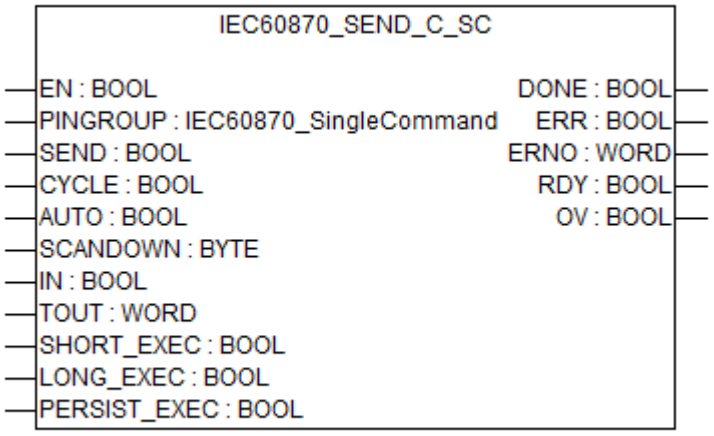
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SingleCommand (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec, see *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown)

Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN BOOL (input)

Input value Bit 1.

TOUT WORD (timeout)

Timeout in ms until ACTERM is expected; 0 = infinite.

SHORT_EXEC BOOL (short-execution)

If input SHORT_EXEC = TRUE, short execution is enabled on controlled station.

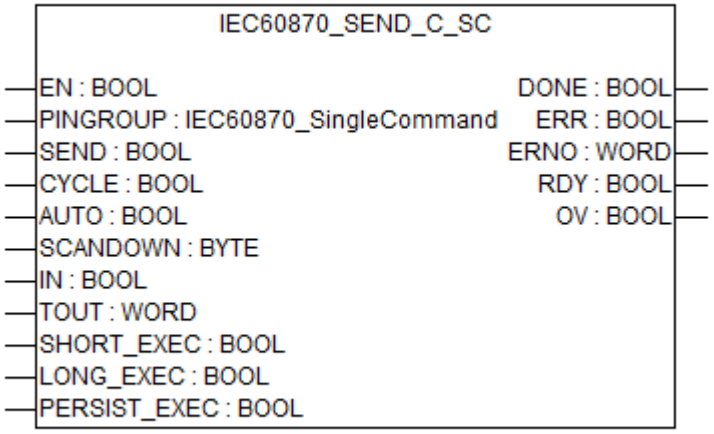
LONG_EXEC BOOL (longexecution)

If input LONG_EXEC = TRUE, long execution is enabled on controlled station.

PERSIST_EXEC BOOL (persistent execution)

If input PERSISTENT_EXEC = TRUE, persistent execution is enabled on controlled station.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

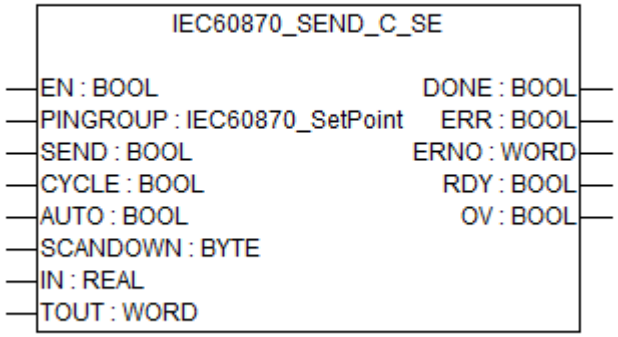
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
PINGROUP     := SEND_PINGROUP,
EXEC_TIME    := SEND_EXEC_TIME,
SEND         := SEND_SEND,
CYCLE        := SEND_CYCLE,
AUTO         := SEND_AUTO,
SCANDOWN     := SEND_SCANDOWN,
IN           := SEND_IN,
TOUT         := SEND_TOUT,
SHORT_EXEC   := SEND_SHORT_EXEC,
LONG_EXEC    := SEND_LONG_EXEC,
PERSIST_EXEC := SEND_PERSIST_EXEC);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_C_SE



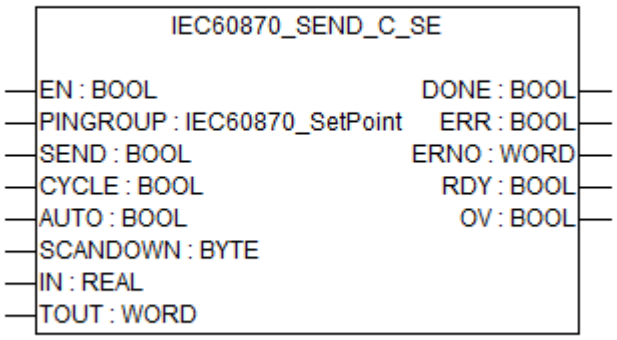
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SetPoint (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown)

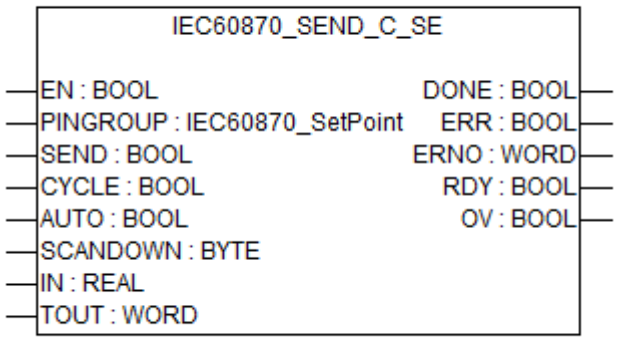
Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN BOOL (input) Input value Bit 1.

TOUT WORD (timeout) Timeout in ms until ACTERM is expected; 0 = infinite.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

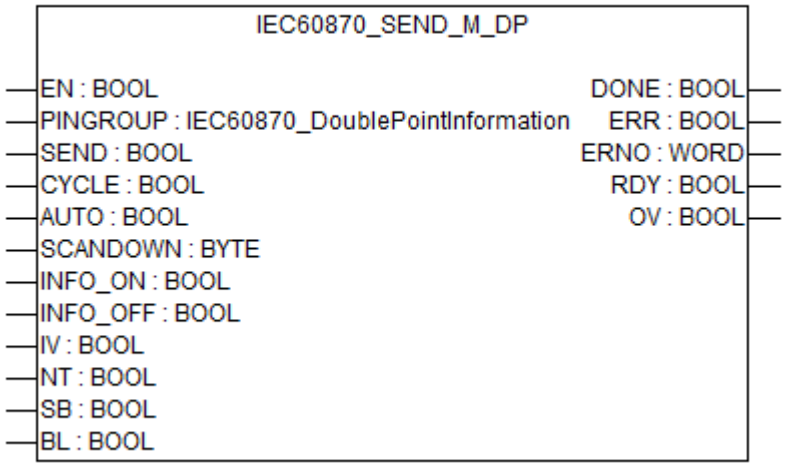
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
PINGROUP     := SEND_PINGROUP,
EXEC_TIME    := SEND_EXEC_TIME,
SEND         := SEND_SEND,
CYCLE        := SEND_CYCLE,
AUTO         := SEND_AUTO,
SCANDOWN     := SEND_SCANDOWN,
IN           := SEND_IN,
TOUT         := SEND_TOUT);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_DP



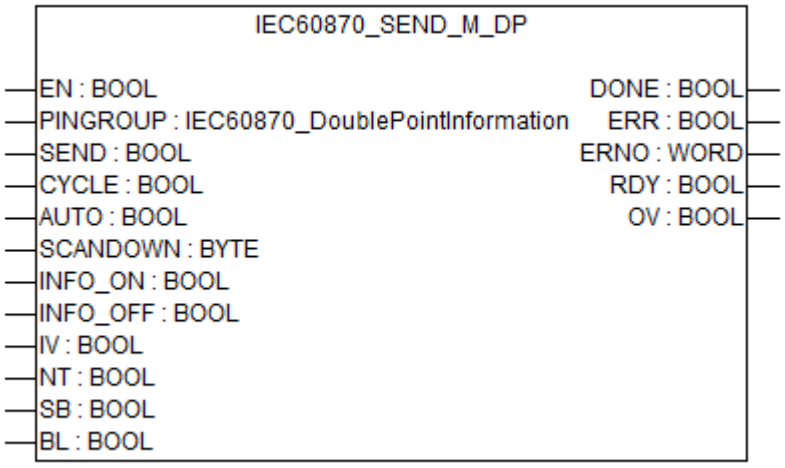
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_DoublePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC 60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

INFO_ON BOOL (information on) Input value of bit 1.

INFO_OFF BOOL (information off) Input value of bit 2.

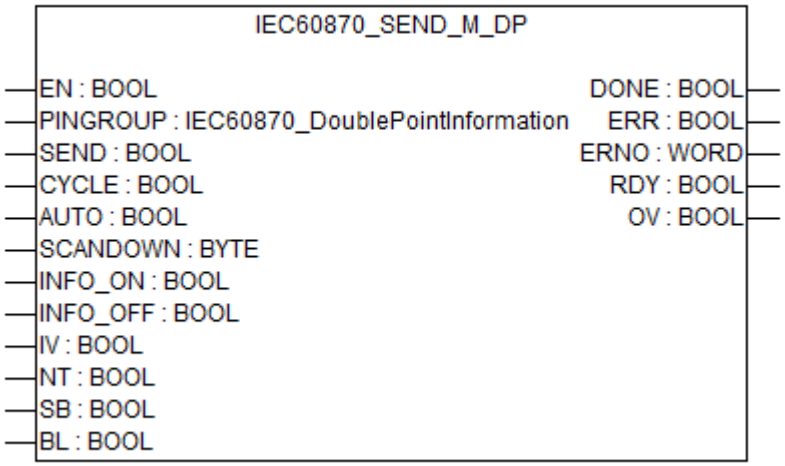
IV BOOL (invalid) Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

SB BOOL (substituted) Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked) Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

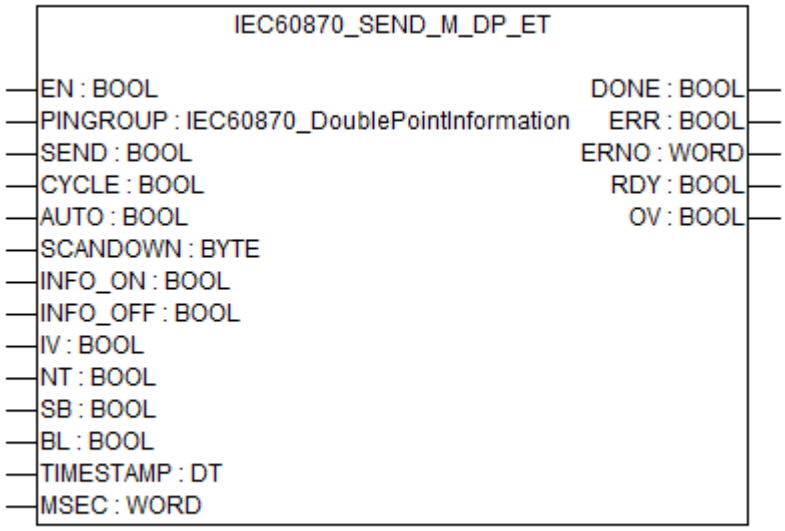
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND     (EN       := SEND_EN,
 PINGROUP := SEND_PINGROUP,
 SEND     := SEND_SEND,
 CYCLE    := SEND_CYCLE,
 AUTO     := SEND_AUTO,
 SCANDOWN := SEND_SCANDOWN,
 INFO_ON  := SEND_INFO_ON,
 INFO_OFF := SEND_INFO_OFF,
 IV       := SEND_IV,
 NT       := SEND_NT,
 SB       := SEND_SB,
 BL       := SEND_BL);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_DP_ET



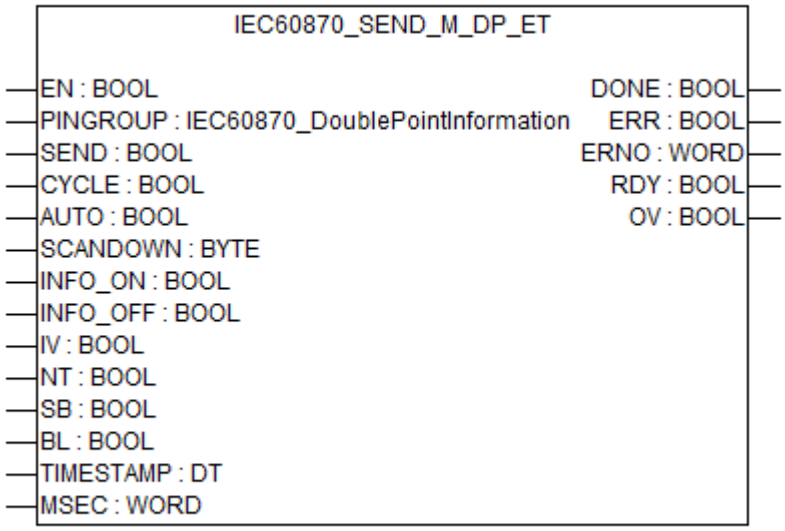
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_DoublePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC 60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

INFO_ON BOOL (information on) Input value of bit 1.

INFO_OFF BOOL (information off) Input value of bit 2.

IV BOOL (invalid) Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

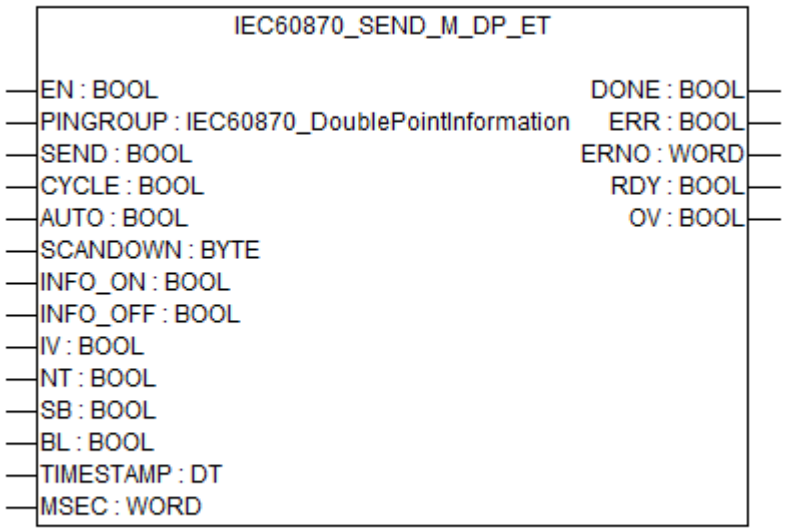
SB BOOL (substituted) Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked) Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

TIMESTAMP DT (timestamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

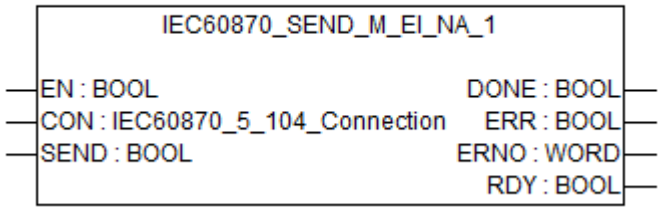
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND     (EN         := SEND_EN,
          PINGROUP   := SEND_PINGROUP,
          SEND       := SEND_SEND,
          CYCLE      := SEND_CYCLE,
          AUTO       := SEND_AUTO,
          SCANDOWN   := SEND_SCANDOWN,
          INFO_ON    := SEND_INFO_ON,
          INFO_OFF   := SEND_INFO_OFF,
          IV         := SEND_IV,
          NT         := SEND_NT,
          SB         := SEND_SB,
          BL         := SEND_BL,
          TIMESTAMP  := SEND_TIMESTAMP,
          MSEC       := SEND_MSEC);


SEND_DONE           := SEND.DONE;
SEND_ERR            := SEND.ERR;
SEND_ERNO           := SEND.ERNO;
SEND_RDY            := SEND.RDY;
SEND_OV             := SEND.OV;
```

IEC60870_SEND_M_EI_NA_1



Sending of data messages of the data type according to IEC 60870-5 protocol.

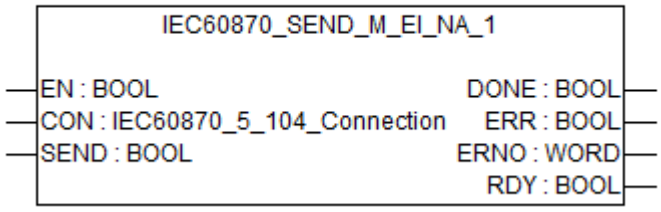
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	System_Information

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

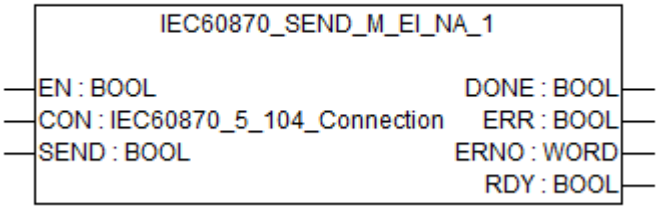
At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send) Start a send request on a rising edge.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

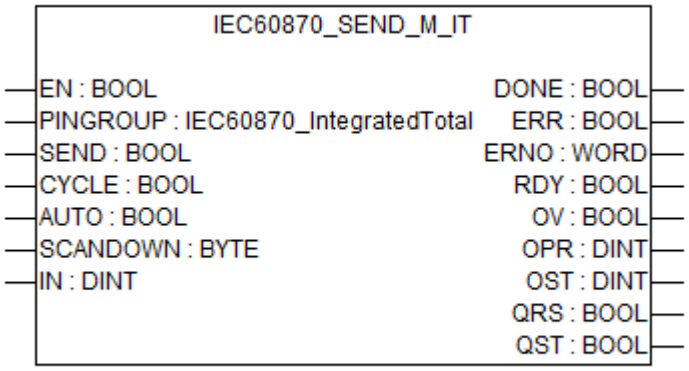
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

Function call in ST

```
SEND    (EN          := SEND_EN,
         CON         := SEND_CON,
SEND        := SEND_SEND);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
```

IEC60870_SEND_M_IT



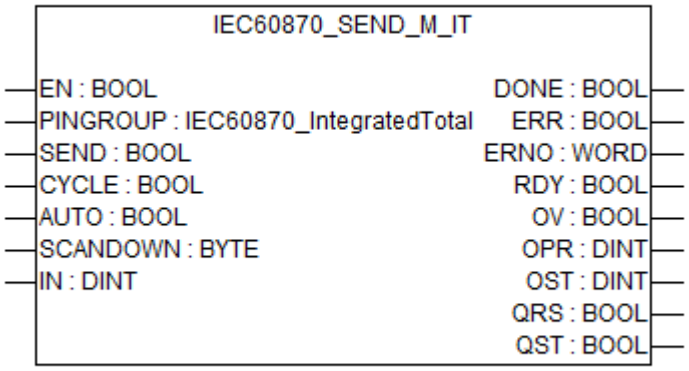
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown)

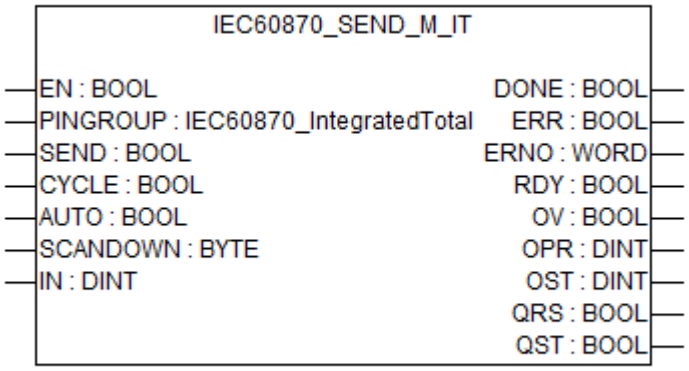
Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN BOOL (input)

The input value.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

OPR DINT (opr)

Preset of discrete value. Output OPR receives the current discrete value on reset.

The output always has to be considered together with output RDY.

OST DINT (ost) Relocated discrete value. Output OST receives the current discrete value on relocation.
The output always has to be considered together with output RDY.

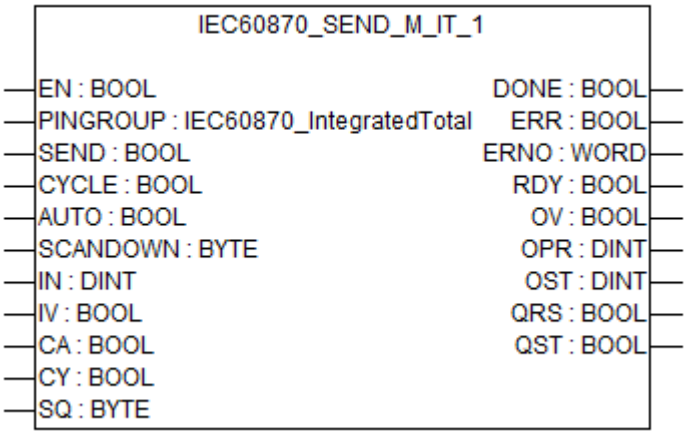
QRS BOOL (qrs) The output QRS indicates a reset of the discrete value by changing from FALSE to TRUE.
The output always has to be considered together with output RDY.

QST BOOL (qst) Relocate discrete value.
The output QRS indicates a relocation of the discrete value by changing from FALSE to TRUE.
The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
PINGROUP     := SEND_PINGROUP,
SEND         := SEND_SEND,
CYCLE        := SEND_CYCLE,
AUTO         := SEND_AUTO,
SCANDOWN     := SEND_SCANDOWN,
IN           := SEND_IN);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
SEND_OPR             := SEND.OPR;
SEND_OST             := SEND.OST;
SEND_QRS             := SEND.QRS;
SEND_QST             := SEND.QST;
```

IEC60870_SEND_M_IT_1



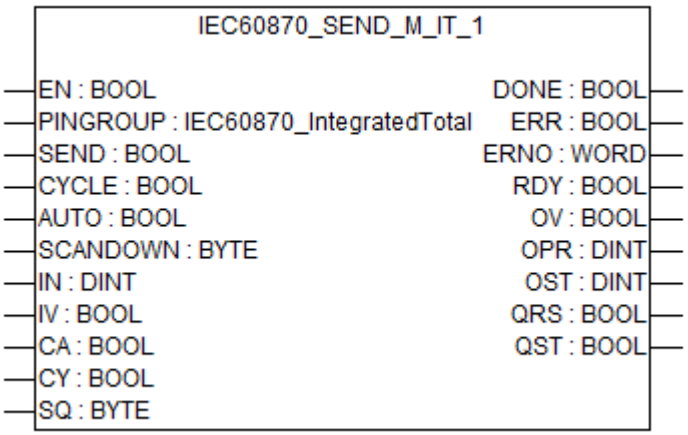
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

> *A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!*

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN BOOL (input) The input value.

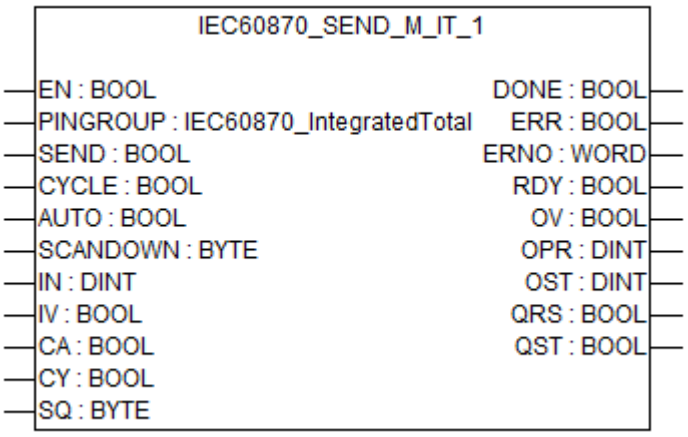
IV BOOL (invalid) Input IV is the status/information of the data. If IV = TRUE, the data is invalid.

CA BOOL (counter adjusted) Input CA is the status/information of the data. If CA = TRUE, the counter has been preset since the last reading.

CY BOOL (carry) Input CY is the status/informations of the data. If CY = TRUE, the transmission divided into associated measurement periods.

SQ BYTE (sequence) Input SQ is the sequence number.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

OPR DINT (opr)

Preset of discrete value. Output OPR receives the current discrete value on reset.

The output always has to be considered together with output RDY.

OST DINT (ost)

Relocated discrete value. Output OST receives the current discrete value on relocation.

The output always has to be considered together with output RDY.

QRS BOOL (qrs)

The output QRS indicates a reset of the discrete value by changing from FALSE to TRUE.

The output always has to be considered together with output RDY.

QST BOOL (qst)

Relocate discrete value.

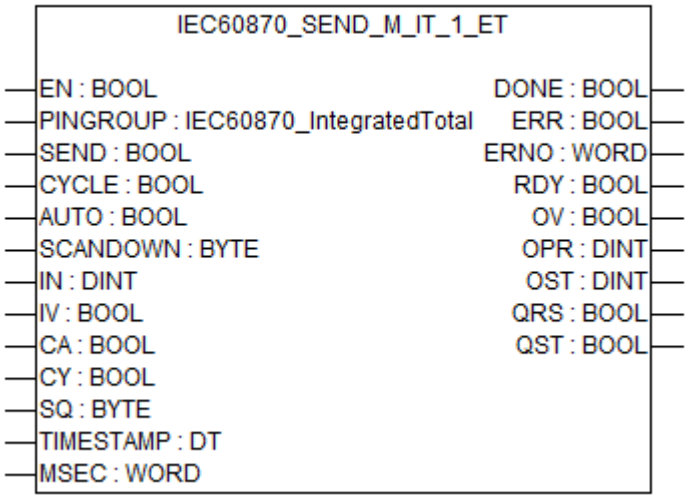
The output QRS indicates a relocation of the discrete value by changing from FALSE to TRUE.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
         PINGROUP       := SEND_PINGROUP,
         SEND           := SEND_SEND,
         CYCLE          := SEND_CYCLE,
         AUTO           := SEND_AUTO,
         SCANDOWN       := SEND_SCANDOWN,
         IN             := SEND_IN,
         IV             := SEND_IV,
         CA             := SEND_CA,
         CY             := SEND_CY,
         SQ             := SEND_SQ);


SEND_DONE               := SEND.DONE;
SEND_ERR                := SEND.ERR;
SEND_ERNO               := SEND.ERNO;
SEND_RDY                := SEND.RDY;
SEND_OV                 := SEND.OV;
SEND_OPR                := SEND.OPR;
SEND_OST                := SEND.OST;
SEND_QRS                := SEND.QRS;
SEND_QST                := SEND.QST;
```

IEC60870_SEND_M_IT_1_ET



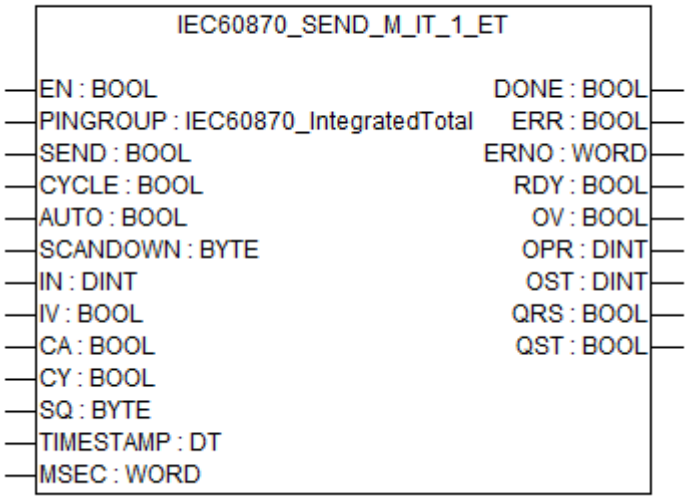
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN BOOL (input) The input value.

IV BOOL (invalid) Input IV is the status/information of the data. If IV = TRUE, the data is invalid.

CA BOOL (counter adjusted) Input CA is the status/information of the data. If CA = TRUE, the counter has been preset since the last reading.

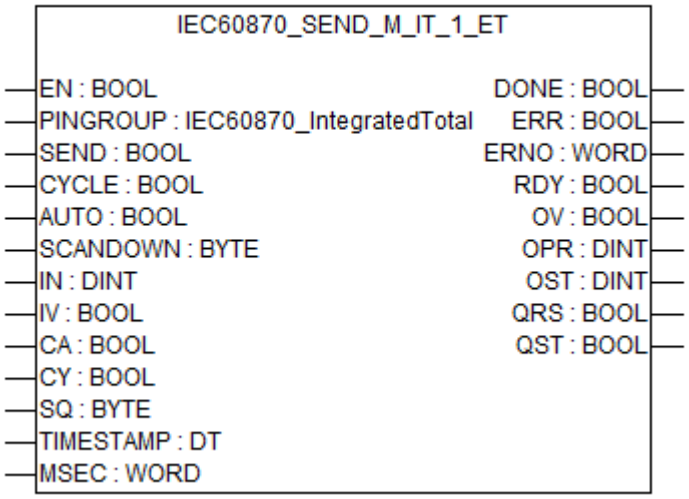
CY BOOL (carry) Input CY is the status/informations of the data. If CY = TRUE, the transmission divided into associated measurement periods.

SQ BYTE (sequence) Input SQ is the sequence number.

TIMESTAMP DT (timestamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

OPR DINT (opr) Preset of discrete value. Output OPR receives the current discrete value on reset.

The output always has to be considered together with output RDY.

OST DINT (ost) Relocated discrete value. Output OST receives the current discrete value on relocation.

The output always has to be considered together with output RDY.

QRS BOOL (qrs) The output QRS indicates a reset of the discrete value by changing from FALSE to TRUE.

The output always has to be considered together with output RDY.

QST BOOL (qst) Relocate discrete value.

The output QRS indicates a relocation of the discrete value by changing from FALSE to TRUE.

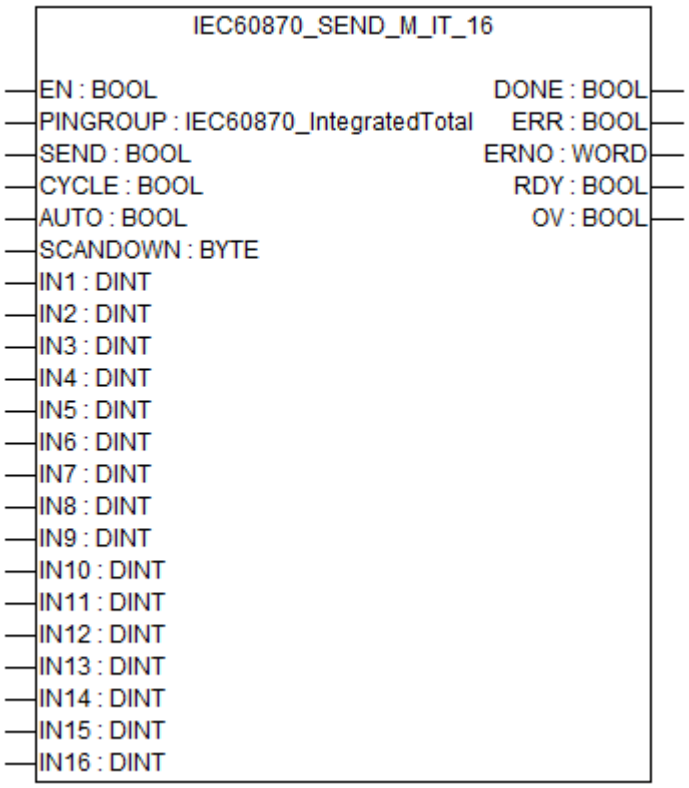
The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        SCANDOWN     := SEND_SCANDOWN,
        IN           := SEND_IN,
        IV           := SEND_IV,
        CA           := SEND_CA,
        CY           := SEND_CY,
        SQ           := SEND_SQ,
        TIMESTAMP    := SEND_TIMESTAMP,
        MSEC         := SEND_MSEC);
```

```
SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
SEND_OV               := SEND.OV;
SEND_OPR              := SEND.OPR;
SEND_OST              := SEND.OST;
SEND_QRS              := SEND.QRS;
SEND_QST              := SEND.QST;
```

IEC60870_SEND_M_IT_16



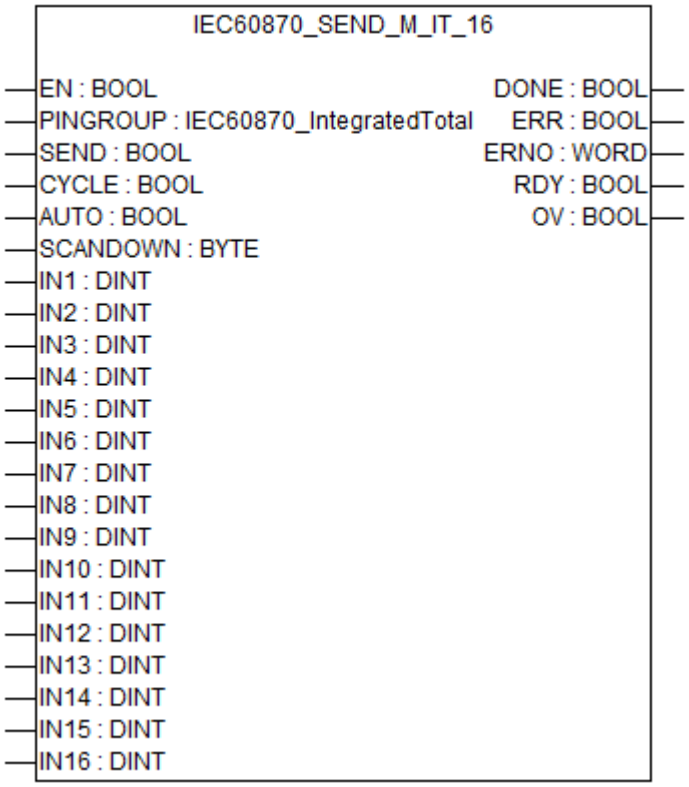
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

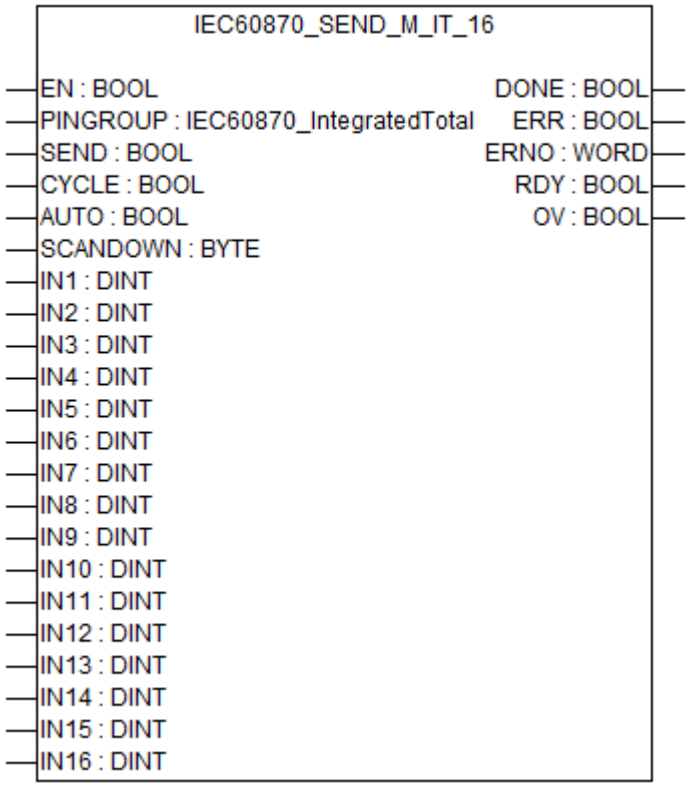
CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 DINT (input 1...16) Input value for member 1 to 16.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

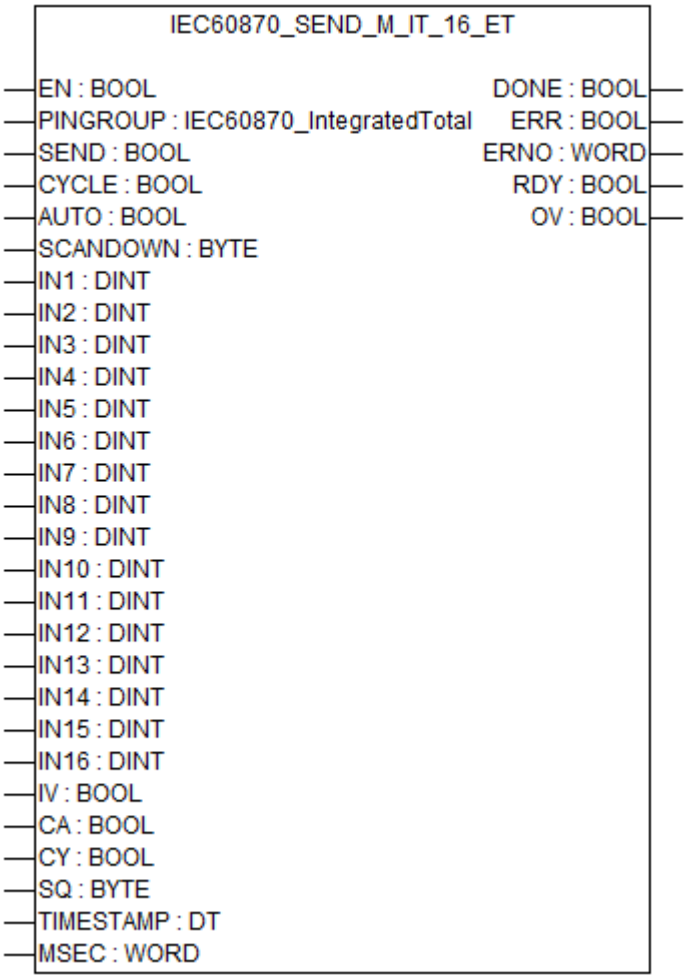
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
         PINGROUP       := SEND_PINGROUP,
         SEND           := SEND_SEND,
         CYCLE          := SEND_CYCLE,
         AUTO           := SEND_AUTO,
         SCANDOWN       := SEND_SCANDOWN,
         IN1            := SEND_IN1,
         IN2            := SEND_IN2,
         IN3            := SEND_IN3,
         IN4            := SEND_IN4,
         IN5            := SEND_IN5,
         IN6            := SEND_IN6,
         IN7            := SEND_IN7,
         IN8            := SEND_IN8,
         IN9            := SEND_IN9,
         IN10           := SEND_IN10,
         IN11           := SEND_IN11,
         IN12           := SEND_IN12,
         IN13           := SEND_IN13,
         IN14           := SEND_IN14,
         IN15           := SEND_IN15,
         IN16           := SEND_IN16);


SEND_DONE               := SEND.DONE;
SEND_ERR                := SEND.ERR;
SEND_ERNO               := SEND.ERNO;
SEND_RDY                := SEND.RDY;
SEND_OV                 := SEND.OV;
```

IEC60870_SEND_M_IT_16_ET



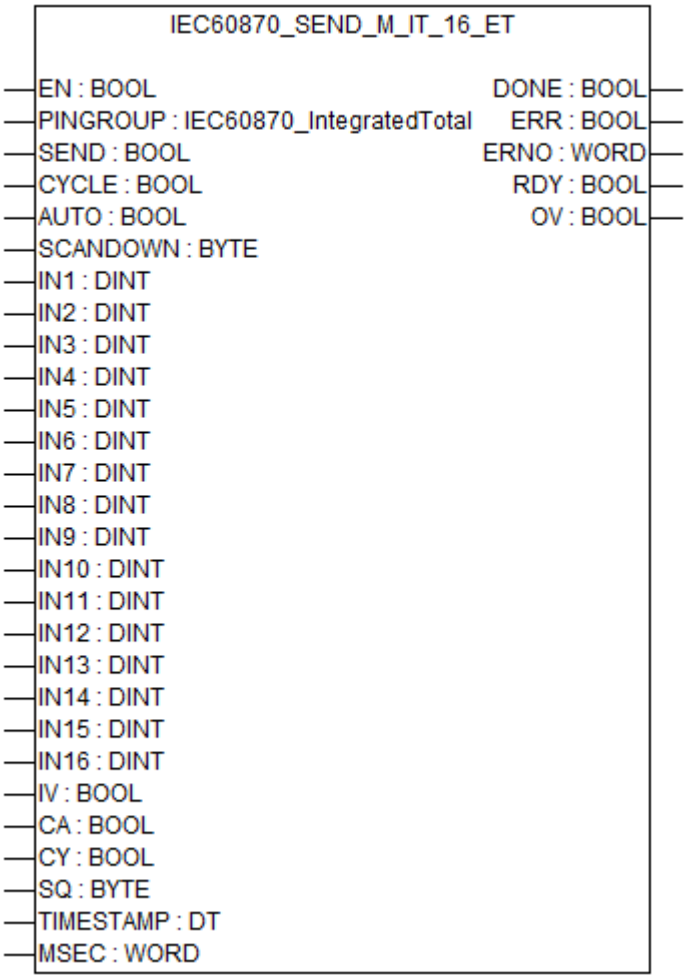
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 DINT (input 1...16) Input value for member 1 to 16.

CA BOOL (counter adjusted) Input CA is the status/information of the data. If CA = TRUE, the counter has been preset since the last reading.

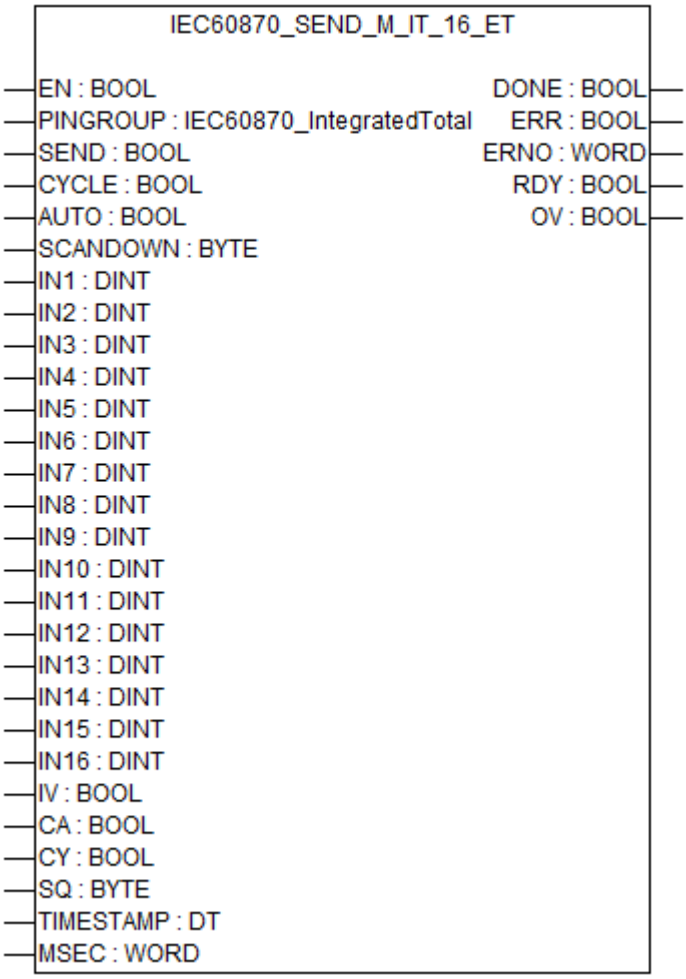
CY BOOL (carry) Input CY is the status/informations of the data. If CY = TRUE, the transmission divided into associated measurement periods.

SQ BYTE (sequence) Input SQ is the sequence number.

TIMESTAMP DT (timestamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

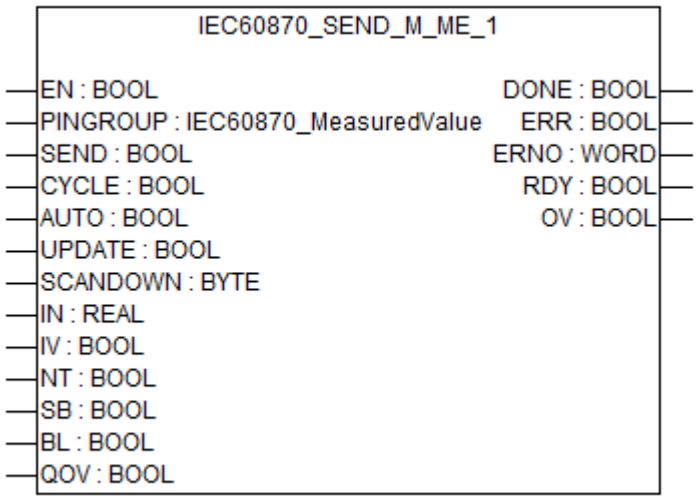
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        SCANDOWN     := SEND_SCANDOWN,
        IN1          := SEND_IN1,
        IN2          := SEND_IN2,
        IN3          := SEND_IN3,
        IN4          := SEND_IN4,
        IN5          := SEND_IN5,
        IN6          := SEND_IN6,
        IN7          := SEND_IN7,
        IN8          := SEND_IN8,
        IN9          := SEND_IN9,
        IN10         := SEND_IN10,
        IN11         := SEND_IN11,
        IN12         := SEND_IN12,
        IN13         := SEND_IN13,
        IN14         := SEND_IN14,
        IN15         := SEND_IN15,
        IN16         := SEND_IN16,
        IV           := SEND_IV,
        CA           := SEND_CA,
        CY           := SEND_CY,
        SQ           := SEND_SQ,
        TIMESTAMP    := SEND_TIMESTAMP,
        MSEC         := SEND_MSEC);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_ME_1



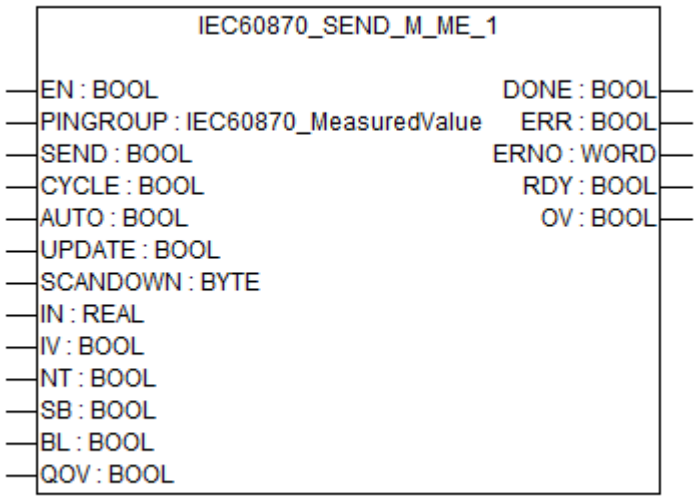
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

UPDATE BOOL (update) If UPDATE = TRUE, updates of threshold via control station are allowed.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN REAL (input) Input value.

IV BOOL (invalid) Input IV is the status/information of the data. If IV = TRUE, the data is invalid.

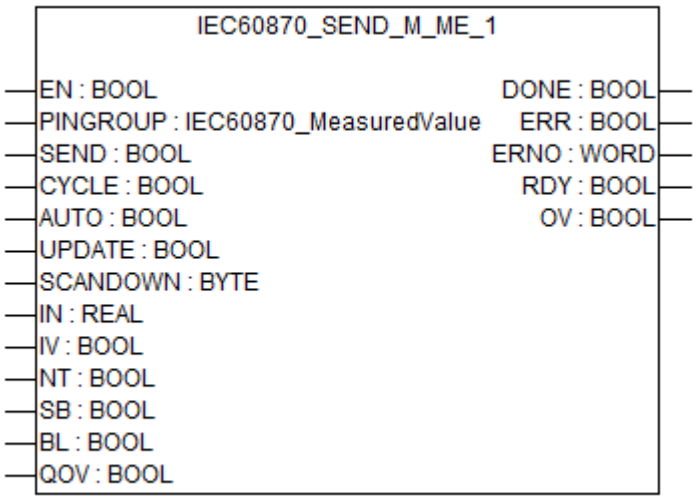
NT BOOL (not topical) Input NT is the status/information of the data. If NT = TRUE, the data is not topical/actual.

SB BYTE (substituted) Input SB is the status/information of the data. If SB = TRUE, the data is substituted.

BL BYTE (blocked) Input BL is the status/information of the data. If BL = TRUE, the data is blocked.

QOV BYTE (overflow/no overflow) Input QOV is the status/information of the data. If QOV = TRUE, the data signals an overflow.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

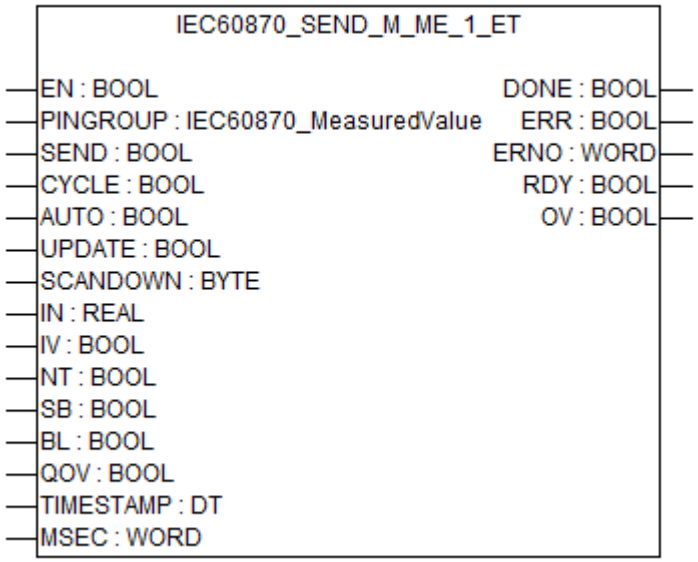
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
         PINGROUP       := SEND_PINGROUP,
         SEND           := SEND_SEND,
         CYCLE          := SEND_CYCLE,
         AUTO           := SEND_AUTO,
         UPDATE         := SEND_UPDATE,
         SCANDOWN       := SEND_SCANDOWN,
         IN             := SEND_IN,
         IV             := SEND_IV,
         NT             := SEND_NT,
         SB             := SEND_SB,
         BL             := SEND_BL,
         QOV            := SEND_QOV);


SEND_DONE               := SEND.DONE;
SEND_ERR                := SEND.ERR;
SEND_ERNO               := SEND.ERNO;
SEND_RDY                := SEND.RDY;
SEND_OV                 := SEND.OV;
```

IEC60870_SEND_M_ME_1_ET



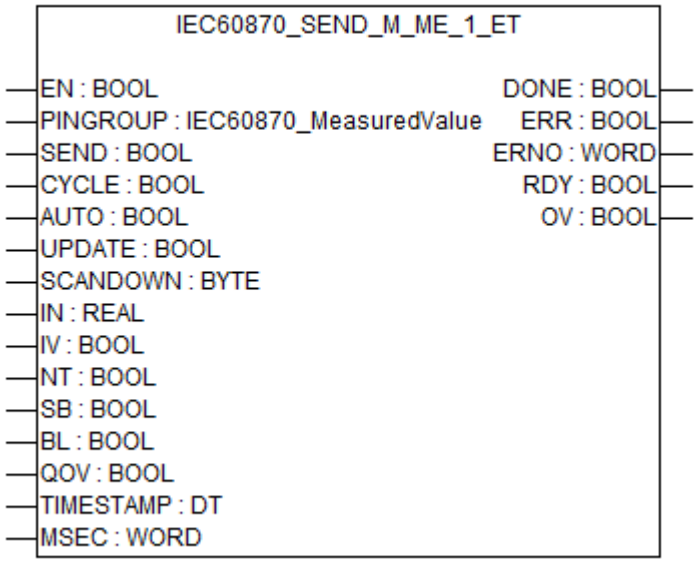
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

UPDATE BOOL (update) If UPDATE = TRUE, updates of threshold via control station are allowed.

SCANDOWN BYTE (scan-down) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN REAL (input) Input value.

IV BOOL (invalid) Input IV is the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT is the status/information of the data. If NT = TRUE, the data is not topical/actual.

SB BYTE (substituted) Input SB is the status/information of the data. If SB = TRUE, the data is substituted.

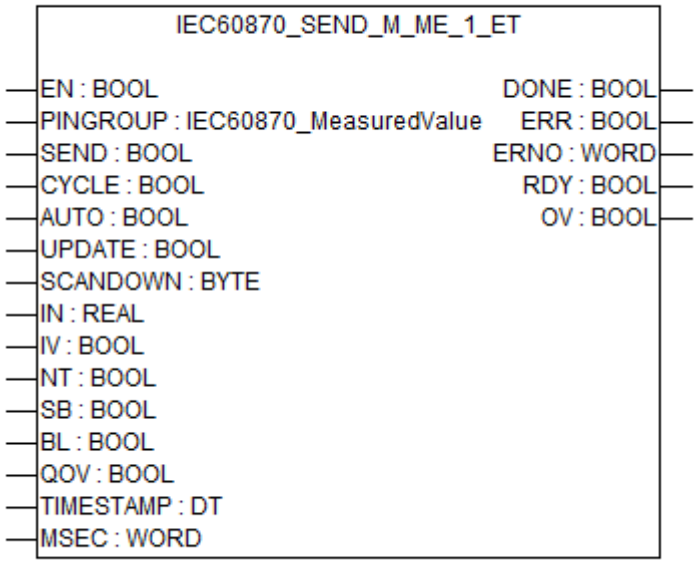
BL BYTE (blocked) Input BL is the status/information of the data. If BL = TRUE, the data is blocked.

QOV BYTE (overflow/no overflow) Input QOV is the status/information of the data. If QOV = TRUE, the data signals an overflow.

TIMESTAMP DT (time-stamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

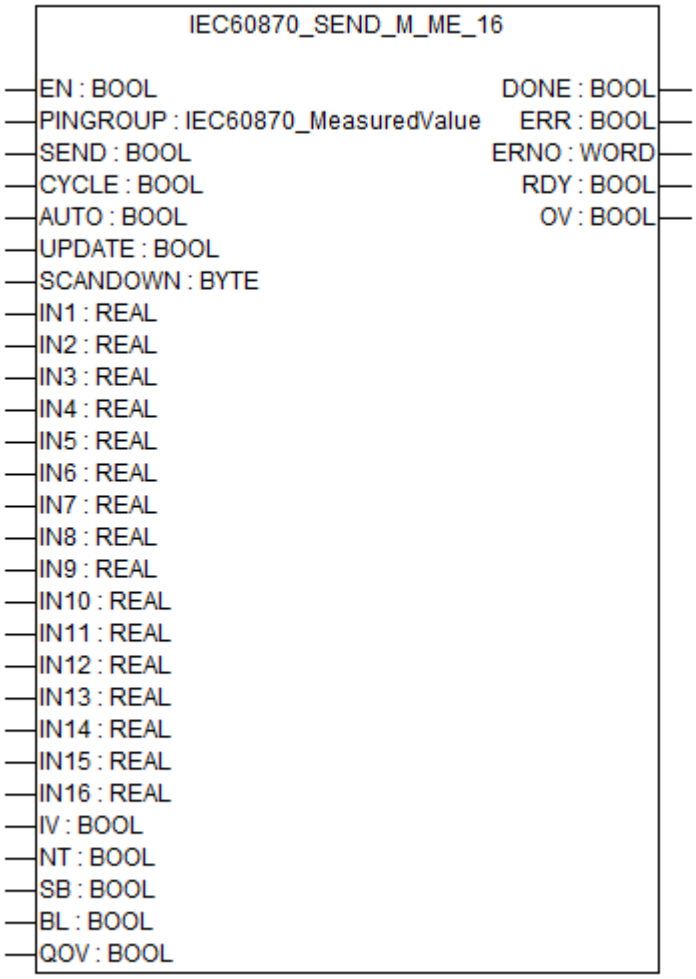
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
         PINGROUP       := SEND_PINGROUP,
         SEND           := SEND_SEND,
         CYCLE          := SEND_CYCLE,
         AUTO           := SEND_AUTO,
         UPDATE         := SEND_UPDATE,
         SCANDOWN       := SEND_SCANDOWN,
         IN             := SEND_IN,
         IV             := SEND_IV,
         NT             := SEND_NT,
         SB             := SEND_SB,
         BL             := SEND_BL,
         QOV            := SEND_QOV,
         TIMESTAMP      := SEND_TIMESTAMP,
         MSEC           := SEND_MSEC);


SEND_DONE               := SEND.DONE;
SEND_ERR                := SEND.ERR;
SEND_ERNO               := SEND.ERNO;
SEND_RDY                := SEND.RDY;
SEND_OV                 := SEND.OV;
```

IEC60870_SEND_M_ME_16



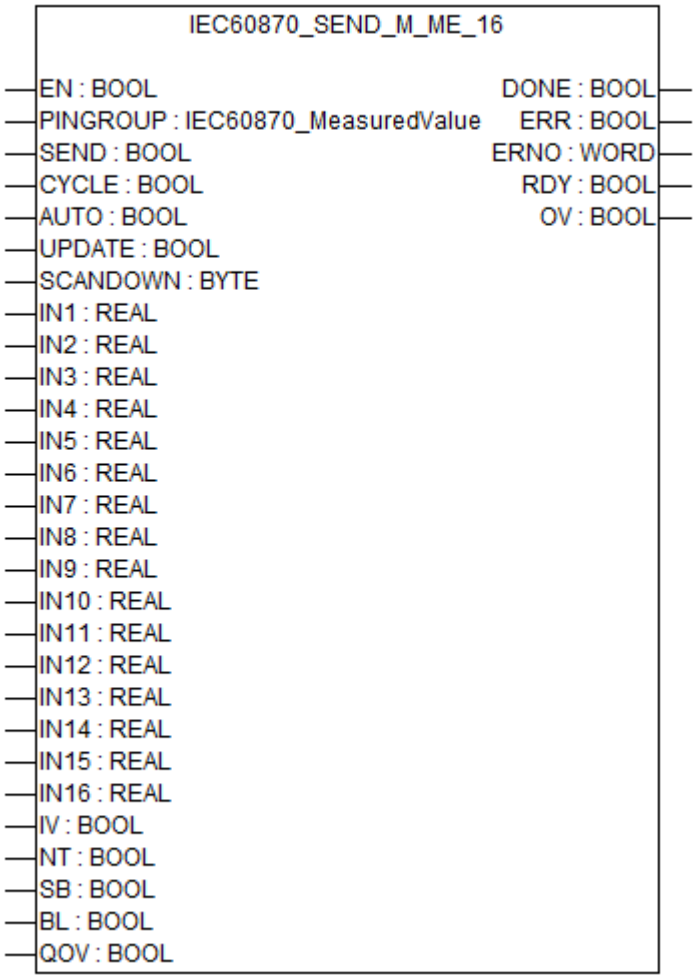
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

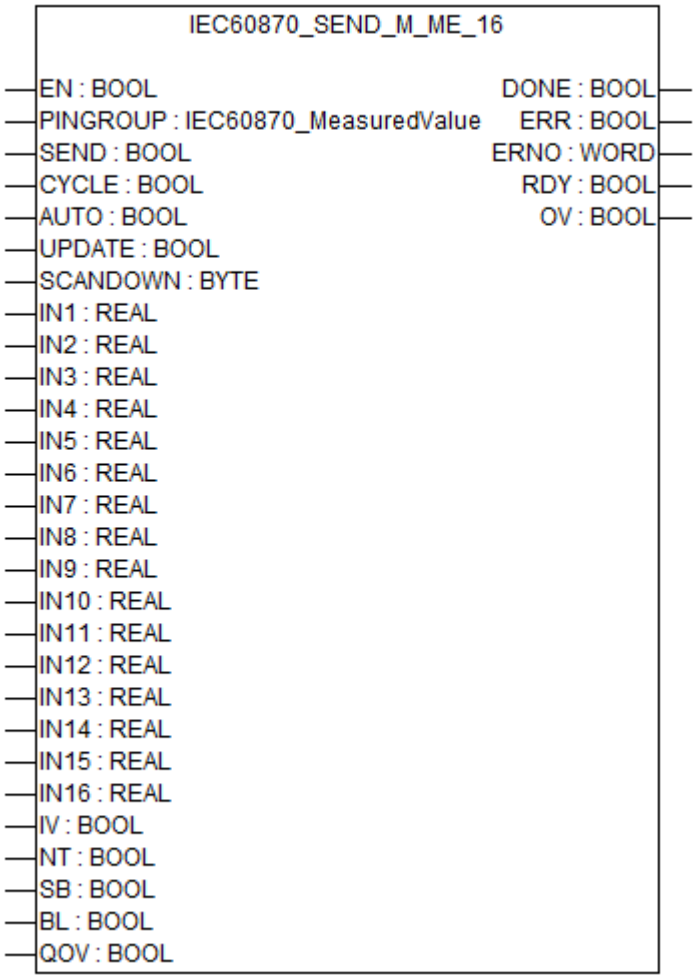
AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

UPDATE BOOL (update) If UPDATE = TRUE, updates of threshold via control station are allowed.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 REAL (input 1...16) Input value for member 1 to 16.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

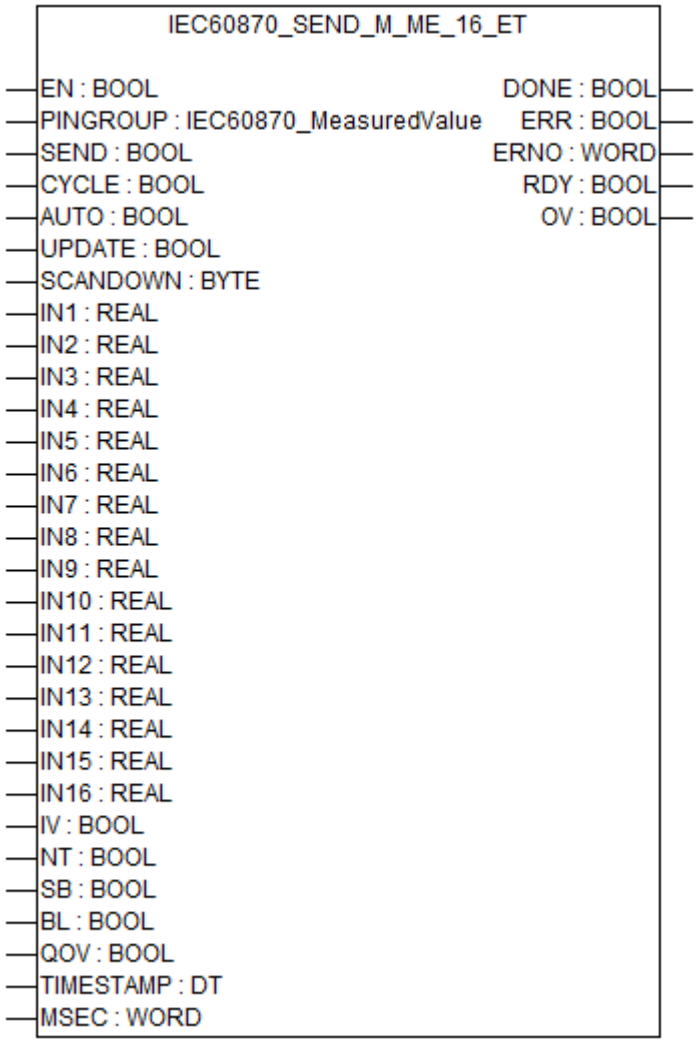
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        UPDATE       := SEND_UPDATE,
        SCANDOWN     := SEND_SCANDOWN,
        IN1          := SEND_IN1,
        IN2          := SEND_IN2,
        IN3          := SEND_IN3,
        IN4          := SEND_IN4,
        IN5          := SEND_IN5,
        IN6          := SEND_IN6,
        IN7          := SEND_IN7,
        IN8          := SEND_IN8,
        IN9          := SEND_IN9,
        IN10         := SEND_IN10,
        IN11         := SEND_IN11,
        IN12         := SEND_IN12,
        IN13         := SEND_IN13,
        IN14         := SEND_IN14,
        IN15         := SEND_IN15,
        IN16         := SEND_IN16)

SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_ME_16_ET



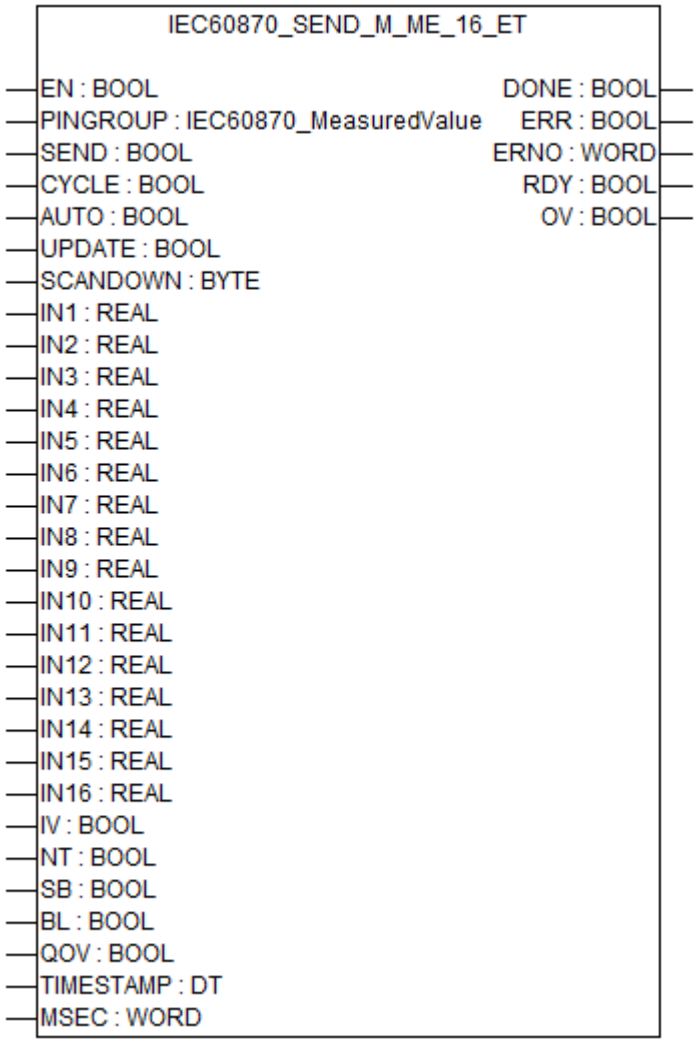
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_MeasuredValue (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

UPDATE BOOL (update) If UPDATE = TRUE, updates of threshold via control station are allowed.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 REAL (input 1...16) Input value for member 1 to 16.

IV BOOL (invalid) Input IV is the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT is the status/information of the data. If NT = TRUE, the data is not topical/actual.

SB BYTE (substituted) Input SB is the status/information of the data. If SB = TRUE, the data is substituted.

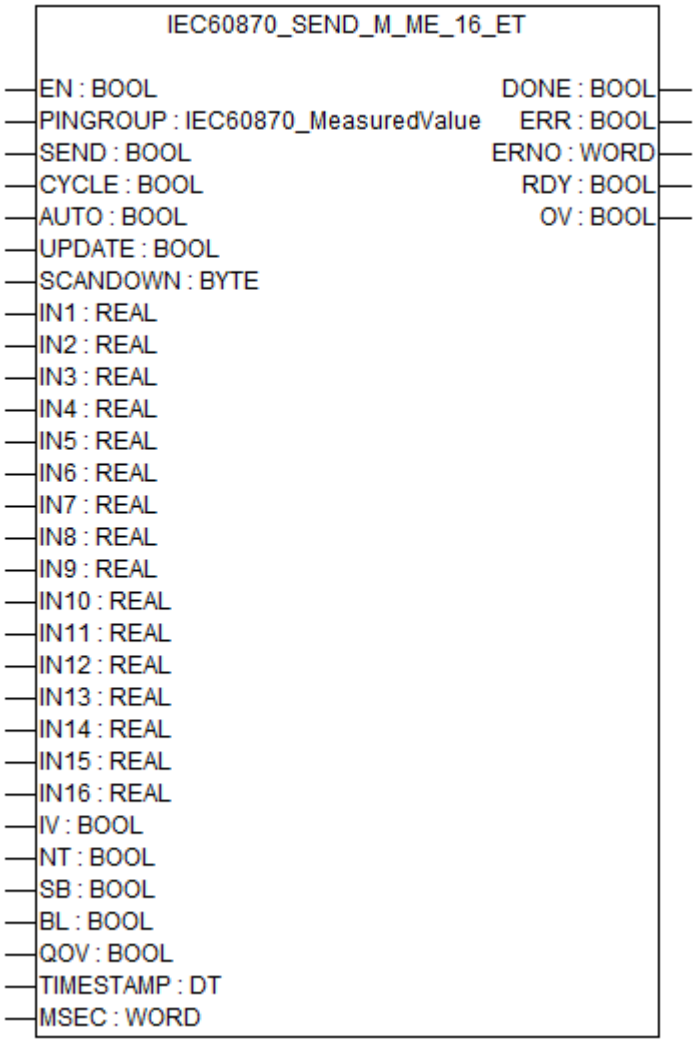
BL BYTE (blocked) Input BL is the status/information of the data. If BL = TRUE, the data is blocked.

QOV BYTE (overflow/no overflow) Input QOV is the status/information of the data. If QOV = TRUE, the data signals an overflow.

TIMESTAMP DT (timestamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

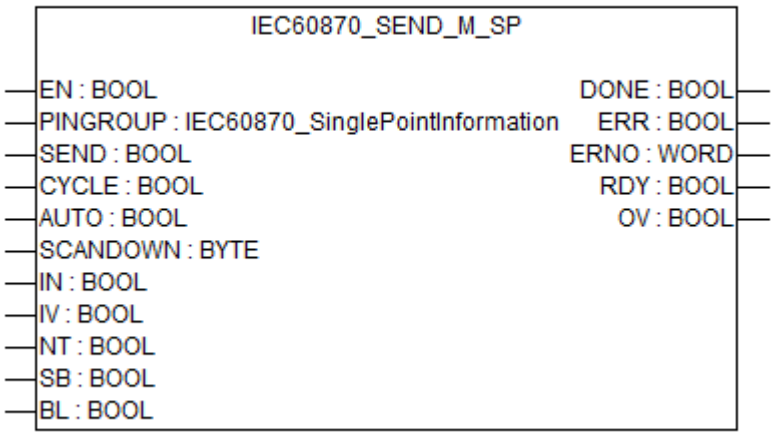
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        UPDATE       := SEND_UPDATE,
        SCANDOWN     := SEND_SCANDOWN,
        IN1          := SEND_IN1,
        IN2          := SEND_IN2,
        IN3          := SEND_IN3,
        IN4          := SEND_IN4,
        IN5          := SEND_IN5,
        IN6          := SEND_IN6,
        IN7          := SEND_IN7,
        IN8          := SEND_IN8,
        IN9          := SEND_IN9,
        IN10         := SEND_IN10,
        IN11         := SEND_IN11,
        IN12         := SEND_IN12,
        IN13         := SEND_IN13,
        IN14         := SEND_IN14,
        IN15         := SEND_IN15,
        IN16         := SEND_IN16,
        IV           := SEND_IV,
        NT           := SEND_NT,
        SB           := SEND_SB,
        BL           := SEND_BL,
        QOV          := SEND_QOV,
        TIMESTAMP    := SEND_TIMESTAMP,
        MSEC         := SEND_MSEC)

SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_SP



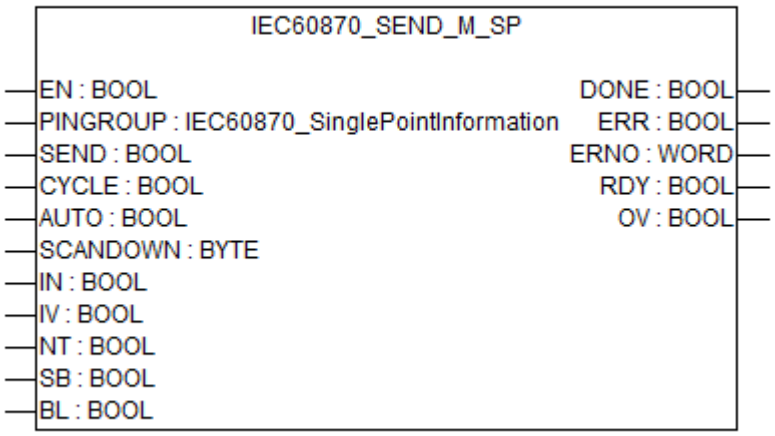
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SinglePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown)

Input SCANDOWN is valid when input CYCLE = TRUE.

On cyclic sending only send within each scandown cycle.

IN BOOL (input)

Input value.

IV BOOL (invalid)

Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical)

Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

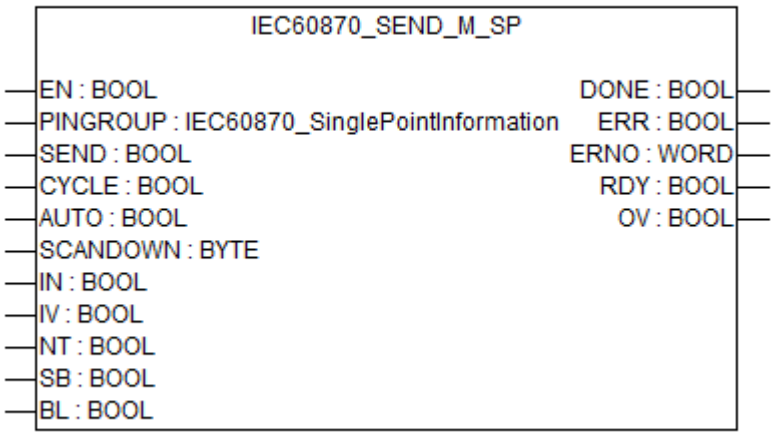
SB BOOL (substituted)

Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked)

Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

DONE BOOL (done)

The output variable DONE shows the processing state of the function block. It shows TRUE when the processing ended or an error took place. This variable DONE should be checked together with the variable ERR. If DONE and ERR are set to TRUE then the output ERNO should be checked to see which error occurred.

ERR BOOL (error)

ERR shows TRUE if the function block was terminated as a consequence of an error, or FALSE if it ended normally.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

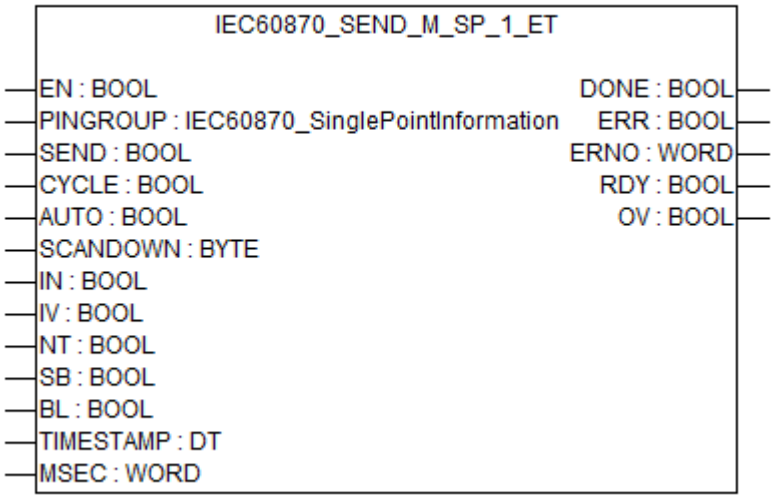
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND_DONE           := SEND.DONE;
SEND_ERR            := SEND.ERR;
SEND_ERNO           := SEND.ERNO;
SEND_RDY            := SEND.RDY;
SEND_OV             := SEND.OV;
```

IEC60870_SEND_M_SP_1_ET



Sending of data messages of the data type according to IEC 60870-5 protocol.

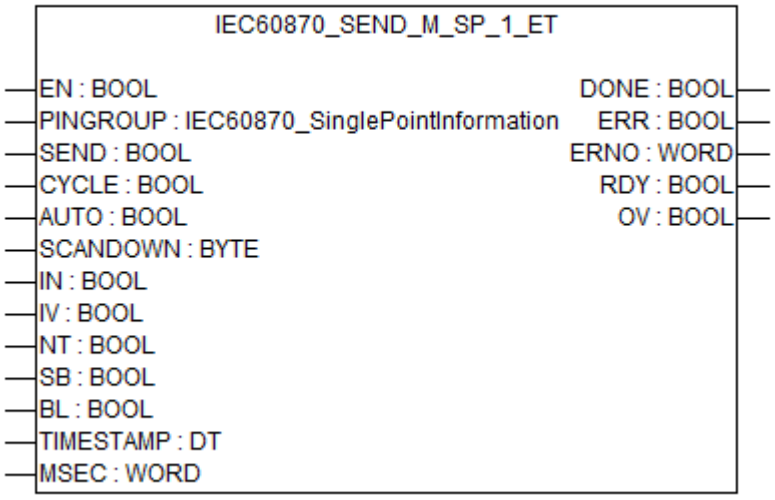
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SinglePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send)

Start a send request on a rising edge.

CYCLE BOOL (cycle)

Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto)

If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scan-down) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN BOOL (input) Input value.

IV BOOL (invalid) Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

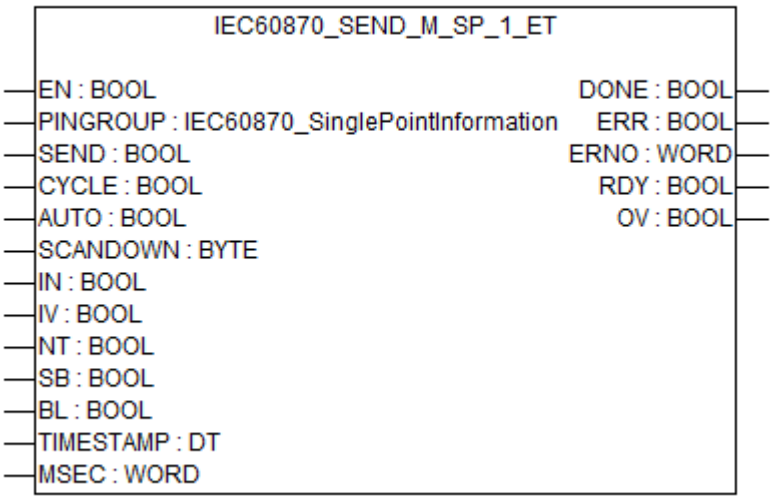
SB BOOL (substituted) Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked) Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

TIMESTAMP DT (time-stamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

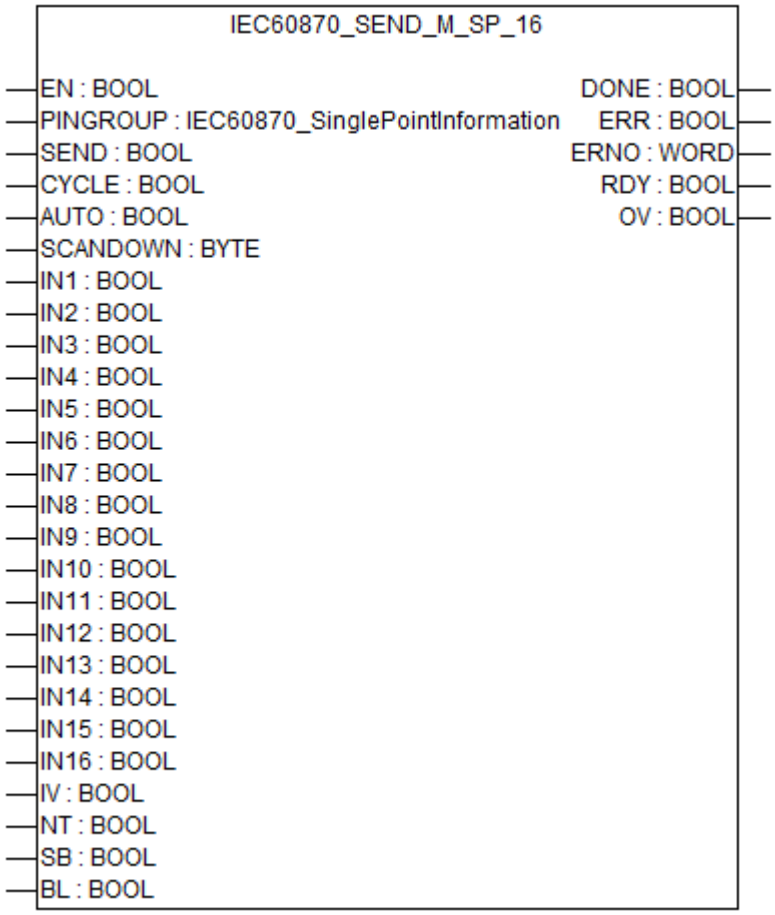
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
         PINGROUP       := SEND_PINGROUP,
         SEND           := SEND_SEND,
         CYCLE          := SEND_CYCLE,
         AUTO           := SEND_AUTO,
         SCANDOWN       := SEND_SCANDOWN,
         IN             := SEND_IN,
         IV             := SEND_IV,
         NT             := SEND_NT,
         SB             := SEND_SB,
         BL             := SEND_BL,
         TIMESTAMP      := SEND_TIMESTAMP,
         MSEC           := SEND_MSEC);


SEND_DONE               := SEND.DONE;
SEND_ERR                := SEND.ERR;
SEND_ERNO               := SEND.ERNO;
SEND_RDY                := SEND.RDY;
SEND_OV                 := SEND.OV;
```

IEC60870_SEND_M_SP_16



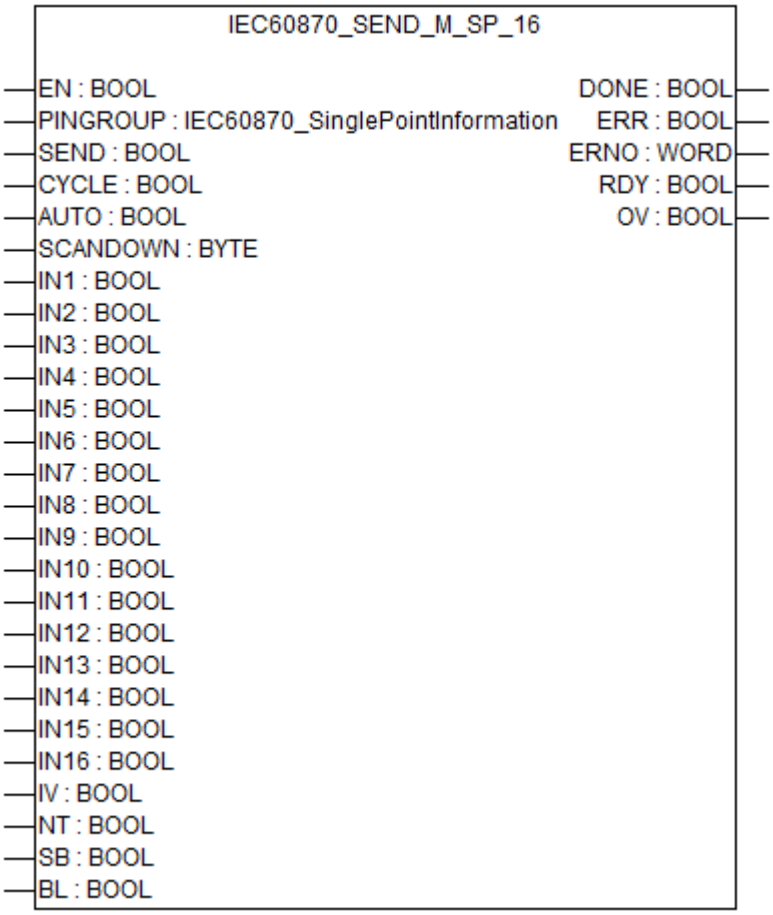
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SinglePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC 60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 BOOL (input 1...16) Input value for member 1 to 16.

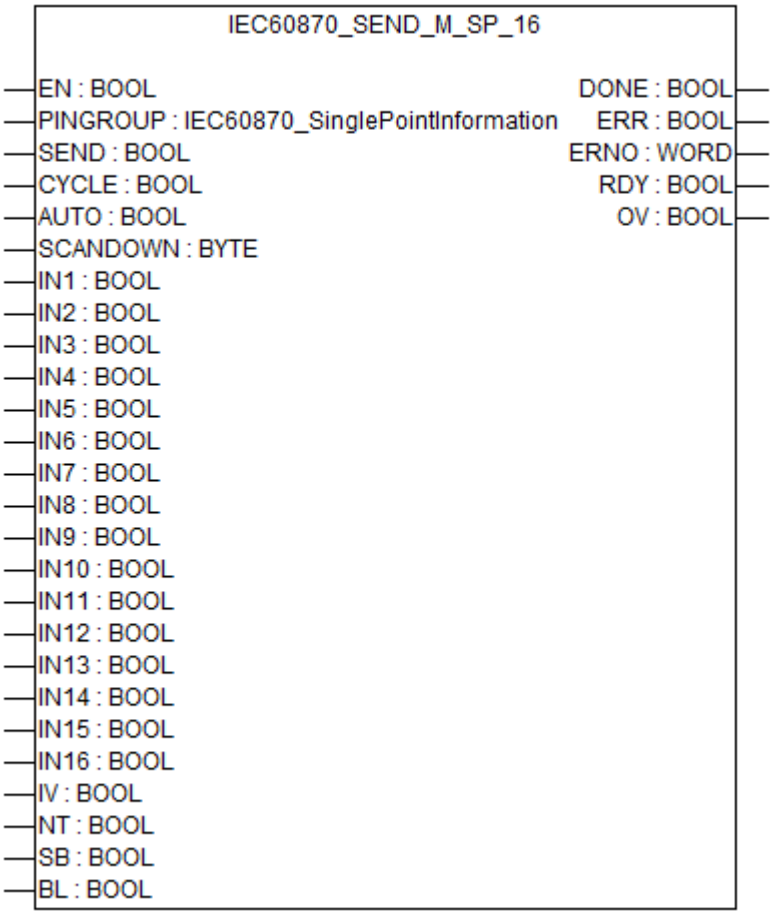
IV BOOL (invalid) Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

SB BOOL (substituted) Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked) Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

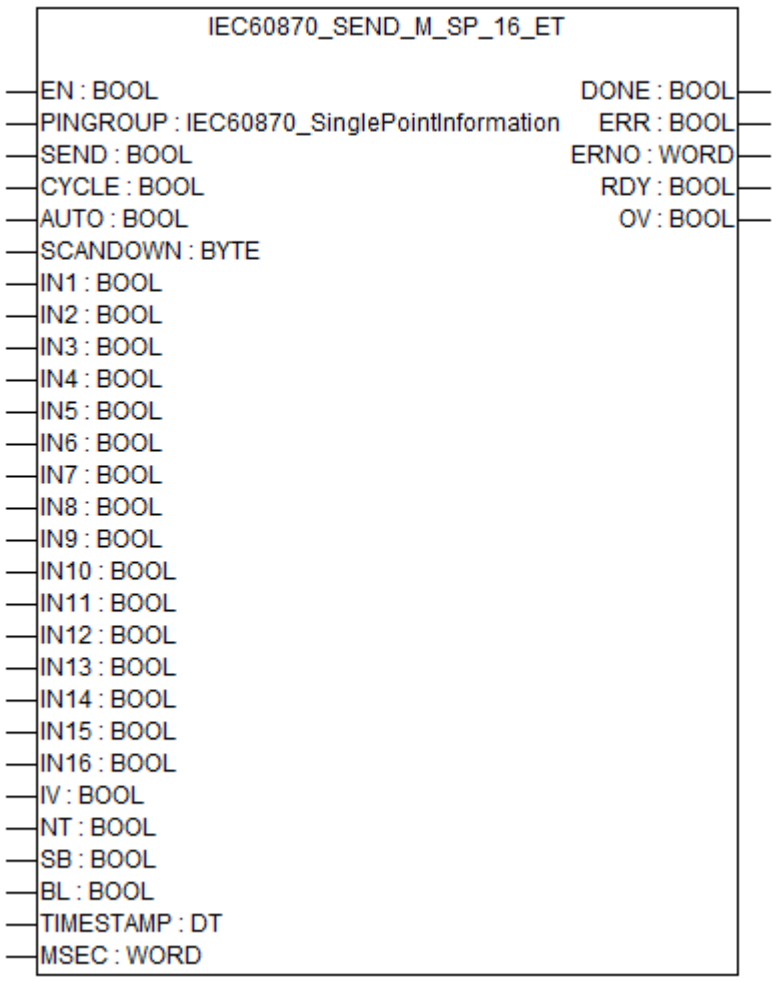
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        SCANDOWN     := SEND_SCANDOWN,
        IN1          := SEND_IN1,
        IN2          := SEND_IN2,
        IN3          := SEND_IN3,
        IN4          := SEND_IN4,
        IN5          := SEND_IN5,
        IN6          := SEND_IN6,
        IN7          := SEND_IN7,
        IN8          := SEND_IN8,
        IN9          := SEND_IN9,
        IN10         := SEND_IN10,
        IN11         := SEND_IN11,
        IN12         := SEND_IN12,
        IN13         := SEND_IN13,
        IN14         := SEND_IN14,
        IN15         := SEND_IN15,
        IN16         := SEND_IN16,
        IN           := SEND_IN,
        IV           := SEND_IV,
        NT           := SEND_NT,
        SB           := SEND_SB,
        BL           := SEND_BL);

SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_M_SP_16_ET



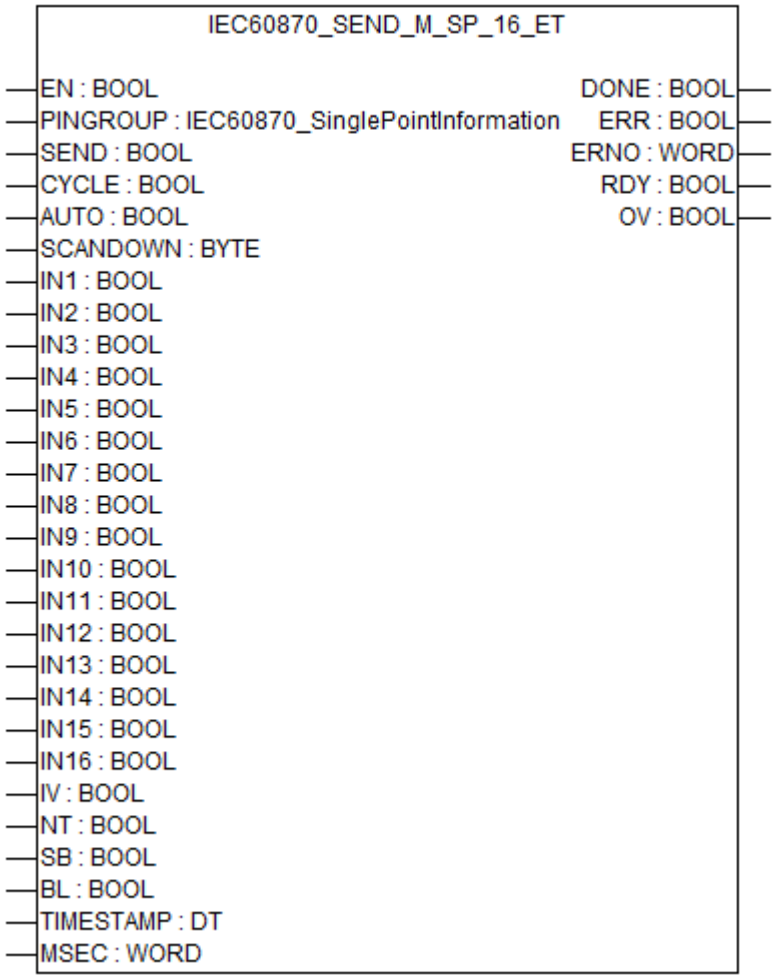
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Data

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP IEC60870_SinglePointInformation (pin group)

At input PINGROUP the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec, see *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*).

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

SEND BOOL (send) Start a send request on a rising edge.

CYCLE BOOL (cycle) Input CYCLE performs a cyclic send each scandown cycle.

AUTO BOOL (auto) If input AUTO = TRUE, each input value change might trigger a sending process.

SCANDOWN BYTE (scandown) Input SCANDOWN is valid when input CYCLE = TRUE.
On cyclic sending only send within each scandown cycle.

IN1 ... IN16 BOOL (input 1...16) Input value for member 1 to 16.

IV BOOL (invalid) Input IV sets the status/information of the data. If IV = TRUE, the data is invalid.

NT BOOL (not topical) Input NT sets the status/information of the data. If NT = TRUE, the data is not topical/actual.

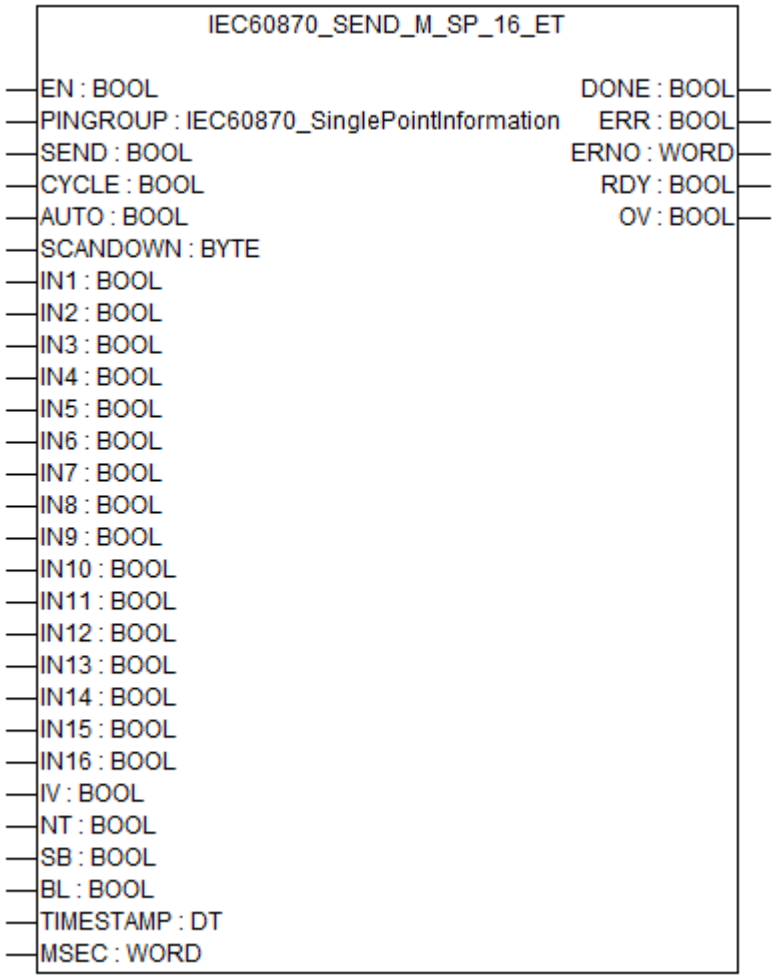
SB BOOL (substituted) Input SB sets the status/information of the data. If SB = TRUE, the data is substituted.

BL BOOL (blocked) Input BL displays the status/information of the data. If BL = TRUE, the data is blocked.

TIMESTAMP DT (timestamp) TIMESTAMP which should be sent with the data.

MSEC WORD (milliseconds) Millisecond part of TIMESTAMP which should be sent with the data.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command has been received if RDY = TRUE.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

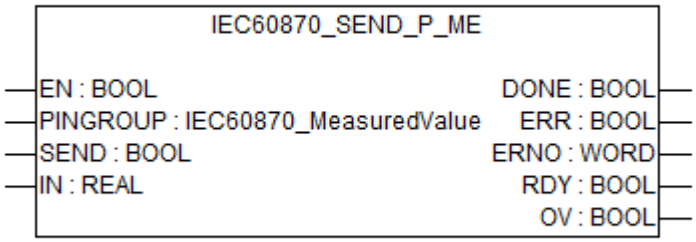
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND   (EN           := SEND_EN,
        PINGROUP     := SEND_PINGROUP,
        SEND         := SEND_SEND,
        CYCLE        := SEND_CYCLE,
        AUTO         := SEND_AUTO,
        SCANDOWN     := SEND_SCANDOWN,
        IN1          := SEND_IN1,
        IN2          := SEND_IN2,
        IN3          := SEND_IN3,
        IN4          := SEND_IN4,
        IN5          := SEND_IN5,
        IN6          := SEND_IN6,
        IN7          := SEND_IN7,
        IN8          := SEND_IN8,
        IN9          := SEND_IN9,
        IN10         := SEND_IN10,
        IN11         := SEND_IN11,
        IN12         := SEND_IN12,
        IN13         := SEND_IN13,
        IN14         := SEND_IN14,
        IN15         := SEND_IN15,
        IN16         := SEND_IN16,
        IN           := SEND_IN,
        IV           := SEND_IV,
        NT           := SEND_NT,
        SB           := SEND_SB,
        BL           := SEND_BL,
        TIMESTAMP    := SEND_TIMESTAMP,
        MSEC         := SEND_MSEC);


SEND_DONE            := SEND.DONE;
SEND_ERR             := SEND.ERR;
SEND_ERNO            := SEND.ERNO;
SEND_RDY             := SEND.RDY;
SEND_OV              := SEND.OV;
```

IEC60870_SEND_P_ME



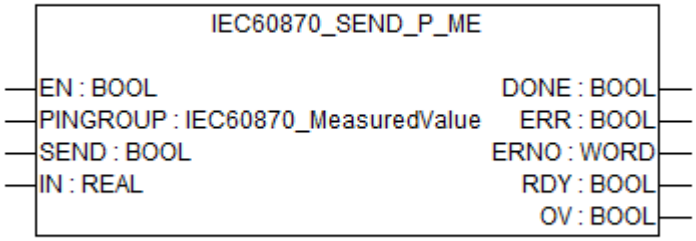
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Parameter_settings

Sending of data messages of the data type according to IEC 60870-5.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

The output RDY indicates the state of the transmission. When new data are received, this is indicated by a change from FALSE to TRUE. This signal is applied for one computation cycle.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PINGROUP

At input PINGROUP (IEC60870_IntegratedTotal) the corresponding data point is set, which gets received by this function block.

The pin group of the data type IEC60870_DoubleCommand corresponds to the defined data type at the global address list due to the Control-/Substation in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

The IEC60870 data is sent/recorded to the associated datapoint which is defined by the address in Automation Builder (address datapoint send has to be the same as datapoint rec.

A pin group can only be assigned to one defined function block! It is not possible to use the pin group twice (e.g. Send and Rec function blocks)!

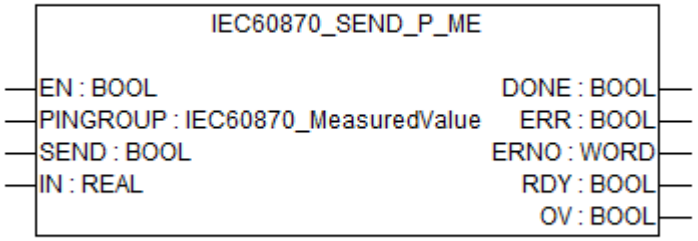
SEND BOOL (send)

Start a send request on a rising edge.

IN REAL (input)

Input value.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command has been received if RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

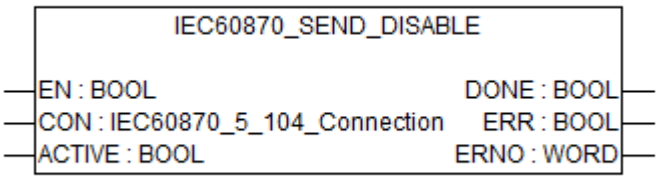
The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND    (EN             := SEND_EN,
PINGROUP      := SEND_PINGROUP,
SEND          := SEND_SEND,
IN            := SEND_IN);


SEND_DONE             := SEND.DONE;
SEND_ERR              := SEND.ERR;
SEND_ERNO             := SEND.ERNO;
SEND_RDY              := SEND.RDY;
SEND_OV               := SEND.OV;
```

IEC60870_SEND_DISABLE



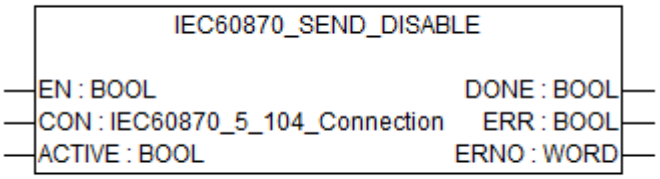
Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6.2 and above
Type	Function block with historical values
Group	Data

IEC60870_SEND_DISABLE can be used to disable sending any data packets from a IEC 60870 station.

No data will be send in monitored direction on an active link any more if IEC60870_SEND_DISABLE has been executed with inputs EN=TRUE and ACTIVE=FALSE.

To select the IEC60870 connection on which no data will be sent any more, the connection needs to be specified on input CON. On the SEND_DISABLED connection no inquiries will be answered with data any more. The normal ACT_CON, ACT_TERM for an inquiry will be still answered to fulfill the normal command state machine. The sending of data can be enabled any time by executing the IEC60870_SEND_DISABLE function block with EN=TRUE, CON=<desired connection> and ACTIVE=TRUE any time. From that point in time data will be send again in monitored direction. This also applies for inquiries, read requests and background scan data.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

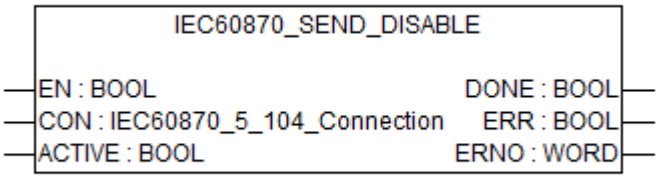
ACTIVE

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

If the function block gets executed with ACTIVE=FALSE the PLC will disable all IEC60870 connections.

If the function block gets executed with ACTIVE=TRUE the PLC will enable all IEC60870 connections if they were disabled before.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

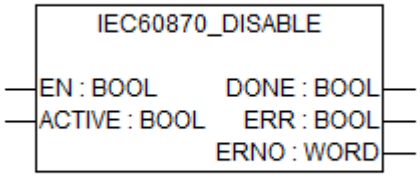
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
SEND_DISABLE (EN  := SEND_DISABLE_EN,
CON    := SEND_DISABLE_CON, ACTIVE  := SEND_DISABLE_ACTIVE;)

SEND_DISABLE_DONE            := SEND_DISABLE.DONE;
SEND_DISABLE_ERR             := SEND_DISABLE.ERR;
SEND_DISABLE_ERNO            := SEND_DISABLE.ERNO;
```

IEC60870_DISABLE



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6.2 and above
Type	Function block with historical values
Group	Data

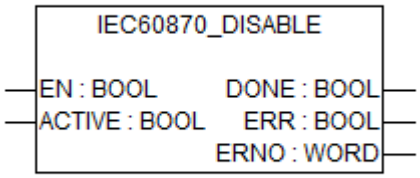
The function block IEC60870_DISABLE can be used to shut down all IEC60870 connections on a PLC.

No further connections can be established after the function block IEC60870_DISABLE has been executed with EN=TRUE and ACTIVE=FALSE.

All already established connections will be aborted. The PLC won’t answer any IEC60870-5-104 connections request any more.

When the IEC60870_DISABLE function block gets executed with EN=TRUE and ACTIVE=TRUE the PLC will listen for new IEC60870-5-104 connections again, and also try to establish connections again.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

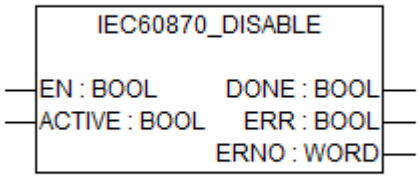
ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

If the function block gets executed with ACTIVE=FALSE the PLC will disable all IEC60870 connections.

If the function block gets executed with ACTIVE=TRUE the PLC will enable all IEC60870 connections if they were disabled before.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

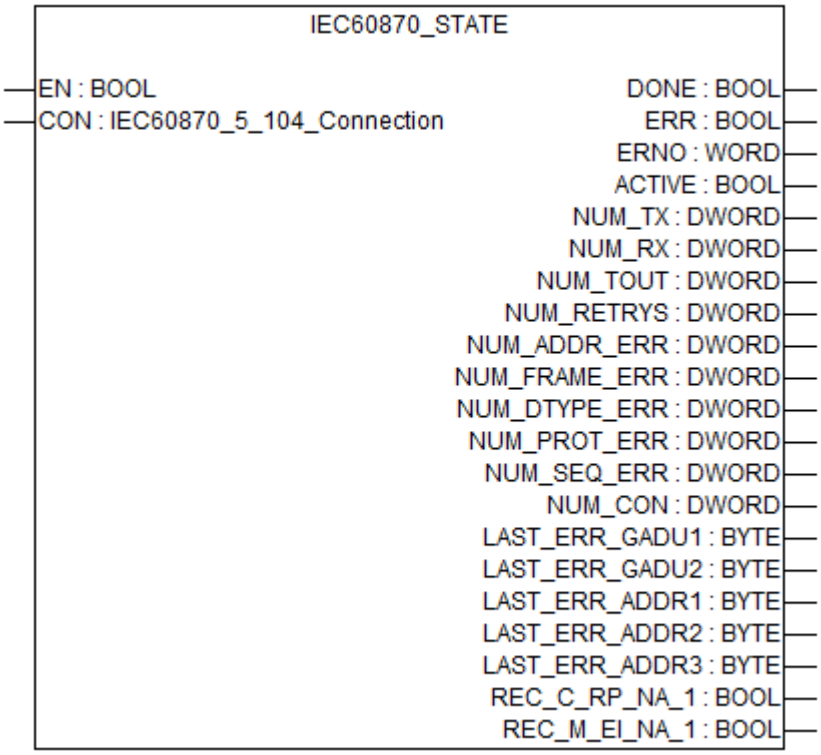
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IEC60870_STATE



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	General

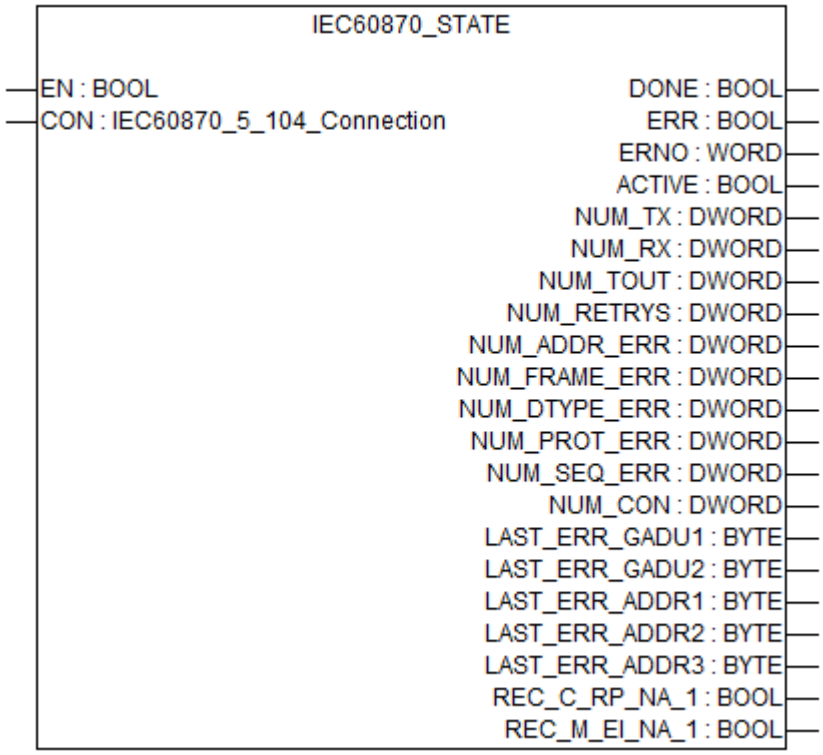
The function block IEC60870_STATE outputs statistically information from the IEC60870-5 protocol.

The function block has to enable to use it. Only if EN = TRUE the specific IEC 60870-5 functions can be activated and used.

For each cyclical processing of the module, the statistical outputs were updated. The actual telecontrol communication via the hardware interface runs in the background independently of the user task.

The data received are available at the corresponding output pins.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

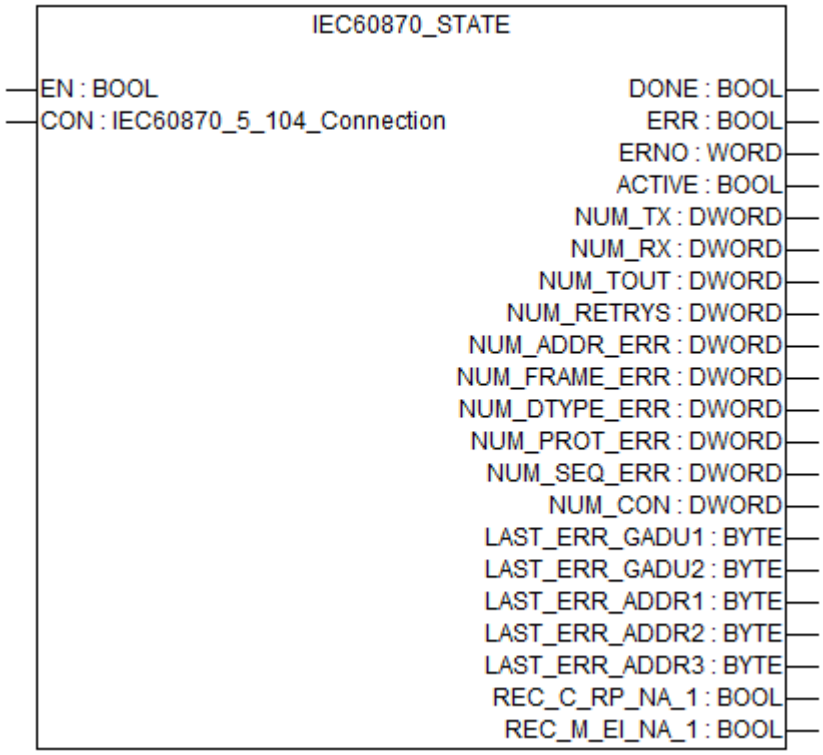
CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACTIVE (active) BOOL

If ACTIVE = TRUE, the communication is established and active.

NUM_TX DWORD (number tx)	Counter of data transmitted.
NUM_RX DWORD (number rx)	Counter of data received.
NUM_TOUT DWORD (number timeout)	Counter of transmission/reception timeouts.
NUM_RETRYS DWORD (number retries)	Counter of transmission/reception retries.
NUM_ADDR_ERR DWORD (number address error)	Counter of address errors.
NUM_FRAME_ERR DWORD (number frameerror)	Counter of frame errors.
NUM_DTYPE_ERR DWORD (number data typeerror)	Counter of data type errors.
NUM_PROT_ERR DWORD (number protocol error)	Counter of protocol errors.
NUM_SEQ_ERR DWORD (number sequence error)	Counter of sequence errors.
NUM_CON DWORD (number connection)	Amount of established connections.
LAST_ERR_GADU1 BYTE (lasterror global address unit 1)	Last Global Address 1 with error.
LAST_ERR_GADU2 BYTE (lasterror global address unit 2)	Last Global Address 2 with error.
LAST_ERR_ADDR1 BYTE (last local error address unit 1)	Last Global Address 1 with error.
LAST_ERR_ADDR2 BYTE (last local error address unit 2)	Last Global Address 2 with error.

LAST_ERR_ADDR3 BYTE (last local error address unit 3) Last Global Address 3 with error.

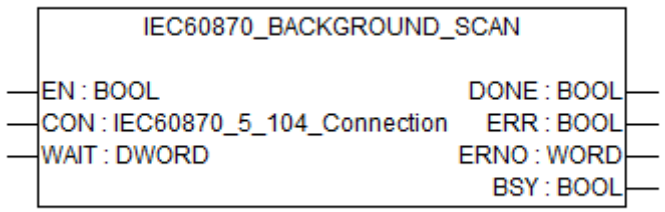
REC_C_RP_NA_1 BOOL (receive command reset process) If REC_C_RP_NA_1 = TRUE, a reset process command has been received.

REC_M_EI_NA_1 BOOL (receive monitored init end) If REC_M_EI_NA_1 = TRUE, an init end has been received after a connection was established.

Function call in ST

```
STATE (EN              := STATE_EN,
CON            := STATE_PINGROUP);


STATE_DONE             := STATE.DONE;
STATE_ERR              := STATE.ERR;
STATE_ERNO             := STATE.ERNO;
STATE_ACTIVE           := STATE.ACTIVE;
STATE_NUM_TX           := STATE.NUM_TX;
STATE_NUM_RX           := STATE.NUM_RX;
STATE_NUM_TOUT         := STATE.NUM_TOUT;
STATE_NUM_RETRYS       := STATE.NUM_RETRYS;
STATE_NUM_ADDR_ERR     := STATE.NUM_ADDR_ERR;
STATE_NUM_FRAME_ERR    := STATE.NUM_FRAME_ERR;
STATE_NUM_DTYPE_ERR    := STATE.NUM_DTYPE_ERR;
STATE_NUM_PROT_ERR     := STATE.NUM_PROT_ERR;
STATE_NUM_SEQ_ERR      := STATE.NUM_SEQ_ERR;
STATE_NUM_CON          := STATE.NUM_CON;
STATE_LAST_ERR_GADU1   := STATE.LAST_ERR_GADU1;
STATE_LAST_ERR_GADU2   := STATE.LAST_ERR_GADU2;
STATE_LAST_ERR_ADDR1   := STATE.LAST_ERR_ADDR1;
STATE_LAST_ERR_ADDR2   := STATE.LAST_ERR_ADDR2;
STATE_LAST_ERR_ADDR3   := STATE.LAST_ERR_ADDR3;
STATE_REC_C_RP_NA_1    := STATE.REC_C_RP_NA_1;
STATE_REC_M_EI_NA_1    := STATE.REC_M_EI_NA_1;
```

IEC60870_BACKGROUND_SCAN



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6.x

Parameter	Value
Type	Function block with historical values
Group	General

The background scan allows a monitoring station to report all monitored data without any request.

The data will be sent with a defined delay, so also slow monitoring stations can be handled. The data the monitored station needs to send must be stored within the protocol stack by utilizing SEND_ function blocks without any active send reason.

Then on any event the monitored station may start the background scan by executing the IEC60870_BACKGROUND_SCAN function block.

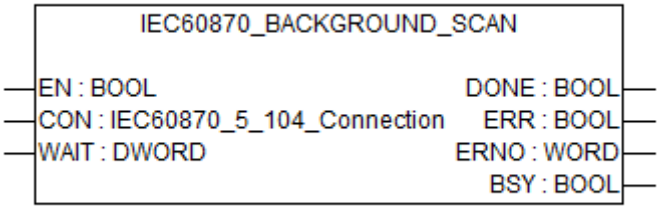
If an active connection exists, the monitored station will then send all monitored data to the monitoring station with a send reason “background scan (2)”.

The data types being used are the same as for a general interrogation.

The receiving station must be able to handle the send reason. For AC500 this send reason is supported since V2.6 and V3.0.

Also data send actively before will be resend by a background scan. If a monitored station sends actively the monitoring station can only distinguish the received data by controlling the send reason value.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

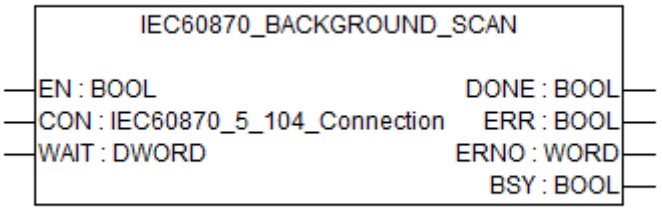
The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

WAIT

This input value denotes a wait time between sending each information object. This allows overcoming any congestion on either the link or the opposite station. The time is given in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

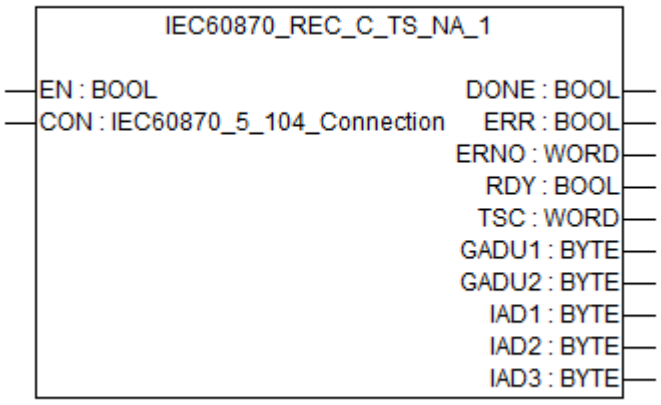
BSY (busy)

The output will be TRUE as long as a background scan is already running.

Function call in ST

```
IEC60870_BACKGROUND_SCAN
                    EN                                  := TRUE,
                    CON                        := ADR(Measured_value),
                    WAIT                             := FORMAT_HEX_ASC,
                    SEPARATOR     := ';'
);

BACKGROUND_SCAN_DONE                      := BACKGROUND_SCAN.DONE;
BACKGROUND_SCAN_ERR                    := BACKGROUND_SCAN.ERR;
BACKGROUND_SCAN_ERNO                  := BACKGROUND_SCAN.ERNO;
BACKGROUND_SCAN_BSY                    := BACKGROUND_SCAN.BSY;
```

IEC60870_REC_C_TS_NA_1



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	System Information

The test command within IEC60870-5 is used to test the whole loop from monitoring station to the application on the monitored station.

For the AC500 this implies to have an IEC1131 action on receiving a test command in the user application. So the test request must be received by the user application and must also be answered by the user application.

The interface for this is build from 3 function blocks:

- IEC60870_SEND_C_TS_NA_1_ACT to request a test and verify if the correct response was received.

2 function blocks on the tested end.

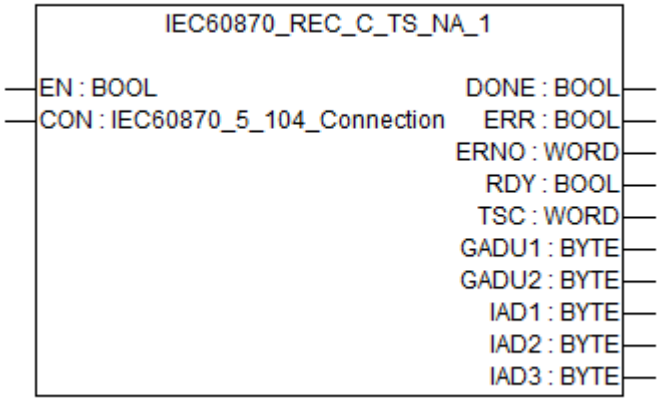
- IEC60870_REC_C_TS_NA_1 to receive an actual test request.
- IEC60870_SEND_C_TS_NA_1_ACTCON to answer an actual test request

For a successful test the testing station needs to request a test by sending a C_TS_NA_1 ACT request to the tested station.

The tested station needs to detect the test request and answer by sending a C_TS_NA_1 ACTCON confirmation with the data received.

The protocol implementation will make sure the correct time stamps are being send, but for the test sequence and address the tested station is responsible

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

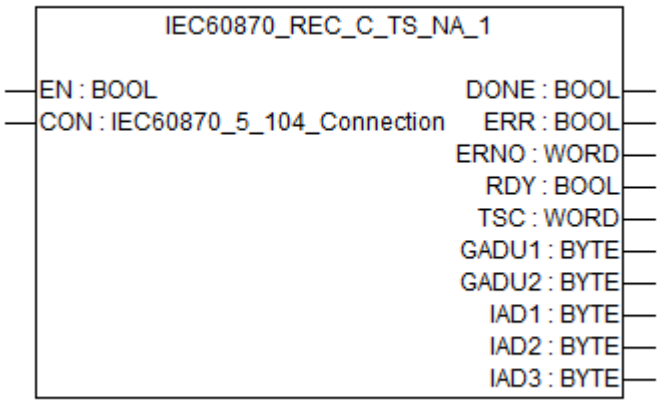
CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command was received if the output RDY = TRUE.

TSC WORD (Test Sequence Counter)

The requesting station may choose any value of TSC. The TSC in the response shall match the request.

The following 5 outputs provide the address the testing station requested.

The tested station needs to answer with this address.

GADU1 (Common Address of ASDU Byte 1)

GADU2 (Common Address of ASDU Byte 2)

IAD1 (Information Object Address Byte 1)

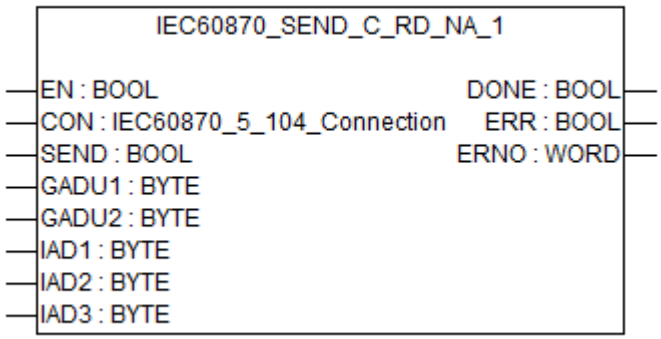
IAD2 (Information Object Address Byte 2)

IAD3 (Information Object Address Byte 3)

Function call in ST

```
REC_C_TS_NA_1 (EN  := REC_C_TS_NA_1_EN,
CON := REC_C_TS_NA_1_CON);


REC_C_TS_NA_1_DONE            := REC_C_TS_NA_1.DONE;
REC_C_TS_NA_1_ERR             := REC_C_TS_NA_1.ERR;
REC_C_TS_NA_1_ERNO            := REC_C_TS_NA_1.ERNO;
REC_C_TS_NA_1_RDY             := REC_C_TS_NA_1.RDY;
REC_C_TS_NA_1_TSC             := REC_C_TS_NA_1.TSC;
REC_C_TS_NA_1_GADU1           := REC_C_TS_NA_1.GADU1;
REC_C_TS_NA_1_GADU2           := REC_C_TS_NA_1.GADU2;
REC_C_TS_NA_1_IAD1            := REC_C_TS_NA_1.IAD1;
REC_C_TS_NA_1_IAD2            := REC_C_TS_NA_1.IAD2;
REC_C_TS_NA_1_IAD3            := REC_C_TS_NA_1.IAD3;
```

IEC60870_SEND_C_RD_NA_1



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6
Type	Function block with historical values
Group	Sytem Information

The read command C_RD_NA_1 is used to request data from a monitoring station.

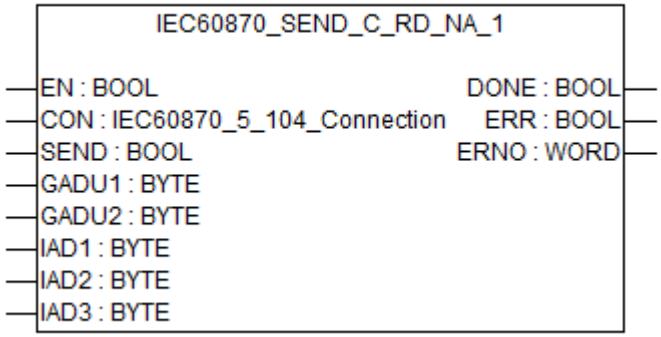
Only monitored data can be requested.

The monitored station will answer a read command with a send of a monitored data type and the send reason being “requested”.

The data that will be send is the last value provided to the protocol with either an active send request, or an inactive send (no send reason active on execution of send function block).

Any data send by normal active sends can be used on the same data point as requested. To detect the requested data, the send reason must be checked. See also background scan or interrogation.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

A true in conjunction with EN being TRUE will send a C_RD_NA_1 requesting the information object with the given address (GADU1 ,GADU2, IAD1, IAD2, IAD3)

GADU1 (Common Address of ASDU Byte 1)

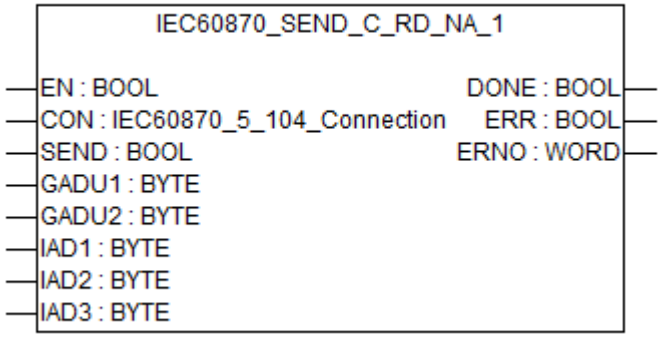
GADU2 (Common Address of ASDU Byte 2)

IAD1 (Information Object Address Byte 1)

IAD2 (Information Object Address Byte 2)

IAD3 (Information Object Address Byte 3)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

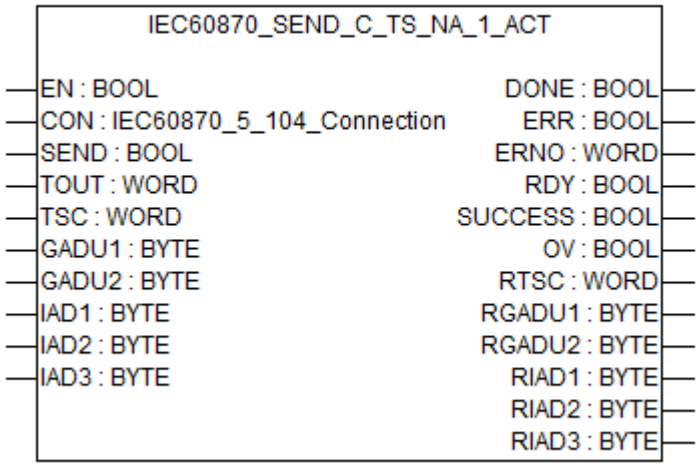
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

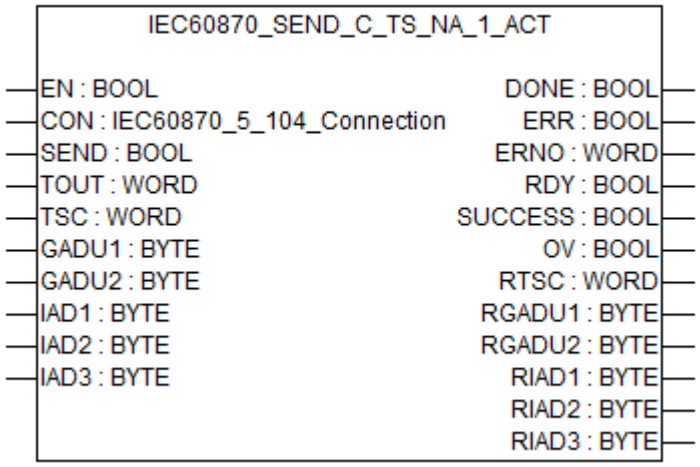
```
SEND_C_RD_NA_1 (EN              := SEND_C_RD_NA_1CON_EN,
CON             := SEND_C_RD_NA_1_CON,  GADU1  :=
SEND_C_RD_NA_1_GADU1,
GADU2  := SEND_C_RD_NA_1_GADU2,
IAD1  := SEND_C_RD_NA_1_IAD1, IAD2  := SEND_C_RD_NA_1_IAD2,
IAD3  := SEND_C_RD_NA_1_IAD3;)


SEND_C_RD_NA_1_DONE             := SEND_C_RD_NA_1.DONE;
SEND_C_RD_NA_1_ERR              := SEND_C_RD_NA_1.ERR;
SEND_C_RD_NA_1_ERNO             := SEND_C_RD_NA_1.ERNO;
```

IEC60870_SEND_C_TS_NA_1_ACT



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6
Type	Function block with historical values
Group	System Information

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

Start a send request on a rising edge.

TOUT WORD (timeout)

Timeout in cycles until ACTERM is expected; 0 = infinite.

TSC WORD (Test Sequence Counter)

The requesting station may choose any value of TSC. The TSC in the response shall match the request.

GADU1 (Common Address of ASDU Byte 1)

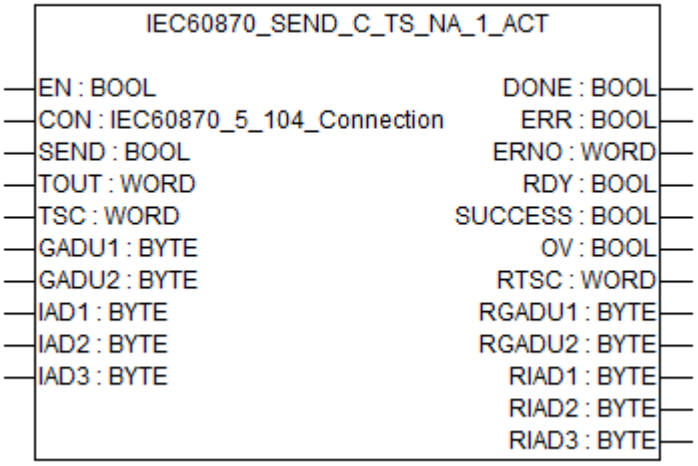
GADU2 (Common Address of ASDU Byte 2)

IAD1 (Information Object Address Byte 1)

IAD2 (Information Object Address Byte 2)

IAD3 (Information Object Address Byte 3)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready) A command was send if the output RDY = TRUE.

SUCCESS BOOL (receive monitored init end) The test sequence finished successful. The matching answer was received within the specified timeout.

OV BOOL (overrun) Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

RTSC BOOL Received test sequence counter, this must match the requested TSC (Input)

RGADU1 BYTE Received adress must match the requested adress at input.

RGADU2 BYTE Received adress must match the requested adress at input.

RIAD1 BYTE Received adress must match the requested adress at input.

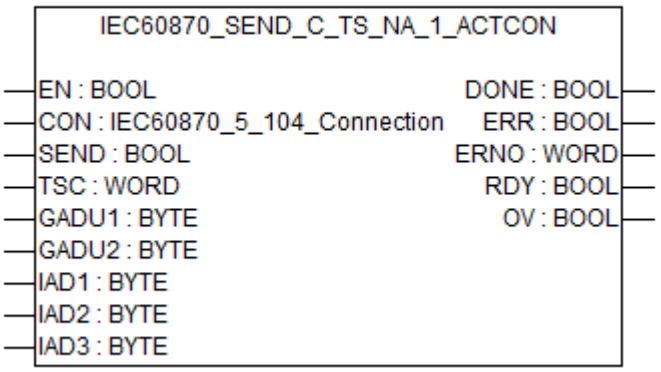
RIAD2 BYTE Received adress must match the requested adress at input.

RIAD3 BYTE Received adress must match the requested adress at input.

Function call in ST

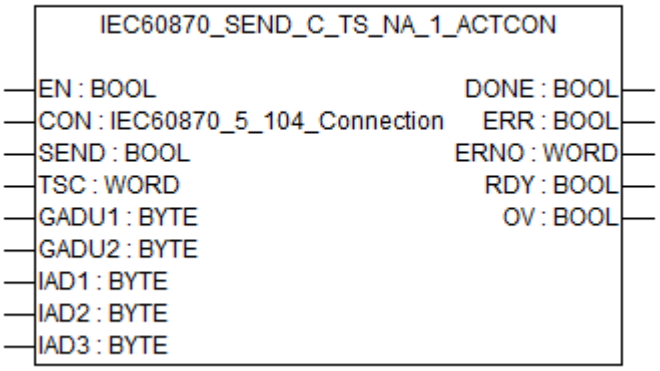
```
SEND_C_TS_NA_1_ACT (EN              := SEND_C_TS_NA_1_ACT_EN,
CON             := SEND_C_TS_NA_1_ACT_CON, SEND  :=
SEND_C_TS_NA_1_ACT_SEND;
TOUT   := SEND_C_TS_NA_1_ACT_TOUT,
 TSC  := SEND_C_TS_NA_1_ACT_TSC, GADU1  := SEND_C_TS_NA_1_ACT_GADU1,
GADU2  := SEND_C_TS_NA_1_ACT_GADU2,
IAD1  := SEND_C_TS_NA_1_ACT_IAD1, IAD2  := SEND_C_TS_NA_1_ACT_IAD2,
IAD3  := SEND_C_TS_NA_1_ACT_IAD3;)


SEND_C_TS_NA_1_ACT_DONE             := SEND_C_TS_NA_1_ACT.DONE;
SEND_C_TS_NA_1_ACT_ERR              := SEND_C_TS_NA_1_ACT.ERR;
SEND_C_TS_NA_1_ACT_ERNO             := SEND_C_TS_NA_1_ACT.ERNO;
SEND_C_TS_NA_1_ACT_TOUT                   := SEND_C_TS_NA_1_ACT.TOUT;
SEND_C_TS_NA_1_ACT_TSC              := SEND_C_TS_NA_1_ACT.TSC;
SEND_C_TS_NA_1_ACT_GADU1            := SEND_C_TS_NA_1_ACT.GADU1;
SEND_C_TS_NA_1_ACT_GADU2            := SEND_C_TS_NA_1_ACT.GADU2;
SEND_C_TS_NA_1_ACT_IAD1             := SEND_C_TS_NA_1_ACT.IAD1;
SEND_C_TS_NA_1_ACT_IAD2             := SEND_C_TS_NA_1_ACT.IAD2;
SEND_C_TS_NA_1_ACT_IAD3             := SEND_C_TS_NA_1_ACT.IAD3;
```

IEC60870_SEND_C_TS_NA_1_ACTCON



Parameter	Value
Included in library	IEC60870_AC500_V20.lib
Available as of firmware	V2.6
Type	Function block with historical values
Group	Sytem Information

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CON IEC60870_5_104_Connection (connection)

At input CON the corresponding connection is set to gather information.

The data type of the type IEC60870_5_104_Connection corresponds to the defined Control-/Substation name in Automation Builder, see ↳ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*.

After the declaration in Automation Builder the data type is available at the global variables constants list.

SEND BOOL (send)

Start a send request on a rising edge.

> *The following values should be connected to the associated ↳ Chapter 1.5.4.18.2.39 “IEC60870_REC_C_TS_NA_1” on page 1488 function block.*

TSC WORD (Test Sequence Counter)

The requesting station may choose any value of TSC. The TSC in the response shall match the request.

GADU1 (Common Address of ASDU Byte 1)

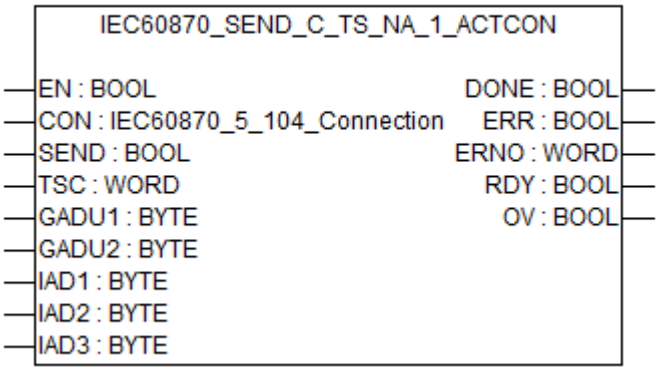
GADU2 (Common Address of ASDU Byte 2)

IAD1 (Information Object Address Byte 1)

IAD2 (Information Object Address Byte 2)

IAD3 (Information Object Address Byte 3)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RDY BOOL (ready)

A command was send if the output RDY = TRUE.

OV BOOL (overrun)

Overrun was detected if the output OV = TRUE.

The output OV indicates by changing from FALSE to TRUE, that the transmission requests are coming too fast, i.e. the task cycle time is configured as too fast.

The output always has to be considered together with output RDY.

Function call in ST

```
SEND_C_TS_NA_1_ACT (EN  := SEND_C_TS_NA_1_ACTCON_EN,
CON    := SEND_C_TS_NA_1_ACTCON_CON, SEND  :=
SEND_C_TS_NA_1_ACTCON_SEND,
 TSC  := SEND_C_TS_NA_1_ACTCON_TSC, GADU1  :=
SEND_C_TS_NA_1_ACTCON_GADU1,
GADU2  := SEND_C_TS_NA_1_ACTCON_GADU2,
IAD1  := SEND_C_TS_NA_1_ACTCON_IAD1, IAD2  :=
SEND_C_TS_NA_1_ACTCON_IAD2,
IAD3  := SEND_C_TS_NA_1_ACTCON_IAD3;)


SEND_C_TS_NA_1_ACTCON_DONE            := SEND_C_TS_NA_1_ACTCON.DONE;
SEND_C_TS_NA_1_ACTCON_ERR             := SEND_C_TS_NA_1_ACTCON.ERR;
SEND_C_TS_NA_1_ACTCON_ERNO            := SEND_C_TS_NA_1_ACTCON.ERNO;
SEND_C_TS_NA_1_ACTCON_TSC             := SEND_C_TS_NA_1_ACTCON.TSC;
SEND_C_TS_NA_1_ACTCON_GADU1           := SEND_C_TS_NA_1_ACTCON.GADU1;
SEND_C_TS_NA_1_ACTCON_GADU2           := SEND_C_TS_NA_1_ACTCON.GADU2;
SEND_C_TS_NA_1_ACTCON_IAD1            := SEND_C_TS_NA_1_ACTCON.IAD1;
SEND_C_TS_NA_1_ACTCON_IAD2            := SEND_C_TS_NA_1_ACTCON.IAD2;
SEND_C_TS_NA_1_ACTCON_IAD3            := SEND_C_TS_NA_1_ACTCON.IAD3;
```

1.5.4.19 Internal system library

Library file name: **SysInt_AC500_Vx.lib**

The internal system library contains all generally applicable function blocks for the system, i.e. function blocks for general system diagnosis functions or system information. The only exception is the "Data Storage" subgroup which contains special FLASH and memory card function blocks. Using these function blocks, data can be either stored to the Flash memory or to the memory card, as desired.

When creating a new project, the library is automatically included to the project.

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

1.5.4.19.1 Structure of the file USRDATXX.DAT on the memory card

Depending on the AC500 CPU type, the data are stored to the memory card in the following directory:

Table 88: Example

AC500 CPU	Directory	File
PM591	..\UserData\PM591\UserDat	USRDATxx.dat

An memory card must be inserted in the AC500!

A maximum of 100 files (USRDAT0.DAT...USRDAT99.DAT) can be stored in one directory. Each data file USRDATxx.dat can be divided into individual sectors, if necessary. The "sector label" enclosed in square brackets (such as [Sector_01]<CR><LF>) indicates the start of the sector. Within a sector, data are saved as data sets in ASCII format. The individual elements of a data set are automatically separated by semicolon. Each data set is terminated with <CR><LF> (0dhex, 0ahex).

This allows to directly import/export the data files from/to EXCEL. The data files can be viewed and edited using a standard ASCII editor (such as Notepad).

When saving / loading the data files, the following rules have to be observed:

- Writing on a non-existent file creates that file prior to the first write access.
- Data sets within a sector must always have the same number of values.
- Data sets in different sectors can have a different number of values.
- The values of a data set must have the same data format (BYTE, WORD, INT,..).
- A sector can have data sets with different data format. (Warning: The user must know the structure of the data when reading them.)
- The data sets are always appended to the end of the file when writing them.
- Searching for a "sector label" within a file is possible when reading it.
- Data sets can be read starting from a particular "sector label".
- A particular data set of a sector cannot be read or written.
- If you want to read each data set individually, a "sector label" must be inserted before each data set.
- Reading and writing the data with help of the user program is done with the function blocks SD_READ and SD_WRITE.
- The values of a data set must be available in variables successively arranged in the PLC (e.g. ARRAY, STRING, %M area).
- A data file can be deleted with help of the PLC program.
- Individual data sets and/or sectors cannot be deleted with the user program. This has to be done on the PC using an ASCII editor such as Notepad.

Data file example 1

Data file USRDAT5.dat without sectors:

-> 5 data sets, each with 10 DINT values:

600462;430;506;469;409;465;466;474;476;-1327203

600477;446;521;484;425;480;482;490;491;-1327187

600493;461;537;499;440;496;497;505;507;-1327172

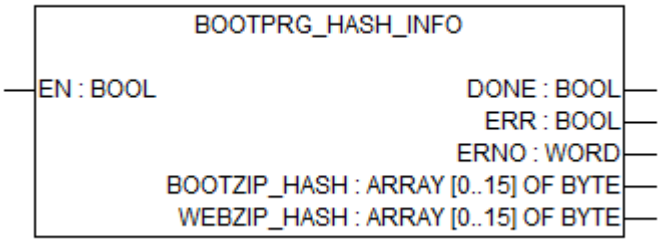
600508;477;552;515;456;511;513;521;522;-1327156

600524;492;568;530;471;527;528;536;538;-1327141

Data file example 2	Data file USRDAT7.dat with sectors: -> 3 sectors, each with 3 data sets and 10 DINT values per data set: [Sector_01] 610439;10408;10483;10446;10387;10442;10444;10452;10453;-1317225 610455;10423;10499;10462;10402;10458;10460;10467;10469;-1317209 610476;10445;10520;10483;10424;10479;10481;10489;10490;-1317188 [Sector_02] 610570;10539;10614;10577;10518;10573;10575;10583;10584;-1317094 610585;10554;10630;10592;10533;10589;10591;10598;10600;-1317078 610602;10571;10646;10609;10550;10605;10607;10615;10616;-1317062 [Sector_03] 610701;10670;10746;10708;10649;10704;10706;10714;10715;-1316963 610717;10686;10761;10724;10665;10720;10722;10730;10731;-1316947 610739;10708;10783;10746;10686;10742;10744;10751;10753;-1316926 The sector name can consist of a maximum of 32 characters (including []).

1.5.4.19.2 Function blocks

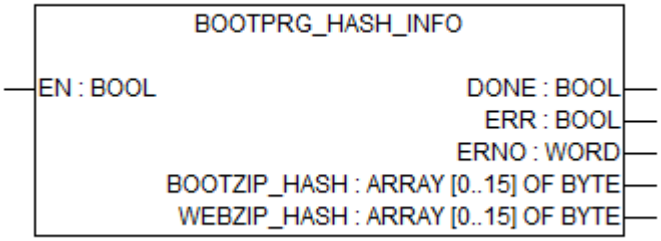
BOOTPRG_HASH_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V2.3.0
Type	Function block with historical values

The function block BOOTPRG_HASH_INFO reads the MD5 hash of the boot project.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

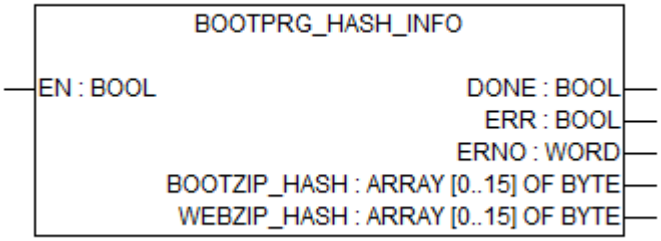
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

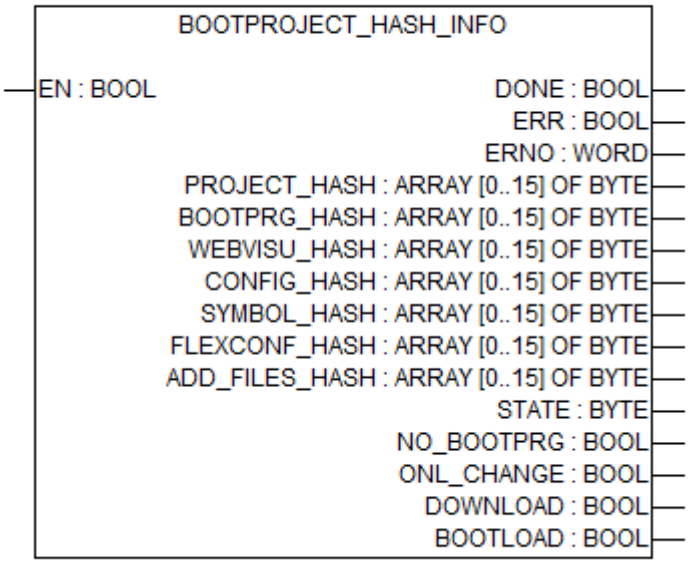
BOOTZIP_HASH BOOTZIP_HASH (array [0..15] of byte) outputs the MD5 hash of the user program part of the boot project (BOOT.ZIP file).

WEBZIP_HASH WEBZIP_HASH (array [0..15] of byte) outputs the MD5 hash of the webvisu part of the boot project (WEB.ZIP file).

Function call in ST

```
Hash (EN := Hash_EN);


Hash_DONE          := Hash.DONE;
Hash_ERR           := Hash.ERR;
Hash_ERNO          := Hash.ERNO;
Hash_BOOTZIP_HASH  := Hash.BOOTZIP_HASH;
Hash.WEBZIP_HASH   := Hash.WEBZIP_HASH;
```

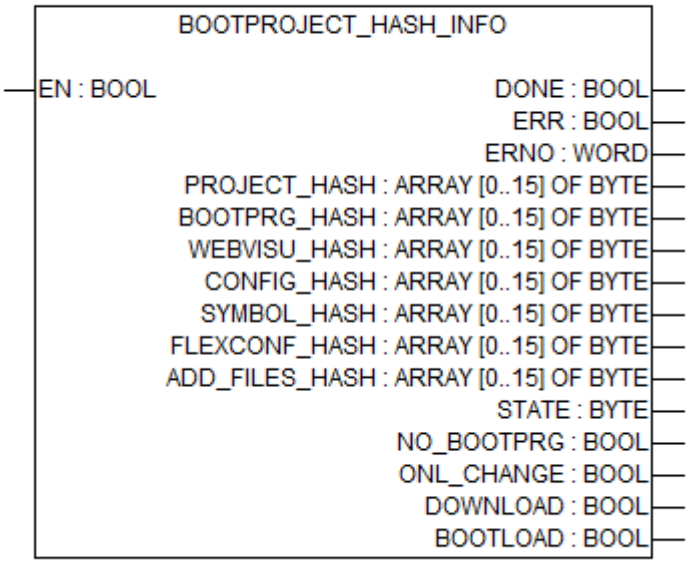
BOOTPROJECT_HASH_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V2.3.2
Type	Function block with historical values

The function block BOOTPROJECT_HASH_INFO reads the MD5 hash of different parts of the boot project.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

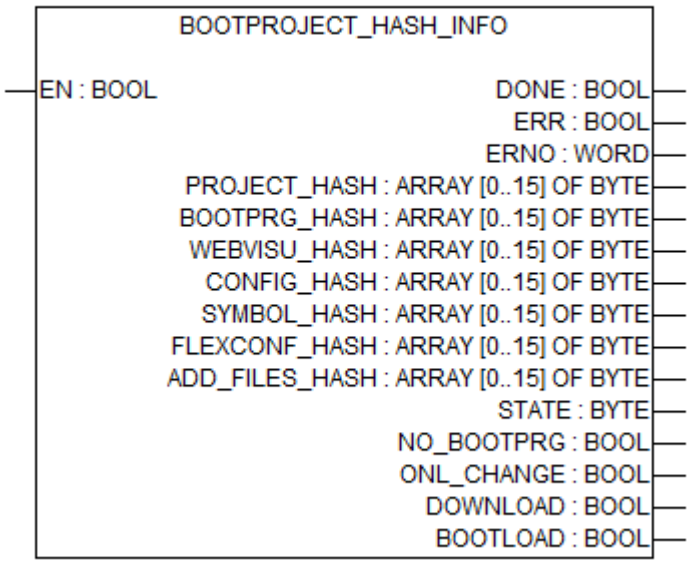
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PROJECT_HASH

PROJECT_HASH (array [0..15] of byte) outputs the MD5 hash of the whole boot project. It is calculated from all the following MD5 sums.

BOOTPRG_HASH

BOOTZIP_HASH (array [0..15] of byte) outputs the MD5 hash of the user program code of the boot project (BOOT.ZIP file).

WEB-VISU_HASH WEBVISU_HASH (array [0..15] of byte) outputs the MD5 hash of all web visualization files in the boot project.

CONFIG_HASH CONFIG_HASH (array [0..15] of byte) outputs the MD5 hash of PLC configuration in the boot project.

SYMBOL_HASH SYMBOL_HASH (array [0..15] of byte) outputs the MD5 hash of the symbol file in the boot project.

FLEX-CONF_HASH FLEXCONF_HASH (array [0..15] of byte) outputs the MD5 hash of the flexible configuration file of the boot project.

ADD_FILES_HASH ADD_FILES_HASH (array [0..15] of byte) outputs the MD5 hash of all additional files contained in the boot project.

STATE BYTE STATE outputs the validity of the hash outputs. It contains status bits for different states the PLC can be in. These states affect whether the hashes at the outputs of this function block will be valid or not and why. All the Status Bits are also shown at the other BOOL outputs of this function block:

NO_BOOTPRG STATE.Bit0

ONL_CHANGE STATE.Bit1

DOWNLOAD STATE.Bit2

BOOTLOAD STATE.Bit3

NO_BOOTPRG

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

NO_BOOTPRG is set to TRUE if no boot project has been found that could be hashed (STATE Bit 0).

ONL_CHANGE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

ONL_CHANGE is set to TRUE if an online change has been done since booting the PLC, so the hashes are outdated (STATE Bit 1).

DOWNLOAD

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

DOWNLOAD is set to TRUE if a project download has been done since booting the PLC, so the hashes are outdated (STATE Bit 2).

BOOTLOAD

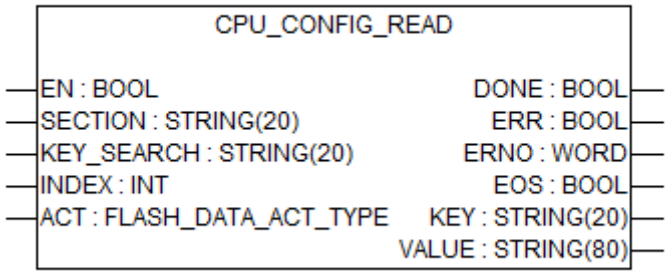
Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

DOWNLOAD is set to TRUE if a boot project download has been flashed since booting the PLC, so the hashes are outdated (STATE Bit 3).

Function call in ST

```
BOOTPROJECT_HASH_INFO (EN := Hash_EN);


Hash_DONE := BOOTPROJECT_HASH_INFO.DONE;
Hash_ERR  := BOOTPROJECT_HASH_INFO.ERR;
Hash_ERNO := BOOTPROJECT_HASH_INFO.ERNO;
Hash_PROJECT_HASH    := BOOTPROJECT_HASH_INFO.PROJECT_HASH;
Hash_BOOTPRG_HASH    := BOOTPROJECT_HASH_INFO.BOOTPRG_HASH;
Hash_WEBVISU_HASH    := BOOTPROJECT_HASH_INFO.WEBVISU_HASH;
Hash_CONFIG_HASH     := BOOTPROJECT_HASH_INFO.CONFIG_HASH;
Hash_SYMBOL_HASH     := BOOTPROJECT_HASH_INFO.SYMBOL_HASH;
Hash_FLEXCONF_HASH   := BOOTPROJECT_HASH_INFO.FLEXCONF_HASH;
Hash_ADD_FILES_HASH  := BOOTPROJECT_HASH_INFO.ADD_FILES_HASH;
Hash_STATE           := BOOTPROJECT_HASH_INFO.STATE;
Hash_NO_BOOTPRG      := BOOTPROJECT_HASH_INFO.NO_BOOTPRG;
Hash_ONL_CHANGE      := BOOTPROJECT_HASH_INFO.ONL_CHANGE;
Hash_DOWNLOAD        := BOOTPROJECT_HASH_INFO.DOWNLOAD;
Hash_BOOTLOAD        := BOOTPROJECT_HASH_INFO.BOOTLOAD;
```

CPU_CONFIG_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block reads individual values of configuration data from the Flash memory. One line consists of a pair of key values and is located within the section. The configuration data stored in the Flash are like an ini file in their construction, like for example the sdcard.ini on the memory card of the AC500.

Example for a line with an IP address in the section Common:

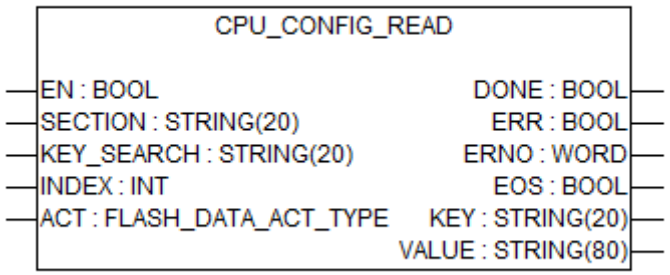
[Common]

IP_ADR=192.168.0.1

The procedure should be performed in three steps:

1. Initialization (ACT = FLASH_DATA_INIT) of the configuration data with a FALSE/TRUE edge at input EN.
2. Reading (ACT = FLASH_DATA_READ) of the individual values (SECTION, KEY_SEARCH, INDEX) with a FALSE/TRUE edge at input EN.
3. Cancel (ACT = FLASH_DATA_SKIP) the reading with a FALSE/TRUE edge at input EN.

Input Description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SECTION

Data type: STRING[20]

Section, within to search/read. A section has always to be specified, except for the initialization step.

KEY_SEARCH

Data type: STRING[20]

If it is searched for the value of a known key, this must be specified at the input KEY_SEARCH. The key is searched within the section which is specified at the input SECTION. In this case, the input INDEX is ignored.

INDEX

Data type: INT

The INDEX input is only evaluated, if the KEY_SEARCH input is 0.

With the INDEX input, a specified line of the configuration data within the section, specified at input SECTION, is output.

If INDEX is unequal to 0, exactly this line number is output, if it exists. If INDEX equals 0 and if there is no value at input KEY_SEARCH, the lines of the section (specified at input SECTION) are output sequentially.

In the latter case each time the next line is read with every FALSE/TRUE edge at input EN as long as the output EOS changes to TRUE and thus terminating the section.

ACT

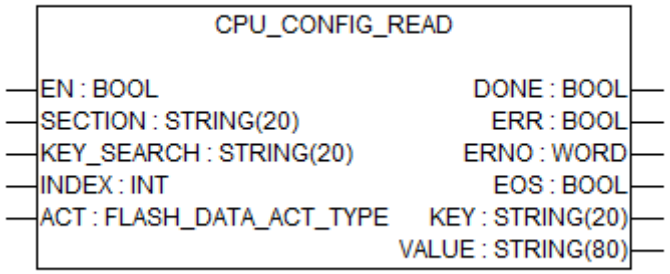
Data type: FLASH_DATA_ACT_TYPE

With the ACT input, the function block can be set to initialize, read the configuration data or to cancel the function block operation.

The input can be set to the following values:

- FLASH_DATA_INIT
  With this input, the function block is initialized for processing of configuration data. The configuration data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input.
  If ACT=FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.
- FLASH_DATA_READ
  Read data from RAM disk with the ACT = FLASH_DATA_READ.
- FLASH_DATA_SKIP
  Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.

Output Description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

EOS

Data type: BOOL

Output EOS (end of section) indicates whether the end of the section was reached while searching sequentially. This output only must be evaluated, if a search is carried out with INDEX=0 and KEY_SEARCH=0.

KEY Data type: STRING[20]

Output KEY indicates the found key.

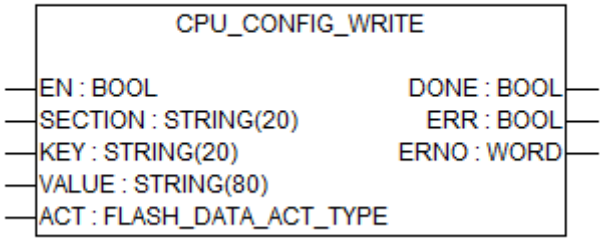
VALUE Data type: STRING[80]

Output VALUE indicates the value of the found key.

Function Call in ST

```
ConfigRead   (EN         := ConfigRead_EN,
    SECTION    := ConfigRead_SECTION,
    KEY_SEARCH := ConfigRead_KEY_SEARCH,
    INDEX      := ConfigRead_INDEX,
    ACT        := ConfigRead_ACT);


ConfigRead_DONE      := ConfigRead.DONE;
ConfigRead_DONE_ERR  := ConfigRead.ERR;
ConfigRead_ERNO      := ConfigRead.ERNO;
ConfigRead_EOS       := ConfigRead.EOS;
ConfigRead_KEY       := ConfigRead.KEY;
ConfigRead_VALUE     := ConfigRead.VALUE;
```

CPU_CONFIG_WRITE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block writes user-defined configuration data to the Flash memory. The configuration data stored in the Flash are like an ini file in their construction, like for example the sdcard.ini on the memory card of the AC500.

Example for a line with an IP address in the section Common:

[Common]

IP_ADR=192.168.0.1

The procedure should be performed in three steps:

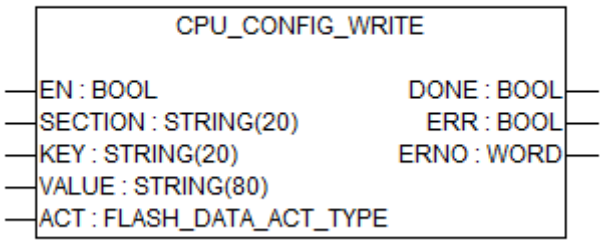
1) Initialization (ACT = FLASH_DATA_INIT) of the configuration data with a FALSE/TRUE edge at input EN.

2) Writing (ACT = FLASH_DATA_WRITE) of the individual values (SECTION, KEY, VALUE) with a FALSE/TRUE edge at input EN. This step is repeated for each entry until all the desired data has been written.

3) Transmit (ACT = FLASH_DATA_SAVE) the data into the Flash and ensuring a permanent data storage in this way.

Only after the third step has been done, the data is stored in the Flash and will be non-volatile in case of power-down.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SECTION Section to write to.

KEY Key to be written.

VALUE Value of the key to be written.

ACT Data type: FLASH_DATA_ACT_TYPE

With the ACT input, the function block can be set to initialize, write the configuration data or to save, to skip or reset the function block operation. The input can be set to following values:

FLASH_DATA_INIT:

With this input, the function block is initialized for processing of configuration data. The configuration data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input. If ACT = FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.

FLASH_DATA_WRITE:

Write data to RAM disk with the ACT = FLASH_DATA_WRITE.

FLASH_DATA_SAVE: Save data from RAM disk to Flash

Using ACT = FLASH_DATA_SAVE, all changes made at the configuration data are saved to Flash. So they are stored non-volatile during power-down. If this step is not carried out, all previous changes are lost. Since writing into the Flash takes some time, the save action should be performed as the last step, after all of the desired values are written, modified or deleted.

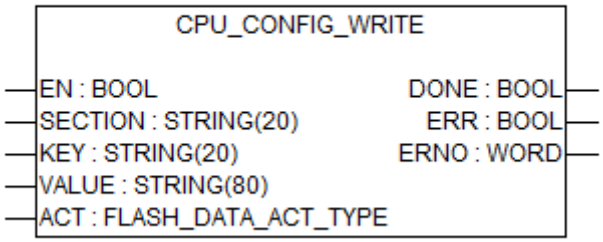
FLASH_DATA_SKIP:

Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.

FLASH_DATA_RES:

With this input, a reset function is carried out, which deletes all of the configuration data completely.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

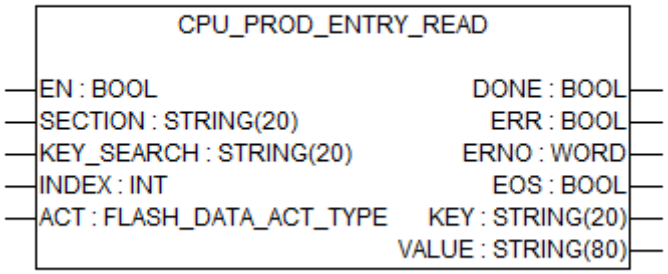
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ConfigWrite  (EN      := ConfigWrite_EN,
     SECTION := ConfigWrite_SECTION,
     KEY     := ConfigWrite_KEY,
     VALUE   := ConfigWrite_VALUE,
     ACT     := ConfigWrite_ACT);


ConfigWrite_DONE      := ConfigWrite.DONE;
ConfigWrite_ERR       := ConfigWrite.ERR;
ConfigWrite_ERNO      := ConfigWrite.ERNO;
```

CPU_PROD_ENTRY_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block reads one line from the production data in the Flash memory. One line consists of a pair of key values and is located within a section.

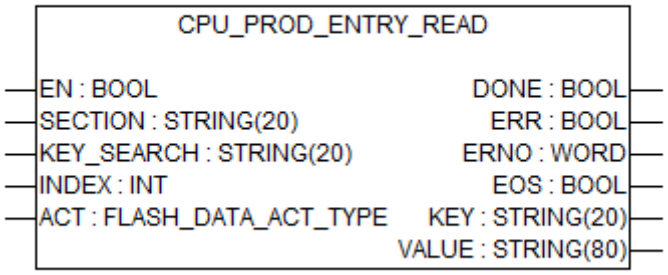
The section name of the production data is called "Common". It consists of many keys with the corresponding values, see table:

Production data		Comment
Common	Section Name	Name of the section
BA_INST	Key and value	BA number
IDENT	Key and value	SAP identnumber
INDEX	Key and value	Index of the module
MAC	Key and value	MAC address of the CPU
MANUF_DATE	Key and value	Date of manufacture
MANUF_YEAR	Key and value	Manufacturing year
MANUF_PLACE	Key and value	Place where the PLC was produced
SERIAL_NR	Key and value	Serial number of the PLC
TYPE	Key and value	CPU type

The procedure should be performed in three steps:

1. Initialization (ACT = FLASH_DATA_INIT) of the production data with a FALSE/TRUE edge at input EN.
2. Reading (ACT = FLASH_DATA_READ) of the individual values (SECTION, KEY_SEARCH, INDEX) with a FALSE/TRUE edge at input EN.
3. Cancel (ACT = FLASH_DATA_SKIP) the reading with a FALSE/TRUE edge at input EN.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SECTION

Data type: STRING[20]

Section, within to search/read. A section has always to be specified, except for the initialization step.

KEY_SEARCH

Data type: STRING[20]

If it is searched for the value of a known key, this must be specified at the input KEY_SEARCH. The key is searched within the section which is specified at the input SECTION. In this case, the input INDEX is ignored.

INDEX

Data type: INT

The INDEX input is only evaluated, if the KEY_SEARCH input is 0.

With the INDEX input, a specified line of the configuration data within the section, specified at input SECTION, is output.

If INDEX is unequal to 0, exactly this line number is output, if it exists. If INDEX equals 0 and if there is no value at input KEY_SEARCH, the lines of the section (specified at input SECTION) are output sequentially.

In the latter case each time the next line is read with every FALSE/TRUE edge at input EN as long as the output EOS changes to TRUE and thus terminating the section.

ACT

Data type: FLASH_DATA_ACT_TYPE

With the ACT input the function block can be set to initialize, read the production data or to cancel the function block operation. The input can be set to following values:

FLASH_DATA_INIT:

With this input, the function block is initialized for processing of production data. The production data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input. If ACT = FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.

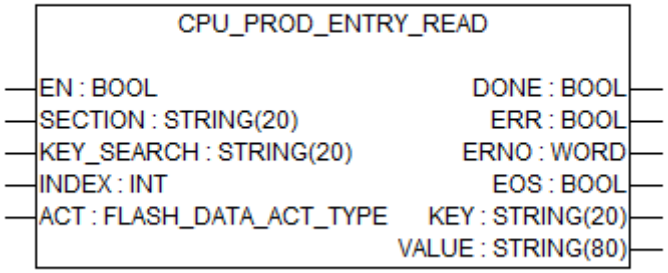
FLASH_DATA_READ:

Read data from RAM disk with the ACT = FLASH_DATA_READ.

FLASH_DATA_SKIP:

Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

EOS Data type: BOOL

Output EOS (end of section) indicates whether the end of the section was reached while searching sequentially. This output only must be evaluated, if a search is carried out with INDEX=0 and KEY_SEARCH=0.

KEY Data type: STRING[20]

Output KEY indicates the found key.

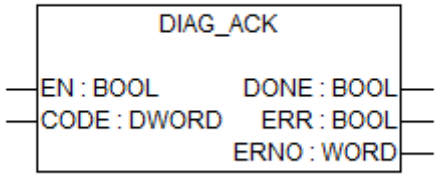
VALUE Data type: STRING[80]

Output VALUE indicates the value of the found key.

Function call in ST

```
CPU_PROD_ENTRY_READ(EN := EN_CPU_PROD_ENTRY_READ,
 SECTION := SECTION_CPU_PROD_ENTRY_READ,
 KEY_SEARCH := KEY_SEARCH_CPU_PROD_ENTRY_READ,
 INDEX := INDEX_CPU_PROD_ENTRY_READ,
 ACT := ACT_CPU_PROD_ENTRY_READ);


DONE_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.DONE;
ERR_CPU_PROD_ENTRY_READ  := CPU_PROD_ENTRY_READ.ERR;
ERNO_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.ERNO;
EOS_CPU_PROD_ENTRY_READ  := CPU_PROD_ENTRY_READ.EOS;
KEY_CPU_PROD_ENTRY_READ  := CPU_PROD_ENTRY_READ.KEY;
VALUE_CPU_PROD_ENTRY_READ:= CPU_PROD_ENTRY_READ.VALUE;
```

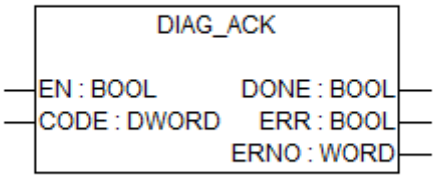
DIAG_ACK



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block DIAG_ACK can be used to acknowledge any error. Selection of the error to be acknowledged is performed using a 32-bit code. If the error list contains several entries with the selected error code number, acknowledgement is always performed for the oldest entry in the list.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

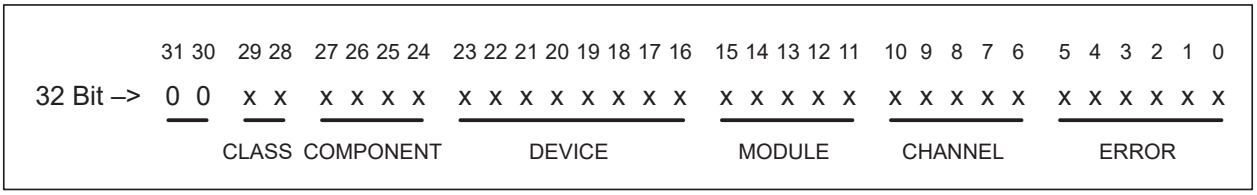
In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

CODE

Data type	Default value	Range	Unit
DWORD	-	-	-

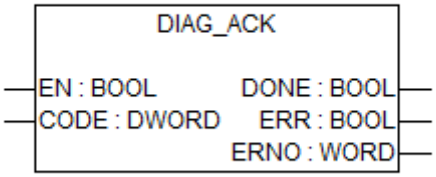
At input CODE, the code number of the error to be acknowledged is specified. The code number of an existing error can be read using the function block DIAG_GET or calculated manually. The structure of the error coding is as follows:



Bit 0 to 5	- Error number	Valid range: 0...63
Bit 6 to 10	- Channel number	Valid range: 0...31
Bit 11 to 15	- Module number	Valid range: 0...31
Bit 16 to 23	- Device number	Valid range: 0...255
Bit 24 to 27	- Component number	Valid range: 0...15
Bit 28 to 29	- Error class	Valid range: 1...4
Bit 30 to 31	- Reserved; both bits always must be zero.	

Valid range: 16#3E9...16#3FFFFFFF

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

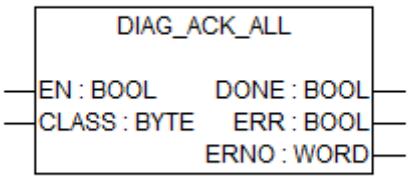
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
DiagAck(EN := DiagAck_EN,
      CODE := DiagAck_CODE);


DiagAck_DONE := DiagAck.DONE;
DiagAck_ERR  := DiagAck.ERR;
DiagAck_ERNO := DiagAck.ERNO;
```

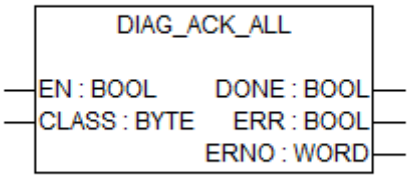
DIAG_ACK_ALL



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block DIAG_ACK_ALL can be used to acknowledge all errors of an error class simultaneously. Selection of the error class, the errors of which are to be acknowledged, is done using the input CLASS.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

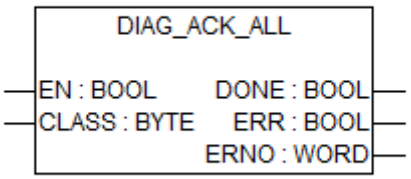
Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

CLASS

Data type	Default value	Range	Unit
BYTE	-	1...4	-

At input CLASS, the error class is specified, the errors of which are to be acknowledged.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

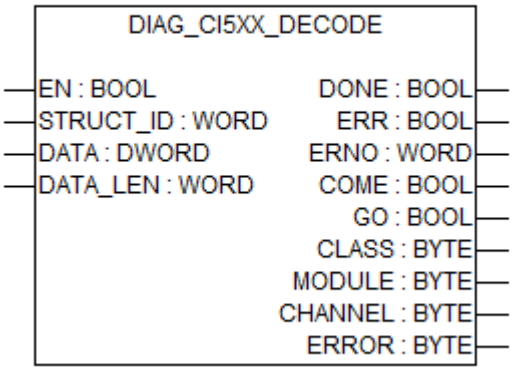
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
DiagAckAll(EN := DiagAckAll_EN,
        CLASS := DiagAckAll_CLASS);


DiagAckAll_DONE := DiagAckAll.DONE;
DiagAckAll_ERR  := DiagAckAll.ERR;
DiagAckAll_ERNO := DiagAckAll.ERNO;
```

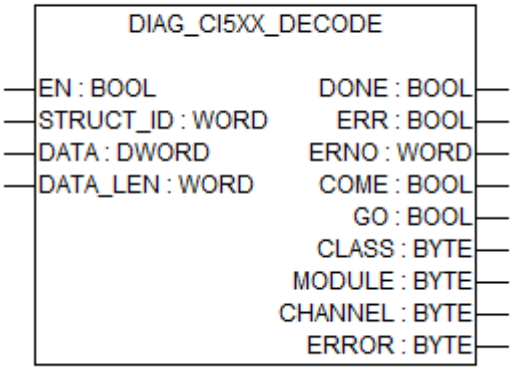
DIAG_CI5XX_DECODE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block decodes CI5XX diagnosis data to AC500 diagnosis system information.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

STRUCT_ID

Data type	Default value	Range	Unit
WORD	0	-	-

Identifier of the diagnosis format to read from CI5XX.

DATA

Data type	Default value	Range	Unit
DWORD	0	-	-

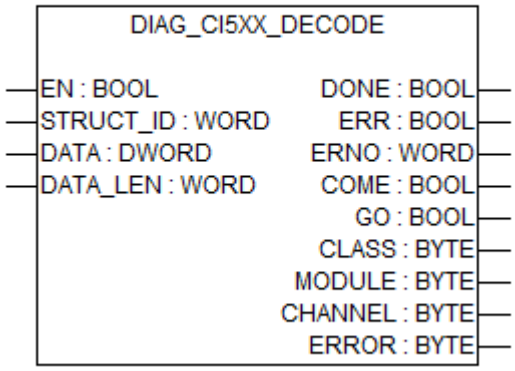
Address to the memory where the diagnosis message should be read from.

DATA_LEN

Data type	Default value	Range	Unit
WORD	0	-	-

Length of the data to read from CI5XX in bytes. Depends on the ID. For available structures see your CI5XX documentation.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

COME

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

The output COME is the flag of standard AC500 diagnosis. It indicates that a diagnosis messages has appeared. For details see *Chapter 1.7.1 “The diagnosis system” on page 6365*.

GO

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

The output GO is the flag of standard AC500 diagnosis. It indicates that a diagnosis messages has disappeared. For details see *Chapter 1.7.1 “The diagnosis system” on page 6365*.

CLASS

Data type	Default value	Range	Unit
BYTE	0	-	-

The output CLASS indicates the severity of the diagnosis message. For details see *Chapter 1.7.1 “The diagnosis system” on page 6365*.

MODULE

Data type	Default value	Range	Unit
BYTE	0	-	-

The output MODULE shows the module the read error is assigned to.

CHANNEL

Data type	Default value	Range	Unit
BYTE	0	-	-

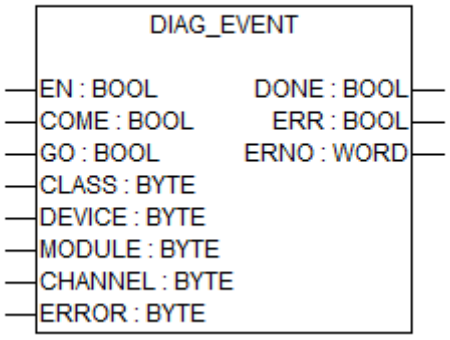
The output CHANNEL shows the channel the read error is assigned to.

ERROR

Data type	Default value	Range	Unit
BYTE	0	-	-

The outputs ERROR provides the error number of the read error.

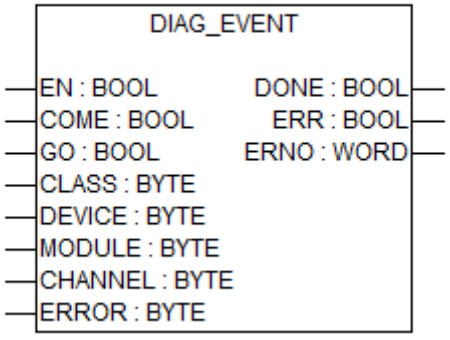
DIAG_EVENT



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block DIAG_EVENT can be used to generate any error of component 15 (*Chapter 1.7.1.5 "Structure of error numbers" on page 6369*). Each error can have three possible states: 1) "Error has come" (COME), 2)"Error has gone" (GO) and 3) "Error has been acknowledged". Using the function block DIAG_EVENT, errors of the states 1) and 2) can be generated. Error acknowledgement is done using the function blocks DIAG_ACK *Chapter 1.5.4.19.2.6 "DIAG_ACK" on page 1517* or DIAG_ACK_ALL *Chapter 1.5.4.19.2.7 "DIAG_ACK_ALL" on page 1520* or using the *Chapter 1.7.1.5 "Structure of error numbers" on page 6369* or directly using the display.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COME

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Input COME is used to specify the status of the error to be generated. This input can be used in combination with input GO.

COME = TRUE = "Error has come"

GO

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Input GO is used to specify the status of the error to be generated. This input can be used in combination with input COME.

GO = TRUE = "Error has gone"

CLASS

Data type	Default value	Range	Unit
BYTE	-	1...4	-

At input CLASS, the error class is specified, the errors of which are to be acknowledged.

DEVICE

Data type	Default value	Range	Unit
BYTE	-	0...255	-

Input "DEVICE" is used to specify the device number the generated error should be assigned to.

MODULE

Data type	Default value	Range	Unit
BYTE	-	0...31	-

Input "MODULE" is used to specify the module the generated error should be assigned to.

CHANNEL

Data type	Default value	Range	Unit
BYTE	-	0...31	-

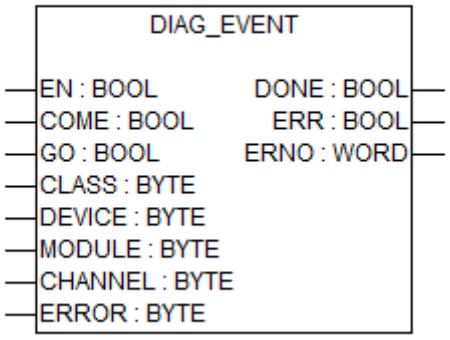
Input "CHANNEL" is used to specify the channel the generated error should be assigned to.

ERROR

Data type	Default value	Range	Unit
BYTE	-	0...63	-

At input "ERROR", the error number to be generated is specified. In order to facilitate later assignment of the errors, it is recommended to use the error numbers already defined. Please refer to the corresponding chapter for a more detailed overview of possible system errors and the used error numbers. If an error is not yet declared by an error number, it is recommended to use a free error number.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

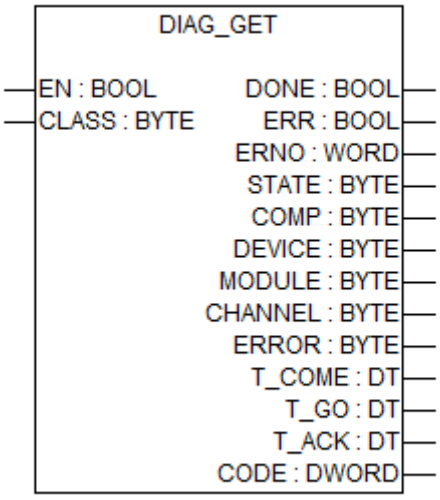
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
DiagEvent(EN := DiagEvent_EN,
        COME := DiagEvent_COME,
        GO := DiagEvent_GO,
        CLASS := DiagEvent_CLASS,
        DEVICE := DiagEvent_DEVICE,
        MODULE := DiagEvent_MODULE,
        CHANNEL := DiagEvent_CHANNEL,
        ERROR := DiagEvent_ERROR);


DiagEvent_DONE := DiagEvent.DONE;
DiagEvent_ERR  := DiagEvent.ERR;
DiagEvent_ERNO := DiagEvent.ERNO;
```

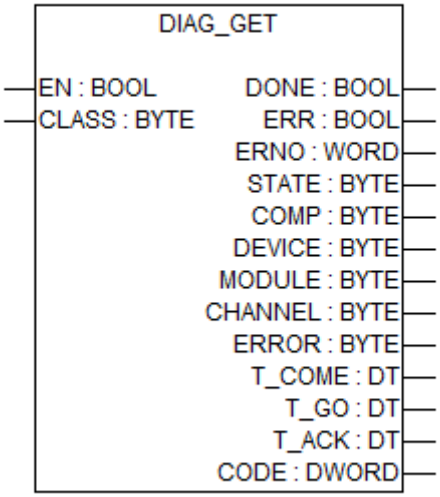
DIAG_GET



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block "DIAG_GET" can be used to read an error of any error class. Each error can only be read once. If this function block is used more than once for a specific error class, the next error output is the oldest error that has not been read yet. If all errors were already read or if there is no existing error available, this is accordingly indicated at output STATE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

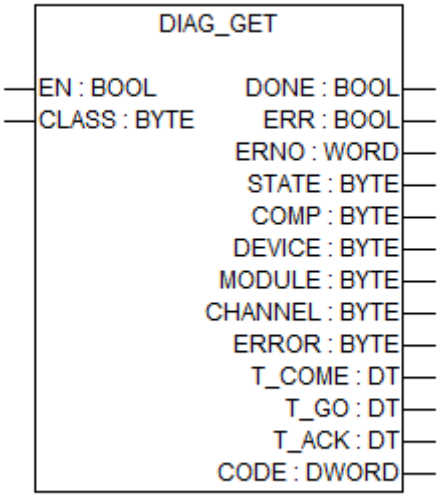
Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

CLASS

Data type	Default value	Range	Unit
BYTE	-	1...4	-

At input CLASS, the error class is specified, the errors of which are to be acknowledged.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
BYTE	-	-	-

STATE outputs the current status of the read error. The error status is a combination of the states "Error has come", "Error has gone" and "Error has been acknowledged". The following status numbers are possible:

State	Error	come	gone	acknowledged
16#02		x		
16#04			x	
16#06		x	x	
16#08				x
16#0A		x		x
16#0C			x	x
16#0E		x	x	x
16#F0	Alle errors read or no error available			

COMP

Data type	Default value	Range	Unit
BYTE	-	-	-

"COMP" outputs the component number the read error is assigned to.

DEVICE

Data type	Default value	Range	Unit
BYTE	-	-	-

"DEVICE" outputs the device number the read error is assigned to.

MODULE

Data type	Default value	Range	Unit
BYTE	-	-	-

"MODULE" outputs the module the read error is assigned to.

CHANNEL

Data type	Default value	Range	Unit
BYTE	-	-	-

"CHANNEL" outputs the channel the read error is assigned to.

ERROR

Data type	Default value	Range	Unit
BYTE	-	-	-

"ERROR" outputs the error number of the read error.

T_COME Data type: DT

T_COME (time come) outputs the time stamp, when the read error occurred ("has come"). If no time stamp is available for the error status "come" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

T_GO Data type: DT

T_GO (time go) outputs the time stamp, when the read error "has gone". If no time stamp is available for the error status "gone" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

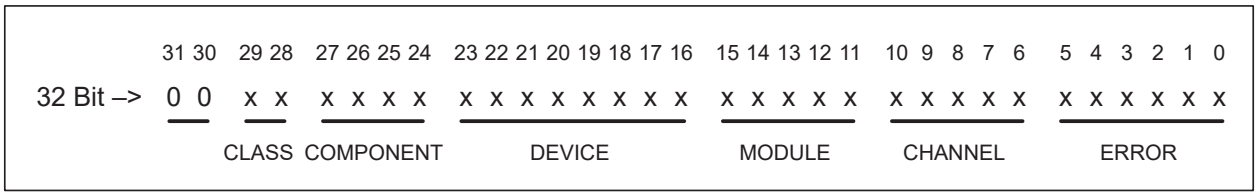
T_ACK Data type: DT

T_ACK (time acknowledge) outputs the time stamp, when the read error was "acknowledged". If no time stamp is available for the error status "acknowledged" (see also the description of output STATE), no value is written to this output. In this case, the output value remains at the default value DT#1970-01-01-00:00.

CODE

Data type	Default value	Range	Unit
DWORD	-	-	-

CODE outputs the code number of the read error. The structure of the error encoding is as follows:

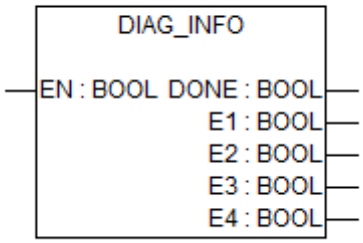


Function call in ST

```
DiagGet(EN := DiagGet_EN,
        CLASS := DiagGet_CLASS);


DiagGet_DONE := DiagGet.DONE;
DiagGet_ERR := DiagGet.ERR;
DiagGet_ERNO := DiagGet.ERNO;
DiagGet_STATE := DiagGet.STATE;
DiagGet_COMP := DiagGet.COMP;
DiagGet_DEVICE := DiagGet.DEVICE;
DiagGet_MODULE := DiagGet.MODULE;
DiagGet_CHANNEL := DiagGet.CHANNEL;
```

```
DiagGet_ERROR := DiagGet.ERROR;
DiagGet_T_COME := DiagGet.T_COME;
DiagGet_T_GO := DiagGet.T_GO;
DiagGet_T_ACK := DiagGet.T_ACK;
DiagGet_CODE := DiagGet.CODE;
```

DIAG_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block DIAG_INFO can be used to display an overview of all errors that were not read yet. The output is sorted according to the error classes E1 to E4. If at least one error is present in any error class and if this error has not been read yet using the function block DIAG_GET, this is displayed at the corresponding Ex output.

Input description

EN

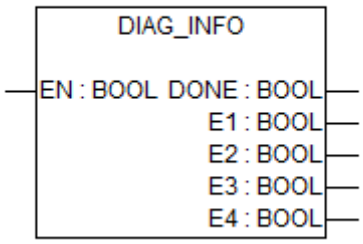
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE (done)

Data type: BOOL

Operation done = TRUE, as long as EN = TRUE

Output DONE indicates the state of job processing. If the processing is finished, DONE is set to TRUE.

E1 (error class 1)

Data type: BOOL

Output E1 is set to TRUE, if at least 1 error is present in error class 1 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E1 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E2 (error class 2)

Data type: BOOL

Output E2 is set to TRUE, if at least 1 error is present in error class 2 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E2 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E3 (error class 3)

Data type: BOOL

Output E3 is set to TRUE, if at least 1 error is present in error class 3 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E3 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E4 (error class 4)

Data type: BOOL

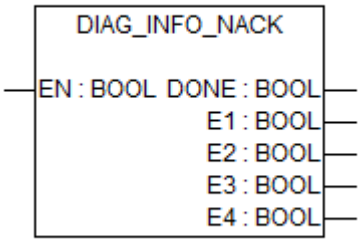
Output E4 is set to TRUE, if at least 1 error is present in error class 4 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E4 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

Function call in ST

```
DiagInfo(EN := DiagInfo_EN);


DiagInfo_DONE := DiagInfo.DONE;
DiagInfo_E1 := DiagInfo.E1;
DiagInfo_E2 := DiagInfo.E2;
DiagInfo_E3 := DiagInfo.E3;
DiagInfo_E4 := DiagInfo.E4;
```

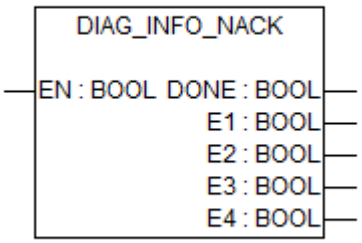
DIAG_INFO_NACK



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block DIAG_INFO_NACK displays an overview of all errors that were not acknowledged yet by buttons/display or function block DIAG_EVENT. The output is sorted according to the error classes E1 to E4. If at least one error is present in any error class and if this error has not been acknowledged yet, this is displayed at the corresponding Ex output.

Input description



EN

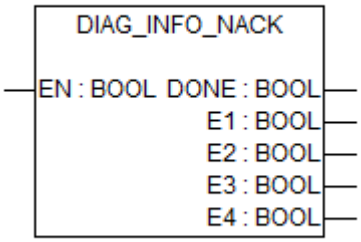
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE (done)

Data type: BOOL

Operation done = TRUE, as long as EN = TRUE

Output DONE indicates the state of job processing. If the processing is finished, DONE is set to TRUE.

E1 (error class 1)

Data type: BOOL

Output E1 is set to TRUE, if at least 1 error is present in error class 1 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E1 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E2 (error class 2)

Data type: BOOL

Output E2 is set to TRUE, if at least 1 error is present in error class 2 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E2 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E3 (error class 3)

Data type: BOOL

Output E3 is set to TRUE, if at least 1 error is present in error class 3 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E3 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

E4 (error class 4)

Data type: BOOL

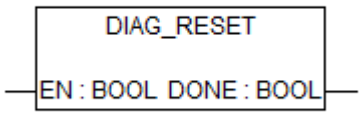
Output E4 is set to TRUE, if at least 1 error is present in error class 4 and if this error has not been read yet using the function block DIAG_GET.

Consequently, if E4 is FALSE, either no errors are available in this error class or, if errors are available, all errors of this class were already read.

Function call in ST

```
DiagInfo(EN := DiagInfo_EN);
DiagInfo_DONE := DiagInfo.DONE;
DiagInfo_E1 := DiagInfo.E1;
DiagInfo_E2 := DiagInfo.E2;
DiagInfo_E3 := DiagInfo.E3;
DiagInfo_E4 := DiagInfo.E4;
```

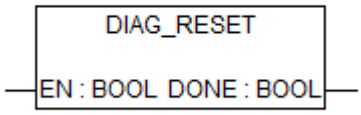
DIAG_RESET



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

Function block DIAG_RESET is used for resetting the diagnosis system of the CPU completely.

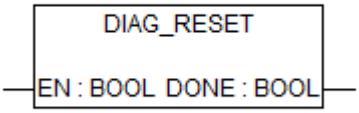
Input description



EN (enable) Data type: BOOL

The function block is activated by a TRUE at the input EN. A FALSE keeps the function block deactivated. Is the function block activated, the values being present at the inputs are processed and the output values are delivered.

Output description



DONE (done) Data type: BOOL

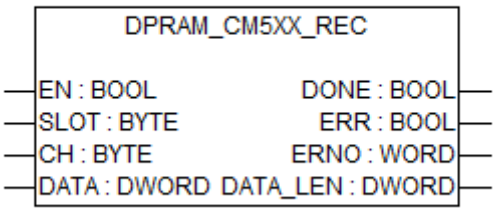
Operation done = TRUE, as long as EN = TRUE

Output DONE indicates the state of job processing. If the processing is finished, DONE is set to TRUE.

Function call in ST

```
DiagReset(EN := DiagReset_EN);
DiagReset_DONE := DiagReset.DONE;
```

DPRAM_CM5XX_REC



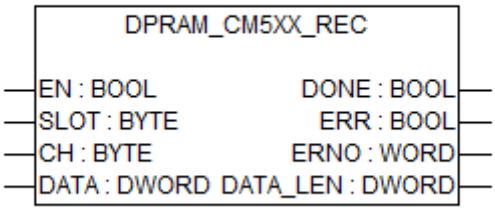
Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block DPRAM_CM5XX_REC is used for receiving data from an CM5xx. This data is stored in the configured memory area (DATA), i.e. the memory address for the received data via ADR operator. The function block is triggered by a high signal at input EN. It remains active until the input EN is set to FALSE. The slot number and the channel of the CM5xx are put to the inputs SLOT and CH. The output DATA_LEN shows the length of the received data in bytes. The successful data reception is indicated by DONE=TRUE and ERR=FALSE. If during operation an error was detected, this is indicated at the outputs ERR and ERNO.

CAUTION!
The reception using the function function block DPRAM_CM5XX_REC is not edge-triggered. The input EN must therefore be at TRUE as long as data has to be received.

DPRAM message-based data exchange allows a maximum of 260 Bytes per message to be sent or received.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type: BYTE

At input CH (channel), the addressed channel is specified.

Valid values: 1 and 2.

DATA (data)

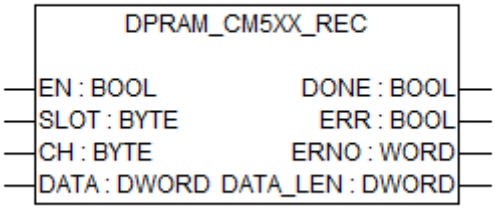
Data type: DWORD

At input DATA, the address of the variable is specified, of which the user data shall be received.

DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION!
Memory area overlapping
Avoid memory area overlappings by specifying the size of the variable to the max. data expected.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DATA_LEN

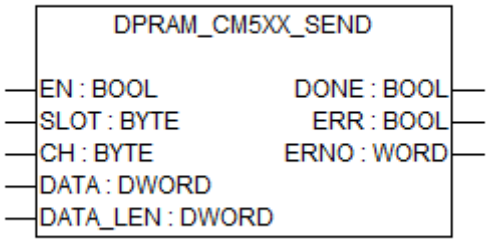
Data type	Default value	Range	Unit
DWORD	-	-	-

DATA_LEN (datalength) outputs the length of the received data in bytes. DATA_LEN is only valid if DONE = TRUE.

Function call in ST

```
CM5xxRec (EN := CM5xxRec_EN,
          SLOT := CM5xxRec_SLOT,
          CH := CM5xxRec_CH,
          DATA := ADR(CM5xxRec_DATA),
          DONE => CM5xxRec_DONE,
          ERR => CM5xxRec_ERR,
          ERNO => CM5xxRec_ERNO,
          DATA_LEN => CM5xxRec_DATA_LEN);


CM5xxRec_DONE := CM5xxRec.DONE;
CM5xxRec_ERR := CM5xxRec.ERR;
CM5xxRec_ERNO := CM5xxRec.ERNO;
CM5xxRec_DATA_LEN := CM5xxRec.DATA_LEN;
```

DPRAM_CM5XX_SEND



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block DPRAM_CM5XX_SEND is used for sending data to the CM5xx. This data is provided in the configured memory area (DATA), i.e. the memory address for the data to be sent via ADR operator. The function block is activateded by a high signal at input EN. It remains active until the input EN is set to FALSE. The slot number and the channel of the CM5xx are put to the inputs SLOT and CH. At the input DATA_LEN the length of the data to be sent is specified in bytes. The successful data transmission is indicated by DONE=TRUE and ERR=FALSE. If during operation an error was detected, this is indicated at the outputs ERR and ERNO.

Sending data with the function block DPRAM_CM5XX_SEND is edge-triggered, i.e. that each sending process is triggered by a FALSE/TRUE edge at input EN.

DPRAM message-based data exchange allows a maximum of 260 Bytes per message to be sent or received.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type: BYTE

At input CH (channel), the addressed channel is specified.

Valid values: 1 and 2.

DATA (data)

Data type: DWORD

At input DATA, the address of the variable is specified, of which the user data shall be received.

DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION!
Memory area overlapping
Avoid memory area overlappings by specifying the size of the variable to the max. data expected.

DATA_LEN

Data type: DWORD

At input DATA_LEN the length of the data to be sent is specified in bytes.

Output description

DONE

Data type: BOOL

Output DONE indicates that data has been sent. The output must always be considered together with the output ERR.

The following is valid:

DONE = TRUE and ERR = FALSE:

The sending process has been completed. A data set has been sent correctly.

DONE = TRUE and ERR = TRUE:

An error occurred while sending data. The error number is indicated at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during Function Block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO

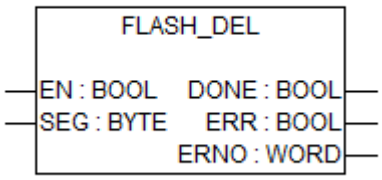
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
CM5xxSend (EN := CM5xxSend_EN,
           SLOT := CM5xxSend_SLOT,
           CH := CM5xxSend_CH,
           DATA := ADR(CM5xxSend_DATA),
           DATA_LEN := CM5xxSend_DATA_LEN,
           DONE => CM5xxSend_DONE,
           ERR => CM5xxSend_ERR,
           ERNO => CM5xxSend_ERNO);


CM5xxSend_DONE := CM5xxSend.DONE;
CM5xxSend_ERR := CM5xxSend.ERR;
CM5xxSend_ERNO := CM5xxSend.ERNO;
```

FLASH_DEL



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

This function block deletes a user data segment from the Flash. All data in this data segment are lost after deletion.

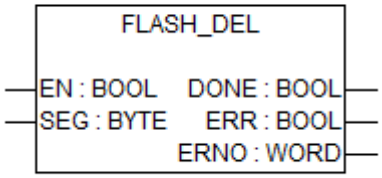
Access to the Flash is only possible using the function blocks FLASH_DEL, FLASH_WRITE and FLASH_READ.

The input SEG defines the data segment within the Flash. In the AC500, two segments with the number 1 and 2 are reserved for the user, each providing 64 kB. Deleting a data segment within the Flash may take several PLC cycles.

A FALSE/TRUE edge at input EN triggers the deletion of the data segment once. Input EN will not be evaluated again, until the delete operation is completed (DONE = TRUE).

After completion of the delete procedure, all function block outputs are updated. The deletion was successful, if DONE = TRUE and ERR = FALSE. If the outputs show DONE = TRUE and ERR = TRUE, the data segment could not be deleted.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

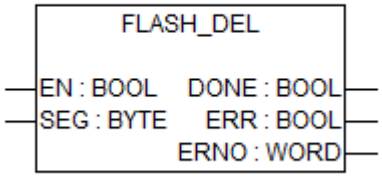
SEG

Data type: BYTE

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO WORD (error number)

The output ERNO indicates an error number (see *Chapter 1.5.3 "Error messages of the AC500 V2 function block libraries" on page 735*). This output has always to be considered together with the outputs DONE and ERR.

The functions FLASH_DEL, FLASH_WRITE and FLASH_READ are executed in the background by the operating system. This procedures can take quite long time, since the PLC user program is processed with priority. Output ERNO then indicates that the block execution is in progress (0x0FFF).

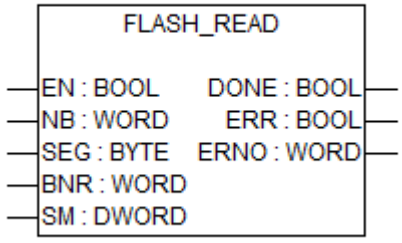
During this phase, the outputs ERR and DONE are set to FALSE.

The inputs and outputs can neither be duplicated nor inverted.

Function call in ST

```
DEL_FLASH(EN := EN_FLASH_DEL,
SEG := SEG_FLASH_DEL);
DONE_FLASH_DEL := DEL_FLASH.DONE;
ERR_FLASH_DEL := DEL_FLASH.ERR;
ERNO_FLASH_DEL := DEL_FLASH.ERNO;
```

FLASH_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block reads a data set from a data segment of the Flash and stores the read data set beginning at the start flag defined by SM. The data of the data set have been previously stored to the Flash using the function block FLASH_WRITE.

Important note:

Access to the Flash is only possible using the function blocks FLASH_WRITE and FLASH_READ.

NB function blocks are read starting at function block BNR within segment SEG and stored starting at address SM.

Either 32 binary data or 16 word data or 8 double word data are read per function block.

One function block contains 34 bytes:

32 bytes of data

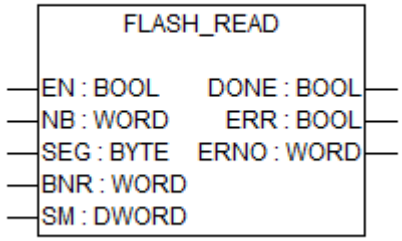
1 byte CRC checksum

1 byte "written identifier"

Reading a data set is triggered once by a FALSE/TRUE edge at input EN. If no error occurred when reading the data, DONE is set to TRUE and the outputs ERR and ERNO are set to FALSE. The data set is stored beginning at the defined start flag SM. Storing the data set can take several PLC cycles.

If an error occurs during reading, DONE and ERR are both set to TRUE. The error type is indicated at output ERNO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

NB

Data type	Default value	Range	Unit
WORD	-	1 ... 1927	-

The number of the data set blocks is specified at input NB. Either 32 binary data or 16 word data or 8 double word data are read per block.

Example:

- SM = ADR(%MW0.0) and NB = 1: Storing data from %MW0.0 to %MW0.15
(1 block = 16 word data)
- SM = ADR(%MW0.0) and NB = 2: Storing data from %MW0.0 to %MW0.31
(2 blocks = 32 word data)

SEG

Data type: BYTE

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

BNR

Data type	Default value	Range	Unit
WORD	-	-	-

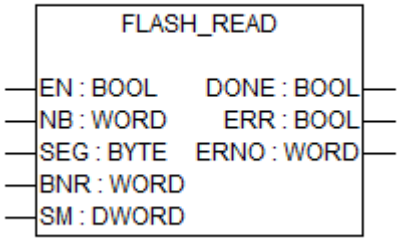
The block number in the data segment is specified at input BNR. Valid values: 0...1926

SM

Data type	Default value	Range	Unit
DWORD	-	-	-

At input SM (source memory), the address of the first variable for storing the data set is specified using an ADR operator.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO WORD The output ERNO indicates an error number(see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*). This output has always to be considered together with the outputs DONE and ERR.

The functions FLASH_DEL, FLASH_WRITE and FLASH_READ are executed in the background by the operating system. This procedures can take quite long time, since the PLC user program is processed with priority. Output ERNO then indicates that the block execution is in progress (0x0FFF).

During this phase, the outputs ERR and DONE are set to FALSE.

The inputs and outputs can neither be duplicated nor inverted.

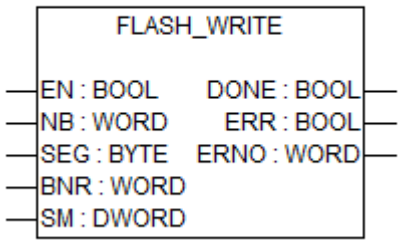
The following figure shows the structure of a Flash segment.

Byte:		**1 | 2**	**3 | 4**	**5 | 6**	**....**	**29 | 30**	**31 | 32**	**33**	**34**
Byte-Offset	**Block-No.**	**Word 1**	**Word 2**	**Word 3**	**....**	**Word 15**	**Word 16**	**CRC**	**Written Identi-fier**
0	0								
34	1								
68	2								
...	...								
65450	1925								
65484	1926								

Function call in ST

```
READ_FLASH(EN := EN_FLASH_READ,
     NB := NB_FLASH_READ,
     SEG := SEG_FLASH_READ,
     BNR := BNR_FLASH_READ,
     SM := SM_FLASH_READ)
DONE_FLASH_READ := READ_FLASH.DONE;
ERR_FLASH_READ := READ_FLASH.ERR;
ERNO_FLASH_READ := READ_FLASH.ERNO;
```

FLASH_WRITE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block writes a data set to a data segment in the Flash. For that purpose, two data segments are available for the AC500. A delete operation (function block FLASH_DEL) always deletes a complete data segment. A data segment consists of 1927 blocks (0 ... 1926). Each block is composed of 34 bytes.

After a delete operation, data can be written only once to each of these 1927 data segment blocks. If a block containing data is to be overwritten with new data, the entire data segment has to be deleted first. In doing so, all data in this segment are lost.

Access to the Flash is only possible using the function blocks FLASH_DEL, FLASH_WRITE and FLASH_READ.

NB blocks are read starting at address SM and stored in the segment SEG starting at block BNR.

Either 32 binary data or 16 word data or 8 double word data are read per block.

One block contains 34 bytes:

32 bytes of data

1 byte CRC checksum

1 byte "written identifier"

(see figure at the end of this block description)

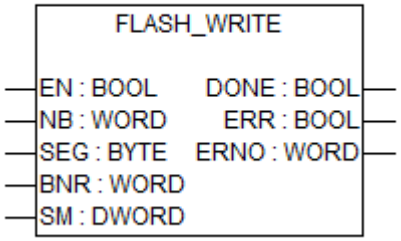
When a write operation of a data set is started (FALSE/TRUE edge at input EN), the data of the data set must not be changed until the end of the write procedure (DONE = TRUE). Storing the data set in the FLASH can take several PLC cycles.

With a FALSE/TRUE edge at input EN, the data set is written once. Until the storage procedure has not been finished (DONE = TRUE), input EN will not be evaluated again.

After the write operation is completed, the block outputs DONE, ERR and ERNO are updated. The storage was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

A new FALSE/TRUE edge at input EN starts a new write operation. Since without a previous deletion of the data segment no new data can be written to blocks which already contain data, the input BNR must point to the next free block for the next write procedure.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

NB

Data type	Default value	Range	Unit
WORD	-	1 ... 1927	-

The number of the data set blocks is specified at input NB. Either 32 binary data or 16 word data or 8 double word data are read per block.

Example:

- SM = ADR(%MW0.0) and NB = 1: Storing data from %MW0.0 to %MW0.15

(1 block = 16 word data)

- SM = ADR(%MW0.0) and NB = 2: Storing data from %MW0.0 to %MW0.31

(2 blocks = 32 word data)

SEG

Data type: BYTE

At input SEG, the number of the data segment in the Flash is specified. In the AC500, controller two data segments are available for the user.

Valid values: 1 and 2 respectively

BNR

Data type	Default value	Range	Unit
WORD	-	-	-

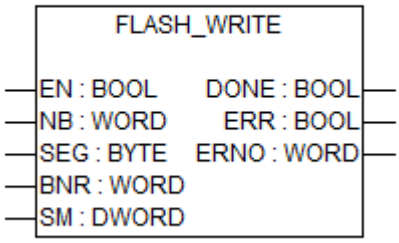
The block number in the data segment is specified at input BNR. Valid values: 0...1926

SM

Data type	Default value	Range	Unit
DWORD	-	-	-

At input SM (source memory), the address of the first variable for storing the data set is specified using an ADR operator.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO WORD

The output ERNO indicates an error number(see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*). This output has always to be considered together with the outputs DONE and ERR.

The functions FLASH_DEL, FLASH_WRITE and FLASH_READ are executed in the background by the operating system. This procedures can take quite long time, since the PLC user program is processed with priority. Output ERNO then indicates that the block execution is in progress (0x0FFF).

During this phase, the outputs ERR and DONE are set to FALSE.

The inputs and outputs can neither be duplicated nor inverted.

The following figure shows the structure of a Flash segment.

Byte:		**1 | 2**	**3 | 4**	**5 | 6**	**....**	**29 | 30**	**31 | 32**	**33**	**34**
Byte-Offset	**Block-No.**	**Word 1**	**Word 2**	**Word 3**	**....**	**Word 15**	**Word 16**	**CRC**	**Written Identifier**
0	0								
34	1								
68	2								
...	...								

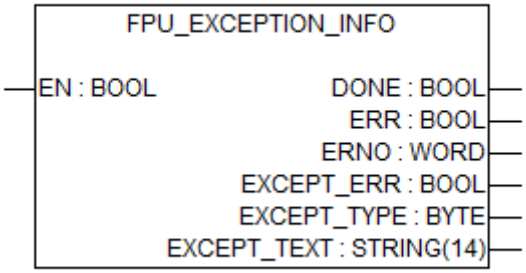
Byte:		1 | 2	3 | 4	5 | 6	....	29 | 30	31 | 32	33	34
Byte-Offset	Block-No.	Word 1	Word 2	Word 3	....	Word 15	Word 16	CRC	Written Identi-fier
65450	1925								
65484	1926								

Function call in ST

Function call in ST

```
WRITE_FLASH(EN := EN_FLASH_WRITE,
     NB := NB_FLASH_WRITE,
     SEG := SEG_FLASH_WRITE,
     BNR := BNR_FLASH_WRITE,
     SM := SM_FLASH_WRITE)
DONE_FLASH_WRITE := WRITE_FLASH.DONE;
ERR_FLASH_WRITE := WRITE_FLASH.ERR;
ERNO_FLASH_WRITE := WRITE_FLASH.ERNO;
```

FPU_EXCEPTION_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Function block with historical values

The function block FPU_EXCEPTION_INFO reads information which has been stored during an FPU exception. The reading operation takes 1 PLC cycle. If data are available, they are provided by setting the outputs DONE = TRUE and ERR = FALSE. If several errors have appeared up to the call of the function block, the last error is shown.

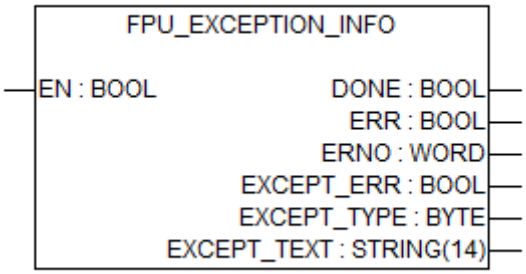
This function block only has to be used with CPUs which have an FPU. The CPU parameter "Reaction on floating point exception" has to be set to "No failure".

The following table represents an overview of the possible return values of the function block.

EXCEPT_ERR	EXCEPT_TYPE	EXCEPT_TEXT
FALSE	16#00	"No error"
TRUE	16#01	"Zero Divide"
TRUE	16#02	"Overflow"
TRUE	16#03	"Underflow"
TRUE	16#04	"Invalid"
TRUE	16#05	"Inexact"
TRUE	16#06	"Function"

The error "Function" occurs, if a result of a function (e.g. SQRT, LN or ACOS) cannot be calculated.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

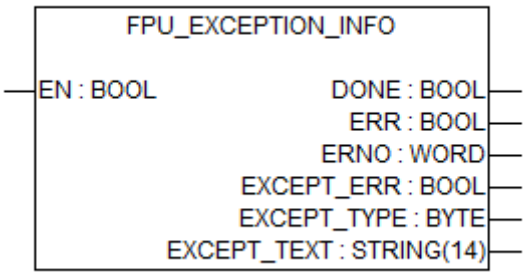
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

EXCEPT_ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output EXCEPT_ERR (exception error) indicates that an exception error has occurred in the FPU.

EXCEPT_TYPE BYTE

Data type	Default value	Range	Unit
BYTE	-	-	-

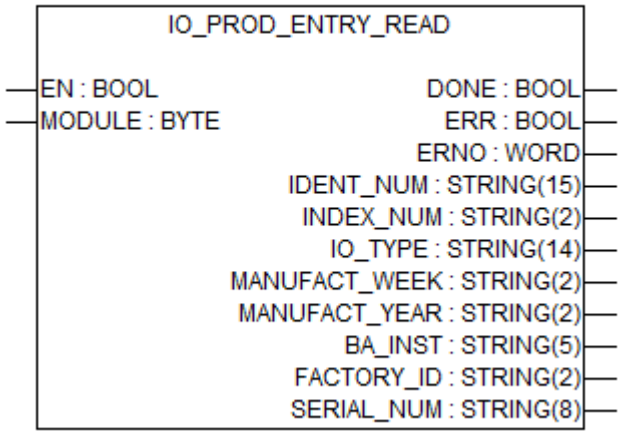
Output EXCEPT_TYPE (exception type) indicates the type of the occurred exception error.

EXCEPT_TEXT Output EXCEPT_TEXT (exception text) indicates in text form which exception error has occurred.

Function call in ST

```
FpuInfo(EN := FpuInfo_EN);


FpuInfo_DONE := FpuInfo.DONE;
FpuInfo_ERR := FpuInfo.ERR;
FpuInfo_ERNO := FpuInfo.ERNO;
FpuInfo_EXCEPT_ERR := FpuInfo.EXCEPT_ERR;
FpuInfo_EXCEPT_TYPE := FpuInfo.EXCEPT_TYPE;
FpuInfo_EXCEPT_TEXT := FpuInfo.EXCEPT_TEXT;
```

IO_PROD_ENTRY_READ

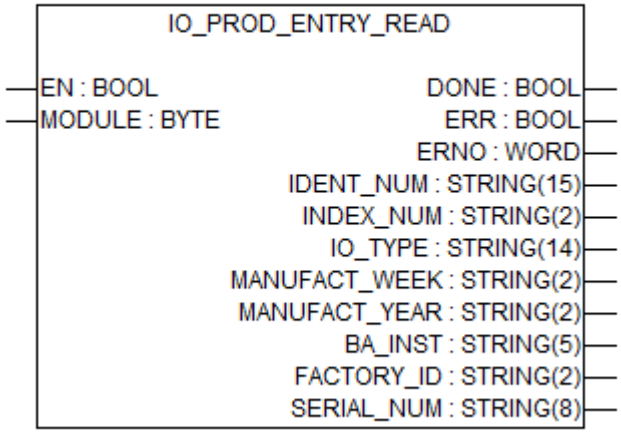


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V2.2.0
Type	Function block with historical values

The function block reads production data from I/O modules.

This function block can be used only for I/O modules which are attached to the local I/O bus master.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

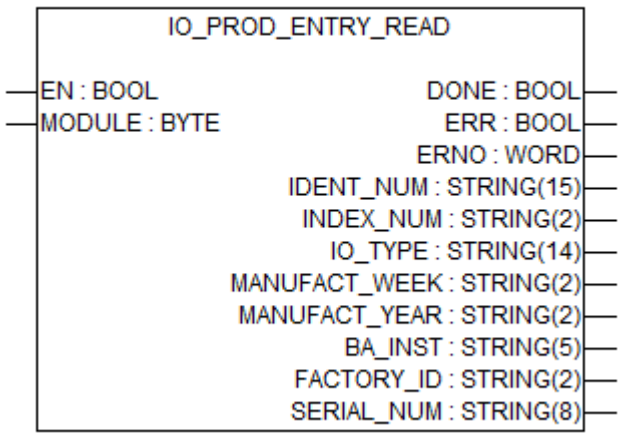
MODULE

Data type	Default value	Range	Unit
BYTE	-	1...n*	-

*) n is the max. number of I/O modules which can be attached to the local I/O bus master

Input MODULE selects the I/O module attached to the local I/O bus master. Valid values are counted from left to right, starting with 1 as the first I/O module right to the CPU.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IDENT_NUM Identification number.

INDEX_NUM Index number.

IO_TYPE Type of the I/O module in plain text.

MANU-FACT_WEEK Calendar week when the I/O module has been manufactured.

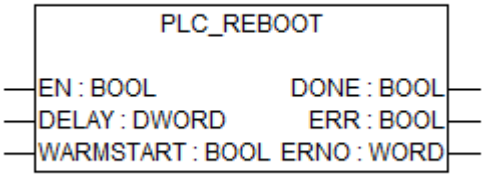
MANU-FACT_YEAR Year when the I/O module has been manufactured.

BA_INST Reserved for future extensions.

FACTORY_ID Factory, where the I/O module has been manufactured.

SERIAL_NUM Serial number of the I/O module.

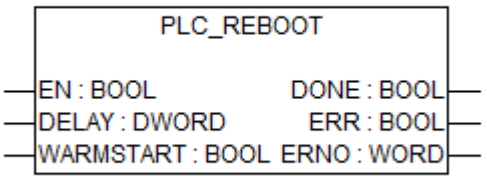
PLC_REBOOT



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block reboots the PLC. Using the input DELAY, a delay can be specified how long to wait until the new start. At the input WARMSTART it can be specified, whether only the firmware shall be re-started or if a cold start shall be performed. A cold start has the same effect as power-off/on.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

DELAY

Data type	Default value	Range	Unit
DWORD	-	-	-

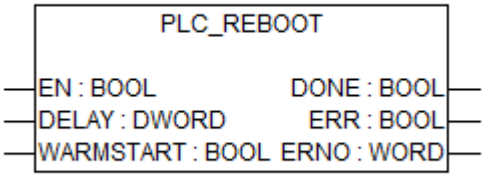
The value at the input DELAY (delay until reboot) specifies how much time will elapse until the new start after the re-boot was triggered at the input EN.

WARMSTART

Data type: BOOL

The input WARMSTART specifies how the PLC behaves in case of a new start. If WARMSTART = TRUE, only the firmware of the PLC is started anew. If WARMSTART = FALSE, the new start is equal to a cold start (like power off/on).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

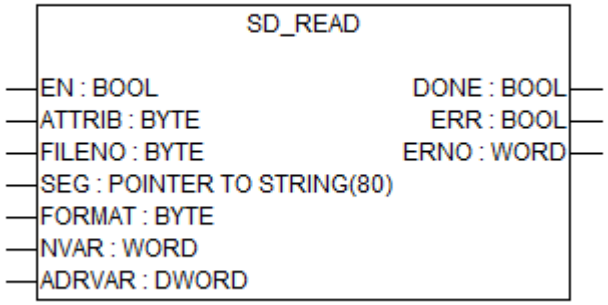
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
PLC_REBOOT(EN := EN_PLC_REBOOT,
           DELAY := DELAY_PLC_REBOOT,
           WARMSTART := WARMSTART_PLC_REBOOT);


DONE_PLC_REBOOT := PLC_REBOOT.DONE;
ERR_PLC_REBOOT  := PLC_REBOOT.ERR;
ERNO_PLC_REBOOT := PLC_REBOOT.ERNO;
```

SD_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The function block SD_READ reads a data set from a file on the memory card and stores the read data set beginning at the start flag defined by ADRVAR.

In this context it has to be observed that the function blocks are mutually interlocked, i.e. it must be ensured that only one function block is active at the same time.

The function block reads a data set in the file on the memory card:

...\UserData\PM5x1\UserDat\USRDATxx.DAT ↳ *Chapter 1.5.4.19.1 “Structure of the file USRDATXX.DAT on the memory card” on page 1500*

Access to the memory card is only possible by using the function blocks SD_WRITE and SD_READ.

The inputs ATTRIB, FILENO, FORMAT, ADRVAR and NVAR determine how many values should be read from which file and in which format on the memory card as well as to which target address they should be stored. Always a complete data set must be read.

Reading a data set from the memory card can take several PLC cycles.

With a FALSE/TRUE edge at input EN, the data set reading is triggered once. Input EN is not evaluated again until the ready message DONE = TRUE is available, i.e. the state of EN is ignored during reading.

After the read operation is completed, the function block outputs DONE, ERR and ERNO are updated. Reading was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

After reading a data set from the memory card, the function block outputs are valid for one cycle. In the next cycle, the outputs DONE, ERR and ERNO are reset to zero. A new FALSE/TRUE edge at input EN starts a new read operation.

Reading example 1:

To read user data from a data file without sectors from the memory card and write them to the PLC, proceed as follows:

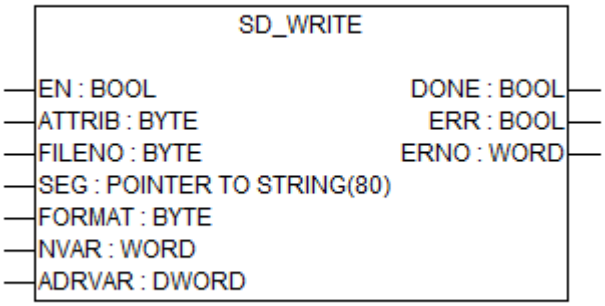
1. Insert the memory card.
2. Read a data set by calling the function block SD_READ with the following settings: EN := TRUE (* FALSE/TRUE edge triggers reading *) ATTRIB := 2 (* open / read *) FILENO := 0..99 (* number of file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data are to be stored.
3. 3. Further data sets can be read with the following settings after the completion message (DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge triggers reading*) ATTRIB := 3 (* continue read *) FILENO := 0..99 (* number of file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data are to be stored If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.
4. To read a further data set and to close the file afterwards, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) is displayed and start the process with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge triggers reading*) ATTRIB := 4 (* read / close *) FILENO := 0...99 (* number of file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data are to be stored If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.
5. To close the file without reading it, call the function block SD_READ with the following settings after the completion message (DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge closes the file *) ATTRIB := 5 (* close *) FILENO := 0..99 (* number of file to be closed *)

Reading example 2:

To read user data from a data file with sectors from the memory card and write them to the PLC, proceed as follows:

1. Insert the memory card.
2. Seek a sector label and read a data set by calling the function block SD_READ with the following settings: EN := TRUE (* FALSE/TRUE edge triggers reading *) ATTRIB := 1 (* open / seek / read *) FILENO := 0..99 (* number of the file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data should be written The read operation is finished successfully if output DONE = TRUE and output ERR = FALSE. A seek error is indicated with ERR = TRUE and ERNO <> 0.
3. Further data sets can be read with the following settings after the completion message (DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge triggers reading*) ATTRIB := 3 (* continue read *) FILENO := 0..99 (* number of file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data are to be stored If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.
4. If you want to read further sectors / data sets, repeat steps 2 and 3.
5. To read a further data set and to close the file afterwards, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) is displayed and start the process with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge triggers reading*) ATTRIB := 4 (* read / close *) FILENO := 0..99 (* number of file to be read *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to which data are to be written If an unexpected sector name or the end of file (EOF) is detected during reading, an appropriate error message is generated.
6. To close the file without reading it, call the function block SD_READ with the following settings after the completion message (DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN: EN := TRUE (* FALSE/TRUE edge closes the file *) ATTRIB := 5 (* close *) FILENO := 0..99 (* number of file to be closed *)

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ATTRIB

Data type	Default value	Range	Unit
BYTE	-	-	-

At input ATTRIB (attribute), the function block operation (action) is specified.

Possible values:

1 - Open file, search sector, read data set (Open, Seek, Read), additionally needed inputs:

FILENO, SEG, FORMAT, NVAR, ADRVAR

2 - Open file, read data set (Open, Read), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

3 - Read next data set (Read), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

4 - Read data set, close file (Read, Close), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

5 - Close file (Close), additionally needed inputs:

FILENO

SEG

Data type	Default value	Range	Unit
DWORD	-	-	-

At input SEG (segment), the start address of the segment label to be searched is specified. A segment label must be enclosed in brackets "[...]".

Examples:

[Values_Tab1]

[Temperature_12]

The length is limited to 32 characters.

FILENO

Data type	Default value	Range	Unit
BYTE	-	0...99	-

At input FILENO (filenumber), the number of the file is specified from which data are to be read.

FORMAT

Data type	Default value	Range	Unit
BYTE	-	-	-

Input Format is used to define the format of the data elements. All elements of one data set must have the same format.

Valid data formats:

00 hex - 0 - BYTE

01 hex - 1 - CHAR

10 hex - 16 - WORD

11 hex - 17 - INT

20 hex - 32 - DWORD

21 hex - 33 - DINT

NVAR

Data type	Default value	Range	Unit
WORD	-	-	-

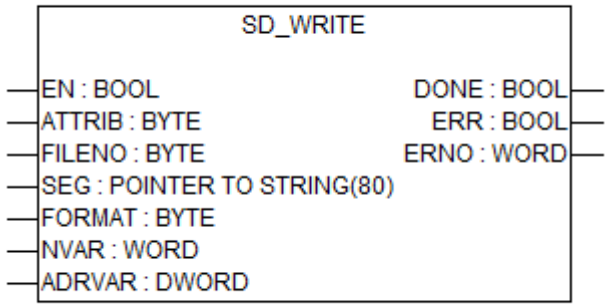
At input NVAR (number of variable), the number of elements of the data set to be read is specified.

ADRVAR

Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADRVAR (address of variable) is used to specify the target start address of the data set. The values of a data set are stored in variables successively arranged in the PLC (e.g. ARRAY, STRING, %M area).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

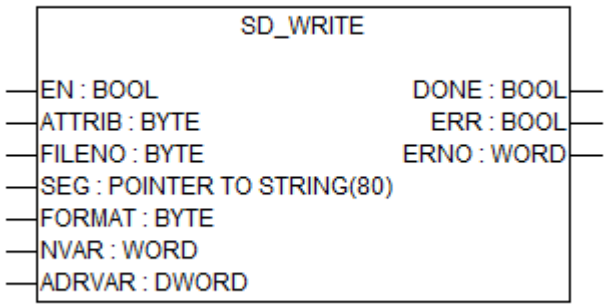
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
READ_SD(EN := EN_SD_READ,
     ATTRIB := ATTRIB_SD_READ,
     FILENO := FILENO_SD_READ,
     SEG := SEG_SD_READ,
     FORMAT := FORMAT_SD_READ,
     NVAR := NVAR_SD_READ,
     ADRVAR := ADRVAR_SD_READ);
DONE_SD_READ := READ_SD.DONE;
ERR_SD_READ := READ_SD.ERR;
ERNO_SD_READ := READ_SD.ERNO;
```

SD_WRITE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The AC500 control system contains a memory card of the type "SD Memory Card" (in short memory card) as external storage medium which is accessed by the PLC like a floppy disk drive. The memory card is used to transfer data between a commercially available PC with memory card interface and the AC500 control system.

Read and write accesses take quite long time, since they are handled in the background by the internal file system of the operating system. When performing a write access, the current file is always copied to a backup file.

In this context it has to be observed that the function blocks are mutually interlocked, i.e. it must be ensured that only one function block is active at the same time.

The function block SD_WRITE writes a data set to a file USRDATxx.DAT on the memory card.

The function block writes a data set to a file on the memory card: ...\User-Data\PM5x1\UserDat\USRDATxx.DAT (see also Structure of the file USRDATXX.DAT on the memory card *Chapter 1.5.4.19.1 “Structure of the file USRDATXX.DAT on the memory card” on page 1500*)

Access to the memory card is only possible by using the function blocks SD_WRITE and SD_READ.

The inputs ATTRIB, FILENO, FORMAT, ADRVAR and NVAR determine how many values should be written to which file and in which format on the memory card as well as from which source address they should be read. To create a readable and EXCEL-compatible file, the individual values are stored in ASCII format, automatically separated by a semicolon. The last value is automatically terminated by a <CR><LF>.

When a write operation of a data set is started (FALSE/TRUE edge at input EN), the data of the data set must not be changed until the end of the write procedure (DONE = TRUE). Storing a data set on the memory card can take several PLC cycles.

Input EN is not evaluated again until the ready message DONE = TRUE is available, i.e. the state of EN is ignored during writing.

After the write operation is completed, the function block outputs DONE, ERR and ERNO are updated. The storage was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred. The error type is signalized at output ERNO.

After storing a data set on the memory card, the function block outputs are valid for one cycle. In the next cycle, the outputs DONE, ERR and ERNO are reset to zero. A new FALSE/TRUE edge at input EN starts a new write operation.

Note: In case of a power failure during the write access, the file USRDATxx.DAT will be corrupted. In order to backup at least the already stored data sets, the file USRDATxx.BAK has to be copied from the SC Card prior to restarting the program. The file can be renamed to USRDATxx.DAT on the PC and can then be used for further storage.

Writing example 1: To store user data to the memory card in a data file without sectors, proceed as follows:

1. Insert the memory card.
2. Write a data set by calling the function block SD_WRITE with the following settings: EN := TRUE (* FALSE/TRUE edge triggers write operation *) ATTRIB := 2 (* write append *) FILENO := 0..99 (* number of the file to be written *) SEG := address of the variable sector name (* any *) FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to be written If no appropriate file can be found, it will be created. The write process is successfully completed if output DONE = TRUE and output ERR = FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.
3. Further data sets can be written with the same function block settings after the completion message (output DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN.

Writing example 2: To store user data to the memory card in a data file with sectors, proceed as follows:

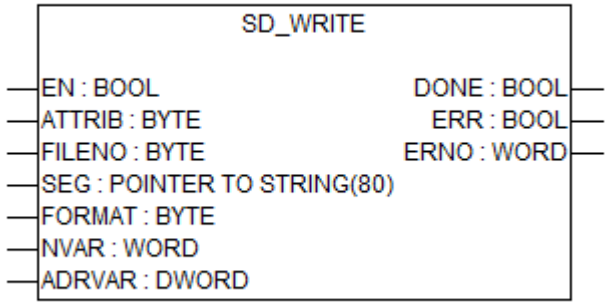
1. Insert the memory card.
2. Write the sector label by calling the function block SD_WRITE with the following settings: EN := TRUE ATTRIB := 3 (* write sector *) FILENO := 0..99 (* number of the file to be written *) SEG := address of the variable sector name If no appropriate file can be found, it will be created. The sector is successfully written when the output DONE:=TRUE and the output ERR:=FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.
3. Write a data set by calling the function block SD_WRITE with the following settings: EN := TRUE (* FALSE/TRUE edge triggers write operation *) ATTRIB := 2 (* write append *) FILENO := 0..99 (* number of the file to be written *) SEG := address of the variable sector name FORMAT := data format NVAR := number of data in the data set ADRVAR := address of the first variable to be written The write process is successfully completed if output DONE = TRUE and output ERR = FALSE. A write error is indicated with ERR = TRUE and ERNO <> 0.
4. Further data sets can be written with the same function block settings after the completion message (output DONE=TRUE) is displayed. This process is started with a FALSE/TRUE edge at input EN.
5. If you want to write further sectors and data sets, repeat steps 2...4. Note:

The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process and a "Open file / Write file / Close file" procedure is performed.

Deleting a file: To delete a data file from the memory card, proceed as follows:

1. Insert the memory card.
2. Call the function block SD_WRITE with the following settings: EN := TRUE ATTRIB := 1 (* delete *) FILENO := 0..99 (* number of the file to be deleted *) SEG, FORMAT, NVAR, ADRVAR – any

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

ATTRIB

Data type	Default value	Range	Unit
BYTE	-	-	-

At input ATTRIB (attribute), the function block operation (action) is specified.

Possible values:

1 - Delete file (Delete), additionally needed inputs:

FILENO

2 - Write data set (Open(create), Write(append), Close), additionally needed inputs:

FILENO, FORMAT, NVAR, ADRVAR

3 - Write segment label (Open(create), Write(append), Close), additionally needed inputs:

FILENO, SEG

SEG

Data type	Default value	Range	Unit
DWORD	-	-	-

At input SEG (segment), the start address of the segment label to be searched is specified. A segment label must be enclosed in brackets "[...]".

Examples: [Values_Tab1]

[Temperature_12]

The length is limited to 32 characters.

FILENO

Data type	Default value	Range	Unit
BYTE	-	0...99	-

At input FILENO, the number of the file is specified to which data are to be written or which should be created or deleted respectively.

FORMAT

Data type	Default value	Range	Unit
BYTE	-	-	-

Input Format is used to define the format of the data elements. All elements of one data set must have the same format.

Valid data formats:

00 hex - 0 - BYTE

01 hex - 1 - CHAR

10 hex - 16 - WORD

11 hex - 17 - INT

20 hex - 32 - DWORD

21 hex - 33 - DINT

NVAR

Data type	Default value	Range	Unit
WORD	-	-	-

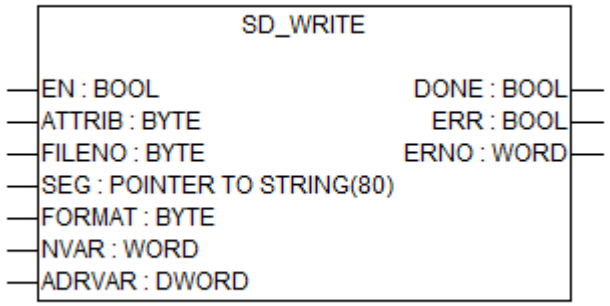
At input NVAR, the number of elements of a data set is specified.

ADRVAR

Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADRVAR is used to specify the start address of the data set. The values of a data set must be available in variables successively stored in the PLC (e.g. ARRAY, STRING, %M area).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

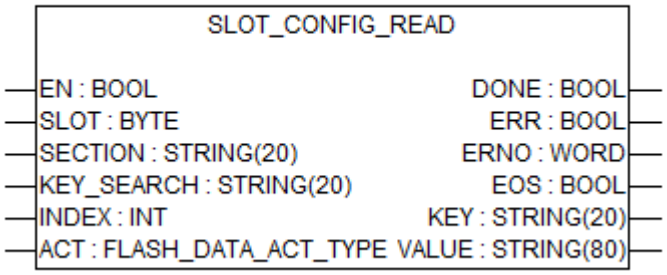
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
WRITE_SD(EN := EN_SD_WRITE,
     ATTRIB := ATTRIB_SD_WRITE,
     FILENO := FILENO_SD_WRITE,
     SEG := SEG_SD_WRITE,
     FORMAT := FORMAT_SD_WRITE,
     NVAR := NVAR_SD_WRITE,
     ADRVAR := ADRVAR_SD_WRITE);
DONE_SD_WRITE := WRITE_SD.DONE;
ERR_SD_WRITE := WRITE_SD.ERR;
ERNO_SD_WRITE := WRITE_SD.ERNO;
```

SLOT_CONFIG_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

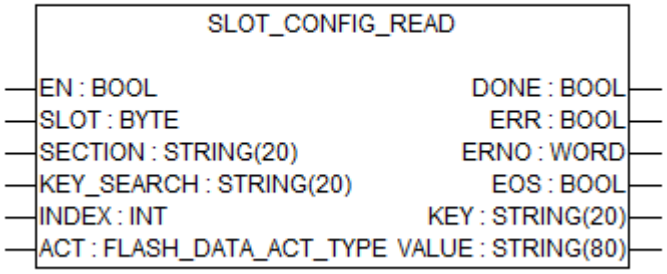
Function block SLOT_CONFIG_READ reads configuration data from flash of a Communication Module in form of KEY/VALUE pairs. One line consists of a pair of key values and is located within the section. The configuration data stored in the Flash are like an "ini" file in their construction, like for example the sdcard.ini on the memory card of the AC500. Example for a line with an IP address in the section Common:

```
[Common]
IP_ADR=192.168.0.1
```

The procedure should be performed in three steps:

1. Initialization (ACT = FLASH_DATA_INIT) of the configuration data with a FALSE/TRUE edge at input EN.
2. Reading (ACT = FLASH_DATA_READ) of the individual values (SECTION, KEY_SEARCH, INDEX) with a FALSE/TRUE edge at input EN.
3. Cancel (ACT = FLASH_DATA_SKIP) the reading with a FALSE/TRUE edge at input EN.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SECTION Data type: STRING[20]

Section, within to search/read. A section has always to be specified, except for the initialization step.

KEY_SEARCH Data type: STRING[20]

If it is searched for the value of a known key, this must be specified at the input KEY_SEARCH. The key is searched within the section which is specified at the input SECTION. In this case, the input INDEX is ignored.

INDEX Data type: INT

The INDEX input is only evaluated, if the KEY_SEARCH input is 0.

With the INDEX input, a specified line of the configuration data within the section, specified at input SECTION, is output.

If INDEX is unequal to 0, exactly this line number is output, if it exists. If INDEX equals 0 and if there is no value at input KEY_SEARCH, the lines of the section (specified at input SECTION) are output sequentially.

In the latter case each time the next line is read with every FALSE/TRUE edge at input EN as long as the output EOS changes to TRUE and thus terminating the section.

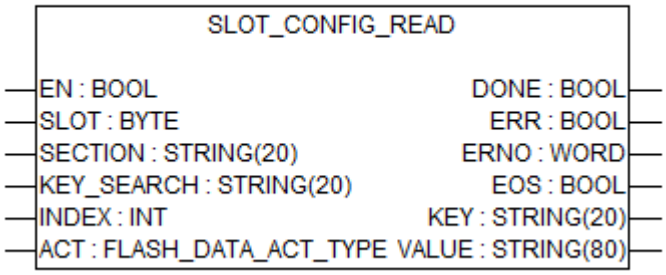
ACT Data type: FLASH_DATA_ACT_TYPE

With the ACT input, the function block can be set to initialize, read the configuration data or to cancel the function block operation.

The input can be set to the following values:

- FLASH_DATA_INIT
  With this input, the function block is initialized for processing of configuration data. The configuration data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input.
  If ACT=FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.
- FLASH_DATA_READ
  Read data from RAM disk with the ACT = FLASH_DATA_READ.
- FLASH_DATA_SKIP
  Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

The function blocks SLOT_CONFIG_READ, SLOT_CONFIG_WRITE and SLOT_PROD_ENTRY_READ are executed in the background by the operating system. This procedures can take quite a long time, since the PLC user program is processed with priority. During this phase ERR=FALSE and DONE=FALSE.

EOS

Data type: BOOL

Output EOS (end of section) indicates whether the end of the section was reached while searching sequentially. This output only must be evaluated, if a search is carried out with INDEX=0 and KEY_SEARCH=0.

KEY

Data type: STRING[20]

Output KEY indicates the found key.

VALUE

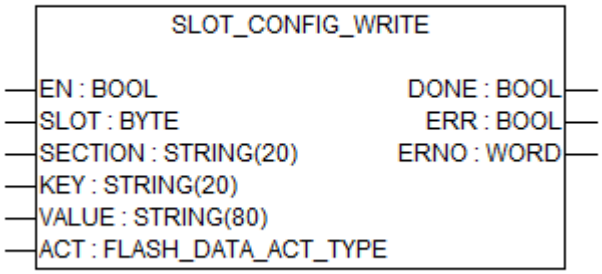
Data type: STRING[80]

Output VALUE indicates the value of the found key.

Function call in ST

```
ConfigRead   (EN         := ConfigRead_EN,
SLOT       := ConfigRead_SLOT,
SECTION    := ConfigRead_SECTION,
KEY_SEARCH := ConfigRead_KEY_SEARCH,
INDEX      := ConfigRead_INDEX,
ACT        := ConfigRead_ACT);
ConfigRead_DONE      := ConfigRead.DONE;
ConfigRead_DONE_ERR  := ConfigRead.ERR;
ConfigRead_ERNO      := ConfigRead.ERNO;
ConfigRead_EOS       := ConfigRead.EOS;
ConfigRead_KEY       := ConfigRead.KEY;
ConfigRead_VALUE     := ConfigRead.VALUE;
```

SLOT_CONFIG_WRITE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

Function block SLOT_CONFIG_WRITE writes configuration data to the flash of a Communication Module in form of KEY/VALUE pairs. The configuration data stored in the Flash are like an "ini" file in their construction, like for example the sdcard.ini on the memory card of the AC500.

Example for a line with an IP address in the section Common:

[Common]

IP_ADR=192.168.0.1

The procedure should be performed in three steps:

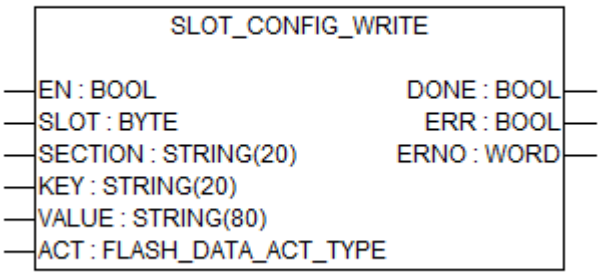
1. Initialization (ACT = FLASH_DATA_INIT) of the configuration data with a FALSE/TRUE edge at input EN
2. Writing (ACT = FLASH_DATA_WRITE) of the individual values (SECTION, KEY, VALUE) with a FALSE/TRUE edge at input EN

   This step is repeated for each entry until all the desired data has been written.

3. Transmit (ACT = FLASH_DATA_SAVE) the data into the Flash and ensuring a permanent data storage in this way.

   Only after the third step has been done, the data is stored in the Flash and will be non-volatile in case of power-down.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SECTION

Data type: STRING[20]

Section to write to.

KEY (key to write)

Data type: STRING[20]

Key to be written.

VALUE (value to write)

Data type: STRING[80]

Value of the key to be written.

ACT

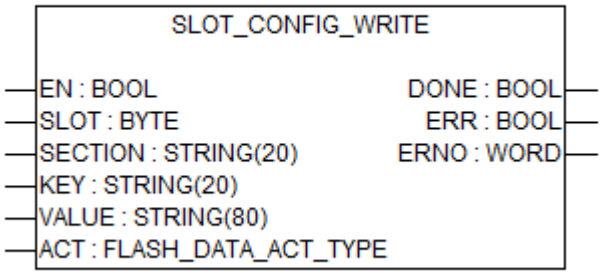
Data type: FLASH_DATA_ACT_TYPE

With the ACT (actual) input, the function block can be set to initialize, write the configuration data or to save, to skip or reset the function block operation.

The input can be set to the following values:

- FLASH_DATA_INIT
  With this input, the function block is initialized for processing of configuration data. The configuration data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input.
  If ACT=FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.
- FLASH_DATA_WRITE
  Write data to a RAM disk with the ACT = FLASH_DATA_WRITE.
- FLASH_DATA_SKIP
  Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.
- FLASH_DATA_RES
  With this input, a reset function is carried out, which deletes all of the configuration data completely.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

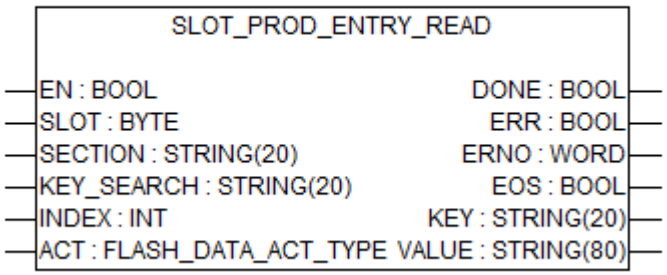
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

The functions SLOT_CONFIG_READ, SLOT_CONFIG_WRITE and SLOT_PROD_ENTRY_READ are executed in the background by the operating system. This procedures can take quite a long time, since the PLC user program is processed with priority. During this phase ERR=FALSE and DONE=FALSE.

Function call in ST

```
ConfigWrite (EN := ConfigWrite_EN,
SLOT := ConfigWrite_SLOT,
SECTION := ConfigWrite_SECTION,
KEY := ConfigWrite_KEY,
VALUE := ConfigWrite_VALUE,
ACT := ConfigWrite_ACT);
ConfigWrite_DONE := ConfigWrite.DONE;
ConfigWrite_ERR := ConfigWrite.ERR;
ConfigWrite_ERNO := ConfigWrite.ERNO;
```

SLOT_PROD_ENTRY_READ



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function block with historical values

The function block reads one line from the production data in the Flash memory of a Communication Module.

One line consists of a pair of key values and is located within a section. The section name of the production data is called "Common".

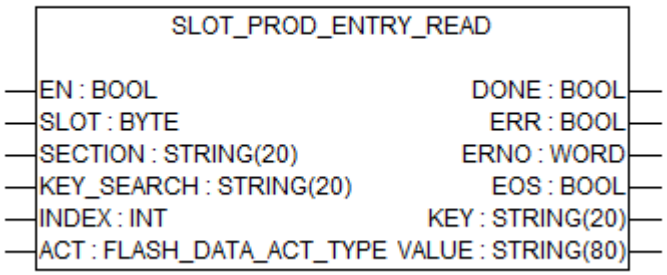
It consists of many keys with the corresponding values, see table:

Production data		Comment
Common	Section Name	Name of the section
BA_INST	Key and value	BA number
IDENT	Key and value	SAP identnumber
IDENT	Key and value	Index of the module
MAC	Key and value	MAC address of the CPU
MANUF_DATE	Key and value	Date of manufacture
MANUF_YEAR	Key and value	Manufacturing year
MANUF_PLACE	Key and value	Place where the PLC was produced
SERIAL_NR	Key and value	Serial number of the PLC
TYPE	Key and value	CPU type

The procedure should be performed in three steps:

1. Initialization (ACT = FLASH_DATA_INIT) of the production data with a FALSE/TRUE edge at input EN.
2. Reading (ACT = FLASH_DATA_READ) of the individual values (SECTION, KEY_SEARCH, INDEX) with a FALSE/TRUE edge at input EN.
3. Cancel (ACT = FLASH_DATA_SKIP) the reading with a FALSE/TRUE edge at input EN.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SECTION Data type: STRING[20]

Section, within to search/read. A section has always to be specified, except for the initialization step.

KEY_SEARCH Data type: STRING[20]

If it is searched for the value of a known key, this must be specified at the input KEY_SEARCH. The key is searched within the section which is specified at the input SECTION. In this case, the input INDEX is ignored.

INDEX Data type: INT

The INDEX input is only evaluated, if the KEY_SEARCH input is 0.

With the INDEX input, a specified line of the configuration data within the section, specified at input SECTION, is output.

If INDEX is unequal to 0, exactly this line number is output, if it exists. If INDEX equals 0 and if there is no value at input KEY_SEARCH, the lines of the section (specified at input SECTION) are output sequentially.

In the latter case each time the next line is read with every FALSE/TRUE edge at input EN as long as the output EOS changes to TRUE and thus terminating the section.

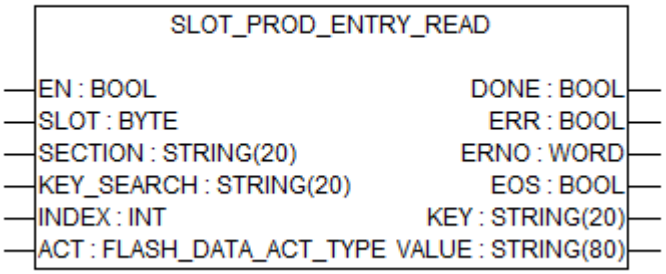
ACT Data type: FLASH_DATA_ACT_TYPE

With the ACT input, the function block can be set to initialize, read the configuration data or to cancel the function block operation.

The input can be set to the following values:

- FLASH_DATA_INIT
  With this input, the function block is initialized for processing of configuration data. The configuration data is copied from the Flash to the RAM, where it can be processed until a re-write into the Flash is performed via the SAVE input.
  If ACT=FLASH_DATA_INIT, all the other inputs are ignored. If the initialization process is performed, all the previous changes are discarded and the original configuration data is copied from the Flash.
- FLASH_DATA_READ
  Read data from RAM disk with the ACT = FLASH_DATA_READ.
- FLASH_DATA_SKIP
  Skip current changes in RAM disk by using FLASH_DATA_SKIP at the input ACT.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

The functions CPU_CONFIG_READ, CPU_CONFIG_WRITE and CPU_PROD_ENTRY_READ are executed in the background by the operating system. This procedures can take quite a long time, since the PLC user program is processed with priority. During this phase ERR=FALSE and DONE=FALSE.

EOS

Data type: BOOL

Output EOS (end of section) indicates whether the end of the section was reached while searching sequentially. This output only must be evaluated, if a search is carried out with INDEX=0 and KEY_SEARCH=0.

KEY

Data type: STRING[20]

Output KEY indicates the found key.

VALUE

Data type: STRING[80]

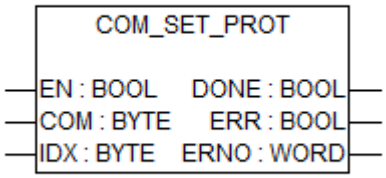
Output VALUE indicates the value of the found key.

Function call in ST

```
CPU_PROD_ENTRY_READ(EN := EN_CPU_PROD_ENTRY_READ,
SECTION := SECTION_CPU_PROD_ENTRY_READ,
KEY_SEARCH := KEY_SEARCH_CPU_PROD_ENTRY_READ,
INDEX := INDEX_CPU_PROD_ENTRY_READ,
ACT := ACT_CPU_PROD_ENTRY_READ);
DONE_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.DONE;
ERR_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.ERR;
ERNO_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.ERNO;
EOS_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.EOS;
KEY_CPU_PROD_ENTRY_READ := CPU_PROD_ENTRY_READ.KEY;
VALUE_CPU_PROD_ENTRY_READ:= CPU_PROD_ENTRY_READ.VALUE;
```

1.5.4.19.3 Programs

COM_SET_PROT



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V2.0
Type	Program

With the function block COM_SET_PROT, the serial interfaces of the CPU can actively be set on a predefined protocol or changed between several protocols from the user program.

The block COM_SET_PROT can be used for different functions:

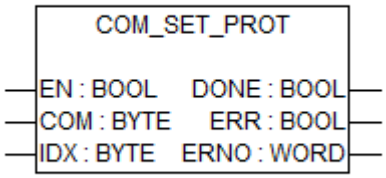
- Switching between two different protocols, for example ASCII/Modbus
- Switching the interface parameters of a protocol, for example changing the transmission rate
- Re-initialization of an interface protocol (for example, if an interface "hangs up")
- Switching between "Online access" and ASCII/Modbus/SysLibCom depending on the current PLC mode, for example STOP=Online access, RUN=Modbus (or ASCII, SysLibCom). In this case, the parameter "Enable login" does not have to be activated and the interface can use other interface parameters than required for "Online access" (see the following program example).

With the setting "COMx "“ Multi[SLOT]" in the configuration, several protocols can be predefined per COM. Using the function block COM_SET_PROT, the user can switch between these protocols from the user program. The rising edge (FALSE->TRUE) at the input EN activates the protocol which was selected over the inputs COM and IDX.

If the function block is used with a serial interface on which only one protocol was defined, the interface is initialized newly and the protocol started again.

Other applications of the function block can be e.g. a change of the transmission rate or also a RESET defined of a protocol.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At input COM (communication), the number of the serial input is defined.

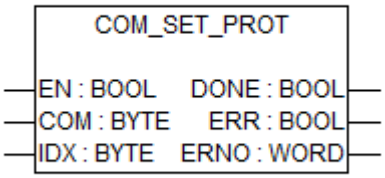
COM = 1: COM1, COM = 2: COM2

IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

The index number of the protocol, which is to be activated on the serial interface COM, is indicated at the input IDX. The assignment of the index number to the individual protocols is carried out in the configuration. If several protocols are defined in the setting "COMx "“ Multi[SLOT]", the index number IDX = 0 corresponds to the protocol in the first place, and IDX = 1 corresponds to the protocol in the second place. In all other cases, if only one protocol is defined per interface, the index number IDX = 0 must be used.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

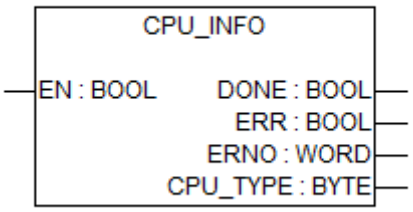
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
COM_SET_PROT (EN  := ComSetProtocol_EN,
              COM := ComSetProtocol_COM,
              IDX := ComSetProtocol_IDX);


ComSetProtocol_DONE := COM_SET_PROT.DONE;
ComSetProtocol_ERR  := COM_SET_PROT.ERR;
ComSetProtocol_ERNO := COM_SET_PROT.ERNO;
```

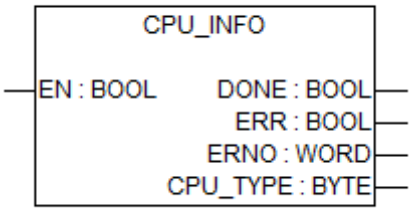
CPU_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program CPU_INFO reads the CPU type.

Input description



EN

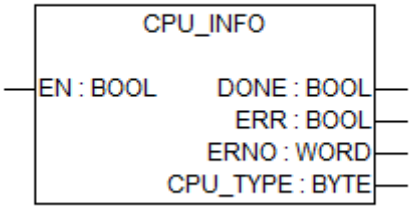
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

CPU_TYPE

Data type	Default value	Range	Unit
BYTE	-	-	-

CPU_TYPE outputs the type of the CPU.

The following values are possible:

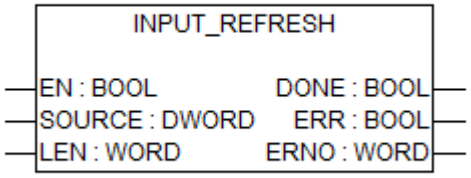
```
Value CPU type


20   PM571
21   PM581
22   PM591
23   PM582
25   CM574
26   PM590
27   not used
28   not used
29   PM583-ETH
30   not used
31   PM554
32   PM564
33   PM554-ETH
34   PM564-ETH
35   SM560
36   PM572
37   PM573-ETH
38   PM592-ETH
39   not used
40   PM590-ETH
41   PM591-ETH
42   PM556-ETH
44   PM590-ARC
45   PM591-2ETH
46   PM595-4ETH-F (internal Communication Module via Communication
Module bus)
47   PM595-4ETH-M (internal Communication Module via Communication
```

```
Module bus)
48   PM595-4ETH-F (internal Communication Module via PCIe)
49   PM595-4ETH-M (internal Communication Module via PCIe)
50   PM585-ETH
51   PM590-ARC-ETH
```

Function call in ST

```
CPU_INFO(EN := CpuInfo_EN);


CpuInfo_DONE := CPU_INFO.DONE;
CpuInfo_ERR := CPU_INFO.ERR;
CpuInfo_ERNO := CPU_INFO.ERNO;
CpuInfo_CPU_TYPE := CPU_INFO.CPU_TYPE;
```

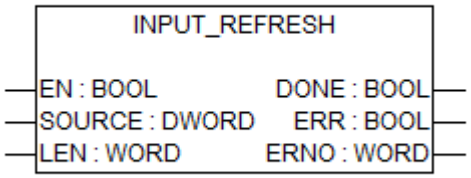
INPUT_REFRESH



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Program

Program INPUT_REFRESH copys the IO input data image into the image area consistently.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

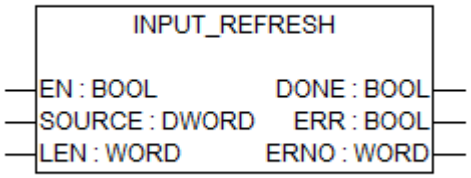
SOURCE Data type: DWORD

At input SOURCE, the address of the first data byte in the source IO area has to be specified.

LEN (length) Data type: BOOL

At input LEN, the number of bytes to be copied has to be specified.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

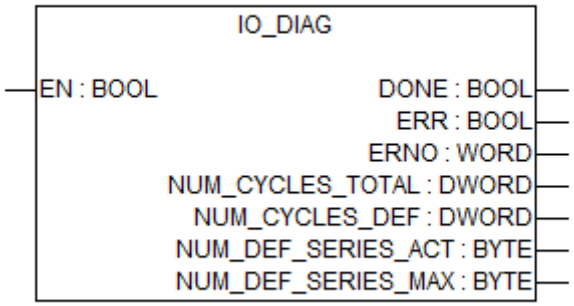
ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during Function Block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input, or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

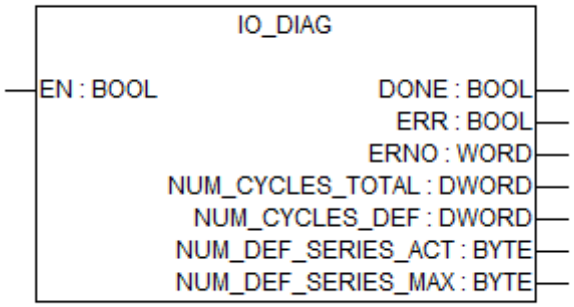
IO_DIAG



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values

The Program IO_DIAG reads the diagnosis data of the I/O bus.

Input description



EN

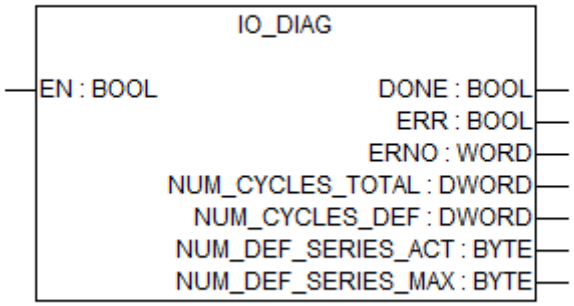
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_CYCLES_TOTAL

Data type	Default value	Range	Unit
DWORD	-	-	-

NUM_CYCLES_TOTAL displays the total number of I/O bus cycles performed since system start.

NUM_CYCLES_DEF

Data type	Default value	Range	Unit
DWORD	-	-	-

NUM_CYCLES_DEF displays the total number of defective I/O bus cycles occurred since system start.

NUM_DEF_SERIES_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_DEF_SERIES_ACT displays the actual number of successively occurred defective I/O bus cycles.

NUM_DEF_SERIES_MAX

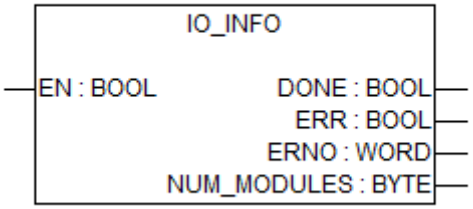
Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_DEF_SERIES_MAX displays the maximum number of successively occurred defective I/O bus cycles captured since system start.

Function call in ST

```
IO_DIAG (EN := IoDiag_EN);


IoDiag_DONE := IO_DIAG.DONE;
IoDiag_ERR := IO_DIAG.ERR;
IoDiag_ERNO := IO_DIAG.ERNO;
IoDiag_NUM_CYCLES_TOTAL := IO_DIAG.NUM_CYCLES_TOTAL;
IoDiag_NUM_CYCLES_DEF := IO_DIAG.NUM_CYCLES_DEF;
IoDiag_NUM_DEF_SERIES_ACT := IO_DIAG. NUM_DEF_SERIES_ACT;
IoDiag_NUM_DEF_SERIES_MAX := IO_DIAG. NUM_DEF_SERIES_MAX;
```

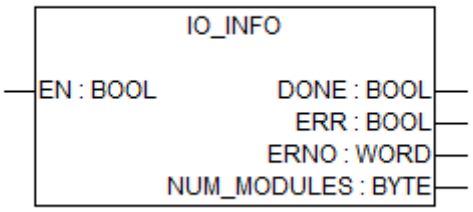
IO_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program IO_INFO displays the number of devices connected to the I/O bus.

Input description



EN

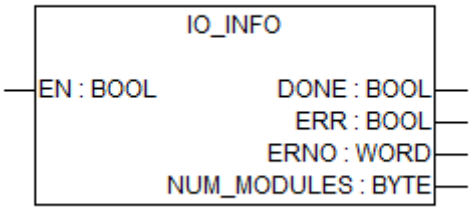
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

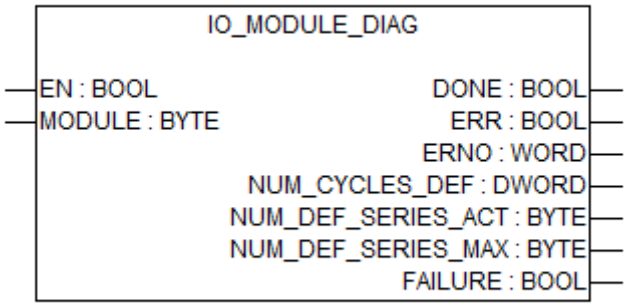
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_MODULES BYTE NUM_MODULES displays the number of devices connected to the I/O bus.

Function call in ST

```
IO_INFO (EN := IoInfo_EN);


IoInfo_DONE := IO_INFO.DONE;
IoInfo_ERR := IO_INFO.ERR;
IoInfo_ERNO := IO_INFO.ERNO;
IoInfo_NUM_ MODULES := IO_INFO.NUM_ MODULES;
```

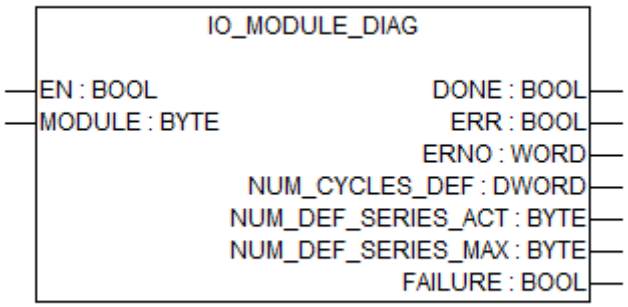
IO_MODULE_DIAG



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program IO_MODULE_DIAG reads the module diagnosis data of the I/O Bus.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

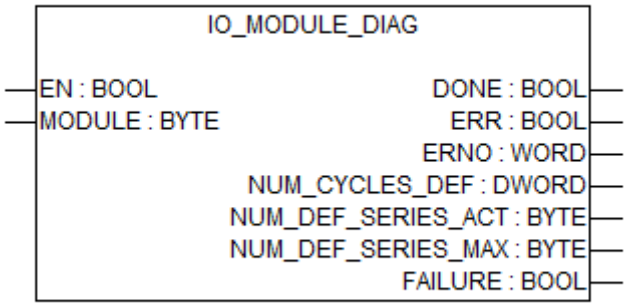
While it is executed its inputs are continuously evaluated.

MODULE

Data type	Default value	Range	Unit
BYTE	-	1..10	-

Module position on bus.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_CYCLES_DEF

Data type	Default value	Range	Unit
DWORD	-	-	-

NUM_CYCLES_DEF (number of cyclesdefective) displays the total number of defective I/O bus cycles occurred for the selected module since system start.

NUM_DEF_SERIES_ACT

Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_DEF_SERIES_ACT (number of defective cycles in seriesactual) displays the actual number of successive defective I/O bus cycles occurred for the selected module.

NUM_DEF_SERIES_MAX

Data type	Default value	Range	Unit
BYTE	-	-	-

NUM_DEF_SERIES_MAX (number of defective cycles in seriesmaximal) displays the maximum number of successive defective I/O bus cycles occurred for the selected module since system start.

FAILURE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

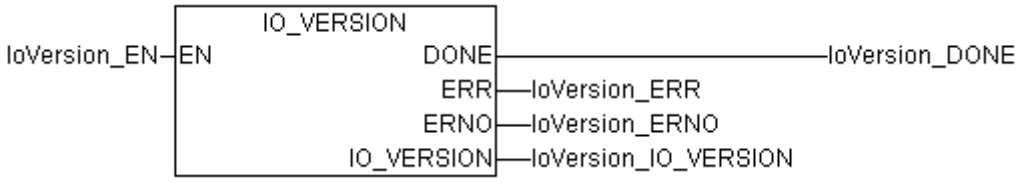
FAILURE indicates that the maximum allowed number of successively occurring defective I/O bus cycles has been exceeded. In this case the module has to be considered as failed. A failure of one module results in a stop of the entire I/O bus.

Function call in ST

```
IO_MODULE_DIAG (EN := IoModuleDiag_EN,
                MODULE := IoModuleDiag_MODULE);


IoModuleDiag_DONE := IO_ MODULE_DIAG.DONE;
IoModuleDiag_ERR := IO_ MODULE_DIAG.ERR;
IoModuleDiag_ERNO := IO_ MODULE_DIAG.ERNO;
```

```
IoModuleDiag_NUM_CYCLES_DEF := IO_ MODULE_DIAG.NUM_CYCLES_DEF;
IoModuleDiag_NUM_DEF_SERIES_ACT := IO_ MODULE_DIAG.
NUM_DEF_SERIES_ACT;
IoModuleDiag_NUM_DEF_SERIES_MAX:= IO_ MODULE_DIAG. NUM_DEF_SERIES_MAX;
IoModuleDiag_FAILURE := IO_ MODULE_DIAG.FAILURE;
```

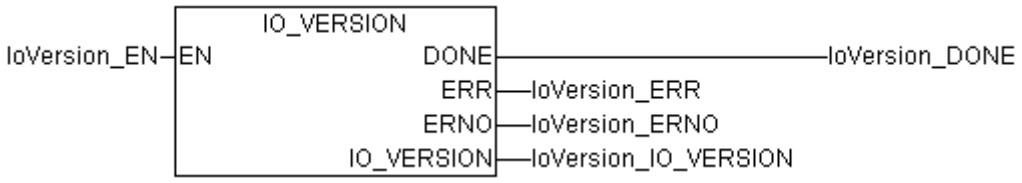
IO_DRIVER_VERSION



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program IO_DRIVER_VERSION reads the version of the I/O bus driver.

Input description



EN

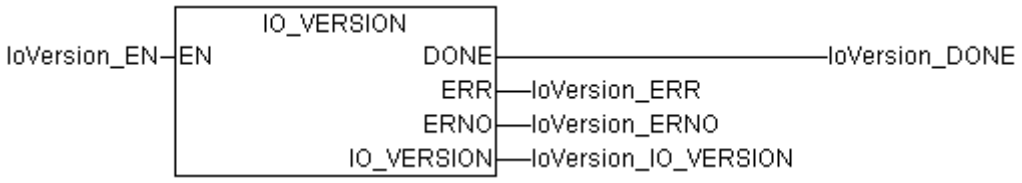
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IO_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

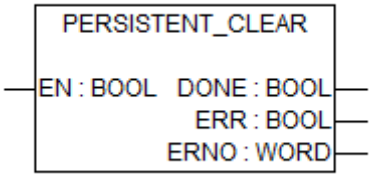
IO_VERSION (I/O bus driver version) outputs the version of the I/O bus driver.

Example: IO_VERSION = 1000 -> V1.0.0.0

Function call in ST

```
IO_VERSION (EN := IoVersion_EN);


IoVersion_DONE := IO_VERSION.DONE;
IoVersion_ERR := IO_VERSION.ERR;
IoVersion_ERNO := IO_VERSION.ERNO;
IoVersion_IO_VERSION := IO_VERSION.IO_VERSION;
```

PERSISTENT_CLEAR

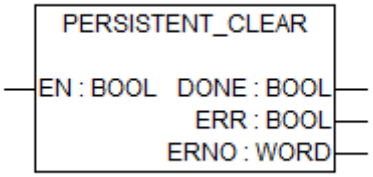


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the function block PERSISTENT_CLEAR, all data in the areas PERSISTENT or %R are deleted.

The use of the Program requires that a valid PERSISTENT area or %R area is set in the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

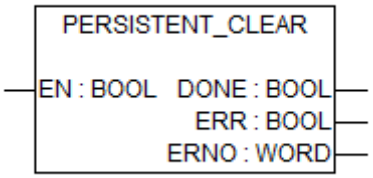
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

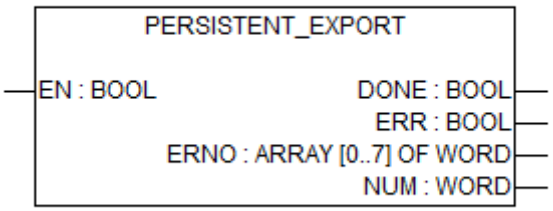
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
PERSISTENT_CLEAR (EN := PersistentClear_EN);


PersistentClear_DONE := PERSISTENT_CLEAR.DONE;
PersistentClear_ERR  := PERSISTENT_CLEAR.ERR;
PersistentClear_ERNO := PERSISTENT_CLEAR.ERNO;
```

PERSISTENT_EXPORT

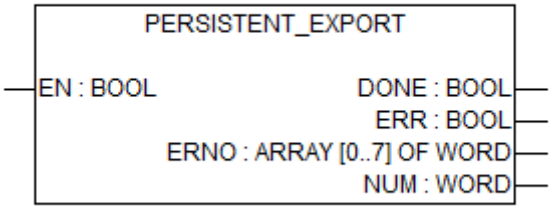


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program PERSISTENT_EXPORT, all data in the areas PERSISTENT or %R are written from the RAM-DISC to the SD Card.

The use of the Program requires that a valid PERSISTENT area or %R area is set in the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

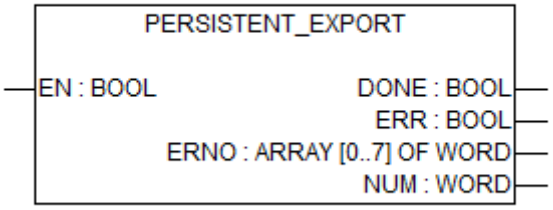
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

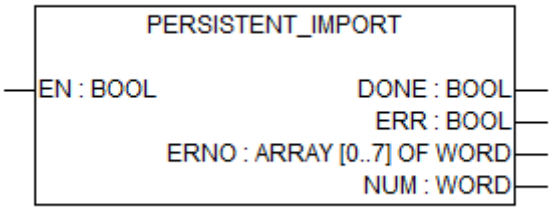
NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in ST

```
PERSISTENT_EXPORT (EN := PersistentExport_EN);


PersistentExport_DONE := PERSISTENT_EXPORT.DONE;
PersistentExport_ERR  := PERSISTENT_EXPORT.ERR;
PersistentExport_ERNO := PERSISTENT_EXPORT.ERNO;
PersistentExport_NUM  := PERSISTENT_EXPORT.NUM;
```

PERSISTENT_IMPORT

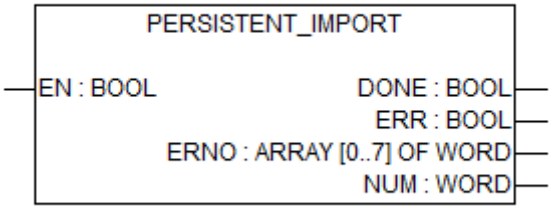


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program PERSISTENT_IMPORT, all data in the areas PERSISTENT or %R are written from the memory card to the RAM-DISC.

The use of the Program requires that a valid PERSISTENT area or %R area is set in the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

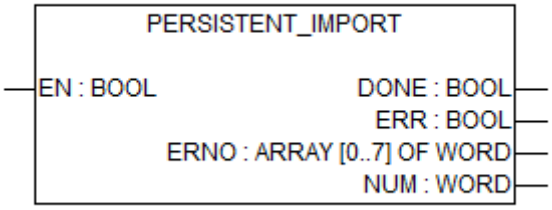
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

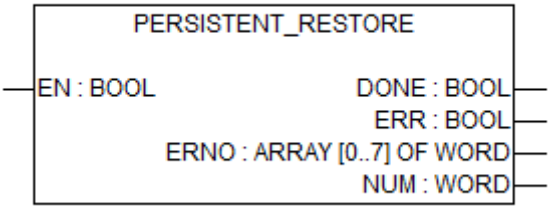
NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in ST

```
PERSISTENT_IMPORT (EN := PersistentImport_EN);


PersistentImport_DONE := PERSISTENT_IMPORT.DONE;
PersistentImport_ERR  := PERSISTENT_IMPORT.ERR;
PersistentImport_ERNO := PERSISTENT_IMPORT.ERNO;
PersistentImport_NUM  := PERSISTENT_IMPORT.NUM;
```

PERSISTENT_RESTORE

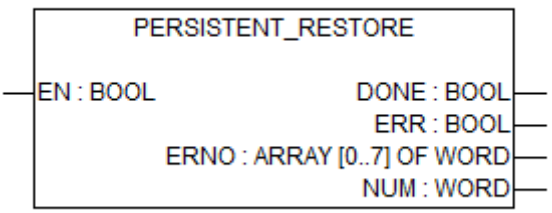


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program PERSISTENT_RESTORE, all data in the areas PERSISTENT or %R are written from the RAM-DISC to the SRAM.

The use of the Program requires that a valid PERSISTENT area or %R area is set in the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

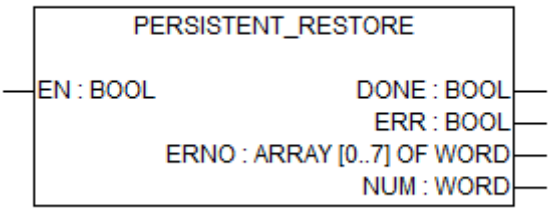
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

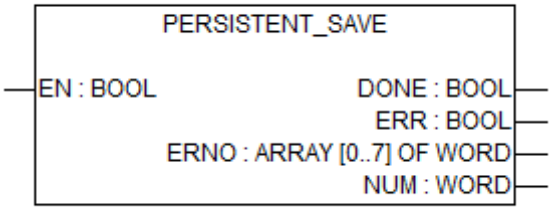
NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in ST

```
PERSISTENT_RESTORE (EN := PersistentRestore_EN);


PersistentRestore_DONE := PERSISTENT_RESTORE.DONE;
PersistentRestore_ERR  := PERSISTENT_RESTORE.ERR;
PersistentRestore_ERNO := PERSISTENT_RESTORE.ERNO;
PersistentRestore_NUM  := PERSISTENT_RESTORE.NUM;
```

PERSISTENT_SAVE

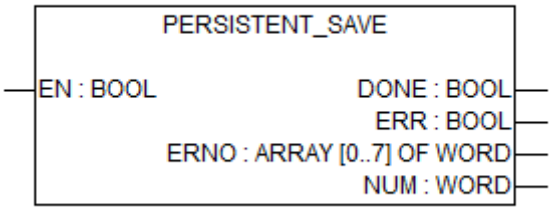


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program PERSISTENT_SAVE, all data in the areas PERSISTENT or %R are written from the SRAM to the RAM-DISC.

The use of the Program requires that a valid PERSISTENT area or %R area is set in the PLC Configuration of the CPU.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

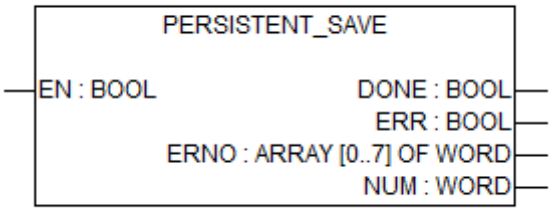
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

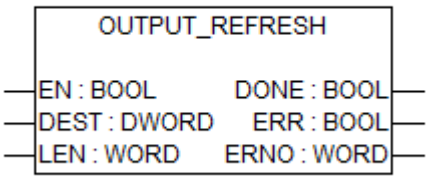
NUM WORD (number of areas)

The number of written segments is available at output NUM.

Function call in ST

```
PERSISTENT_SAVE (EN := PersistentSave_EN);


PersistentSave_DONE := PERSISTENT_SAVE.DONE;
PersistentSave_ERR  := PERSISTENT_SAVE.ERR;
PersistentSave_ERNO := PERSISTENT_SAVE.ERNO;
PersistentSave_NUM  := PERSISTENT_SAVE.NUM;
```

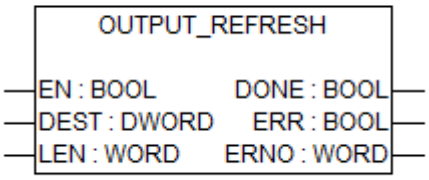
OUTPUT_REFRESH



Program OUPUT_REFRESH copys the image area into the IO image consistently.

Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Program

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

DEST (destination)

Data type: BOOL

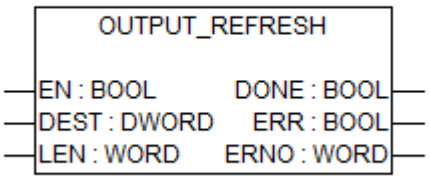
At input DEST, the address of the first data byte in the destination IO area has to be specified.

LEN (length)

Data type: BOOL

At input LEN, the number of bytes to be copied has to be specified.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR (error)

Data type: BOOL

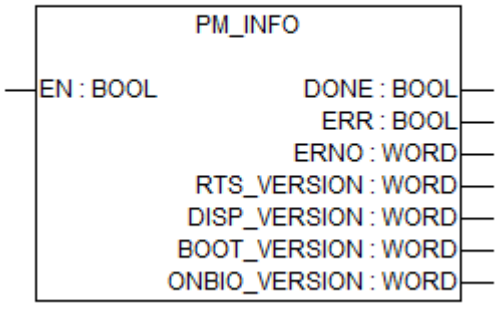
Output ERR indicates whether an error occurred during Function Block processing. This output always has to be considered together with output DONE. If DONE is TRUE and ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input, or if an error occurred during job processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

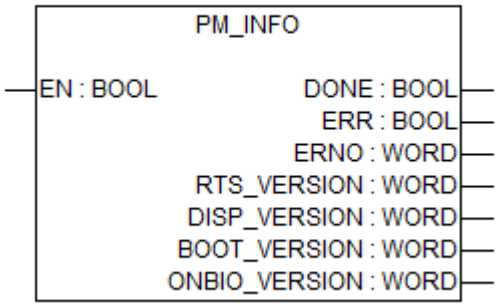
PM_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V2.2.0
Type	Program

The Program PM_INFO reads the software version information of a PLC.

Input description



EN

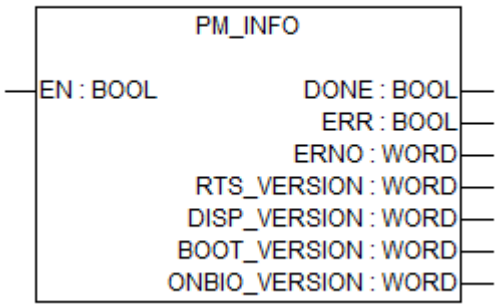
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RTS_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

Output RTS_VERSION (runtime system version) delivers the version of the runtime system in BCD code (e. g. 16#10 means version 1.0)

DISP_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

Output DISP_VERSION (display version) delivers the version of the PLC display firmware (not for AC500 eCo-CPUs) in BCD code (e. g. 16#10 means version 1.0).

BOOT_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

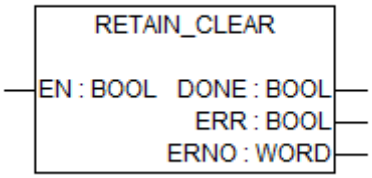
Output BOOT_VERSION (bootcode version) delivers the version of the PLC bootcode in BCD code (e. g. 16#10 means version 1.0).

ONBIO_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

Output ONBIO_VERSION (onboard IO version) delivers the version of the PLC onboard IOs (AC500 eCo-CPUs only) in BCD code (e. g. 16#10 means version 1.0).

RETAIN_CLEAR

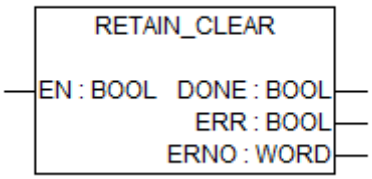


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program RETAIN_CLEAR, all data in the retain area are deleted.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

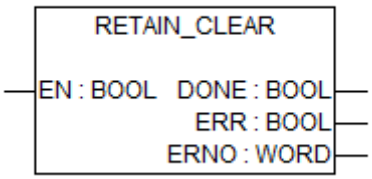
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

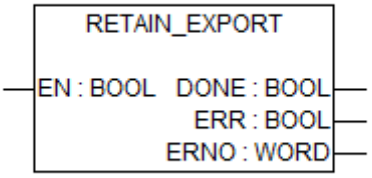
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RETAIN_CLEAR (EN := RetainClear_EN);


RetainClear_DONE := RETAIN_CLEAR.DONE;
RetainClear_ERR  := RETAIN_CLEAR.ERR;
RetainClear_ERNO := RETAIN_CLEAR.ERNO;
```

RETAIN_EXPORT

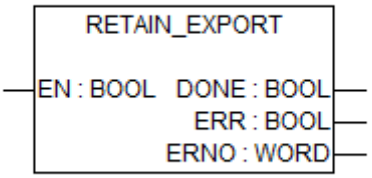


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program RETAIN_EXPORT, all data in the retain area are written from the RAM-DISC to the memory card.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

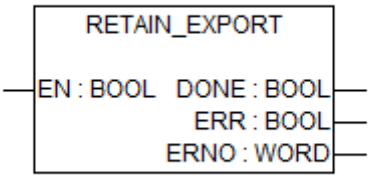
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

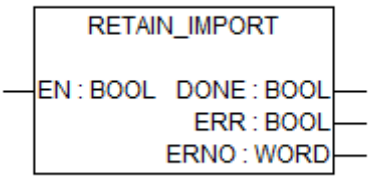
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RETAIN_EXPORT (EN := RetainExport_EN);


RetainExport_DONE := RETAIN_EXPORT.DONE;
RetainExport_ERR  := RETAIN_EXPORT.ERR;
RetainExport_ERNO := RETAIN_EXPORT.ERNO;
```

RETAIN_IMPORT

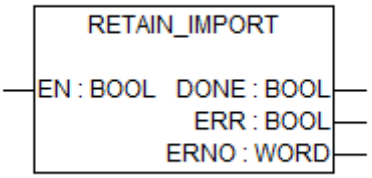


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program RETAIN_IMPORT, all data in the retain area are written from the memory card to the RAM-DISC.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

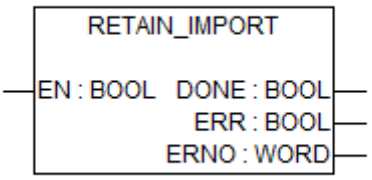
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

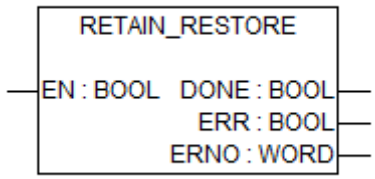
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RETAIN_IMPORT (EN := RetainImport_EN);


RetainImport_DONE := RETAIN_IMPORT.DONE;
RetainImport_ERR  := RETAIN_IMPORT.ERR;
RetainImport_ERNO := RETAIN_IMPORT.ERNO;
```

RETAIN_RESTORE

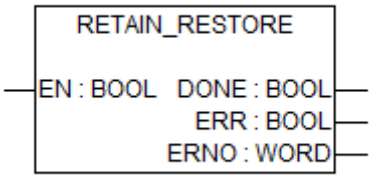


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1:2
Type	Program

With a rising edge (False -> True) at input EN of the Program RETAIN_RESTORE, all data in the retain area are written from the RAM-DISC to the SRAM.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

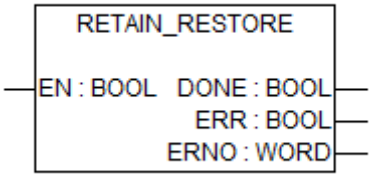
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

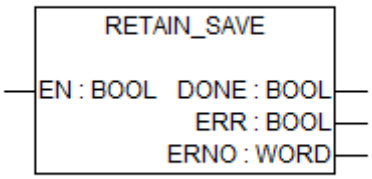
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RETAIN_RESTORE (EN := RetainRestore_EN);


RetainRestore_DONE := RETAIN_RESTORE.DONE;
RetainRestore_ERR  := RETAIN_RESTORE.ERR;
RetainRestore_ERNO := RETAIN_RESTORE.ERNO;
```

RETAIN_SAVE

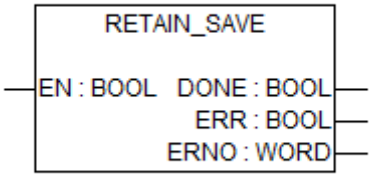


Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.2
Type	Program

With a rising edge (False -> True) at input EN of the Program RETAIN_SAVE, all data in the retain area are written from the SRAM to the RAM-DISC.

Retain variables are declared with the keyword RETAIN. These variables keep their values after an uncontrolled abort as well as after a normal switch off/on of the control system (or with the command 'online' 'reset'). At a new start of the program work is continued with the stored values.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

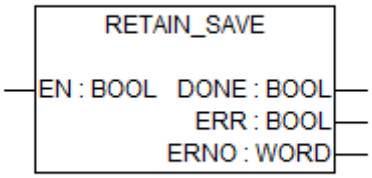
The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

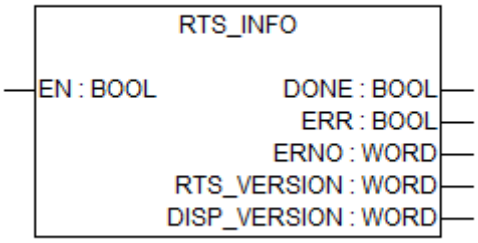
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RETAIN_SAVE (EN := RetainSave_EN);


RetainSave_DONE := RETAIN_SAVE.DONE;
RetainSave_ERR  := RETAIN_SAVE.ERR;
RetainSave_ERNO := RETAIN_SAVE.ERNO;
```

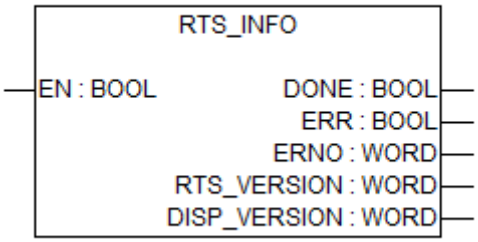
RTS_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program RTS_INFO reads the version of the CPU runtime system.

Input description



EN

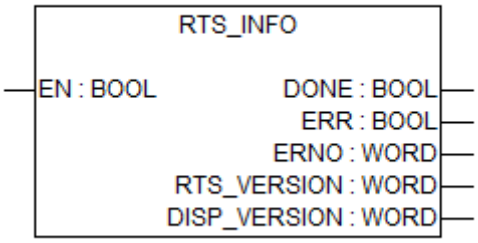
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

RTS_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

RTS_VERSION (RTSversion) outputs the version of the CPU runtime system. The upper BYTE of the entry represents the main version, the lower BYTE represents the subversion of the runtime system.

> ***Different data format display***
> *Examples:*
> *Runtime system ≤ V2.3.3 = RTS_VERSION 16#0233*
> *Runtime system ≥ V2.4.0 = RTS_VERSION 16#2400*

DISP_VERSION

Data type	Default value	Range	Unit
WORD	-	-	-

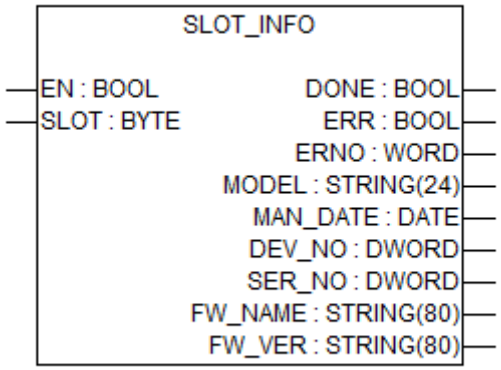
DISP_VERSION (display version) outputs the software version of the display. The upper BYTE of the entry represents the main version, the lower BYTE represents the subversion of the display software.

Example: DISP_VERSION = 16#0110 -> V01.16

Function call in ST

```
RTS_INFO (EN := RtsInfo_EN);


RtsInfo_DONE := RTS_INFO.DONE;
RtsInfo_ERR := RTS_INFO.ERR;
RtsInfo_ERNO := RTS_INFO.ERNO;
RtsInfo_RTS_VERSION := RTS_INFO.RTS_VERSION;
RtsInfo_DISP_VERSION := RTS_INFO.DISP_VERSION;
```

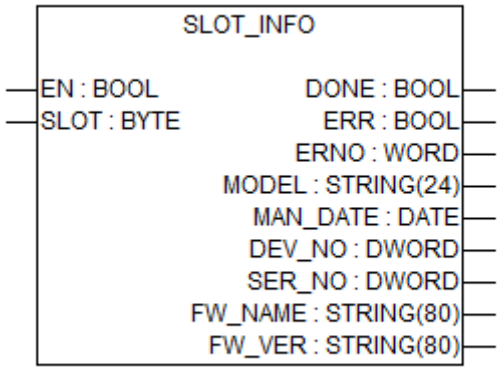
SLOT_INFO



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program SLOT_INFO reads information from the slot from the connected device (Communication module).

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

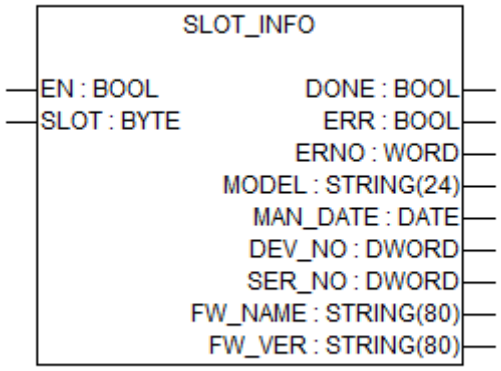
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

MODEL

Data type: STRING(24)

The output MODEL provides the type of the device which is connected to the selected slot. The output is in plain text.

MAN_DATE

Data type: DATE

The output MAN_DATE (date of manufacture)provides the date of manufacture of the device which is connected to the selected slot.

DEV_NO

Data type	Default value	Range	Unit
DWORD	-	-	-

The output DEV_NO (device number) provides the manufacturer-specific number of the device which is connected to the selected slot.

SER_NO

Data type	Default value	Range	Unit
DWORD	-	-	-

The output SER_NO (serial number) provides the serial number of the device which is connected to the selected slot.

FW_NAME

Data type: STRING(16)

The output FW_NAME (firmware name) provides the firmware name of the device which is connected to the selected slot. The output is in plain text.

FW_VER

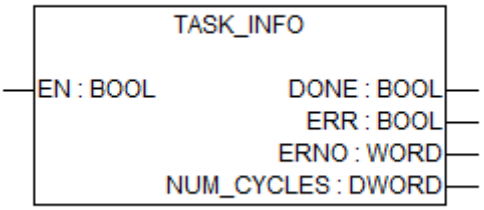
Data type: STRING(16)

The output FW_VER (firmware version) provides the firmware version of the device which is connected to the selected slot. The output is in plain text.

Function call in ST

```
SLOT_INFO (EN := SlotInfo_EN, SLOT := SlotInfo_SLOT);


SlotInfo_DONE := SLOT_INFO.DONE;
SlotInfo_ERR := SLOT_INFO.ERR;
SlotInfo_ERNO := SLOT_INFO.ERNO;
SlotInfo_MODEL := SLOT_INFO.MODEL;
SlotInfo_MAN_DATE := SLOT_INFO.MAN_DATE;
SlotInfo_DEV_NO := SLOT_INFO.DEV_NO;
SlotInfo_SER_NO := SLOT_INFO.SER_NO;
SlotInfo_FW_NAME := SLOT_INFO.FW_NAME;
SlotInfo_FW_VER := SLOT_INFO.FW_VER;
```

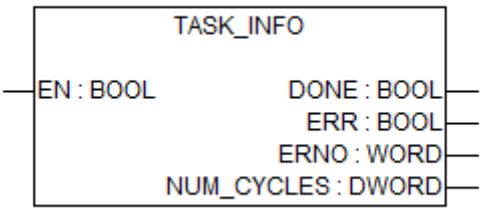
TASK_INFO read number of completed task cycles



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Program

The Program TASK_INFO outputs the number of completed cycles for the task it is called in.

Input description



EN

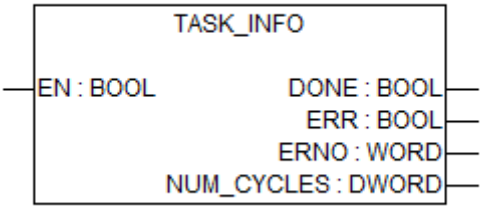
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_CYCLES DWORD (number of cycles)

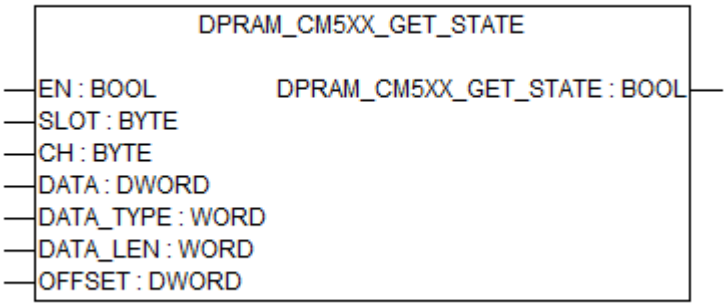
Output NUM_CYCLES displays a counter of the number of times the current task has completed a cycle. The number is reset to 0 on PLC STOP, download of new user program or reset of the PLC.

Function call in ST

```
TaskInfo (EN         := TaskInfo_EN);


TaskInfo_DONE        := TaskInfo.DONE;
TaskInfo_ERR         := TaskInfo.ERR;
TaskInfo_ERNO        := TaskInfo.ERNO;
TaskInfo_NUM_CYCLES  := TaskInfo.NUM_CYCLES;
```

1.5.4.19.4 Functions

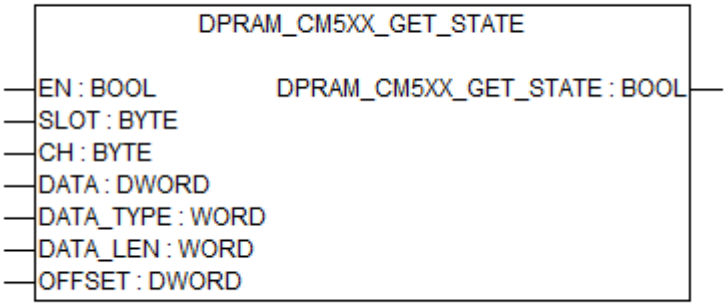
DPRAM_CM5XX_GET_STATE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function

Function DPRAM_CM5XX_GET_STATE is used for getting value(s) in extended status from a Communication Module.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type: BYTE

At input CH (channel), the addressed channel is specified.

Valid values: 1 and 2.

DATA (data)

Data type: DWORD

At input DATA, the address of the variable is specified, of which the user data shall be received.

DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION!

Memory area overlapping

Avoid memory area overlappings by specifying the size of the variable to the max. data expected.

DATA_TYPE (data type)

Data type: WORD

At input DATA_TYPE, the type of data has to be specified. The following allocation is used:

- 0 = Bit
- 1 = Byte
- 2 = Word
- 4 = DWord

DATA_LEN (data length)

Data type: DWORD

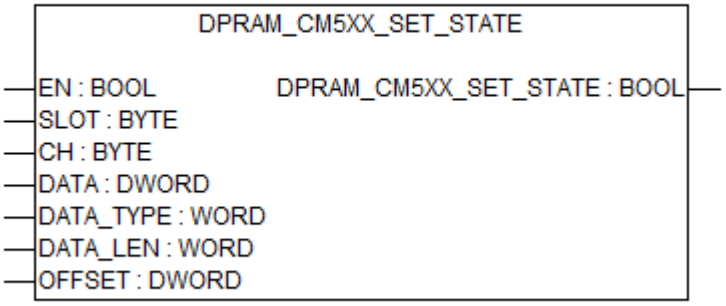
At input DATA_LEN, the length of the data which should be read has to be specified.

OFFSET (offset)

Data type: DWORD

At input OFFSET, the offset within the block in bytes has to be specified.

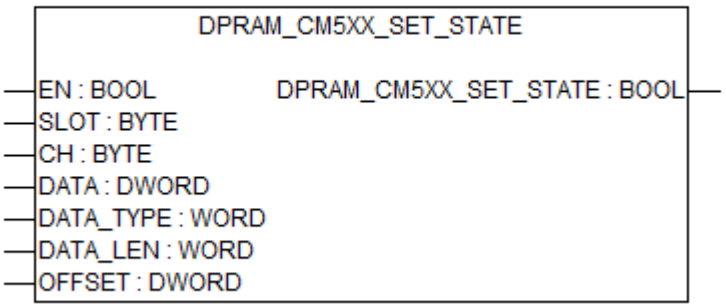
DPRAM_CM5XX_SET_STATE



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.3.0
Type	Function

Function DPRAM_CM5XX_SET_STATE is used for setting value(s) in extended status from a Communication Module.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

CH (channel)

Data type: BYTE

At input CH (channel), the addressed channel is specified.

Valid values: 1 and 2.

DATA (data)

Data type: DWORD

At input DATA, the address of the variable is specified, of which the user data shall be received.

DATA must be the address of a variable of the type ARRAY or STRUCT.

CAUTION!
Memory area overlapping
Avoid memory area overlappings by specifying the size of the variable to the max. data expected.

DATA_TYPE (data type)

Data type: WORD

At input DATA_TYPE, the type of data has to be specified. The following allocation is used:

- 0 = Bit
- 1 = Byte
- 2 = Word
- 4 = DWord

DATA_LEN (data length)

Data type: DWORD

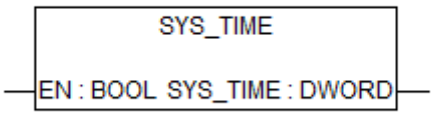
At input DATA_LEN, the length of the data which should be read has to be specified.

OFFSET (offset)

Data type: DWORD

At input OFFSET, the offset within the block in bytes has to be specified.

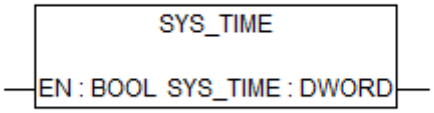
SYS_TIME



Parameter	Value
Included in library	SysInt_AC500_V10.lib
Available as of firmware	V1.0
Type	Function

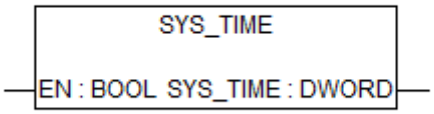
The Function SYS_TIME outputs the system tick in milliseconds as a double word. The system tick is provided with a resolution of a millisecond and also serves as a time basis for the PLC application program and all time-dependent function blocks. After a PLC reset the system tick always starts with 0. An overflow is reached after 49 days. After this, the counter restarts at 0.

Input description



EN BOOL (enable) The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

Output description



(Output) DWORD The function block output delivers the system tick in ms.

Function call in ST

```
SysTime := SYS_TIME(SysTime_EN);
```

1.5.4.20 Extended internal system library

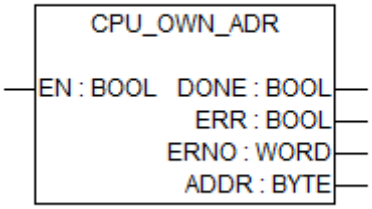
Library file name: **SysIntExt_AC500_Vx.lib**

The extended internal system library contains all function blocks that are generally applicable to the system, i.e. function blocks for general system diagnosis functions or for system information.

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

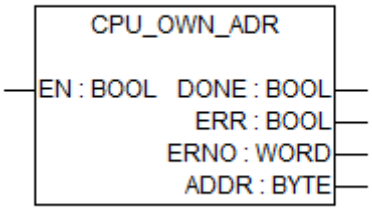
1.5.4.20.1 Function blocks

CPU_OWN_ADR



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V1.3
Type	Function block with historical values
Group	System information

Input description



EN

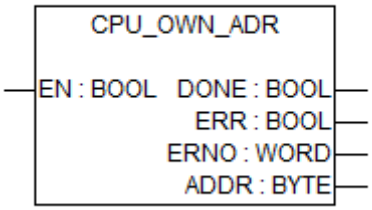
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADDR

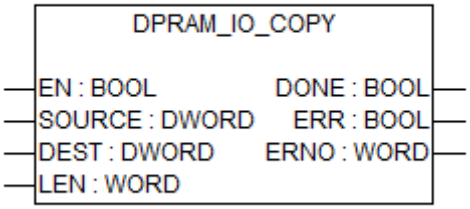
Data type	Default value	Range	Unit
BYTE	-	-	-

Output ADDR provides the current address value set with the rotary switch on the CM574-RS.

Function call in ST

```
OWN_ADR (EN := OWN_ADR_EN);
OWN_ADR_DONE := OWN_ADR.DONE;
OWN_ADR_ERR := OWN_ADR.ERR;
OWN_ADR _ERNO := OWN_ADR.ERNO;
OWN_ADR_MAP := OWN_ADR.ADDR;
```

DPRAM_IO_COPY



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	DPRAM communication

This function block is used to copy IO data areas.

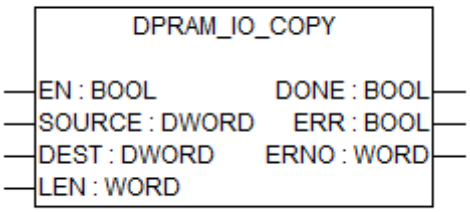
Using this function block it is possible to transfer for instance the inputs of the CS31 to the CM574-RS outputs without occupying any PLC code.

To use this function block, symbolic names have to be assigned to each first byte, e.g. "Input1" for %IB1000 = SOURCE and "Output1" for %QB0 = DEST.

If enabled, the function block first reads all IO data assigned from the I/O image into the image area. Then it copies the image area to the assigned IO data. Input LEN specifies the number of bytes to be copied. Output DONE = TRUE indicates that the operation is finished.

See also *Chapter 1.6.4.1.1 “Inputs, outputs and flags for AC500 V2 products” on page 5395* in the Multitasking System.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SOURCE

Data type	Default value	Range	Unit
DWORD	-	-	-

Input SOURCE specifies the address of the first data byte of the IO area to be read.

DEST

Data type	Default value	Range	Unit
DWORD	-	-	-

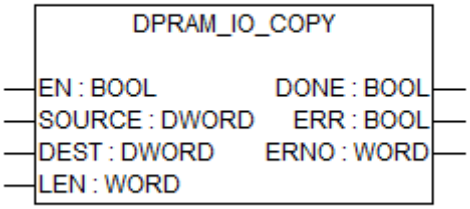
Input DEST specifies the address of the first data byte of the destination IO area the data should be copied to.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Input LEN specifies the length (in bytes) of the data to be copied.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

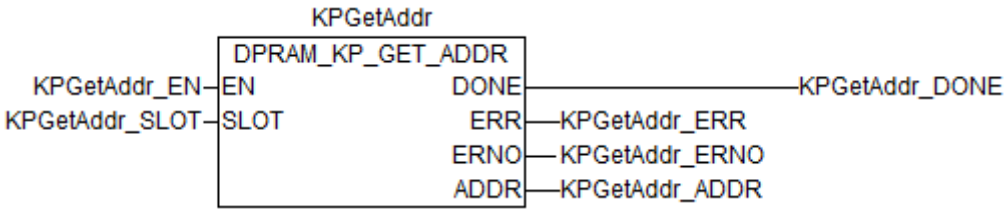
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
DPRAM_IO_COPY(EN:= IOcopy_EN,
              SOURCE:= IOcopy_SOURCE,
              DEST:= IOcopy_DEST,
              LEN:= IOcopy_LEN);
DPRAM_IO_COPY_DONE:= DPRAM_IO_COPY.DONE;
DPRAM_IO_COPY_ERR:= DPRAM_IO_COPY.ERR;
DPRAM_IO_COPY_ERNO:= DPRAM_IO_COPY.ERNO;
```

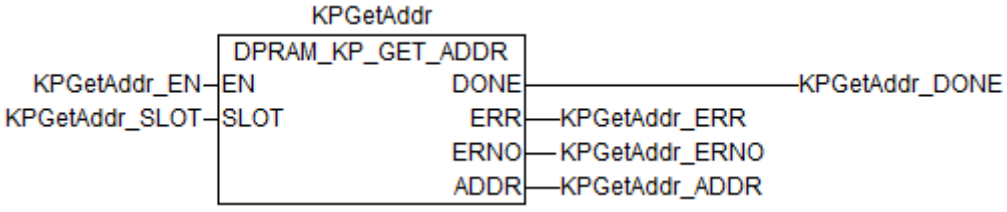
DPRAM_KP_GET_ADDR



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	DPRAM communication

This function block is used to get the start address of DPRAM for KP-based Communication Modules.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

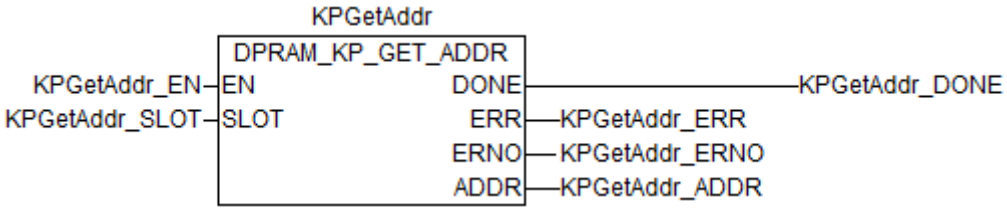
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADDR

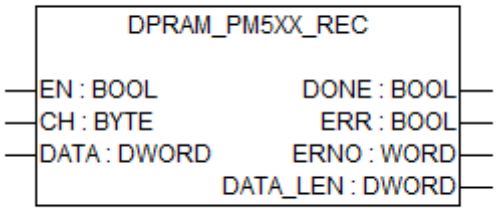
Data type	Default value	Range	Unit
DWORD	-	-	-

Output ADDR returns the start address of the DPRAM. If KP has not been completely detected yet, output ERNO will indicate busy state and set DONE to TRUE.

Function call in ST

```
KPGetAddr(EN:= KPGetAddr_EN,
              SLOT:= KPGetAddr_SLOT);

KPGetAddr_DONE:= KPGetAddr.DONE;
KPGetAddr_ERR := KPGetAddr.ERR;
KPGetAddr_ERNO:= KPGetAddr.ERNO;
KPGetAddr_ADDR:= KPGetAddr.ADDR;
```

DPRAM_PM5XX_REC



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	DPRAM communication

This function block is used to receive data from a AC500 CPU.

The data received are stored in the configured memory area, i.e. at the memory address specified at input DATA via the ADR operator.

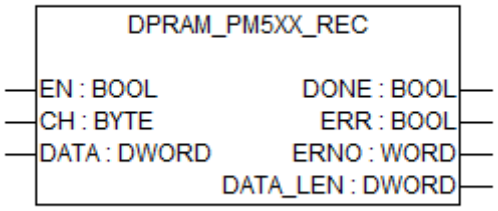
The function block is enabled by a TRUE signal at input EN. It remains active until input EN is set to FALSE. The channel of the AC500 CPU is specified at input CH.

Output DATA_LEN displays the length of the received data in bytes. Successful data reception is indicated by DONE = TRUE and ERR = FALSE. The outputs ERR and ERNO indicate whether an error occurred during Function Block processing.

Data reception using the DPRAM_PM5XX_REC function block is not edge-triggered. Therefore, input EN must be TRUE during data reception.

DPRAM message-based data exchange allows a maximum of 260 Bytes per message to be sent or received.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CH

Data type	Default value	Range	Unit
BYTE	-	-	-

Input CH (channel) is used to specify the channel of the AC500 CPUs to be addressed.

Valid values: 1 and 2.

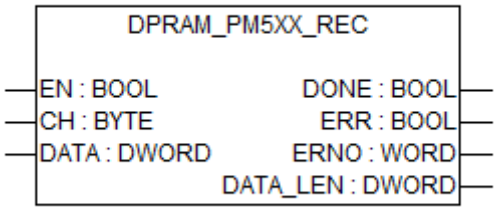
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable the user data are to be copied to. DATA must contain an address of a variable of the type ARRAY or STRUCT.

Specify the size of the variable according to the maximum amount of data expected in order to avoid memory area overlapping.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DATA_LEN

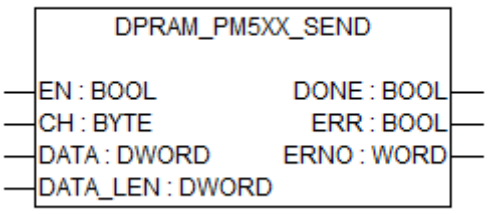
Data type	Default value	Range	Unit
WORD	-	-	Byte

Output DATA_LEN displays the length of the received data. The output value at DATA_LEN is only valid if DONE = TRUE.

Function call in ST

```
PM5xxRec (EN := PM5xxRec_EN,
CH := PM5xxRec_CH,
DATA := ADR(PM5xxRec_DATA),
DONE => PM5xxRec_DONE,
ERR => PM5xxRec_ERR,
ERNO => PM5xxRec_ERNO,
DATA_LEN => PM5xxRec_DATA_LEN);
PM5xxRec_DONE := PM5xxRec.DONE;
PM5xxRec_ERR := PM5xxRec.ERR;
PM5xxRec_ERNO := PM5xxRec.ERNO;
PM5xxRec_DATA_LEN := PM5xxRec.DATA_LEN;
```

DPRAM_PM5XX_SEND



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	DPRAM communication

This function block is used to send data to the AC500 CPU.

The data to be sent are provided in the configured memory area, i.e. at the memory address specified at input DATA via the ADR operator.

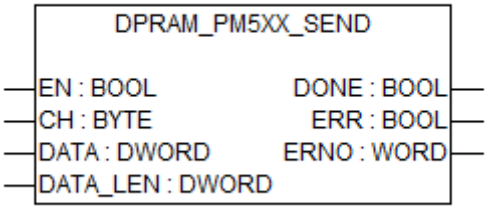
The function block is enabled by a TRUE signal at input EN. It remains active until input EN is set to FALSE. The channel of the AC500 CPU is specified at input CH. The data length (in bytes) to be sent is specified at input DATA_LEN.

Successful data transfer is indicated by DONE = TRUE and ERR = FALSE. The outputs ERR and ERNO indicate whether an error occurred during Function Block processing.

Sending data using the DPRAM_PM5XX_SEND function block is edge-triggered, i.e. each sending process is initiated by a FALSE/TRUE edge at input EN.

DPRAM message-based data exchange allows a maximum of 260 Bytes per message to be sent or received.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

CH

Data type	Default value	Range	Unit
BYTE	-	-	-

Input CH (channel) is used to specify the channel of the AC500 CPUs to be addressed.

Valid values: 1 and 2.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA is used to specify the address of the variable the user data are to be copied to. DATA must contain an address of a variable of the type ARRAY or STRUCT.

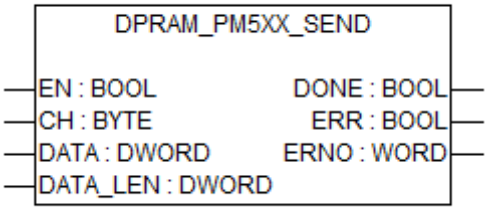
Specify the size of the variable according to the maximum amount of data expected in order to avoid memory area overlapping.

DATA_LEN

Data type	Default value	Range	Unit
WORD	-	-	Byte

Output DATA_LEN displays the length of the received data. The output value at DATA_LEN is only valid if DONE = TRUE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

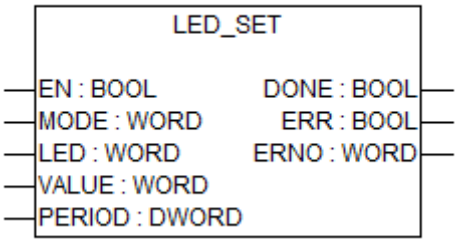
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
PM5xxSend (EN := PM5xxSend_EN,
SLOT := PM5xxSend_SLOT,
CH := PM5xxSend_CH,
DATA := ADR(PM5xxSend_DATA),
DATA_LEN := PM5xxSend_DATA_LEN,
DONE => PM5xxSend_DONE,
ERR => PM5xxSend_ERR,
ERNO => PM5xxSend_ERNO);
PM5xxSend_DONE := PM5xxSend.DONE;
PM5xxSend_ERR := PM5xxSend.ERR;
PM5xxSend_ERNO := PM5xxSend.ERNO;
```

LED_SET



Parameter	Value
Included in library	SysIntExt_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	LED

This function block is used to control the LED states of the AC500 PLCs. Using this function block it is possible to access each LED on the AC500 PLC and to change its state and blinking period.

Each LED has to be controlled individually.

The LED assignment depends on PLC type.

CM574:

1 : RUN LED

2 : ERROR LED

3 : COM1 Status LED

4 : COM2 Status LED

5 : READY LED

6 : STAT LED

PM57x, PM58x, PM59x:

1 : RUN LED

2 : ERROR LED

Eco:

1 : RUN LED

2 : ERROR LED

3 : COM1 Status LED

PM595:

1 : RUN LED (green only)

2 : ERROR LED (red only)

3 : CAN LED(green/red/orange)

8 : Batt State LED (green/red/orange)

9 : Function 1 LED (green/red/orange)

10 : Function 2 LED (green/red/orange)

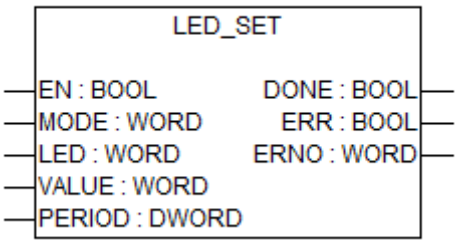
11 : Function 3 LED (green/red/orange)

12 : Function 4 LED (green/red/orange)

13 : Function 5 LED (green/red/orange)

If the LED should be controlled by the runtime system again, just switch the MODE back to 0. (MODE 0 = runtime system , MODE 1 = user program).

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Mode

Data type	Default value	Range	Unit
WORD	-	-	-

Access control for the LED:

MODE = 0 = runtime system

MODE = 1 = user program

LED

Data type	Default value	Range	Unit
WORD	-	-	-

Input LED is used to specify the LED:

CM574:

1 : RUN LED

2 : ERROR LED

3 : COM1 Status LED

4 : COM2 Status LED

5 : READY LED

6 : STAT LED

PM57x, PM58x, PM59x:

1 : RUN LED

2 : ERROR LED

Eco:

1 : RUN LED

2 : ERROR LED

3 : COM1 Status LED

PM595:

1 : RUN LED (green only)

2 : ERROR LED (red only)

3 : CAN LED (green/red/orange)

8 : Batt State LED (green/red/orange)

9 : Function 1 LED (green/red/orange)

10 : Function 2 LED (green/red/orange)

11 : Function 3 LED (green/red/orange)

12 : Function 4 LED (green/red/orange)

13 : Function 5 LED (green/red/orange)

VALUE

Data type	Default value	Range	Unit
WORD	-	-	-

Input VALUE is used to specify the static value of the LED. Only applicable if the LED is not in mode = 0.

Following values are possible:

0 = LED off

1 = LED on (Default colour used if applicable)

2 = LED on, green (only at PM595)

3 = LED on, red (only at PM595)

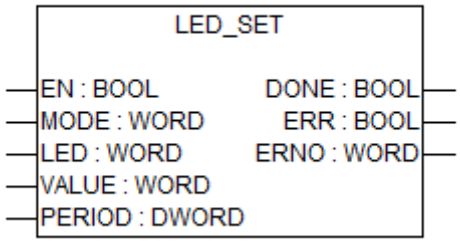
4 = LED on, orange (only at PM595)

PERIOD

Data type	Default value	Range	Unit
DWORD	-	-	ms

Input PERIOD is used to set the blinking period of the LED in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
LedSet (EN := LedSet_EN,
MODE:= LedSet_MODE,
LED := LedSet_LED,
VALUE := LedSet_VALUE,
PERIOD := LedSet_PERIOD,
DONE => LedSet_DONE,
ERR => LedSet_ERR,
ERNO => LedSet_ERNO);
LedSet_DONE := LedSet.DONE;
LedSet_ERR := LedSet.ERR;
LedSet_ERNO := LedSet.ERNO;
```

1.5.4.21 JSON library

1.5.4.21.1 System technology

JSON is an acronym for *“JavaScript Object Notation”*. It is a compact data format that is often used to transfer data between clients and a server (e. g. via MQTT). JSON has arisen from JavaScript, whereby many implementations for several programming languages exist.

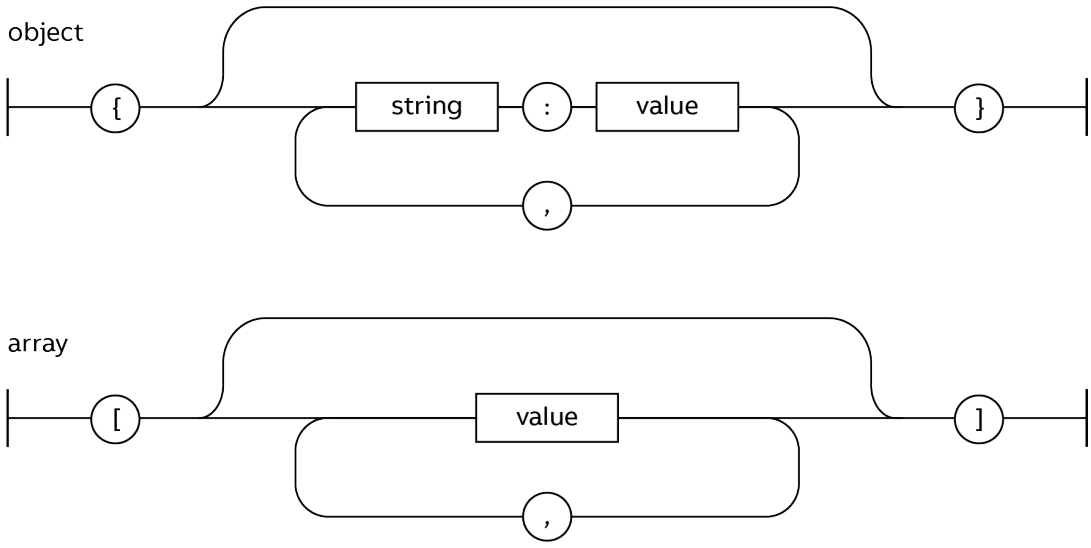
JSON embeds several data types as *“values”* into either *“JSON OBJECTS”* or *“JSON ARRAYS”*.

In objects each value is identified by a key string, while arrays make values accessible via indices.

JSON also allows the nesting of objects and arrays as values providing a recursive parent-child hierarchy.

The possible types for a value are namely: *“string”*, *“number”*, *“object”* and *“array”*. Values can also consist of the constants `true`, `false` and `null`.

The following figure displays the structure of JSON objects and arrays. Detailed information on the JSON format is given at *https://www.json.org*.



The use of the JSON library allows creating and parsing of JSON strings without manual string operations and detailed knowledge about JSON syntax. This reduces the risk of data transfer errors, as syntax of JSON strings will always be correct.

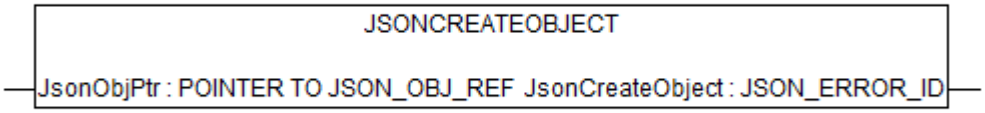
To realize those benefits, there are functions implemented in the JSON library.

A detailed description for each function is given in the section *Chapter 1.5.4.21.2 “Function block description” on page 1647*.

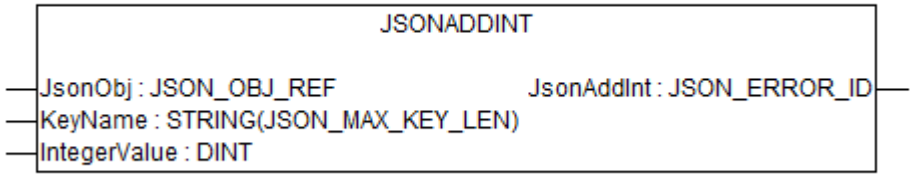
Creating and sending

One JSON create and send use case could look like this:

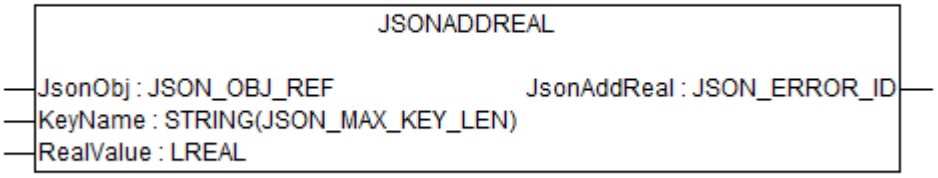
1. Create an empty JSON object with the function block *"JSONCREATEOBJECT"*.



2. Add a sender ID with the function block *"JSONADDINT"*.

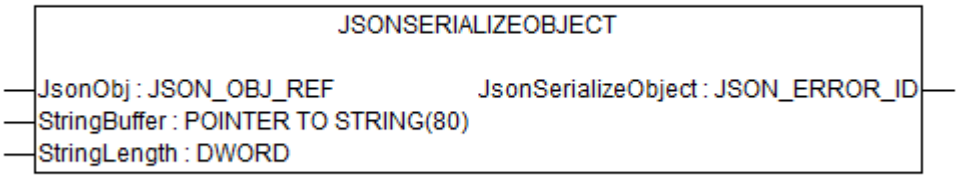


3. Add a telemetry value (i.e. "temperature":21.0) with the function block *"JSONADDREAL"*.

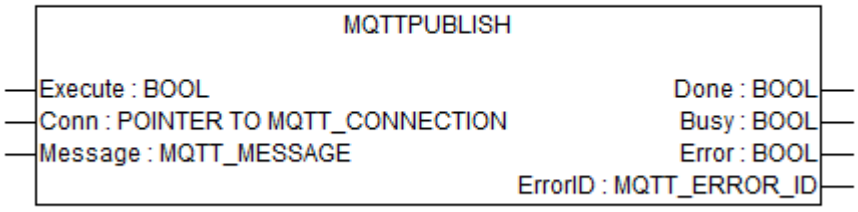


4. Serialize the created object into a string for sending with the function block *"JSONSERIALIZEOBJECT"*

.



5. Send the JSON string for instance to a MQTT broker with the function block *"MQTTPUBLISH"*.

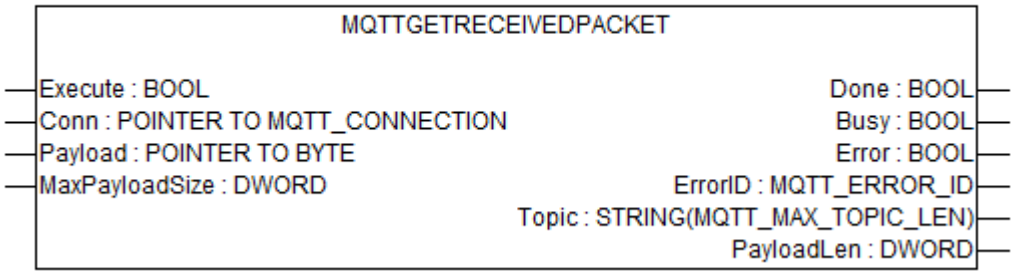


6. If no longer needed, free the created JSON object.

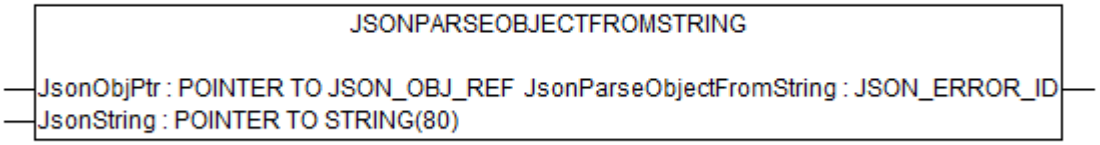
Receiving and parsing

One JSON receive and parse use case could look like this:

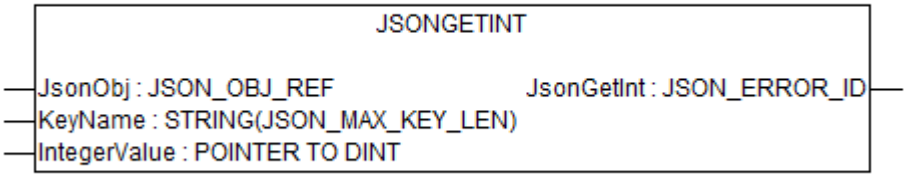
1. Receive a JSON string for instance from a MQTT broker with the function block *"MQTTGETRECEIVEDPACKET"*.



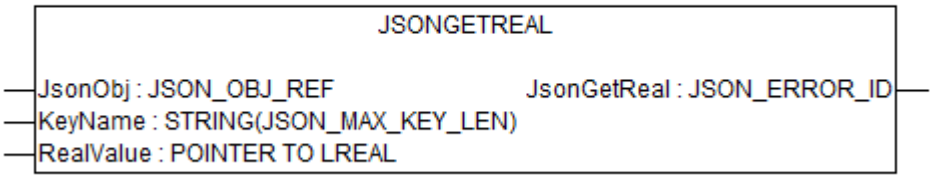
2. Parse the received string into a JSON object with the function block *"JSONPARSEOBJECTFROMSTRING"*.



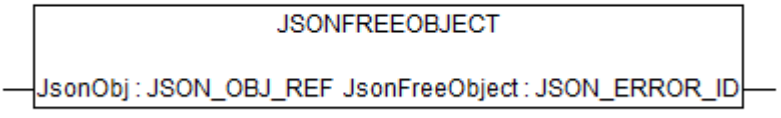
3. Get the ID of the sender with the function block *"JSONGETINT"*.



4. Get settings for the PLC (i.e. "frequency":2.7) with the function block JSONGETREAL.



5. If no longer needed, free the created JSON object.



Configuration in Automation Builder

For the JSON library no special configuration is needed.

Task configuration in CODESYS

All functions have to be called in tasks with cyclically processing.

Limitations

- Function calls are synchronous. Processing large objects might stall the execution of the task.
- The maximum length of each JSON key is limited to 100 characters (see *“JSON_MAX_KEY_LEN”* constant).
- The maximum recursion level for the nesting of JSON OBJECTS and JSON ARRAYS is limited to a depth of 64.

Functions

The functions in this library can only be executed in RUN mode of the processor module, not in simulation mode.

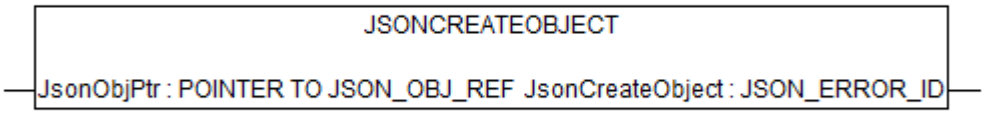
Standard

JSON library is based on *[RFC 8259]*.

1.5.4.21.2 Function block description

JSON object functions

JSONCREATEOBJECT

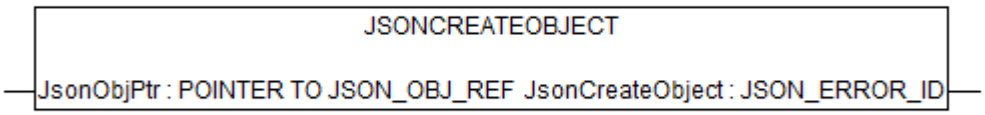


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function *"JSONCREATEOBJECT"* creates an empty JSON OBJECT. This function allocates memory for the new OBJECT. The new OBJECT reference is assigned to the given pointer.

Input description



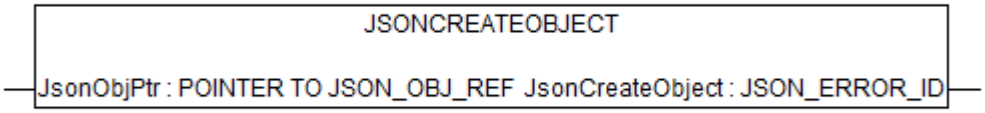
JsonObjPtr

Data type	Default value	Range	Unit
POINTER TO JSON_OBJ_REF	-	-	-

The pointer to a JSON OBJECT reference to assign the new OBJECT.

In case of any error, the data at the provided pointer stays unchanged.

Output description

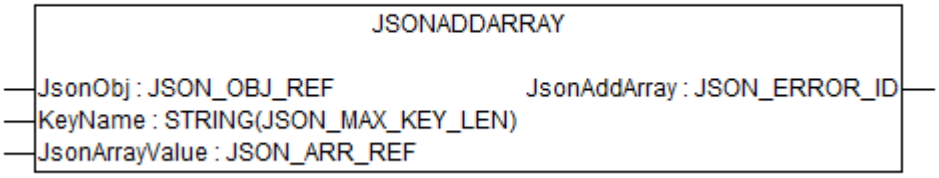


JsonCreate Object

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDARRAY

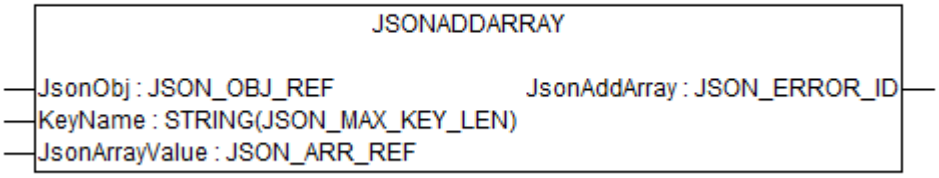


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDARRAY nests a JSON ARRAY reference to a JSON OBJECT. The given JSON ARRAY is not copied into the OBJECT, only meta information is added. The JSON ARRAY will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the JSON ARRAY. Memory of the old value gets freed in the replacement.

Input Description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT to which the JSON ARRAY is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

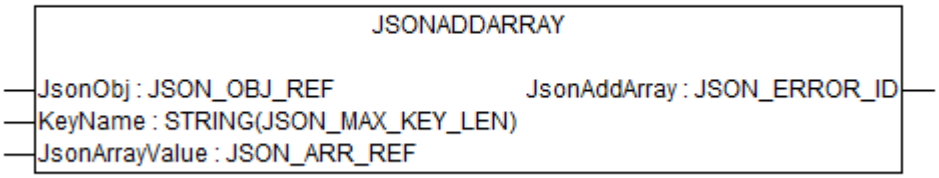
The key string by which the added JSON ARRAY is identified.

JsonArrayValue

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY which gets inserted.

Output Description

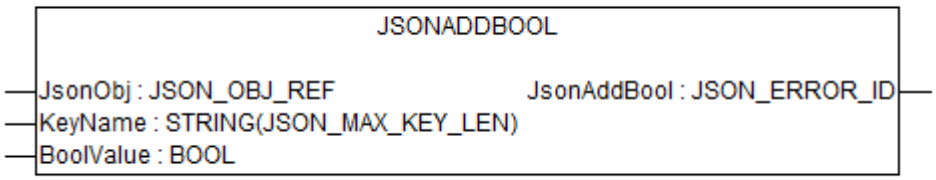


JsonAddArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDBOOL

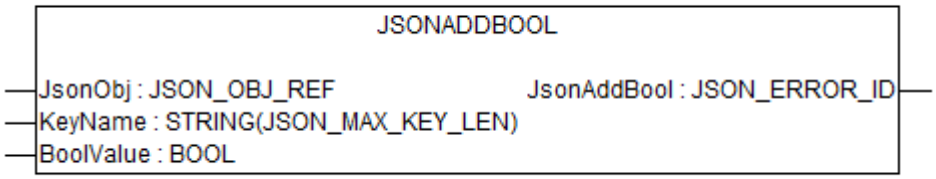


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDBOOL adds a BOOLEAN value to a JSON OBJECT. The given BOOLEAN is copied into the OBJECT. The BOOLEAN will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the BOOLEAN. Memory of the old value gets freed in the replacement.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT to which the BOOLEAN value is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

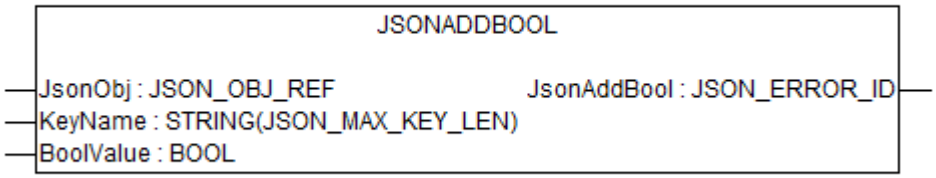
The key string by which the added BOOLEAN is identified.

BoolValue

Data type	Default value	Range	Unit
BOOL	-	-	-

The BOOLEAN value which gets added.

Output description

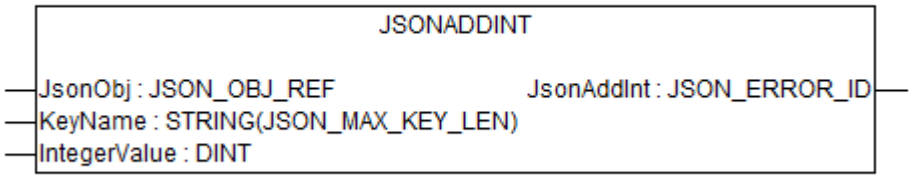


JsonAddBool

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDINT

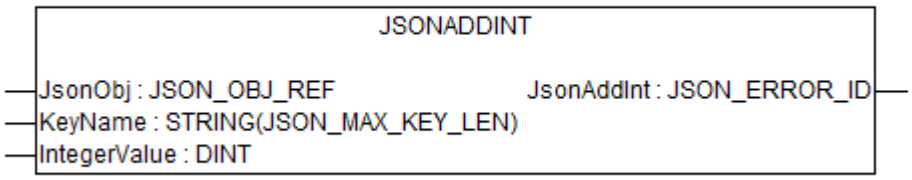


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDINT adds an INTEGER value to a JSON OBJECT. The given INTEGER is copied into the OBJECT. The INTEGER will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the INTEGER. Memory of the old value gets freed in the replacement.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT to which the INTEGER value is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

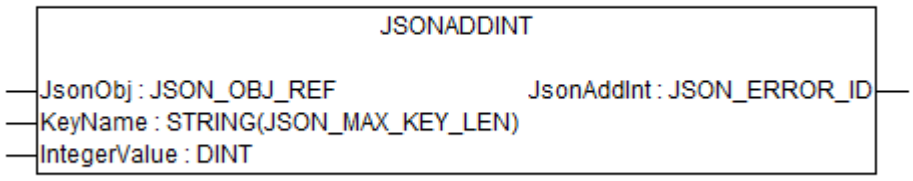
The key string by which the added INTEGER is identified.

IntegerValue

Data type	Default value	Range	Unit
DINT	-	-	-

The INTEGER value which gets added. The input is DINT. In case an INT variable in CODESYS is added the function INT_TO_DINT has to be used.

Output description

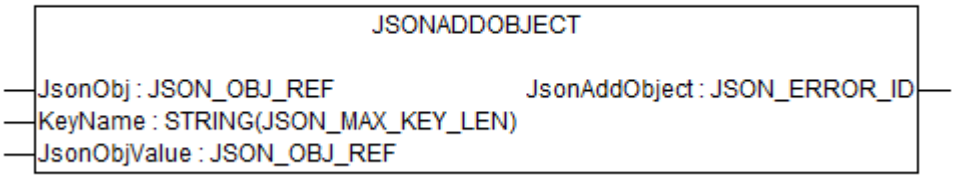


JsonAddInt

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDOBJECT

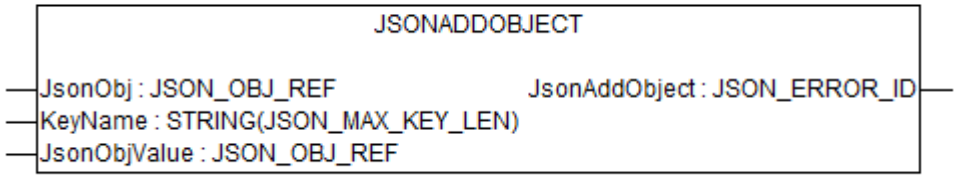


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDOBJECT nests a JSON OBJECT reference to another JSON OBJECT. The child JSON OBJECT is not copied into the parent JSON OBJECT, only meta information is added. The child JSON OBJECT will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the new JSON OBJECT. Memory of the old value gets freed in the replacement.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the parent JSON OBJECT to which the child JSON OBJECT is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

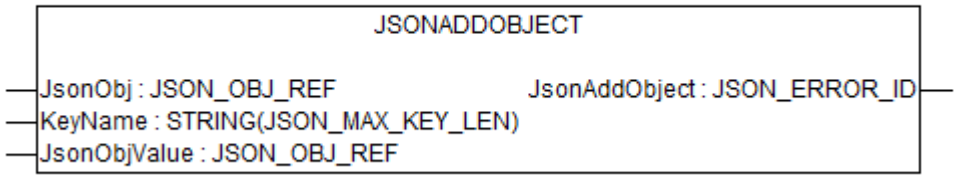
The key string by which the added child JSON OBJECT is identified.

JsonObjValue

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The child JSON OBJECT which gets added.

Output description

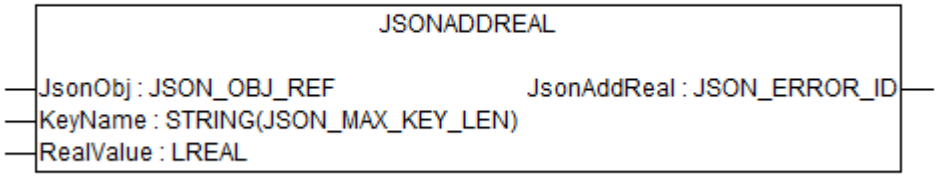


JsonAddObject

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDREAL

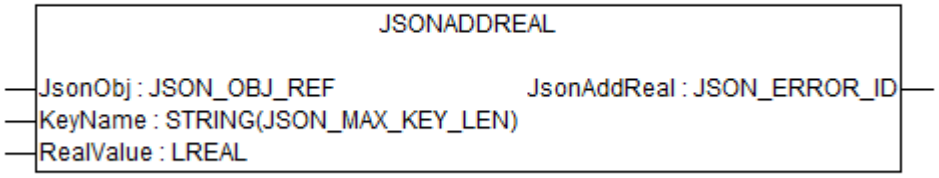


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDREAL adds a REAL value to a JSON OBJECT. The given REAL is copied into the OBJECT. The REAL will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the REAL. Memory of the old value gets freed in the replacement.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT to which the REAL value is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

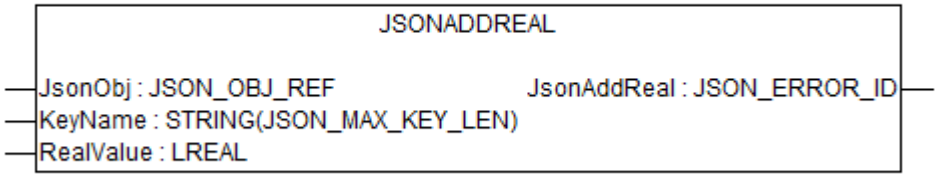
The key string by which the added REAL is identified.

RealValue

Data type	Default value	Range	Unit
LREAL	-	-	-

The REAL value which gets added. The input is LREAL. In case a REAL variable in CODESYS is added the function REAL_TO_LREAL has to be used.

Output description

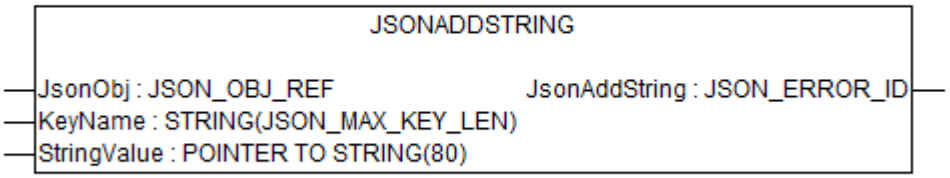


JsonAddReal

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONADDSTRING

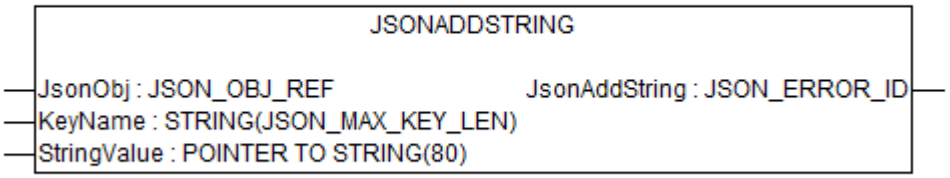


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONADDSTRING adds a STRING value to a JSON OBJECT. The given STRING is copied into an internal buffer of the OBJECT. The STRING will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the STRING. Memory of the old value gets freed in the replacement. Different from the function figure, STRING values are allowed to have any viable length, including STRING that are longer than 80 characters.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT to which the STRING value is added.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string by which the added STRING is identified.

StringValue

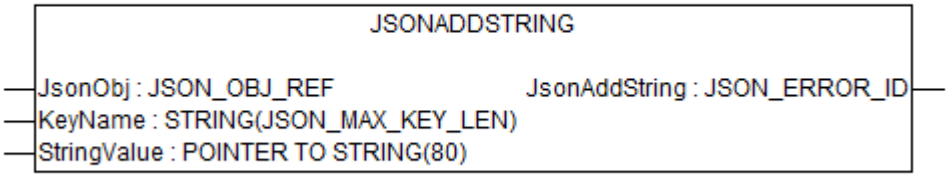
Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING value which gets added.

Contrary to the figure the STRING value is not limited to 80 characters.

Strings of any viable length can be provided.

Output description

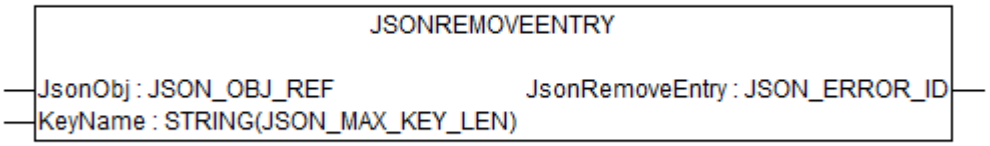


JsonAddString

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONREMOVEENTRY

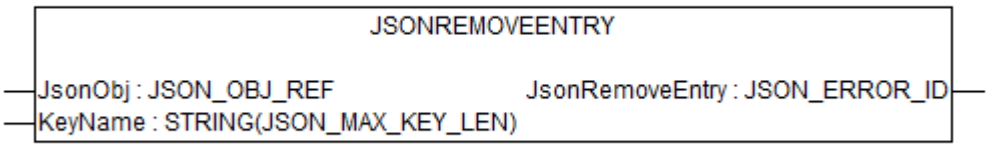


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONREMOVEENTRY removes a JSON value from a JSON OBJECT, that is identified by the given key name. All memory used by that value is freed. If the entry contains a JSON OBJECT or JSON ARRAY all contained recursive entries get freed as well.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

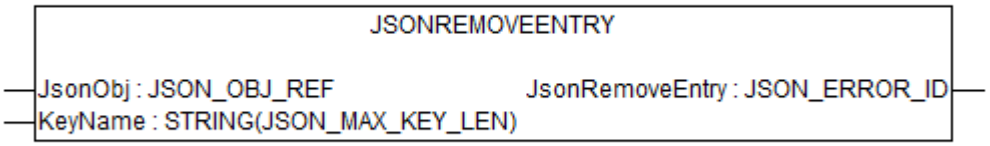
The reference to the JSON OBJECT from which the value is removed.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string by which the removed JSON value is identified.

Output description

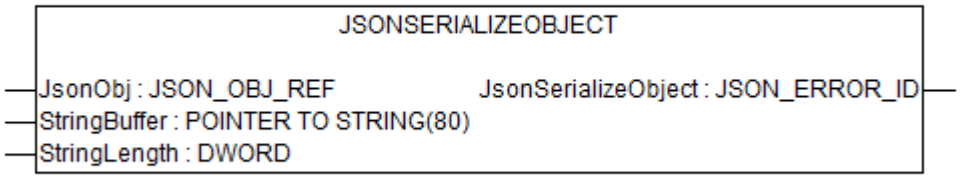


JsonRemoveEntry

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONSERIALIZEOBJECT

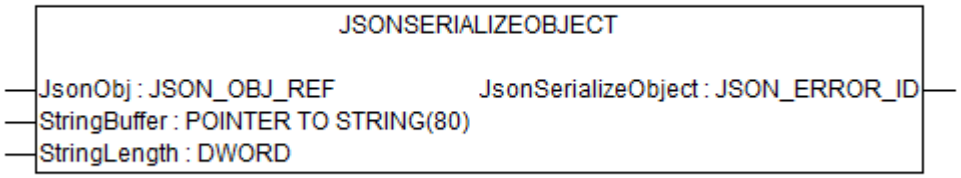


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONSERIALIZEOBJECT serializes a JSON OBJECT into a given STRING buffer. Different from the figure, the STRING buffer is allowed to have any viable length, including STRINGS that are longer than 80 characters.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT which is serialized.

StringBuffer

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING buffer which is the target of the serialization.

> *Contrary to the figure the STRING buffer is not limited to 80 characters.*
>
> *STRING buffers of any viable length can be provided.*
>
> *No memory will be acquired so a target buffer has to be provided.*

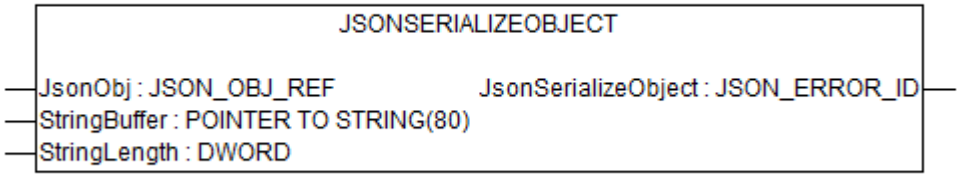
StringLength

Data type	Default value	Range	Unit
DWORD	-	-	-

The StringLength parameter specifies the limit of the provided STRING buffer.

If the size of the STRING buffer is not sufficient to contain the JSON OBJECT representation, the buffer stays unchanged.

Output description



JsonSerialize Object

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONFREEOBJECT

General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONFREEOBJECT frees all memory used by a JSON OBJECT. All recursively contained JSON OBJECTS and JSON ARRAYS are freed as well.

Input description

JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT which is freed. After the function returned without error, this reference becomes invalid.

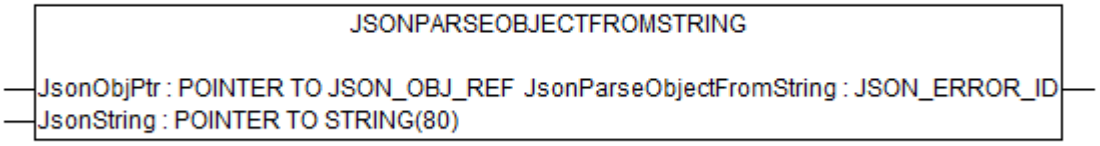
Output description

JsonFreeObject

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONPARSEOBJECTFROMSTRING

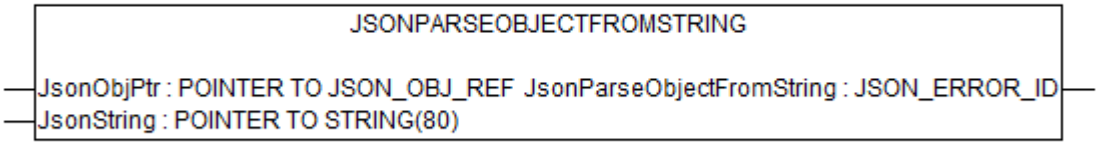


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONPARSEOBJECTFROMSTRING parses a STRING, which represents a JSON OBJECT and creates a corresponding OBJECT from it. The new OBJECT contains all values from the STRING representation. The function expects a reference to a JSON OBJECT to assign the new OBJECT.

Input description



JsonObjPtr

Data type	Default value	Range	Unit
POINTER TO JSON_OBJ_REF	-	-	-

The pointer to a JSON OBJECT reference to assign the parsed OBJECT.

In case of any error, the data at the provided pointer stays unchanged.

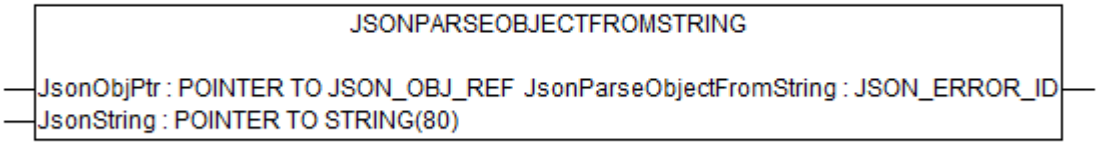
JsonString

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

JsonString is a pointer to a STRING representing a JSON OBJECT.

> *Contrary to the figure the STRING buffer is not limited to 80 characters.*
> *Strings of any viable length can be provided.*

Output description

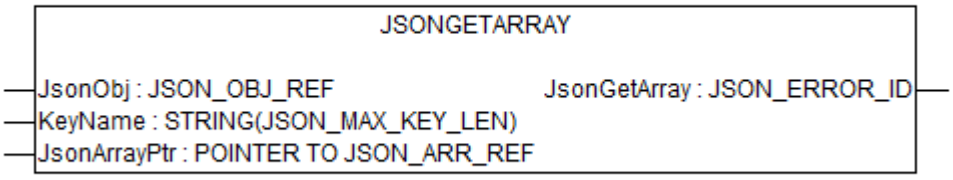


JsonParse ObjectFrom String

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETARRAY

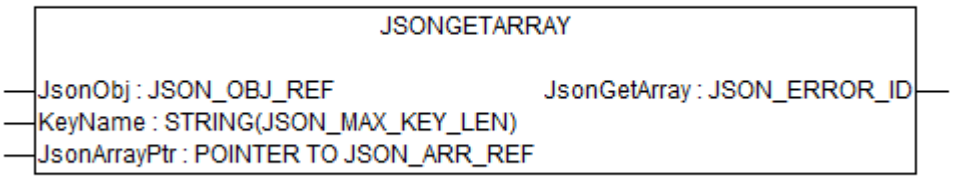


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETARRAY gets a JSON ARRAY identified by a key name from a JSON OBJECT. The ARRAY is not copied, only a reference is retrieved. This reference is assigned to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT from which the JSON ARRAY is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string which identifies the retrieved JSON ARRAY.

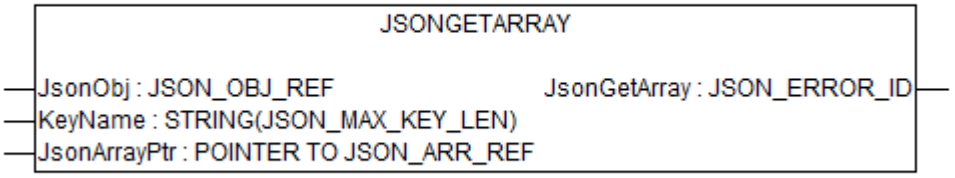
JsonArrayPtr

Data type	Default value	Range	Unit
POINTER TO JSON_ARR_REF	-	-	-

The pointer to a JSON ARRAY to assign the retrieved ARRAY reference.

In case of any error, the data at the provided pointer stays unchanged.

Output description

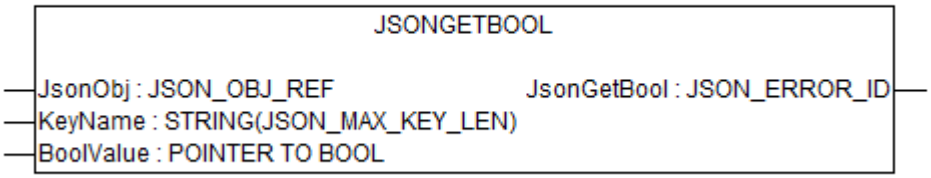


JsonGetArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETBOOL

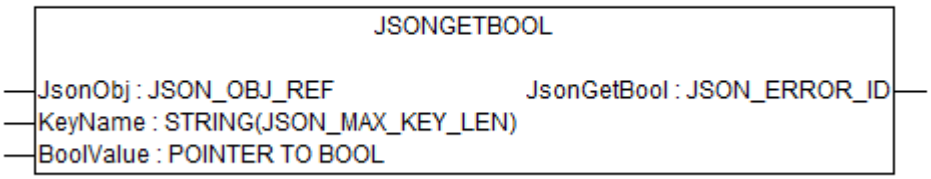


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETBOOL gets a BOOLEAN value identified by a key name from a JSON OBJECT. The retrieved value is assigned to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT from which the BOOLEAN value is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string by which identifies the retrieved BOOLEAN value.

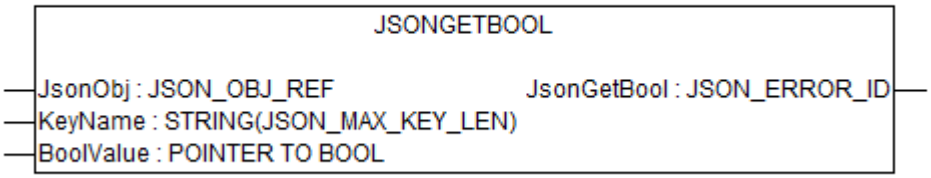
BoolValue

Data type	Default value	Range	Unit
POINTER TO BOOL	-	-	-

The pointer to a BOOLEAN variable to assign the retrieved value.

In case of any error, the data at the provided pointer stays unchanged.

Output description

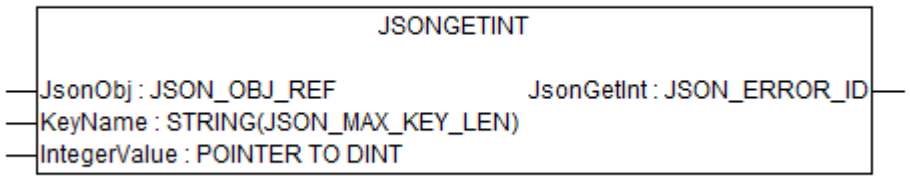


JsonGetBool

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETINT

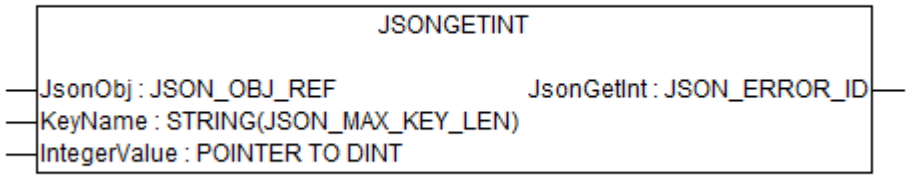


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETINT gets an INTEGER value identified by a key name from a JSON OBJECT. The retrieved value is assigned to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT from which the INTEGER value is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string which identifies the retrieved INTEGER value.

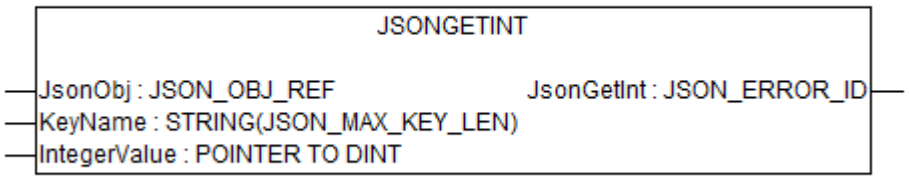
IntegerValue

Data type	Default value	Range	Unit
POINTER TO DINT	-	-	-

The pointer to an INTEGER variable to assign the retrieved value. In CODESYS a DINT variable has to be used.

In case of any error, the data at the provided pointer stays unchanged.

Output description

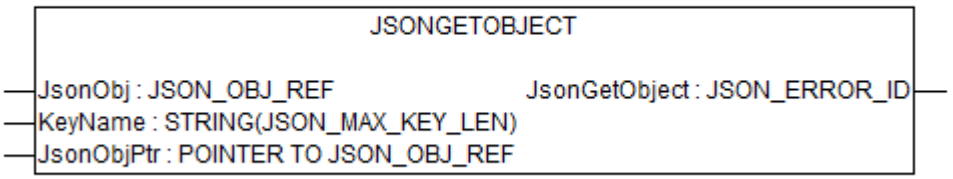


JsonGetInt

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETOBJECT

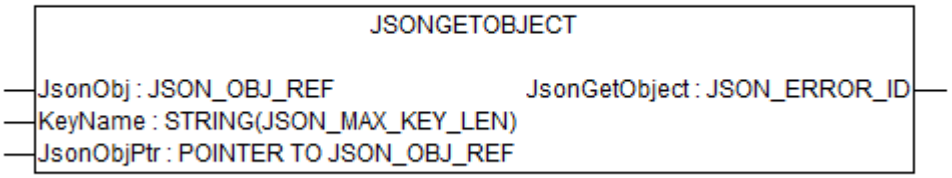


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETOBJECT gets a nested child JSON OBJECT identified by a key name from a parent JSON OBJECT. The OBJECT is not copied, only a reference is retrieved. This reference is assigned to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the parent JSON OBJECT from which the child JSON OBJECT is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string which identifies the retrieved child JSON OBJECT.

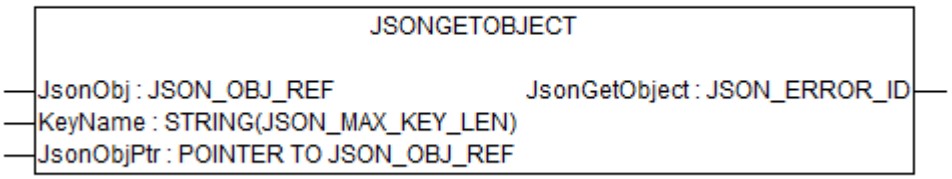
JsonObjPtr

Data type	Default value	Range	Unit
POINTER TO JSON_OBJ_REF	-	-	-

The pointer to a JSON OBJECT to assign the retrieved child OBJECT reference.

In case of any error, the data at the provided pointer stays unchanged.

Output description

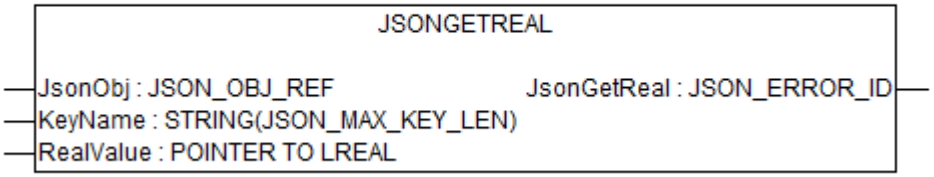


JsonGetObject

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETREAL

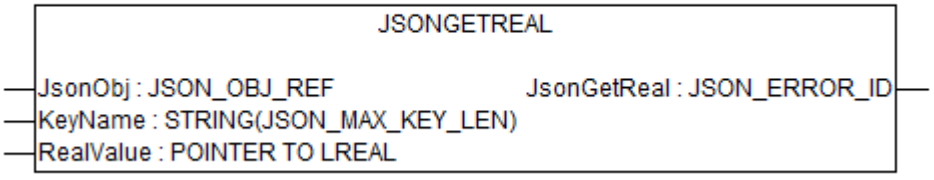


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETREAL gets a REAL value identified by a key name from a JSON OBJECT. The retrieved value is assigned to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT from which the REAL value is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string which identifies the retrieved REAL value.

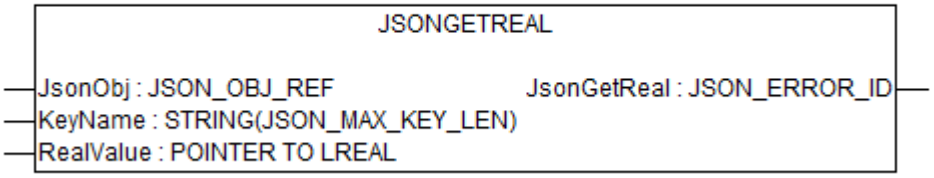
RealValue

Data type	Default value	Range	Unit
POINTER TO LREAL	-	-	-

The pointer to a REAL variable to assign the retrieved value. In CODESYS a LREAL variable has to be used.

In case of any error, the data at the provided pointer stays unchanged.

Output description

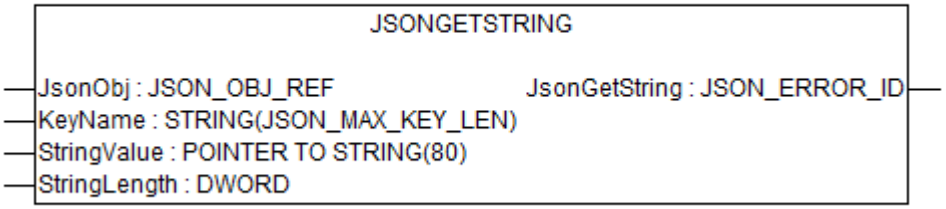


JsonGetReal

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONGETSTRING

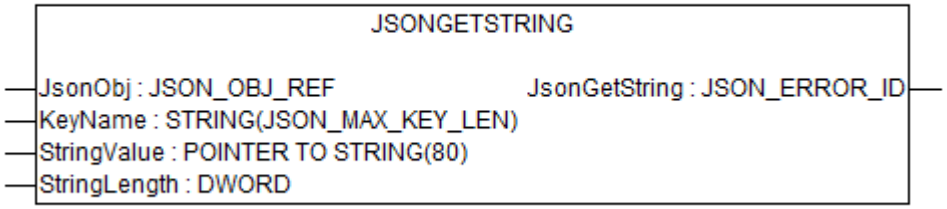


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONGETSTRING gets a STRING value identified by a key name from a JSON OBJECT. The STRING value is copied to the given pointer.

Input description



JsonObj

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The reference to the JSON OBJECT from which the STRING value is retrieved.

KeyName

Data type	Default value	Range	Unit
STRING(JSON_ MAX_KEY_LEN)	-	-	-

The key string which identifies the retrieved STRING value.

StringValue

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING buffer to copy the retrieved value.

Contrary to the figure the STRING value is not limited to 80 characters.
STRING buffers of any viable length can be provided.

In case of any error, the data at the provided pointer stays unchanged.

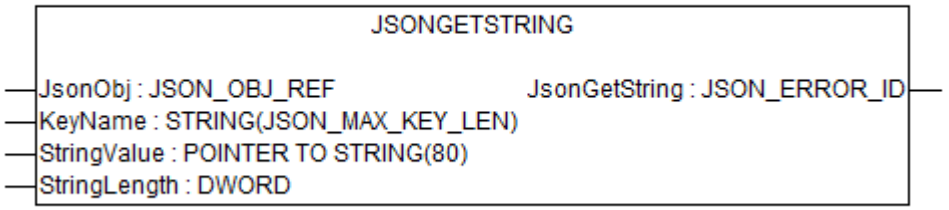
StringLength

Data type	Default value	Range	Unit
DWORD	-	-	-

The StringLength parameter specifies the limit of the provided STRING value.

If the size of the STRING value is not sufficient, it stays unchanged.

Output description



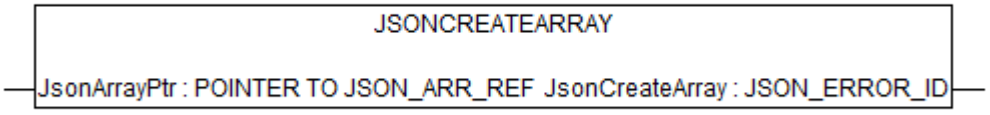
JsonGetString

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSON array functions

JSONCREATEARRAY

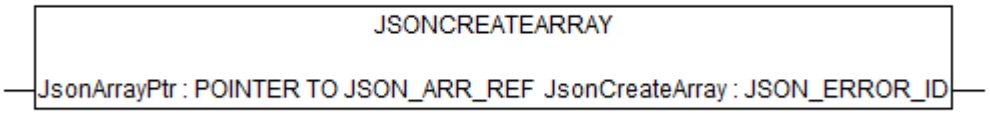


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function *"JSONCREATEARRAY"* creates an empty JSON ARRAY. This function allocates memory for the new ARRAY. The new ARRAY reference is assigned to the given pointer.

Input description



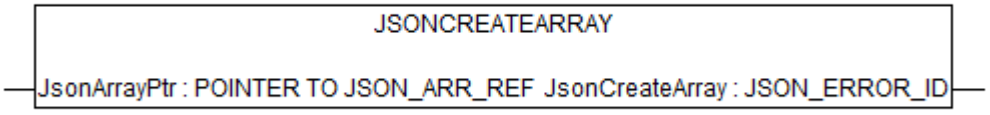
JsonArrayPtr

Data type	Default value	Range	Unit
POINTER TO JSON_ARR_REF	-	-	-

The pointer to a JSON ARRAY reference to assign the new ARRAY.

In case of any error, the data at the provided pointer stays unchanged.

Output description

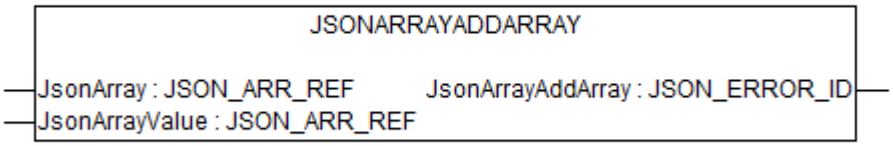


JsonCreateArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDARRAY

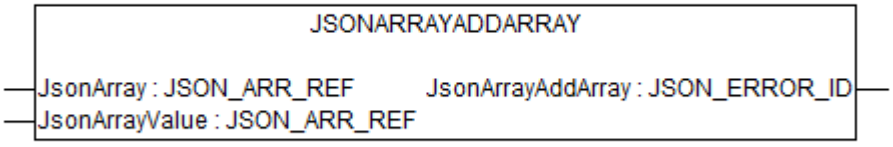


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDARRAY nests a JSON ARRAY reference to another JSON ARRAY. The child JSON ARRAY is not copied into the parent JSON ARRAY, only meta information is appended. The child JSON ARRAY will be identified by the last index after insertion.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

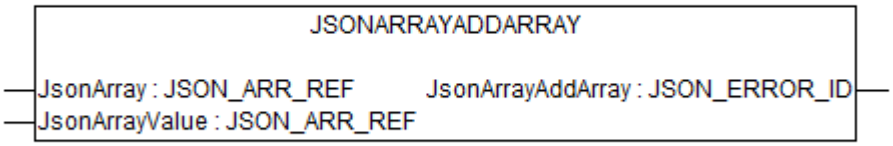
The reference to the parent JSON ARRAY to which the child JSON ARRAY is appended.

JsonArrayValue

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The child JSON ARRAY which gets appended.

Output description

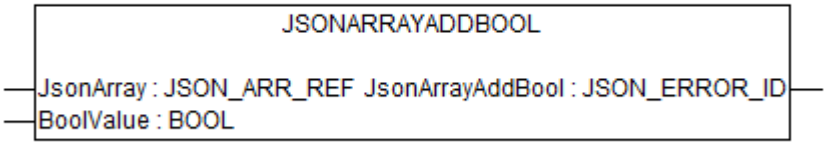


JsonArrayAddArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDBOOL

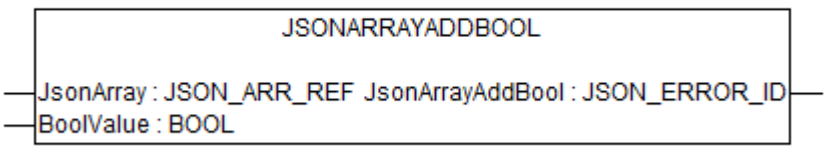


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDBOOL adds a BOOLEAN value to a JSON ARRAY. The given BOOLEAN is copied into the ARRAY. The BOOLEAN will be identified by the last index after insertion.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

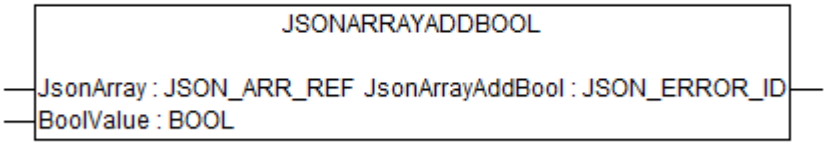
The reference to the JSON ARRAY to which the BOOLEAN is appended.

BoolValue

Data type	Default value	Range	Unit
BOOL	-	-	-

The BOOLEAN value which gets appended.

Output description

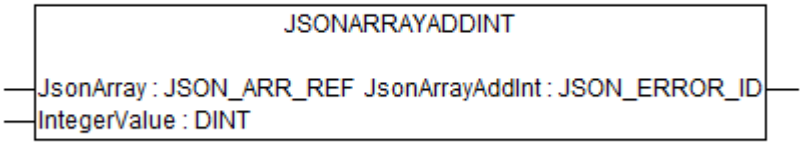


JsonArrayAdd Bool

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDINT

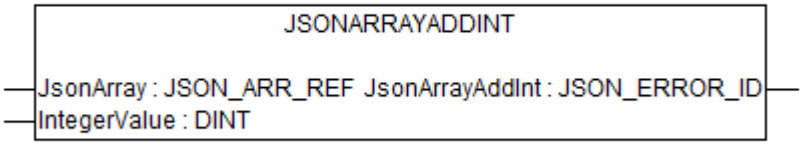


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDINT adds an INTEGER value to a JSON ARRAY. The given INTEGER is copied into the ARRAY. The INTEGER will be identified under the provided key name. If the key value pair already exists, the value gets replaced by the INTEGER. Memory of the old value gets freed in the replacement.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

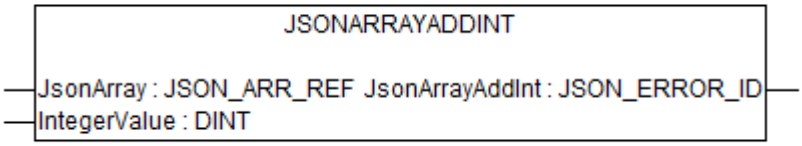
The reference to the JSON ARRAY to which the INTEGER is appended.

IntegerValue

Data type	Default value	Range	Unit
DINT	-	-	-

The INTEGER value which gets appended. The input is DINT. In case an INT variable in CODESYS is appended the function INT_TO_DINT has to be used.

Output description

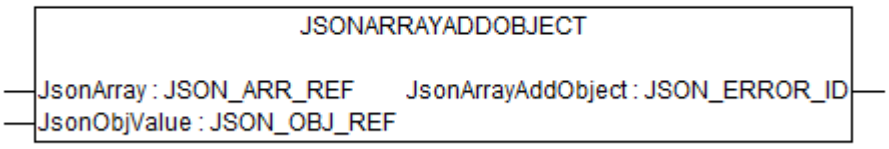


JsonArrayAddInt

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDOBJECT

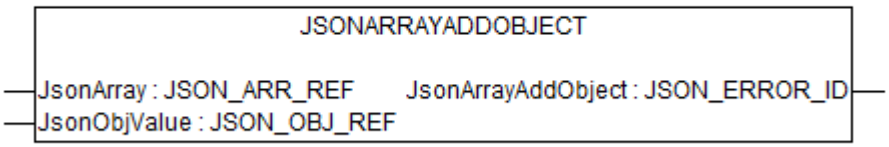


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDOBJECT nests a JSON OBJECT reference to a JSON ARRAY. The JSON OBJECT is not copied into the JSON ARRAY, only meta information is appended. The JSON OBJECT will be identified by the last index after insertion.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

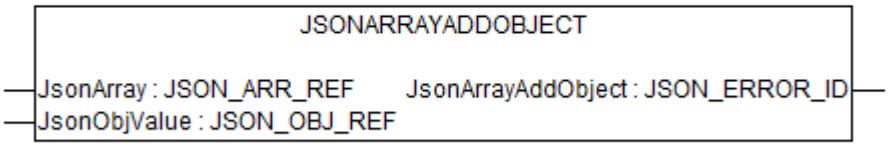
The reference to the JSON ARRAY to which the JSON OBJECT is appended.

JsonObjValue

Data type	Default value	Range	Unit
JSON_OBJ_REF	-	-	-

The JSON OBJECT reference which gets appended.

Output description

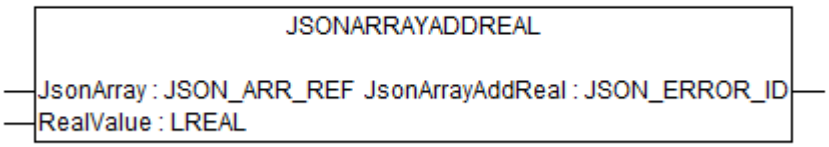


JsonArrayAdd Object

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDREAL

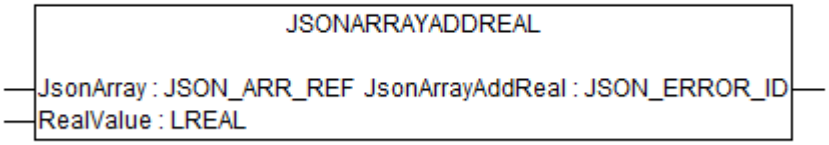


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDREAL adds a REAL value to a JSON ARRAY. The given REAL is copied into the ARRAY. The REAL will be identified by the last index after insertion.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

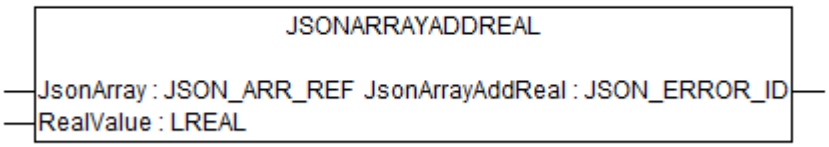
The reference to the JSON ARRAY to which the REAL is appended.

RealValue

Data type	Default value	Range	Unit
LREAL	-	-	-

The REAL value which gets appended. The input is LREAL. In case a REAL variable in CODESYS is appended the function REAL_TO_LREAL has to be used.

Output description

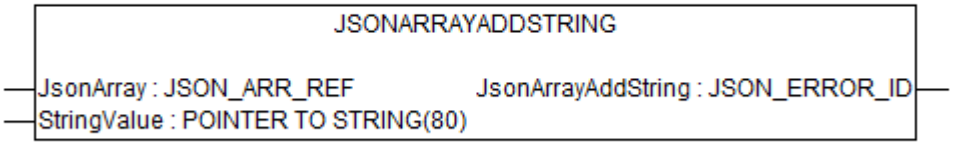


JsonArrayAddReal

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYADDSTRING

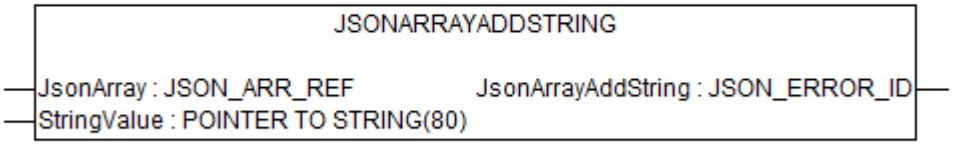


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYADDSTRING adds a STRING value to a JSON ARRAY. The given STRING is copied into an internal buffer of the ARRAY. The STRING will be identified by the last index after insertion. Different from the function figure, STRING values are allowed to have any viable length, including STRINGS that are longer than 80 characters.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY to which the STRING value is appended.

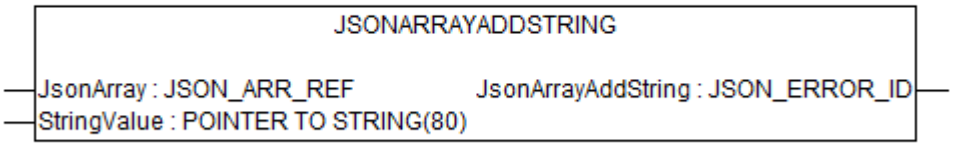
StringValue

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING value which gets appended.

Contrary to the figure the STRING value is not limited to 80 characters.
Strings of any viable length can be provided.

Output description

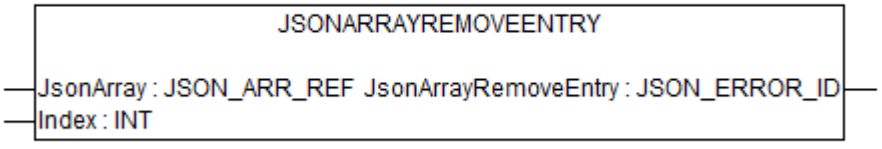


JsonArrayAdd String

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYREMOVEENTRY

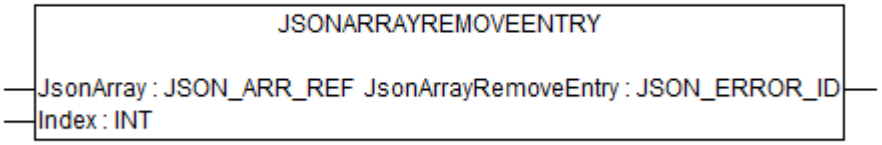


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYREMOVEENTRY removes a JSON value from a JSON ARRAY, that is identified by the given key name. All memory used by that value is freed. If the entry contains a JSON OBJECT or JSON ARRAY all contained recursive entries get freed as well.

Input description

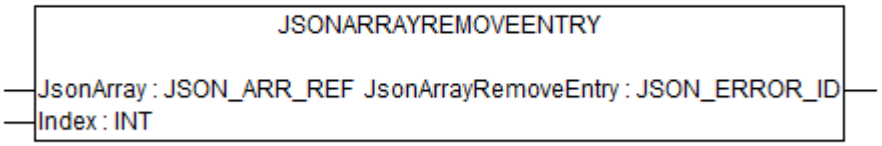


JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the value is removed.

Output description

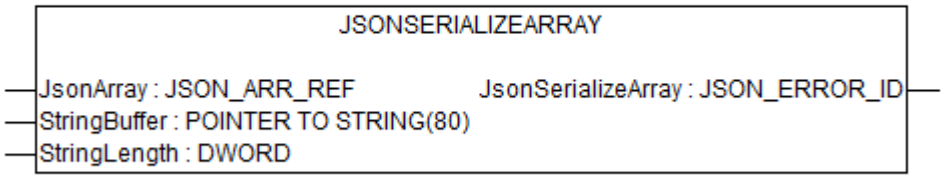


JsonArray RemoveEntry

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONSERIALIZEARRAY

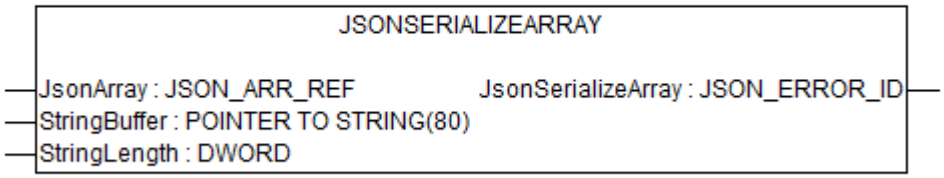


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONSERIALIZEARRAY serializes a JSON ARRAY into a given STRING buffer. Different from the figure, the STRING buffer is allowed to have any viable length, including STRINGS that are longer than 80 characters.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY which is serialized.

StringBuffer

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING buffer which is the target of the serialization.

> *Contrary to the figure the STRING buffer is not limited to 80 characters.*
>
> *STRING buffers of any viable length can be provided.*
>
> *No memory will be acquired so a target buffer has to be provided.*

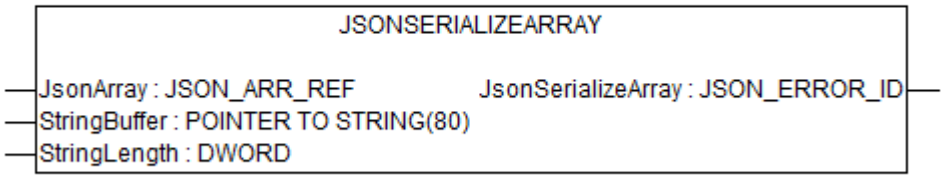
StringLength

Data type	Default value	Range	Unit
DWORD	-	-	-

The StringLength parameter specifies the limit of the provided STRING buffer.

If the size of the STRING buffer is not sufficient to contain the JSON ARRAY representation, the buffer stays unchanged.

Output description

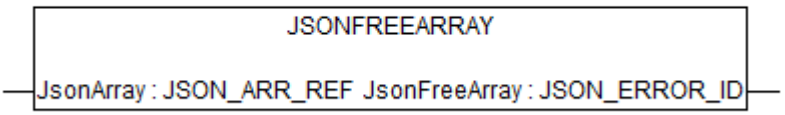


JsonSerialize Object

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONFREEARRAY

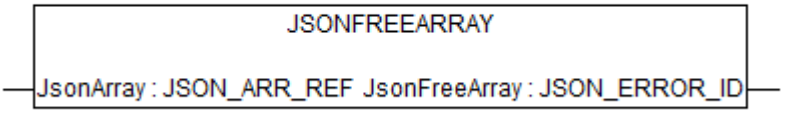


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONFREEOBJECT frees all memory used by a JSON OBJECT. All recursively contained JSON OBJECTS and JSON ARRAYS are freed as well.

Input description

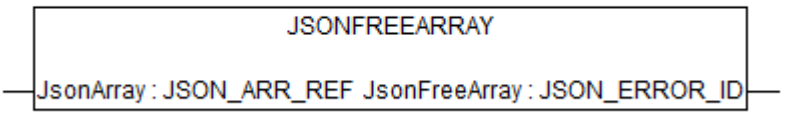


JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY which is freed. After the function returned without error, this reference becomes invalid.

Output description

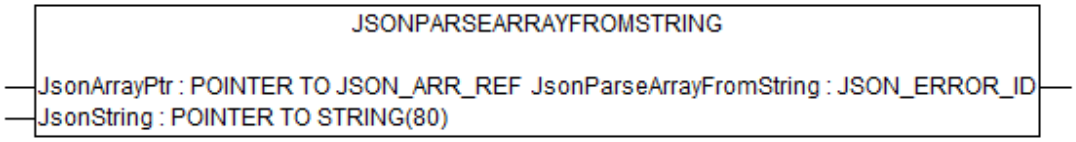


JsonFreeArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONPARSEARRAYFROMSTRING

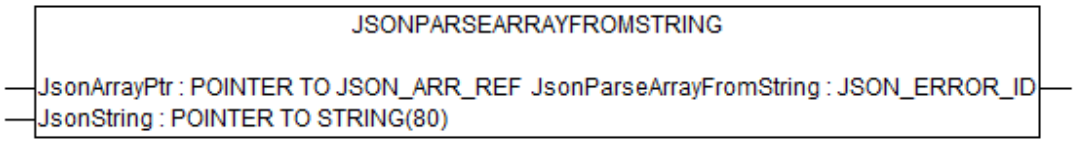


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONPARSEARRAYFROMSTRING parses a STRING, which represents a JSON ARRAY and creates a corresponding ARRAY from it. The new ARRAY contains all values from the STRING representation. The function expects a reference to a JSON ARRAY to assign the new ARRAY.

Input description



JsonArrayPtr

Data type	Default value	Range	Unit
POINTER TO JSON_ARR_REF	-	-	-

The pointer to a JSON ARRAY reference to assign the parsed ARRAY.

In case of any error, the data at the provided pointer stays unchanged.

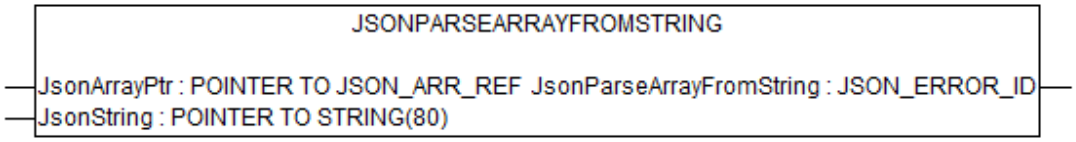
JsonString

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

JsonString is a pointer to a STRING representing a JSON ARRAY.

> *Contrary to the figure the STRING buffer is not limited to 80 characters.*
> *Strings of any viable length can be provided.*

Output description

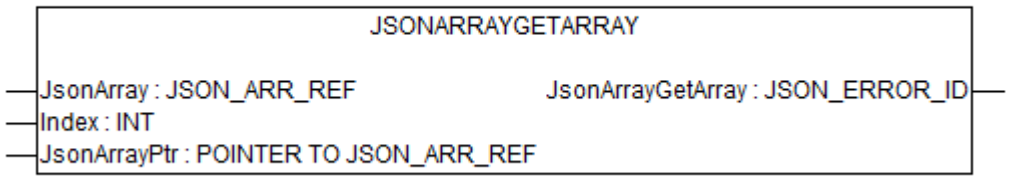


JsonParseArrayFromString

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETARRAY

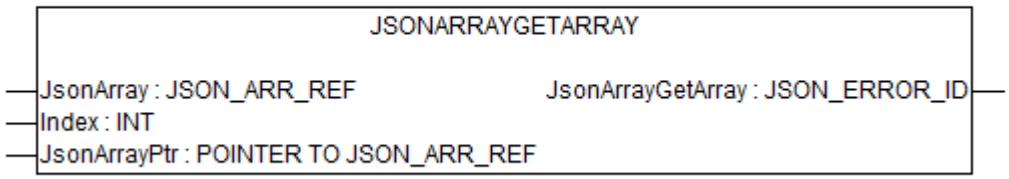


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETARRAY gets a nested child JSON ARRAY identified by an index from a parent JSON ARRAY. The ARRAY is not copied, only a reference is retrieved. This reference is assigned to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the parent JSON ARRAY from which the child JSON ARRAY is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved child JSON ARRAY.

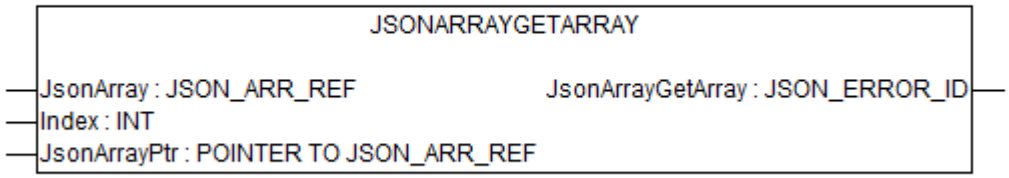
JsonArrayPtr

Data type	Default value	Range	Unit
POINTER TO JSON_ARR_REF	-	-	-

The pointer to a JSON ARRAY to assign the retrieved child ARRAY reference.

In case of any error, the data at the provided pointer stays unchanged.

Output description

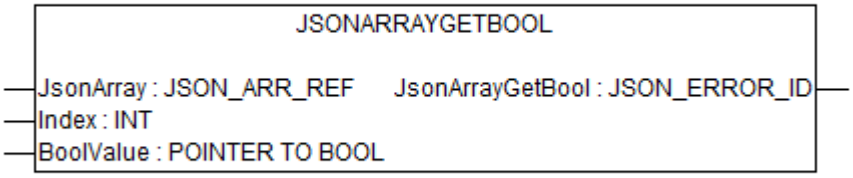


JsonArrayGetArray

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETBOOL

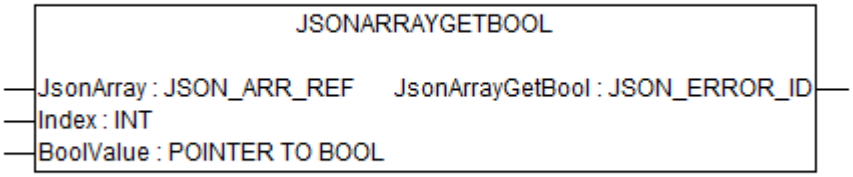


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETBOOL gets a BOOLEAN value identified by an index from a JSON ARRAY. The retrieved value is assigned to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the BOOLEAN is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved BOOLEAN value.

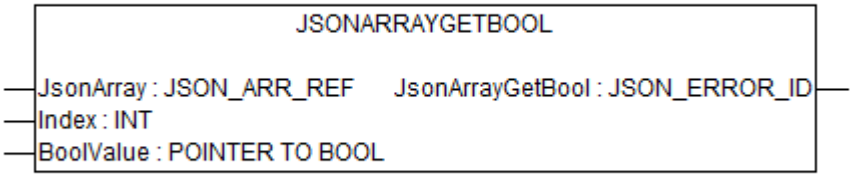
BoolValue

Data type	Default value	Range	Unit
POINTER TO BOOL	-	-	-

The pointer to a BOOLEAN variable to assign the retrieved value.

In case of any error, the data at the provided pointer stays unchanged.

Output description

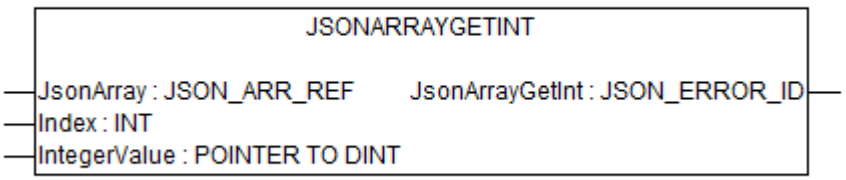


JsonArrayGetBool

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETINT

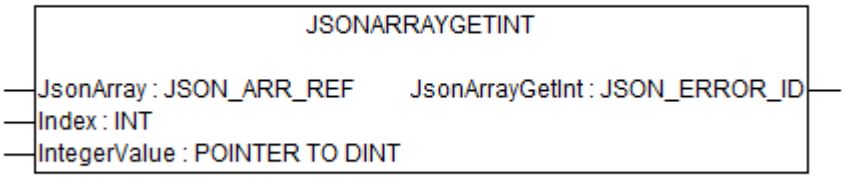


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETINT gets an INTEGER value identified by an index from a JSON ARRAY. The retrieved value is assigned to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the INTEGER value is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved INTEGER value.

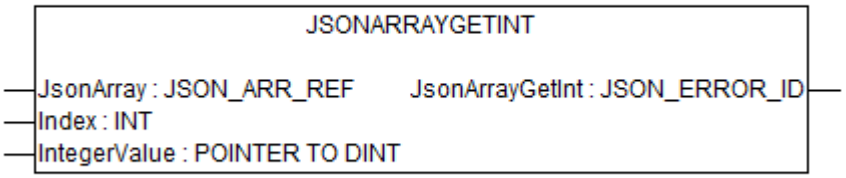
IntegerValue

Data type	Default value	Range	Unit
POINTER TO DINT	-	-	-

The pointer to an INTEGER variable to assign the retrieved value. In CODESYS a DINT variable has to be used.

In case of any error, the data at the provided pointer stays unchanged.

Output description

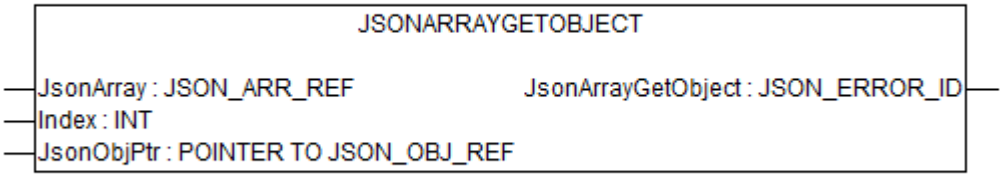


JsonArrayGetInt

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETOBJECT

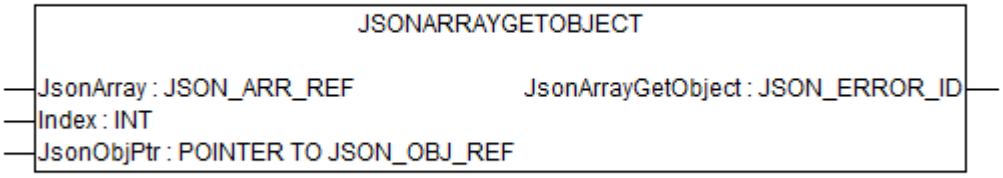


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETOBJECT gets a JSON OBJECT identified by an index from a JSON ARRAY. The OBJECT is not copied, only a reference is retrieved. This reference is assigned to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the JSON OBJECT is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved JSON OBJECT.

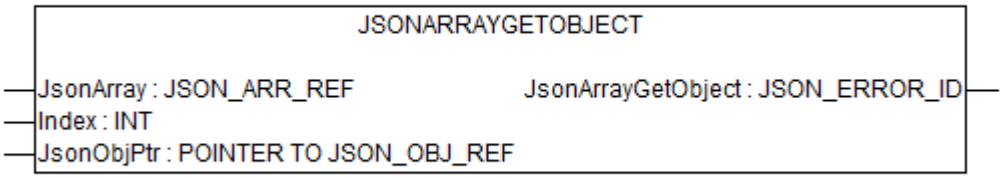
JsonObjPtr

Data type	Default value	Range	Unit
POINTER TO JSON_OBJ_REF	-	-	-

The pointer to a JSON OBJECT to assign the retrieved OBJECT reference.

In case of any error, the data at the provided pointer stays unchanged.

Output description

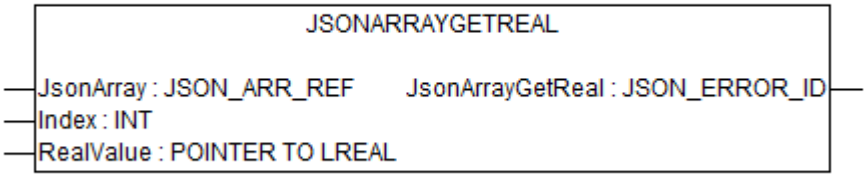


JsonArrayGetObject

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETREAL

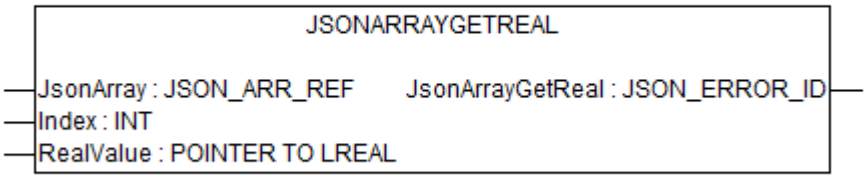


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETREAL gets a REAL value identified by an index from a JSON ARRAY. The retrieved value is assigned to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the REAL is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved REAL value.

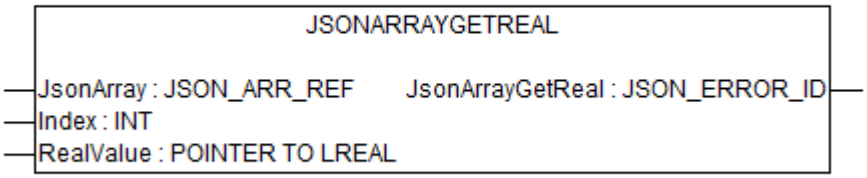
RealValue

Data type	Default value	Range	Unit
POINTER TO LREAL	-	-	-

The pointer to a REAL variable to assign the retrieved value. In CODESYS a LREAL variable has to be used.

In case of any error, the data at the provided pointer stays unchanged.

Output description

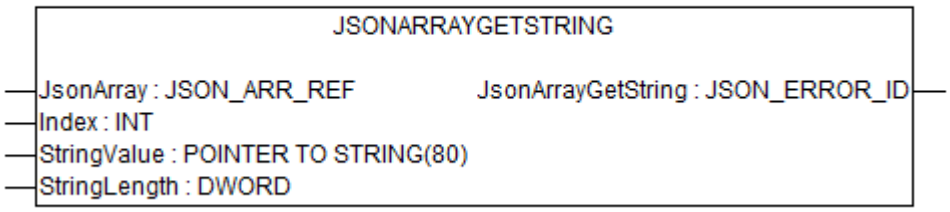


JsonArrayGetReal

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

JSONARRAYGETSTRING

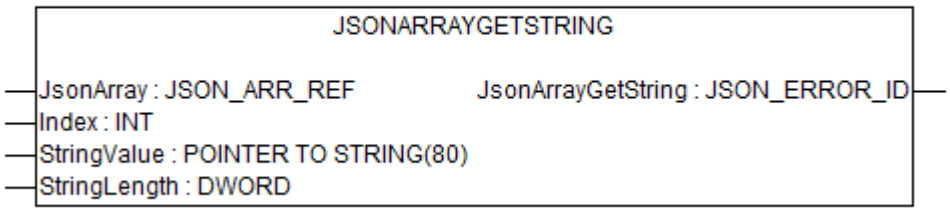


General information

Parameter	Value
Included in library	JSON_AC500_V28.lib
Available as of firmware	V2.8
Type	Function
Group	C interface

The function JSONARRAYGETSTRING gets a STRING value identified by an index from a JSON ARRAY. The STRING value is copied to the given pointer.

Input description



JsonArray

Data type	Default value	Range	Unit
JSON_ARR_REF	-	-	-

The reference to the JSON ARRAY from which the STRING value is retrieved.

Index

Data type	Default value	Range	Unit
INT	-	-	-

The index which identifies the retrieved STRING value.

StringValue

Data type	Default value	Range	Unit
POINTER TO STRING(80)	-	-	-

The pointer to a STRING buffer to copy the retrieved value.

> *Contrary to the figure the STRING value is not limited to 80 characters.*
> *STRING buffers of any viable length can be provided.*

In case of any error, the data at the provided pointer stays unchanged.

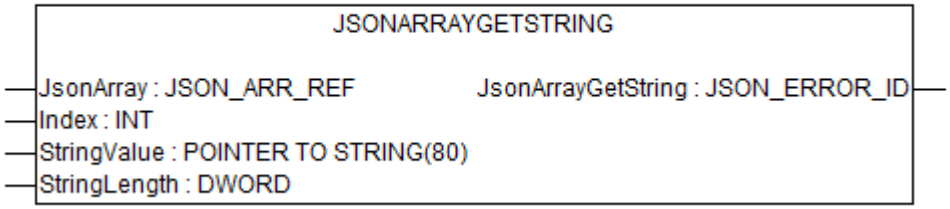
StringLength

Data type	Default value	Range	Unit
DWORD	-	-	-

The StringLength parameter specifies the limit of the provided STRING value.

If the size of the STRING value is not sufficient, it stays unchanged.

Output description



JsonArrayGetString

Data type	Default value	Range	Unit
JSON_ERROR_ID	JSON_ERR_NO_ERR	-	-

Provides an error number from enumeration JSON_ERROR_ID if an error occurred while processing the function.

Structures and enumerations

JSON_ERROR_ID

Enumeration

Parameter	Default value	Description
JSON_ERR_NO_ERR	0	No error.
JSON_ERR_PARSE_ERR	16#3015	Error on parsing given STRING. STRING does not represent a valid JSON OBJECT/ARRAY or not enough memory is available.
JSON_ERR_STRING_LEN_ERR	16#3017	Error on given STRING buffer. JSON serialization or STRING value is longer than STRING length.
JSON_ERR_KEY_NOT_EXISTING	16#301B	Error on given key. The key does not exist in the JSON OBJECT.
JSON_ERR_INDEX_NOT_EXISTING	16#301C	Error on given index. The index does not exist in the JSON ARRAY.
JSON_ERR_TYPE_ERR	16#3027	Type mismatch of value at given key in JSON OBJECT or at given index in JSON ARRAY.
JSON_ERR_MEM_ERR	16#3028	Error on allocating memory on JSON value creation or insertion.

JSON_OBJ_REF

Type

Type name	Data type	Description
JSON_OBJ_REF	DWORD	Address handle for JSON OBJECTS.

JSON_ARR_REF

Type

Type name	Data type	Description
JSON_ARR_REF	DWORD	Address handle for JSON ARRAYS.

Global variables

JSON_CONSTANTS

Parameter	Data type	Default value	Description
JSON_MAX_KEY_LEN	DWORD	100	Maximum size of a key string in a JSON OBJECT.

Hardware

The JSON Library requires an suitable processor module.

The firmware of the processor module must be minimum FW 2.8.0.

The processor module must have an Ethernet connection.

The following **processor modules** can be used:

AC500-eCo

- PM556-TP-ETH
- PM566-TP-ETH

AC500

- PM573-ETH
- PM583-ETH
- PM585-ETH
- PM590-ETH
- PM591-ETH
- PM591-2ETH
- PM592-ETH
- PM595-4ETH

Examples

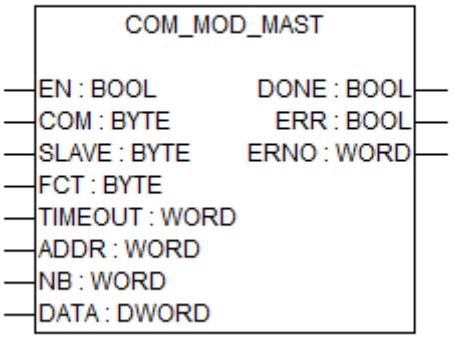
Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.4.22 Modbus library

Library file name: **Modbus_AC500_Vx.lib**

1.5.4.22.1 Function blocks

COM_MOD_MAST



Parameter	Value
Included in library	Modbus_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	Modbus

The function block COM_MOD_MAST realizes the Modbus master function for the Modbus interface of the controller (COM1, COM2) specified at input COM.

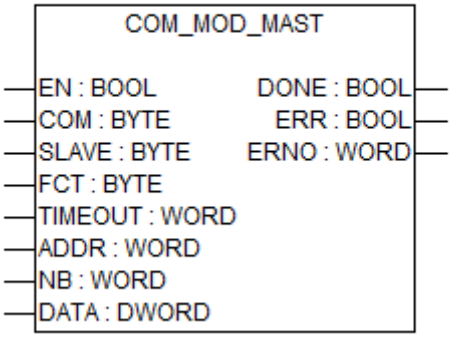
For each interface, a separate COM_MOD_MAST function block must be used.

Prior to the use of COM_MOD_MAST for an interface, the particular interface has to be configured via the controller configuration of the Control Builder (PS501) as the Modbus master interface.

With each FALSE > TRUE edge at input EN, the function block COM_MOD_MAST reads the values at the inputs, generates a telegram according to the inputs and sends this telegram to the slave.

For detailed information on Modbus RTU, see ↬ Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At input COM, the Modbus interface number is specified.

COM = 1: COM1

COM = 2: COM2

SLAVE

Data type	Default value	Range	Unit
BYTE	-	-	-

At input SLAVE, the address of the slave to which the telegram shall be sent is specified.

FCT

Data type	Default value	Range	Unit
BYTE	-	-	-

The function code of the request telegram is specified at input FCT.

01 or 02	Read n bits
03 or 04	Read n words
05	Write one bit (encoded in one word)
06	Write one word
07	Read 8 bit

15	Write n bits (encoded in one byte)
16	Write n words
22	Mask write
23	Read/write multiple words in one telegram

TIMEOUT

Data type	Default value	Range	Unit
WORD	-	-	ms

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified in TIMEOUT, the procedure is aborted and an error identifier is output.

Keeping the timeout depends on the cycle time of the task in which the MODMAST function block is processed. The real time may deviate from the specification in worst case by task cycle time - 1 ms.

ADDR

Data type	Default value	Range	Unit
WORD	-	-	-

The operand/register address in the slave from which data should be read or written is specified at input ADDR.

The access to operands of AC500 devices in Modbus slave mode is defined via the Modbus cross-reference list. Only operands that are listed in the cross-reference list may be used (*Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*).

Only operands that are listed in the Modbus address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

NB

Data type	Default value	Range	Unit
DWORD	Empty string	-	-

At input NB (number), the number of data to be written or read is specified. The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB. The following restrictions apply to the length:

FCT		NBmax	
Dec	**Hex**	**Serial**	**Modbus on TCP/IP**
01 or 02	01 or 02	2000 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
03 or 04	03 or 04	125 words / 62 double words	125 words / 62 double words
05	05	1 bit	1 bit
06	06	1 word	1 word

FCT		NBmax	
Dec	**Hex**	**Serial**	**Modbus on TCP/IP**
07	07	8 bits	8 bits
15	0F	1968 bits	255 bits (up to firmware version V01.33) 1800 bits (from firmware version V01.41) 1536 bits (PM573/PM583 only)
16	10	123 words / 61 double words	123 words / 61 double words
22	16	Write: 1 word	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 123 words / 61 double words	Read: 125 words / 62 double words Write: 123 words / 61 double words

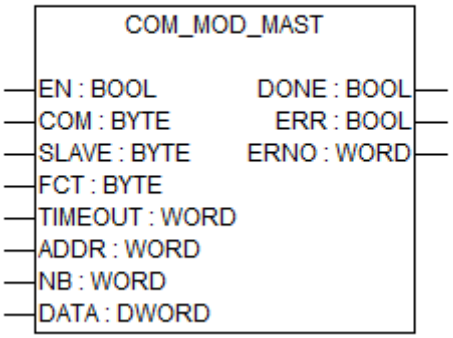
DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

At input DATA, the address of the first operand in the master is specified, from which data are copied/written to the slave or to which the data read by the slave should be stored. For this purpose it is necessary that the operand type (e.g. bit) matches the selected function (e.g. FCT 1, read n bits).

If using Modbus function codes 22 or 23, the according data structures COM_MOD_FCT22_TYPE *Chapter 1.5.4.22.2.1 “COM_MOD_FCT22_TYPE” on page 1702* or COM_MOD_FCT23_TYPE *Chapter 1.5.4.22.2.2 “COM_MOD_FCT23_TYPE” on page 1703* must be defined and applied to DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ModMast  (EN      := ModMast_EN,
          COM     := ModMast_COM,
          SLAVE   := ModMast_SLAVE,
          FCT     := ModMast_FCT,
          TIMEOUT := ModMast_TIMEOUT,
          ADDR    := ModMast_ADDR,
          NB      := ModMast_NB,
          DATA    := ADR(ModMast_DATA));


ModMast_DONE := ModMast.DONE;
ModMast_ERR := ModMast.ERR;
ModMast_ERNO:= ModMast.ERNO;
```

1.5.4.22.2 Structures

COM_MOD_FCT22_TYPE

Parameter	Value
Included in library	Modbus_AC500_V10.lib
Available as of firmware	V2.2

This structure is used to handle the Modbus function 22 with the function block COM_MOD_MAST *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*. The structure has to be used at the DATA input with an ADR(instance_of_struct) operator.

Type and the content need to be created and defined by the user.

Visible variable	Type	Default value	Description
wAND_Mask	WORD	0	AND mask
wOR_Mask	WORD	0	OR mask

COM_MOD_FCT23_TYPE

Parameter	Value
Included in library	Modbus_AC500_V10.lib
Available as of firmware	V2.2

This structure is used to handle the Modbus function 23 with the function block COM_MOD_MAST ↬ *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*. The structure has to be used at the DATA input with an ADR(instance_of_struct) operator.

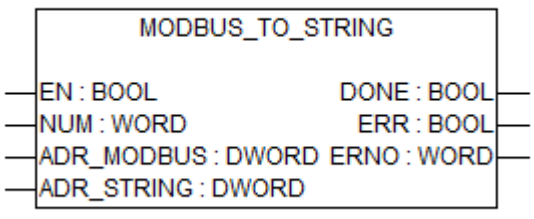
Modbus function 23 could only be used with CPUs with onboard Ethernet.

Type and the content need to be created and defined by the user.

Visible variable	Type	Default value	Description
pByDataWrite	POINTER TO BYTE	0	Pointer to buffer containing data to write. Compare FCT 16 input DATA
pByDataRead	POINTER TO BYTE	0	Pointer to buffer to store data to read. Compare FCT 03 input DATA
wDataAddress-Read	WORD	0	Address of the data to be read. Compare FCT 16 input ADDR
wNumDataUnits-Read	WORD	0	Number of data units to be read. Compare FCT 16 input NB

1.5.4.22.3 Programs

MODBUS_TO_STRING



Parameter	Value
Included in library	Modbus_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Program

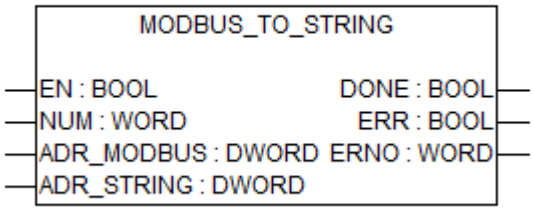
The Program MODBUS_TO_STRING converts data received or data to be send via Modbus to a String.

It does not change the value of the data but swaps and aligns data correctly. So it finally just rearranges the original data.

The output String is NULL-terminated. Any Byte in input data containing value ‘0’ is also set to ‘\0’ in output String. In this case output String is shorter than NUM or variable referenced at input ADR_STRING contains multiple strings with an overall length of NUM.

The Program MODBUS_TO_STRING can be used for Modbus Master as well as for Modbus Slave.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

NUM

Data type	Default value	Range	Unit
WORD	-	0..65536	-

The input NUM specifies the number of Bytes contained in variable referenced at input ADR_MODBUS to be converted and put to variable referenced at input ADR_STRING.

ADR_MODBUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADR_MODBUS specifies the start address of the Modbus payload to be converted.

The total length of the data is assumed to be NUM Bytes long. It may contain any values. The value 0 is treated as string separator/terminator.

ADR_STRING

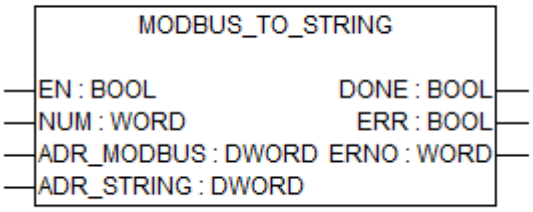
Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADR_STRING specifies the start address of the String to contain the converted Modbus payload.

The total size/length of this variable is assumed to be at least NUM characters.

Since the input data may contain any values (even 0), after conversion ADR_STRING may contain one String shorter than NUM or multiple ones of NUM characters overall length.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

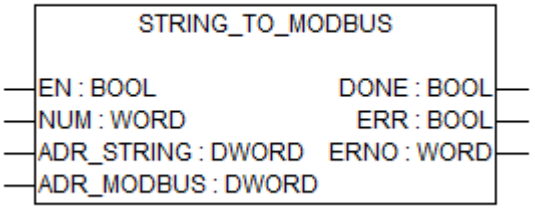
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STRING_TO_MODBUS



Parameter	Value
Included in library	Modbus_AC500_V10.lib
Available as of firmware	V1.0.2
Type	Program

The Program STRING_TO_MODBUS converts a String to data to be send via Modbus.

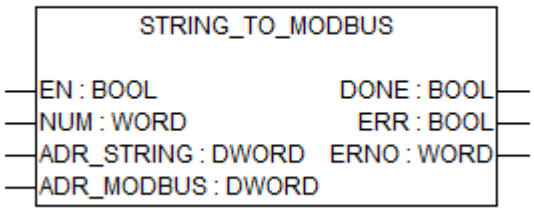
It does not change the value of the data but swaps and aligns data correctly. So it finally just re-arranges the original String.

Any terminating character (‘\0’) in input String is also set to ‘0’ in output data.

Since Program converts NUM characters in any case (independent of any terminating characters in between) input data may contain multiple Strings in this case.

The Program STRING_TO_MODBUS can be used for Modbus Master as well as for Modbus Slave.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

NUM

Data type	Default value	Range	Unit
WORD	-	0..65536	-

The input NUM specifies the number of Characters contained in variable referenced at input ADR_STRING to be converted and put to variable referenced at input ADR_MODBUS.

ADR_STRING

Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADR_STRING specifies the start address of the String(s) to contain the converted to Modbus payload.

The total size/length of this variable is assumed to be NUM characters.

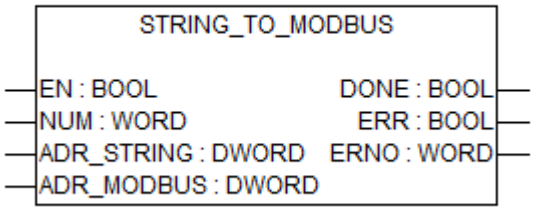
ADR_MODBUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Input ADR_MODBUS specifies the start address of the Modbus payload to contain the converted input String(s).

The total size of the data is assumed to be at least NUM Bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

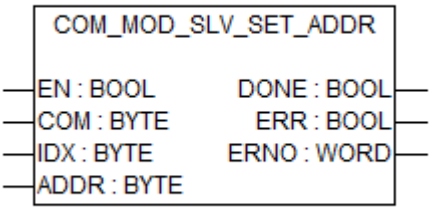
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

1.5.4.23 Extended Modbus library

Library file name: **MODBUS_Ext_AC500_Vx.lib**

1.5.4.23.1 Function blocks

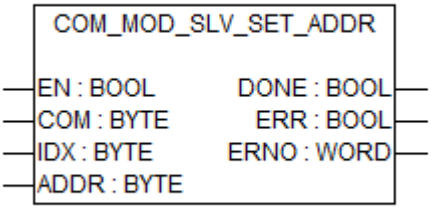
COM_MOD_SLV_SET_ADDR



Parameter	Value
Included in library	MODBUS_Ext_AC500_V10.lib MODBUS_Ext_AC500_V20.lib
Available as of firmware	V1.2.4 (MODBUS_Ext_AC500_V10.lib) V2.0.0 (MODBUS_Ext_AC500_V20.lib)
Type	Function block
Group	Modbus

The function block allows to set the address of a Modbus slave in serial mode during run time using the MULTI protocol.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

Input COM specifies the COM port where the MULTI protocol is configured.

IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

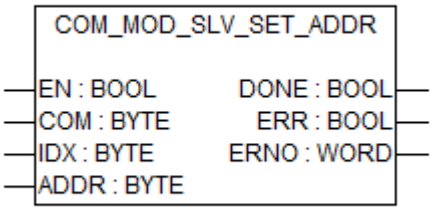
Input IDX specifies the index of the sub-module that should be used for the Modbus communication.

ADDR

Data type	Default value	Range	Unit
BYTE	-	-	-

Input ADDR specifies the address that should be set for the corresponding Modbus slave.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

1.5.4.24 MQTT client library

1.5.4.24.1 Function blocks

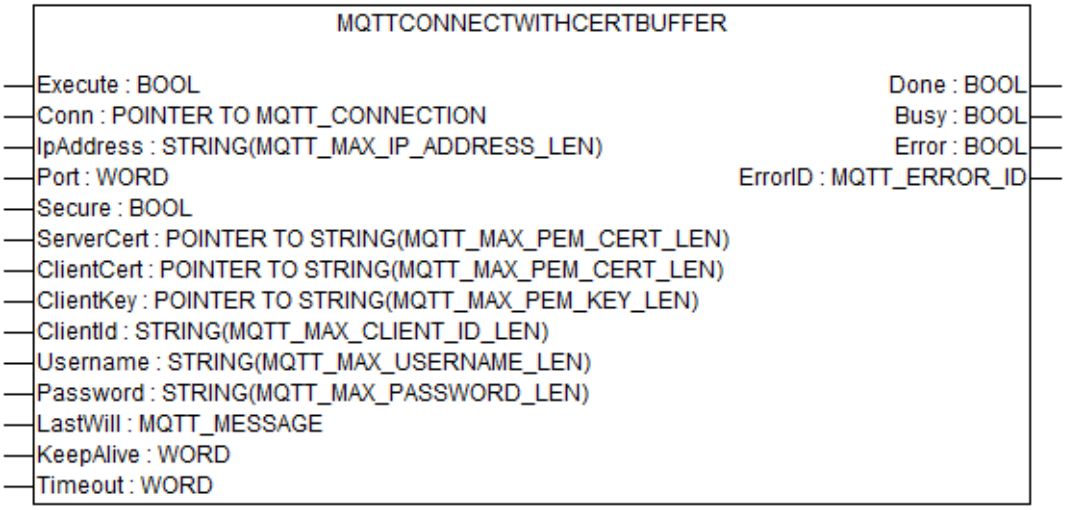
The function blocks in this library can only be executed in RUN mode of the processor module, not in simulation mode.

MQTT client library is based on OASIS MQTT specification (v3.1.1).

MQTT client protocol

For further information see *Chapter 1.6.5.3.4 “MQTT client protocol” on page 6178*

MqttConnectWithCertBuffer



Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

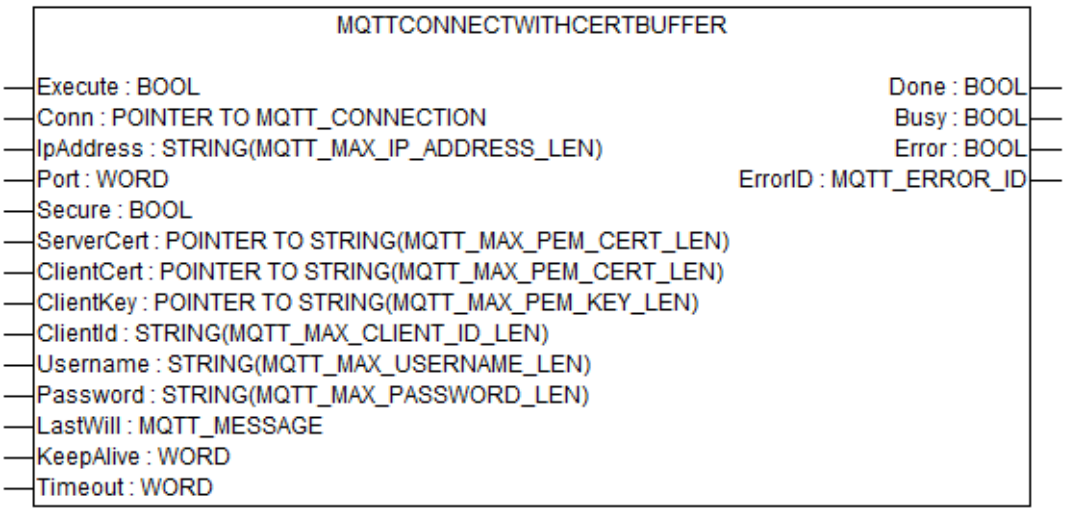
The function block MqttConnectWithCertBuffer establishes a connection to a MQTT broker. This function block only has to be called once per connection. The input parameters of the function block are used to access to the broker.

Comparing to function block MqttConnectWithCertFile it is possible to establish a TLS connection with certificates from buffer.

In this case copy the content of the certificate .pem file into a STRING variable. All line breaks from the file must be replace with the ST specific line break character ‘$n’.

Each time a TLS connection is established to a broker, the AC500 calculates some key pairs. For AC500-eCo PLCs, this calculation can take up to 7 seconds. Hence, ensure correct configuration of your watchdog.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNECTION	0	-	-

Set a reference to the connection.

MQTT_CONNECTION is used for other function blocks to reference to the established connection.

IpAddress

Data type	Default value	Range	Unit
STRING	Empty string	-	-

Set the IP address of the MQTT Broker. The IP address needs to be a string like ‘192.168.0.1’.

Port

Data type	Default value	Range	Unit
WORD	8883	-	-

Set the port number of the MQTT broker.

Secure

Data type	Default value	Range	Unit
BOOL	TRUE	-	-

Decide if using secure channel (TLS, Secure := TRUE) for communication.

ServerCert

Data type	Default value	Range	Unit
POINTER TO STRING(MQTT_MAX_PEM_CERT_LEN)	0	-	-

Set server certificate in PEM format. Only necessary if using a TLS connection. MQTT_MAX_PEM_CERT_LEN is an internal constant which is set to 3072.

ClientCert

Data type	Default value	Range	Unit
POINTER TO STRING(MQTT_MAX_PEM_CERT_LEN)	0	-	-

Set (optional) client certificate in PEM format. Only necessary if using a TLS connection. MQTT_MAX_PEM_CERT_LEN is an internal constant which is set to 3072.

ClientKey

Data type	Default value	Range	Unit
POINTER TO STRING(MQTT_MAX_PEM_CERT_LEN)	0	-	-

Set (optional) client certificate in PEM format. Only necessary if using a TLS connection and if using a client certificate. MQTT_MAX_PEM_CERT_LEN is an internal constant which is set to 3072.

ClientId

Data type	Default value	Range	Unit
POINTER TO STRING(MQTT_MAX_PEM_CERT_LEN)	0	-	-

Set client identifier which is used to identify the client to the server. Only necessary if using a TLS connection. MQTT_MAX_CLIENT_ID_LEN is an internal constant which is set to 250.

Username

Data type	Default value	Range	Unit
STRING(MQTT_MAX_USERNAME_LEN)	Empty string	-	-

Set username for MQTT broker. MQTT_MAX_USERNAME_LEN is an internal constant which is set to 250.

Password

Data type	Default value	Range	Unit
STRING(MQTT_MAX_PASSWORD_LEN)	Empty string	-	-

Set password for MQTT broker. MQTT_MAX_PASSWORD_LEN is an internal constant which is set to 250.

LastWill

Data type	Default value	Range	Unit
MQTT_MESSAGE	-	-	-

Optional Last Will message.

KeepAlive

Data type	Default value	Range	Unit
WORD	600	-	-

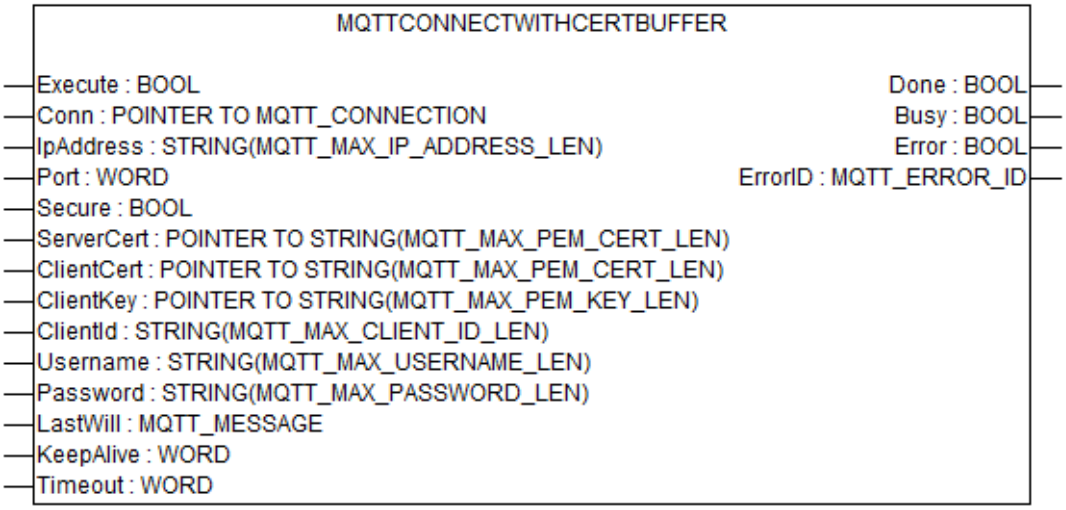
Time in seconds, which will be passed to the MQTT Broker and which is normally used by the Broker to disconnect clients if no communication was made for 1,5 * KeepAlive.

Timeout

Data type	Default value	Range	Unit
WORD	30000	-	Millisecond

Timeout for connect and all subsequent calls (ping, subscribe, unsubscribe, publish with QoS > 0).

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

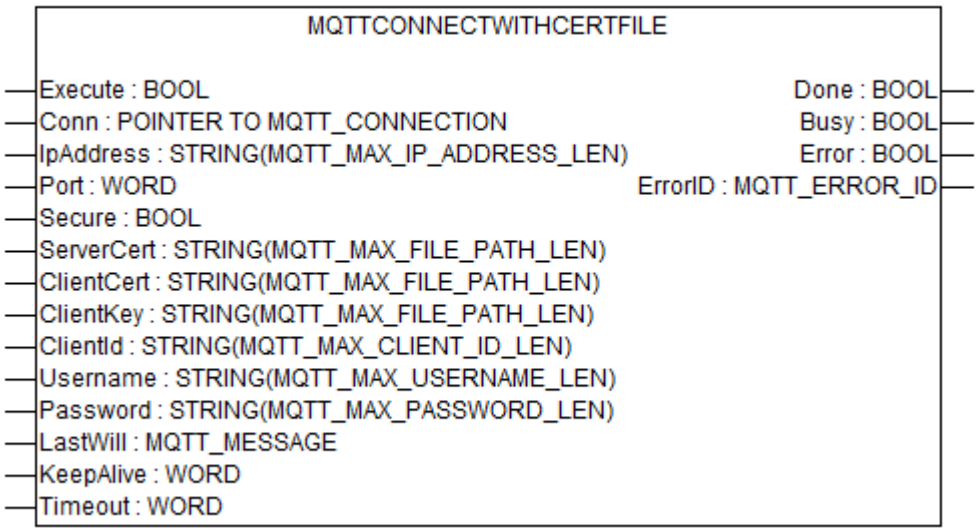
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

MqttConnectWithCertFile



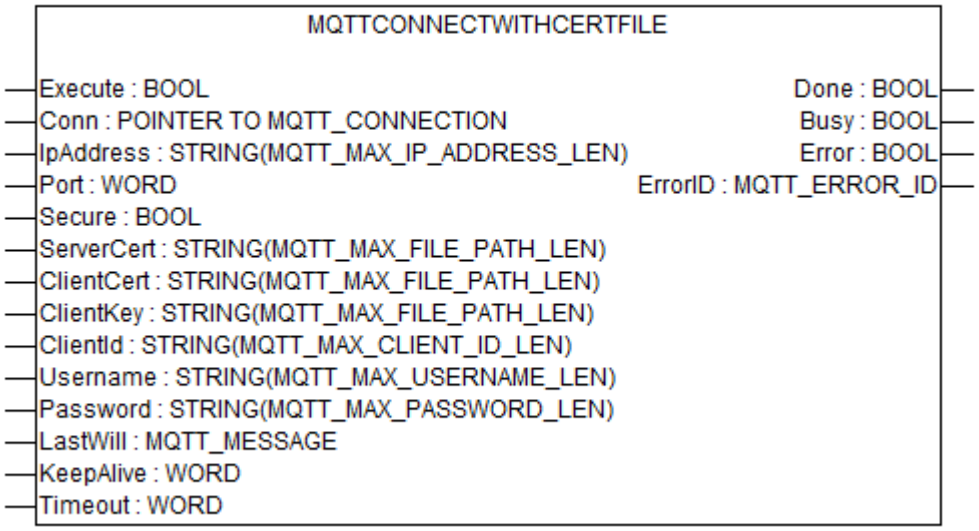
Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

The function block MqttConnectWithCertFile establishes a connection to a MQTT broker. This function block only has to be called once per connection. The input parameters of the function block are used to access to the broker.

Comparing to function block MqttConnectWithCertBuffer it is possible to establish a TLS connection with certificates from file.

If your PLC does not have a persistent file storage like memory card or flash disk, consider using MqttConnectWithCertBuffer.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNEC-TION	0	-	-

Set a reference to the connection.

MQTT_CONNECTION is used for other function blocks to reference to the established connection.

IpAddress

Data type	Default value	Range	Unit
STRING	Empty string	-	-

IP address of the MQTT Broker. The IP address needs to be a string like ‘192.168.0.1’.

Port

Data type	Default value	Range	Unit
WORD	8883	-	-

Set the port number of the MQTT broker.

Secure

Data type	Default value	Range	Unit
BOOL	TRUE	-	-

Decide if using secure channel (TLS, Secure := TRUE) for communication.

ServerCert

Data type	Default value	Range	Unit
STRING(MQTT_MAX_FILE_PATH_LEN)	Empty string	-	-

File name of server certificate in PEM format. Only necessary if using a TLS connection.

ClientCert

Data type	Default value	Range	Unit
STRING(MQTT_MAX_FILE_PATH_LEN)	Empty string	-	-

File name of client certificate in PEM format. Only necessary if using a TLS connection.

ClientKey

Data type	Default value	Range	Unit
STRING(MQTT_MAX_FILE_PATH_LEN)	Empty string	-	-

File name of client private key in PEM format (optional). Only necessary if using a TLS connection and if using a client certificate.

ClientId

Data type	Default value	Range	Unit
POINTER TO STRING(MQTT_MAX_PEM_CERT_LEN)	0	-	-

Set client identifier which is used to identify the client to the server. Only necessary if using a TLS connection. MQTT_MAX_CLIENT_ID_LEN is an internal constant which is set to 250.

Username

Data type	Default value	Range	Unit
STRING(MQTT_MAX_USERNAME_LEN)	Empty string	-	-

Set username for MQTT broker. MQTT_MAX_USERNAME_LEN is an internal constant which is set to 250.

Password

Data type	Default value	Range	Unit
STRING(MQTT_MAX_PASSWORD_LEN)	Empty string	-	-

Set password for MQTT broker. MQTT_MAX_PASSWORD_LEN is an internal constant which is set to 250.

LastWill

Data type	Default value	Range	Unit
MQTT_MESSAGE	-	-	-

Optional Last Will message.

KeepAlive

Data type	Default value	Range	Unit
WORD	600	-	-

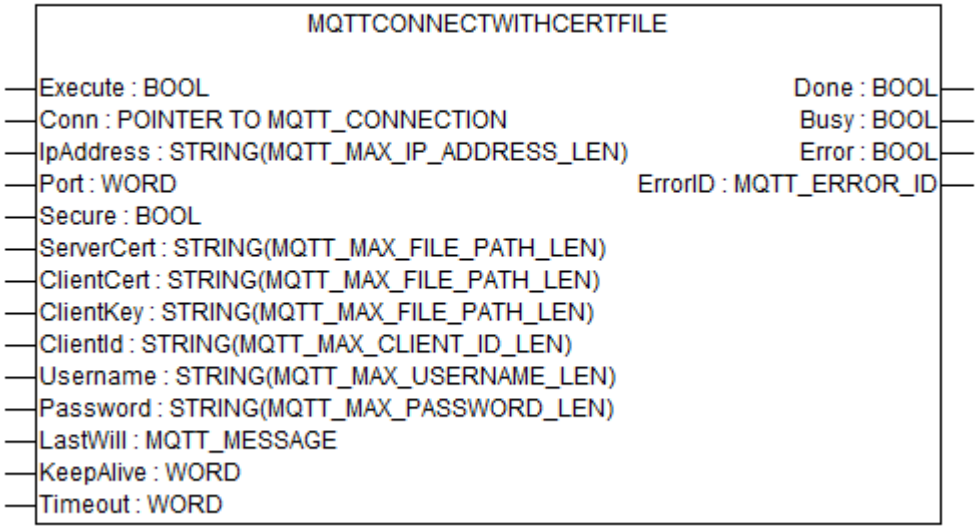
Time in seconds, which will be passed to the MQTT Broker and which is normally used by the Broker to disconnect clients if no communication was made for 1,5 * KeepAlive.

Timeout

Data type	Default value	Range	Unit
WORD	30000	-	Millisecond

Timeout for connect and all subsequent calls (ping, subscribe, unsubscribe, publish with QoS > 0).

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

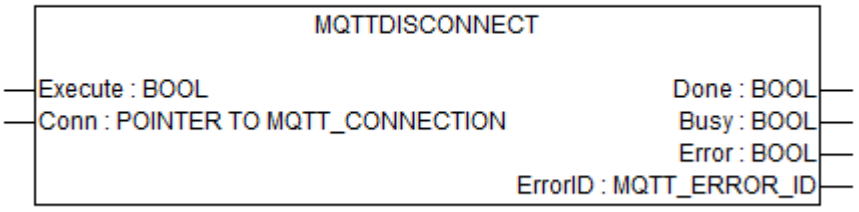
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

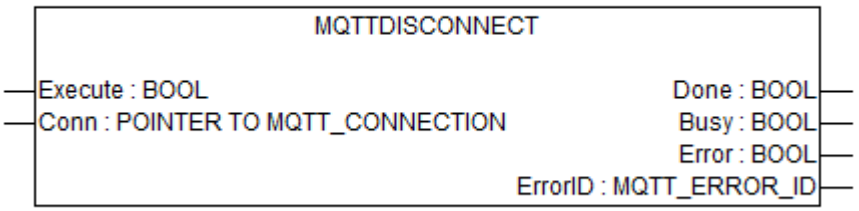
MqttDisconnect



Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

The function block MqttDisconnect is used to disconnect from the MQTT Broker. This function block resets the used socket. In case of an error during the connection, the used socket will be reset automatically.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

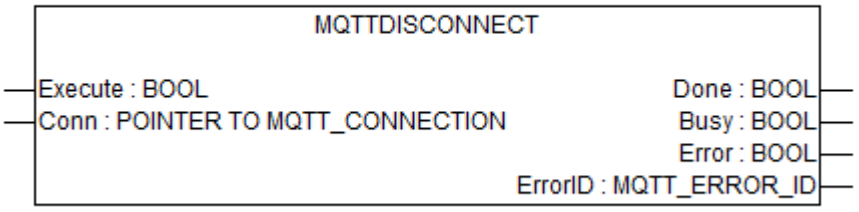
A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNECTION	0	-	-

Pointer to a valid connection structure created by MqttConnect.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

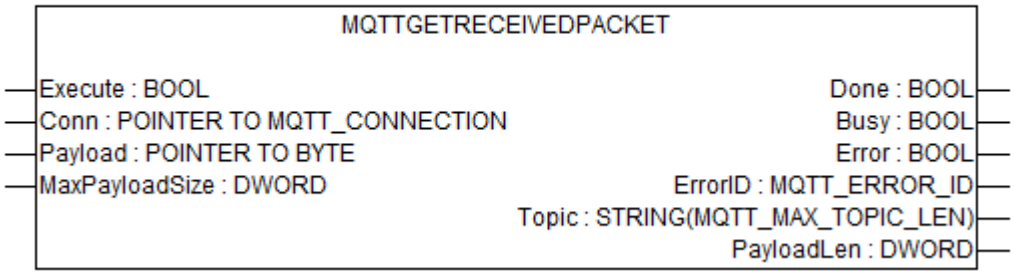
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

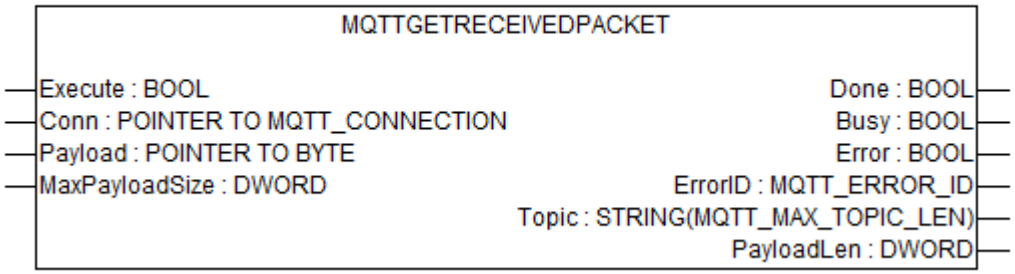
MqttGetReceivedPacket



Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

The function block MqttGetReceivedPacket returns the first packet received for any subscribed topic. Done = TRUE with PayloadLen = 0 indicates that nothing new has been received since the last call..

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNEC-TION	0	-	-

Pointer to a valid connection structure created by MqttConnect.

Payload

Data type	Default value	Range	Unit
POINTER TO BYTE	0	-	-

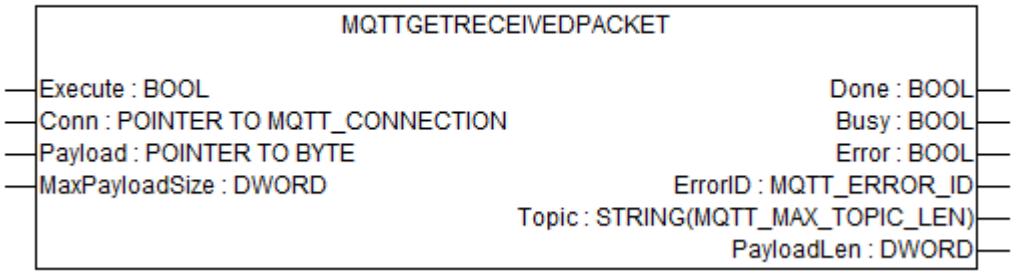
Pointer to the data area where the received packet can be stored.

MaxPayloadSize

Data type	Default value	Range	Unit
WORD	0	-	-

Size of the data area where the received packet can be stored.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *"MQTT_ERROR_ID (Enum) " on page 1729* if an error occurred while processing the function block.

Topic

Data type	Default value	Range	Unit
STRING(MQTT_MAX_TOPIC_LEN)	Empty string	-	-

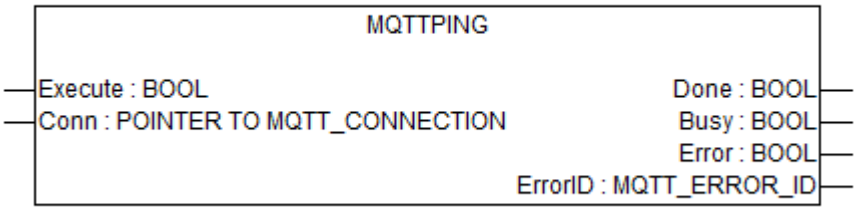
Topic where the payload data belongs to.

PayLoadLen

Data type	Default value	Range	Unit
STRING(MQTT_MAX_TOPIC_LEN)	Empty string	-	-

Actual length of the payload.

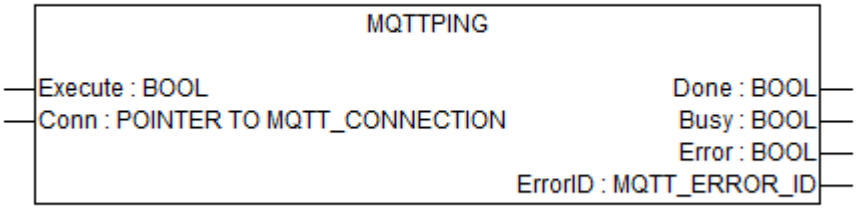
MqttPing



Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

MqttPing sends an MQTT Ping request to the broker and waits for the response packet. If output Done is TRUE and output Error is FALSE, the client still reaches the broker.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

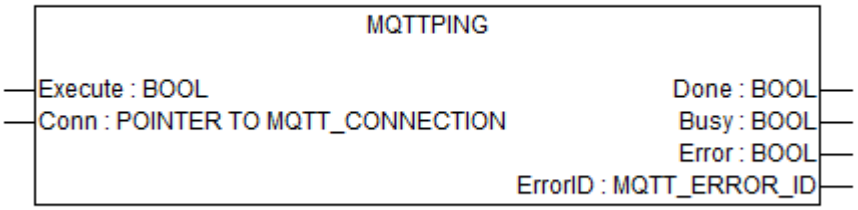
A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNECTION	0	-	-

Pointer to a valid connection structure created by MqttConnect.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

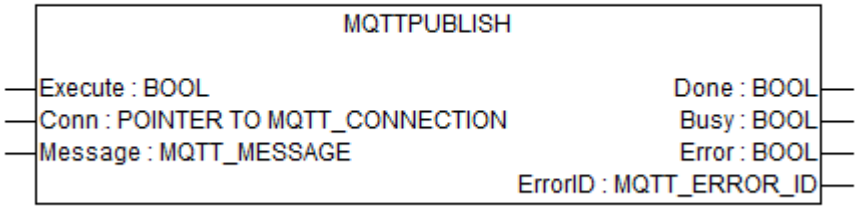
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

MqttPublish



Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

MqttPublish publishes a MQTT message to MQTT Broker. The properties of the message (Topic, Payload, QOS, Retain Flag) can be set with the message input.

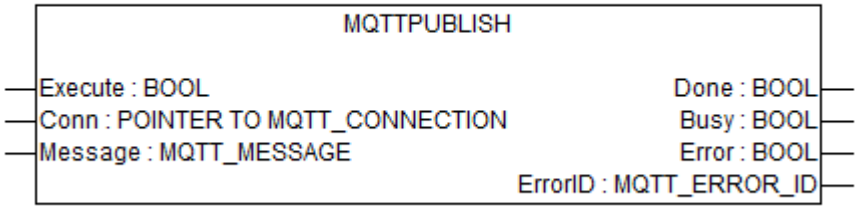
The *Timeout* that was set on the *MqttConnect POUs* is used when a QoS Level of 1 or 2 is set for a message.

This means that this POU will wait for an answer from the broker until the *Timeout* is reached when the message is either Q0s Level 1 or Level 2.

It will not wait for any answer when QoS Level is 0.

The application using this FB has to define if the message shall be re-published and how often that shall be done.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNECTION	0	-	-

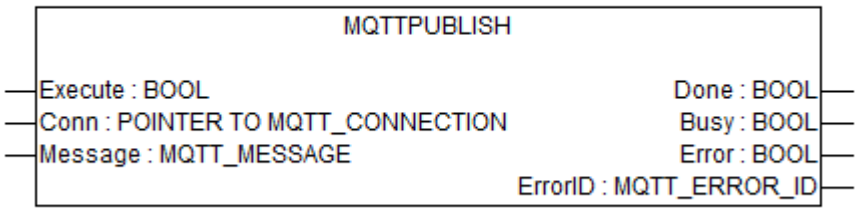
Pointer to a valid connection structure created by MqttConnect.

Message

Data type	Default value	Range	Unit
MQTT_MESSAGE	See description of MQTT_MESSAGE	-	-

Defines the Message to be published.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

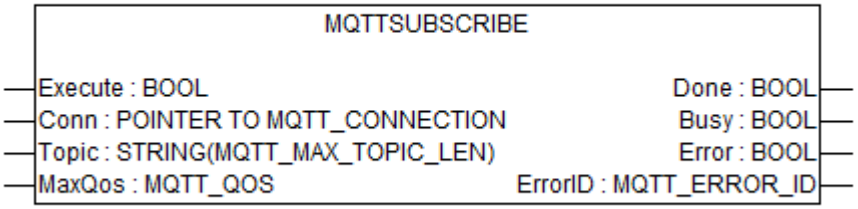
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

MqttSubscribe

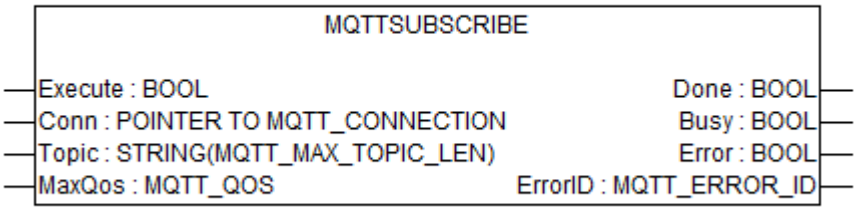


Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

The function block MqttSubscribe is used to subscribe to a topic name like 'topic/name'. Wildcard symbols (#/+).

It can be used for subscribing like city/# or city/+/temperature. If the connection between the client and the broker is interrupted, the client needs to subscribe again to a topic after a new connect.

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNECTION	0	-	-

Pointer to a valid connection structure created by MqttConnect.

Topic

Data type	Default value	Range	Unit
STRING(MQTT_MAX_TOPIC_LEN)	Empty string	-	-

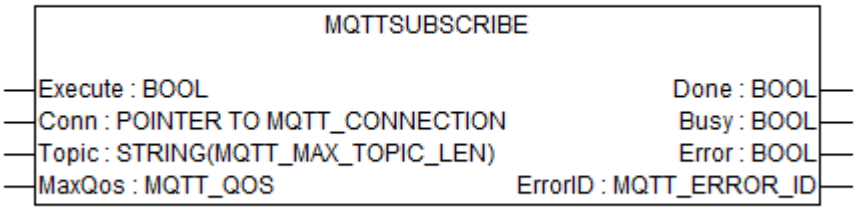
Set the Topic name as a string. The topic name must look like ‘topic/name’. Wildcard symbols can also be used for subscribing (for example: ‘city/#’ or ‘city/+/temperature’). For further information see the MQTT specification.

MaxQos

Data type	Default value	Range	Unit
MQTT_QOS	QOS_0	BASC_NORMAL, BASC_REVERSE	-

This input is used to signal the server which is the highest QoS level which can be handled. Can be reduced to relive the client from high network load.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

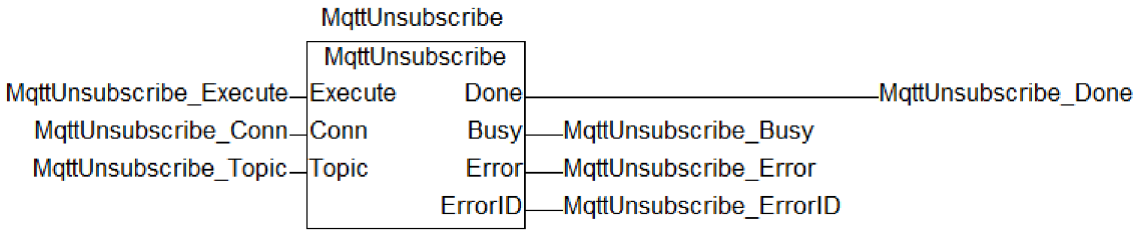
Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *“MQTT_ERROR_ID (Enum) ” on page 1729* if an error occurred while processing the function block.

MqttUnsubscribe

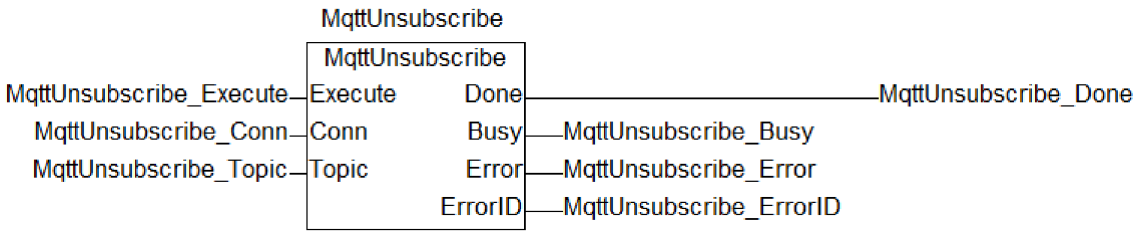


Parameter	Value
Included in library	ABB_MqttClient_AC500.lib
Available as of firmware	V2.8
Type	Function block with historical values
Group	C interface

The function block MqttUnsubscribe is used to unsubscribe from a topic name like 'topic/name'.

Wildcard symbols (#/+) can be used for subscribing like city/# or city/+/temperature

Input description



Execute

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

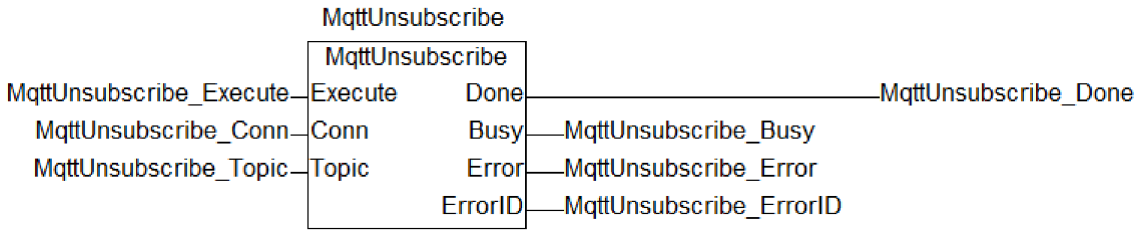
A rising edge starts the operation, the output Busy goes to TRUE. In the first cycle all other inputs are read and stored, afterwards they are ignored. A falling edge does not stop the operation. After Done = TRUE or Error = TRUE and Execute = FALSE all outputs will be reset.

Conn

Data type	Default value	Range	Unit
POINTER TO MQTT_CONNEC-TION	0	-	-

Pointer to a valid connection structure created by MqttConnect.

Output description



Done

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is completed without error (while outputs Busy and Error are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

Busy

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

Error

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is stopped with error (while outputs Busy and Done are FALSE). This output is TRUE for at least one cycle or until Execute is set to FALSE.

ErrorID

Data type	Default value	Range	Unit
MQTT_ERROR_ID	MQTT_ERR_NO_ERROR	-	-

Provides an error number from enumeration *"MQTT_ERROR_ID (Enum) " on page 1729* if an error occurred while processing the function block.

1.5.4.24.2 Structures and enumerations

MQTT_ERROR_ID (Enum)

Parameter	Value	Description
MQTT_ERR_NO_ERROR	0	No error.
MQTT_ERR_CONN_SERVICE_UNAVAIL	16#3001	The Network Connection has been made but the MQTT service is unavailable on the specified port.
MQTT_ERR_COMMUNICATION_TIMEOUT	16#3013	The timeout value for the communication has been exceeded.
MQTT_ERR_REC_PACKET_TOO_LONG	16#3017	Received topic is too long.
MQTT_ERR_PING_NO_ANSWER	16#301A	The MQTT broker did not answer the ping. MQTT client has passed the KeepAlive or MQTT broker is unreachable.
MQTT_ERR_CONN_CLIENT_ID_NOT_ALLOWED	16#301F	The Client identifier is correct UTF-8 but not allowed by the Server.

Parameter	Value	Description
MQTT_ERR_CONN_ REFUSED_PROTOCOL	16#3020	The Server does not support the level of the MQTT protocol requested by the Client.
MQTT_ERR_CONN_REFU SED_CONNECTION	16#3025	Connection refused, maybe the IP address is malformed.
MQTT_ERR_UNSPECI-FIED_ERROR	16#302B	Internal library returned an unspecified error.
MQTT_ERR_NET-WORK_ERROR	16#302D	General network error.
MQTT_ERR_CONN_AUTH _FAILED	16#3217	Authentication failed: Bad username, password OR client id.
MQTT_ERR_CONN_TLS_ HANDSHAKE_FAILED	16#3230	Error on TLS handshake.
MQTT_ERR_CONN_SERV ER_CERT_NOT_VALID	16#3231	Server certificate not valid. Check if PLC date has been set correctly.
MQTT_ERR_CONN_SERV ER_CERT_NOT_PEM	16#3232	Server certificate format is not formatted as PEM.
MQTT_ERR_CONN_SERV ER_CERT_EXPIRED	16#3233	Server certificate has expired.
MQTT_ERR_CONN_CLIE NT_CERT_NOT_VALID	16#3234	Client certificate not valid. Check if PLC date has been set correctly.
MQTT_ERR_CONN_CLIE NT_CERT_NOT_PEM	16#3235	Client certificate or client key format is not formatted as PEM.
MQTT_ERR_CONN_CLIE NT_CERT_EXPIRED	16#3236	Client certificate has expired.
MQTT_ERR_INPUT_02_0	16#4020	Function block Input 02 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Parameter Conn of function block was not set. ● MqttConnectWithCertFile (FB): Parameter Conn of function block was not set. ● MqttGetReceivedPacket (FB): Parameter Conn of function block was not set. ● MqttPublish (FB): Parameter Conn of function block was not set. ● MqttSubscribe (FB): Parameter Conn of function block was not set. ● MqttUnsubscribe (FB): Parameter Conn of function block was not set. MqttPing (FB): Parameter Conn of function block was not set.

Parameter	Value	Description
MQTT_ERR_INPUT_03_0	16#4030	Function block Input 03 error (error case 0), specific error depends on used function block: ● MqttGetReceivedPacket (FB): Pointer payload not initialized. ● MqttPublish (FB): Publish topic name must not contain wildcard characters (+ or #). ● MqttSubscribe (FB): Topic is missing. ● MqttUnsubscribe (FB): Topic is missing.
MQTT_ERR_INPUT_03_1	16#4031	Function block Input 03 error (error case 1), specific error depends on used function block: ● MqttPublish (FB): Payload is not set in MQTT_MESSAGE.
MQTT_ERR_INPUT_04_0	16#4040	Function block Input 04 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Check if Port number has been set correctly (0 is not accepted). ● MqttConnectWithCertFile (FB): Check if Port number has been set correctly (0 is not accepted).
MQTT_ERR_INPUT_06_0	16#4060	Function block Input 06 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Server certificate file was not found.
MQTT_ERR_INPUT_07_0	16#4070	Function block Input 07 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Client certificate file was not found.
MQTT_ERR_INPUT_08_0	16#4080	Function block Input 08 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertFile (FB): Client key file was not found.
MQTT_ERR_INPUT_12_0	16#4120	Function block Input 12 error (error case 0), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Couldn't initialize Last Will message because the topic is not set. ● MqttConnectWithCertFile (FB): Couldn't initialize Last Will message because the payload is not set.

Parameter	Value	Description
MQTT_ERR_INPUT_12_1	16#4121	Function block Input 12 error (error case 1), specific error depends on used function block: ● MqttConnectWithCertBuffer (FB): Couldn't initialize Last Will message because the topic is not set. ● MqttConnectWithCertFile (FB): Couldn't initialize Last Will message because the payload is not set.
MQTT_ERR_FATAL_ERROR	16#5FFFF	Fatal error state machine.

MQTT_QOS (Enum)

Parameter	Value	Description
QOS_0	-	Fire and forget (At most once delivered).
QOS_1	-	Simple acknowledgement (At least once delivered).
QOS_2	-	Complex acknowledgement (Exactly once delivered).

MQTT_MESSAGE

This structure is used for messages which can be published or used for LastWill on MqttConnect(FB).

Variable name	Data type	Default value	Description
sTopic	STRING(MQTT_MAX_TOPIC_LEN)	Empty string	Topic where this message belongs to.
pbyPayload	POINTER TO BYTE	0	Payload which should be sent.
dwLen	DWORD	0	Length of the payload.
eQos	MQTT_QOS	QOS_0	Quality of Service level.
xRetainFlag	BOOL	FALSE	True = message must be stored by the server, False = server must not store this message.

MQTT_CONNECTION

Internal data required by the library to operate. This structure allocates memory and it is used to identify the MQTT connection you want to work with

Parameter	Data type	Range
abyConn	Array	MQTT_CLIENT_STRUCT_SIZE
abyTxBuf	Array	MQTT_TX_BUF_SIZE
abyRxBuf	Array	MQTT_RX_BUF_SIZE
abyMsgBuf	Array	MQTT_MSG_BUF_SIZE

1.5.4.24.3 Global variables

MQTT_CONSTANTS

Parameter	Datatype	Value	Description
MQTT_MAX_IP_ADDRESS_LEN	Word	15	Maximum length of the IP address.
MQTT_MAX_PEM_KEY_LEN	Word	2048	Maximum length of the PEM key.
MQTT_MAX_PEM_CERT_LEN	Word	3072	Maximum length of the PEM certificate.
MQTT_MAX_FILE_PATH_LEN	Word	255	Maximum length of the file path to the certificate files.
MQTT_MAX_CLIENT_ID_LEN	Word	250	Maximum length of the client id.
MQTT_MAX_USERNAME_LEN	Word	250	Maximum length of the username.
MQTT_MAX_PASSWORD_LEN	Word	250	Maximum length of the password.
MQTT_MAX_TOPIC_LEN	Word	255	Maximum length of the topic.
MQTT_CLIENT_STRUCT_SIZE	Word	336	Size of the internal connection structure representing the connection state.
MQTT_TX_BUF_SIZE	Word	1024	Size of the internally used output buffer.
MQTT_RX_BUF_SIZE	Word	1024	Size of the internally used input buffer.
MQTT_MSG_BUF_SIZE	Word	2148	Size of the internally used message buffer.

1.5.4.25 Onboard IO library

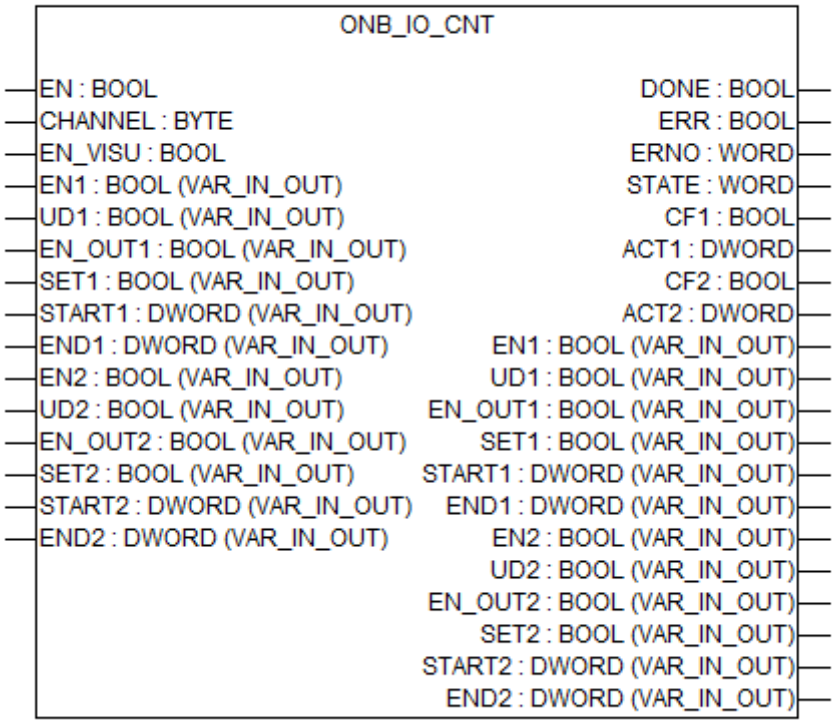
Library file name: **OnBoardIO_AC500_Vx.lib**

The Onboard I/O Library contains all function blocks necessary for using the Onboard I/Os of AC500 CPUs PM55x and PM56x.

The library is automatically included if a project has an processor module PM55x/PM56x.

1.5.4.25.1 Function blocks

ONB_IO_CNT

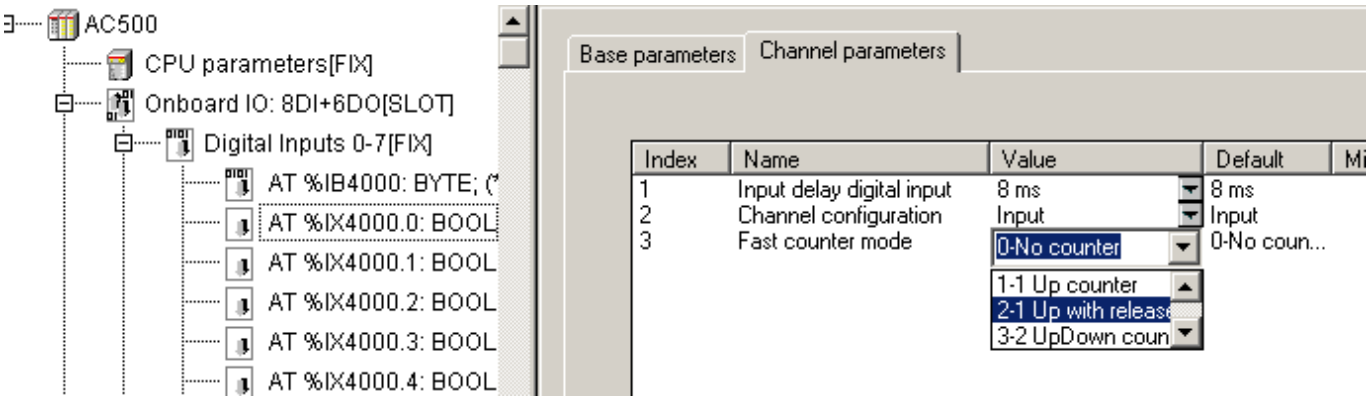


Parameter	Value
Included in library	OnBoardIO_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Counter_OBIO

The function block ONB_IO_CNT is used to control the fast counters which are integrated in AC500 CPUs PM55x and PM56x.

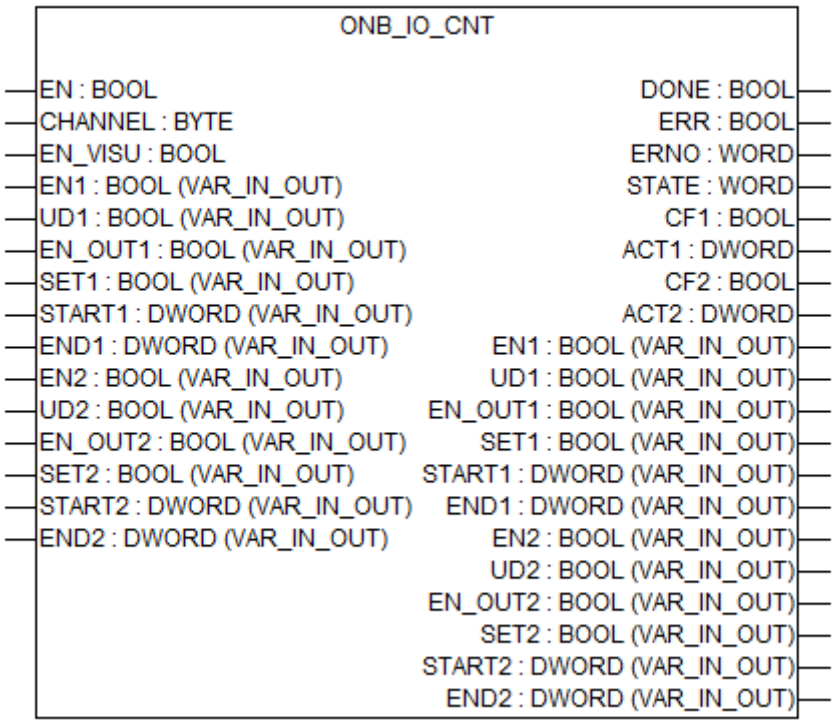
The operating modes of the fast counters are described in detail in *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

To activate the fast counters of AC500 CPUs PM55x and PM56x, the parameter "Fast Counter" of the I/O module must be set to the desired counting mode in the control system configuration.



The function block ONB_IO_CNT has an integrated visualization block that can be used to control all block functions in parallel to the user program (if input EN_VISU = TRUE). A detailed functional description of the visualization and how to integrate it can be found at the end of this Function Block description.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CHANNEL

Data type	Default value	Range	Unit
BYTE	-	-	-

Input CHANNEL is used to select the input for the counter. Currently the only valid value is 0. The device occupies the inputs I0...I1. If an invalid value is entered at input CHANNEL or if the selected channel is not configured as a counter, the function block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

CHANNEL contains the channel number managed by the function block:

- CHANNEL = 0: the output O2 is managed
- CHANNEL = 1: the output O3 is managed

EN_VISU

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN_VISU = TRUE, it is also possible to control the function block inputs (except CHANNEL and EN_VISU) via the integrated visualization of the function block. If input EN_VISU = FALSE, control via the visualization is disabled and the labelling of the corresponding control elements is displayed in gray. The actual values are always displayed.

EN1

Data type	Default value	Range	Unit
BOOL	-	-	-

If input EN1 = TRUE, pulse counting of counter 1 is enabled. If EN1 = FALSE, no pulse counting is performed and the pulses are lost.

Input EN1 corresponds to bit 1 in control byte 0.

UD1

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD1, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD1 = FALSE → counter 1 counts up

UD1 = TRUE → counter 1 counts down

If input SET1 = TRUE, the counter takes this value.

Input UD1 corresponds to bit 0 in control byte 0.

EN_OUT1

Data type	Default value	Range	Unit
BOOL	-	-	-

Input EN_OUT1 is used to enable/disable the output control for the operating modes with direct output activation (modes 1 and 2).

The functionality of the input variable is not supported before Onboard I/O firmware V1.04.

Only for fast counter operating modes 1 and 2:

If EN_OUT1 = FALSE, the related digital output DO acts as an output indicating "end value reached" of the fast counter.

If EN_OUT1 = TRUE, the related digital output DO can be used as normal digital output.

The value change of EN_OUT1 is only effective when EN = TRUE.

Input EN_OUT1 corresponds to bit 3 in control byte 0.

SET1

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET1 = TRUE, the counter takes the values from the inputs UD1, START1 and END1.

As long as input SET1 = TRUE, no pulses are counted because the counter is always over-written by the start value START1.

Input SET1 corresponds to bit 2 in control byte 0.

START1

Data type	Default value	Range	Unit
DWORD			

At input START1, the start value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

END1

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END1, the end value of counter 1 is entered.

If input SET1 = TRUE, counter 1 takes this value.

UD2

Data type	Default value	Range	Unit
BOOL	-	-	-

At input UD2, the counting direction is set for operating modes with up/down counters (modes 3...6).

The following applies:

UD2 = FALSE → counter 2 counts up

UD2 = TRUE → counter 2 counts down

If input SET2 = TRUE, the counter takes this value.

Input UD2 corresponds to bit 0 in control byte 1.

EN_OUT2

Data type	Default value	Range	Unit
BOOL	-	-	-

The functionality of the input variable is not supported before Onboard I/O firmware V1.04.

Input EN_OUT2 is reserved. The value FALSE has to be applied.

Input EN_OUT2 corresponds to bit 3 in control byte 1.

SET2

Data type	Default value	Range	Unit
BOOL	-	-	-

If set input SET2 = TRUE, the counter takes the values from the inputs UD2, START2 and END2.

As long as input SET2 = TRUE, no pulses are counted because the counter is always overwritten by the start value START2.

Input SET2 corresponds to bit 2 in control byte 1.

START2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input START2, the start value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

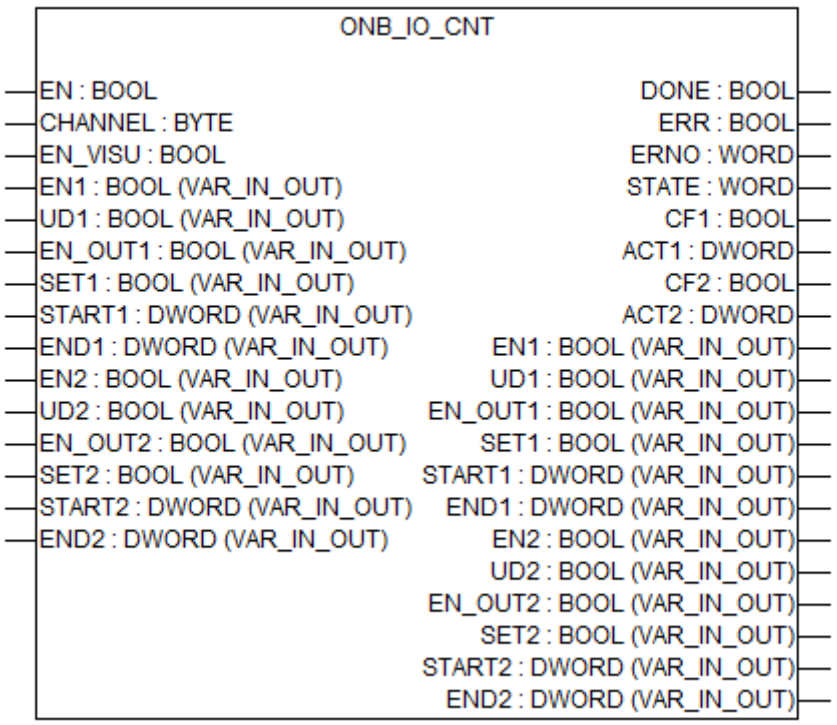
END2

Data type	Default value	Range	Unit
DWORD	-	-	-

At input END2, the end value of counter 2 is entered.

If input SET2 = TRUE, counter 2 takes this value.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE

Data type	Default value	Range	Unit
WORD	-	-	-

Output STATE indicates the current mode of fast counter. The mode of fast counter (0-10) can be configured in the channel parameter of the PLC configuration (please refer to fast counter mode in system technology chapter 9.3).

CF1

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 1 has reached the programmed end value (input END1), output CF1 (end value reached) is set to TRUE and stored. When setting the counter (via input SET1), CF1 is set to FALSE.

Output CF1 corresponds to bit 0 in status byte 0.

ACT1

Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT1, the actual value = counter reading of counter 1 is output as double word.

CF2

Data type	Default value	Range	Unit
BOOL	-	-	-

If counter 2 has reached the programmed end value (input END2), output CF2 (end value reached) is set to TRUE and stored. When setting the counter (via input SET2), CF2 is set to FALSE.

Output CF2 corresponds to bit 0 in status byte 1.

ACT2

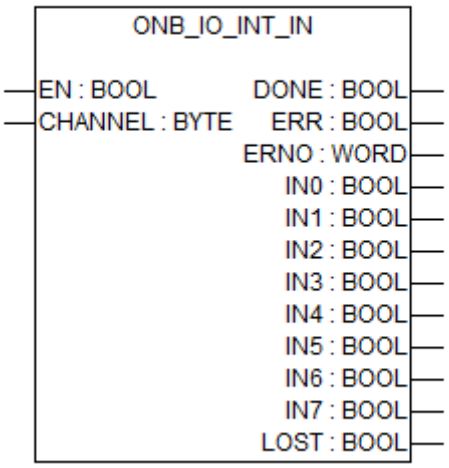
Data type	Default value	Range	Unit
DWORD	-	-	-

At output ACT2, the actual value = counter reading of counter 2 is output as double word.

Function call in ST

```
ONB_IO_CNT   (EN      := ONB_IO_CNT_EN,
CHANNEL := ONB_IO_CNT_CHANNEL,
EN_VISU := ONB_IO_CNT_EN_VISU,
EN1     := ONB_IO_CNT_EN1,
UD1     := ONB_IO_CNT_UD1,
EN_OUT1 := ONB_IO_CNT_EN_OUT1,
SET1    := ONB_IO_CNT_SET1,
START1  := ONB_IO_CNT_START1,
END1    := ONB_IO_CNT_END1,
EN2     := ONB_IO_CNT_EN2,
UD2     := ONB_IO_CNT_UD2,
EN_OUT2 := ONB_IO_CNT_EN_OUT2,
SET2    := ONB_IO_CNT_SET2,
START2  := ONB_IO_CNT_START2,
END2    := ONB_IO_CNT_END2);


ONB_IO_CNT_DONE     := ONB_IO_CNT.DONE;
ONB_IO_CNT_ERR      := ONB_IO_CNT.ERR;
ONB_IO_CNT_ERNO     := ONB_IO_CNT.ERNO;
ONB_IO_CNT_STATE    := ONB_IO_CNT.STATE;
ONB_IO_CNT_CF1      := ONB_IO_CNT.CF1;
ONB_IO_CNT_ACT1     := ONB_IO_CNT.ACT1;
ONB_IO_CNT_CF2      := ONB_IO_CNT.CF2;
ONB_IO_CNT_ACT2     := ONB_IO_CNT.ACT2;
```

ONB_IO_INT_IN

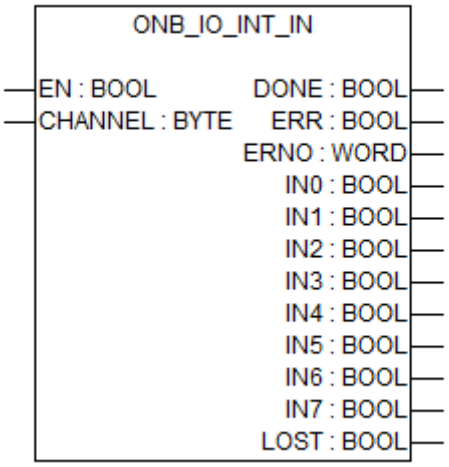


Parameter	Value
Included in library	OnBoardIO_AC500_V13.lib
Available as of firmware	V1.3.0

Parameter	Value
Type	Function block with historical values
Group	Interrupt_Input_OBIO

Using the function block ONB_IO_INT_IN, the interrupt program can be polled which inputs (IN0...IN3) triggered interrupts since the last function block calling. The corresponding outputs IN0...IN3 are set to TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CHANNEL

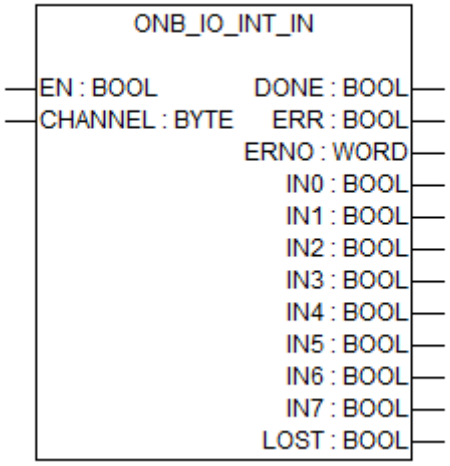
Data type	Default value	Range	Unit
BYTE	-	-	-

Input CHANNEL is used to select the input for the counter. Currently the only valid value is 0. The device occupies the inputs I0...I1. If an invalid value is entered at input CHANNEL or if the selected channel is not configured as a counter, the function block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

CHANNEL contains the channel number managed by the function block:

- CHANNEL = 0: the output O2 is managed
- CHANNEL = 1: the output O3 is managed

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

IN0...IN7

Data type	Default value	Range	Unit
BOOL	-	-	-

The outputs IN0...IN7 display which inputs triggered an interrupt since the last function block calling. 4 channels (I0..I3) of AC500-eCo Onboard I/O can be configured as interrupt inputs. Therefore, IN0…IN3 will be used to display the status of interrupt inputs.

LOST

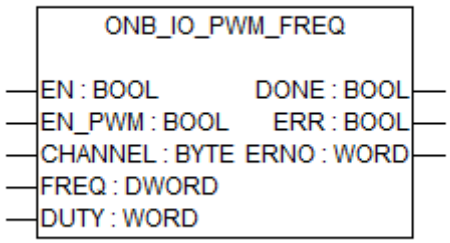
Data type	Default value	Range	Unit
BOOL	-	-	-

If interrupt pulse was lost, LOST=TRUE. Otherwise, LOST=FALSE.

Function call in ST

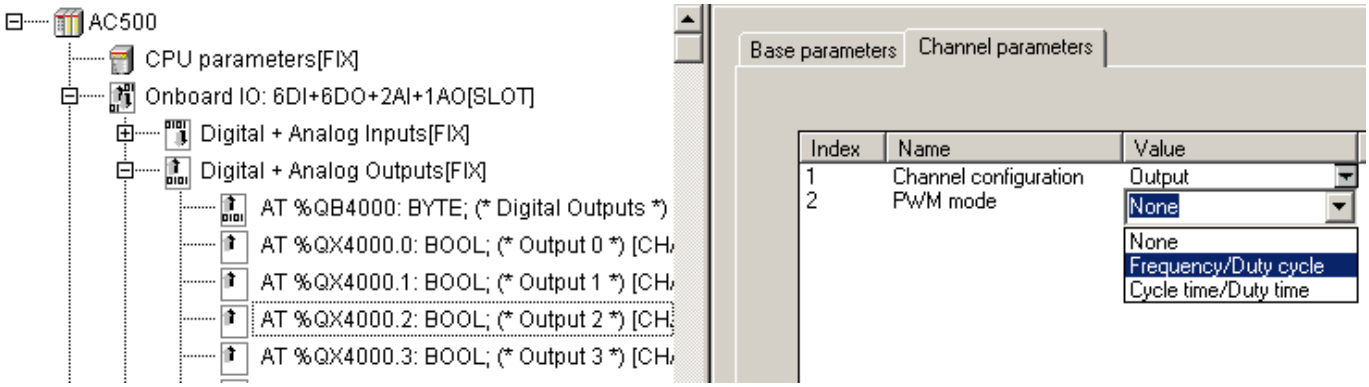
```
ONB_IO_INT_IN    (EN      := ONB_IO_INT_IN_EN,
    CHANNEL := ONB_IO_INT_IN_CHANNEL);


ONB_IO_INT_IN_DONE     := ONB_IO_INT_IN.DONE;
ONB_IO_INT_IN_ERR      := ONB_IO_INT_IN.ERR;
ONB_IO_INT_IN_ERNO     := ONB_IO_INT_IN.ERNO;
ONB_IO_INT_IN_IN0      := ONB_IO_INT_IN.IN0;
ONB_IO_INT_IN_IN1      := ONB_IO_INT_IN.IN1;
ONB_IO_INT_IN_IN2      := ONB_IO_INT_IN.IN2;
ONB_IO_INT_IN_IN3      := ONB_IO_INT_IN.IN3;
ONB_IO_INT_IN_IN4      := ONB_IO_INT_IN.IN4;
ONB_IO_INT_IN_IN5      := ONB_IO_INT_IN.IN5;
ONB_IO_INT_IN_IN6      := ONB_IO_INT_IN.IN6;
ONB_IO_INT_IN_IN7      := ONB_IO_INT_IN.IN7;
ONB_IO_INT_IN_LOST     := ONB_IO_INT_IN.LOST;
```

ONB_IO_PWM_FREQ



Parameter	Value
Included in library	OnBoardIO_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	PWM_OBIO

The onboard digital outputs O2…O3 can be used to control output pulses signals with an adjustable duty cycle (ON/OFF ratio).The PWM operating mode is configured in PLC Configuration using channel parameters (see figure below).



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically when target is set as CPU PM55x or PM56x.

The CPU PM55x or PM56x provides 2 outputs which can be used in PWM mode (O2 and O3). Both have the same specification and work with the same frequency. The duty time can be adjusted independently.

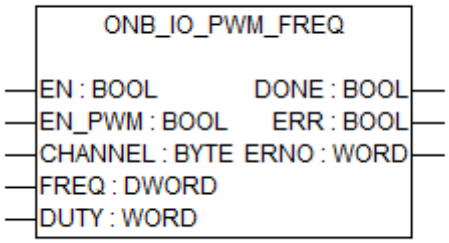
The function block ONB_IO_PWM_FREQ should be used to control with input EN_PWM, configure the frequency with input FREQ and the input duty cycle DUTY of PWM outputs (pulse-width modulator).

The PWM function can be realized by using two function blocks:

- ONB_IO_PWM_FREQ (this function block)
- ONB_IO_PWM_TIME *Chapter 1.5.4.25.1.4 “ONB_IO_PWM_TIME” on page 1746*

> *The frequency for both PWM channels must be the same. If both channels are used, the frequency of the first PWM channel (CHANNEL=0) is valid for both channels. The frequency input of the second channel (CHANNEL=1) must be set to 0.*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

EN_PWM

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_PWM = TRUE, the pulse-width modulator is enabled. If EN_PWM = FALSE, no pulse-width modulation is performed.

Input EN_PWM corresponds to output bit 7 in "control byte PWM x ".

CHANNEL

Data type	Default value	Range	Unit
BYTE	-	-	-

Input CHANNEL is used to select the input for the counter. Currently the only valid value is 0. The device occupies the inputs I0...I1. If an invalid value is entered at input CHANNEL or if the selected channel is not configured as a counter, the function block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

CHANNEL contains the channel number managed by the function block:

- CHANNEL = 0: the output O2 is managed
- CHANNEL = 1: the output O3 is managed

FREQ

Data type	Default value	Range	Unit
DWORD	-	125 - 20000	Hz

The input FREQ is used to specify the frequency of PWM.

The frequency value is defined with a double word (DWORD), but the internal communication is realized with WORD type. For frequencies greater than 65535, an additional bit is used internally as a multiplier. Thus, for such frequencies the resolution becomes 10 Hz.

The additional multiplier bit is the bit 0..1 of "control byte PWM x ".

Input FREQ corresponds to output word "PWM x -Frequency/Cycle time".

DUTY

Data type	Default value	Range	Unit
WORD	-	0.0 - 100.0	%

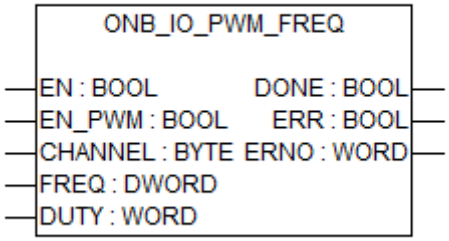
The input DUTY is used to specify the percentage of time set to TRUE. The duty cycle is from 0 to 1000 (0.0 % to 100.0 %); the value is written without dot. That means, for example, for 75.8 % the value written will be 758. The max. value authorized will be 1000 to specify a duty cycle = 100.0 %.

On fast outputs O2 and O3, the brightness of the yellow LED depends on the value of duty cycle specified (from 0 to 100 %).

If the written value is greater than 1000, the value 1000 will be read. If the written value is less than 0, the value 0 will be read. No error message will be generated in these two cases. The last entered valid value will be used or the value '0' if no value has been entered before.

Input DUTY corresponds to output word "PWM x - Duty cycle/Duty time".

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

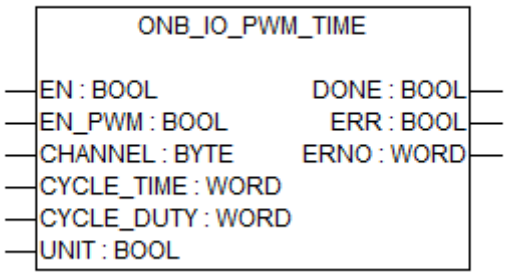
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

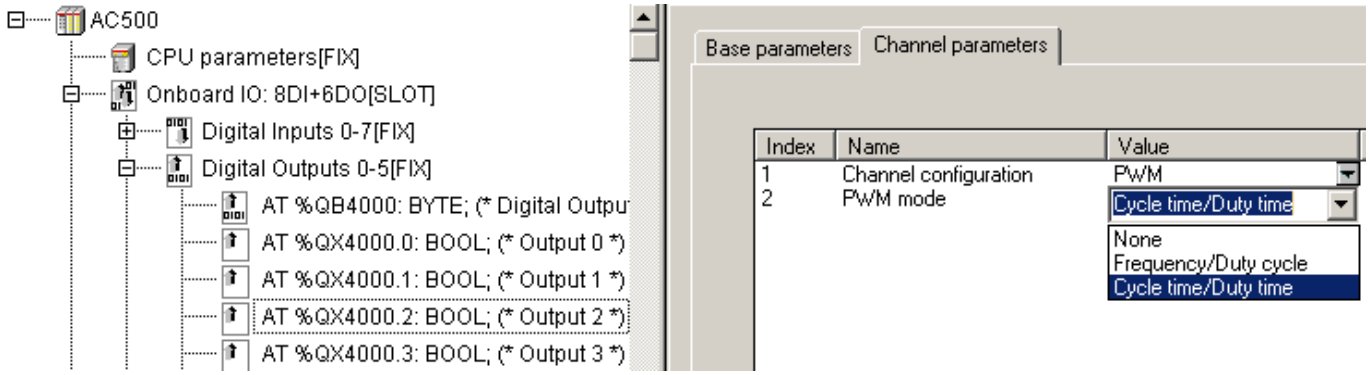
```
ONB_IO_PWM_FREQ    (EN       := ONB_IO_PWM_FREQ_EN,
EN_PWM  := ONB_IO_PWM_EN_PWM,
CHANNEL := ONB_IO_PWM_FREQ_CHANNEL,
FREQ    := ONB_IO_PWM_FREQ_FREQ,
DUTY    := ONB_IO_PWM_FREQ_DUTY);


ONB_IO_PWM_FREQ_DONE      := ONB_IO_PWM_FREQ.DONE;
ONB_IO_PWM_FREQ_ERR       := ONB_IO_PWM_FREQ.ERR;
ONB_IO_PWM_FREQ_ERNO      := ONB_IO_PWM_FREQ.ERNO;
```

ONB_IO_PWM_TIME



Parameter	Value
Included in library	OnBoardIO_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	PWM_OBIO

The onboard digital outputs O2…O3 can be used to control output pulses signals with an adjustable duty cycle (ON/OFF ratio).The PWM operating mode is configured in PLC Configuration using channel parameters (see figure below).



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically when target is set as CPU PM55x or PM56x.

The CPU PM55x or PM56x provides two outputs which can be used in PWM mode (O2 and O3). Both have the same specification and work with the same frequency. The duty time can be adjusted independently.

The function block ONB_IO_PWM_TIME should be used to control with input EN_PWM, configure the frequency with input CYCLE_TIME and the input CYCLE_DUTY of PWM outputs (pulse-width modulator).

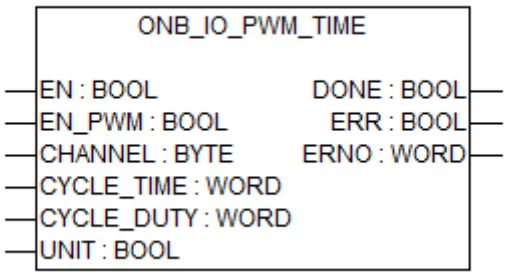
The PWM function can be realized by using two function blocks:

- ONB_IO_PWM_FREQ ↳ *Chapter 1.5.4.25.1.3 “ONB_IO_PWM_FREQ” on page 1743*
- ONB_IO_PWM_TIME (this function block)

For PWM functionality two channels share the same frequency. With this by using the second channel, frequency on its function block won't work. To resolve this, enable the function block of the first channel and set the frequency.

The cycle time for both PWM channels must be the same. If both channels are used, the cycle time of the first PWM channel (CHANNEL=0) is valid for both channels. The cycle time input of the second channel (CHANNEL=1) must be set to 0.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

EN_PWM

Data type	Default value	Range	Unit
BOOL	-	-	-

If EN_PWM = TRUE, the pulse-width modulator is enabled. If EN_PWM = FALSE, no pulse-width modulation is performed.

Input EN_PWM corresponds to output bit 7 in "control byte PWM x ".

CHANNEL

Data type	Default value	Range	Unit
BYTE	-	-	-

Input CHANNEL is used to select the input for the counter. Currently the only valid value is 0. The device occupies the inputs I0...I1. If an invalid value is entered at input CHANNEL or if the selected channel is not configured as a counter, the function block is aborted with DONE=ERR=TRUE and a corresponding error number at ERNO.

CHANNEL contains the channel number managed by the function block:

- CHANNEL = 0: the output O2 is managed
- CHANNEL = 1: the output O3 is managed

CYCLE_TIME

Data type	Default value	Range	Unit
WORD	-	50 µs (20 kHz) - 8000 µs (125 Hz)	µs (Hz)

The input CYCLE_TIME is used to specify the cycle time of PWM. The unit can be also set as "ms" by configuring the input variable UNIT=TRUE.

Input CYCLE_TIME corresponds to output word "PWM x -Frequency/Cycle time".

CYCLE_DUTY

Data type	Default value	Range	Unit
WORD	-	-	-

Input CYCLE_DUTY is used to preset the duty time for the output. The input CYCLE_DUTY is used to specify the time of TRUE signal. By normal operation mode, the value of CYCLE_DUTY should be smaller or equal to the value of CYCLE_TIME.

On fast outputs O2 and O3 the brightness of the yellow LED depends on the value of duty time specified.

The last entered valid value will be used or the value '0' if no value has been entered before.

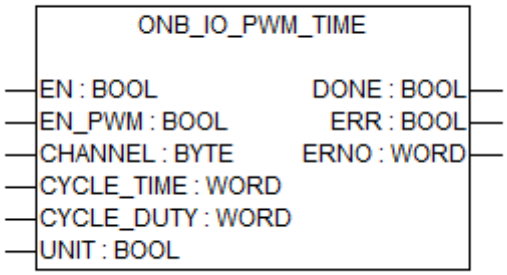
Input CYCLE_DUTY corresponds to output word "PWM x - Duty cycle/Duty time".

UNIT

Data type	Default value	Range	Unit
BOOL	-	-	-

Input UNIT is used to preset the timebase for the PWM output. FALSE sets the timebase to µs and TRUE to ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
ONB_IO_PWM_TIME (EN          := ONB_IO_PWM_TIME_EN,
EN_PWM      := ONB_IO_PWM_TIME_EN_PWM,
CHANNEL     := ONB_IO_PWM_TIME_CHANNEL,
CYCLE_TIME  := ONB_IO_PWM_TIME_CYCLE_TIME,
CYCLE_DUTY  := ONB_IO_PWM_TIME_CYCLE_DUTY,
UNIT        := ONB_IO_PWM_TIME_UNIT);


ONB_IO_PWM_TIME_DONE      := ONB_IO_PWM_TIME.DONE;
ONB_IO_PWM_TIME_ERR       := ONB_IO_PWM_TIME.ERR;
ONB_IO_PWM_TIME_ERNO      := ONB_IO_PWM_TIME.ERNO;
```

1.5.4.26 PROFIBUS DP library

Library file name: **PROFIBUS_AC500_Vx.lib**

The PROFIBUS DP Communication Module in the AC500 controller series can solely be operated in operating mode DP master.

The function blocks provided by PROFIBUS DP library can be used to work with Communication Module CM592-DP. Depending on the functionality runtime system or directly the Communication Module is accessed. Required definitions are provided by the libraries *SysExt_AC500_V10.lib*, *PROFIBUS_CME_AC500_V25.lib*, *PROFIBUS_CMN_AC500_V25.lib* and *CMN_AC500_V24.lib*. The library is automatically included into the user program due to Communication Module configuration.

Basically this library replaces the library *PROFIBUS_AC500_V10.lib* and provides function block types with identical type names and input and output layouts. The function blocks act in compatible behavior for both Communication Module types. Thus description of the functionality is valid regardless of the Communication Module type. Already existing user programs can be reused without changes. However, CM592-DP uses different PROFIBUS protocol stack implementations which acts different in a few details. If such differences are to be considered in using the function blocks, this is described at the certain Function Block descriptions.

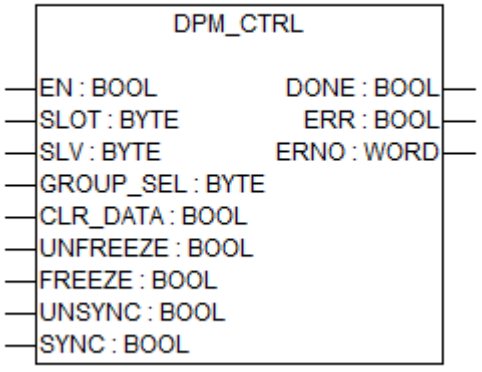
Neither the function blocks of the PROFIBUS DP library nor the PROFIBUS DP communication can be run in simulation mode. The PROFIBUS DP communication only runs in the PLC mode RUN, but not in the modes Single Cycle, Step and Breakpoint.

It is not necessary to include the library for normal cyclic data exchange via PROFIBUS DP. The library contains additional function blocks which allow an easy handling of the PROFIBUS DP Communication Module. Additionally, various data types are defined in this library. These structures enable a clear presentation of data sets.

If you update an Automation Builder project from Automation Builder Version 1.2 to a Automation Builder Version ≥ 1.2, delete the PROFIBUS_AC500_V10.lib from your CODESYS V2.3 project in advance.

1.5.4.26.1 Function blocks

DPM_CTRL



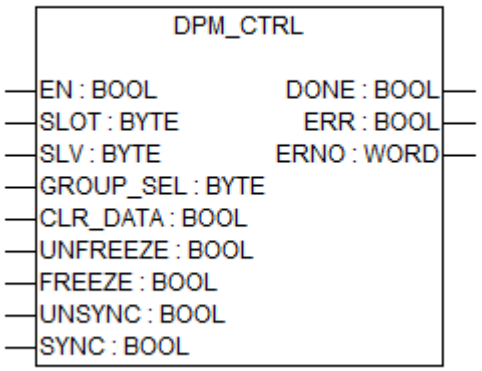
Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Control

The function block DPM_CTRL implements thePROFIBUS DP function DDLM_Global_Control. DDLM_Global_Control is a broadcast function.

Using global control commands, the output data of one, several or all slaves can be reset and input or output data of slaves can be synchronized. The commands are selected by different combinations of the outputs CLR_DATA, FREEZE / UNFREEZE and SYNC / UNSYNC. The called slaves are selected using three parameters. First, during project planning, the slaves can be divided into logical groups. Then, during run time, the slaves can be called individually or in groups via the function block inputs SLV and GROUP_SLV.

Every time a FALSE → TRUE edge is applied to input EN, DPM_CTRL reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV

Data type	Default value	Range	Unit
BYTE	-	-	-

At input SLV the bus address of the DP slave is specified, to which a global control command shall be sent.

With SLV = 0..126, a particular slave with the corresponding bus address is called directly, independent of the group to which it was assigned during configuration and independent of the value applied at function block input GROUP_SEL.

If SLV = 127 and GROUP_SEL = 0, all slaves are called simultaneously. The selection of individual slave groups is done with SLV = 127 and a combination of GROUP_SEL and the group assignment made during configuration. For that reason, the function block outputs SLV and GROUP_SEL always have to be considered together with the group assignment made during configuration.

For possible combinations see *Possible combinations of global control commands*.

GROUP_SEL

Data type: BYTE

At input GROUP_SEL (slave group selection) the slave groups are selected, to which a global control command shall be sent.

With SLV = 0..126, a particular slave with the corresponding bus address is called directly, independent of the group assignment which was made during configuration and independent of the value applied at function block input GROUP_SEL. If GROUP_SEL = 0 and SLV = 127, all slaves are called simultaneously.

The selection of individual slave groups is done with SLV = 127 and a combination of GROUP_SEL and the group assignment made during configuration. For that reason, the function block outputs SLV and GROUP_SEL always have to be considered together with the group assignment made during configuration.

For possible combinations see *Possible combinations of global control commands*.

CLR_DATA

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Using input CLR_DATA (clear data), the output data of slaves can be reset on site.

The called slave resets its outputs on-site to 0 when it receives a global control command with CLR_DATA = TRUE.

If the CLR_DATA part of a command is FALSE, the outputs of the called slaves keep their current state. Slaves which are not called ignore the entire command.

For possible combinations see *Possible combinations of global control commands*.

CLR_DATA is not supported on using function block DPM_CTRL with Communication Module of type CM592-DP.

UNFREEZE Data type: BOOL

When input UNFREEZE is TRUE, the synchronization mode for input data of the called slaves is terminated, regardless of the current state of input FREEZE (TRUE or FALSE).

Then, the called slaves forward their input values directly to the master again. If the UNFREEZE part of a command is FALSE, the called slaves keep their current state. The UNFREEZE command is ignored, if the called slave is not in the FREEZE state. Slaves, which are not called, ignore the entire command.

UNFREEZE always has to be considered together with FREEZE.

For possible combinations see *Possible combinations of global control commands*.

FREEZE Data type: BOOL

When input FREEZE is TRUE and input UNFREEZE is FALSE at the same time, the called slaves change to input data synchronization mode.

This mode is activated with the first FREEZE command. As a result, the called slaves simultaneously freeze the values currently applying at their local inputs and store them temporarily. During the subsequent process data cycles, the temporarily stored input values are transmitted to the master, regardless of possible changes of the input values in the mean time.

When another FREEZE command is received, the temporarily stored input values are updated, i.e. the called slaves simultaneously store the present input values into an internal buffer once again and then transmit these values to the master during the subsequent cycles.

FREEZE always has to be considered together with UNFREEZE.

For possible combinations see *Possible combinations of global control commands*.

UNSYNC Data type: BOOL

If input UNSYNC is TRUE, the synchronization mode for the output data of the called slaves is terminated, regardless of the current state of input SYNC (TRUE or FALSE). From now on, the called slaves immediately forward the output data received from the master to their own outputs again.

If the UNSYNC part of a command is FALSE, the called slaves keep their current state. The UNSYNC command is ignored, if the called slave is not in the SYNC state. Slaves which are not called ignore the entire command.

UNSYNC always has to be considered together with SYNC.

For possible combinations see *Possible combinations of global control commands*.

SYNC Data type: BOOL

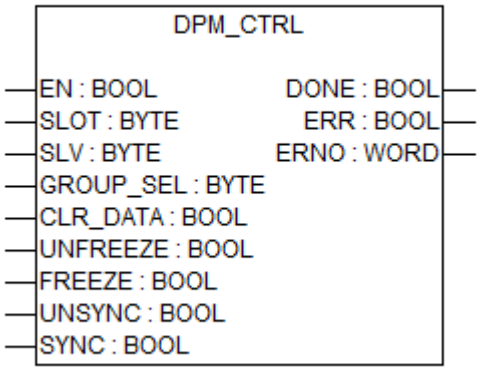
If input SYNC is TRUE and input UNSYNC is FALSE at the same time, the called slaves change to output data synchronization mode.

This mode is activated with the first SYNC command. As a result, the called slaves freeze the current states of their local outputs. The output data sent during the subsequent process data cycles are first stored only locally in these slaves.

On reception of another SYNC command, the slaves then simultaneously apply these temporarily stored values to their outputs. SYNC always has to be considered together with UNSYNC.

For possible combinations see *Possible combinations of global control commands*.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
CTRL
     (EN      := CTRL_EN,
     SLOT  := CTRL_SLOT,
     SLV      := CTRL_SLV,
     GROUP_SEL  := CTRL_GROUP_SEL,
     CLR_DATA      := CTRL_CLR_DATA,
     UNFREEZE      := CTRL_UNFREEZE,
     FREEZE  := CTRL_FREEZE,
     UNSYNC  := CTRL_UNSYNC,
     SYNC  := CTRL_SYNC);
```

```
CTRL_DONE := CTRL.DONE;
CTRL_ERR  := CTRL.ERR;


SYSDIAG_ERNO:= SYSDIAG.ERNO;
```

Selection of the called slaves

Group assignment is part of the PROFIBUS DP slave configuration procedure and will be loaded to the PROFIBUS DP master as part of the slave parameter set. As default a slave is not connected to a specific group. This has to be done manually. The group assignment parameter is coded of type byte and provides a bitfield. Each bit points to a certain group and several bits can be set in parallel. Thus a slave can be assigned to multiple groups at the same time which is limited to 8 groups in total.

G8	G7	G6	G5	G4	G3	G2	G1
7	6	5	4	3	2	1	0

For instance, if a slave is to be assigned to the groups 7 and 1, the master sends a byte with the decimal value 65 (= binary value 01000001) to this slave.

The function block inputs SLV and GROUP_SEL specify which slaves are to be called by a global control command during run time.

SLV = 0..126 calls only the slave with a bus address = SLV, independent of the group assignment and of the value applied at input GROUP_SEL. All other slaves discard this telegram. Applying SLV = 127 and GROUP_SEL = 0 calls all slaves connected to the bus, independent of their group assignment. The group assignment defined during the configuration is only considered by the slaves if they receive a global control command with SLV = 127 and GROUP_SEL unequal to 0. In this case, each slave compares the GROUP_SEL value with the group assignment byte received during parameterization. The slave accepts the command if a bitwise collation of these two values delivers a result unequal to 0 and discards the command if the collation delivers a value of 0. For instance, if a slave which is assigned to groups 1 and 7 (see above) receives a global control command with SLV = 127 and GROUP_SEL = 64 dec. (= 01000000 bin.), the command is also (among others) addressed to this slave.

Group assignment		0	1	0	0	0	0	0	1
GROUP_SEL	AND	0	1	0	0	0	0	0	0
Result of the comparison		0	1	0	0	0	0	0	0

The following table shows the possible combinations of the three parameters SLV, GROUP_SEL and group assignment and lists the slaves called with these combinations.

SLV	GROUP_SEL	Group assignment	Called slaves
0...126	X	X	Only slave with bus address = SLV
127	0	X	All slaves
127	1 - 255	1 - 255	Slaves, for which a bitwise collation of group assignment and GROUP_SEL delivers a value unequal to 0.

Group assignment does not have impact on cyclic data exchange.

Possible combinations of global control commands

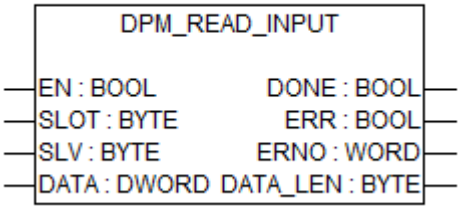
During the process data exchange, the master cyclically transmits the output data to the corresponding slave, which applies these data immediately to its outputs. In return, a slave transmits the values currently applied at its inputs to the master. During this process, the slaves are called one after the other within one bus cycle of the master. As a result, a small time difference appears between the points of time at which the individual slaves apply the received data at their local outputs. In the same way, the points of time differ at which the acquisition of values at the slave inputs and their transmission to the master takes place. A time-synchronization of the inputs or outputs is achieved with the help of global commands. While a CLR_DATA causes all called slaves to set their outputs to 0 once and at the same time, the combinations of SYNC / UNSYNC or FREEZE / UNFREEZE must be considered together.

The following table shows the possible combinations within a global control command and explains their effects.

CLR_ DATA	SYNC	UN-SYNC	FREEZE	UN-FREEZE	Effect
1	X	X	X	X	All called slaves set their outputs to 0
X	0	0	X	X	No effect to SYNC mode
X	0	1	X	X	SYNC mode for output data is terminated
X	1	0	X	X	SYNC mode; the output data received last are applied to the outputs
X	1	1	X	X	SYNC mode for output data is terminated
X	X	X	0	0	No effect to FREEZE mode
X	X	X	0	1	FREEZE mode for input data is terminated
X	X	X	1	0	
X	X	X	1	1	

CLR_DATA is not supported on using function block DPM_CTRL with Communication Module of type CM592-DP.

DPM_READ_INPUT

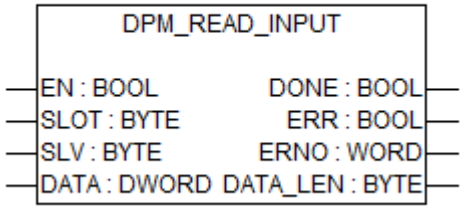


Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Data

The function block DPM_READ_INPUT implements the acyclic PROFIBUS DP function DDLM_RD_Inp which can be used to read input data of a slave. Using this function the master can read also input data of slaves which are assigned to other masters. DPM_READ_INPUT works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPM_READ_INPUT reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is applied, the input data of which shall be read.

DATA (data)

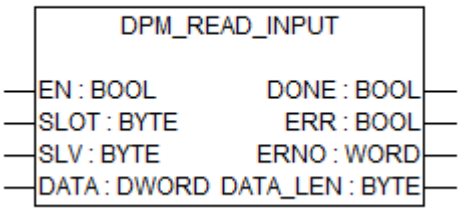
Data type: DWORD

At input DATA the address of the variable to be used to store the received input data is specified via the address operator ADR.

The size of this variable must be big enough to store all input data of the slave (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the slave inputs must be considered.

If the slave has mixed inputs of different types, it is recommended to first define a STRUCT data type which represents an image of the slave's input structure (see I/O configuration of the slave) and then to declare a variable of this type.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DATA_LEN (data length)

Data type: BYTE

Output DATA_LEN displays the length (in bytes) of the input data read by the slave.

DATA_LEN is only valid, if DONE is TRUE and ERR is FALSE. If DATA_LEN contains a value X which is not 0, the function block has stored X bytes of input data in the variable specified at DATA.

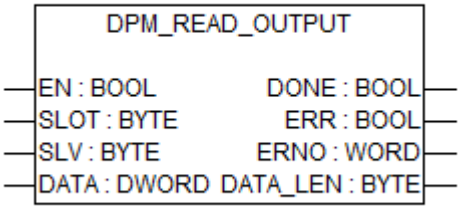
Example

If DATA is a byte array with start index 1, the valid input data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in ST

```
READ_INPUT
     (EN := RAED_INPUT_EN,
     SLOT:= RAED_INPUT_SLOT,
     SLV := RAED_INPUT_SLV,
     DATA:= ADR(RAED_INPUT_DATA));


READ_INPUT_DONE:= READ_INPUT.DONE;
READ_INPUT_ERR := READ_INPUT.ERR;
SYSDIAG_ERNO   := SYSDIAG.ERNO;
READ_INPUT_DATA_LEN:= READ_INPUT.DATA_LEN;
```

DPM_READ_OUTPUT

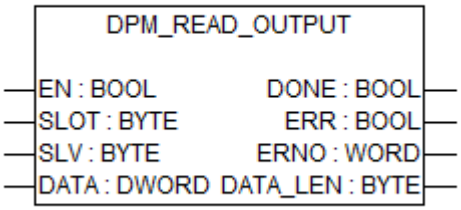


Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Data

The function block DPM_READ_OUTPUT implements the acyclic PROFIBUS DP function DDLM_RD_Outp which can be used to read output data of a slave. Using this function the master can also read output data of slaves which are assigned to other masters. Write access to output data of these slaves is not possible. DPM_READ_OUTPUT works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPM_READ_OUTPUT reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is applied, the output data of which shall be read.

DATA (data)

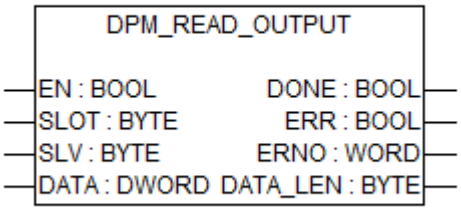
Data type: DWORD

At input DATA the address of the variable to be used to store the received output data is specified via the address operator ADR.

The size of this variable must be big enough to store all output data of the slave (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the slave outputs must be considered.

If the slave has mixed outputs of different types, it is recommended to first define a STRUCT data type which represents an image of the slave's output structure (see I/O configuration of the slave) and then to declare a variable of this type.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DATA_LEN (data length)

Data type: BYTE

Output DATA_LEN displays the length (in bytes) of the output data read by the slave.

DATA_LEN is only valid, if DONE is TRUE and ERR is 0. If DATA_LEN contains a value X which is not 0, the function block has stored X bytes of output data in the variable specified at DATA.

Example

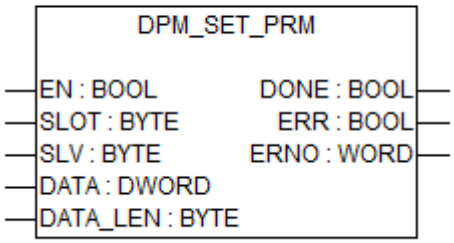
If DATA is a byte array with start index 1, the valid output data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in ST

```
READ_OUTPUT
     (EN   := READ_OUTPUT_EN,
     SLOT  := READ_OUTPUT_SLOT,
     SLV   := READ_OUTPUT_SLV,
     DATA  := ADR(READ_OUTPUT_DATA));


READ_OUTPUT_DONE:= READ_OUTPUT.DONE;
```

```
READ_OUTPUT_ERR := READ_OUTPUT.ERR;
SYSDIAG_ERNO:= SYSDIAG.ERNO;
READ_OUTPUT_DATA_LEN:= READ_OUTPUT.DATA_LEN;
```

DPM_SET_PRM



Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Parameter

The function block DPM_SET_PRM implements the PROFIBUS function DDLM_Set_Prm.

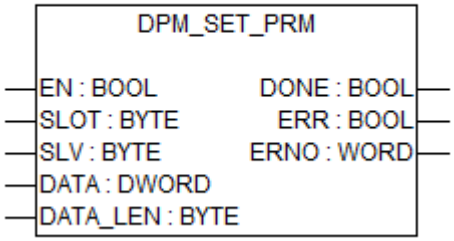
With function block DPM_SET_PRM the user parameter of a slave can be modified during run time. The parameters are immediately sent to the slave. This slave parameter set is used temporarily only. It is not taken into the corresponding slave parameter set stored in master configuration. In case of restarting the PROFIBUS DP communication for any reason the master will configure the slave using the parameter set defined in slave configuration data. The parameter set written to the slave by using DPM_SET_PRM is lost in this case.

Format and length of the user parameters are slave-specific. Due to this, the function block DPM_SET_PRM provides inputs where only the variable address and the length of the user parameters to be sent must be specified. It is in the responsibility of the user that the data comply with the requirements of the corresponding device regarding format and length (e.g. by defining a structure).

Every time a FALSE → TRUE edge is applied to input EN, DPM_SET_PRM reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Due to temporarily validity of salve parameters loaded by DPM_SET_PRM Communication Module of type CM592-DP does not support this function.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is specified, to which the user parameters shall be sent.

Valid addresses are values between 0 and 126.

DATA (data)

Data type: DWORD

At input DATA the address of the variable from which onwards the parameters to be sent are stored, is specified via the address operator ADR.

The data format and the length of the variable must correspond to the structure of the user parameters of the slave. It is recommended to first define a STRUCT data type which represents an image of the slave's structure, and then to declare a variable of this type.

Please note that only the user parameters are to be specified at this point. The standard parameters cannot be modified during run time. The standard parameter settings specified during configuration are automatically completed by the function block.

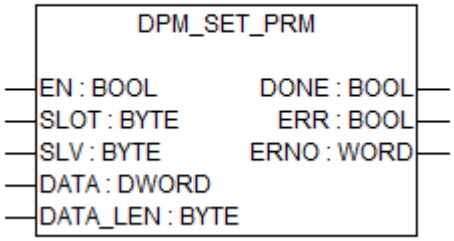
DATA_LEN (data length)

Data type: BYTE

The input DATA_LEN informs the function block about the length (number of bytes) of the user parameters to be transmitted.

The maximum length is 237 bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

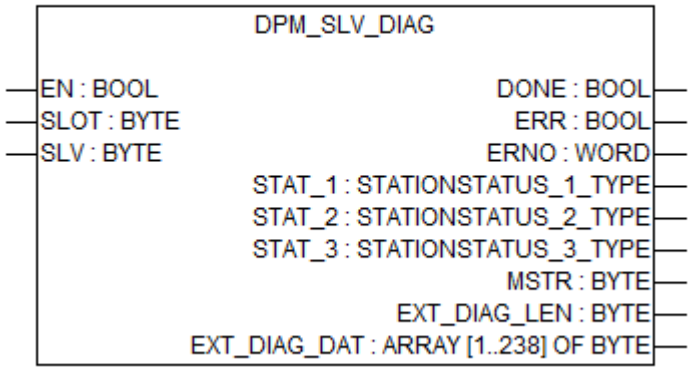
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
SETPRM
      (EN  := SETPRM_EN,
      SLOT := SETPRM_SLOT,
      SLV  := SETPRM_SLV,
      DATA := SETPRM_DATA,
      DATA_LEN := SETPRM_DATA_LEN);


SETPRM_DONE :=  SETPRM.DONE;
SETPRM_ERR  :=  SETPRM.ERR;


SYSDIAG_ERNO:= SYSDIAG.ERNO;
```

DPM_SLV_DIAG



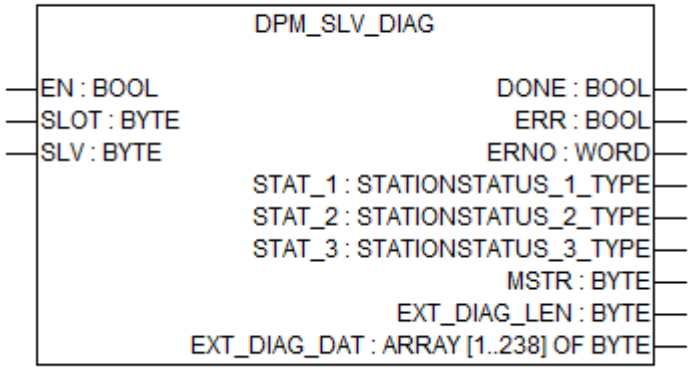
Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Diagnosis

The function block DPM_SLV_DIAG implements the PROFIBUS DP function DDLM_Slave_Diag.

Every time a FALSE → TRUE edge is applied to input EN, DPM_SLV_DIAG reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished.

The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

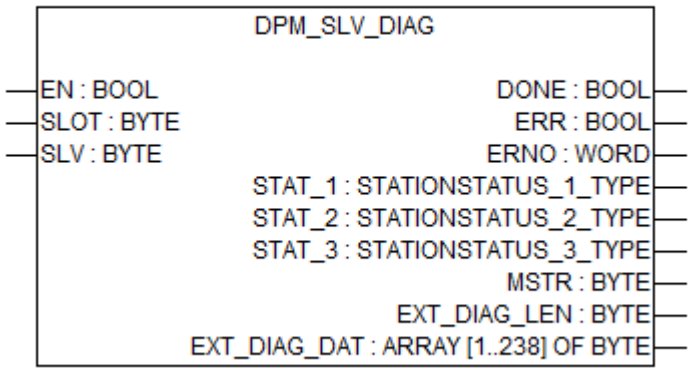
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is specified, for which diagnosis data are requested.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STAT_1 (station status)

Data type: STATIONSTATUS_1_TYPE

STAT_1 outputs the first octet of the DP slave diagnosis data.

STAT_1 is only valid, if DONE = TRUE and ERR = FALSE. The structure of the type STATIONSTATUS_1_TYPE corresponds to the octet Stationstatus_1 defined in the standard as described below.

STAT_2 (station status)

Data type: STATIONSTATUS_2_TYPE

STAT_2 outputs the second octet of the DP slave diagnosis data.

STAT_2 is only valid, if DONE = TRUE and ERR = FALSE. The structure of the type STATIONSTATUS_2_TYPE corresponds to the octet Stationstatus_2 defined in the standard as described below.

STAT_3 (station status)

Data type: STATIONSTATUS_3_TYPE

STAT_3 outputs the third octet of the DP slave diagnosis data.

STAT_3 is only valid, if DONE = TRUE and ERR = FALSE. The structure of the type STATIONSTATUS_3_TYPE corresponds to the octet Stationstatus_3 defined in the standard as described below.

MSTR (master)

Data type: BYTE

MSTR outputs the bus address of the DP master to which has done the configuration of this slave.

MSTR is only valid, if DONE = TRUE and ERR = FALSE. Using the function block DPM_SLV_DIAG, diagnosis data for all DP slaves can be polled which are connected to the bus. If the diagnosis data are requested by a DP slave which was assigned to the PLC during configuration, the PLC bus address is output at MSTR.

In multi-master systems it is possible that diagnosis data are also requested by DP slaves which are assigned to other DP masters. In this case, MSTR outputs the bus address of the DP master to which the requesting DP slave is assigned.

EXT_DIAG_LEN

Data type	Default value	Range	Unit
BYTE	-	-	-

EXT_DIAG_LEN (extended diagnosis data length) outputs the number of valid bytes, following in EXT_DIAG_DAT.

If EXT_DIAG_LEN = 0, no extended diagnosis data are available. Otherwise, EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN] contain the extended diagnosis data reported by the DP slave. EXT_DIAG_LEN is only valid, if DONE = TRUE and ERR = FALSE.

EXT_DIAG_DAT

Data type: ARRAY[1..238] OF BYTE

At output EXT_DIAG_DAT (extended diagnosis data) the extended diagnosis data reported by the DP slave are applied as a byte array.

The data in EXT_DIAG_DAT are only valid, if DONE = TRUE, ERR = FALSE and EXT_DIAG_LEN > 0. The extended diagnosis data are structured according to the standard. Since the meaning of these data strongly depends on the used devices, no automatic interpretation by the function block is possible at this point.

The evaluation of the data must be performed by the user, with the aid of the particular device description and the GSD file respectively. The general structure of the extended diagnosis data (according to the standard) is described in *Structure of DP slave diagnostic data*.

Function call in ST

```
DIAG
     (EN           := DIAG_EN,
     SLOT          := DIAG_SLOT,
     SLV           := DIAG_SLV);


DIAG_DONE          := DIAG.DONE;
DIAG_ERR           := DIAG.ERR;
DIAG_ERNO          := DIAG.ERNO;
DIAG_STAT_1        := DIAG.STAT_1;
DIAG_STAT_2        := DIAG.STAT_2;
DIAG_STAT_3        := DIAG.STAT_3;
DIAG_MSTR          := DIAG.MSTR;
DIAG_EXT_DIAG_LEN := DIAG.EXT_DIAG_LEN;
DIAG_EXT_DIAG_DAT := DIAG.EXT_DIAG_DAT;
```

Structure of DP slave diagnosis data

The function block DPM_SLV_DIAG divides the diagnosis data of a DP slave into sections and applies them to the corresponding outputs. The structure of the DP slave diagnosis data is prescribed by the standard as follows:

Octet 1	Stationstatus_1 ↳ *Chapter 1.5.4.26.1.5.4.1 “Stationstatus_1” on page 1769*	STAT_1
Octet 2	Stationstatus_2 ↳ *Chapter 1.5.4.26.1.5.4.2 “Stationstatus_2” on page 1770*	STAT_2
Octet 3	Stationstatus_3 ↳ *Chapter 1.5.4.26.1.5.4.3 “Stationstatus_3” on page 1771*	STAT_3
Octet 4	Master_Add ↳ *Chapter 1.5.4.26.1.5.4.4 “Master_Add” on page 1771*	MSTR
Octet 5 to 6	Ident_Number ↳ *Chapter 1.5.4.26.1.5.4.5 “Ident_Number” on page 1771*	
Octet 7 to 244	Ext_Diag_Data ↳ *Chapter 1.5.4.26.1.5.4.6 “Ext_Diag_Data” on page 1771*	EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN]

Stationstatus_1

STATION-STATUS_1_TYPE

The diagnostic byte Stationstatus_1 (defined within the standard) is output at STAT_1 of the function block DPM_SLV_DIAG. It is represented as a structure of the data type STATION-STATUS_1_TYPE.

Within the PROFIBUS DP library the structure STATIONSTATUS_1_TYPE is declared as follows:

```
TYPE STATIONSTATUS_1_TYPE:
STRUCT
    NON_EXISTENT:     BOOL;
    NOT_READY:        BOOL;
    CFG_FAULT:        BOOL;
    EXT_DIAG:         BOOL;
    NOT_SUPPORTED:    BOOL;
    INVALID_RESPONSE: BOOL;
    PRM_FAULT:        BOOL;
    MASTER_LOCK:      BOOL;
END_STRUCT
END_TYPE
```

NON_EXISTENT Data type: BOOL

If this bit is set, the PROFIBUS DP Communication Module has not found a DP slave at the bus address applied at function block input SLV.

NOT_READY Data type: BOOL

This bit is set to TRUE by the DP slave if the DP slave is not ready for I/O data exchange.

CFG_FAULT Data type: BOOL

This bit is set to TRUE by the DP slave, if the configuration data (nominal configuration) received from the DP master do not match the data stored in the DP slave (actual configuration).

EXT_DIAG Data type: BOOL

If this bit is TRUE, extended diagnosis data are available in EXT_DIAG_DAT. If this bit is not set (FALSE) and EXT_DIAG_LEN > 0, possibly a status message is available in EXT_DIAG_DAT.

The meaning of such a status message is device-dependent.

NOT_SUPPORTED Data type: BOOL

This bit is set to TRUE by the DP slave, if an unsupported function was requested before.

INVALID_RESPONSE Data type: BOOL

If this bit is TRUE, the PROFIBUS Communication Module has not received a plausible response from the requested DP slave.

PRM_FAULT Data type: BOOL

The DP slave sets this bit to TRUE, if the parameter data received last were faulty.

MASTER_LOCK Data type: BOOL

This bit is set to TRUE, if the DP slave is assigned to another DP master. In this case, MSTR contains the bus address of this DP master.

Stationstatus_2

STATION-STATUS_2_TYPE

The diagnostic byte Stationstatus_2 (defined within the standard) is output at STAT_2 of the function block DPM_SLV_DIAG. It is represented as a structure of the data type STATION-STATUS_2_TYPE.

The structure STATIONSTATUS_2_TYPE is declared as follows in the PROFIBUS library:

```
TYPE STATIONSTATUS_2_TYPE:
STRUCT
     PRM_REQ:          BOOL;
     STAT_DIAG:        BOOL;
     DP_Slave:         BOOL;
     WD_ON:            BOOL;
     FREEZE_MODE:      BOOL;
     SYNC_MODE:        BOOL;
     reserved:         BOOL;
     DEACTIVATED:      BOOL;
END_STRUCT
END_TYPE
```

PRM_REQ Data type: BOOL

This bit is set to TRUE by the DP slave, if reparameterization and reconfiguration of the slave is required (e.g. when adding an additional I/O module). The bit remains set until reparameterization is done.

STAT_DIAG Data type: BOOL

This bit is set to TRUE by the DP slave, if the slave has a static diagnosis. A DP slave with static diagnosis is not ready for I/O data exchange.

DP_SLAVE Data type: BOOL

This bit is permanently set to TRUE.

WD_ON Data type: BOOL

This bit is set to TRUE by the DP slave, if the slave response monitoring is active.

FREEZE_MODE Data type: BOOL

This bit is set to TRUE by the DP slave, if the slave is currently running in Freeze mode. SYNC_MODE BOOL

SYNC_MODE Data type: BOOL

This bit is set to TRUE by the DP slave, if the slave is currently running in Sync mode.

reserved Data type: BOOL

This bit is reserved and currently not used.

DEACTIVATED Data type: BOOL

This bit is set to TRUE, if the DP slave is marked as non-active in the DP master's configuration data and has been taken out of the cyclic I/O data exchange.

Stationstatus_3

STATION-STATUS_3_TYPE

The diagnostic byte Stationstatus_3 (defined within the standard) is output at STAT_3 of the function block DPM_SLV_DIAG. It is represented as a structure of the data type STATION-STATUS_3_TYPE.

The structure STATIONSTATUS_3_TYPE is declared as follows within the PROFIBUS DP library:

```
TYPE STATIONSTATUS_3_TYPE:
STRUCT
    reserved0:          BOOL;
    reserved1:          BOOL;
    reserved2:          BOOL;
    reserved3:          BOOL;
    reserved4:          BOOL;
    reserved5:          BOOL;
    reserved6:          BOOL;
    EXT_DIAG_OVERFLOW: BOOL;
END_STRUCT
END_TYPE
```

Master_Add

MSTR

Data Type: BYTE

The DP slave enters the DP master's address into this octet by which it was parameterized (i.e. to which it is assigned). If the DP slave was not yet parameterized by any DP master, MSTR is set to 255.

Ident_Number

Here, the DP slave enters its identification number in the diagnosis telegram. The identification number is assigned by the PNO (PROFIBUS DP Nutzerorganisation e.V. = PROFIBUS DP user organization) for each device type. This number is a firm component of the GSD file. The function block DPM_SLV_DIAG does not output the device identification number, because the number is not necessary for evaluating the diagnosis data.

Ext_Diag_Data

EXT_DIAG_DAT

Data Type: ARRAY [1..238] OF BYTE

The six bytes of standard diagnosis data described above have to be supported by each DP slave. Optionally, a DP slave can additionally provide extended diagnosis data. This is the case if a value greater than 6 is assigned to the item Max_Diag_Data_Len in the GSD file of the DP slave. The format of the extended diagnosis is defined by the standard. Since the extended diagnosis data are not static on the one side and can contain manufacturer-specific entries on the other side, no automatic data evaluation can be performed by the function block DM_SLV_DIAG.

The evaluation of extended diagnosis data must be performed by the user with the aid of the GSD file for the particular DP slave and the description of the general data format given below.

The extended diagnosis data are divided into three parts (sections):

- device-related diagnosis
- module-related diagnosis
- channel-related diagnosis

Device-related diagnosis

The device-related diagnosis section contains general diagnosis information, such as overtemperature or undervoltage. This section starts with a header byte, the highest two bits of which are permanently set to 00. The lower six bits indicate the length of the following function block, including the header byte itself.

0	0	L	e	n	g	t	h
7	6	5	4	3	2	1	0

The device-related diagnosis data are defined by the manufacturer. For detailed information about their meaning please refer to the particular device documentation.

Module-related diagnosis

The module-related diagnosis section contains diagnosis information, which can be assigned directly to the particular I/O modules of the device. This section containing the module-related diagnosis starts with a header byte, the highest two bits of which are permanently set to 01. The lower six bits indicate the length of the following function block, including the header byte itself.

0	1	L	e	n	g	t	h
7	6	5	4	3	2	1	0

In the following function block, one single bit is assigned to each module. The module index is represented by the bit offset within the function block (please refer to the example). A bit which is set to TRUE means that diagnosis is required for the related I/O module.

Channel-related diagnosis

In the channel-related diagnosis section, the diagnosed channels and the cause for the diagnosis are entered in sequence. Each entry consists of three bytes and starts with a header byte, the highest two bits of which are permanently set to 10. The lower six bits contain the index of the module for which the following diagnosis was made.

1	0	M	o	d	u	l	e
7	6	5	4	3	2	1	0

The lower six bits of the following byte contain the number of the channel which reports a diagnosis. The highest two bits indicate whether the specific channel is an input channel, an output channel or an I/O channel.

I	O	C	h	a	n	n	l
7	6	5	4	3	2	1	0

The direction identifier in the bits 6 and 7 is encoded as follows:

0	0	Reserved
0	1	Input
1	0	Output
1	1	Input / Output

The third byte of each entry contains the channel type in its upper three bits and the error type in the lower five bits.

C	h	n	E	r	r	o	r
7	6	5	4	3	2	1	0

The channel type (channel width) is encoded as follows:

0	0	0	Reserved
0	0	1	Bit
0	1	0	2 bit
0	1	1	4 bit
1	0	0	Byte
1	0	1	Word
1	1	0	2 Word
1	1	1	Reserved

The encoding of the error type is as follows:

0	Reserved	0	0	0	0	0
1	Short circuit	0	0	0	0	1
2	Undervoltage	0	0	0	1	0
3	Overvoltage	0	0	0	1	1
4	Overload	0	0	1	0	0
5	Overtemperature	0	0	1	0	1
6	Cable brake	0	0	1	1	0
7	Upper limit exceeded	0	0	1	1	1
8	Lower limit exceeded	0	1	0	0	0
9	Error	0	1	0	0	1
10	Reserved	0	1	0	1	0
:	:	0	1	x	x	x
15	Reserved	0	1	1	1	1
16	Manufacturer-specific	1	0	0	0	0
:	:	1	x	x	x	x
31	Manufacturer-specific	1	1	1	1	1

The valid length of the complete extended diagnosis data is indicated at output EXT_DIAG_LEN of the function block DPM_SLV_DIAG. When evaluating the diagnosis, only data have to be considered which are contained in the range EXT_DIAG_DAT[1] to EXT_DIAG_DAT[EXT_DIAG_LEN].

Example for extended diagnosis data

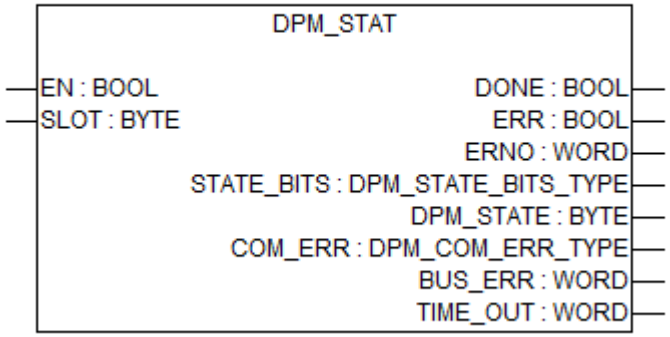
EXT_DIAG_LEN = 15

	7	6	5	4	3	2	1	0	
EXT_DIAG_DAT[1]	0	0	0	0	0	1	0	0	Device-related diagnosis; Length: 4 bytes incl. header byte
EXT_DIAG_DAT[2]	X	x	x	X	X	x	X	X	Device-related diagnosis
EXT_DIAG_DAT[3]	X	x	x	X	X	x	X	X	Length: 3 bytes
EXT_DIAG_DAT[4]	X	x	x	X	X	x	X	X	Meaning of the data is manufacturer-specific
EXT_DIAG_DAT[5]	0	1	0	0	0	1	0	1	ID-related diagnosis; Length: 5 bytes incl. header byte
EXT_DIAG_DAT[6]	0 7	0 6	0 5	0 4	0 3	0 2	0 1	1 0	Module 0 with diagnosis
EXT_DIAG_DAT[7]	015	014	013	112	011	010	0 9	0 8	Module 12 with diagnosis
EXT_DIAG_DAT[8]	023	022	021	020	019	118	017	016	Module 18 with diagnosis
EXT_DIAG_DAT[9]	031	130	029	028	027	026	025	024	Module 30 with diagnosis
EXT_DIAG_DAT[10]	1	0	0	0	0	0	0	0	Channel-related diagnosis module 0
EXT_DIAG_DAT[11]	0	1	0	0	0	0	1	0	Channel 2, Input
EXT_DIAG_DAT[12]	0	0	1	0	0	1	0	0	Overload, channel organized bitwise
EXT_DIAG_DAT[13]	1	0	0	0	1	1	0	0	Channel-related diagnosis module 12
EXT_DIAG_DAT[14]	1	0	0	0	0	1	1	0	Channel 6, Output
EXT_DIAG_DAT[15]	1	0	1	0	0	1	1	1	Upper limit exceeded, channel organized wordwise

Example program for evaluating extended diagnosis data

A detailed example program for the evaluation of extended diagnosis data can be found onABB website under *Download Application Examples*.

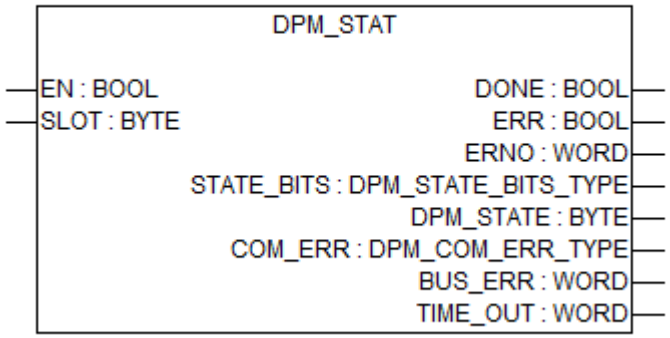
DPM_STAT



Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Diagnosis

The function block DPM_STAT outputs the current PROFIBUS DP Communication Module status. DPM_STAT is active, if input EN = TRUE. While the function block is active, the current values are permanently displayed at the outputs. The outputs provide information about the communication state and error events.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

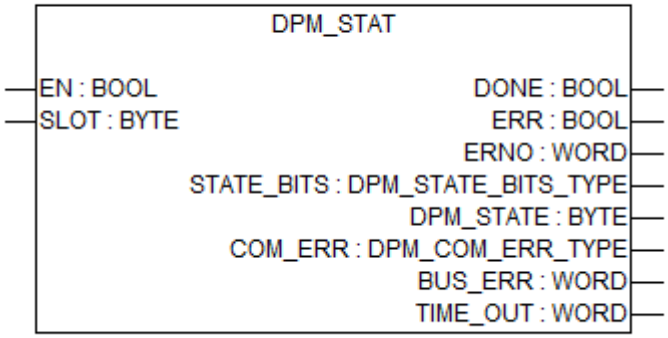
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE_BITS Data type: DPM_STATE_BITS_TYPE

The Output STATE_BITS provides error or diagnosis flags of the PROFIBUS DP Communication Module.

DPM_STATE Data type: BYTE

At output DPM_STATE, the DP master state is output according to the standard. The following states are defined:

```
OFFLINE  00HEX / 00DEC
STOP     40HEX / 64DEC
CLEAR    80HEX / 128DEC
OPERATE  C0HEX / 192DEC
DPM_STATE = OFFLINE
```

The PROFIBUS DP Communication Module is in initialization state, if DPM_STATE has the value OFFLINE. After the initialization phase is completed, the Communication Module changes to STOP state.

DPM_STATE = STOP

The Communication Module is completely initialized, if DPM_STATE has the value STOP. In this state the Communication Module is ready to receive configuration data. No data are exchanged with the DP slaves. The Communication Module has this state, if no user program is running.

DPM_STATE = CLEAR

When starting the user program, the Communication Module changes from STOP into CLEAR state and begins, via the bus, to parameterize (set into operation) the DP slaves assigned during configuration. When the setup has been completed successfully, the Communication Module moves to OPERATE state. If an error occurs during parameterization, the Communication Module changes back to STOP state.

DPM_STATE = OPERATE

Normally, the Communication Module is in OPERATE state, while a user program is running. In this state, the DP master exchanges I/O data with the DP slaves. If an error occurs during this process, and if *Auto Clear Mode* was selected during configuration, the Communication Module changes back to CLEAR state and tries to parameterize the DP slaves again.

If *Auto Clear Mode* was not selected, the Communication Module remains (in case of an error) in OPERATE state. When stopping the user program, the Communication Module also changes back to STOP state.

DPM_STATE is only valid, if EN = TRUE and ERR = FALSE.

COM_ERR Data type: DPM_COM_ERR_TYPE

The Output COM_ERR provides detailed information to the corresponding output STATE_BITS.

The element address contains the node address of the faulty device and the element event the error code. If several errors occur simultaneously, the output contains the error of the device with the lowest address.

The structure of the type DPM_COM_ERR_TYPE is defined in the PROFIBUS library and described in the following chapter together with the possible errors.

BUS_ERR Data type: WORD

BUS_ERR outputs the number of serious bus errors since system startup, such as transmission line short circuits.

BUS_ERR is only valid, if EN = TRUE and ERR = FALSE.

TIME_OUT Data type: WORD

TIME_OUT outputs the number of timeout errors since system startup. A timeout error occurs, if a DP slave does not respond to a DP master's request within the configured time.

TIME_OUT is only valid, if EN = TRUE and ERR = FALSE.

On CM592-DP BUS_ERR and TIME_OUT are incremented always at once. Due to internal handling of error events the counters are updated with steps greater one always.

Function call in ST

```
STAT
    (EN := STAT_EN,
   SLOT := STAT_SLOT);


STAT_DONE       := STAT.DONE;
STAT_ERR        := STAT.ERR;
STAT_ERNO       := STAT.ERNO;
STAT_STATE_BITS := STAT.STATE_BITS;
STAT_DPM_STATE  := STAT.DPM_STATE;
STAT_COM_ERR    := STAT.COM_ERR;
STAT_BUS_ERR    := STAT.BUS_ERR;
STAT_TIME_OUT   := STAT.TIME_OUT;
```

STATE_BITS DPM_STATE_BITS_TYPE

The structure STATE_BITS includes four Boolean variables which display different communication states. Within the PROFIBUS DP library, the data type DPM_STATE_BITS_TYPE is declared as follows:

```
TYPE DPM_STATE_BITS_TYPE:
STRUCT
    CTRL:       BOOL;
    AUTO_CLR:   BOOL;
    NO_EXCH:    BOOL;
    FATAL:      BOOL;
    EVENT:      BOOL;
    TIMEOUT:    BOOL;
END_STRUCT
END_TYPE
```

CTRL Data type: BOOL

If this bit is TRUE, a parameter setting error occurred. During normal operation, CTRL should be FALSE. If this is not the case, the parameter and configuration data have to be checked.

AUTO_CLR Data type: BOOL

This bit is only valid, if *Auto Clear Mode* was set during the configuration. If AUTO_CLR is true, an error occured during communication with at least one DP slave. As a result, the Communication Module stopped the data exchange with all DP slaves and changed back to CLEAR state (see DPM_STATE).

NO_EXCH Data type: BOOL

This bit is set to TRUE, if process data exchange with one or several DP slaves is not possible. The error can be caused by the configuration data or by the DP slaves.

If no slave devices are connected, the Communication Modules signal state the following information:

CM592-DP: the bit NO_EXCH remains not set and has FALSE value.

FATAL Data type: BOOL

If FATAL is TRUE, no communication via the PROFIBUS DP is possible due to a serious bus error (e.g. bus line short circuit). In this case, all bus lines have to be checked.

EVENT Data type: BOOL

This bit is set to TRUE, if a bus short circuit was detected. The number of detected short circuits can be read at the BUS_ERR output. After setting the bit once, its state remains TRUE.

State bit EVENT is not supported by Communication Module CM592-DP. EVENT is always set to FALSE .

TIMEOUT Data type: BOOL

This bit is set to TRUE, if a telegram timeout was detected. The number of detected timeouts can be read at the TIME_OUT output. After setting the bit once, its state remains TRUE.

COM_ERR DPM_COM_ERR_TYPE

Communication errors can be located more detailed using COM_ERR. COM_ERR outputs a structure of the type DPM_COM_ERR_TYPE. This data type is declared as follows within the PROFIBUS DP library:

```
TYPE DPM_COM_ERR_TYPE:
STRUCT
    ADDRESS:  BYTE;
    EVENT:    BYTE;
END_STRUCT
END_TYPE
```

COM ERR is not supported by Communication Module CM592-DP. ADDRESS and EVENT are always set to 0.

ADDRESS Data type: BYTE

If an error occurs, ADDRESS contains the bus address of the faulty device (0 to 125). If ADDRESS has the value 255, the error is located in the Communication Module itself.

EVENT Data type; BYTE

EVENT displays the cause of an error.

The following tables show the encoding of the various errors.

Table 89: ADDRESS = 255 / Communication module error

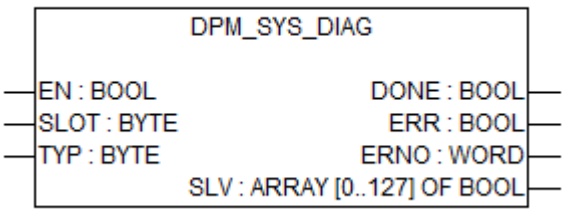
Event	Description	Error source	Remedy
50	USR_INTF task not found	Communication Module	Contact ABB
51	No global data field	Communication Module	Contact ABB
52	FDL task not found	Communication Module	Contact ABB
53	PLC task not found	Communication Module	Contact ABB
54	No master parameter record	Configuration	Generate configuration in the project and reload program to controller
55	Faulty value in master parameter record	Configuration	Check configuration data for Communication Module in the project and/or reload program to controller
56	No slave parameter records	Configuration	Add DP slaves to configuration data and reload program to controller
57	Faulty value in a slave parameter record	Configuration	Check configuration data of subordinate DP slaves in the project and/or reload program to controller
58	Doubled slave address	Configuration	Check configuration data of subordinate DP slaves in the project for doubled bus addresses and/or reload program to controller
59	Invalid offset address output data	Configuration	Check configuration data of subordinate DP slaves in the project for invalid IEC addresses and/or reload program to controller
60	Invalid offset address input data	Configuration	Check configuration data of subordinate DP slaves in the project for invalid IEC addresses and/or reload program to controller
61	Range overlapping in output data	Configuration	Check configuration data of subordinate DP slaves in the project for overlapping IEC address ranges and/or reload program to controller
62	Range overlapping in input data	Configuration	Check configuration data of subordinate DP slaves in the project for overlapping IEC address ranges and/or reload program to controller

Event	Description	Error source	Remedy
63	Unknown process data handshake	Controller	Supply voltage OFF/ON, otherwise contact ABB
64	Insufficient memory	Communication Module	Contact ABB
65	Faulty slave parameter record	Configuration	Check configuration data of subordinate DP slaves in the project and/or reload program to controller
202	No segment available	Communication Module	Contact ABB
212	Error while reading database	Configuration/ Communication Module	Reload program with configuration data to controller
213	Faulty transfer structure operating system	Communication Module	Contact ABB

Table 90: ADDRESS = 0..125 / Error at subscriber with bus address ADDRESS

Event	Description	Error source	Remedy
2	Subscriber reports overflow	DP master telegram	Check configuration data of subordinate DP slave in the project and/or reload program to controller
3	Subscriber does not support requested function	DP master telegram	Check DP slave for conformity according to PROFIBUS DP standard
9	No data in response telegram	DP slave	Compare configuration data of subordinate DP slave in the project with actual configuration and reload program to controller if necessary
17	Subscriber does not response	DP slave	Check bus line and DP slave bus address
18	DP master not in logical token ring	DP master	Check the configured DP master bus address, the highest station address (HSA) in the other system DP masters and/or bus line for short circuits
21	Faulty parameter in request telegram	DP master telegram	Contact ABB

DPM_SYS_DIAG

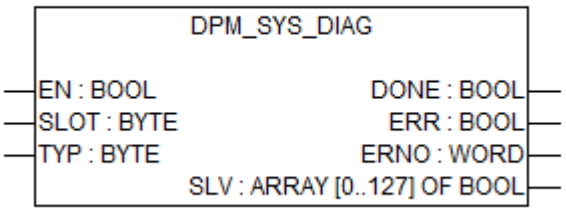


Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Diagnosis

The function block DPM_SYS_DIAG outputs different surveys reporting the status of all DP slaves. Three survey types can be selected:

- configuration survey
- I/O data exchange survey
- diagnosis survey

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

TYP (type)

Data type: BYTE

TYP=1, configuration survey

Output SLV indicates which DP slaves have been configured successfully (i.e. set into operation) by the DP master. Please note, that the DP master only sets DP slaves into operation which were assigned to the master when generating the configuration data.

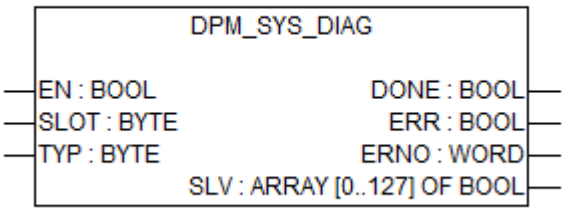
TYP=2, data exchange survey

SLV outputs all DP slaves with which the DP master exchanges data. The data exchange can only be performed with DP slaves which were configured by the DP master itself. The data exchange survey can only be requested, if the communication module is in OPERATE state.

TYP=3, diagnosis survey

Output SLV indicates all DP slaves which have signalized an available diagnosis. The diagnosis survey can only be requested, if the Communication Module is in OPERATE state.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

SLV (slave)

Data type: ARRAY[0..127] OF BOOL

SLV outputs the status survey as array of BOOL values. Every index within this array represents a DP slave. The index itself corresponds to the DP slave bus address. When a certain value is set to TRUE, corresponding slave signals information regarding selected survey type in input TYP.

If e.g. TYP = 1 is selected and SLV[2] = TRUE, the DP slave was successfully configured with bus address 2 by the DP master. If SLV[2] = FALSE, the configuration of the specific DP slave has not yet been completed or the DP slave is not part of the DP master configuration data.

If TYP = 2 was selected and SLV[2] = TRUE, the DP master exchanges I/O data with the DP slave having bus address 2. If SLV[2] = FALSE, the DP master currently does not exchange I/O data with the DP slave. The DP master is only able to exchange data with DP slaves which it has successfully set into operation before.

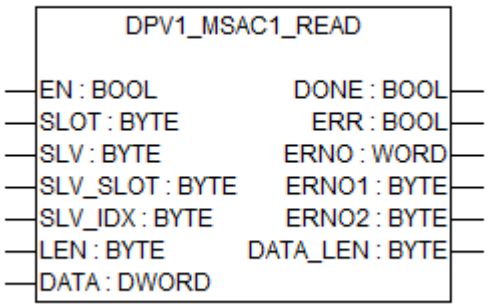
If TYP = 3, SLV[2] = TRUE means, that the DP slave with bus address 2 has signalized a diagnosis. The diagnosis description can then be requested using the function block DPM_SLV_DIAG.

Output SLV is only valid, if EN = TRUE and ERR = FALSE.

Function call in ST

```
SYSDIAG
       (EN := SYSDIAG_EN,
       SLOT:= SYSDIAG_SLOT,
       TYP := SYSDIAG_TYP);


SYSDIAG_DONE:= SYSDIAG.DONE;
SYSDIAG_ERR := SYSDIAG.ERR;
SYSDIAG_ERNO:= SYSDIAG.ERNO;
SYSDIAG_SLV := SYSDIAG.SLV;
```

DPV1_MSAC1_READ

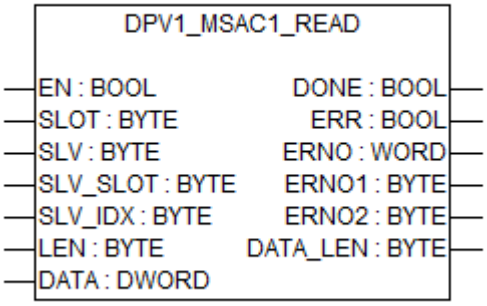


Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Data

The function block DPV1_MSAC1_READ implements the acyclic PROFIBUS DP DPV1 service MSAC1_READ. Using this function, the master has read access to slot and index-related data of slaves supporting DPV1. DPV1_MSAC1_READ works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPV1_MSAC1_READ reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is applied, the data of which shall be read.

Different range of values have to be considered depending on the used communication module. For CM592-DP it is 0..125.

SLV_SLOT

Data type: BYTE

At input SLV_SLOT the number of the slot within the slave is specified, the data of which shall be read.

Valid values: 0..254.

SLV_IDX

Data type: BYTE

At input SLV_IDX the number of the index within the slot is specified, the data of which shall be read.

Valid values: 0..254.

LEN (length)

Data type: BYTE

The length of the data block to be read is specified at input LEN.

Valid values: 0..240.

Function block reads a complete data block independent of input LEN.

If length of data block is unknown, set LEN = 240 (max. data LEN).

CAUTION!

Overwriting of variables to data!

If LEN is lower than length of data block the following variables to data will be overwritten!

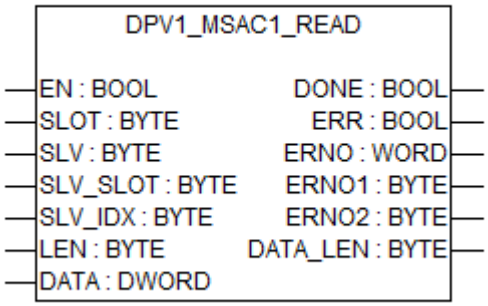
DATA (data)

Data type: DWORD

At input DATA the address of the variable where the received data block shall be stored is specified via the ADR address operator.

The size of the variable must be big enough to store the complete data block (e.g. BYTE array). Furthermore, the format (BYTE, WORD, etc.) of the data must be considered.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1)

Data type: BYTE

Output ERNO1 provides an additional DPV1-specific error information in case that an error occurred during processing.

ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO1 of the DPV1 function blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

7	6	5	4	3	2	1	0
Error class				Error code			

ERNO1		Error class/Error code
DEC	HEX	
0	0	Reserved
...	...	...
159	9F	Reserved
160	A0	10 Application / 0 Read error
161	A1	10 Application / 1 Write error
162	A2	10 Application / 2 Error module
163	A3	Reserved
...	...	...
167	A7	Reserved
168	A8	10 Application / 8 Version conflict
169	A9	10 Application / 9 Function not supported
170	AA	10 Application / 10 Manufacturer-specific
...	...	...
175	AF	10 Application / 15 Manufacturer-specific
176	B0	11 Access / 0 Invalid index
177	B1	11 Access / 1 Invalid length of data to be written
178	B2	11 Access / 2 Invalid slot
179	B3	11 Access / 3 Type conflict
180	B4	11 Access / 4 Invalid range
181	B5	11 Access / 5 Status conflict
182	B6	11 Access / 6 Access denied
183	B7	11 Access / 7 Invalid value range

184	B8	11 Access / 8 Invalid parameter
185	B9	11 Access / 9 Invalid type
186	BA	11 Access / 10 Manufacturer-specific
...	...	...
191	BF	11 Access / 15 Manufacturer-specific
192	C0	12 Resources / 0 Read conflict
193	C1	12 Resources / 1 Write conflict
194	C2	12 Resources / 2 Resource used
195	C3	12 Resources / 3 Resource not available
196	C4	Reserved
...	...	...
199	C7	Reserved
200	C8	12 Resources / 10 Manufacturer-specific
...	...	...
207	CF	12 Resources / 15 Manufacturer-specific
208	D0	Reserved
...	...	...
255	FF	Reserved

ERNO2 (error number 2)

Data type: BYTE

Output ERNO2 provides an additional DPV1-specific error information, if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO.

The value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

The encoding of ERNO2 is completely manufacturer-specific.

DATA_LEN (data length)

Data type: BYTE

Output DATA_LEN displays the length (in bytes) of the input data read by the slave.

DATA_LEN is only valid, if DONE is TRUE and ERR is FALSE. If DATA_LEN contains a value X which is not 0, the function block has stored X bytes of input data in the variable specified at DATA.

Example

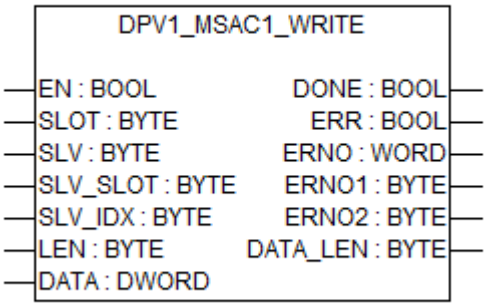
If DATA is a byte array with start index 1, the valid input data of the slave are contained in the entries DATA[1] to DATA[X].

Function call in ST

```
DPV1_READ
     (EN  := DPV1_READ_EN,
     SLOT := DPV1_READ_SLOT,
     SLV  := DPV1_READ_SLV,
     SLV_SLOT := DPV1_READ_SLV_SLOT,
     SLV_IDX  := DPV1_READ_SLV_IDX,
     LEN  := DPV1_READ_LEN,
     DATA := ADR(DPV1_READ_DATA));
```

```
DPV1_READ_DONE     := DPV1_READ.DONE;
DPV1_READ_ERR      := DPV1_READ.ERR;
DPV1_READ_ERNO     := DPV1_READ.ERNO;
DPV1_READ_ERNO1    := DPV1_READ.ERNO1;
DPV1_READ_ERNO2    := DPV1_READ.ERNO2;
DPV1_READ_DATA_LEN := DPV1_READ.DATA_LEN;
```

DPV1_MSAC1_WRITE

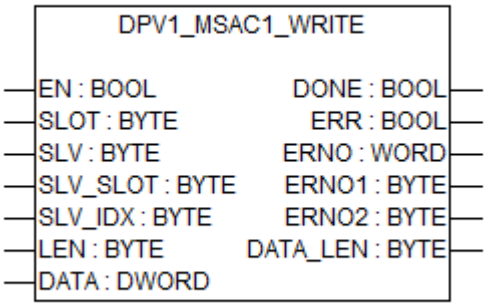


Parameter	Value
Included in library	PROFIBUS_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	Data

The function block DPV1_MSAC1_WRITE implements the acyclic PROFIBUS DP DPV1 service MSAC1_WRITE. Using this function, the master has write access to slot and index-related data of slaves supporting DPV1. DPV1_MSAC1_WRITE works outside the cyclic process data exchange.

Every time a FALSE → TRUE edge is applied to input EN, DPV1_MSAC1_WRITE reads the values at its inputs and the data to be written and sends a corresponding request message to the Communication Module. Further FALSE → TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

SLV (slave)

Data type: BYTE

At input SLV the bus address of the DP slave is specified to which the data shall be written.

Different range of values have to be considered depending on the used communication module. For CM592-DP it is 0..125.

SLV_SLOT

Data type: BYTE

At input SLV_SLOT the number of the slot within the slave is specified to which the data shall be written.

Valid values: 0..254.

SLV_IDX

Data type: BYTE

At input SLV_IDX the number of the index within the slot is specified to which the data shall be written.

Valid values: 0..254.

LEN (length)

Data type: BYTE

At input LEN the length of the data block to be written is specified.

Valid values: 0..240.

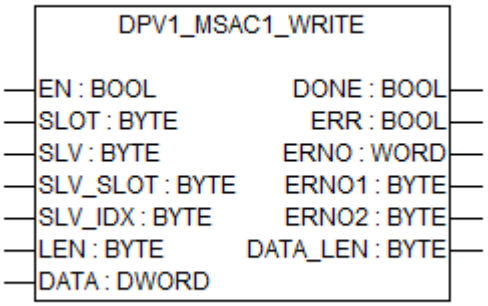
DATA (data)

Data type: DWORD

At input DATA the address of the variable containing the data block to be transmitted is specified via the ADR address operator. The size of the variable must be big enough to store the complete data block (e.g. BYTE array).

Furthermore, the format (BYTE, WORD, etc.) of the data must be considered.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1)

Data type: BYTE

Output ERNO1 provides an additional DPV1-specific error information in case that an error occurred during processing.

ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO1 of the DPV1 function blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

7	6	5	4	3	2	1	0
Error class				Error code			

ERNO1		**Error class/Error code**
DEC	HEX	
0	0	Reserved
...	...	...
159	9F	Reserved
160	A0	10 Application / 0 Read error
161	A1	10 Application / 1 Write error
162	A2	10 Application / 2 Error module
163	A3	Reserved
...	...	...
167	A7	Reserved
168	A8	10 Application / 8 Version conflict
169	A9	10 Application / 9 Function not supported
170	AA	10 Application / 10 Manufacturer-specific
...	...	...
175	AF	10 Application / 15 Manufacturer-specific
176	B0	11 Access / 0 Invalid index
177	B1	11 Access / 1 Invalid length of data to be written
178	B2	11 Access / 2 Invalid slot
179	B3	11 Access / 3 Type conflict
180	B4	11 Access / 4 Invalid range
181	B5	11 Access / 5 Status conflict
182	B6	11 Access / 6 Access denied
183	B7	11 Access / 7 Invalid value range
184	B8	11 Access / 8 Invalid parameter
185	B9	11 Access / 9 Invalid type
186	BA	11 Access / 10 Manufacturer-specific
...	...	...
191	BF	11 Access / 15 Manufacturer-specific
192	C0	12 Resources / 0 Read conflict
193	C1	12 Resources / 1 Write conflict
194	C2	12 Resources / 2 Resource used
195	C3	12 Resources / 3 Resource not available

196	C4	Reserved
...	...	...
199	C7	Reserved
200	C8	12 Resources / 10 Manufacturer-specific
...	...	...
207	CF	12 Resources / 15 Manufacturer-specific
208	D0	Reserved
...	...	...
255	FF	Reserved

ERNO2 (error number 2)

Data type: BYTE

Output ERNO2 provides an additional DPV1-specific error information, if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO.

The value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

The encoding of ERNO2 is completely manufacturer-specific.

DATA_LEN (data length)

Data type: BYTE

The output DATA_LEN displays the actual length (in bytes) of the data written to the slave.

DATA_LEN is only valid, if DONE = TRUE and ERR = 0.

Function call in ST

```
DPV1_WRITE
     (EN  := DPV1_WRITE_EN,
     SLOT := DPV1_WRITE_SLOT,
     SLV  := DPV1_WRITE_SLV,
     SLV_SLOT := DPV1_WRITE_SLV_SLOT,
     SLV_IDX  := DPV1_WRITE_SLV_IDX,
     LEN  := DPV1_WRITE_LEN,
     DATA := ADR(DPV1_WRITE_DATA));


DPV1_WRITE_DONE     := DPV1_WRITE.DONE;
DPV1_WRITE_ERR      := DPV1_WRITE.ERR;
DPV1_WRITE_ERNO     := DPV1_WRITE.ERNO;
DPV1_WRITE_ERNO1    := DPV1_WRITE.ERNO1;
DPV1_WRITE_ERNO2    := DPV1_WRITE.ERNO2;
DPV1_WRITE_DATA_LEN := DPV1_WRITE.DATA_LEN;
```

1.5.4.26.2 Data types

Data type	Description
DPM_COM_ERR_TYPE	Communication error
DPM_STATE_BITS_TYPE	Bits for Communication Module state description
STATIONSTATUS_1_TYPE	Stationstatus_1 (DP slave diagnosis according to standard)

Data type	Description
STATIONSTATUS_2_TYPE	Stationstatus_2 (DP slave diagnosis according to standard)
STATIONSTATUS_3_TYPE	Stationstatus_3 (DP slave diagnosis according to standard)

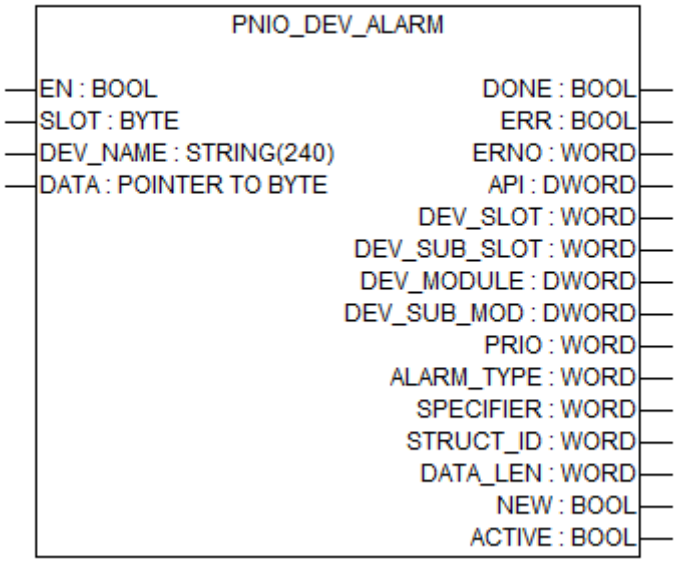
1.5.4.27 PROFINET IO library

Library file name: **PROFINET_AC500_Vx.lib.**

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

1.5.4.27.1 Function blocks

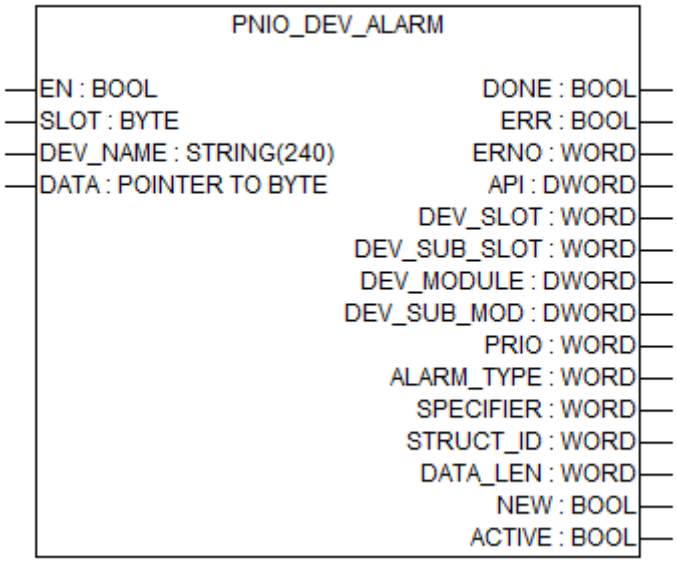
PNIO_DEV_ALARM



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Alarm

The function block PNIO_DEV_ALARM provides diagnostic and emergency information describing the current condition of a certain PROFINET IO device.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

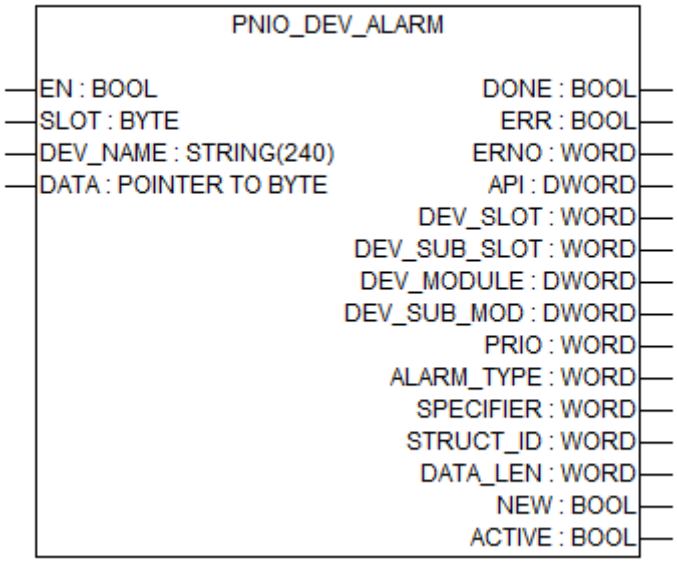
ci502-pn-##

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

API

Data type	Default value	Range	Unit
DWORD	-	-	-

The output API provides the ALARM API of the current alarm.

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_SUB_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SUB_SLOT provides the device sub slot which sent the alarm.

DEV_MODULE

Data type	Default value	Range	Unit
DWORD	-	-	

The output DEV_MODULE provides the ID of the module which sent the alarm.

DEV_SUB_MOD

Data type	Default value	Range	Unit
DWORD	-	-	-

The output DEV_SUB_MOD provides the ID of the sub module which sent the alarm.

PRIO

Data type	Default value	Range	Unit
WORD	-	-	-

The output PRIO provides the priority of the alarm sent by the device.

ALARM_TYPE

Data type	Default value	Range	Unit
WORD	-	-	-

The output ALARM_TYPE describes the kind of the reason for diagnosis message received. The ABB I/O modules CI501-PNIO and CI502-PNIO currently only deliver messages of the type (2): process.

Value	Diagnosis type
16#0000	reserved
16#0001	Diagnosis appears
16#0002	Process
16#0003	Pull
16#0004	Plug
16#0005	Status
16#0006	Update
16#0007	Redundancy

Value	Diagnosis type
16#0008	Controlled by supervisor
16#0009	Released
16#000A	Plug wrong submodule
16#000B	Return of submodule
16#000C	Diagnosis disappears
16#000D	Multicast communication mismatch
16#000E	Port data change notification
16#000F	Sync data change notification
16#0010	Isochronous mode notification
16#0011	Network component problem notification
16#0012	Time data changed notification
16#0013	Dynamic Frame Packing problem notification
16#0014	MRPD problem notification
16#0015	System Redundancy notification
16#0016	Multiple interface mismatch notification
16#0017 - 16#001D	Reserved
16#001E	Upload and retrieval notification
16#001F	Pull mode
16#0020 - 16#007F	Manufacturer specific
16#0080 - 16#00FF	Reserved for profiles
16#0100 - 16#FFFF	Reserved

SPECIFIER

Data type	Default value	Range	Unit
WORD	-	0-15	bit

The bits of the diagnosis specifier describe the type and specification of the current diagnosis:

Bit	Description
0 ... 10	Sequence number
11	Channel diagnosis
12	Manufacturer specific diagnosis
13	Submodule diagnosis state
14	Reserved
15	AR diagnosis state

The SPECIFIER can be displayed in plain text using the function PNIO_DEV_SPECIFIER
↳ *Chapter 1.5.4.27.1.3 “PNIO_DEV_INFO” on page 1803*.

SRUCT_ID

Data type	Default value	Range	Unit
WORD	-	-	-

The output STRUCT_ID provides the manufacture's proprietary data structure ID.

DATA_LEN

Data type	Default value	Range	Unit
WORD	-	-	-

The output DATA_LEN provides the size of / numbers of bytes which were written to DATA.

NEW

Data type	Default value	Range	Unit
BOOL	-	-	-

Signals whether new valid diagnosis information is available. If DONE and NEW are TRUE, the outputs marked with an asterisk (*) are valid and the data transferred to DATA is up to date.

ACTIVE

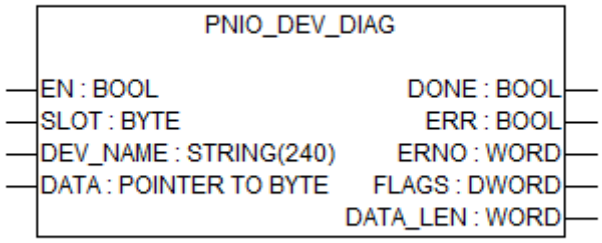
Data type	Default value	Range	Unit
BOOL	-	-	-

The inquired device is currently active. This output does not depend on the NEW output.

Function call in ST

```
PNIO_DEVICE_ALARM(
EN         := PNIO_DEVICE_ALARM_EN,
SLOT       := PNIO_DEVICE_ALARM_SLOT,
SLV_NAME   := PNIO_DEVICE_ALARM_DEV_NAME,
DATA       := ADR(PNIO_DEVICE_ALARM_DATA));
PNIO_DEVICE_ALARM_DONE                  := PNIO_DEVICE_ALARM.DONE;
PNIO_DEVICE_ALARM_ERR                   := PNIO_DEVICE_ALARM.ERR;
PNIO_DEVICE_ALARM_ERNO                  := PNIO_DEVICE_ALARM.ERNO;
PNIO_DEVICE_ALARM_API                   := PNIO_DEVICE_ALARM.API;
PNIO_DEVICE_ALARM_DEV_SLOT              := PNIO_DEVICE_ALARM.DEV_SLOT;
PNIO_DEVICE_ALARM_DEV_SUB_SLOT          :=
PNIO_DEVICE_ALARM.DEV_SUBSLOT;
PNIO_DEVICE_ALARM_DEV_MODULE            :=
PNIO_DEVICE_ALARM.DEV_MODULE;
PNIO_DEVICE_ALARM_DEV_SUB_MOD           :=
PNIO_DEVICE_ALARM.DEV_SUB_MOD;
PNIO_DEVICE_ALARM_PRIO                  := PNIO_DEVICE_ALARM.PRIO;
PNIO_DEVICE_ALARM_ALARM_TYPE            :=
PNIO_DEVICE_ALARM.ALARM_TYPE;
PNIO_DEVICE_ALARM_SPECIFIER             :=
PNIO_DEVICE_ALARM.SPECIFIER;
PNIO_DEVICE_ALARM_STRUCT_ID             :=
PNIO_DEVICE_ALARM.STRUCT_ID;
PNIO_DEVICE_ALARM_DATA_LEN              := PNIO_DEVICE_ALARM.DATA_LEN;
PNIO_DEVICE_ALARM_NEW                   := PNIO_DEVICE_ALARM.NEW;
PNIO_DEVICE_ALARM_ACTIVE                := PNIO_DEVICE_ALARM.ACTIVE;
```

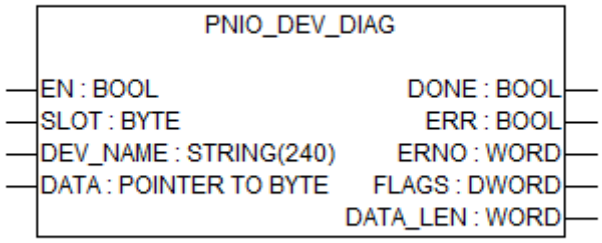
PNIO_DEV_DIAG



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Diagnosis

The function block PNIO_DEV_DIAG provides diagnostic information of a certain PROFINET IO device.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

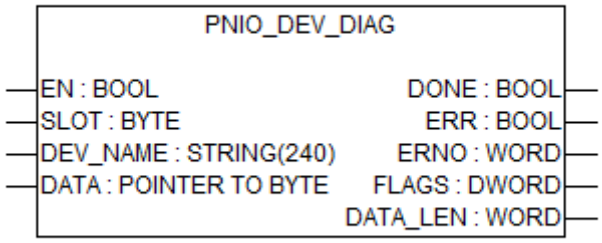
ci502-pn-##

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

FLAGS

Data type	Default value	Range	Unit
DWORD	-	0-11	Bit

The output FLAGS provides the status flags of the PROFINET IO device chosen by DEV_NAME.

The individual bits have the following meaning:

Bit	Description
0	The I/O module does not exist / The I/O module does not respond to DCP Ident requests
1	The I/O module is not ready
2	A configuration fault exists for this I/O module (e. g. NameOf-Station or IP is used more than once in one network)
3	The I/O module transmits an invalid response (e. g. DCP Set IP was not successful)
4	A parameter fault exists for this I/O module (e. g. wrong Module ID)
5	This I/O module is deactivated
6	Diagnosis data exists for this I/O module
7	The I/O module sends a "Diagnosis disappeared" alert
8	The diagnosis buffer of the Communication Module was too small for the diagnosis data sent by the I/O module
9	The diagnosis buffer of the Communication Module was over-written by new diagnosis data of the I/O module before the old data was read out
10	The packet requesting diagnosis data is too small to carry the diagnosis data for this I/O module
11	The I/O module reported a ModuleDiffBlock during connection establishment

DATA_LEN

Data type	Default value	Range	Unit
DWORD	-	-	-

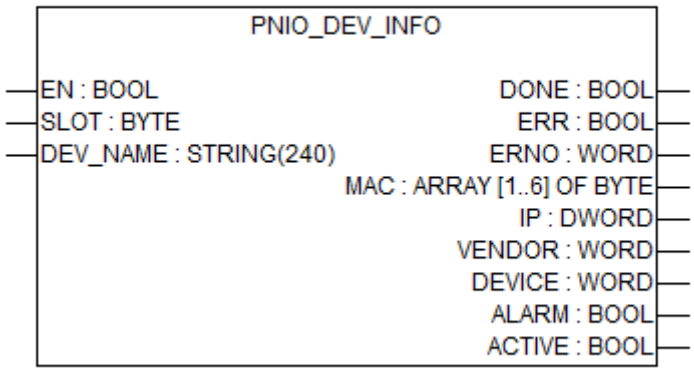
The output DATA_LEN provides the size of numbers / of bytes which were written to DATA.

Function call in ST

```
PNIO_DEVICE_DIAG(EN := PNIO_DEVICE_DIAG_EN,
```

```
    SLOT := PNIO_DEVICE_DIAG_SLOT,
    DATA:= ADR(PNIO_DEVICE_DIAG_DATA));
PNIO_DEVICE_DIAG_DONE            := PNIO_DEVICE_DIAG.DONE;
PNIO_DEVICE_DIAG_ERR             := PNIO_DEVICE_DIAG.ERR;
PNIO_DEVICE_DIAG_ERNO            := PNIO_DEVICE_DIAG.ERNO;
PNIO_DEVICE_DIAG_FLAGS           := PNIO_DEVICE_DIAG.FLAGS;
PNIO_DEVICE_DIAG_DATA_LEN        := PNIO_DEVICE_DIAG.DATA_LEN;
```

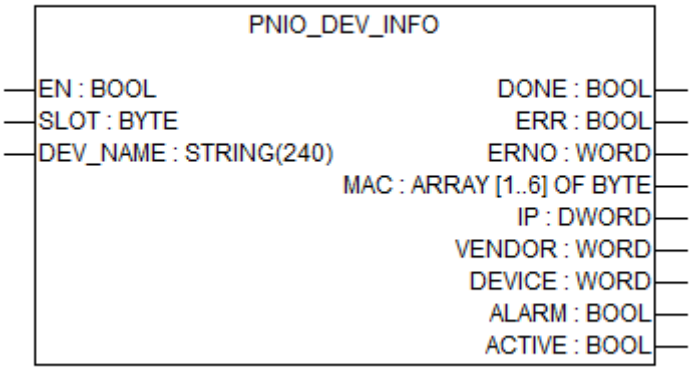
PNIO_DEV_INFO



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Info

The function block PNIO_DEV_INFO provides general information of a certain PROFINET device.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

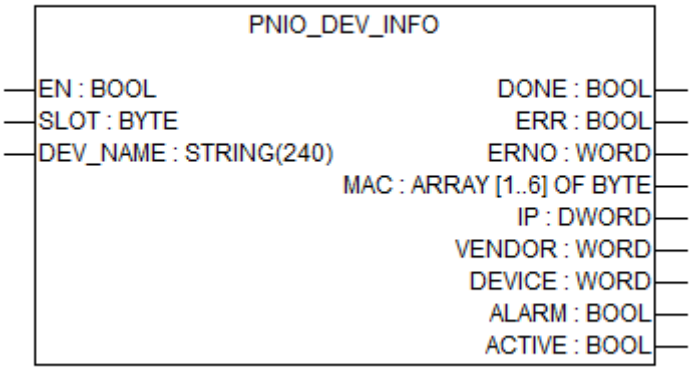
The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

MAC

Data type	Default value	Range	Unit
ARRAY	-	-	-

The output MAC provides the MAC address of the device chosen by DEV_NAME.

IP

Data type	Default value	Range	Unit
DWORD	-	-	-

The output IP provides the IP address of the device chosen by DEV_NAME.

VENDOR

Data type	Default value	Range	Unit
WORD	-	-	-

The output VENDOR provides the manufacturers VENDOR ID of the device chosen by DEV_NAME.

DEVICE

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEVICE provides the Device ID of the device chosen by DEV_NAME.

ALARM

Data type	Default value	Range	Unit
BOOL	-	-	-

If this output is TRUE an alarm is pending on the device chosen by DEV_NAME.

ACTIVE

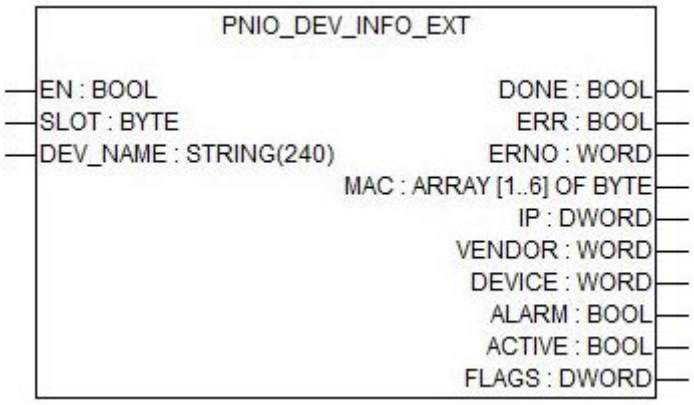
Data type	Default value	Range	Unit
BOOL	-	-	-

The inquired device is currently active. This output does not depend on the NEW output.

Function call in ST

```
PNIO_DEVICE_INFO(EN := PNIO_DEVICE_INFO_EN,
    SLOT := PNIO_DEVICE_INFO_SLOT,
DEV_NAME := PNIO_DEVICE_INFO_DEV_NAME);
PNIO_DEVICE_INFO_DONE             := PNIO_DEVICE_INFO.DONE;
PNIO_DEVICE_INFO_ERR              := PNIO_DEVICE_INFO.ERR;
PNIO_DEVICE_INFO_ERNO             := PNIO_DEVICE_INFO.ERNO;
PNIO_DEVICE_INFO_MAC              := PNIO_DEVICE_INFO.MAC;
PNIO_DEVICE_INFO_IP               := PNIO_DEVICE_INFO.IP;
PNIO_DEVICE_INFO_VENDOR           := PNIO_DEVICE_INFO.VENDOR;
PNIO_DEVICE_INFO_DEVICE           := PNIO_DEVICE_INFO.DEVICE;
PNIO_DEVICE_INFO_ALARM            := PNIO_DEVICE_INFO.ALARM;
PNIO_DEVICE_INFO_ACTIVE           := PNIO_DEVICE_INFO.ACTIVE;
```

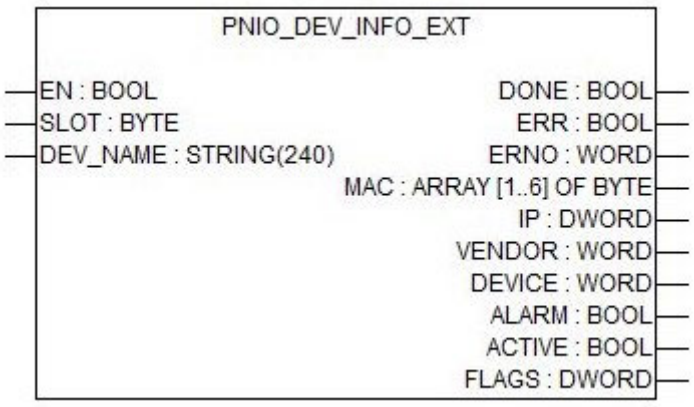
PNIO_DEV_INFO_EXT



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Info

The function block PNIO_DEV_INFO_EXT provides general information of a certain PROFINET IO device.

Input Description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

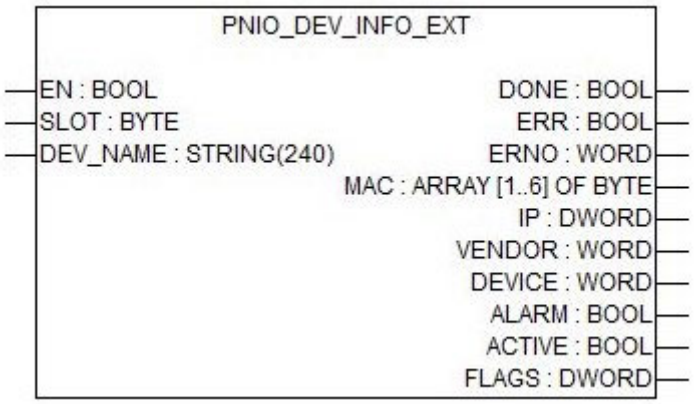
The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

Output Description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

MAC

Data type	Default value	Range	Unit
ARRAY	-	-	-

The output MAC provides the MAC address of the device chosen by DEV_NAME.

IP

Data type	Default value	Range	Unit
DWORD	-	-	-

The output IP provides the IP address of the device chosen by DEV_NAME.

VENDOR

Data type	Default value	Range	Unit
WORD	-	-	-

The output VENDOR provides the manufacturers VENDOR ID of the device chosen by DEV_NAME.

DEVICE

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEVICE provides the Device ID of the device chosen by DEV_NAME.

ALARM

Data type	Default value	Range	Unit
BOOL	-	-	-

If this output is TRUE an alarm is pending on the device chosen by DEV_NAME.

ACTIVE

Data type	Default value	Range	Unit
BOOL	-	-	-

The inquired device is currently active. This output does not depend on the NEW output.

FLAGS

Data type	Default value	Range	Unit
DWORD	-	0-11	Bit

The output FLAGS provides the status flags of the PROFINET IO device chosen by DEV_NAME.

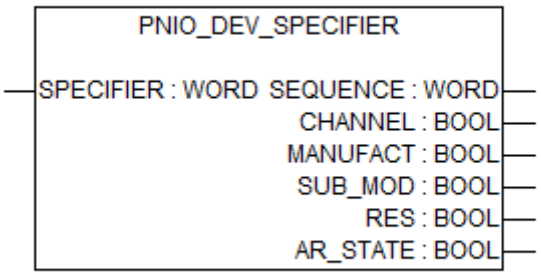
The individual bits have the following meaning:

Bit	Description
0	The I/O module does not exist / The I/O module does not respond to DCP Ident requests
1	The I/O module is not ready
2	A configuration fault exists for this I/O module (e. g. NameOf-Station or IP is used more than once in one network)
3	The I/O module transmits an invalid response (e. g. DCP Set IP was not successful)
4	A parameter fault exists for this I/O module (e. g. wrong Module ID)
5	This I/O module is deactivated
6	Diagnosis data exists for this I/O module
7	The I/O module sends a "Diagnosis disappeared" alert
8	The diagnosis buffer of the Communication Module was too small for the diagnosis data sent by the I/O module
9	The diagnosis buffer of the Communication Module was over-written by new diagnosis data of the I/O module before the old data was read out
10	The packet requesting diagnosis data is too small to carry the diagnosis data for this I/O module
11	The I/O module reported a ModuleDiffBlock during connection establishment

Function call in ST

```
PNIO_DEVICE_INFO_EXT(EN   := PNIO_DEVICE_INFO_EXT_EN,
                     SLOT := PNIO_DEVICE_INFO_EXT_SLOT);
PNIO_DEVICE_INFO_EXT_DONE             := PNIO_DEVICE_INFO_EXT.DONE;
PNIO_DEVICE_INFO_EXT_ERR              := PNIO_DEVICE_INFO_EXT.ERR;
PNIO_DEVICE_INFO_EXT_ERNO             := PNIO_DEVICE_INFO_EXT.ERNO;
PNIO_DEVICE_INFO_EXT_MAC              := PNIO_DEVICE_INFO_EXT.MAC;
PNIO_DEVICE_INFO_EXT_IP               := PNIO_DEVICE_INFO_EXT.IP;
PNIO_DEVICE_INFO_EXT_VENDOR           := PNIO_DEVICE_INFO_EXT.VENDOR;
PNIO_DEVICE_INFO_EXT_DEVICE           := PNIO_DEVICE_INFO_EXT.DEVICE;
PNIO_DEVICE_INFO_EXT_ALARM            := PNIO_DEVICE_INFO_EXT.ALARM;
PNIO_DEVICE_INFO_EXT_ACTIVE           := PNIO_DEVICE_INFO_EXT.ACTIVE;
PNIO_DEVICE_INFO_EXT_FLAGS            := PNIO_DEVICE_INFO_EXT.FLAGS;
```

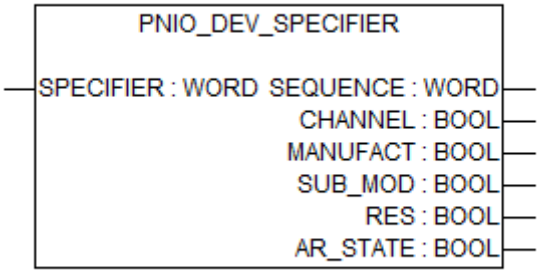
PNIO_DEV_SPECIFIER



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block without historical values
Group	Alarm

The function PNIO_DEV_SPECIFIER displays the PNIO_DEV_ALARM SPECIFIER in plain text.

Input description



SPECIFIER

Data type	Default value	Range	Unit
WORD	-	0-15	bit

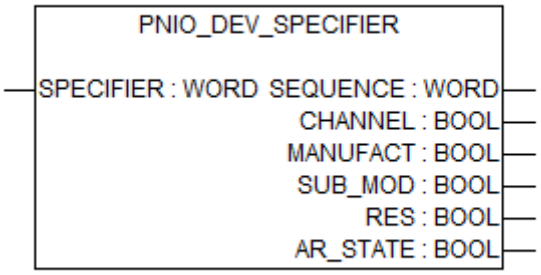
The bits of the diagnosis specifier describe the type and specification of the current diagnosis:

Bit	Description
0 ... 10	Sequence number
11	Channel diagnosis
12	Manufacturer specific diagnosis
13	Submodule diagnosis state

Bit	Description
14	Reserved
15	AR diagnosis state

The SPECIFIER can be displayed in plain text using the function PNIO_DEV_SPECIFIER
↳ *Chapter 1.5.4.27.1.3 “PNIO_DEV_INFO” on page 1803*.

Output description



SEQUENCE

Data type	Default value	Range	Unit
WORD	-	-	-

The output SEQUENCE provides the alarm sequence number.

CHANNEL

Data type	Default value	Range	Unit
BOOL	-	-	-

The output CHANNEL shows if the current alarm is a channel diagnosis. If CHANNEL=TRUE, the current alarm is a channel diagnosis.

MANUFACT

Data type	Default value	Range	Unit
BOOL	-	-	-

The output MANUFACT shows if the current alarm is a manufacturer-specific diagnosis. If MANUFACT=TRUE, the current alarm is a manufacturer-specific diagnosis.

SUB_MOD

Data type	Default value	Range	Unit
BOOL	-	-	-

The output SUB_MOD shows if the current alarm is a sub module diagnosis. If SUB_MOD=TRUE, the current alarm is a sub module diagnosis.

RES

Data type	Default value	Range	Unit
BOOL	-	-	-

The output res in not used. It is reserved for future use.

AR_STATE

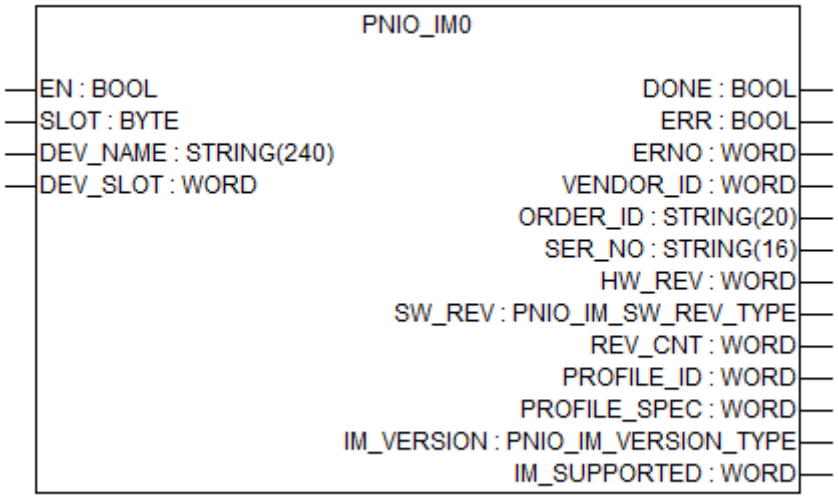
Data type	Default value	Range	Unit
BOOL	-	-	-

The output AR_STATE displays the AR diagnosis state.

Function call in ST

```
PNIO_DEV_SPECIFIER                  ( EN :=
PNIO_DEV_SPECIFIER_SPECIFIER);

PNIO_DEV_SPECIFIER_SEQUENCE             :=
PNIO_DEV_SPECIFIER.SEQUENCE;

PNIO_DEV_SPECIFIER_CHANNEL              := PNIO_DEV_SPECIFIER.CHANNEL;

PNIO_DEV_SPECIFIER_MANUFACT             :=
PNIO_DEV_SPECIFIER.MANUFACT;

PNIO_DEV_SPECIFIER_SUB_MOD              := PNIO_DEV_SPECIFIER.SUB_MOD;

PNIO_DEV_SPECIFIER_AR_STATE             :=
PNIO_DEV_SPECIFIER.AR_STATE;
```

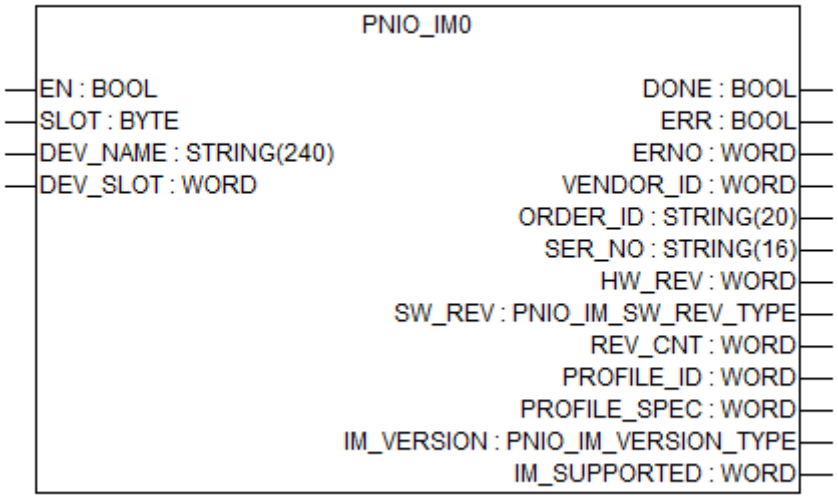
PNIO_IM0



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Info

The function block PNIO_IM0 provides Identification & Maintenance information, describing the manufacturer and the current version and revision of a certain PROFINET IO device.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

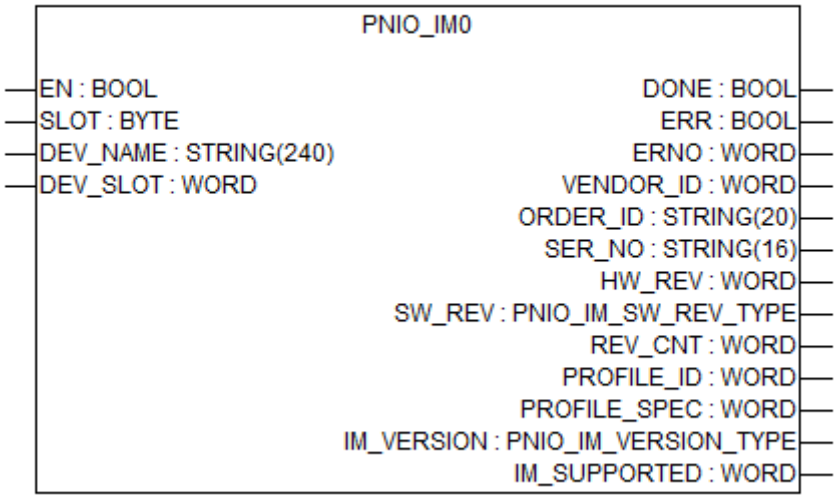
The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

VENDOR_ID

Data type	Default value	Range	Unit
WORD	0000h	-	-

The output parameter VENDOR_ID (or MANUFACTURER_ID) returns the ID of a specific manufacturer that had been assigned by the PROFIBUS/PROFINET business office. Default value is 0000h in case a manufacturer does not want to handle an ID. With this ID the manufacturer of a device can be identified from a list, hosted at and available from the PROFIBUS community. See *http://www.profibus.com* for further details.

ORDER_ID

Data type	Default value	Range	Unit
STRING	-	-	

The ORDER_ID output contains the complete order number or at least a relevant part that allows unambiguous identification of the device/module within the manufacturer's web site. It consists of a string of 20 octets, whereas unused octets are set to "20h" (blank).

SER_NO

Data type	Default value	Range	Unit
STRING	-	-	-

The SER_NO output contains an unique production number assigned by the manufacturer to this device during production. It consists of a string of 16 octets, whereas unused octets are set to "20h" (blank).

HW_REV

Data type	Default value	Range	Unit
WORD	-	-	-

The content of this parameter characterizes the edition of the hardware only. Default value is 0000h. FFFFh indicates availability of profile specific information.

SW_REV

Data type	Default value	Range	Unit
STRUCT PNIO_IM_SW_REV_TYPE	-	-	-

The content of this structure characterizes the edition of the software or firmware of a device or module. The structure supports coarse and detailed differentiation that may be defined by the manufacturer: Vx.y.z.

Any component that carries software shall present its software version even if the user may not be aware of it. V255.255.255 indicates the availability of profile specific information.

Valid values for SW_REV.REV_TYPE are:

V = officially released version

R = Revision (of a virtual or physically modular product)

P = Prototype

U = Under Test (field test)

T = Test Device

SW_REV.FUNCTIONAL (0..255) reflects the functional enhancement level

SW_REV.BUG_FIX (0..255) reflects the bug fix release level SW_REV.INTERNAL (0..255) reflects a certain internal change level

REV_CNT

Data type	Default value	Range	Unit
WORD	-	-	-

The output value of the REV_CNT parameter of a given module reflects the number of changes of the hardware or of its parameters. At production time the counter is set to 0. This value is reserved for the first installation and the first increment. Thus, the counter increments from

1... 65535, wrapping over back to 1 at the end.

PROFILE_ID

Data type	Default value	Range	Unit
WORD	-	-	-

A module following a special profile may offer extended information (PROFILE_SPEC) about its function and/or sub devices, e. g. HART. Devices not following any PROFINET application profile shall use F600h. In this case, the device is generic and may indicate its type via the parameter PROFILE_SPEC.

PROFILE_SPEC

Data type	Default value	Range	Unit
WORD	-	-	-

In case a module follows a special profile, this output parameter offers information about the usage of its channels and/or sub devices. The module/channel information is according to the respective definitions of the application profile. Devices/modules not following any PROFINET application profile are generic and are using F600h "“ F6FFh.

IM_SUPPORTED

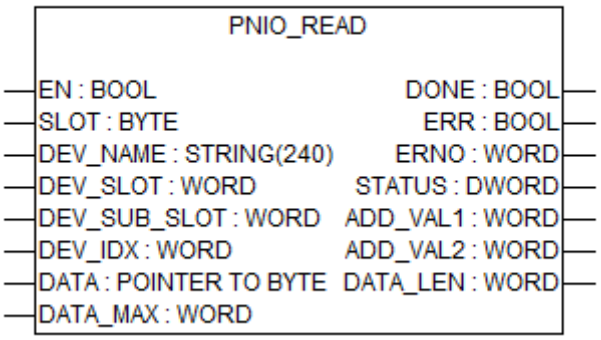
Data type	Default value	Range	Unit
DWORD	-	-	Bit

The output value of IM_SUPPORTED reflects the availability of I&M records. Each of the 16 bits of this value represents an I&M record. Starting with Bit 0, which represents the availability of a profile specific I&M information, the Bits 1...15 are counted upwards: Bit 1 equals to I&M1, Bit 2 equals to I&M2 etc. A bit set to 1 identifies an I&M being available.

Function call in ST

```
PNIO_IM(
          EN       := PNIO_IM_EN,
          SLOT     := PNIO_IM_SLOT,
          DEV_SLOT := PNIO_IM_DEV_SLOT,
          DEV_NAME := PNIO_IM_DEV_NAME);
PNIO_IM_DONE               := PNIO_IM.DONE;
PNIO_IM_ERR                := PNIO_IM.ERR;
PNIO_IM_ERNO               := PNIO_IM.ERNO;
PNIO_IM_VENDOR_ID          := PNIO_IM.VENDOR_ID;
PNIO_IM_ORDER_ID           := PNIO_IM.ORDER_ID;
PNIO_IM_SER_NO             := PNIO_IM.SER_NO;
PNIO_IM_HW_REV             := PNIO_IM.HW_REV; PNIO_IM_SW_REV
:= PNIO_IM.SW_REV; PNIO_IM_REV_CNT            := PNIO_IM.REV_CNT;
PNIO_IM_PROFILE_ID         := PNIO_IM.PROFILE_ID;
PNIO_IM_PROFILE_SPEC       := PNIO_IM.PROFILE_SPEC;
PNIO_IM_IM_VERSION         := PNIO_IM.IM_VERSION;
PNIO_IM_IM_SUPPORTED       := PNIO_IM.IM_SUPPORTED;
```

PNIO_READ

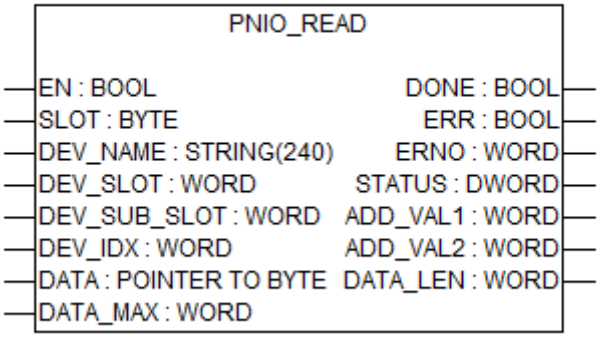


Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Data

The function block PNIO_READ implements the acyclic PROFINET READ service. Using this function, the controller has read access to slot, sub slot and index-related data of PROFINET IO devices. PNIO_READ works outside the cyclic process data exchange.

Every time a FALSE->TRUE edge is applied to input EN, PNIO _READ reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_SUB_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SUB_SLOT provides the device sub slot which sent the alarm.

DEV_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_IDX specifies the number of the index within the sub slot, the data of which shall be read.

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

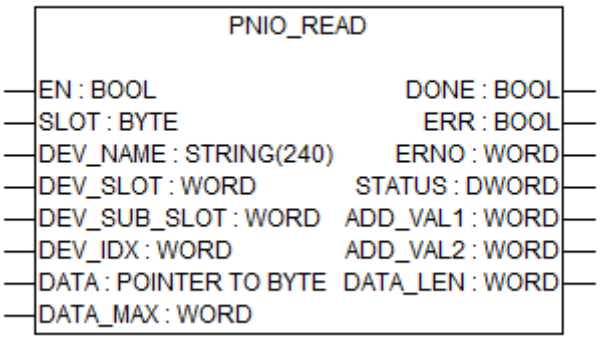
Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

DATA_MAX

Data type	Default value	Range	Unit
WORD	-	1-1024	-

Input DATA_MAX specifies the length of the data block to be read. Valid values are 1..1024.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

PNIO_STATUS contains the PNIO error code and error decode (see, *Chapter 1.5.4.27.1.13 “PROFINET status*” on page 1839*).

ADD_VAL1

Data type	Default value	Range	Unit
WORD	1	-	-

PNIO_ADD_VAL1 contains the PNIO error code 1.

ADD_VAL2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO error code 2.

DATA_LEN

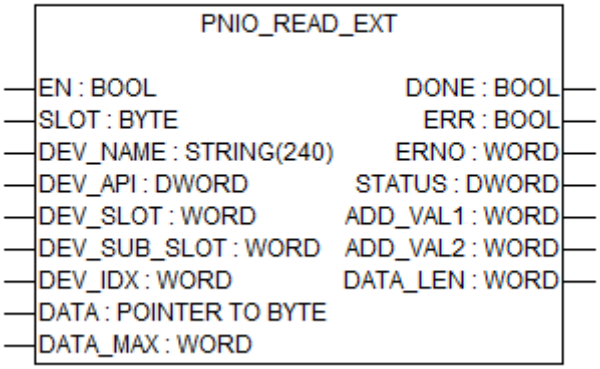
Data type	Default value	Range	Unit
DWORD	-	-	-

The output DATA_LEN provides the size of numbers / of bytes which were written to DATA.

Function call in ST

```
PNIO_READ (EN           := PNIO_READ_EN,
           SLOT         := PNIO_READ_SLOT,
           DEV_NAME     := PNIO_READ_DEV_NAME,
           DEV_SLOT     := PNIO_READ_DEV_SLOT,
           DEV_SUB_SLOT := PNIO_READ_DEV_SUB_SLOT,
           DEV_IDX      := PNIO_READ_DEV_IDX,
           DATA         := ADR(PNIO_READ_DATA),
           DATA_MAX     := PNIO_READ_DATA_MAX);
PNIO_READ_DONE     := PNIO_READ.DONE;
PNIO_READ_ERR      := PNIO_READ.ERR;
PNIO_READ_ERNO     := PNIO_READ.ERNO;
PNIO_READ_STATUS   := PNIO_READ.STATUS;
PNIO_READ_ADD_VAL1 := PNIO_READ.ADD_VAL1;
PNIO_READ_ADD_VAL2 := PNIO_READ.ADD_VAL2;
PNIO_READ_DATA_LEN := PNIO_READ.DATA_LEN;
```

PNIO_READ_EXT

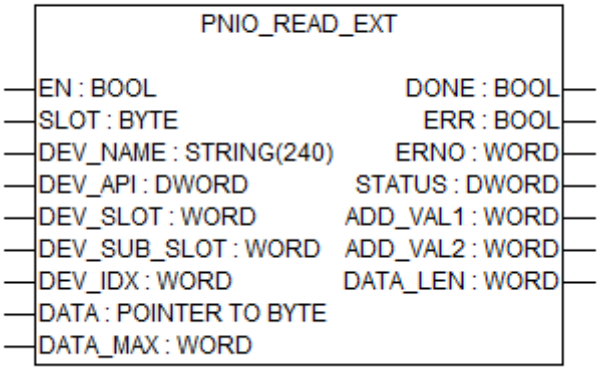


Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V2.2.0
Type	Function block with historical values
Group	Data

The function block PNIO_READ_EXT implements the acyclic PROFINET READ service. Using this function, the controller has read access to API, slot, sub slot and index-related data of PROFINET IO devices. PNIO_READ_EXT works outside the cyclic process data exchange.

Every time a FALSE->TRUE edge is applied to input EN, PNIO_READ_EXT reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

DEV_API

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_API specifies the Application Process Identifier of the AP to which the data block belongs.

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_SUB_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SUB_SLOT provides the device sub slot which sent the alarm.

DEV_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_IDX specifies the number of the index within the sub slot, the data of which shall be read.

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

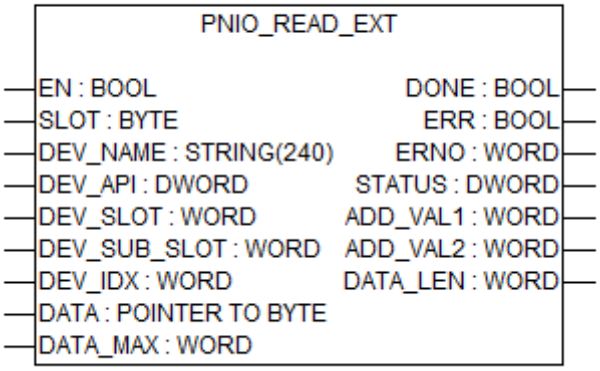
Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

DATA_MAX

Data type	Default value	Range	Unit
WORD	-	1-1024	-

Input DATA_MAX specifies the length of the data block to be read. Valid values are 1..1024.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

PNIO_STATUS contains the PNIO error code and error decode (see, *Chapter 1.5.4.27.1.13 “PROFINET status*” on page 1839*).

ADD_VAL1

Data type	Default value	Range	Unit
WORD	1	-	-

PNIO_ADD_VAL1 contains the PNIO error code 1.

ADD_VAL2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO error code 2.

DATA_LEN

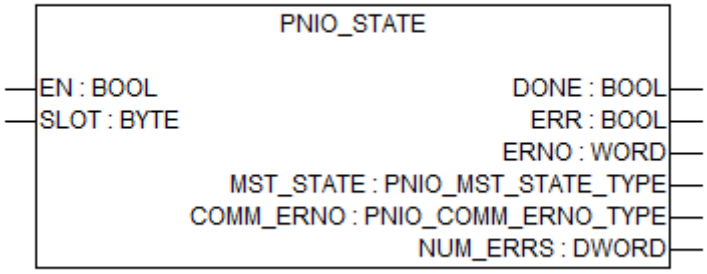
Data type	Default value	Range	Unit
DWORD	-	-	-

The output DATA_LEN provides the size of numbers / of bytes which were written to DATA.

Function call in ST

```
PNIO_READ_EXT (EN           := PNIO_READ_EXT_EN,
               SLOT         := PNIO_READ_EXT_SLOT,
               DEV_NAME     := PNIO_READ_EXT_DEV_NAME,
               DEV_API      := PNIO_READ_EXT_DEV_API,
               DEV_SLOT     := PNIO_READ_EXT_DEV_SLOT,
               DEV_SUB_SLOT := PNIO_READ_EXT_DEV_SUB_SLOT,
               DEV_IDX      := PNIO_READ_EXT_DEV_IDX,
               DATA         := ADR(PNIO_READ_EXT_DATA),
               DATA_MAX     := PNIO_READ_EXT_DATA_MAX);
PNIO_READ_EXT_DONE     := PNIO_READ_EXT.DONE;
PNIO_READ_EXT_ERR      := PNIO_READ_EXT.ERR;
PNIO_READ_EXT_ERNO     := PNIO_READ_EXT.ERNO;
PNIO_READ_EXT_STATUS   := PNIO_READ_EXT.STATUS;
PNIO_READ_EXT_ADD_VAL1 := PNIO_READ_EXT.ADD_VAL1;
PNIO_READ_EXT_ADD_VAL2 := PNIO_READ_EXT.ADD_VAL2;
PNIO_READ_EXT_DATA_LEN := PNIO_READ_EXT.DATA_LEN;
```

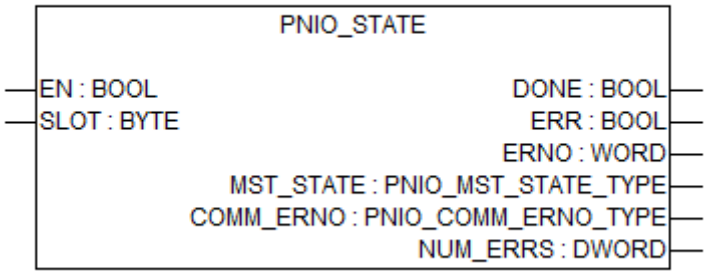
PNIO_STATE



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Info

The function block PNIO_STATE provides diagnostic and emergency information describing the current condition of a PROFINET I/O bus.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

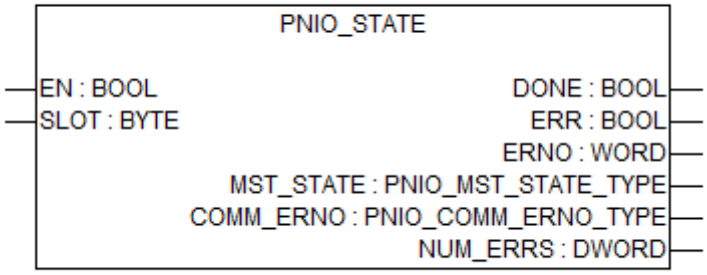
SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

MST_STATE

Data type	Default value	Range	Unit
PNIO_MST_STATE_TYPE	-	-	-

The MST_STATE output reflects the major state of the PROFINET Communication Module. This value is valid with DONE=TRUE and ERR=FALSE.

The MST_STATE is displayed in plain text with aid of the enumeration PNIO_MST_STATE_TYPE.

PNIO_MST_STATE_UNKOWN	Unknown state
PNIO_MST_STATE_NOT_CONFIGURED	Pre-configuration
PNIO_MST_STATE_STOP	Configured but not running
PNIO_MST_STATE_IDLE	Idle
PNIO_MST_STATE_OPERATE	All up, normal operational state

COMM_ERNO

Data type	Default value	Range	Unit
PNIO_COMM_ERNO_TYPE	-	-	-

The COMM_ERNO output value represents the current PROFINET bus condition. A transition of this output with DONE=TRUE to a non-zero-value signals a communication error caused by one of the reasons listed in the table below.

The COMM_ERNO is displayed in plain text with aid of the enumeration PNIO_COMM_ERNO_TYPE.

PNIO_COMM_ERNO_NONE	No error
PNIO_CONIFGURATION_FAULT	Configuration fault
PNIO_PARAMETER_ERROR	Parameter error
PNIO_INV_NETWORK_ERROR	Invalid network address
PNIO_NETWORK_FAULT	Network fault
PNIO_CONNECTION_CLOSED	Connection closed
PNIO_CONNECTION_TIMEOUT	Connection timeout
PNIO_LONELY_NETWORK	Lonely network
PNIO_DUBLICATE_NODE	Duplicate node
PNIO_CABLE_DISCONNECTED	Cable disconnected
PNIO_COMM_ERNO_UNKNOWN	Unknown communication error

NUM_ERRS

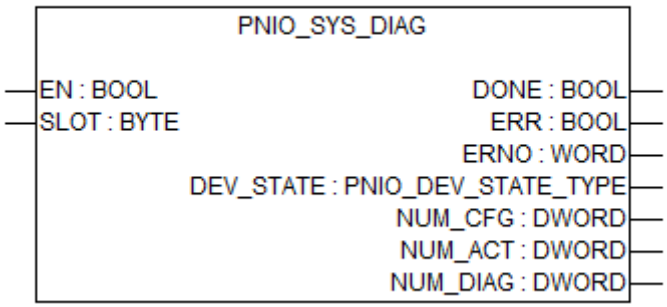
Data type	Default value	Range	Unit
DWORD	-	-	-

The total number of errors detected by the Communication Module since last power up or reset.

Function call in ST

```
PNIO_GENERAL_STATE( EN   := PNIO_GENERAL_STATE_EN,
SLOT := PNIO_GENERAL_STATE_SLOT);
PNIO_GENERAL_STATE_DONE := PNIO_GENERAL_STATE.DONE;
PNIO_GENERAL_STATE_ERR := PNIO_GENERAL_STATE.ERR;
PNIO_GENERAL_STATE_ERNO := PNIO_GENERAL_STATE.ERNO;
PNIO_GENERAL_STATE_MST_STATE := PNIO_GENERAL_STATE.MST_STATE;
PNIO_GENERAL_STATE_COMM_ERNO := PNIO_GENERAL_STATE.COMM_ERNO;
PNIO_GENERAL_STATE_NUM_ERRS := PNIO_GENERAL_STATE.NUM_ERRS;
```

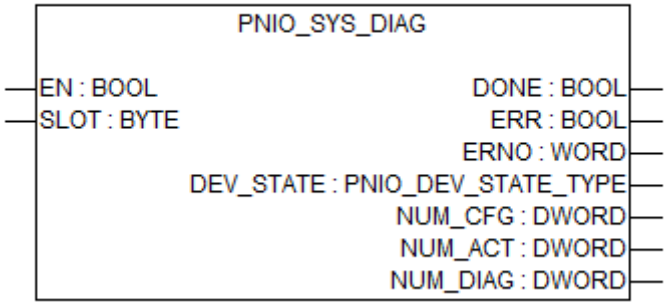
PNIO_SYS_DIAG



Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Simple function block with cycle persistent data.
Group	Info

The function block PNIO_SYS_DIAG provides diagnostic and emergency information describing the current condition of all of the PROFINET IO devices, connected to a certain PROFINET communication module.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

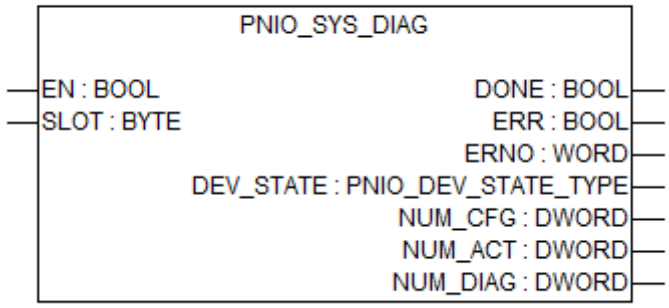
SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

DEV_STATE

Data type	Default value	Range	Unit
PNIO_DEV_STATE_TYPE	-	-	-

The output DEV_STATE indicates whether the cyclic process data exchange with all configured I/O modules is working, and no module has any diagnosis issue to be solved. On any I/O module missing or having a diagnosis issue pending, this output will change to FAILED.

After all slaves are back to normal operation, DEV_STATE will change back to OK

The DEV_STATE is displayed in plain text with aid of the enumeration PNIO_DEV_STATE_TYPE.

Value	Description
PNIO_DEV_STATE_UNDEFINED	Undefined
PNIO_DEV_STATE_OK	OK
PNIO_DEV_STATE_FAILED	Failed (at least one I/O module) or Warning (at least one I/O module)

NUM_CFG

Data type	Default value	Range	Unit
DWORD	-	-	-

The output NUM_CFG indicates the number of configured I/O modules, known by the communication module.

NUM_ACT

Data type	Default value	Range	Unit
DWORD	-	-	-

The output NUM_ACT indicates the number of currently active I/O modules, known by the communication module.

NUM_DIAG

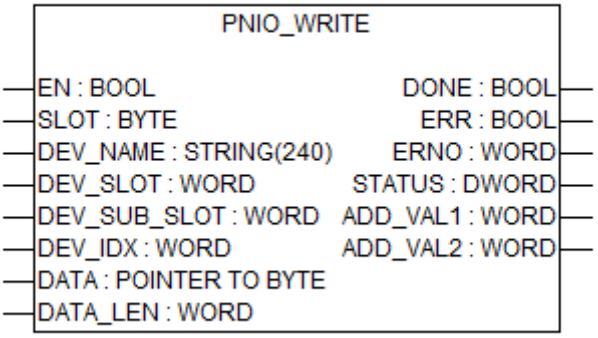
Data type	Default value	Range	Unit
DWORD	-	-	-

The output NUM_DIAG indicates the number I/O modules which have current alarms and current diagnosis information to be proceeded.

Function call in ST

```
PNIO_SYSTEM_DIAG( EN := PNIO_SYSTEM_DIAG_EN,
SLOT := PNIO_SYSTEM_DIAG_SLOT);
PNIO_SYSTEM_DIAG_DONE             := PNIO_SYSTEM_DIAG.DONE;
PNIO_SYSTEM_DIAG_ERR              := PNIO_SYSTEM_DIAG.ERR;
PNIO_SYSTEM_DIAG_ERNO             := PNIO_SYSTEM_DIAG.ERNO;
PNIO_SYSTEM_DIAG_DEV_STATE        := PNIO_SYSTEM_DIAG.DEV_STATE;
PNIO_SYSTEM_DIAG_NUM_CFG          := PNIO_SYSTEM_DIAG.NUM_CFG;
PNIO_SYSTEM_DIAG_NUM_ACT          := PNIO_SYSTEM_DIAG.NUM_ACT;
PNIO_SYSTEM_DIAG_NUM_DIAG         := PNIO_SYSTEM_DIAG.NUM_DIAG;
```

PNIO_WRITE

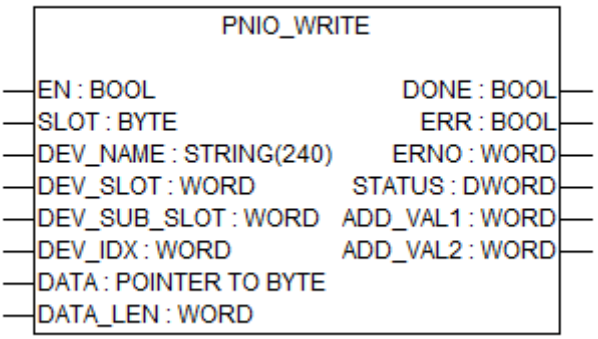


Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V1.3.0
Type	Function block with historical values
Group	Data

The function block PNIO_WRITE implements the acyclic PROFINET WRITE service. Using this function, the controller has write access to slot, sub slot and index-related data of PROFINET IO devices. PNIO_WRITE works outside the cyclic process data exchange.

Every time a FALSE->TRUE edge is applied to input EN, PNIO_WRITE reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_SUB_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SUB_SLOT provides the device sub slot which sent the alarm.

DEV_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_IDX specifies the number of the index within the sub slot, the data of which shall be read.

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

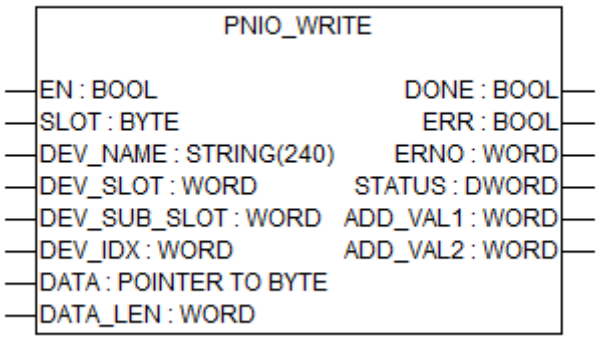
Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

DATA_LEN

Data type	Default value	Range	Unit
DWORD	-	-	-

The output DATA_LEN provides the size of numbers / of bytes which were written to DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

PNIO_STATUS contains the PNIO error code and error decode (see, *Chapter 1.5.4.27.1.13 “PROFINET status*” on page 1839*).

ADD_VAL1

Data type	Default value	Range	Unit
WORD	1	-	-

PNIO_ADD_VAL1 contains the PNIO error code 1.

ADD_VAL2

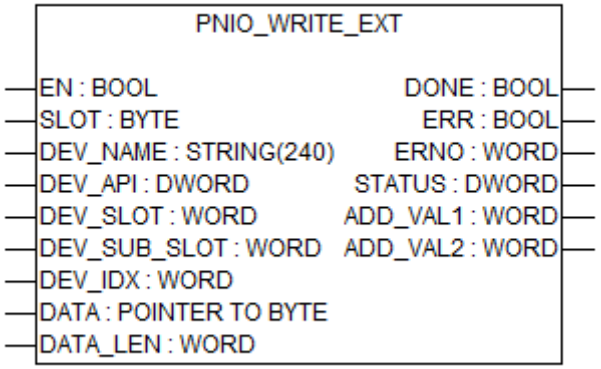
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO error code 2.

Function call in ST

```
PNIO_WRITE (EN           := PNIO_WRITE _EN,
            SLOT         := PNIO_WRITE_SLOT,
            DEV_NAME     := PNIO_WRITE_DEV_NAME,
            DEV_SLOT     := PNIO_WRITE_DEV_SLOT,
            DEV_SUB_SLOT := PNIO_WRITE_DEV_SUB_SLOT,
            DEV_IDX      := PNIO_WRITE_DEV_IDX,
            DATA         := ADR(PNIO_WRITE_DATA),
            DATA_LEN     := PNIO_WRITE_DATA_LEN);
PNIO_WRITE_DONE     := PNIO_WRITE.DONE;
PNIO_WRITE_ERR      := PNIO_WRITE.ERR;
PNIO_WRITE_ERNO     := PNIO_WRITE.ERNO;
PNIO_WRITE_STATUS   := PNIO_WRITE.STATUS;
PNIO_WRITE_ADD_VAL1 := PNIO_WRITE.ADD_VAL1;
PNIO_WRITE_ADD_VAL2 := PNIO_WRITE.ADD_VAL2;
```

PNIO_WRITE_EXT

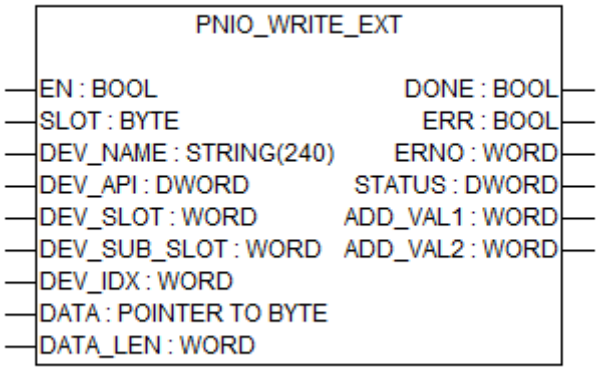


Parameter	Value
Included in library	PROFINET_AC500_V13.lib
Available as of firmware	V2.2.0
Type	Function block with historical values
Group	Data

The function block PNIO_WRITE_EXT implements the acyclic PROFINET WRITE service. Using this function, the controller has write access to API, slot, sub slot and index-related data of PROFINET IO devices. PNIO_WRITE_EXT works outside the cyclic process data exchange.

Every time a FALSE->TRUE edge is applied to input EN, PNIO_WRITE_EXT reads the data at its inputs and sends a corresponding request message to the Communication Module. Further FALSE->TRUE edges at input EN are ignored until the processing of the active requests is finished. The completion of the request processing is indicated by DONE = TRUE.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	-	1 ... 6	-

The SLOT input selects the communication module serving the PROFINET IO device described by DEV_NAME. Valid values are 1...6, counting from right to left, starting with 1 as the first communication module left to the CPU.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

The input DEV_NAME describes the PROFINET IO device to be queried. The string may have a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target IO module of this request.

Valid values are (examples for the ABB module, where the hash marks # are place holders for the address selected by the rotary switch at the front of the module):

ci501-pn-##

ci502-pn-##

DEV_API

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_API specifies the Application Process Identifier of the AP to which the data block belongs.

DEV_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SLOT provides the device slot which sent the alarm.

DEV_SUB_SLOT

Data type	Default value	Range	Unit
WORD	-	-	-

The output DEV_SUB_SLOT provides the device sub slot which sent the alarm.

DEV_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input DEV_IDX specifies the number of the index within the sub slot, the data of which shall be read.

DATA

Data type	Default value	Range	Unit
BYTE	-	-	-

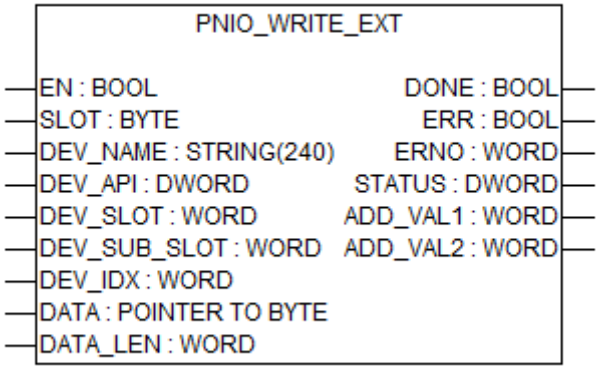
Input DATA specifies via the ADR address operator the address of the variable where the received data block shall be stored. The size of the variable must be big enough to store the complete data block (e.g. BYTE array). And the format (BYTE, WORD etc.) of the data must be considered.

DATA_LEN

Data type	Default value	Range	Unit
DWORD	-	-	-

The output DATA_LEN provides the size of numbers / of bytes which were written to DATA.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

PNIO_STATUS contains the PNIO error code and error decode (see, *Chapter 1.5.4.27.1.13 “PROFINET status*” on page 1839*).

ADD_VAL1

Data type	Default value	Range	Unit
WORD	1	-	-

PNIO_ADD_VAL1 contains the PNIO error code 1.

ADD_VAL2

Data type	Default value	Range	Unit
WORD	2	-	-

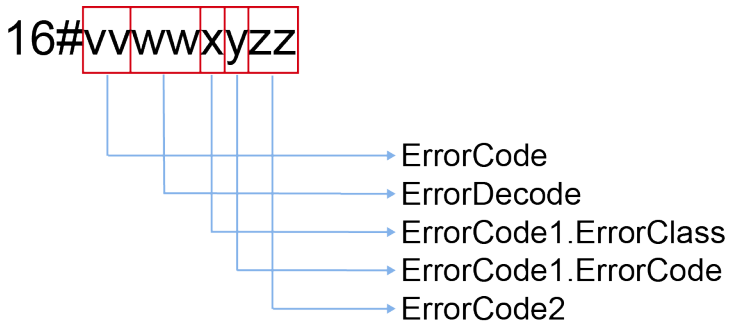
PNIO_ADD_VAL2 contains the PNIO error code 2.

Function call in ST

```
PNIO_WRITE_EXT (EN           := PNIO_WRITE_EXT_EN,
                SLOT         := PNIO_WRITE_EXT_SLOT,
                DEV_NAME     := PNIO_WRITE_EXT_DEV_NAME,
                DEV_API      := PNIO_WRITE_EXT_DEV_API,
                DEV_SLOT     := PNIO_WRITE_EXT_DEV_SLOT,
                DEV_SUB_SLOT := PNIO_WRITE_EXT_DEV_SUB_SLOT,
                DEV_IDX      := PNIO_WRITE_EXT_DEV_IDX,
                DATA         := ADR(PNIO_WRITE_EXT_DATA),
                DATA_LEN     := PNIO_WRITE_EXT_DATA_LEN);
PNIO_WRITE_EXT_DONE     := PNIO_WRITE_EXT.DONE;
PNIO_WRITE_EXT_ERR      := PNIO_WRITE_EXT.ERR; PNIO_WRITE_EXT_ERNO
:= PNIO_WRITE_EXT.ERNO;
PNIO_WRITE_EXT_STATUS   := PNIO_WRITE_EXT.STATUS;
PNIO_WRITE_EXT_ADD_VAL1 := PNIO_WRITE_EXT.ADD_VAL1;
PNIO_WRITE_EXT_ADD_VAL2 := PNIO_WRITE_EXT.ADD_VAL2;
```

PROFINET status*

The PROFINET status of read / write service is described in the following structures:



ErrorCode	
16#DE	PNIO READ Response
16#DF	PNIO WRITE Respone

ErrorDecode	
16#80	PNIO READ / WRITE Service

ErrorClass	
16#0 - 16#9	Not Specified / Reserved
16#A	Application
16#B	Access
16#C	Resource
16#D - 16#F	User / Module specific

ErrorCode inside ErrorCode1 for ErrorClass 16#A (Application)	
16#0	Read error
16#1	Write error
16#2	Module failure
16#3 - 16#6	Not specific / Reserved
16#7	Module busy
16#8	Version conflict
16#9	Feature not supported
16#A - 16#F	User / Module specific

ErrorCode inside ErrorCode1 for ErrorClass 16#B (Access)	
16#0	Invalid Index
16#1	Write lenhth error
16#2	Invalid slot / subslot
16#3	Type conflict
16#4	Invalid area / API
16#5	State conflict
16#6	Access denied
16#7	Invalid range
16#8	Invalid parameter
16#9	Invalid type
16#A	Backup
16#B - 16#F	User / Module specific

ErrorCode inside ErrorCode1 for ErrorClass 16#C (Resource)	
16#0	Read constrain confict
16#1	Write constrain confict
16#2	Resource busy
16#3	Resource unavailable
16#4 - 16#7	Not specific / Reserved
16#8 - 16#16	User / Module specific

ErrorCode2	
16#00 - 16#FF	User / Module specific

*)This chapter corresponds in substance to the standard PN-AL-protocol_2722.

1.5.4.28 Extended PROFINET IO library

Library file name: **PROFINET_Ext_AC500_Vx.lib**

The function blocks of this library provide access to the Communication Modules CI504-PNIO and CI506-PNIO and the beneath connected devices.

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

1.5.4.28.1 Function blocks

Error messages of the CI506-PNIO CANopen master

The following table defines error messages of the CI506-PNIO CANopen Master. They will be set as last error code if communication to a CANopen slave fails. The function block CI506_CANOM_NODE_DIAG provides this as last error code. This last error is also displayed in the Control Builder Plus online diagnosis.

Hexadecimal Value	Definition	Description
0x00000000	TLR_S_OK	Status ok.
0xC0000001	TLR_E_FAIL	Common error, detailed error information optionally present in the data area of packet.
0xC0420003	TLR_E_CANOPEN_MASTER_DATA_COUNT	Invalid data count.
0xC0420004	TLR_E_CANOPEN_MASTER_DATA_OFFSET	Invalid data offset.
0xC0420005	TLR_E_CANOPEN_MASTER_DATA_COUNT_WITH_OFFSET	Invalid data count in combination with offset.
0xC0420006	TLR_E_CANOPEN_MASTER_MODE	Invalid mode in command.
0xC0420007	TLR_E_CANOPEN_MASTER_STATE	Command is not allowed in current state.
0xC0420008	TLR_E_CANOPEN_MASTER_NO_VALID_BUS_PARAM	No valid bus configuration parameterized.
0xC042000A	TLR_E_CANOPEN_MASTER_BUS_RUNNING	Command is not allowed because CANopen is running.
0xC042000B	TLR_E_CANOPEN_MASTER_BUS_PARAM_ALREADY_SET	Bus parameters are already configured.
0xC042000C	TLR_E_CANOPEN_MASTER_LOCAL_NODE_ID	Invalid Node ID for CANopen Master
xC042000D	TLR_E_CANOPEN_MASTER_BAUDRATE	Invalid Baud rate.

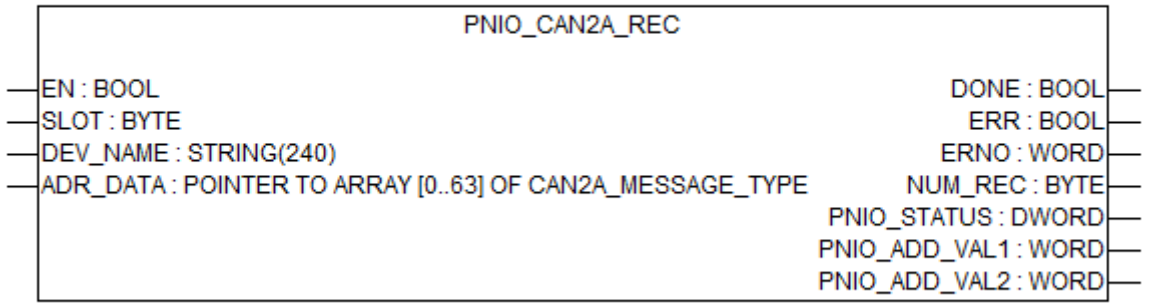
Hexadecimal Value	Definition	Description
0xC042000B	TLR_E_CANOPEN_MASTER_BUS_PARAM_ALREADY_SET	Bus parameters are already configured.
0xC042000C	TLR_E_CANOPEN_MASTER_LOCAL_NODE_ID	Invalid Node ID for CANopen Master.
xC042000D	TLR_E_CANOPEN_MASTER_BAUDRATE	Invalid Baud rate.
0xC042000E	TLR_E_CANOPEN_MASTER_29BIT_SELECTOR	Invalid parameter for 29 bit selector.
0xC042000F	TLR_E_CANOPEN_MASTER_SYNC_TIMER_VALUE	Invalid parameter for SYNC timer
0xC0420010L	TLR_E_CANOPEN_MASTER_COB_ID_SYNC	Invalid parameter for COB-ID SYNC.
0xC0420011	TLR_E_CANOPEN_MASTER_PROD_HEARTBEAT_TIME	Invalid parameter for Producer Heartbeat time.
0xC0420013	TLR_E_CANOPEN_MASTER_NODE_PARAM_SET_SIZE	Invalid size of Node parameter set.
0xC0420014	TLR_E_CANOPEN_MASTER_NODE_PARAM_HEADER_SIZE	Invalid size of Node parameter header.
0xC0420015	TLR_E_CANOPEN_MASTER_NODE_ALREADY_CONFIGURED	Node is already configured.
0xC0420016	TLR_E_CANOPEN_MASTER_SLAVE_NODE_ID	Invalid Node ID for Slave.
0xC0420017	TLR_E_CANOPEN_MASTER_NODE_ID_EQUAL	Node ID of Slave is equal to Master Node ID.
0xC0420018	TLR_E_CANOPEN_MASTER_PARAMETER_SET_LENGTH	Length of parameter set is different from length in parameter header.
0xC0420019	TLR_E_CANOPEN_MASTER_SDO_PARAMETER_SET_LENGTH	Invalid size of SDO parameter set:
0xC042001A	TLR_E_CANOPEN_MASTER_PDO_PARAMETER_SET_LENGTH	Invalid size of PDO parameter set:
0xC042001B	TLR_E_CANOPEN_MASTER_ADDR_TABLE_SET_LENGTH	Invalid size of address table
0xC042001C	TLR_E_CANOPEN_MASTER_ADDR_TABLE_LENGTH_INCONSISTENT	Address table size is inconsistent.
0xC042001E	TLR_E_CANOPEN_MASTER_TPDO_CNT	Invalid number of transmitted PDOs.
0xC042001F	TLR_E_CANOPEN_MASTER_RPDO_CNT	Invalid number of received PDOs.
0xC0420020	TLR_E_CANOPEN_MASTER_COB_ID_EMCY	Invalid value for COB-ID Emergency.
0xC0420021	TLR_E_CANOPEN_MASTER_COB_ID_GUARD	Invalid value for COB-ID Guard.
0xC0420022	TLR_E_CANOPEN_MEMORY_ALLOCATION	No memory for parameter set.
0xC0420023	TLR_E_CANOPEN_SDO_DATA_CNT	Invalid value for SDO data count.
0xC0420024	TLR_E_CANOPEN_PDO_DATA_CNT	Invalid value for PDO data count.

Hexadecimal Value	Definition	Description
0xC0420025	TLR_E_CANOPEN_ADDR_TAB_DATA_CNT	Invalid value for address table data count.
0xC0420026	TLR_E_CANOPEN_ADDR_TAB_PDO_CNT	Invalid value for address table PDO count.
0xC0420027	TLR_E_CANOPEN_MASTER_NODE_SDO_TIMEOUT	Timeout during SDO transfer.
0xC0420028	TLR_E_CANOPEN_MASTER_NODE_SDO_ERROR	Error during SDO transfer.
0xC0420029	TLR_E_CANOPEN_MASTER_NO_PDO_AVAILABLE	No further PDO available.
0xC042002A	TLR_E_CANOPEN_MASTER_AUTO_CLEAR_ACTIVE	Master is in auto clear state.
0xC042002B	TLR_E_CANOPEN_MASTER_WATCHDOG_FAIL	Watchdog failure detected.
0xC042002C	TLR_E_CANOPEN_MASTER_INVALID_INDEX	Invalid index for request.
0xC042002D	TLR_E_CANOPEN_MASTER_NODE_STATE	Request not possible in current Node state.
0xC042002F	TLR_E_CANOPEN_MASTER_SDO_REQUEST_FAILED	SDO request failed.
0x40420030	TLR_I_CANOPEN_MASTER_ALREADY_IN_STATE	Master is already in requested state.
0xC0420031	TLR_E_CANOPEN_MASTER_COB_ID_PDO	Invalid value for PDO COB-ID.
0xC0420032	TLR_E_CANOPEN_MASTER_SEND_EMCY	Send emergency-telegram failed.
0xC0420033	TLR_E_CANOPEN_MASTER_INIT_SDO_REQUEST	Failed to initialize SDO request.
0xC0420034	TLR_E_CANOPEN_MASTER_SET_NMT_STATE	Set NMT state failed.
0xC0420035	TLR_E_CANOPEN_MASTER_ERROR_PASSIVE	CANopen is in error-passive state.
0xC0420036	TLR_E_CANOPEN_MASTER_BUS_OFF	CANopen is in bus-off state.
0x40420037	TLR_I_CANOPEN_MASTER_NODE_DEACTIVATED	Node is deactivated in configuration.
0xC0420038	TLR_E_CANOPEN_MASTER_DL_REQ_FAILED	CAN-DL request failed.
0xC0420039	TLR_E_CANOPEN_MASTER_PUT_OBJECT_DATA	Failed to write object data.
0xC042003A	TLR_E_CANOPEN_MASTER_SET_OBJECT_DATA_VALID	Failed to set object data valid.
0xC042003B	TLR_E_CANOPEN_MASTER_INIT_lib	Failed to initialize CANopen library.
0xC042003C	TLR_E_CANOPEN_MASTER_SET_COB_ID_FAILED	COB-ID could not be set.
0xC042003D	TLR_E_CANOPEN_MASTER_ADD_REMOTE_NODE_REQUEST	Failed to add remote Node.

Hexadecimal Value	Definition	Description
0xC042003E	TLR_E_CANOPEN_MASTER_SET_HEARTBEAT_TIME	Heartbeat time could not be set.
0xC042003F	TLR_E_CANOPEN_MASTER_DD_GUARDING_SLAVE	Node could not be added to Node guarding list.
0xC0420040	TLR_E_CANOPEN_MASTER_SET_GUARDING_TIME	Node guard time could not be set.
0xC0420041	TLR_E_CANOPEN_MASTER_START_NODE_GUARD	Node guarding could not be started.
0xC0420042	TLR_E_CANOPEN_MASTER_RESET_NODE	Reset Node failed.
0xC0420043	TLR_E_CANOPEN_MASTER_RESET_COMMUNICATION	Failed to reset communication of Node.
0xC0420044	TLR_E_CANOPEN_MASTER_SET_NODE_PREOPERATIONAL	Failed to set Node to preoperational state.
0xC0420045	TLR_E_CANOPEN_MASTER_STOP_NODE	Failed to set Node to stop state.
0xC0420046	TLR_E_CANOPEN_MASTER_START_NODE	Failed to set Node to operational state.
0xC0420047	TLR_E_CANOPEN_MASTER_SET_EMCY_COB_ID	Failed to set Emergency COB-ID.
0xC0420048	TLR_E_CANOPEN_MASTER_START_SYNC	Failed to start SYNC-telegram.
0xC0420049	TLR_E_CANOPEN_MASTER_STOP_SYNC	Failed to stop SYNC-telegram.
0xC042004A	TLR_E_CANOPEN_MASTER_NODE_UNEXPECTED_STATE	Node is not in expected state.
0xC042004B	TLR_E_CANOPEN_MASTER_NODE_LOST_CONNECTION	Connection to Node lost.
0xC042004C	TLR_E_CANOPEN_MASTER_NODE_GUARDING_ERROR	Node guarding error.
0xC042004D	TLR_E_CANOPEN_MASTER_NODE_HEARTBEAT_ERROR	Heartbeat error.
0x4042004E	TLR_I_CANOPEN_MASTER_NODE_HEARTBEAT_STARTED	Heartbeat supervision of Node started
0xC042004F	TLR_E_CANOPEN_MASTER_NODE_UNEXPECTED_BOOTUP	Unexpected Boot up message from Node received.
0xC0420050	TLR_E_CANOPEN_MASTER_WRITE_PDO_REQ	Failed to transmit PDO.
0xC0420051	TLR_E_CANOPEN_MASTER_READ_PDO_REQ	Failed to request PDO.
0xC0420052	TLR_E_CANOPEN_MASTER_INIT_BUFFER	Initialization of buffer failed.
0x40420053	TLR_I_CANOPEN_MASTER_NODE_STATE_NOT_HANDLED	State of Node not handled.
0xC0420054	TLR_E_CANOPEN_MASTER_NODE_DEVICE_TYPE	Node Device Type unequal to configured Device Type.
0x40420055	TLR_I_CANOPEN_MASTER_NODE_EMERGENCY_RECEIVED	Emergency message received from Node

Hexadecimal Value	Definition	Description
0x40420056	TLR_I_CANOPEN_MASTER_INITIALIZE	Master is initializing.
0x40420057	TLR_I_CANOPEN_MASTER_NODE_BOOTUP	Boot up message from Node received.

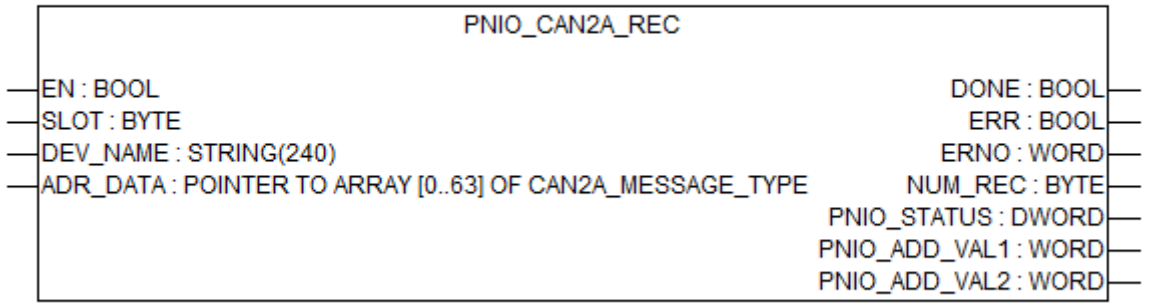
PNIO_CAN2A_REC



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CAN2A

The function block PNIO_CAN2A_REC provides all CAN2A messages received on the addressed CI506- PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

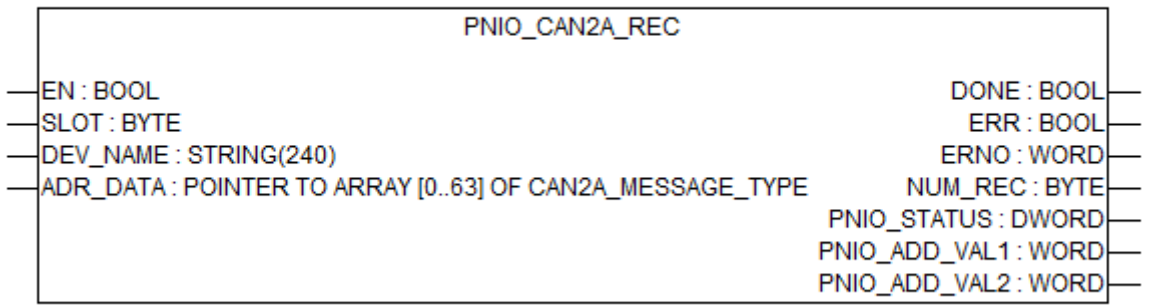
ci506-pn-##

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the total number of CAN 2.0A telegrams received.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

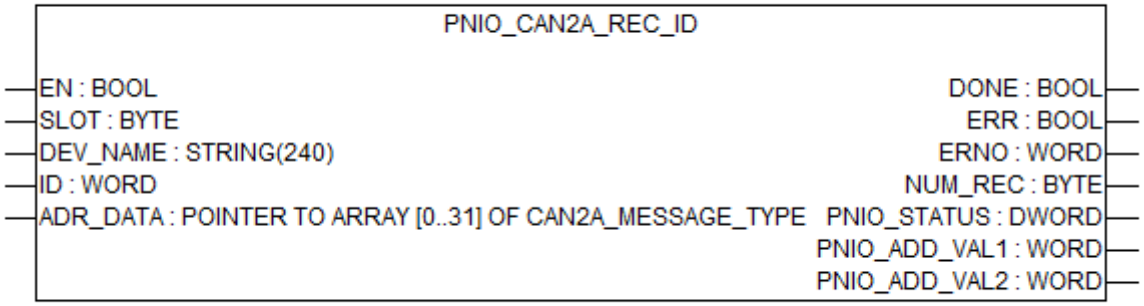
PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
REC (EN       := REC_EN,
     SLOT     := REC_SLOT,
     DEV_NAME := REC_DEV_NAME,
     ADR_DATA := REC_ADR_DATA);


REC_ERR           := REC.ERR;
REC_ERNO          := REC.ERNO;
```

```
REC_NUM_REC        := REC.NUM_REC;
REC_PNIO_STATUS    := REC.PNIO_STATUS;
REC_PNIO_ADD_VAL1  := REC.PNIO_ADD_VAL1;
REC_PNIO_ADD_VAL2  := REC.PNIO_ADD_VAL2;
REC_DONE           := REC.DONE;
```

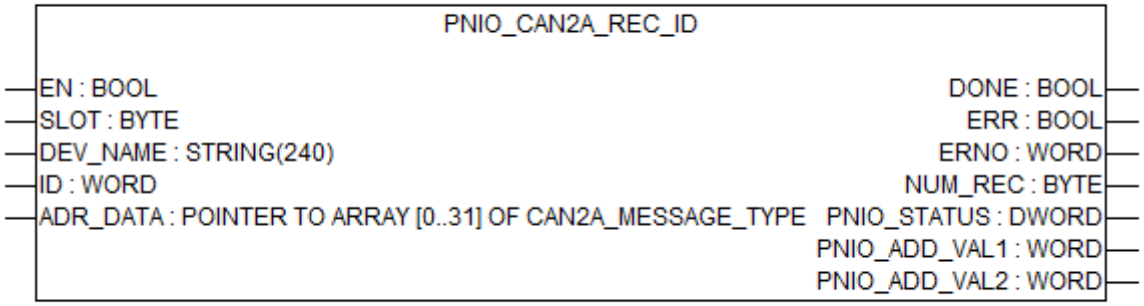
PNIO_CAN2A_REC_ID



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CAN2A

The function block PNIO_CAN2A_REC_ID provides one CAN2A message of the defined CAN identifier received on the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

ID

Data type	Default value	Range	Unit
WORD	-	-	-

Input ID specifies the identifier of the CAN 2.0A telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the function block outputs.

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the total number of CAN 2.0A telegrams received.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

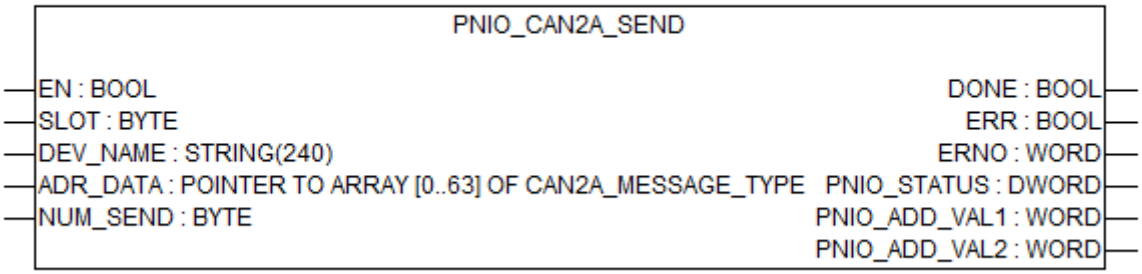
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
REC_ID(EN       := REC_ID_EN,
       SLOT     := REC_ID_SLOT,
       DEV_NAME := REC_ID_DEV_NAME,
       ID       := REC_ID_ID,
       ADR_DATA := REC_ID_ADR_DATA);


REC_ID_ERR           := REC_ID.ERR;
REC_ID_ERNO          := REC_ID.ERNO;
REC_ID_NUM_REC       := REC_ID.NUM_REC;
REC_ID_PNIO_STATUS   := REC_ID.PNIO_STATUS;
REC_ID_PNIO_ADD_VAL1 := REC_ID.PNIO_ADD_VAL1;
REC_ID_PNIO_ADD_VAL2 := REC_ID.PNIO_ADD_VAL2;
REC_ID_DONE          := REC_ID.DONE;
```

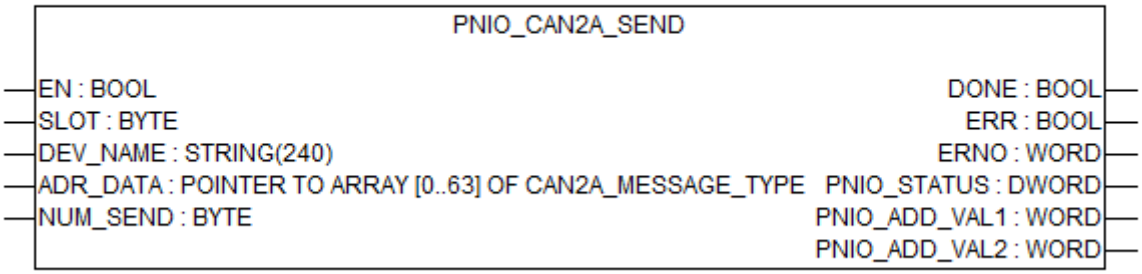
PNIO_CAN2A_SEND



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CAN2A

The function block PNIO_CAN2A_SEND sends CAN2A messages via the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

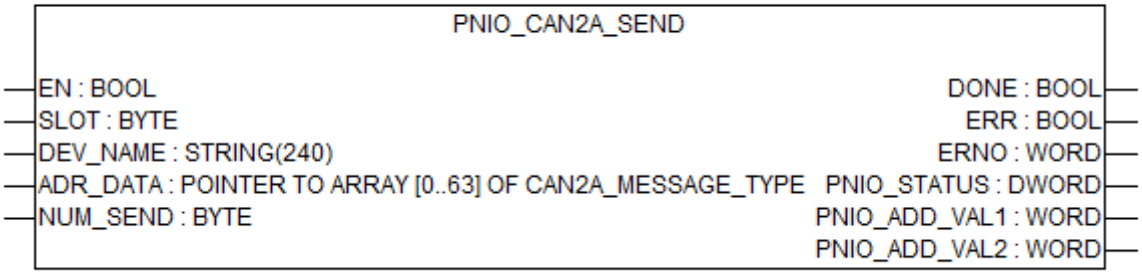
Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

NUM_SEND

Data type	Default value	Range	Unit
BYTE	-	1 - 64	-

Input NUM_SEND specifies the number of valid telegrams to be transmitted and stored starting at address ADR_DATA. The valid values for NUM_SEND are 1 to 64.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

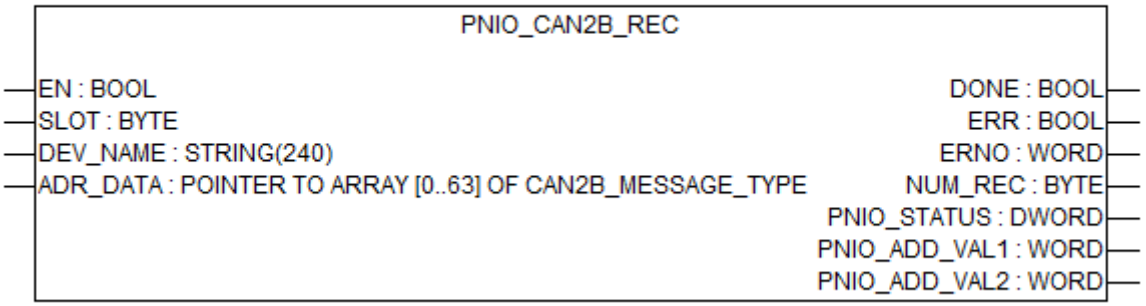
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
SEND (EN       := SEND_EN,
      SLOT     := SEND_SLOT,
      DEV_NAME := SEND_DEV_NAME,
      ADR_DATA := SEND_DEV_ADR_DATA,
      NUM_SEND := SEND_NUM_SEND);


SEND_ERR           := SEND.ERR;
SEND_ERNO          := SEND.ERNO;
SEND_STATUS        := SEND.PNIO_STATUS;
SEND_PNIO_ADD_VAL1 := SEND.PNIO_ADD_VAL1;
SEND_PNIO_ADD_VAL2 := SEND.PNIO_ADD_VAL2;
SEND_DONE          := SEND.DONE;
```

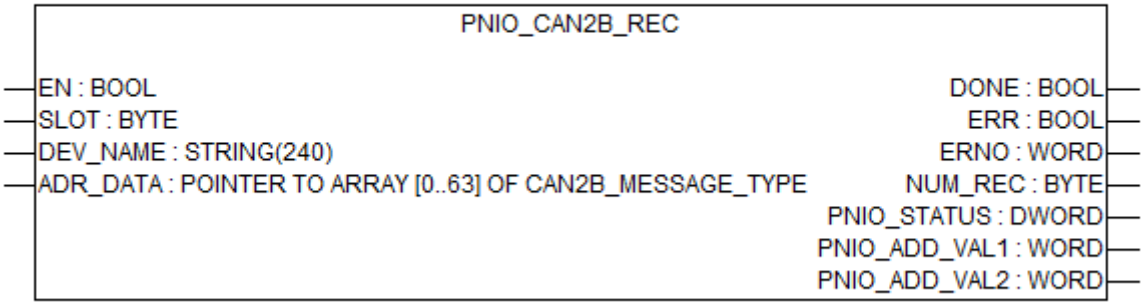
PNIO_CAN2B_REC



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	Group/Subgroup

The function block PNIO_CAN2B_REC provides all CAN2B messages received on the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

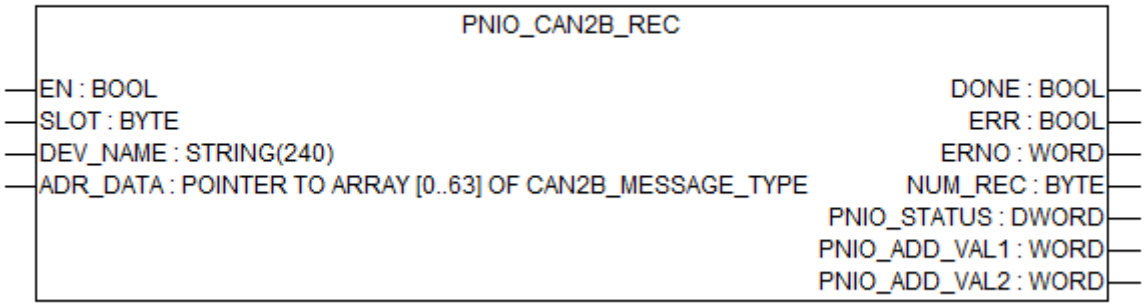
ci506-pn-##

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the total number of CAN 2.0A telegrams received.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

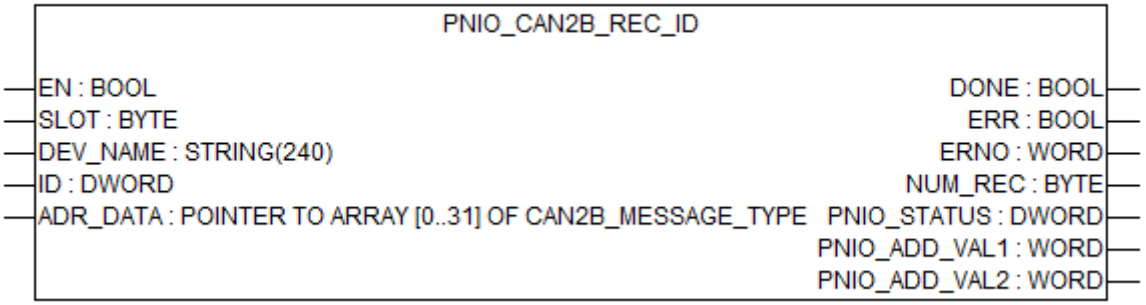
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
REC (EN       := REC_EN,
     SLOT     := REC_SLOT,
     DEV_NAME := REC_DEV_NAME,
     ADR_DATA := REC_ADR_DATA);


REC_ERR           := REC.ERR;
REC_ERNO          := REC.ERNO;
REC_NUM_REC       := REC.NUM_REC;
REC_PNIO_STATUS   := REC.PNIO_STATUS;
REC_PNIO_ADD_VAL1 := REC.PNIO_ADD_VAL1;
REC_PNIO_ADD_VAL2 := REC.PNIO_ADD_VAL2;
REC_DONE          := REC.DONE;
```

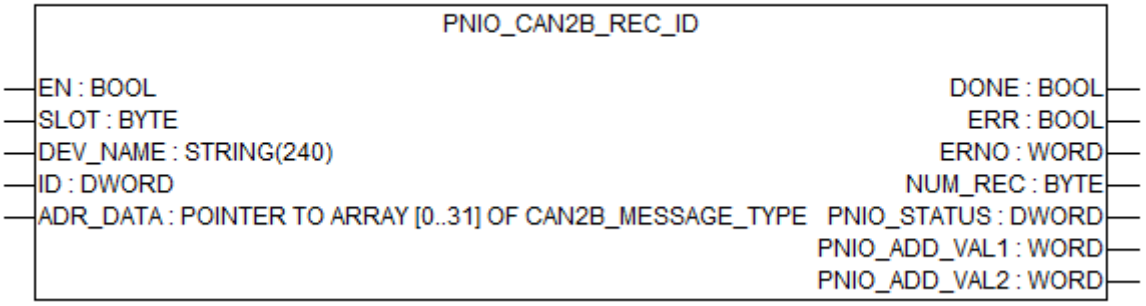
PNIO_CAN2B_REC_ID



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CAN2B

The function block PNIO_CAN2B_REC_ID provides one CAN2B message of the defined CAN identifier received on the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

ID

Data type	Default value	Range	Unit
WORD	-	-	-

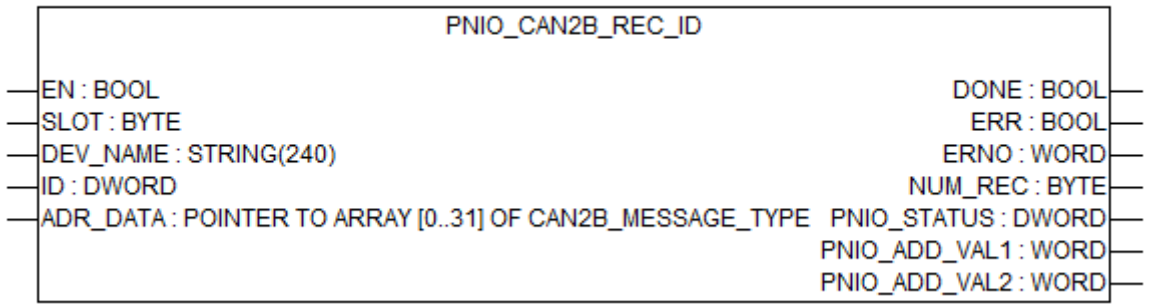
Input ID specifies the identifier of the CAN 2.0A telegrams to be read from the buffer. If no buffer has been specified for the selected identifier using the controller configuration, this is indicated accordingly at the function block outputs.

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the total number of CAN 2.0A telegrams received.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

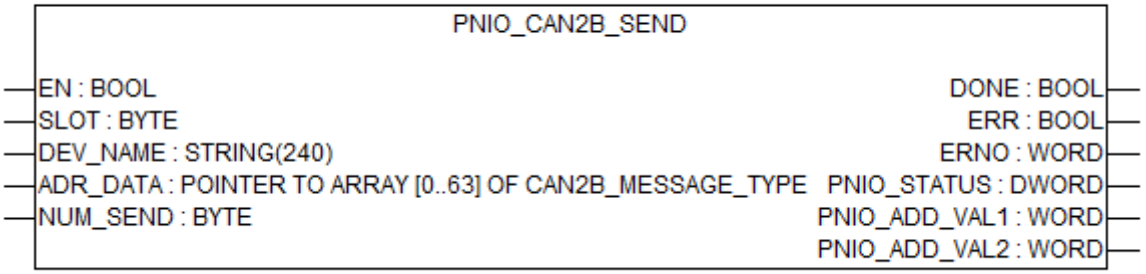
PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
REC_ID(EN       := REC_ID_EN,
       SLOT     := REC_ID_SLOT,
       DEV_NAME := REC_ID_DEV_NAME,
       ID       := REC_ID_ID,
       ADR_DATA := REC_ID_ADR_DATA);


REC_ID_ERR           := REC_ID.ERR;
REC_ID_ERNO          := REC_ID.ERNO;
```

```
REC_ID_NUM_REC        := REC_ID.NUM_REC;
REC_ID_PNIO_STATUS    := REC_ID.PNIO_STATUS;
REC_ID_PNIO_ADD_VAL1  := REC_ID.PNIO_ADD_VAL1;
REC_ID_PNIO_ADD_VAL2  := REC_ID.PNIO_ADD_VAL2;
REC_ID_DONE           := REC_ID.DONE;
```

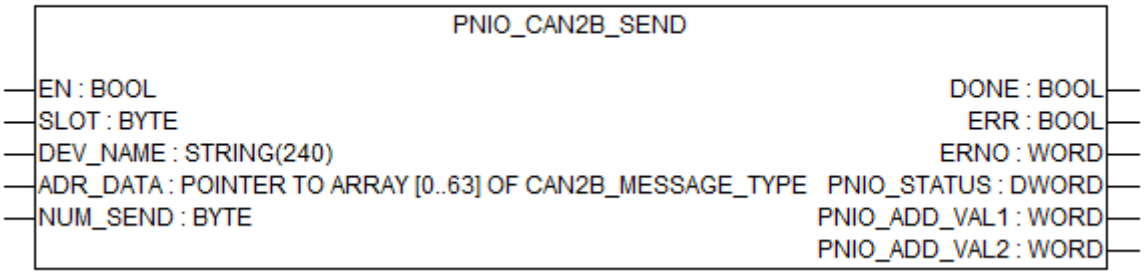
PNIO_CAN2B_SEND



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CAN2B

The function block PNIO_CAN2B_SEND sends CAN2B messages via the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

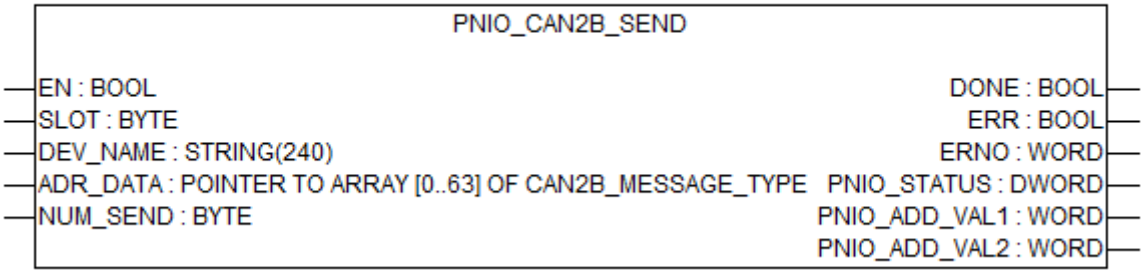
Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

NUM_SEND

Data type	Default value	Range	Unit
BYTE	-	1 - 64	-

Input NUM_SEND specifies the number of valid telegrams to be transmitted and stored starting at address ADR_DATA. The valid values for NUM_SEND are 1 to 64.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

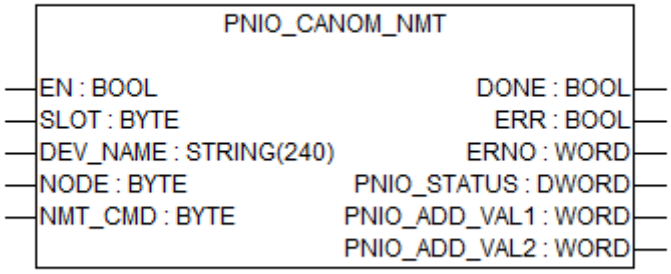
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
SEND (EN       := SEND_EN,
      SLOT     := SEND_SLOT,
      DEV_NAME := SEND_DEV_NAME,
      ADR_DATA := SEND_DEV_ADR_DATA,
      NUM_SEND := SEND_NUM_SEND);


SEND_ERR           := SEND.ERR;
SEND_ERNO          := SEND.ERNO;
SEND_STATUS        := SEND.PNIO_STATUS;
SEND_PNIO_ADD_VAL1 := SEND.PNIO_ADD_VAL1;
SEND_PNIO_ADD_VAL2 := SEND.PNIO_ADD_VAL2;
SEND_DONE          := SEND.DONE;
```

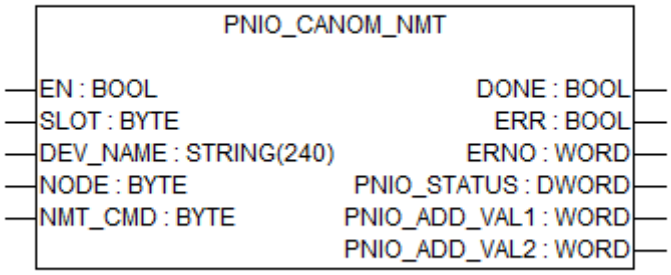
PNIO_CANOM_NMT



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_NMT sends a CANopen Network Management (NMT) command to a CANopen slave that is connected to the CI506-PNIO CANopen master interface.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input NODE describes the target CANopen slave node ID for which the NMT command is intended.

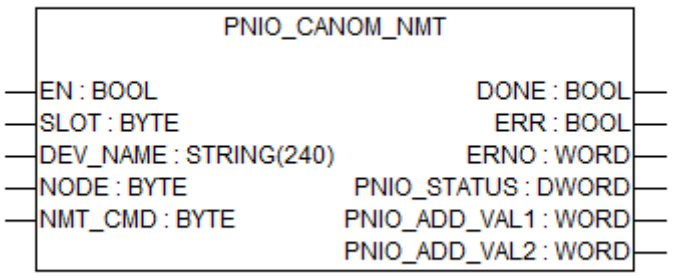
NMT_CMD

Data type	Default value	Range	Unit
BYTE	-	-	-

Input NMT_CMD describes the type of the NMT command to be transmitted to the CANopen slave. The range of values of node ID is as follows:

NMT Command	NMT_CMD value in hex	NMT_CMD value in decimal
Start remote node	16#01	1
Stop remote node	16#02	2
Enter pre-operational	16#80	128
Reset remote node	16#81	129

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

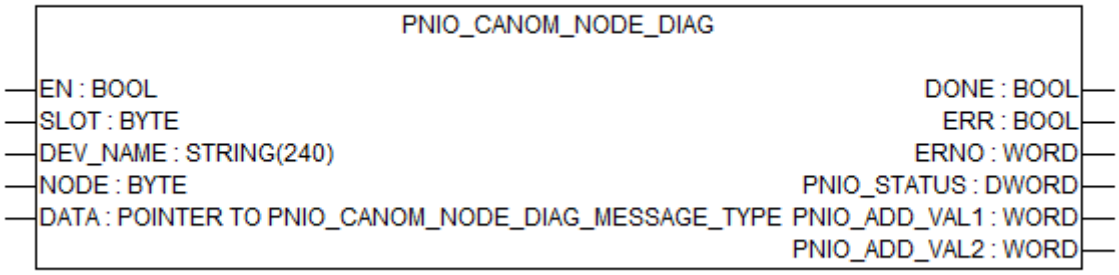
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_CANOM_NMT(EN        := NMT_EN,
               SLOT      := NMT_SLOT,
               DEV_NAME  := NMT_DEV_NAME,
               NODE      := NMT_NODE,
               NMT_CMD   := NMT_CMD);


NMT_ERR          := PNIO_CANOM_NMT.ERR;
NMT_ERNO         := PNIO_CANOM_NMT.ERNO;
NMT_PNIO_STATUS  := PNIO_CANOM_NMT.PNIO_STATUS;
NMT_ADD_VAL1     := PNIO_CANOM_NMT.PNIO_ADD_VAL1;
NMT_ADD_VAL2     := PNIO_CANOM_NMT.PNIO_ADD_VAL2;
NMT_DONE         := PNIO_CANOM_NMT.DONE;
```

PNIO_CANOM_NODE_DIAG



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_NODE_DIAG reads the diagnosis data from a CANopen slave that is connected to the CANopen master of the addressed CI506-PNIO.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

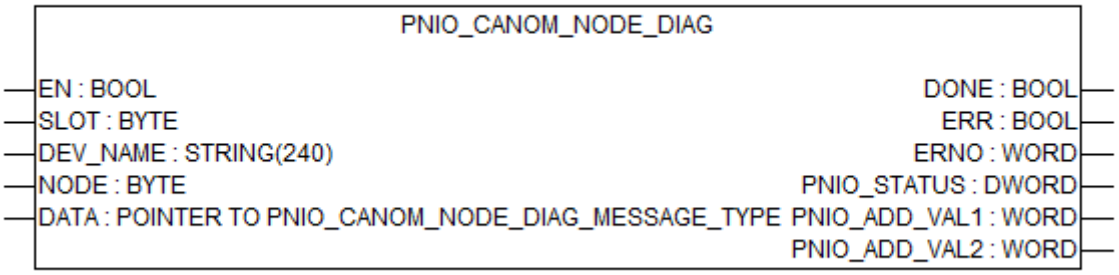
Input NODE describes the target CANopen slave node ID for which the NMT command is intended.

DATA

Data type	Default value	Range	Unit
DWORD POINTER	-	-	-

Input DATA specifies the address of the variable containing the telegrams to be transmitted. This specification is usually done via the ADR operator. The node diagnostic data have to be of the data type PNIO_CANOM_NODE_DIAG_MESSAGE_TYPE defined in the library.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

PNIO_CANOM_ NODE_DIAG_M ESSAGE_TYPE

Data type	Default value	Range	Unit
-	-	-	-

The CANopen slave node diagnostic can be displayed properly by using the structure PNIO_CANOM_NODE_DIAG_MESSAGE_TYPE which is declared as follows.

TYPE PNIO_CANOM_NODE_DIAG_MESSAGE_TYPE:

STRUCT

NODE_FLAG: DWORD;

LAST_DIAG: DWORD;

DEV_TYPE_VALID: DWORD;

DEV_TYPE: DWORD;

NMT_STATE: DWORD;

EMCY_CNT: DWORD;

EMCY_DATA: ARRAY [1..5] OF CANOM_EMCY_TYPE;

ADD_INFO: DWORD;

RESERVED : ARRAY [1..4] OF DWORD;

END_STRUCT

END_TYPE

NODE_FLAG

Data type	Default value	Range	Unit
DWORD	-	0 - 31	-

Each bit in this variable describes an occurrence of a certain event for the node. The bit organization is displayed in the following table:

Bit	Name	Description
31	FLAG_DEACTIVATED	Node is deactivated and not handled by the master. This bit does not generate an entry in the diagnostic list.
30	FLAG_STATE_NOT_HANDLED	At least one state has been omitted during the initialization sequence of the node. This bit does not generate an entry in the diagnostic list.
13..29	Reserved	Reserved
12	FLAG_INVALID_PARAMETER	Parameter set of node is invalid.
11	FLAG_UNEXPECTED_BOOTUP	Unexpected boot-up message from node received.
10	FLAG_BOOTUP	Expected boot-up message from node received.
9	FLAG_EMCY_BUFF_OVER	Emergency buffer overflow
8	FLAG_EMCY_RECEIVED	Emergency telegram received
7	FLAG_UNEXPECTED_STATE	Node is in unexpected NMT state
6	FLAG_HEARTBEAT_ERROR	Error in heartbeat protocol
5	FLAG_CON_LOST	Node guarding has been lost
4	FLAG_GUARD_ERROR	A guarding message has been lost. This bit does not generate an entry in the diagnostic list.
3	FLAG_HEARTBEAT_STARTED	Heartbeat protocol started. This bit does not generate an entry in the diagnostic list.
2	FLAG_CFG_FAULT	Configuration fault
1	FLAG_SDO_ERROR	Error during SDO transfer
0	FLAG_SDO_TIMEOUT	Timeout during SDO transfer

LAST_DIAG

Data type	Default value	Range	Unit
DWORD	-	-	-

Last error code recorded by the CANopen master for the slave. See 'Error Messages of the CI506 CANopen Master' *Chapter 1.5.4.28.1.1 “Error messages of the CI506-PNIO CANopen master” on page 1841*.

DEV_TYPE_VALID

Data type	Default value	Range	Unit
DWORD	-	-	-

DEV_TYPE_VALID describes the validity of the device type displayed in the variable DEV_TYPE.

If DEV_TYPE_VALID = 1, the device type is valid.

If DEV_TYPE_VALID = 0, the device type is invalid.

DEV_TYPE

Data type	Default value	Range	Unit
DWORD	-	-	-

DEV_TYPE describes the device type read from the CANopen slaves upon node start-up. The values are predefined in the CANopen specification. Example for device type values are:

Device Profile for I/O modules: 401 decimal

Device Profile for Drives: 402 decimal

Device Profile Encoder modules: 406 decimal

NMT_STATE

Data type	Default value	Range	Unit
DWORD	-	0 - 6	-

NMT_STATE describes the current CANopen slave NMT state. The following table describes the possible value of the NMT_STATE.

NMT State	Value
Unknown	0
Initializing	1
Stopped	2
Operational	3
Pre-operational	4
Reset application	5
Reset communication	6

EMCY_CNT

Data type	Default value	Range	Unit
DWORD	-	-	-

EMCY_CNT outputs the number of valid emergency messages of the slave output at EMCY_DATA according to the CANopen specification. Up to 5 emergency messages per slave can be buffered in the CANopen master.

EMCY_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

EMCY_DATA outputs up to 5 buffered emergency messages of the slave. The number of valid messages is output by NUM_EMCY. The structure of the type CANOM_EMCY_TYPE is defined in the AC500 CANopen library.

ADD_INFO

Data type	Default value	Range	Unit
DWORD	-	-	-

Not used.

RESERVED

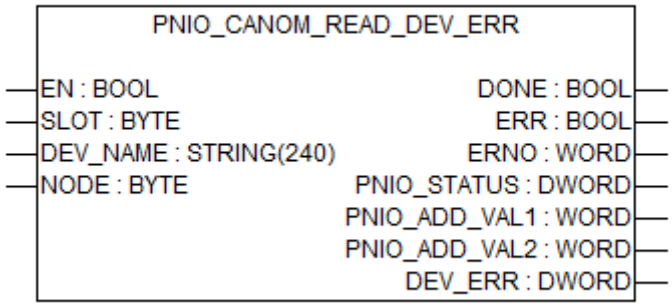
Data type	Default value	Range	Unit
DWORD	-	-	-

Reserved and not used.

Function call in ST

```
PNIO_CANOM_NODE_DIAG(EN       := DIAG_EN,
                     SLOT     := DIAG_SLOT,
                     DEV_NAME := DIAG_DEV_NAME,
                     NODE     := DIAG_NODE,
                     DATA     := DIAG_DATA);


DIAG_ERR           := PNIO_CANOM_NODE_DIAG.ERR;
DIAG_ERRNO         := PNIO_CANOM_NODE_DIAG.ERNO;
DIAG_STATUS        := PNIO_CANOM_NODE_DIAG.PNIO_STATUS;
DIAG_PNIO_ADD_VAL1 := PNIO_CANOM_NODE_DIAG.PNIO_ADD_VAL1;
DIAG_PNIO_ADD_VAL2 := PNIO_CANOM_NODE_DIAG.PNIO_ADD_VAL2;
DIAG_DONE          := PNIO_CANOM_NODE_DIAG.DONE;
```

PNIO_CANOM_READ_DEV_ERR



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_READ_DEV_ERR reads the CANopen slave device error object (index 16#1001) of a CANopen slave connected to the CANopen master of the addressed CI506-PNIO.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

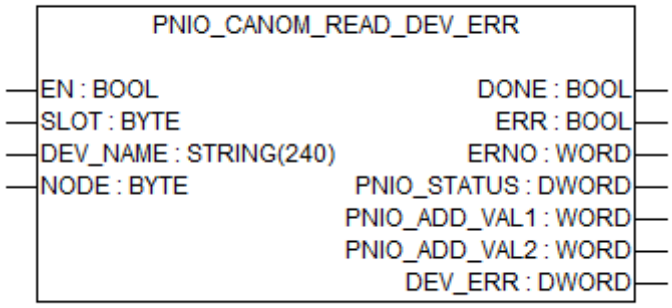
ci506-pn-##

NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input NODE describes the target CANopen slave node ID for which the NMT command is intended.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

DEV_ERR

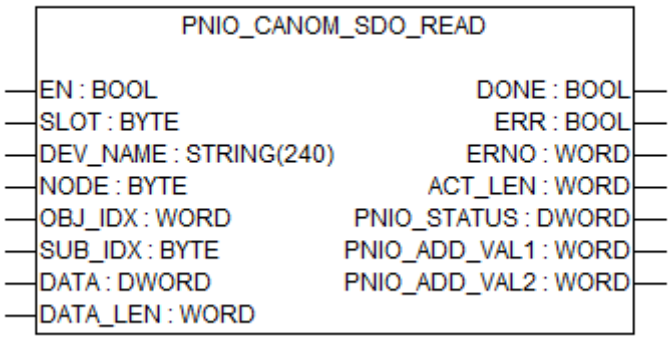
Data type	Default value	Range	Unit
DWORD	-	-	-

Value of the CANopen slave device error object (index 16#1001).

Function call in ST

```
PNIO_CANOM_READ_DEV_ERR(EN       := DEV_ERR_EN,
                        SLOT     := DEV_ERR_SLOT,
                        DEV_NAME := DEV_ERR_DEV_NAME,
                        NODE     := DEV_ERR_NODE);


DEV_ERR_ERR           := PNIO_CANOM_READ_DEV_ERR.ERR;
DEV_ERR_ERNO          := PNIO_CANOM_READ_DEV_ERR.ERNO;
DEV_ERR_PNIO_STATUS   := PNIO_CANOM_READ_DEV_ERR.PNIO_STATUS;
DEV_ERR_PNIO_ADD_VAL1 := PNIO_CANOM_READ_DEV_ERR.PNIO_ADD_VAL1;
DEV_ERR_PNIO_ADD_VAL2 := PNIO_CANOM_READ_DEV_ERR.PNIO_ADD_VAL2;
DEV_ERR_DEV_ERR       := PNIO_CANOM_READ_DEV_ERR.DEV_ERR;
DEV_ERR_DONE          := PNIO_CANOM_READ_DEV_ERR.DONE;
```

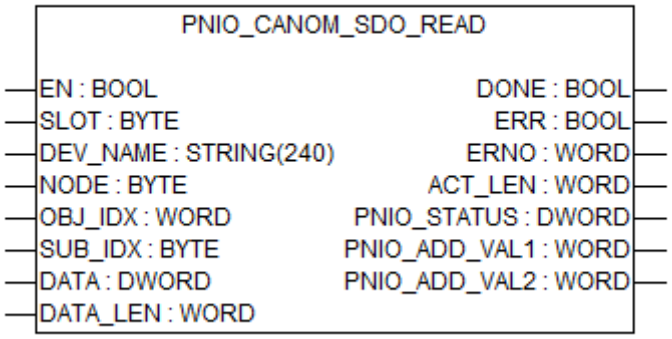
PNIO_CANOM_SDO_READ



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_SDO_READ reads "service data object" (SDO) of a CANopen slave connected to the CANopen master of the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input NODE describes the target CANopen slave node ID for which the NMT command is intended.

OBJ_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input OBJ_IDX describes the index of the CANopen object in the slave dictionary (refer to the CANopen slave EDS file) to be read.

SUB_IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

Input OBJ_IDX describes the sub-index of the CANopen object in the slave dictionary (refer to the CANopen slave EDS file) to be read.

DATA

Data type	Default value	Range	Unit
DWORD POINTER	-	-	-

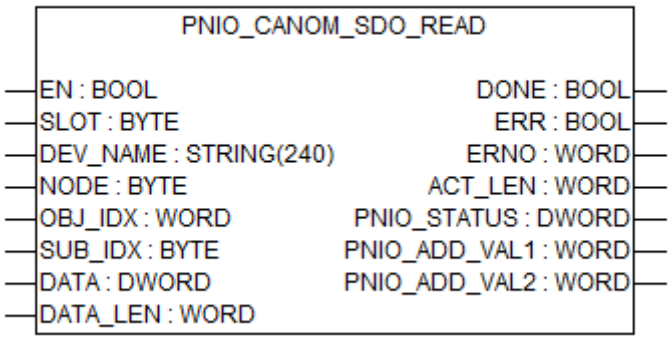
Input DATA specifies the address of the variable containing the telegrams to be transmitted. This specification is usually done via the ADR operator. The node diagnostic data have to be of the data type PNIO_CANOM_NODE_DIAG_MESSAGE_TYPE defined in the library.

DATA_LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Input DATA_LEN describes the maximum length of the SDO data to be uploaded in bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT_LEN

Data type	Default value	Range	Unit
WORD	-	-	-

ACT_LEN outputs the length of the received object data (in bytes), after the procedure has been completed successfully. Since function block execution requires bus access, the data are available in the next cycle after activating the function block at the earliest. The output value is only valid, if DONE = TRUE and ERR = FALSE.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

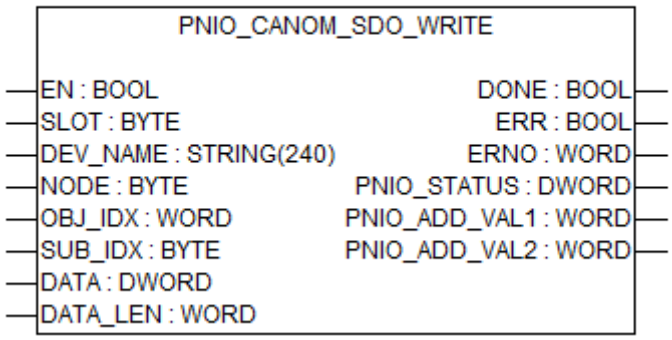
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_CANCOM_SDO_READ(EN       := SDO_READ_EN,
                     SLOT     := SDO_READ_SLOT,
                     DEV_NAME := SDO_READ_DEV_NAME,
                     NODE     := SDO_READ_NODE,
                     OBJ_IDX  := SDO_READ_OBJ_IDX,
                     SUB_IDX  := SDO_READ_SUB_IDX,
                     DATA     := SDO_READ_DATA,
                     DATA_LEN := SDO_READ_DATA_LEN);


SDO_RED_ERR             := PNIO_CANCOM_SDO_READ.ERR;
SDO_READ_ERNO           := PNIO_CANCOM_SDO_READ.ERNO;
SDO_READ_ACT_LEN        := PNIO_CANCOM_SDO_READ.ACT_LEN;
SDO_READ_PNIO_STATUS    := PNIO_CANCOM_SDO_READ.PNIO_STATUS;
SDO_READ_PNIO_ADD_VAL1  := PNIO_CANCOM_SDO_READ.PNIO_ADD_VAL1;
SDO_READ_PNIO_ADD_VAL2  := PNIO_CANCOM_SDO_READ.PNIO_ADD_VAL2;
SDO_READ_DONE           := PNIO_CANCOM_SDO_READ.DONE;
```

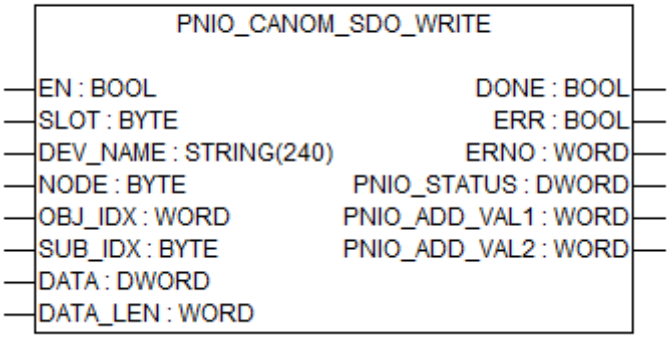
PNIO_CANOM_SDO_WRITE



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_SDO_WRITE writes a "service data object" (SDO) of a CANopen slave connected to the CANopen master of the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input NODE describes the target CANopen slave node ID for which the NMT command is intended.

OBJ_IDX

Data type	Default value	Range	Unit
WORD	-	-	-

Input OBJ_IDX describes the index of the CANopen object in the slave dictionary (refer to the CANopen slave EDS file) to be read.

SUB_IDX

Data type	Default value	Range	Unit
BYTE	-	-	-

Input OBJ_IDX describes the sub-index of the CANopen object in the slave dictionary (refer to the CANopen slave EDS file) to be read.

DATA

Data type	Default value	Range	Unit
DWORD POINTER	-	-	-

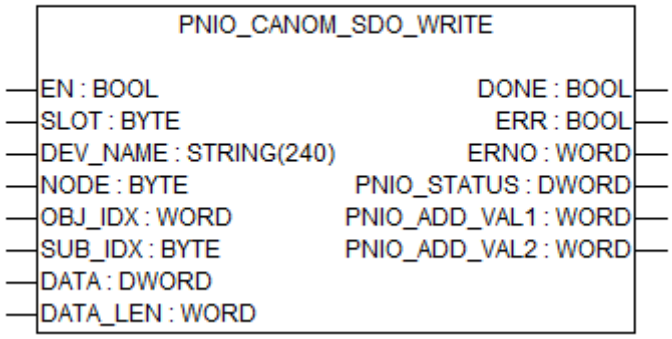
Input DATA specifies the address of the variable containing the telegrams to be transmitted. This specification is usually done via the ADR operator. The node diagnostic data have to be of the data type PNIO_CANOM_NODE_DIAG_MESSAGE_TYPE defined in the library.

DATA_LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Input DATA_LEN describes the maximum length of the SDO data to be uploaded in bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

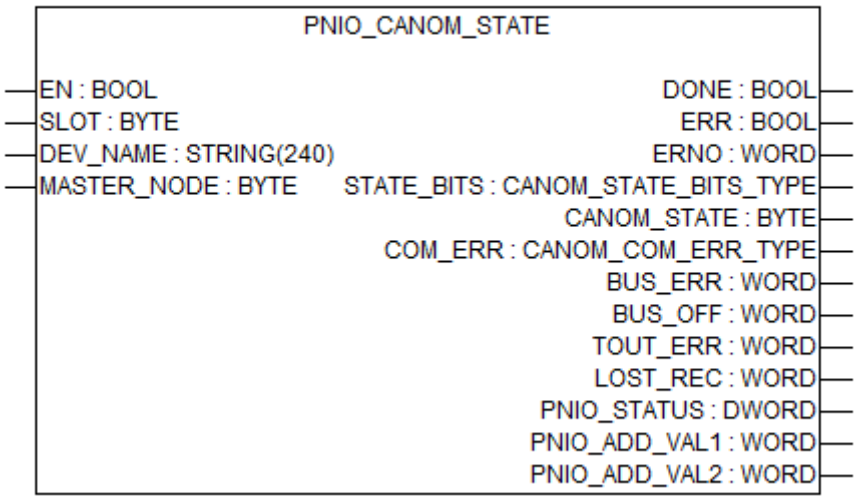
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_CANCOM_SDO_WRITE(EN       := SDO_WRITE_EN,
                      SLOT     := SDO_WRITE_SLOT,
                      DEV_NAME := SDO_WRITE_DEV_NAME,
                      NODE     := SDO_WRITE_NODE,
                      OBJ_IDX  := SDO_WRITE_OBJ_IDX,
                      SUB_IDX  := SDO_WRITE_SUB_IDX,
                      DATA     := SDO_WRITE_DATA,
                      DATA_LEN := SDO_WRITE_DATA_LEN);


SDO_WRITE_ERR           := PNIO_CANCOM_SDO_WRITE.ERR;
SDO_WRITE_ERNO          := PNIO_CANCOM_SDO_WRITE.ERNO;
SDO_WRITE_PNIO_STATUS   := PNIO_CANCOM_SDO_WRITE.PNIO_STATUS;
SDO_WRITE_PNIO_ADD_VAL1 := PNIO_CANCOM_SDO_WRITE.PNIO_ADD_VAL1;
SDO_WRITE_PNIO_ADD_VAL2 := PNIO_CANCOM_SDO_WRITE.PNIO_ADD_VAL2;
SDO_WRITE_DONE          := PNIO_CANCOM_SDO_WRITE.DONE;
```

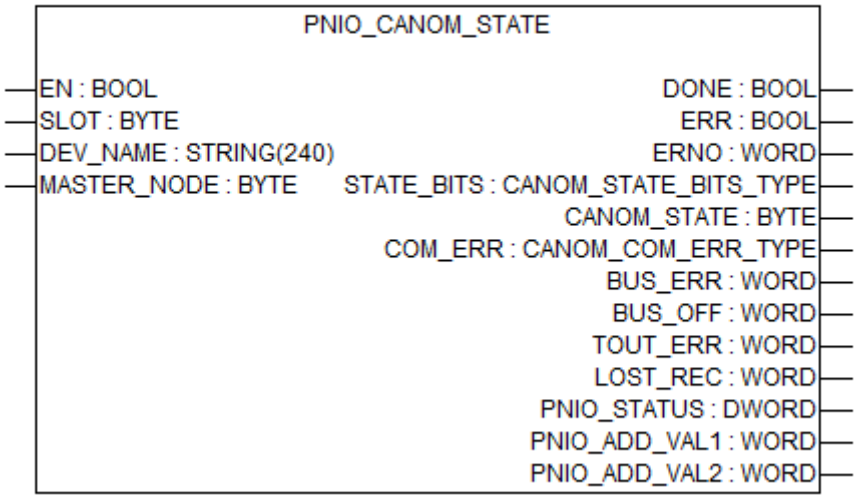
PNIO_CANOM_STATE



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_STATE provides state information of the CANopen master of the addressed CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

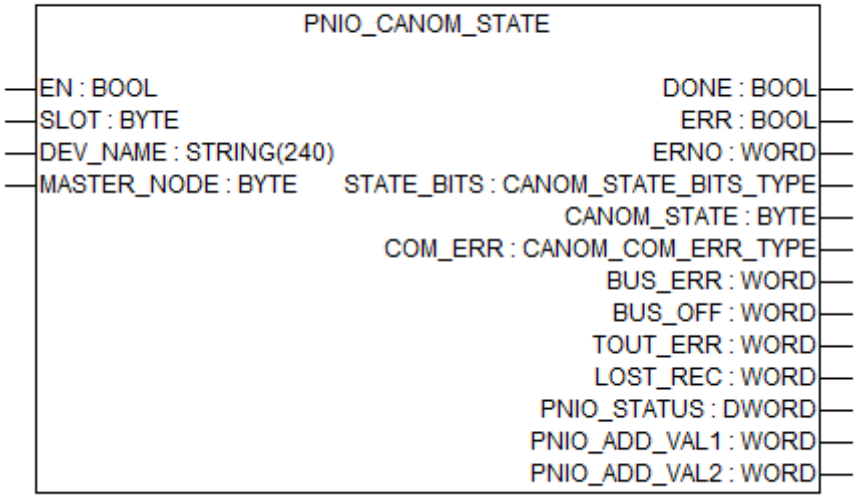
ci506-pn-##

MASTER_NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input MASTER_NODE describes the CANopen master node ID in CI506.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATE_BITS

Data type	Default value	Range	Unit
CANOM_STATE_BITS_TYPE	-	-	-

Output STATE_BITS indicates a typical communication states of the CANopen master. STATE_BITS is only valid if EN = TRUE and ERR = FALSE.

CANOM_STATE

Data type	Default value	Range	Unit
BYTE	-	-	-

CANOM_STATE outputs the general communication state of the CANopen master in CI506. CANOM_STATE is only valid, if EN = TRUE and ERR = FALSE. The following states are defined:

State		Meaning
Dec	Hex	
0	00	OFFLINE
64	40	STOP
128	80	CLEAR
192	C0	OPERATE

CANOM_STATE = OFFLINE

If CANOM_STATE is set to OFFLINE, the CANopen master performs an initialization. After the initialization phase is completed, the CANopen master changes to STOP state.

CANOM_STATE = STOP

If CANOM_STATE has the value STOP, the CANopen master is completely initialized. In this state the CANopen master is ready to receive configuration data. There is no data exchange with the slaves. CANOM_STATE = CLEAR

If a CANopen network configuration is received the CANopen master changes from STOP to CLEAR and starts to establish the connections defined during configuration. When the setup has been completed successfully, the CANopen master moves to OPERATE state. If an error occurs during parameterization, the CANopen master changes back to STOP state.

CANOM_STATE = OPERATE

If CANOM_STATE has the value OPERATE, the master exchanges I/O data with the slaves.

COM_ERR

Data type	Default value	Range	Unit
CANOM_COM_ERR_TYPE	-	-	-

Not used in CI506.

BUS_ERR

Data type	Default value	Range	Unit
WORD	-	-	-

BUS_ERR outputs the number of occurred bus errors. A bus error occurs if the internal error frame counter exceeds a specific value. BUS_ERR is only valid if EN = TRUE and ERR = FALSE.

BUS_OFF

Data type	Default value	Range	Unit
WORD	-	-	-

BUS_OFF is incremented if the CAN interface reports that it detaches itself from CAN bus due to exceeded internal bus error counters and must be reinitialized. BUS_OFF is only valid if EN = TRUE and ERR = FALSE.

LOST_REC

Data type	Default value	Range	Unit
WORD	-	-	-

LOST_REC outputs the number of received telegrams that were rejected because they could not be processed successfully due to a CAN chip overload. LOST_REC is only valid, if EN = TRUE and ERR = FALSE.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

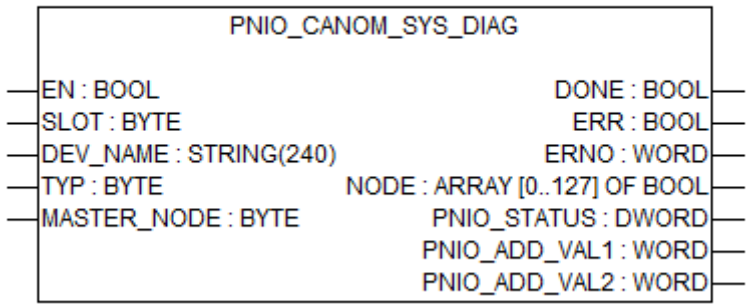
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_CANOM_STATE(EN          := STATE_EN,
                 SLOT         := STATE_SLOT,
                 DEV_NAME     := STATE_DEV_NAME,
                 MASTER_NODE  := STATE_MASTER_NODE);


STATE_ERR           := PNIO_CANOM_STATE.ERR;
STATE_ERNO          := PNIO_CANOM_STATE.ERNO;
STATE_BITS          := PNIO_CANOM_STATE.STATE_BITS;
STATE_CANCOM_STATE  := PNIO_CANOM_STATE.CANOM_STATE;
STATE_CANCOM_ERR    := PNIO_CANOM_STATE.COM_ERR;
STATE_BUS_ERR       := PNIO_CANOM_STATE.BUS_ERR;
STATE_BUS_OFF       := PNIO_CANOM_STATE.BUS_OFF;
STATE_TOUT_ERR      := PNIO_CANOM_STATE.TOUT_ERR;
STATE_LOST_REC      := PNIO_CANOM_STATE.LOST_REC;
STATE_PNIO_STATUS   := PNIO_CANOM_STATE.PNIO_STATUS;
STATE_PNIO_ADD_VAL1 := PNIO_CANOM_STATE.PNIO_ADD_VAL1;
STATE_PNIO_ADD_VAL2 := PNIO_CANOM_STATE.PNIO_ADD_VAL2;
STATE_DONE          := PNIO_CANOM_STATE.DONE;
```

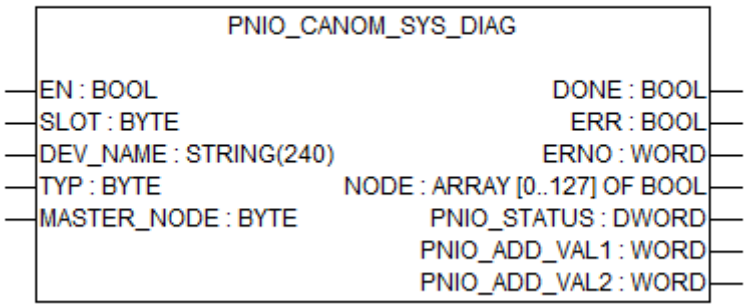
PNIO_CANOM_SYS_DIAG



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	CANopen

The function block PNIO_CANOM_SYS_DIAG gets the states of all slaves connected to the CI506-PNIO CANopen master.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

TYP

Data type	Default value	Range	Unit
BYTE	-	-	-

Input TYP selects the type of CANopen system diagnosis displayed at output NODE.

TYP = 1 Configured slaves

Output NODE displays which slaves are configured for the CANopen master in CI506 (TRUE).

TYP = 2 Operational slaves

Output NODE displays which slaves are error-free and in operation. A slave can only be operational if it has been configured in the master. The type of system diagnosis can only be requested if the CANopen master is in OPERATE state.

TYP = 3 Slaves with diagnosis

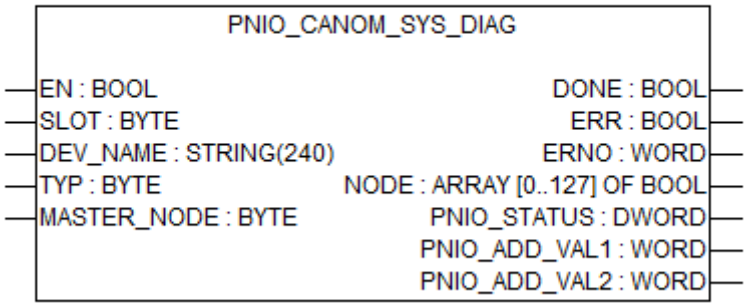
Output NODE indicates which slaves report a diagnosis. The diagnosis survey can only be requested if the CANopen master is in OPERATE state.

MASTER_NODE

Data type	Default value	Range	Unit
BYTE	-	1 - 127	-

Input MASTER_NODE describes the CANopen master node ID in CI506.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NODE

Data type	Default value	Range	Unit
ARRAY ARRAY	-	-	-

Output NODE outputs the status of each slave as a bitfield. Each individual bit within this field represents one slave. The index number corresponds to the slave's node address. If a bit is set to TRUE, the state selected using TYP applies for the slave. If e.g. TYP = 1 is selected and NODE[2] = TRUE, the slave with a configuration for this node was received and stored in master. If NODE[2] = FALSE, the slave is not part of the master's configuration data. If TYP = 3, NODE[2] = TRUE means that the slave with the node address 2 has received an emergency message or that the diagnosis indication of the slave has changed. In this case, a diagnosis description can be requested using the function block PNIO_CANOM_NODE_DIAG ↳ *Chapter 1.5.4.28.1.9 “PNIO_CANOM_NODE_DIAG” on page 1866*.

The bitfield output at NODE is only valid, if EN = TRUE and ERR = FALSE.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

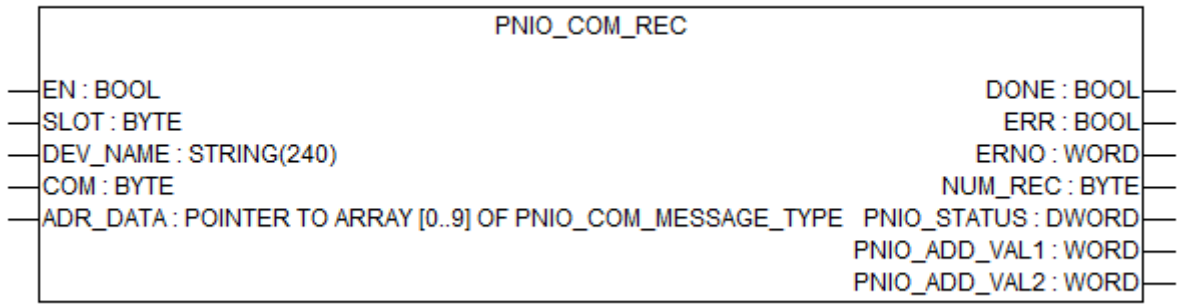
Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_CANOM_SYS_DIAG(EN          := SYS_DIAG_EN,
                    SLOT         := SYS_DIAG_SLOT,
                    DEV_NAME     := SYS_DIAG_DEV_NAME,
                    TYP          := SYS_DIAG_TYP,
                    MASTER_NODE := SYS_DIAG_MASTER_NODE);


SYS_DIAG_ERR           := PNIO_CANOM_SYS_DIAG.ERR;
SYS_DIAG_ERNO          := PNIO_CANOM_SYS_DIAG.ERNO;
SYS_DIAG_NODE          := PNIO_CANOM_SYS_DIAG.NODE;
SYS_DIAG_PNIO_STATUS   := PNIO_CANOM_SYS_DIAG.PNIO_STATUS;
SYS_DIAG_PNIO_ADD_VAL1 := PNIO_CANOM_SYS_DIAG.PNIO_ADD_VAL1;
SYS_DIAG_PNIO_ADD_VAL2 := PNIO_CANOM_SYS_DIAG.PNIO_ADD_VAL2;
SYS_DIAG_DONE          := PNIO_CANOM_SYS_DIAG.DONE;
```

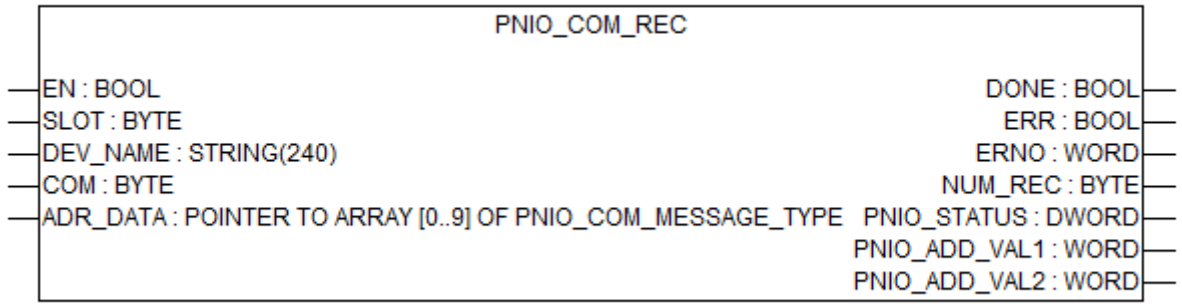
PNIO_COM_REC



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	COM

The function block PNIO_COM_REC provides all serial messages received on the addressed CI504-PNIO or CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At the COM input, the number of the serial interface is specified.

COM = 1: COM1

COM = 2: COM2

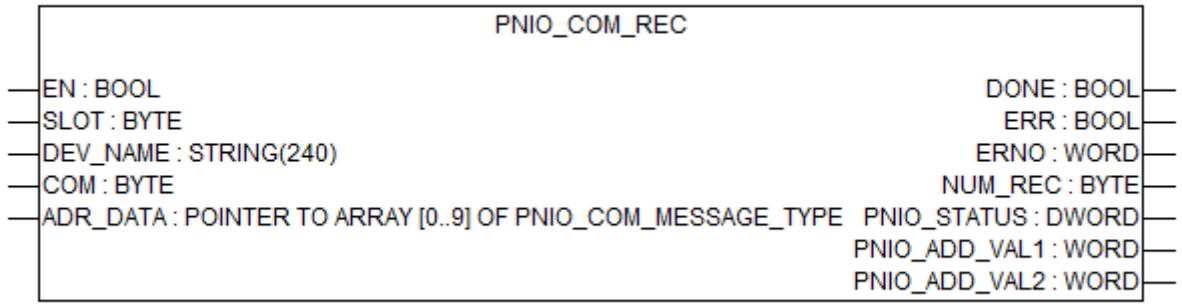
COM = 3: COM3 (CI504 only)

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_REC

Data type	Default value	Range	Unit
DWORD	-	-	-

Output NUM_REC displays the total number of CAN 2.0A telegrams received.

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

PNIO_COM_MESSAGE_TYPE

This data type is declared as follows:

```
TYPE PNIO_COM_MESSAGE_TYPE:
STRUCT
    HEADER: PNIO_COM_MESSAGE_HEADER_TYPE; (*HEADER*)
    DATA: ARRAY [1..256] OF BYTE; (*data if any, depends on LENGTH in
HEADER*)
END_STRUCT
END_TYPE
```

PNIO_COM_MESSAGE_HEADER_TYPE

This data type is declared as follows:

```
TYPE PNIO_COM_MESSAGE_HEADER_TYPE:
STRUCT
    STATUS: WORD; (*STATUS*)
    LENGTH: WORD; (*LENGTH*)
END_STRUCT
END_TYPE
```

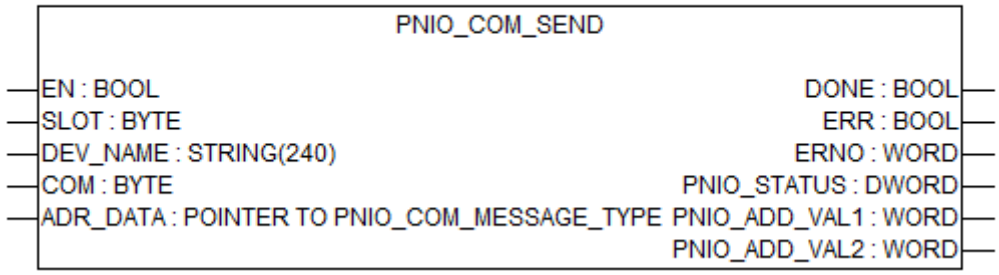
Possible values for STATUS are:

```
CDRV_FRAME_BUFFER_FULL = 0x101,
CDRV_FRAME_END_CHAR = 0x102,
CDRV_FRAME_TIMEOUT = 0x103,
CDRV_FRAME_BREAK = 0x104,
CDRV_FRAME_FRAMING = 0x105,
CDRV_FRAME_PARITY = 0x106,
CDRV_FRAME_CANCELED = 0x107,
CDRV_FRAME_ERROR = 0x108,
CDRV_FRAME_CHKSUM = 0x109,
CDRV_FRAME_LEN = 0x10A
```

Function call in ST

```
PNIO_COM_REC(EN       := REC_EN,
                      SLOT     := REC_SLOT,
                      DEV_NAME := REC_DEV_NAME,
                      COM      := REC_COM,
                      ADR_DATA := REC_ADR);


REC_ERR           := PNIO_COM_REC.ERR;
REC_ERNO          := PNIO_COM_REC.ERNO;
REC_NUM_REC       := PNIO_COM_REC.NUM_REC;
REC_PNIO_STATUS   := PNIO_COM_REC.PNIO_STATUS;
REC_PNIO_ADD_VAL1 := PNIO_COM_REC.PNIO_ADD_VAL1;
REC_PNIO_ADD_VAL2 := PNIO_COM_REC.PNIO_ADD_VAL2;
REC_DONE          := PNIO_COM_REC.DONE;
```

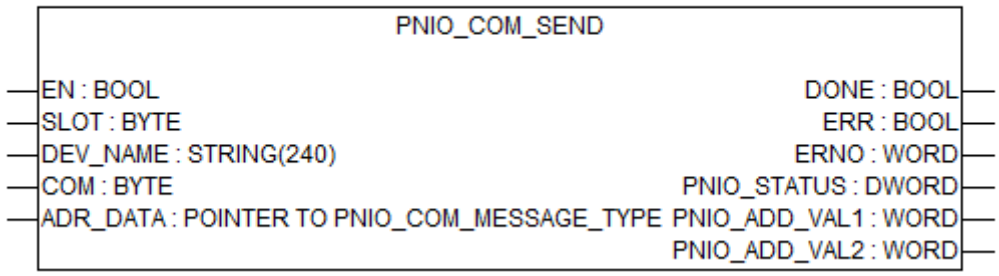
PNIO_COM_SEND



Parameter	Value
Included in library	PROFINET_Ext_AC500_V20.lib
Available as of firmware	V2.1.x
Type	Function block with historical values
Group	COM

The function block PNIO_COM_SEND provides all serial messages via the addressed CI504-PNIO or CI506-PNIO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

DEV_NAME

Data type	Default value	Range	Unit
STRING	-	-	-

Input DEV_NAME describes the PROFINET IO device to be queried. The string has a maximum length of 240 characters, including the zero terminating char. In conjunction with the SLOT value this string selects the target I/O module of this request.

Valid values are (examples for the ABB modules, the hash marks # are place holders for the address selected by the rotary switch at the front of the device):

ci506-pn-##

COM

Data type	Default value	Range	Unit
BYTE	-	-	-

At the COM input, the number of the serial interface is specified.

COM = 1: COM1

COM = 2: COM2

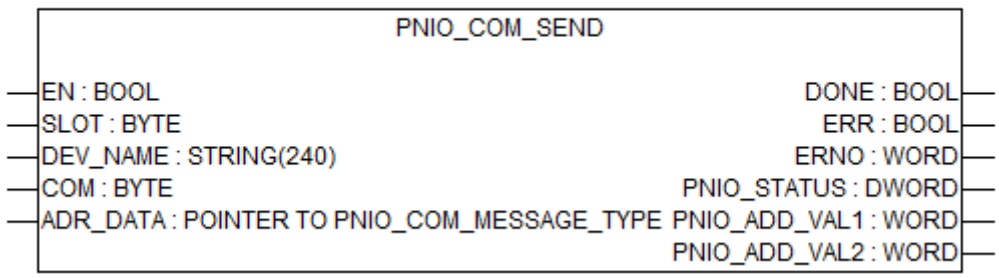
COM = 3: COM3 (CI504 only)

ADR_DATA

Data type	Default value	Range	Unit
ARRAY	-	-	-

Input ADR_DATA specifies the address starting from which the received CAN 2.0A telegrams should be written. Usually, this specification is done via the ADR operator and should point to an array with 64 entries of the type CAN2A_MESSAGE_TYPE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE. (see *Chapter 1.5.3 "Error messages of the AC500 V2 function block libraries" on page 735*).

PNIO_STATUS

Data type	Default value	Range	Unit
DWORD	-	-	-

Output PNIO_STATUS contains the PNIO ErrorCode and ErrorDecode.

PNIO_ADD_VAL 1

Data type	Default value	Range	Unit
WORD	1	-	-

Output PNIO_ADD_VAL1 contains the PNIO ErrorCode1.

PNIO_ADD_VAL 2

Data type	Default value	Range	Unit
WORD	2	-	-

PNIO_ADD_VAL2 contains the PNIO ErrorCode2.

Function call in ST

```
PNIO_COM_SEND(EN       := SEND_EN,
                      SLOT     := SEND_SLOT,
                      DEV_NAME := SEND_DEV_NAME,
                      COM      := SEND_COM,
                      ADR_DATA := SEND_ADR_DATA);


SEND_ERR           := PNIO_COM_SEND.ERR;
SEND_ERNO          := PNIO_COM_SEND.ERNO;
SEND_PNIO_STATUS   := PNIO_COM_SEND.PNIO_STATUS;
SEND_PNIO_ADD_VAL1 := PNIO_COM_SEND.PNIO_ADD_VAL1;
SEND_PNIO_ADD_VAL2 := PNIO_COM_SEND.PNIO_ADD_VAL2;
SEND_DONE          := PNIO_COM_SEND.DONE;
```

1.5.4.29 Profinet_Ext2 library

Library file name: **Profinet_Ext2_AC500_Vx.lib**

This library contains function blocks which are used to control the PROFINET IO network and the connected devices.

Preconditions

All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.

The following libraries are required to use the function blocks of the Profinet_Ext2 library:

- *CMN_AC500_V24.lib (V 1.0.0)*
- *SysInt_AC500_V10.lib (V 1.4.3)*
- *Profinet_AC500_V13.lib (V1.2.1)*

These libraries are part of the Automation Builder.

1.5.4.29.1 Function blocks

PNIO_CNTL_START_COM

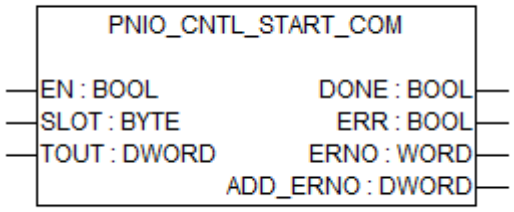
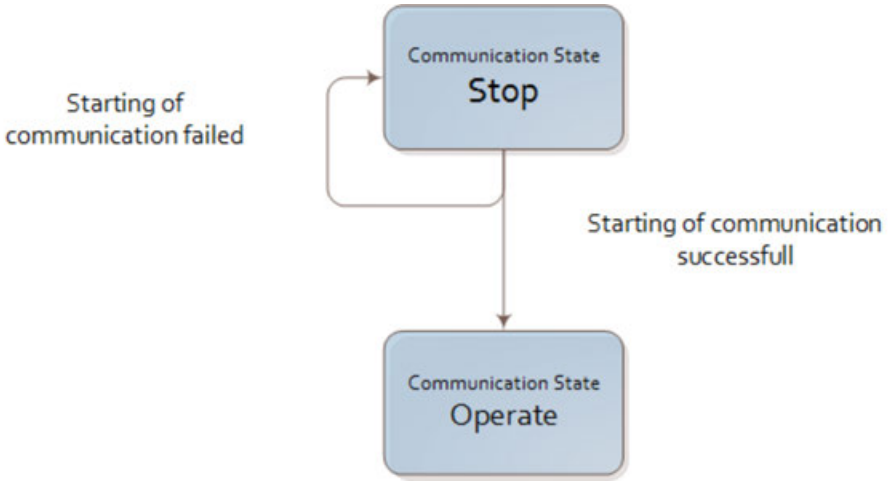


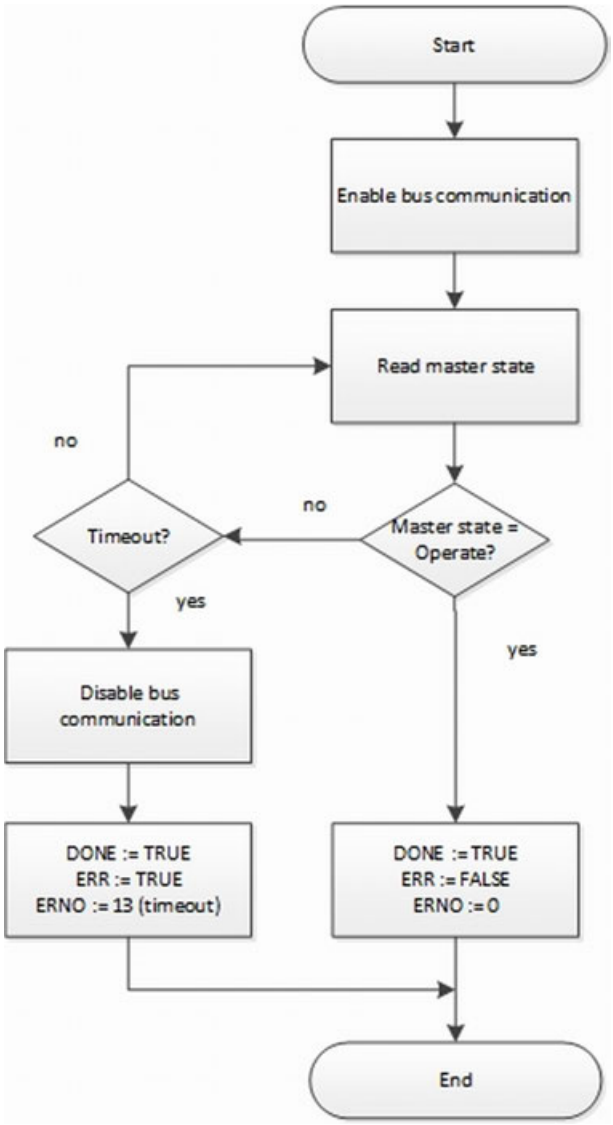
Table 91: General information

Available as of runtime system	V2.6.2 and above
Available as of AB version	V2.1 and above
Available as of CM579-PNIO version	V2.8.4.20 and above
Included in library	Profinet_Ext2_AC500_V26.lib
Type	Function block with historical values.

The function block PNIO_CNTL_START_COM can be used to start the communication of the PROFINET IO Controller. That means the communication state is changed from Stop to Operate.



The next figure shows the flowchart of the execution of the function block PNIO_CNTL_START_COM.:

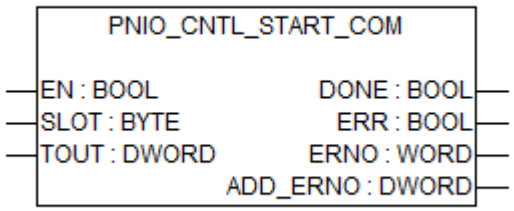


When the execution is started the bus communication will be enabled. After that the communication state of the PROFINET IO Controller is checked and it's waited until the communication state is Operate.

The execution of the FB was successful if the PROFINET IO Controller was changed to the communication state Operate.

If the PROFINET IO Controller is not in the communication state Operate within the specified timeout the execution of the FB is failed and the bus communication is disabled again.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

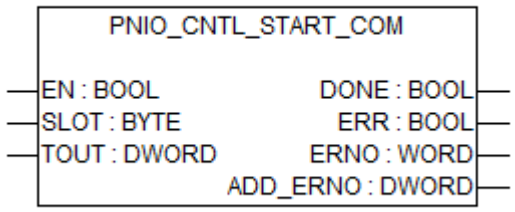
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

PNIO_CNTL_STOP_COM

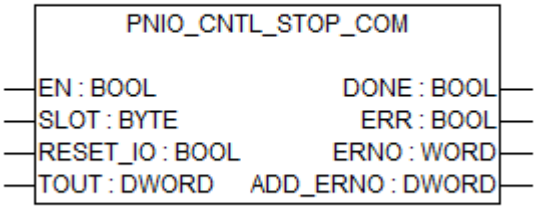
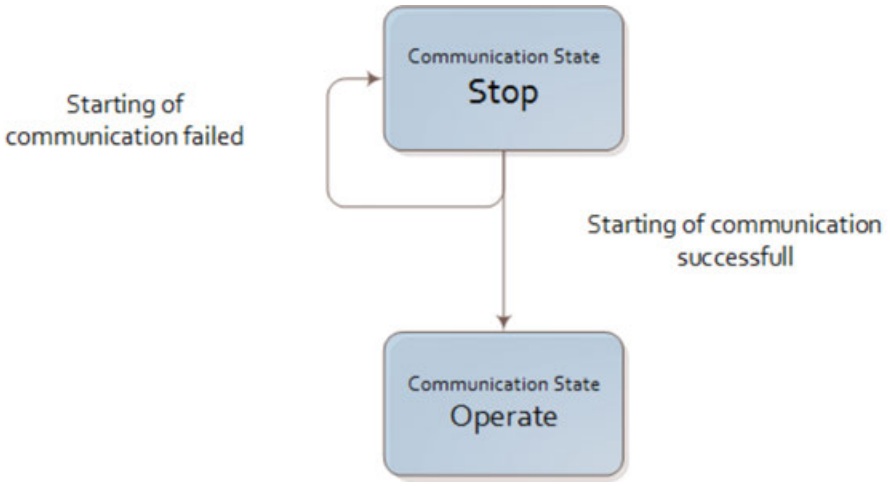


Table 92: General information

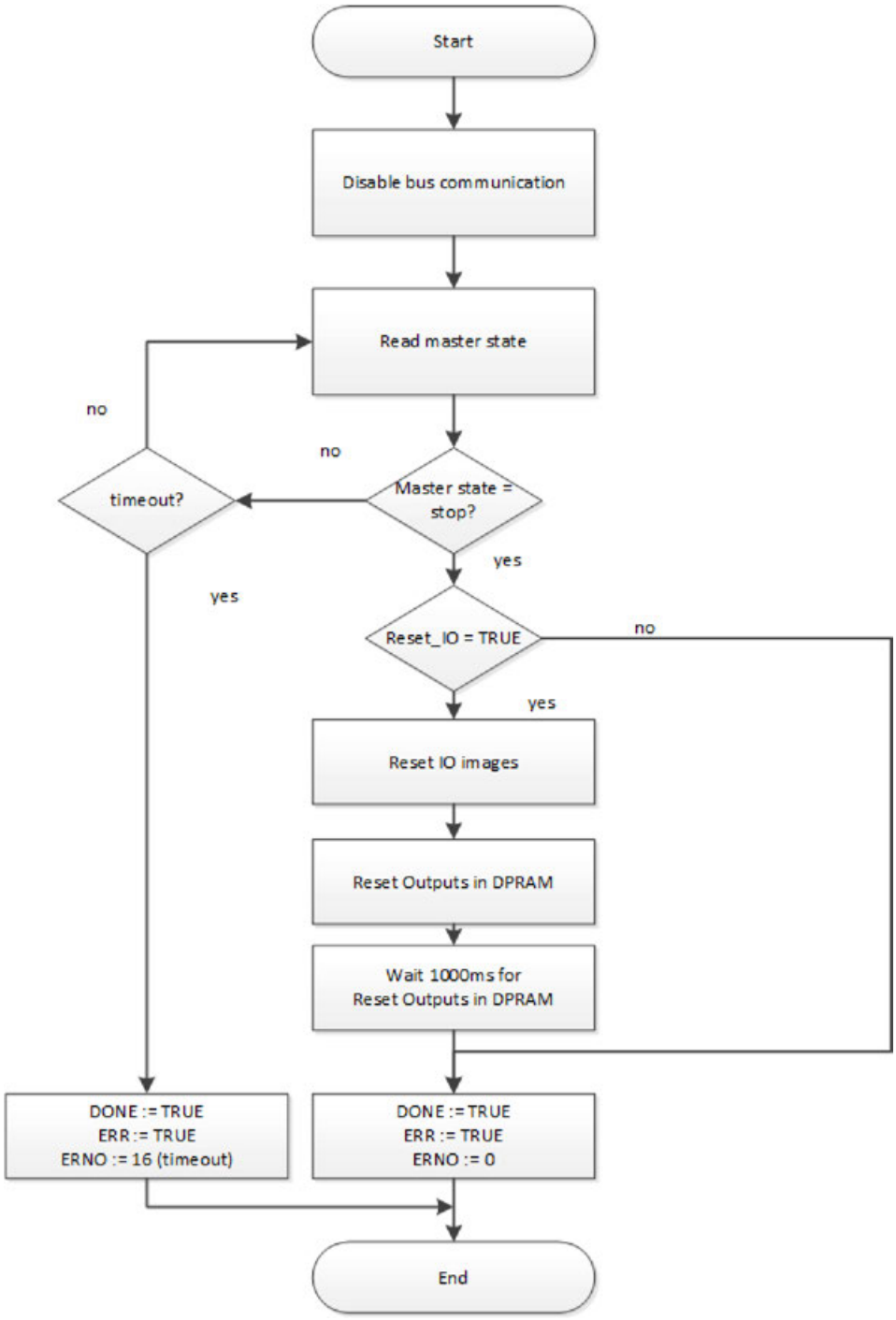
Available as of runtime system	V2.6.2 and above
Available as of AB version	V2.1 and above
Available as of CM579-PNIO version	V2.8.4.20 and above
Included in library	Profinet_Ext2_AC500_V26.lib
Type	Function block with historical values.

The function block PNIO_CNTL_STOP_COM can be used to stop the communication of the PROFINET IO Controller. That means the communication state is changed from Operate to Stop.



The user is responsible for resetting of the PROFINET IO related part of the PLC application when the bus communication is re-started

The next figure shows the flowchart of the execution of the function block PNIO_CNTL_STOP_COM.:



When the execution is started the bus communication will be disabled. After that the communication state is checked and it's waited until the communication state is Stop.

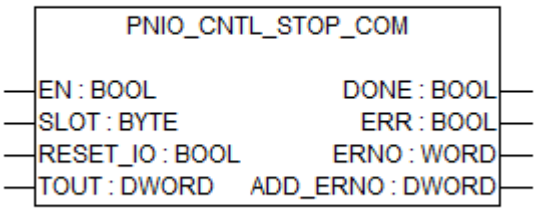
If the PROFINET IO Controller is in communication Stop and the input RESET_IO is TRUE the IOs will be reset.

The execution of the FB was successful if the communication state was changed to Stop and no error is occurred during resetting the IOs. If the PROFINET IO Controller is not in the communication state Stop within the specified timeout the execution of the FB is failed.

The IO reset functionality of the FB PNIO_CNTL_STOP_COM is not synchronized. That means there is no notification when the reset is finished.

The FB PNIO_CNTL_STOP_COM has an internal waiting time of 1000ms to be sure the IOs were reset. If this time is not sufficient it has to be waited an additional time before the communication is re-started with the FB PNIO_CNTL_START_COM.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

RESET_IO

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

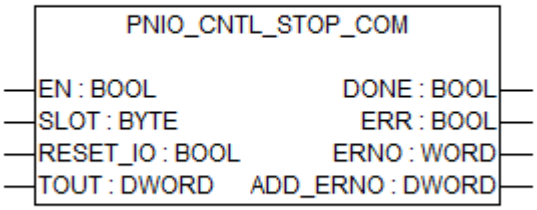
IO values will be reset if set to true.

TOUT

Data type	Default value	Range	Unit
DWORD	0	0 … 4294967295	ms

Value of timeout in ms.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ADD_ERNO

Data type	Default value	Range	Unit
DWORD	-	$0…2^{32-1}$	-

At output ADD_ERNO an additional error code is provided when output ERNO is 0 x 0031.

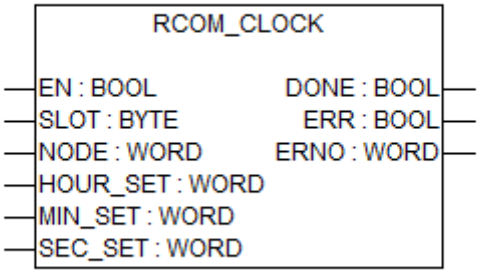
1.5.4.30 RCOM/RCOM+ library

Library file name: **RCOM_AC500_Vx.lib**

> *All function blocks of this library can only be executed in RUN mode of the processor module, not in simulation mode.*

1.5.4.30.1 Function blocks

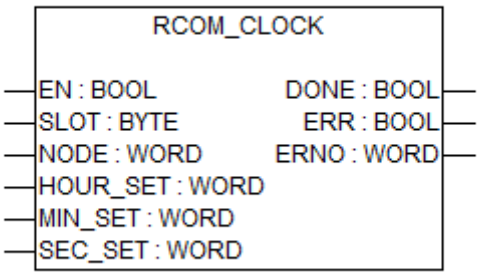
RCOM_CLOCK



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block transmits a telegram that sets the clock of the called slave(s) to the time specified at the function block inputs. The master clock is set to the same value.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

HOUR_SET

Data type	Default value	Range	Unit
WORD	-	-	-

Input HOUR_SET specifies the value of the hour component of the time.

MIN_SET

Data type	Default value	Range	Unit
WORD	-	-	-

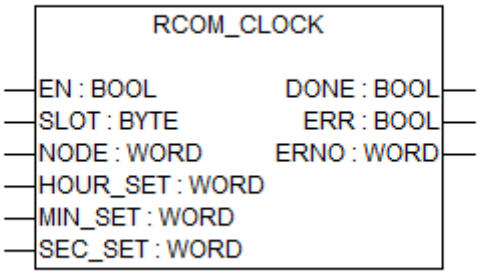
Input MIN_SET specifies the value of the minutes component of the time.

SEC_SET

Data type	Default value	Range	Unit
WORD	-	-	-

Input SEC_SET specifies the value of the seconds component of the time.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

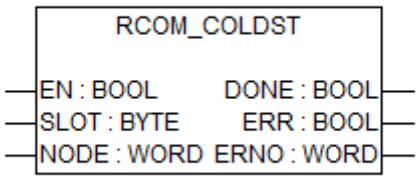
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMClock(
     EN := RCOMClock_EN,
     SLOT := RCOMClock_SLOT,
     NODE := RCOMClock_NODE,
     HOUR_SET := RCOMClock_HOUR_SET,
     MIN_SET := RCOMClock_MIN_SET,
     SEC_SET := RCOMClock_SEC_SET);
RCOMClock_DONE := RCOMClock.DONE;
RCOMClockart_ERR   := RCOMClock.ERR;
RCOMClock_ERNO := RCOMClock.ERNO;
```

RCOM_COLDST



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

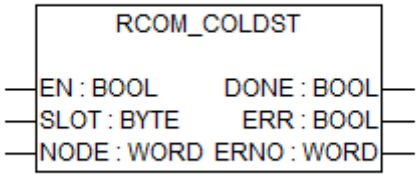
This function block performs a cold start of the called slave(s). After the cold start, the slave(s) are partially reinitialized.

At the called slave(s) the sequence marks (RCOM-specific protocol parameter) are reset and the transmission of data records is disabled. All entries in the event queue are deleted, and, after this, a cold start event is generated.

After a cold start, always a normalization is required.

This function block has to be used for the initialization of the RCOM network, e.g. after the initialization of the master's CM574-RCOM device.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

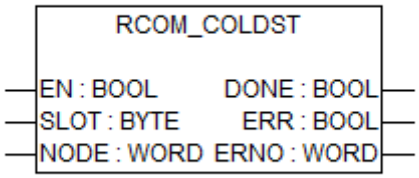
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

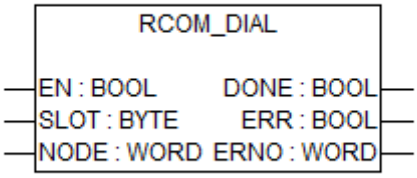
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMColdstart(
     EN := RCOMColdstart_EN,
     SLOT := RCOMColdstart_SLOT,
     NODE := RCOMColdstart_NODE);
RCOMColdstart_DONE := RCOMColdstart.DONE;
RCOMColdstart_ERR  := RCOMColdstart.ERR;
RCOMColdstart_ERNO := RCOMColdstart.ERNO;
```

RCOM_DIAL



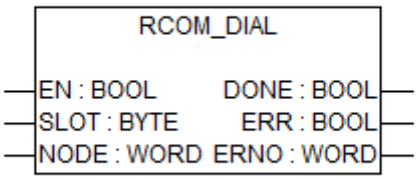
Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block calls a specified subscriber in a network realized with dial lines. If used in the RCOM master, the function block has to be processed before transmitting system services or data records to a slave.

If used in a slave, the function block can be used to call the master in order to request event polling.

The dial line has to be terminated using the RCOM_HANGUP function block after transmission is completed.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

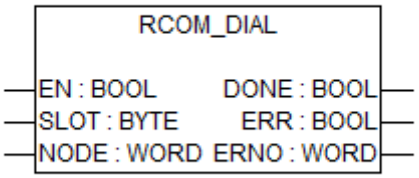
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

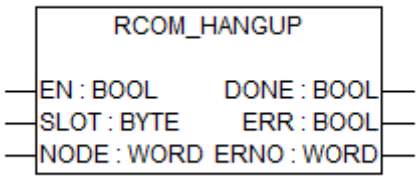
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMDial(
```

```
      EN := RCOMDial_EN,
      SLOT := RCOMDial_SLOT,
      NODE := RCOMDial_NODE);
RCOMDial_DONE := RCOMDial.DONE;
RCOMDial_ERR  := RCOMDial.ERR;
RCOMDial_ERNO := RCOMDial.ERNO;
```

RCOM_HANGUP



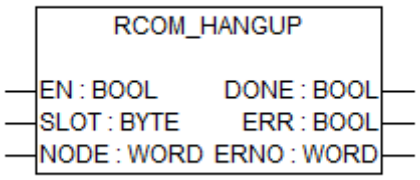
Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block hangs up the connection in networks realized with dial lines.

The function block has to be used to terminate the connection after transmitting data records to or from a slave.

The function block cannot be used until all jobs for a slave are processed completely. Furthermore, the function block can only be used, if the RCOM network is realized by dial lines ("Hayes compatible dial modem"; has to be selected as modem type in PLC configuration).

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

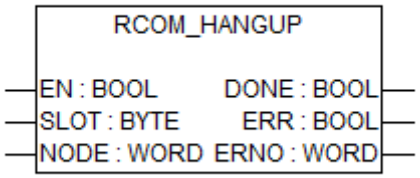
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

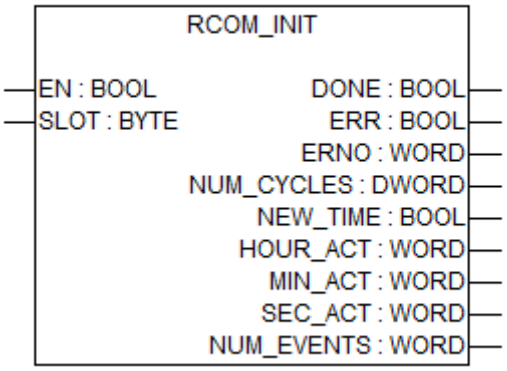
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMHangup(
     EN := RCOMHangup_EN,
     SLOT := RCOMHangup_SLOT,
     NODE := RCOMHangup_NODE);
RCOMHangup_DONE := RCOMHangup.DONE;
RCOMHangup_ERR  := RCOMHangup.ERR;
RCOMHangup_ERNO := RCOMHangup.ERNO;
```

RCOM_INIT



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block is used to initialize the CM574-RCOM. It sets all necessary network parameters and starts initialization.

Once the function block is started, the input parameters must not be changed anymore. Due to this, the parameters should be preset by assigning constants.

A rising edge at input EN initiates the function block. The signal TRUE must be applied until the function block is processed completely.

If DONE = TRUE and ERR = FALSE, the initialization process was successful. If DONE = TRUE and ERR = TRUE, an error occurred and the error number is indicated at output ERNO.

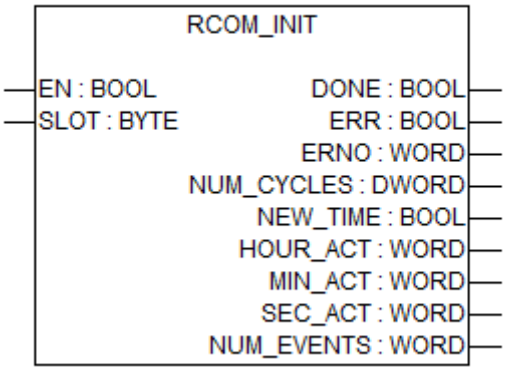
After initialization, the outputs ERR, ERNO, NUM_CYCLES, NEW_TIME, HOUR_ACT, MIN_ACT, SEC_ACT and NUM_EVENTS are updated with each PLC cycle as long as EN = TRUE. This way, the PLC permanently receives the cycle counter value and error messages from the CM574-RCOM. Therefore, input EN should constantly remain TRUE, except when the Communication Module has to be reinitialized with new parameters.

A falling edge at input EN resets the outputs ERR and ERNO.

Output NUM_CYCLES indicates the current cycle counter value of the CM574-RCOM. The outputs NEW_TIME, HOUR_ACT, MIN_ACT and SEC_ACT indicate the current RCOM system time which can also be used in the PLC program. Time starts at switching on with 00:00.00. If a ‘set clock’ command is received, the time is set to the new value and output NEW_TIME is set for approx. 5 seconds (e.g. for setting a real-time clock).

For RCOM slaves, output NUM_EVENTS indicates the number of events stored in the event queue. For the RCOM master, NUM_EVENTS is always zero.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

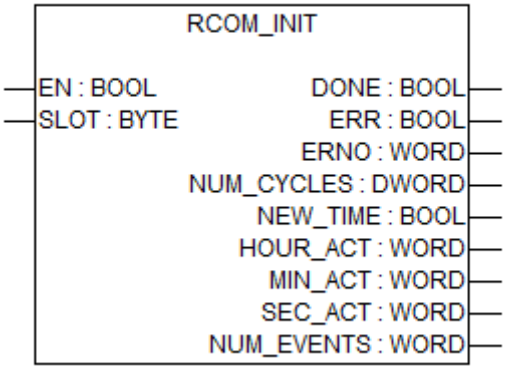
SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_CYCLES

Data type	Default value	Range	Unit
WORD	-	-	-

Output NUM_CYCLES provides the current value of the CM574-RCOM cycle counter. This value is permanently updated as long as EN = TRUE. The cycle counter should be used to monitor the CM574-RCOM (life identifier).

NEW_TIME

Data type	Default value	Range	Unit
BOOL	-	-	-

Output NEW_TIME is set to TRUE for approx. 5 seconds, if a new RCOM time has been set using the RCOM_CLOCK function block. This output can be used to adjust a real-time clock to the time applied at HOUR_ACT, MIN_ACT and SEC_ACT.

HOUR_ACT

Data type	Default value	Range	Unit
WORD	-	-	-

Output HOUR_ACT indicates the hours component of the RCOM system time. The master can adjust the time using the RCOM_CLOCK function block.

MIN_ACT

Data type	Default value	Range	Unit
WORD	-	-	-

Output MIN_ACT indicates the minutes component of the RCOM system time. The master can adjust the time using the RCOM_CLOCK function block.

SEC_ACT

Data type	Default value	Range	Unit
WORD	-	-	-

Output SEC_ACT indicates the seconds component of the RCOM system time. The master can adjust the time using the RCOM_CLOCK function block.

NUM_EVENTS

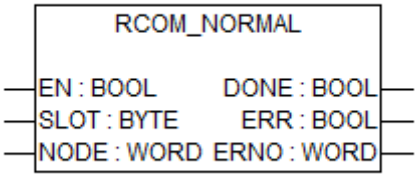
Data type	Default value	Range	Unit
WORD	-	-	-

For RCOM slaves, output NUM_EVENTS indicates the number of events stored in the event queue. For the RCOM master, NUM_EVENTS is always zero.

Function call in ST

```
RCOMInit(
EN := RCOMInit_EN,
SLOT := RCOMInit_SLOT);
RCOMInit_DONE := RCOMInit.DONE;
RCOMInit_ERR := RCOMInit.ERR;
RCOMInit_ERNO := RCOMInit.ERNO;
RCOMInit_NUM_CYCLES := RCOMInit.NUM_CYCLES;
RCOMInit_NEWTIME := RCOMInit.NEWTIME;
RCOMInit_HOUR_ACT := RCOMInit.HOUR_ACT;
RCOMInit_MIN_ACT := RCOMInit.MIN_ACT;
RCOMInit_SEC_ACT := RCOMInit.SEC_ACT;
RCOMInit_NUM_EVENTS := RCOMInit.NUM_EVENTS;
```

RCOM_NORMAL

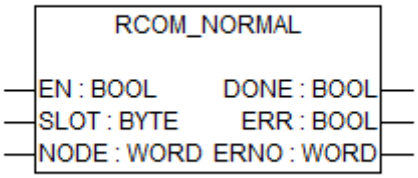


Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block performs a normalization at the called slave and thus enables the transmission of data records.

This function block has to be processed each time after switching on and after a cold start or a warm start.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

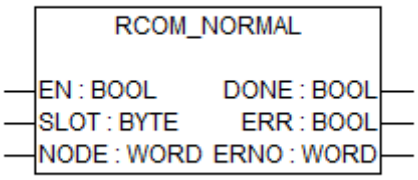
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

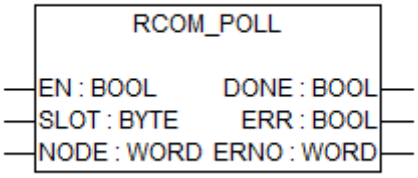
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMNormal(
     EN := RCOMNormal_EN,
     SLOT := RCOMNormal_SLOT,
     NODE := RCOMNormal_NODE);
RCOMNormal_DONE := RCOMNormal.DONE;
RCOMNormal_ERR  := RCOMNormal.ERR;
RCOMNormal_ERNO := RCOMNormal.ERNO;
```

RCOM_POLL

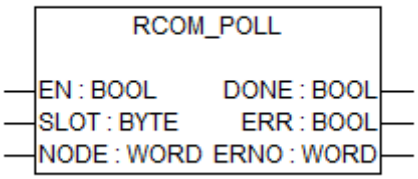


Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block performs event polling on the called slave, i.e. the slave is asked whether it wants to transfer data records to the master.

The data records returned by the slave appear in the master in a correspondingly configured here *Chapter 1.5.4.30.1.10 “RCOM_REC” on page 1927* function block.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

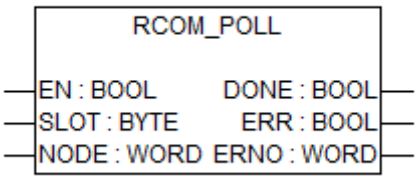
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

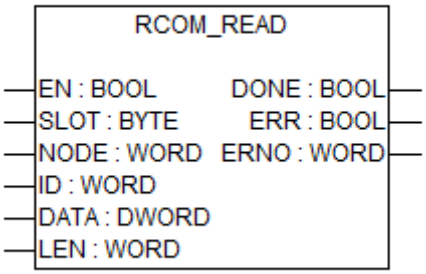
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMPoll(
     EN := RCOMPoll_EN,
     SLOT := RCOMPoll_SLOT,
     NODE := RCOMPoll_NODE);
RCOMPoll_DONE := RCOMPoll.DONE;
RCOMPoll_ERR  := RCOMPoll.ERR;
RCOMPoll_ERNO := RCOMPoll.ERNO;
```

RCOM_READ



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

The function block reads data from an RCOM slave. For this, a READ job is sent to the communication partner via the CM574-RCOM communication module. As a result, the communication partner returns an acknowledgement including the data.

The data received from the slave are stored in a flag area specified at input DATA.

Data transmission (i.e. reading data) using the RCOM_READ function block is only possible, if the CM574-RCOM communication module has been initialized (RCOM function block) and if normalization has been performed previously.

Once a job is started (FALSE/TRUE edge at input EN), the data at the inputs NODE, ID, DATA and LEN must not be changed until the job is completed (i.e. until DONE = TRUE).

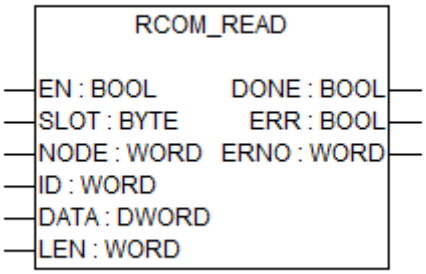
With each FALSE/TRUE edge at input EN the reading operation is performed once.

Further FALSE/TRUE edges at input EN are ignored until the reading operation is completed. The outputs DONE, ERR and ERNO are set when reading is completed.

Input NODE specifies the RCOM subscriber providing the data to be read. Input ID specifies the data record number within this subscriber and input LEN specifies the number of words to be transmitted.

If DONE = TRUE and ERR = FALSE, the reading operation could be completed successfully. If DONE = TRUE and ERR = TRUE, an error occurred and the error number is indicated at output ERNO. The outputs DONE, ERR and ERNO are reset with each falling edge at input EN.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

ID

Data type	Default value	Range	Unit
WORD	-	-	-

Input ID specifies the number of the data record to be read.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

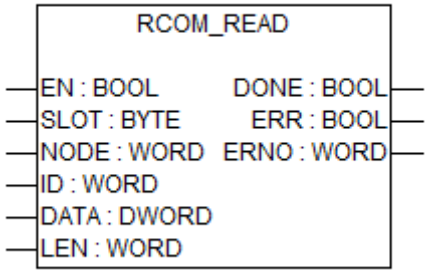
Input DATA specifies the first flag of the flag area where the data should be stored.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Input LEN specifies the number of words to be read (has to be even-numbered).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

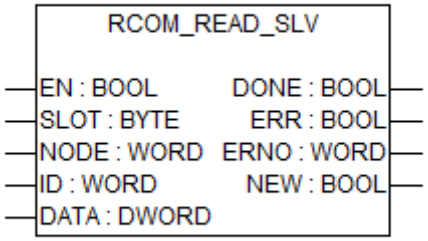
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMRead(
EN := RCOMRead_EN,
SLOT := RCOMRead_SLOT,
NODE := RCOMRead_NODE,
IDT := RCOMRead_IDT,
DATA := ADR(RCOMRead_DATA),
LEN := RCOMRead_LEN);
RCOMRead_DONE := RCOMRead.DONE;
RCOMRead_ERR  := RCOMRead.ERR;
```

```
RCOMRead_ERNO := RCOMRead.ERNO;
```

RCOM_READ_SLV



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block provides the data to be read by the master using an RCOM_READ function block.

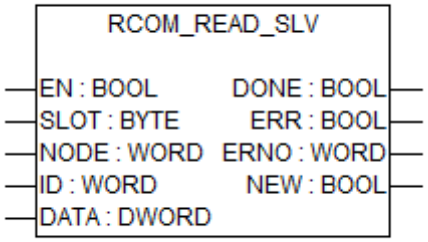
The data are provided to the Communication Module with an incoming READ job for the specified flag area.

If EN = FALSE or if the slave has not been normalized before, the function block rejects reading jobs (answer: ‘application part not ready’).

The data record number must match the number in the RCOM_READ function block.

The sender of the reading job is specified at input NODE. Because only the master of an RCOM network can send reading jobs, input NODE always has to be zero.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

ID

Data type	Default value	Range	Unit
WORD	-	-	-

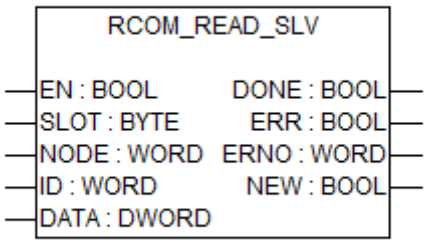
Input ID specifies the number of the data record to be read.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA specifies the first flag of the flag area where the data should be stored.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NEW

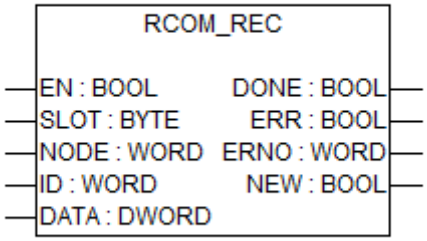
Data type	Default value	Range	Unit
BOOL	-	-	-

Output NEW is set to TRUE, if the master has read the requested data record. This way, the PLC is able to detect the reception of data by the master. Output NEW is reset during the next PLC cycle.

Function call in ST

```
RCOMReadSlv(
     EN := RCOMReadSlv_EN,
     SLOT := RCOMReadSlv_SLOT,
     NODE := RCOMReadSlv_NODE,
     IDT := RCOMReadSlv_IDT,
     DATA := ADR(RCOMReadSlv_DATA));
RCOMReadSlv_DONE := RCOMReadSlv.DONE;
RCOMReadSlv_ERR  := RCOMReadSlv.ERR;
RCOMReadSlv_ERNO := RCOMReadSlv.ERNO;
RCOMReadSlv_NEW  := RCOMReadSlv.NEW;
```

RCOM_REC



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block is used to receive the data records transmitted using the RCOM_TRANSMIT function block.

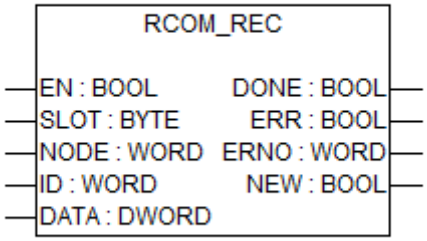
Output NEW is TRUE for one PLC cycle, if new data are available for the called data record. After this, the new data are stored to the specified flag area.

The data record number (ID) must match the number in the RCOM_TRANSMIT function block.

If EN = FALSE or if the slave has not been normalized before, the function block rejects writing requests (answer ‘application part not ready’).

The sender of the writing job is specified at input NODE. Data records for a slave are always sent by the master, therefore input NODE hast to be zero. Data records for the master (events) can be sent by different slaves, therefore input NODE has to be specified here. This enables the CM574-RCOM in the master to distribute data records correctly.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

ID

Data type	Default value	Range	Unit
WORD	-	-	-

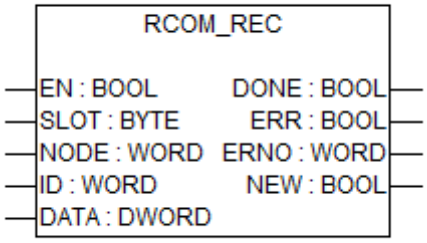
Input ID specifies the number of the data record to be read.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

Input DATA specifies the first flag of the flag area where the data should be stored.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NEW

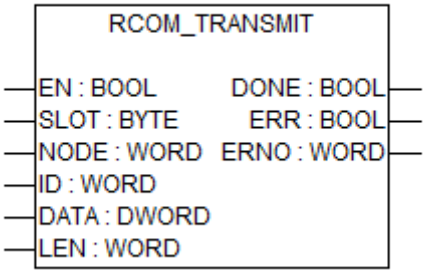
Data type	Default value	Range	Unit
BOOL	-	-	-

Output NEW is set to TRUE, if the master has read the requested data record. This way, the PLC is able to detect the reception of data by the master. Output NEW is reset during the next PLC cycle.

Function call in ST

```
RCOMRec(
     EN := RCOMRec_EN,
     SLOT := RCOMRec_SLOT,
     NODE := RCOMRec_NODE,
     ID := RCOMRec_ID,
     DATA := ADR(RCOMRec_DATA));
RCOMRec_DONE := RCOMRec.DONE;
RCOMRec_ERR  := RCOMRec.ERR;
RCOMRec_ERNO := RCOMRec.ERNO;
RCOMRec_NEW  := RCOMRec.NEW;
```

RCOM_TRANSMIT



Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block transmits a data record to an RCOM communication partner.

If used in an RCOM master, a send job including the user data is sent to the called slave. As a result, the slave sends an acknowledgement.

If the function block is used in a slave, the data record is transferred to the event queue and transmitted to the master with the next event poll.

At the receiver, the transmitted data appear at an accordingly programmed RCOM_REC function block.

Once a job has been started (FALSE/TRUE edge at input EN), the data at the inputs NODE, ID, LEN and DATA must not be changed until the job is completed (DONE = TRUE).

Data transmission using the RCOM_TRANSMIT function block is only possible, if the CM574-RCOM has been initialized (RCOM function block) and if normalization has been performed previously.

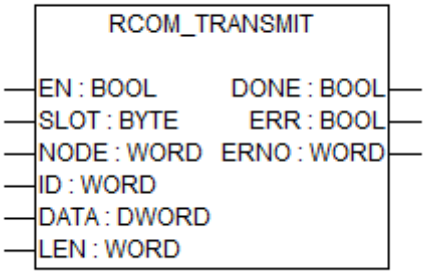
With each FALSE/TRUE edge at input EN, transmission is performed once. With this edge, the outputs DONE, ERR and ERNO are also set to zero.

Further FALSE/TRUE edges at input EN are ignored until transmission is completed (i.e. until DONE = TRUE). When transmission is completed, the outputs DONE, ERR and ERNO are set. DONE, ERR and ERNO are reset with a falling edge at input EN.

Input NODE specifies the RCOM subscriber that should receive the data. ID specifies the data record number within this subscriber, LEN has to be even-numbered and specifies the number of transmitted words. If the function block is used in a slave (event transmission), NODE has to be set to zero. LEN can be max. 14, because the timing mark is stored in addition to the user data.

Transmission was successful, if DONE = TRUE and ERR = FALSE. If DONE = TRUE and ERR = TRUE, an error occurred and the error type is indicated at output ERNO.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

ID

Data type	Default value	Range	Unit
WORD	-	-	-

Input ID specifies the number of the data record to be read.

DATA

Data type	Default value	Range	Unit
DWORD	-	-	-

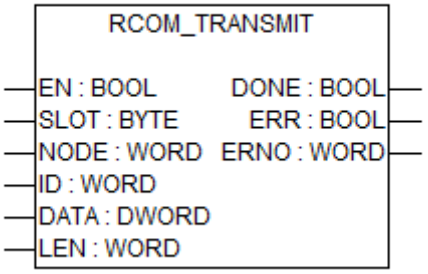
Input DATA specifies the first flag of the flag area where the data should be stored.

LEN

Data type	Default value	Range	Unit
WORD	-	-	-

Input LEN specifies the number of words to be read (has to be even-numbered).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

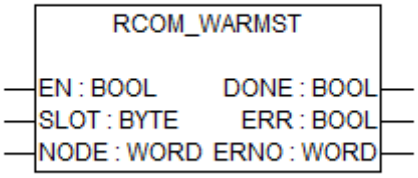
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RCOMTransmit(
EN := RCOMTransmit_EN,
SLOT := RCOMTransmit_SLOT,
NODE := RCOMTransmit_NODE,
ID := RCOMTransmit_ID,
DATA := ADR(RCOMTransmit_DATA),
LEN := RCOMTransmit_LEN);
RCOMTransmit_DONE := RCOMTransmit.DONE;
RCOMTransmit_ERR := RCOMTransmit.ERR;
RCOMTransmit_ERNO := RCOMTransmit.ERNO;
```

RCOM_WARMST



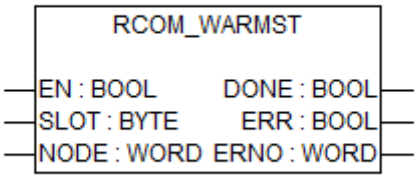
Parameter	Value
Included in library	RCOM_AC500_V13.lib
Available as of firmware	V1.3.1
Type	Function block with historical values
Group	RCOM/RCOM+ communication

This function block performs a warm start of the called slave. This deletes the event queue and disables the transmission of data records.

The function block can be used to resume communication, e.g. after a transmission error.

After a warm start, always a normalization has to be performed.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

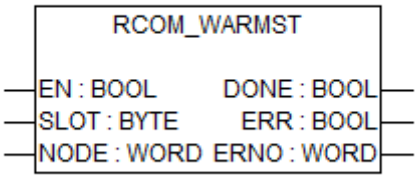
The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

NODE

Data type	Default value	Range	Unit
WORD	-	-	-

Input NODE specifies the number of the slave. The value 255 sets all slaves to the same time (all slaves are called).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

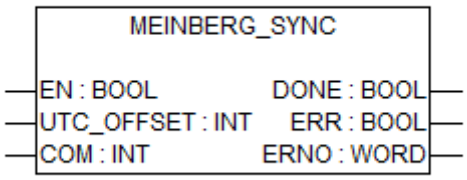
```
RCOMWarmst(
     EN := RCOMWarmst_EN,
     SLOT := RCOMWarmst_SLOT,
     NODE := RCOMWarmst_NODE);
RCOMWarmst_DONE := RCOMWarmst.DONE;
RCOMWarmst_ERR  := RCOMWarmst.ERR;
RCOMWarmst_ERNO := RCOMWarmst.ERNO;
```

1.5.4.31 RTC library

Library file name: **RTC_AC500_Vx.lib**

1.5.4.31.1 Function blocks

MEINBERG_SYNC



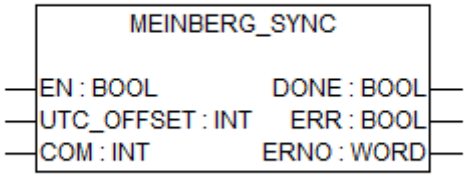
Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Radio clock

The function block MEINBERG_SYNC reads a telegram out off the Meinberg Radio Clock. The connection of Meinberg Radio Clock to an AC500 PLC is done via RS-232 interface (COM 1 or COM 2) and ASCII Protocol. The AC500 time management receives this telegram and disassembles the telegram.

The function block MEINBERG_SYNC will adjust the current time to the time received from the Meinberg Radio Clock.

MEINBERG_SYNC is called cyclic in users application.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

UTC_OFFSET

Data type	Default value	Range	Unit
INT	-	-	-

If time format of Radio Clock is UTC, with UTC_OFFSET you can calculate the time in MEZ / MESZ.

The offset UTC_OFFSET is in minutes.

COM

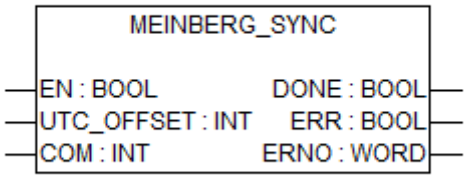
Data type	Default value	Range	Unit
INT	-	-	-

Specifies the COM-Port the real time Radio Clock is connected to:

COM:= 1 à COM-Port 1

COM:= 2 à COM-Port 2

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

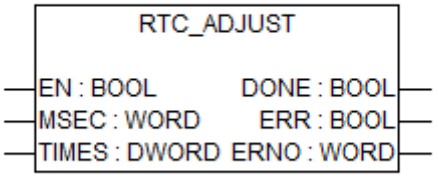
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
MEINBERG_SYNC(
EN:= Meinberg_Sync_EN,
   UTC_OFFSET:= Meinberg_Sync_UTC,
   COM:= Meinberg_Sync_COM);

Meinberg_Sync_DONE:= Meinberg_Sync.DONE;
Meinberg_Sync_ERR:= Meinberg_Sync.ERR;
Meinberg_Sync_ERNO:= Meinberg_Sync.ERNO;
```

RTC_ADJUST

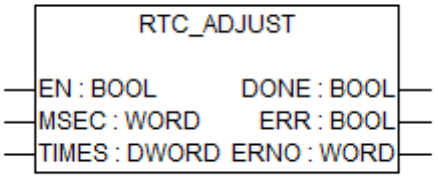


Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Real-time clock

The function block RTC_ADJUST is used to adjust the current time with the supplied time. It also evaluates the correction values for the time tick adjustment.

Adjustment will be done smoothly during the interval set by RTC_SET_ADJUST_INTERVAL. Should be called on each readout of the external time source.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

MSEC

Data type	Default value	Range	Unit
WORD	-	-	-

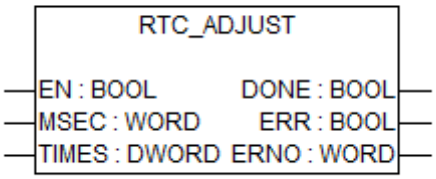
Contains the actual milliseconds within this second.

TIMES

Data type	Default value	Range	Unit
DWORD	-	-	-

Contains the actual time in seconds since 01.01.1970.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

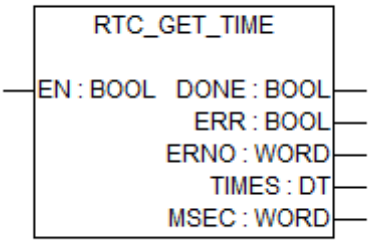
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RTC_ADJUST(EN:= RTC_Adjust_EN,
  MSEC:= RTC_Adjust_MSEC,
        TIMES:= RTC_Adjust_TIMES);

RTC_Adjust_DONE:= RTC_Adjust.DONE;
RTC_Adjust_ERR:= RTC_Adjust.ERR;
RTC_Adjust_ERNO:= RTC_Adjust.ERNO;
```

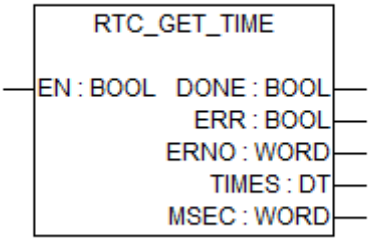
RTC_GET_TIME



Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Real-time clock

With this function block the user is able to read the actual time out of AC500. The format of time is in seconds since 01.01.1970 and a word containing the milliseconds within the current second.

Input description



EN

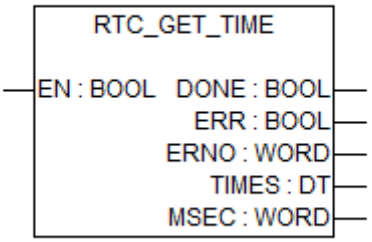
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

TIMES

Data type	Default value	Range	Unit
DWORD	-	-	-

Contains the actual time in seconds since 01.01.1970.

MSEC

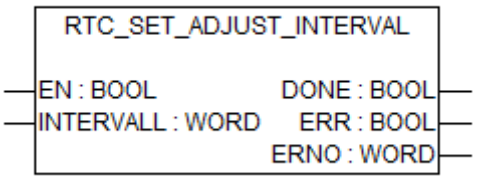
Data type	Default value	Range	Unit
WORD	-	-	-

Contains the actual milliseconds within this second.

Function call in ST

```
RTC_GET_TIME(
EN:= RTC_Get_Time_EN);

RTC_Get_Time_DONE:= RTC_Get_Time.DONE;
RTC_Get_Time_ERR:= RTC_Get_Time.ERR;
RTC_Get_Time_ERNO:= RTC_Get_Time.ERNO;
RTC_Get_Time_TIMES:= RTC_Get_Time.TIMES;
RTC_Get_Time_MSEC:= RTC_Get_Time.MSEC;
```

RTC_SET_ADJUST_INTERVAL

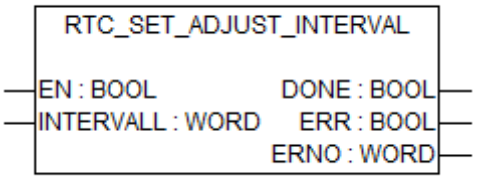


Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Real-time clock

The function block RTC_SET_ADJUST_INTERVAL is used to set the adjustment interval and calculate the maximum correction possible in that interval.

Smallest value is 2 sec. For the Meinberg real-time clock as time source a value of 60 sec is chosen.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

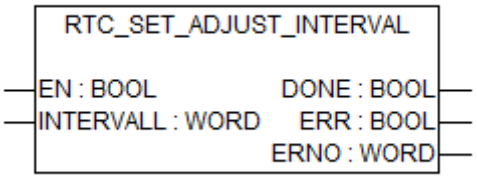
While it is executed its inputs are continuously evaluated.

INTERVALL

Data type	Default value	Range	Unit
WORD	-	-	-

Adjustment interval time in seconds.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

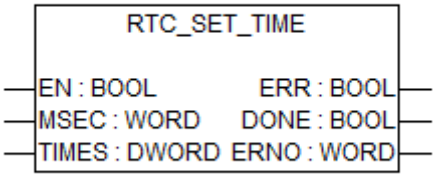
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RTC_SET_ADJUST_INTERVAL(
      EN:= RTC_Set_Adjust_Interval_EN,
      INTERVALL:= RTC_Set_Adjust_Interval_INTERVALL);

RTC_Set_Adjust_Interval_DONE:= RTC_Set_Adjust_Interval.DONE;
RTC_Set_Adjust_Interval_ERR:= RTC_Set_Adjust_Interval.ERR;
RTC_Set_Adjust_Interval_ERNO:= RTC_Set_Adjust_Interval.ERNO;
```

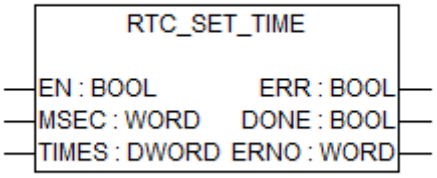
RTC_SET_TIME



Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Group/Subgroup

The function block RTC_SET_TIME is used to set the current time in AC500. The time must be supplied in seconds since 01.01.1970 and milliseconds of current second.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

MSEC

Data type	Default value	Range	Unit
WORD	-	-	-

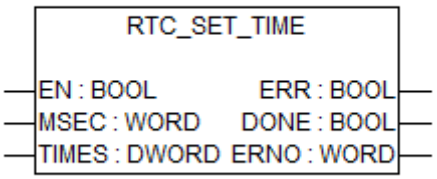
Contains the actual milliseconds within this second.

TIMES

Data type	Default value	Range	Unit
DWORD	-	-	-

Contains the actual time in seconds since 01.01.1970.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

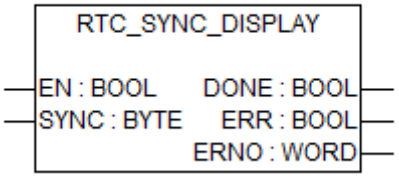
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

```
RTC_SET_TIME(
EN:= RTC_Set_Time_EN,
MSEC:= RTC_Set_Time_MSEC,
TIMES:= RTC_Set_Time_Times);


RTC_Set_Time_DONE:= RTC_Set_Time.DONE;
RTC_Set_Time_ERR:= RTC_Set_Time.ERR;
RTC_Set_Time_ERNO:= RTC_Set_Time.ERNO;
```

RTC_SYNC_DISPLAY

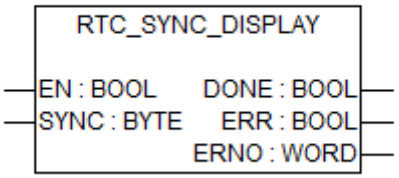


Parameter	Value
Included in library	RTC_AC500_V20.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	Real-time clock

This Funktion Block is used to inform the time management of AC500, if the source of its current time shall be the RTC of the display unit or if an external time source is used.

When using an external time source, this function block is called in the user program if the time of the external source is valid.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

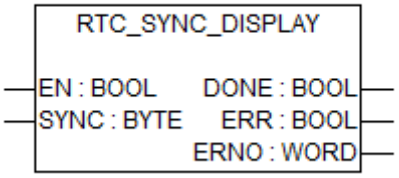
SYNC

Data type	Default value	Range	Unit
BYTE	-	-	-

0: External time source (e.g. Meinberg real-time clock) will be master.

1: RTC on the display will be master.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function call in ST

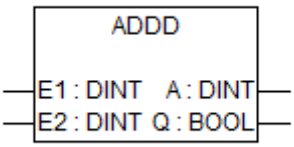
```
RTC_SYNC_DISPLAY(
 EN:= RTC_Sync_Display_EN,
 SYNC:= RTC_Sync_Display_SYNC);


RTC_Sync_Display_DONE:= RTC_Sync_Display.DONE;
RTC_Sync_Display_ERR:= RTC_Sync_Display.ERR;
RTC_Sync_Display_ERNO:= RTC_Sync_Display.ERNO;
```

1.5.4.32 Series90 AC500 library

Library file name: **Serie90_AC500_Vx.lib**

1.5.4.32.1 Function blocks

ADDD

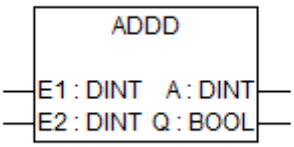


Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	-

The value of the operand at input E1 is added to the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (-2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q. The inputs and outputs can neither be duplicated nor negated.

Input and output description

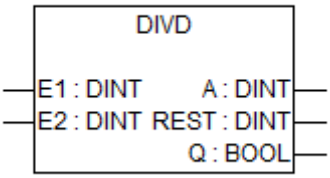


Parameter	Type	Data type	Description
E1	Input	DINT	Summand 1
E2	Input	DINT	Summand 2
A	Output	DINT	Total
Q	Output	BOOL	Total, limited

Function call in ST

```
ADDD1(E1 := ADDD_E1, E2 := ADDD_E2);
ADDD_Q:=ADDD1.Q;
ADDD_A:=ADDD1.A;
```

DIVD



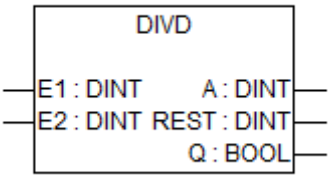
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	-

The value of the operand at input E1 is divided by the value of the operand at input E2 and the result is assigned to the operand at output A. The remainder is assigned to the operand at output REST. If a remainder is produced, the result will always be rounded down. If the result lies outside of the permissible number range, it will be limited to the maximum or minimum value of the number range: -2147483647 (8000 0001H) … 2147483647 (7FFF FFFFH). If a limiting has been performed, a TRUE signal is assigned to the binary operand at output Q and the value 0 is assigned to output REST. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

Division by »zero« is therefore also signalized at the binary output Q.

The inputs and outputs can neither be duplicated nor negated.

Input and output Description



Instance		DIVD	Instance name
E1	Input	DINT	Dividend
E2	Input	DINT	Divisor
A	Output	DINT	Result (quotient)
REST	Output	DINT	Remainder
Q	Output	BOOL	Result limited

Remainder handling

If the division results in a remainder, this is available at the double word output REST. The result of the division is always rounded down if a remainder occurs.

Example:

3 : 3 = 1 Remainder 0

4 : 3 = 1 Remainder 1

5 : 3 = 1 Remainder 2

6 : 3 = 2 Remainder 0

As the remainder is available at the output REST, the user can compare this to the divisor and can round the result at output A according to his own requirements.

Example:

Remainder > divisor/2 → round up the result at output A.

Division by »zero«

If the divisor has the value »zero«, the positive or negative limit of the number range is assigned to output A.

For the division by »zero« the following applies:

A = -2147483647 (8000 0001H) if dividend is negative

A = +2147483647 (7FFF FFFFH) if dividend is positive

REST = 0 Output for the remainder

Q = TRUE Output to signalize that the value at output A has been limited

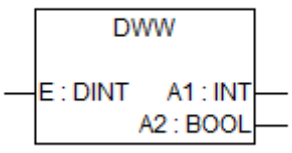
Invalid result value

If the invalid value 8000 0000H is the result of the division, this will be corrected to the permissible limit 8000 0001H (-2 147 483 647), the binary output Q will be set to the value TRUE and the output REST will be set to the value 0.

Function call in ST

```
DIVD1(E1 := DIVD_E1,E2 := DIVD_E2);
DIVD_REST:=DIVD1.REST;
DIVD_Q:=DIVD1.Q;
DIVD_A:=DIVD1.A;
```

DWW



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	-

The value of the double word operand at input E1 is converted to a word variable and the result is assigned to the word operand at output A1.

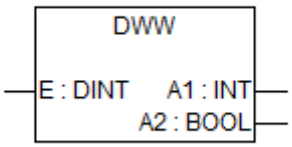
The result is limited to the maximum or minimum number range.

- max. number range: +32767 (7FFFH)
- min. number range: -32767 (8001H)

If limiting occurred, a TRUE signal is assigned to the binary operand at output A2. If no limiting occurred, a FALSE signal is assigned to the binary operand at output A2.

The input and the outputs can neither be duplicated nor negated.

Input and output description

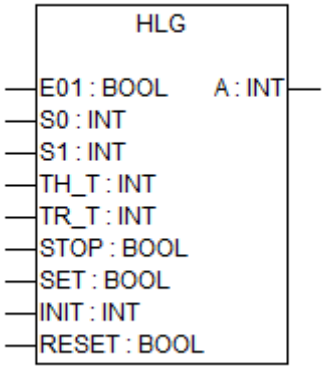


Instance		DWW	Instance name
E1	Input	DINT	Double word variable to be converted
A1	Output	INT	Result of conversion, word variable
A2	Output	BOOL	Result limited

Function call in ST

```
DWW1(E := DWW_E);
DWW_A2:=DWW1.A2;
DWW_A1:=DWW1.A1;
```

HLG



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	-

The ramp function generator is used for the rampshaped adaption of the current actual value at the output to a specified set point.

The value at the HLG output is adapted linearly from the current actual value to the specified set point with the slope y'.

In doing so, the value at the output precisely runs through the amount of the set point during the time TH or TR. If the value at the HLG output has reached the set point, it no longer changes unless a new set point is specified.

The inputs and the output can neither be duplicated nor negated/inverted.

The slope y' of the ramp results from the specified time TH (start up time) or TR (return time) and the amount of the set point:

Slope y' =	Set point amount
	TH or TR

The slope is

- positive, if set point > actual value
- negative, if set point < actual value
- 0, if set point = 0

Therefore, the specified set point has two functions

- its amount defines the slope of the ramp in conjunction with the specified time TH or TR;
- it represents the value to which the current actual value must be adapted in a ramp shape.

The user can specify the start up time TH and the return time TR separately. The direction of the slope is defined on the basis of the set point. The direction of the slope then defines whether or not the running time TH or TR is used.

Slope y' positive → TH, i.e. the ramp runs upwards.

Slope y' negative → TR, i.e. the ramp runs downwards.

The start up time TH and the return time TR must be scaled to the program cycle time TZ, i.e. the following must be specified at the corresponding function block inputs:

- Start up time: TH/TZ
- Return time: TR/TZ

The times are specified in milliseconds. The following applies to the time constants TH or TR:

0 ≤ TH ≤ 32767

0 ≤ TR ≤ 32767

Two set points can be planned (S0 and S1), whereby one of these set points is selected by the binary input E01 (set point selection).

The set points can assume the following values:

-32767 ≤ set point ≤ +32767

At any time, the output of the ramp function generator can be

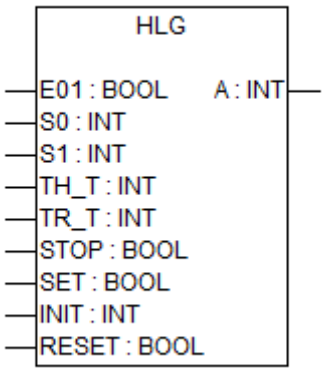
- stopped at the current value
- set to an initial value
- reset (output = 0)

The STOP input has the highest priority and the R input has the lowest.

The values at the HLG inputs can be changed at any time in the user program. In this way, any (non-linear) adaption to the set point can be realized on the basis of the linear adaption of the actual value.

> *Set point = 0 means that the slope of the ramp is also 0, i. e. the current actual value does not change. If it is intended to switch from an actual value unequal to 0 to an actual value of 0, a set point unequal to 0 must be specified and the output of the ramp function generator must be limited to 0 by a subsequent limiter. (On interpolation, the rounding transitions are based on calculation of integral numbers only).*

Input description



E01

Data type	Default value	Range	Unit
BOOL	-	-	-

One of the two set points is selected with input E01.

E01 = FALSE → set point S0

E01 = TRUE → set point S1

S0

Data type	Default value	Range	Unit
INT	-	-	-

The set point 0 is specified at input S0.

S1

Data type	Default value	Range	Unit
INT	-	-	-

The set point 1 is specified at input S1.

TH_T

Data type	Default value	Range	Unit
INT	-	-	-

The start up time is specified at input TH_T. At the same time, the start up time TH must be scaled to the cycle time T.

TR_T

Data type	Default value	Range	Unit
INT	-	-	-

STOP

Data type	Default value	Range	Unit
BOOL	-	-	-

The output can be latched to the current value by means of the STOP input.

STOP = FALSE → Output not latched

STOP = TRUE → Output is latched

The STOP input has higher priority than the inputs S and R.

S

Data type	Default value	Range	Unit
BOOL	-	-	-

INIT

Data type	Default value	Range	Unit
INT	-	-	-

The initial value to which the output is to be set if required is specified at input INIT.

R

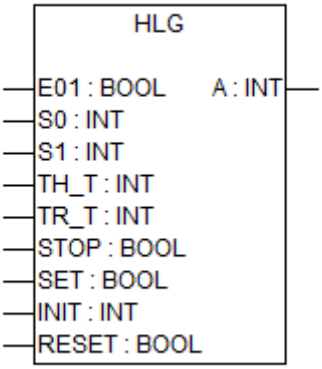
Data type	Default value	Range	Unit
BOOL	-	-	-

The output can be set to the value 0 with the input R.

R = FALSE → Output is not reset

R = TRUE → Output is reset to value 0.

Output description



A

Data type	Default value	Range	Unit
BOOL	-	-	-

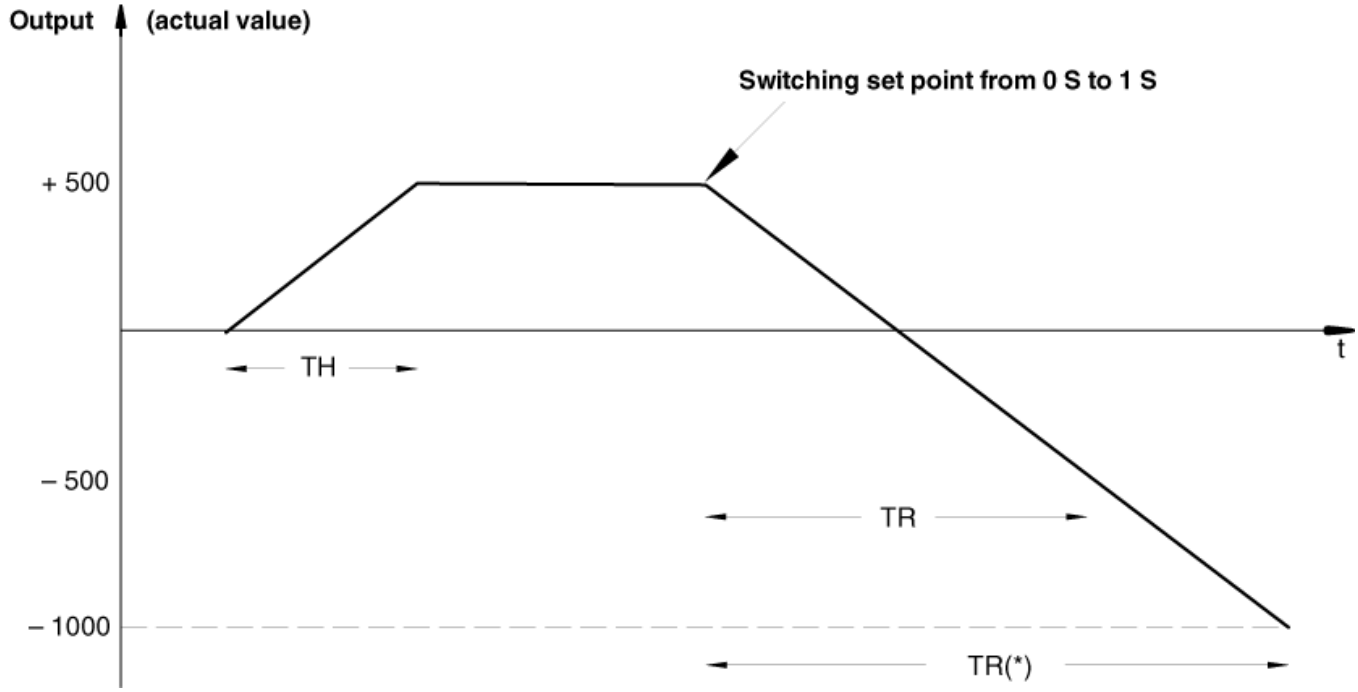
Output of the function block.

Example 1

The actual value is to be changed from 0 to the set point +500 (set point amount 500) and then from +500 to the set point -1000 (set point amount 1000).

The start up time is TH and the return time TR.

0 S: 500 1 S: -1000



TR(*) is the actual time until the actual value has reached -1000. During the time TH or TR, the actual value changes by the amount of the applied set point.

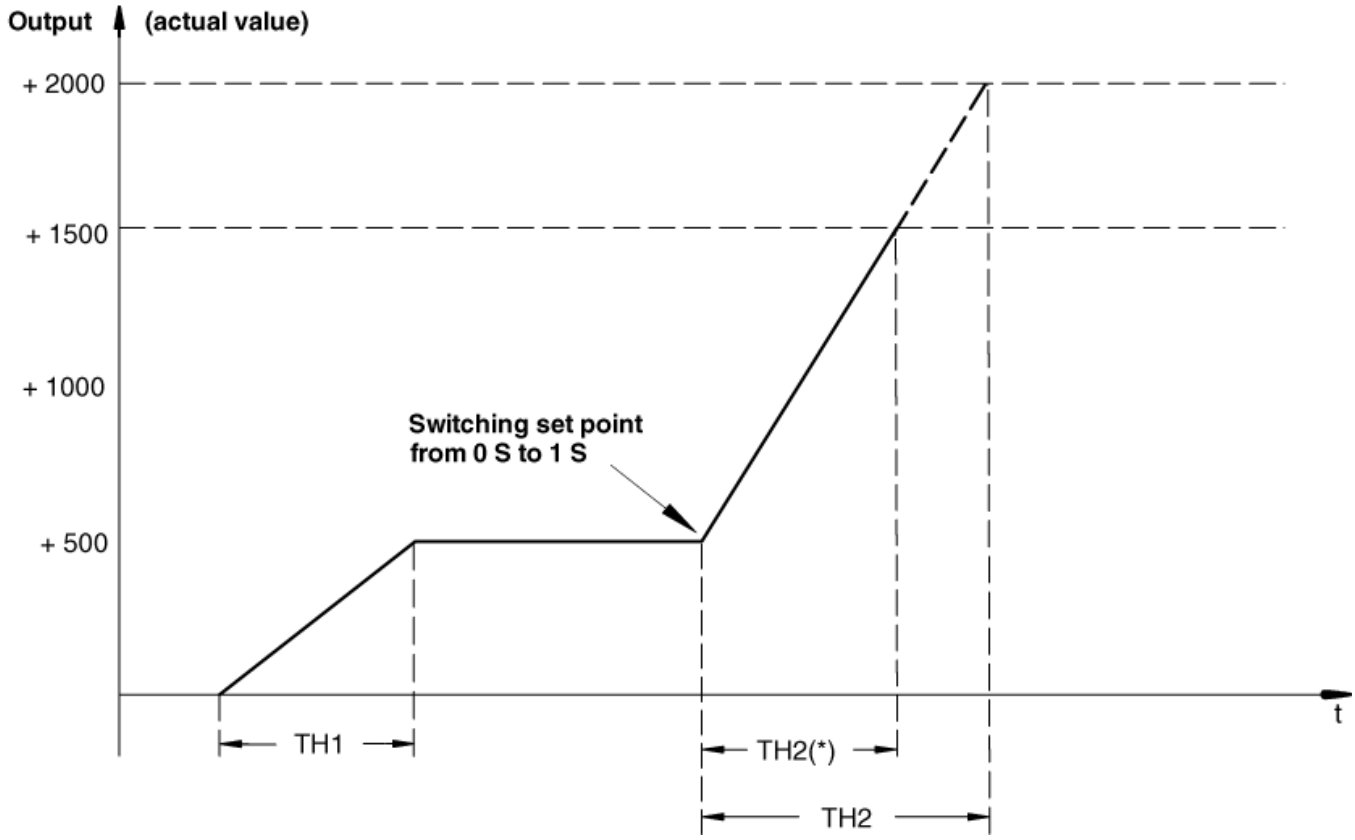
Example 2

The actual value is to be changed from 0 to the set point +500 (set point amount 500) and then from +500 to the set point 1500 (set point amount 1500).

The start up time is TH and the return time TR.

0 S: 500

1 S: 1500

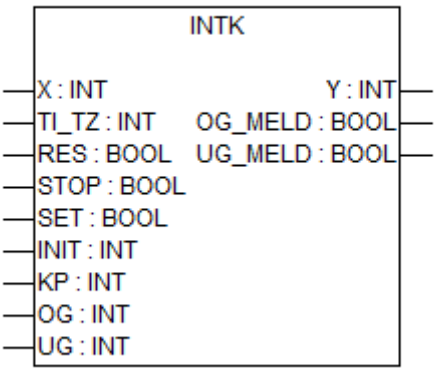


TR(*) is the actual time until the actual value has reached 1500. During the time TH or TR, the actual value changes by the amount of the applied set point.

Function call in ST

```
HLG1(E01   := HLG_E01,
S1    := HLG_S1,S0    := HLG_S0,
TH_T  := HLG_THT,
TR_T  := HLG_TRT,
STOP  := HLG_STOP,
SET   := HLG_SET,
INIT  := HLG_INIT,
RESET := HLG_RES);
HLG_A    :=    HLG1.A;
```

INTK



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	-

The function block generates the integral of the controlled variable X multiplied by the coefficient of proportionality KP. The integrator output Y can be manipulated as follows:

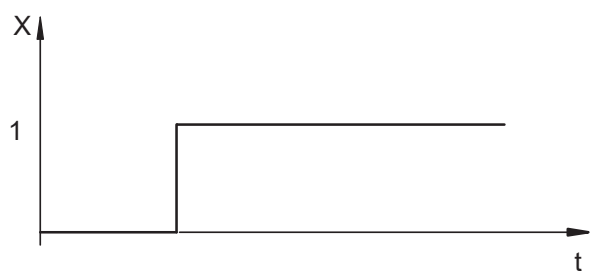
- It can be set to the value 0 by a 1 signal at input RES (reset).
- It can be latched to a current value by a TRUE signal at input STOP.
- It can be set to the initial value at input INIT by a TRUE signal at input SET.
- It can be limited to a maximum value specified at input OG (high limit).
- It can be limited to a minimum value specified at input UG (low limit).

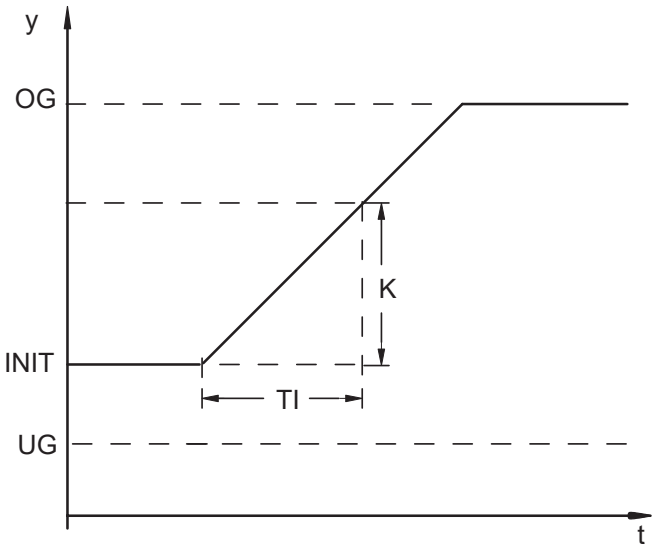
The inputs and outputs can neither be duplicated nor negated/inverted.

Transfer function

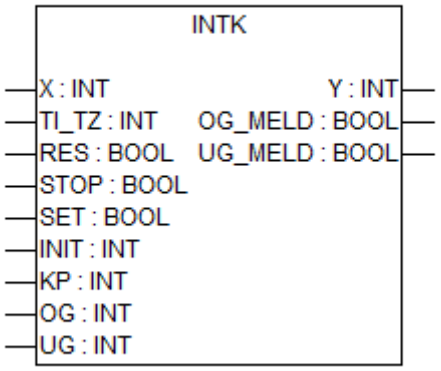
$$F(s) = \frac{KP}{s * T}$$

Transient function:





Input description



X

Data type	Default value	Range	Unit
INT	-	-	-

The operand for the controlled variable is specified at input X.

T1_TZ

Data type	Default value	Range	Unit
INT	-	-	-

The integration time is specified at input T1_TZ. In this case, it must be scaled to the cycle time. During the time T1, the output Y of the integrator changes by the value KP * X.

Range of values: 0 ≤ T1_TZ ≤ 328

- If values are specified which are beyond the admissible range of values, the PLC generally uses the value 328.
- A large integration time (T1) can be achieved by choosing a large cycle time, too. If the function block is used within a run number block, the cycle time of the run number block is valid for INTK and not the cycle time of the PLC program.

RES

Data type	Default value	Range	Unit
BOOL	-	-	-

The output Y can be reset to the value 0 with the input RES. Integration then begins as from the value 0.

STOP

Data type	Default value	Range	Unit
BOOL	-	-	-

The output can be latched to the current value by means of the STOP input.

STOP = FALSE → Output not latched

STOP = TRUE → Output is latched

The STOP input has higher priority than the inputs S and R.

SET

Data type	Default value	Range	Unit
BOOL	-	-	-

With the input SET, the manipulated value Y can be set to the initial value at input INIT. Integration then begins as from the initial value.

SET = FALSE

→ No setting

SET = TRUE

→ Output Y is set to the specified initial value.

*) Priority sequence for the inputs STOP, SET and RES:

RES highest priority

STOP

SET lowest priority

INIT

Data type	Default value	Range	Unit
INT	-	-	-

The initial value to which the output is to be set if required is specified at input INIT.

KP

Data type	Default value	Range	Unit
INT	-	-	-

The coefficient of proportionality is specified at input KP. It serves to weight the controlled variable at input X. Weighting is achieved by multiplying the controlled variable by the coefficient of proportionality. The coefficient of proportionality is specified as a percentage.

Example:

KP	is equal to	Description
1	1 percent	The function block multiplies the value at input X by the factor 0.01
55	55 percent	The function block multiplies the value at input X by the factor 0.55
100	100 percent	The function block multiplies the value at input X by the factor 1
1000	1000 percent	The function block multiplies the value at input X by the factor 10
-100	-100 percent	The function block multiplies the value at input X by the factor -1

OG

Data type	Default value	Range	Unit
INT			

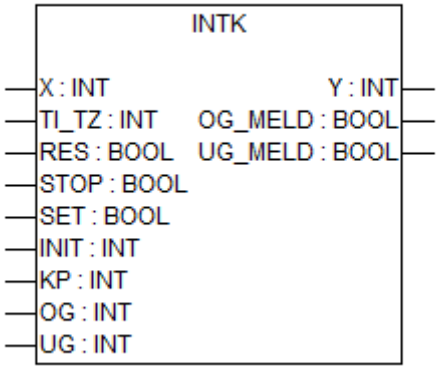
The manipulated variable Y can be limited to a range of values. The high limit for the manipulated variable Y is specified at input OG.

UG

Data type	Default value	Range	Unit
INT	-	-	-

The manipulated variable Y can be limited to a range of values. The low limit for the manipulated variable Y is specified at input UG.

Output description



OG_MELD

Data type	Default value	Range	Unit
BOOL	-	-	-

Whether the value at output Y has reached the specified high limit is signalized at output OG_MELD. Integration is stopped automatically when the limit is reached.

OG_MELD = FALSE

→ Output Y has not reached the limit (yet).

OG_MELD = TRUE

→ Output Y has reached the limit.

UG_MELD

Data type	Default value	Range	Unit
BOOL	-	-	-

Whether the value at output Y has reached the specified low limit is signalized at output UG_MELD. Integration is stopped automatically when the limit is reached.

UG_MELD = FALSE

→ Output Y has not reached the limit (yet).

UG_MELD = TRUE

→ Output Y has reached the limit.

Y

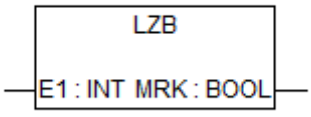
Data type	Default value	Range	Unit
INT	-	-	-

The manipulated variable (output value of the integrator) is provided at the output Y.

Function call in ST

```
INTK1(X := INTK_X, T1_TZ := INTK_T1TZ,
  RES := INTK_RES, STOP := INTK_STOP,
SET := INTK_SET, INIT := INTK_INIT,
KP := INTK_KP, OG := INTK_OG, UG := INTK_UG);
INTK_MOG := INTK1.OG_MELD;
INTK_MUG := INTK1.UG_MELD;
INTK_Y := INTK1.Y;
```

LZB

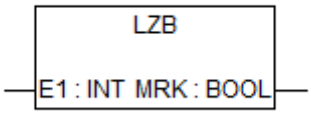


Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block with historical values
Group	-

The function block controls processing of a program part. This program part is called run number function block and begins with the function block LZB and ends with the appropriate target label specified at the function block output MRK. This program part is processed as follows depending on the value of the operand at input E1:

E1 = 0:	Program part is not processed.
E1 = 1:	Program part is processed during every cycle.
E1 = 2:	Program part is processed during every second cycle.
:	:
E1 = n:	Program part is processed during every nth cycle.

Input description



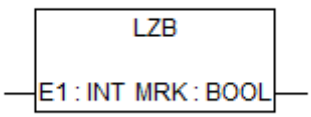
E1

Data type	Default value	Range	Unit
INT	-	-	-

This program part is processed as follows depending on the value of the operand at input E1:

E1 = 0:	Program part is not processed.
E1 = 1:	Program part is processed during every cycle.
E1 = 2:	Program part is processed during every second cycle.
:	:
E1 = n:	Program part is processed during every nth cycle.

Output description



MRK

Data type	Default value	Range	Unit
BOOL	-	-	-

At this output, a jump instruction with a corresponding jump destination must be specified. Output MRK only signalizes, whether the subsequent program part is processed or not.

The following applies:

MRK = FALSE

→ Processing of program part

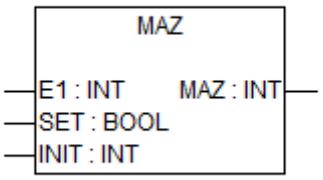
MRK = TRUE

→ No processing of program part

Function call in ST

```
LZB(E1 := LZB_E1);
IF (LZB1.MRK)
       THEN .....;
END_IF
```

MAZ



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	-

This function block determines the maximum value of a signal up to the current point of time by evaluating the time behavior of the signal.

The value of the operand at input E1 is compared to the previously occurred maximum value. If the input value E1 is higher than the previously occurred maximum, the input value is the new maximum value and is assigned to the operand at output MAZ.

If the input value E1 is less than the previously occurred maximum value, the previous maximum value is assigned to the output.

Output MAZ is set to the value of the operand at input INIT (initial value) with the FALSE → TRUE edge at binary input SET.

The following applies:

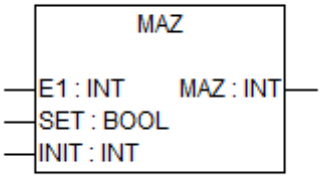
E1 < MAZ → MAZ = MAZ

E1 > MAZ → MAZ = E1

SET = FALSE→TRUE edge → MAZ = INIT

The inputs and the output can not be negated/inverted.

Input and output description

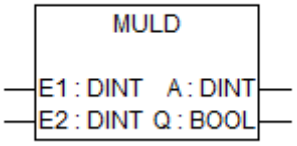


Instance		MAZ	Instance name
E1	Input	INT	Input value whose time maximum has to be determined
SET	Input	BOOL	Set input
INIT	Input	INT	Initial value
MAZ	Output	INT	Result of conversion, word variable

Function call in ST

```
IF E1 > MAZ THEN
 MAZ
   := E1; (* new max value *)
END_IF;

   IF SET
   AND NOT SET_old THEN
(* rising
   edge of input SET *)
MAZ
   := INIT; (* set INIT value *)
END_IF;
SET_old
   := SET; (* for edge detection *)
MAZ1(E1 := MAZ_E1, SET := MAZ_SET,INIT := MAZ_INIT);
MAZ_MAZ:=MAZ1.MAZ;
```

MULD



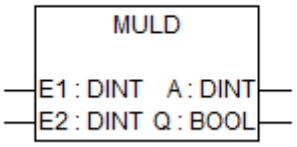
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	Group/Subgroup

The value of the operand at input E1 is multiplied by the value of the operand at input E2 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: -2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

The inputs and outputs can neither be duplicated nor negated.

Input and output description

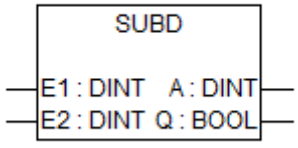


Instance		MULD	Instance name
E1	Input	DINT	Multiplicand
E2	Input	DINT	Multiplier
A	Output	DINT	Result (product)
Q	Output	BOOL	Result limited

Function call in ST

```
MULD1(E1 := MULD_E1, E2 := MULD_E2);
  MULD_Q:=MULD1.Q;
MULD_A:=MULD1.A;
```

SUBD



The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: -2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

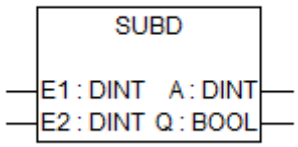
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block without historical values
Group	-

The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range (Number range: -2147483647 … 2147483647). If limiting occurred, a TRUE signal is assigned to the binary operand at output Q. If no limiting occurred, a FALSE signal is assigned to the binary operand at output Q.

The inputs and outputs can neither be duplicated nor negated.

Input and output description

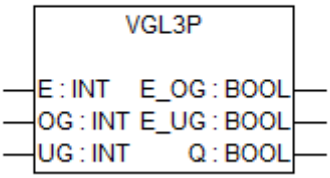


Instance		SUBD	Instance name
E1	Input	DINT	Minuend
E2	Input	DINT	Subtrahend
A	Output	DINT	Result (difference)
Q	Output	BOOL	Result limited

Function call in ST

```
SUBD1(E1 := SUBD_E1, E2 := SUBD_E2);
  SUBD_Q:=SUBD1.Q;
  SUBD_A:=SUBD1.A;
```

VGL3P



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V1.0
Type	Function block without historical values
Group	-

The value of the operand at input E is compared to the values of the operands at the inputs OG and UG.

The possible results are signalled at the outputs E_OG, E_UG and Q.

The following applies:

E < UG

→ E_OG = FALSE, E_UG = TRUE, Q = FALSE

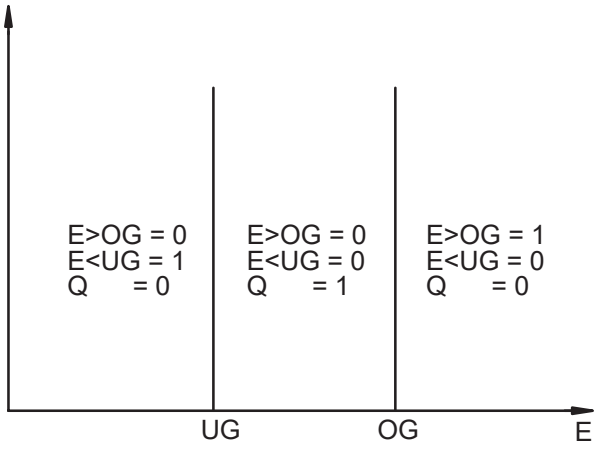
UG ≤ E ≤ OG

→ E_OG = FALSE, E_UG = FALSE, Q = TRUE

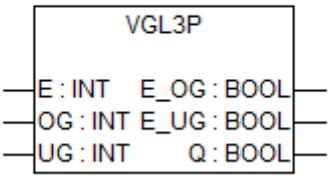
E > OG

→ E_OG = TRUE, E_UG = FALSE, Q = FALSE

The inputs and outputs can neither be duplicated nor negated/inverted.



Input and output description

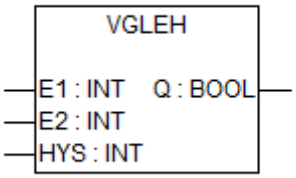


Instance		VGL3P	Instance name
E	Input	INT	Input value
OG	Input	INT	High limit
UG	Input	INT	Low limit
E_OG	Output	BOOL	Value > high limit
E_UG	Output	BOOL	Value < low limit
Q	Output	BOOL	Low limit ≤ input value ≤ high limit

Function call in ST

```
VGL3P1(E := V3P_E, OG := V3P_OG,
  UG := V3P_UG);
  V3P_EUG := VGL3P1.E_UG;
V3P_Q := VGL3P1.Q
V3P_EOG := VGL3P1.E_OG;
```

VGLEH



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function block with historical values
Group	-

The values of the operands at the inputs E1 and E2 are compared to each other. Taking the hysteresis at input HSY into account, the result is signaled at output Q.

The following applies:

E1 ≥ E2

→ Q = 1

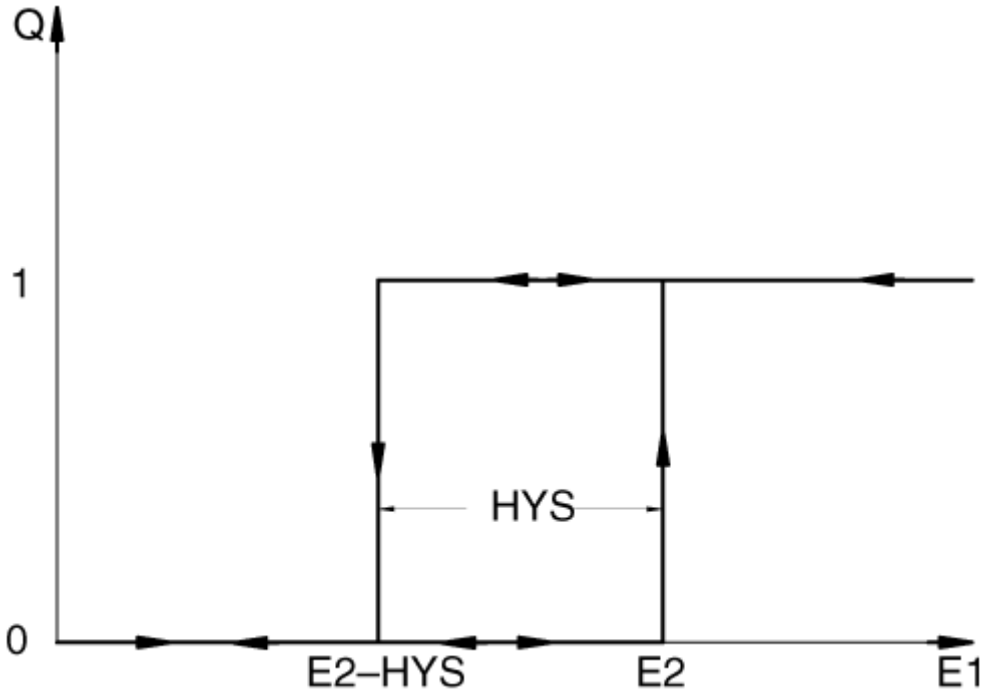
E1 < E2 "“ HYS

→ Q = FALSE

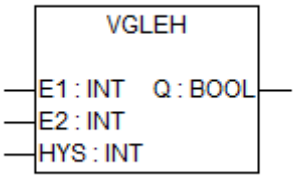
E2 - HYS ≤ E1 < E2

→ Q as in the previous cycle

The inputs can neither be duplicated nor inverted. The output can neither be duplicated nor inverted.



Input and output description



Instance		VGLEH	Instance name
E1	Input	INT	Input value 1
E2	Input	INT	Input value 2
HYS	Input	INT	Hysteresis
Q	Output	BOOL	Result of the comparison

Number range

Integer word (16 bits)

Low limit	8001H	-32767
High limit	7FFFH	+32767
Inadmissible value	8000H	---

The following especially applies here to the specification for the left edge of the hysteresis:

E2 - HYS ≥ -32767 (8001H)

In the two's complement arithmetic, the value 8000H (-32768) lies outside of the number range and is neither generated nor processed correctly by the PLC. If this forbidden value reaches the PLC

- by bit manipulations of the user or
- by reading from outside the PLC or
- by an indirect word constant

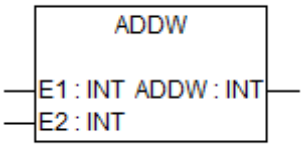
it is absolutely not allowed to carry out a negation or subtraction on this value.

Function call in ST

```
VGLEH1(E1 := VEH_E1, E2 := VEH_E2,HYS := VEH_HYS);
VEH_Q:=VGLEH1.Q;
```

1.5.4.32.2 Functions

ADDW



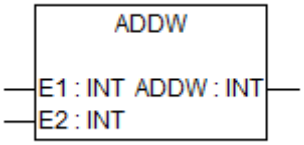
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

The values of the operands at the inputs of the function are added and the result is assigned to the operand at the output.

The result is limited to the maximum value 32767 and the minimum value ""32767.

The inputs and the output can neither be duplicated nor negated.

Input and output description

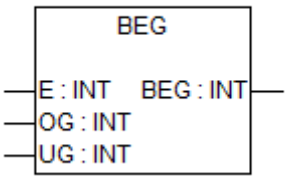


E1	Input	INT	Summand 1
E2	Input	INT	Summand 2
	Output	INT	Total

Function call in ST

```
ADDW_A:=ADDW(ADDW_E1,ADDW_E2);
```

BEG



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	Group/Subgroup

The value of the operand at input E is limited to the range between the upper and lower limits.

The upper limit is specified by the operand at the OG input and the lower limit is specified by the one at the UG input.

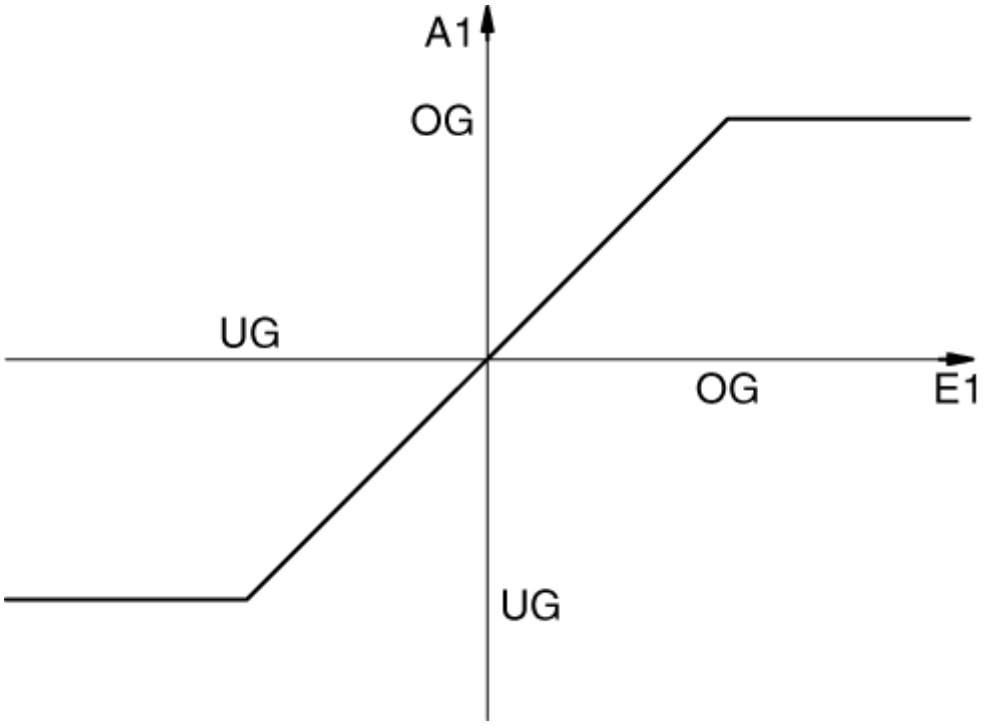
The following applies:

A = UG for E < UG

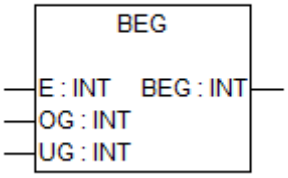
A = E for UG ≤ E ≤ OG

A = OG for E > OG

The inputs and the output can neither be duplicated nor negated.



Input and output description

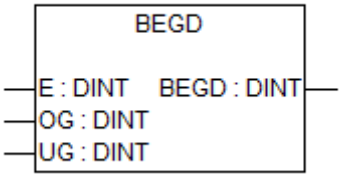


E	Input	INT	Input value
OG	Input	INT	Upper limit
UG	Input	INT	Lower limit
	Output	INT	Limited value

Function call in ST

```
BEG_A:=BEG(BEG_E, BEG_OG,BEG_UG);
```

BEGD



Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

The value of the operand at input E is limited to the range between the upper and lower limits.

The upper limit is specified by the operand at the OG input and the lower limit is specified by the one at the UG input.

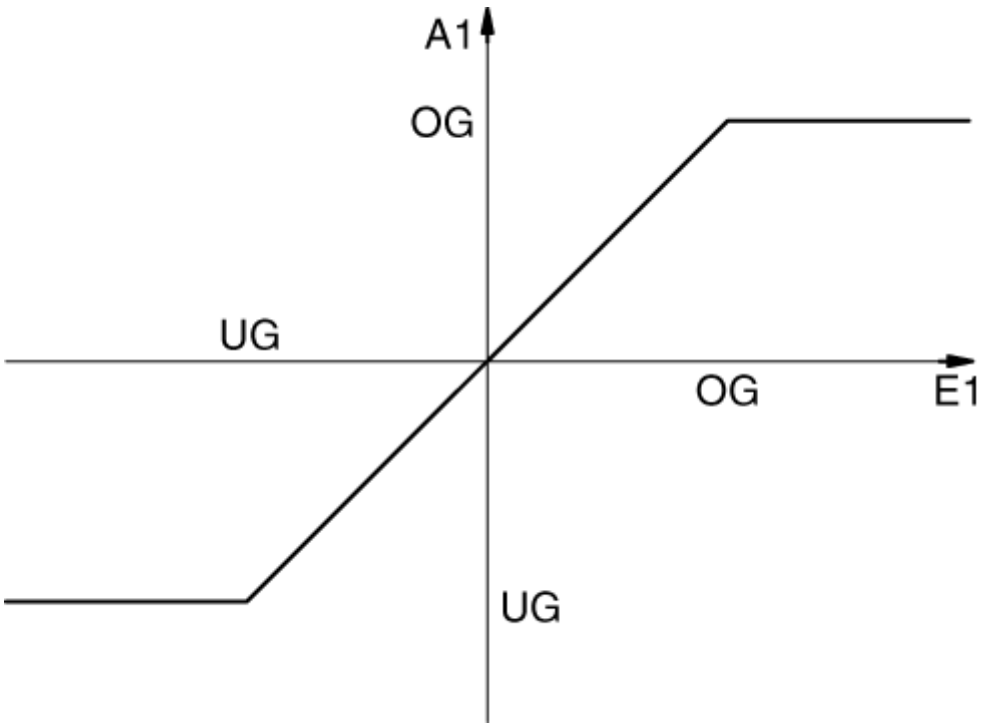
The following applies:

A = UG for E < UG

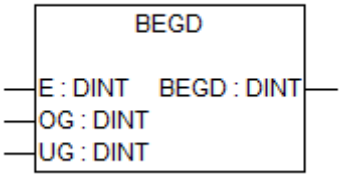
A = E for UG ≤ E ≤ OG

A = OG for E > OG

The inputs and the output can neither be duplicated nor negated.



Input and output description

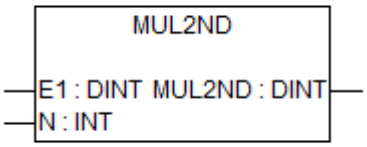


E	Input	DINT	Input value
OG	Input	DINT	Upper limit
UG	Input	DINT	Lower limit
	Output	DINT	Limited value

Function call in ST

```
BEGD_A:=BEGD(BEGD_E, BEGD_OG,BEGD_UG);
```

MUL2ND



The value of the operand at input E1 is shifted bitwise N times.

If the value at input N is positive, the value is shifted to the left. Each shift by 1 bit position corresponds to a multiplication of the current value by 2.

If the value at input N is negative, the value is shifted to the right. Each shift by 1 bit position corresponds to a division of the current value by 2.

The result is assigned to the operand at output A1.

Parameter	**Value**
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

The value of the operand at input E1 is shifted bitwise N times.

If the value at input N is positive, the value is shifted to the left. Each shift by 1 bit position corresponds to a multiplication of the current value by 2.

If the value at input N is negative, the value is shifted to the right. Each shift by 1 bit position corresponds to a division of the current value by 2.

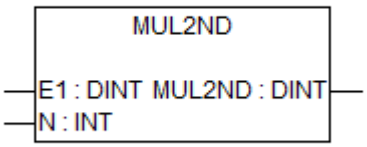
The result is assigned to the operand at output A1.

The inputs and the output can neither be duplicated nor negated.

Reasonable range for N: -30 ≤ N ≤ +30

If N = 0, the value at input E1 is passed directly to output A1.

Input and output description



E1	Input	DINT	Input operand
N	Input	INT	Quantity
	Output	DINT	Result

Sign of the value at input E1

The sign of value E1 is not influenced by the shift operation. I.e. the sign of the output value is always identical with the sign of the input value.

Shift to the left (Multiplication)

When the value at the input is shifted to the left, the released bit 0 is filled with 0. The sign bit (bit 31) is not changed because a limiting to the limit of the number range is performed before.

Limiting the value at output A1 when shifting to the left:

- The following applies to positive values at input E1: If bit 30 has a »1« and if shift operations still have to be carried out on the basis of the value at input N, these are no longer executed. Instead, the output is set to the limit of the positive number range. I.e. the limit has been reached in any case at the latest after 30 shifts.
  Limit value: Output A1 = +2147483647 (7FFFFFFFH).
- The following applies to negative values at input E1: If bit 30 has a »0« and if shift operations still have to be carried out on the basis of the value at input N, these are no longer executed. Instead, the output is set to the limit of the negative number range. I.e. the limit has been reached in any case at the latest after 30 shifts.
  Limit value: Output A1 = -2147483647 (80000001H).

Shift to the right (Division)

When shifting to the right, every bit moves to the right by one position. At the same time, the sign bit (bit 31) always retains its value. The released bit (bit 30) is filled in each case with the value of the sign bit.

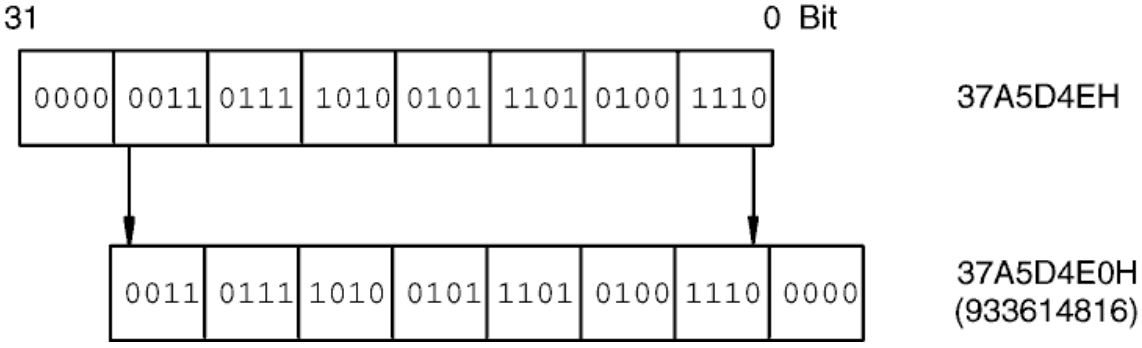
Limiting the value at the output when shifting to the right:

- The following applies to positive values at input E1: If now only bit 0 has a »1« and shift operations still have to be carried out because of the value at input N, the output will be set to the value 0. I.e. value 0 has been reached in any case at the latest after 30 shifts.
  Output A1 = 0.
- The following applies to negative values at input E1: If bit 0 ... bit 31 has a »1« as the result of the shift, the limit value (-1) has been reached. Further shifts have no effect. I.e. the value -1 has been reached at the latest at 31 shifts.
  Output A1 = -1 (FFFFFFFFH).

The inputs and the output can neither be duplicated nor negated.

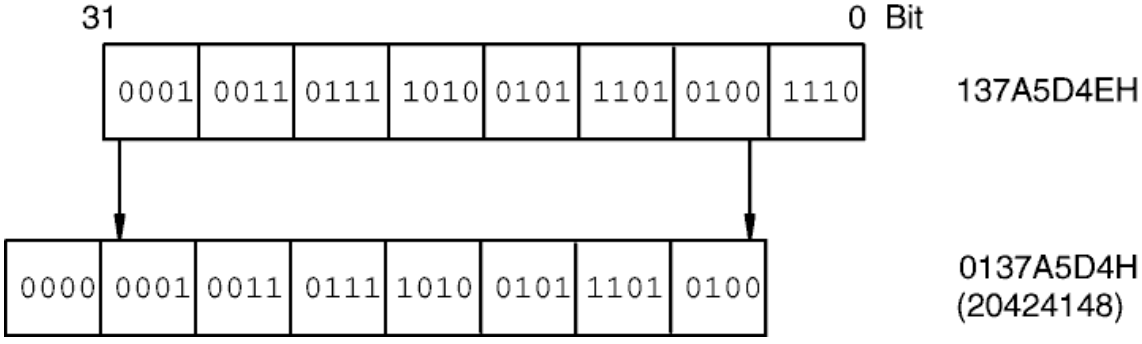
Examples

1. Input value E1 = 58350926 (37A5D4EH) Exponent N = 4 4 * Left shift



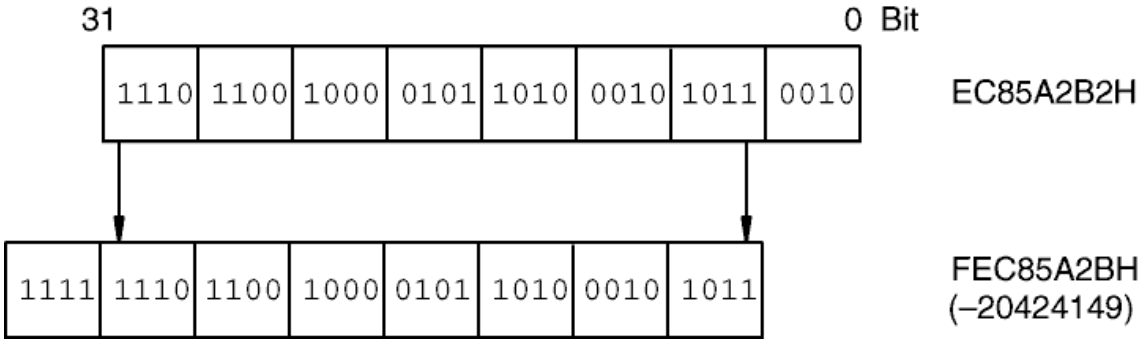
2. Input value E1 = 326786382 (137A5D4EH)

Exponent N = -4 4 * Right shift



3. Input value E1 = -326786382 (EC85A2B2H)

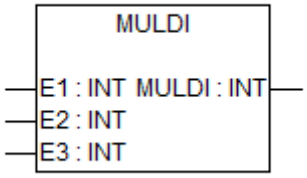
Exponent N = -4 4 * Right shift



Function call in ST

```
MUL2ND_A:=MUL2ND(MUL2ND_E1, MUL2ND_N);
```

MULDI



The operand value at input E1 is multiplied by the operand value at input E2, the intermediate result is divided by the operand value at E3 and then the result is assigned to output A.

The result is limited to the maximum or minimum value of the number range.

Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

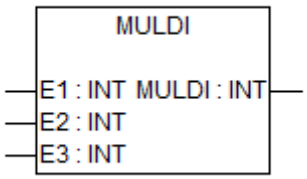
The operand value at input E1 is multiplied by the operand value at input E2, the intermediate result is divided by the operand value at E3 and then the result is assigned to output A.

Internal, the function block performs the multiplication and division with the accuracy of a double word (32 bit). Only when assigning the result to output A, the limiting to the accuracy of a word (16 bit) is carried out. If the remainder of the division is > 0,5, the result is rounded up. If a numerical overflow occurs during the division (e.g. division by zero), the correct signed limit of the number range is applied at output A.

The result is limited to the maximum value 32767 and the minimum value -32767.

The inputs and the output can neither be duplicated nor negated.

Input and output description

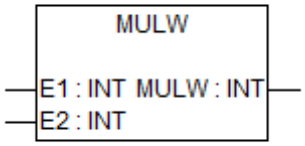


E1	Input	INT	Multiplicand
E2	Input	INT	Multiplier
E3	Input	INT	Divisor
	Output	INT	Result

Function call in ST

```
MULDI_A := MULDI(MULDI_E1, MULDI_E2,MULDI_E3);
```

MULW



The values of the operands at the inputs of the function are multiplied and the result is assigned to the operand at the output.

The result is limited to the maximum or minimum value of the number range.

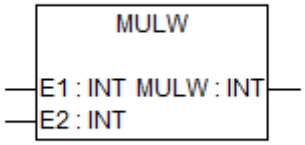
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

The values of the operands at the inputs of the function are multiplied and the result is assigned to the operand at the output.

The result is limited to the maximum value 32767 and the minimum value -32767.

The inputs and the output can neither be duplicated nor negated.

Input and output description

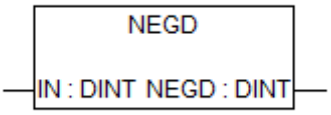


E1	Input	INT	Multiplicand
E2	Input	INT	Multiplier
	Output	INT	Result (product)

Function call in ST

```
MULW_A := MULW(MULW_E1, MULW_E2);
```

NEGD



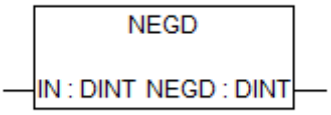
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	-

The value of the operand at input IN is negated and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range. (Number range: -2147483647 … 2147483647).

The input and the output can neither be duplicated nor negated.

Input and output description

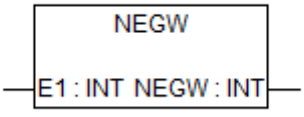


IN	Input	DINT	Input value
	Output	DINT	Negated value

Function call in ST

```
NEGD_A:=NEGD(NEGD_E1);
```

NEGW



The value of the operand at input E1 is negated and the result is assigned to the operand at output A.

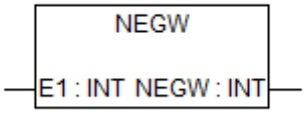
Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0
Type	Function
Group	Group/Subgroup

The value of the operand at input E1 is negated and the result is assigned to the operand at output A.

The result is limited to the maximum or minimum value of the number range. (Number range: -2147483647 … 2147483647).

The input and the output can neither be duplicated nor negated.

Input and output description

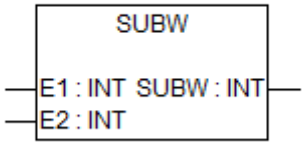


E1	Input	INT	Input value
	Output	INT	Negated value

Function call in ST

```
NEGW_A:=NEGW(NEGW_E1);
```

SUBW



The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

Parameter	Value
Included in library	Serie90_AC500_V10.lib
Available as of firmware	V2.0

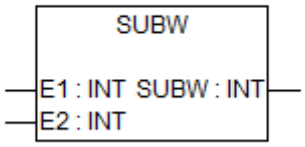
Parameter	Value
Type	Function
Group	-

The value of the operand at input E2 is subtracted from the value of the operand at input E1 and the result is assigned to the operand at output A.

The result is limited to the maximum value 32767 and the minimum value -32767.

The inputs and the output can neither be duplicated nor negated.

Input and output description



E1	Input	INT	Minuend
E2	Input	INT	Subtrahend; input can be dublicated
	Output	INT	Result (difference)

Function call in ST

```
SUBW_A:=SUBW(SUBW_E1, SUBW_E2);
```

1.5.4.33 Glossary

BOOL Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE BYTE belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 93: For integer data types the following range limits are valid:

TYPE	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 94: For integer data types the following range limits are valid:

TYPE	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 95: For integer data types the following range limits are valid:

TYPE	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on. For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function 'Function Blocks' must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block. As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated. In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blockscan have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

INT

INT belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 96: For integer data types the following range limits are valid:

TYPE	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 97: For integer data types the following range limits are valid:

TYPE	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

1.5.5 AC500 HA High Availability System

1.5.5.1 AC500 HA-CS31 based on serial communication

For the AC500 HA-CS31 high availability based on ABBs CS-31 Bus with serial communication (2 parallel lines) see High Availability System Technology Chapter in Automation Builder online documentation.

1.5.5.1.1 Introduction

Safety instructions

Consider the following before using the libraries:

- All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.
- Read the complete safety instructions of the user's manuals for the drives you are using, before installation and commissioning.
- Read all *Chapter 1.6.1.1 “Safety instructions” on page 3697* for the AC500 PLC.
- Read the user information of the devices and functions you are using, see Automation Builder online help.

Preconditions: HA-CS31 library

The library package has been released for the software and firmware versions listed in the readme file of Automation Builder only (see *“Help ➔ Automation Builder Release Notes”*) .

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the HA library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e.g. of example programs.

CAUTION!
Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

The function blocks contained in the library can be executed only in RUN mode of the PLC, but not in simulation mode.

1.5.5.1.2 AC500 High Availability CS31 system technology

Introduction

What is the AC500 High Availability CS31 system?

The AC500 High Availability CS31 system is designed for the demand of automation systems that require a higher availability, which is realized by redundant devices and communications. The redundancy concept reduces the risk of losing production due to failure of parts of the automation system and thereby minimizes scheduled idle times. For instance, control can be taken over by the secondary station if the primary station fails.

AC500 High Availability CS31 system helps to implement redundancy based on standard AC500 PLCs:

- CPU
- Field communication
- SCADA communication

HA-CS31 system

Greater availability to the field connection is provided by redundant fieldbus (CS31) connection to CI590-CS31-HA, a redundant CS31 slave as S500 I/O module. The CI590-CS31-HA includes two CS31 slave interfaces: One is connected to the CS31 master system on station A, the other is connected to the CS31 master system on station B. Further information on the device is provided in the device description for ↬ *Chapter 1.6.2.8.3.1 “CI590-CS31-HA” on page 4745*.

Both automation systems require activated HA-CS31 function blocks to achieve that the control sequences can be switched over bumplessly, meaning they continue in the event of failures. The total application program has to be made considering high availability.

The HA-CS31 system has hot-standby redundancy as a quality level:

- Both CPUs are running in parallel
- Both CPUs are continuously synchronized
- The fieldbus is redundant and continuously running (updated by both CPUs)
- The CI590 remote I/O decides itself which CPU and route to trust (voter) which ensures very fast switch over times while keeping the process running.

When to use AC500 High Availability CS31 system?

Centralized and particularly important system components require greater levels of availability. To keep the control sequences operational in the event of a failure, the mechanism of HA-CS31 system automatically switches over to the redundant backup system.

HA-CS31 system can be used in any situation in which system components require greater levels of availability by redundancy and automatic switch over, where a brief freeze of the system can be tolerated by the process while switching from one station to another. See ↬ *Chapter 1.5.5.1.2.4.1 “Use case and reaction time” on page 2004*.

AC500 HA system can be used to manage the following types of failure:

- Failure of CPU components:
  - Failure of the power supply unit
  - Failure of the terminal base of the CPU
  - Failure of the CS31 master
  - CPU failure due to hardware or software faults
- Failure of field device connection:
  - Breaks/short circuit in the bus cable for the data link or CS31 slave interface faults on CI590-CS31-HA module
- Failure of HMI connection (if using HMI and switches):
  - Switch
  - Connection between CPU and switch
  - Connection between switch and HMI

Requirements

Hardware

The communication interface module *Chapter 1.6.2.8.3.1 “CI590-CS31-HA” on page 4745* is required.

Two AC500 CPUs are required as central hardware components. Each CPU must be equipped with an integrated Ethernet port (or Ethernet communication module) and a connection for a CS31 master system. These two CPUs, called station A and station B, are linked by means of an Ethernet bus system through which they can exchange information. Connection to the peripheral devices is performed via two CS31 master systems - one on station A and another on station B.

AC500 I/O modules on CI590-CS31-HA are connected to both CS31 master systems. The CS31 slave interface enables switching from the first to the second interface in the event of a fault, thereby enabling process status data to be passed to the peripherals by the second CS31 master.

Until version V2.3.0 (HA_CS31_AC500_V23.lib) the library supported high availability only for I/O modules connected on CPU COM1 of the CS31 bus.

Only one PLC COM CS31 bus

The following diagram represents HA-CS31 system recommendation for applications using one CPU COM CS31 Bus only.

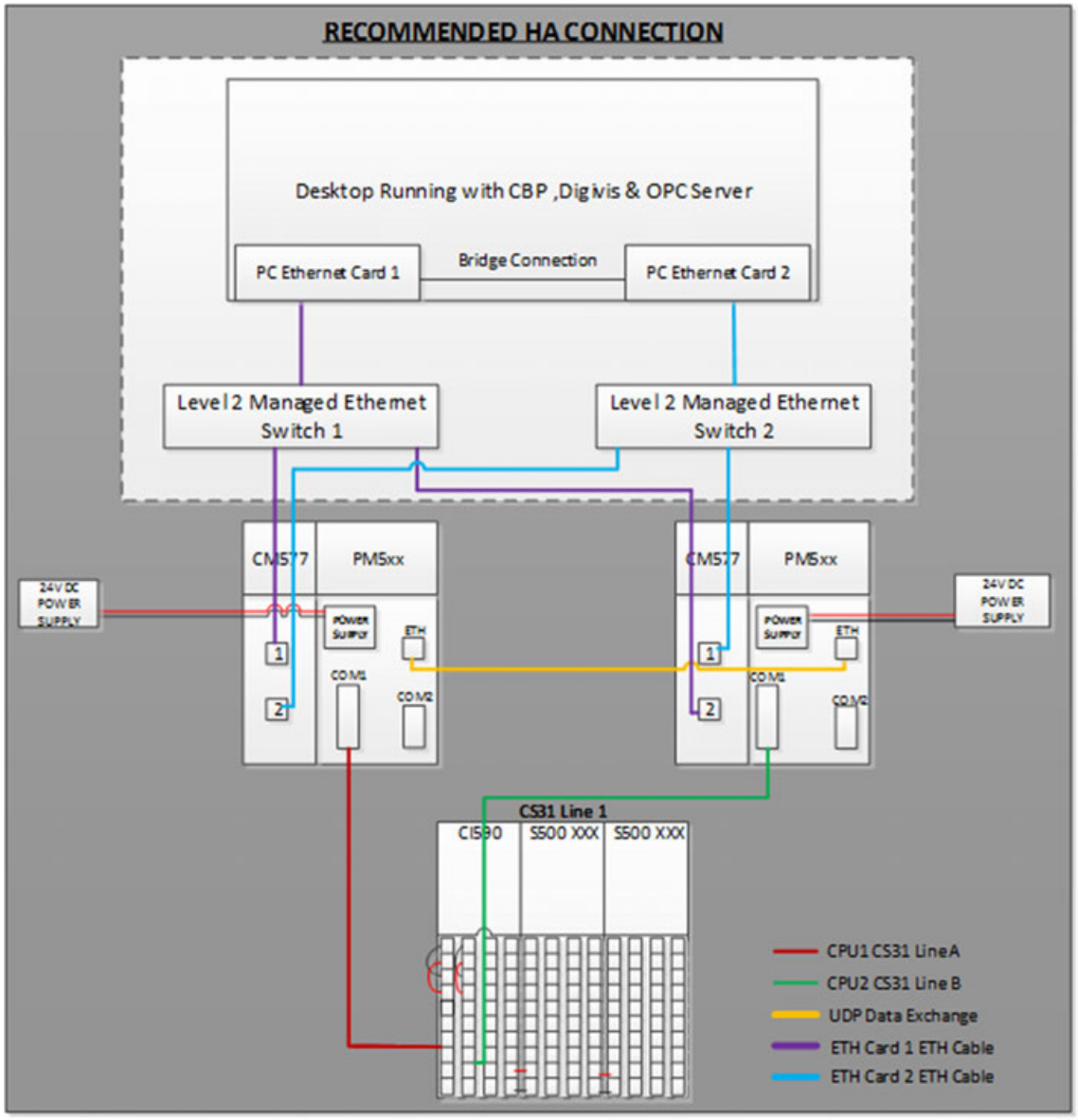


Fig. 31: Recommended high availability connection.

> *Blocks inside the doted box are not mandatory. However, if users require Ethernet redundancy with a control system/SCADA/ DigiVis 500, the customer has to use Level 2 Managed Ethernet switch.*

PLC COM CS31 bus and CM574-RS COM CS31 bus

As of HA-CS31 library version V2.4.0 (HA_CS31_AC500_V23.lib) and higher, high availability also supports the I/O modules connected on both CPU COM ports and CM574-RS COM ports.

The HA-CS31 library supports up to three CM574-RS modules and each CM574-RS supports two CS31 lines. So it supports a maximum of seven CS31 lines including CPU COM1 CS31 line.

> *PM595 supports maximum 2 CM574-RS modules. So it supports a maximum of 5 CS31 lines including CPU COM1 CS31 line.*

PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.

The following diagram represents one CPU COM CS31 Bus and only one CM574-RS COM CS31 bus. Users can add up to three CM574-RS as per the requirements:

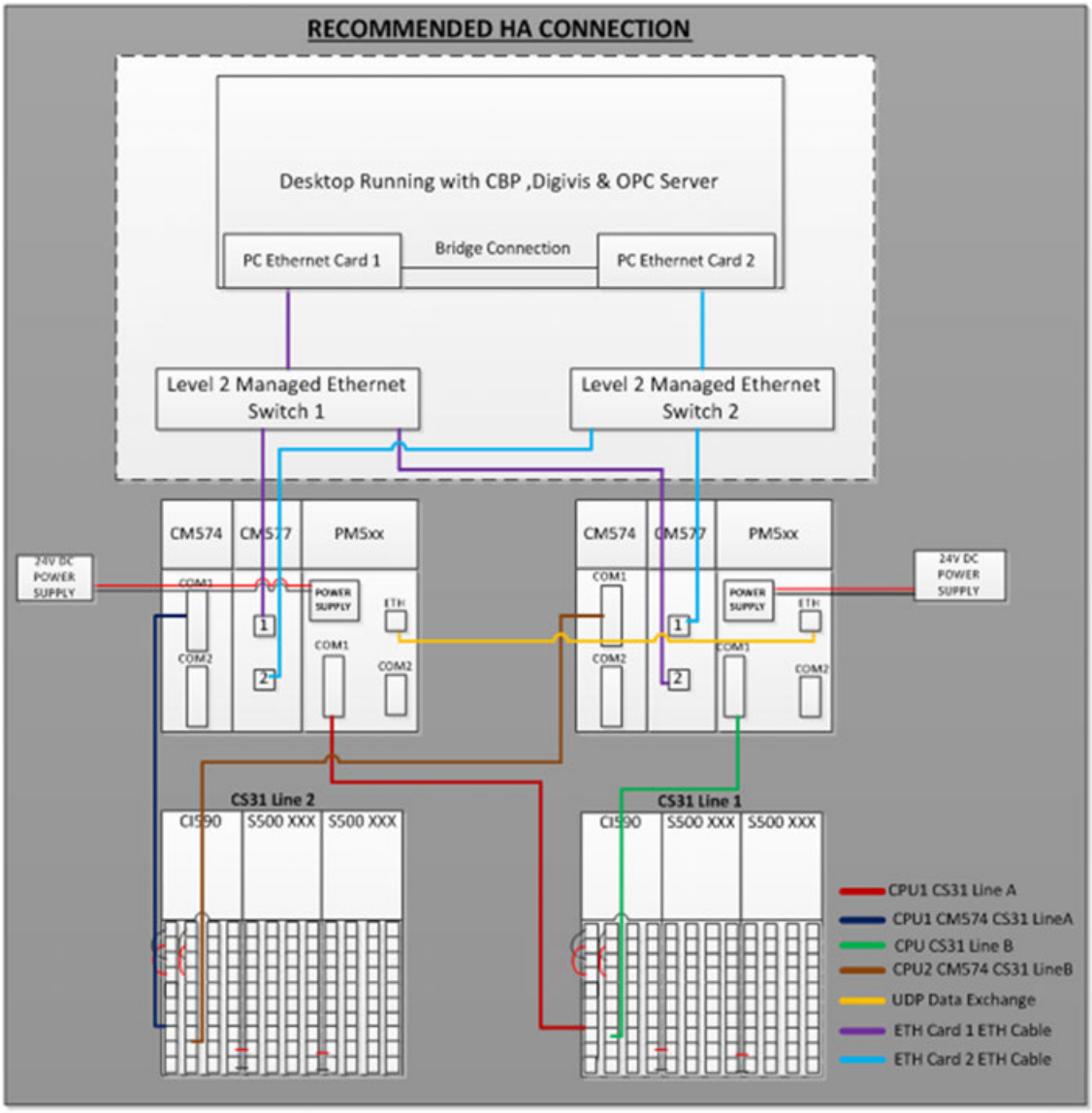


Fig. 32: Recommended high availability connection. Option 2.

Blocks inside the doted box are not mandatory. However, if the application requires Ethernet redundancy with DigiVis 500, the user has to use Level 2 Managed Ethernet switch.

Recommended connections for a HA-CS31 system

CPU redundancy is ensured by using two CPUs in combination with the HA-CS31 library. Redundancy of the Ethernet connection between CPUs and HMI can be achieved depending on the type of HMI and use of managed switches:

Sl no.	HMI (DigiVis 500/ CP6xx)	CPU Redundancy	Ethernet Redundancy	Recommendation
1	DigiVis 500 with two Ethernet ports on PC and Layer 2 managed switch. Refer to *Chapter 1.5.5.1.2.2.1.1.1 "DigiVis 500 with two Ethernet ports" on page 1987*.	Yes (with the HA-CS31 library)	Yes (complete Ethernet network is redundant with Layer 2 managed switches).	Recommended for users who require CPU and Ethernet network redundancy.
2	HMI CP6xx with two Ethernet ports on HMI with same IP and Layer 2 managed switch. Refer to *Chapter 1.5.5.1.2.2.1.1.2 "HMI CP6xx with two Ethernet ports" on page 1988*.	Yes (with the HA-CS31 library)	Yes (complete Ethernet network is redundant with Layer 2 managed switches, HMI IP switching functionality is achieved by 'Node Override IP' functionality within the script programming. Refer to Example_AC500_HA_CS31_V242.project, HA_CS31_Example_OnlyCPU_HMI)	Recommended for users who require CPU and Ethernet network redundancy.
3	DigiVis 500 with standard Ethernet switch (unmanaged switch). Refer to *Chapter 1.5.5.1.2.2.1.1.3 "DigiVis 500 with standard Ethernet switch" on page 1989*.	Yes (with the HA-CS31 library)	No	Recommended for users who require CPU redundancy but no Ethernet network redundancy.
4	HMI CP6xx with one Ethernet port on HMI with standard Ethernet switch (unmanaged switch). Refer to *Chapter 1.5.5.1.2.2.1.1.4 "HMI CP6xx with one/two Ethernet ports" on page 1990*.	Yes (with the HA-CS31 library)	No	Recommended for users who require CPU redundancy but no Ethernet network redundancy.

DigiVis 500 with two Ethernet ports

DigiVis 500 with two Ethernet ports on PC and Layer 2 managed switch.

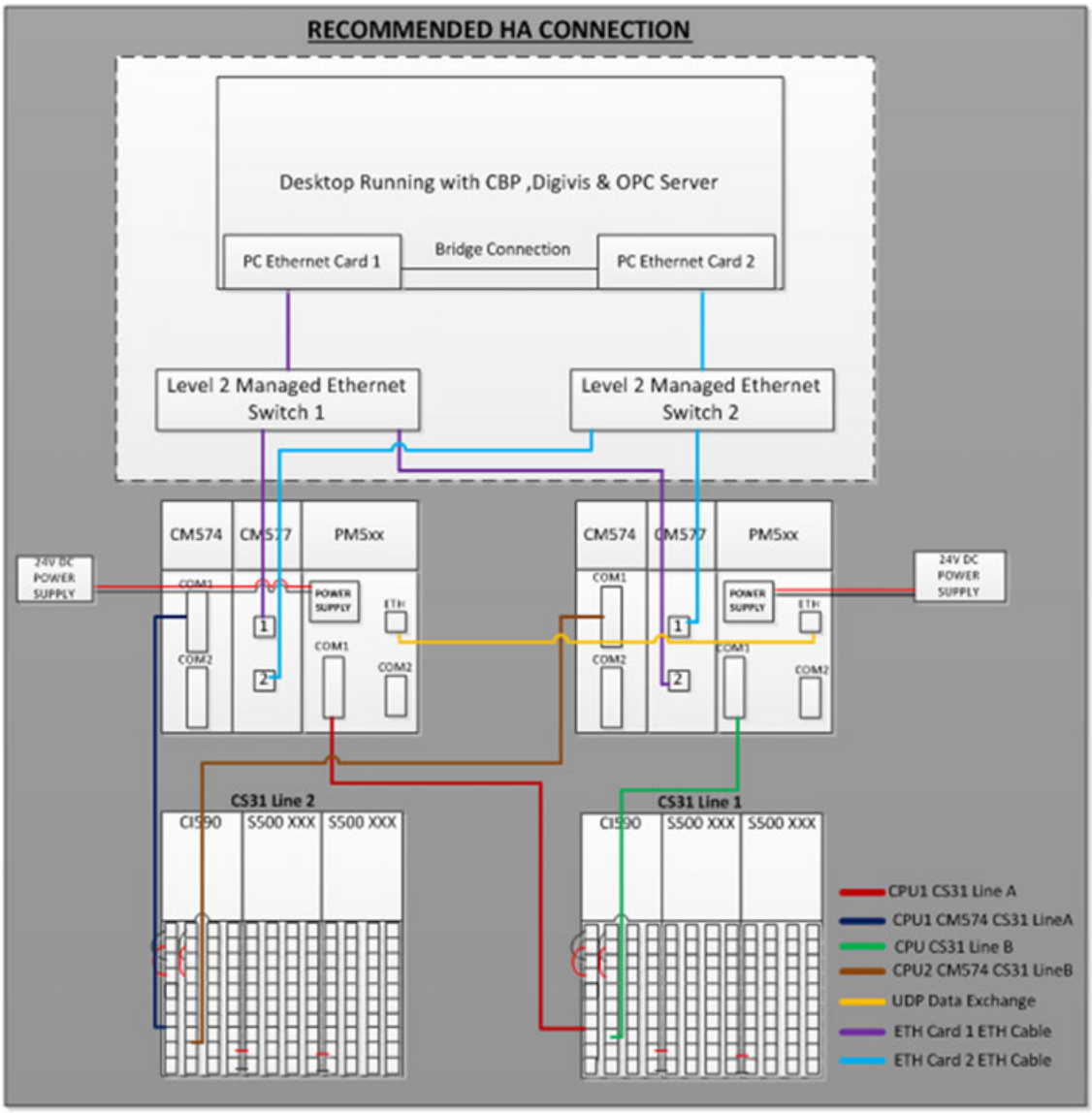


Fig. 33: Recommended high availability connection. Option 1.

HMI CP6xx with two Ethernet ports

HMI CP6xx with two Ethernet ports on HMI with same IP and Layer 2 managed switch:

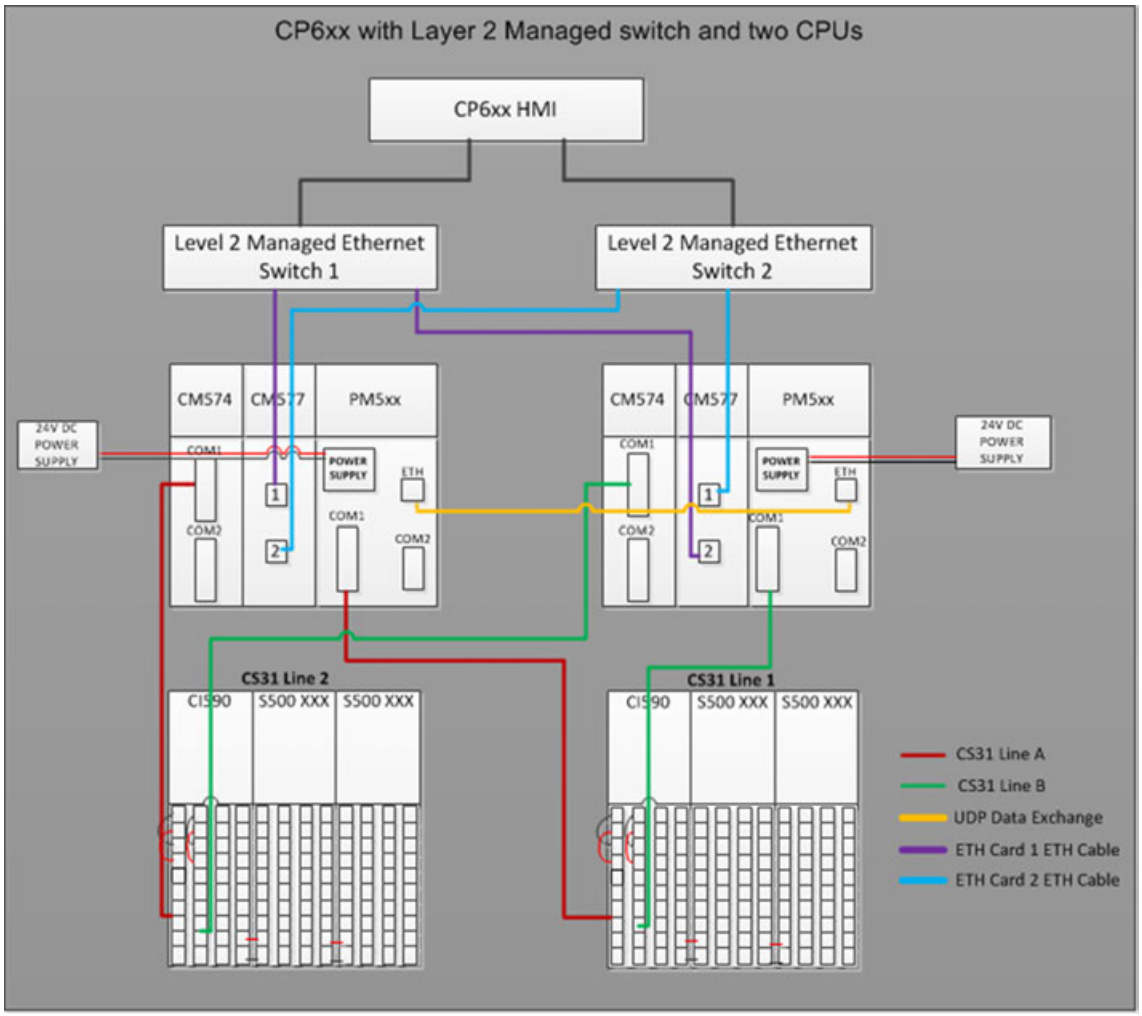


Fig. 34: Recommended high availability connection. Option 2.

Ethernet redundancy must be done by the application and Layer 2 managed switches.

HA example application program (Example_AC500_HA_CS31_V242.project) provides a sample application program for CP620 HMI. It has the required scripts to read data from the primary PLC. Also it provides an option for the user to select one of the two PLCs to read data. This is done using the IP override functionality. With minor modifications the HMI application program can be made application specific.

Data writing from HMI to PLC is not recommended as it will write data only to the primary CPU.

DigiVis 500 with standard Ethernet switch

DigiVis 500 with standard Ethernet switch (unmanaged switch):

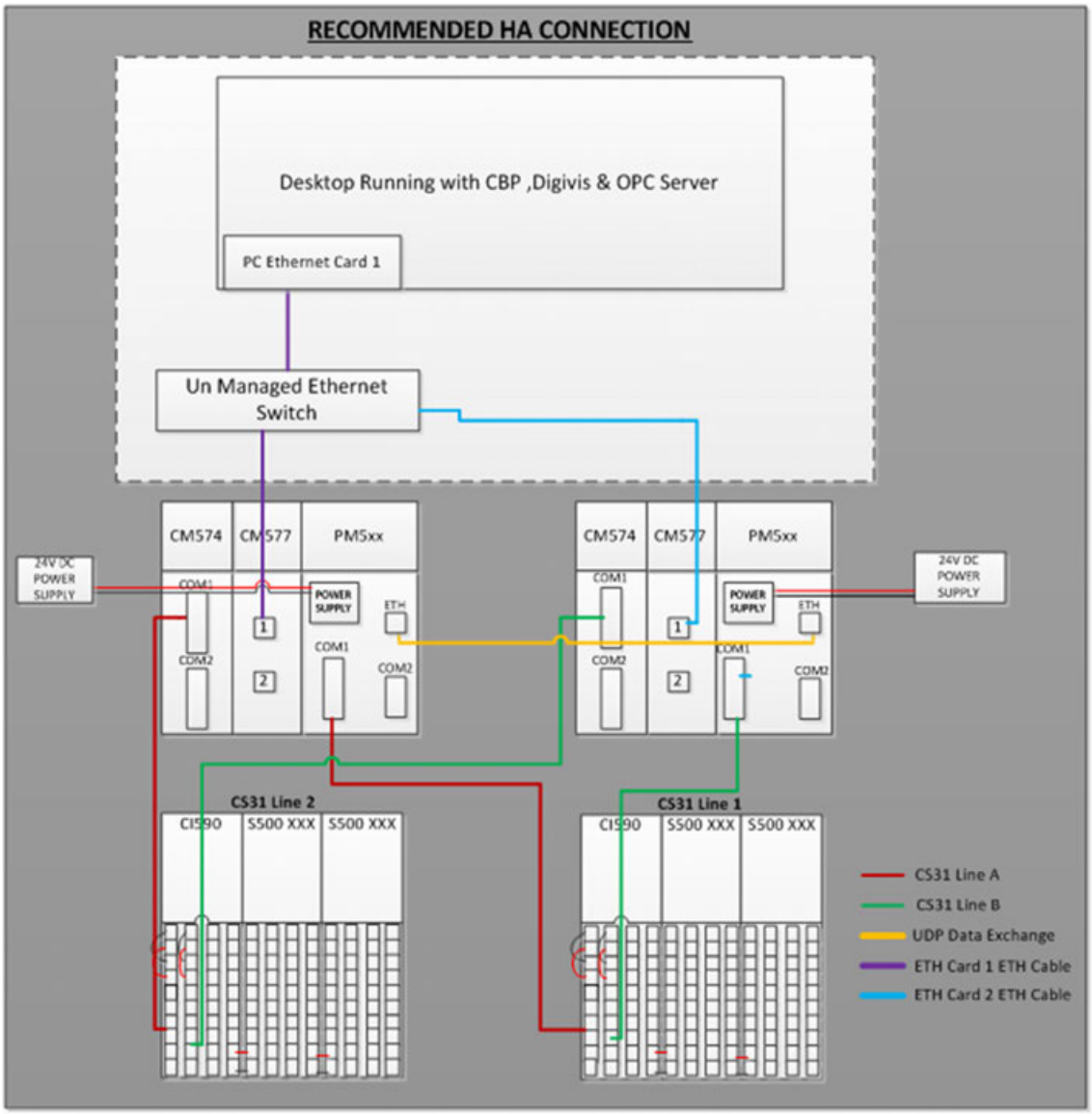


Fig. 35: Recommended high availability connection. Option 3.

If HA application requires CPU redundancy but no Ethernet redundancy, users can use a standard Ethernet switch (unmanged switch) with Digivis 500.

HMI CP6xx with one/two Ethernet ports

HMI CP6xx with One/Two Ethernet ports on HMI with standard Ethernet switch (unmanaged switch):

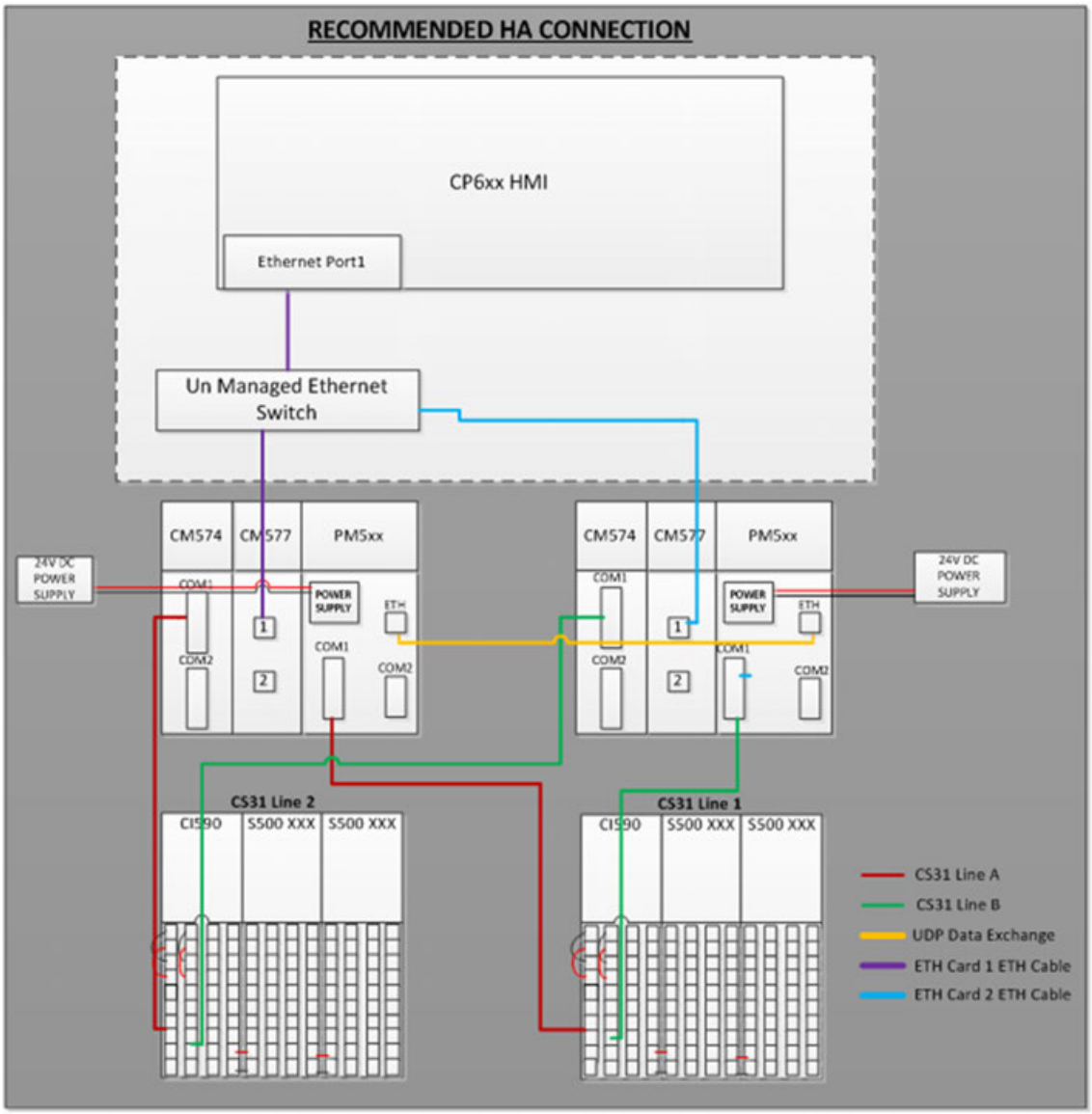


Fig. 36: Recommended high availability connection. Option 4.

If application requires only CPU redundancy and no Ethernet redundancy, the users can use standard Ethernet switch (unmanaged switch) with CP6xx HMI.

HA example application program (Example_AC500_HA_CS31_V242.project) provides a sample application program for CP620 HMI. It has the required scripts to read the data from the primary PLC. Also it provides an option for the user to select one of the two PLCs to read data. This is done using the IP override functionality. With minor modifications the HMI application program can be made application specific.

Data writing from HMI to PLC is not recommended as it will write data only to the primary CPU.

Recommended use of Ethernet connections

The following types of Ethernet communication is required:

- UDP: Communication between the two CPUs for HA synchronization.
- AB: Communication between CPUs and Engineering PC for program download.
- OPC: Communication between CPUs and HMI (optionally).
- WEB: Web server communication between CPUs and HMI (optionally).

Ethernet ports are available on the CPU (onboard) and on the CM597-ETH communication module. For an optimal load balance the following connections are recommended:

CPU Type	Web server required	Onboard ETH1	Onboard ETH2	CM597-ETH
PM57x, PM58x, PM591, PM592	No	UDP	n/a	AB, OPC
PM57x, PM58x, PM591, PM592	Yes	AB, OPC, WEB	n/a	UPD
PM592-2ETH, PM595	No	UDP	AB, OPC	-
PM592-2ETH, PM595	Yes	UDP	AB, OPC, WEB	-

PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.

Software

The HA-CS31 library V2.4.2 (HA_CS31_AC500_V23.lib) has been tested with the following versions:

- Automation Builder: 1.1
- CODESYS: 2.3.9.46
- CPU and CM574 Firmware: 2.4.2
- CI590 Firmware: 3.0.15

Guidelines for usage

Introduction

The following sections give an overview of the engineering steps (hardware configuration, programming, task configuration and program download) and the operation.

If only one CS31 line is required, please consider the information provided in the chapters “Single CS31 Bus on CPU COM Port”. If more than one CS31 line is required (with CM574 extension), please consider the information provided in the chapters “CS31 Bus Extension with CM574-RS communication module”.

For further information on HA-CS31 library preconditions, please refer to Chapter 1.5.5.1.3.2 “Prerequisites for the use of HA-CS31 library” on page 2020.

For further information on HA configuration, please refer to the provided HA examples - typically located in: C:\Users\Public\Documents\Automation-Builder\Examples.

Hardware configuration in Automation Builder

Single CS31 bus on CPU COM port

In order to configure a single CS31 Bus on a CPU COM port, build-up the hardware configuration in Automation Builder. The following figure demonstrates necessary configuration:

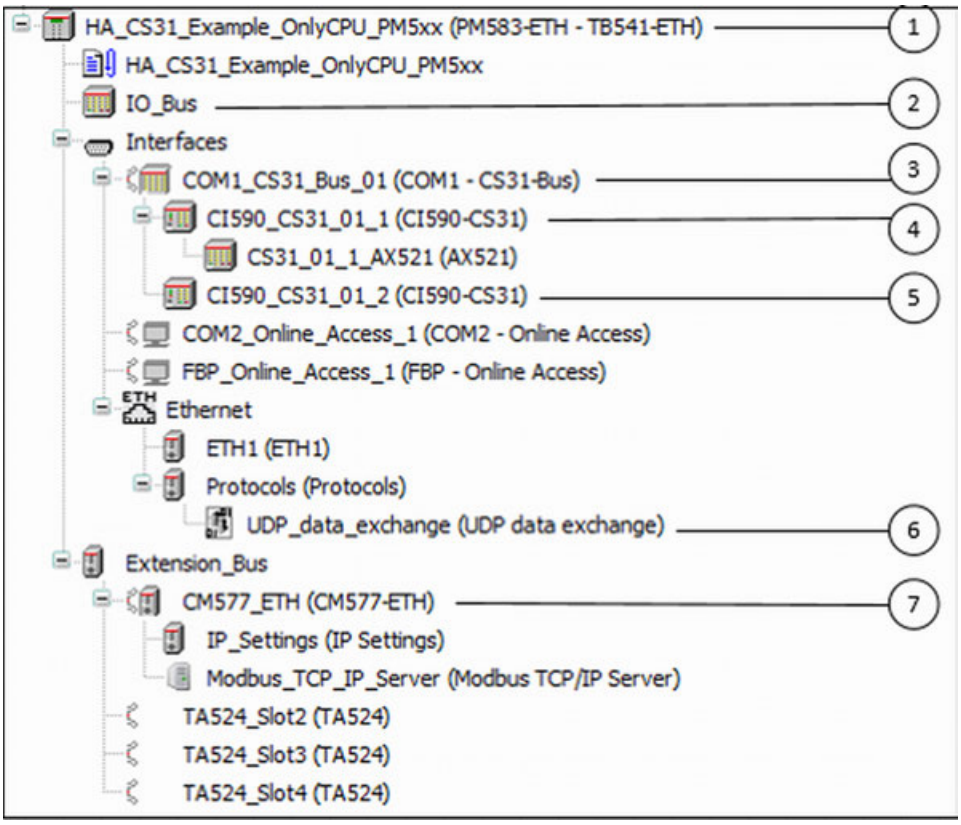


Fig. 37: Hardware Configuration Tree. Option 1.

1. Shows the configured PLC type.
2. Addition of local I/O Bus modules is possible but the I/O modules will not be redundant.
3. CS31 Bus module below CPU COM1 interface.
4. CI590-CS31-HA module and I/O modules below CS31 Bus.
5. Maximum limit is 31 CI590-CS31-HA modules in one CS31 line.
6. UDP data exchange below the Ethernet interface. This is mandatory for a HA-CS31 system to run since it is responsible for synchronization of CPU A and CPU B.

7. CM597-ETH module below the communication module slot for OPC server and Automation Builder communication etc. If the PLC has more than one onboard ETH port (Example: PM595, PM591-2ETH) the user can use ETH2 port instead of CM597-ETH. Refer to *Chapter 1.5.5.1.2.2.1.2 “Recommended use of Ethernet connections” on page 1991*.

PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.

We recommend you, to use separate Ethernet slots for UDP data exchange and PC connectivity. Refer to Chapter 1.5.5.1.2.2.1.2 “Recommended use of Ethernet connections” on page 1991.

For using CI590 2FC (fast counter) users have to add CI590 2FC under CS31 network. Then, the “Fast Counter” selection must be updated with the appropriate counter type. With the default setting “No Counter” the system will become unstable.

CS31 bus extension with CM574-RS module

If more than one CS31 line is needed, CM574 can be used for extension. The following figure demonstrates an example of CPU and CM574 hardware configuration tree:

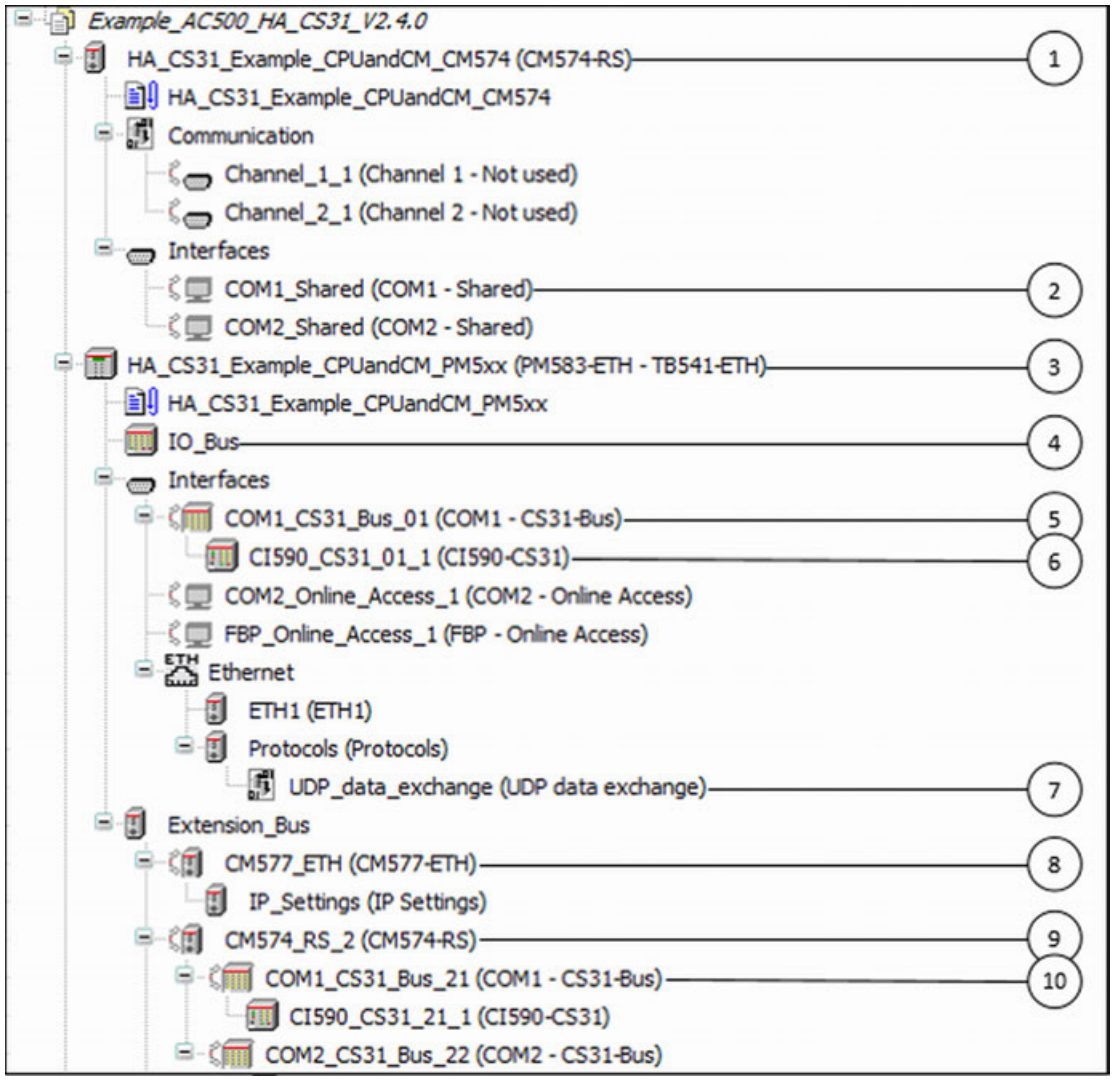


Fig. 38: Hardware Configuration Tree. Option 2.

1. Shows the configured device type: CM574-RS.
2. Configure Interfaces as “shared” com ports. Engineering of underlying CS31 lines and CI590-CS31-HA is done in the CPU hardware tree.
3. Shows the configured PLC type.
4. Addition of local I/O Bus modules is possible but the I/O modules will not be redundant.
5. CS31 Bus module below CPU COM1 interface.
6. CI590-CS31-HA module and I/O modules below CS31 Bus.
7. UDP data exchange below the Ethernet interface. This is mandatory for a HA-CS31 system to RUN, it does the synchronization between CPU A and CPU B.
8. CM597-ETH module below the communication module slot for program download, OPC server communication etc. We recommend you, to have separate Ethernet slot for UDP data exchange and PC connectivity. In another case if a user has a PLC with more than one onboard ETH port (Example: PM595, PM591-2ETH) the user can use ETH2 port for downloading e.g. the program, web server, OPC server communication etc. Refer to ↬ *Chapter 1.5.5.1.2.2.1.2 “Recommended use of Ethernet connections” on page 1991*.
9. CM574-RS for CS31 extension.
10. CS31 Bus module below CM574-RS COM1 interface and CI590-CS31-HA module and additional I/O modules.

PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.

The following hardware configuration (CS31 lines below CM574-RS) is not supported for a HA-CS31 system even though it is supported by Automation Builder:

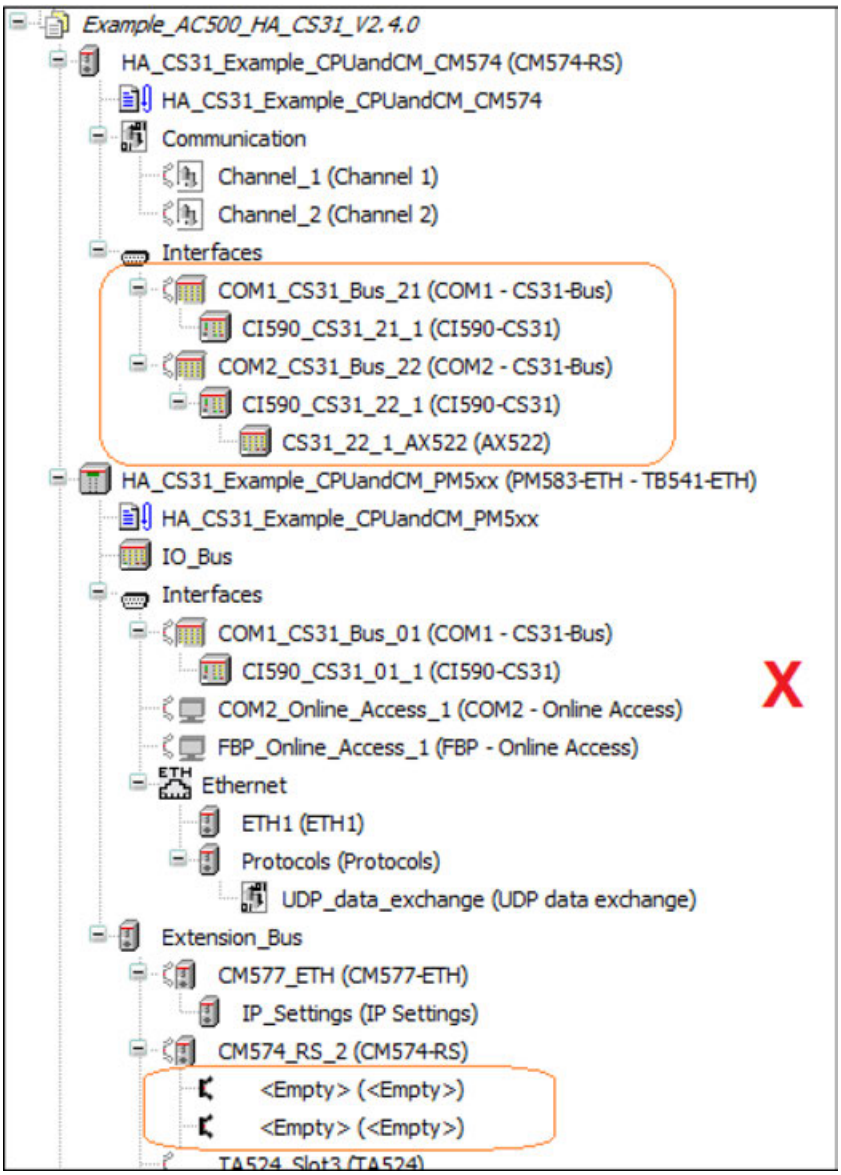


Fig. 39: Not supported/ not recommended hardware configuration.

For more details about the configuration please refer to the Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

Programming

Single CS31 bus on CPU COM port

The CPU program contains different types of HA-CS31 function blocks.

The following figure describes function blocks and functions for a HA-CS31 system to work with a CPU COM1 CS31 Bus connected with I/O modules. The function blocks indicated with dotted lines are not mandatory for a HA-CS31 system to run.

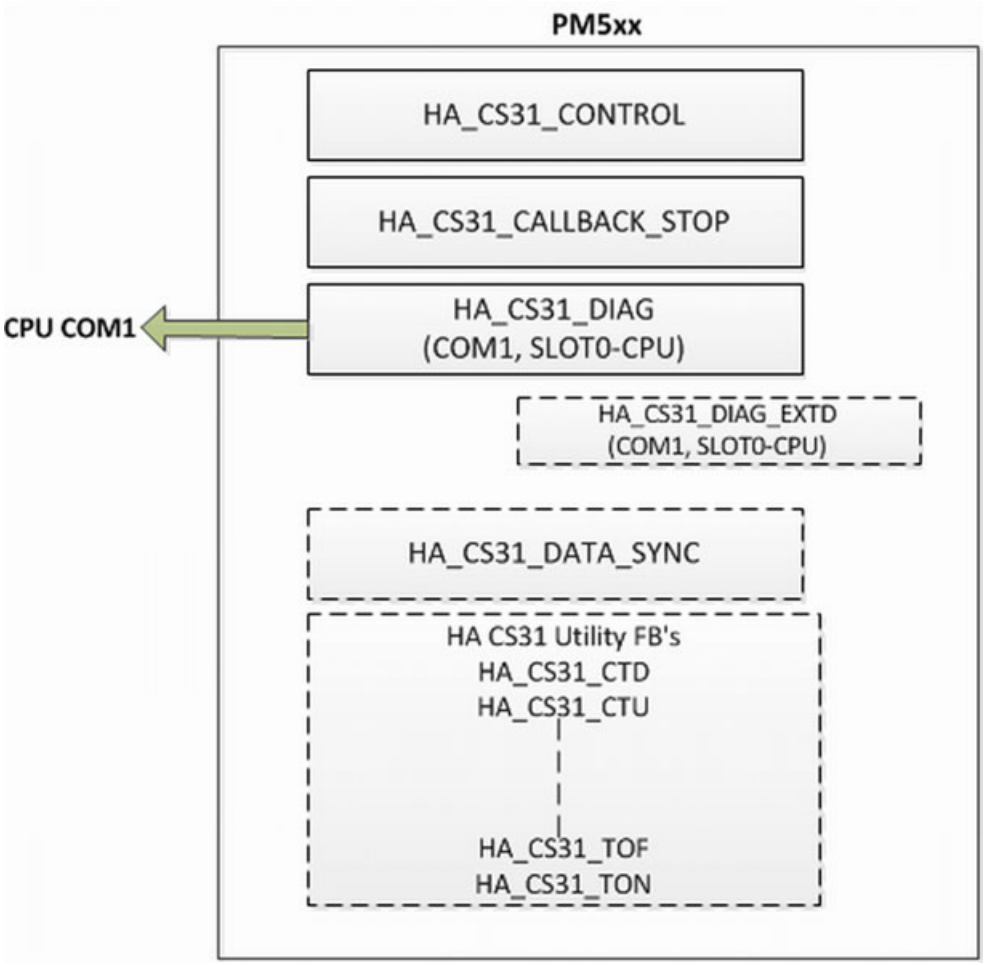


Fig. 40: Function blocks in CPU

The following function blocks and functions are required to be called and downloaded to the PLC:

- HA_CS31_CONTROL
- HA_CS31_DIAG
- HA_CS31_CALLBACK_STOP

CS31 bus extension with CM574-RS module

If configuration requires more than one CS31 network, then the CM574-RS is used to extend the CS31 lines. In this case, users have to create two separate programs: one for the AC500 CPU and one for the CM574-RS including the respective function blocks called from the library.

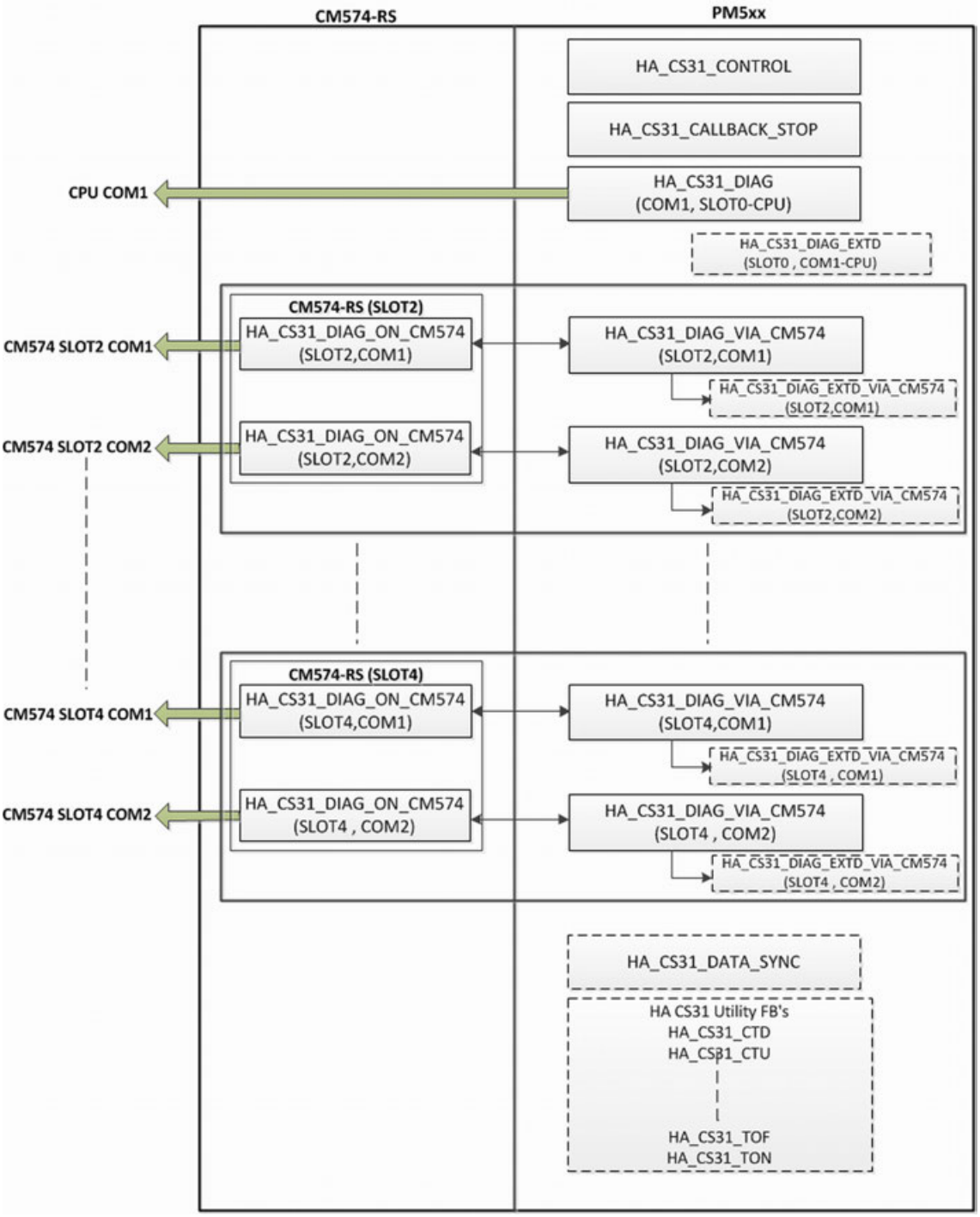


Fig. 41: Function blocks in CPU and CM574-RS

The following sections explain mandatory steps required while programming such a set-up.

CPU program

The right part of the figure above describes function blocks for the CPU. The function blocks indicated with dotted lines are not mandatory for a HA-CS31 system to run.

The following function blocks and functions are required to be downloaded to the CPU:

- HA_CS31_CONTROL
- HA_CS31_DIAG
- HA_CS31_DIAG_VIA_CM574
- HA_CS31_CALLBACK_STOP

> *For HA programming using SFC language (Sequential Function Chart) is not recommended as too much and too spread data is to be synchronized. For further information about programming, please refer to Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.*

- Application will start on CPU A connected to CS31 Bus A if both CPUs start together or on CPU CS31 B and Bus B if only CPU B is connected.
- If CPU A stops / turns off or crashes -> application will run on CPU B and Bus B.
- If CS31 Bus A is open or in short circuit -> application will run on CPU B and Bus B.

CM574-RS program

The left part of the figure above describes the function blocks for CM574-RS.

Function block HA_CS31_DIAG_ON_CM574 must be downloaded to CM574 module. This function block sends CS31 Bus diagnosis information to host CPU.

The CM574-RS program is common and can be used for each CM574-RS module without any specific changes (please refer to Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf).

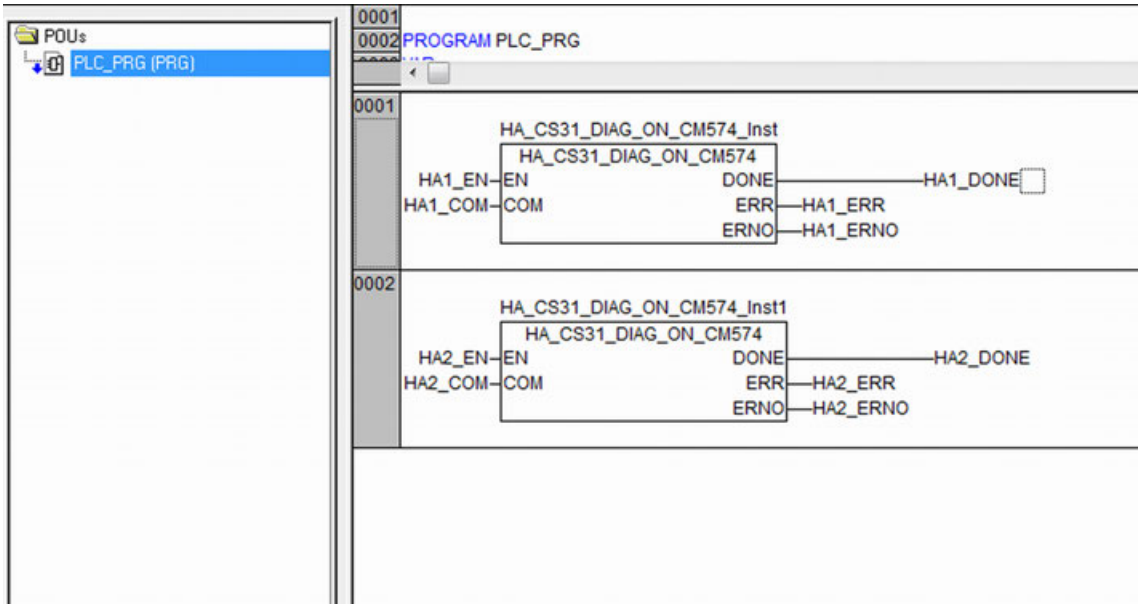


Fig. 42: Common program for CM574-RS

Task configuration

Single CS31 bus on CPU COM port

Task configuration for the CPU program

At least, two tasks must be defined:

- HA_Task for the mandatory HA-CS31 function blocks HA_CS31_CONTROL, HA_CS31_DIAG, HA_CS31_DIAG_VIA_CM574. Recommended HA task cycle time is equal to CS31 cycle time (as indicated in Automation Builder), but at least 30 ms. Priority must be higher than priority of the other tasks in the project.
- Main task for the application itself: lower priority than the HA_Task. Cycle time to be decided based on CPU loading, but not less than HA cycle time.

HA_Task priority must be higher than the main task priority.

CS31 bus extension with CM574-RS module

Task configuration for the CPU program

At least, two tasks must be defined:

- HA_Task for the mandatory HA-CS31 function blocks HA_CS31_CONTROL, HA_CS31_DIAG, HA_CS31_DIAG_VIA_CM574. Recommended HA task cycle time is equal to CS31 cycle time (as indicated in Automation Builder), but at least 30 ms. Priority must be higher than priority of the other tasks in the project.
- Main task for the application itself: lower priority than the HA_Task. Cycle time to be decided based on CPU loading, but not less than HA cycle time.

HA_Task priority must be higher than the main task priority.

Task configuration for the CM574 program

The program which calls function block HA_C31_DIAG_ON_CM574 must be called and assigned to a cyclic task.

The cycle time of this task must not be more than the cycle time of the HA_Task above.

For further information, please refer to Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

Program download

To avoid any difference in configuration, we recommend you to create a single Automation Builder project for high availability. Download this project to both CPUs.

Single CS31 bus on CPU COM port

Download for CPU program

The CPU program can be downloaded to the PLC using normal TCP/IP gateway. We recommend you, to use a dedicated CM597-ETH slot for downloading the program to both PLCs. If no CM597-ETH is available, onboard Ethernet slot can be used. In another case if user has a PLC with more than one onboard ETH port (Example: PM595, PM591-2ETH) the user can use ETH2 port for downloading e.g. the program, webserver, OPC server communication etc. After the download, create a boot project.

For further information, please refer to Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

CS31 bus extension with CM574-RS module

Download for CPU program

The CPU program can be downloaded to the PLC using normal TCP/IP gateway. We recommend you, to use a dedicated CM597-ETH slot for downloading the program to both PLCs. If no CM597-ETH is available, onboard Ethernet slot can be used. In another case if user has a PLC with more than one onboard ETH port (Example: PM595, PM591-2ETH) the user can use ETH2 port for downloading e.g. the program, webserver, OPC server communication etc. After the download, create a boot project.

For further information, please refer to Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

Download for CM574 program

Download the configuration and program to CM574 via Ethernet level 2 root driver. The following figure describes the settings of a gateway channel for connection via AC500 CPU. The Ethernet port uses the IP address 192.168.3.10 with routing to CM574-RS plugged into slot 2 (line 2).

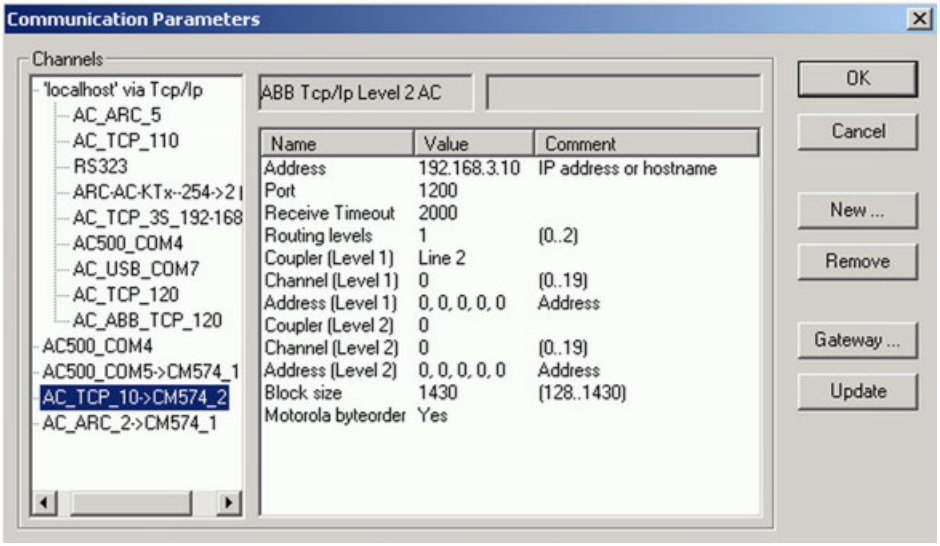


Fig. 43: Communication Parameters

Table 98: The following settings have to be specified:

Address	IP address of the AC500 CPU, in the example: 192.168.3.10
Routing levels	1
Communication module (level 1)	Line 2
Motorola byte order	Yes

After downloading the program, create a boot project. For further information on providing access to CM574, please refer to *Chapter 1.6.4.2.6.5 “Programming access to the CM574-RS” on page 5610*

Operation

Single CS31 bus on CPU COM port

HA_CS31_CONTROL function block handles HA-CS31 operation such as change over from primary to secondary CPU in case of fault with related diagnostics and also data transfer between HA-CS31 CPUs. This is a mandatory function block for High Availability to run. For further information on this function block, please refer to *Chapter 1.5.5.1.3.3.2 “HA_CS31_CONTROL - HA control FB” on page 2026*.

HA_CS31_DIAG function block will be used to get diagnosis information only from CPU COM1 CS31 line. For further information on this function block, please refer to *Chapter 1.5.5.1.3.3.4 “HA_CS31_DIAG - Reading HA diagnosis” on page 2035*.

HA_CS31_DATA_SYNC function block is used for synchronizing non High Availability function blocks or any other data. For further information on this function block, please refer to ↳ *Chapter 1.5.5.1.3.3.3 “HA_CS31_DATA_SYNC - HA data synchronization FB” on page 2028*.

High Availability Utility function blocks outputs are internally synchronized and can be used directly without using HA_CS31_SYNC function block. For further information on these function blocks, please refer to ↳ *Chapter 1.5.5.1.3.4 “Utility function blocks” on page 2047*.

Visualization

High Availability overview visualization can be used for monitoring purpose but it will show data only for the CPU on which we are logged into.

The following figure describes the visualization screen of a running HA-CS31 system with the CS31 lines only on the PLC COM port:

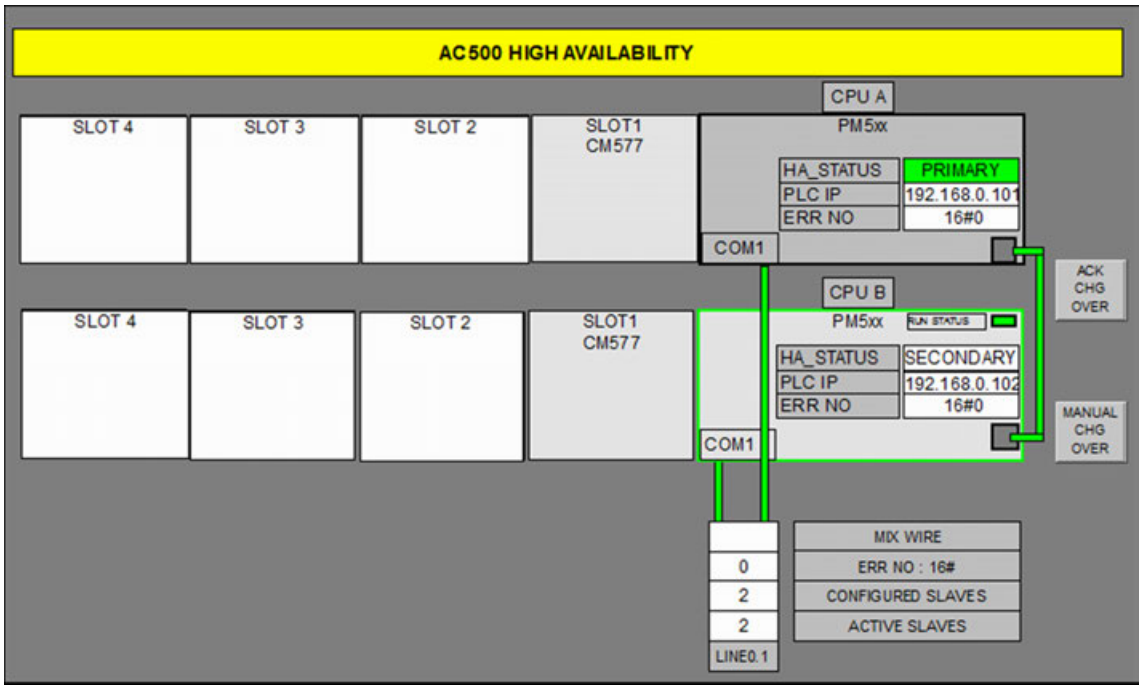


Fig. 44: Visualization of AC500 High Availability System. Option 1.

DigiVis and OPC server can be used to write data as it will allow the user to write data to both PLCs at the same time. Data is read out from the primary PLC only. We recommend you, not to use HMI for writing data as it is not written to both PLCs.

> *If during the startup of the HA-CS31 system none CI590-CS31-HA module is powered on, or if all CI590-CS31-HA modules have a wrong module address, the system does not become stable.*
>
> *If any CI590-CS31-HA module address has been changed after startup of the HA-CS31 system the system will continue to run as it is. In order to activate the new address the complete system must be restarted.*
>
> *If any CI590-CS31-HA module is powered off and on, there is no need to power restart the complete system from library version V2.4.2 onwards. Sync LED of failed CI590 line will blink and has to be acknowledged by the user.*

For further information, please refer to both, the online help for HA-CS31 function blocks (↳ *Chapter 1.5.5.1.3 “AC500 High Availability CS31 library” on page 2017*) and the Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

CS31 bus extension with CM574-RS communication module

Apart from all functions explained in chapter *Chapter 1.5.5.1.2.3.6.1 "Single CS31 bus on CPU COM port" on page 2001* there are CM574 related function blocks which are called in the programs of this configuration.

CPU program Function block HA_CS31_DIAG_VIA_CM574 reads the extended diagnostics of CM574-RS CS31 Bus in high availability operation.

CM574 program Function block HA_CS31_DIAG_ON_CM574 sends CS31 Bus diagnosis information to host CPU and receives CS31 configuration information from host CPU.

Visualization The following figure describes visualization of a running HA-CS31 system with the CS31 lines both on the PLC COM port and the CM574 COM ports.

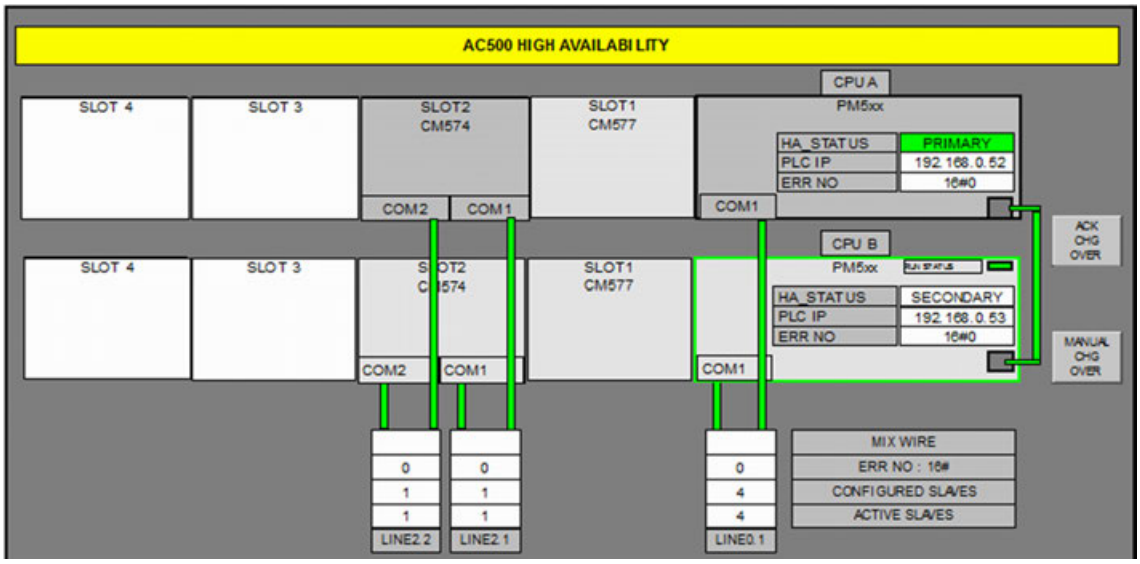


Fig. 45: Visualization of AC500 High Availability System. Option 2.

Example of a utility function block - Switch over

Consider the on-delay timer HA_CS31_TON (refer to *Chapter 1.5.5.1.3.4.12 "HA_CS31_TON - HA turn-on delay timer" on page 2071*).

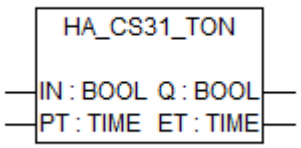


Fig. 46: HA_CS31_TON

Both CPUs require the same function block called in the program. Under normal operating conditions the elapsed time ET and output, Q of the timer is synchronized internally for both CPUs. ET and Q data are available and can be attached to local or global variables in the program as per application requirements. If CPU A shuts down due to a fault, the primary status switches over to CPU B.

In the event of a switch over, the moment the CPU B becomes the primary, the timer on this PLC will keep running. Since till the time of CPU A failure the timer on the CPU B was synchronized, the actual process remains unaffected by the switch over.

All possible use cases for switch over are described under *Chapter 1.5.5.1.2.4.1 “Use case and reaction time” on page 2004*.

Fault on primary and secondary PLC

HA-CS31 system only takes care of the first fault. In case of a second fault the primary CPU remains primary CPU until the second fault occurs. This results in no further switch overs (manual switch overs included).

Example: If an error occurs on PLC A (primary) due to CS31 cable disconnection, the HA-CS31 system will switch over to PLC B (primary). If there is any cable disconnection happened in either PLC A (secondary) or PLC B (primary) the HA-CS31 system will remain in PLC B as primary.

In case of a second error generated in the HA-CS31 system, function block HA_CS31_CONTROL will generate an error with error code 16#2006.

If one fault occurs in both, CPU A and CPU B, error code 16#2006 occurs. This error code does not consider two faults on the same PLC. In the following the most common scenarios that trigger error code 16#2006 (named as wHA_ER_LOCAL_AND_REMOTE_FAULT) are listed:

- CS31 cable is removed from PLC A and PLC B (from same corresponding ports).
- CS31 cable is removed from PLC A and PLC B (from different ports).
- Cable of CS31 network is removed from any CI590-CS31-HA related to PLC A and afterwards it is removed from PLC B (coming from the same corresponding ports).
- Cable of CS31 network is removed from any CI590-CS31-HA related to PLC A and afterwards it is removed from PLC B (coming from different ports).
- PLC A is stopped and PLC B CS31 line is removed.
- PLC A CS31 line is removed and PLC B is stopped.
- PLC A is powered off and PLC B CS31 line is removed.
- PLC A CS31 line is removed and PLC B is powered off.
- PLC A CS31 line is removed and PLC B CI-590 is powered off.

All outputs on the remote modules keep on working properly as long as at least one CS31 line is active - regardless of CPU primary or secondary status. This is because of the self-primary action in CI590-CS31-HA module.

Functionality

Use case and reaction time

The HA-CS31 system performs a switch over when the primary CPU is powered off, crashed or stopped or if the primary CS31 bus is disconnected. The total switch over time depends on the use case. The following table gives an overview of the switch over times assuming that HA cycle = CS31 cycle time (as indicated in Automation Builder) or at least 30 ms according to the recommendation. After CI590 switchover period the outputs will be written from the new CPU. CI590 LED will be steadily green after CPU switch over period.

	Capacitor discharge period	**CI590 switch over period**	**CPU switch over period, afterwards remaining CI590 are switched over**
Primary CPU powered off	Depending on system size	≤ 4 CS31 cycles	Not relevant, all CI590 already switched over
Primary CPU stopped	0	≤ 2 CS31 cycles	Not relevant, all CI590 already switched over

	Capacitor discharge period	CI590 switch over period	CPU switch over period, afterwards remaining CI590 are switched over
Primary CS31 bus cut near CPU	0	≤ 4 CS31 cycles	35 HA cycles
Primary CS31 bus cut at local CI590	0	≤ 4 CS31 cycles	35 HA cycles

Details are described in the following sections:

Primary CPU off or primary CPU crash

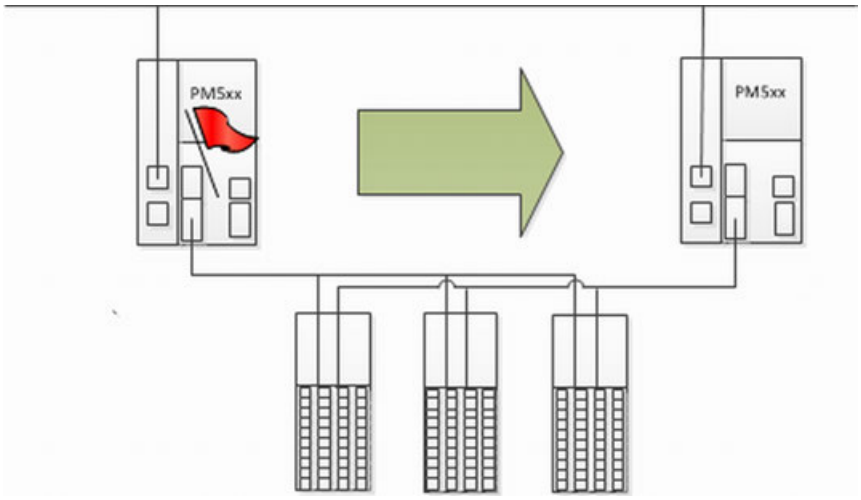


Fig. 47: Use case 1

Reaction	Switch over to secondary CPU. All CI590 are immediately switched over.
Total switch over time	CI590 switch over period (≤ 4 CS31 cycles).
Comment	In case of power off there is a capacitor discharge period before CI590 switch over period. Duration depends on the system size. Outputs are frozen during CI590 switch over period. It takes some more HA cycles until the other CPU takes over primary status. However, this is not relevant for the process, because all CI590 were already switched over individually in the first CI590 switch over period.
Diagnosis message on function block	• Primary CPU: Power OFF • Secondary CPU: HA_CS31_CONTROL: 16#101B (remote CPU failure: other CPU is OFF or out of order.)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” OFF, “CB” ON, "RUN A" OFF, "RB" ON, "SYNC-ERR" RED (Blinking), "S-ERR" RED (Blinking).

Primary CPU stop

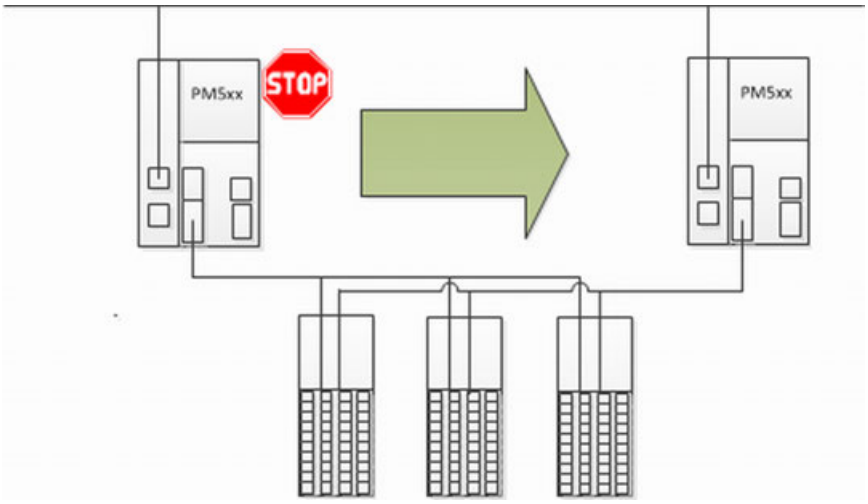


Fig. 48: Use case 2

Reaction	Switch over to secondary CPU. All CI590 are immediately switched over.
Total switch over time	CI590 switch over period (≤ 2 CS31 cycles).
Comment	Outputs are frozen during CI590 switch over period. It takes some more High Availability cycles until the other CPU takes over primary status. However, this is not relevant for the process, because all CI590 were already switched over individually in the first CI590 switch over period.
Diagnosis message on function block	• Primary CPU: STOP • Secondary CPU: HA_CS31_CONTROL: 16#101B (remote CPU failure: other CPU is OFF or out of order.)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON (Blinking), “CB” ON, "RUN A" OFF, "RB" ON, "SYNC-ERR" RED (Blinking), "S-ERR" OFF.

Secondary CPU off or secondary CPU crash

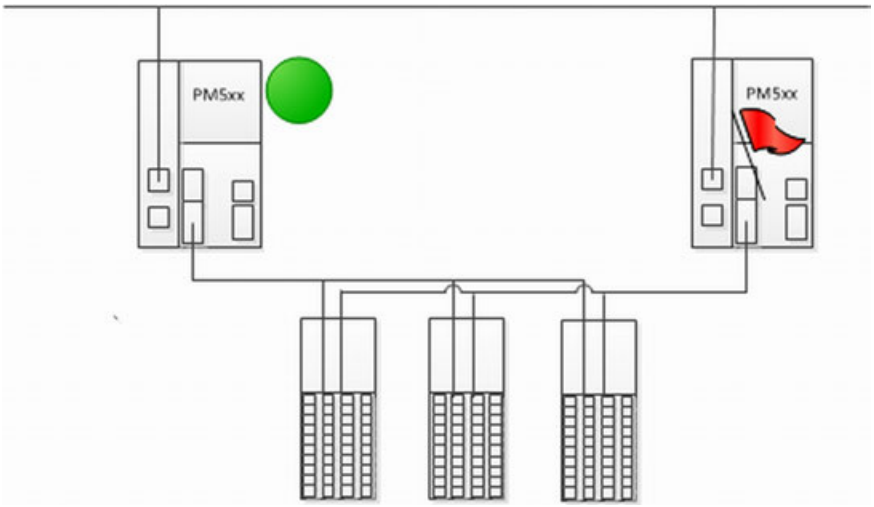


Fig. 49: Use case 3

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues
Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#101B (remote CPU failure: other CPU is OFF or out of order.) • Secondary CPU: Power OFF
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” OFF, "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" ON (Blinking).

Secondary CPU stop

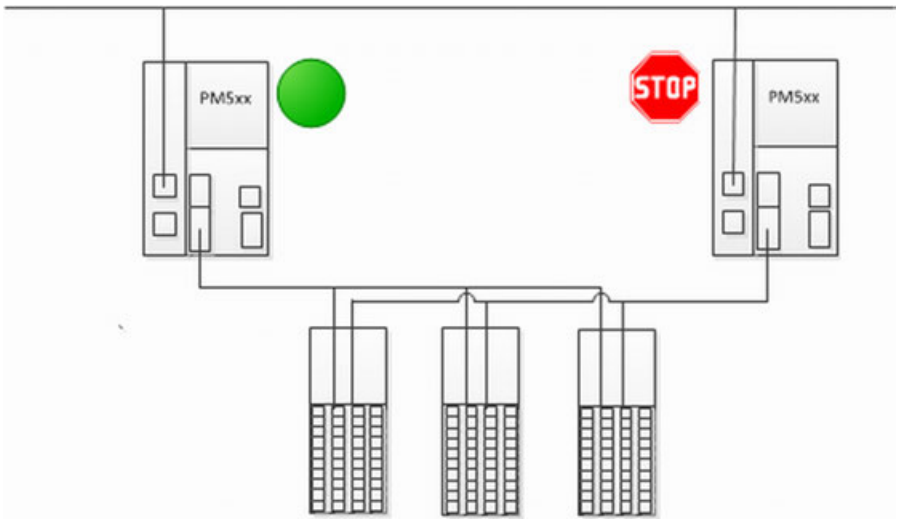


Fig. 50: Use case 4

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues
Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#101B (remote CPU failure: other CPU is OFF or out of order.) • Secondary CPU: STOP
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” ON (Blinking), "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" OFF.

Primary CS31 bus off / Disconnected / Short circuit near CPU / No module connected

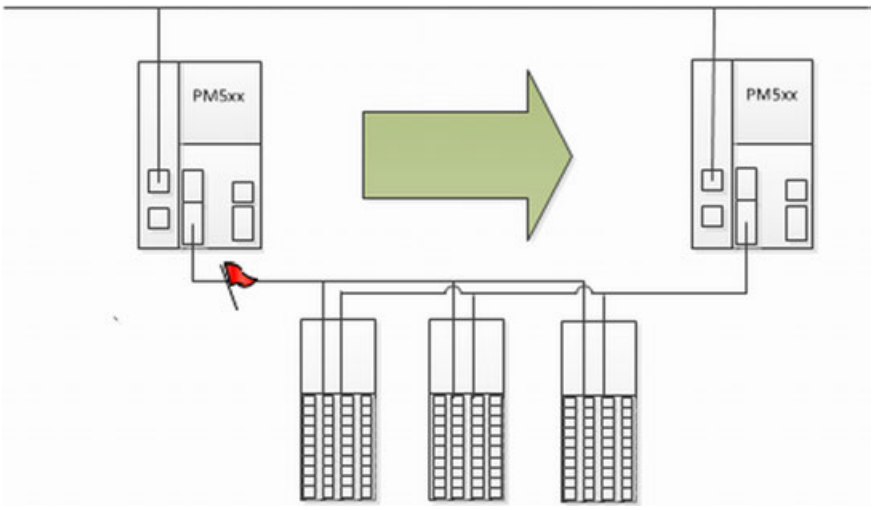


Fig. 51: Use case 5

Reaction	Switch over to secondary CPU. All CI590 from the broken CS31 bus are immediately switched over. After the CPU switch over period all CI590 from the other CS31 lines are switched over.
Total switch over time	CI590 switch over period (≤ 4 CS31 cycles) + CPU switch over period (~ 35 HA cycles)
Comment	Outputs are frozen during CI590 switch over period. It takes ~ 35 HA cycles until the other CPU takes over primary status. Thus short disturbances of the CS31 bus cannot cause an unintended switch over.
Diagnosis message on function block	● Primary CPU: HA_CS31_CONTROL: 16#2005 (CS31 Bus failure), HA_CS31_DIAG / HA_CS31_DIAG_VIA_CM574: 16#1021 (one or more CI590-CS31-HA slave(s) is inactive) ● Secondary CPU: STOP
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” ON (Blinking), "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" OFF.

Primary CS31 bus off / Disconnected / Short circuit at local CI590/ Some Modules still connected

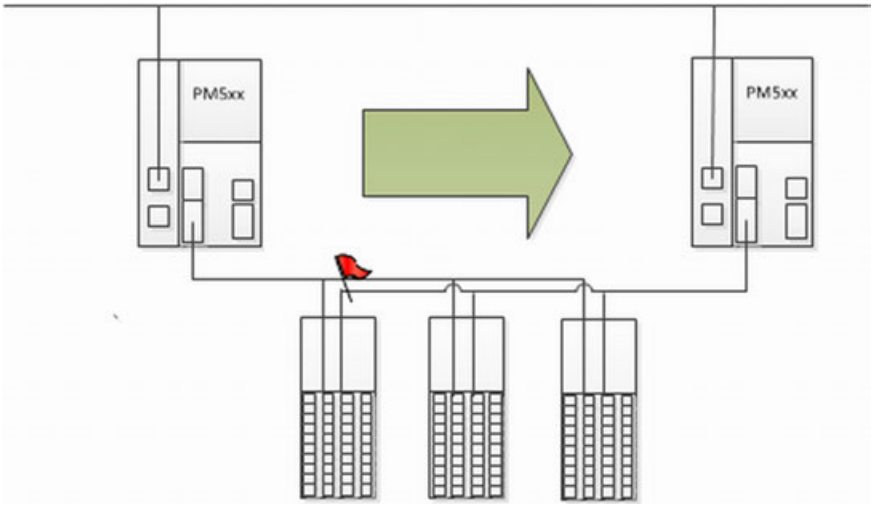


Fig. 52: Use case 6

Reaction	Switch over to secondary CPU. All disconnected CI590 are immediately switched over. After the CPU switch over period all remaining CI590 of the same CS31 line and all CI590 from the other CS31 lines are switched over.
Total switch over time	CI590 switch over period (≤ 4 CS31 cycles) + CPU switch over period (~ 35 HA cycles)
Comment	Outputs are frozen during CI590 switch over period. It takes ~ 35 HA cycles until the other CPU takes over primary status. Thus short disturbances of the CS31 bus cannot cause an unintended switch over.
Diagnosis message on function block	● Primary CPU: HA_CS31_CONTROL: 16#1021 (own CI590-CS31-HA slave failure (missing module), HA_CS31_DIAG / HA_CS31_DIAG_VIA_CM574: 16#1021 (one or more CI590-CS31-HA slave(s) is inactive) ● Secondary CPU: HA_CS31_CONTROL: 16#201B (remote CS31 Bus failure: other CPUs' CS31 Bus is out of order)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” OFF, “CB” ON, "RUN A" OFF, "RB" ON, "SYNC-ERR" ON, "S-ERR" ON (Blinking).

Secondary CPU CS31 bus disconnected / Short Circuit / At least one Module is disconnected

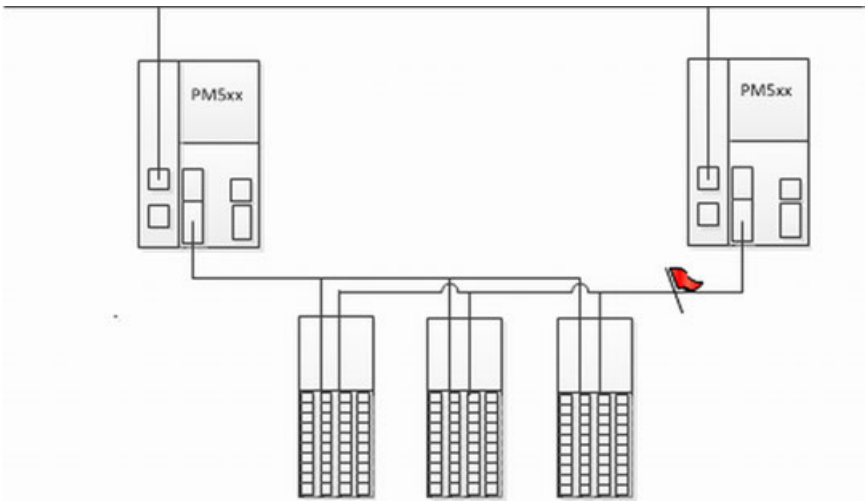


Fig. 53: Use case 7

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues
Diagnosis message on function block	● Primary CPU: HA_CS31_CONTROL: 16#201B (remote CS31 Bus failure: other CPUs' CS31 Bus is out of order) ● Secondary CPU: HA_CS31_CONTROL: 16#1021 (own CI590-CS31-HA slave failure (missing module), HA_CS31_DIAG / HA_CS31_DIAG_VIA_CM574: 16#1021 (one or more CI590-CS31-HA slave(s) is inactive.)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” OFF, "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" ON (Blinking).

Ethernet cable disconnected or no Ethernet communication

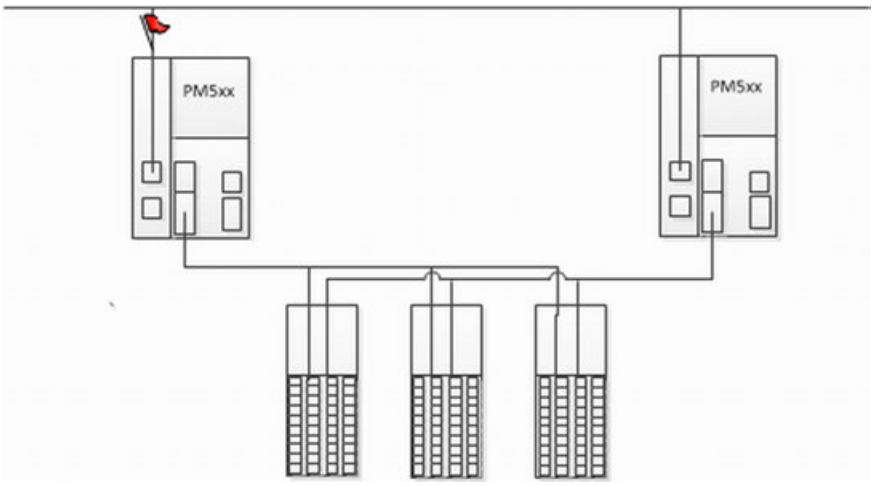


Fig. 54: Use case 8

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues
Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#2013 (no Ethernet Link) • Secondary CPU: HA_CS31_CONTROL: 16#2013 (No Ethernet Link)
Diagnosis message on CI590-CS31 LEDs	CI590-CS31 LEDs status: “CS31 A” ON, “CB” ON, "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" OFF (considering PM1 is Primary).

One or more CI590-CS31-HA are off or not connected to both CPUs

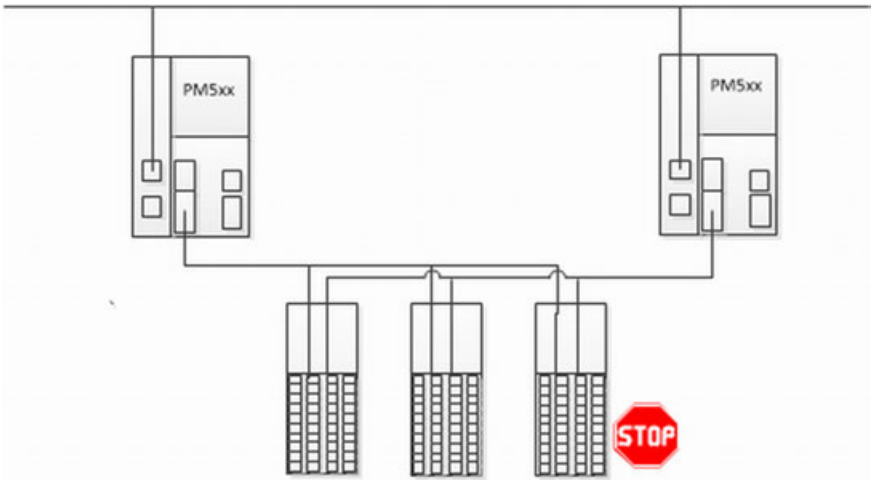


Fig. 55: Use case 9

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues on existing module according user's choice

Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#2006 (fault on both local and remote CPU), HA_CS31_DIAG / HA_CS31_DIAG_VIA_CM574: 16#1021(one or more CI590-CS31-HA slave(s) is inactive),HA_CS31_DIAG_VIA_CM574: 16#1021(one or more CI590 are inactive) • Secondary CPU: HA_CS31_CONTROL: 16#2006 (fault on both local and remote CPU), HA_CS31_DIAG / HA_CS31_DIAG_VIA_CM574: 16#1021(one or more CI590-CS31-HA slave(s) is inactive), HA_CS31_DIAG_VIA_CM574: 16#1021 (one or more CI590 are inactive)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” ON, "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" OFF.

CM574-RS is not in RUN mode in primary

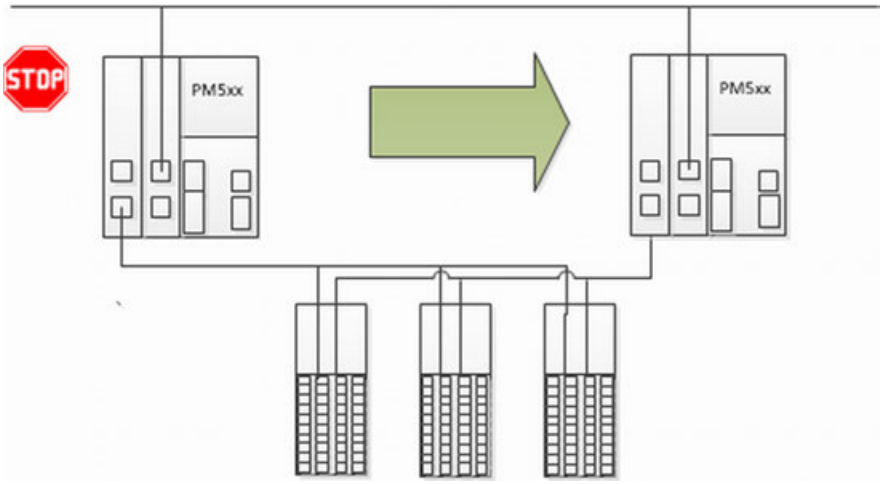


Fig. 56: Use case 10

Reaction	Switch over to secondary PLC
Total switch over time	5 ms
Comment	Outputs freeze to last state during switch over time. Then process continues on secondary CPU.
Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#2005 (CS31 Bus failure), HA_CS31_DIAG_VIA_CM574: 16#2013 (error in DPRAM communication between CM574-RS and AC500 CPU) • Secondary CPU: HA_CS31_CONTROL: 16#201B (remote CS31 Bus failure: other CPUs' CS31 Bus is out of order)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON (Blinking), “CB” ON, "RUN A" OFF, "RB" ON, "SYNC-ERR" ON (Blinking), "S-ERR" OFF.

CM574-RS not in RUN mode in secondary

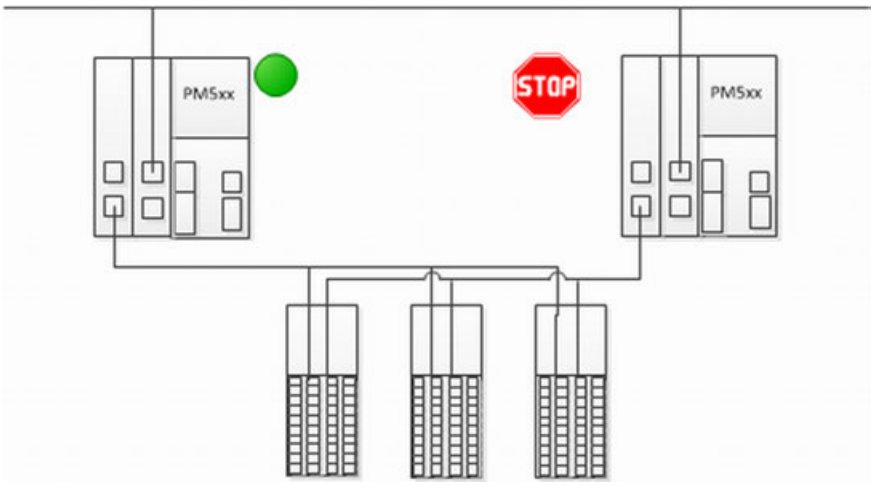


Fig. 57: Use case 11

Reaction	No switch over
Total switch over time	N.A.
Comment	Process continues
Diagnosis message on function block	• Primary CPU: HA_CS31_CONTROL: 16#201B (remote CS31 Bus failure: other CPUs' CS31 Bus is out of order) • Secondary CPU: HA_CS31_CONTROL: 16#2005 (CS31 Bus failure), HA_CS31_DIAG_VIA_CM574: 16#2013 (error in DPRAM communication between CM574-RS and AC500 CPU)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” ON (Blinking), "RUN A" ON, "RB" OFF, "SYNC-ERR" OFF, "S-ERR" OFF.

CM574-RS not in RUN mode in both systems

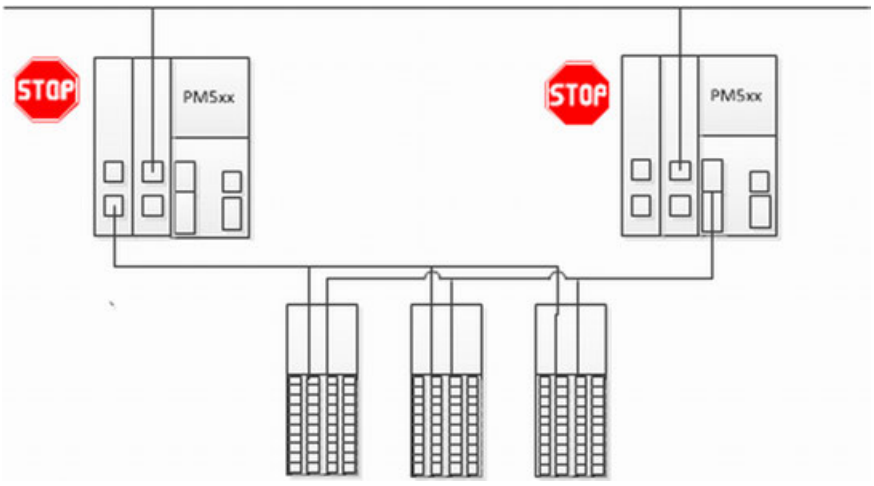


Fig. 58: Use case 12

Reaction	No switch over as fault is in both system
Total switch over time	N.A.
Comment	-

Diagnosis message on function block	● Primary CPU: HA_CS31_CONTROL: 16#2006 (fault on both local and remote CPU), HA_CS31_DIAG_VIA_CM574: 16#2013 (error in DPRAM communication between CM574-RS and AC500 CPU) ● Secondary CPU: HA_CS31_CONTROL: 16#2006 (fault on both local and remote CPU), HA_CS31_DIAG_VIA_CM574: 16#2013 (error in DPRAM communication between CM574-RS and AC500 CPU)
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON (Blinking), “CB” ON (Blinking), "RUN A" ON (Blinking), "RB" OFF, "SYNC-ERR" OFF, "S-ERR" ON (Blinking).

Manual switch over by users

Reaction	Switch over to secondary CPU at user's request
Total switch over time	N.A.
Comment	Process continues, primary and secondary CPUs are swapped
Diagnosis message on function block	● Primary CPU: No Error ● Secondary CPU: No Error
Diagnosis message on CI590-CS31 LEDs	“CS31 A” ON, “CB” ON, "RUN A" OFF, "RB" ON, "SYNC-ERR" ON, "S-ERR" OFF (Considering Manual switch over request to PM1).

Procedure for modifying hardware and application program

In order to modify hardware configuration or application program while HA-CS31 system is running, proceed as follows:

1. Execute all necessary changes.
2. Compile the project.
3. Make sure that there is no error in the program.
4. We recommend you, to carry out a 'Clean All' and 'Rebuild All'.
5. Select the communication channel of the primary CPU.
6. Download the modified program.
   ⇨ Now the CPU goes to STOP mode and there will be a changeover to secondary CPU.
7. After a successful download, set the primary CPU to RUN mode and create a boot project.
   ⇨ Now, the primary CPU is updated with the new program.
8. Select the communication channel of the second CPU which is now the primary. Download the modified program.
   ⇨ The CPU switches to STOP mode which results in a change over to the original CPU. The original CPU will start working as Primary.
9. After a successful download, set the CPU to RUN mode and create a boot project.
   ⇨ The complete system works in the same status as before modification.

After adding or deleting the HA_CS31_DATA_SYNC function block or any HA-CS31 utility function block, restart the complete system.

It is absolutely necessary that both CPUs have identically user programs.

The replacement of CI590 is possible with a normal HA-CS31 system, which otherwise has no error: PLC A has to be the primary.

For replacement of CI590 when PLC B is the primary, the following pins of TU522-CS31 must be bridged before:

- *2.2 to 2.5*
- *2.3 to 2.6*
- *2.4 to 2.7*

System structure

A HA-CS31 system is characterized by two AC500 CPUs with the following features:

- Synchronized by means of UDP Ethernet Bus system.
- Identical application program that is loaded in both CPUs.
- Two CS31 master systems with redundant CI590-CS31-HA slave(s) and additionally connected I/O devices.

Single CS31 bus on CPU COM port

The following picture describes basic operation of HA-CS31 system - a typical scan cycle of an AC500 CPU with redundant CS31 slave (CI590-CS31-HA).

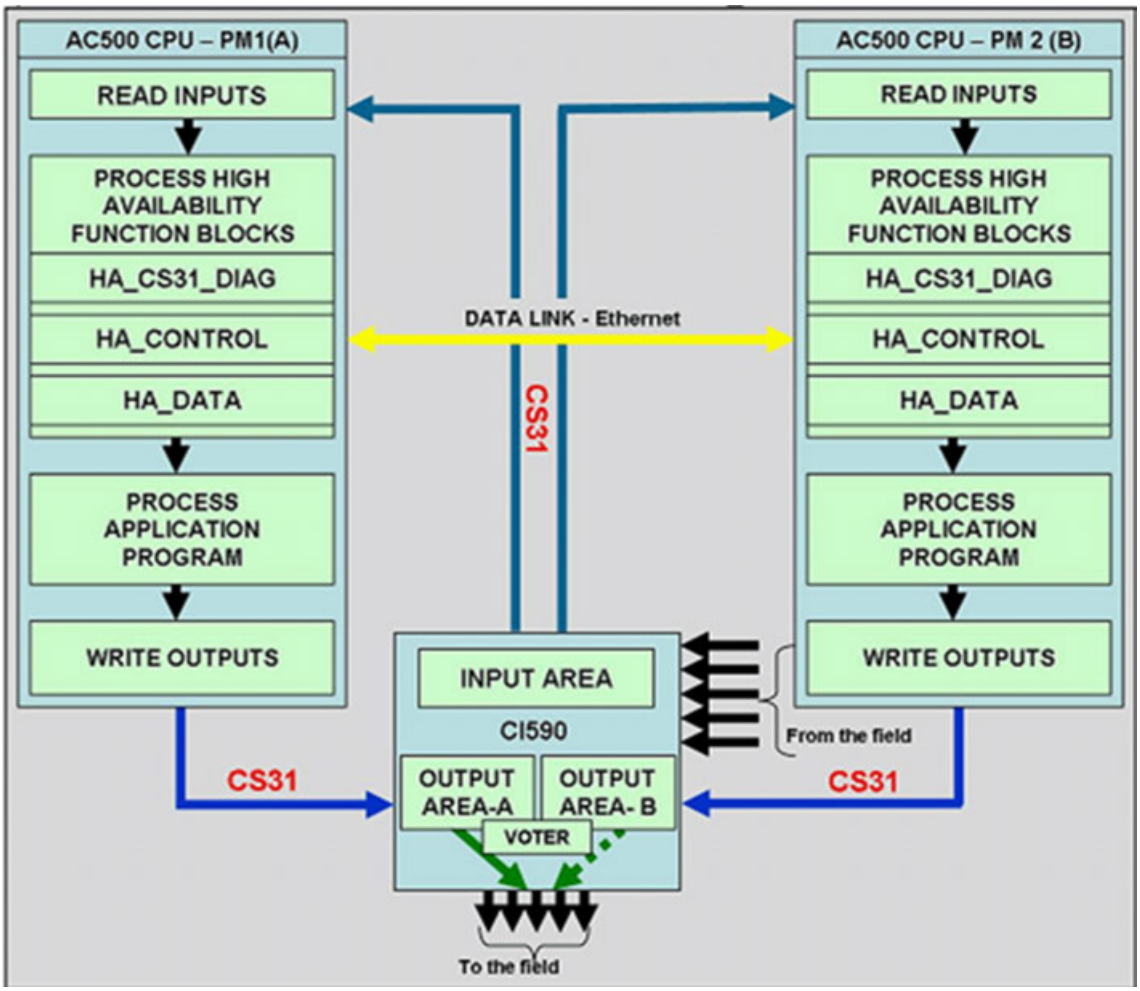


Fig. 59: System Structure AC500 CPU

Status of the inputs connected to CI590-CS31-HA is transferred to both CPUs simultaneously in every CS31 cycle. The CPUs process HA-CS31 function blocks along with the application program. At the end of the program, the generated outputs are transferred to respective buffers in the CI590-CS31-HA via CS31 Bus. Depending on the control byte and on diagnosis events, the CI590-CS31-HA selects one of the output buffers. Then, the buffer content is transferred to physical outputs through the I/O Bus.

Data transfer between the CPUs is watched by HA_CS31_CONTROL function block. HA_CS31_DATA_SYNC function block collects data from the primary CPU and creates a data table which is transferred to the secondary CPU via Ethernet link. Depending on data size, data transfer may take more than one cycle.

CS31 bus extension with CM574-RS module

As of HA-CS31 library version V2.4.0 (HA_CS31_AC500_V23.lib) the library supports CM574-RS CS31 Bus. Data transfer between CPU and CM574-RS is done with the help of the HA-CS31 library function blocks HA_CS31_DIAG_VIA_CM574 and HA_CS31_DIAG_ON_CM574.

The following picture describes the basic operation for this type of configuration:

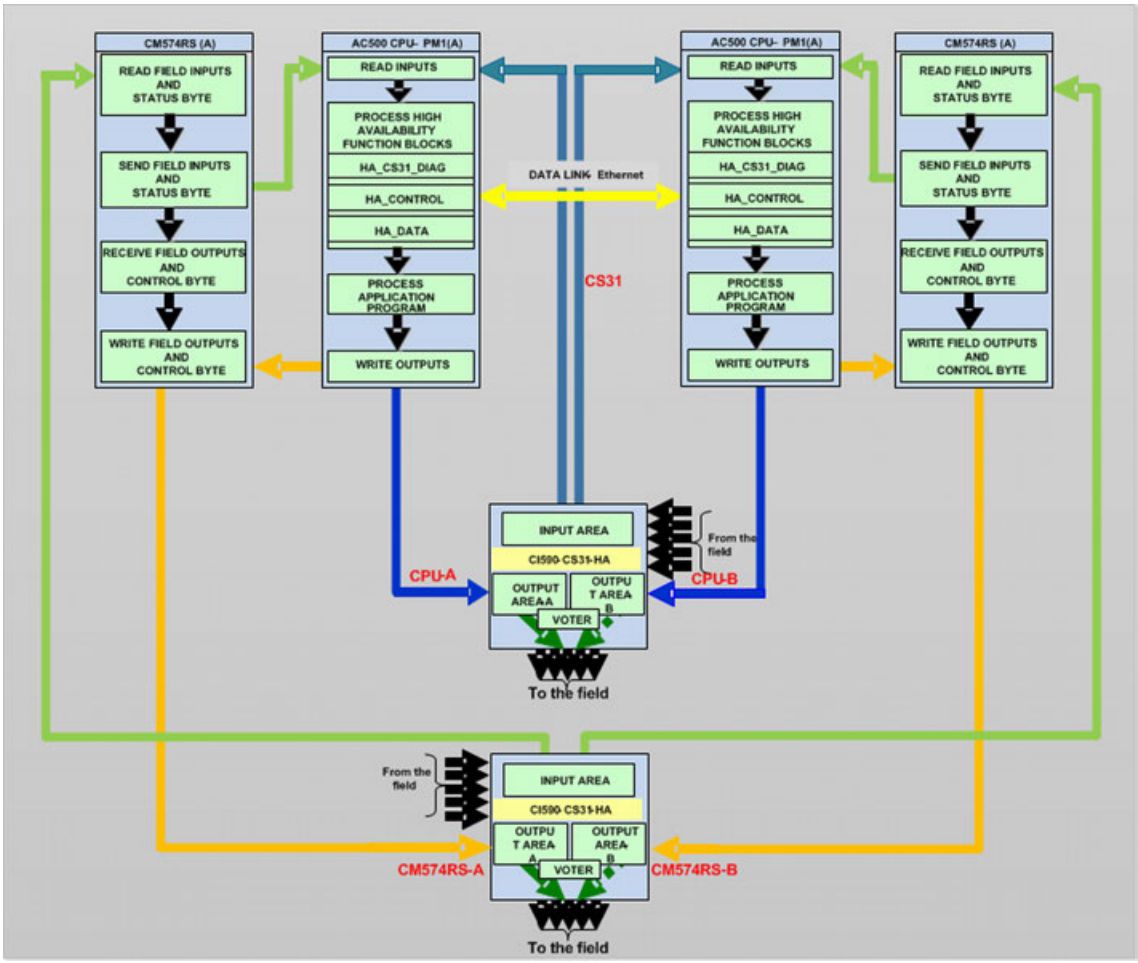


Fig. 60: Operation Process for CS31 and CM574-RS

For further information on function block operations, please refer to the online help for HA-CS31 function blocks (*Chapter 1.5.5.1.2 “AC500 High Availability CS31 system technology” on page 1983*).

Details of control and state byte

These bytes are managed by HA_CS31_CONTROL function block.

Table 99: Structure of the control byte

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit5	Bit6	Bit 7
CPU status: 0: CPU in STOP mode 1: CPU in RUN mode	Not used	Manual change over request: 0: No request 1: Request	Switch over acknowl-edge-ment: 0: No request 1: Request	Primary request 0: No pri-mary request 1: Primary request	Not used	Not used	Life status: Toggles when CS31 Bus is active

Table 100: Structure of status byte

Bit 0	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7
Line information: 0: Line B 1: Line A	Other Bus Active: 0: Not Active 1: Active	Manual change over request: 0: No Request 1: Request	Digital O/P buffer different 0: Identical 1: Different	Other Bus Primary 0: Not primary 1: Primary	Analog O/P buffer different 0: Identical 1: Different	CI590-CS31-HA Initialization 0: Not done 1: Done	Life status: Toggles when CS31 Bus is active

HA-CS31 limitations

- The HA-CS31 library does not recommend more than three CM574-RS modules for HA-CS31 Bus. For PM57x CPU only one CM574-RS is recommended.
- The HA-CS31 library will not support more than 1024 HA_CS31_DATA_SYNC function blocks in one POU.
- AC500-eCo PLCs are not supported for HA-CS31 library.
- The HA-CS31 system takes care of the first fault only. If more than one error occurs in the same PLC, the user must restart the system to reset the error.
- To avoid compatibility issues with HA-CS31 library, users must have recommended hardware with runtime system and software *Chapter 1.5.5.1.2.2 “Requirements” on page 1984*.
  For further information on CS31 limitations please refer to *Chapter 1.6.2.8.3.1 “CI590-CS31-HA” on page 4745*.

1.5.5.1.3 AC500 High Availability CS31 library

Components of HA-CS31 library

The HA-CS31 library contains the following function blocks, structures, visualizations, constants and variables.

HA-CS31 library - Overview diagram

The following table gives an overview which function blocks and functions run on CPU and on CM574 (slot 2 to 4). In the example slot 1 is used for CM597-ETH for program download. Function blocks marked in italic font are only available from HA-CS31 library version V2.4.0. In another case if a user has a PLC with more than one onboard ETH (Example: PM595, PM591-2ETH) the user can use ETH2 port for downloading e.g. the program, web server, OPC server communication etc.

CM574 program (slot 4)	CM574 program (slot 3)	CM574 program (slot 2)	CM597-ETH (slot 1)	CPU program	CS31 line
-	-	-	-	**Application**	-
-	-	-	-	HA_CS31_PID	-
-	-	-	-	HA_CS31_CTD	-

CM574 program (slot 4)	CM574 program (slot 3)	CM574 program (slot 2)	CM597-ETH (slot 1)	CPU program	CS31 line
-	-	-	-	... (further utility function blocks)	-
-	-	-	-	HA_CS31_DATA_SYNC	-
-	-	-	-	**HA Functionality**	-
-	-	-	-	HA_CONTROL	-
-	-	-	-	HA_CS31_CALLBACK_STOP	-
-	-	-	-	HA_CS31_DIAG(_EXT	01 (local)
-	-	*HA_CS31_DIAG_ON_CM574*	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*2.1 (slot 2, COM1)*
-	-	*HA_CS31_DIAG_ON_CM574*	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*2.2 (slot 2, COM2)*
-	*HA_CS31_DIAG_ON_CM574*	-	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*3.1 (slot 3, COM1)*
-	*HA_CS31_DIAG_ON_CM574*	-	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*3.2 (slot 3, COM2)*
HA_CS31_DIAG_ON_CM574	-	-	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*4.1 (slot 4, COM1)*
HA_CS31_DIAG_ON_CM574	-	-	-	*HA_CS31_DIAG(_EXT)_VIA_CM574*	*4.2 (slot 4, COM2)*

Function blocks

POU Name	Function
Group: Control	
HA_CS31_CONTROL	High Availability control block
HA_CS31_DATA_SYNC	High Availability data synchronization block
Group: Diagnosis_PM5xx	
HA_CS31_DIAG	High Availability diagnosis block for CPU COM1 CS31 Line
HA_CS31_DIAG_EXTD	High Availability extended diagnosis block for CPU COM1 CS31 Line
HA_CS31_DIAG_VIA_CM574	High Availability diagnosis block used in CPU for CM574-RS CS31 Line
HA_CS31_DIAG_EXTD_VIA_CM574	High Availability extended diagnosis block for CM574 CS31 Line
Group: Diagnosis_CM574-RS	

POU Name	Function
HA_CS31_DIAG_ON_CM574	High Availability diagnosis block used in CM574 for CM574-RS CS31 Line
Group: Utility	
HA_CS31_CTD	High Availability count down counter
HA_CS31_CTU	High Availability count up counter
HA_CS31_CTUD	High Availability Up/down counter
HA_CS31_INTEGRAL	High Availability integral function
HA_CS31_PID	High Availability PID controller
HA_CS31_PID_DV500	High Availability PID controller for DigiVis 500 face-plates
HA_CS31_PID_FIXCYCLE	High Availability PID controller with fix cycle
HA_CS31_PID_FIXCYCLE_DV500	High Availability PID controller for DigiVis 500 face-plates
HA_CS31_RAMP_INT	High Availability ramp with integer
HA_CS31_RAMP_REAL	High Availability ramp with real
HA_CS31_TOF	High Availability off delay timer
HA_CS31_TON	High Availability on delay timer

Function

Group: Callback	
HA_CS31_CALLBACK_STOP	High Availability CPU STOP event function

Visualizations

HA_CS31_DIAG_VISU_PH	Faceplate for function block HA_CS31_DIAG
HA_CS31_OVERVIEW_VISU	Visualization for HA-CS31 library overview. This is available as of HA-CS31 library version 2.4.0 (HA_CS31_AC500_V23.lib)
Group: Control	
HA_CS31_CONTROL_VISU_PH	Faceplate for function block HA_CS31_CON-TROL
HA_CS31_DATA_SYNC_VISU_PH	Faceplate for function block HA_CS31_DATA_SYNC
Group: Diagnosis_PM5xx	
HA_CS31_DIAG_EXTD_VIA_CM574_VISU_PH	Faceplate for function block HA_CS31_DIAG_EXTD_VIA_CM574
HA_CS31_DIAG_EXTD_VISU_PH	Faceplate for function block HA_CS31_DIAG_EXTD
HA_CS31_DIAG_VIA_CM574_VISU_PH	Faceplate for function block HA_CS31_DIAG_VIA_CM574

Structures

stTON_TOFFSyncData	Structure for synchronized TON and TOFF data
zHA_CS31_PID_DV500_DATA_TYPE	Structure for synchronization data for HA_CS31_PID_DV500
zHA_CS31_PID_FIX-CYCLE_DV500_DATA_TYPE	Structure for synchronization data for HA_CS31_PID_FIXCYCLE_DV500

Global variables

Group: HA_Global_Variables	
fG_HA_PRIMARY	State of the AC500 CPU (FALSE -> PM acts as secondary, TRUE -> PM acts as primary)
fG_HA_PM1_PRIMARY	Indication of primary PM (FALSE -> PM1 / IP1 is not primary, TRUE -> PM1 / IP1 acts as primary
fG_HA_CPU_STOP	Indication of PLC STOP status (FALSE -> indicates the CPU in RUN mode. If TRUE -> indicates the CPU in STOP MODE)
fG_HA_Err	High Availability error state
wG_HA_ErNo	High Availability error code
bitG_Data_ERR	High Availability error state for data synchronization
wG_DATA ERNO	High Availability error code for data synchronization
dwG_HA_OwnIP	Own IP address on sync link connection
dwG_HA_OtherIP	Other PMs IP address on sync link connection
bG_HA_Slot	Slot of interface to sync link connection
dwG_HA_ServerAlive	Life counter incremented by OPC server
byLastDataDelay	Variable to store last delay in data exchange
bitRefreshDataDelay	Bit to refresh data delay
byCntDataDelay	Data delay counts
wETH_Life	Ethernet Life Count
dwHATimersBaseTime	High Availability base timer
Group: HA_VISU_COLOR_INFO	
dwHaVisuBackgroundColor	Visualization elements background color 16#00<B><G><R>
dwHaVisuTitleColor	Visualization elements title background color 16#00<B><G><R>

Prerequisites for the use of HA-CS31 library

Introduction

This library is intended to be used for a HA-CS31 project along with two AC500 CPUs with Ethernet port (or with a Ethernet Communication Module) and CI590-CS31-HA communication interface modules. For general information on how to use CI590-CS31-HA please refer to ↳ *Chapter 1.5.5.1.2.3.4 “Task configuration” on page 1999*.

The following table specifies the mandatory function blocks to be used for HA-CS31 library in order to work with different configurations:

Serial No.	Hardware Configuration Type	Mandatory functions and function blocks	
		To be Downloaded on CPU	To be Downloaded on CM574-RS
1	Only CPU	HA_CALLBACK_STOP HA_CS31_CONTROL HA_CS31_DIAG	Not applicable
2	CPU + CM574-RS	HA_CALLBACK_STOP HA_CS31_CONTROL HA_CS31_DIAG HA_CS31_DIAG_VIA_CM574	HA_CS31_DIAG_ON_CM574

Only the I/O channels of the CI590-CS31-HA and those of connected I/O modules are covered under high availability. It is possible to use local I/O modules, but only with standard functionality and not with high availability feature.

The limitations of the HA-CS31 Bus are the same as of the standard CS31 Bus.

It is not possible to use or mix any other CS31 slave (e.g. DC551-CS31) with a CI590-CS31-HA slave.

The HA-CS31 system only takes care of the first occurring error. If both PLCs are erroneous, then the CPU which generates the second error will remain primary CPU - independent of further errors or manual switch over command.

Hardware configuration

General preconditions

The AC500 I/O configuration on redundant CS31 slave interface (CI590-CS31-HA) must be identical in both CPUs. To avoid any differences in configurations we recommend you, to create a single Automation Builder project for high availability. Then import this project into both CPUs.

CS31 Bus wiring: The CPU connected to CI590-CS31-HA Bus A will be CPU A and will be the primary CPU in usual operation mode. The CPU connected to CI590-CS31-HA Bus B will be CPU B and will be the secondary CPU.

We recommend you, to use separate CM597-ETH modules for larger systems, refer to
↳ *Chapter 1.5.5.1.2.2.1.2 “Recommended use of Ethernet connections” on page 1991*.

In order to use CI590-CS31-2FC (Fast counter) the users have to perform the following:

- CI590-CS31-2FC must be added to their CS31 network.
- Section “Fast Counter” must be updated with the appropriate counter type. The default setting "No Counter” leads the system to become unstable.

For further information please refer to
Example_AC500_HA_CS31_V242_3ADR023070M0201.pdf.

Preconditions when using more than one CS31 line (AC500 CPU configuration + CM574-RS)

- Download the CPU configuration before downloading configuration and program to CM574-RS.
- CM574-RS is a programmable co–processor module with two serial communication ports. Configuration and programming of the CM574-RS module is done by creating a separate node in Automation Builder project. In case of problems, please check configuration of CM574-RS below AC500 CPU node: “Enable debug” must be set to “On”.
- The AC500 CPU node in Automation Builder project must be downloaded to AC500 CPU before downloading CM574-RS configuration and the program to CM574-RS.
- If more than one CS31 line is configured, the users have to set the “Max Wait Run” under each CS31 Bus line based on the actual time taken for all CS31 lines to come online. This will ensure that all outputs of the I/O modules get activated at the same time.
- If the CS31 lines have different sizes their startup time will be different. In order to avoid an unsynchronized I/O module startup, users have to set the “Max Wait Run” under each CS31 Bus line and it must be greater than the actual time taken for all CS31 lines to come online. This will ensure that all I/O module outputs will get activated at the same time.

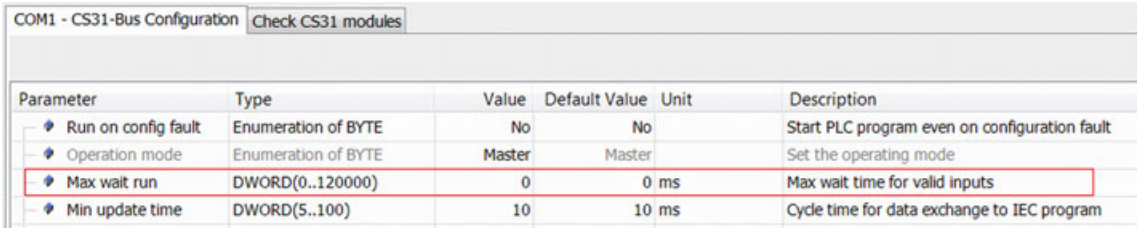


Fig. 61: Configuration in Automation Builder

Program

We recommend you, to use an AC500 PLC node (Automation Builder configuration + program) download to both PLCs, create boot project in both and restart the complete system.

We recommend you, to use the IP address of CPU A as IP1 in HA_CS31_CONTROL function block.

HA-CS31 utility function blocks are synchronized internally. Any other synchronization requirement is to be done through HA_CS31_DATA_SYNC function block.

Restart the complete system if any of the HA-CS31 library function blocks changed. Else the system may become unstable as HA library function blocks are not supporting online changes.

If online program modification is required on a running high availability application (e.g. additional variable, number or type of the variable in HA_CS31_DATA_SYNC), it is necessary to restart the whole system to recreate synchronization.

SFC language (Sequential Function Chart) is not recommended for HA programming as too much and too spread data is to synchronize.

If web visualization is required, the following setup is recommended: Onboard Ethernet for web server and OPC & Automation Builder communication, CM597-ETH for HA data synchronization, refer to *Chapter 1.5.5.1.2.2.1.2 “Recommended use of Ethernet connections” on page 1991*.

For web visualization users have to use onboard Ethernet slot. If a user has a PLC with more than one onboard ETH port (Example: PM595, PM591-2ETH) the user can use ETH1 port for UDP data exchange and ETH2 for web server.

If a user is using a PLC with one onboard ETH port and requires web visualization, then use onboard ETH for web visualization (CM597-ETH does not support web server) and CM597-ETH for UDP data exchange. This change may be able to make some impact on UDP data exchange performance as CM597-ETH port has a lower priority than the PLC on board port.

Use DigiVis and OPC server for writing data as it will be written to both PLCs at the same time. We recommend you, not to use HMI for writing data as it is not written to both PLCs.

PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.

Task configuration

The default task in the CPU can be used for the application program, containing the utility blocks. Beyond this the following tasks have to be created:

- HA_Task on CPU, containing the CONTROL and DIAG function blocks.
- HA_Task on CM574, containing DIAG_ON_CM574 function blocks.

The cycle times are dependent on the CS31 cycle time which is calculated by the Automation Builder and indicated in the CS31 Bus node (tab “Check CS31 modules”). Minimum HA CPU cycle time must be either 30 ms or (CS31 cycle time) whichever value is higher. HA-CM574-RS cycle time must be always lesser than the CPU cycle time.

HA_CS31_CALLBACK_STOP function must be called in CPU program as a Stop Event under SYSTEM EVENTS with the name starting with "callback" (e.g. "callback_stop").

Proceed as follows:

- Select: *Task configuration -> system event*.
- Enable the system event **Stop**.
- In the ‘Called POU’ column type in the POU name, starting with CALLBACK (e.g. CALLBACK_STOP).
- Click the ‘Create POU <individual POU name>’ button to create the CALLBACK_STOP (FUN) function in the POU project.
- Switch to CALLBACK_STOP POU and insert HA_CS31_CALLBACK_STOP function.

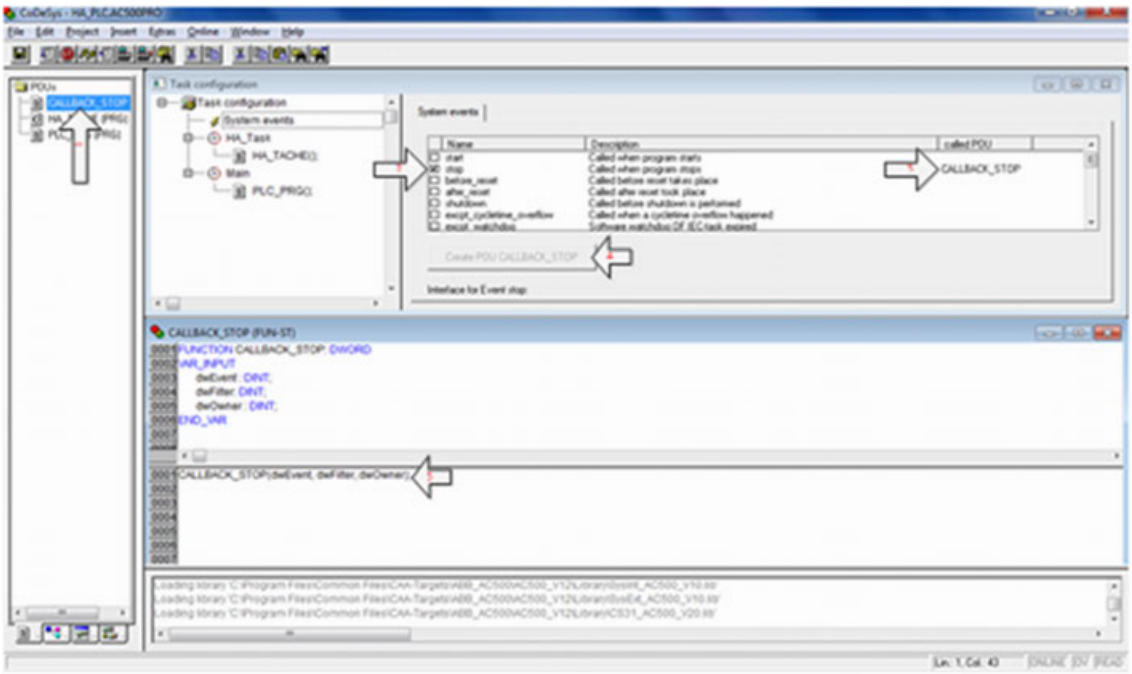


Fig. 62: Callback stop event

If a customer has downloaded the program to the PLC with a wrong callback stop event name then the CPU freezes and is unable to log in/on to the PLC using Ethernet cable. Users have to download a blank project to the PLC using RS-232 communication cable (TK501) in order to return the PLC into healthy stage. After this, the users have to correct the callback stop event name as recommended and restart the download.

The callback stop event should follow the name pattern “CALLBACK_xxx”.

The blocks contained in the library can only be executed in RUN mode of the PLC, but not in 'simulation' mode.

Control and diagnosis - Function blocks and functions

HA_CS31_CALLBACK_STOP - HA CPU STOP event function

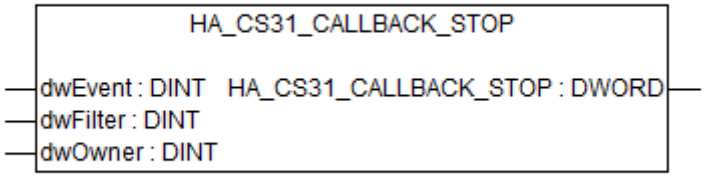


Fig. 63: Function HA_CS31_CALLBACK_STOP

Table 101: General information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4). In previous library versions this component was a program (not a function).
Type	DWORD (previously this function was a program)

This mandatory function is intended to detect the CPU stop event and process logic related to HA-CS31 project. For further information please refer to *Chapter 1.5.5.1.3.2.4 “Task configuration” on page 2023*.

HA_CS31_CALLBACK_STOP function is programmed to process logic related to HA-CS31 project in case of CPU switching into stop mode.

If there is a STOP event initiated in the PLC, HA_CS31_CALLBACK_STOP function is processed before the CPU switches to STOP state. This function sets/resets some HA-CS31 library global variables, to provide the one CPU’s STOP information to the other CPU.

Input description

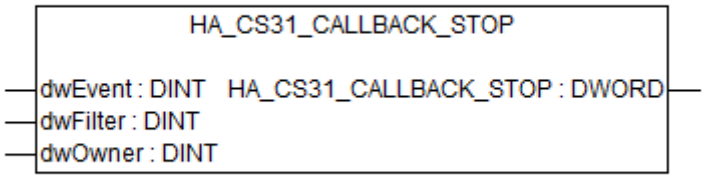


Fig. 64: Function HA_CS31_CALLBACK_STOP

dwEvent Data type: DINT.
Runtime system event in which the function block is to be called.

dwFilter Data type: DINT.
Runtime system event filter.

dwOwner Data type: DINT.
Runtime event system source.

Output description

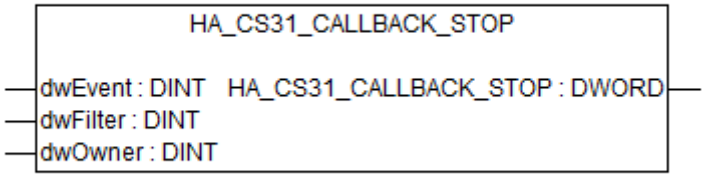


Fig. 65: Function HA_CS31_CALLBACK_STOP

dwHACS31Call-backStop Data type: DWORD.

Function call in ST

```
HA_CS31_CALLBACK_STOP(dwEvent, dwFilter, dwOwner);
```

HA_CS31_CONTROL - HA control FB

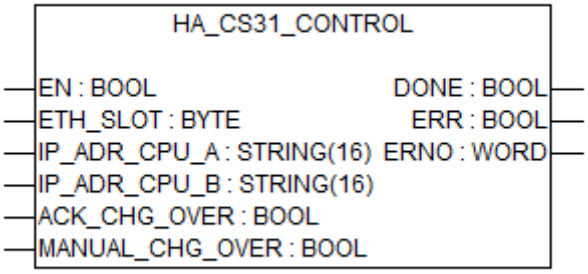


Fig. 66: Function Block HA_CS31_CONTROL

Table 102: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block handles HA-CS31 operation such as change over from primary to secondary CPU in case of an error with related diagnosis. Further it is used for data transfer between high availability CPUs. This is a mandatory function block for high availability application. HA_CS31_CONTROL receives the status data from all seven CS31 lines and sends back the control data.

This function block works with the respective diagnosis function blocks for sending and receiving diagnosis information.

Input description

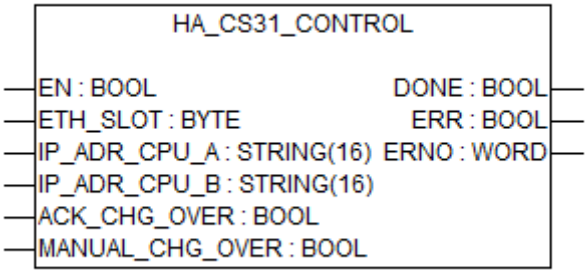


Fig. 67: Function Block HA_CS31_CONTROL

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

ETH_SLOT Data type: BYTE, default value: 0, range: 0 to 4.

Ethernet slot number. At input ETH_SLOT the communication module slot (module number) is selected which shall be used by the block.

The internal communication module always has the module number 0. All external communication modules are serially numbered from right to left, starting with module number 1.

> *PM591-2 ETH and PM595 PLCs must use onboard ETH1 for UDP data communication.*

IP_ADR_CPU_A Data type: STRING, default value: 0.0.0.0, range: 000.000.000.000 to 255.255.255.255.

IP address of the AC500 CPU connected to line A.

IP_ADR_CPU_B Data type: STRING, default value: 0.0.0.0, range: 000.000.000.000 to 255.255.255.255

IP address of the AC500 CPU connected to line B.

ACK_CHG_OVER Data type: BOOL, default value: FALSE, range: TRUE/FALSE

The user can acknowledge the changeover event.

MANUAL_CHG_OVER Data type: BOOL, default value: FALSE, range: TRUE/FALSE

A manual change in the primary CPU to value TRUE forces a changeover from the primary CPU to the secondary CPU.

Output description

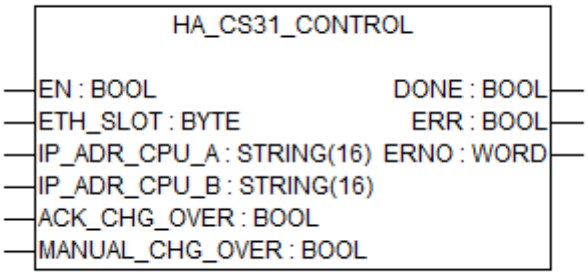


Fig. 68: Function Block HA_CS31_CONTROL

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function block call in ST

```
HACS31control(
EN                      := hacs31_EN,
ETH_SLOT                := hacontrol_ethslot,
IP_ADR_CPU_A            := hacontrol_ipadr_a,
IP_ADR_CPU_B            := hacontrol_ipadr_b,
ACK_CHG_OVER            := hacontrol_ackn,
MANUAL_CHG_OVER         := hacontrol_man_chnge_ovr);
hacontrol_done          := HACS31control.DONE;
hacontrol_err           := HACS31control.ERR;
hacontrol_erno          := HACS31control.ERNO;
```

HA_CS31_DATA_SYNC - HA data synchronization FB

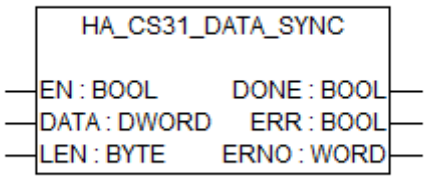


Fig. 69: Function block HA_CS31_DATA_SYNC

Table 103: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

The function block HA_CS31_DATA_SYNC is intended to be used for synchronizing different instances of function blocks in the HA-CS31 project. It is needed only if the user is not using HA-CS31 utility function blocks.

HA_CS31_DATA_SYNC is part of the HA-CS31 library and is used for synchronization. It collects data and size information of the connected function block instance and delivers those details to the HA_CS31_CONTROL function block for data synchronization. Users can use up to 1024 instances of HA_CS31_DATA_SYNC in one program. The maximum of 1024 bytes (user data) can be synchronized in one cycle. In previous library versions (< 2.4.0) the users can use only up to 256 instances of HA_CS31_DATA_SYNC.

To start synchronization HA_CS31_DATA_SYNC should be enabled in both CPUs at least once. Once the function block is enabled, it runs till the system is powered off (regardless of an enabled input status). We recommend you, always to keep ENABLE input as TRUE instead of connecting a variable.

Functionality

The following figure describes the function blocks which are mandatory for data synchronization in all switchover conditions and which are necessary for a complete redundant system.

HA_CS31_SYNC will help users to synchronize different function block instances, variables, arrays etc. HA_CS31_TON is a utility function block available in HA-CS31 library and the outputs of this function block are internally synchronized. All utility function blocks which are available in HA-CS31 library are internally synchronized.

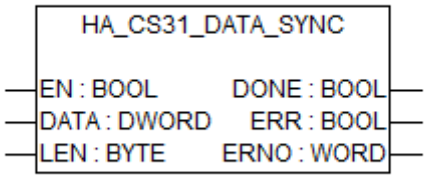


Fig. 70: Function block PUMP_LEVEL_CTRL

Data synchronization is used for writing the values of the defined variables or internal historical values of function blocks from the primary CPU to the secondary CPU. Users need this in case of external change or after repair. Please note that the data is always written from primary to secondary CPU.

Rules for data synchronization

- All instances on HA_CS31_DATA_SYNC must be executed at least once to have a correct data synchronization table.
- We recommend you, to have all instances of HA_CS31_DATA_SYNC and HA-CS31 function blocks to be called in one separate POU. This POU should be called in PLC_PRG.
- During commissioning the users have to use OPC client to tune the parameters of HA-CS31 utility function blocks to make sure that both PLCs are sending/receiving data at the same time without any time delay.

Synchronization of function blocks

The following figure describes how function blocks are provided by HA_CS31_AC500_V23 library. Internal historical values are placed in an exchange table and copied from the primary to the secondary CPU. Then, the secondary CPU is synchronized with the primary and the process outputs are identically on both CPUs.

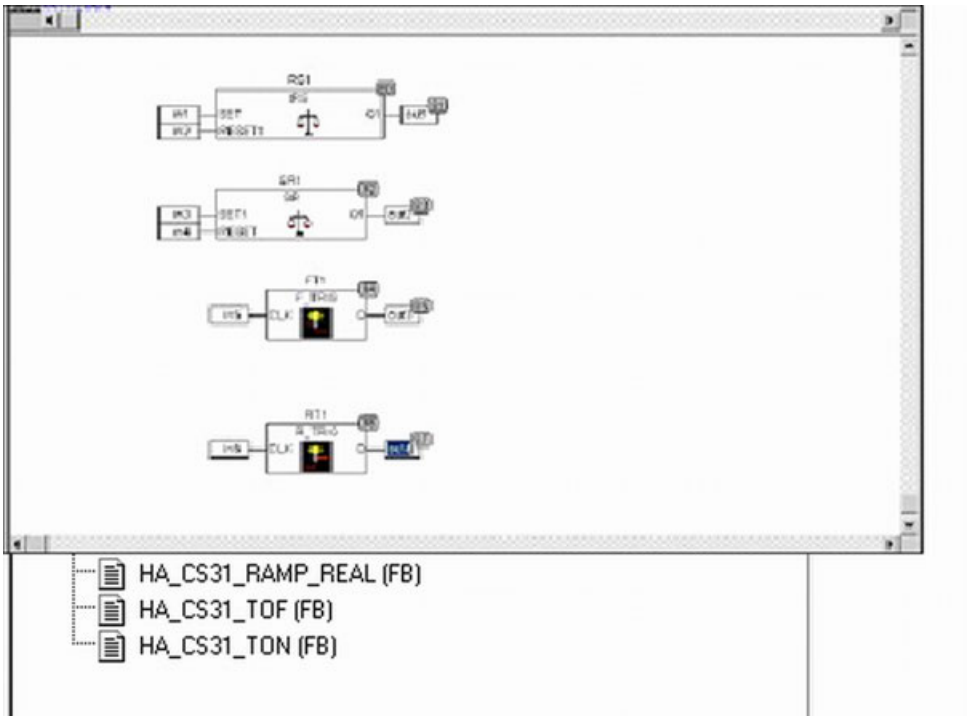


Fig. 71: Function block PUMP_LEVEL_CTRL

The following function blocks with historical values are not included in the HA_CS31_AC500_V23 library:

- Set Reset (SR)
- Reset Set (RS)
- Rising edge trigger (R_TRIG)
- Falling edge trigger (F_TRIG)
- etc.

To be able to resynchronize these values after an error or a PLC stop, these internal values can be added in a synchronization table.

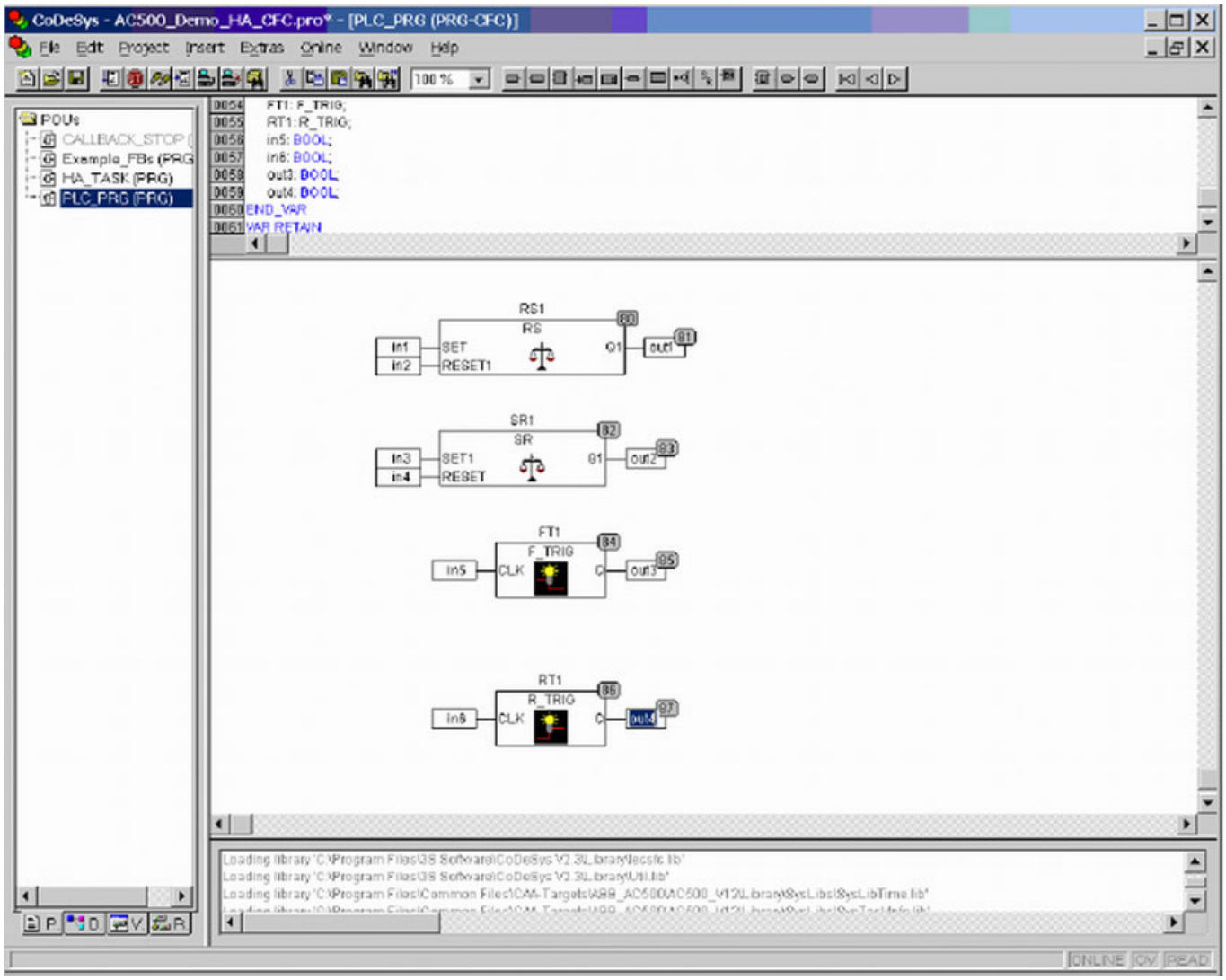


Fig. 72: Function block PUMP_LEVEL_CTRL

Internal values to synchronize are:

- RS1.Q1 (internal value of RS1)
- SR1.Q1 (internal value of SR1)
- FT1.Q (internal value of FT1)
- RT1.Q (internal value of RT1)

User data synchronization

The variables which are changed by a SCADA/ HMI on the primary CPU should be synchronized, e.g:

- Set points, limits, on/off command, etc.
- Variables used to manage program sequences.
- Sequence number, condition with historical values, etc.

Tips for data synchronization:

To optimize data exchange and synchronization in a High Availability program data can be grouped in a structure as shown in the following figure to avoid multiple entries in synchronization table.

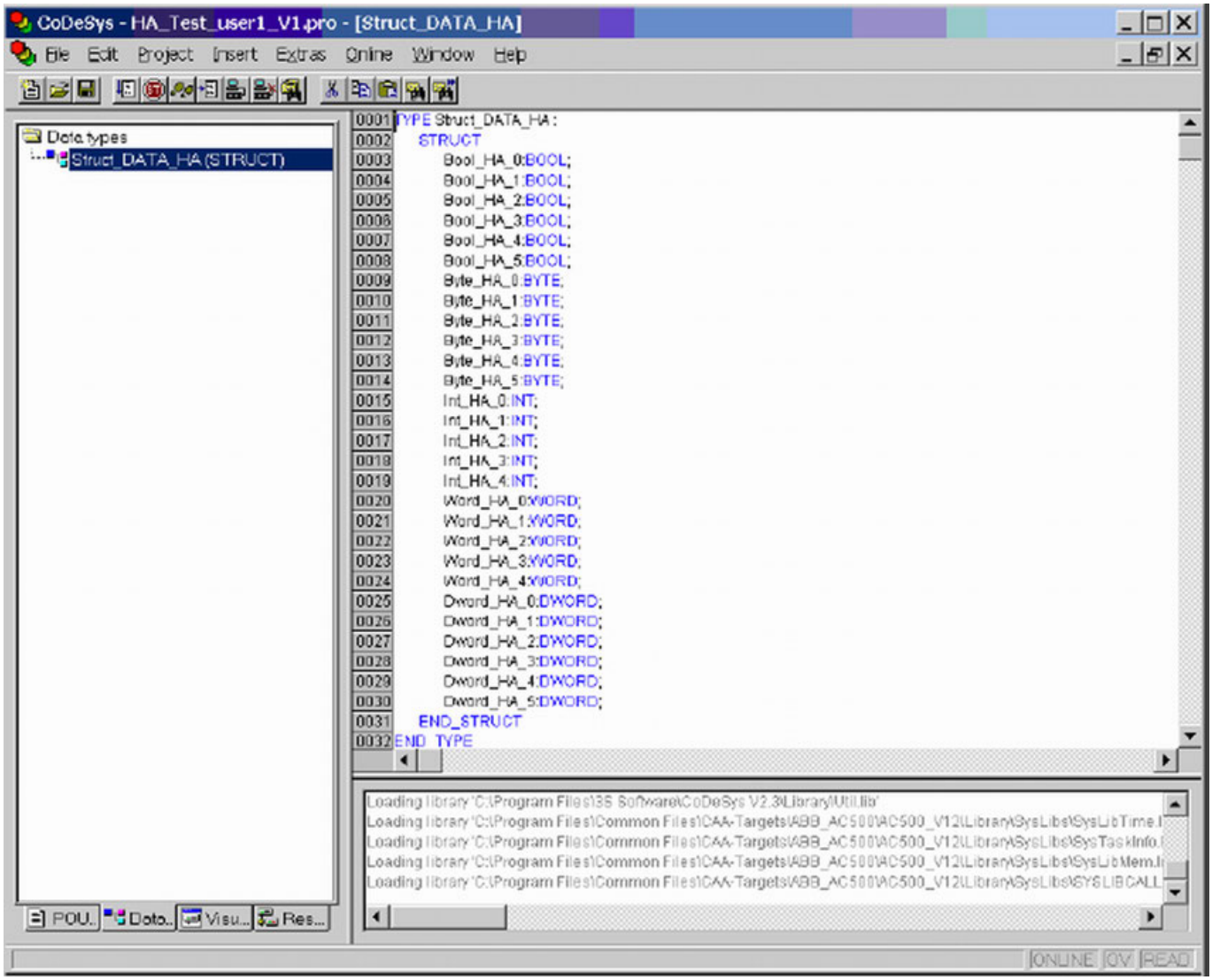


Fig. 73: Function block PUMP_LEVEL_CTRL

Then, one complete structure can be placed in a synchronization table. In the example all variables of the structure data will be synchronized using only one instance of HA_CS31_DATA_SYNC.

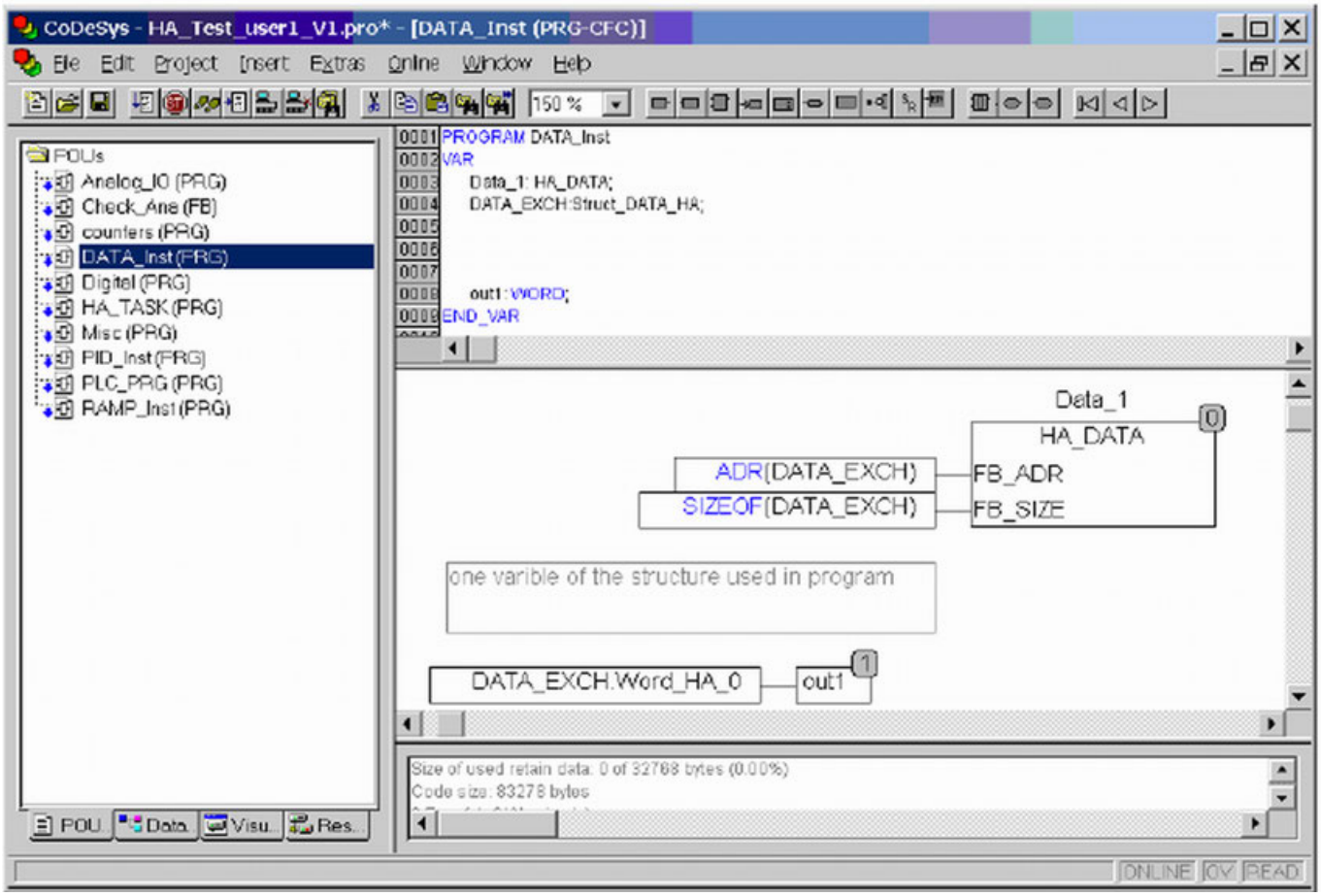


Fig. 74: Function block PUMP_LEVEL_CTRL

Input description

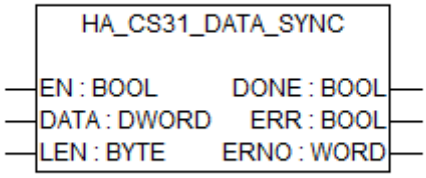


Fig. 75: Function block HA_CS31_DATA_SYNC

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

DATA

Data type: DWORD, default value: 0, range: 0-255.

Start address of the parameters/ variables to be synchronized (via ADR-operator).

LEN (length) Data type: BYTE, default value: 0, range: 0-255.

Length of the parameters/variables to be synchronized (via SIZEOF-operator).

Output description

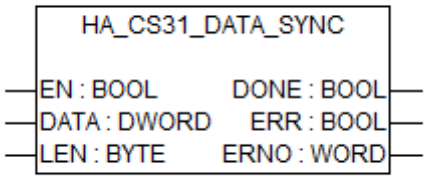


Fig. 76: Function block HA_CS31_DATA_SYNC

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function block call in ST

```
HACS31_datasync(
EN       := hacs31_sync_EN,
DATA     := ADR(VAR1),
LEN      := SIZEOF(VAR1),
DONE     := hacs31_data_done,
ERR      := hacs31_data_err,
ERNO     := hacs31_data_erno);
```

HA_CS31_DIAG - Reading HA diagnosis

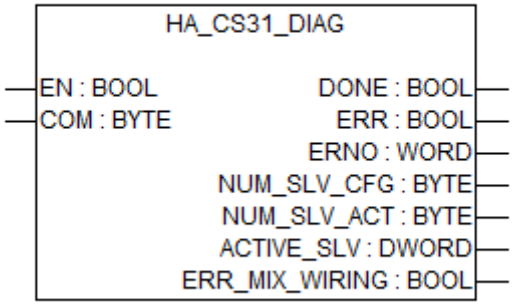


Fig. 77: Function block HA_CS31_DIAG

Table 104: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block handles the control and status bytes of each CI590-CS31-HA module on the CPU COM1 CS31 Bus and related diagnostics. It is a mandatory function block for CPU COM CS31 Bus to work.

HA_CS31_DIAG will be used to get diagnosis information from CPU COM1 CS31 line only. It handles the control and status bytes of each CI590-CS31-HA module on the CS31 Bus. It is also possible to find information such as configured and active CI590-CS31-HA slaves available on CPU COM1 CS31 Bus and related diagnosis.

Input description

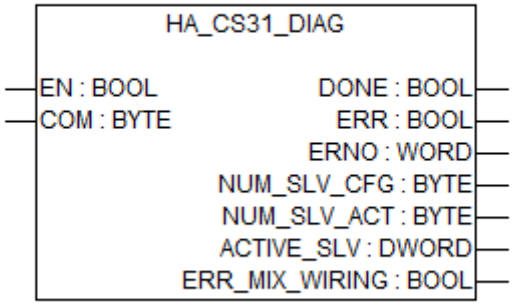


Fig. 78: Function block HA_CS31_DIAG

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

COM (communication port)

Data type: BYTE, default value: 1, range: 1.

CS31 communication port number. At input COM, the port number of the serial interface on AC500 CPU for CS31 protocol is selected.

Output description

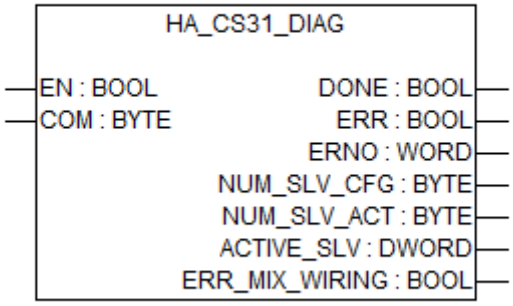


Fig. 79: Function block HA_CS31_DIAG

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SLV_CFG (number of slaves configured) Number of CI590-CS31-HA slaves configured in the PLC configuration. The output NUM_SLV_CFG indicates the number of CI590-CS31-HA slaves configured in the PLC configuration.

NUM_SLV_ACT (number of active slaves) Number of active CI590-CS31-HA slaves in the CS31 Bus. The output NUM_SLV_ACT indicates the number of active CI590-CS31-HA slaves available in the CS31 Bus.

ACTIVE_SLV (active slaves) Each bit of this DWORD represents an active CI590-CS31-HA device on the CS31 Bus. Each bit of the output ACTIVE_SLV indicates an active CI590-CS31-HA in the CS31 Bus.

ERR_MIX_WIRING (error of mix-wiring) Error in case of mix-wiring between Bus line A and Bus line B of CS31 slaves. Output ERR_MIX_WIRING indicates a mix-wiring (cross-wiring) between line A and line B of CI590-CS31-HA slaves.

Function block call in ST

```
HACS31diag(
EN                  := hacs31_EN,
COM                 := hacs31_COM);
hacs31_done         := HACS31diag.DONE;
hacs31_err          := HACS31diag.ERR;
hacs31_erno         := HACS31diag.ERNO;
hacs31_num_cfg_slv  := HACS31diag.NUM_SLV_CFG;
hacs31_num_act_slv  := HACS31diag.NUM_SLV_ACT;
hacs31_act_slv      := HACS31diag.ACTIVE_SLV;
hacs31_err_mix_wire := HACS31diag.ERR_MIX_WIRING;
```

HA_CS31_DIAG_EXTD - HA Extended Diagnosis FB

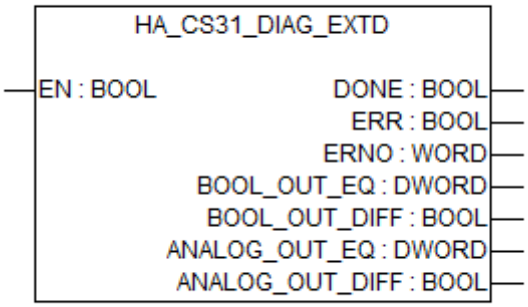


Fig. 80: Function block HA_CS31_DIAG_EXTD

Table 105: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block reads the extended diagnosis from CPU COM1 CS31 Bus. It works only if HA_CS31_DIAG is enabled in the CPU program.

HA_CS31_DIAG_EXTD is an optional function block which can be used to read the extended diagnosis from CI590-CS31-HA slave module. It is used in order to get extended diagnosis information from CPU COM1 CS31 line only.

Input description

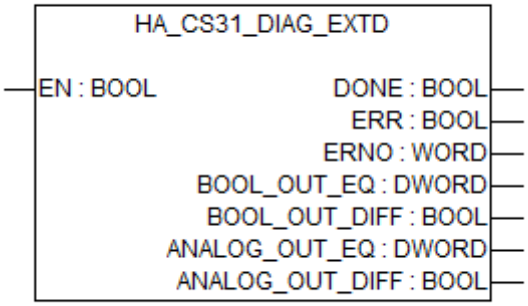


Fig. 81: Function block HA_CS31_DIAG_EXTD

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

Output description

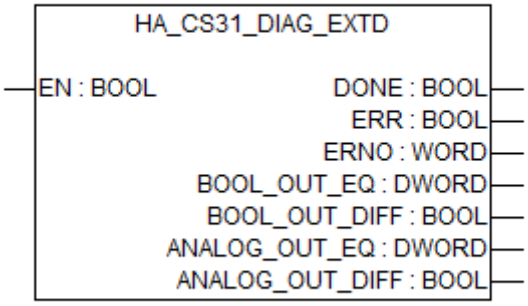


Fig. 82: Function block HA_CS31_DIAG_EXTD

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BOOL_OUT_EQ (bool out equality)

Data type: DWORD.

Each bit of this DWORD represents a CI590-CS31-HA with identical buffer for digital type.

BOOL_OUT_DIFF (bool out difference)

Data type: BOOL.

Error bit to indicate that the output buffer for digital type data is different in one or more CI590-CS31-HA which are active on the Bus.

ANALOG_OUT_EQ (analog out equality)

Data type: DWORD.

Each bit of this DWORD represents a CI590-CS31-HA with identical buffer for analog type.

ANALOG_OUT_DIFF (analog out difference)

Data type: BOOL.

Error bit to indicate that the output buffer for analog type data is different in one or more CI590-CS31-HA which are active on the Bus.

Function block call in ST

```
HACS31_extdiag(
EN                    :=    hacs31_ext_EN);
hacs31_ext_done       :=    HACS31_extdiag.DONE;
hacs31_ext_err        :=    HACS31_extdiag.ERR;
hacs31_ext_erno       :=    HACS31_extdiag.ERNO;
hacs31_ext_out_eq     :=    HACS31_extdiag.BOOL_OUT_EQ;
hacs31_ext_out_diff   :=    HACS31_extdiag.BOOL_OUT_DIFF;
hacs31_ext_ana_out_eq :=    HACS31_extdiag.ANALOG_OUT_EQ;
hacs31_ext_ana_out_diff :=  HACS31_extdiag.ANALOG_OUT_DIFF;
```

HA_CS31_DIAG_EXTD_VIA_CM574 - HA extended diagnosis FB

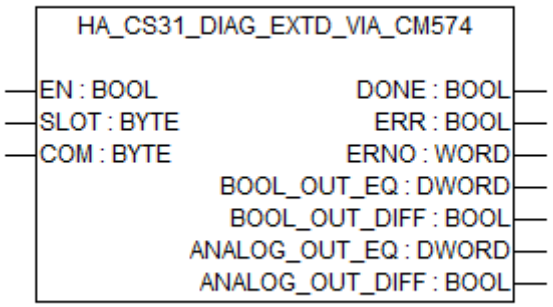


Fig. 83: Function block HA_CS31_DIAG_EXTD_VIA_CM574

Table 106: General Information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4) and above
Type	Function block with historical values.

This function block reads the extended diagnosis of CM574-RS CS31 Bus in High Availability operation. HA_CS31_DIAG_EXTD_VIA_CM574 works only if HA_CS31_DIAG_VIA_CM574 is enabled in the CPU program.

Input description

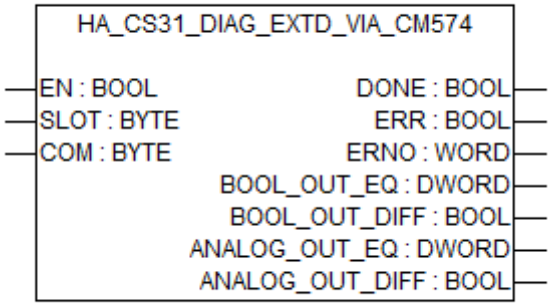


Fig. 84: Function block HA_CS31_DIAG_EXTD_VIA_CM574

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

COM

Data type: BYTE, default value: 1, range: 1 to 2.

CS31 communication port number of the CS31 master.

Output description

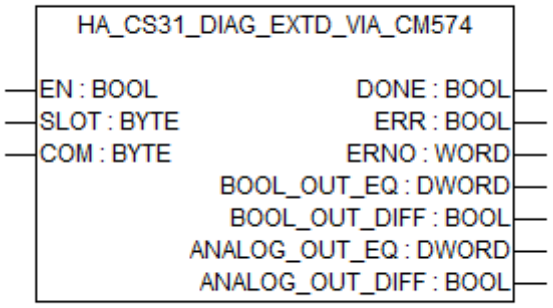


Fig. 85: Function block HA_CS31_DIAG_EXTD_VIA_CM574

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BOOL_OUT_EQ (BOOL out equal) Data type: DWORD.
Each bit of the DWORD represents identical output buffer for BOOL data in each CS31 slave.

BOOL_OUT_DIFF (BOOL out difference) Data type: DWORD.
If PLC A and PLC B bool output buffer is different, then this bit will turn on to TRUE.

ANALOG_OUT_EQ (analog out equal) Data type: DWORD.
Each bit of the DWORD represents identical output buffer for analog data in each CS31 slave.

ANALOG_OUT_DIFF (analog out different) Data type: DWORD.
If PLC A and PLC B analog output buffer is different, then this bit will turn on to TRUE.

Function block call in ST

```
HACS31_extdiagviaCM574-RS(
EN                                  :=    hacs31_extdiagviaCM574_EN;
SLOT                                :=    hacs31_extdiagviaCM574_EN;
COM                                 :=    hacs31_extdiagviaCM574_EN);
hacs31_extdiagviaCM574_done         :=    HACS31_extdiagviaCM574.DONE;
hacs31_extdiagviaCM574_err          :=    HACS31_extdiagviaCM574.ERR;
hacs31_extdiagviaCM574_erno         :=    HACS31_extdiagviaCM574.ERNO;
hacs31_extdiagviaCM574_out_eq       :=
HACS31_extdiagviaCM574.BOOL_OUT_EQ;
hacs31_extdiagviaCM574_out_diff     :=
HACS31_extdiagviaCM574.BOOL_OUT_DIFF;
hacs31_extdiagviaCM574_ana_out_eq   :=
HACS31_extdiagviaCM574.ANALOG_OUT_EQ;
hacs31_extdiagviaCM574_ana_out_diff :=
HACS31_extdiagviaCM574.ANALOG_OUT_DIFF;
```

HA_CS31_DIAG_ON_CM574 - HA diagnosis FB on CM574-RS

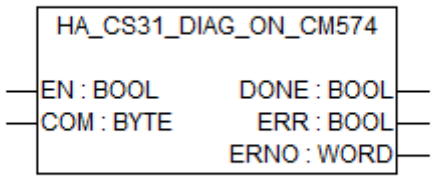


Fig. 86: Function block HA_CS31_DIAG_ON_CM574

Table 107: General Information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4) and above
Type	Function block with historical values.

This function block will work only on CM574-RS and in combination with enabled HA_CS31_DIAG_VIA_CM574 on the CPU. It is a mandatory function block for CM574-RS CS31 Bus to work. HA_CS31_DIAG_ON_CM574-RS sends CS31 Bus diagnosis information to host CPU and receives CS31 configuration information from host CPU.

This function block has to be called separately for each CM574-RS CS31 line (number of CM574-RS CS31 = number of HA_CS31_DIAG_ON_CM574).

Input description

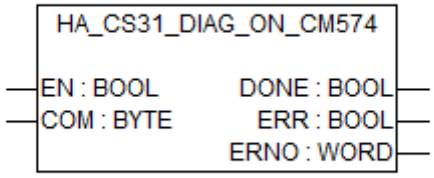


Fig. 87: Function block HA_CS31_DIAG_ON_CM574

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

COM

Data type: BYTE, default value: 1, range: 1 to 2.

CS31 communication port number of the CS31 master.

Output description

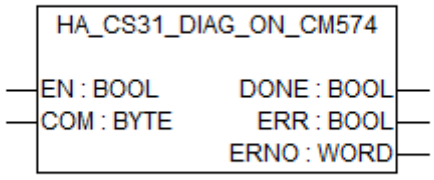


Fig. 88: Function block HA_CS31_DIAG_ON_CM574

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Function block call in ST

```
HACS31diagonCM574(
EN                      :=    hacs31diagonCM574_EN,
COM                     :=    hacs31diagonCM574_COM);
hacs31diagonCM574_done  :=    HACS31diagonCM574.DONE;
hacs31diagonCM574_err   :=    HACS31diagonCM574.ERR;
hacs31diagonCM574_erno  :=    HACS31diagonCM574.ERNO;
```

HA_CS31_DIAG_VIA_CM574 - HA diagnosis FB for CM574-RS

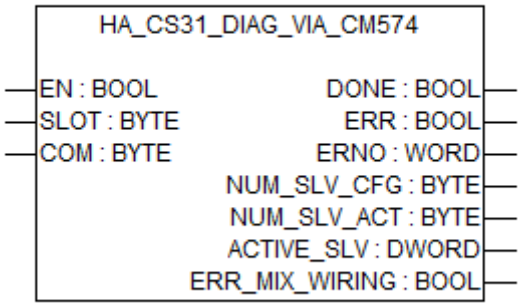


Fig. 89: Function block HA_CS31_DIAG_VIA_CM574

Table 108: General Information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4) and above
Type	Function block with historical values.

This function block will work only on the CPU and in combination with an enabled HA_CS31_DIAG_ON_CM574 in CM574-RS program. HA_CS31_DIAG_VIA_CM574 reads the status byte and writes the control byte of all CI590-CS31-HA connected on CM574-RS COM with relevant diagnosis. This is a mandatory function block for CM574-RS CS31 Bus to work.

This function block has to be called separately for each CM574-RS CS31 line (number of CM574-RS CS31 = number of HA_CS31_DIAG_ON_CM574).

Input description

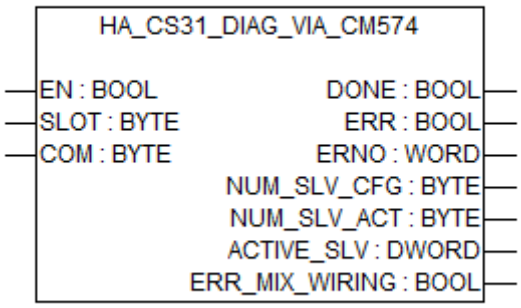


Fig. 90: Function block HA_CS31_DIAG_VIA-CM574

EN (enable)

Data type	Default value	Range	Unit
BOOL	-	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and an error is displayed.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

COM

Data type: BYTE, default value: 1, range: 1 to 2.

Communication port number of the CS31 master.

Output description

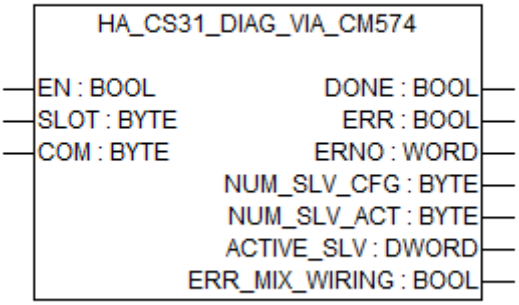


Fig. 91: Function block HA_CS31_DIAG_VIA-CM574

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NUM_SLV_CFG (number of slave configured)

Data type: BYTE.

This output indicates the number of CI590-CS31-HA slaves configured in the PLC configuration.

NUM_SLV_ACT (number of slave active)

Data type: BYTE

This output indicates the number of active CI590-CS31-HA slaves available in CS31 Bus.

ACTIVE_SLV (active slaves)

Data type: DWORD

Each bit of this output indicates an active CI590-CS31-HA slave in the CS31 Bus.

ERR_MIX_WIRING (Error Mix Wiring)

Data type: BOOL

This output indicates a mix-wiring (cross-wiring) between line A and line B of CI590-CS31-HA slaves.

Function block call in ST

```
HACS31diagviaCM574(
EN                          :=    hacs31diagviaCM574_EN,
COM                         :=    hacs31diagviaCM574_COM);
hacs31_viaCM574_done        :=    HACS31diagviaCM574.DONE;
hacs31_viaCM574_err         :=    HACS31diagviaCM574.ERR;
hacs31_viaCM574_erno        :=    HACS31diagviaCM574.ERNO;
hacs31_viaCM574_num_cfg_slv :=    HACS31diagviaCM574.NUM_SLV_CFG;
hacs31_viaCM574_num_act_slv :=    HACS31diagviaCM574.NUM_SLV_ACT;
hacs31_viaCM574_act_slv     :=    HACS31diagviaCM574.ACTIVE_SLV;
hacs31_viaCM574_err_mix_wire :=   HACS31diagviaCM574.ERR_MIX_WIRING;
```

Utility function blocks

HA_CS31_CTD - HA count down counter

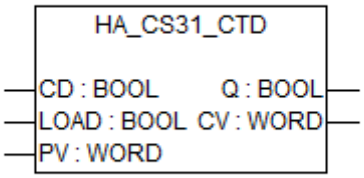


Fig. 92: Function block HA_CS31_CTD

Table 109: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a standard count down counter with automatic data synchronization in a high availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

If parameter LOAD=TRUE, parameter CV (**C**ounter **V**ariable) will be initialized with the upper limit of PV (**P**reset **V**alue). If parameter CD (**C**ount **D**own) has a rising edge from FALSE to TRUE, CV will be lowered by 1, provided that CV > 0 (i.e., it does not cause the value to fall below 0).

Input description

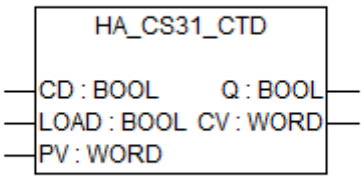


Fig. 93: Function block HA_CS31_CTD

CD Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

Count down with rising edge.

LOAD Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

This parameter loads the parameter PV (preset value) into the parameter CV (counter value) with rising edge at LOAD.

PV Data type: WORD, default value: 0, range: 0 to 65535.

Preset value.

Output description

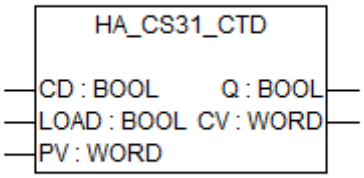


Fig. 94: Function block HA_CS31_CTD

Q Data type: BOOL.

This parameter returns TRUE if parameter CV is 0.

CV Data type: WORD.

Actual counter value.

Function block call in ST

```
HACS31CTD (
CD              := hactd_cd,
LOAD            := hactd_load,
PV              := hactd_pv);
hactd_q         := HACS31CTD.Q;
hactd_cv        := HACS31CTD.CV;
```

HA_CS31_CTU - HA count up counter

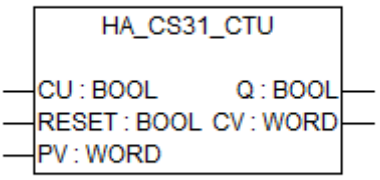


Fig. 95: Function block HA_CS31_CTU

Table 110: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a standard count up counter with automatic data synchronization in a high availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

Parameter CV (**C**ounter **V**ariable) will be initialized with 0 if RESET=TRUE. If parameter CU (**C**ount **U**p) has a rising edge from FALSE to TRUE, parameter CV will be raised by 1. Parameter Q will return TRUE when CV is greater or equal to the upper limit PV (**P**reset **V**alue).

Input description

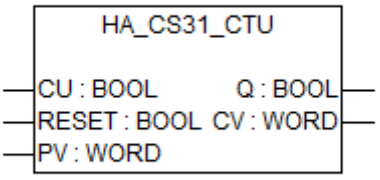


Fig. 96: Function block HA_CS31_CTU

CU Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Count up with rising edge.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Loads value '0' into CV with a rising edge at RESET.

PV Data type: WORD, default value: 0, range: 0 - 65535.
Preset value.

Output description

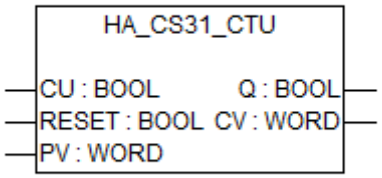


Fig. 97: Function block HA_CS31_CTU

Q Data type: BOOL, range: TRUE/FALSE.
Parameter Q returns TRUE when parameter CV (counter value) is greater than or equal to parameter PV (preset value).

CV Data type: WORD.
Actual counter value.

Function block call in ST

```
HACS31CTU (
CU              := hactu_cu,
RESET           := hactu_reset,
PV              := hactu_pv);
hactu_q         := HACS31CTU.Q;
hactu_cv        := HACS31CTU.CV;
```

HA_CS31_CTUD - HA up/down counter

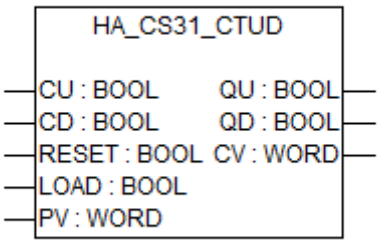


Fig. 98: Function block HA_CS31_CTUD

Table 111: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is standard bidirectional counter with automatic data synchronization in a high availability application (high availability bidirectional counter).

> *Only internal variables and outputs are synchronized. Input variables or parameters are not synchronized. If necessary, use HA_CS31_DATA_SYNC function block to synchronize the input variables.*

If the CU (Count Up) parameter has a rising edge from FALSE to TRUE, parameter CV (Counter Variable) will be increased by 1. If parameter CD (Count Down) has a rising edge from FALSE to TRUE, parameter CV will be decreased by 1, provided that this does not cause the value to fall below 0.

If RESET=TRUE, CV will be initialized with 0. If LOAD=TRUE, parameter CV will be initialized with PV (Preset Value). QU returns TRUE when CV has become greater or equal PV. QD returns TRUE when CV has become equal 0.

Input description

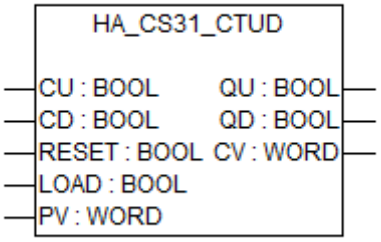


Fig. 99: Function block HA_CS31_CTUD

CU Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Count up with rising edge.

CD Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Count down with rising edge.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Loads value 0 into parameter CV with a rising edge at RESET.

LOAD Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Loads parameter PV into parameter CV with rising edge at LOAD.

PV Data type: WORD, default value: 0, range: 0-65535.
Preset value.

Output description

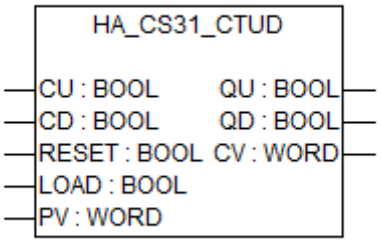


Fig. 100: Function block HA_CS31_CTUD

QU Data type: BOOL
QU is TRUE when parameter CV is greater than or equal to parameter PV.

QD Data type: BOOL
QD is TRUE when parameter CV is 0.

CV Data type: WORD, default value: , range:
Actual counter value.

Function call in ST

```
HACS31CTUD(
CU          := hactud_cu,
CD          := hactud_cd,
RESET       := hactud_reset,
LOAD        := hactud_load,
PV          := hactud_pv);
hactud_qu   := HACS31CTUD.QU;
hactud_qd   := HACS31CTUD.QD;
hactud_cv   := HACS31CTUD.CV;
```

HA_CS31_INTEGRAL - HA integral function block

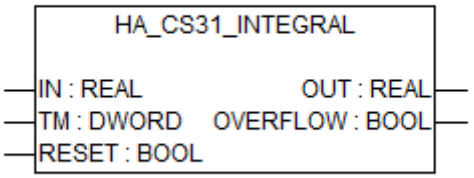


Fig. 101: Function block HA_CS31_INTEGRAL

Table 112: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block approximately determines the integral of the function with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

In an analogous fashion to DERIVATIVE, the function value is delivered as a REAL variable by using IN. TM contains the time which has passed in a DWORD (in ms). The input of RESET of the bool type allows the function block to start anew with value TRUE. The integral is approximated by two step functions. The average of these is delivered as the approximated integral.

Input description

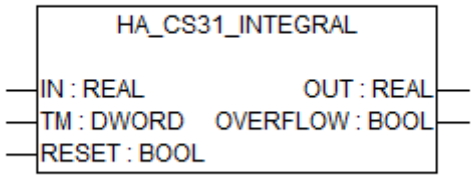


Fig. 102: Function block HA_CS31_INTEGRAL

IN Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Input function value.

TM Data type: REAL, default value: 0, range: 0 to 4294967295.
Time in ms.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Rising edge starts with a new value.

Output description

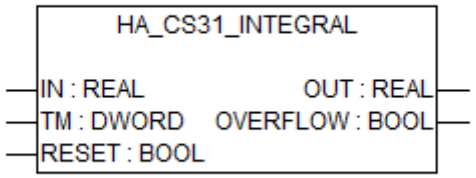


Fig. 103: Function block HA_CS31_INTEGRAL

OUT Data type: REAL.
Integral output.

OVERFLOW Data type: BOOL.
Overflow in the integral part.

Function block call in ST

```
HACS31INTEGRAL(
IN                      := haintegral_in,
TM                      := haintegral_tm,
RESET                   := haintegral_reset);
haintegral_out          := HACS31INTEGRAL.OUT;
haintegral_overflow     := HACS31INTEGRAL.OVERFLOW;
```

HA_CS31_PID - HA PID controller

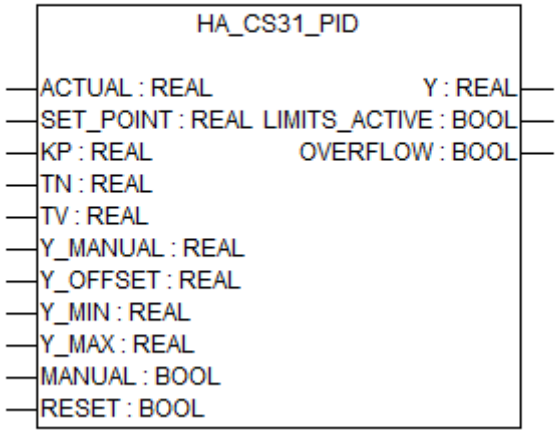


Fig. 104: Function block HA_CS31_PID

Table 113: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a standard PID controller with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

Parameter description

- The parameters Y_OFFSET, Y_MIN and Y_MAX serve for transformation of the manipulated variable within a prescribed range.
- MANUAL parameter can be used to switch to manual operation.
- RESET parameter can be used to re-initialize the controller.
- In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error as difference from SET_POINT parameter – ACTUAL parameter, generates the derivation with respect to time de/dt and stores these values internally.
- The output, i.e. the manipulated variable Y unlike the PD controller contains an additional integral part and is calculated as follows: $Y = KP \times (D + 1/TN \int edt + TV\ dD/dt) + Y\_OFFSET$. So besides the P-part also the current change of the controller error (D-part) and the history of the controller error (I-part) influence the manipulated variable. The PID controller can be easily converted to a PI controller by setting TV=0. Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller if the integral of error D becomes too great. Therefore for the sake of security a BOOLEAN output called OVERFLOW is present, which in this case would have the value TRUE. This only will happen if the control system is instable due to incorrect parameterization. At the same time, the controller will be suspended and will only be activated again by re-initialization.

Input description

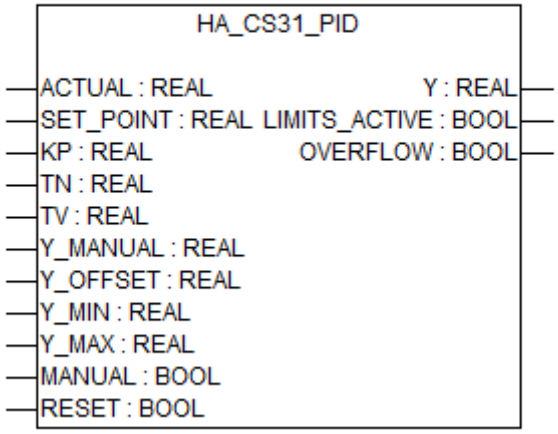


Fig. 105: Function block HA_CS31_PID

ACTUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Current value of the controlled variable.

SET_POINT Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Desired value, command variable.

KP Data type: REAL, default value: 0, range; 0 to 3.402823466e+38.
Coefficient of proportionality, unity gain of the P-part.

TN Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.
Reset time, reciprocal unity gain of the I-part; given in seconds, e.g. "0.5" for 500 ms.

TV Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.
Derivative action time, unity gain of the D-part in seconds, e.g. "0.5" for 500 ms.

Y_MANUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Defines output value Y in case of MANUAL=TRUE.

Y_OFFSET Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Offset for the manipulated variable Y.

Y_MIN Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Lower limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN < Y_MAX.

Y_MAX Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.

Upper limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN < Y_MAX.

MANUAL Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

If TRUE, manual operation will be active, i.e. the manipulated value will be defined by Y_MANUAL.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

TRUE resets the controller; during re-initialization Y = Y_OFFSET.

Output description

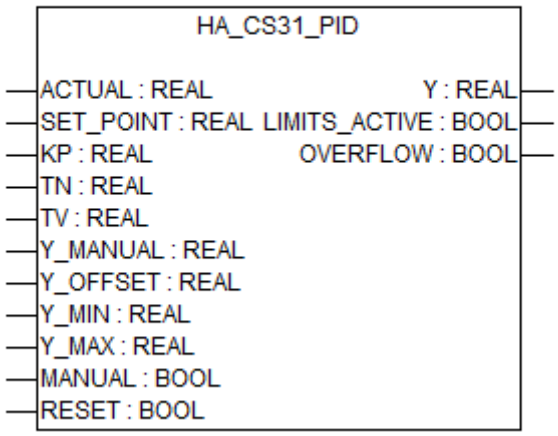


Fig. 106: Function block HA_CS31_PID

Y Data type: REAL.

Manipulated value, calculated by the function block.

LIMITS_ACTIVE Data type: BOOL.

TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).

OVERFLOW Data type: BOOL.

TRUE indicates an overflow in the integral part.

Function block call in ST

```
HACS31PID(
ACTUAL                  := hapid_actual,
SET_POINT               := hapid_set_point,
KP                      := hapid_kp,
TN                      := hapid_tn,
```

```
TV                      := hapid_tv,
Y_MANUAL                := hapid_y_manual,
Y_OFFSET                := hapid_y_offset,
Y_MIN                   := hapid_y_min,
Y_MAX                   := hapid_y_max,
MANUAL                  := hapid_manual,
RESET                   := hapid_reset);
hapid_y                 := HACS31PID.A;
hapid_limits_active     := HACS31PID.LIMITS_ACTIVE;
hapid_overflow          := HACS31PID.OVERFLOW;
```

HA_CS31_PID_DV500 - HA PID controller for DigiVis

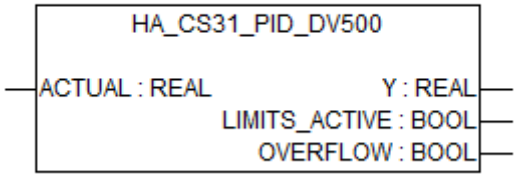


Fig. 107: Function block HA_CS31_PID_DV500

Table 114: General Information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4) and above
Type	Function block with historical values.

This function block is a standard PID controller with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

In terms of functionality this function block is similar to HA_CS31_PID, though, users can only access the ACTUAL parameter. Users get the complete configuration faceplate on the DigiVis, which is identical to the faceplate of HA_CS31_PID.

Parameter description

- The parameters Y_OFFSET, Y_MIN and Y_MAX serve for transformation of the manipulated variable within a prescribed range.
- MANUAL parameter can be used to switch to manual operation.
- RESET parameter can be used to re-initialize the controller.

- In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error as difference from SET_POINT parameter – ACTUAL parameter, generates the derivation with respect to time de/dt and stores these values internally.
- The output, i.e. the manipulated variable Y unlike the PD controller contains an additional integral part and is calculated as follows: Y = KP × (D + 1/TN ∫edt + TV dD/dt) + Y_OFFSET. So besides the P-part also the current change of the controller error (D-part) and the history of the controller error (I-part) influence the manipulated variable. The PID controller can be easily converted to a PI controller by setting TV=0. Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller if the integral of error D becomes too great. Therefore for the sake of security a BOOLEAN output called OVERFLOW is present, which in this case would have the value TRUE. This only will happen if the control system is instable due to incorrect parameterization. At the same time, the controller will be suspended and will only be activated again by re-initialization.

Input description

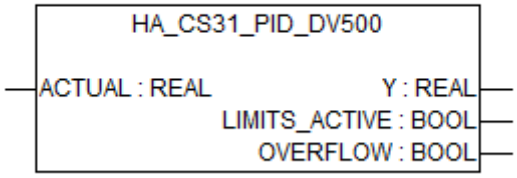


Fig. 108: Function block HA_CS31_PID_DV500

ACTUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Current value of the controlled variable.

Output description

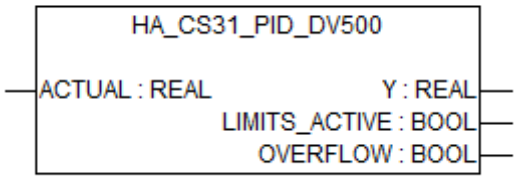


Fig. 109: Function block HA_CS31_PID_DV500

Y Data type: REAL.
Manipulated value, calculated by the function block.

LIMITS_ACTIVE Data type: BOOL.
TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).

OVERFLOW Data type: BOOL.
TRUE indicates an overflow in the integral part.

Function block call in ST

```
HACS31PIDDV500(
ACTUAL                  := haPID_actual,
hapid_y                 := HACS31PID.A;
hapid_limits_active     := HACS31PID.LIMITS_ACTIVE;
hapid_overflow          := HACS31PID.OVERFLOW;
```

HA_CS31_PID_FIXCYCLE - HA PID controller with fix cycle

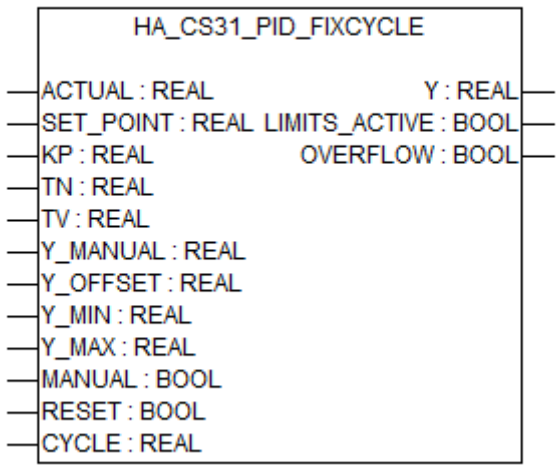


Fig. 110: Function block HA_CS31_PID_FIXCYCLE

Table 115: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a standard PID controller with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

Parameter description

- The parameters Y_OFFSET, Y_MIN and Y_MAX serve for transformation of the manipulated variable within a prescribed range.
- MANUAL parameter can be used to switch to manual operation.
- CYCLE ia an input to to the function block and is to be set by the user (in ms).
- RESET parameter can be used to re-initialize the controller.
  The cycle time is not measured automatically. It is set by the input CYCLE (in seconds).

- In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error as difference from SET_POINT parameter – ACTUAL parameter, generates the derivation with respect to time de/dt and stores these values internally.
- The output, i.e. the manipulated variable Y unlike the PD controller contains an additional integral part and is calculated as follows: Y = KP × (D + 1/TN ∫edt + TV dD/dt) + Y_OFFSET. So besides the P-part also the current change of the controller error (D-part) and the history of the controller error (I-part) influence the manipulated variable. The PID controller can be easily converted to a PI controller by setting TV=0. Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller if the integral of error D becomes too great. Therefore for the sake of security a boolean output called OVERFLOW is present, which in this case would have the value TRUE. This only will happen if the control system is instable due to incorrect parameterization. At the same time, the controller will be suspended and will only be activated again by re-initialization.

Input description

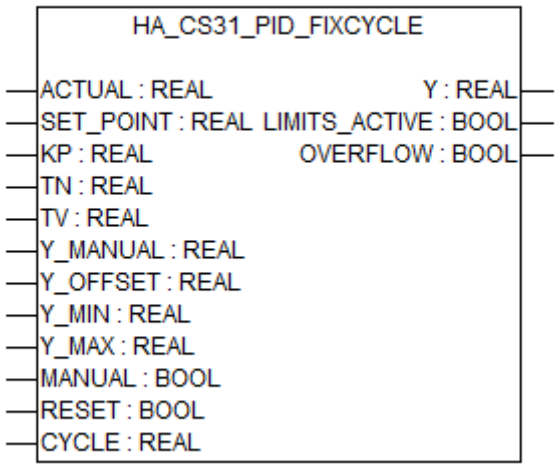


Fig. 111: Function block HA_CS31_PID_FIXCYCLE

ACTUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Current value of the controlled variable.

SET_POINT Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Desired value, command variable.

KP Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.
Coefficient of proportionality, unity gain of the P-part.

TN Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.
Reset time, reciprocal unity gain of the I-part; given in seconds, e.g. "0.5" for 500 ms.

TV Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.
Derivative action time, unity gain of the D-part in seconds, e.g. "0.5" for 500 ms.

Y_MANUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.

Defines output value Y in case of MANUAL = TRUE.

Y_OFFSET Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.

Offset for the manipulated variable Y.

Y_MIN Data type: REAL, default value: 0.

Lower limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN < Y_MAX.

Y_MAX Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.

Upper limit for the manipulated variable Y. If Y exceeds these limits, output LIMITS_ACTIVE will be set to TRUE and Y will be kept within the prescribed range. This control will only work if Y_MIN < Y_MAX.

MANUAL Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

If TRUE, manual operation will be active, i.e. the manipulated value will be defined by Y_MANUAL.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.

TRUE resets the controller; during re-initialization Y = Y_OFFSET.

CYCLE Data type: REAL, default value: 0, range: 0 to 3.402823466e+38.

Number of cycles.

Output description

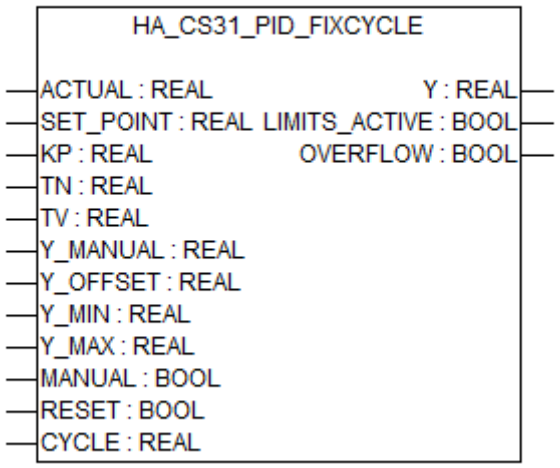


Fig. 112: Function block HA_CS31_PID_FIXCYCLE

Y Data type: REAL

Manipulated value, calculated by the function block.

LIMITS_ACTIVE Data type: BOOL

TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).

OVERFLOW Data type: BOOL

TRUE indicates an overflow in the integral part.

Function block call in ST

```
HACS31PIDFIXCYCLE(
ACTUAL          := hapid_actual,
SET_POINT       := hapid_set_point,
KP              := hapid_kp,
TN              := hapid_tn,
TV              := hapid_tv,
Y_MANUAL        := hapid_y_manual,
Y_OFFSET        := hapid_y_offset,
Y_MIN           := hapid_y_min,
Y_MAX           := hapid_y_max,
MANUAL          := hapid_manual,
RESET           := hapid_reset);
```

HA_CS31_PID_FIXCYCLE_DV500 - HA PID controller for DigiVis

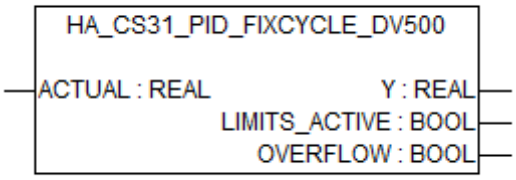


Fig. 113: Function block HA_CS31_PID_FIXCYCLE_DV500

Table 116: General Information

Available as of runtime system	V2.3 and above
Included in library	HA_CS31_AC500_V23.lib (V2.4) and above
Type	Function block with historical values.

This function block is a standard PID controller with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

In terms of functionality this function block is similar to HA_CS31_PID_FIXCYCLE, though, users can only access the ACTUAL parameter. Users get the complete configuration faceplate on the DigiVis, which is identical to the faceplate of HA_CS31_PID.

Parameter description

- The parameters Y_OFFSET, Y_MIN and Y_MAX serve for transformation of the manipulated variable within a prescribed range.
- MANUAL parameter can be used to switch to manual operation.
- RESET parameter can be used to re-initialize the controller.
- In normal operation (MANUAL = RESET = LIMITS_ACTIVE = FALSE) the controller calculates the controller error as difference from SET_POINT parameter – ACTUAL parameter, generates the derivation with respect to time de/dt and stores these values internally.
- The output, i.e. the manipulated variable Y unlike the PD controller contains an additional integral part and is calculated as follows: Y = KP × (D + 1/TN ∫edt + TV dD/dt) + Y_OFFSET. So besides the P-part also the current change of the controller error (D-part) and the history of the controller error (I-part) influence the manipulated variable. The PID controller can be easily converted to a PI controller by setting TV=0. Because of the additional integral part, an overflow can come about by incorrect parameterization of the controller if the integral of error D becomes too great. Therefore for the sake of security a BOOLEAN output called OVERFLOW is present, which in this case would have the value TRUE. This only will happen if the control system is instable due to incorrect parameterization. At the same time, the controller will be suspended and will only be activated again by re-initialization.

Input description

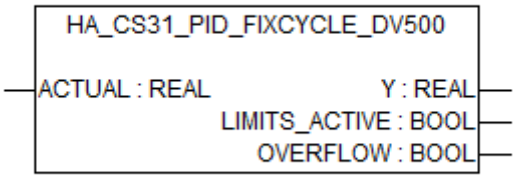


Fig. 114: Function block HA_CS31_PID_FIXCYCLE_DV500

ACTUAL Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.

Current value of the controlled variable.

Output description

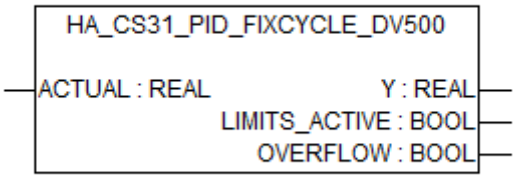


Fig. 115: Function block HA_CS31_PID_FIXCYCLE_DV500

Y Data type: REAL

Manipulated value, calculated by the function block.

LIMITS_ACTIVE Data type: BOOL

TRUE indicates that Y has exceeded the given limits (Y_MIN, Y_MAX).

OVERFLOW Data type: BOOL

TRUE indicates an overflow.

Function block call in ST

```
HACS31PIDFIXCYCLEDV500(
ACTUAL                      := hapid_actual,
hapid_y                     := HACS31PID.A;
hapid_limits_active         := HACS31PID.LIMITS_ACTIVE;
hapid_overflow              := HACS31PID.OVERFLOW;
```

HA_CS31_RAMP_INT - HA ramp with integer

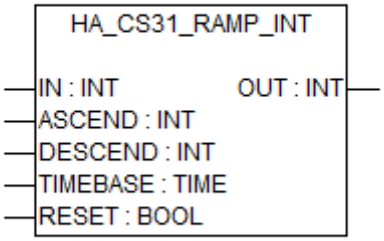


Fig. 116: Function block HA_CS31_RAMP_INT

Table 117: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is an integer ramp generator with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

This function block serves to limit ascending or descending of a function:

The input consists of three INT values: IN (function input), and ASCEND and DESCEND, the maximum increase or decrease for a given time interval, which is defined by the TIMEBASE parameter. Setting RESET parameter to TRUE causes the function block to be initialized anew.

The output OUT of the type INT contains ascends and descends limited function value. If the TIMEBASE parameter is set to t#0s, ASCEND and DESCEND are not related to the time interval, but remain the same.

Input description

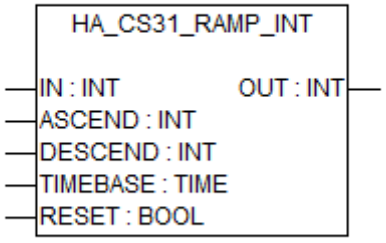


Fig. 117: Function block HA_CS31_RAMP_INT

IN Data type: INT, default value: 0, range: -32767 to 32767.
Function input.

ASCEND Data type: INT, default value: 0, range: -32767 to 32767.
Max. ascension per time interval.

DESCEND Data type: INT, default value: 0, range: -32767 to 32767.
Max. descend per time interval.

TIMEBASE Data type: TIME, default value: 0, range: 0 to 4194967295 ms.
Time interval for the maximum increase/decrease.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Reset input to reinitialize the function block.

Output description

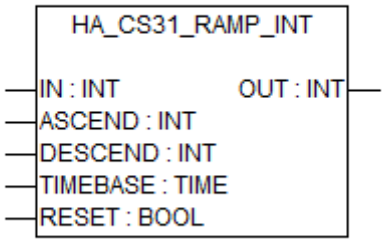


Fig. 118: Function block HA_CS31_RAMP_INT

OUT Data type: INT.

Output with ascended or descended function value.

Function block call in ST

```
HACS31RAMPINT(
IN                  := harampint_in,
ASCEND              := harampint_ascend,
DESCEND             := harampint_descend,
TIMEBASE            := harampint_timebase,
RESET               := harampint_reset);
harampint_out       := HACS31RAMPINT.OUT;
```

HA_CS31_RAMP_REAL - HA Ramp with Real

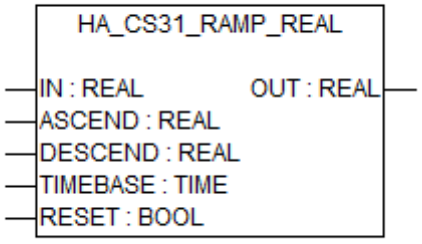


Fig. 119: Function block HA_CS31_RAMP_REAL

Table 118: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a real ramp generator with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

This function block serves to limit ascending or descending of a function:

The input consists of three REAL values: IN (function input), and ASCEND and DESCEND, the maximum increase or decrease for a given time interval, which is defined by the TIMEBASE parameter. Setting RESET parameter to TRUE causes the function block to be initialized anew.

The output OUT of the type REAL contains ascends and descends limited function value. When the TIMEBASE parameter is set to t#0s, ASCEND and DESCEND are not related to the time interval, but remain the same.

Input description

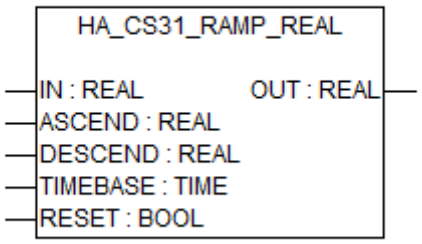


Fig. 120: Function block HA_CS31_RAMP_REAL

IN Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Function input.

ASCEND Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Max. ascension per time interval.

DESCEND Data type: REAL, default value: 0, range: 1.175494351e-38 to 3.402823466e+38.
Max. descend per time interval.

TIMEBASE Data type: TIME, default value: 0, range: 0 to 4194967295 ms.
Time interval for the maximum increase/decrease.

RESET Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
Reset input to re-initialize the function block.

Output description

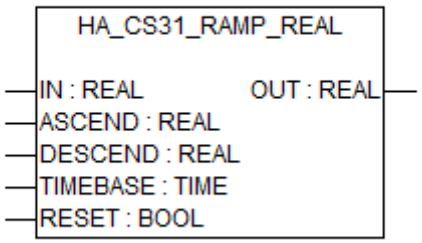


Fig. 121: Function block HA_CS31_RAMP_REAL

OUT Data type: REAL.

Output with ascended or descended function value.

Function block call in ST

```
HACS31RAMPREAL(
IN                  := harampreal_in,
SCEND               := harampreal_ascend,
DESCEND             := harampreal_descend,
TIMEBASE            := harampreal_timebase,
RESET               := harampreal_reset);
harampreal_out      := HACS31RAMPREAL.OUT;
```

HA_CS31_TOF - HA Turn-Off delay timer

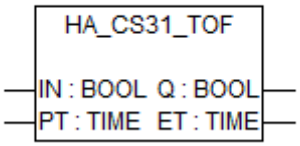


Fig. 122: Function block HA_CS31_TOF

Table 119: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is standard turn-off delay with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

This function block implements a turn-off delay timer. As soon as the IN parameter becomes FALSE, time counting in ET will start (unit: ms). Counting will proceed until the value in ET is equal to PT value. Then, it will remain constant. Q is FALSE if IN is FALSE and ET is equal to PT. Otherwise it is TRUE. Thus, the Q parameter has a falling edge as soon as time indicated in PT has run out (unit: ms).

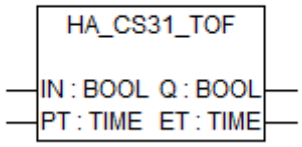


Fig. 123: A_CS31_TOF behavior over time

Input description

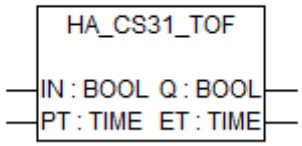


Fig. 124: Function block HA_CS31_TOF

IN Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
If IN=FALSE, the turn-off delay starts.

PT Data type: TIME, default value: 0, range: 0 to 4194967295 ms.
Preset time for turn-off delay.

Output description

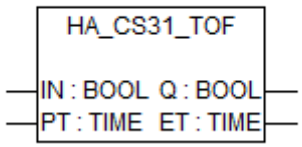


Fig. 125: Function block HA_CS31_TOF

Q Data type: BOOL, range: TRUE/FALSE.
Q=FALSE if IN=FALSE and ET=PT.

ET Data type: TIME.
Actual time of turn-off delay.

Function block call in ST

```
HACS31TOF(
IN              := hatof_in,
PT              := hatof_pt);
hatof_q         := HACS31TOF.Q;
hatof_et        := HACS31TOF.ET;
```

HA_CS31_TON - HA turn-on delay timer

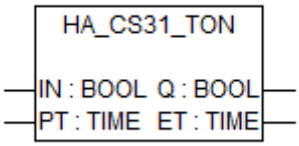


Fig. 126: Function block HA_CS31_TON

Table 120: General information

Available as of runtime system	V1.3 and above
Included in library	HA_CS31_AC500_V13.lib (V1.3) and above
Type	Function block with historical values.

This function block is a standard turn-on delay with automatic data synchronization in a High Availability application.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_CS31_DATA_SYNC to achieve the same.*

This function block implements a turn-on delay timer. As soon as the IN parameter becomes TRUE, time counting in ET will start (unit: ms). Counting will proceed until the value in ET is equal to PT value. Then, it will remain constant. Q is TRUE when IN is TRUE and ET is equal to PT. Otherwise it is FALSE. Thus, the Q parameter has a rising edge as soon as time indicated in PT has run out (unit: ms).

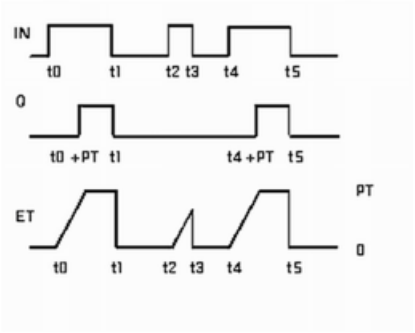


Fig. 127: HA_CS31_TON behavior over time.

Input description

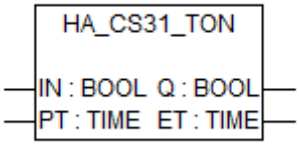


Fig. 128: Function block HA_CS31_TON

IN Data type: BOOL, default value: FALSE, range: TRUE/FALSE.
If IN=TRUE, the turn-on delay starts.

PT Data type: TIME, default value: 0, range: 0 to 4194967295 ms.
Preset time for turn-on delay.

Output description

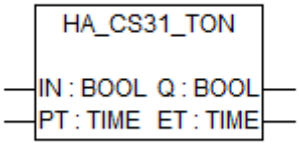


Fig. 129: Function block HA_CS31_TON

Q Data type: BOOL.
Q=TRUE if IN=TRUE and ET=PT.

ET Data type: TIME.
Actual time of turn-on delay.

Function block call in ST

```
HACS31TON(
IN              := haton_in,
PT              := haton_pt);
haton_q         := HACS31TON.Q;
haton_et        := HACS31TON.ET;
```

Visualizations

In the application program, the user can add the visualization object in his project. The following figure describes how to add the faceplate of the required function block to the visualization object:

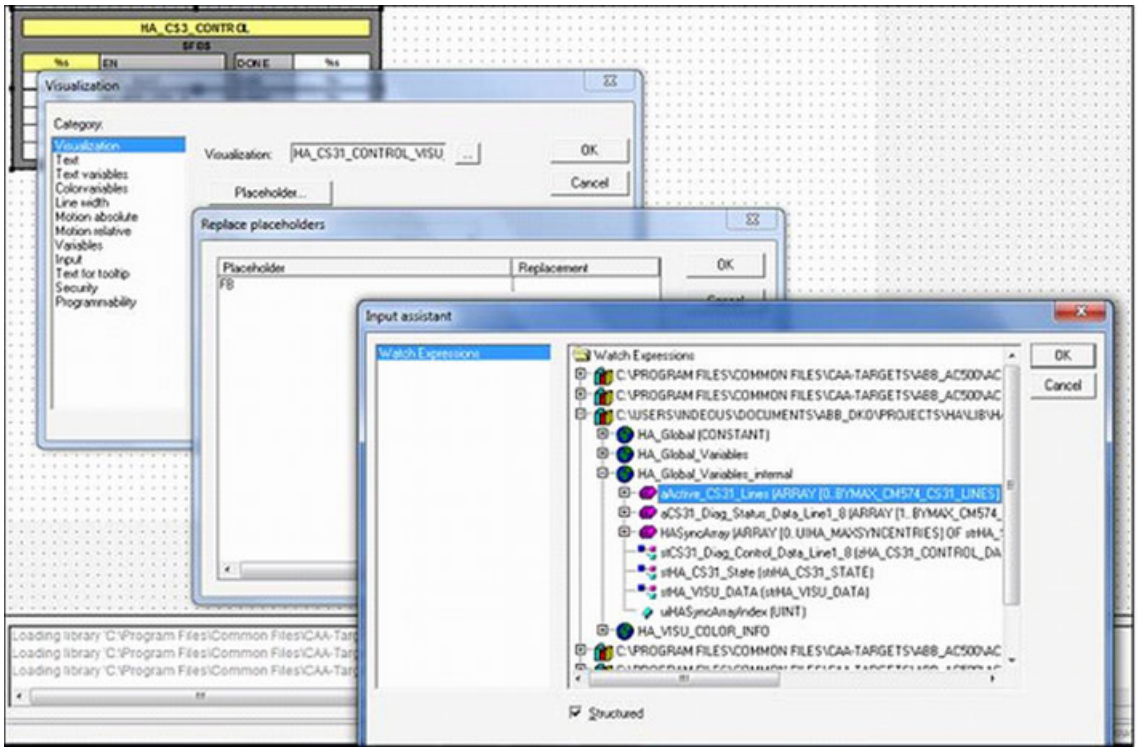


Fig. 130: HA_CS31_Visualization

In the following chapter, overall visualization and visualization for each individual function block are discussed in detail.

HA_CS31_OVERVIEW_VISU - Visualization

This overview visualization for a AC500 High Availability CS31 system provides information for easy diagnosis in case of an error. It contains information about Ethernet cable status, PLC primary and secondary status, status for each CS31 line and diagnosis information, the HA _CS31_CONTROL status etc.

The following figure describes the visualization in offline and online mode:

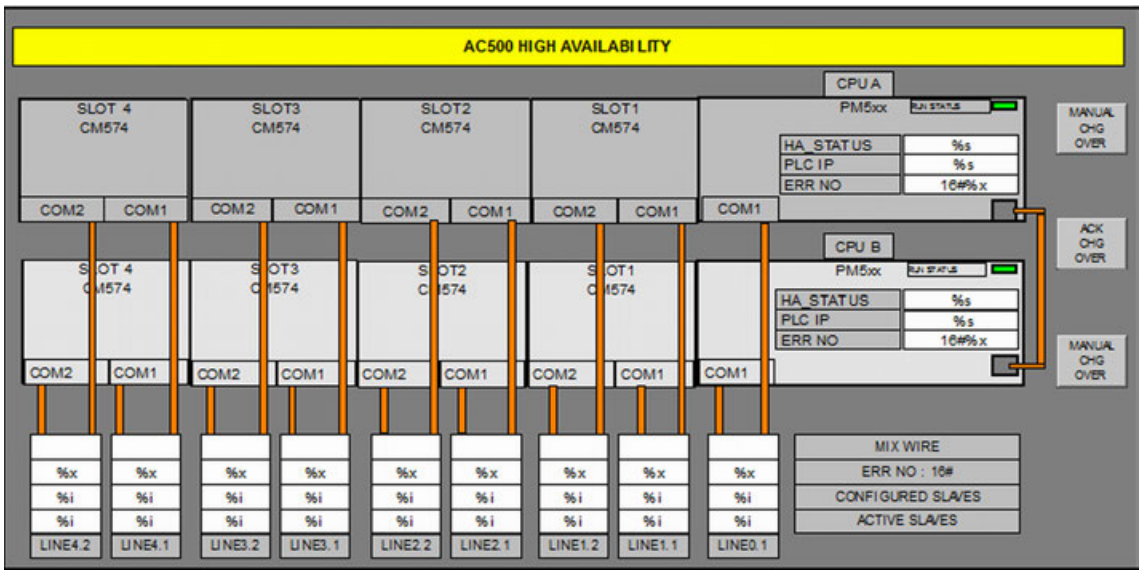


Fig. 131: HA_CS31_Visualization_offline

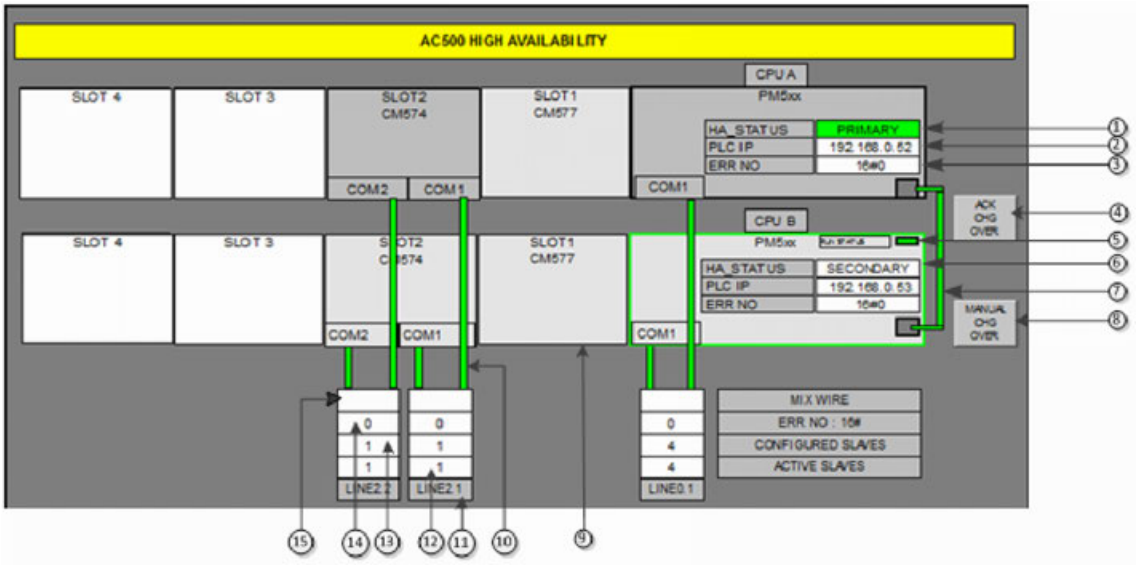


Fig. 132: HA_CS31_Visualization_online

No.	Description
1	Shows the PLC status as PRIMARY or SECONDARY.
2	Shows PLC IP address.
3	Shows HA_CS31_CONTROL error code.
4	Acknowledge button for change over.
5	Shows PLC RUN status (green: running, red: stopped).
6	A green frame surrounding a PLC indicates the PLC that is connected to the programming software.
7	Shows UDP data exchange Ethernet status (green: no error, red: error).
8	Changeover button (only available in primary PLC).
9	Shows CM597-ETH if configured.
10	Shows CS31 line status (green: active, red: not active).
11	Shows CS31 line number.
12	Shows active salves in that particular CS31 line.
13	Shows configured slaves in that particular CS31 line.
14	Shows errors related to that particular CS31 line.
15	A red box indicates mix wire errors (white box = no error).

HA-CS31 visualization can be used only for monitoring purpose. Users can only view the logged PLC information using visualization. The only possible action is a manual changeover and the acknowledgment. Users can use DigiVis and OPC server to read out data from or to write data to the HA-CS31 system. Writing data through OPC server will affect both PLCs. Reading out data is done from primary PLC.

HA_CS31_CONTROL_VISU_PH - Visualization

Visualization element HA_CS31_CONTROL_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_CS31_CONTROL function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figures demonstrate visualization in the offline and online mode.

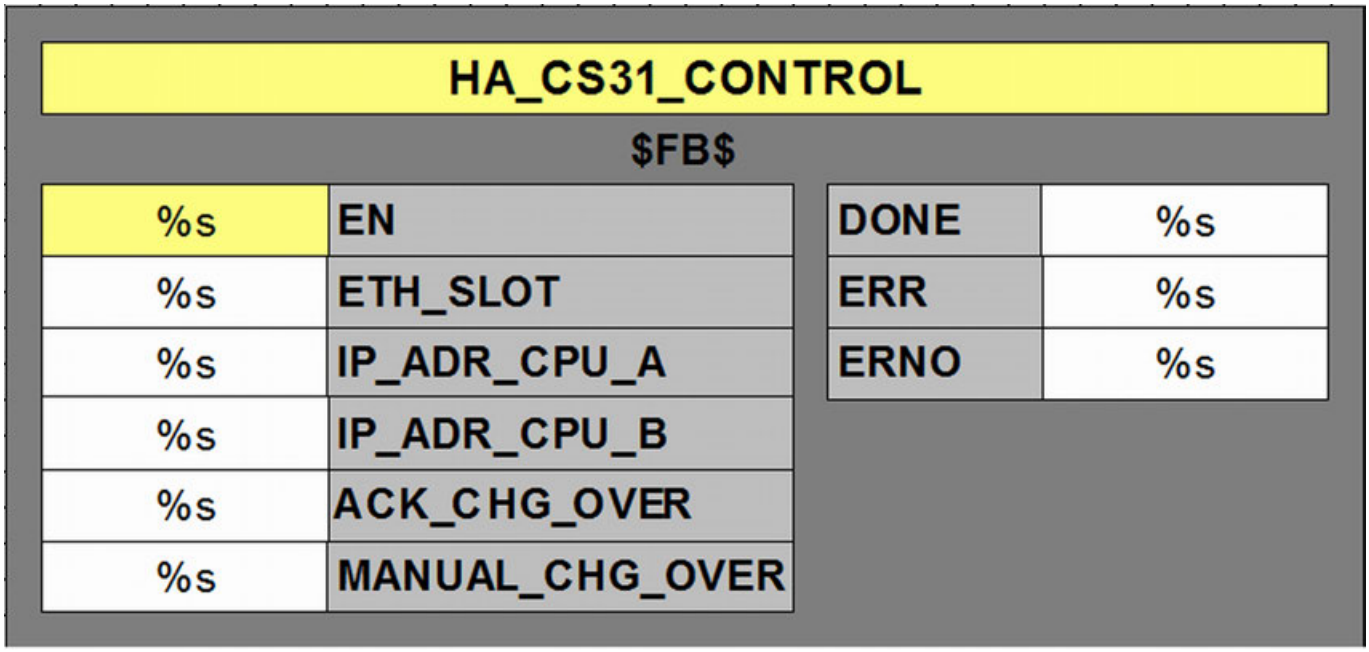


Fig. 133: Faceplate HA_CS31_CONTROL_VISU_PH offline mode

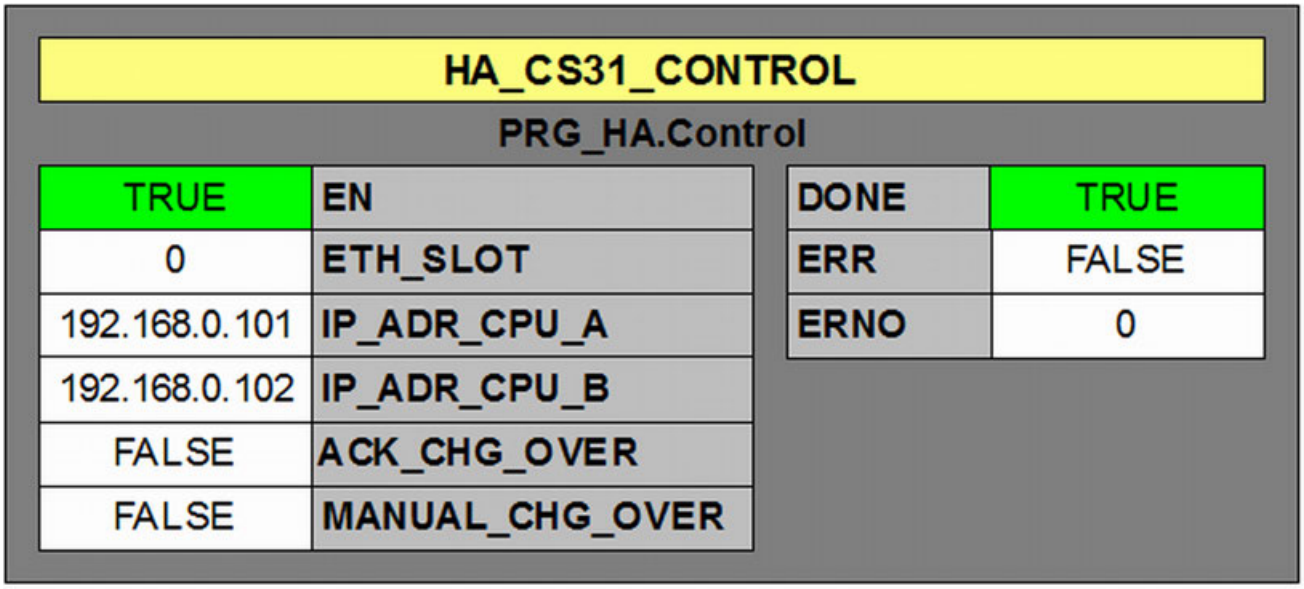


Fig. 134: Faceplate HA_CS31_CONTROL_VISU_PH online mode

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable	Access	Way of Access	Description *)
EN	R		To enable the function block with value TRUE.
ETH_SLOT	R		Slot number for UDP data exchange.
IP_ADR_CPU_A	R		IP address of CPU A.
IP_ADR_CPU_B	R		IP address of CPU B.
ACK_CHG_OVER	R/W	Toggle	Acknowledgement for switch over.
MANUAL_CHG_OVER	R/W	Toggle	Manual change over
DONE	R		Execution finished when output DONE=TRUE.
ERR	R		Error occurred during execution when output ERR=TRUE.
ERNO	R		Error code.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_DATA_SYNC function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The color of the background can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. The color of the title can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

All inputs of the HA_CS31_ DIAG_EXTD_VIA_CM574-RS function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The color of the background can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. The color of the title can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

HA_CS31_DATA_SYNC_VISU_PH - Visualization

Description

Visualization element HA_CS31_DATA_SYNC_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_CS31_DATA_SYNC function block. Visualization can also be used to control the function block by those inputs which are not connected inside the program.

The following figures demonstrate visualization in the offline and online mode:

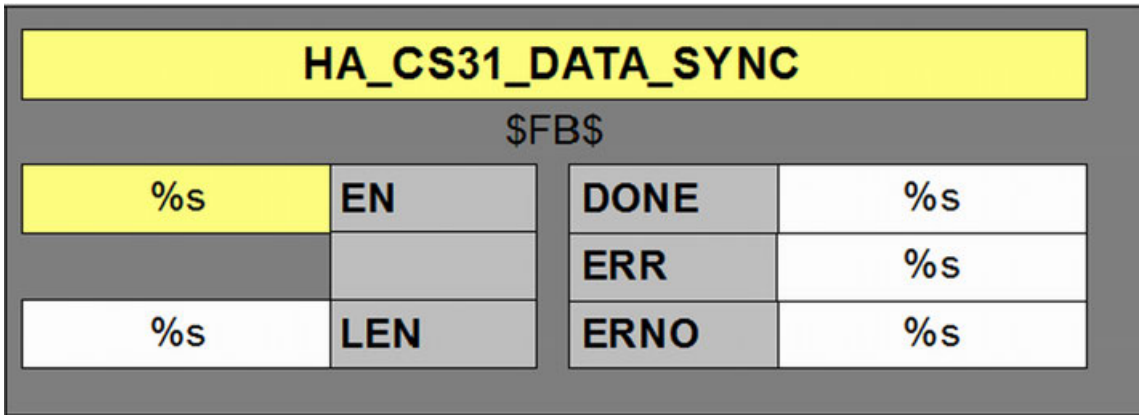


Fig. 135: HA_CS31_Visualization_SYNC_offline

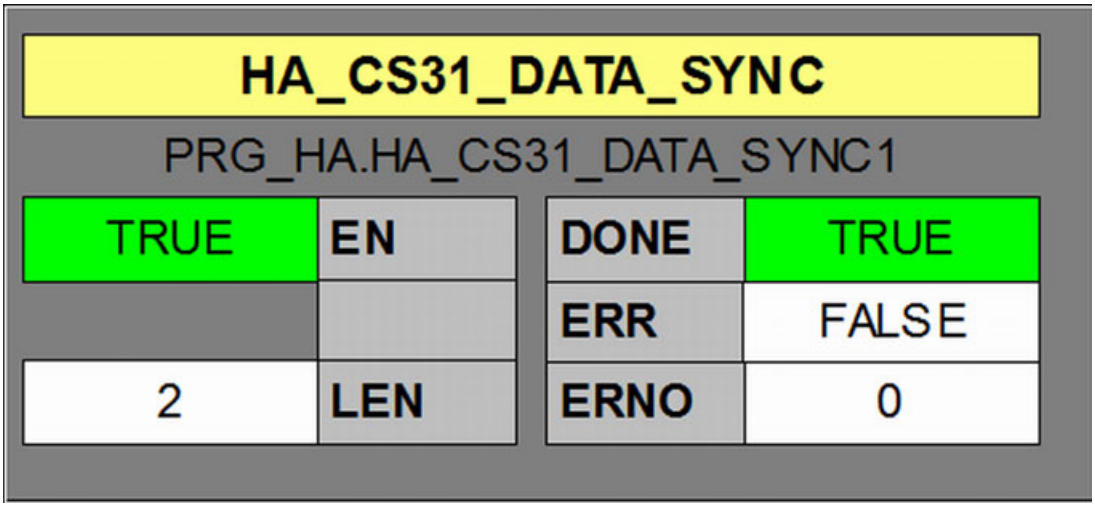


Fig. 136: HA_CS31_Visualization_SYNC_online

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable	Access	Description *)
EN	R	To enable the function block with value TRUE.
LEN	R	Function block instance input (via SIZEOF-operator) for size.
DONE	R	Execution finished when output DONE=TRUE.
ERR	R	Error occurred during execution when output ERR=TRUE.
ERNO	R	Error code.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_DATA_SYNC function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. Background color can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. Title color can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

HA_CS31_DIAG_EXTD_VIA_CM574_VISU_PH - Visualization

Description

Visualization element HA_CS31_DIAG_EXTD_VIA_CM574_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_CS31_DIAG_EXTD_VIA_CM574 function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figure describes visualization in the offline and online mode:

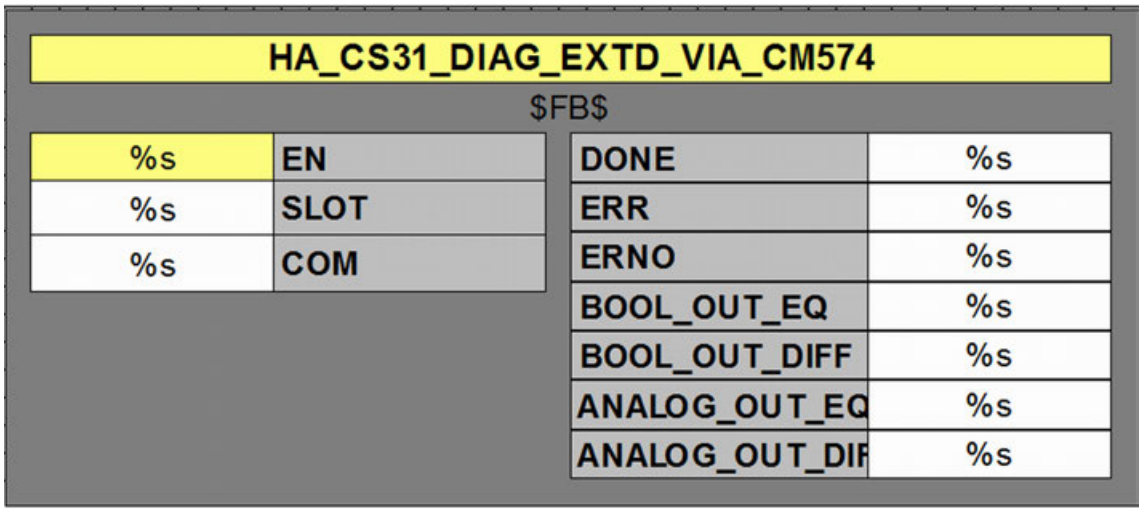


Fig. 137: HA_CS31_DIAG_EXTD_VIA_CM574_offline

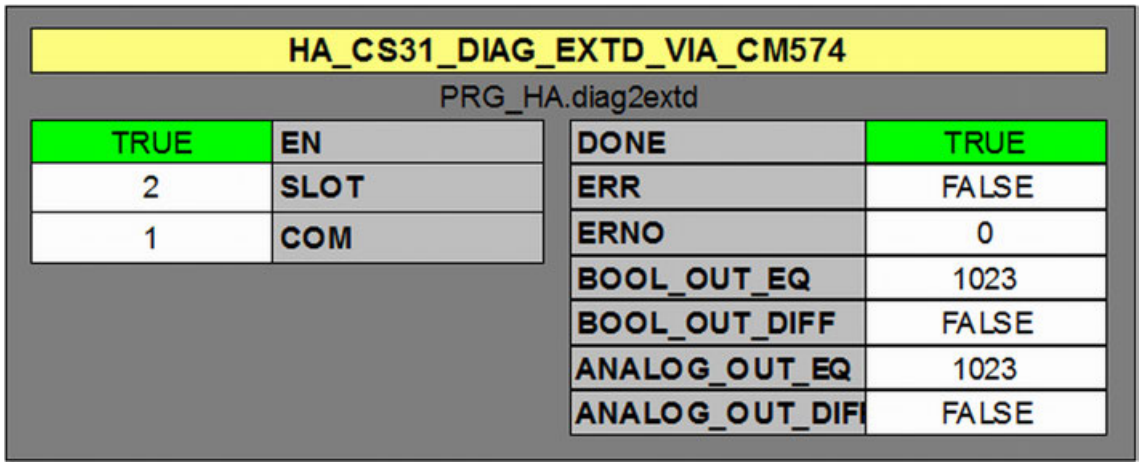


Fig. 138: HA_CS31_DIAG_EXTD_VIA_CM574_online

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization Parameters

Variable	Access	Description *)
EN	R	To enable the function block with value TRUE.
SLOT	R	Slot number of the CS31 master.
COM	R	COM port number of the CS31 master.
DONE	R	Execution finished when output DONE=TRUE.
ERR	R	Error occurred during execution when output ERR=TRUE.
ERNO	R	Error code.
BOOL_OUT_EQ	R	Each bit of the DWORD represents identical output buffer for BOOL data.
BOOL_OUT_DIFF	R	Output buffer with different BOOL data.
ANALOG_OUT_EQ	R	Each bit of the DWORD represents identical output buffer for analog data.
ANALOG_OUT_DIFF	R	Output buffer with different analog data.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_ DIAG_EXTD_VIA_CM574 function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. Background color can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. Title color can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

HA_CS31_DIAG_EXTD_VISU_PH - Visualization

Description

Visualization element HA_CS31_DIAG_EXTD_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_CS31_DIAG_EXTD function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figure describes the visualization in the offline and online mode:

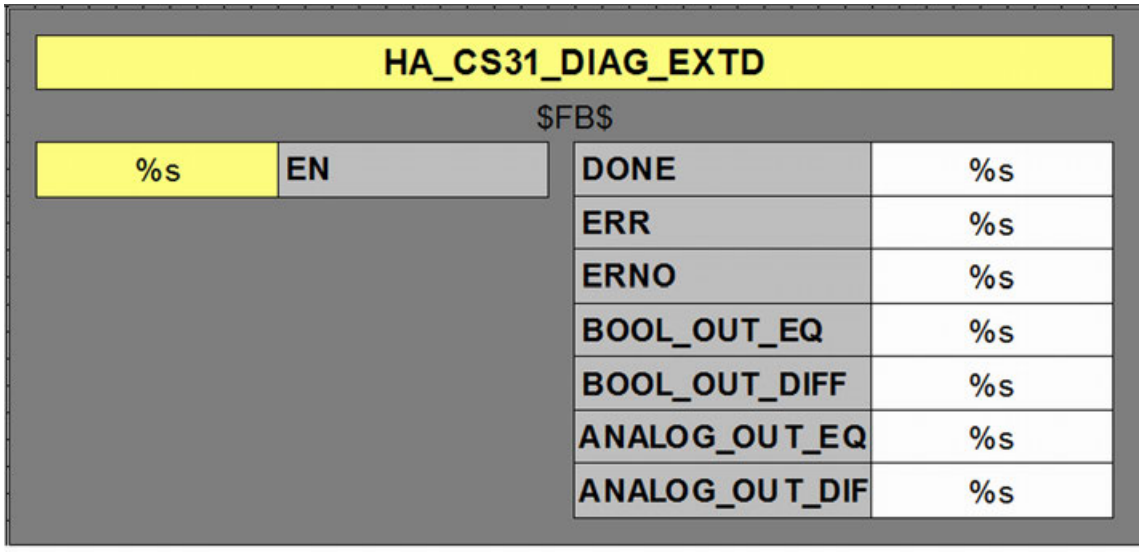


Fig. 139: HA_CS31_DIAG_EXTD_VISU_PH_offline

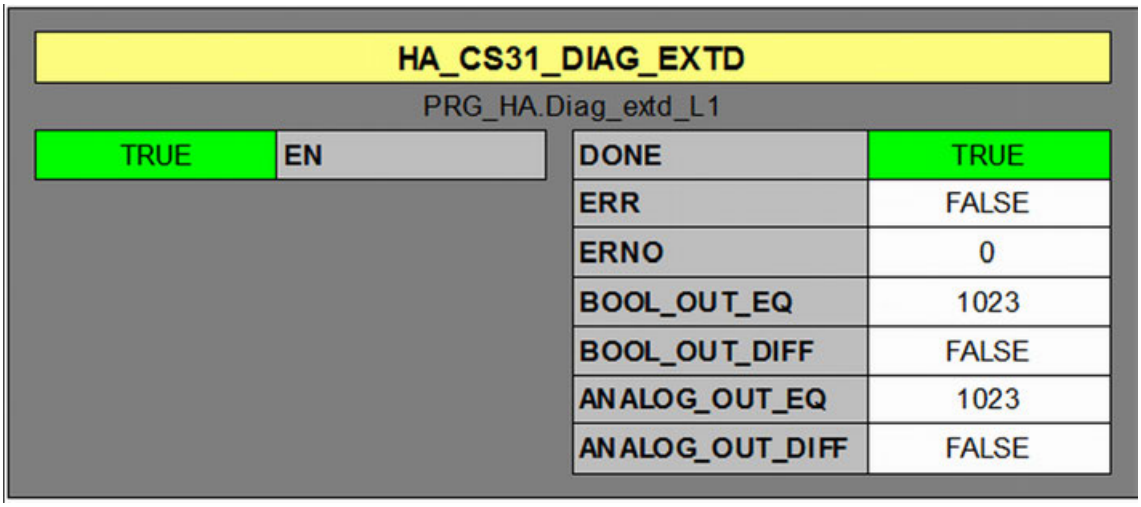


Fig. 140: HA_CS31_DIAG_EXTD_VISU_PH_online

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable	Access	Description *)
EN	R	To enable the function block with value TRUE.
DONE	R	Execution finished when output DONE=TRUE.
ERR	R	Error occurred during execution when output ERR=TRUE.
ERNO	R	Error code.
BOOL_OUT_EQ	R	Each bit of the DWORD represents identical output buffer for BOOL data.
BOOL_OUT_DIFF	R	Output buffer with different BOOL data.
ANALOG_OUT_EQ	R	Each bit of the DWORD represents identical output buffer for analog data.
ANALOG_OUT_DIFF	R	Output buffer with different analog data.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_ DIAG_EXTD function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. Background color can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. Title color can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

HA_CS31_DIAG_VIA_CM574_VISU_PH - Visualization

Description

Visualization element HA_CS31_DIAG_VIA_CM574_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_CS31_DIAG_VIA_CM574 function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figures describe visualization in the offline and online mode:

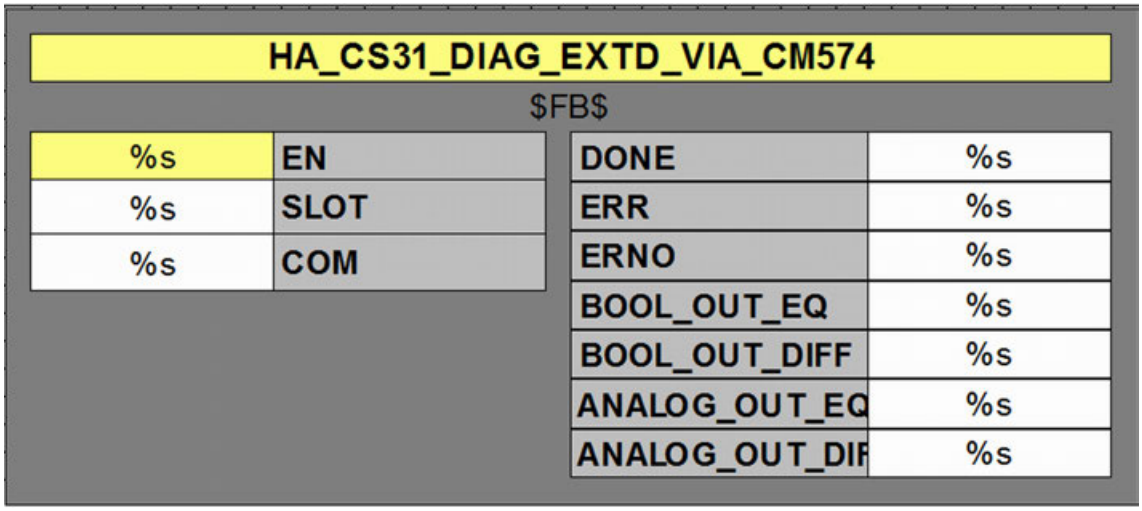


Fig. 141: HA_CS31_DIAG_VIA_CM574_VISU_PH_offline

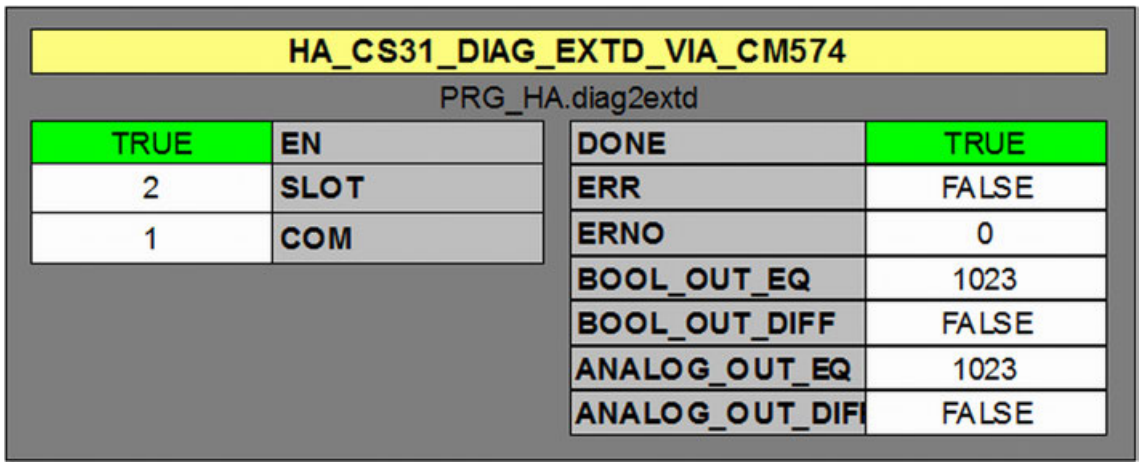


Fig. 142: HA_CS31_DIAG_VIA_CM574-RS_VISU_PH_online

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable	Access	Description *)
EN	R	To enable the function block with value TRUE.
SLOT	R	Slot number of the CS31 master.
COM	R	COM port number of the CS31 master.
DONE	R	Execution finished when output DONE=TRUE.

Variable	Access	Description *)
ERR	R	Error occurred during execution when output ERR=TRUE.
ERNO	R	Error code.
NUM_SLV_CFG	R	Number of configured CI590-CS31-HA slave modules on COM.
NUM_SLV_ACT	R	Number of CI590-CS31-HA slave modules active on CS31 Bus.
ACTIVE_SLV	R	Each bit of the DWORD indicates the active slave modules CI590-CS31-HA.
ERR_MIX_WIRING	R	Error bit indicating mix wiring between Bus 1 and Bus 2.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_ DIAG_VIA_CM574 function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. Background color can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. Title color can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

HA_CS31_DIAG_VISU_PH - Visualization

Description

Visualization element HA_CS31_DIAG_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of HA_CS31_DIAG function block. The visualization can also be used to control the function block by those inputs which are not connected inside the program.

The following figure describes visualization in offline and online mode:

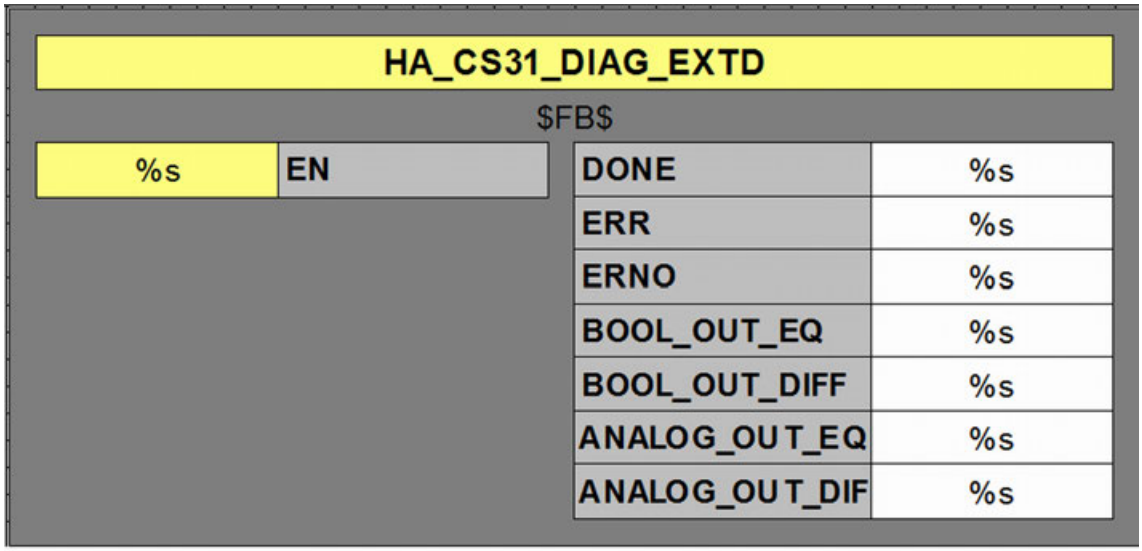


Fig. 143: HA_CS31_DIAG_VISU_PH_offline

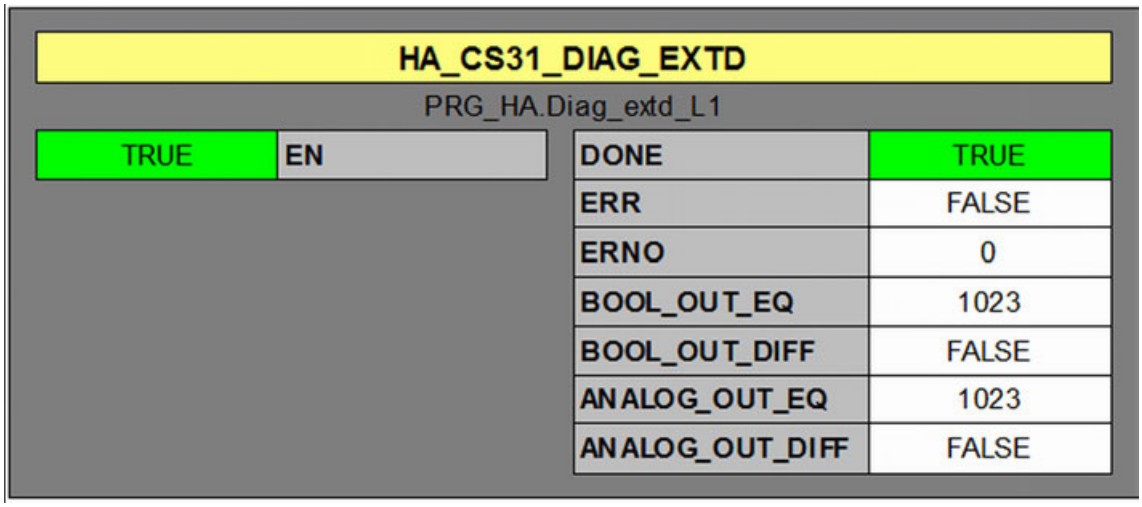


Fig. 144: HA_CS31_DIAG_VISU_PH_online

Colors

The color of the variables has the following meaning:

- White: Actual FALSE and should be FALSE in normal operation.
- Green: Actual TRUE and should be TRUE in normal operation.
- Yellow: Actual FALSE but should be TRUE in normal operation.
- Red: Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable	Access	Description *)
EN	R	To enable the function block with value TRUE.
COM	R	COM port number of CS31 master.
DONE	R	Execution finished when output DONE=TRUE.
ERR	R	Error occurred during execution when output ERR=TRUE.
ERNO	R	Error code.
NUM_SLV_CFG	R	Number of configured CI590-CS31-HA slave modules on COM.
NUM_SLV_ACT	R	Number of CI590-CS31-HA active slave modules on CS31 Bus.
ACTIVE_SLV	R	Each bit of the DWORD indicates the active CI590-CS31-HA slave modules.
ERR_MIX_WIRING	R	Error bit indicating mix wiring between Bus 1 and Bus 2.

*) all elements refer to the function block instance replaced for the placeholder FB.

All inputs of the HA_CS31_ DIAG function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. Background color can be changed by writing a value to the global variable *dwAcsVisuBackgroundColor*. Title color can be changed by writing a value to the global variable *dwAcsVisuTitleColor*.

Structures

stTON_TOFFSyncData	Structure for synchronized TON and TOFF data.
zHA_CS31_PID_DV500_DATA_TYPE	Structure for synchronization data for HA_CS31_PID_DV500.
zHA_CS31_PID_FIX-CYCLE_DV500_DATA_TYPE	Structure for synchronization data for HA_CS31_PID_FIXCYCLE_DV500.

Global variables

Group: HA_Global_Variables	**Description**
fG_HA_PRIMARY	State of the AC500 CPU: FALSE: PM acts as secondary. TRUE: PM acts as primary.
fG_HA_PM1_PRIMARY	Indication of primary PM. FALSE: PM1/ IP1 is not primary. TRUE: PM1/ IP1 acts as primary.
fG_HA_CPU_STOP	Indication of PLC Stop status. FALSE: Indicates the CPU in Run mode. TRUE: Indicates the CPU in Stop mode.
fG_HA_Err	High Availability error state.
wG_HA_ErNo	High Availability error code.
bitG_Data_ERR	High Availability error state for data synchronization.
wG_Data_ERNO	High Availability error code for data synchronization.
dwG_HA_OwnIP	Own IP address on synchronization link connection.
dwG_HA_OtherIP	Other PMs IP address on synchronization link connection.
bG_HA_Slot	Slot of interface to synchronization link connection.
dwG_HA_ServerAlive	Life counter incremented by OPC server.
byLastDataDelay	Variable to store last delay in data exchange.
bitRefreshDataDelay	Bit to refresh data delay.
byCntDataDelay	Data delay counts.
wETH_Life	Ethernet life count.
dwHATimersBaseTime	High Availability base timer.

Group: HA_VISU_COLOR_INFO	**Description**
dwHaVisuBackgroundColor	Visualization elements background color 16#00<B><G><R>.
dwHaVisuTitleColor	Visualization elements title background color 16#00<B><G><R>.

Appendix

Table: Call names of HA function blocks

Used abbreviations:

- FBhv: Function block with historical values
- FBnohv: Function block without historical values
- F: Function

POU Name	Type	Function
HA_CS31_CALLBACK_STOP	F	High Availability CPU STOP event function
HA_CS31_CONTROL	FBhV	High Availability control function block
HA_CS31_CTD	FBhV	High Availability count down counter function block
HA_CS31_CTU	FBhV	High Availability count up counter function block
HA_CS31_CTUD	FBhV	High Availability Up/down counter function block
HA_CS31_DATA_SYNC	FBhV	High Availability data synchronization function block
HA_CS31_DIAG_ON_CM574	FBhV	High Availability diagnosis function block for CM574-RS CS31 line used in CM574-RS
HA_CS31_DIAG	FBhV	High Availability diagnosis function block for CPU COM1 CS31 line
HA_CS31_DIAG_EXTD	FBhV	High Availability extended diagnosis function block for CPU COM1 CS31 line
HA_CS31_DIAG_EXTD_VIA_CM574	FBhV	High Availability extended diagnosis function block for CM574-RS CS31 line used in CPU
HA_CS31_DIAG_VIA_CM574	FBhV	High Availability diagnosis function block for CM574-RS CS31 line used in CPU
HA_CS31_INTEGRAL	FBhV	High Availability integral function block
HA_CS31_PID	FBhV	High Availability PID controller function block
HA_CS31_PID_FIXCYCLE	FBhV	High Availability PID controller with fix cycle function block
HA_CS31_RAMP_INT	FBhV	High Availability ramp function block with integer
HA_CS31_RAMP_REAL	FBhV	High Availability ramp function block with real
HA_CS31_TOF	FBhV	High Availability off delay timer function block
HA_CS31_TON	FBhV	High Availability on delay timer function block

Table: HA library versions and runtime system details

High Availability Library				
Library name	HA_CS31_AC500__V.V13.lib (V1.3.0)	HA_CS31_AC500_V20.lib (V2.0.0)	HA_CS31_AC500_V23.lib (V2.3.0)	HA_CS31_AC500_V23.lib (V2.4)
Tested with firmware version	V13	V20	V23	V24
Number of supported CS31 buses	1	1	1	7 (6 via 3 CM574 modules)
Functions and function blocks				

HA_CS31_CALL-BACK_STOP (pro-gram)	x	x	NA	NA
HA_CS31_CALL-BACK_STOP (func-tion)	NA	NA	x	x
HA_CS31_CONTROL	x	x	x	x
HA_CS31_DATA_SY NC	x	x	x	x
HA_CS31_DIAG	x	x	x	x
HA_CS31_DIAG_VIA _CM574	NA	NA	NA	x
HA_CS31_DIAG_ON _CM574	NA	NA	NA	x
HA_CS31_DIAG_EXT D	x	x	x	x
HA_CS31_DIAG_EXT D_VIA_CM574	NA	NA	NA	x
HA_CS31_CTD	x	x	x	x
HA_CS31_CTU	x	x	x	x
HA_CS31_CTUD	x	x	x	x
HA_CS31_INTE-GRAL	x	x	x	x
HA_CS31_PID	x	x	x	x
HA_CS31_PID_DV50 0	NA	NA	NA	x
HA_CS31_PID_FIX-CYCLE	x	x	x	x
HA_CS31_PID_FIX-CYCLE_DV500	NA	NA	NA	x
HA_CS31_RAMP_IN T	x	x	x	x
HA_CS31_RAMP_RE AL	x	x	x	x
HA_CS31_TOF	x	x	x	x
HA_CS31_TON	x	x	x	x

Glossary

BOOL Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

BYTE BYTE belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 121: For integer data types the following range limits are valid:

TYPE	BYTE
Lower limit	0
Upper limit	255
Memory space	8 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DINT

DINT belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 122: For integer data types the following range limits are valid:

TYPE	DINT
Lower limit	-2147483648
Upper limit	2147483647
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

DWORD

DWORD belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 123: For integer data types the following range limits are valid:

TYPE	DWORD
Lower limit	0
Upper limit	4294967295
Memory space	32 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

INT

INT belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 124: For integer data types the following range limits are valid:

TYPE	INT
Lower limit	-32768
Upper limit	32767
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

WORD

WORD belongs to the integer data types. The different numerical types are responsible for a different numerical range.

Table 125: For integer data types the following range limits are valid:

TYPE	WORD
Lower limit	0
Upper limit	65535
Memory space	16 bits

Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on. For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function 'Function Blocks' must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block. As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated. In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blockscan have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory)

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory)

For function blocks without historical values only one instance has to be defined for the function block type. This instance can be used for several calls of the function block (also with different I/O values).

1.5.5.2 AC500 HA-Modbus TCP

1.5.5.2.1 Safety instructions and preconditions to use HA-Modbus TCP library

Safety instructions

Consider the following before using the libraries:

- All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.
- Read the complete safety instructions of the user's manuals for the drives you are using, before installation and commissioning.
- Read all *Chapter 1.6.1.1 “Safety instructions” on page 3697* for the AC500 PLC.
- Read the user information of the devices and functions you are using, see Automation Builder online help.

Preconditions of HA-Modbus TCP library

The library package has been released for the software and firmware versions listed in the readme file of Automation Builder only (see *“Help ➔ Automation Builder Release Notes”*) .

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the HA library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e.g. of example programs.

CAUTION!
Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

The function blocks contained in the library can be executed only in RUN mode of the PLC, but not in simulation mode.

1.5.5.2.2 HA-Modbus TCP - System technology

The AC500 High Availability system

The AC500 High Availability system is designed for the demand of automation systems that require a higher availability, which is realized by redundant devices and communications. The redundancy concept reduces the risk of losing production due to failure of parts of the automation system and thereby minimizes scheduled idle times.

For instance, control can be taken over by the secondary station automatically if the primary station fails.

AC500 High Availability system implements redundancy based on standard AC500 PLCs:

- PLC
- Field communication
- SCADA communication

General differences in high availability / redundancy systems are in which way and how fast the switchover between redundancies happens.

- Cold standby: A replacement system is there but not up and running - Process has (to allow) to completely stop for switchover – e.g. outputs may go to zero.
- Warm standby: Both CPU may be running (= warm) but e.g. communication need to be started/stopped for switch-over - Process needs to tolerate longer freeze times e.g. on outputs - e.g. several seconds.
- AC500 High Availability systems are "hot-standby":
  - Redundant CPUs and all communications are always up and running (hot)
  - Continuous failure detection in both CPU´s and mutual exchange of status
  - Continuous synchronization of critical/historical data from primary to secondary
  - Automatic switch-over in very short time in case of any failure in primary CPU

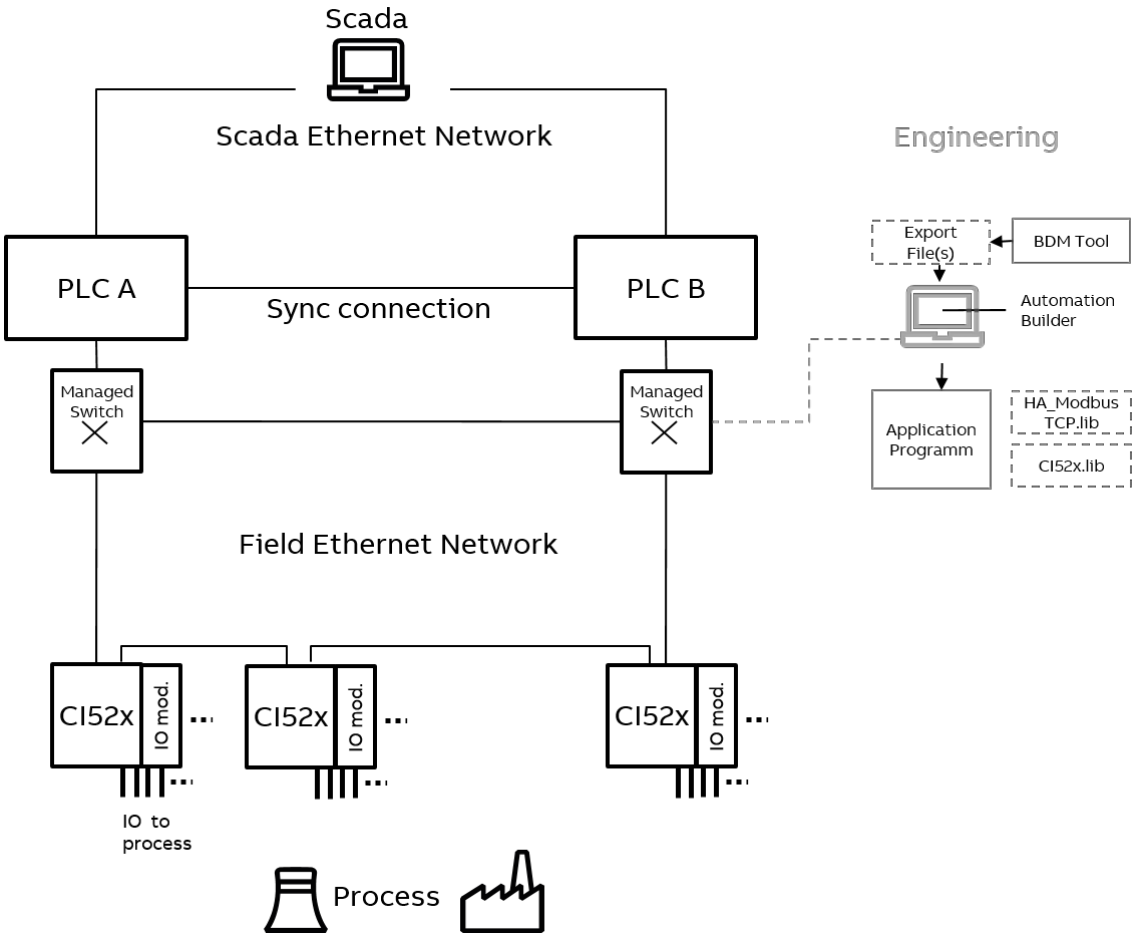


Fig. 145: Principle AC500 HA-Modbus TCP architecture example based on Ethernet redundancy

Details of AC500 HA operation along the figure above:

- PLC redundancy: The two PLCs (A and B) are running in parallel and calculating and reading.
  One is *"primary"* = active, which means also writing data to field devices.
  The other one is *"secondary"* (= stand-by), also calculating but only reading data from field and receiving synchronization (or short = sync) data from the primary.
- Synchronisation data are critical internal variables with e.g. historical content, which will be transmitted from primary to secondary CPU over the sync connection, so that secondary always has the latest data and can take over immediately. Automatically synced are the historic data of the special HA library function blocks (like counters, timers, integral controllers, …), additional Data e.g. of events and diagnosis can be synced by the user with sync blocks. The sync connection also transmits a *"lifecom1"* signal (back and forth) containing diagnosis data of each CPU, so that both CPU know the status of the other CPU. If secondary CPU receives no *"lifecom1"* anymore it assumes that primary CPU has a failure and takes over primary status. If the sync connection is broken both CPUs would try to adopt primary status, therefore, a separate connection *"lifecom2"* is used to differential a "sync link" failure from an "other PLC" failure. The *"lifecom2"* should be routed via a different physical communication path than the data sync/lifecom1, e.g. the Field or SCADA network.
- The field I/O connection is performed via the Ethernet protocol *ModbusTCP* - connecting the CI52x devices (↳ *Chapter 1.6.2.8.5.1 "CI521-MODTCP" on page 4864* or ↳ *Chapter 1.6.2.8.5.2 "CI522-MODTCP" on page 4904*).

For high availability/redundancy of the field or SCADA network, proven Ethernet network redundancy mechanisms are used. (In AC500 this is assumed to be realized by at least 2 (to avoid a single point of failure) external, managed switches), which has the advantage to be able to use AC500 HA with any faster redundancy mechanism / protocol.

- For the I/O communication with CI52x modules two variants exist (see online help: PLC Automation with V2 CPUs ➔ PLC integration ➔ Device specifications ➔ Communication interface modules (S500) ➔ Modbus XY)
  For smaller systems, the CI52x modules can be directly daisy chained (as in previous figure above) if MRP (Media Redundancy Protocol) or DLR (Device Level Ring) is used. Ci52x are not actively participating in ring recovery however, a special FW allows fast ring detection and very short freeze times. Larger systems with e.g. many IO and clusters typically anyway connect to the network via a dedicated managed switch.
- SCADA connection is redundant by nature of the two Ethernet ports and can be extended with further redundancy level as well by managed switches. SCADA itself can also switch the primary PLC to ensure communication to the active PLC in case of a simple connection and a connection failure. If the redundancy mechanism of the OPC DA server is not used, SCADA level itself must be able to handle and differentiate primary and secondary PLC and IP addresses based on the HA-status bits. For CP600 a script exists to do the same for Modbus or AC500 communication protocol.

In most PLC applications the critical components to fail are, beneath PLC, typically the power supply or communication components such as wires or switches. Therefore a *SPOF (Single Point Of Failure)* has to be avoided by adding redundant devices or redundancy functions wherever a failure likelihood is high and failures are not tolerable.

HA core functionality typically can tolerate only a single failure in the different levels. Then, a repair of the failed part is highly advised to achieve and ensure redundancy again. As shown in the above figure, the I/O-network cabling already provides a second independent redundancy layer e.g. for cable failure by its redundancy mechanism (e.g. ring), which can keep up communication without switching the PLCs: There a second failure in the PLC level could be tolerated as long as both connecting, managed switched still work, but it is highly advised to repair immediately anyway.

The AC500 High Availability system itself only takes care of the first fault. For example, in case of a second fault the primary PLC remains primary PLC until the second fault occurs. This results in no further switchovers (manual switchovers included).

Due to the efficient data sync mechanism, which allows data sync over normal and shared ethernet networks, with a well-planned communication network, the PLCs can operate geographically separated (by many 10^{th} of kilometers). So even in catastrophic events with full mechanical destruction still one PLC will be available to control the process or infrastructure.

The secondary PLC or single CI52x modules can be exchanged in a running system without interruption of the primary PLC or the process. (Check document in “Examples” directory of Automation Builder if HA package was installed.)

Libraries

In order to achieve high availability, the CODESYS application must be enhanced with HA function blocks, from the HA-Modbus TCP library and the CI52x library. If the bulk data manager tool (BDM) is used for configuring the System and I/O modules - this is done automatically for the basic initial configuration step by code creation resulting in a prepared user specific “template” application (see below).

- HA-Modbus TCP library contains *HA control* and *HA utility* function blocks
  - *HA control* function blocks manage the core HA functionality by collecting diagnosis and switching if necessary.
  - *HA utility* function blocks provide standard functions in the application program with internal *sync* for integral data e.g. timers, counters, PI control.
- *CI52x library* contains a function block to configure and communicate to the communication interface modules and ensures that only the primary PLC writes to the outputs. The inputs are read by both PLCs.
- For both PLCs the same application must be used/downloaded.

Bulk data manager tool (BDM)

For configuration of the CI52x Modbus TCPs, a separate Bulk Data Manager tool (BDM) is provided. Especially in larger systems usage of BDM is recommended to comfortably engineer HA and create CI52x related configuration and variable data in one place:

- Configuration and parameters of the used I/O modules
- Program code creationfor variable naming, configuration, communication and all basic HA functionality

The BDM tool can serve SCADA programming and documentation as well in an efficient manner.

Hardware, requirements and options overview

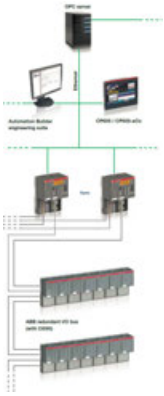
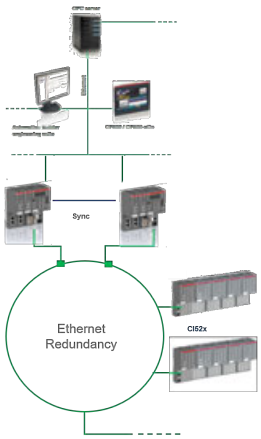
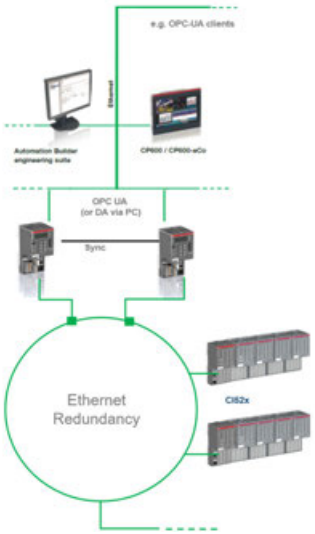
Two same type AC500 PLCs are required as central hardware components. Each PLC is equipped with at least two Ethernet ports at a processor module or at a communication module. The two PLCs, called PLC A and PLC B, are linked by Ethernet to exchange and synchronize information (*Sync*). Connections to the AC500 peripheral field devices (I/O) are performed via Ethernet as well.

For further information on which CPU type and library to be used refer to *Table 126 “Overview of AC500 HA systems and options” on page 2093*.

The following table gives an overview of the different High Availability variants possible with AC500.

The figures are indicative, depend on chosen architectures, system size, network and CPU/CM modules used.

Table 126: Overview of AC500 HA systems and options

Library version		HA-CS31	HA-Modbus TCP					
CPU version		V2 CPUs	V2 CPUs			V3 CPUs		
I/O communication		Parallel serial	Ethernet			Ethernet		
CPUs		PM573 - 595	PM573	PM591	PM595	PM5630	PM5650	PM5670
		Parallel serial	I/O network based on Ethernet and ext.edundancy mechanism					
Max. system size CI52x [1, 6]		3 - 50 [2]	3	< 25 / 50	< 60 / 92	< 30	< 50	< 120
I/O modules		CI590: S500	CI52x: S500 and S500-eCo usable [4]					
Switch-over times	CPU	25 -120 ms [3]	Typically < 50 ms ~[6]					
	Field	15 - 120 ms [3]	Depends mainly on network size, redundancy mechanism of external switches [7]					
SCADA connectivity		OPC DA, IEC60870, …	OPC DA, IEC60870, …			OPC DA, OPC UA, IEC60870, …		
Interfaces		Several CS31 and Ethernet	Several ETH ports, via CM597			2 ETH ports [5] + 1 CAN Interface		
Sync		UDP	UDP			UDP		
Lifecom1		-	UDP			UDP		
Lifecom2		-	Modbus TCP			Modbus TCP / CAN		
Overview of AC500 HA system								

[1] Number of CI52x recommendation based on performance or max. number of sockets (CPU and CM modules).

For more details of sockets supported in AC500 V2 PLCs refer to *Chapter 1.6.4.1.6.1.1.2 “Numbers and usage of Ethernet sockets” on page 5448*.

[2] Limited by CPU performance, number of CM574 modules number of CS31 clients and process data limits.

[3] Depends on system size and CPU type.

[4] For details on certain S500-eCo modules not supported, see the Automation Builder release notes, Appendix 1.

[5] CM597 not available for V3 CPUs.

[6] Based on HA bits switchover, depending on failure case *Chapter 1.5.5.2.2.3.2 “Use case descriptions” on page 2100* *Chapter 1.5.5.2.2.5.2 “Task configuration recommendations for HA system” on page 2113*.

[7] Field network: If CI52x are used with their 2 ports as part of a ring: In the moment of a network switchover single telegrams may be destroyed: - for V2 ETH onboard: Standard TCP delays repeats by 500 ms - for V3 CPUs onboard or V2 CPU using CM597: A special HA-FW ensures fast repeats of typ. ~50ms (settable).

CPU choice, system size and performance indications

The diagrams below indicate the example choices of AC500 CPU's (horizontal axes) based on the number of communication interface CI-remote I/O clusters (Communication Interface modules; numbers see legend) used in a system and resulting application cycle times (vertical axes).

Further details can be found in *Chapter 1.5.5.2.2.5.2 “Task configuration recommendations for HA system” on page 2113*. The values in below graphs base on the assumption to use max. 50-60% as CPU loading by the bare fast IO communication and HA functionality. So the application load would come on top and cycle times (especially HA, Modbus) need to be relaxed (made higher) compared to below indication.

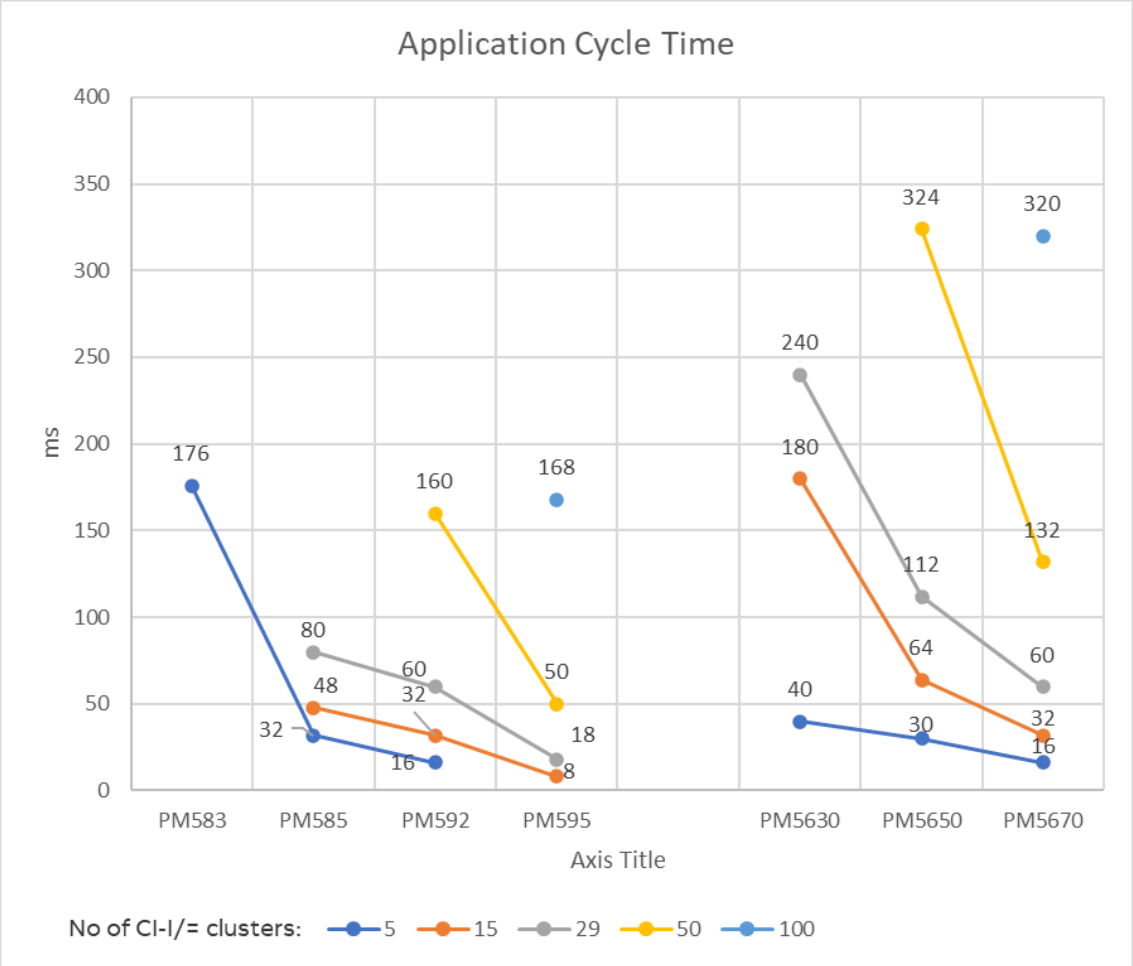


Fig. 146: Indication of AC500 CPU's performance (horizontal axes) based on the number of communication interface CI-remote I/O clusters (Communication Interface modules; numbers see legend) used in a system and resulting application cycle times (vertical axes).

Example: If you need a system supporting min. 25 CI at application cycle time around 120 ms, suitable options based on above graph would be V2 PLCs - PM592 or PM595 and V3 PLCs – PM5650 or PM5670. The main parameter in the application cycle determination is the amount of overall Sync data, which is assumed 160 bytes per CI for the smaller systems, up to 250 bytes per CI for the larger ones. Sync data of the project of in total more than ~1200 byte necessitates several HA cycles to transfer within one application cycle.

The V2 or V3 PLCs types, also differ in available interfaces, protocols supported and memory size.

CI521-MODTCP or CI522-MODTCP can be used as peripheral devices which communicate via the Modbus TCP protocol with the PLCs. The HA-Modbus TCP library supports currently up to 120 CI52x, depending on the CPU type as listed in ↳ *Further information on page 2093*. Each CI52x supports up to a maximum of 10 S500-I/O modules. Nevertheless the standard Modbus TCP communication of the HA library transfers only 120 words per cycle: Therefore please check if for your module configuration matches: In case of many analog IO modules with high-density - like 16 channel AI523/AO523 or modules with fast counters - this limit might be surpassed by roughly 5-6 such modules (to help calculate exactly, there is an Excel sheet provided in the HA “Examples” subfolder of Automation Builder once installed).

For more details of sockets supported in AC500 V2 PLCs refer to ↳ *Chapter 1.6.4.1.6.1.1.2 “Numbers and usage of Ethernet sockets” on page 5448*.

Local I/O on a CPU can signal / interact for diagnosis or service with / from this CPU. This local I/O is not redundant and won't be available to communicate to in case of a CPU failure.

Hardware connections

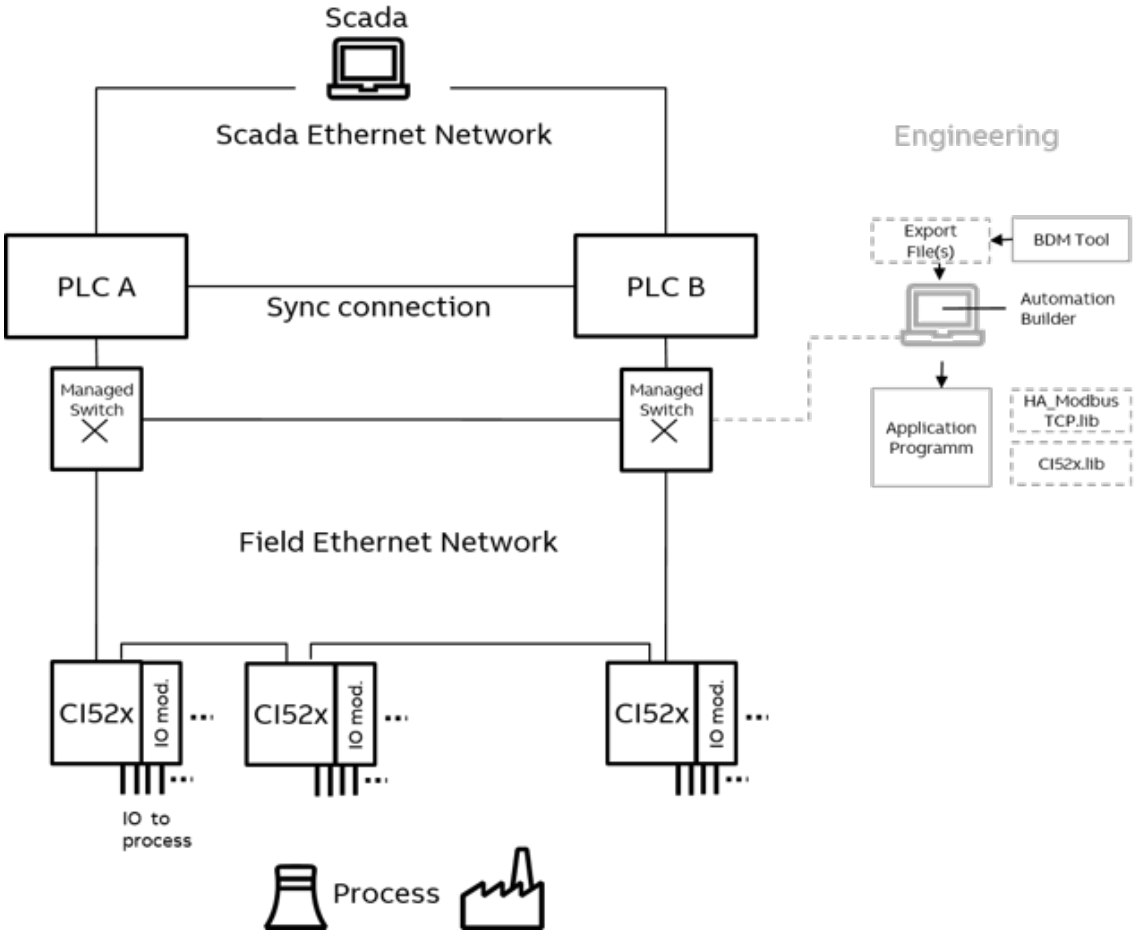


Fig. 147: AC500 HA and SCADA connection

SCADA/ Engineering connection is done using ETH ports of both PLCs and one or several managed Ethernet switches depending on the redundancy requirements in the Ethernet levels.

- HA communication between PLC A and PLC B must be done via two physical connections between PLC A and PLC B in order to distinguish a “sync link” failure from another PLC failure:
  - Sync (including *“lifecom1”*) over Ethernet
  - *“Lifecom2”*
    over Ethernet (Modbus TCP): Can be combined with Field or SCADA network or a separate Ethernet network
    over CAN (only possible with AC500 V3 CPU)
- Field devices (CI52x modules) will be connected via Ethernet switches, forming a redundant network (if requested). For details on network configuration see ↳ *Chapter 1.5.5.2.2.5.3 “Field I/O network topologies” on page 2115*.

The following table shows possible combinations of connections for different CPU types. There must be at least two physical connections. The availability can be increased with a third physical connection, e.g. CM597 for AC500 V2 CPUs or CAN for AC500 V3 CPUs.

	V2 CPU with 1 ETH onboard + CM597		V2 CPU with 2 ETH onboard (+CM597)			V3 CPU with 2 ETH onboard (+ CAN)			
Scada / Engineering	1	1	1	1	1	1	1	1	1
Sync / lifecom1	1	11	1	2	2	1	2	1	2
lifecom2	11	1	2	1	11	2	1	3	3
Field (Modbus TCP)	11	11	2	2	2	2	2	2	2
# of physical connections	2	2	2	2	3	2	2	3	3

1 ETH1 (orange)
2 ETH2 (green)
3 CAN (blue, applicable only in V3)
11 CM597 communication module at slot1 (grey)
The blue box indicates the example which is used in the next chapters.

The numbers in the figure above define the slot on which the connection is made. Last line # of physical connections define how many physical interfaces are used or connected between the PLCs.

It is also possible to realize an HA system without a communication interface CI module see chapter ↳ *Chapter 1.5.5.2.2.6.1.1 "Configuration without communication interface modules to establish redundancy" on page 2119*.

Hardware Example

HA hardware configuration based on V3 PLC to explain the minimal recommended Ethernet port configuration.

- The *Sync* connection is performed via SCADA network, the
- *"lifecom2"* is performed via field network (or the other way around).

The following figure represents the connection example with the details from the highlighted box (see previous figure).

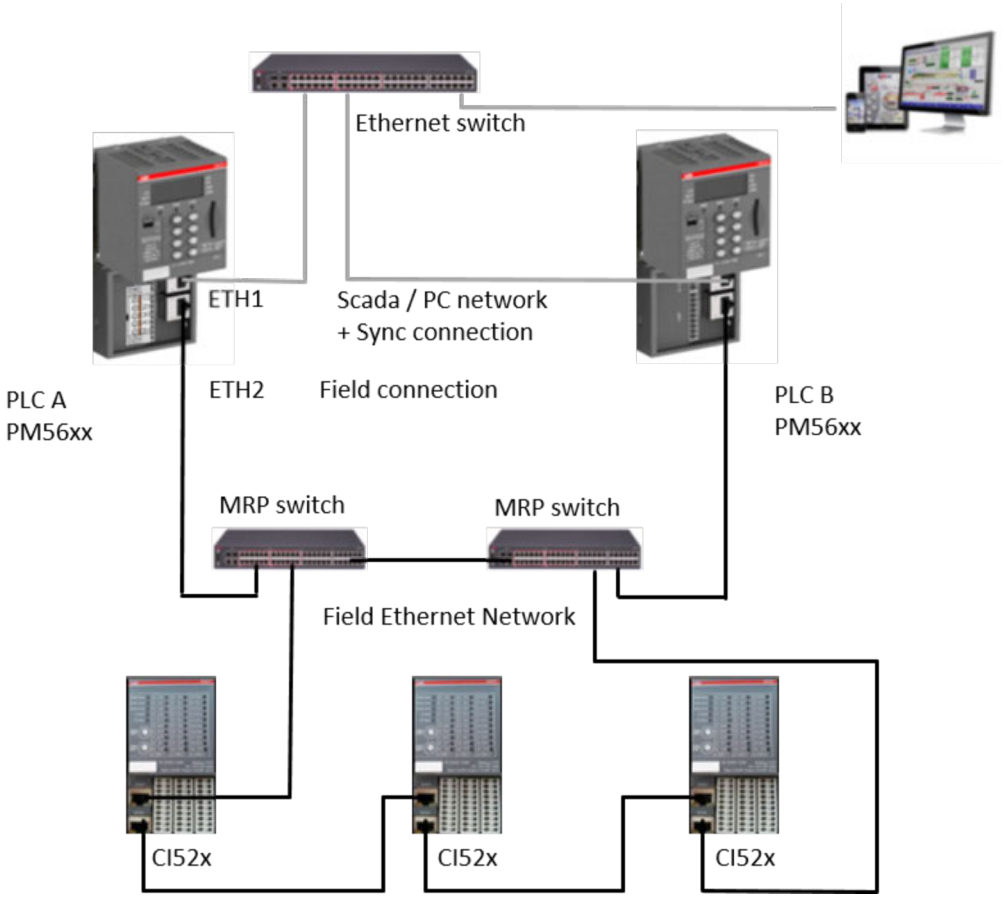


Fig. 148: Physical connection example: 2 ETH CPUs combining sync data / "lifefcom2" with SCADA / Field I/O network

Support of I/O modules (S500/S500-eCo) depends on the version of the library package. See the version details of the library in the Automation Builder release notes.

Functionality

Failures and use cases

The AC500 High Availability system performs a switch-over whenever the primary PLC is powered off, crashed or stopped or if the primary PLC loses fieldbus communication (cut of ETH or defect MRP switch) while the secondary PLC still has connection.

In the following the different use cases and reaction times are outlined.

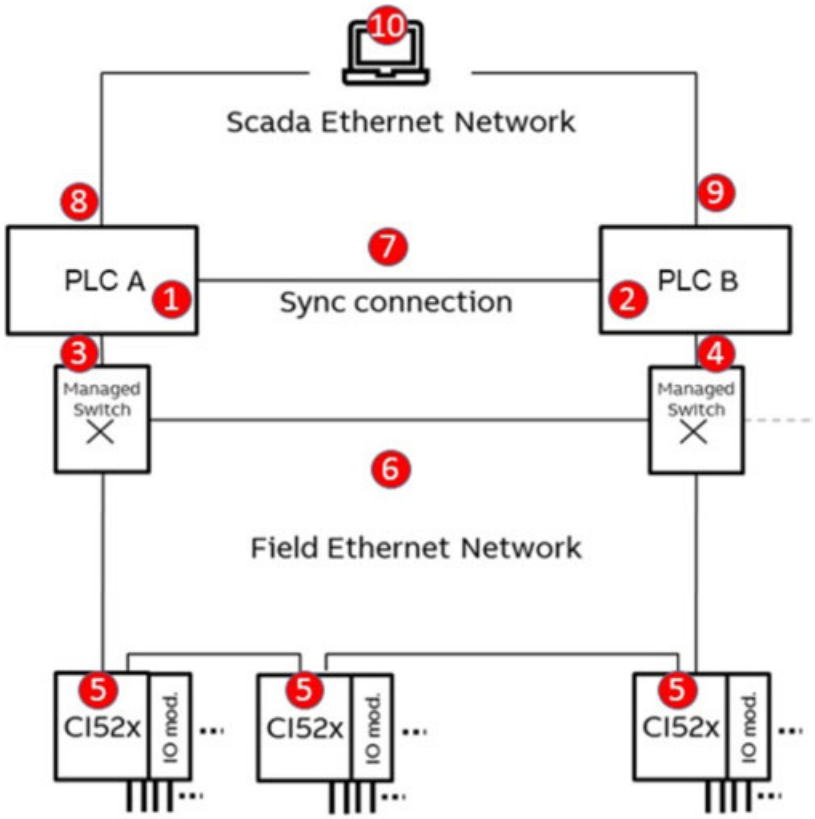


Fig. 149: HA use cases – failures, assuming PLC A is primary and "lifecom2" over field network

The below use case table with reaction and diagnosis messages are based on the setup where Sync is via SCADA network, *"lifecom2"* over field network and PLC A is primary.

Case	Use case	Reaction	Diagnosis message on *)
1	Primary PLC is powered off, crashed or stopped.	Switchover to secondary PLC. CI52x outputs are frozen during switchover period.	Secondary
2	Secondary PLC is powered off, crashed or stopped.	No switchover, process continues.	Primary
3	Primary PLC loses connection to fieldbus CI52x modules while secondary PLC still has a connection.	Switchover to the secondary PLC. CI52x outputs are frozen during switchover period.	Primary
4	Secondary PLC loses connection to one or more CI52x modules.	No switchover, process continues.	Secondary
5	CI52x module is stopped/ powered off.	No switchover, process continues.	Primary and secondary
6	Connection lost in Field Ethernet network.	Depending on Ethernet network structure, and redundancy mechanisms used a reconfiguration time exists.	Lifecom2 lost and CI module lost errors will be generated in primary and secondary.
7	*Sync* and/ or *"lifecom2"* are broken.	No switchover, process continues.	Primary and secondary

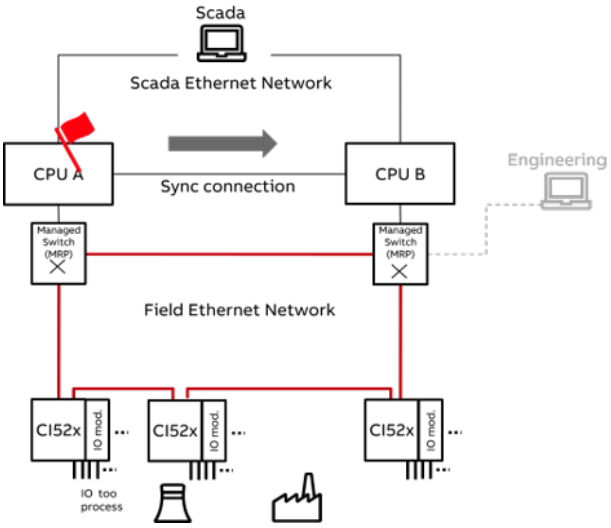
Case	Use case	Reaction	Diagnosis message on *)
8	Primary PLC loses connection to SCADA.	SCADA is responsible to detect and to switch over.	-
9	Secondary PLC loses connection to SCADA.	SCADA is responsible to detect and to switch over.	-
10	SCADA is broken	SCADA is responsible to detect and to switch over.	-
11	Manual switchover by the user.	Switchover to the secondary PLC. CI52x outputs are frozen during switchover period.	-
*) Diagnosis description, see function block description.			

Use case descriptions

The below cases explain the behavior of the system during different use cases.

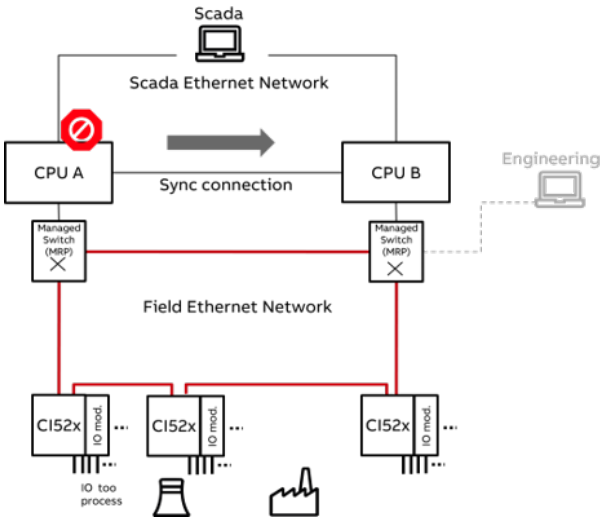
Basic diagnosis information is provided for each case. For diagnosis description refer to *Chapter 1.5.5.2.2.8 “Diagnosis” on page 2124*.

Case 1 a): Primary PLC is powered off or crashes



Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary PLC.
Comment	CI52x outputs are frozen during switchover period.
Diagnosis message on function block	Primary PLC is powered off. Secondary PLC: control block output Runtime Error = 16#001E and xHaModPrimary = TRUE

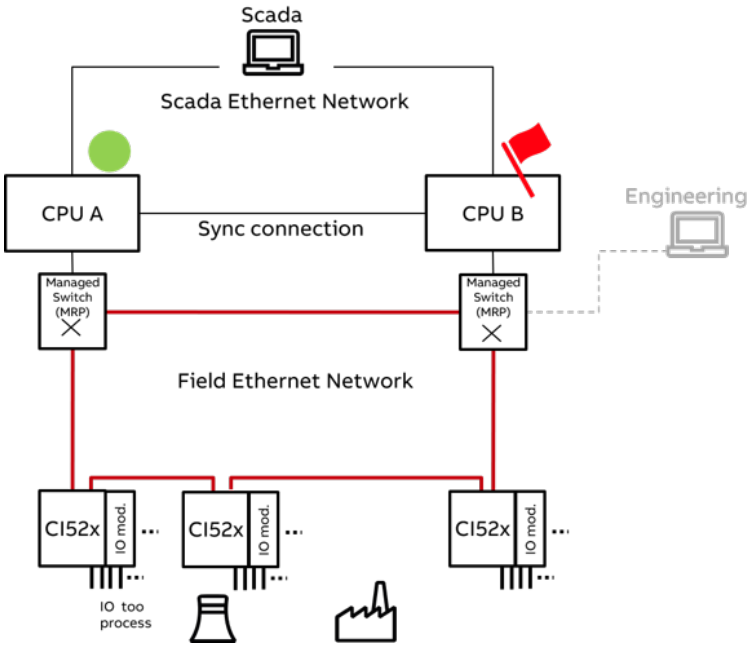
Case 1 b): Primary PLC is stopped



Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary PLC.
Comment	CI52x outputs are frozen during switchover period.
Diagnosis message on function block	Primary PLC is stopped. Secondary PLC: control block output Runtime Error = 16#0016 and xHaModPrimary = TRUE

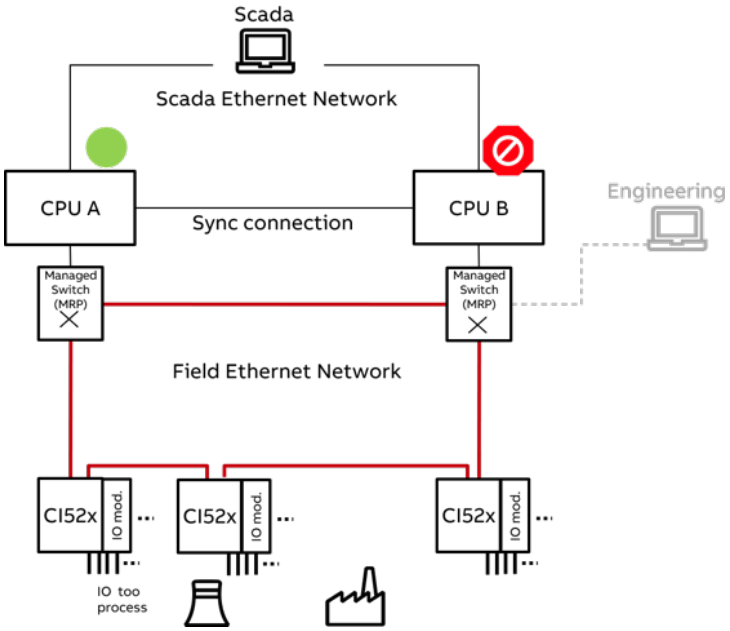
If “lifecom2” is lost and the PLC is in STOP mode RUNTIME ERROR will not be TRUE. This is because Modbus is still responding even if PLC is in STOP mode.

Case 2 a): Secondary PLC is powered off or crashes



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#001E and xHaModPrimary = TRUE Secondary PLC is stopped.

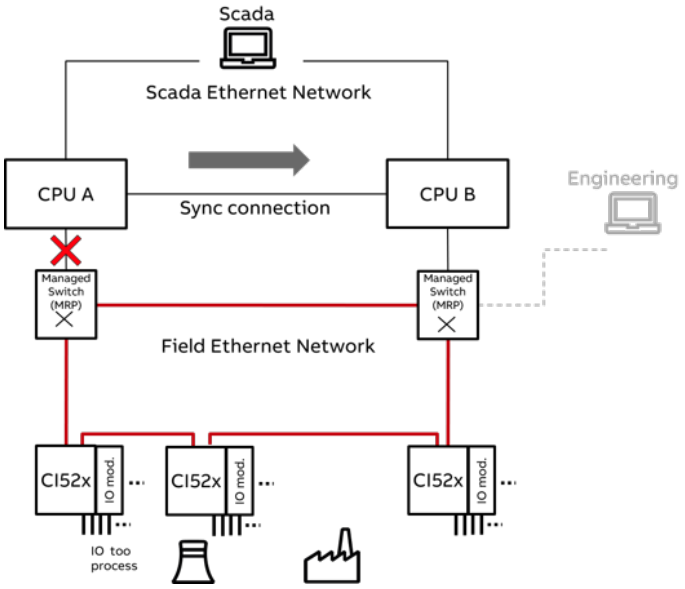
Case 2 b): Secondary PLC stop



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0016 and xHaModPrimary = TRUE Secondary PLC is stopped.

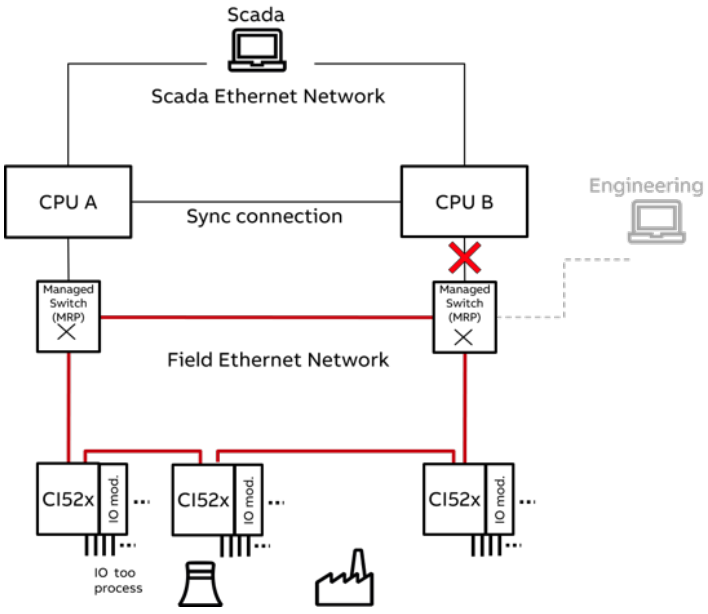
If “lifecom2” is lost and the PLC is in STOP mode RUNTIME ERROR will not be TRUE. This is because Modbus is still responding even if PLC is in STOP mode.

Case 3: Primary PLC loses connection to fieldbus CI52x modules



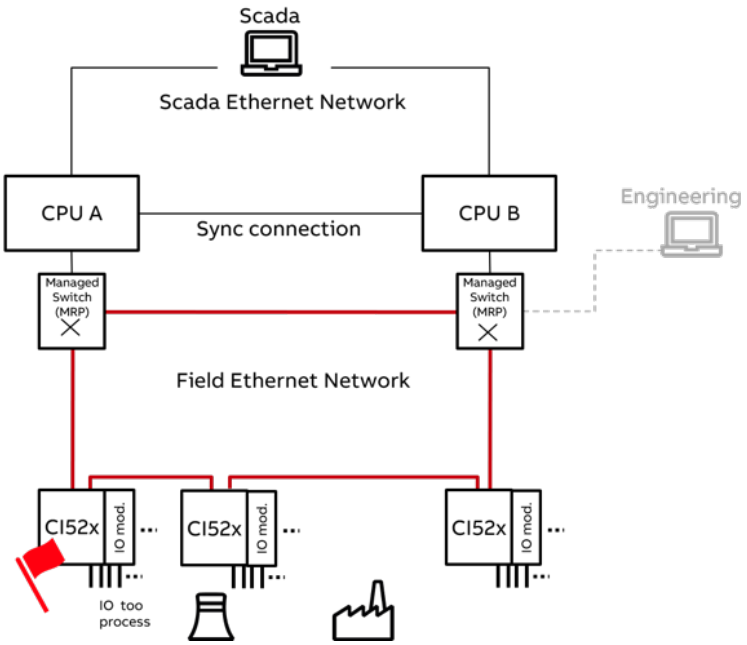
Reaction	Switchover to secondary PLC. The communication interface modules are updated by the new primary.
Comment	CI52x outputs are frozen during the switchover period.
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0094 and xHaModPrimary = FALSE Secondary PLC: control block output Runtime Error = 16#0015 and xHaModPrimary = TRUE

Case 4: Secondary PLC loses connection to fieldbus CI52x modules



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0015 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0094 and xHaModPrimary = FALSE

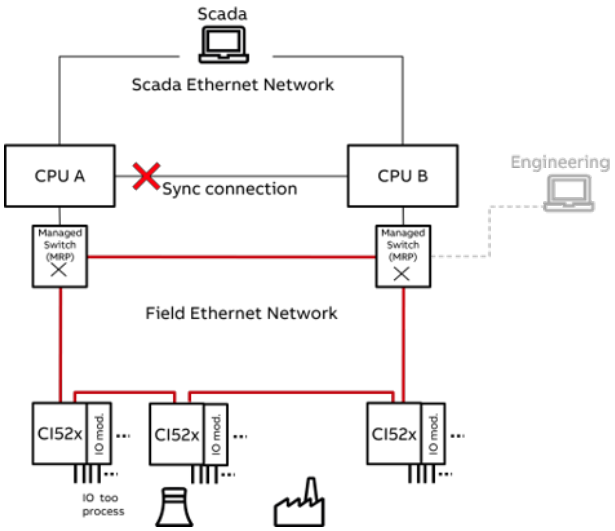
Case 5: CI52x is powered off or stopped



Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0081 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0081 and xHaModPrimary = FALSE

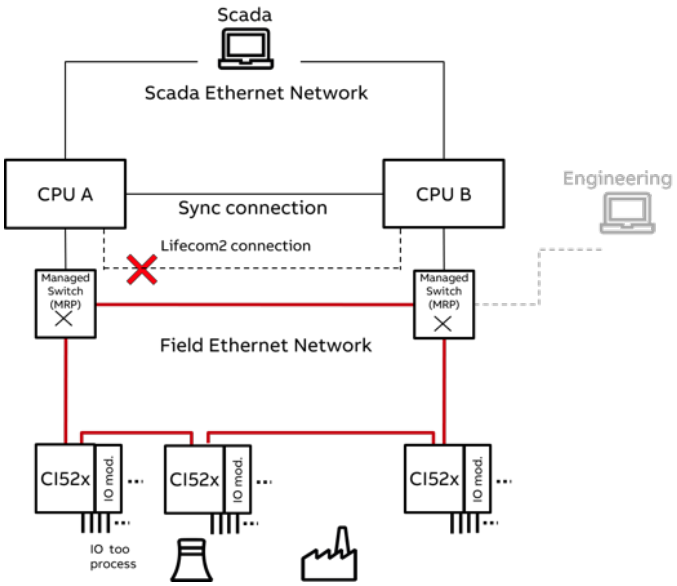
If any CI52x-MODTCP module is powered off and on, there is no need to power restart the complete system. The module will be recognized once the communication is reestablished.

Case 7 a): Sync connection is broken between the PLCs



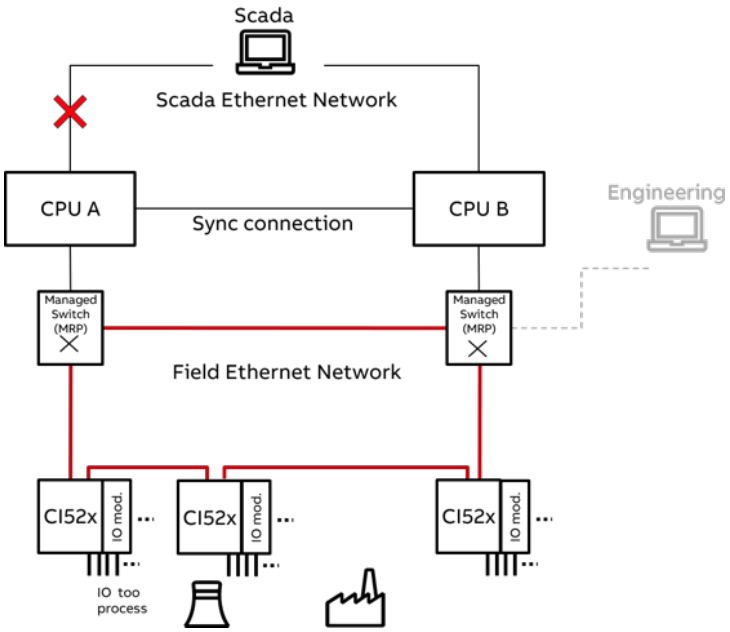
Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0014 / 16#0094 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0014 / 16#0094 and xHaModPrimary = FALSE

Case 7 b): Lifecom2 connection is lost between the PLCs



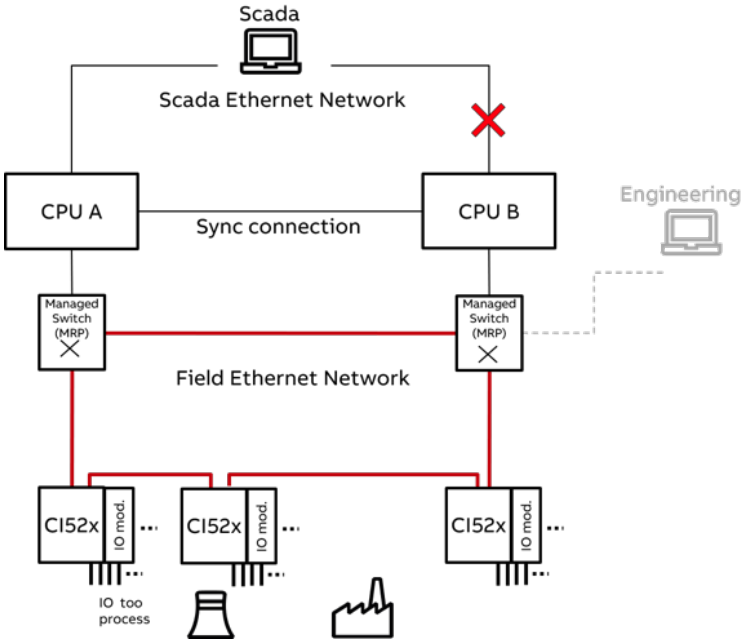
Reaction	No switchover
Comment	Process continues
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0008 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0008 and xHaModPrimary = FALSE

Case 8: Primary PLC loses SCADA connection



Reaction	No switchover
Comment	Process continues, SCADA is responsible to detect and switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE Secondary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = FALSE

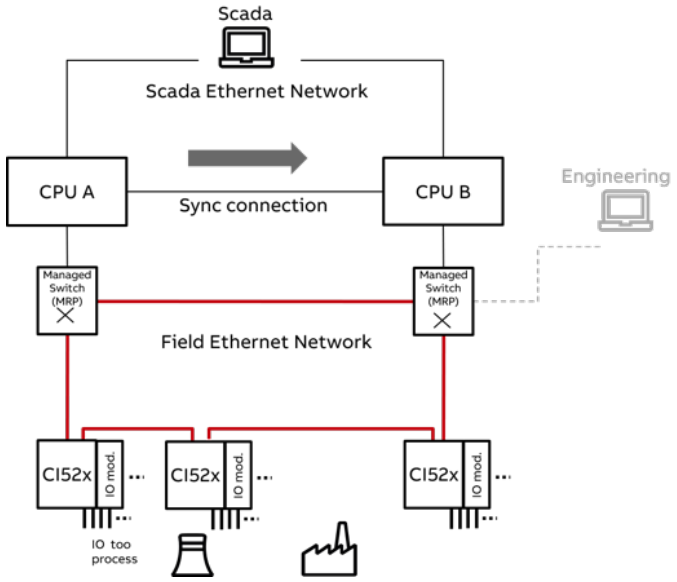
Case 9: Secondary PLC loses SCADA connection



Reaction	No switchover
Comment	Process continues, SCADA is responsible to detect and switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE Secondary PLC: control block output Run-time Error = 16#0000 and xHaModPrimary = FALSE

SCADA link may be combined with sync connection or “lifecom2” connection. In that case runtime error and system behavior will be as described in the cases above (Sync connection lost / “lifecom2” connection broken).

Case 11: Manual changeover by user



Reaction	Changeover from primary PLC to secondary PLC.
Comment	CI52x outputs will be frozen during switchover
Diagnosis message on function block	Primary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = FALSE Secondary PLC: control block output Runtime Error = 16#0000 and xHaModPrimary = TRUE

A manual switchover can be triggered from both PLCs. For each trigger a switchover from primary PLC to secondary PLC will take place.

How to get and install the AC500 High Availability system package

The PS5601- High Availability Modbus library package can be installed from the *Automation Builder Installation Manager* by selecting the component.

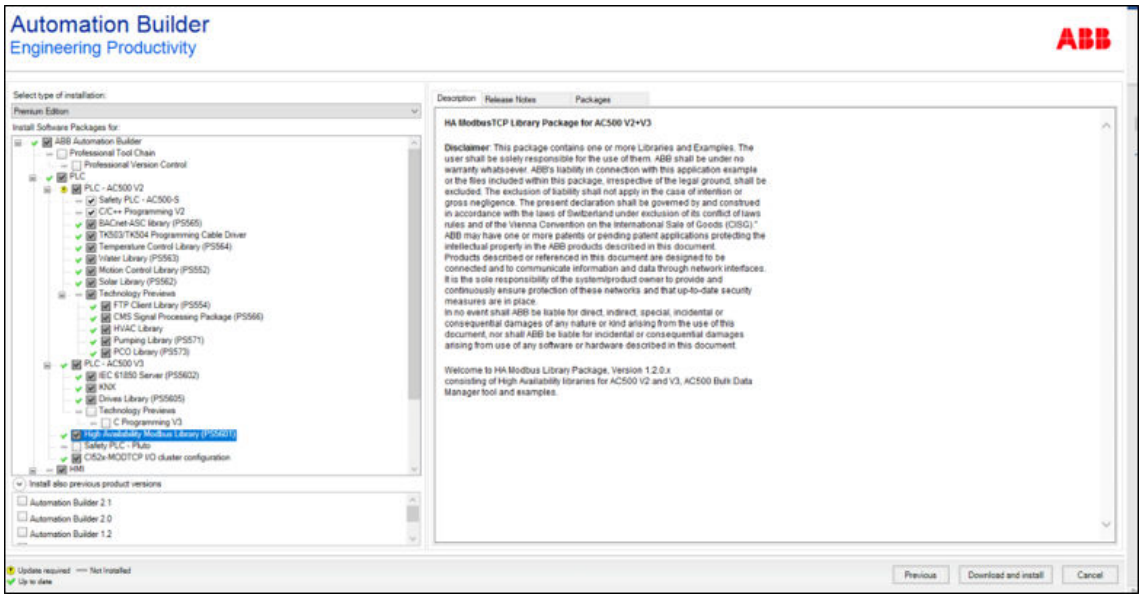


Fig. 150: Automation Builder Installation Manager

The following components are installed:

- Libraries
  - AC500 V2 libraries: *C:\Program Files (x86)\Common Files\CAA-Targets\ABB_AC500\AC500_V12\library\PS5601-HA-MTCP*
    CI52x_AC500_Vxx.lib, HAModbus_AC500_Vxx.lib.
  - AC500 V3 libraries available in library repository:
    ABB_CI52x_AC500.compiled-library, ABB_HaModbus_AC500.compiled-library
- Online help: HA-CS31, HA Modbus V2 function block description
- Automation Builder Example folder: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP*
  - AC500_V2: Examples for AC500 V2 including documentation
  - AC500_V3: Examples for AC500 V3 including documentation
  - BulkDataManager: Bulk Data Manager (BDM) tool which helps efficient engineering in larger projects. This requires a separate installation. Further information can be found in the document: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
  - HA-Modbus TCP System Technology.pdf (this document)

System structure

This chapter explains the detailed structure of the HA system in CODESYS. A HA-Modbus TCP system is characterized by two AC500 PLCs with the following features:

- Identical programs (application with additional HA and Modbus function blocks) that are loaded to both PLCs.
- Communication interface modules CI52x-MODTCP that are connected via Modbus TCP.
- Synchronization of both PLCs (*sync/lifecom1* and *lifecom2* logical connections).

Programming

Each PLC contains at least three main tasks/ programs:

- HA program
- Application program
- Modbus program

The programs in one PLC communicate via internal structures of the libraries and dedicated internal memory areas for HA-Sync array and the Modbus CI52x memory(ies) CiModDataxx.

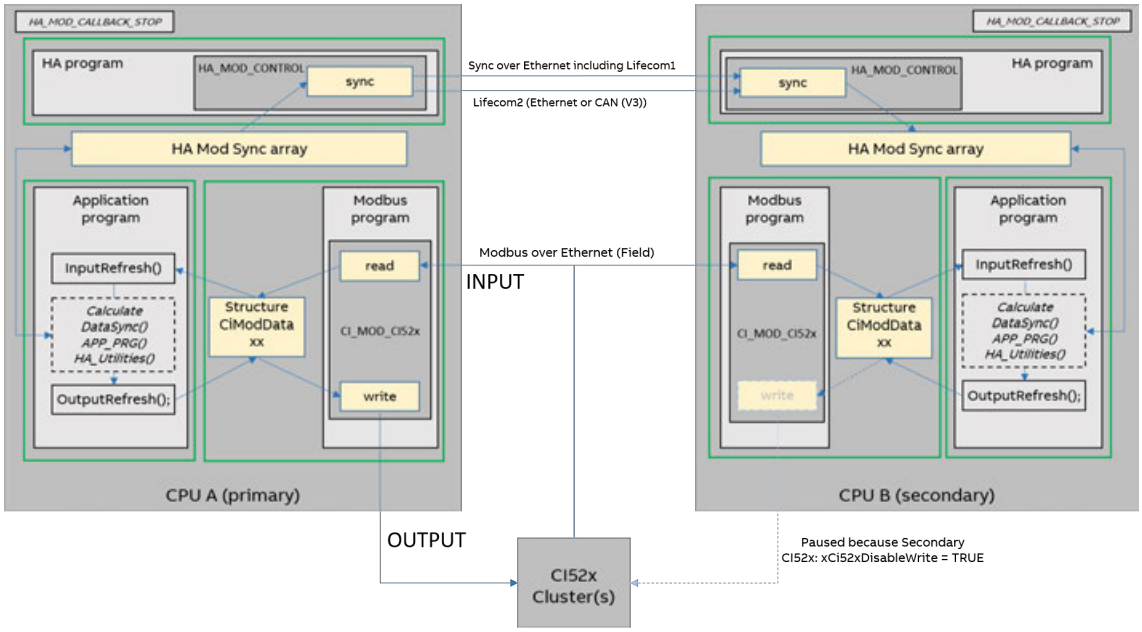


Fig. 151: Principle structure of the HA system and recommended tasks: HA, Modbus, Application

Table 127: Image description

Layout element	Meaning
Dotted outline box	Indicates optional function block or programs.
Solid outline box	Indicates the mandatory function blocks or programs. All mandatory blocks are called when an export is created from Bulk data manager.
Italic font	Indicates the program or functions user should call in his project and not created by Bulk data manager.
Light yellow background block / blue arrow	Indicates the operations which are handled internally in the library.
Green solid box	Indicates the three different tasks which user has to configure.

Modbus program

The function block CIModCI52x (V3) / CI_MOD_CI52x (V2) reads the input values from the CI52x modules and stores them in the structure CiModDataxx. If the CPU is primary it also writes the outputs to the CI52x modules. The Function block also parametrizes the CI Module as configured in e.g. Bulk data manager tool during the first startup or when a CI module is exchanged.

Normally the HA-Modbus TCP library takes care of communication monitoring. Nevertheless if com- munication is cut completely, the CI52x communication interfaces and its I/O modules have to react on their own to achieve a bumpless or desired behavior: The following parameters for the CI52x com- munication interfaces and I/O modules need to be considered:

- *– CI52x: parameter "Timeout" for Bus supervision: [2)]*
  Allows to detect errors from communication interface side as well and take action to ensure a fail- safe behavior if communication is cut. It can be set in 10 ms steps. If set to 0 no bus supervision is active. Proposed value: 50 = 500 ms = default in Bulk data manager; this value should be increased, e.g. to value 65 if AC500 V2 CPU ports are used for field communication to take care of the larger TCP retransmit time.
- *"Behaviour Outputs" at "Timeout for Bus supervision" [1), 2)]. This fail-safe parameter has to be con- sciously set: separate settings are possible for each module (and communication interface): "off"; "last" or "substitute": 5 s, 10 s, ∞ s [1)].*

Remarks:

[1)] The parameters "Behaviour Outputs at comm. Error" is only analyzed if the Failsafe-mode is [ON].

[2)] Both are CI52x parameters set e.g. via Bulk data manager tool in the program.

Application program

- At the start of the application task the InputRefresh program has to be called. It copies data from Modbus via the structure CiModDataxx to the user variables, which were defined in BDM as signals. For further information refer to BDM documentation, chapter 7 which is available in the path: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
- Only the main application programs should be in this task and use these variables for the user defined functions. E.g here the user programs and logic should be called and use the HA libraries utility blocks (which *sync* their historic data automatically) and HA_MOD_DATA_SYNC blocks for further user data which should be synchronized.
- Data of utility blocks and HA_MOD_DATA_SYNC blocks are copied to the HA Sync array of the primary CPU (which is sent to the secondary CPU by the HA program).
- OutputRefresh program is called as a last step. It copies data from the user variables via structure CiModDataxx to Modbus.

Example of a utility function block (with integrated sync data)

Consider the on-delay timer HA_MOD_TON (V2)/ HaModTon (V3).

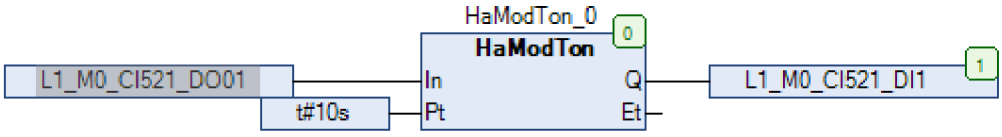


Fig. 152: HaModTon utility function block with internal synchronization

Both PLCs require the same function block called in the program. Under normal operating conditions the elapsed time ET and output Q of the timer is synchronized internally from primary to the secondary CPU. ET and Q data are available and can be attached to local or global variables in the program as per application requirements. If PLC A shuts down due to a fault, the primary status switches over to PLC B.

In the event of a switchover, the moment PLC B becomes the primary, the timer on this PLC will keep running. Until the time of PLC A failure, the timer on PLC B was synchronized. This is most important in cases when one CPU was not in run or off and needs to “catch up” such integral or historic system values (timers, counters, operator settings, …). The actual process remains then unaffected by the switchover.

HA program

HA_MOD_CONTROL has two functions:

- Exchange status data (lifecom1 and lifecom2) and switch from secondary to primary PLC (or vice versa) based on the status according to the use cases described in *Chapter 1.5.5.2.2.3.1 “Failures and use cases” on page 2098*.
- Send sync “HA SYNC” array from primary to secondary PLC to ensure that the secondary PLC is always in hot-stand-by and can take over immediately. UDP protocol is used for data synchronization between the CPUs.

Data synchronization via UDP

This chapter explains how the data synchronization happens between primary and secondary PLC via UDP.

All prepared sync data is synchronized with the secondary PLC. Typically only integral values (timers, counters, PID, …) or settings which might have been received have to be synchronized. For example for fast start-up cases when a secondary CPU was restarted, as both PLCs are running and calculating closely in parallel and based on the same input values, synchronization will make the secondary start with current value instead of default value. For details on how to configure or use the data sync function block refer example projects.

Following steps are performed:

- HA SYNC array is transferred via UDP to the secondary CPU. This includes the exchange of lifecom1 status between primary and secondary CPU.
- In the HA program the HA_MOD_CONTROL function block collects all diagnosis, sync and lifecom2 data from the field and/ or the other PLC. Whether a switchover is necessary is decided based on a simple decision matrix.
- Lifecom2 is exchanged between CPUs over Modbus TCP every cycle.
- One task per program, see figure above.
- Status of the inputs connected to CI52x decentralized I/O stations is transferred to both PLCs simultaneously in every PLC cycle. They are received by the CI52x function block.
- At the end of the program, the generated output values are sent, by transferring from the primary PLC respective buffers to the CI52x-MODTCP module(s) via CI52x function block and Modbus TCP. The secondary PLC is prepared to send but stays “silent” (not sending output values).

PLC needs one HA cycle to send one ETH frame data from primary to secondary CPU and receive acknowledge from secondary CPU. Similarly V3 PLC needs two HA cycles.

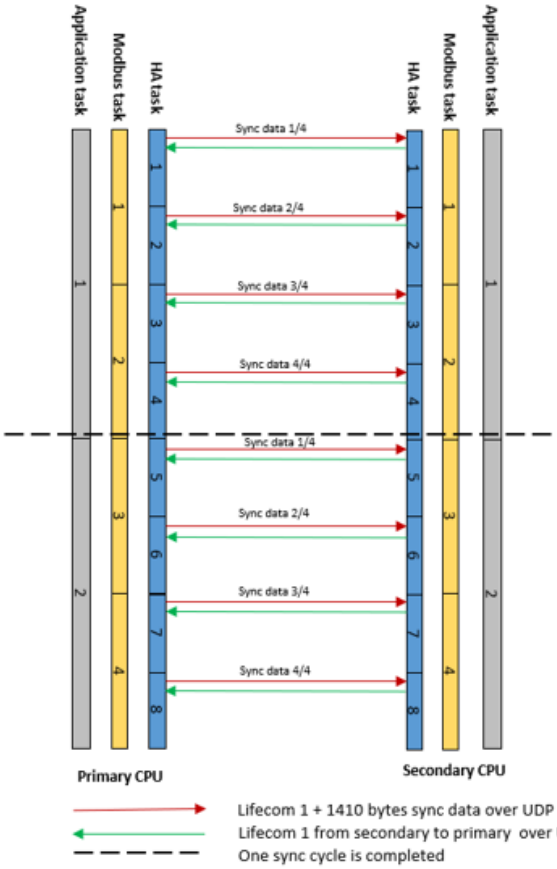
One ETH frame copies approx. 1412 data bytes. The number of ETH frames needed to synchronize HA Sync Array completely depends on the number of data sync bytes. Global variable *iNoOfEthFrames* gives the user this information, which should be used to calculate the cycle time for the application task.

Chapter 1.5.5.2.2.5.2 “Task configuration recommendations for HA system” on page 2113

Up to max. 60 kB of Sync data can be synchronized.

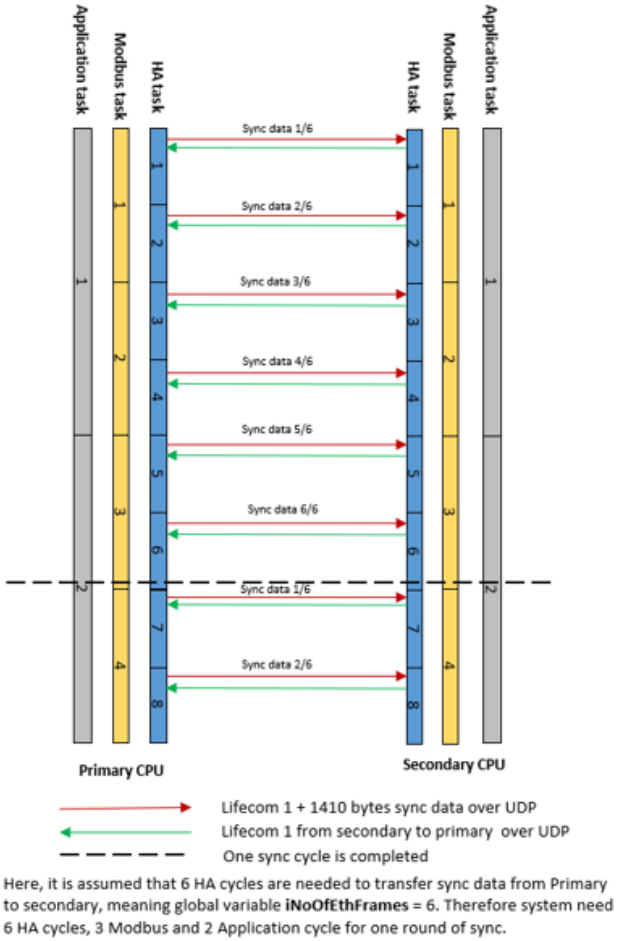
Synchronization between the primary and the secondary PLC happens over a few cycles of HA task time depending on the total sync data bytes configured in the system. Lifecom1 is also exchanged between the primary and the secondary PLC. The primary PLC sends lifecom1 to the secondary PLC along with sync data. Backwards the secondary PLC sends lifecom1 to the primary PLC every cycle.

The following figures shows an example for V2 PLC. When in the project the sync data is equal to 4 *iNoOfEthFrames* then it takes 4 HA cycles to synchronize the data between the PLCs.



Here, it is assumed that 4 HA cycles are needed to transfer sync data from Primary to secondary, meaning global variable **iNoOfEthFrames** = 4. Therefore system need 4 HA cycles, 2 Modbus and 1 Application cycle for one round of sync.

When sync data in the project is equal to 6 *iNoOfEthFrames* then it takes 6 HA cycles to synchronize the data between the PLCs.



Here, it is assumed that 6 HA cycles are needed to transfer sync data from Primary to secondary, meaning global variable **iNoOfEthFrames** = 6. Therefore system need 6 HA cycles, 3 Modbus and 2 Application cycle for one round of sync.

Task configuration recommendations for HA system

For a balanced performance of the HA system consider the following recommendations in your project task configuration:

General

- Use the real time priorities for all HA related tasks. The HA program/ task should be called at highest priority as it is responsible for the core HA functionality and should be the fastest task.
- The Modbus task contains the Modbus communication function blocks at lower priority and (depending on CPU performance) also a faster cycle time to ensure sufficient update rates on Modbus without over- loading the CPU with communication.
- The application program parts should be called in the application task with even lower priority and a larger cycle time than above tasks.
- Configuration to improve standard Modbus TCP for a fast switch over between PLCs.
- AC500 V2
  - CM597ETH_SET_TCP_RTO function block from CM597_ETH_AC500_V28.lib needs to be called inside HA task. User needs to call this function block for each CM597 module connected. For recommended values see example description.
- AC500 V3
  - RTO retransmission time function block “EthSetRtoMin” for the ETH port where fieldbus communication is configured. By default, minimum retransmission time configured is 15 ms.

Task	Priority	PM57x, PM58x, PM59x	PM595-4ETH	V3 PLCs
HA	10 (high)	4 ms or higher	2 ms or higher	4 ms or higher
Modbus	11 (medium)	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))	Maximum of (HA cycle time *2), (3 ms + roundup (#CI/2))
Application	12 (low)	Maximum of (Modbus cycle time *2), (iNoOfEthFrames * HA cycle time)	(iNoOfEthFrames * HA cycle time)	Maximum of (Modbus cycle time *2), (iNoOfEthFrames * HA cycle time *2)

Procedure for task configuration

1. Choose suitable CPU type according to chapter CPU choice, system size, performance indications
2. Configure task priorities according to the table
3. Set HA task to minimum according to above table
4. Calculate Modbus cycle time according formulas in the table, based on HA cycle and number of CI modules “#CI”
5. Calculate Application cycle time according to formulas in the table, based on Modbus cycle time and variable iNoOfEthFrames, which is defined in the global variables of HA-Modbus TCP library.
6. Measure PLC and CPU load during trial operation.

   V2: CPU load : *Chapter 1.5.4.12.1.2 “CPU_LOAD” on page 1168*

   If the PLC load is higher than 40 % or CPU load higher than 60 % then increase HA cycle time (e.g. to 8 ms / 12 ms / 24 ms, …) and go to step 4, repeat the steps until loading is within defined range.

   *A new **V3 CPU configuration option** is introduced from Automation Builder 2.4.1 and onwards which allows to change the priority for Ethernet communication in PLCs.*

   Set this configuration in the device tree of the CPU in Automation Builder double click on PLC “CPU_Parameters Parameters ➔ Communication Schema ➔ Select “Onboard Ethernet””.

   The above parameter should be set to “Onboard” Ethernet for HA systems and it will consequently increase the loading due to the higher priority. PLC Load < 50 % and CPU load < 70 % should be considered as guidelines here instead, while setting the task times while setting the task times.

7. Following timeout values has to be defined in the user project according to the relation defined.

Timeout variables (see definitions in box below table)	HA in V2	HA in V3
timCI52xTimeOut	1 * Modbus Task time	50 ms or Modbus Task time, whichever is higher
timHaModSyncTimeOut	1* HA Task time	2 * HA Task time
timResponseTimeout	Not applicable	50ms or (2 * Modbus Task time), whichever is higher
timCanTimeOut	Not applicable	100 ms or (2 * Application Task time) whichever is higher

8. Add additional applications and SCADA communication: Check PLC and CPU load again vs. your requirements.

In the HA Modbus system different timeouts must be configured for the fine operation of the system as described above in the task configuration for V2 and V3 PLCs. These different timeouts meaning, and relation is explained below:

timHaModSyncTimeOut:

Time limit to check if the new sync data is received or not in the secondary PLC. If this timeout is not defined properly, Sync lost error/ “lifecom1” lost error will be generated.

timCanTimeOut:

Time used for the check whether “lifecom2” is received when configured via CAN. This value is applicable only in AC500 V3. Lifecom2 via CAN won’t be stable between the PLCs and runtime error "lifecom2 lost" will be flickering if not the right value is configured.

timCI52xTimeOut:

Time limit to check whether new data is received in the Modbus field modules. It is also used to check whether “lifecom2” is received when configured via Modbus TCP. If 'timCI52xTimeOut' is not defined as described, “lifecom2” error / communication interface diagnosis error will not be generated as expected.

timResponseTimeOut:

Timeout value to check whether CPU has lost the communication interface modules connected in the network. If this value is not defined as described, communication interface module lost detection will not be indicated properly.

Field I/O network topologies

Modbus TCP communication between PLC and communication interface modules CI521-MODTCP or CI522-MODTCP can be done using different network topologies. In the following subchapters different simple combinations with their pros and cons are explained.

If a CI52x module of a daisy chain is powered off, next following modules will lose connection/ data provided there is no redundancy in the Ethernet network (e.g. ring and managed switch).

Simple ring topology (smaller systems)

In a simple configuration, CI52x modules can be part of a ring if MRP (or DLR) protocol is used in the managed switches. Then the CI52x are connected from one to another device (“daisy chained") through e.g. two network switches. The redundancy protocol detects a closed ring and opens one port of a managed switch to avoid the ring. The user has to configure the necessary ring configurations and enable the ring manager for the used ring ports in one switch.

It is recommended that time interval between ETH cable disconnection and re-connection should be greater than 2-3 seconds.

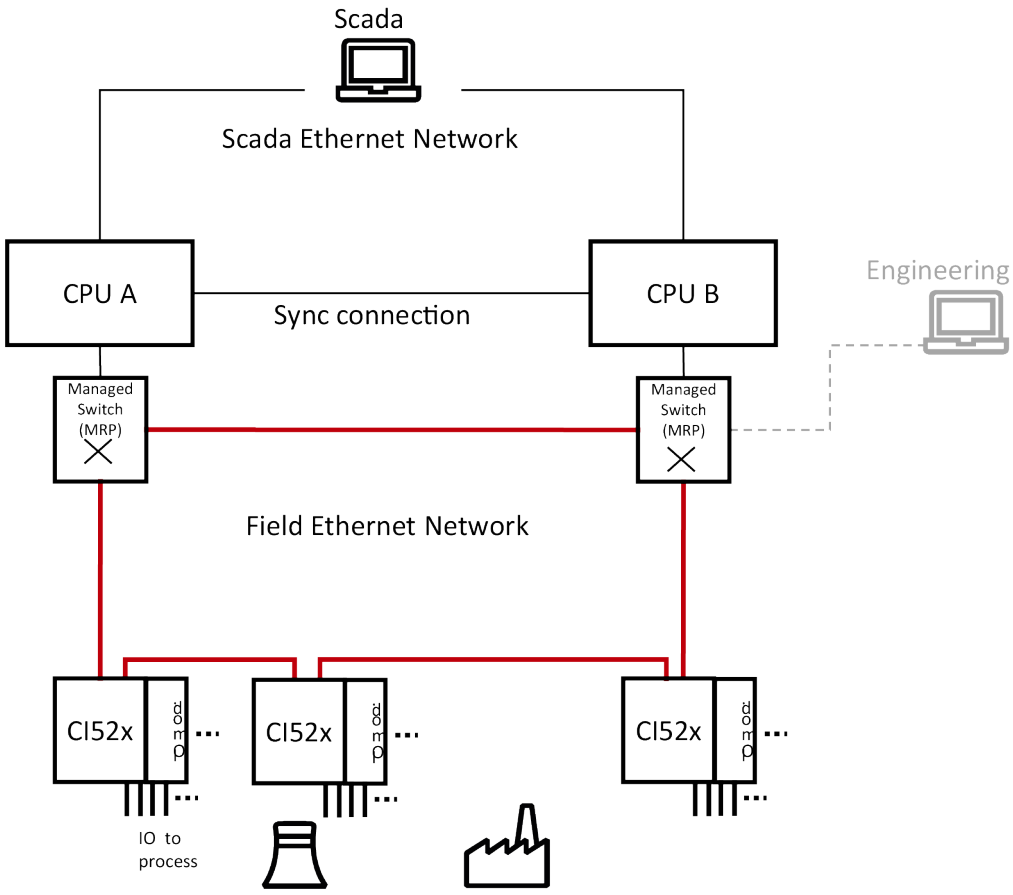


Fig. 153: Redundant ring topology with 2 MRP switches (avoids a SPOF (Single Point Of Failure))

Standard network topology (large systems)

In the standard redundant network, which is often done by third party dedicated telecommunication companies, managed switches are used for every connection point to this network. It’s the network's (and operator's) responsibility to repair any failure fast enough so that no influence on the HA system or its outputs occur.

The network can use other fast redundancy algorithms, also having other than ring structures, if redundancy links are activated fast enough.

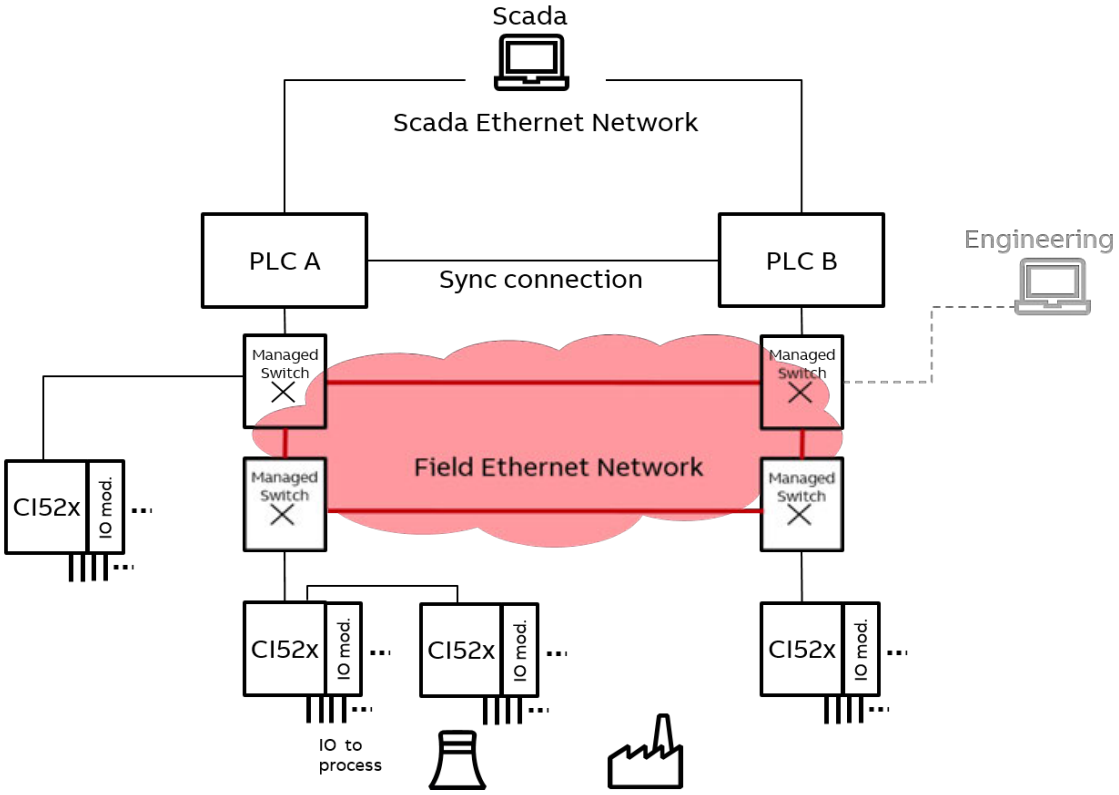


Fig. 154: Redundant ring topology with independent network (using any fast redundancy mechanism internally in a ring or meshed network)

Parallel network topology (using PRP)

Each CI52x module and PLCs as single ended devices are connected by PRP switches to both networks. Here the failure of the switch which connects the primary CPU will also lead to a switchover.

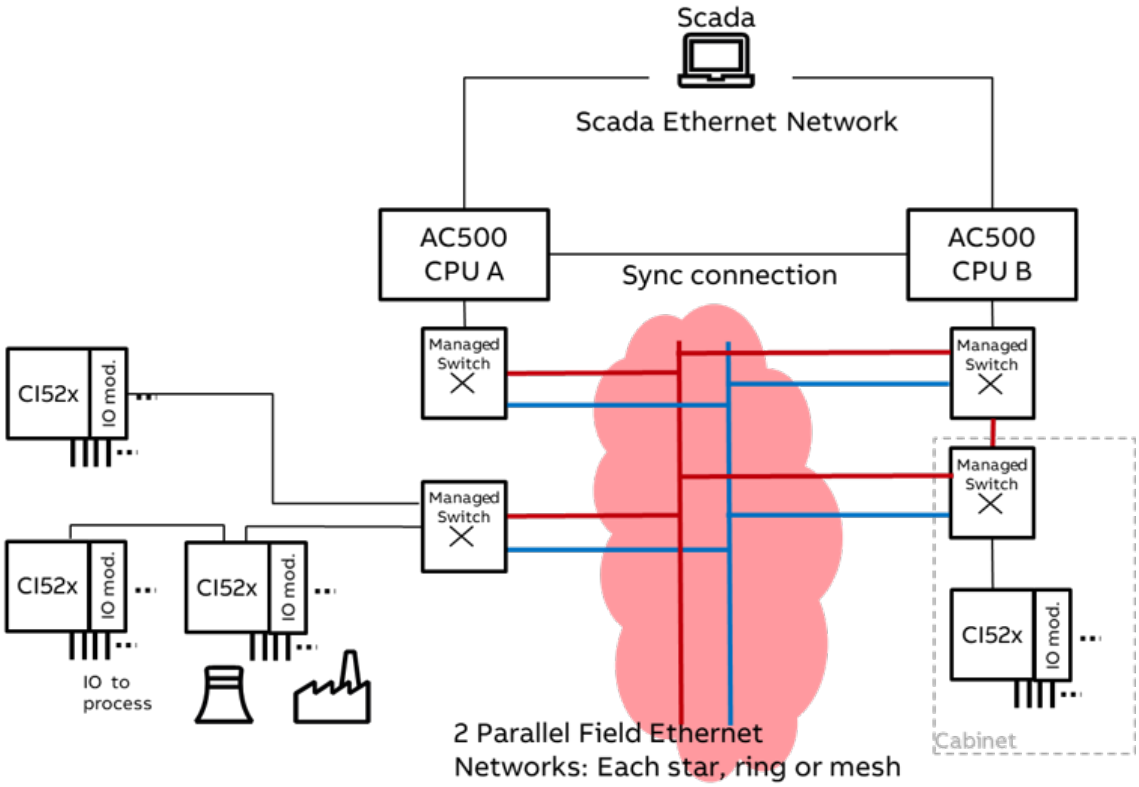


Fig. 155: Parallel-redundant network, PRP switches to connect each device, CPU and communication inter- face modules

Redundancy switchover timing should match the settings in the program and communication interface modules for time-out and freeze periods. The networks for larger systems are often seen as a separate entity and done by a separate company. Make sure to have the redundancy status information of the network at least in SCADA, to repair in time. If the I/O field network responsibility is with the automation/ PLC part, the redundancy status should be also monitored by the PLC. A warning to initiate repair may be created from the managed switches in the I/O field network.

Examples

- Alarm output(s) wired (e.g. to a CI52x input and related settings of the switch(es)).
- Settings of the switch(es) to send (e.g. SNMP traps, which can be received in PLC (AC500 SNMP library)).
- Use of “automation switches” which can also communicate their status directly via Modbus.

It is also possible to connect switches in ring combination with CI modules connected to them in daisy chain. User needs to do the relevant setting based on type of switch and protocol (Ex: MRP, RSTP).

If RSTP ring configuration is used in the system, ring reconfiguration time is slower than other ring protocols. During this reconfiguration, connection to the CI modules will be lost.

HA Modbus system without communication interface modules in the network

It is also possible to have a HA Modbus system without connecting any field devices, CI521-MODTCP / CI522-MODTCP in the network. This system can be used for establishing a redundant PLC system with data synchronization between two AC500 controllers, either without field IO or with user integration of other protocols to field-IO or “intelligent” IO: CPUs as field devices.

Secondary will be on hot standby with primary PLC, during a power off/ stop of the primary PLC. Secondary will take over the control and continue the process. Any user integrated field-IO or CPUs can establish communication mapped with the primary bit: parallel reading but prevent parallel writing.

HA without CI modules can be also used during commissioning to check the data sync, OPC and SCADA related communications without any field devices configured. The user has to set the global variable 'xNoCiBus' to TRUE defined in the HA_GLOBAL_VARIABLES. This variable has to be set to TRUE in both PLCs. Note: It is not advised to update this variable during run time.

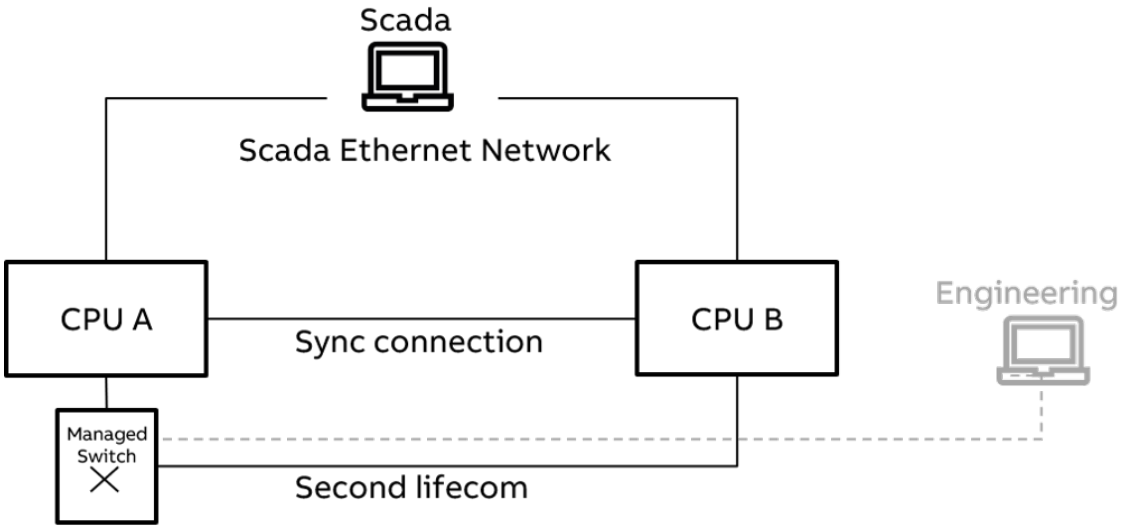


Fig. 156: Simple SCADA connection

Getting started

Quick start list and guidelines

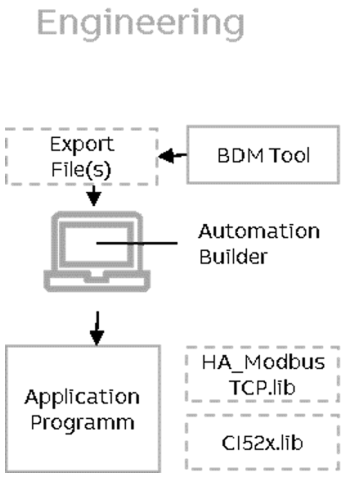


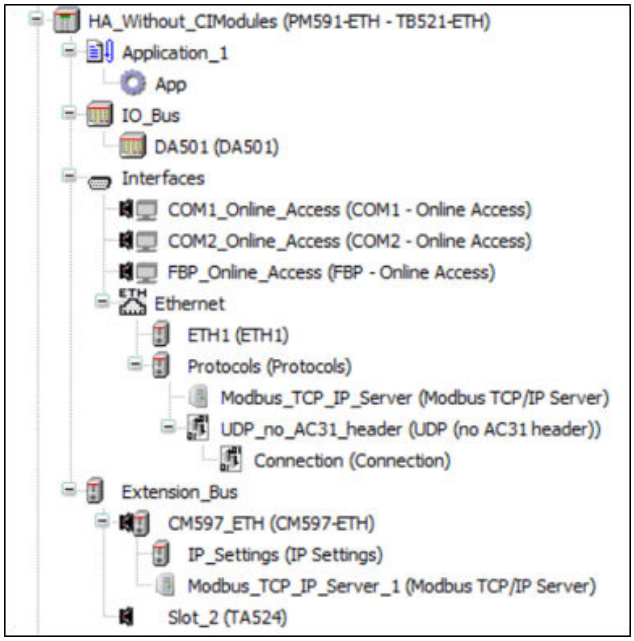
Fig. 157: Engineering workflow using Automation Builder

Simple steps to engineer the HA Modbus system is explained in the following chapters.

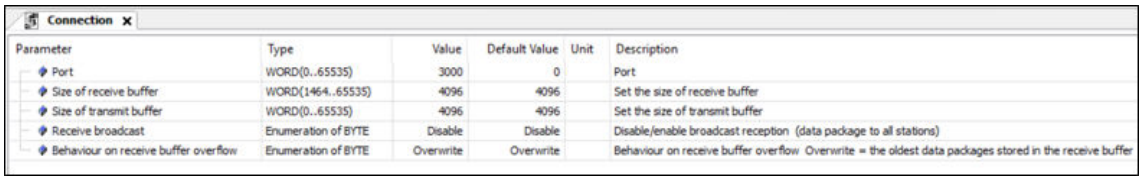
Configuration without communication interface modules to establish redundancy

Configuration of the HA system without communication interface modules to establish redundancy is done by the following steps (for details see the example documentations):

1. Install the hardware ↳ *Chapter 1.5.5.2.2.2 “Hardware, requirements and options overview” on page 2092*.
2. Select the CPUs based on the requirements ↳ *Chapter 1.5.5.2.2.2.1 “CPU choice, system size and performance indications” on page 2094*.
3. Install Automation Builder including the latest libraries ↳ *Chapter 1.5.5.2.2.4 “How to get and install the AC500 High Availability system package” on page 2107*.
4. Create a new project in Automation Builder for the chosen CPUs.
5. Configure the required Modbus and UDP configuration in the Automation Builder device tree of the CPU.



6. For UDP in AC500 V2 PLC, configure *“UDP_no_AC31_header”* and set the port number to value '3000'.

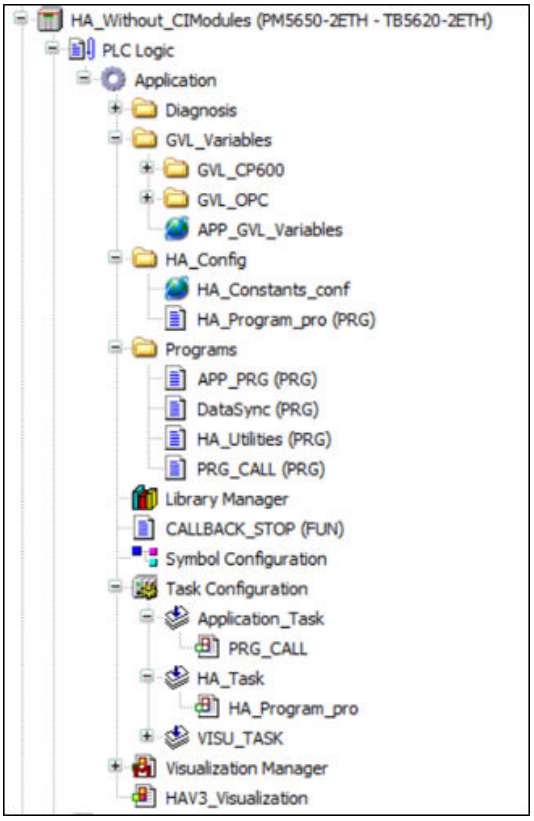


Connection

Parameter	Type	Value	Default Value	Unit	Description
Port	WORD(0..65535)	3000	0		Port
Size of receive buffer	WORD(1464..65535)	4096	4096		Set the size of receive buffer
Size of transmit buffer	WORD(0..65535)	4096	4096		Set the size of transmit buffer
Receive broadcast	Enumeration of BYTE	Disable	Disable		Disable/enable broadcast reception (data package to all stations)
Behaviour on receive buffer overflow	Enumeration of BYTE	Overwrite	Overwrite		Behaviour on receive buffer overflow Overwrite = the oldest data packages stored in the receive buffer

7. Assign the IP addresses in ≥ 2 different Ethernet networks:
   - SCADA network: SCADA, connected PLC A and PLC B
   - Field network: connected CI52x module(s)
8. Configure the mandatory HA_MOD_CONTROL function block for the HA task ↳ *“HA program” on page 2111*.
9. Add Callback stop function HA_MOD_CALLBACK_STOP and call it in the system event “stop”.
10. Add optional HA utility function blocks or function block HA_MOD_DATASYNC.

11. Make the global variable xNoCiBus = TRUE to run the system without communication interface module configured in the system. Refer to *Chapter 1.5.5.2.2.5.3.4 “HA Modbus system without communication interface modules in the network” on page 2118*.



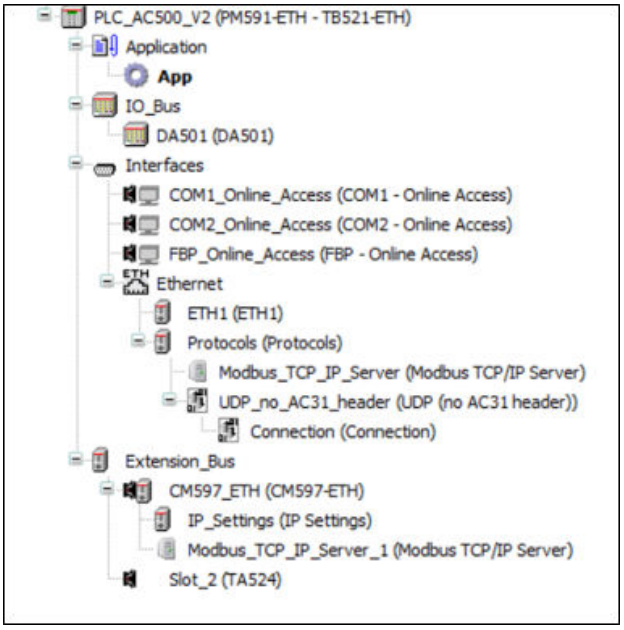
12. Add the task configuration *Chapter 1.5.5.2.2.5.2 “Task configuration recommendations for HA system” on page 2113*.
13. Compile and download to both PLCs (simplified in V3 via integrated download manager).
14. Create a boot project, restart the complete system and RUN.
15. Operation: Test use cases (e.g. by putting the primary PLC to STOP mode and observe the switchover). For different use cases and behavior refer to .
16. Runtime error and diagnosis function block can be used to monitor the system . For details refer to chapter Diagnosis *Chapter 1.5.5.2.2.8 “Diagnosis” on page 2124*.

Configuration with communication interface modules and redundancy

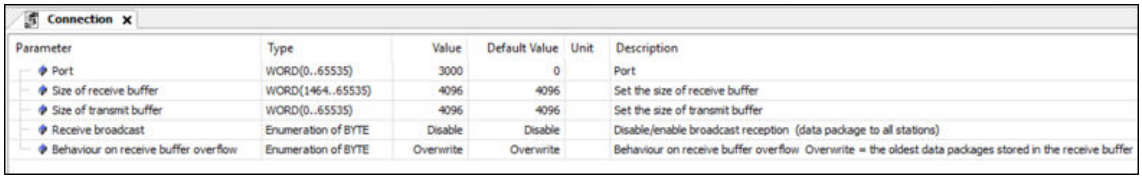
For medium or large HA systems the configuration with communication interface modules and redundancy is done by the following steps. For details see the example documentations:

1. Install the hardware *Chapter 1.5.5.2.2.2 “Hardware, requirements and options overview” on page 2092*.
2. Select the CPUs based on the requirements *Chapter 1.5.5.2.2.2.1 “CPU choice, system size and performance indications” on page 2094*.
3. Install Automation Builder including the latest libraries *Chapter 1.5.5.2.2.2 “Hardware, requirements and options overview” on page 2092*.
4. Install the Bulk Data Manager tool (BDM) *Chapter 1.5.5.2.2.4 “How to get and install the AC500 High Availability system package” on page 2107*.
5. Create a new project in Automation Builder for the chosen CPUs.

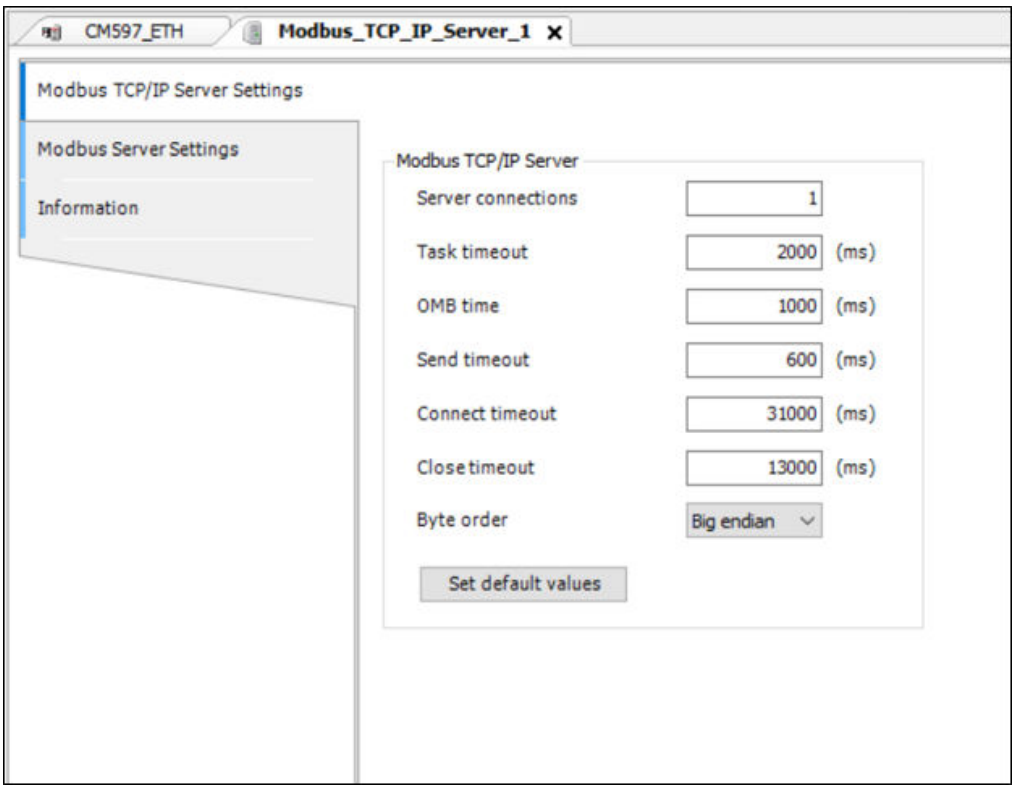
6. Configure the required Modbus and UDP configuration in the Automation Builderdevice tree of the CPU. UDP settings are only required in AC500 V2 PLCs.



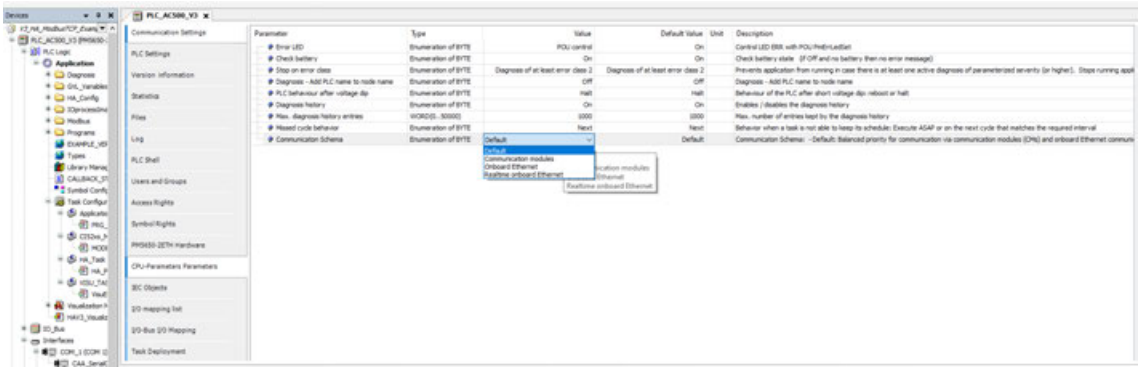
7. For UDP in AC500 V2 PLC, configure *"UDP_no_AC31_header"* and define the port number as '3000'.



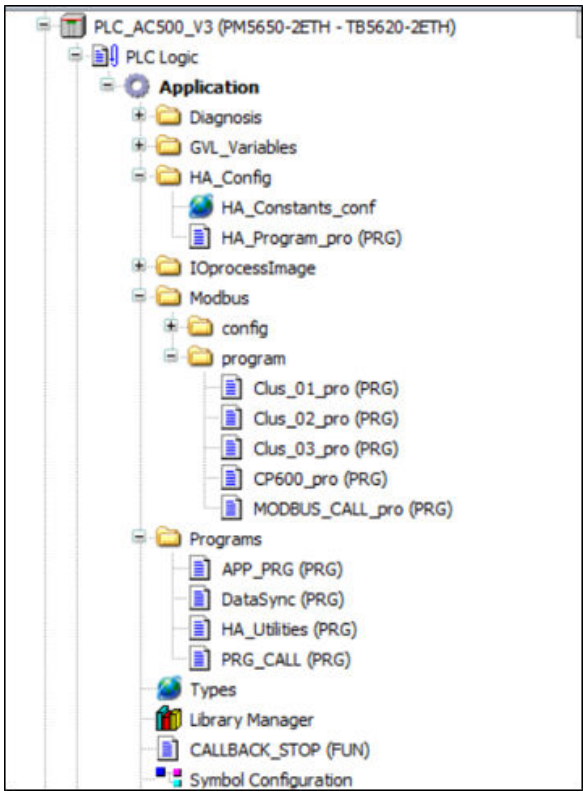
8. In AC500 V2 PLCs for each CM597-ETH communication module added the *"Send timeout"* value has to be changed to 600 ms for the Modbus TCP server.



9. Assign the IP addresses in ≥ 2 different Ethernet networks:
   - SCADA network: SCADA, connected PLC A and PLC B.
   - Field network: connected CI52x module(s).
10. Configure a network switch in the field network (if managed /redundant) based on network redundancy required ↳ *Chapter 1.5.5.2.2.5.3 “Field I/O network topologies” on page 2115*.
11. Run BDM tool to configure CI52x network.
12. Export the files. Refer for details in the document: *C:\Users\Public\Documents\AutomationBuilder\Examples\PS5601-HA-MTCP\BulkDataManager\Documentation*.
13. Import the Bulk data export files to the Automation Builder project.
14. Add Modbus TCP configuration for the ETH ports.
15. For the system with V3 PLCs, set the Communication Schema to “Onboard Ethernet” *“CPU-Parameters Parameters”* for better performance.



16. Add Callback stop function HA_MOD_CALLBACK_STOP and call it in the system event *“stop”*.
17. Add optional HA utility function blocks or function block HA_MOD_DATASYNC.



18. Add the task configuration ↳ *Chapter 1.5.5.2.2.5.2 “Task configuration recommendations for HA system” on page 2113*.
19. Compile and download to both PLCs (simplified in V3 via integrated download manager).

20. Create a boot project, restart the complete system and RUN.
21. Operation: Test use cases (e.g. by putting the primary PLC to STOP mode and observe the switchover).
22. For different use cases and behavior refer to *Chapter 1.5.5.2.2.3.1 “Failures and use cases” on page 2098*.
23. Runtime error and diagnosis function block can be used to monitor the system. For details refer to *Chapter 1.5.5.2.2.8 “Diagnosis” on page 2124*.

HA-Modbus TCP Limits

HA-Modbus TCP is supported as of Automation Builder 2.0 or higher and the corresponding AC500 CPUs mentioned previously. AC500 V3 PLC is currently not supporting external ETH communication modules. Therefore, onboard ETH1, ETH2 (and eventually CAN) ports are to be used for communication.

3000 sync instances can be used: Either 3000 HA_MOD_DATA_SYNC function block instances alone or together 3000 instances of HA_MOD_DATA_SYNC inclouding + HA utility function block can be used. If more than 3000 instances are configured user can see the error at xHaModDataErr = True and wHaModDataErNo = 16#2022 in HA_GLOBAL_VARIABLES.

The maximum length of sync data at an instance of HA_MOD_DATA_SYNC function block would be 1412 bytes. The maximum size of sync data which can be synced between PLCA and PLCB in total can be max. 60 000 bytes.

The HA-Modbus TCP system takes care of the first fault only. This fault must be visualized by the programmer and overall system (e.g. HMI, SCADA) to the operator, to plan and repair as soon as possible as redundancy might be lost. If more than one error occurs, system may not react to second or following faults.

SCADA/ HMI has to be configured/programmed to:

- Only read data from the primary PLC.
- Parameters and control data should be always written to both PLCs or has to be synchronized via the function block.

This is given automatically when using OPC DA, where the CODESYS OPC Server does this switching for the connected clients according to the primary status. For CP600 HMI a script is available to switch likewise (connected via the internal AC500 protocol or Modbus). Zenon as a SCADA also uses the AC500 protocol to automatically switchover.

Diagnosis

This chapter explains the diagnosis information available to the user in the HA Modbus library and CI52x library. Diagnosis information is available at the outputs of HA control function block, HA Diagnosis function block and at the CI52x function block.

Depending on the use case defined in *Chapter 1.5.5.2.2.3 “Functionality” on page 2098* different diagnosis information can be accessed.

Primary CPU currently can read-out the diagnosis information (CI52x function block outputs) from communication interface module only once, hence secondary PLC will not be able to read the diagnosis information from the CI52x module.

So if any change happens in CI52x diagnosis it is not reflected in the secondary CPU.

This can lead to different diagnosis information of CI52x module in the primary and the secondary CPU. Hence it is recommended to customers that diagnosis information should be handled in the application (e.g. SCADA).

Diagnosis in HA-Modbus TCP library

In the HA Modbus library diagnosis information is available at the control block and diagnosis block.

Output System Configuration

This output at the HA control block gives the information of system configuration. Each bit of the word represents a different configuration.

Bit	Description
0	Sync is configured via CAN
1	Sync is configured via UDP
2	Lifecom2 is configured via CAN
3	Lifecom2 is configured via UDP
4	Lifecom2 is configured via Modbus TCP
5	Initialization for Ethernet configuration

Output System Configuration error

This output at the HA control block gives the details of error in the configuration. Each bit of the word represents different configuration errors. It is valid only when Error = TRUE.

Bit	Description
0	Communication interface module is not configured properly
1	1< SyncSlot >3. Invalid value at input sync slot
2	1<SecSlot>3. Invalid value at input second slot
3	Value at IpAdrCpuASync is invalid
4	Value at IpAdrCpuBSync is invalid
5	Value at IpAdrCpuALifecom2 is invalid
6	Value at IpAdrCpuBLifecom2 is invalid
7	IpAdrCpuASync = IpAdrCpuBSync or IpAdrCpuALifecom2 = IpAdrCpuBLifecom2 The IP addresses assigned at sync or lifecom2 inputs are wrong

Output Runtime error

This output at the HA control block gives the details of the error during run time of the system. Each bit of the word represents different runtime errors. It will not set Error = FALSE.

Bit	Description
0	Communication interface modules are lost
1	Other CPU is not active
2	Lifecom1 is lost (part of sync)
3	Lifecom2 is lost. This error will not be TRUE if the PLC is in STOP status. This is because Modbus is still responding even when PLC is in STOP
4	Synchronization is lost
5	Error in synchronization
6	Ethernet status error

Bit	Description
7	Other PLC lost communication to CI52x modules
8	CAN_HEADER function block has error
9	CAN_DATA function block has error
10	fbGetOwnIP function block has error

Diagnosis function block

Outputs at the HA Diagnosis function block, HaModDiag (V3) / HA_MOD_DIAG (V2) provides the following diagnosis information of the HA system.

Output	Description
CpuAPrimary / CPUA_PRIMARY	TRUE indicates CPU A is primary
CpuBPrimary / CPUB_PRIMARY	TRUE indicates CPU B is primary
CpuARun / CPUA_RUN	TRUE means CPU A is in RUN mode
CpuBRun / CPUB_RUN	TRUE means CPU B is in RUN mode
CpuACI52xBusActive / CPUA_CI52x_BUS_ACTIVE	Modbus TCP CI52x bus active on CPU A
CpuBCI52xBusActive / CPUB_CI52x_BUS_ACTIVE	Modbus TCP CI52x bus active on CPU B
CpuACi52xCfg / CPUA_CI52x_CFG	Total number of CI52x configured on CPU A
CpuBCi52xCfg / CPUB_CI52x_CFG	Total number of CI52x configured on CPU B
CpuACi52xAct / CPUA_CI52x_ACT	Total number of CI52x active on CPU A line
CpuBCi52xAct / CPUB_CI52x_ACT	Total number of CI2x active on CPU B line
SyncInstances / SYNC_INSTANCES	Number of data sync and utility blocks initialized in the system
SyncDataCheckSum / SYNC_DATA_SUM	Checksum of all address pointer blocks in bytes, indicates total number of bytes getting synchronized.

Output	Description
StHACpuStatus / stHA_CPU_STATUS	HA own CPU status. It will show the status details of logged in CPU for the following parameters: ● HA1: CPU A is primary ● HA2: CPU B is primary ● bit_CI52x_BUS_active: CI52x bus with one or more communication interface modules active ● bit_CI52x_BUS_err: CI52x bus one or more communication interface modules powered off / connection lost ● RUN: Run status of CI52x ● cnt: Count of data sync communication, indicates data sync between CPUs is okay.
StHAotherCpuStatus / stHA_OTHER_CPU_STATUS	HA other CPU status. It will show the status details of other CPU for the following parameters: ● HA1: CPU A is primary ● HA2: CPU B is primary ● bit_CI52x_BUS_active: CI52x bus with one or more communication interface modules active ● bit_CI52x_BUS_err: CI52x bus one or more communication interface modules powered off / connection lost ● RUN: Run status of CI52x ● byETH_ACT_CI52x_Count: CI52x alive identification count.

Other diagnosis variables

Apart from the errors / diagnosis information available in the control and diagnosis block, few other variables can be monitored too.

Variable	Value	Description
wHA_ER_NO_SYNC_LINK	16#7487	No sync link between the PLCs
HA_MOD_INVALID_LENGTH	16#2017	Invalid length at the input of the data sync block
HA_MOD_ERNO_TBL_OVERFLOW	16#2022	HA data reference table is full
xHaModDataErr	TRUE	IF TRUE – HA data sync is in error state
wHaModDataErNo		HA data sync error code
xHaModErr	TRUE	HA system is in error state
dwHaModServerAlive		Life counter incremented by OPC DA server

Diagnosis in CI52x library

In addition to the diagnosis information in the HA Modbus library, additional diagnosis information for each communication interface module can be obtained from the CI52x library.

System Configuration error

This output at the CI52x function block gives the details of the configuration error in the CI52x module. Each bit of the byte represents different configuration errors:

Bit	Description
0	Reserved
1	Wrong ETH port is configured at input Config ETH
2	Wrong IP address is configured for communication interface module

Runtime error

RuntimeError (v3) / RUNTIME_ERROR (v2) of the function block CiModCi52x (v3) / CI_MOD_CI52x (v2). Runtime error is a combination of error bits that are described in the following:

Runtime Error	Description
Bit 0	Indicates communication error i.e., when CPU is not able to get any response from CI52x module. This error will get reset when communication is reestablished.
Bit 1	Indicates parameter state is not equal to 2 (PARA_STATE_PARA_DONE). If not true, then system gives I/O bus error. System resets this error when parameter state is equal to 2.
Bit 2	Indicates the cluster error [1]) in the system, if there is an error in the diagnosis buffer. ACK input is needed to reset this error.
Bit 3	Indicates the hardware configuration error, mismatch between configuration and actual hardware detected. System automatically resets this error when the hardware matches.
S-ERR (LED on communication interface module)	Indicates that there is some issue with channel configuration in the cluster [1]). It is not linked with Runtime Error. User can read DiagBuffer (v3) / DIAG_BUFFER (v2) from CiModDiag (v3)/ CI_MOD_DIAG (v2) function block to get more information. This error does not get reset using ACK. It will only reset when all channel errors are removed.
[1]) "Cluster" means a combination of one communication interface module and several I/O modules attached to it.	

Runtime error in different scenarios:

Error	Run-time error	PLC A: Primary				PLC B: Secondary			
		Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error	Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error
Wrong IP address configured	16#1	TRUE	FALSE	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE
Wrong slot address configured [1])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Communi-cation cable dis-connected	16#2	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
Wrong I/O module plugged in the CI module	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Wrong hotswap I/O module plugged at the start	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Wrong hotswap I/O module swapped online	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Configured I/O module not con-nected at start [2])	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
Configured hotswap I/O module not con-nected at start [2])	16#B	BLINK	TRUE	FALSE	TRUE	BLINK	TRUE	FALSE	TRUE
I/O module powered off in CI module [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Hotswap I/O module powered off in CI module [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE
Remove hotswap I/O module when online [2])	16#4	FALSE	FALSE	TRUE	FALSE	FALSE	FALSE	FALSE	FALSE

Error	Run-time error	PLC A: Primary				PLC B: Secondary			
		Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error	Bit0 - comm error	Bit1 - I/O bus error	Bit2 - cluster error	Bit3 - HW config error
CI module is powered off	16#2	TRUE	TRUE	FALSE	FALSE	TRUE	TRUE	FALSE	FALSE
Mismatch in Channel configura-tion and wiring [3])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
Regular I/O module mounted on hotswap terminal unit [4])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
During an error stage if HA system changeover is initiated [5])	16#0	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE

[1]) Slot input in the block can be ignored. Similar to ETH input of the ModMast blocks.

[2]) Error generated only in the primary PLC, to reset ACK input to be used.

[3]) No runtime error in function block. Module generates S-Err and ZP Blinks.

[4]) No runtime error in function block. Module generates S-Err.

[5]) Runtimer Error bit2 gets reset when the PLC is switched over and error won’t be available in any of the PLC regardless of its Primary status.

Communication interface diagnosis

Table 128: Function block CiModDiag (V3) and CI_MOD_DIAG (V2)

Output	Description
DevState / DEV_STATE	CI521 or CI522 device current status is displayed. ● STATE_PREOP: Device is booting ● STATE_OPERATION: Device is operational, no bus supervision is active ● STATE_ERROR: Device detected a bus error, bus supervision is active ● STATE_IP_ERROR: Device has an IP address error ● STATE_CYLIC_OPERATION: Device is operational, bus supervision is active ● STATE_NA: Not available
ParaState / PARA_STATE	CI521 or CI522 device parameter status. ● PARA_STATE_NO_PARA: Device has no parameters ● PARA_STATE_PARA_ACTIVE: Parameterization process is running ● PARA_STATE_PARA_DONE: Device used valid parameters and parameterization is done ● PARA_STATE_ERROR: Device has invalid parameters ● PARA_STATE_NA: Not available
DeviceInfo / DEVICE_INFO	CI521 or CI522 type and extended module types. This will give the details of the module configured in the communication interface module including the I/O modules. If module is with suffix F, then fast counter is enabled for that module.
DiagBuffer / DIAG_BUFFER	CI521 or CI522 module diagnosis buffer. Refer to *Chapter 1.6.4.3.1.2.3.2 “Diagnosis data” on page 5659*.
ErClass / ERR_CLASS	Communication interface error class. Refer to *Chapter 1.7.3 “Diagnosis messages” on page 6429*
ErNo / ERR_NO	Communication interface error number. Refer to *Chapter 1.7.3 “Diagnosis messages” on page 6429*
ModMastErr / MODMAST_ERR	Latest 22 Modbus TCP error message status of the ModMastTcp (V2) / COM_MOD_MAST (V2) function block.
ModMastErNo / MODMAST_ERR_NO	Latest 22 Modbus TCP error numbers. Refer to the error details in Modbus library *Chapter 1.5.5.2.2.8.1 “Diagnosis in HA-Modbus TCP library” on page 2125*. V2: Refer to the error messages related to *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*

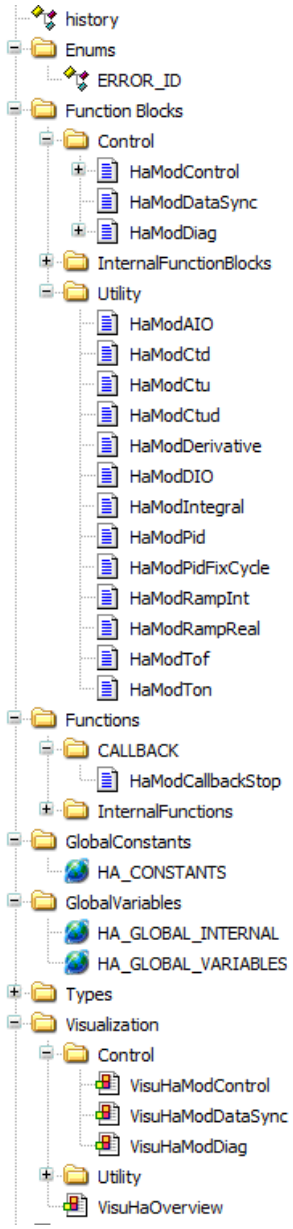
Library overview

Documentation

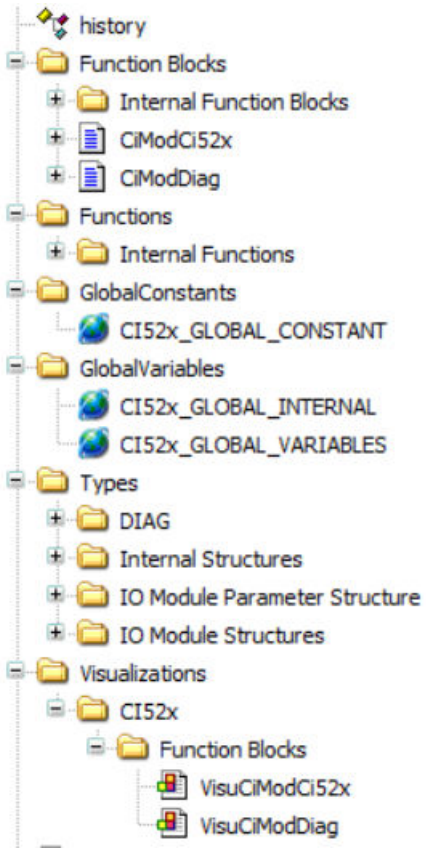
CODESYS V2 libraries are described in a separate library documentation: *Chapter 1.5.5.2.3 “HA-Modbus TCP V2 library function block description” on page 2133*.

The following function blocks are contained in the libraries:

HA-Modbus TCP library



HA_CI52x library



1.5.5.2.3 HA-Modbus TCP V2 library function block description

Scope and structure of this document

The purpose of this libraries description is to explain different components of HA-Modbus TCP and CI52x library.

The libraries description is valid for AC500 V2 products (using CODESYS V2 libraries).

For AC500 V3 products (using CODESYS V3 libraries) the library description is integrated in the Library Manager of Automation Builder.

HA-Modbus TCP library

Components of HA-Modbus TCP library

The HA-Modbus TCP library contains the following function, function blocks, structures, visualizations, constants and variables.

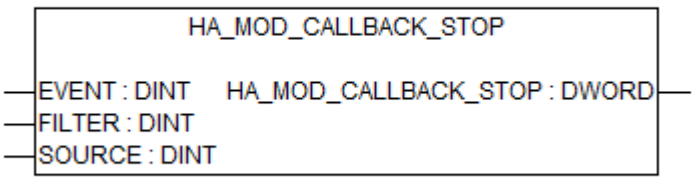
- Function
  - HA_MOD_CALLBACK_STOP
- Function blocks

  Main:
  - HA_MOD_DATA_SYNC
  - HA_MOD_CONTROL
  - HA_MOD_DIAG

- Function blocks
  Utility:
  – HA_MOD_AIO
  – HA_MOD_CTD
  – HA_MOD_CTU
  – HA_MOD_CTUD
  – HA_MOD_DERIVATIVE
  – HA_MOD_DIO
  – HA_MOD_INTEGRAL
  – HA_MOD_PID
  – HA_MOD_PID_FIXCYCLE
  – HA_MOD_RAMP_INT
  – HA_MOD_RAMP_REAL
  – HA_MOD_TOF
  – HA_MOD_TON
- Visualizations
  – HA_MOD_DATA_SYNC_VISU_PH
  – HA_MOD_CONTROL_VISU_PH
  – HA_MOD_DIAG_VISU_PH
  – HA_MOD_AIO_VISU_PH
  – HA_MOD_CTD_VISU_PH
  – HA_MOD_CTU_VISU_PH
  – HA_MOD_CTUD_VISU_PH
  – HA_MOD_DERIVATIVE_VISU_PH
  – HA_MOD_DIO_VISU_PH
  – HA_MOD_INTEGRAL_VISU_PH
  – HA_MOD_PID_VISU_PH
  – HA_MOD_PID_FIXCYCLE_VISU_PH
  – HA_MOD_RAMP_INT_VISU_PH
  – HA_MOD_RAMP_REAL_VISU_PH
  – HA_MOD_TOF_VISU_PH
  – HA_MOD_TON_VISU_PH
  – HA_OVERVIEW_VISU
- Global variables
  – HA_GLOBAL (constant)
  – HA_GLOBAL_VARIABLES
- Structures

Function

HA_MOD_CALLBACK_STOP



Parameter	Value
Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Event Function, Return DWORD

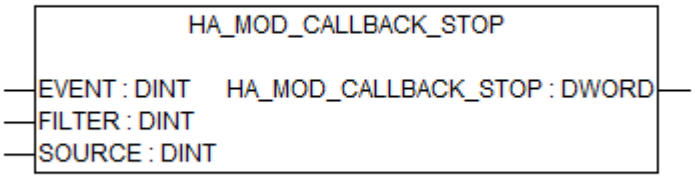
HA_MOD_CALLBACK_STOP function is programmed to process logic related to AC500 High Availability project in case of CPU switching into stop mode.

If there is a STOP event initiated in the PLC, HA_MOD_CALLBACK_STOP function is processed before the CPU switches to STOP state. This function sets/resets some HA library global variables, to provide the CPU's STOP information to the other CPU.

This function needs to be called under runtime system event. For further information on how to call the call back stop function in system events please refer to example project and documentation of the library.

- *The Callback stop event should follow the name pattern "CALLBACK_xxx".*
  - *If a user has downloaded the program to the PLC with a wrong callback stop event name then the CPU freezes and is unable to log in/on to the PLC using Ethernet cable.*
  - *User has to download a blank project using RS232 communication cable (TK501/TK502/TK503) in order to return the PLC into healthy stage.*
  - *After this, the user has to correct the callback stop event name as recommended and restart the download*
- *When a PLC type is changed in Automation Builder project, assigning of callback stop to "Stop event" in the System events, checkbox stop gets unchecked.*
  The user has to select the stop event again manually.

Input description



EVENT

Data type	Default value	Range	Unit
DINT	-	-	-

Runtime system event in which the function block is to be called.

FILTER

Data type	Default value	Range	Unit
DINT	-	-	-

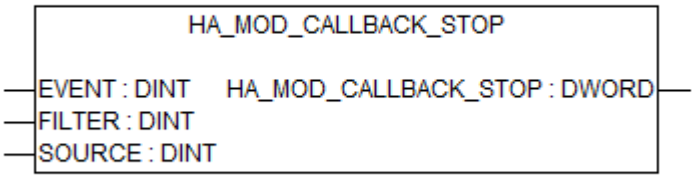
Runtime system event filter.

OWNER

Data type	Default value	Range	Unit
DINT	-	-	-

Runtime event system source.

Output description

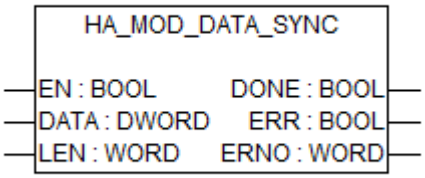


HA_MOD_CALL-BACK_STOP

Data type	Default value	Range	Unit
DWORD	-	-	-

Function blocks

HA_MOD_DATA_SYNC



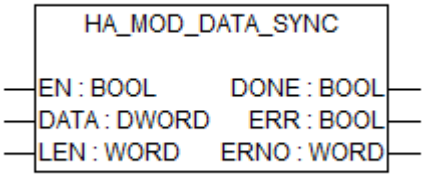
Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_DATA_SYNC is used for synchronizing different instances of function blocks, or any other data that need to be synchronized, in the HA application.

It collects data and size information from the connected function block and delivers details to the HA_MOD_CONTROL function block.

User can use up to 1024 instances of HA_MOD_DATA_SYNC function block in one program. This number depends on the number of High Availability utility function blocks used. The utility function blocks are synchronizing relevant data automatically.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

DATA

Data type	Default value	Range	Unit
DWORD	0	0 ... 255	-

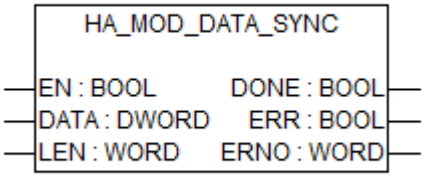
Function block instance input (via ADR-operator) for pointer data.

LEN (length)

Data type	Default value	Range	Unit
WORD	0	0 ... 255	-

Function block instance input (via SIZEOF-operator) for size, maximum 1412 bytes.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

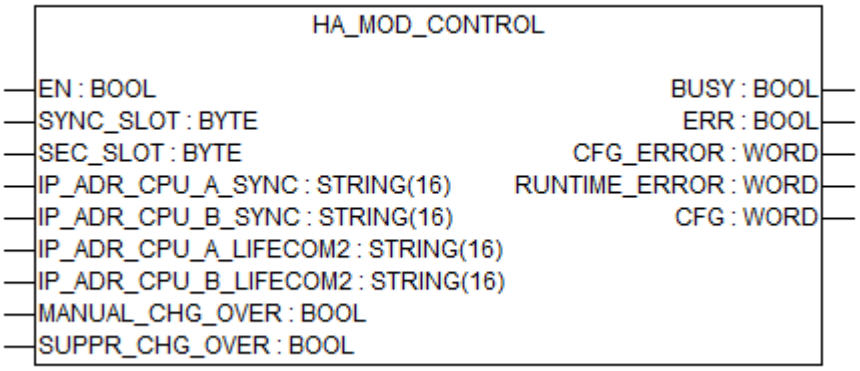
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

HA_MOD_CONTROL

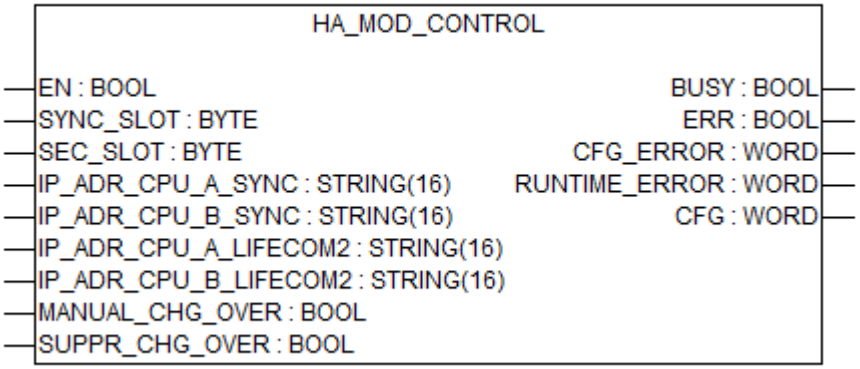


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

HA_MOD_CONTROL function block handles the AC500 High Availability operation such as change over from primary to secondary CPU in case of an error with related diagnostics.

Further it is used to data transfer between high availability CPUs. This is a mandatory function block for HA application.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SYNC_SLOT

Data type	Default value	Range	Unit
BYTE	0	1 ... 61	-

Slot number on which Sync Communication is configured. Possible 1, 2, 3, 4, 11, 21, 31, 41, 51, 61 dependent on CPU type and communication module position of ETH port.

SEC_SLOT

Data type	Default value	Range	Unit
BYTE	0	1 ... 61	-

Slot number on which second communication LifeCom2 is configured. Possible 1, 2, 3, 4, 11, 21, 31, 41, 51, 61 dependent on CPU type and communication module position of ETH port.

IP_ADR_CPU_A_SYNC

Data type	Default value	Range	Unit
STRING(16)	‘0.0.0.0’	-	-

IP Address (on which SYNC communication is configured) of AC500 CPU connected to PLC A.

IP_ADR_CPU_B_SYNC

Data type	Default value	Range	Unit
STRING(16)	‘0.0.0.0’	-	-

IP Address (on which SYNC Communication is configured) of AC500 CPU connected to PLC B.

IP_ADR_CPU_A_LIFECOM2

Data type	Default value	Range	Unit
STRING(16)	‘0.0.0.0’	-	-

IP Address (on which LifeCom2 Communication is configured) of AC500 CPU connected to PLC A.

IP_ADR_CPU_B_LIFECOM2

Data type	Default value	Range	Unit
STRING(16)	'0.0.0.0'	-	-

IP Address (on which LifeCom2 Communication is configured) of AC500 CPU connected to PLC B.

MANUAL_CHG_OVER

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

TRUE- Manual changeover from primary to secondary PLC.

SUPPR_CHG_OVER

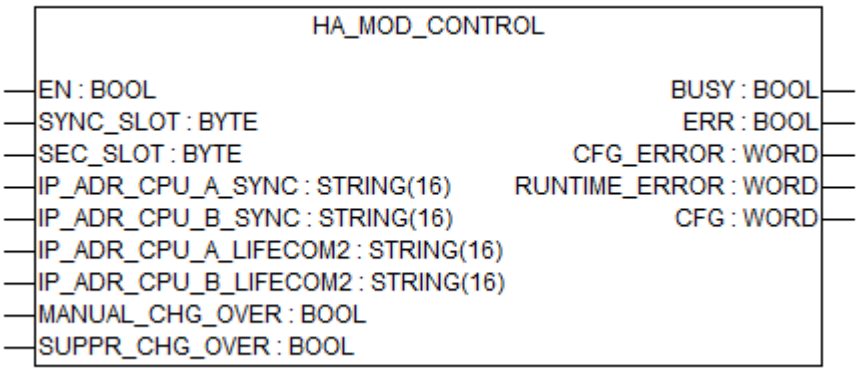
Data type	Default value	Range	Unit
BOOL	FALSE	-	-

TRUE- Suppress changeover (from primary to secondary PLC)

If used this input should be activated in both the CPU at the same time else system may behave randomly.

> *Suppress changeover feature is recommended to use only during network reconfiguration and not during any other switchover scenarios for example PLC STOP.*

Output description



BUSY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Error occurred during execution when output ERR = TRUE.

CFG_ERROR

Data type	Default value	Range	Unit
WORD	0	-	-

Each bit of the WORD represents different configuration errors:

Bit0 - *CI module* configuration mismatch,

Bit1 - *SYNC_SLOT* invalid,

Bit2 - *SEC_SLOT* invalid,

Bit3 - *IP_ADR_CPU_A_SYNC* invalid,

Bit4 - *IP_ADR_CPU_B_SYNC* invalid,

Bit5 - *IP_ADR_CPU_A_LIFECOM2* invalid,

Bit6 - *IP_ADR_CPU_B_LIFECOM2* invalid,

Bit7 - *IP_ADR_CPU_A_SYNC* = *IP_ADR_CPU_B_SYNC* or *IP_ADR_CPU_A_LIFECOM2*= *IP_ADR_CPU_B_LIFECOM2*.

CFG_ERROR is valid only when ERR = TRUE

RUN-TIME_ERROR

Data type	Default value	Range	Unit
WORD	0	-	-

Each bit of the WORD represents different Runtime errors:

Bit0 - CI module lost,

Bit1 - Other CPU not active,

Bit2 - Lifecom1 is lost (part of Sync),

Bit3 - Lifecom2 is lost,

Bit4 - Sync lost,

Bit5 - Error in Sync,

Bit6 - Ethernet status,

Bit7- Other PLC Lost communication to CI52x,

Bit8- Not used,

Bit9- Not used,

Bit10- Error reading own IP address.

RUNTIME Errors may occur during operation, but they will not generate ERR output = TRUE. This allows the system to run and revert to an error free state.

Bit3 - Lifecom2 is lost in RUNTIME ERROR will not be TRUE if the PLC is in STOP status. This is because Modbus is still responding even when PLC is in STOP.

CFG

Data type	Default value	Range	Unit
WORD	0	-	-

Bit0 - Not used,

Bit1 - Sync via UDP enabled,

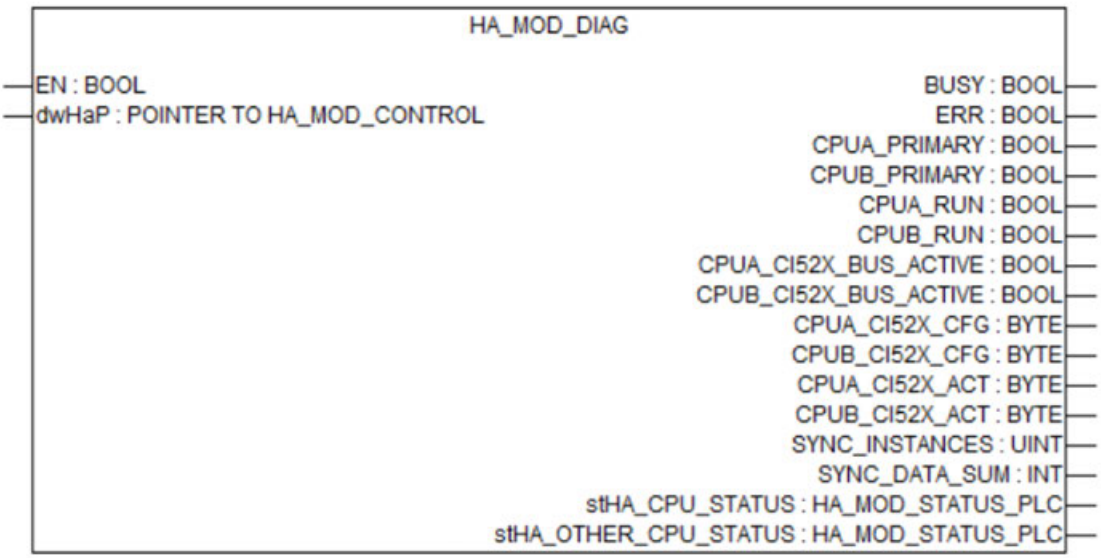
Bit2 - Not used,

Bit3 - Not used,

Bit4 - LifeCom2 via Modbus TCP is enabled,

Bit5 - Initialization for Ethernet configuration.

HA_MOD_DIAG

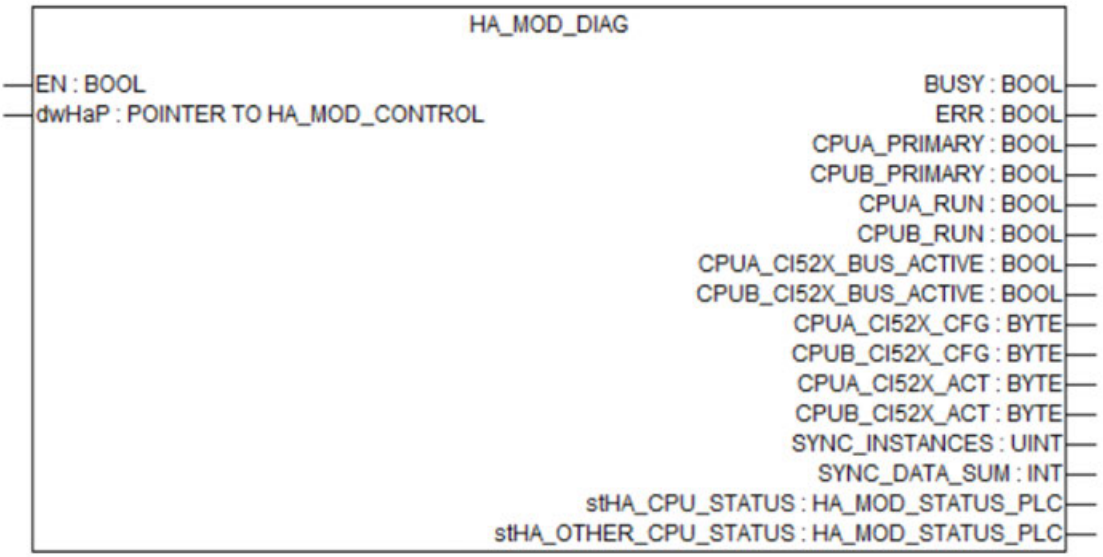


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

HA_MOD_DIAG function block provides the diagnosis information other than the configuration and runtime errors displayed in the HA_MOD_CONTROL function block. Using this function block, user will be able to get the diagnosis information such as primary status, active and configured CI modules, CI communication status, sync instances and number of sync bytes configured.

This function block must be connected to HA_MOD_CONTROL function block instance called for the system using ADR operator to populate the diagnosis information.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

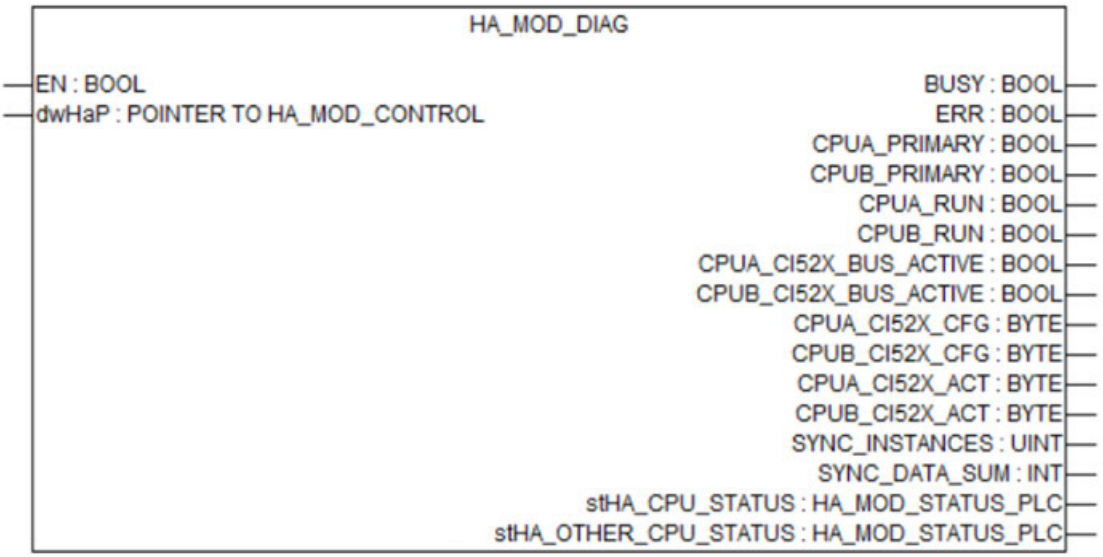
While it is executed its inputs are continuously evaluated.

dwHaP

Data type	Default value	Range	Unit
POINTER TO HA_MOD_CONTROL	-	-	-

This input points to the address of the control block instance from which the diagnosis information has to be read. Use the ADR operator and connect the instance of HA_MOD_CONTROL function block.

Output description



BUSY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Error occurred during execution when output ERR = TRUE.

CPUA_ PRIMARY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

CPU A primary status.

CPUB_ PRIMARY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

CPU B primary status.

CPUA_RUN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

CPU A RUN status.

CPUB_RUN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

CPU B RUN status.

CPUA_CI52X_BUS_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Modbus TCP CI52x – Bus active on CPU A.

CPUB_CI52X_BUS_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Modbus TCP CI52x – Bus active on CPU B.

CPUA_CI52X_CFG

Data type	Default value	Range	Unit
BYTE	0	-	-

Number of CI52x configured on CPU A.

CPUB_CI52X_CFG

Data type	Default value	Range	Unit
BYTE	0	-	-

Number of CI52x configured on CPU B.

CPUA_CI52X_ACT

Data type	Default value	Range	Unit
BYTE	0	-	-

Number of CI52x active on CPU A.

CPUB_CI52X_ACT

Data type	Default value	Range	Unit
BYTE	0	-	-

Number of CI52x active on CPU B.

SYNC_INSTANCES

Data type	Default value	Range	Unit
UINT	0	-	-

Number of data sync and utility blocks initialized in project.

SYNC_DATA_SUM

Data type	Default value	Range	Unit
INT	0	-	-

Total number of sync data in bytes.

stHA_CPU_STATUS

Data type	Default value	Range	Unit
HA_MOD_STATUS_PLC	-	-	-

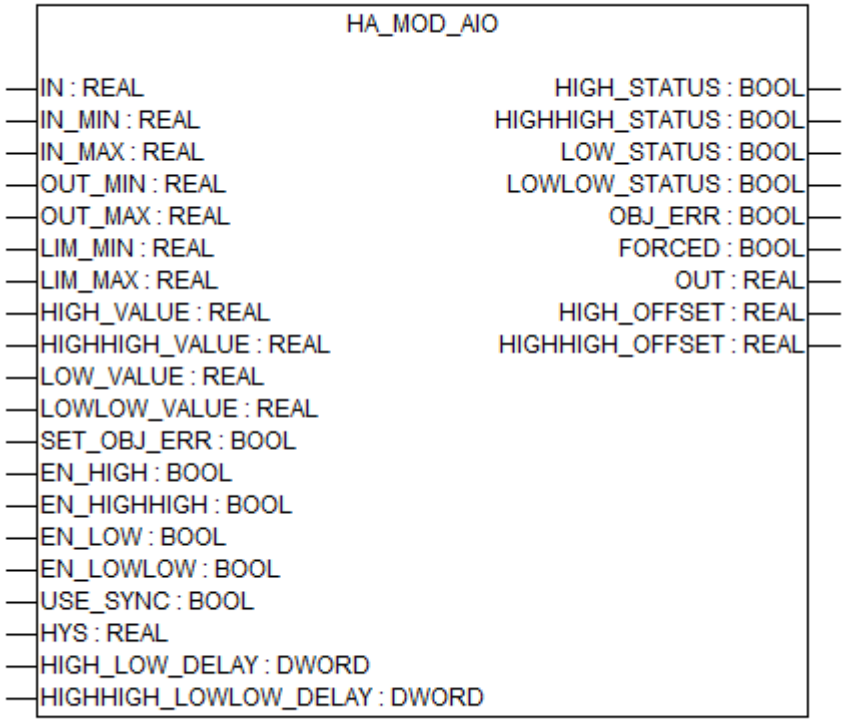
HA own CPU status. It will show the status details of logged in CPU.

stHA_OTHER_CPU_STATUS

Data type	Default value	Range	Unit
HA_MOD_STATUS_PLC	-	-	-

HA other CPU status. It will show the status of another CPU.

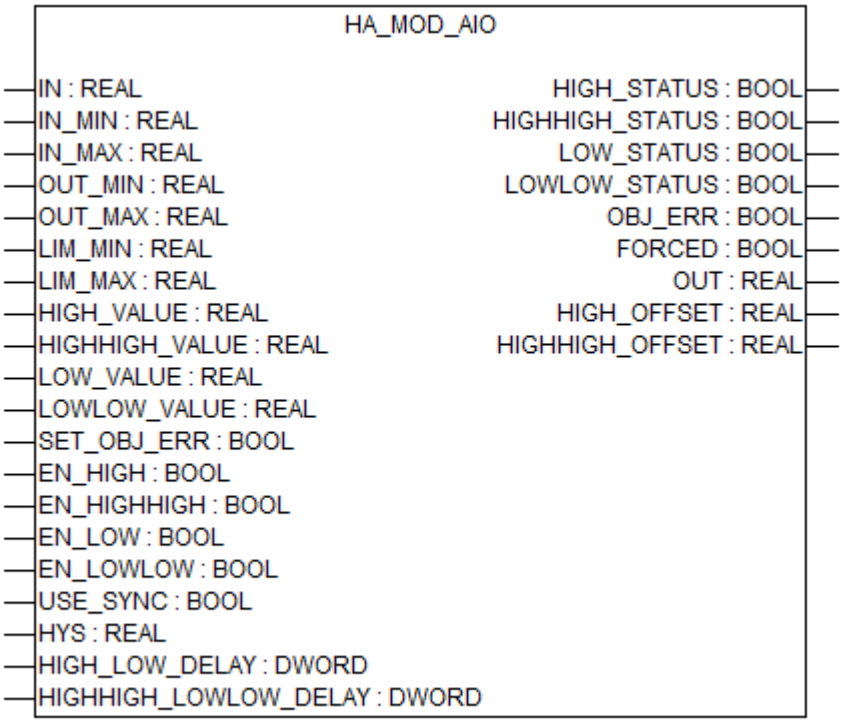
HA_MOD_AIO



Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

This function block can add in a standardized way scaling, limiting, alarming and other functions as often available in DCS type environment for analog inputs and outputs.

Input description



IN

Data type	Default value	Range	Unit
REAL	0	-	-

Signal input.

IN_MIN

Data type	Default value	Range	Unit
REAL	0	-	-

Lower limit of input range of values.

IN_MAX

Data type	Default value	Range	Unit
REAL	0	-	-

Upper limit of input range of values.

OUT_MIN

Data type	Default value	Range	Unit
REAL	0	-	-

Lower limit of output value range.

OUT_MAX

Data type	Default value	Range	Unit
REAL	0	-	-

Upper limit of output value range.

LIM_MIN

Data type	Default value	Range	Unit
REAL	0	-	-

Signal minimum range.

LIM_MAX

Data type	Default value	Range	Unit
REAL	0	-	-

Signal maximum range.

HIGH_VALUE

Data type	Default value	Range	Unit
REAL	0	-	-

Value for High alarm event.

HIGH_HIGH_VALUE

Data type	Default value	Range	Unit
REAL	0	-	-

Value for High high alarm event.

LOW_VALUE

Data type	Default value	Range	Unit
REAL	0	-	-

Value for Low alarm event.

LOW_LOW_VALUE

Data type	Default value	Range	Unit
REAL	0	-	-

Value for Low low alarm event.

SET_OBJ_ERROR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Set object error

EN_HIGH

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Enable High alarm event.

EN_HIGHHIGH

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Enable High high alarm event.

EN_LOW

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Enable Low alarm event.

EN_LOWLOW

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Enable Low low alarm event.

USE_SYNC

Data type	Default value	Range	Unit
BOOL	TRUE	-	-

Use HA sync.

HYS

Data type	Default value	Range	Unit
REAL	0	-	-

Hysteresis alarm event.

HIGH_LOW_ DELAY

Data type	Default value	Range	Unit
DWORD	-	-	-

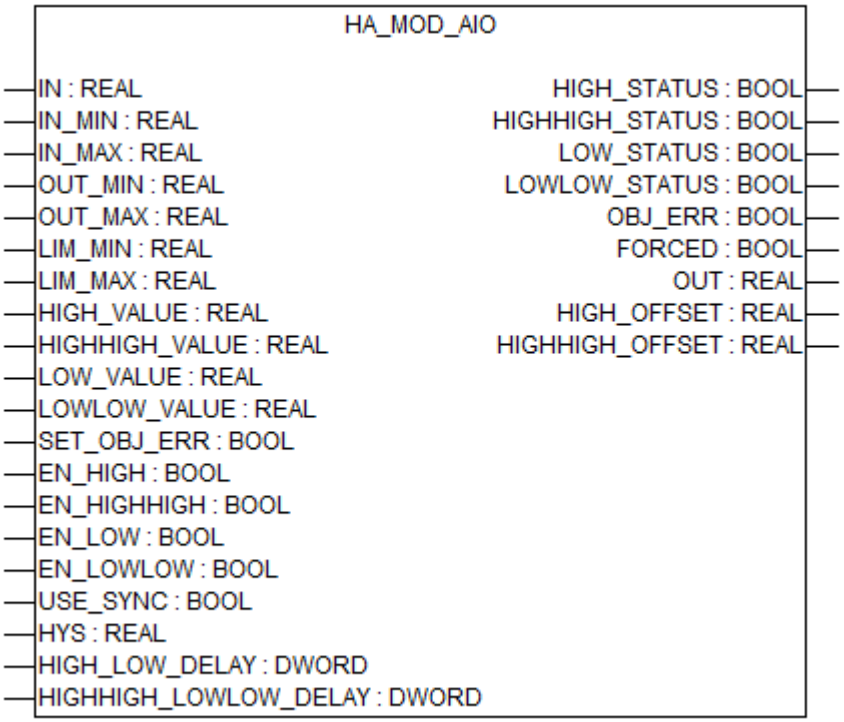
High & Low On delay [ms].

HIGHHIGH_ LOWLOW_ DELAY

Data type	Default value	Range	Unit
DWORD	0	-	-

High High & Low Low On delay [ms].

Output description



HIGH_STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

High alarm active.

HIGH-HIGH_STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

High high alarm active.

LOW_STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Low alarm active.

LOWLOW_STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Low low alarm active.

OBJ_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal object error.

FORCED

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Simulation ON.

OUT

Data type	Default value	Range	Unit
REAL	0	-	-

Signal out.

HIGH_OFFSET

Data type	Default value	Range	Unit
REAL	0	-	-

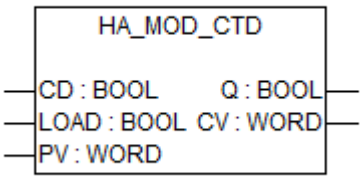
High offset.

HIGH-HIGH_OFFSET

Data type	Default value	Range	Unit
REAL	0	-	-

High high offset.

HA_MOD_CTD

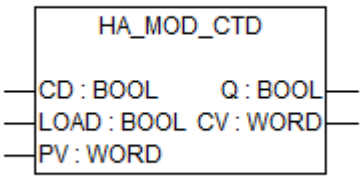


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_CTD is a standard count down counter with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



CD

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Count down on rising edge.

LOAD

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

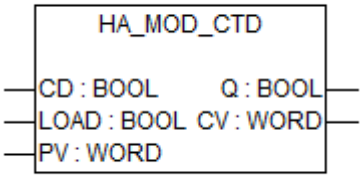
Load start value.

PV

Data type	Default value	Range	Unit
WORD	0	-	-

Start value

Output description



Q

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

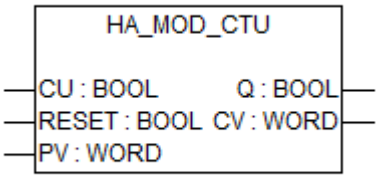
Counter reached 0.

CV

Data type	Default value	Range	Unit
WORD	0	-	-

Current counter value.

HA_MOD_CTU

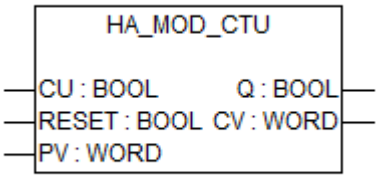


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_CTU is a standard count up counter with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



CU

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Count up.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

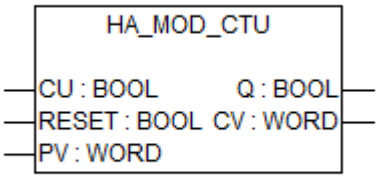
Reset counter to 0.

PV

Data type	Default value	Range	Unit
WORD	0	-	-

Counter limit

Output description



Q

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

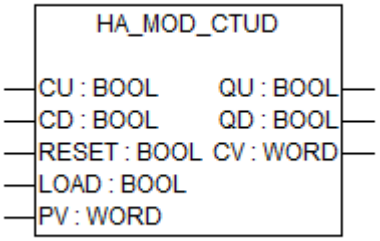
Counter reached the limit.

CV

Data type	Default value	Range	Unit
WORD	0	-	-

Current counter value.

HA_MOD_CTUD



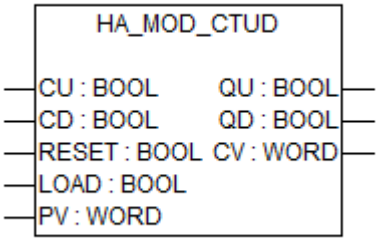
Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_CTUD is a standard count up and count down counter with automatic data synchronization in a high availability system.

QD is TRUE, if counter is 0. QU is TRUE, if counter is PV.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



CU

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Count up.

CD

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Count down.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Reset counter to 0.

LOAD

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

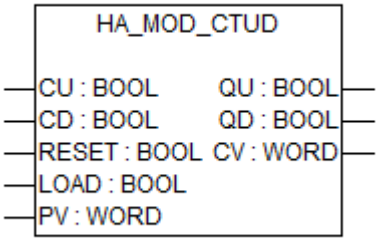
Load start value.

PV

Data type	Default value	Range	Unit
WORD	0	-	-

Start value / counter limit.

Output description



QU

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Counter reached limit.

QD

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

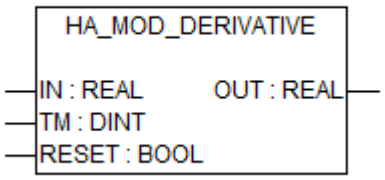
Counter reached 0.

CV

Data type	Default value	Range	Unit
WORD	0	-	-

Current counter value.

HA_MOD_DERIVATIVE

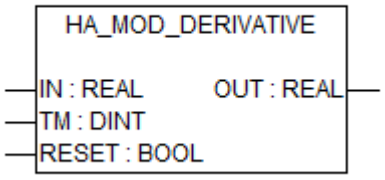


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_DERIVATIVE is a standard derivative with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



IN

Data type	Default value	Range	Unit
REAL	0	-	-

Input variable.

TM

Data type	Default value	Range	Unit
DINT	0	-	-

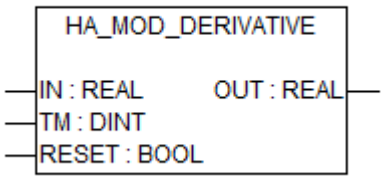
Time since last call [ms].

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

OUT is set to zero.

Output description

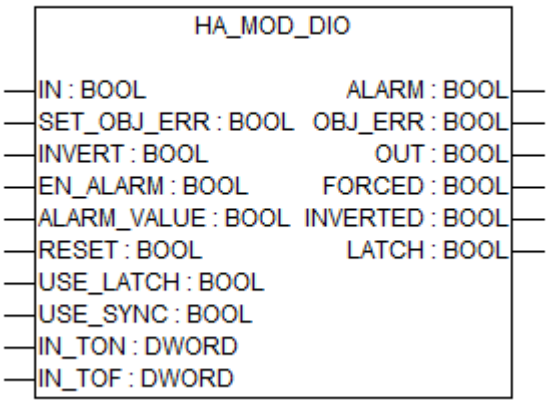


OUT

Data type	Default value	Range	Unit
REAL	0	-	-

Value of the derivative.

HA_MOD_DIO

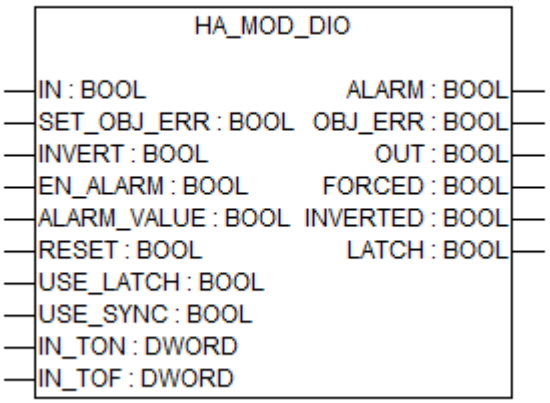


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

This function block can add in a standardized way event handling, alarming and other functions as often available in DCS type environment for digital in- and outputs.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



IN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal input.

SET_OBJ_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Set object error.

INVERT

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal inversion.

EN_ALARM

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal alarm configuration enabled.

ALARM_VALUE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal alarm configuration, alarm value 0 / 1 active.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Reset latched state.

USE_LATCH

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Automatically reset latched state.

USE_SYNC

Data type	Default value	Range	Unit
BOOL	TRUE	-	-

HA sync is used.

IN_TON

Data type	Default value	Range	Unit
DWORD	0	-	-

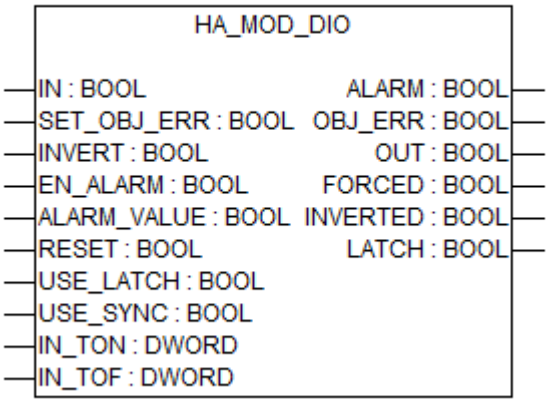
Signal input switch ON delay [ms].

IN_TOF

Data type	Default value	Range	Unit
DWORD	0	-	-

Signal input switch OFF delay [ms].

Output description



ALARM

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal alarm.

OBJ_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal object error.

OUT

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Signal out.

FORCED

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Simulation ON.

INVERTED

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

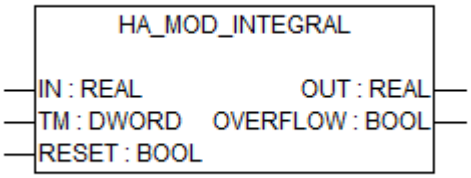
Signal inversion.

LATCH

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Latched output.

HA_MOD_INTEGRAL

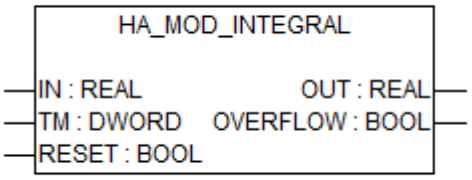


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_INTEGRAL is a standard integral with automatic data synchronization in a high availability system.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.*

Input description



IN

Data type	Default value	Range	Unit
REAL	0	-	-

Input variable.

TM

Data type	Default value	Range	Unit
DINT	0	-	-

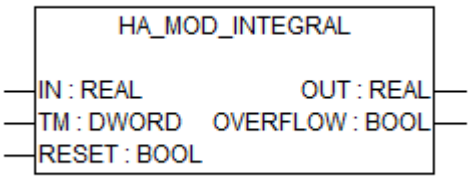
Time since last call [ms].

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

OUT is set to zero and OVERFLOW to false.

Output description



OUT

Data type	Default value	Range	Unit
REAL	0	-	-

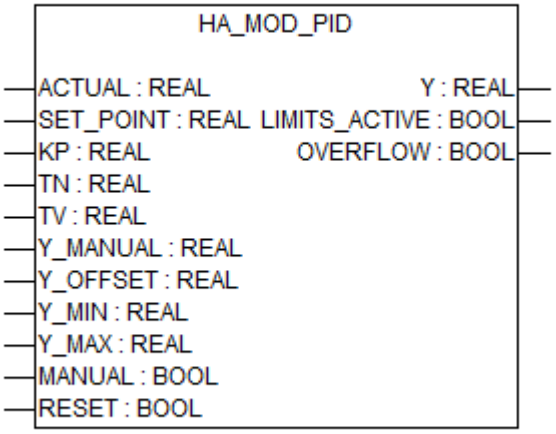
Value of the integral.

OVERFLOW

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Overflow.

HA_MOD_PID

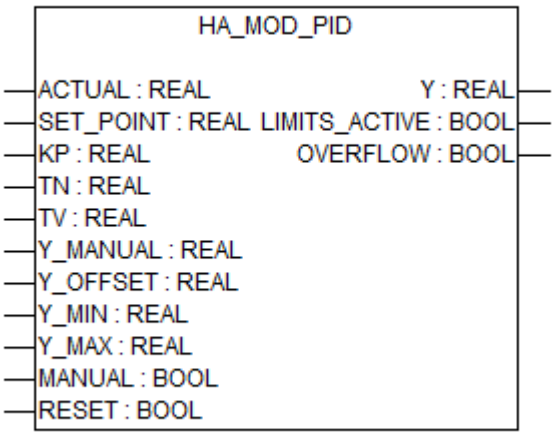


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_PID is a standard PID with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



ACTUAL

Data type	Default value	Range	Unit
REAL	0	-	-

Actual value, process variable.

SET_POINT

Data type	Default value	Range	Unit
REAL	0	-	-

Desired value, set point.

KP

Data type	Default value	Range	Unit
REAL	0	-	-

Coefficient of proportionality, unity gain of the P-part (P).

TN

Data type	Default value	Range	Unit
REAL	0	-	-

Reset time (I) [s].

TV

Data type	Default value	Range	Unit
REAL	0	-	-

Rate time, derivative time (D) [s].

Y_MANUAL

Data type	Default value	Range	Unit
REAL	0	-	-

Y is set to this value as long as MANUAL=TRUE.

Y_OFFSET

Data type	Default value	Range	Unit
REAL	0	-	-

Offset for manipulated variable.

Y_MIN

Data type	Default value	Range	Unit
REAL	0	-	-

Minimum value for manipulated variable.

Y_MAX

Data type	Default value	Range	Unit
REAL	0	-	-

Maximum value for manipulated variable.

MANUAL

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

TRUE: Y is not influenced by controller.

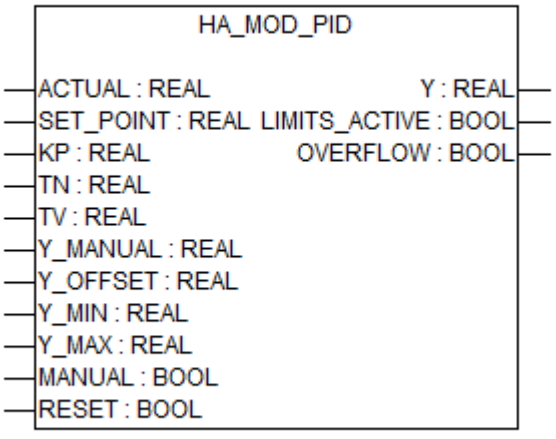
FALSE: controller determines Y.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Set Y output to Y_OFFSET, reset integral part.

Output description



Y

Data type	Default value	Range	Unit
REAL	0	-	-

Manipulated variable, set value.

LIMITS_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

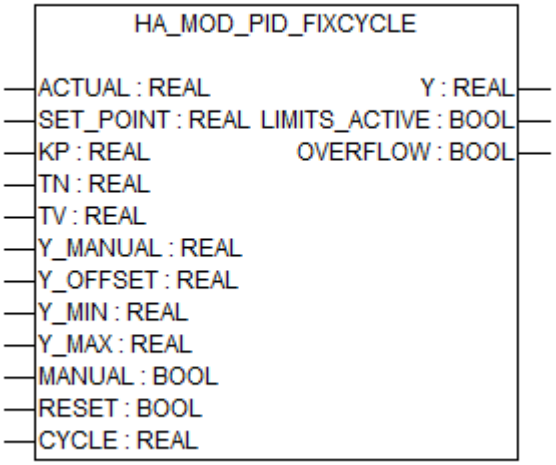
TRUE set value would exceed limits Y_MIN, Y_MAX.

OVERFLOW

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Overflow in integral part.

HA_MOD_PID_FIXCYCLE

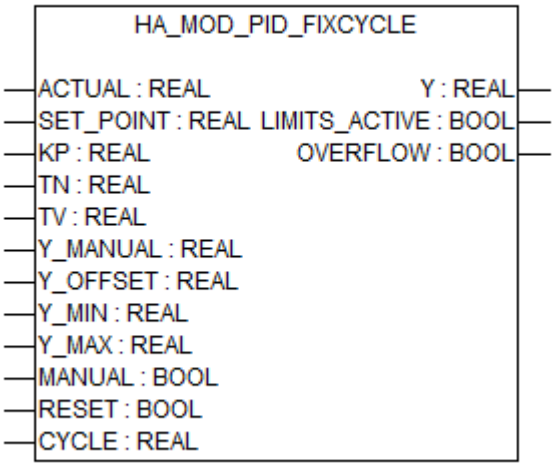


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_PID_FIXCYCLE is a standard PID in fix cycle with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



ACTUAL

Data type	Default value	Range	Unit
REAL	0	-	-

Actual value, process variable.

SET_POINT

Data type	Default value	Range	Unit
REAL	0	-	-

Desired value, set point.

KP

Data type	Default value	Range	Unit
REAL	0	-	-

Coefficient of proportionality, unity gain of the P-part (P).

TN

Data type	Default value	Range	Unit
REAL	0	-	-

Reset time (I) [s].

TV

Data type	Default value	Range	Unit
REAL	0	-	-

Rate time, derivative time (D) [s].

Y_MANUAL

Data type	Default value	Range	Unit
REAL	0	-	-

Y is set to this value as long as MANUAL=TRUE.

Y_OFFSET

Data type	Default value	Range	Unit
REAL	0	-	-

Offset for manipulated variable.

Y_MIN

Data type	Default value	Range	Unit
REAL	0	-	-

Minimum value for manipulated variable.

Y_MAX

Data type	Default value	Range	Unit
REAL	0	-	-

Maximum value for manipulated variable.

MANUAL

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

TRUE: Y is not influenced by controller.

FALSE: controller determines Y.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

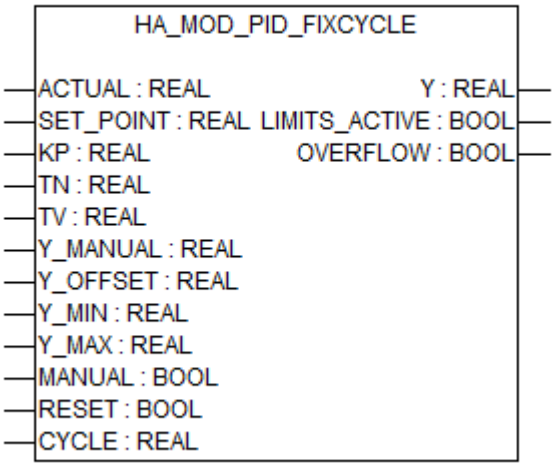
Set Y output to Y_OFFSET, reset integral part.

CYCLE

Data type	Default value	Range	Unit
REAL	0	-	-

Time in [s] between two calls.

Output description



Y

Data type	Default value	Range	Unit
REAL	0	-	-

Manipulated variable, set value.

LIMITS_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

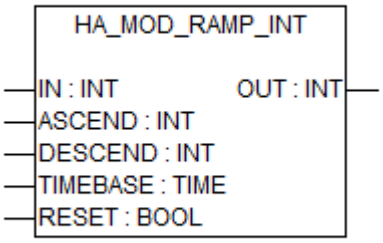
TRUE set value would exceed limits Y_MIN, Y_MAX.

OVERFLOW

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Overflow in integral part.

HA_MOD_RAMP_INT

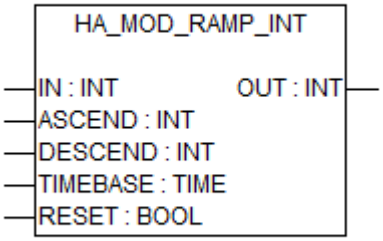


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_RAMP_INT is a standard integer ramp generator with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



IN

Data type	Default value	Range	Unit
INT	0	-	-

Input variable.

ASCEND

Data type	Default value	Range	Unit
INT	0	-	-

Maximum positive slope.

DESCEND

Data type	Default value	Range	Unit
INT	0	-	-

Maximum negative slope (non-negative!).

TIMEBASE

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Reference for ASCEND / DESCEND : t#0s : ASCEND / DESCEND defined per call

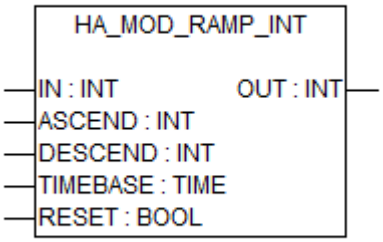
else : ASCEND / DESCEND defined per specified time.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Resets the OUT value.

Output description

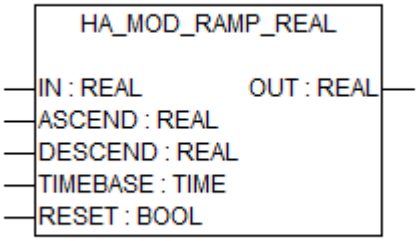


OUT

Data type	Default value	Range	Unit
INT	0	-	-

Value of function with limited slope.

HA_MOD_RAMP_REAL

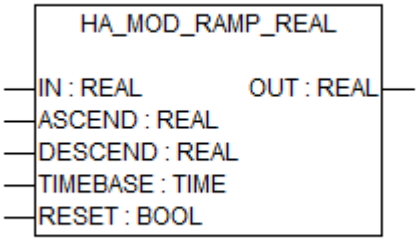


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_RAMP_REAL is a standard real ramp generator with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



IN

Data type	Default value	Range	Unit
REAL	0	-	-

Input variable.

ASCEND

Data type	Default value	Range	Unit
REAL	0	-	-

Maximum positive slope.

DESCEND

Data type	Default value	Range	Unit
REAL	0	-	-

Maximum negative slope (non-negative!).

TIMEBASE

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Reference for ASCEND / DESCEND : t#0s : ASCEND / DESCEND defined per call

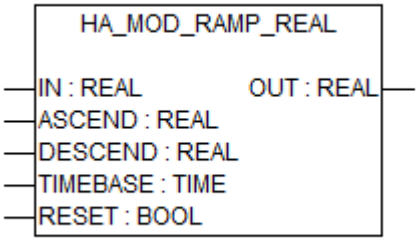
else : ASCEND / DESCEND defined per specified time.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Resets the OUT value.

Output description

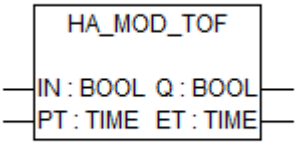


OUT

Data type	Default value	Range	Unit
REAL	0	-	-

Value of function with limited slope.

HA_MOD_TOF

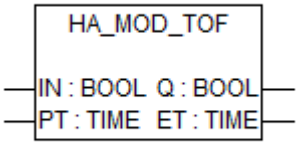


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_TOF is a standard off delay timer with automatic data synchronization in a high availability system.

> *Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.*

Input description



IN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Starts timer with falling edge.

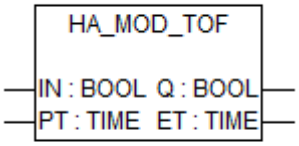
Resets timer with rising edge.

PT

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Time to pass, before Q is set.

Output description



Q

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

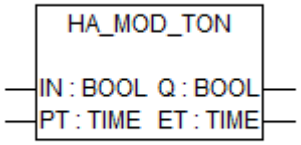
Is FALSE, PT seconds after IN had a falling edge.

ET

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Elapsed time.

HA_MOD_TON

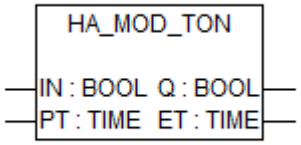


Available as of firmware	V2.6
Included in library	HaModbus_AC500_V26.lib
Type	Function block with historical values

Function block HA_MOD_TON is a standard on delay timer with automatic data synchronization in a high availability system.

Only internal variables and outputs are synchronized. Input variables and parameters are not synchronized. If needed, use HA_MOD_DATA_SYNC.

Input description



IN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Starts timer with rising edge.

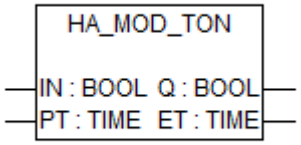
Resets timer with falling edge.

PT

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Time to pass, before Q is set.

Output description



Q

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Is TRUE, PT seconds after IN had a rising edge.

ET

Data type	Default value	Range	Unit
TIME	TIME#0ms	-	-

Elapsed time.

Visualization

In the application program, the user can add visualization objects in his project. In the following chapter, overall visualization and visualization for each individual function block are discussed in detail.

HA_OVERVIEW_VISU

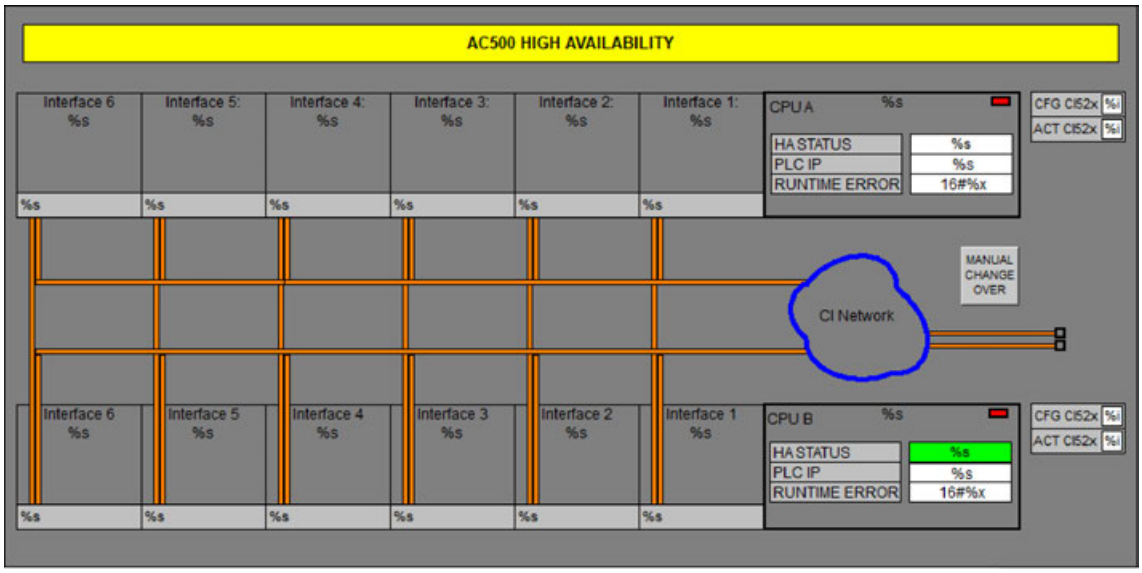
Visualization element HA_OVERVIEW_VISU gives an overview of the HA system state.

Based on the configuration, the state of the interfaces which are used is indicated.

Run state, PLC type, Primary or Secondary state, IP Address, Runtime Error, configured and actual count of CI52x modules are represented for both CPUs.

The button *[MANUAL CHANGE OVER]* may be used to perform a manual switch over.

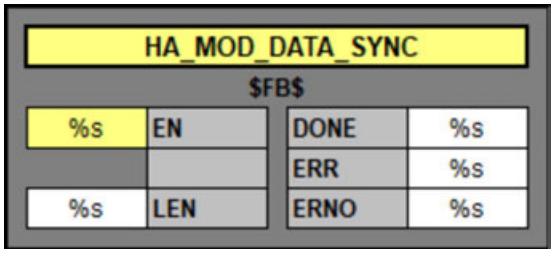
The following figure demonstrates visualization in the offline mode:



HA_MOD_DATA_SYNC_VISU_PH

Visualization element HA_MOD_DATA_SYNC_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the HA_MOD_DATA_SYNC function block. Visualization can also be used to control the function block by those inputs which are not connected inside the program.

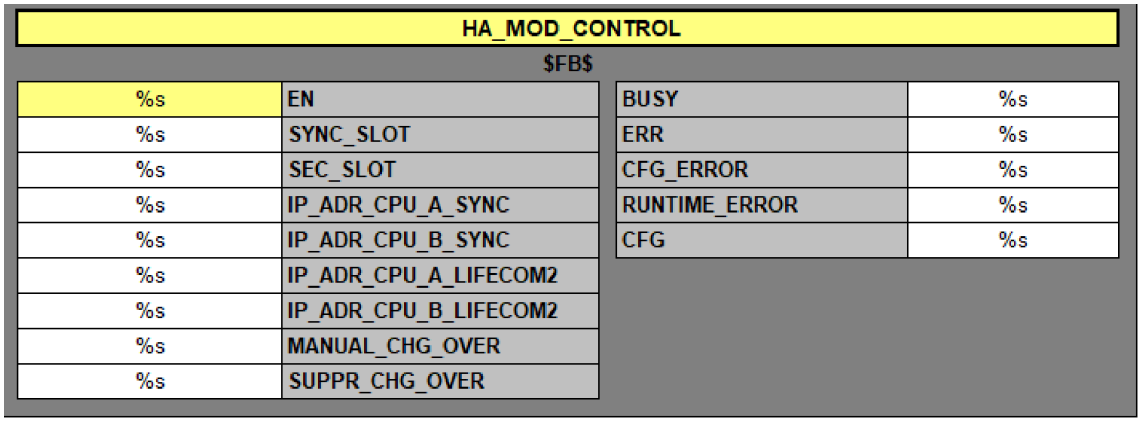
The following figure demonstrates visualization in the offline mode:



HA_MOD_CONTROL_VISU_PH

Visualization element HA_MOD_CONTROL_VISU_PH can be used to show the actual values of all inputs and outputs of the HA_MOD_CONTROL function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figure demonstrates visualization in the offline mode:



For all the utility and diagnosis function blocks visualizations are available in the library. All these visualization works same like the main function block visualizations defined above.

Global variables

HA_GLOBAL - errors/constants

List of all error numbers generated by HA-Modbus TCP library and specific constants declared in the library.

scope	Name	Type	Initial	Comment
Error Numbers	HA_MOD_INVALID_LENGTH	WORD	16#2017	Invalid length at the input of data sync block.
	HA_MOD_ERNO_TBL_OVERFLOW	WORD	16#2022	HA data reference table is full.
	HA_MOD_FRAME_TYPE_STATUS	BYTE	16#42	HA status frame.
Constants	HA_MOD_FRAME_TYPE_STATUS_DATA	BYTE	16#42	HA status and data frame 16#DD.
	HA_MOD_MAX_BUFFER_SIZE	UINT	1464	HA sync max. frame size (1400 original).
	HA_MOD_DELAY_CI52x_ERR	UINT	0	Number of cycle delay before declaring the CI52x failure error.
	HA_MOD_FILTER_TIME_HA_SYNC_ARRAY_INIT	UINT	200	Number of cycles to wait before calculating HA_SyncArray ptrData CS.
	HA_MOD_MAX_SYNC_ENTRIES	UINT	3000	Total size of the sync entry array (NoSyncFB-Pins* MaxSyncFBs.) in bytes.
	HA_MOD_MAX_DATA_IN_ETH_FRAME	UINT	1416	Ethernet Frame Length - Size of Header (1416 UDP) in bytes.

HA_GLOBAL_VARIABLES

Global variables for the status of high availability and Ethernet data exchange.

Name	Type	Initial	Comment
xHaModPrimary	BOOL	FALSE	State of the AC500 CPU. TRUE = PM acts as Primary. FALSE = PM acts as Secondary
xHaModCpuStop	BOOL	FALSE	If TRUE -> Indicates the CPU in STOP Mode
xHaModDataErr	BOOL	FALSE	If TRUE -> HA data sync is in error state.
wHaModDataErNo	WORD	0	HA data sync error code.
dwHaModTimersBaseTime	DINT	0	HA base time value for the HA timers
xHaModErr	BOOL	FALSE	If TRUE -> HA error state.
dwHaModOwnIP	DWORD	0	Own IP address configured for sync link connection.
dwHaModOtherIP	DWORD	0	Other PMs IP address configured for sync link connection.
dwHaModServerAlive	DWORD	0	Life counter incremented by OPC server.
wHaModEthLife	WORD	0	Ethernet Life Count.
timHaModSyncTimeOut	TIME	T#10MS	LifeCom1 UDP synchronization time out. May be changed by the user based on Cycle Time of the HA Task.
uiHaModSyncArrayIndex:	UINT	0	Synchronization ARRAY index
stHAEthRecData: HA_MOD_ETH_FRAME_HEADER;			Received Ethernet header
iTotalSyncData	UINT	0	Total sync data for the HA system
iNoOfEthFrames	INT	0	Number of packages sent for all Sync data.
xNoCiBus	BOOL	FALSE	No CI Bus configured. HA standalone system.

CI52x library

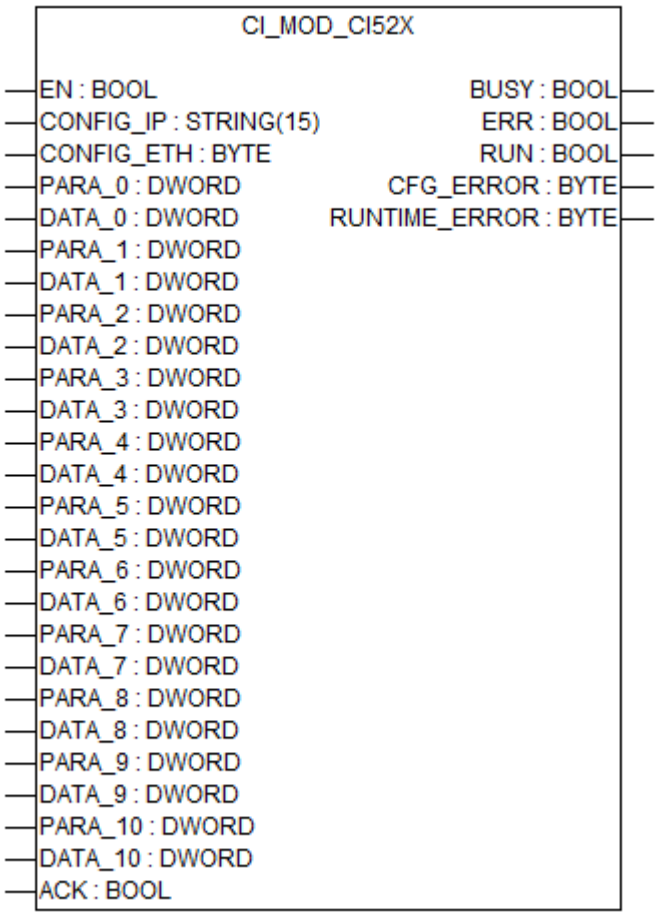
CI52x library is a part of HA-Modbus TCP library package.

Components of CI52x library:

- Function block:
  - CI_MOD_CI52x
  - CI_MOD_DIAG
- Visualization:
  - CI_MOD_CI52x_VISU_PH
  - CI_MOD_DIAG_VISU_PH
  - AI523, AI531, AI561, AI562, AI563
  - AO523, AO561
  - AX521, AX522, AX561
  - CI521, CI522
  - DA501, DA502
  - DC522, DC523, DC532, DC561
  - DI524, DI561, DI562, DI571, DI572
  - DO524, DO526, DO561, DO571, DO572, DO573
  - DX522, DX531, DX561, DX571
- Global variables *Chapter 1.5.5.2.3.3.3 “Global variable list (GVL)” on page 2191*
  - CI52x_GLOBAL_CONSTANT
  - CI52x_GLOBAL_VARIABLES

Function blocks

CI_MOD_CI52x



Available as of firmware	V2.6
Included in library	CI52x_AC500_V26.lib
Type	Function block with historical values

CI_MOD_CI52x function block is to establish the communication between AC500 PLC and communication interface module CI521 or CI522. Using this function block, status of the CI52x module configured in the network can be known.

The diagnosis info (CI_MOD_CI52x function block outputs) from CI52x communication interface module can be read out only once, so e.g. the secondary CPU once it was read.

So if change happens in CI52x diagnosis, it is not reflected in secondary CPU, leading to different diagnosis information in Primary and Secondary CPU.

Hence it is recommended to the user that the diagnosis info handled in application program e.g. should be synchronized in a separate structure OR handled e.g. on SCADA side

NOTICE!

For updating the state of inputs and outputs actions of the CI_MOD_CI52x function block must be called:

- Action “ACTION_INPUT_REFRESH” for refreshing the inputs,
- Action “ACTION_OUTPUT_REFRESH” for refreshing the outputs.

Input refresh should be done at the beginning of the task, output refresh at the end.

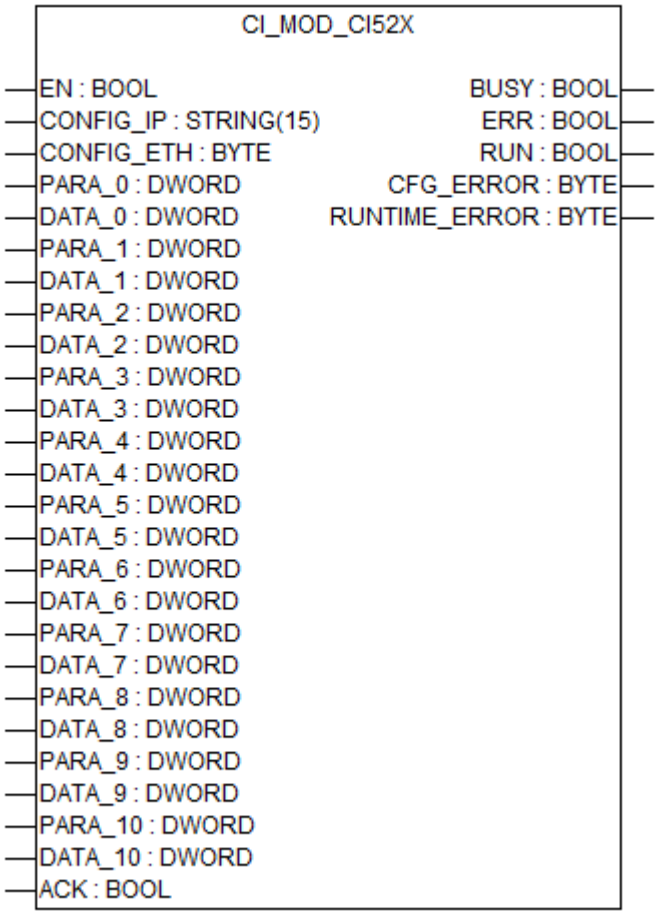
When there is a different task for the application logic, where the inputs and outputs are read/written it is mandatory that these actions are called from the task which is reading/writing the IO data.

For example, two Tasks: Modbus task and application task. In the Modbus Task the CI_MOD_CI52x function block is called.

The action “ACTION_INPUT_REFRESH” of the function block is called in beginning of the application task. Then all necessary calls for the application logic are done.

At the end of the application task the action “ACTION_OUTPUT_REFRESH” of the function block is called.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CI_MOD_CI52x function block disabling (EN = FALSE) will not set output RUN-TIME_ERROR Bit 0 (Bit 0 - CI module lost error) on HA_MOD_CONTROL function block.

CONFIG_IP

Data type	Default value	Range	Unit
STRING(15)	-	-	-

IP Address of the I/O module.

CONFIG_ETH

Data type	Default value	Range	Unit
BYTE	0	-	-

Ethernet slot used for communication on the PLC (see *Chapter 1.5.4.13.1.19 “ETHx_MOD_MAST” on page 1250*).

PARA_0

Data type	Default value	Range	Unit
DWORD	0	-	-

Communication interface module configuration address

DATA_0

Data type	Default value	Range	Unit
DWORD	0	-	-

Communication interface module IO structure address

PARA_1

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 1 configuration address.

DATA_1

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 1 IO structure address

PARA_2

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 2 configuration address.

DATA_2

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 2 IO structure address

PARA_3

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 3 configuration address.

DATA_3

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 3 IO structure address

PARA_4

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 4 configuration address.

DATA_4

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 4 IO structure address

PARA_5

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 5 configuration address.

DATA_5

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 5 IO structure address

PARA_6

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 6 configuration address.

DATA_6

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 6 IO structure address

PARA_7

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 7 configuration address.

DATA_7

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 7 IO structure address

PARA_8

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 8 configuration address.

DATA_8

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 8 IO structure address

PARA_9

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 9 configuration address.

DATA_9

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 9 IO structure address

PARA_10

Data type	Default value	Range	Unit
DWORD	0	-	-

I/O module 10 configuration address.

DATA_10

Data type	Default value	Range	Unit
DWORD	0	-	-

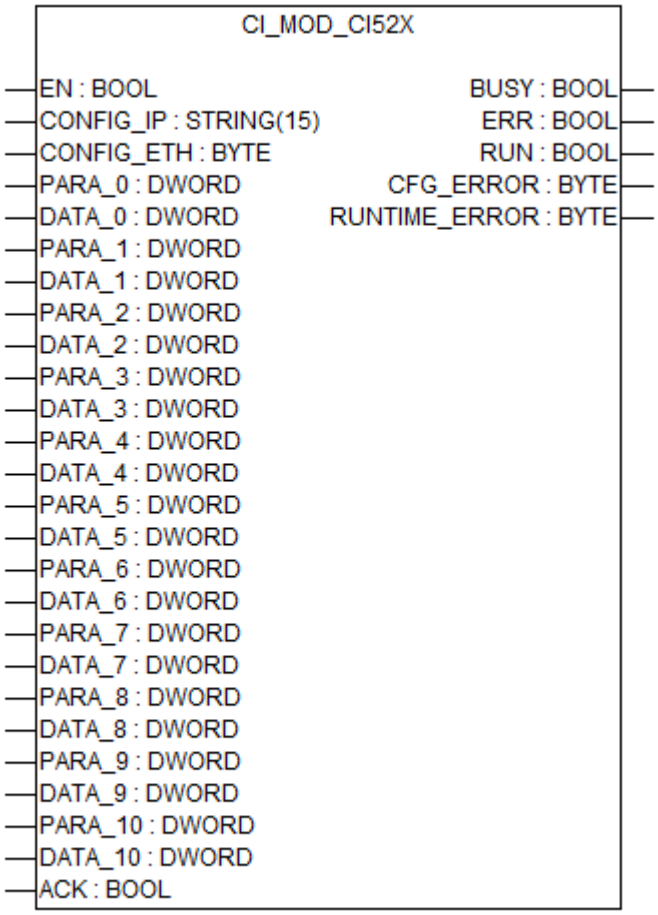
I/O module 10 IO structure address

ACK

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

To acknowledge errors.

Output description



BUSY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

If TRUE - Error condition reached, mismatch between configuration and actual hardware detected.

RUN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

If TRUE - CI52x module is running.

CFG_ERROR

Data type	Default value	Range	Unit
BYTE	0	-	-

Each bit of the byte represents different configuration errors:

- Bit 0: CPU_INFO error
- Bit 1: CONFIG_ETH invalid
- Bit 2: CONFIG_IP invalid

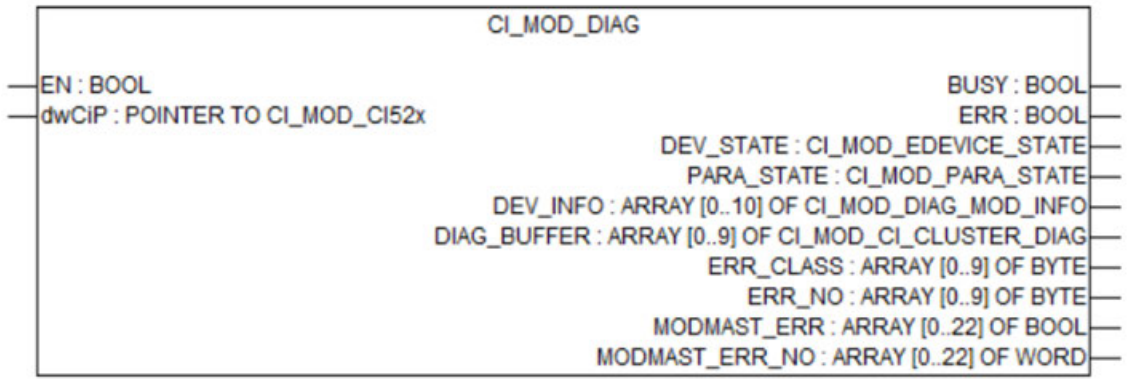
RUNTIME_ERROR

Data type	Default value	Range	Unit
BYTE	0	-	-

Each bit of the byte represents different runtime errors:

- Bit 0: Communication error
- Bit 1: I/O bus error
- Bit 2: Cluster error
- Bit 3: Hardware configuration error, mismatch between configuration and actual hardware detected

CI_MOD_DIAG

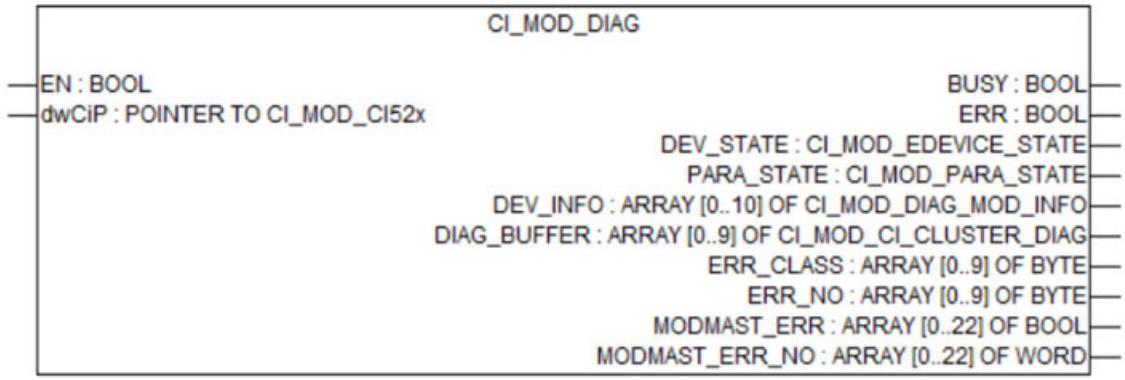


Available as of firmware	V2.6
Included in library	CI52x_AC500_V26.lib
Type	Function block with historical values

CI_MOD_DIAG function block can be used to get the additional diagnosis information related to the CI521 / CI522 communication module. This block provides addition information like device state, parameter state, device information, CI module diagnosis buffer, CI module related errors and details of error generated in the COM_MOD_MAST core block.

This function block must be connected to CI_MOD_CI52x function block instance using ADR operator to read the diagnosis of particular communication interface module.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

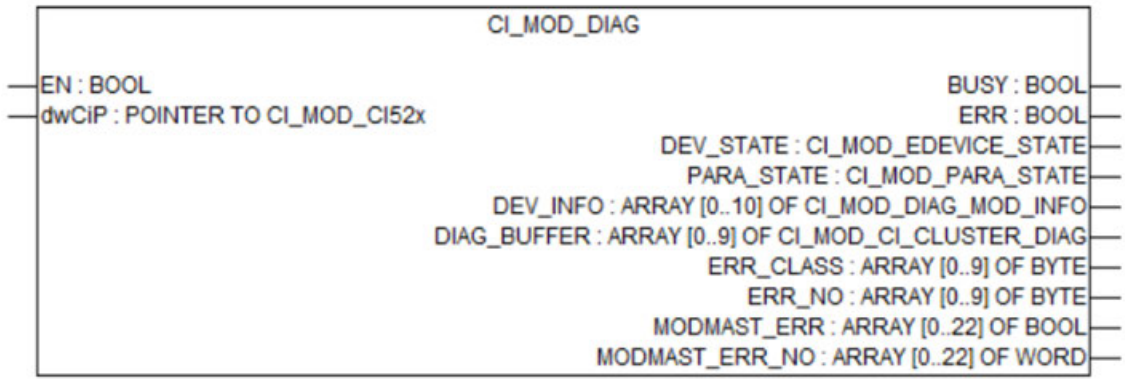
While it is executed its inputs are continuously evaluated.

dwCiP

Data type	Default value	Range	Unit
POINTER TO CI_MOD_CI52x	-	-	-

This input points to the address of the CI communication block instance from which the diagnosis information has to be read. Use the ADR operator and connect the instance of CI_MOD_CI52x function block.

Output description



BUSY

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Operation is running.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

Error occurred during execution when output ERR = TRUE.

DEV_STATE

Data type	Default value	Range	Unit
CI_MOD_DEVICE_STATE	-	-	-

CI521 or CI522 device current status is displayed.

- STATE_PREOP – Device is booting
- STATE_OPERATION – Device is operational, no bus monitoring is active
- STATE_ERROR – Device detected a bus error, bus monitoring is active
- STATE_IP_ERROR – Device has an IP address error
- STATE_CYCLIC_OPERATION – Device is operational, bus monitoring is active
- STATE_NA – Not available

PARA_STATE

Data type	Default value	Range	Unit
CI_MOD_PARA_STATE	-	-	-

CI521 or CI522 device current status is displayed.

- PARA_STATE_NO_PARA – Device has no parameters
- PARA_STATE_PARA_ACTIVE – Parameterization process is running
- PARA_STATE_PARA_DONE– Device used valid parameters and parameterization is done
- PARA_STATE_ERROR– Device has invalid parameters
- PARA_STATE_NA – Not available

DEV_INFO

Data type	Default value	Range	Unit
ARRAY[0..10] OF CI_MOD_DIAG_MOD_INFO	-	-	-

CI521 or CI522 type and extended module types. This will give the details of the module configured in the communication interface module including the I/O modules.

If module is with suffix F, then fast counter is enabled for that module.

DIAG_BUFFER

Data type	Default value	Range	Unit
ARRAY[0..9] OF CI_MOD_CI_CLUSTER_DIAG	-	-	-

CI521 or CI522 module diagnosis buffer. More details on this are provided in ↳ *Chapter 1.6.4.3.1.2.3.2 “Diagnosis data” on page 5659*

ERR_CLASS

Data type	Default value	Range	Unit
ARRAY[0..9] OF BYTE	-	-	-

CI error class. Refer to the diagnostics ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

ERR_NO

Data type	Default value	Range	Unit
ARRAY[0..9] OF BYTE	-	-	-

CI error number. Refer to the diagnostics *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

MODMAST_ERR

Data type	Default value	Range	Unit
ARRAY[0..22] OF BOOL	-	-	-

Latest 22 Modbus TCP error messages of the block.

MODMAST_ERR_NO

Data type	Default value	Range	Unit
ARRAY[0..22] OF WORD	-	-	-

Latest 22 Modbus TCP error numbers. Refer to the error details in Modbus *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

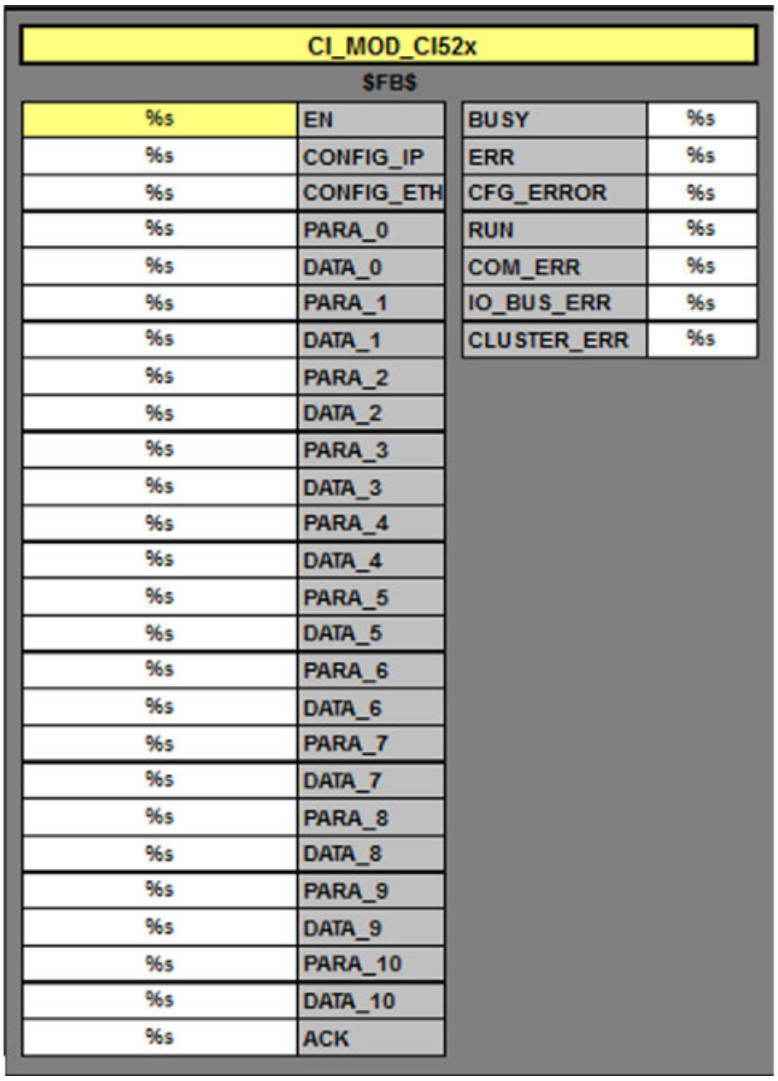
Visualization

In the application program, the user can add the visualization object in his project. In the following chapter, overall visualization and visualization for each individual function block are discussed in detail.

CI_MOD_CI52x_VISU_PH

The visualization element CI_MOD_CI52x_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of the CI52x function block. Visualization can also be used to control the function block by those inputs which are not connected inside the program.

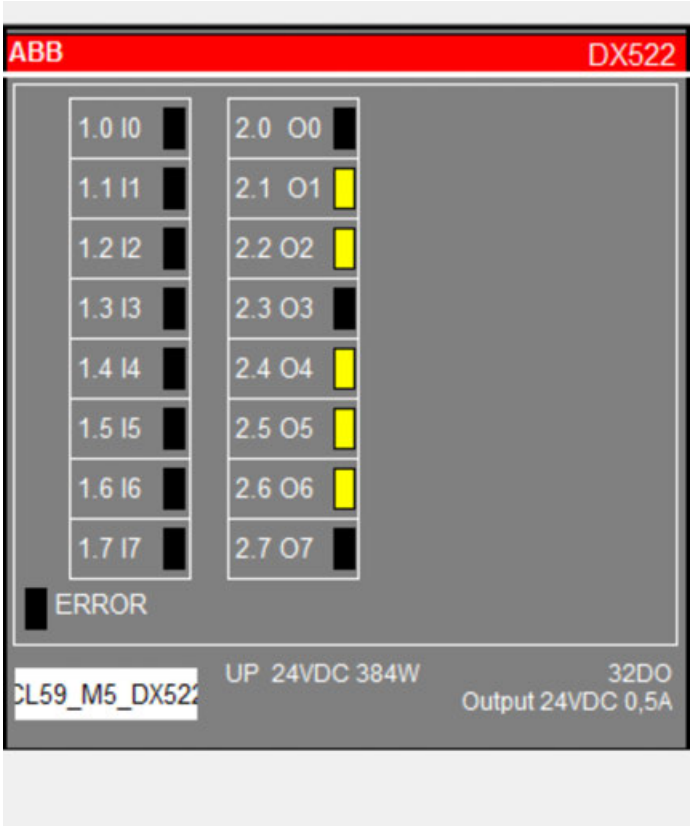
The following picture demonstrates visualization in offline mode:



IO visualization

Visualization elements can be used to show the actual values of all inputs and outputs on the hardware module. Values will be updated only on the primary PLC.

The following example figure demonstrates visualization in the online mode:



In the similar way for all the IO modules visualizations are available in the library which can be used to view the state of the LEDs on hardware module.

Values will be updated only on the Primary PLC.

Global variable list (GVL)

List of all global variables declared in the library.

CI52x_GLOBAL_CONSTANT

List of all the global constants declared in the library.

Scope	Name	Type	Initial	Comment
Constants	CI52x_MOD_SEND	BYTE	16	Write the values to the CI modules selected
	CI52x_MOD_REC	BYTE	3	Read the values from the CI modules selected
	CI52x_MOD_FC23	BYTE	23	Read or write the values to the CI modules selected
	CI52x_MOD_NOT_VALID	BYTE	16#0	Status to indicate that module is not available.

CI52x_GLOBAL_VARIABLES

List of all global variables declared in the library which are intended for the user

Name	Type	Initial	Comment
byCI52xActive-CICluster	BYTE	0	Number of active CI52x clusters in the network.
aCI52xActiveCIClus-terStatus	ARRAY [1..100] OF POINTER TO CI_MOD_CLUSTER_STATUS		Status of the clusters configured.
timCI52xTimeOut	TIME	T#10ms	Modbuscommunica-tion time out.

1.5.5.3 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.6 ACS / DCS drives libraries

1.5.6.1 System technology

In the following chapter a guide is given how to select the appropriate function blocks to set up a working system of AC500 together with ACS drives. This guide is given in form of questions which answers will lead step by step to the selection of correct components and function blocks.

Additional information on preconditions of the configuration of the drives can then be found at the documentation of the selected function blocks.

1.5.6.1.1 Application selection

Question: What kind of application should be realized?

- Motion control application using PLC open motion control function blocks --> Use the PS552-MC motion control library
- Speed and/or torque applications using ABB Drives Profile in the ACS drive --> Go to chapter Connection Selection ↳ *Chapter 1.5.6.1.2 “Connection selection” on page 2192*

1.5.6.1.2 Connection selection

Question: What kind of connection between AC500 and ACS drive should be used?

- discrete or analog wiring to start/stop and give reference values --> no special function blocks needed. Please see drive manuals how to connect.
  PROFIBUS ↳ *Chapter 1.5.6.1.3 “PROFIBUS” on page 2192*
  PROFINET ↳ *Chapter 1.5.6.1.4 “PROFINET” on page 2194*
  CANopen ↳ *Chapter 1.5.6.1.5 “CANopen” on page 2194*
  EtherCAT ↳ *Chapter 1.5.6.1.6 “EtherCAT” on page 2195*
  Modbus TCP ↳ *Chapter 1.5.6.1.7 “Modbus TCP” on page 2195*
  Modbus RTU ↳ *Chapter 1.5.6.1.8 “Modbus RTU” on page 2200*

1.5.6.1.3 PROFIBUS

The following hardware components must be available:

- AC500 with a PROFIBUS master Communication Module
- ACSxxx / DCSxxx drive with FPBA-01 or RPBA-01 PROFIBUS Communication Module
- PROFIBUS cable with DSUB9 for the PROFIBUS master Communication Module and DSUB9 for drive Communication Modules

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500.

- Drive --> AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 --> drive: Control word and reference value 1 (speed) and optional reference value 2 (torque)

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- Optional ACSDrivesComPB_AC500_V24.lib *Chapter 1.5.6.6 “ACS / DCS Drives communication via PROFIBUS” on page 2410*
- DCS Drives Library *Chapter 1.5.6.8 “DCS drives library” on page 2467* (if DCS drives are used)

The following communication profile should be used:

- ABB Drives profile

The following function blocks can be used in AC500 program

- Control function blocks:
  - Generic control blocks – only ACSDrivesBase_AC500_V20.lib “ACS Drives Base Library” and DCSDrivesBase_AC500_V24.lib is needed.
  - ACS_DRIVES_CTRL_STANDARD_GEN *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*
  - DCS_DRIVES_CTRL_GEN *Chapter 1.5.6.8.4.2 “DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus” on page 2477*(if DCS drives are used)
  - Standard control blocks toghether with communication blocks
    ACS_DRIVES_CTRL_STANDARD
    DCS_DRIVES_CTRL_STANDARD
- scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- Communication function blocks:
  - ACS_COM_PB *Chapter 1.5.6.6.3.1 “ACS_COM_PB communication block via PROFIBUS” on page 2412* Communication for ACS drives via PROFIBUS DP for PZD1..3
  - ACS_COM_PB_PZD *Chapter 1.5.6.6.3.2 “ACS_COM_PB_PZD communication block for direct access to PZD4..12” on page 2415* (Communication for ACS drives via PROFIBUS DP for Process Data PZD4..12)
- Parameter READ / WRITE function blocks:
  PPO-Types 3, 4, 6 or 8 (DPV1)
  - ACS_PB_READ_N_PRM_DPV1 *Chapter 1.5.6.6.3.5 “ACS_PB_N_READ_PRM_DPV1 read parameters from ABB drives via PROFIBUS DPV1” on page 2425*
  - ACS_PB_WRITE_N_PRM_DPV1 *Chapter 1.5.6.6.3.6 “ACS_PB_N_WRITE_PRM_DPV1 write parameters from ABB drives via PROFIBUS DPV1” on page 2431*
    These function blocks can be used stand-alone without any communication block.
  - ACS_PB_READ_PRM_DPV0 *Chapter 1.5.6.6.3.3 “ACS_PB_READ_PRM_DPV0 read parameters from ABB drives via PROFIBUS DPV0” on page 2420*
  - ACS_PB_WRITE_PRM_DPV0 *Chapter 1.5.6.6.3.4 “ACS_PB_WRITE_PRM_DPV0” on page 2423*
    These function blocks must be used together with FB ACS_COM_PB “ACS_COM_PB”

1.5.6.1.4 PROFINET

The following hardware components must be available:

- AC500 with CM579-PNIO (PROFINET master Communication Module)
- ACSxxx drive with FENA-01, FENA-11, FENA-21 or RETA-02 realtime Ethernet communication module
- Ethernet cable with RJ45 plugs

The following values should be mapped in the fieldbus configuration of the drive and the configuration of AC500.

- Drive --> AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 --> drive: Control word and reference value 1 (speed) and optional reference value 2 (torque)

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- DCS Drives Library *Chapter 1.5.6.8 “DCS drives library” on page 2467*(if DCS drives are used)
- Optional: ACSDrivesComPN_AC500_V24.lib *Chapter 1.5.6.7 “ACS / DCS Drives read / write parameter via PROFINET library” on page 2451*

The following function blocks can be used in AC500 program:

- Control function block:

The following communication profile should be used:

ABB Drives profile

- ACS_DRIVES_CTRL_STANDARD_GEN *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*
- DCS_DRIVES_CTRL_GEN (if DCS drives are used) *Chapter 1.5.6.8.4.2 “DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus” on page 2477*
- scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- Parameter READ / WRITE function blocks:
  - ACS_PN_READ_N_PRM_DPV1 *Chapter 1.5.6.7.4.1 “ACS_PN_READ_N_PRM_DPV1 read parameters from ABB drives via PROFINET DPV1” on page 2453*
  - ACS_PN_WRITE_N_PRM_DPV1 *Chapter 1.5.6.7.4.2 “ACS_PN_WRITE_N_PRM_DPV1 write parameters from ABB drives via PROFINET DPV1” on page 2457*

1.5.6.1.5 CANopen

The following hardware components must be available:

- AC500 with CM598-CN (CANopen master Communication Module)
- ACSxxx drive with FCAN-01 or RCAN-01 CANopen slave Communication Module
- CANopen cable

The following values should be mapped in the fieldbus configuration of the drive and the Automation Builder configuration of AC500.

- Drive --> AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 --> drive: Control word and reference value 1 (speed) and optional reference value 2 (torque)

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- DCS Drives Library (if DCS drives are used) *Chapter 1.5.6.8 “DCS drives library” on page 2467*

The following communication profile should be used:

- ABB Drives profile

The following function blocks can be used in AC500 program:

- Control function blocks:
  - ACS_DRIVES_CTRL_STANDARD_GEN *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*
  - DCS_DRIVES_CTRL_GEN (if DCS drives are used) *Chapter 1.5.6.8.4.2 “DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus” on page 2477*
- scaling: ACS_REF_SCALING *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*(optional)

1.5.6.1.6 EtherCAT

The following hardware components must be available:

- AC500 with CM579-ETHCAT (EtherCAT master Communication Module)
- ACSxxx drive with FECA01 or RECA-01 EtherCAT slave Communication Module
- Ethernet cable with RJ45 plugs

The following values should be mapped in the fieldbus configuration of the drive and the Automation Builder configuration of AC500.

- Drive --> AC500: Status word and actual value 1 (speed) and optional actual value 2 (torque).
- AC500 --> drive: Control word and reference value 1 (speed) and optional reference value 2 (torque)

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- DCS Drives Library (if DCS drives are used) *Chapter 1.5.6.8 “DCS drives library” on page 2467*

The following communication profile should be used:

- ABB Drives profile

The following function blocks can be used in AC500 program

- Control function blocks:
  - ACS_DRIVES_CTRL_STANDARD_GEN *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*
  - DCS_DRIVES_CTRL_GEN (if DCS drives are used) *Chapter 1.5.6.8.4.2 “DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus” on page 2477*
- scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*

1.5.6.1.7 Modbus TCP

The following hardware components must be available:

- AC500 or AC500-eCo with Ethernet option
- Drive with FENA-01 or FENA-11 or RETA-01 or RETA-02 realtime Ethernet communication module
- Ethernet cable with RJ45 plugs

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib ↳ *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- In case the Firmware is less then V2.4 and CPU has only one Ethernet Interface: ACSDrivesComModTCP_AC500_V22.lib ↳ *Chapter 1.5.6.4 “ACS / DCS drives communication via Modbus TCP library” on page 2359*
- In case the CPU has two Ethernet Interfaces e.g. PM595 or PM591-2ETH and at least with Firmware V2.4. ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*

Question:Which ACSxxx drive should be used?

- ACS355 ↳ *Chapter 1.5.6.1.7.1 “ACS355 with Modbus TCP” on page 2196*
- ACS550 or ACH550 ↳ *Chapter 1.5.6.1.7.2 “ACS550 or ACH550 with Modbus TCP” on page 2197*
- ACS580 ↳ *Chapter 1.5.6.1.7.3 “ACS580 with Modbus TCP” on page 2197*
- ACS800 ↳ *Chapter 1.5.6.1.7.4 “ACS800 with Modbus TCP” on page 2198*
- ACS850, ACQ810 or ACSM1 ↳ *Chapter 1.5.6.1.7.5 “ACS850, ACQ810 or ACSM1 with Modbus TCP ” on page 2199*
- ACS880 ↳ *Chapter 1.5.6.1.7.6 “ACS880 with Modbus TCP” on page 2200*
- DCS800 or DCS550 ↳ *Chapter 1.5.6.1.7.7 “DCS800 or DCS550 with Modbus TCP” on page 2200*

ACS355 with Modbus TCP

Used drive Communication Module: FENA-x1:

Question: How many values should be exchanged?

- only status word, actual speed, control word and speed reference:
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS3XX_DRIVES_CTRL_BASIC ↳ *Chapter 1.5.6.2.4.4 “ACS3XX_DRIVES_CTRL_BASIC” on page 2220*
  - scaling: the scaling is included in the ACS3XX_DRIVES_CTRL_BASIC ↳ *Chapter 1.5.6.2.4.4 “ACS3XX_DRIVES_CTRL_BASIC” on page 2220* function block
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*(optional)

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 10 more values read from drive and up to 10 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS550 or ACH550 with Modbus TCP

Used drive Communication Module: RETA-01

Question: How many values should be exchanged?

- only status word, actual speed, control word and speed reference:
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS3XX_DRIVES_CTRL_BASIC ↳ *Chapter 1.5.6.2.4.4 “ACS3XX_DRIVES_CTRL_BASIC” on page 2220*
  - scaling: the scaling is included in the ACS3XX_DRIVES_CTRL_BASIC function block
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS580 with Modbus TCP

Used drive Communication Module: FENA-x1

Question: How many values should be exchanged?

- only status word, actual speed, control word and speed reference:
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS_DRIVES_CTRL ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program:
    Use ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*

    “ACS_COM_MOD_TCP_ENHANCED Communication for ACS Drives via Modbus TCP”, if PLC firmware is less than V2.4, else use

    ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*

    “ACS_COM_MOD_TCPx_ENHANCED Communication for ACS Drives via Modbus TCP”
  - control function block in AC500 program:
    ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*

    “ACS_DRIVES_CTRL_STANDARD Control of ACS Drives with ABB-Drives Profile”
  - scaling: ACS_REF_SCALING (optional)
    “ACS_REF_SCALING ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246* Scaling for ACS Reference and Actual Values”
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS800 with Modbus TCP

Used drive Communication Module: RETA-01

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*
-

ACS850, ACQ810 or ACSM1 with Modbus TCP

Used drive Communication Module: FENA-x1

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*.
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246* (optional)
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*
-

ACS880 with Modbus TCP

Used drive Communication Module: FENA-11

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP_ENHANCED *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx_ENHANCED *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP_ENHANCED *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367* “ACS_COM_MOD_TCP_ENHANCED, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx_ENHANCED *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*

DCS800 or DCS550 with Modbus TCP

Used drive Communication Module: RETA-01

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication function block in AC500 program: Use ACS_COM_MOD_TCP *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, if PLC firmware is less than V2.4, else use ACS_COM_MOD_TCPx *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*
  - control function block in AC500 program: DCS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - to exchange more than above mentioned values use additionally the following blocks: ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

1.5.6.1.8 Modbus RTU

The following hardware components must be available:

- AC500 or AC500-eCo
- Drive with embedded Modbus RTU or FMBA-01 or FSCA-01 or RMBA-01 RS-485 Communication Module
- Twisted pair serial cable

The following libraries have to be loaded into the project:

- ACSDrivesBase_AC500_V20.lib *Chapter 1.5.6.2 “ACS drives base library” on page 2204*
- ACSDrivesComModRTU_AC500_V22.lib *Chapter 1.5.6.3 “ACS / DCS Drives communication via Modbus RTU library” on page 2288*
- DCS Drives Library *Chapter 1.5.6.8 “DCS drives library” on page 2467* (if DCS drives are used)

Question:Which Drive should be used?

- ACS310, ACS350, ACS355, ACS550 or ACH550 *Chapter 1.5.6.1.8.1 “ACS310, ACS350, ACS355, ACS550 or ACH550 with Modbus RTU” on page 2201*
- ACS800 *Chapter 1.5.6.1.8.2 “ACS800 with Modbus RTU” on page 2202*
- ACS850 or ACQ810 *Chapter 1.5.6.1.8.3 “ACS850, ACQ810 with Modbus RTU” on page 2202*
- ACS880 or ACSM1 or ACS580 *Chapter 1.5.6.1.8.4 “ACS880 or ACSM1 or ACS580 with Modbus RTU” on page 2203*
- DCS800 *Chapter 1.5.6.1.8.5 “DCS800 with Modbus RTU” on page 2203* or DCS550

ACS310, ACS350, ACS355, ACS550 or ACH550 with Modbus RTU

Used drive Communication Module: embedded, only ACS350 or ACS355 with FMBA-01:

Question: Are the Emergency stops needed and is the change to Ext1/2 control location needed?

- Emergency stops or change to external control location is NOT needed
  - READ / WRITE variables: status word, actual speed, control word and speed reference and up to 7 more values read from the drive
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS3XX_COM_MOD_RTU *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293*
  - control function block in AC500 program: ACS3XX_DRIVES_CTRL_BASIC *Chapter 1.5.6.2.4.4 “ACS3XX_DRIVES_CTRL_BASIC” on page 2220*
  - scaling:the scaling is included in the ACS3XX_DRIVES_CTRL_BASIC function block
- Emergency stops or change to external control location is needed
  - read/write variables: status word, actual speed, control word and speed reference and up to 7 more values read from the drive:
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS3XX_COM_MOD_RTU *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS800 with Modbus RTU

Used drive Communication Module: RMBA-01

Question: How many values should be exchanged?

- READ / WRITE variables: status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque)
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS850, ACQ810 with Modbus RTU

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque) and up to 24 more values read from the drive
  - used drive Communication Module: embedded Modbus using the D2D plug
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque)
  - used drive Communication Module: embedded Modbus using the D2D plug or FSCA-01
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - used drive Communication Module: FSCA-01
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: ACS_COM_MOD_RTU_ENHANCED *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

ACS880 or ACSM1 or ACS580 with Modbus RTU

Used drive Communication Module: FSCA-01:

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque)
  - communication profile in drive parameters: ABB Drives classic
  - communication function block in AC500 program: ACS_COM_MOD_RTU *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed), reference value2 (torque) and up to 12 more values read from drive and up to 12 more values write to the drive
  - communication profile in drive parameters: ABB Drives enhanced
  - communication function block in AC500 program: ACS_COM_MOD_RTU_ENHANCED *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312*
  - control function block in AC500 program: ACS_DRIVES_CTRL_STANDARD *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*
  - scaling: ACS_REF_SCALING (optional) *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

DCS800 with Modbus RTU

Used drive Communication Module: RMBA-01

Question: How many values should be exchanged?

- status word, actual value1 (speed), actual value2 (torque), control word, reference1 (speed) and reference value2 (torque):
  - communication function block in AC500 program: Use ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*
  - control function block in AC500 program: DCS_DRIVES_CTRL ↳ *Chapter 1.5.6.8.4.1 “DCS_DRIVES_CTRL Control of DCS Drives with ABB-Drives profile using a communication block” on page 2470*
  - scaling: ACS_REF_SCALING (optional) ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*
- to exchange more than above mentioned values use additionally the following blocks:
  - ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*
  - ACS_MOD_WRITE_N_PRM ↳ *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*

1.5.6.2 ACS drives base library

1.5.6.2.1 Preconditions for the use of the ACS drives base library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

Some of the function blocks can only be used in combination with one of the ACSDrivesComXXX_AC500_V20 libraries, e.g. the ACSDrivesComModRTU_AC500_V20.lib.

The library is released for the following products:

CPUs: AC500, AC500-eCo

Modbus communication is tested for connection of 7 drives in total. Connection of more drives depends on performance of used CPU, communication type and settings.

Fieldbuses: Modbus RTU (Serial Modbus)

Modbus TCP

CANopen: For CANopen communication we have to use FCAN-01 module firmware version 1050 or above has to be used.

PROFIBUS

PROFINET

EtherCAT

General blocks Following blocks can be used independent of used fieldbus:

- ACS_DRIVES_CTRL_STANDARD_GEN ↳ *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*
- ACS_REF_SCALING ↳ *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*

Compatibility

To check the compatiblity of drives and their communication modules please refer to the following table, that shows the tested combinations:

Communication	PLC communication modules		PLC Fieldbus Adapter			Drive		
	PLC communication module	Firmware Version	Fieldbus Adapter (FBA)	FBA comm sw ver	FBA appl sw ver	Drive	Firmware Version	Drive Rating ID
Modbus RTU	Onboard		FMBA			ACS355	5060 / 5090	ACS355-03E-01A2-4
						ACS355	5060	ACS355-01E-02A4-2
						ACS355	5040	ACS355-01E-02A4-2
			Onboard	Onboard	Onboard	ACS310	402A	ACS310-03E-01A3-4
			RMBA			ACS800	SW Ver AS7R7 365. Appl Ver ASAR F018	ACS800-01-0005-3
			Onboard	Onboard	Onboard	ACS550	313D / 314E	ACH550-01-03A3-4
			Onboard	Onboard	Onboard	ACH550	313D / 314E	ACH550-01-02A4-4
			FSCA-01	300C	042A	ACS850	UIFI 2110 / 2700	ACS850-04-03A0-5
			FSCA-01	300C	042A	ACSM1	UMFI 1510 / 1600	ACSM1-04Ax-03A0-4
			Onboard	Onboard	Onboard	ACQ810	UIFQ 2010 / 2200	ACQ810_04-02A7-4
Modbus RTU - Enhanced	Onboard		FSCA-01	300C	042A / 163	ACQ810	UIFQ 2010 / 2200	ACQ810_02A7

Communication	PLC communication modules		PLC Fieldbus Adapter			Drive		
	PLC communication module	Firmware Version	Fieldbus Adapter (FBA)	FBA comm sw ver	FBA appl sw ver	Drive	Firmware Version	Drive Rating ID
Modbus TCP	CM577-ETH	V01.100	FENA-11	0072	0252 / 0302	ACS850	UIFI 2110	ACS850-04-03A0-5
	Onboard	V01.100	FENA-11	0072	0252 / 0302	ACSM1	UMFI 1510 / 1600	ACSM1-04Ax-03A0-4
Modbus TCP - Enhanced	Onboard	V 1.3.2	FENA-01	0062	0252/302	ACQ810	UIFQ 2010 / 2200	ACQ810_02A7
PROFIBUS	CM572-DP	V01.097	FPBA-01	205B	0200B / 0300	ACSM1	UMFI 1510 / 1600	ACSM1-02A5-4
			FPBA-01	3102	2145 / 0300	ACS880	AINF01.11.0.0 / 1.92.200.3	AINF01.11.0.0
			FPBA-01	3102	21450 / 300	ACS850	UIFI 2400	ACS850_03A0
PROFIBUS	CM592-DP	V3.00	FPBA-01	-	0x0300	ACS880	AINFC 2.82.0.0	ACS880-01-02A4-3
PROFINET	CM579-PNIO	2.6.5 (0)	FENA-11	072	0255 / 0302	ACS880	AINF01.11.0.0	ACS880-01-02A4-3(301)
			FENA-11	062	0246 / 0302	ACQ810	UIFQ 2020	ACQ810_02A7
EtherCAT	CM579_ECAT	2.4.11 (0)	FECA-01	0073	0109 / 121	ACS355	5060 / 5090	ACS355-03E-01A2-4
CANopen	CM578-CAN	V01.101	FCAN-01	3120	1050	ACS850	UIFI 2400	ACS850_03A0
CANopen	CM598-CAN	V1.16	FCAN-01	-	0x0116	ACS880	AINFC 2.82.0.0	ACS880-01-02A4-3

1.5.6.2.2 Special characteristics of the ACS drives base library

The types and constants defined in this library are base types and can be used in other ACSDrivesXXX libraries, e.g. ACSDrivesComModRTU_AC500_V20.lib.

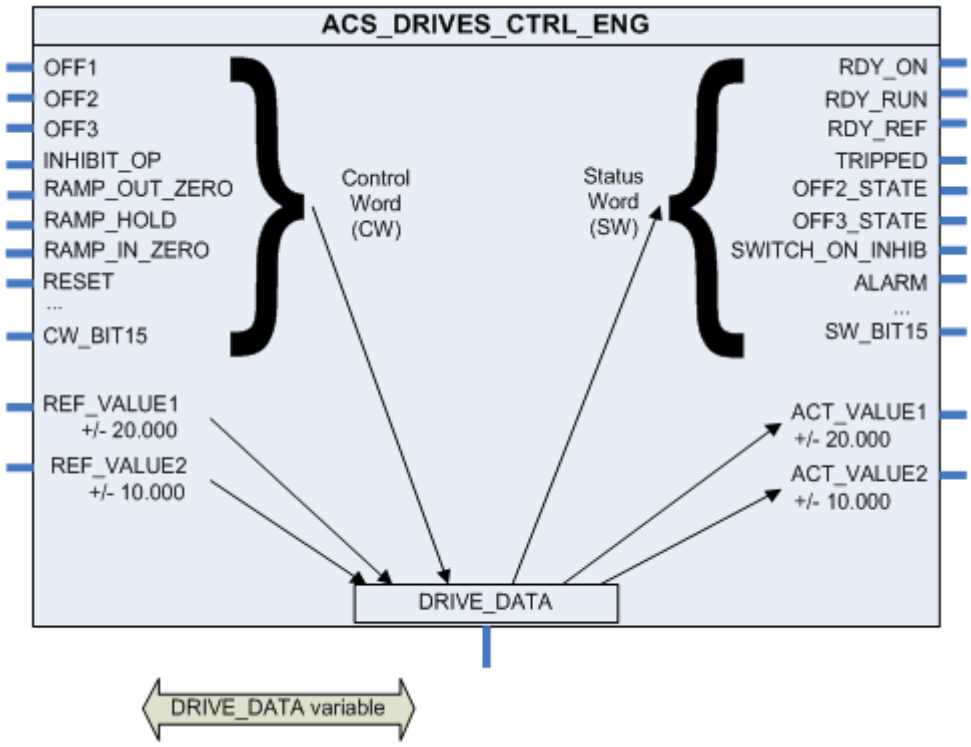
The ACSDrivesBase_AC500_V20 Library includes 3 different control function blocks to be used with ACS drives configured for ABB Drives Profile.

For three of these blocks the communication to the drive is made via the DRIVE_DATA variable which must be connected to an communication function block. The communication function blocks for different fieldbuses can be found in separate libraries, e.g. ACSCom-ModRTU_AC500_V20.lib for Modbus RTU connection.

One control function block (ACS_DRIVES_CTRL_STANDARD_GEN) is a generic block which doesn´t need a communication block. The input of Status Word and output of Control Word must be mapped somehow to any fieldbus.

The control function blocks work independent of the used fieldbus connection. The principle and differences of the 4 control blocks is shortly described below:

ACS_DRIVES_CTRL_ENG

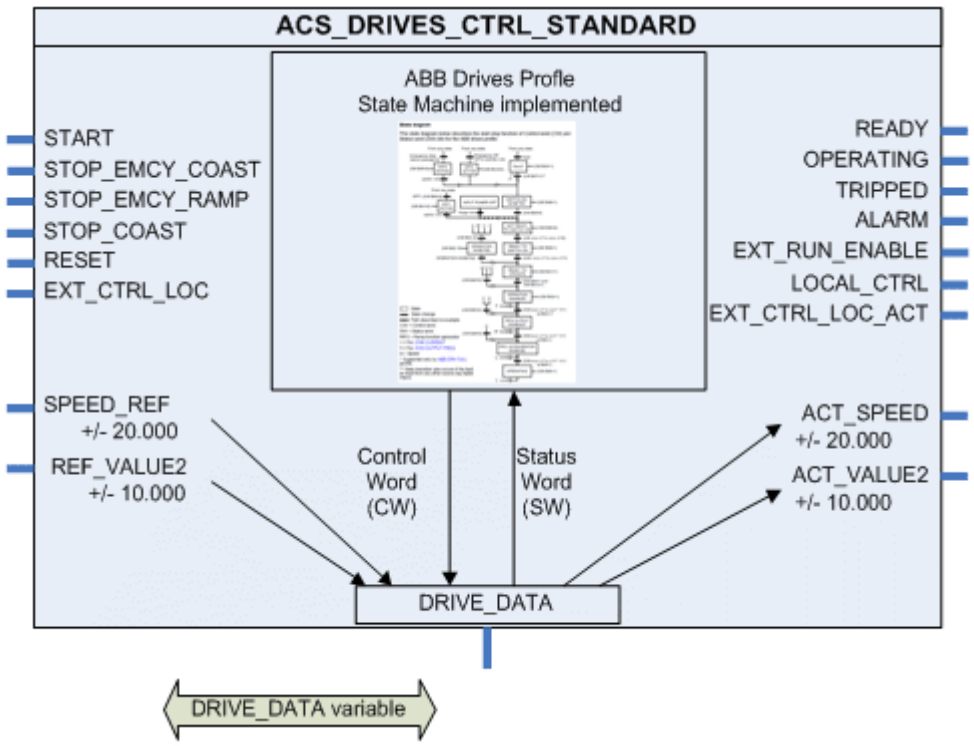


The ACS_DRIVES_CTRL_ENG function block is designed for user specific control of the drive setting the Control Word (CW) by the user in the program.

Therefore the user should have a detailed knowledge of the ABB Drives Profile handling.

The reference and actual values must be given in fieldbus equivalent, e.g. +/- 20000 for the reference value 1.

ACS_DRIVES_CTRL_STANDARD

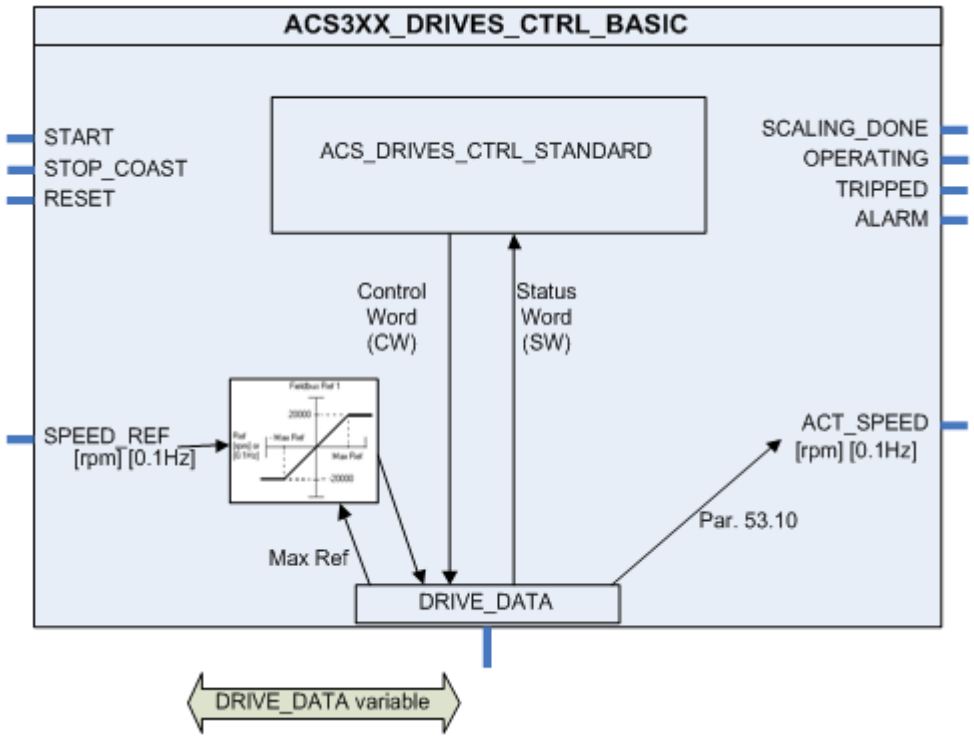


The ACS_DRIVES_CTRL_STANDARD function block can be used for standard control functions.

The handling of the ABB Drives Profile State Machine is done inside the block. The Control Word is built according to ABB Drives Profile State Machine, the Status Word (SW) and the inputs of the function block.

The reference and actual values must be given in fieldbus equivalent, e.g. +/- 20000 for the reference value 1.

ACS3XX_DRIVES_CTRL_BASIC



The ACS3XX_DRIVES_CTRL_BASIC function block is designed especially for ACS3XX and ACX550 drives and provides additionally scaling functions for the speed reference.

The handling of the ABB Drives Profile State Machine is done inside the block. The Control Word is built according to ABB Drives Profile State Machine, the Status Word (SW) and the inputs of the function block.

The speed reference input can be given in rpm or in 0.1 Hz. The act speed output is also in rpm or 0.1 Hz.

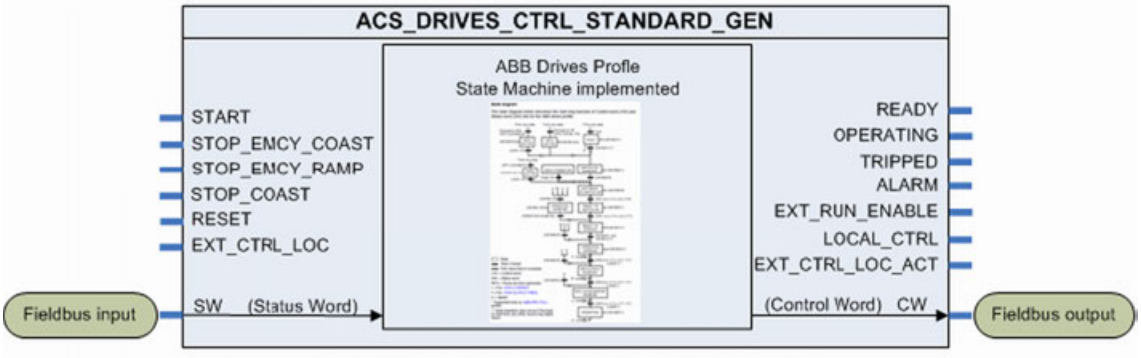
ACS_DRIVES_CTRL_STANDARD_GEN

The ACS_DRIVES_CTRL_STANDARD_GEN function block can be used for standard control functions, using any generic fieldbus.

The handling of the ABB Drives Profile State Machine is done inside the block. The Control Word is built according to ABB Drives Profile State Machine, the Status Word (SW) and the inputs of the function block.

The block does not have a DRIVE_DATA input. The SW and CW can be retrieved from any where. So any generic fieldbus can be used to get the SW and send the CW to the drive.

The block does not have any input or output for actual- or reference speed values. These can be send directly via the fieldbus to the drive.

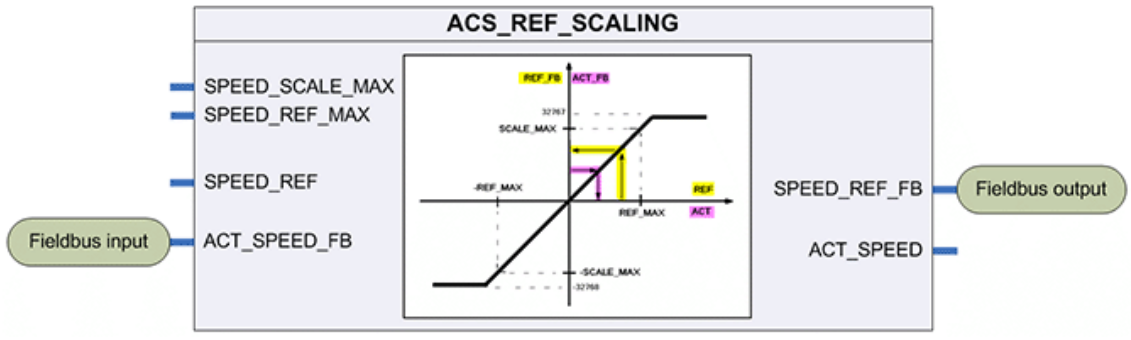


ACS_REF_SCALING

This block can be used to scale the actual and reference values from fieldbus equivalent (+/- 20000 or +/- 10000) to the physical value ("rpm" or "Hz").

Figure bellow shows an overview for the speed values. Same functionality is also provided in the same block for the torque values.

This block can also be used together with other ACS or DCS blocks, e.g. ACS_DRIVES_CTRL_STANDARD or DCS_DRIVES_CTRL to scale the fieldbus values to physical values used in the PLC program.



1.5.6.2.3 Overview of the ACS drives base components according to their call names

Used abbreviations:

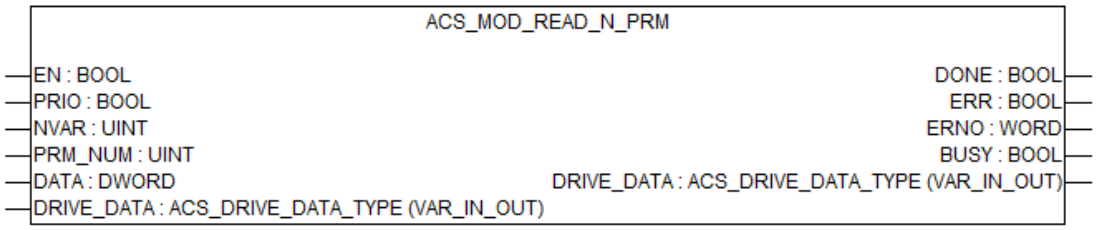
FBhv	Function block with historical values
FBnohv	Function block without historical values
F:	Function
Enum:	Enumeration
Struct:	Structure
Visu:	Visualization

VE name	Type	Function
ACS_CW_VISU_PH ↳ *Chapter 1.5.6.2.8.9 “ACS_CW_VISU_PH visualization for the ABB drives profile control word” on page 2286*	Visu	Visualization for the ABB Drives Profile Control Word
ACS_DRIVE_CONFIG_TYPE ↳ *Chapter 1.5.6.2.6.1 “ACS_DRIVE_CONFIG_TYPE structure including configurations parameters of the ACS3XX drive” on page 2252*	Struct	Structure Including Configurations Parameters of the ACS3XX Drive
ACS_DRIVE_DATA_TYPE ↳ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*	Struct	Structure to Exchange Data between function blocks for 1 drive
ACS_DRIVES_CTRL_ENG ↳ *Chapter 1.5.6.2.4.5 “ACS_DRIVES_CTRL_ENG ” on page 2226*	FBhv	Control of ACS Drives with ABB-Drives Profile
ACS_DRIVES_CTRL_ENG_VISU_PH ↳ *Chapter 1.5.6.2.8.4 “ACS_DRIVES_CTRL_ENG_VISU_PH faceplate of function block ACS_DRIVES_CTRL_ENG” on page 2266*	Visu	Faceplate of function block ACS_DRIVES_CTRL_ENG
ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234*	FBhv	Control of ACS Drives with ABB-Drives Profile
ACS_DRIVES_CTRL_STANDARD_VISU_PH ↳ *Chapter 1.5.6.2.8.5 “ACS_DRIVES_CTRL_STANDARD_VISU_PH faceplate of function block ACS_DRIVES_CTRL_STANDARD” on page 2271*	Visu	Faceplate for the function block
ACS_DRIVES_CTRL_STANDARD_GEN ↳ *Chapter 1.5.6.2.4.7 “ACS_DRIVES_CTRL_STANDARD_GEN” on page 2241*	FBhv	Control of ACS Drives with ABB-Drives Profile
ACS_DRIVES_CTRL_STANDARD_GEN_VISU_P ↳ *Chapter 1.5.6.2.8.6 “ACS_DRIVES_CTRL_STANDARD_GEN_VISU_PH faceplate for the function block” on page 2275*	Visu	Faceplate for the function block

VE name	Type	Function
ACS_DRIVE_ENUM *Chapter 1.5.6.2.5.1 “ACS_DRIVE_ENUM enumerations to select the type of drive used” on page 2251*	Enum	Enumerations to Select the Type of Drive Used
ACS_MOD_PRM_NUM_32BIT *Chapter 1.5.6.2.4.3 “ACS_MOD_PRM_NUM_32BIT” on page 2219*	FBhv	Function creates the Modbus Address 32-bit Parameters of ACSxxx Drives
ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212*	FBhv	Read 1 or More Parameters from an ACS Drive via Modbus RTU
ACS_MOD_READ_N_PRM_VISU_PH *Chapter 1.5.6.2.8.1 “ACS_MOD_READ_N_PRM_VISU_PH faceplate for the function block ACS_MOD_READ_N_PRM” on page 2256*	Visu	Faceplate of function block ACS_MOD_READ_N_PRM
ACS_MOD_TOKEN_TYPE *Chapter 1.5.6.2.6.3 “ACS_MOD_TOKEN_TYPE structure to exchange the internal Modbus token for Modbus RTU communication with more than 1 Drive” on page 2254*	Struct	Structure to Exchange the Internal Modbus Token for Modbus RTU Communication With More Than 1 Drive
ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*	FBhv	Write 1 or More Parameters to an ACS Drive via Modbus RTU
ACS_MOD_WRITE_N_PRM_VISU_PH *Chapter 1.5.6.2.8.2 “ACS_MOD_WRITE_N_PRM_VISU_PH faceplate for the function block ACS_MOD_WRITE_N_PRM” on page 2260*	Visu	Faceplate for the function block ACS_MOD_WRITE_N_PRM
ACS_REF_SCALING *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246*	FBhv	Scaling for ACS Reference and Actual Values
ACS_REF_SCALING_VISU_PH *Chapter 1.5.6.2.8.7 “ACS_REF_SCALING_VISU_PH faceplate for the function block ACS_REF_SCALING” on page 2279*	Visu	Faceplate of function block ACS_REF_SCALING
ACS_SW_VISU_PH *Chapter 1.5.6.2.8.8 “ACS_SW_VISU_PH visualization for the ABB drives profile status word” on page 2282*	Visu	Visualization for the ABB Drives Profile Status Word
ACS3XX_DRIVES_CTRL_BASIC *Chapter 1.5.6.2.4.4 “ACS3XX_DRIVES_CTRL_BASIC” on page 2220*	FBhv	Control of ACS3XX Drives with ABB-Drives Profile
ACS3XX_DRIVES_CTRL_BASIC_VISU_PH *Chapter 1.5.6.2.8.3 “ACS3XX_DRIVES_CTRL_BASIC_VISU_PH faceplate of function block ACS3XX_DRIVES_CTRL_BASIC” on page 2262*	Visu	Faceplate for the function block ACS3XX_DRIVES_CTRL_BASIC

1.5.6.2.4 Function blocks

ACS_MOD_READ_N_PRM



Function block ACS_MOD_READ_N_PRM is used for reading n parameters from a drive via Modbus.

Function block information

Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_MOD_READ_N_PRM reads n parameters of the drive. The number of parameters to be read is specified at the input NVAR. The first parameter number is specified at the input PRM_NUM. All parameters must be accessible from consecutive Modbus registers in the drive. The values of the parameters are stored in the PLC memory area, defined at the input DATA. The values in the PLC memory area are updated when the read job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

As long as the EN = TRUE a new read job is requested each time the further read job is terminated.

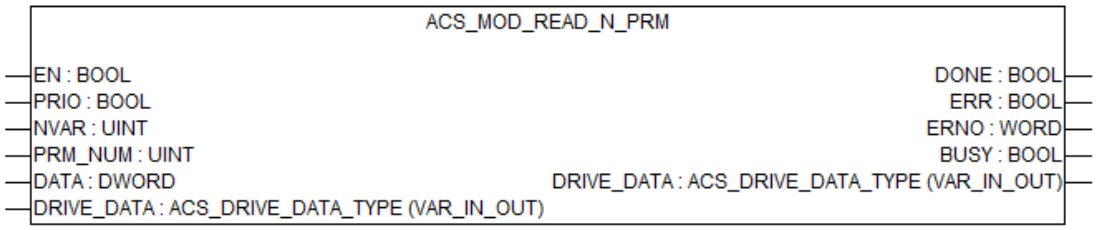
The Modbus job is started from the communication block which is connected to the same DRIVE_DATA variable. It uses the Modbus function code 03 (Read n words).

The drive (Modbus slave) from which the parameter is read is specified at this communication block.

The communication blocks are available from other ACSDrivesComXXX libraries e.g. ACS_3XX_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293* in ACSDrivesComModRTU_AC500_V20.lib. ↳ *Chapter 1.5.6.3 “ACS / DCS Drives communication via Modbus RTU library” on page 2288* or ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367* in ACSDrivesComModTCP_V22.lib ↳ *Chapter 1.5.6.4 “ACS / DCS drives communication via Modbus TCP library” on page 2359*ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392* in ACSDrivesComModTCP_Ext_AC500_V24.lib.

If the connected communication block (via DRIVE_DATA variable) is disabled or not parametrized correctly and EN = TRUE, the variables or arrays that are attached via DATA input are reset to zero. This can be checked by the ERNO output.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

If a TRUE state is applied to the input EN, a request to perform a Modbus read job is set to the DRIVE_DATA variable.

All further inputs are read in.

If the input values are valid, a request to perform a Modbus job is send to the communication block via the DRIVE_DATA variable.

If at least 1 input is invalid, no job is generated and the error is displayed at the outputs ERR and ERNO instead.

If the state of EN stays TRUE a new read job is requested each time the previous job is terminated, indicated by the DONE = TRUE flag.

PRIO (priority)

Data type: BOOL

Input PRIO is reserved for future usage. It can be left open.

NVAR (number of variables)

Data type: UINT, Default value: 1, Range: 1 to 125

Input NVAR defines the number of variables to be read by the function block.

If the read job was finished successfully the first 15 parameter values are written to the internal array VALUE, which can be made visible in the visualization element ACS_MOD_READ_N_PRM_VISU_PH *Chapter 1.5.6.2.8.1 “ACS_MOD_READ_N_PRM_VISU_PH faceplate for the function block ACS_MOD_READ_N_PRM” on page 2256*or used via <instance of the function block>.VALUE[1..15].

PRM_NUM (parameter number)

Data type: UINT, Range 0 to 65535

For 16bit Parameters it is set in the format 100 * GG + ii as decimal integer value. (GG = parameter group, ii = parameter index of the drive, e.g. for parameter 12.02, 100 * GG + ii = 1202.)

(GG = parameter group, ii = parameter index of the drive)

For 32bit Parameters it must be set according to the formula 20000 + 200 x GG + 2 x ii.(see description of related drive fieldbus module), e.g. for parameter 14.54 = 22908.

This calculation is provided in the function ACS_MOD_PRM_NUM_32BIT *Chapter 1.5.6.2.4.3 “ACS_MOD_PRM_NUM_32BIT” on page 2219*, which output can then be connected to the PRM_NUM input. The Modbus address which is used at the Communication Block is PRM_NUM - 1.

> *The parameter number must be set prior or at the same cycle when input EN is set to TRUE.*

The Modbus address which is used at the communication block is PRM_NUM - 1.

DATA (data)

Data type: DWORD

At input DATA, the address of the first operand in the PLC is specified to which the data read by the slave should be stored.

For this purpose it is necessary that all PLC variables, which are written to the drive, have consecutive addresses. This can be obtained by declaration of each variable within the %MW0.xxx area or declaration of an array containing all variables.

Declaration of each variable has the advantage, that the types (integer or word) can be selected individually.

DRIVE_DATA (drive data)

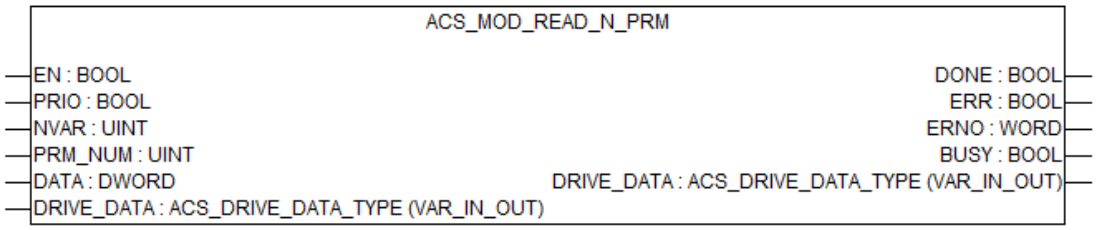
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

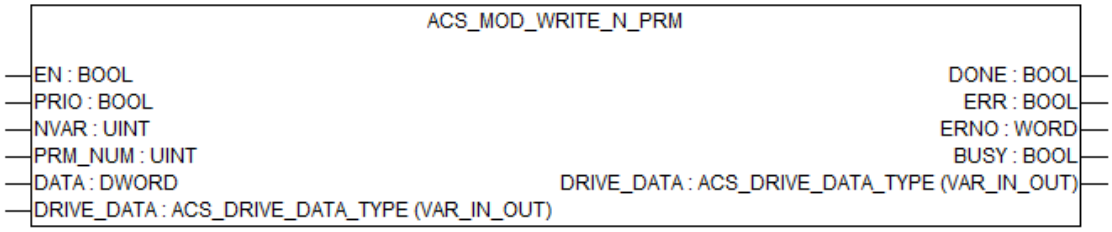
Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

Function call in ST:

```
dwAcsModReadNPrm_DATA              := ADR(aiAcsModReadNPrm_VALUE[1]);


aiAcsModReadNPrm_DATA (EN          := xAcsModReadPrm_EN,
                       PRM_NUM     := uiAcsModReadPrm_PRM_NUM,
                       DATA        := dwAcsModReadNPrm_DATA,
                       DRIVE_DATA := tsDriveData);


xAcsModReadNPrm_DONE := AcsModReadNPrm.DONE;
xAcsModReadNPrm_ERR  := AcsModReadNPrm.ERR;
wAcsModReadNPrm_ERNO := AcsModReadNPrm.ERNO;
xAcsModReadNPrm_BUSY := AcsModReadNPrm.BUSY;
```

ACS_MOD_WRITE_N_PRM



Function block ACS_MOD_WRITE_N_PRM is used for writing n parameters to a drive via Modbus.

Function block information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_MOD_WRITE_PRM writes n parameter to the drive. The number of parameters to be written is specified must be available in the PLC memory area, defined at the input DATA. The write job has been performed without error if DONE=TRUE and ERR=FALSE.

To start a new write job the input EN has to be set from FALSE to TRUE (edge sensitive).

The Modbus job is started from the Communication Block which is connected to the same DRIVE_DATA variable. It uses the Modbus function code 16 (Write n words).

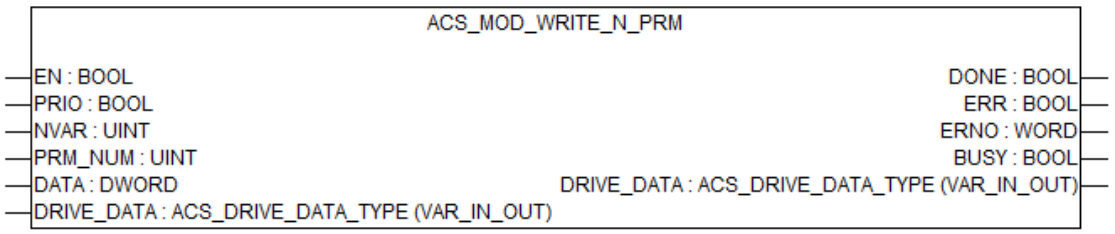
The drive (Modbus Slave) to which the parameter is written is specified at this communication block.

The communication blocks are available from other ACSDrivesComXXX libraries e.g. ACS3XX_COM_MOD_RTU *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293* in ACSDrivesComModRTU_AC500_V20.lib *Chapter 1.5.6.3 “ACS / DCS Drives communication via Modbus RTU library” on page 2288*. or ACS_COM_MOD_TCP_ENHANCED *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367* in ACSDrivesComModTCP_V22.lib *Chapter 1.5.6.4 “ACS / DCS drives communication via Modbus TCP library” on page 2359*. See description of these blocks for further information.

> *ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set. Please see drive manuals and following table which parameter has to be set.*

Save valid parameters to permanent memory in drive	**ACS3XX, ACX550, ACQ810, ACS850, ACSM1, ACS800**	**ACS880**
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1

Input description



EN (enable) Data type: BOOL

If a FALSE->TRUE edge is applied to input EN, all further inputs are read in (edge sensitive).

If the input values are valid, a request telegram is sent to the communication block via the DRIVE_DATA variable.

If at least 1 input is invalid, no telegram is generated and the error is displayed at the outputs ERR and ERNO instead.

While the request is processed, state changes at input EN are recognized but not evaluated.

PRIO (priority) Data type: BOOL

Input PRIO is reserved for future usage. It can be left open.

NVAR (number of variables)

Data type: UINT, Default value: 1, Range: 1 to 125

Input NVAR defines the number of variables to be read by the function block.

If the read job was finished successfully the first 15 parameter values are written to the internal array VALUE, which can be made visible in the visualization element ACS_MOD_READ_N_PRM_VISU_PH *Chapter 1.5.6.2.8.1 “ACS_MOD_READ_N_PRM_VISU_PH faceplate for the function block ACS_MOD_READ_N_PRM” on page 2256*or used via <instance of the function block>.VALUE[1..15].

PRM_NUM (parameter number)

Data type: UINT, Range 0 to 65535

For 16bit Parameters it is set in the format 100 * GG + ii as decimal integer value. (GG = parameter group, ii = parameter index of the drive, e.g. for parameter 12.02, 100 * GG + ii = 1202.)

(GG = parameter group, ii = parameter index of the drive)

For 32bit Parameters it must be set according to the formula 20000 + 200 x GG + 2 x ii.(see description of related drive fieldbus module), e.g. for parameter 14.54 = 22908.

This calculation is provided in the function ACS_MOD_PRM_NUM_32BIT *Chapter 1.5.6.2.4.3 “ACS_MOD_PRM_NUM_32BIT” on page 2219*, which output can then be connected to the PRM_NUM input. The Modbus address which is used at the Communication Block is PRM_NUM - 1.

The parameter number must be set prior or at the same cycle when input EN is set to TRUE.

The Modbus address which is used at the communication block is PRM_NUM - 1.

DATA (data)

Data type: DWORD

At input DATA, the address of the first operand in the PLC is specified to which the data read by the slave should be stored.

For this purpose it is necessary that all PLC variables, which are written to the drive, have consecutive addresses. This can be obtained by declaration of each variable within the %MW0.xxx area or declaration of an array containing all variables.

Declaration of each variable has the advantage, that the types (integer or word) can be selected individually.

DRIVE_DATA (drive data)

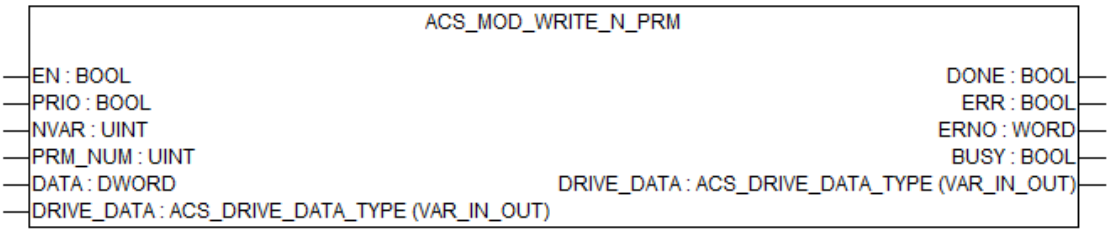
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

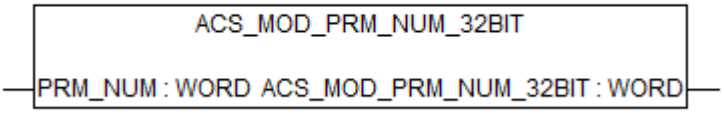
BUSY (busy) Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

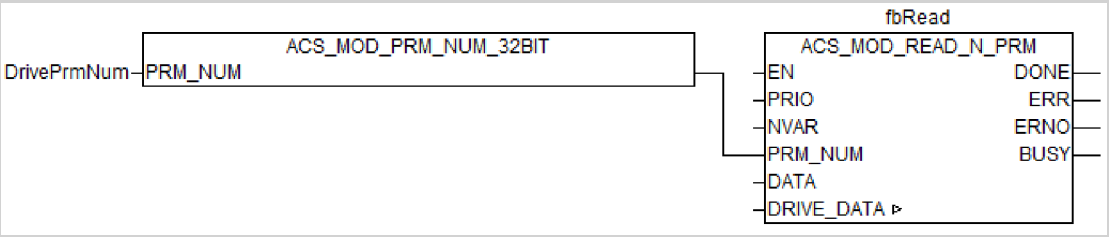
Function call in ST:

```
dwAcsModWriteNPrm_DATA := ADR(aiAcsModWriteNPrm_VALUE[1]);


AcsModWriteNPrm (EN         := xAcsModWritePrm_EN,
                 PRM_NUM    := uiAcsModWritePrm_PRM_NUM,
                 DATA       := dwAcsModWriteNPrm_DATA,
                 DRIVE_DATA := tsDriveData);


xAcsModWritePrm_DONE := AcsModWritePrm.DONE;
xAcsModWritePrm_ERR  := AcsModWritePrm.ERR;
xAcsModWritePrm_ERNO := AcsModWritePrm.ERNO;
xAcsModWritePrm_BUSY := AcsModWritePrm.BUSY;
```

ACS_MOD_PRM_NUM_32BIT



Example



Function to create the Modbus address for a 32-bit parameter of ACSxxx drives.

Function information

Available as of runtime system:	V1.3.2 and above
Included in library:	ACSDrivesBase_AC500_V20.lib
Function type:	Function block with historical values.

Function description

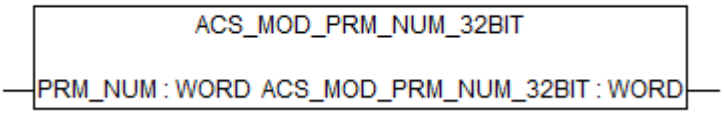
This function will create the Modbus address for 32-bit parameters of the ACSXXX / ACx550 drives using the format for 16 bit parameter as input.

Output is the calculated address for 32-bit parameters according the following rule:

ACS_MOD_PRM_NUM_32BIT = 20000 + (200 * GG) + (2 * ii)

Where GG = parameter group and ii = the index of input PRM_NUM in format of 4 digits: GGii.

Input description



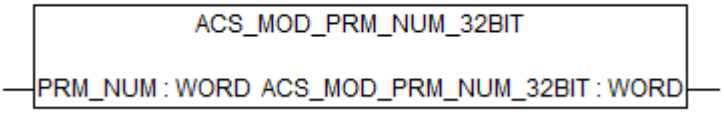
PRM_NUM (parameter number)

Data type: WORD

Input for function. Parameter number in format: 100 * GG + ii

(GG = parameter group, ii = parameter index of the drive, e.g. for parameter 12.02, 100 * GG + ii = 1202.)

Output description



ACS_MOD_PRM_NUM_32BIT (acs mod prm num 32 bit)

Data type: WORD

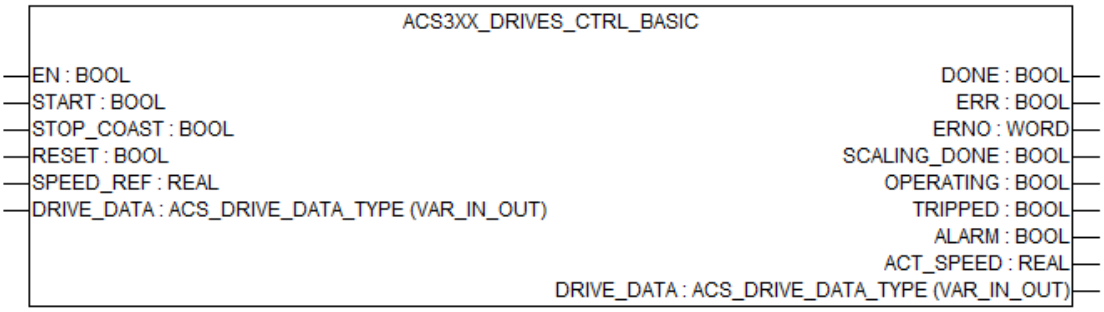
Output of function. Modbus address for 32-bit parameter access, which can be used at input PRM_NUM of ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* or ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215*.

E.g. ACQ810 Par.14.54 (32-bit parameter) Modbus Address is 22908. This is the output of the function if PRM_NUM input is set to 1454.

Function call in ST

```
uiAcsModReadNPrm_PRM_NUM  := ACS_MOD_PRM_NUM_32BIT (

          PAR_NUM         := wAcsModPrmNum32Bit_PAR_NUM);
```

ACS3XX_DRIVES_CTRL_BASIC

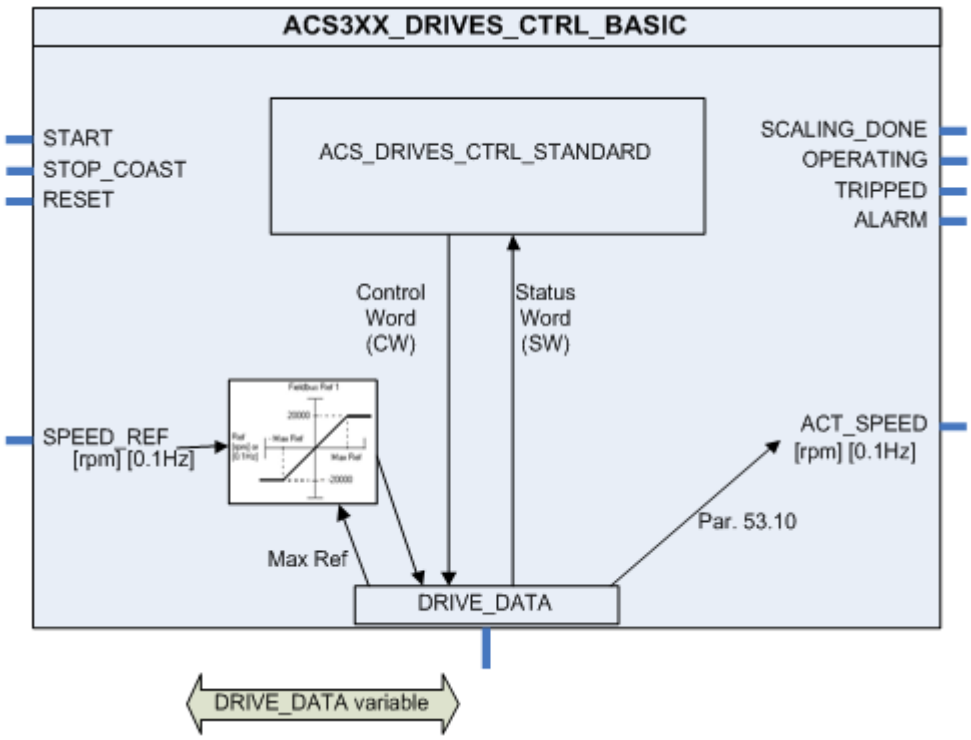


Function block ACS3XX_DRIVES_CTRL_BASIC is used for the control of ACS3XX and ACX550 drives with ABB Drives profile.

Function block information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type:	Function block with historical values

Block description



Function block ACS_DRIVES_CTRL_BASIC is used for controlling ACS3XX and ACX550 drives with ABB Drives profile.

The Function block provides basic start/stop signals, basic diagnosis signals and scaling of the SPEED_REF input and ACT_SPEED to the ACS fieldbus scaling range -20000 to +20000.

The scaling is done according to the drive parameter 11.05 (Ref1 Max) which is read automatically at the first start of the function block form the drive. A successful reading of this and some other configuration parameters is indicated by the output SCALING_DONE. These parameter values are available at the DRIVE_DATA variable element “config” e.g. tsDriveData.config. iRefScaleMax.

The function block internally calls the function block ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234* which is included in the same library.

If the connected communication block (via DRIVE_DATA variable) is disabled or not parametrized correctly all outputs except DONE, ERR and ERNO are reset to zero. This can be checked by the ERNO output.

PRECONDITIONS

The function block is only working for ACS3XX (ACS310, ACS350 and ACS355) and ACX550 (ACS550 and ACH550) drives via Modbus RTU communication.

> *This function block cannot be used with ACS380 drive.*
>
> *Instead for ACS380 drive use ACS_DRIVES_CTRL_STANDARD function block.*

The data transfer to the ACS3XX / ACX550 drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to a communication block such as ACS3XX_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293* or an ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*, ACS_COM_MOD_RTU_ENHANCED ↳ *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives*

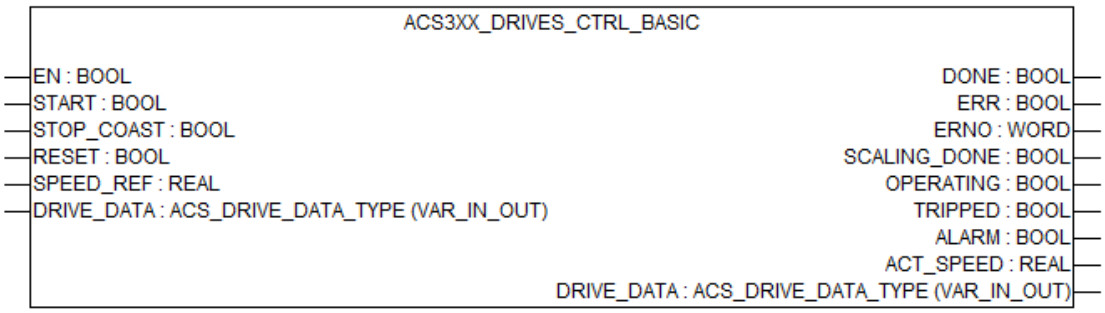
via Modbus RTU using ABB drives profile enhanced" on page 2312, ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 "ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP" on page 2360*or ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 "ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP" on page 2367*.

The drive parameters to select the source of the Control Word must be set to fieldbus control. Also the source for reference 1 and the RESET signal must be set to fieldbus.

For an ACS3XX / ACX550 drive the setting must be as follows:

```
10.01 = 10          (EXT1 COMMANDS = COMM)

11.02 = 8           (EXT1/EXT2 SEL = COMM)

11.03 = 8           (REF1 SEL = COMM)

16.04 = 8           (FAULT RESET SEL = COMM)

53.05 = 2           (ABB drives profile)
```

Input description



EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0.

After a rising edge (FALSE -> TRUE) of input EN some parameters are read from the ACS3XX / ACX550 drive. Included is parameter 11.05 which reflects the maximum speed reference value of the drive. After successful reading of the ACS3XX / ACX550 parameters output SCALING_DONE is set to TRUE.

START (start) Data type: BOOL

With a rising edge at input START (FALSE -> TRUE) the ACS Drive is started. If START = FALSE the drive is stopped along the normal stop ramp, defined in the drive (e.g. Par. 22.03 for ACS355).

> *According to the ABB Drives Profile a new rising edge of START input will be ignored by the drive until zero speed was reached.*

After an emergency stop of the drive a new rising edge of the START input is required to restart the drive.

STOP_COAST (stop coast)

Data type: BOOL

Input STOP_COAST = TRUE will coast the drive immediately (STOP_COAST = inverted Bit 3 of the Control Word – INHIBIT_OP). Setting STOP_COAST=FALSE will restart the drive immediately without need of an rising edge at input START.

RESET (reset)

Data type: BOOL

Input RESET is packed to bit 7 of the Control Word as long as input USE_CW=FALSE. RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

SPEED_REF (speed reference)

Data type: REAL

Input SPEED_REF must be given according to ACS3XX / ACX550 drive motor control mode. The motor control mode is set in the drive (e.g. Par. 99.04 for ACS355). In case of a scalar motor control mode (99.04 = 3 for ACS355) the input SPEED_REF reflects the frequency reference in 0.1 Hz (10 = 1Hz). In case of vector speed control mode (99.04 = 2 for ACS35) the input SPEED_REF reflects the speed reference in rpm.

The function block includes a linear scaling of input SPEED_REF to the fieldbus equivalent value between -20000 to +20000. 20000 = the value defined in the drive parameter 11.05 (Ref1 Max). This parameter 11.05 is automatically read by the function block after a rising edge is given at the input START.

The input range is from negative to positive value of maximum speed (Par.11.05 “Ref1 Max”), e.g. –1500 .. +1500 rpm.

If the input value exceeds the input range, the speed reference value is limited to maximum value. ERR is set to TRUE, ERNO is set to the message that the input of SPEED_REF is out of range. Nevertheless, the function block is processed normally.

DRIVE_DATA (drive data)

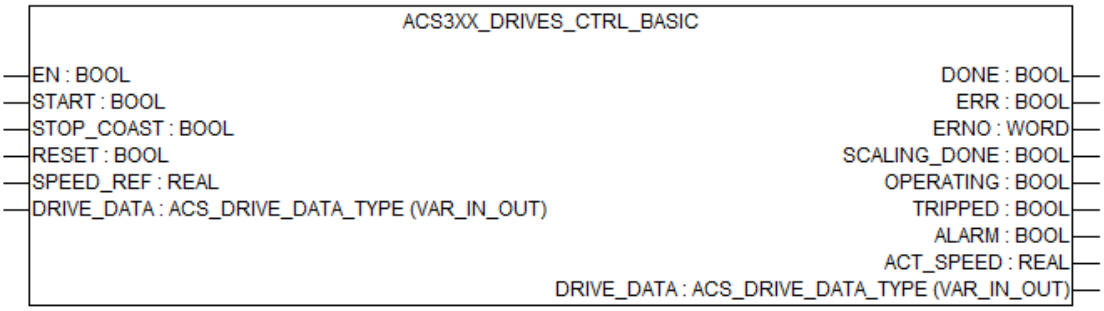
Data type: ACS_DRIVE_DATA_TYPE ↳ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

SCALING_DONE (scaling done)

Data type: BOOL

Output SCALING_DONE is set to TRUE when the reading of the scaling parameters from the ACS3XX / ACX550 drive has been finished. SCALING_DONE will be reset to FALSE if EN = FALSE. With a rising edge of EN the reading of the scaling parameters will be started. If the reading of a scaling parameter is not successful the ERR output is set to TRUE and the ERNO indicates that the reading of the scaling parameters was erroneous.

Scaling parameters for ACS3XX / ACX550 are written to the DRIVE_DATA variable structure under DRIVE_DATA.config (see description of DRIVE_DATA). They are only valid if SCALING_DONE = TRUE.

The parameters can be accessed via the DRIVE_DATA variable using the Point-Operator. Scaling parameters are Par.11.05 (Ref1 Max); Par 99.04 (Motor Ctrl Mode), Par 99.07 (Motor Nom Freq), Par. 99.08 (Motor Nom Speed).

OPERATING (operating)

Data type: BOOL

Output OPERATING=TRUE indicates that the drive is controlled by this function block. The drive is enabled and running (Status Word of drive bits: RDY_ON = TRUE, RDY_RUN = TRUE, RDY_REF = TRUE).

If the drive is controlled from another control place, e.g. local panel, output OPERATING is reset to FALSE, even if the drive might be enabled and running.

To get a control place independent indication, evaluate if the following bits of input SW are set: SW.0 AND SW.1 AND SW.2

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates that the drive is tripped (Bit 3 in the Status Word from the drive).

ALARM (alarm)

Data type: BOOL

Output ALARM=TRUE indicates that the drive has an alarm (Bit 7 in the Status Word from the drive).

ACT_SPEED (actual speed)

Data type: REAL

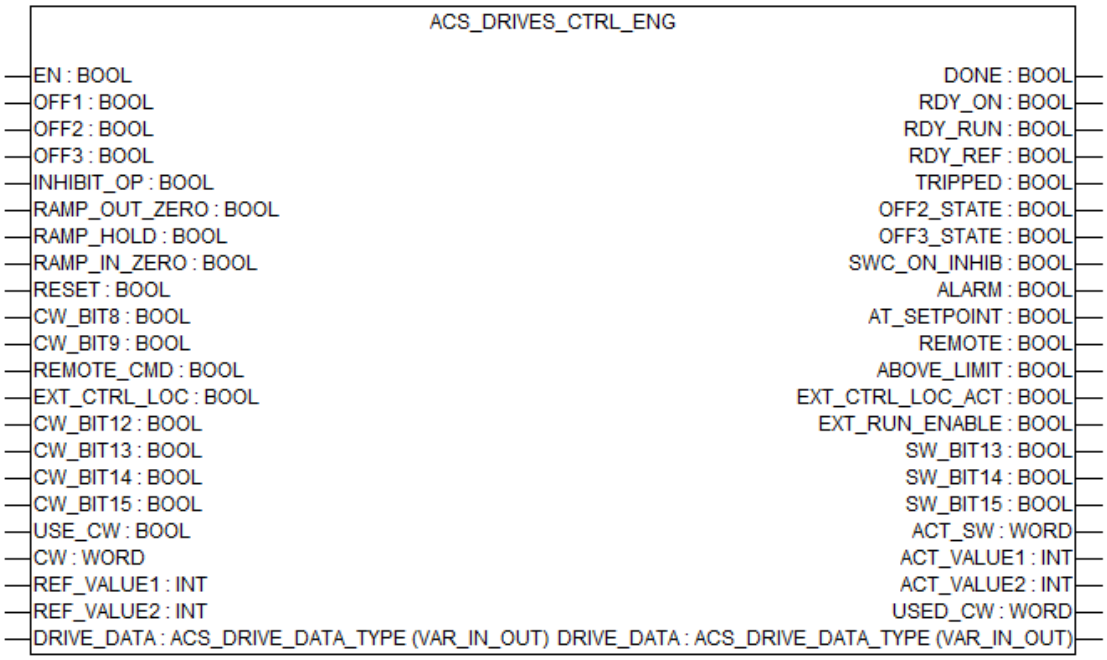
Output ACT_SPEED returns the actual speed value from the drive. The scaling depends on the drive settings.

Drive	Fieldbus	Parameter Settings	Scaling
ACS3XX / ACX550	Modbus RTU	53.10=101	Actual speed [rpm] 1 = 1 rpm
ACS3XX / ACX550	Modbus RTU	53.10=103	Absolute frequency 10 = 1 Hz

Function call in ST

```
ACS3XXDrivesCtrlBasic  (EN                  :=  xACS3XXDrivesCtrlBasic_EN,
                        START          := xACS3XXDrivesCtrlBasic_START,
                       STOP_COAST    := xACS3XXDrivesCtrlBasic_STOP_COAST,
                        RESET          := xACS3XXDrivesCtrlBasic_RESET,
                        SPEED_REF     := rACS3XXDrivesCtrlBasic_SPEED_REF,
                                       DRIVE_DATA     :=  tsDriveData);


xACS3XXDrivesCtrlBasic_DONE               :=  ACS3XXDrivesCtrlBasic.DONE;
xACS3XXDrivesCtrlBasic_ERR                 :=  ACS3XXDrivesCtrlBasic.ERR;
wACS3XXDrivesCtrlBasic_ERNO               :=  ACS3XXDrivesCtrlBasic.ERNO;
xACS3XXDrivesCtrlBasic_SCALING_DONE := ACS3XXDrivesCtrlBasic.SCALING_DONE;
xACS3XXDrivesCtrlBasic_OPERATING      :=  ACS3XXDrivesCtrlBasic.OPERATING;
xACS3XXDrivesCtrlBasic_TRIPPED          :=  ACS3XXDrivesCtrlBasic.TRIPPED;
xACS3XXDrivesCtrlBasic_ALARM             :=  ACS3XXDrivesCtrlBasic.ALARM;
rACS3XXDrivesCtrlBasic_ACT_SPEED      :=  ACS3XXDrivesCtrlBasic.ACT_SPEED;
```

ACS_DRIVES_CTRL_ENG



Function block ACS_DRIVES_CTRL_ENG is used as an engineering interface for ACS Drives with ABB Drives profile.

Function block information

Available as of runtime system:	V1.3.2
Available as of runtime system:	ACSDrivesBase_AC500_V20.lib
Function block type	Function block with historical values

Block description

Function block ACS_DRIVES_CTRL_ENG is used as an engineering interface for ACS Drives with ABB Drives profile.

Inputs REF_VALUE1 and REF_VALUE2 and the generated Control Word are written to the DRIVE_DATA variable which transfers these values to a communication function block, e.g. ACS3XX_MOD-MAST_RTU. That communication function block writes them to the drive. In the same way ACT_VALUE1, ACT_VALUE2 and the Status Word are transferred from the communication function block to the ACS_DRIVES_CTRL_ENG block where they are written to the outputs.

As long as EN=FALSE no values are read nor written to the DRIVE_DATA variable.

The Control Word can be generated in 2 ways. First way is to set the single bits of the Control Word separately at the inputs OFF1 ... CW_BIT15 while the input USE_CW=FALSE. Second way is to set the input USE_CW=TRUE and write the Control Word as a whole word directly to the input CW. The generated Control Word is written to the DRIVE_DATA variable and for diagnosis purpose also available at output USED_CW.

PRECONDITIONS

The data transfer to the ACS drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to an ACS3XX_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293*, ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*, ACS_COM_MOD_RTU_ENHANCED ↳ *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312*, ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* or ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter*

1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367, ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385* or ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*.

Table 129: Necessary configuration of parameters in the drive

Drive Parameter	ACS3XX, ACX550	ACS850, ACQ810 embedded	ACS850, ACQ810	ACSM1	ACS580, ACS880	ACS800	Comment
EXT1 COMMANDS	10.01 = COMM	10.01 = FBA	10.01 = FB	10.01 = FBA	20.01 = Fieldbus A	10.01 = COMM.CW	Fieldbus interface as source for start and stop.
EXT1/ EXT2 SEL	11.02 = COMM	12.01 = P.02.36 bit 15	12.01 = P.02.22 bit 15	34.01 = P.02.12 bit 15	19.11 = MCW Bit11 (06.01)	11.02 = COMM.CW	Fieldbus interface as source to switch to EXT2 control place.
REF1 SELECT	11.03 = COMM	21.01 = EFB REF1	21.01 = FBA REF1	24.01 = FBA Ref1	22.11 = FBA ref1	11.03 = COMM.REF	Fieldbus interface as source for speed reference.
FAULT RESET SEL	16.04 = COMM	10.10 = P.02.36 bit 8	10.10 = P.02.22 bit 8	10.08 = P.02.12 bit 8	31.11 = P.06.01 bit 7	16.04 = COMM.CW	Fieldbus interface as source for fault reset.
PROFILE	53.05 = ABB DRV FULL	58.06 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	98.07 = ABB DRIVES	Control Profile to ABB Drives Profile classic or enhanced.

Only one of the function blocks ACS3XX_DRIVES_CTRL_BASIC, ACS_DRIVES_CTRL_STANDARD or ACS_DRIVES_CTRL_ENG must be enabled (EN=TRUE) at the same time.

The data transfer to the ACS drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to an ACS3XX_COM_MOD_RTU or an ACS_COM_xxx function block.

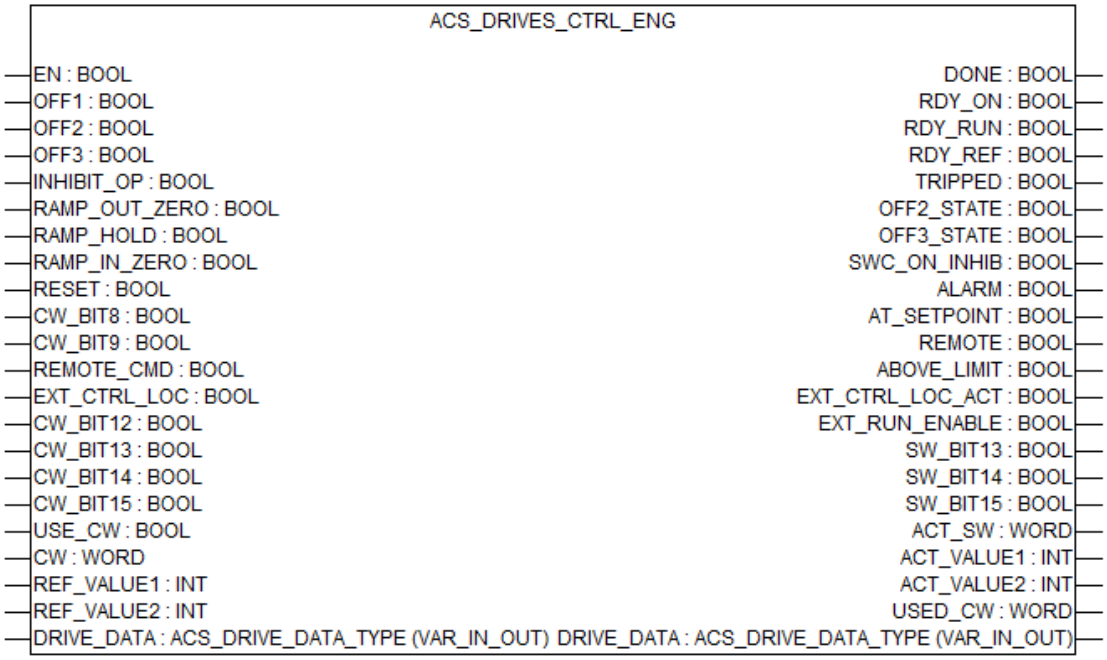
The input and output names of the bits of the Control Word and Status Word reflect the functions used with ABB Drive profile. So the block should be used with ABB Drives profile setting in the drive.

The function block does not execute any functionality expect data transfer to and from the DRIVE_DATA variable. So there is no special drive parameter setting necessary to use this block.

The programmer using this block should have a detailed understanding of how to set the Control Word according to the Status Word and the description of the used drive.

For standard speed & torque control application it is recommended to the use the ACS_DRIVES_CTRL_STANDARD instead.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0, with the exception of the USED_CW output and the DRIVE_DATA.cw, which are set to 1024 (hex 0400 - only remote bit).

OFF1 (off1) Data type: BOOL

Input OFF1 is packed to bit 0 of the Control Word as long as input USE_CW=FALSE.

OFF2 (off2) Data type: BOOL

Input OFF2 is packed to bit 1 of the Control Word as long as input USE_CW=FALSE.

OFF3 (off3) Data type: BOOL

Input OFF3 is packed to bit 2 of the Control Word as long as input USE_CW=FALSE.

INHIBIT_OP (inhibit operation)
Data type: BOOL
Input INHIBIT_OP is packed to bit 3 of the Control Word as long as input USE_CW=FALSE.

RAMP_OUT_ZERO (ramp out zero)
Data type: BOOL
Input RAMP_OUT_ZERO is packed to bit 4 of the Control Word as long as input USE_CW=FALSE.

RAMP_HOLD (ramp hold)
Data type: BOOL
Input RAMP_HOLD is packed to bit 5 of the Control Word as long as input USE_CW=FALSE.

RAMP_IN_ZERO (ramp in zero)
Data type: BOOL
Input RAMP_IN_ZERO is packed to bit 6 of the Control Word as long as input USE_CW=FALSE.

RESET (reset)
Data type: BOOL
Input RESET is packed to bit 7 of the Control Word as long as input USE_CW=FALSE. RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

CW_BIT8 (control word bit 8)
Data type: BOOL
Input CW_BIT8 is packed to bit 8 of the Control Word as long as input USE_CW=FALSE.

CW_BIT9 (control word bit 9)
Data type: BOOL
Input CW_BIT9 is packed to bit 9 of the Control Word as long as input USE_CW=FALSE.

REMOTE (remote command)
Data type: BOOL
Input REMOTE_CMD is packed to bit 10 of the Control Word as long as input USE_CW=FALSE.

EXT_CTRL_LOC (external control location)
Data type: BOOL
Input EXT_CTRL_LOC is packed to bit 11 of the Control Word as long as input USE_CW=FALSE.

Input EXT_CTRL_LOC=TRUE sets the control place for the drive to EXT2 (Bit 11 in the Control Word to the drive). For normal control of the drive from the PLC EXT_CTRL_LOC should be set to FALSE.

CW_BIT12 (control word bit 12)
Data type: BOOL
Input CW_BIT12 is packed to bit 12 of the Control Word as long as input USE_CW=FALSE.

CW_BIT13 (control word bit 13)
Data type: BOOL
Input CW_BIT13 is packed to bit 13 of the Control Word as long as input USE_CW=FALSE.

CW_BIT14 (control word bit 14)
Data type: BOOL
Input CW_BIT14 is packed to bit 14 of the Control Word as long as input USE_CW=FALSE.

CW_BIT15 (control word bit 15)

Data type: BOOL

Input CW_BIT15 is packed to bit 15 of the Control Word as long as input USE_CW=FALSE.

USE_CW (use control word)

Data type: WORD

Output USED_CW shows the used Control Word which is written to the DRIVE_DATA variable.

E.g. the packed word from the input bits OFF1 .. CW_BIT15 if USE_CW=FALSE.

CW (control word)

Data type: WORD

Input CW is written to the DRIVE_DATA variable as Control Word as long as USE_CW=TRUE.

REF_VALUE1 (reference value 1)

Data type: INT

Input REF_VALUE1 is written to the DRIVE_DATA variable as Reference Value 1 (SPEED_REF).

REF_VALUE2 (reference value 2)

Data type: INT

Input REF_VALUE2 is written to the DRIVE_DATA variable as Reference Value 2. Input REF_VALUE2 must be given in fieldbus equivalent value between -10000 .. +10000. 10000 = the value defined in the drive as Ref2 Max (e.g. Par.11.08 Ref2 Max for ACS3XX or Par Gr. 32.for ACSM1).

DRIVE_DATA (drive data)

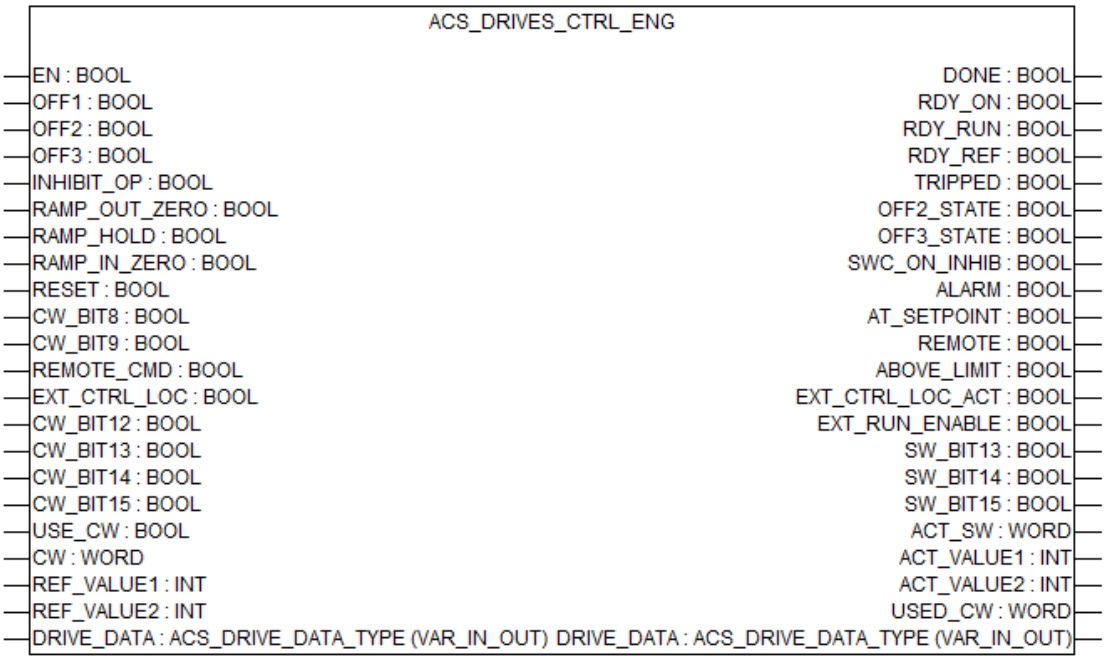
Data type: ACS_DRIVE_DATA_TYPE ↯ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

RDY_ON (ready on)

Data type: BOOL

Output RDY_ON is bit 0 of actual Status Word from the drive.

RDY_RUN (ready run)

Data type: BOOL

Output RDY_RUN is bit 1 of actual Status Word from the drive.

RDY_REF

Data type: BOOL

Output RDY_REF (ready reference) is bit 2 of actual Status Word from the drive.

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates that the drive is tripped (Bit 3 in the Status Word from the drive).

OFF2_STATE (off state 2)

Data type: BOOL

Output OFF_STATE2 is bit 4 of actual Status Word from the drive.

OFF3_STATE (off state 3)

Data type: BOOL

Output OFF_STATE3 is bit 5 of actual Status Word from the drive.

SWC_ON_INHIB (switch on inhibit) Data type: Bool

Output SWC_ON_INHIB is bit 6 of actual Status Word from the drive.

ALARM (alarm) Data type: BOOL

Output ALARM=TRUE indicates that the drive has an alarm (Bit 7 in the Status Word from the drive).

AT_SETPOINT (actual tolerance setpoint) Data type: BOOL

Output AT_SETPOINT is bit 8 of actual Status Word from the drive.

REMOTE (remote) Data type: BOOL

Output REMOTE is bit 9 of actual Status Word from the drive.

ABOVE_LIMIT (above limit) Data type: BOOL

Output ABOVE_LIMIT is bit 10 of actual Status Word from the drive.

EXT_CTRL_LOC_ACT (external control location active) Data type: BOOL

Output EXT_CTRL_LOC_ACT is bit 11 of actual Status Word from the drive. Output EXT_CTRL_LOC_ACT = TRUE indicates that the drive is controlled from the control place EXT2 (Bit 11 in Status Word from the drive). EXT_CTRL_LOC_ACT might be set to TRUE due to the input EXT_CTRL_LOC = TRUE.

EXT_RUN_ENABLE (external run enable) Data type: BOOL

Output EXT_RUN_ENABLE is bit 12 of actual Status Word from the drive. Output EXT_RUN_ENABLE=TRUE indicates that the drive received External Run Enable signal. For normal control of the drive from the PLC EXT_RUN_ENABLE must be set to TRUE.

SW_BIT13 (status word bit 13) Data type: BOOL

Output SW_BIT13 is bit 13 of actual Status Word from the drive.

SW_BIT14 (status word bit 14) Data type: BOOL

Output SW_BIT14 is bit 14 of actual Status Word from the drive.

SW_BIT15 (status word bit 15) Data type: BOOL

Output SW_BIT15 is bit 15 of actual Status Word from the drive.

ACT_SW (actual status word) Data type: WORD

Output ACT_SW is the actual Status Word from the drive.

ACT_VALUE1 (actual value 1) Data type: INT

Output ACT_VALUE1 is the Actual Value 1 (actual speed value) from the drive.

ACT_VALUE2 (actual value 2)

Data type: INT

Output ACT_VALUE2 is the Actual Value 2 from the drive.

USED_CW (used control word)

Data type: WORD

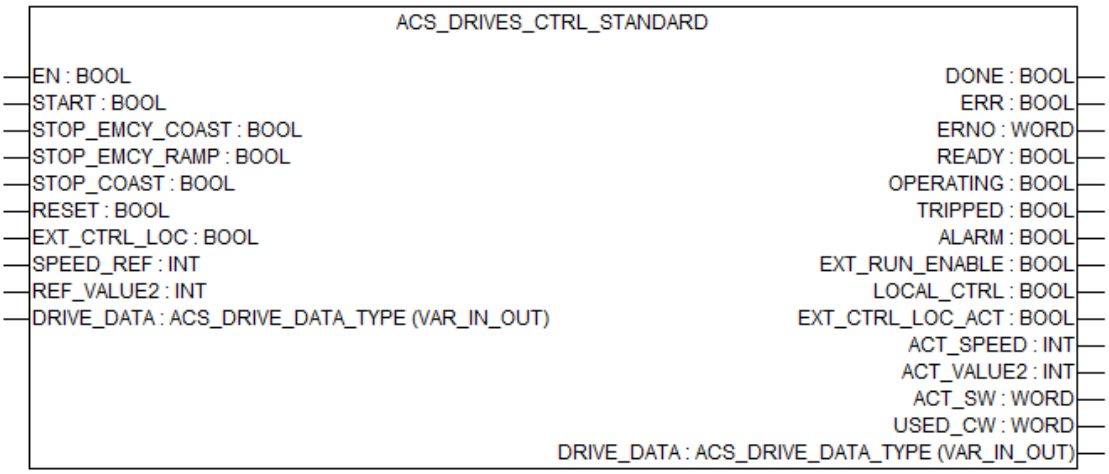
Output USED_CW shows the used Control Word which is written to the DRIVE_DATA variable.

E.g. the packed word from the input bits OFF1 .. CW_BIT15 if USE_CW=FALSE.

Function call in ST

```
ACSDrivesCtrlEng (EN            := xACSDrivesCtrlEng_EN,
                  OFF1          := xACSDrivesCtrlEng_OFF1,
                  OFF2          := xACSDrivesCtrlEng_OFF2,
                  OFF3          := xACSDrivesCtrlEng_OFF3,
                  INHIBIT_OP    := xACSDrivesCtrlEng_INHIBIT_OP,
                  RAMP_OUT_ZERO := xACSDrivesCtrlEng_RAMP_OUT_ZERO,
                  RAMP_HOLD     := xACSDrivesCtrlEng_RAMP_HOLD,
                  RAMP_IN_ZERO  := xACSDrivesCtrlEng_RAMP_IN_ZERO,
                  RESET         := xACSDrivesCtrlEng_RESET,
                  CW_BIT8       := xACSDrivesCtrlEng_CW_BIT8,
                  CW_BIT9       := xACSDrivesCtrlEng_CW_BIT9,
                  REMOTE_CMD    := xACSDrivesCtrlEng_REMOTE_CMD,
                  EXT_CTRL_LOC  := xACSDrivesCtrlEng_EXT_CTRL_LOC,
                  CW_BIT12      := xACSDrivesCtrlEng_CW_BIT12,
                  CW_BIT13      := xACSDrivesCtrlEng_CW_BIT13,
                  CW_BIT14      := xACSDrivesCtrlEng_CW_BIT14,
                  CW_BIT15      := xACSDrivesCtrlEng_CW_BIT15,
                  USE_CW        := xACSDrivesCtrlEng_USE_CW,
                  CW            := xwACSDrivesCtrlEng_CW,
                  REF_VALUE1    := iACSDrivesCtrlEng_REF_VALUE1,
                  REF_VALUE2    := iACSDrivesCtrlEng_REF_VALUE2,
                  DRIVE_DATA    := tsDRIVE_DATA);


xACSDrivesCtrlEng_DONE             := ACSDrivesCtrlEng.DONE;
xACSDrivesCtrlEng_RDY_ON           := ACSDrivesCtrlEng.RDY_ON;
xACSDrivesCtrlEng_RDY_RUN          := ACSDrivesCtrlEng.RDY_DONE;
xACSDrivesCtrlEng_RDY_REF          := ACSDrivesCtrlEng.RDY_REF;
xACSDrivesCtrlEng_TRIPPED          := ACSDrivesCtrlEng.TRIPPED;
xACSDrivesCtrlEng_OFF2_STATE       := ACSDrivesCtrlEng.OFF2_STATE;
xACSDrivesCtrlEng_OFF3_STATE       := ACSDrivesCtrlEng.OFF3_STATE;
xACSDrivesCtrlEng_SWC_ON_INHIB     := ACSDrivesCtrlEng.SWC_ON_INHIB;
xACSDrivesCtrlEng_ALARM            := ACSDrivesCtrlEng.ALARM;
xACSDrivesCtrlEng_AT_SETPOINT      := ACSDrivesCtrlEng.AT_SETPOINT;
xACSDrivesCtrlEng_REMOTE           := ACSDrivesCtrlEng.REMOTE;
xACSDrivesCtrlEng_ABOVE_LIMIT      := ACSDrivesCtrlEng.ABOVE_LIMIT;
xACSDrivesCtrlEng_EXT_CTRL_LOC_ACT :=
ACSDrivesCtrlEng.EXT_CTRL_LOC_ACT;
xACSDrivesCtrlEng_EXT_RUN_ENABLE   := ACSDrivesCtrlEng.EXT_RUN_ENABLE;
xACSDrivesCtrlEng_SW_BIT13         := ACSDrivesCtrlEng.SW_BIT13;
xACSDrivesCtrlEng_SW_BIT14         := ACSDrivesCtrlEng.SW_BIT14;
xACSDrivesCtrlEng_SW_BIT15         := ACSDrivesCtrlEng.SW_BIT15;
wACSDrivesCtrlEng_ACT_SW           := ACSDrivesCtrlEng.ACT_SW;
iACSDrivesCtrlEng_ACT_VALUE1       := ACSDrivesCtrlEng.ACT_VALUE1;
iACSDrivesCtrlEng_ACT_VALUE2       := ACSDrivesCtrlEng.ACT_VALUE2;
wACSDrivesCtrlEng_USED_CW          := ACSDrivesCtrlEng.USED_CW;
```

ACS_DRIVES_CTRL_STANDARD

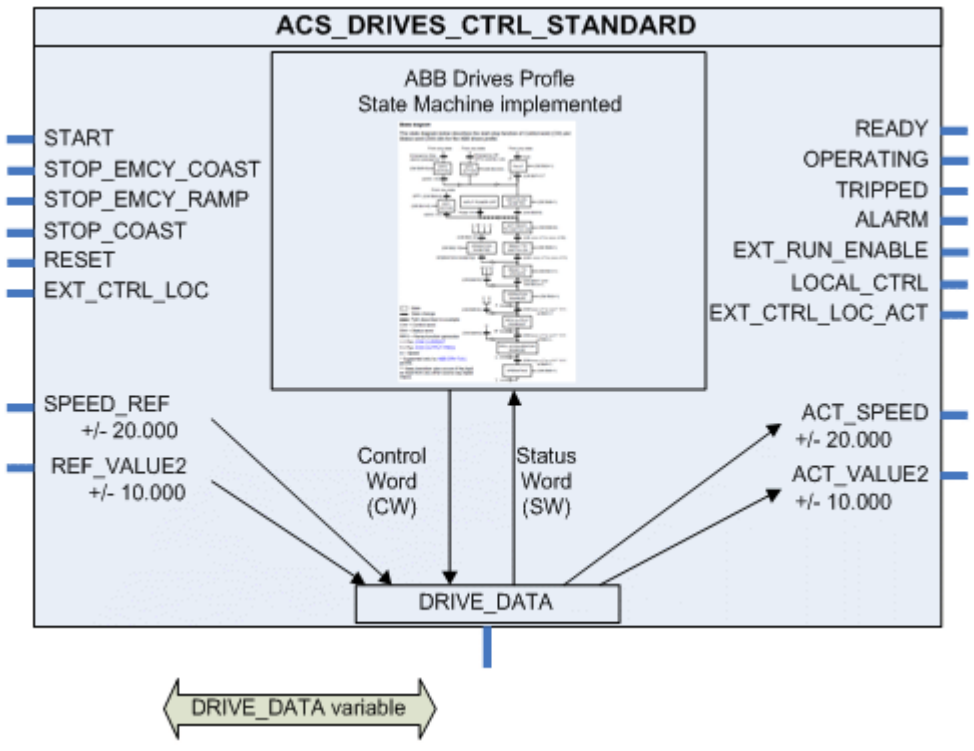


Function block ACS_DRIVES_CTRL_STANDARD is used for controlling ACS Drives with ABB Drives profile.

Function block information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type	Function block with historical values

Block description



Function block ACS_DRIVES_CTRL_STANDARD is used for controlling ACS Drives with ABB Drives profile.

The function block provides standard start/stop signals to control the drive and standard diagnosis signals read from the drive.

According to the start/stop input signals and the actual Status Word (SW), read from the DRIVE_DATA variable, the ABB Drives profile state machine is executed. The Control Word (CW) is built and written to the DRIVE_DATA variable. For diagnosis purpose the CW is also written to the output USED_CW.

SPEED_REF input has to be given in the ACS fieldbus range.of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355)

ACT_SPEED provides the actual speed in the ACS fieldbus range of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355).

ACS_REF_SCALING *Chapter 1.5.6.2.4.8 “ACS_REF_SCALING” on page 2246* function block could be used to scale the fieldbus range to a physical value.

If the connected communication block (via DVIVE_DATA variable) is disabled or not parameterized correctly all outputs except DONE, ERR and ERNO are reset to zero. This can be checked by the ERNO output.

PRECONDITIONS

The function block is only working for ACS drives using ABB Drive profile.

The data transfer to the ACS drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to an ACS3XX_COM_MOD_RTU *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293*, ACS_COM_MOD_RTU *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*, ACS_COM_MOD_RTU_ENHANCED *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312*, ACS_COM_MOD_TCP *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*, ACS_COM_MOD_TCP_ENHANCED *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*, ACS_COM_MOD_TCPx *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385* or ACS_COM_MOD_TCPx_ENHANCED *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392*.

Please refer the respective drives manual for parameter setting if the drive is not mentioned in below table.

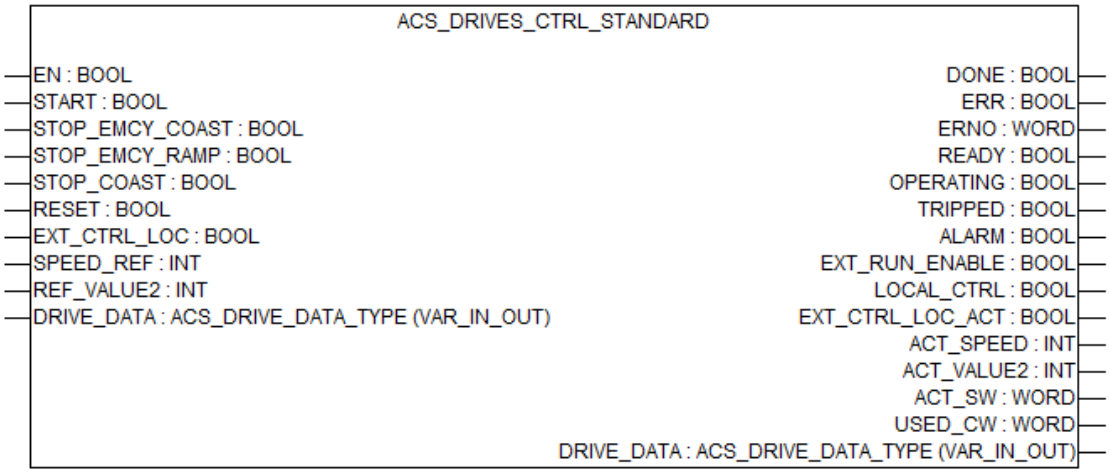
Table 130: Necessary configuration of parameters in the drive

Drive Parameter	ACS3XX, ACX550	ACS850, ACQ810 embedded	ACS850, ACQ810	ACSM1	ACS580, ACS880	ACS800	Comment
EXT1 COMMANDS	10.01 = COMM	10.01 = FBA	10.01 = FB	10.01 = FBA	20.01 = Fieldbus A	10.01 = COMM.CW	Fieldbus interface as source for start and stop
EXT1/ EXT2 SEL	11.02 = COMM	12.01 = P.02.36 bit 15	12.01 = P.02.22 bit 15	34.01 = P.02.12 bit 15	19.11 = MCW Bit11 (06.01)	11.02 = COMM.CW	Fieldbus interface as source to switch to EXT2 control place
REF1 SELECT	11.03 = COMM	21.01 = EFB REF1	21.01 = FBA REF1	24.01 = FBA Ref1	22.11 = FBA ref1	11.03 = COMM.REF	Fieldbus interface as source for speed reference

Drive Parameter	ACS3XX, ACX550	ACS850, ACQ810 embedded	ACS850, ACQ810	ACSM1	ACS580, ACS880	ACS800	Comment
FAULT RESET SEL	16.04 = COMM	10.10 = P.02.36 bit 8	10.10 = P.02.22 bit 8	10.08 = P.02.12 bit 8	31.11 = P.06.01 bit 7	16.04 = COMM.CW	Fieldbus interface as source for fault reset
PROFILE	53.05 = ABB DRV FULL	58.06 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	98.07 = ABB DRIVES	Control Profile to ABB Drives Profile classic or enhanced.

Only one of the function blocks ACS3XX_DRIVES_CTRL_BASIC, ACS_DRIVES_CTRL_STANDARD or ACS_DRIVES_CTRL_ENG must be enabled (EN=TRUE) at the same time.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0, with the exception of the USED_CW output and the DRIVE_DATA.cw, which are set to 1024 (hex 0400 - only remote bit).

START (start) Data type: BOOL

With a rising edge at input START (FALSE -> TRUE) the ACS Drive is started. If START = FALSE the drive is stopped along the normal stop ramp, defined in the drive (e.g. Par. 22.03 for ACS355).

According to the ABB Drives Profile a new rising edge of START input will be ignored by the drive until zero speed was reached.

After an emergency stop of the drive a new rising edge of the START input is required to restart the drive.

STOP_EMCY_COAST (stop emergency coast)

Data type: BOOL

Input STOP_EMCY_COAST=FALSE will coast the drive (bit 1 of the Control Word OFF2). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_COAST=TRUE enables normal operation of the drive.

Default value = TRUE.

STOP_EMCY_RAMP (stop emergency ramp)

Data type: BOOL

Input STOP_EMCY_RAMP = FALSE will stop the drive along the emergency ramp, defined in the drive (bit 2 of the Control Word OFF3). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_RAMP = TRUE enables normal operation of the drive.

Default value = TRUE.

STOP_COAST (stop coast)

Data type: BOOL

Input STOP_COAST = TRUE will coast the drive immediately (STOP_COAST = inverted Bit 3 of the Control Word – INHIBIT_OP). Setting STOP_COAST=FALSE will restart the drive immediately without need of an rising edge at input START.

RESET (reset)

Data type: BOOL

Input RESET is packed to bit 7 of the Control Word as long as input USE_CW=FALSE. RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

EXT_CTRL_LOC (external control location)

Data type: BOOL

Input EXT_CTRL_LOC is packed to bit 11 of the Control Word as long as input USE_CW=FALSE.

Input EXT_CTRL_LOC=TRUE sets the control place for the drive to EXT2 (Bit 11 in the Control Word to the drive). For normal control of the drive from the PLC EXT_CTRL_LOC should be set to FALSE.

SPEED_REF (speed reference)

Data type: REAL

Input SPEED_REF must be given according to ACS3XX / ACX550 drive motor control mode. The motor control mode is set in the drive (e.g. Par. 99.04 for ACS355). In case of a scalar motor control mode (99.04 = 3 for ACS355) the input SPEED_REF reflects the frequency reference in 0.1 Hz (10 = 1Hz). In case of vector speed control mode (99.04 = 2 for ACS35) the input SPEED_REF reflects the speed reference in rpm.

The function block includes a linear scaling of input SPEED_REF to the fieldbus equivalent value between -20000 to +20000. 20000 = the value defined in the drive parameter 11.05 (Ref1 Max). This parameter 11.05 is automatically read by the function block after a rising edge is given at the input START.

The input range is from negative to positive value of maximum speed (Par.11.05 “Ref1 Max”), e.g. –1500 .. +1500 rpm.

If the input value exceeds the input range, the speed reference value is limited to maximum value. ERR is set to TRUE, ERNO is set to the message that the input of SPEED_REF is out of range. Nevertheless, the function block is processed normally.

REF_VALUE2 (reference value 2)

Data type: INT

Input REF_VALUE2 is written to the DRIVE_DATA variable as Reference Value 2. Input REF_VALUE2 must be given in fieldbus equivalent value between -10000 .. +10000. 10000 = the value defined in the drive as Ref2 Max (e.g. Par.11.08 Ref2 Max for ACS3XX or Par Gr. 32.for ACSM1).

DRIVE_DATA (drive data)

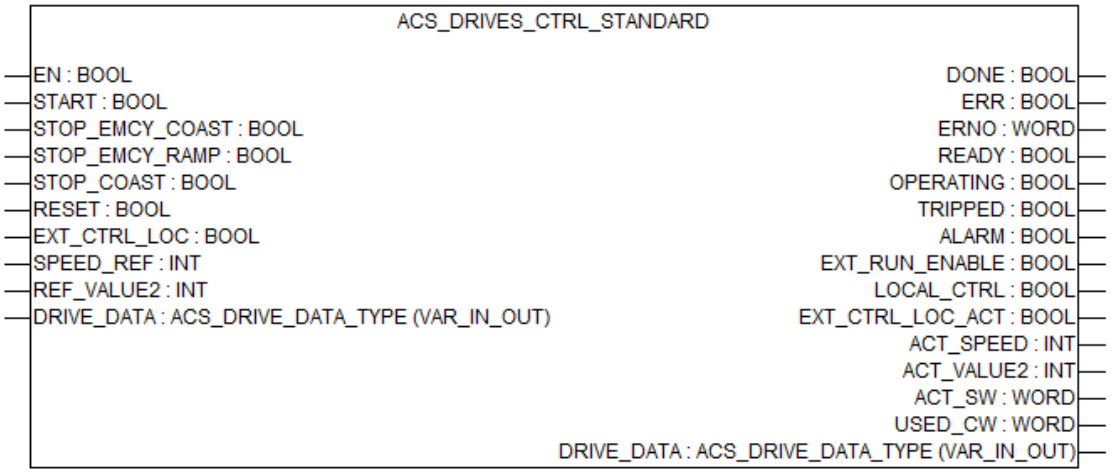
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

READY (ready)

Data type: BOOL

Output READY=TRUE indicates that the drive is ready to switch on (Bit 0 in the status word from the drive).

OPERATING (operating)

Data type: BOOL

Output OPERATING=TRUE indicates that the drive is controlled by this function block. The drive is enabled and running (Status Word of drive bits: RDY_ON = TRUE, RDY_RUN = TRUE, RDY_REF = TRUE).

If the drive is controlled from another control place, e.g. local panel, output OPERATING is reset to FALSE, even if the drive might be enabled and running.

To get a control place independent indication, evaluate if the following bits of input SW are set: SW.0 AND SW.1 AND SW.2

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates that the drive is tripped (Bit 3 in the Status Word from the drive).

ALARM (alarm)

Data type: BOOL

Output ALARM=TRUE indicates that the drive has an alarm (Bit 7 in the Status Word from the drive).

EXT_RUN_ENABLE (external run enable)

Data type: BOOL

Output EXT_RUN_ENABLE is bit 12 of actual Status Word from the drive. Output EXT_RUN_ENABLE=TRUE indicates that the drive received External Run Enable signal. For normal control of the drive from the PLC EXT_RUN_ENABLE must be set to TRUE.

LOCAL_CTRL (local control)

Data type: BOOL

Output LOCAL_CTRL = TRUE indicates that the drive is not controlled from remote control e.g. PLC (LOCAL_CTRL = inverted bit 9 in the Status Word from the drive - REMOTE). LOCAL_CTRL might be set to TRUE due to no connection to the drive or the drive was set to local control via the drive panel or a drive configuration tool from PC.

EXT_CTRL_LOC_ACT (external control location active)

Data type: BOOL

Output EXT_CTRL_LOC_ACT is bit 11 of actual Status Word from the drive. Output EXT_CTRL_LOC_ACT = TRUE indicates that the drive is controlled from the control place EXT2 (Bit 11 in Status Word from the drive). EXT_CTRL_LOC_ACT might be set to TRUE due to the input EXT_CTRL_LOC = TRUE.

ACT_SPEED (actual speed)

Data type: INT, Range: -20000 .. +20000

Output ACT_SPEED returns the actual speed value from the drive. The scaling depends on the drive settings.

Drive	Fieldbus	Parameter Settings	Scaling
ACS3XX / ACX550	Modbus RTU	53.10=101	Actual speed [rpm] 1 = 1 rpm
ACS3XX / ACX550	Modbus RTU	53.10=103	Absolute frequency 10 = 1 Hz

ACT_VALUE2 (actual value 2)

Data type: INT

Output ACT_VALUE2 is the Actual Value 2 from the drive.

ACT_SW (actual status word)

Data type: WORD

Output ACT_SW is the actual Status Word from the drive.

USED_CW (used control word)

Data type: WORD

Output USED_CW shows the used Control Word which is written to the DRIVE_DATA variable.

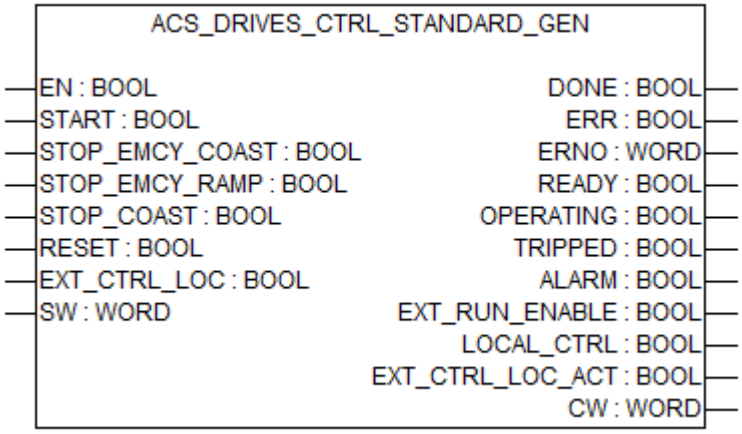
E.g. the packed word from the input bits OFF1 .. CW_BIT15 if USE_CW=FALSE.

Function call in ST

```
ACS_Drives_Ctrl_Standard (EN               :=
xACS_Drives_Ctrl_Standard_EN,
                         START             := xACS_Drives_Ctrl_Standard_START,
                         STOP_EMCY_COAST   := xACS_Drives_Ctrl_Standard_STOP_E
                         STOP_EMCY_RAMP    := xACS_Drives_Ctrl_Standard_STOP_E
                         STOP_COAST        :=
xACS_Drives_Ctrl_Standard_STOP_COAST,
                         RESET             := xACS_Drives_Ctrl_Standard_RESET,
                         EXT_CTRL_LOC      := xACS_Drives_Ctrl_Standard_EXT_CT
                         SPEED_REF         := iACS_Drives_Ctrl_Standard_SPEED_
                         REF_VALUE2        :=
iACS_Drives_Ctrl_Standard_REF_VALUE2,
                         DRIVE_DATA        := tsDRIVE_DATA);


xACS_Drives_Ctrl_Standard_DONE             :=
ACS_Drives_Ctrl_Standard.DONE;
xACS_Drives_Ctrl_Standard_ERR              :=
ACS_Drives_Ctrl_Standard.ERR;
wACS_Drives_Ctrl_Standard_ERNO             :=
ACS_Drives_Ctrl_Standard.ERNO;
xACS_Drives_Ctrl_Standard_READY            :=
ACS_Drives_Ctrl_Standard.READY;
xACS_Drives_Ctrl_Standard_OPERATING        :=
ACS_Drives_Ctrl_Standard.OPERATING;
xACS_Drives_Ctrl_Standard_TRIPPED          :=
```

```
ACS_Drives_Ctrl_Standard.FAULT;
xACS_Drives_Ctrl_Standard_ALARM            :=
ACS_Drives_Ctrl_Standard.WARN;
xACS_Drives_Ctrl_Standard_EXT_RUN_ENABLE   :=
ACS_Drives_Ctrl_Standard.EXT_RUN_ENABLE;
xACS_Drives_Ctrl_Standard_LOCAL_CTRL       :=
ACS_Drives_Ctrl_Standard.LOCAL_CTRL;
xACS_Drives_Ctrl_Standard_EXT_CTRL_LOC_ACT :=
ACS_Drives_Ctrl_Standard.EXT_CTRL_ACT;
iACS_Drives_Ctrl_Standard_ACT_SPEED        :=
ACS_Drives_Ctrl_Standard.ACT_SPEED;
iACS_Drives_Ctrl_Standard_ACT_VALUE2       :=
ACS_Drives_Ctrl_Standard.ACT_VALUE2;
wACS_Drives_Ctrl_Standard_ACT_SW           :=
ACS_Drives_Ctrl_Standard.ACT_SW;
wACS_Drives_Ctrl_Standard_USED_CW          :=
ACS_Drives_Ctrl_Standard.USED_CW;
```

ACS_DRIVES_CTRL_STANDARD_GEN

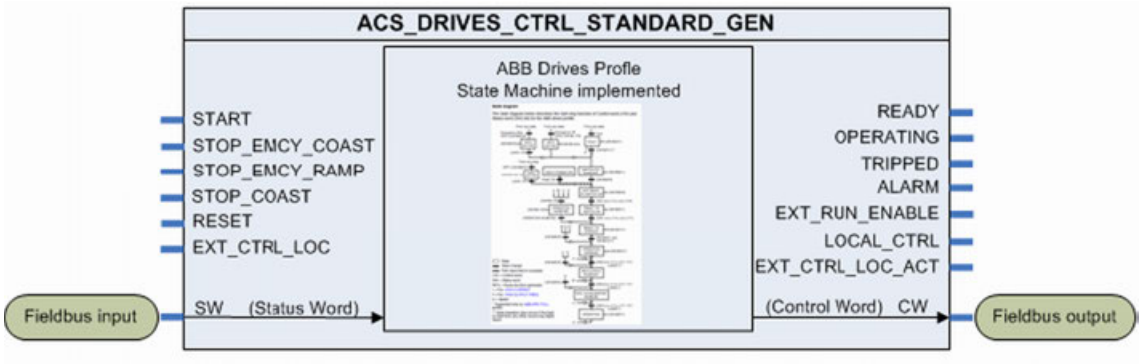


Function block ACS_DRIVES_CTRL_STANDARD_GEN is used for controlling ACS Drives with ABB Drives profile. The Status Word (SW) and Control Word (CW) can be transferred via any generic fieldbus.

Function block information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type	Function block with historical values

Block description



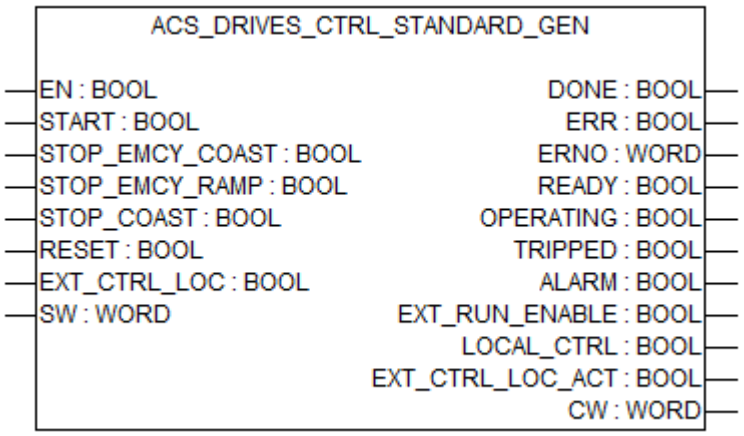
Function block ACS_DRIVES_CTRL_STANDARD_GEN is used for controlling ACS Drives with ABB Drives profile. User has to connect Status Word from the fieldbus to this function block input. Function block generates Control Word, which has to be connected to the fieldbus.

Please refer the respective drives manual for parameter setting, if the drive is not mentioned in below table.

Table 131: Necessary configuration of parameters in the drive

Drive Parameter	ACS3XX, ACX550	ACS850, ACQ810	ACSM1	ACS580, ACS880	ACS800	Comment
EXT1 COM-MANDS	10.01 = COMM	10.01 = FB	10.01 = FBA	20.01 = Fieldbus A	10.01 = COMM.CW	Fieldbus interface as source for start and stop
EXT1/EXT2 SEL	11.02 = COMM	12.01 = P.02.22 bit 15	34.01 = P.02.12 bit 15	19.11 = MCW Bit11 (06.01)	11.02 = COMM.CW	Fieldbus interface as source to switch to EXT2 con-trol place
REF1 SELECT	11.03 = COMM	21.01 = FBA REF1	24.01 = FBA Ref1	22.11 = FBA ref1	11.03 = COMM.REF	Fieldbus interface as source for speed refer-ence
FAULT RESET SEL	16.04 = COMM	10.10 = P.02.22 bit 8	10.08 = P.02.12 bit 8	31.11 = P.06.01 bit 7	16.04 = COMM.CW	Fieldbus interface as source for fault reset
PROFILE	53.05 = ABB DRV FULL	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	51.02 = ABB Drives classic/ enhanced	98.07 = ABB DRIVES	Control Pro-file to ABB Drives Pro-file classic or enhanced.

Input description



EN (enable)

Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0, with the exception of the CW output to 1024 (hex 0400 - only remote bit).

START (start)

Data type: BOOL

With a rising edge at input START (FALSE -> TRUE) the ACS Drive is started. If START = FALSE the drive is stopped along the normal stop ramp, defined in the drive (e.g. Par. 22.03 for ACS355).

According to the ABB Drives Profile a new rising edge of START input will be ignored by the drive until zero speed was reached.

After an emergency stop of the drive a new rising edge of the START input is required to restart the drive.

STOP_EMCY_COAST (stop emergency coast)

Data type: BOOL

Input STOP_EMCY_COAST=FALSE will coast the drive (bit 1 of the Control Word OFF2). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_COAST=TRUE enables normal operation of the drive.

Default value = TRUE.

STOP_EMCY_RAMP (stop emergency ramp)

Data type: BOOL

Input STOP_EMCY_RAMP = FALSE will stop the drive along the emergency ramp, defined in the drive (bit 2 of the Control Word OFF3). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_RAMP = TRUE enables normal operation of the drive.

Default value = TRUE.

STOP_COAST (stop coast)

Data type: BOOL

Input STOP_COAST = TRUE will coast the drive immediately (STOP_COAST = inverted Bit 3 of the Control Word – INHIBIT_OP). Setting STOP_COAST=FALSE will restart the drive immediately without need of an rising edge at input START.

RESET (reset)

Data type: BOOL

Input RESET is packed to bit 7 of the Control Word as long as input USE_CW=FALSE. RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

EXT_CTRL_LOC (external control location)

Data type: BOOL

Input EXT_CTRL_LOC is packed to bit 11 of the Control Word as long as input USE_CW=FALSE.

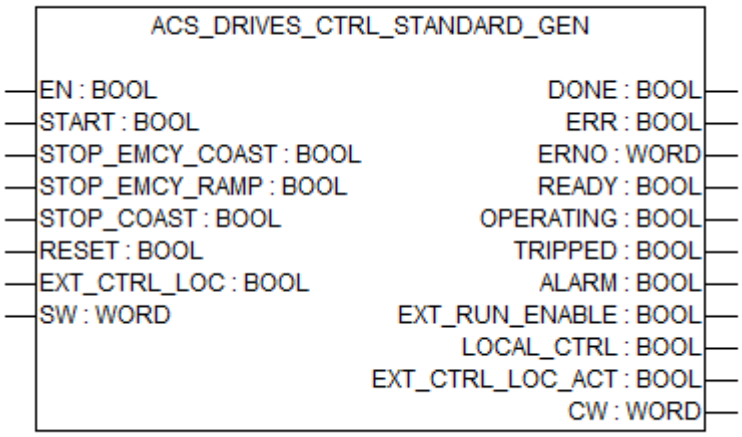
Input EXT_CTRL_LOC=TRUE sets the control place for the drive to EXT2 (Bit 11 in the Control Word to the drive). For normal control of the drive from the PLC EXT_CTRL_LOC should be set to FALSE.

SW (status word)

Data type: WORD

Status Word from drive. Connect Status Word from fieldbus to this input.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

READY (ready)

Data type: BOOL

Output READY=TRUE indicates that the drive is ready to switch on (Bit 0 in the status word from the drive).

OPERATING (operating)

Data type: BOOL

Output OPERATING=TRUE indicates that the drive is controlled by this function block. The drive is enabled and running (Status Word of drive bits: RDY_ON = TRUE, RDY_RUN = TRUE, RDY_REF = TRUE).

If the drive is controlled from another control place, e.g. local panel, output OPERATING is reset to FALSE, even if the drive might be enabled and running.

To get a control place independent indication, evaluate if the following bits of input SW are set: SW.0 AND SW.1 AND SW.2

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates that the drive is tripped (Bit 3 in the Status Word from the drive).

ALARM (alarm)

Data type: BOOL

Output ALARM=TRUE indicates that the drive has an alarm (Bit 7 in the Status Word from the drive).

EXT_RUN_ENABLE (external run enable)

Data type: BOOL

Output EXT_RUN_ENABLE is bit 12 of actual Status Word from the drive. Output EXT_RUN_ENABLE=TRUE indicates that the drive received External Run Enable signal. For normal control of the drive from the PLC EXT_RUN_ENABLE must be set to TRUE.

LOCAL_CTRL (local control)

Data type: BOOL

Output LOCAL_CTRL = TRUE indicates that the drive is not controlled from remote control e.g. PLC (LOCAL_CTRL = inverted bit 9 in the Status Word from the drive - REMOTE). LOCAL_CTRL might be set to TRUE due to no connection to the drive or the drive was set to local control via the drive panel or a drive configuration tool from PC.

EXT_CTRL_LOC_ACT (external control location active)

Data type: BOOL

Output EXT_CTRL_LOC_ACT is bit 11 of actual Status Word from the drive. Output EXT_CTRL_LOC_ACT = TRUE indicates that the drive is controlled from the control place EXT2 (Bit 11 in Status Word from the drive). EXT_CTRL_LOC_ACT might be set to TRUE due to the input EXT_CTRL_LOC = TRUE.

CW (control word)

Data type: WORD

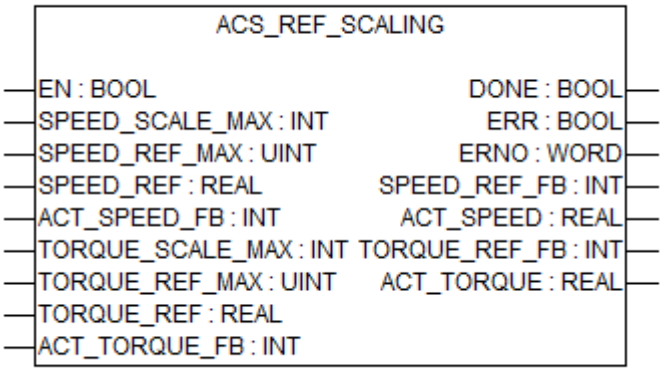
Input CW is written to the DRIVE_DATA variable as Control Word as long as USE_CW=TRUE.

Function call in ST

```
ACS_Drives_Ctrl_Standard_Gen    (EN    :=
xACS_Drives_Ctrl_Standard_Gen_EN,



     START                                          :=
xACS_Drives_Ctrl_Standard_Gen_START,

     STOP_EMCY_COAST    :=
xACS_Drives_Ctrl_Standard_Gen_STOP_EMCY_COAST,

     STOP_EMCY_RAMP        :=
xACS_Drives_Ctrl_Standard_Gen_STOP_EMCY_RAMP,

     STOP_COAST                         :=
xACS_Drives_Ctrl_Standard_Gen_STOP_COAST,
```

```
     RESET                                                    :=
xACS_Drives_Ctrl_Standard_Gen_RESET,


     EXT_CTRL_LOC                :=
xACS_Drives_Ctrl_Standard_Gen_EXT_CTRL_LOC,


     SW                                                                :=
wACS_Drives_Ctrl_Standard_Gen_SW);

xACS_Drives_Ctrl_Standard_Gen_DONE
                 := ACS_Drives_Ctrl_Standard_Gen.DONE;
xACS_Drives_Ctrl_Standard_Gen_ERR
                    := ACS_Drives_Ctrl_Standard_Gen.ERR;
wACS_Drives_Ctrl_Standard_Gen_ERNO
                 := ACS_Drives_Ctrl_Standard_Gen.ERNO;
xACS_Drives_Ctrl_Standard_Gen_READY
              := ACS_Drives_Ctrl_Standard_Gen.READY;
xACS_Drives_Ctrl_Standard_Gen_OPERATING
 := ACS_Drives_Ctrl_Standard_Gen.OPERATING;
xACS_Drives_Ctrl_Standard_Gen_TRIPPED
       := ACS_Drives_Ctrl_Standard_Gen.FAULT;
xACS_Drives_Ctrl_Standard_Gen_ALARM
              := ACS_Drives_Ctrl_Standard_Gen.WARN;
xACS_Drives_Ctrl_Standard_Gen_EXT_RUN_ENABLE            :=
ACS_Drives_Ctrl_Standard_Gen.EXT_RUN_ENABLE;
xACS_Drives_Ctrl_Standard_Gen_LOCAL_CTRL                          :=
ACS_Drives_Ctrl_Standard_Gen.LOCAL_CTRL;
xACS_Drives_Ctrl_Standard_Gen_EXT_CTRL_LOC_ACT    :=
ACS_Drives_Ctrl_Standard_Gen.EXT_CTRL_ACT;
wACS_Drives_Ctrl_Standard_Gen_CW
                      := ACS_Drives_Ctrl_Standard_Gen.CW;
```

ACS_REF_SCALING

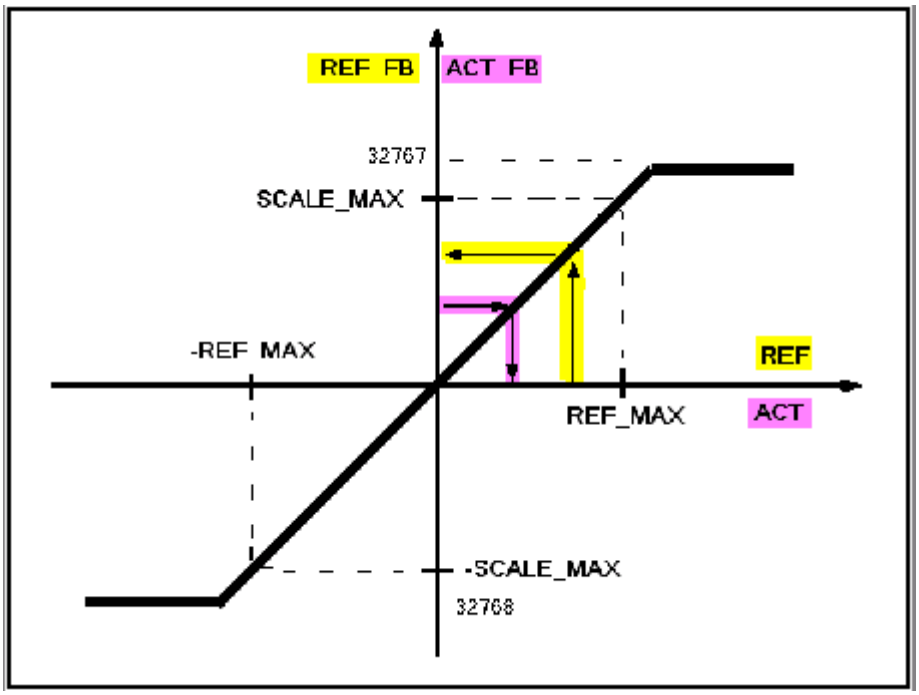


Function block ACS_REF_SCALING is used for scaling of fieldbus integer reference and actual values to real values.

Function block information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib
Function block type:	Function block with historical values

Block description



Function block ACS_REF_SCALING can be used to scale the variables from fieldbus equivalent values to values used in the program. With the scaling also a conversion from INT to REAL is performed.

Fieldbus variables are mostly given in fieldbus equivalent values as INT values.

Reference1 and Actual Value1 (speed) are mostly given in the range of -20000 .. + 20000.

Reference2 and Actual Value2 (torque) are mostly given in the range of -10000 .. + 10000.

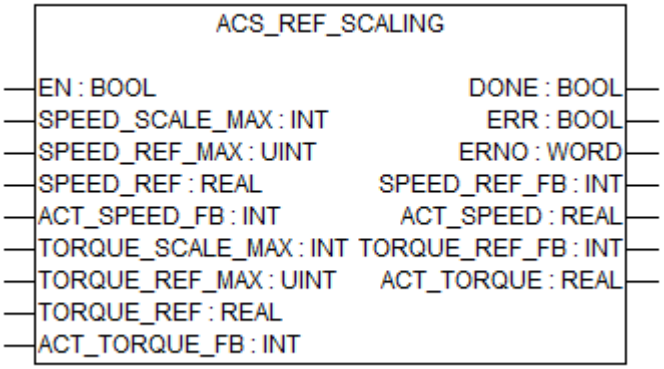
In the program it´s often useful to work with real physical values such as "rpm", "Hz", "%" or "Nm" as REAL values.

The function block provides scaling for two scaling settings: Speed (Reference1 and Actual Value1) and Torque (Reference2 and Actual Value2).

The Scaling maximum and the according reference maximum can be set at the inputs of the function block. This is independent of the drives profile used and could also be utilized for transparent mode or any drive independent linear scaling.

The linear calculation limits the outputs only at the maximum of the INT range at -32768 and +32767. If the scaling would result in a higher value the ERR and ERNO output are indicating the overflow.

Input description



EN (enable)

Data type: BOOL

Enabling of the function block processing.

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is deactivated all outputs are set to zero.

SPEED_SCALE_MAX

Data type: UINT, Default value: 20000; Range: Minimum=1

Scaling maximum for speed. With ABB Drives Profile mostly maximum fieldbus equivalent for maximum of Reference1 and Actual Value1 = +20000.

This value (speed reference maximum) can be adapted at this input by setting a different value, e.g. if transparent mode is used, or speed is used as the Reference2 and Actual Value2.

SPEED_REF_MAX (speed reference maximum)

Data type: INT, Default value: 20000, Range: Minimum=1

Value which corresponds to 100% of the SPEED_REF range.

Should be the speed scaling parameter from the drive. ACS850 = Par.19.01, ACSM1 = Par25.02, ACS355 = Par11.05. e.g. 1500[rpm] or 500 [(0.1)Hz].

SPEED_REF

Data type: REAL

Input SPEED_REF (speed reference) is a REAL value, in rpm or 0.1 Hz or %.

Output SPEED_REF_FB will be calculated according to this input and scaled to +/- SPEED_SCALE_MAX. The scaled value is available at ouput SPEED_REF_FB.

If SPEED_REF exeeds 32767, an error (16#4042) will be indicated and SPEED_REF is set to 32767.

ACT_SPEED_FB (actual speed in fieldbus equivalent)

Data type: INT

Actual speed input from fieldbus in fieldbus equivalent. 100% = +SPEED_SCALE_MAX.

TORQUE_SCALE_MAX (torque reference maximum)

Data type: INT

Scaling maximum for torque. With ABB Drives Profile mostly maximum fieldbus equivalent for maximum of Reference2 and Actual Value2 = +10000.

This value can be adapted at this input by setting a different value, e.g. if transparent mode is used, or torque is used as the Reference1 and Actual Value1.

Default: 10000. Minimum = 1.

TORQUE_REF_MAX (torque reference maximum)

Data type: INT

Value which corresponds to 100% of the TORQUE_REF range.

Working with [Nm] instead of [%] the scaling parameter from the drive should be connected here: e.g. ACS850 = Par.1.29, ACSM1= Par98.1.

Default: 100. Minimum = 1.

TORQUE_REF (torque reference)

Data type: UINT

Torque reference input as real value which is used in the program. Can be in [%] or [Nm].

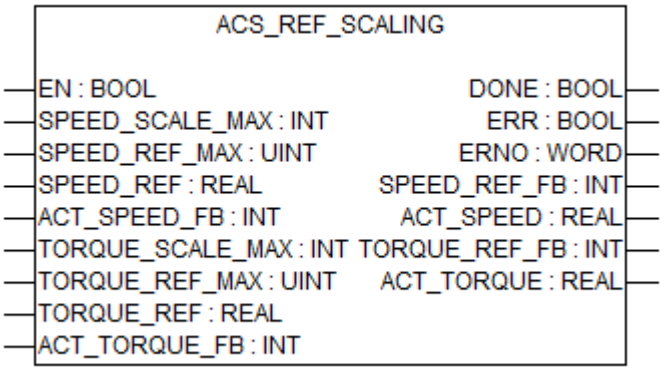
Will be scaled to +/-TORQUE_SCALE_MAX and be available at ouput TORQUE_REF. If TORQUE_REF would be more than 32767 an error 16#4042 will be indicated. TORQUE_REF is set to 32767.

ACT_TORQUE_FB (actual torque in fieldbus equivalent)

Data type: INT

Actual torque input form fieldbus in fieldbus equivalent (+/- TORQUE_SCALE_MAX). TORQUE_SCALE_MAX will be scaled to TORQUE_REF_MAX input and be available at output ACT_TORQUE.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

SPEED_REF_FB (speed reference in fieldbus equivalent)

Data type: INT

Speed reference for fieldbus = SPEED_REF input scaled to (SPEED_SCALE_MAX / SPEED_REF_MAX).

This reference should be connected directly to the fieldbus.

If the result would be higher than 32767 or less than -32768, the output is limit at these values and ERR and ERNO indicating the overflow.

ACT_SPEED (actual speed)

Data type: REAL

Output ACT_SPEED returns the actual speed value from the drive. The scaling depends on the drive settings.

Drive	Fieldbus	Parameter Settings	Scaling
ACS3XX / ACX550	Modbus RTU	53.10=101	Actual speed [rpm] 1 = 1 rpm
ACS3XX / ACX550	Modbus RTU	53.10=103	Absolute frequency 10 = 1 Hz

TORQU_REF_FB (torque reference in fieldbus equivalent)

Data type: INT

Torque reference for fieldbus = TORQUE_REF input scaled to (TORQUE_SCALE_MAX / TORQUE_REF_MAX).

This reference should be connected directly to the fieldbus.

If the result would be higher than 32767 or less than -32768, the output is limit at these values and ERR and ERNO indicating the overflow.

ACT_TORQUE (actual torque)

Data type: REAL

Actual torque = ACT_TORQUE_FB input scaled to (TORQUE_REF_MAX / TORQUE_SCALE_MAX) and converted to REAL.

Function call in ST

```
AcsRefScaling    (EN    := xAcsRefScaling_EN,

           SPEED_SCALE_MAX        := iAcsRefScaling_SPEED_SCALE_MAX,

           SPEED_REF_MAX                :=
uiAcsRefScaling_SPEED_REF_MAX,

           SPEED_REF                                  :=
rAcsRefScaling_SPEED_REF,

           ACT_SPEED_FB                    :=
iAcsRefScaling_ACT_SPEED_FB,

           TORQUE_SCALE_MAX    := iAcsRefScaling_TORQUE_SCALE_MAX,

           TORQUE_REF_MAX            :=
uiAcsRefScaling_TORQUE_REF_MAX,

           TORQUE_REF                              :=
rAcsRefScaling_TORQUE_REF,

           ACT_TORQUE_FB               :=
iAcsRefScaling_ACT_TORQUE_FB);

xAcsRefScaling_DONE                                        :=
AcsRefScaling.DONE;
xAcsRefScaling_ERR                                            :=
AcsRefScaling.ERR;
wAcsRefScaling_ERNO                                        :=
AcsRefScaling.ERNO;
iAcsRefScaling_SPEED_REF_FB        := AcsRefScaling.SPEED_REF_FB;
rAcsRefScaling_ACT_SPEED                   :=
AcsRefScaling.ACT_SPEED;
iAcsRefScaling_TORQUE_REF_FB    := AcsRefScaling.TORQUE_REF_FB;
rAcsRefScaling_ACT_TORQUE               := AcsRefScaling.ACT_TORQUE;
```

1.5.6.2.5 Enumerations

ACS_DRIVE_ENUM enumerations to select the type of drive used

Enumeration information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Enumeration	Value	Description
ACS_DRIVE_ACS800	1	ACS800
ACS_DRIVE_ACSM1	2	ACSM1
ACS_DRIVE_ACS350	3	ACS350
ACS_DRIVE_ACS355	4	ACS355
ACS_DRIVE_ACS310	5	ACS310
ACS_DRIVE_ACS550	6	ACS550
ACS_DRIVE_ACH550	7	ACH550
ACS_DRIVE_ACQ810	8	ACQ810
ACS_DRIVE_ACS850	9	ACS850
ACS_DRIVE_ACS880	10	ACS880
ACS_DRIVE_ACS580	11	ACS580
ACS_DRIVE_DCS800	12	DCS800
ACS_DRIVE_DCS550	13	DCS550
ACS_DRIVE_ACH580	14	ACH580
ACS_DRIVE_ACS380	15	ACS380
ACS_DRIVE_ACS480	16	ACS480
ACS_DRIVE_ACQ580	17	ACQ580

ACS_PB_PN_PRM_TYPE_ENUM

Enumeration information

Available as of runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Enumeration	Value		Description
	Hex	DEC	
ACS_INT16_PRM	16#03	3	Integer of 16 bits
ACS_INT32_PRM	16#04	4	Integer of 32 bits
ACS_UINT16_PRM	16#06	6	Unsigned 16 bits
ACS_UINT32_PRM	16#07	7	Unsigned 32 bits
ACS_ZERO_VALUE	16#40	64	Zero value returned, Ex.Parameter is not availabe inside the Drive

Enumeration	Value		Description
ACS_BYTE_PRM	16#41	65	Byte
ACS_WORD_PRM	16#42	66	Word
ACS_DWORD_PRM	16#43	67	Double Word
ACS_ERROR_VALUE	16#44	68	Error returned, Ex.Not able to read / write because of communication fault or read or write protected parameter etc

Description

For writing any parameter using PROFIBUS/PROFINET DPV1 function blocks, user need to speficy the data type in function block input. For example, if user is writing one parameter to drive which is double word then user need to specify 16#43 in DATA input abyPrmType.

1.5.6.2.6 Structures

ACS_DRIVE_CONFIG_TYPE structure including configurations parameters of the ACS3XX drive

Structure ACS_DRIVE_CONFIG_TYPE is used in the ACS_DRIVE_DATA_TYPE as subelement and contains configuration parameters of the drive.

Structure information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Visible Variable	Type	Default Value	Description
iMotorCtrlMode	INT	0	0=no information, 1=scalar control, 2=vector control or DTC
iRefScaleMax	INT	0	Reference scaling = +20000
iRefScaleMin	INT	0	Reference scaling = -20000
iActScaleMax	INT	0	Actual value scaling = +20000
iActScaleMin	INT	0	Actual value scaling = -20000
iMotorNomFrqHz	INT	0	Nominal motor frequency [0.1 Hz]
iMotorNomSpeedRpm	INT	0	Nominal motor speed [rpm]

Structure description

Structure ACS_DRIVE_CONFIG_TYPE is used in the ACS_DRIVE_DATA_TYPE as subelement and contains configuration parameters of the drive.

The ACS3XX_DRIVES_CTRL_BASIC function block reads scaling and other configuration parameters from the drive. These values are written to subelement CONFIG of the DRIVE_DATA variable.

These values are only valid if output SCALING_DONE at the ACS3XX_DRIVES_CTRL_BASIC is set to TRUE.

The user can read these values via the DRIVE_DATA variable but he must not change these values.

ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive

Structure ACS_DRIVE_DATA_TYPE is used for the DRIVE_DATA variable to exchange the data for one drive.

Structure information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Visible Variable	Type	Default value	User Access *)	Description
sw	WORD	0	R	Actual Status Word from drive
actValue1	INT	0	R	Actual value 1 from drive - mostly actual speed
actValue2	INT	0	R	Actual value 2 from drive - mapping is made in drive configuration
cw	WORD	0	R	Control Word to drive
refValue1	INT	0	R	Reference Value 1 to drive - mostly speed reference
refValue2	INT	0	R	Reference Value 2 to drive - mapping is made in drive configuration
driveType	INT	0	R	Selected drive type – input of communication function block
adapterType	INT	0	R	Seleceted fieldbus adapter type – built in communication function block according to chosen drive type and communication block for specific fieldbus type
name	STRING[20]	"Default Drive Name"	R/W	Name for drive which can be set by user directly to DRIVE_DATA variable
online	BOOL		R	Connection established – set in Modbus communication function block after successful reading and writing one Modbus job
config	ACS_DRIVE_CONFIG_TYPE		R	Some configuration and scaling parameters of the drive – read by the ACS3XX_DRIVES_CTRL_BASIC function block

*) R = read only, R/W = read and write access

Structure description

Structure ACS_DRIVE_DATA_TYPE is used for the DRIVE_DATA variable which must be connected to all function blocks related to the same drive.

Besides the variable “name” all variables should not be written by the user directly. They are read and written within the function blocks.

The ACS_DRIVE_DATA_TYPE contains some more internal, invisible variables which are used for interlocks and data transfer and not meant for user access.

ACS_MOD_TOKEN_TYPE structure to exchange the internal Modbus token for Modbus RTU communication with more than 1 Drive

Structure ACS_MOD_TOKEN_TYPE is used to exchange the internal Modbus token between the ACS3XX_COM_MOD_RTU or ACS_COM_MOD_RTU function blocks.

Structure information

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Structure description

Structure ACS_MOD_TOKEN_TYPE is used for the LINE_TOKEN variable which must be connected to all Modbus communication function blocks of the same physical Modbus line, e.g. COM1. As on the same Modbus line, only one job at a time can be processed, this variable is used to transfer the internal Modbus token to handle this.

ACS_PB_PN_PRM_DPV1_DATA_TYPE

Structure information

Available as of runtime system:	V2.4	Remark:
Included in library:	ACSDrives-Base_AC500_V20.lib	

Visible Variable	Type	Default Value	Description
abyPrmGroup	ARRAY[1..37] OF BYTE	0	Drive parameter Group Number. For example: Parameter 20.12 , Group number is 20.
abyPrmIndex	ARRAY[1..37] OF BYTE	0	Drive parameter Index Number. For example: Parameter 20.12 , Index number is 12.
abyPrmType	ARRAY[1..37] OF BYTE	0	While using READ block, it will act as an output and for WRITE block, it will act as an input. For writing, user need to speficy the Drive Parameter Type here. Refer ACS_PB_PN_PRM_TYPE_ENUM for ENUM for each data type ↬ *Chapter 1.5.6.2.5.2 “ACS_PB_PN_PRM_TYPE_ENUM” on page 2251*.
adwPrmValue	ARRAY[1..37] OF DWORD	0	While using READ block, it will act as an output and for WRITE block, it will act as an input. For writing the value to drive, user need to specify the value here. While reading, user will receive the value here.

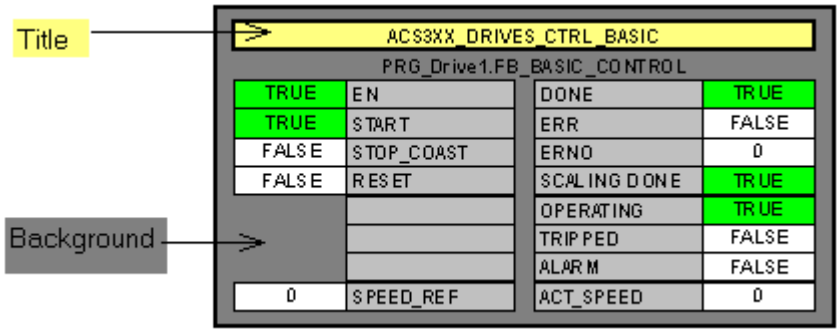
Description

Structure ACS_PB_PN_PRM_DPV1_DATA_TYPE is used to assign assigning Group, Index, Type and Value in input DATA for ACS_PB_WRITE_N_PRM_DPV1 and ACS_PN_WRITE_N_PRM_DPV1.

Same structure is used in ACS_PB_READ_N_PRM_DPV1 and ACS_PN_READ_N_PRM_DPV1 for assigning Group and Index in Input DATA . READ function block reads the values and stores it in type and Value array available in the same structure.

1.5.6.2.7 Global variables

dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements



The background color and the color of the title in the visualization elements of the ACS Drives Libraries can be changed with the two global variables dwAcsVisuBackgroundColor and dwAcs-VisuTitleColor.

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

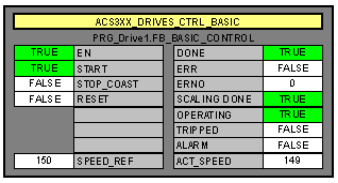
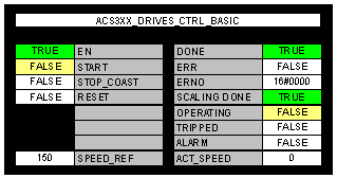
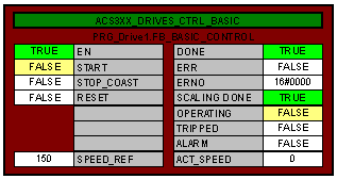
Parameters

The color variables are given in hex format and represent the following colors:

```
variable := 16#00BBGGRR          (where RR = red, GG = green, BB = blue)
```

The color variables can be changed to another value in any part of the users program.

Examples

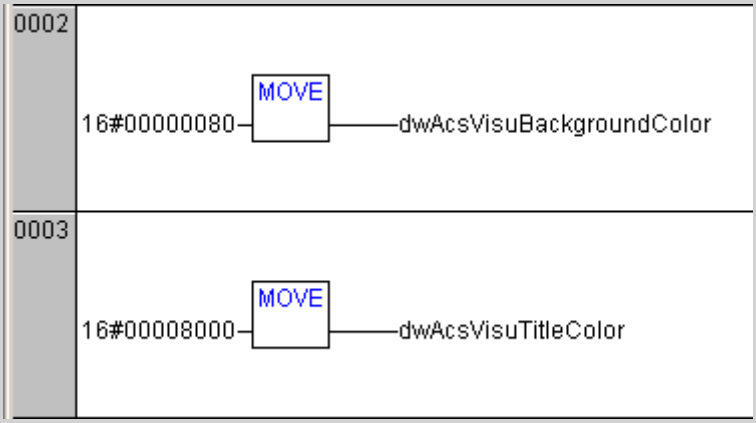
	dwAcsVisuBack-groundColor	dwAcsVisuTitleColor	Example
Example 1	16#00808080	16#0080FFFF	
Example 2	16#00000000 (black)	16#00FFFFFF (white)	
Example 3	16#00000080	16#00008000	

Change Color Variables in ST

Example 3:

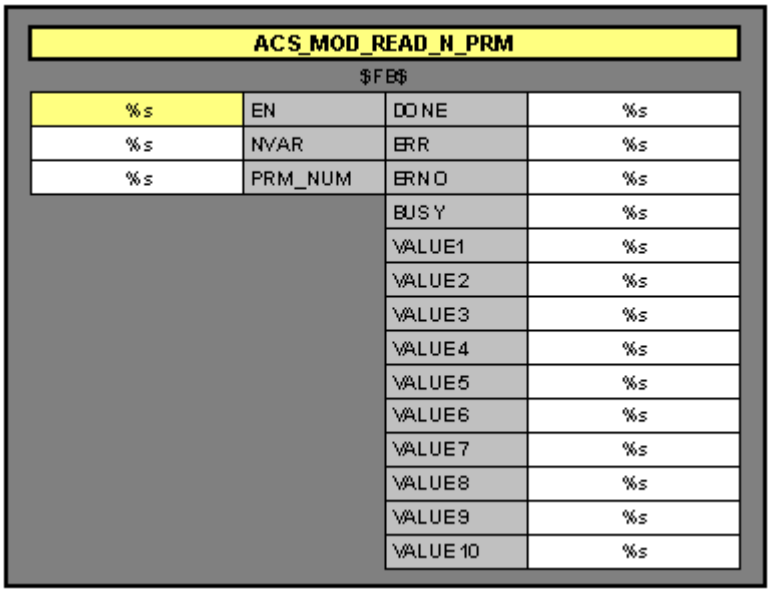
```
dwAcsVisuBackgroundColo := 16#00000080;
dwAcsVisuTitleColor := 16#00008000;
```

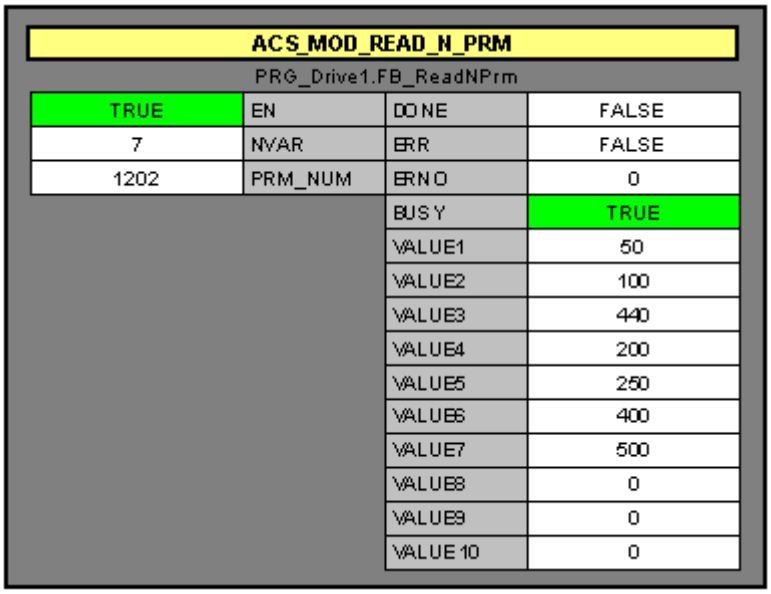
Change Color Variables in FUP



1.5.6.2.8 Visualizations

ACS_MOD_READ_N_PRM_VISU_PH faceplate for the function block ACS_MOD_READ_N_PRM





Visualization element ACS_MOD_READ_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_MOD_READ_N_PRM function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Additionally the first 10 values of the read block are shown.

Visualization information

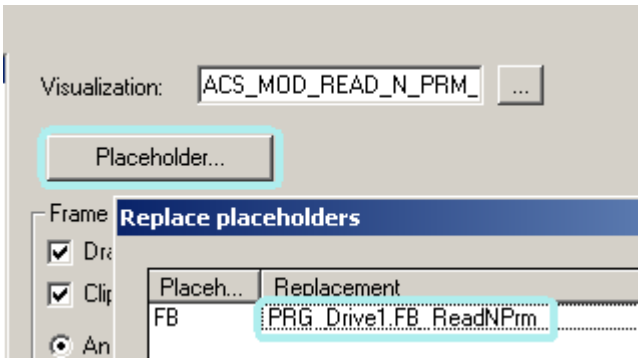
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

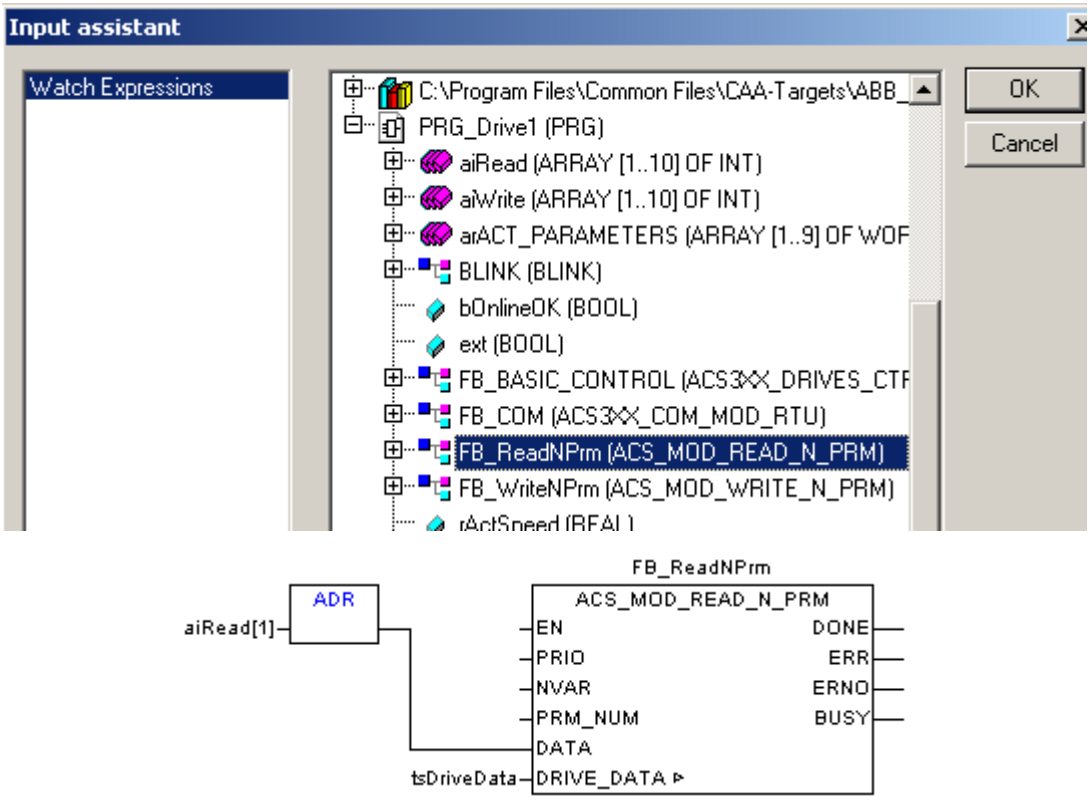
Visualization description

Visualization element ACS_MOD_READ_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_MOD_READ_N_PRM function block which instance was used to replace the placeholder FB.

All inputs of that ACS_MOD_READ_N_PRM ↳ *Chapter 1.5.6.2.4.1 "ACS_MOD_READ_N_PRM" on page 2212* function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

The values to be written can not be displayed with this faceplate, because they have to be accessed by the ADR operator outside the function block ACS_MOD_READ_N_PRM.





Parameters

Access R/W

EN Access via: Toggle

Description: EN input

NVAR Access via: Numpad 1...125

Description: NVAR input

PRM_NUM Access via:

Description: PRM_NUM input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_ReadNPrm

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

VALUE1 Description: Value of 1st read parameter, written if read job was successfully.

VALUE2 Description: Value of 2nd read parameter, written if read job was successfully.

VALUE3 Description: Value of 3rd read parameter, written if read job was successfully.

VALUE4 Description: Value of 4th read parameter, written if read job was successfully.

VALUE5 Description: Value of 5th read parameter, written if read job was successfully.

VALUE6 Description: Value of 6th read parameter, written if read job was successfully.

VALUE7 Description: Value of 7th read parameter, written if read job was successfully.

VALUE8 Description: Value of 8th read parameter, written if read job was successfully

VALUE9 Description: Value of 9th read parameter, written if read job was successfully.

VALUE10 Description: Value of 10th read parameter, written if read job was successfully.

Colors

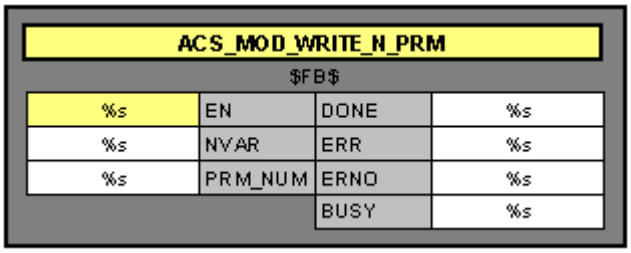
The color of the variables have the following meaning:

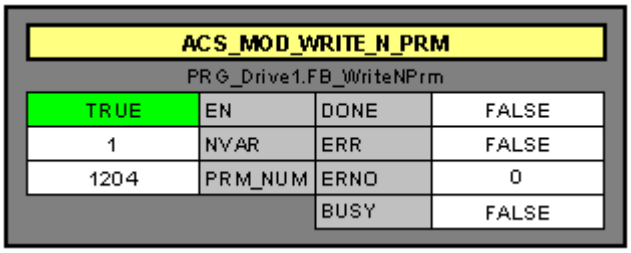
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_MOD_WRITE_N_PRM_VISU_PH faceplate for the function block ACS_MOD_WRITE_N_PRM





Visualization element ACS_MOD_WRITE_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_MOD_WRITE_N_PRM function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Visualization information

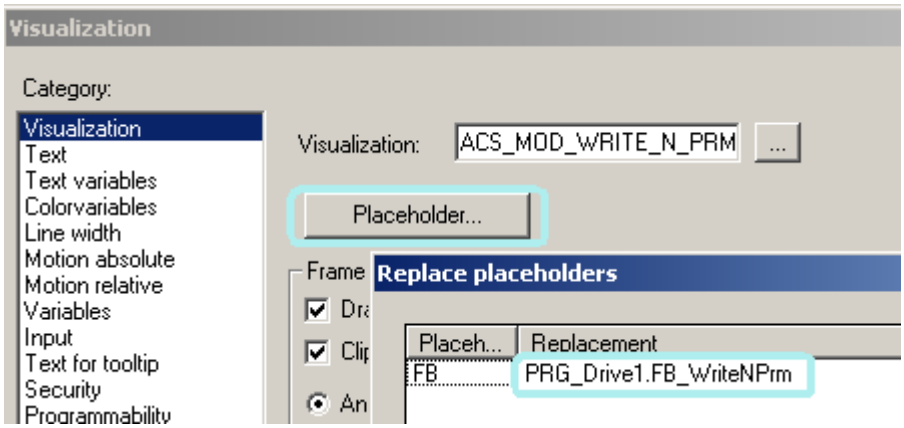
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

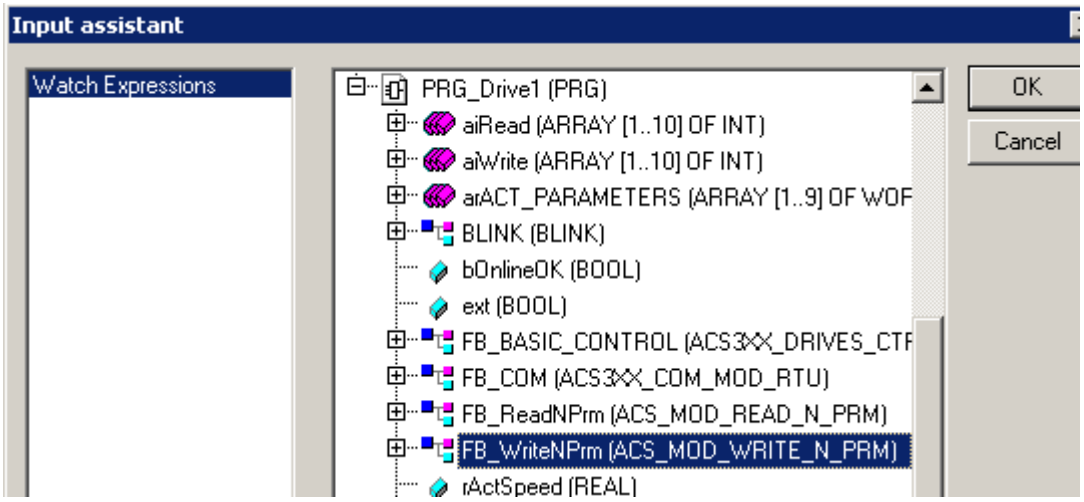
Visualization description

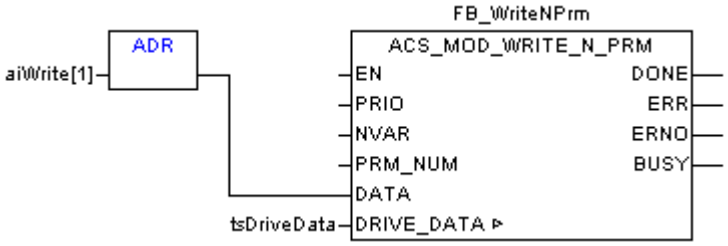
Visualization element ACS_MOD_WRITE_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_MOD_WRITE_N_PRM function block which instance was used to replace the placeholder FB.

All inputs of that ACS_MOD_WRITE_N_PRM_MOD ↬ *Chapter 1.5.6.2.4.2 "ACS_MOD_WRITE_N_PRM" on page 2215* function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

The values to be written can not be displayed with this faceplate, because they have to be accessed by the ADR operator outside the function block ACS_MOD_WRITE_N_PRM.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

NVAR Access via: Numpad 1...125
Description: NVAR input

PRM_NUM Access via:
Description: PRM_NUM input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_WriteNPrm

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

Colors

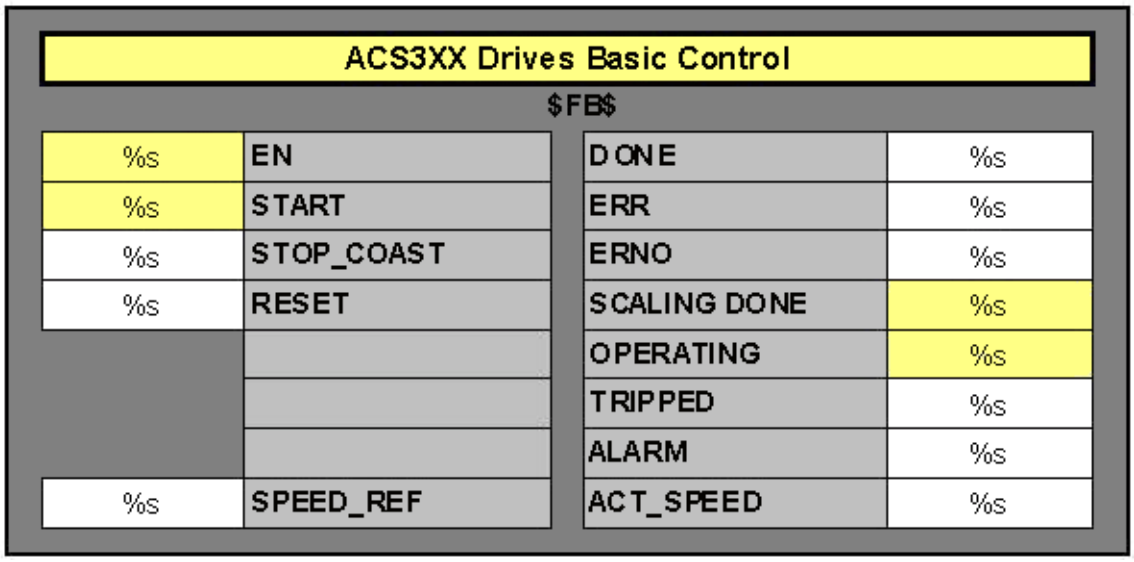
The color of the variables have the following meaning:

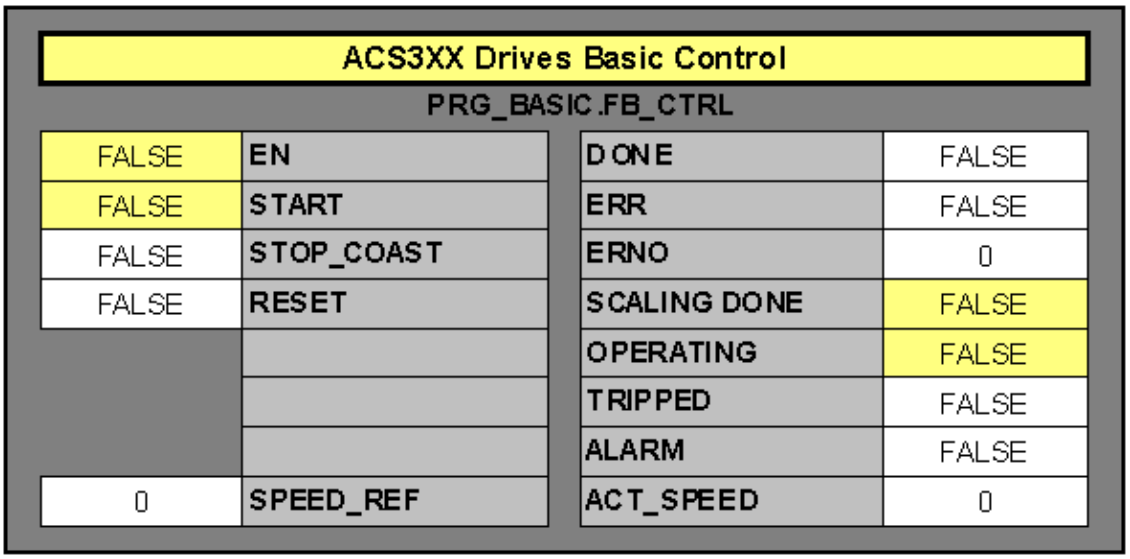
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS3XX_DRIVES_CTRL_BASIC_VISU_PH faceplate of function block ACS3XX_DRIVES_CTRL_BASIC





Visualization element ACS3XX_DRIVES_CTRL_BASIC_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS3XX_DRIVES_CTRL_BASIC function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

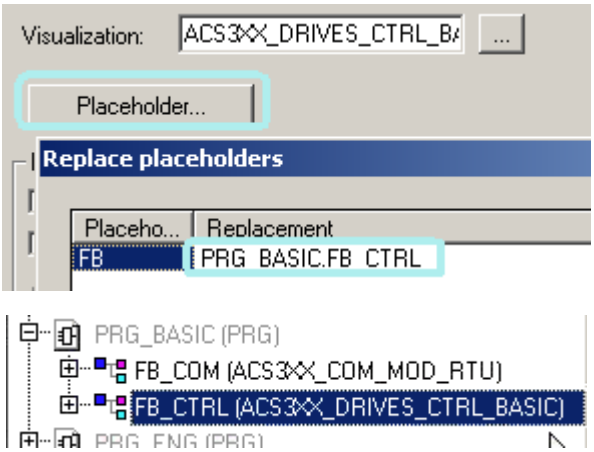
Visualization information

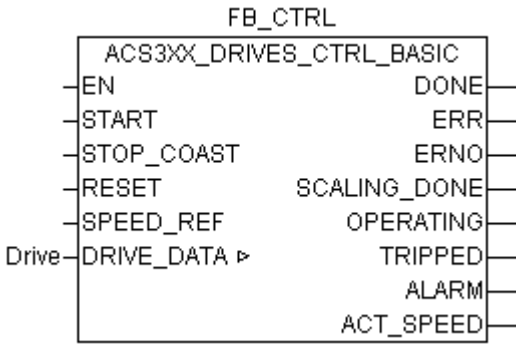
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS3XX_DRIVES_CTRL_BASIC_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS3XX_DRIVES_CTRL_BASIC *Chapter 1.5.6.2.4.4 "ACS3XX_DRIVES_CTRL_BASIC" on page 2220* function block which instance was used to replace the placeholder FB.

All inputs of that ACS3XX_DRIVES_CTRL_BASIC function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.





Parameters

Access R/W

EN Access via: Toggle
Description: EN input

START Access via: Toggle
Description: START input

STOP_COAST Access via: Toggle
Description: STOP_COAST input

RESET Access via: Toggle
Description: RESET input

SPEED_REF Access via: Numpad, no limits
Description: SPEED_REF input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_BASIC_CTRL

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

SCALING_DONE Description: SCALING_DONE output

OPERATING Description: OPERATING output

TRIPPED Description: TRIPPED output

ALARM Description: ALARM output

ACT_SPEED Description: ACT_SPEED output

Colors

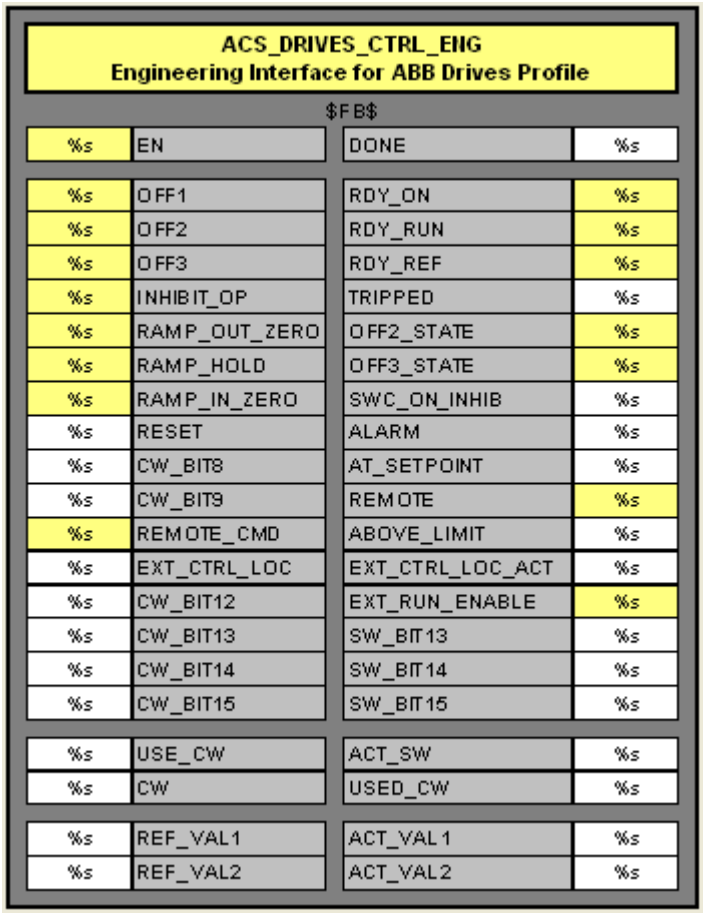
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

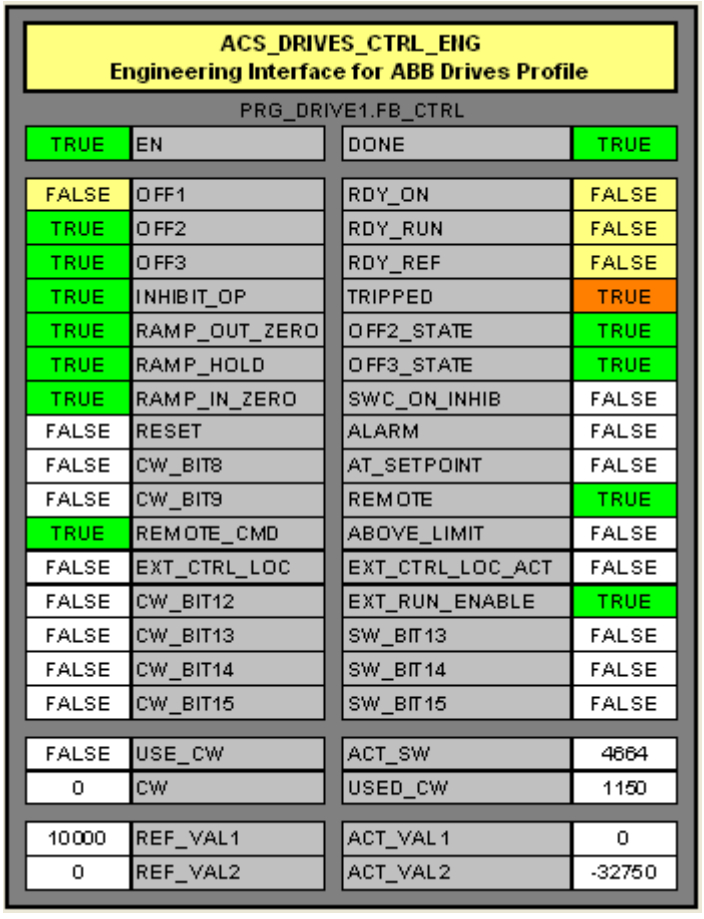
The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_DRIVES_CTRL_ENG_VISU_PH faceplate of function block ACS_DRIVES_CTRL_ENG



ACS_DRIVES_CTRL_ENG Engineering Interface for ABB Drives Profile			
FB			
%s	EN	DONE	%s
%s	OFF1	RDY_ON	%s
%s	OFF2	RDY_RUN	%s
%s	OFF3	RDY_REF	%s
%s	INHIBIT_OP	TRIPPED	%s
%s	RAMP_OUT_ZERO	OFF2_STATE	%s
%s	RAMP_HOLD	OFF3_STATE	%s
%s	RAMP_IN_ZERO	SWC_ON_INHIB	%s
%s	RESET	ALARM	%s
%s	CW_BIT8	AT_SETPOINT	%s
%s	CW_BIT9	REMOTE	%s
%s	REMOTE_CMD	ABOVE_LIMIT	%s
%s	EXT_CTRL_LOC	EXT_CTRL_LOC_ACT	%s
%s	CW_BIT12	EXT_RUN_ENABLE	%s
%s	CW_BIT13	SW_BIT13	%s
%s	CW_BIT14	SW_BIT14	%s
%s	CW_BIT15	SW_BIT15	%s
%s	USE_CW	ACT_SW	%s
%s	CW	USED_CW	%s
%s	REF_VAL1	ACT_VAL1	%s
%s	REF_VAL2	ACT_VAL2	%s



ACS_DRIVES_CTRL_ENG Engineering Interface for ABB Drives Profile			
PRG_DRIVE1.FB_CTRL			
TRUE	EN	DONE	TRUE
FALSE	OFF1	RDY_ON	FALSE
TRUE	OFF2	RDY_RUN	FALSE
TRUE	OFF3	RDY_REF	FALSE
TRUE	INHIBIT_OP	TRIPPED	TRUE
TRUE	RAMP_OUT_ZERO	OFF2_STATE	TRUE
TRUE	RAMP_HOLD	OFF3_STATE	TRUE
TRUE	RAMP_IN_ZERO	SWC_ON_INHIB	FALSE
FALSE	RESET	ALARM	FALSE
FALSE	CW_BIT8	AT_SETPOINT	FALSE
FALSE	CW_BIT9	REMOTE	TRUE
TRUE	REMOTE_CMD	ABOVE_LIMIT	FALSE
FALSE	EXT_CTRL_LOC	EXT_CTRL_LOC_ACT	FALSE
FALSE	CW_BIT12	EXT_RUN_ENABLE	TRUE
FALSE	CW_BIT13	SW_BIT13	FALSE
FALSE	CW_BIT14	SW_BIT14	FALSE
FALSE	CW_BIT15	SW_BIT15	FALSE
FALSE	USE_CW	ACT_SW	4664
0	CW	USED_CW	1150
10000	REF_VAL1	ACT_VAL1	0
0	REF_VAL2	ACT_VAL2	-32750

Visualization element ACS_DRIVES_CTRL_ENG_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_CTRL_ENG ↳ *Chapter 1.5.6.2.4.5 “ACS_DRIVES_CTRL_ENG ” on page 2226* function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

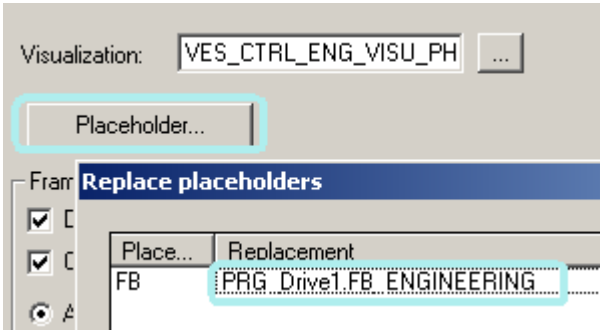
Visualization information

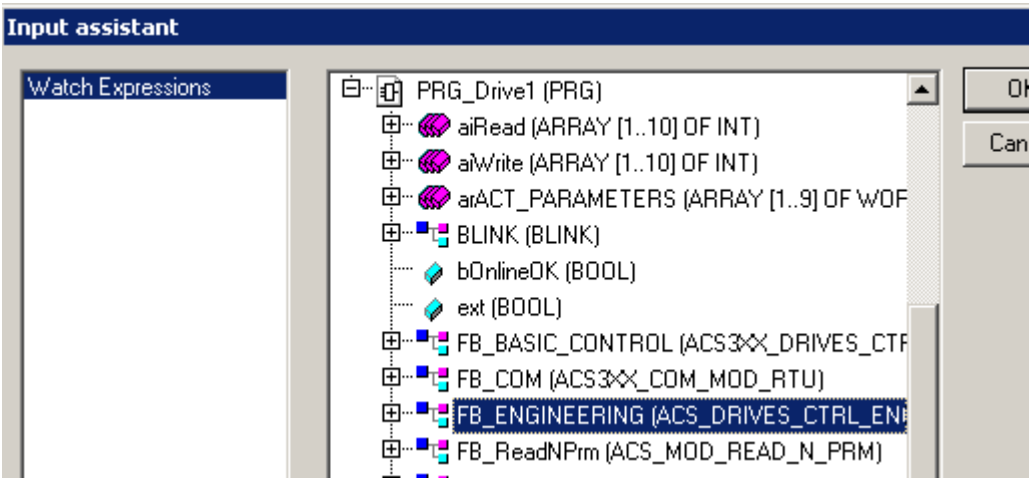
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

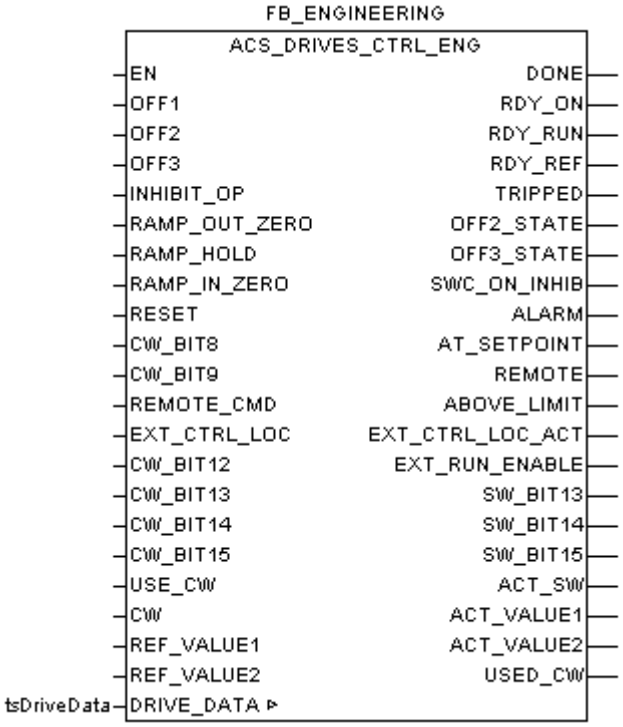
Visualization description

Visualization element ACS_DRIVES_CTRL_ENG_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_ENG_STANDARD function block which instance was used to replace the placeholder FB.

All inputs of that ACS_DRIVES_CTRL_ENG function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

OFF1 Access via: Toggle
Description: OFF1 input

OFF2 Access via: Toggle
Description: OFF2 input

OFF3 Access via: Toggle
Description: OFF3 input

INHIBIT_OP Access via: Toggle
Description: INHIBIT_OP input

RAMP_OUT_ZERO Access via: Toggle
Description: RAMP_OUT_ZERO input

RAMP_HOLD Access via: Toggle
Description: RAMP_HOLD input

RAMP_IN_ZERO Access via: Toggle
Description: RAMP_IN_ZERO input

RESET Access via: Toggle
Description: RESET input

CW_BIT8 Access via: Toggle
Description: CW_BIT8 input

CW_BIT9 Way of Access: Toggle
Description: CW_BIT9 input

REMOTE_CMD Access via: Toggle
Description: CW_BIT9 input

EXT_CTRL_LOC Access via: Toggle
Description: EXT_CTRL_LOC input

CW_BIT12 Access via: Toggle
Description: CW_BIT12 input

CW_BIT13 Access via: Toggle
Description: CW_BIT13 input

CW_BIT14 Access via: Toggle
Description: CW_BIT14 input

CW_BIT15 Access via: Toggle
Description: CW_BIT15 input

USE_CW Access via: Toggle
Description: USE_CW input (switch to CW input instead of single bits above)

CW Access via: Text input -Word: 0...65535
Description: CW input (used if USE_CW = TRUE)

REF_VALUE1 Access via: Text input -32768...+32768
Description: REF_VALUE1 input

REF_VALUE2 Access via: Text input -32768...+32768
Description: REF_VALUE2 input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_ENGINEERING

Access R

DONE Description: DONE output.

RDY_ON Description: RDY_RUN output

RDY_RUN Description: RDY_RUN output

RDY_REF Description: RDY_REF output

TRIPPED Description: TRIPPED output

OFF2_STATE Description: OFF2_STATE output

OFF3_STATE Description: OFF3_STATE output

SWITCH_ON_INHIB Description: SWITCH_ON_INHIB output

ALARM Description: ALARM output

AT_SETPOINT Description: AT_SETPOINT output

REMOTE Description: REMOTE output

ABOVE_LIMIT Description: ABOVE_LIMIT output

EXT_CTRL_LOC Description: EXT_CTRL_LOC output

EXT_RUN_EN Description: EXT_RUN_EN output

SW_BIT13 Description: SW_BIT13 output

SW_BIT14 Description: SW_BIT14 output

SW_BIT15 Description: SW_BIT15 output

ACT_SW Description: ACT_SW output

USED_CW Description: USED_CW output

ACT_VALUE1 Description: ACT_VALUE1 output

ACT_VALUE2 Description: ACT_VALUE2 output

Colors

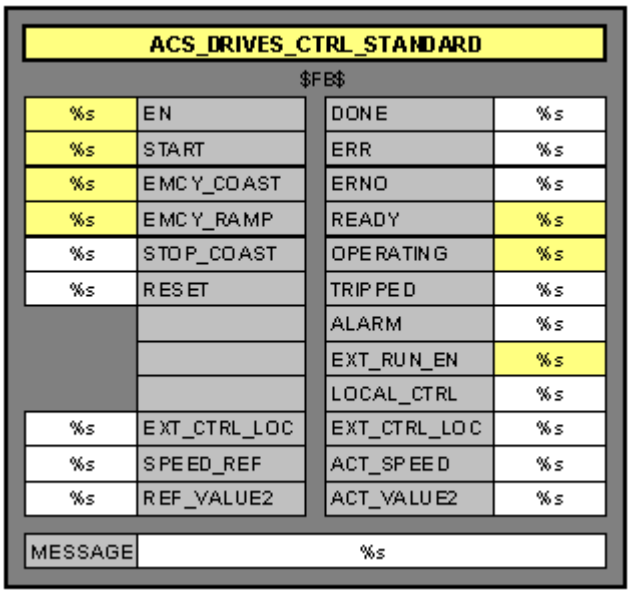
The color of the variables have the following meaning:

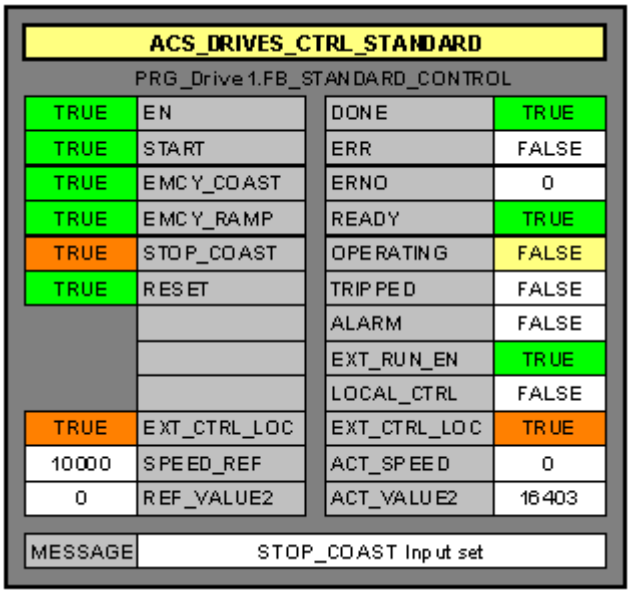
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_DRIVES_CTRL_STANDARD_VISU_PH faceplate of function block ACS_DRIVES_CTRL_STANDARD





Visualization element ACS_DRIVES_CTRL_STANDARD_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_CTRL_STANDARD function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

Visualization information

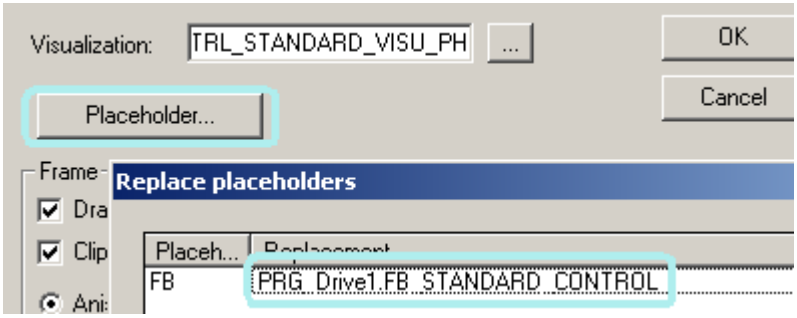
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

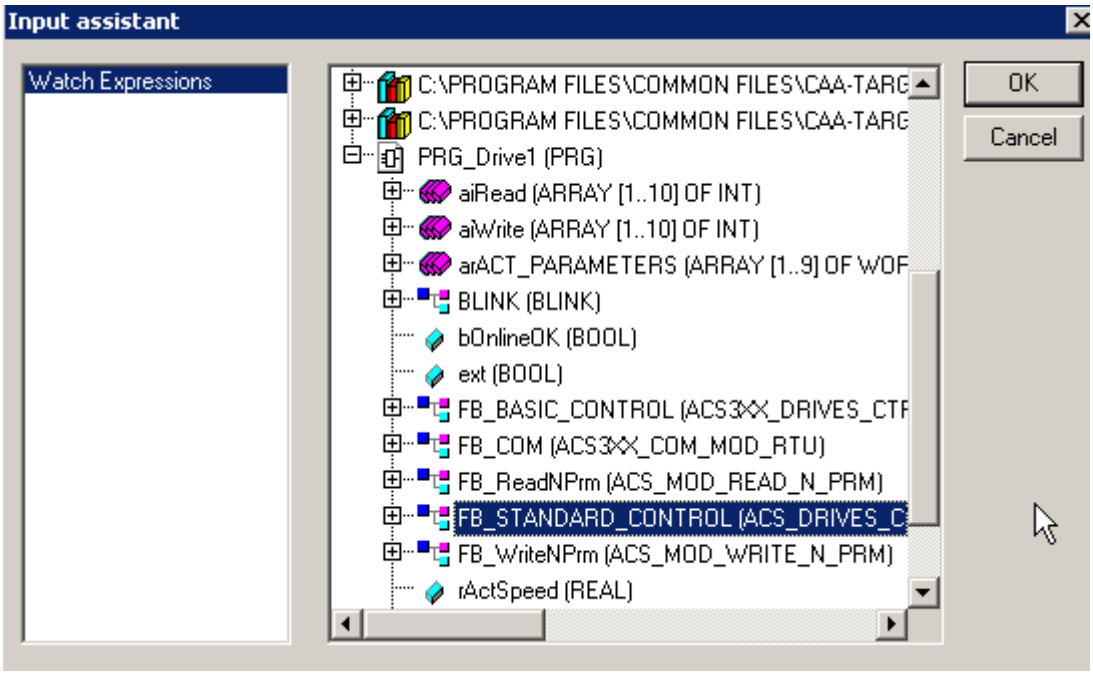
Visualization description

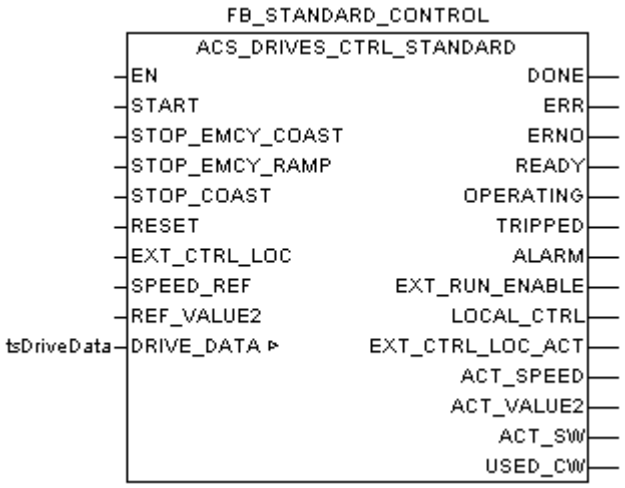
Visualization element ACS_DRIVES_CTRL_STANDARD_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_CTRL_STANDARD ↳ *Chapter 1.5.6.2.4.6 “ACS_DRIVES_CTRL_STANDARD ” on page 2234* function block which instance was used to replace the placeholder FB.

All inputs of that ACS_DRIVES_CTRL_STANDARD function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

Additionally, a text message of the actual state, e.g. missing input or missing feedback from drive, is given by the variable MESSAGE, shown next to label MESSAGE.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

START Access via: Toggle
Description: START input

EMCY_COAST Access via: Toggle
Description: EMCY_COAST input

EMCY_RAMP Access via: Toggle
Description: EMCY_RAMP input

STOP_COAST Access via: Toggle
Description: STOP_COAST input

RESET Access via: Toggle
Description: RESET input

EXT_CTRL_LOC Access via: Toggle
Description: EXT_CTRL_LOC input

SPEED_REF Access via: Numpad, no limits
Description: SPEED_REF input

REF_VALUE2 Access via: Text input -32768...+32768
Description: REF_VALUE2 input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_BASIC_CTRL

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

READY Description: READY output

OPERATING Description: OPERATING output

TRIPPED Description: TRIPPED output

ALARM Description: ALARM output

EXT_RUN_EN Description: EXT_RUN_EN output

LOC_CTRL Description: LOC_CTRL output

EXT_CTRL_LOC Description: EXT_CTRL_LOC output

ACT_SPEED Description: ACT_SPEED output

ACT_VALUE2 Description: ACT_VALUE2 output

MESSAGE Description: MESSAGE output

Colors

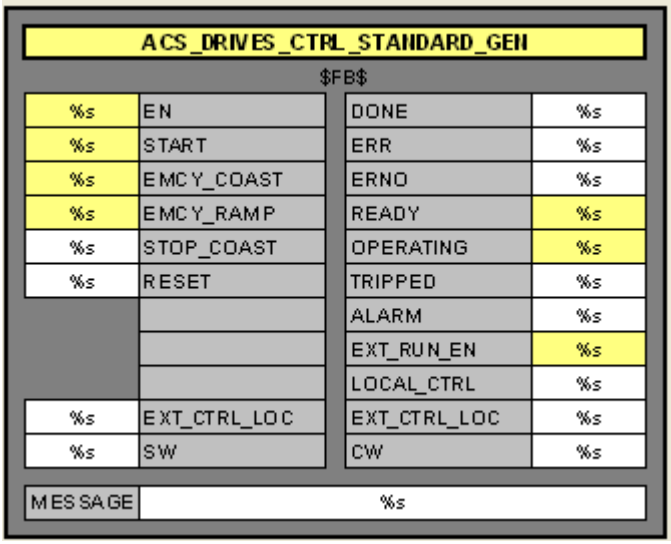
The color of the variables have the following meaning:

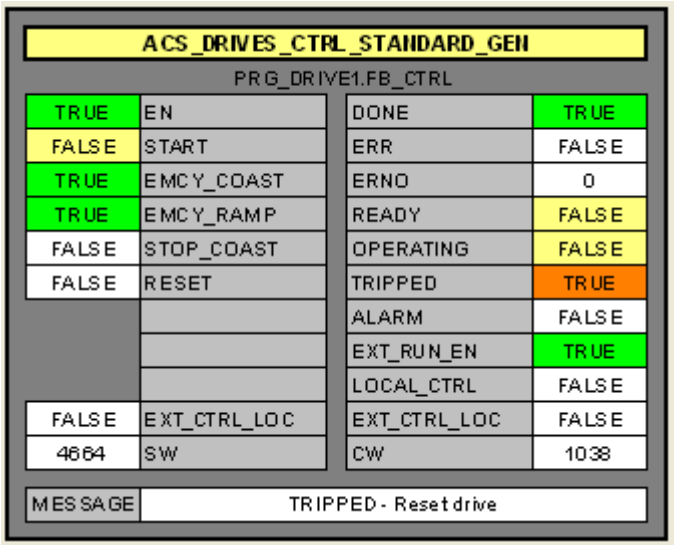
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_DRIVES_CTRL_STANDARD_GEN_VISU_PH faceplate for the function block





Visualization element ACS_DRIVES_CTRL_STANDARD_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_CTRL_STANDARD_GEN ↳ *Chapter 1.5.6.2.4.7 "ACS_DRIVES_CTRL_STANDARD_GEN" on page 2241* function block which instance is used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

Visualization information

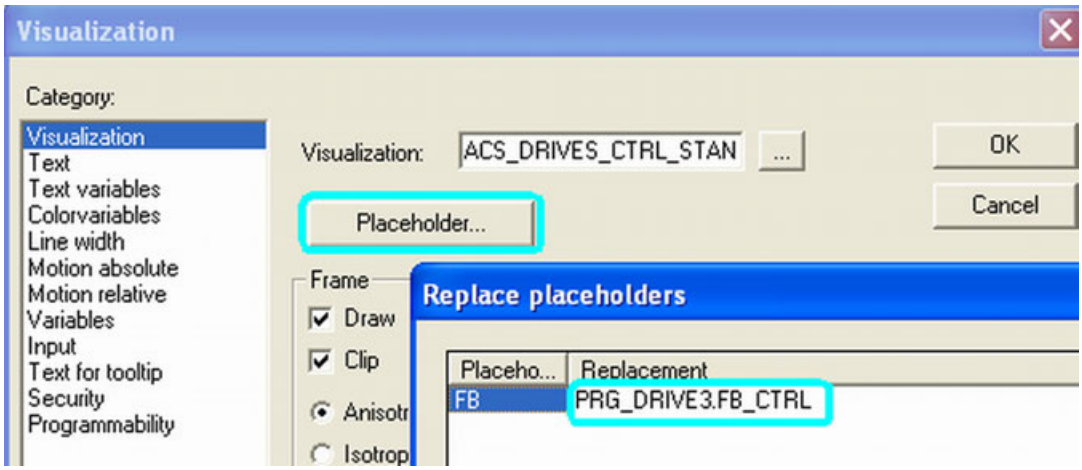
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

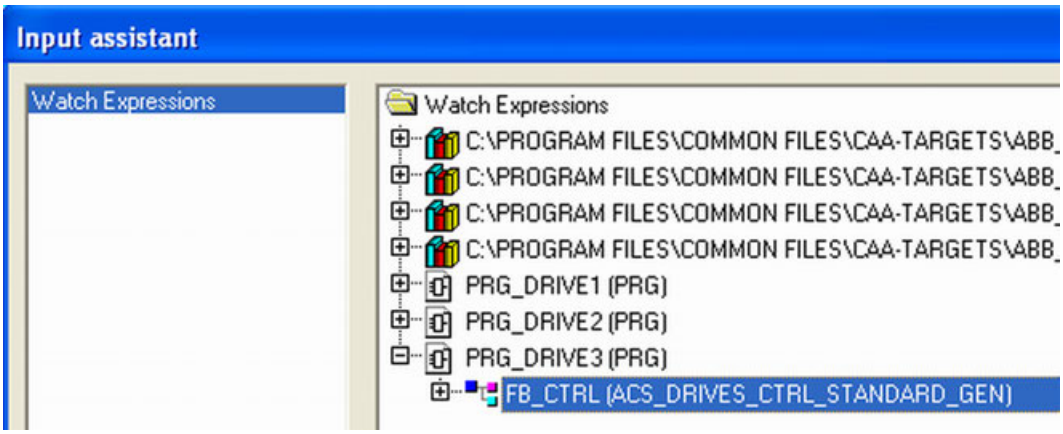
Visualization description

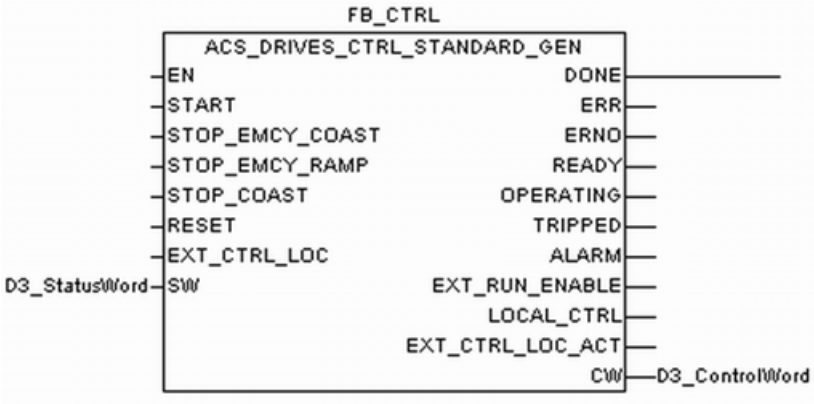
Visualization element ACS_DRIVES_CTRL_STANDARD_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_DRIVES_CTRL_STANDARD_GEN ↳ *Chapter 1.5.6.2.4.7 "ACS_DRIVES_CTRL_STANDARD_GEN" on page 2241*function block which instance was used to replace the placeholder FB.

All inputs of that ACS_DRIVES_CTRL_STANDARD_GEN ↳ *Chapter 1.5.6.2.4.7 "ACS_DRIVES_CTRL_STANDARD_GEN" on page 2241*function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.

Additionally, a text message of the actual state, e.g. missing input or missing feedback from drive, is given by the variable MESSAGE, shown next to the label MESSAGE.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

START Access via: Toggle
Description: START input

EMCY_COAST Access via: Toggle
Description: EMCY_COAST input

EMCY_RAMP Access via: Toggle
Description: EMCY_RAMP input

STOP_COAST Access via: Toggle
Description: STOP_COAST input

RESET Access via: Toggle
Description: RESET input

EXT_CTRL_LOC Access via: Toggle

Description: EXT_CTRL_LOC input

SW Access via: Text

Description: SW input (Status word)

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_BASIC_CTRL

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

READY Description: READY output

OPERATING Description: OPERATING output

TRIPPED Description: TRIPPED output

ALARM Description: ALARM output

EXT_RUN_EN Description: EXT_RUN_EN output

LOC_CTRL Description: LOC_CTRL output

EXT_CTRL_LOC Description: EXT_CTRL_LOC output

CW Description: CW output (Control word)

MESSAGE Description: MESSAGE output

Colors

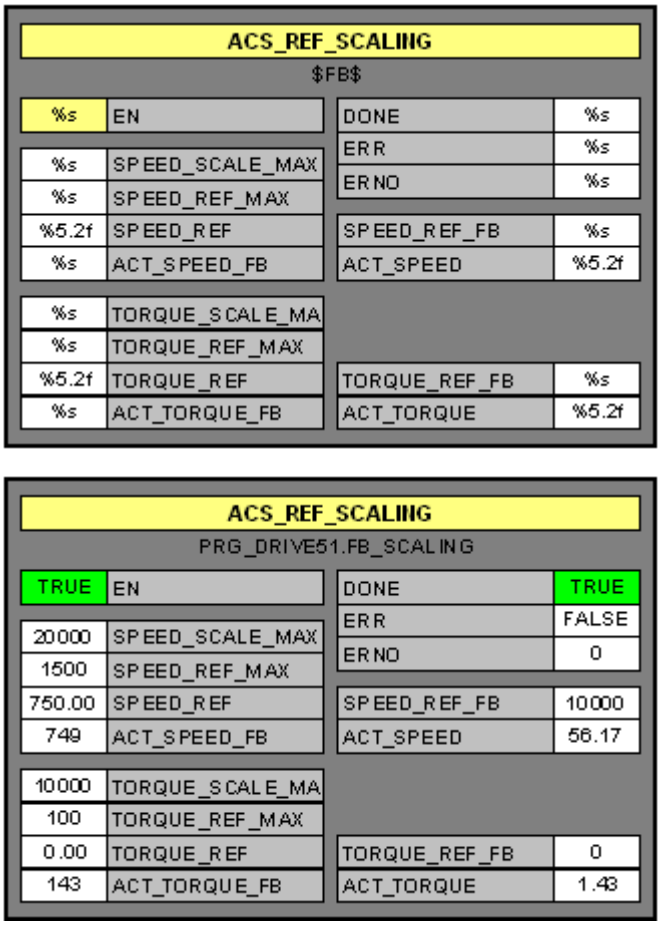
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_REF_SCALING_VISU_PH faceplate for the function block ACS_REF_SCALING



Visualization element ACS_REF_SCALING_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_REF_SCALING function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

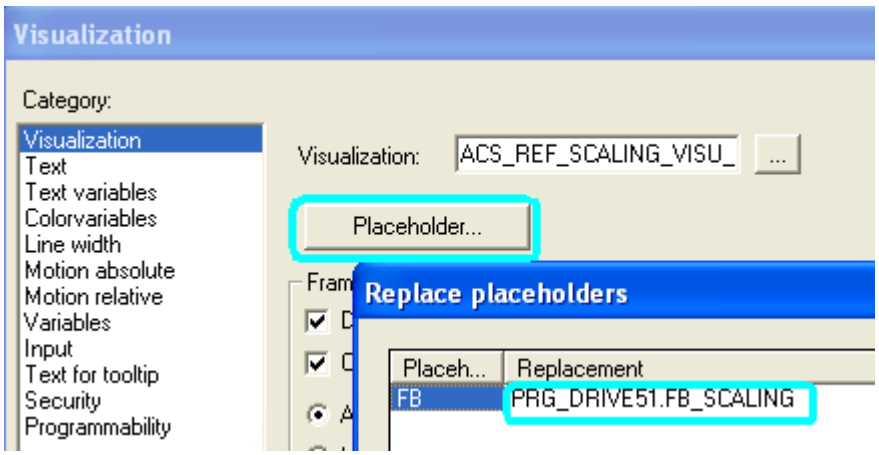
Visualization information

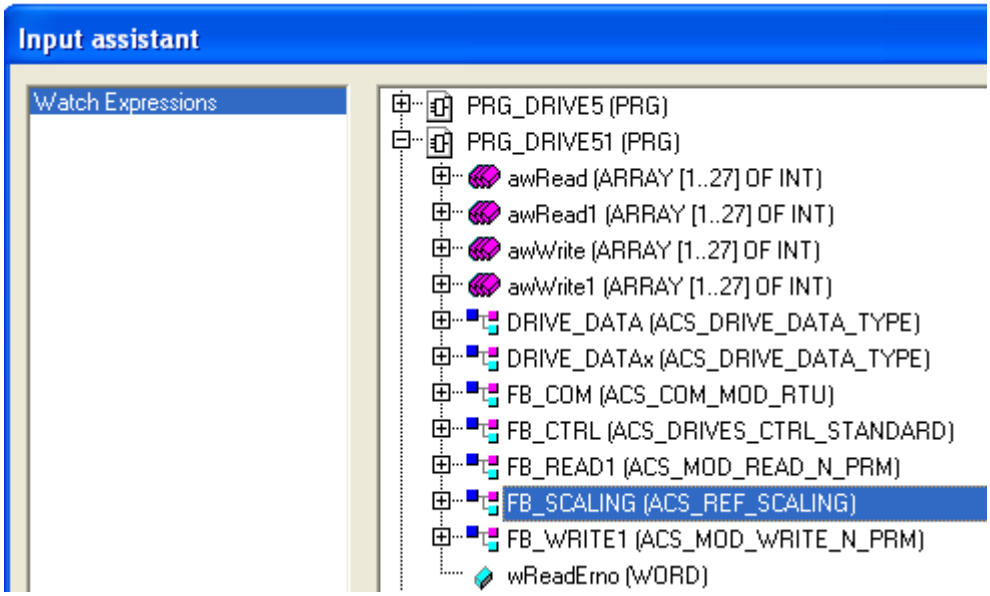
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

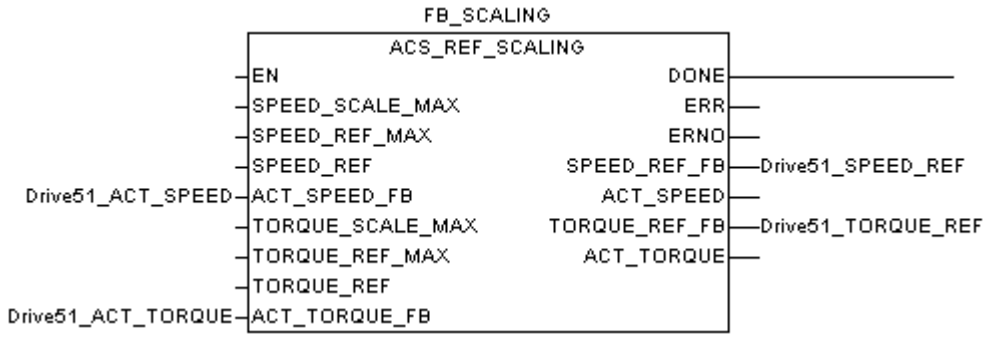
Visualization description

Visualization element ACS_REF_SCALING_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_REF_SCALING *Chapter 1.5.6.2.4.8 "ACS_REF_SCALING" on page 2246*function block which instance was used to replace the placeholder FB.

All inputs of that ACS_REF_SCALING *Chapter 1.5.6.2.4.8 "ACS_REF_SCALING" on page 2246* function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.







Parameters

Access R/W

EN
Access via: Toggle
Description: EN input

SPEED_SCALE_MAX
Access via: Numpad 1 .. 32767
Description: SPEED_SCALE_MAX input

SPEED_REF_MAX
Access via: Numpad 1 .. 32767
Description: SPEED_REF_MAX input

SPEED_REF
Access via: Numpad, no limits
Description: SPEED_REF input

ACT_SPEED_FB
Access via: Numpad -32768.0 .. -32767.0
Description: ACT_SPEED_FB input – should be connected to the fieldbus.

TORQUE_SCALE_MAX
Access via: Numpad 1...32767
Description: TORQUE_SCALE_MAX input

TORQUE_REF_MAX
Access via: Numpad 1...32767
Description: TORQUE_REF_MAX input

TORQUE_REF
Access via: Numpad -32768.0 .. -32767.0
Description: TORQUE_REF input

ACT_TORQUE_FB
Access via: Numpad -32768.0 .. -32767.0
Description: ACT_TORQUE_FB input – should be connected to the fieldbus.

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive51.FB_SCALING

Access R

DONE
Description: DONE output.

ERR
Description: ERR output.

ERNO
Description: ERNO output.

BUSY Description: BUSY output.

SPEED_REF_FB Description: SPEED_REF_FB output – should be connected to the fieldbus.

ACT_SPEED Description: ACT_SPEED output

TORQUE_REF_FB Description: TORQUE_REF_FB output – should be connected to the fieldbus.

ACT_TORQUE Description: ACT_TORQUE output

Colors

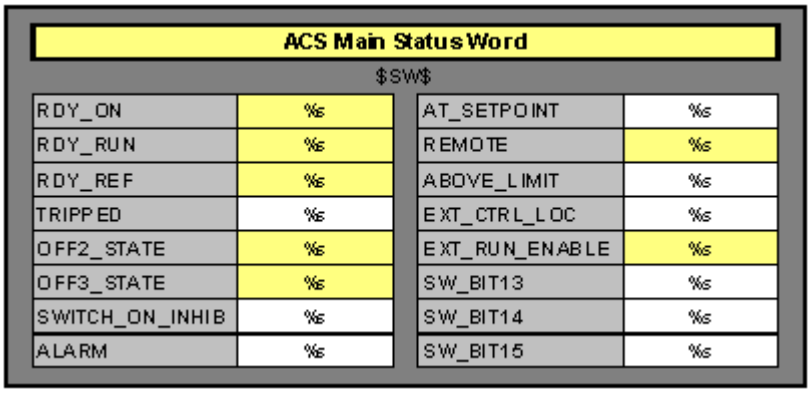
The color of the variables have the following meaning:

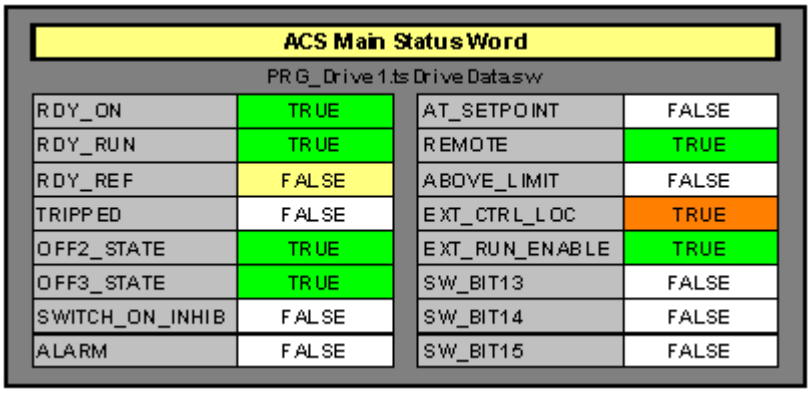
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_SW_VISU_PH visualization for the ABB drives profile status word





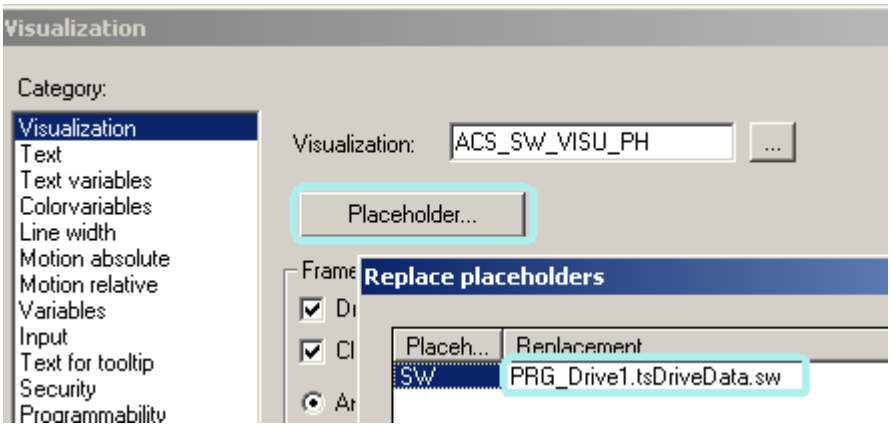
Visualization element ACS_SW_VISU_PH can be used to show the actual values of all bits of the Status Word.

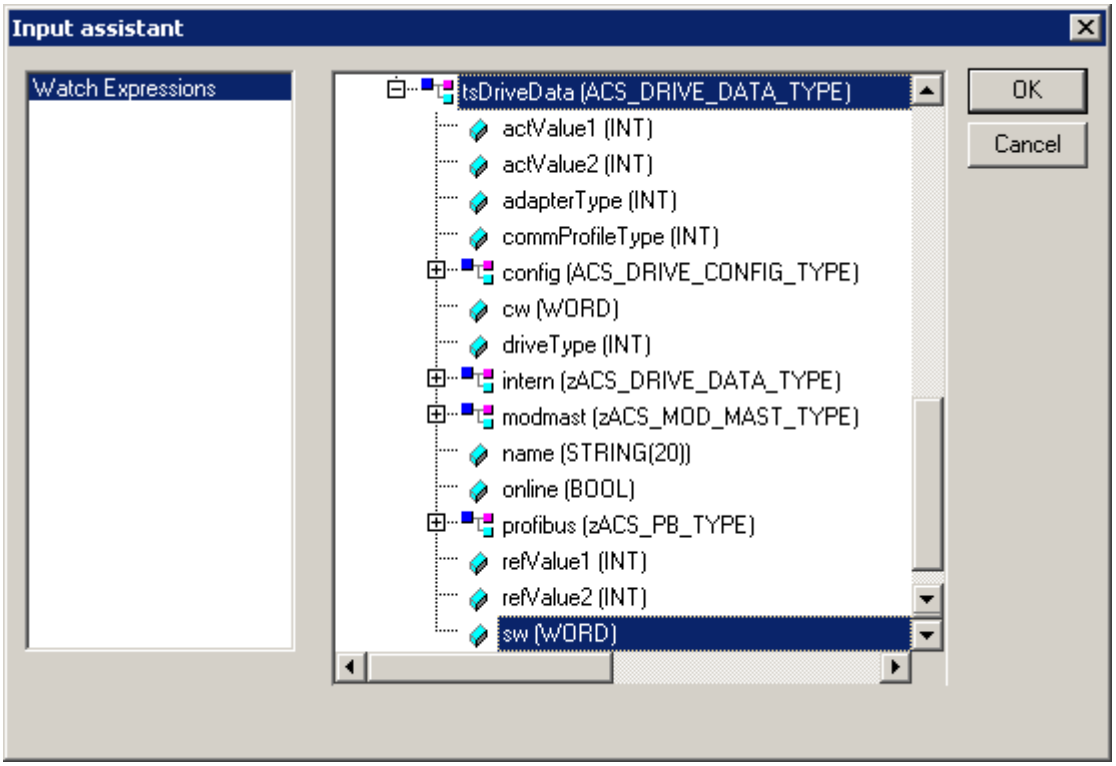
Visualization information

Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_SW_VISU_PH can be used to show the actual values of all bits of the Status Word. The placeholder SW must be replaced by the “sw” element of the DRIVE_DATA variable.






```
PROGRAM PRG_Drive1
VAR
    FB_COM:             ACS3XX_COM_MOD_RTU;
    tsDriveData:        ACS_DRIVE_DATA_TYPE;
    FB_BASIC_CONTROL:   ACS3XX_DRIVES_CTRL_BASIC;
```

Parameters

Access R

RDY_ON	Description: RDY_RUN output
RDY_RUN	Description: RDY_RUN output
RDY_REF	Description: RDY_REF output
TRIPPED	Description: TRIPPED output
OFF2_STATE	Description: OFF2_STATE output
OFF3_STATE	Description: OFF3_STATE output
SWITCH_ON_INHIB	Description: SWITCH_ON_INHIB output
ALARM	Description: ALARM output

AT_SETPOINT Description: AT_SETPOINT output

REMOTE Description: REMOTE output

ABOVE_LIMIT Description: ABOVE_LIMIT output

EXT_CTRL_LOC Description: EXT_CTRL_LOC output

EXT_RUN_EN Description: EXT_RUN_EN output

SW_BIT13 Description: SW_BIT13 output

SW_BIT14 Description: SW_BIT14 output

SW_BIT15 Description: SW_BIT15 output

Placeholder	Replacement	Example
SW	SW element of DRIVE_DATA variable	PRG_Drive1.tsDriveData.sw

Colors

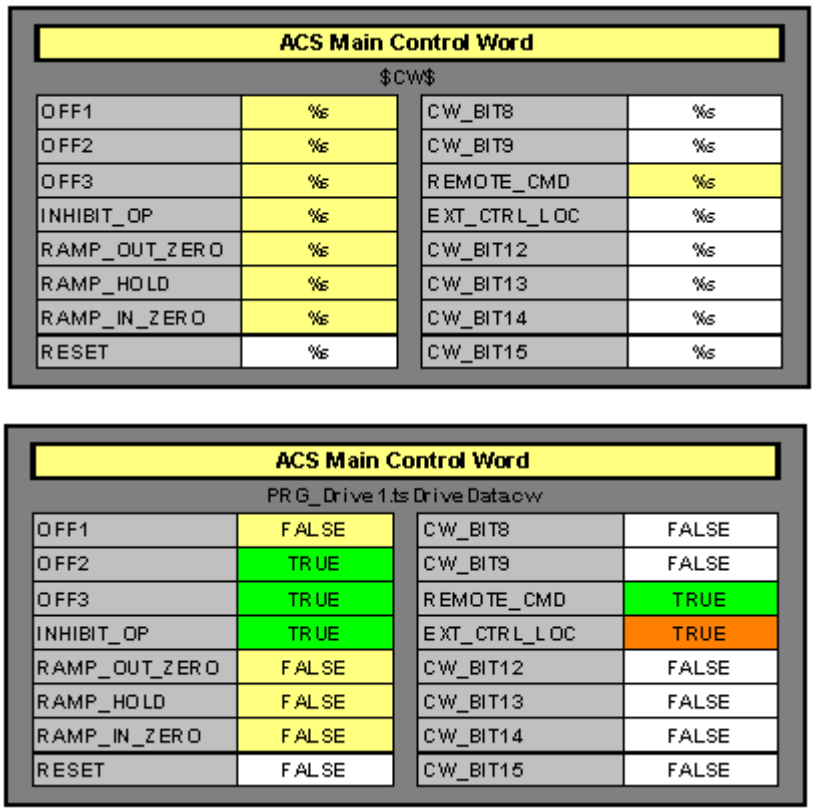
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_CW_VISU_PH visualization for the ABB drives profile control word



ACS Main Control Word			
CW			
OFF1	%s	CW_BIT8	%s
OFF2	%s	CW_BIT9	%s
OFF3	%s	REMOTE_CMD	%s
INHIBIT_OP	%s	EXT_CTRL_LOC	%s
RAMP_OUT_ZERO	%s	CW_BIT12	%s
RAMP_HOLD	%s	CW_BIT13	%s
RAMP_IN_ZERO	%s	CW_BIT14	%s
RESET	%s	CW_BIT15	%s

ACS Main Control Word			
PRG_Drive1.tsDriveData.cw			
OFF1	FALSE	CW_BIT8	FALSE
OFF2	TRUE	CW_BIT9	FALSE
OFF3	TRUE	REMOTE_CMD	TRUE
INHIBIT_OP	TRUE	EXT_CTRL_LOC	TRUE
RAMP_OUT_ZERO	FALSE	CW_BIT12	FALSE
RAMP_HOLD	FALSE	CW_BIT13	FALSE
RAMP_IN_ZERO	FALSE	CW_BIT14	FALSE
RESET	FALSE	CW_BIT15	FALSE

Visualization ACS_CW_VISU_PH can be used to show the actual value of all bits of the Control Word. The naming is made according the ABB Drives Profile.

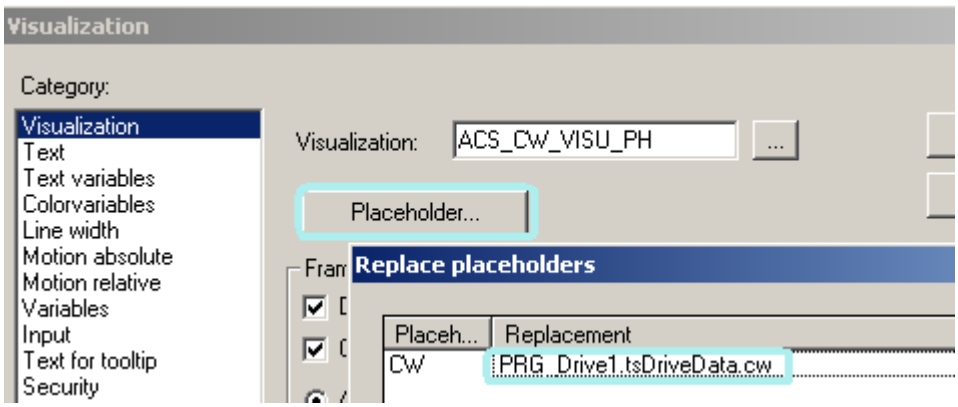
Visualization information

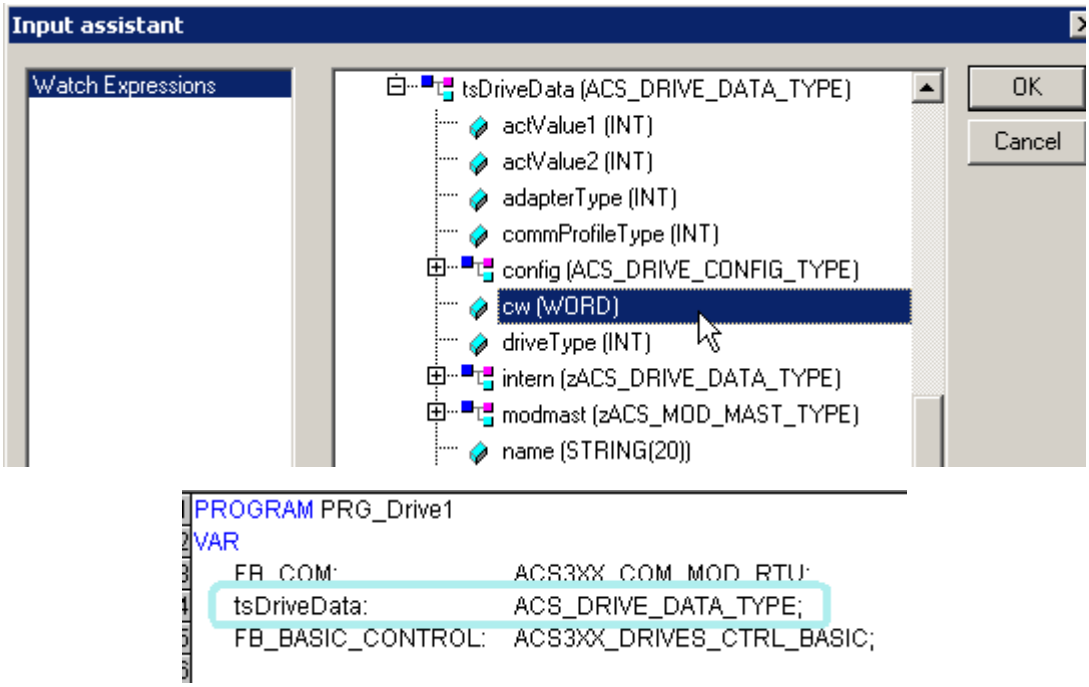
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization ACS_CW_VISU_PH can be used to show the actual value of all bits of the Control Word. The naming is made according the ABB Drives Profile.

The placeholder CW must be replaced by the “cw” element of the DRIVE_DATA variable.





Parameters

Access R

OFF1	Description: OFF1 input
OFF2	Description: OFF2 input
OFF3	Description: OFF3 input
INHIBIT_OP	Description: INHIBIT_OP input
RAMP_OUT_ZERO	Description: RAMP_OUT_ZERO input
RAMP_HOLD	Description: RAMP_HOLD input
RAMP_IN_ZERO	Description: RAMP_IN_ZERO input
RESET	Description: RESET input
CW_BIT8	Description: CW_BIT8 input
CW_BIT9	Description: CW_BIT9 input
REMOTE_CMD	Description: REMOTE_CMD input

EXT_CTRL_LOC Description: EXT_CTRL_LOC input

CW_BIT12 Description: CW_BIT12 input

CW_BIT13 Description: CW_BIT13 input

CW_BIT14 Description: CW_BIT14 input

CW_BIT15 Description: CW_BIT15 input

Placeholder	Replacement	Example
CW	CW element of DRIVE_DATA variable	PRG_Drive1.tsDriveData.cw

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.3 ACS / DCS Drives communication via Modbus RTU library

1.5.6.3.1 Preconditions for the use of the ACS / DCS drives communication via Modbus RTU library

The library is released for the following products:

CPUs:	AC500 and AC500-eCo
Fieldbus:	Modbus RTU (Serial Modbus)
Drives:	ACS800, ACSM1, ACS350, ACS355, ACS310, ACS550, ACH550, ACQ810, ACS850, ACS880, ACS800, DCS800, DCS550

Generic Devices:	No specific limit except that they can work with the Modbus function codes defined in function block *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*.
Modbus RTU configuration:	Prior to the use of the function blocks a Communication Module "COMx_Modbus" has to be configured accordingly using Automation Builder, either at "Interfaces" or at "CM574-RS" module.
ACS3XX_COM_MOD_RTU, ACS_COM_MOD_RTU, ACS_COM_MOD_RTU_ENHANCED:	Modbus communication tested for connection of 7 drives in total. Connection of more drives depends on the performance of used CPU and communication settings. The communication function blocks are designed to be used for one specific drive at run time. So it´s not recommanded to change the COM or SLAVE inputs of the blocks while the program is running.
ACS_COM_MOD_RTU_GEN:	Modbus communication for generic devices. The communication performance depends on performance of used CPU and communication settings. The communication function block is designed to be used for one specific slave device at run time. So it´s not recommanded to change the COM or SLAVE inputs of the block while the program is running.

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks can only be used in combination with the ACSDrives-Base_AC500_V20 Library.

1.5.6.3.2 Special characteristics of the ACS / DCS drives communication via Modbus RTU library

The function blocks in the ACSDrivesComModRTU_AC500_V20 Library are designed to handle the Modbus RTU communication to 1 or more drives.

The function blocks ACSxxxx_COM_MOD_RTU perform all the actions that are needed to read and write Modbus RTU jobs from / to different slaves.

The data handling is automatically done in the ACSxxxx_COM_MOD_RTU function blocks using the COM_MOD_MAST and ACS_COM_READ_N_PRM and ACS_COM_WRITE_N_PRM function blocks internally.

The user just has to connect the communication function blocks together via the common LINE_TOKEN variable of type ACS_MOD_TOKEN_TYPE.

The following block diagram shows the general connection of the function blocks for 3 drives:

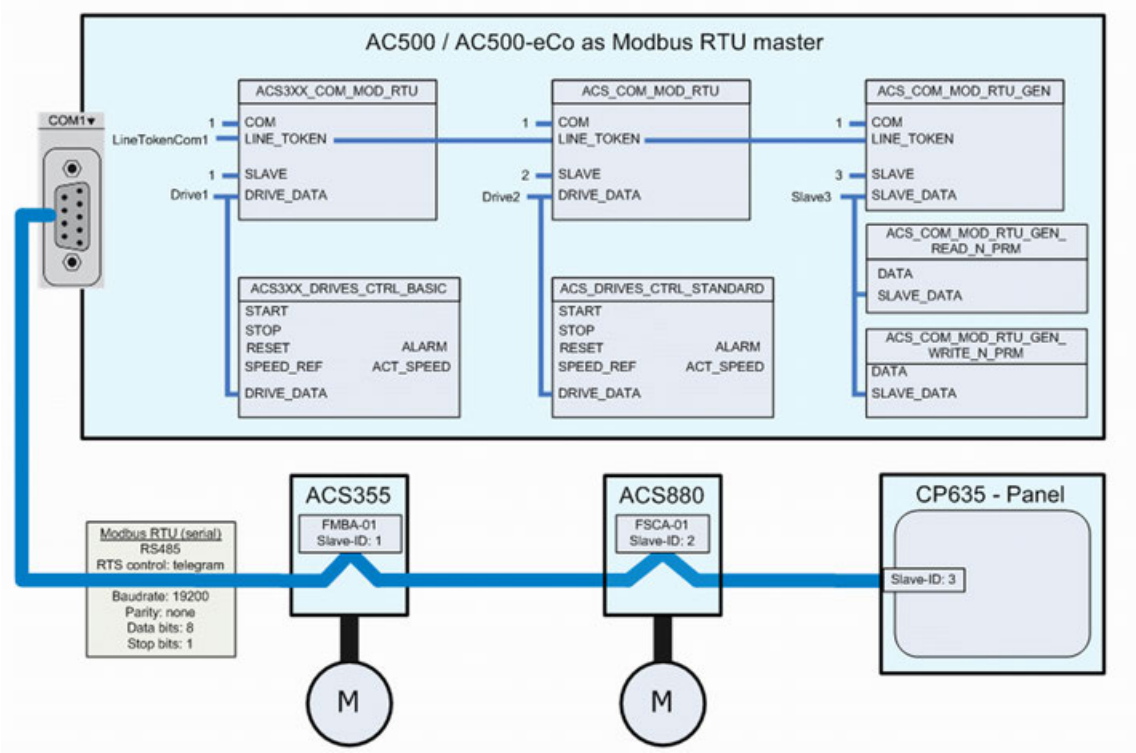


Fig. 158: Function Block Diagram PS553-DRIVES library Modbus RTU connection for three ACS drives

The communication function blocks are connected together via the LineToken variable. All function blocks regarding 1 drive are connected together via the DRIVE_DATA variable of type ACS_DRIVE_DATA_TYPE.

For the Modbus settings in the AC500 see tab PLC configuration \ Interfaces \ COMx – Modbus \ Module parameters – or in ABB Automation Builder: Setting_COMx_Modbus

For the Modbus settings in the ACS drives see Parameter group "EFB Protocoll", e.g. Par. 53.02 .. 53.04 for ACS355.

Performance

The following timing performance has been measured:

Test was done with no change of commands nor reference values. So just a continuous reading of status values from each drive were performed.

Test conditions general: PM554-T, timeout at ACS3XX_COM_MOD_RTU: 1000 ms

Transmission rate	**Number of drives**	**Update time for 1 drive**		
-	-	PLC scan time: 10ms PLC load: 50%	PLC scan time: 5ms PLC load: 74%	PLC scan time: 3ms PLC load: 75%
19,2 kBd	7	450 ms	360 ms	-
19,2 kBd	3	190 ms	-	147 ms
115,2 kBd	7	370 ms	280 ms	-
115,2 kBd	3	160 ms	-	100 ms

Reconnection pause:

When one or more drives in the Modbus RTU lines are offline, all the other drives have to wait for the TIMEOUT to elapse until a line token is assigned to next drive.

Reconnection pause input helps in skipping the drives which are offline from the next Modbus job and execute Modbus job operations only for the drives which are online.

“reconnectPause” is time in seconds before next retry to connect after a timeout was detected. Timeout is detected with ERNO = 8211.

This feature can be used with the ACS_COM_MOD_RTU or ACS_COM_MOD_RTU_ENHANCED function block when Modbus RTU communication is used.

User must configure the reconnect pause input value using the input/output variable “DRIVE_DATA.reconnectPause”.

For the generic RTU function block ACS_COM_MOD_RTU_GEN, value for the reconnect pause must be configured at the input/output variable “SLAVE_DATA.reconnectPause”.

1.5.6.3.3 Components of the ACS / DCS drives communication via Modbus RTU library

The library ACSDrivesComModRTU_AC500_V20.lib contains the following function blocks, structures and visualizations:

Table 132: Function blocks

Group: Modbus RTU -> ACS Drives	
ACS3XX_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293*	Communication for ACS 3xx or ACX550 Drives via Modbus RTU
ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*	Communication for ACS / DCS Drives via Modbus RTU
ACS_COM_MOD_RTU_ENHANCED ↳ *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312*	Communication for ACS / DCS Drives via Modbus RTU
Group: Modbus RTU -> Generic Devices	
ACS_COM_MOD_RTU_GEN ↳ *Chapter 1.5.6.3.5.4 “ACS_COM_MOD_RTU_GEN communication for generic devices via Modbus RTU ” on page 2322*	Communication for Generic Devices via Modbus RTU
ACS_COM_MOD_RTU_GEN_READ_N_PRM ↳ *Chapter 1.5.6.3.5.5 “ACS_COM_MOD_RTU_GEN_READ_N_PRM read N parameters from a generic Modbus RTU device ” on page 2327*	Read N Parameters from a Generic Modbus RTU Device
ACS_COM_MOD_RTU_GEN_WRITE_N_PRM ↳ *Chapter 1.5.6.3.5.6 “ACS_COM_MOD_RTU_GEN_WRITE_N_PRM write N parameters to a generic Modbus RTU device ” on page 2330*	Write N Parameters to a Generic Modbus RTU Device

Table 133: Structures

ACS_GEN_DEV_DATA_TYPE ↳ *Chapter 1.5.6.3.6.1 “ACS_GEN_DEV_DATA_TYPE structure to exchange data between function blocks for 1 generic device ” on page 2334*	Structure to Exchange Data between function blocks for 1 Generic Device

Table 134: Visualizations

Group: Modbus RTU -> ACS Drives	
ACS3XX_COM_MOD_RTU_VISU_PH ↳ *Chapter 1.5.6.3.7.1 “ACS3XX_COM_MOD_RTU_VISU_PH faceplate for the function block ACS3XX_COM_MOD_RTU” on page 2335*	Faceplate for the function block ACS3XX_COM_MOD_RTU
ACS_COM_MOD_RTU_VISU_PH ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301*	Faceplate for the function block ACS_COM_MOD_RTU
ACS_COM_MOD_RTU_ENHANCED_VISU_PH ↳ *Chapter 1.5.6.3.7.3 “ACS_COM_MOD_RTU_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_ENHANCED ” on page 2345*	Faceplate for the function block ACS_COM_MOD_RTU_ENHANCED
Group: Modbus RTU -> Generic Devices	
ACS_COM_MOD_RTU_GEN_VISU_PH ↳ *Chapter 1.5.6.3.7.4 “ACS_COM_MOD_RTU_GEN_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN ” on page 2350*	Faceplate for the function block ACS_COM_MOD_RTU_GEN
ACS_COM_MOD_RTU_GEN_READ_N_PRM_VISU_PH ↳ *Chapter 1.5.6.3.7.5 “ACS_COM_MOD_RTU_GEN_READ_N_PRM_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN_READ_N_PRM ” on page 2353*	Faceplate for the function block ACS_COM_MOD_RTU_GEN_READ_N_PRM
ACS_COM_MOD_RTU_GEN_WRITE_N_PRM_VISU_PH ↳ *Chapter 1.5.6.3.7.6 “ACS_COM_MOD_RTU_GEN_WRITE_N_PRM_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN_WRITE_N_PRM ” on page 2356*	Faceplate for the function block ACS_COM_MOD_RTU_GEN_WRITE_N_PRM

1.5.6.3.4 Overview of the ACS / DCS drives communication via Modbus RTU function blocks according to their call names

Used abbreviations:

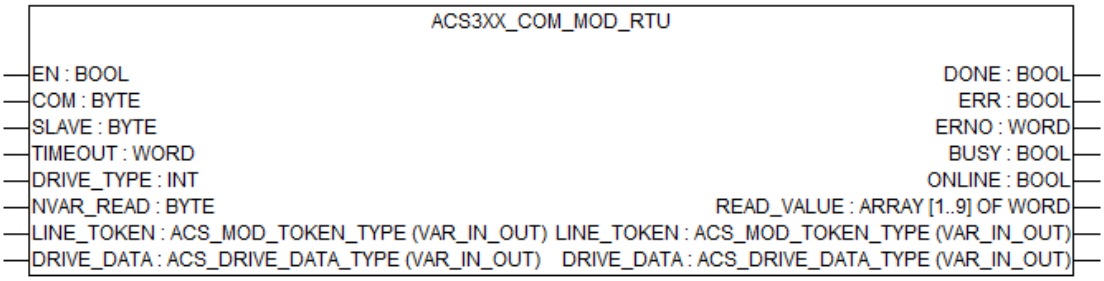
FBhv	Function block with historical values
FBnohv	Function block without historical values
F:	Function
Enum:	Enumeration

Struct:	Structure
Visu:	Visualization

VE Name	Type	Function
ACS3XX_COM_MOD_RTU	FBhv	Communication for ACS3xx and ACX550 Drives via Modbus RTU
ACS_COM_MOD_RTU	FBhv	Communication for ACS / DCS Drives via Modbus RTU
ACS_COM_MOD_RTU_ENHANCED	FBhv	Communication for ACS DCS Drives via Modbus RTU
ACS_COM_MOD_RTU_GEN	FBhv	Communication for Generic Devices via Modbus RTU
ACS_COM_MOD_RTU_GEN_READ_N_PRM	FBhv	Read N Parameters from a Generic Modbus RTU device
ACS_COM_MOD_RTU_GEN_WRITE_N_PRM	FBhv	Write N Parameters to a Generic Modbus RTU Device

1.5.6.3.5 Function blocks

ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU



Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

Function block ACS3XX_COM_MOD_RTU controls the Modbus RTU communication to an ACS3XX / ACX550 drive (except ACS380) and is used for the basic control of ACS3XX / ACX550 drives (except ACS380) with ABB Drives profile.

The function block continuously reads data from the drive starting at Modbus register 40004. So at least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable.

With input NVAR_READ the function block can be configured to read between 3 and 9 signals from the drive. All read data is also written to the array at the READ_VALUE output. The first signal is the Status Word. The following signals are configured in the ACS3XX / ACX550 drive with Par. 53.10 .. 53.17. The array READ_VALUE contains the data as follows:

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	**Mapping configuration in drive**		**Written to in AC500**	**Condition at function block**	
	ACS3XX, ACX550	**ACS355 default values using the Macro "AC500 Modbus" in Par. 99.02**			
Communication module	**embedded fieldbus or FMBA-01**				
40004	Status Word (SW)	Fix	DRIVE_DATA .sw READ_VALUE[1]	EN = TRUE NVAR_READ >= 3	
40005	Par. 53.10 (e.g.101)	Speed & direction (101)	DRIVE_DATA .actValue1 READ_VALUE[2]	EN = TRUE NVAR_READ >= 3	
40006	Par. 53.11 (e.g.105)	FB STS WORD 1 (303)	DRIVE_DATA .actValue2 READ_VALUE[3]	EN = TRUE NVAR_READ >= 3	
40007	Par. 53.12	FAULT WORD 1 (305)	READ_VALUE[4]		EN = TRUE and NVAR_READ >= 4
40008	Par. 53.13	0	READ_VALUE[5]		EN = TRUE and NVAR_READ >= 5
40012	Par. 53.17	0	READ_VALUE[9]		EN = TRUE and NVAR_READ = 9

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore in ACS3XX/ACX550 at least the Par. 53.10 and 53.11 have to be configured to the Actual Value1 and Actual Value2 e.g. 101 and 105.

If e.g. NVAR_READ = 5 the values from READ_VALUE[1 .. 5] are updated. Then none of the configured parameters Par. 53.10 .. 53.13 in the ACS3XX drive must be zero.

The READ_VALUE array is reset to zero in the following cases:

- EN = FALSE
- COM, SLAVE or TIMEOUT are out of range
- Communication error occurs

Writing Control Word and Reference Values to Drive

The function block checks, if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS3XX / ACX550 drive starting at Modbus register 40001.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive	Mapping configuration in drive	Taken form AC500
	For ACS3XX / ACX550 drives	
40001	Control Word (CW)	DRIVE_DATA.cw
40002	Reference Value1	DRIVE_DATA.refValue1
40003	Reference Value2	DRIVE_DATA.refValue2

Read/Write Jobs Coming from Other function blocks

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/ write function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM of this drive.

Communication with Several ACS3XX Drives

If several drives are used, for each drive an communication function block such as ACS3XX_COM_MOD_RTU must be programmed. Also every other generic slave device on the same Modbus RTU line must be programmed with it´s own ACS_COM_MOD_RTU_GEN function block. All those communication function blocks of one Modbus RTU line must be linked together via one variable of type ACS_MOD_TOKEN_TYPE, connected to the IN_OUTPUTs LINE_TOKEN. Via this variable the Modbus token is passed to the next drive / device, so only one drive / device at a time is communicating with the PLC.

Preconditions

The function block is only working with ACS3XX / ACX550 drives (ACS310, ACS350, ACS355, ACS500, ACH550) via Modbus RTU communication.

This function block cannot be used with ACS380 drive.

Instead for ACS380 drive use ACS_COM_MOD_RTU function block or ACS_COM_MOD_RTU_ENHANCED function block based on the drive configuration.

The data transfer to other function blocks for this drive communication to the ACS3XX/ACX550 drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to then ACS3XX_COM_MOD_RTU even if no other function block is connected.

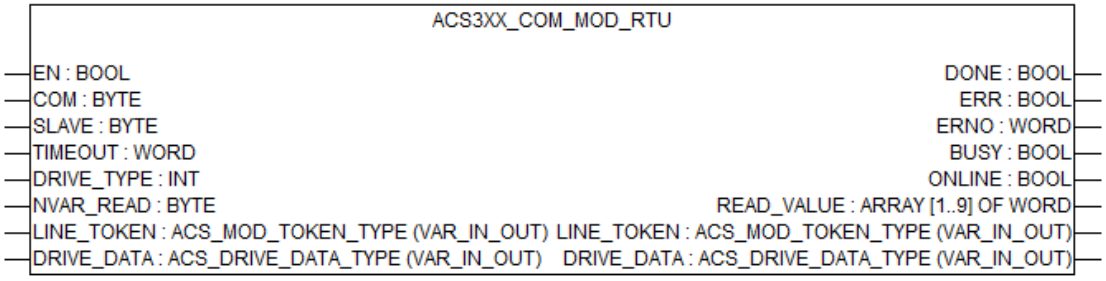
The Modbus token is passed to other Modbus communication blocks such as ACS3XX_COM_MOD_RTU via the IN_OUTPUT variable LINE_TOKEN which must be connected even if no other ACS3XX_COM_MOD_RTU function block is used.

The ACS3XX / ACX550 parameters must be set as follows:

Setting according to AC500 configuration or function block input Communication module:	ACS3XX, ACX550	ACS355 default values using the Macro "AC500 Modbus" in Par. 99.02	Condition at function block
	Embedded fieldbus or FMBA-01		
Fieldbus activation = STD Modbus or Modbus RS-232	98.02	STD Modbus	EN = TRUE and NVAR_READ >= 3
SLAVE number	53.02	2	EN = TRUE and NVAR_READ >= 3

Setting according to AC500 configuration or function block input **Communication module:**	**ACS3XX, ACX550** **Embedded fieldbus or FMBA-01**	**ACS355 default values using the Macro "AC500 Modbus" in Par. 99.02**	**Condition at function block**
Transmission rate = AC500 Modbus configuration	53.03	19.2 kbit/s (192)	EN = TRUE and NVAR_READ >= 3
Parity, Data- and Stopbits = AC500 Modbus configuration	53.04	8 NONE 1 (1)	EN = TRUE and NVAR_READ >= 3
Control profile = ABB Drives (lim or full) but not ABB drives enhanced nor DCU profile	53.05	ABB DRV FULL (2)	EN = TRUE and NVAR_READ >= 3
Mapping of Actual Value1 Modbus register 40005	53.10 e.g. 101	Speed & direction (101)	EN = TRUE and NVAR_READ >= 3
Mapping of Actual Value2 Modbus register 40006	53.11 e.g. 105	FB STS WORD 1 (303)	EN = TRUE and NVAR_READ >= 3
Optional Modbus register 40007	53.12 e.g. 305	FAULT WORD 1 (305)	EN = TRUE and NVAR_READ >= 4
Optional Modbus register 40012	53.17		EN = TRUE and NVAR_READ = 9

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS3XX / ACX550 drive.

After first successful reading and writing parameters, output ONLINE is set to TRUE.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) to the drive is started.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN is reset to FALSE while the function block actually keeps the token of the LINE_TOKEN variable the token will be released for another drive (set to 0).

COM (com)

Data type: BYTE

Interface identifier of Modbus line.

COM = 1: COM1

COM = 2: COM2

...

COM = 11: COM11 (using CM574-RS)

(COM3 = FPB, COM4 .. COM11 = COMs on CM574-RS).

Default value = 1. Minimum = 1, Maximum = 11

The COM input should not be changed while the program is running. If changed nevertheless, the new value will become effective only after a new rising edge of EN input.

SLAVE (slave)

Data type: BYTE

At input SLAVE, the address of the drive (slave) to which the connection shall be established must be specified.

Default value = 2, Minimum = 1, Maximum = 255

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed while the program is running. If changed nevertheless the new value will become effective only after an already running Modbus job is finished.

TIMEOUT (timeout)

Data type: WORD

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified at TIMEOUT, the procedure is aborted and an error identifier is output.

If several drives are connected to the Modbus line and one drive cannot respond, the whole communication is waiting till the timeout is over and the Modbus procedure (job) is aborted. Afterwards the next drive can take the LINE_TOKEN signal. The TIMEOUT should not be set too long if more than one drive is connected, but also not too short for having a chance to respond within the TIMEOUT. A minimum of 100 is required.

Typical values should be between 300 ms and 2000 ms (range: minimum 100, maximum 65535).

Default value = 1000. Minimum = 100, Maximum = 65535.

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

NVAR_READ (number of variables for reading)

Data type: BYTE

With the input NVAR_READ the function block can be configured to read between 3 and 9 signals from the drive. All read data is written to the array at the READ_VALUE output. The first value is the Status Word (SW). The following values are configured in the ACS3XX / ACX550 drive with Par. 53.10 .. 53.17.

If e.g. NVAR_READ = 5 then none of the configuration parameters Par. 53.10 .. 53.13 must be 0. Then the values from READ_VALUE[1] .. READ_VALUE[5] are updated.

Default value = 3. Minimum = 3, Maximum = 9.

For ACS3XX/ACX550 drives, mapping of 53.10 and 53.11 parameters to any of the drive parameter is mandatory, otherwise function block will return error.

When NVAR READ = 3, it reads SW, 53.10 mapped parameter and 53.11 mapped parameter.

See table at description of READ_VALUE output.

LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

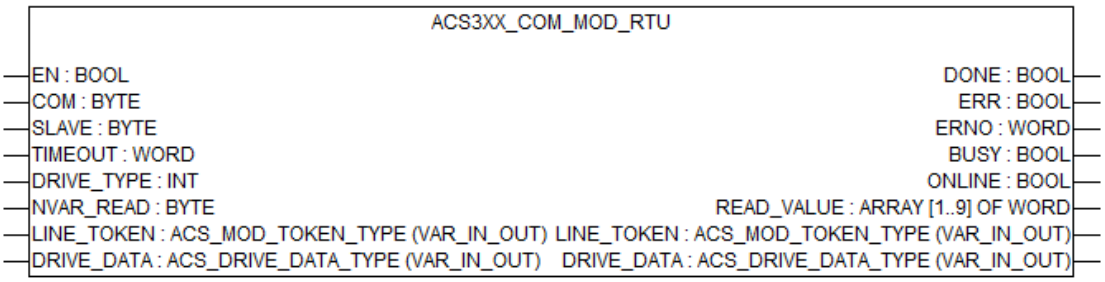
The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

Output description



LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy) Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

READ_VALUE (actual spee-darray of read values) Data type: ARRAY[1..9] OF WORD

At output READ_VALUES the values of the array are updated after the read status information job was terminated successfully (DONE = TRUE, ERR = FALSE, BUSY = FALSE).

The read status information job is requested cyclically. It reads data from the ACS drive starting at Modbus register 400051 up to the number specified at input NVAR_READ +3.

READ_VALUES contains the data as follows:

READ_VALUES[1] = <Modbus register 400054>

READ_VALUES[2] = <Modbus register 400055>

...

READ_VALUES[12] = <Modbus register 400065>.

Function call in ST:

```
ACS3XXComModRTU   (EN         := xACS3XXComModRTU_EN,
                               COM         := byACS3XXComModRTU_COM,
                               SLAVE       := byACS3XXComModRTU_SLAVE,
                               TIMEOUT     := wACS3XXComModRTU_TIMEOUT,
```

```
                                DRIVE_TYPE := eACS3XXComModRTU_DRIVE_TYPE,

                                NVAR_READ  := byACS3XXComModRTU_NVAR_READ,

                                LINE_TOKEN := tsLineToken,

                                DRIVE_DATA := tsDriveData);
xACS3XXComModRTU_DONE          := ACS3XXComModRTU.DONE;
xACS3XXComModRTU_ERR           := ACS3XXComModRTU.ERR;
wACS3XXComModRTU_ERNO          := ACS3XXComModRTU.ERNO;
xACS3XXComModRTU_BUSY          := ACS3XXComModRTU.BUSY;
xACS3XXComModRTU_ONLINE        := ACS3XXComModRTU.ONLINE;
awACS3XXComModRTU_READ_VALUE   := ACS3XXComModRTU.READ_VALUE;
```

ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU

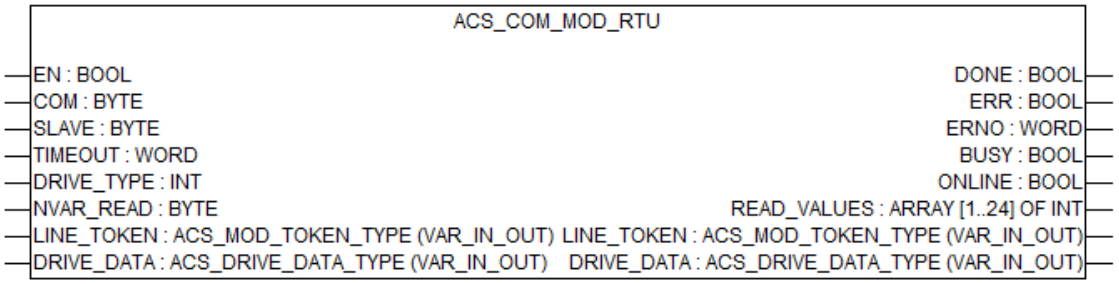


Fig. 159: Function block ACS_COM_MOD_RTU controls the Modbus RTU communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives Profile.

Function block information

Available from runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

Function block ACS_COM_MOD_RTU controls the Modbus RTU communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives profile. Reading Status Information from Drive The function block continuously reads data from the drive starting at Modbus register 40004. At least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable.

With input NVAR_READ the function block can be configured to read in the same job between 0 .. 24 data more from the drive. These additional data is written to the array at the READ_VALUES output. These data have to be configured in the drive and are only accessible if the embedded Modbus is used. See table below for detailed information.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Modbus register address in drive *)	**Mapping configuration in drive**					**Written to in AC500**		**Condition at function block**
	ACS3XX, ACX550	**ACS850, ACQ810**	**ACS580**	**ACS850, ACQ810, ACSM1, ACS880,**	**ACS800**	**DCS550, DCS800**		
Communication module	embedded fieldbus	embedded fieldbus	embedded fieldbus	FSCA-01	RMBA-01	RMBA-01		
40004	Status Word (SW)	Status Word (SW)	Par.: 58.104 = 4 - Status Word (SW)	Status Word (SW)	Status Word (SW)	Par.: 92.01=801 Status Word (SW)	DRIVE_DATA.sw	EN = TRUE
40005	Par. 53.10 (e.g.101)	Actual Value1	Par.: 58.105 = 5 – Actual Value1	Actual Value 1	Par.: 92.02 = Actual Value1 e.g. = 102 (Speed)	Par.: 92.02 = 104 Actual Value1 e.g. = 102 (Speed)	DRIVE_DATA.act-Value1	EN = TRUE
40006	Par. 53.11 (e.g.105)	Actual Value2	Par.: 58.106 = 6 – Actual Value2	Actual Value2	Par.: 92.03 = Actual Value2 e.g. = 105 (Torque) **)	92.03 = 209 Actual Value2 e.g. = 105 (Torque) **)	DRIVE_DATA.act-Value2	EN = TRUE
40007	Par. 53.12	Par. 58.35	Par.: 58.107				READ_VALUES[1]	EN = TRUE and NVAR_READ >= 1
40008	Par. 53.13	Par. 58.36	Par.: 58.108				READ_VALUES[2]	EN = TRUE and NVAR_READ >= 2
40012	Par. 53.17	Par. 58.40	Par.: 58.112				READ_VALUES[6]	EN = TRUE and NVAR_READ >= 6
40014		Par. 58.42	Par.: 58.114				READ_VALUES[8]	

Modbus register address in drive *)	Mapping configuration in drive					Written to in AC500		Condition at function block
	ACS3XX, ACX550	ACS850, ACQ810	ACS580	ACS850, ACQ810, ACSM1, ACS880,	ACS800	DCS550, DCS800		
40030		Par. 58.58					READ_VALUES[24]	EN = TRUE and NVAR_READ = 24
*) For ACS850, ACQ810 and external fieldbus adapters are having 6 digit Modbus address. Eg; ACS3XX Status Word Modbus address is 40004 and ACS850 Modbus address is 400004. **) If 51.19 .. 51.22 (Output 1 .. 4) are set to the actual values the Modbus response will be faster because those values are updated cyclically between RETA-01 and ACS800.								

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore in ACS3XX/ACX550 at least the Par. 53.10 and 53.11 have to be configured to the Actual Value1 and Actual Value2 e.g. 101 and 105.

If e.g. NVAR_READ = 5 the values from READ_VALUES[1 .. 5] are updated. Then all of the configured parameters Par. 53.12 ... 53.16 in the ACS3XX/ACX550 or Par. 58.35 & 58.38 in ACS850/ACQ810 drive must contain valid mapping information (not zero).

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 40001.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Table 135: Reading status information from drive

Modbus register address in drive	Mapping configuration in drive			Taken form AC500	Condition at function block
	For all ACS drives	DCS 800 / DCS550	ACS 580		
40001	Control Word (CW)	Par.: 90.01 = 701 – Control Word (CW)	Par.: 58.101 = 1 – Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
40002	Reference Value1	Par.: 90.02 = 2301 – Reference Value 1	Par.: 58.102 = 2 – Referencec Value 1	DRIVE_DATA.refValue1	EN = TRUE
40003	Reference Value2	Par.: 90.03 = 2501 – Reference Value 2	Par.: 58.103 = 3 – Referencec Value 2	DRIVE_DATA.refValue2	EN = TRUE

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM function blocks of this drive.

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_RTU must be programmed. Also every other generic slave device on the same Modbus RTU line must be programmed with it´s own ACS_COM_MOD_RTU_GEN function block. All those communication function blocks of one Modbus RTU line must be linked together via one variable of type ACS_MOD_TOKEN_TYPE, connected to the IN_OUTPUTs LINE_TOKEN. Via this variable the Modbus token is passed to the next drive / device, so only one drive / device at a time is communicating with the PLC.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>
- Create a visualization element of the function block ACS_COM_MOD_RTU_VISU_PH.

The additional diagnosis variables are:

iWriteErrCnt:	number of errors in write jobs since EN = TRUE
wLastWriteErno:	holds the error number of the last executed write job
iReadErrCnt:	number of errors in read jobs since EN = TRUE
wLastReadErno:	holds the error number of the last executed read job

Preconditions

The function block is working with all ACS / DCS drives via Modbus RTU communication.

The data transfer to other function blocks for this drive communication to the ACS /DCSdrive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to the ACS_COM_MOD_RTU even if no other function block is connected.

The Modbus token is passed to other communication function blocks such as ACS_COM_MOD_RTU or ACS_COM_MOD_RTU_GEN via the IN_OUTPUT variable LINE_TOKEN which must be connected even if no other of those communication function blocks is used.

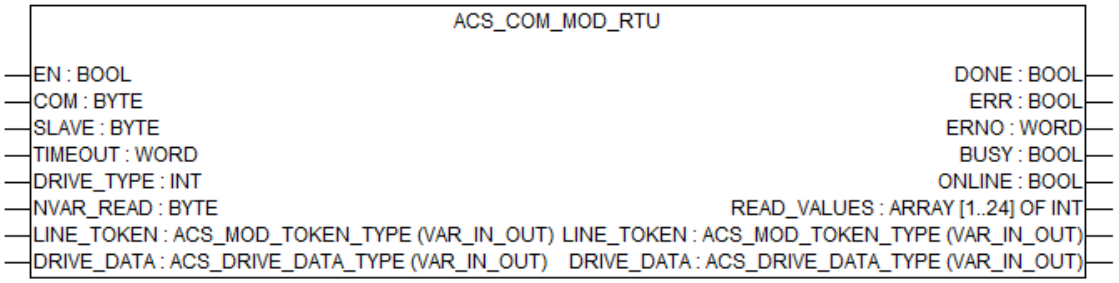
The following ACS drive parameters have to be set according the configuration for the Modbus line and the inputs of the function block.

Setting according to AC500 configuration or function block input	**ACS3XX, ACX550**	**ACS850, ACQ810**	**ACS580**	**ACS850, ACQ810, ACSM1,**	**ACS880**	**ACS800**
Communication module:	direct or FMBA-01	embedded fieldbus	embedded fieldbus	FSCA-01	FSCA-01	RMBA-01
Fieldbus activation = STD Modbus or Modbus RS-232	98.02	58.01	58.01	50.01	50.01	98.02
SLAVE number	53.02	58.03	58.03	51.03	51.03	52.01
Transmission rate = AC500 Modbus configuration	53.03	58.04	58.04	51.04	51.04	52.02
Parity, Data- and Stopbits = AC500 Modbus configuration	53.04	58.05	58.05	51.05	51.05	52.03
Control profile = ABB Drives (lim or full) but not ABB drives enhanced nor DCU profile	53.05	58.06	58.25	51.02	51.02	98.07
Mapping of Control Word Modbus register 40001	Fix	Fix	58.101	Fix		
Mapping of Reference Value 1 Modbus register 40002	Fix	Fix	58.102			
Mapping of Reference Value 2 Modbus register 40003	Fix	Fix	58.103	Fix		

Setting according to AC500 configuration or function block input	ACS3XX, ACX550	ACS850, ACQ810	ACS580	ACS850, ACQ810, ACSM1,	ACS880	ACS800
Mapping of Status Word Modbus register 40004	Fix	Fix	58.104	Fix		
Mapping of Actual Value1 Modbus register 40005	53.10 e.g. 101	Fix	58.105	Fix	Fix	Fix
Mapping of Actual Value2 Modbus register 40006	53.11 e.g. 105	Fix	58.106	Fix	Fix	Fix
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as theses values are only written after changes.		58.08	58.15	51.07	Timeout mode	Fix monitoring of Main and Auxiliary data sets. See Manual Par.30.18.
Modbus timeout. Depending on Timeout mode. Value in 100ms.		58.07	58.16	51.06	Modbus timeout	Fix
Refresh settings in drive		58.10	58.06	51.27	51.27	

For further settings, e.g. reaction of drive at communication error, please see related drive or fieldbus adapter manual.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACSXXX drive.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of controls and Reference Values (CW, SPEED_REF, REF_VALUE2) (if they are changed) to the drive is started.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN is reset to FALSE while the function block actually keeps the token of the LINE_TOKEN variable, the token will be released for another drive (set to 0) as soon as no Modbus job is performed any more.

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero. The elements of the READ_VALUES array are also reset to zero.

COM (com)

Data type: BYTE

Interface identifier of Modbus line.

COM = 1: COM1

COM = 2: COM2

...

COM = 11: COM11 (using CM574-RS)

(COM3 = FPB, COM4 .. COM11 = COMs on CM574-RS).

Default value = 1. Minimum = 1, Maximum = 11

The COM input should not be changed while the program is running. If changed nevertheless, the new value will become effective only after a new rising edge of EN input.

SLAVE (slave)

Data type: BYTE

At input SLAVE, the address of the drive (slave) to which the connection shall be established must be specified.

Default value = 2, Minimum = 1, Maximum = 255

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed while the program is running. If changed nevertheless the new value will become effective only after an already running Modbus job is finished.

TIMEOUT (timeout)

Data type: WORD

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified at TIMEOUT, the procedure is aborted and an error identifier is output.

If several drives are connected to the Modbus line and one drive cannot respond, the whole communication is waiting till the timeout is over and the Modbus procedure (job) is aborted. Afterwards the next drive can take the LINE_TOKEN signal. The TIMEOUT should not be set too long if more than one drive is connected, but also not too short for having a chance to respond within the TIMEOUT. A minimum of 100 is required.

Typical values should be between 300 ms and 2000 ms (range: minimum 100, maximum 65535).

Default value = 1000. Minimum = 100, Maximum = 65535.

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

NVAR_READ (number of variables for reading)

Data type: BYTE

The internal Modbus read job reads 3 + NVAR_READ words starting from Modbus register address 40004 from the drive.

Only for ACS drives using embedded fieldbus option, such as ACS3xx, ACx550, ACS850 and ACQ810 the input NVAR_READ should be set higher than 0.

With the input NVAR_READ the function block can be configured to read between 0 and 24 data more from the drive in addition to the 3 signals that are always read from Modbus registers addresses 40004 .. 40006.

If NVAR_READ input is set to e.g. 5 the internal read job addresses the Modbus registers from 40004 .. 40001.

See table *Table 135 “Reading status information from drive” on page 2303*.

The first 3 signals are always the Status Word (SW), Actual Value1 and Actual Value2 and will be written to the *“DRIVE_DATA (drive data)” on page 2309* variable.

The additional NVAR_READ data will be written to the array at the READ_VALUES output.

Example:

If e.g. NVAR_READ = 5 and all mapping Parameters, Par. 53.10 ... 53.13 for ACS3XX / ACX550 or Par. 58.35 ... 58.39 for ACS850 / ACQ810 are valid (not 0), then the values from READ_VALUES[1] to READ_VALUES[5] are updated.

Default value = 0, Minimum 0, Maximum 24.

To read/write "one" 32-bit data, the NVAR should be equal to 2. Accordingly this has to be followed if we want to read/write more than one data.

LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

DRIVE_DATA (drive data)

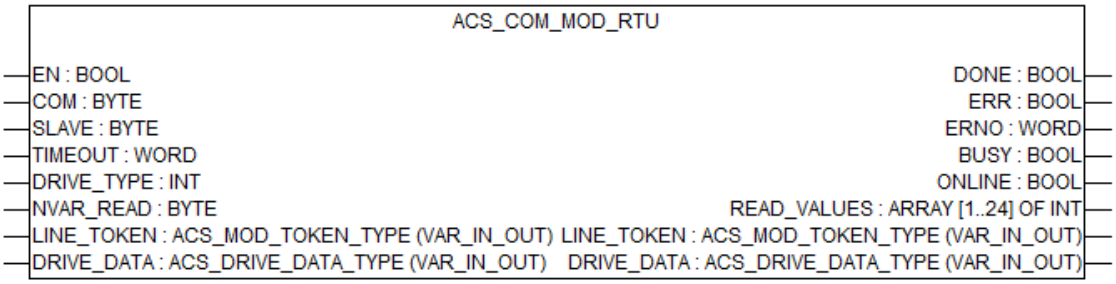
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online)

Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

READ_VALUES (array of read values)

Data type: ARRAY[1..12] OF INT

At output READ_VALUES the values of the array are updated after the read status information job was terminated successfully (DONE = TRUE, ERR = FALSE, BUSY = FALSE).

The read status information job is requested cyclically. It reads data from the ACS drive starting at Modbus register 400051 up to the number specified at input 3 + NVAR_READ

READ_VALUES contains the data as follows:

READ_VALUES[1] = <Modbus register 400054>

READ_VALUES[2] = <Modbus register 400055>

...

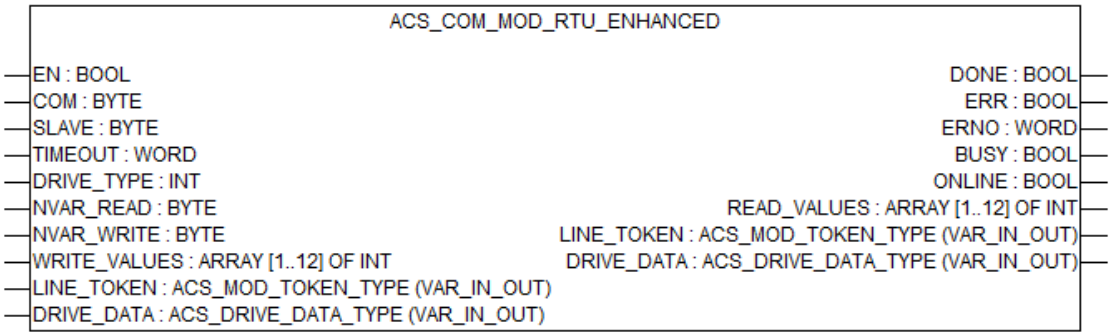
READ_VALUES[12] = <Modbus register 400065>.

Function call in ST

ACSComModRTU	(EN	= xACSComModRTU_EN,
	COM	:= byACSComModRTU_COM,
	SLAVE	:= byACSCom-ModRTU_SLAVE,
	TIMEOUT	:= wACSCom-ModRTU_TIMEOUT,
	DRIVE_TYPE	:= eACSCom-ModRTU_DRIVE_TYPE,

	NVAR_READ	:= byACSCom-ModRTU_NVAR_READ,
	LINE_TOKEN	:= tsLineToken,
	DRIVE_DATA	:= tsDriveData);
xACSComModRTU_DONE		:= ACSComModRTU.DONE;
xACSComModRTU_ERR		:= ACSComModRTU.ERR;
wACSComModRTU_ERNO		:= ACSComModRTU.ERNO;
xACSComModRTU_BUSY		:= ACSComModRTU.BUSY;
xACSComModRTU_ONLINE		:= ACSCom-ModRTU.ONLINE;
awACSCom-ModRTU_READ_VALUES		:= ACSCom-ModRTU.READ_VALUES;

ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced



Function block information

Available from runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

Function block ACS_COM_MOD_RTU_ENHANCED establishes the Modbus RTU communication to an ACS drive and is used for the basic control of ACS drives with ABB Drives Profile Enhanced.

The ABB Drives Profile Enhanced communication profile provides register mapped access to the Control, Status, Reference and Actual Values of the ABB Drives Profile Enhanced. The mapping of the registers has been enhanced to allow additionally writing of up to 12 control and reading of up to 12 additionally status parameters in a single Modbus job.

The function block continuously reads data from the drive starting at Modbus register 400051. The Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable. These values are stored in MSW, ActValue1 and ActValue2.

Apart from these three parameters there is also an option to read additional 12 more drive parameters in the same Modbus job. Using the input NVAR_READ the function block can be configured to read between 1 and 12 more parameters from the drive. All read data is then written to the array at the READ_VALUES output.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Table 136: Reading status information from the drive

Modbus register address in drive	Mapping configuration in drive		Written to in AC500	Condition at function block
	ACS850, ACQ810 embedded fieldbus	**ACS850, ACQ810, ACSM1, ACS880, ACS800, ACS580**		
400051	Status Word (SW)	Status Word (SW)	DRIVE_DATA.sw	EN = TRUE
400052	Actual Value1	Actual Value1	DRIVE_DATA.act Value1	EN = TRUE
400053	Actual Value2	Actual Value2	DRIVE_DATA.act Value2	EN = TRUE
400054	Par. 58.47	FBA DATA IN 1	READ_VALUES[1]	EN = TRUE and NVAR_READ >= 1
400055	Par. 58.48	FBA DATA IN 2	READ_VALUES[2]	EN = TRUE and NVAR_READ >= 2
400065	Par. 58.58	FBA DATA IN 12	READ_VALUES[6]	EN = TRUE and NVAR_READ = 12

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore the drive parameters in FBA DATA IN group have to be configured according to the used NVAR_READ input number.

If 32-bit parameters are mapped to DATA IN,

- *The following field in DATA IN has to be left open (= 0) To retrieve the original 32-bit value from the drive in AC500 the HW and LW from READ_VALUES fields have to be recombined in the program.*
- *The word order of the High-Word and Low-Word can be configured in the drive. If e.g. FSCA-01 is used the configuration is done in Par.51.11.*
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from READ_VALUES fields have to be recombined in the program.*

Function block DATA IN has to be configured in drive in the following groups see also FSCA-01 manual.

Drive	Parameter group
ACS355	FSCA-01 is not released 54.01 .. 54.10
ACS850, ACQ810, ACSM1, ACS880	52.01 .. 52.12

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 400001.

Apart from these three parameters there is also an option to write additional 12 more drive parameters in the same Modbus job. Using the input NVAR_WRITE the function block can be configured to write between 1 and 12 more parameters to the drive. The necessary values have to present in the array connected to WRITE_VALUES input.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area from where the data in the AC500 is taken.

Modbus register address in drive	**Mapping configuration in drive**		**Written to in AC500**	**Condition at function block**
	ACS850, ACQ810 embedded fieldbus	**ACS850, ACQ810, ACSM1, ACS880, ACS800**		
400001	Control Word (CW)	Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
400002	Reference Value1	Actual Value1	DRIVE_DATA.ref Value1	EN = TRUE
400003	Reference Value2	Actual Value2	DRIVE_DATA.ref Value2	EN = TRUE
400004	Par. 58.35	FBA DATA OUT 1	WRITE_VALUES[1]	EN = TRUE and NVAR_READ >= 1
400005	Par. 58.36	FBA DATA OUT 2	WRITE_VALUES[2]	EN = TRUE and NVAR_READ >= 2
400015	Par. 58.46	FBA DATA OUT 12	WRITE_VALUES[6]	EN = TRUE and NVAR_READ = 12

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore the drive parameters in FBA DATA IN group have to be configured according to the used NVAR_READ input number.

If 32-bit parameters are mapped to DATA OUT,

- *The following field in DATA OUT has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. If e.g. FSCA-01 is used the configuration is done in Par.51.11.*
- *The original 32-bit value in AC500 has to be split up in HW and LW in the WRITE_VALUES array.*

Function block DATA IN has to be configured in drive in the following groups see also FSCA-01 manual.

Drive	Parameter group
ACS355	54.01 .. 54.10
ACS850, ACQ810, ACSM1, ACS880	52.01 .. 52.12

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set. Please see drive manuals and following table which parameter has to be set

Save valid parameters to permanent memory in drive	ACS3XX, ACX550, ACQ810, ACS850, ACSM1, ACS800	ACS880
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/ write function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM of this drive.

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_RTU_ENHANCED must be programmed. Also every other generic slave device on the same Modbus RTU line must be programmed with it´s own ACS_COM_MOD_RTU_GEN function block. All those communication function blocks of one Modbus RTU line must be linked together via one variable of type ACS_MOD_TOKEN_TYPE, connected to the IN_OUTPUT LINE_TOKEN. Via this variable the Modbus token is passed to the next drive / device, so only one drive / device at a time is communicating with the PLC.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>
- Create a visualization element of the function block.

The additional diagnosis variables are:

iWriteErrCnt:	number of errors in write jobs since EN = TRUE
wLastWriteErno:	holds the error number of the last executed write job
iReadErrCnt:	number of errors in read jobs since EN = TRUE
wLastReadErno:	holds the error number of the last executed read job

Preconditions

The function block is working with all ACS drives via Enhanced Modbus RTU communication.

The data transfer to other function blocks for this drive communication to the ACS drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to ACS_COM_MOD_RTU_ENHANCED even if no other function block is connected.

The Modbus token is passed to other ACS_COM_MOD_RTU_ENHANCED function via the IN_OUTPUT variable LINE_TOKEN which must be connected even if no other ACS_COM_MOD_RTU_ENHANCED function block is used.

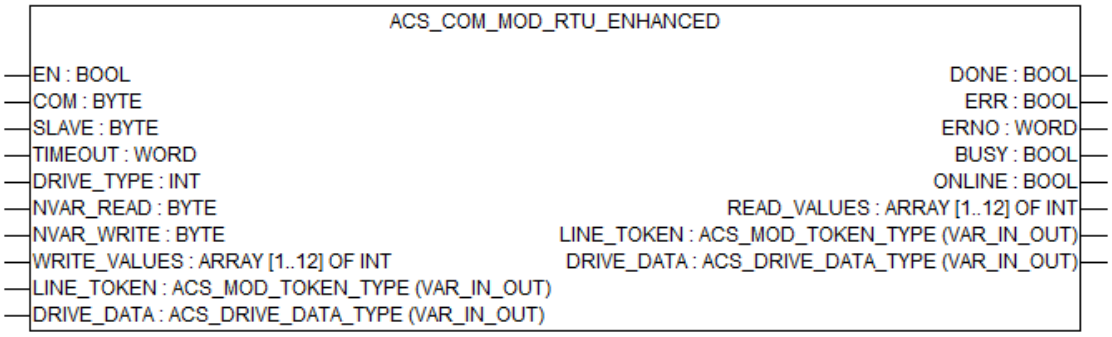
The following ACS drive parameters have to be set according to the configuration of the Modbus line and the inputs of the function block.

Setting according to AC500 configuration or function block input	**ACS355**	**ACS850, ACQ810**	**ACS850, ACQ810, ACSM1,**	**ACS880**	**ACS800**
Communication module:	FSCA-01	embedded fieldbus	FSCA-01	FSCA-01	FSCA-01
Fieldbus activation = STD Modbus or Modbus RS-232 or EXT FBA	98.02	58.01	50.01	50.01	98.02
SLAVE number	51.03	58.03	51.03	51.03	52.01
Transmission rate = AC500 Modbus configuration	51.04	58.04	51.04	51.04	52.02
Parity, Data- and Stopbits = AC500 Modbus configuration	51.05	58.05	51.05	51.05	52.03
Control profile = ABB Drives Profile Enhanced	53.05	58.06	51.02	51.02	98.07
Word order for 32-bit parameter access	No 32-bit access	58.32	51.11	51.11	51.11
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as theses values are only written after changes.	51.07	58.08	51.07	Timeout mode	Fix monitoring of Main and Auxiliary data sets. See Manual Par.30.18.

Setting according to AC500 configuration or function block input	ACS355	ACS850, ACQ810	ACS850, ACQ810, ACSM1,	ACS880	ACS800
Modbus timeout. Depending on Timeout mode. Value in 100ms.	51.06	58.07	51.06	Modbus timeout	Fix
Refresh settings in drive	51.27	58.10	51.27	51.27	

For further settings, e.g. reaction of drive at communication error, please see related drive manual.

Input descriptions



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS drive.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) (if they are changed) to the drive is started.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN is reset to FALSE while the function block actually keeps the token of the LINE_TOKEN variable the token will be released for another drive (set to 0) as soon as no Modbus job is performed any more.

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero. The elements of the READ_VALUES array are also reset to zero.

COM (com)

Data type: BYTE

Interface identifier of Modbus line.

COM = 1: COM1

COM = 2: COM2

...

COM = 11: COM11 (using CM574-RS)

(COM3 = FPB, COM4 .. COM11 = COMs on CM574-RS).

Default value = 1. Minimum = 1, Maximum = 11

The COM input should not be changed while the program is running. If changed nevertheless, the new value will become effective only after a new rising edge of EN input.

SLAVE (slave)

Data type: BYTE

At input SLAVE, the address of the drive (slave) to which the connection shall be established must be specified.

Default value = 2, Minimum = 1, Maximum = 255

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed while the program is running. If changed nevertheless the new value will become effective only after an already running Modbus job is finished.

TIMEOUT (timeout)

Data type: WORD

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified at TIMEOUT, the procedure is aborted and an error identifier is output.

If several drives are connected to the Modbus line and one drive cannot respond, the whole communication is waiting till the timeout is over and the Modbus procedure (job) is aborted. Afterwards the next drive can take the LINE_TOKEN signal. The TIMEOUT should not be set too long if more than one drive is connected, but also not too short for having a chance to respond within the TIMEOUT. A minimum of 100 is required.

Typical values should be between 300 ms and 2000 ms (range: minimum 100, maximum 65535).

Default value = 1000. Minimum = 100, Maximum = 65535.

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6

ENUM	Value
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

NVAR_READ (number of variables for reading)

Data type: BYTE

The internal Modbus read job reads 3 + NVAR_READ words starting from Modbus register address 400051 from the drive.

With the input NVAR_READ the function block can be configured to read between 0 and 12 data more from the drive in addition to the 3 signals that are always read from Modbus registers addresses 400051 .. 400053.

If NVAR_READ input is set to e.g. 5 the internal read job addresses the Modbus registers from 400051 .. 400058.

The first 3 signals are always the Status Word (SW), Actual Value1 and Actual Value2 and will be written to the *“DRIVE_DATA (drive data)” on page 2320* variable. The additional NVAR_READ data will be written to the array at the READ_VALUES output.

See table *Table 136 “Reading status information from the drive” on page 2313* about how to configure the drive and where the data is stored in AC500.

Default value = 0. Minimum = 0, Maximum = 12.

To read/write "one" 32-bit data, the NVAR should be equal to 2. Accordingly this has to be followed if we want to read/write more than one data.

NVAR_WRITE (number of variables for writing)

Data type: BYTE

The internal Modbus write job writes 3 + NVAR_WRITE words starting from Modbus register address 400001 to the drive, every time a change in those variables is detected.

With the input NVAR_WRITE the function block can be configured to write between 0 and 12 variables more to the drive in addition to the 3 controls (CW, ref1 and ref2). These are always written to Modbus registers addresses 400001 .. 400003 if changed.

If NVAR_WRITE input is set to e.g. 5 the internal write job addresses the Modbus registers from 400001 .. 400008.

The first 3 signals are always the Control Word (CW), Reference value1 and Reference value2 and will be taken from the DRIVE_DATA variable.

The additional NVAR_WRITE data will be taken from the array at the WRITE_VALUES input.

Default value = 0. Minimum 0, Maximum 12.

WRITE_VALUES (write values to mapped parameters in drive group DATA_OUT)

Data type: ARRAY[1..12] OF INT

The values from the array at input WRITE_VALUE will be written to Modbus registers 400004 400015 in the drive. The number of data written to the drive is specified at the input NVAR_WRITE.

See table in chapter Writing Control Word and Reference Values to the drive for information about how to configure the drive.

LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

DRIVE_DATA (drive data)

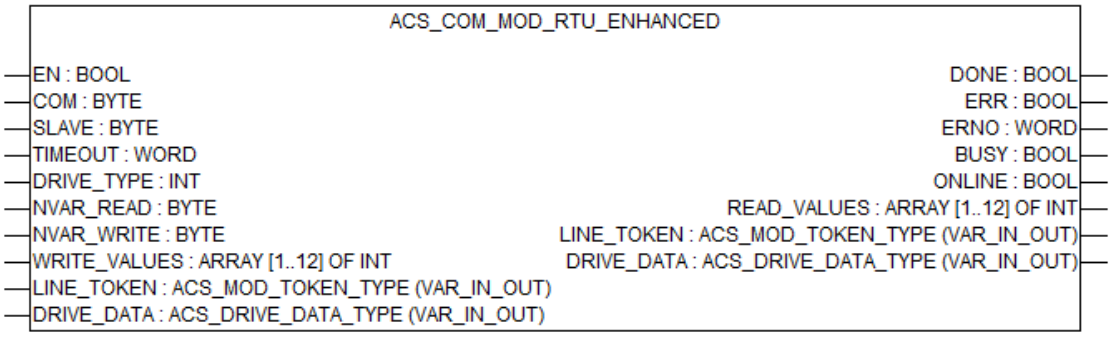
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output descriptions



LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↳ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_RTU_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

READ_VALUES (array of read values) Data type: ARRAY[1..12] OF INT

At output READ_VALUES the values of the array are updated after the read status information job was terminated successfully (DONE = TRUE, ERR = FALSE, BUSY = FALSE).

The read status information job is requested cyclically. It reads data from the ACS drive starting at Modbus register 400051 up to the number specified at input 3 + NVAR_READ

READ_VALUES contains the data as follows:

READ_VALUES[1] = <Modbus register 400054>

READ_VALUES[2] = <Modbus register 400055>

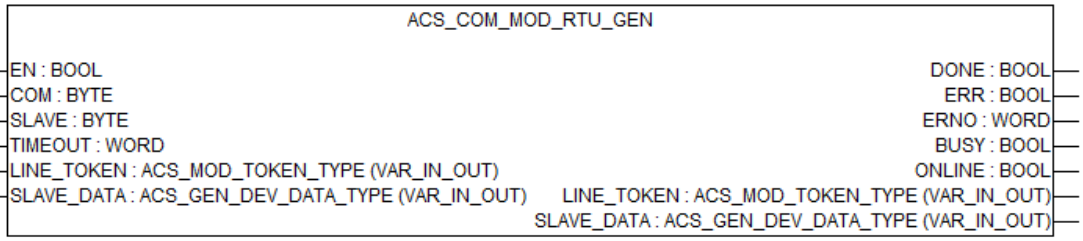
...

READ_VALUES[12] = <Modbus register 400065>.

Function call in ST

```
ACSComModRTUEnhanced    (EN    = xACSComModRTUEnhanced_EN,
     COM    := byACSComModRTUEnhanced_COM,
     SLAVE    := byACSComModRTUEnhanced_SLAVE,
     TIMEOUT    := wACSComModRTUEnhanced_TIMEOUT,
     DRIVE_TYPE    := eACSComModRTUEnhanced_DRIVE_TYPE,
     NVAR_READ    := byACSComModRTUEnhanced_NVAR_READ,
     NVAR_WRITE    := byACSComModRTUEnhanced_NVAR_WRITE,
     WRITE_VALUES    := aiACSComModRTUEnhanced_WRITE_VALUES,
     LINE_TOKEN    := tsLineToken,
     DRIVE_DATA    := tsDriveData);

xACSComModRTUEnhanced_DONE    := ACSComModRTUEnhanced.DONE;
xACSComModRTUEnhanced_ERR    := ACSComModRTUEnhanced.ERR;
wACSComModRTUEnhanced_ERNO    := ACSComModRTUEnhanced.ERNO;
xACSComModRTUEnhanced_BUSY    := ACSComModRTUEnhanced.BUSY;
xACSComModRTUEnhanced_ONLINE    := ACSComModRTUEnhanced.ONLINE;
awACSComModRTUEnhanced_READ_VALUES    :=
ACSComModRTUEnhanced.READ_VALUES;
```

ACS_COM_MOD_RTU_GEN communication for generic devices via Modbus RTU



Function block ACS_COM_MOD_RTU_GEN controls the Modbus RTU communication to a generic slave device. It must be used together with the function blocks ACS_COM_MOD_RTU_GEN_READ_N_PRM and/or ACS_COM_MOD_RTU_GEN_WRITE_N_PRM to exchange Modbus data. function block ACS_COM_MOD_RTU_GEN controls the Modbus RTU communication to a generic Modbus RTU server device. A generic Modbus RTU server device can be any field device which supports Modbus RTU server within it such as PLC, HMI or ABB ACS/DCS drive etc. It must be used together with the function blocks ACS_COM_MOD_RTU_GEN_READ_N_PRM and/or ACS_COM_MOD_RTU_GEN_WRITE_N_and/or ModRtuReadWrite23 to exchange Modbus data.

Block data

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

If more devices are connected to the same Modbus RTU line, for each of them an own instance of ACS_COM_MOD_RTU_GEN or ACS_COM_MOD_RTU_ENHANCED or ACS_COM_MOD_RTU (for connection to an ABB ACS/DCS drive) function block must be used. All these function blocks must be connected to the same LineToken variable of type ACS_MOD_TOKEN_TYPE at their IN_OUT LINE_TOKEN. Via this LINE_TOKEN variable the serial access to the different devices is controlled. All these blocks must be called within the same PLC task.

At the IN_OUT SLAVE_DATA a variable of type ACS_GEN_DEV_DATA_TYPE must be connected, which inturn connected to the Modbus read/write blocks related to the same device. Via these blocks ACS_COM_MOD_RTU_GEN_READ_N_PRM and ACS_COM_MOD_RTU_GEN_WRITE_N_PRM the Modbus jobs are initiated in the order they are programmed. The requests to process these read or write Modbus jobs is transferred via the SLAVE_DATA variable to the ACS_COM_MOD_RTU_GEN block. The Modbus job will be started, when the ACS_COM_MOD_RTU_GEN block of the server has the token, and the read/write block was started before. All these blocks must be called within the same PLC task.

The input TIMEOUT sets the timeout for one Modbus job in ms. After a timeout this server will not be reconnected for the time which can be assigned at the variable SLAVA_DATA<point>reconnectPause in seconds. So a steady delay for a disconnected server can be avoided ↯ *Chapter 1.5.6.2.2 “Special characteristics of the ACS drives base library” on page 2206*.

Diagnosis: The output ERNO, which reflects an actual error number is only valid for one cycle, output ERR is set to TRUE. To capture this error number, an external function must be programmed. However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance or visualization.

The additional diagnosis variables are:

iWriteErrCnt: number of errors in write jobs since Enable = TRUE. wLastWriteErno: holds the error number of the last executed write job.

iReadErrCnt: number of errors in read jobs since Enable = TRUE. wLastReadErno: holds the error number of the last executed read job.

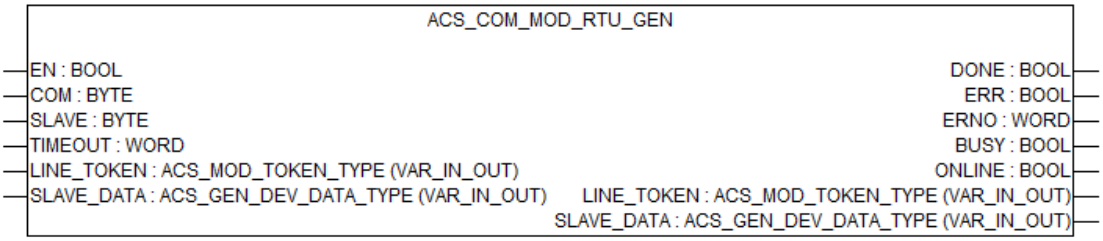
iReadWriteErrCnt: number of errors in readwrite23 with function code 23 jobs since Enable = TRUE. wLastReadWriteErno: holds the error number of the last executed readwrite23 job.

iWriteMaskErrCnt: number of errors in writemask22 with function code 22 jobs since Enable = TRUE.

wLastWriteMaskErno: holds the error number of the last executed readwrite22 job.

This function block must not be used in parallel with COM_MOD_MAST function block for the same RTU line. The function block must be used in same PLC task than other function blocks using the same variable on IN_OUT LINE_TOKEN and IN_OUT SLAVE_DATA.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN is reset to FALSE while the function block actually keeps the token of the LINE_TOKEN variable the token will be released for another device (set to 0).

COM (com)

Data type: BYTE

Interface identifier of Modbus line.

COM = 1: COM1

COM = 2: COM2

...

COM = 11: COM11 (using CM574-RS)

(COM3 = FPB, COM4 .. COM11 = COMs on CM574-RS).

Default value = 1. Minimum = 1, Maximum = 11

The COM input should not be changed while the program is running. If changed nevertheless, the new value will become effective only after a new rising edge of EN input.

SLAVE (slave)

Data type: BYTE

At input SLAVE, the address of the drive (slave) to which the connection shall be established must be specified.

Default value = 2, Minimum = 1, Maximum = 255

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed while the program is running. If changed nevertheless the new value will become effective only after an already running Modbus job is finished.

TIMEOUT (timeout)

Data type: WORD

The telegram timeout in milliseconds (ms) is specified at input TIMEOUT.

If no response is received within the time interval specified at TIMEOUT, the procedure is aborted and an error identifier is output.

If several drives are connected to the Modbus line and one drive cannot respond, the whole communication is waiting till the timeout is over and the Modbus procedure (job) is aborted. Afterwards the next drive can take the LINE_TOKEN signal. The TIMEOUT should not be set too long if more than one drive is connected, but also not too short for having a chance to respond within the TIMEOUT. A minimum of 100 is required.

Typical values should be between 300 ms and 2000 ms (range: minimum 100, maximum 65535).

Default value = 1000. Minimum = 100, Maximum = 65535.

LINE_TOKEN (line token)

Data type: ACS_MOD_TOKEN_TYPE

The combined input/output LINE_TOKEN must be connected to the one variable of ACS_MOD_TOKEN_TYPE of the related Modbus line. Each Modbus line must have its own LINE_TOKEN variable.

The LINE_TOKEN variable contains the token for the Modbus line and must be connected to all related ACS3XX_COM_MOD_RTU function blocks of this line (COMx).

Description:

The ACS_COM_MOD_RTU_ENHANCED function block writes its SLAVE number to the LINE_TOKEN variable (takes the token) when it is 0.

If the token is already occupied it writes its SLAVE number to the next-token-request.

If all Modbus jobs of a drive are terminated, the ACS_COM_MOD_RTU_ENHANCED function block resets the token on the LINE_TOKEN variable to 0 (release the token) if the next-token-request is set (not 0).

If an ACS_COM_MOD_RTU_ENHANCED takes the token it releases the next-token-request at the same time.

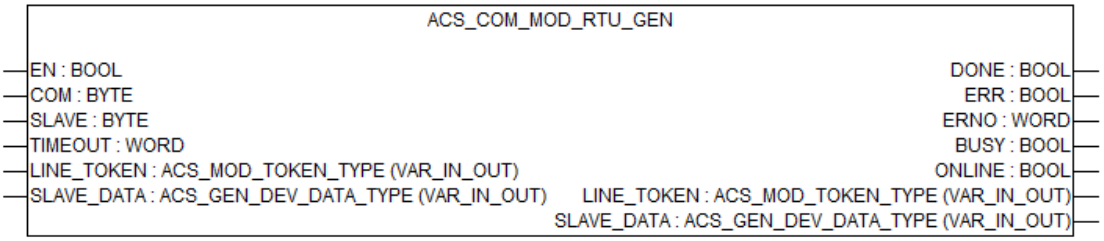
SLAVE_DATA (slave data)

Data type: ACS_DRIVE_DATA_TYPE

The combined input/output SLAVE_DATA must be connected to the variable of type ACS_GEN_DEV_DATA_TYPE of the related slave device. Each device must have its own SLAVE_DATA variable.

The SLAVE_DATA variable transfers the Modbus data from the ACS_COM_MOD_RTU_GEN_READ_N_PRM or ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function blocks to the ACS_COM_MOD_RTU_GEN function block. It contains also some information about the online state of the device and must be connected to all related function blocks of this device.

Output description



DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy) Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

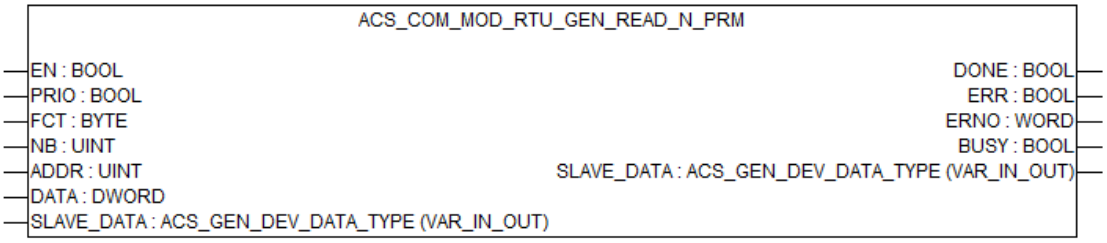
Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

Function call in ST

```
ACSComModRTUGen     (EN                                  :=
xACSComModRTUGen_EN,

 COM                                  := byACSComModRTUGen_COM,

 SLAVE                           := byACSComModRTUGen_SLAVE,

 TIMEOUT                 := wACSComModRTUGen_TIMEOUT,
```

```
 LINE_TOKEN    := tsLineToken,

 SLAVE_DATA    := tsSlaveData);

xACSComModRTUGen_DONE            := ACSComModRTUGen.DONE;
xACSComModRTUGen_ERR                := ACSComModRTUGen.ERR;
wACSComModRTUGen_ERNO            := ACSComModRTUGen.ERNO;
xACSComModRTUGen_BUSY            := ACSComModRTUGen.BUSY;
xACSComModRTUGen_ONLINE    := ACSComModRTUGen.ONLINE;
```

ACS_COM_MOD_RTU_GEN_READ_N_PRM read N parameters from a generic Modbus RTU device



Function block information

Available from runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

Function block ACS_COM_MOD_RTU_GEN_READ_N_PRM reads n data of the slave device. The number of data to be read is specified at the input NB. The first address is specified at the input ADDR. The values of the data is stored in the PLC memory area, defined at the input DATA. The values in the PLC memory area are updated when the read job was performed without error. This is indicated by DONE = TRUE and ERR = FALSE.

As long as the EN = TRUE a new read job is requested each time the further read job is terminated.

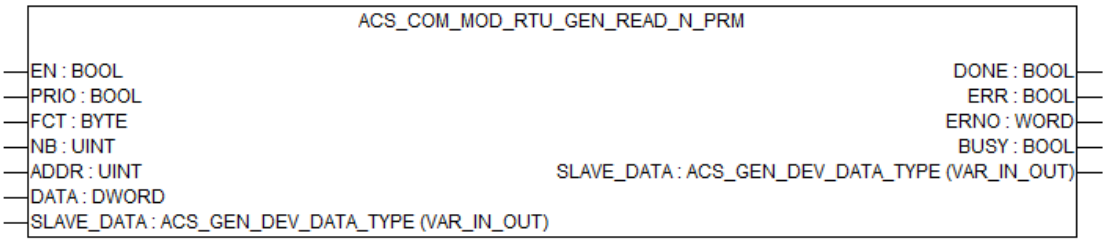
The Modbus job is started from the ACS_COM_MOD_RTU_GEN function block which is connected to the same SLAVE_DATA variable.

The slave address of the device from which the data is read is specified at this ACS_COM_MOD_RTU_GEN block.

If the input values are valid, a request to perform a Modbus RTU job is send to the ACS_COM_MOD_RTU_GEN block via the SLAVE_DATA variable.

The ACS_COM_MOD_RTU_GEN_READ_N_PRM function block does only change the values at the DATA input if the Modbus job was excecuted without errors. If the block is not enabled or the connection to the ACS_COM_MOD_RTU_GEN block is not ok these values are not updated and keep their previous values. This is also the case for the internal VALUE array. However the ACS_MOD_READ_N_PRM for drives does reset its values attached to DATA input and the VALUE array to zero in case of no connection to the related ACS_COM_MOD function block or if the block is not enabled.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

If a TRUE state is applied to the input EN, a request to perform a Modbus read job is set to the SLAVE_DATA variable.

All further inputs are read in.

If the state of EN stays TRUE a new read job is requested each time the previous job is terminated, indicated by the DONE = TRUE flag.

PRIO (priority)

Data type: BOOL

Input PRIO is reserved for future usage. It can be left open.

FCT (number of variables)

Data type: BYTE

The function code of the request telegram is specified at input FCT.

01 or 02	read n bits
03 or 04	read n words
05	write one bit
06	write one word
15	write n bits
16	write n words

NB (number of variables)

Data type: UINT

At input NB, the number of data to be read is specified.

The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB.

Default: 1. Minimum 1, Maximum 96.

ADDR (address)

Data type: UINT

The operand/register address in the slave from which data should be read is specified at input ADDR.

The access to operands of AC500 devices in Modbus slave mode is defined via the Modbus cross-reference list. Only operands that are listed in the cross-reference list may be used (see Communication with Modbus RTU).

Only operands that are listed in the Modbus address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

DATA (data)

Data type: DWORD

At input DATA, the address of the first operand in the PLC is specified to which the data read by the slave should be stored.

For this purpose it is necessary that all PLC variables, which are written to the drive, have consecutive addresses. This can be obtained by declaration of each variable within the %MW0.xxx area or declaration of an array containing all variables.

Declaration of each variable has the advantage, that the types (integer or word) can be selected individually.

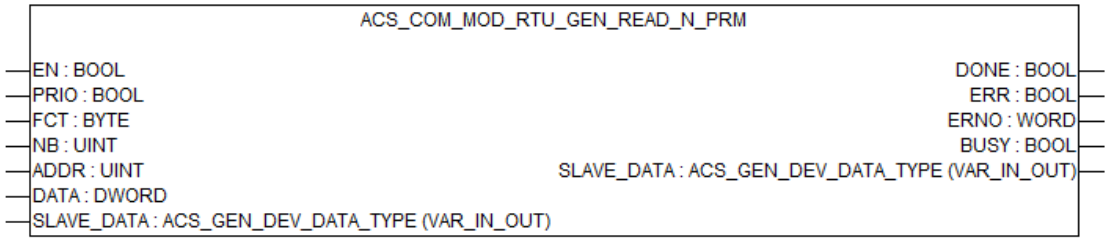
SLAVE_DATA (slave data)

Data type: ACS_DRIVE_DATA_TYPE

The combined input/output SLAVE_DATA must be connected to the variable of type ACS_GEN_DEV_DATA_TYPE of the related slave device. Each device must have its own SLAVE_DATA variable.

The SLAVE_DATA variable transfers the Modbus data from the ACS_COM_MOD_RTU_GEN_READ_N_PRM or ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function blocks to the ACS_COM_MOD_RTU_GEN function block. It contains also some information about the online state of the device and must be connected to all related function blocks of this device.

Output description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

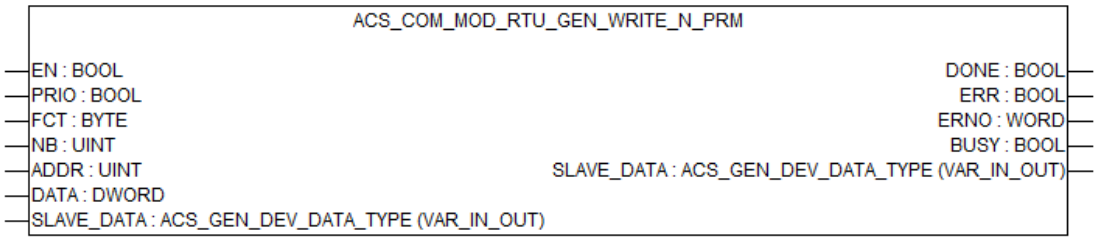
Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

Function call in ST

```
dwAcsComModRTUGenReadNPrm_DATA    :=
ADR(aiAcsComModRTUGenReadNPrm_VALUE[1]);

ACSComModRTUGenReadNPrm    (EN                                    :=
xACSComModRTUGenReadNPrm_EN,

                           FCT                                 :=
byACSComModRTUGenReadNPrm_FCT,

                           NB                                     :=
uiACSComModRTUGenReadNPrm_NB,

                           ADDR                             :=
uiACSComModRTUGenReadNPrm_ADDR,

                           DATA                             :=
dwAcsComModRTUGenReadNPrm_DATA,

                           SLAVE_DATA    := tsSlaveData);

xACSComModRTUGenReadNPrm_DONE    := ACSComModRTUGenReadNPrm.DONE;
xACSComModRTUGenReadNPrm_ERR        := ACSComModRTUGenReadNPrm.ERR;
wACSComModRTUGenReadNPrm_ERNO    := ACSComModRTUGenReadNPrm.ERNO;
xACSComModRTUGenReadNPrm_BUSY    := ACSComModRTUGenReadNPrm.BUSY;
```

ACS_COM_MOD_RTU_GEN_WRITE_N_PRM write N parameters to a generic Modbus RTU device



Function block information

Available from runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib
Function block	with historical values

Function block ACS_COM_MOD_RTU_GEN_WRITE_N_PRM writes n data to the slave device. The number of data to be written is specified at the input NB. The first address is specified at the input ADDR. The values of the data that should be written must be stored in the PLC memory area, defined at the input DATA. A successful writing is indicated by DONE = TRUE and ERR = FALSE.

To initiate a write job the EN input must be given a rising edge. (FALSE -> TRUE).

After termination of this job, even if it was not successful, a next writing can once again only be started with a rising edge at EN input.

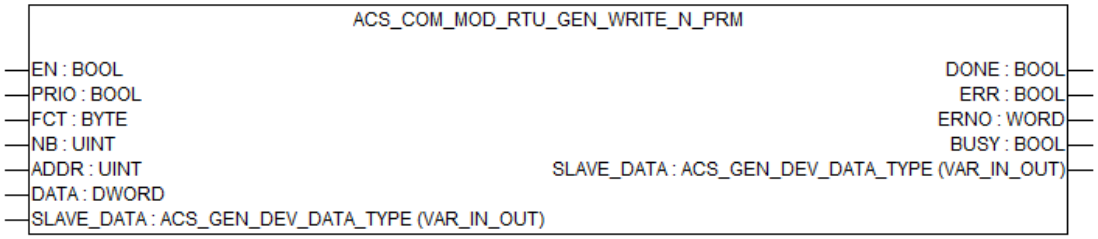
The Modbus job is started from the ACS_COM_MOD_RTU_GEN function block which is connected to the same SLAVE_DATA variable.

The slave address of the device to which the data is written is specified at this ACS_COM_MOD_RTU_GEN block.

If the input values are valid, a request to perform a Modbus RTU job is send to the ACS_COM_MOD_RTU_GEN block via the SLAVE_DATA variable.

If at least 1 input is invalid, no job is generated and the error is displayed at the outputs ERR and ERNO instead.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

If a FALSE -> TRUE edge is applied to input EN, all further inputs are read in (edge sensitive).

If the input values are valid, a request telegram is sent to the communication block via the DRIVE_DATA variable.

If at least 1 input is invalid, no telegram is generated and the error is displayed at the outputs ERR and ERNO instead.

The inputs are read in until the job is processed. This can be seen at the output BUSY.

PRIO (priority)

Data type: BOOL

Input PRIO is reserved for future usage. It can be left open.

FCT (number of variables)

Data type: BYTE

The function code of the request telegram is specified at input FCT.

01 or 02	read n bits
03 or 04	read n words
05	write one bit
06	write one word
15	write n bits
16	write n words

NB (number of variables)

Data type: UINT

At input NB, the number of data to be read is specified.

The unit of NB depends on the selected function. For bit accesses the number of bits, for word and double word accesses the number of words is specified at NB.

Default: 1. Minimum 1, Maximum 96.

ADDR (address)

Data type: UINT

The operand/register address in the slave from which data should be read is specified at input ADDR.

The access to operands of AC500 devices in Modbus slave mode is defined via the Modbus cross-reference list. Only operands that are listed in the cross-reference list may be used (see Communication with Modbus RTU).

Only operands that are listed in the Modbus address list may be used. When accessing other devices, ADDR is freely selectable. The valid ranges have to be gathered from the corresponding device description.

DATA (data)

Data type: DWORD

At input DATA, the address of the first operand in the PLC is specified to which the data read by the slave should be stored.

For this purpose it is necessary that all PLC variables, which are written to the drive, have consecutive addresses. This can be obtained by declaration of each variable within the %MW0.xxx area or declaration of an array containing all variables.

Declaration of each variable has the advantage, that the types (integer or word) can be selected individually.

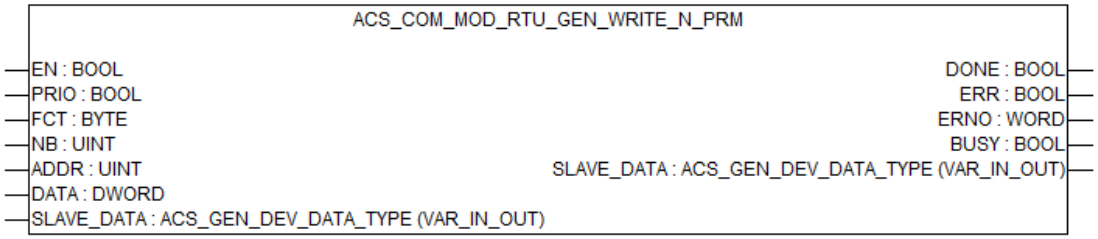
SLAVE_DATA (slave data)

Data type: ACS_DRIVE_DATA_TYPE

The combined input/output SLAVE_DATA must be connected to the variable of type ACS_GEN_DEV_DATA_TYPE of the related slave device. Each device must have its own SLAVE_DATA variable.

The SLAVE_DATA variable transfers the Modbus data from the ACS_COM_MOD_RTU_GEN_READ_N_PRM or ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function blocks to the ACS_COM_MOD_RTU_GEN function block. It contains also some information about the online state of the device and must be connected to all related function blocks of this device.

Output description



> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

SLAVE_DATA (slave data)

Data type: ACS_DRIVE_DATA_TYPE

The combined input/output SLAVE_DATA must be connected to the variable of type ACS_GEN_DEV_DATA_TYPE of the related slave device. Each device must have its own SLAVE_DATA variable.

The SLAVE_DATA variable transfers the Modbus data from the ACS_COM_MOD_RTU_GEN_READ_N_PRM or ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function blocks to the ACS_COM_MOD_RTU_GEN function block. It contains also some information about the online state of the device and must be connected to all related function blocks of this device.

DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter ↬ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

Function call in ST

```
dwAcsComModRTUGenWriteNPrm_DATA    :=
ADR(aiAcsComModRTUGenWriteNPrm_VALUE[1]);

ACSComModRTUGenWriteNPrm    (EN                                    :=
xACSComModRTUGenWriteNPrm_EN,

                             FCT                                   :=
byACSComModRTUGenWriteNPrm_FCT,

                             NB                                      :=
uiACSComModRtuGenWriteNPrm_NB,

                             ADDR                                  :=
uiACSComModRtuGenWriteNPrm_ADDR,

                             DATA                                  :=
dwAcsComModGenWritePrm_DATA,

                             SLAVE_DATA    := tsSlaveData);

xACSComModRTUGenWriteNPrm_DONE    := ACSComModRTUGenWriteNPrm.DONE;
xACSComModRTUGenWriteNPrm_ERR        := ACSComModRTUGenWriteNPrm.ERR;
wACSComModRTUGenWriteNPrm_ERNO    := ACSComModRTUGenWriteNPrm.ERNO;
xACSComModRTUGenWriteNPrm_BUSY    := ACSComModRTUGenWriteNPrm.BUSY;
```

1.5.6.3.6 Structures

ACS_GEN_DEV_DATA_TYPE structure to exchange data between function blocks for 1 generic device

Structure ACS_GEN_DEV_DATA_TYPE is used for the SLAVE_DATA variable to exchange the data for one generic Modbus slave device.

Structure data

Available as of runtime system:	V1.3.2	Remark:
Included in library:	ACSDrivesCom-ModRTU_AC500_V20.lib	ACSDrives-Base_AC500_V20.lib is needed in addition.

Structure

Visible variable	Type	Default value	User access *)	Description
online	BOOL	FALSE	R	Connection established – set in Modbus communication function block after successfull reading and writing one Modbus job.

name	STRING(20)	Default Drive Name	R/W	Name for slave device which can be set by user directly to SLAVE_DATA variable – as information or use in visualizations.
adapterType	INT	1	R	Will be set to 100 in ACS_COM_MOD_RTU_GEN for internal checks.
*) R = read only, R/W = read and write access				

Description

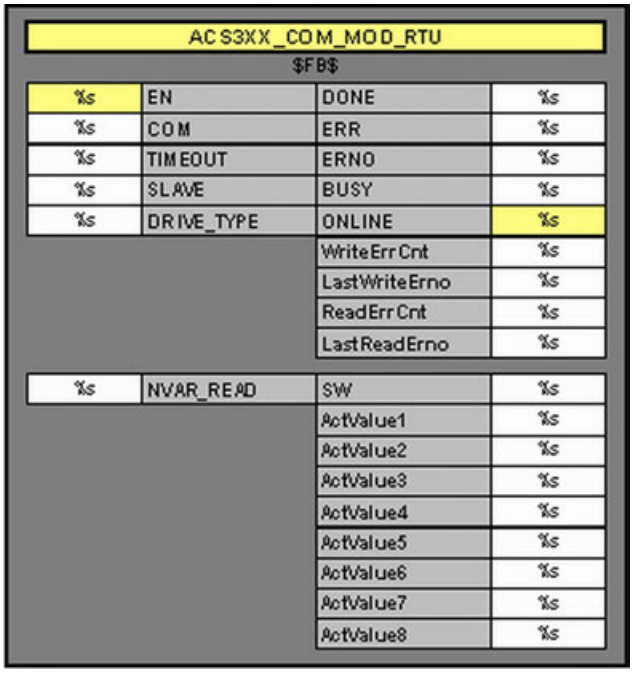
Structure ACS_GEN_DEV_DATA_TYPE is used for the SLAVE_DATA variable which must be connected to all function blocks related to the same device.

Besides the variable "name" all variables should not be written by the user directly. They are read and written within the function blocks.

The ACS_GEN_DEV_DATA_TYPE contains some more internal, invisible variables which are used for interlocks and data transfer and not meant for user access.

1.5.6.3.7 Visualizations

ACS3XX_COM_MOD_RTU_VISU_PH faceplate for the function block ACS3XX_COM_MOD_RTU



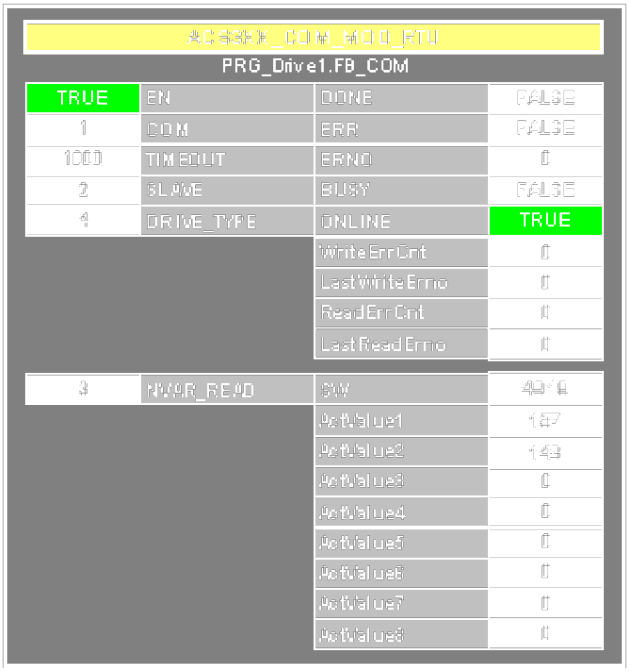


Fig. 160: Visualization offline / online mode

Visualization element ACS3XX_COM_MOD_RTU_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS3XX_COM_MOD_RTU function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

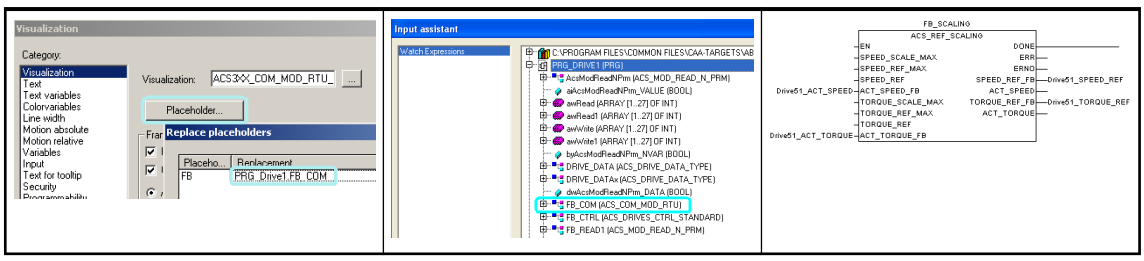
Visualization information

Available in runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20_App.lib

Visualization description

Visualization element ACS3XX_COM_MOD_RTU_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS3XX_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.1 “ACS3XX_COM_MOD_RTU communication for ACS3XXwrite one bit/ACX550 drives via Modbus RTU” on page 2293* function block which instance was used to replace the placeholder FB.

All inputs of that ACS3XX_COM_MOD_RTU function block, which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.



Parameters

Access R/W

EN Access via: Toggle

Description: EN input

COM Access via: Numpad 1..10

Description: COM. After a change the program must be reset (create boot project, reset and start).

TIMEOUT Access via: Numpad min. 1

Description: TIMEOUT input

SLAVE Way of Access: Numpad 1..31

Description: SLAVE input

DRIVE_TYPE Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

NVAR_READ Access via: Numpad 0 .. 12

Description: NVAR_READ input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_COM

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of COM_MOD_MAST function block.

SW_BIT13 Description: Status Word of drive, READ_VALUE[1] output

ACT_VALUE1 Description:Actual Value 1 mapped in Par. 53.10 - READ_VALUE[2] output

ACT_VALUE2 Description:Actual Value 2 mapped in Par. 53.11 - READ_VALUE[3] output

ACT_VALUE3 Description:Actual Value 3 mapped in Par. 53.12 - READ_VALUE[4] output

ACT_VALUE4 Description:Actual Value 4 mapped in Par. 53.13 - READ_VALUE[5] output

ACT_VALUE5 Description:Actual Value 5 mapped in Par. 53.14 - READ_VALUE[6] output

ACT_VALUE6 Description:Actual Value 6 mapped in Par. 53.15 - READ_VALUE[7] output

ACT_VALUE7 Description:Actual Value 7 mapped in Par. 53.16 - READ_VALUE[8] output

ACT_VALUE8 Description:Actual Value 8 mapped in Par. 53.17 - READ_VALUE[9] output

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_RTU_VISU_PH faceplate for the function block ACS_COM_MOD_RTU

ACS_COM_MOD_RTU			
FB			
%s	EN	DONE	%s
%s	COM	ERR	%s
%s	SLAVE	ERNO	%s
%s	TIMEOUT	BUSY	%s
%s	DRIVE_TYPE	ONLINE	%s
%s	NVAR_READ	ReconPause	%s
%s	ReconPause	WriteErrCnt	%s
		LastWriteErr	%s
		ReadErrCnt	%s
		LastReadErr	%s
%s	MCW	MSW	%s
%s	RefValue1	ActValue1	%s
%s	RefValue2	ActValue2	%s
READ_VALUES - for embedded modbus			
%s	DATA_IN1	DATA_IN13	%s
%s	DATA_IN2	DATA_IN14	%s
%s	DATA_IN3	DATA_IN15	%s
%s	DATA_IN4	DATA_IN16	%s
%s	DATA_IN5	DATA_IN17	%s
%s	DATA_IN6	DATA_IN18	%s
%s	DATA_IN7	DATA_IN19	%s
%s	DATA_IN8	DATA_IN20	%s
%s	DATA_IN9	DATA_IN21	%s
%s	DATA_IN10	DATA_IN22	%s
%s	DATA_IN11	DATA_IN23	%s
%s	DATA_IN12	DATA_IN24	%s

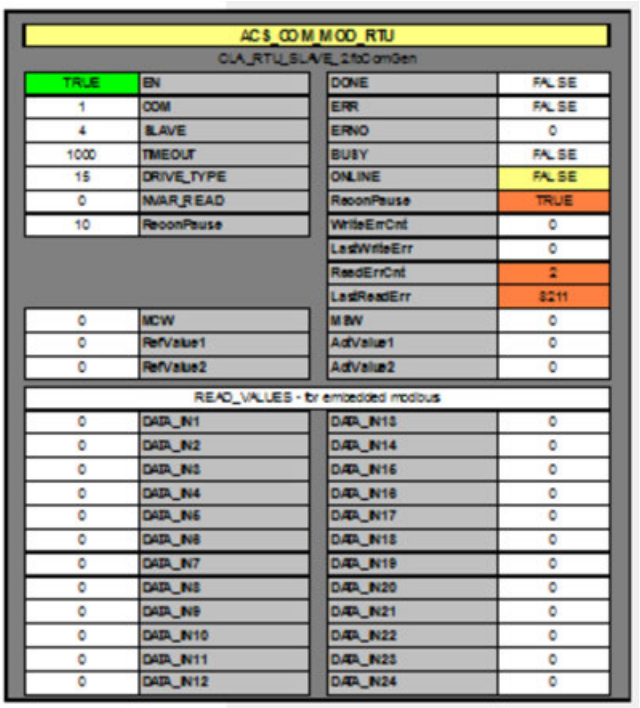


Fig. 161: Visualization offline / online mode

Visualization element ACS_COM_MOD_RTU_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU ↬ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301* function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Visualization data

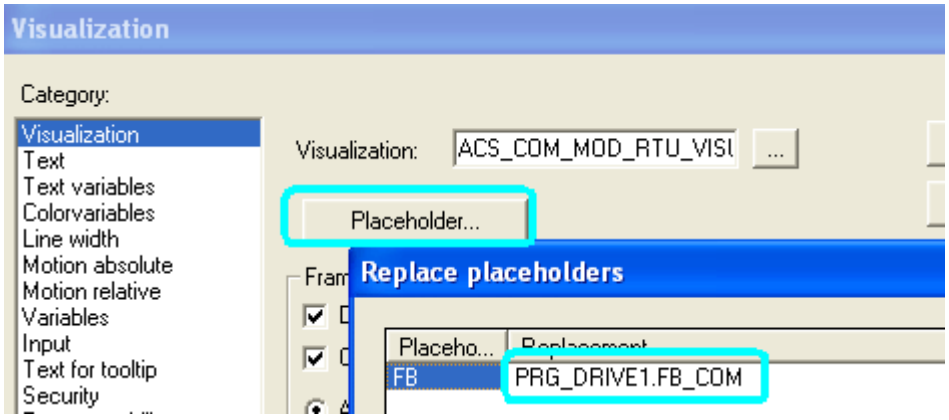
Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib

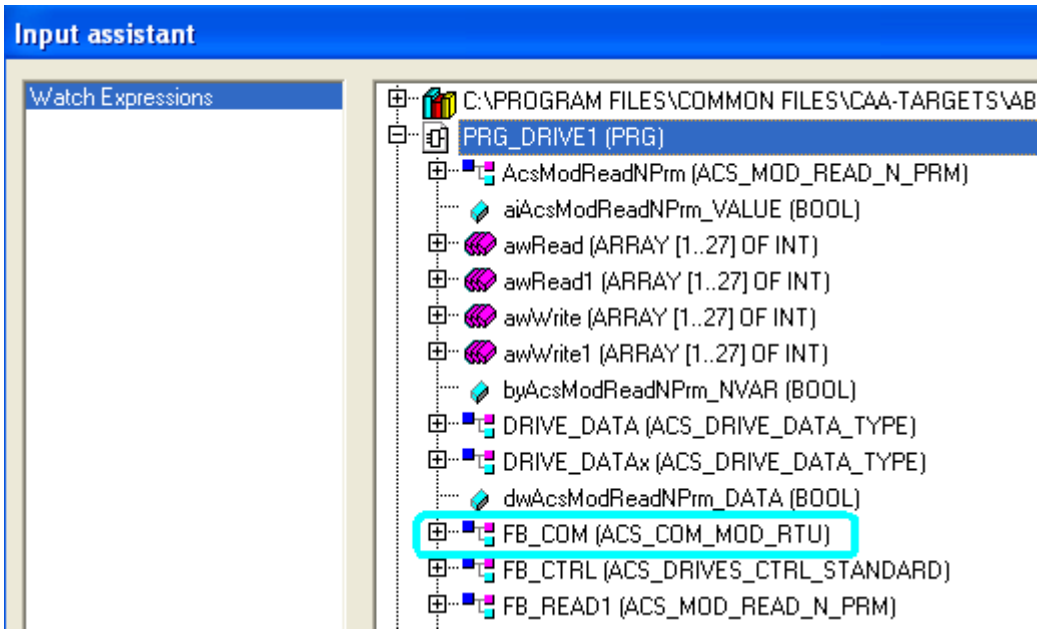
Description

Visualization element ACS_COM_MOD_RTU_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU function block, which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_RTU function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.

The DRIVE_DATA variable must be connected to the function block.





Parameters

Access R/W

EN Access via: Toggle

Description: EN input

COM Access via: Numpad 1..10

Description: COM. After a change the program must be reset (create boot project, reset and start).

TIMEOUT Access via: Numpad min. 1

Description: TIMEOUT input

SLAVE Way of Access: Numpad 1..31

Description: SLAVE input

DRIVE_TYPE Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

NVAR_READ Access via: Numpad 0 .. 12

Description: NVAR_READ input

ReconPause Access via: Numpad min. 0

Description: Reconnect Pause value in seconds.

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_COM

Access R

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

ReconPause Timeout or other SLAVE error detected and reconnection pause is active.

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of COM_MOD_MAST function block.

MSW Description: Status Word of drive, READ_VALUE[1] output.

ACT_VALUE1 Description: ACT_VALUE1 output

ActValue2 Description: Actual Value 2 output; Actual Value 2 of drive

DATA_IN1 Description: READ_VALUE[1] output

DATA_IN2 Description: READ_VALUE[2] output

DATA_IN3 Description: READ_VALUE[3] output

DATA_IN4 Description: READ_VALUE[4] output

DATA_IN5 Description: READ_VALUE[5] output

DATA_IN6 Description: READ_VALUE[6] output

DATA_IN7 Description: READ_VALUE[7] output

DATA_IN8 Description: READ_VALUE[8] output

DATA_IN9 Description: READ_VALUE[9] output

DATA_IN10 Description: READ_VALUE[10] output

DATA_IN11 Description: READ_VALUE[11] output

DATA_IN12 Description: READ_VALUE[12] output

DATA_IN13 Description: READ_VALUE[13] output

DATA_IN14 Description: READ_VALUE[14] output

DATA_IN15 Description: READ_VALUE[15] output

DATA_IN16 Description: READ_VALUE[16] output

DATA_IN17 Description: READ_VALUE[17] output

DATA_IN18 Description: READ_VALUE[18] output

DATA_IN19 Description: READ_VALUE[19] output

DATA_IN20 Description: READ_VALUE[20] output

DATA_IN21 Description: READ_VALUE[21] output

DATA_IN22 Description: READ_VALUE[22] output

DATA_IN23 Description: READ_VALUE[23] output

DATA_IN24 Description: READ_VALUE[24] output

Colors

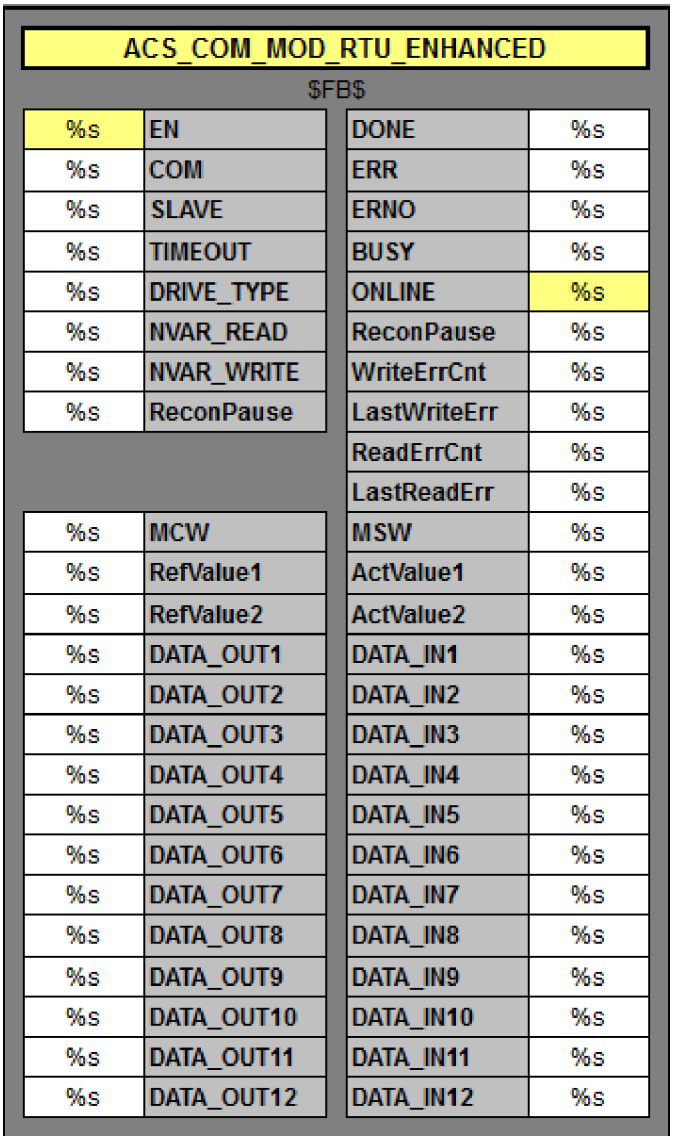
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_RTU_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_ENHANCED



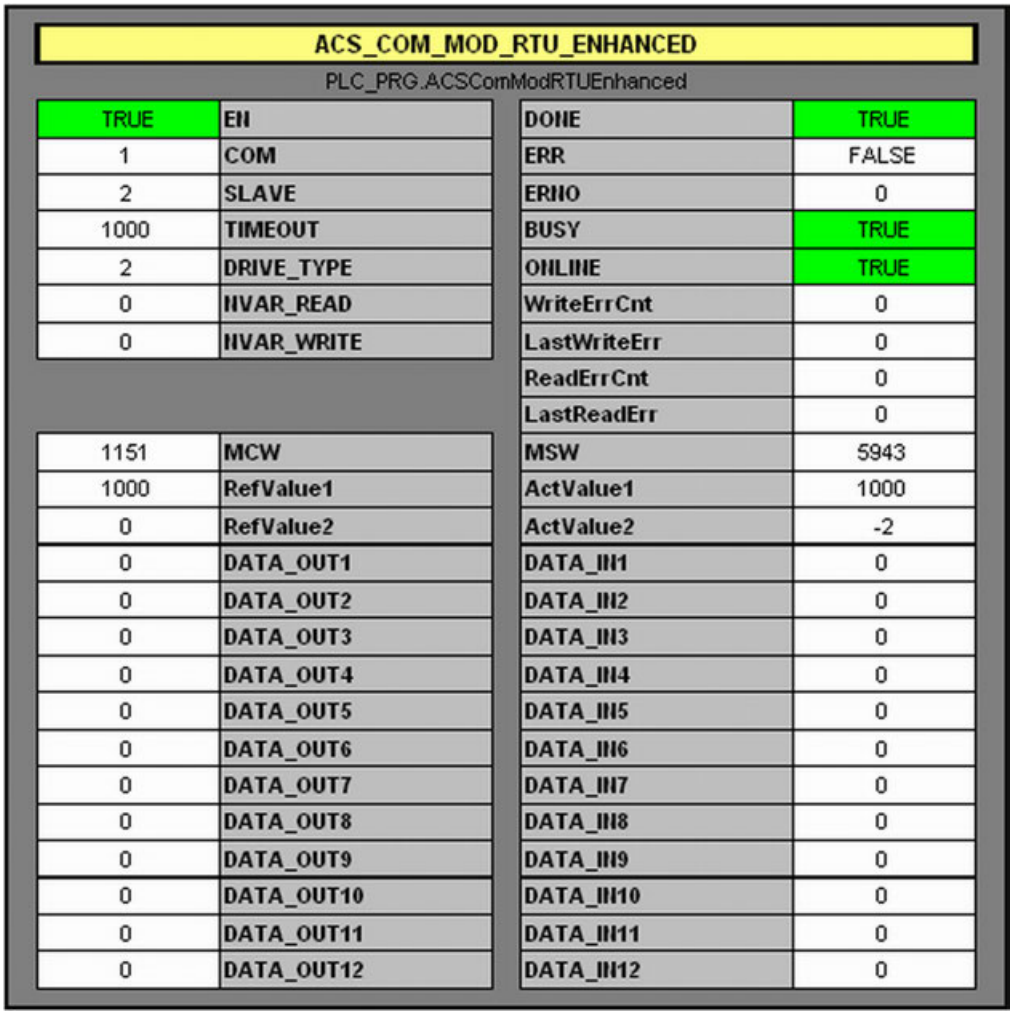


Fig. 162: Visualization offline / online mode

Visualization element ACS_COM_MOD_RTU_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_ENHANCED function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

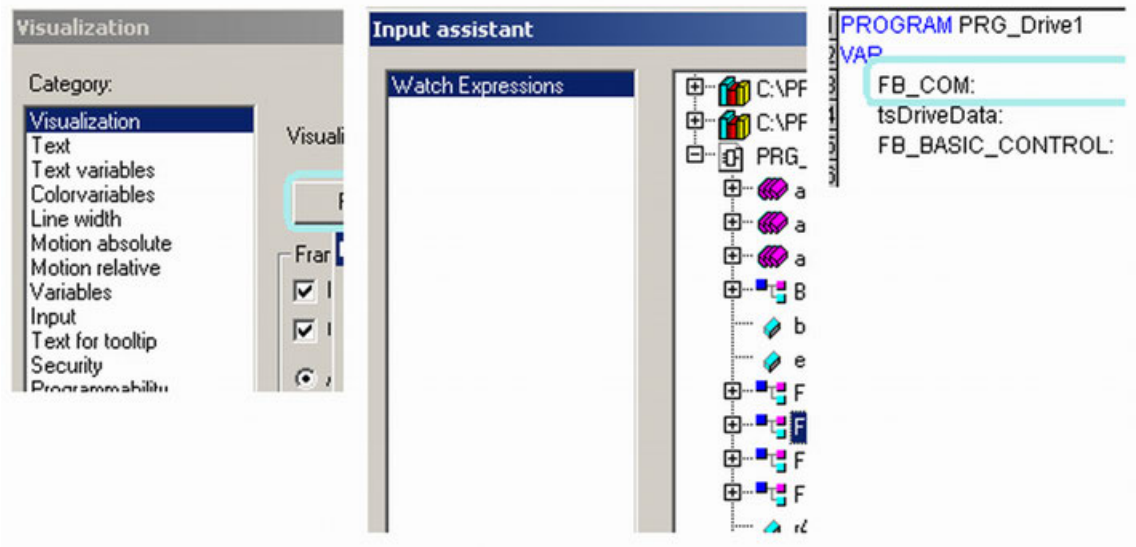
Visualization data

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib

Visualization description

Visualization element ACS_COM_MOD_RTU_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_ENHANCED ↳ *Chapter 1.5.6.3.5.3 “ACS_COM_MOD_RTU_ENHANCED communication for ACS drives via Modbus RTU using ABB drives profile enhanced” on page 2312* function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_RTU_ENHANCED function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.



Parameters

ACCESS R/W

EN
Access via: Toggle
Description: EN input

COM
Access via: Numpad 1..10
Description: COM. After a change the program must be reset (create boot project, reset and start).

TIMEOUT
Access via: Numpad min. 1
Description: TIMEOUT input

SLAVE
Access via: Numpad 1..247
Description: SLAVE input

DRIVE_TYPE
Access via: Numpad 1 ..17
Description: DRIVE_TYPE input

NVAR_READ
Access via: Numpad 0 .. 12
Description: NVAR_READ input

NVAR_WRITE
Access via: Numpad 0 .. 12
Description: NVAR_WRITE input

ReconPause
Access via: Numpad min. 0
Description: Reconnect Pause value in seconds.

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_COM

DATA_OUT1 Description: WRITE_VALUE[1] input

DATA_OUT2 Description: WRITE_VALUE[2] input

DATA_OUT3 Description: WRITE_VALUE[3] input

DATA_OUT4 Description: WRITE_VALUE[4] input

DATA_OUT5 Description: WRITE_VALUE[5] input

DATA_OUT6 Description: WRITE_VALUE[6] input

DATA_OUT7 Description: WRITE_VALUE[7] input

DATA_OUT8 Description: WRITE_VALUE[8] input

DATA_OUT9 Description: WRITE_VALUE[9] input

DATA_OUT10 Description: WRITE_VALUE[10] input

DATA_OUT11 Description: WRITE_VALUE[11] input

DATA_OUT12 Description: WRITE_VALUE[12] input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_COM

ACCESS R

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

ReconPause Timeout or other SLAVE error detected and reconnection pause is active.

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of COM_MOD_MAST function block.

MSW Description: Status Word of drive, READ_VALUE[1] output.

ACT_VALUE1 Description: ACT_VALUE1 output

ACT_VALUE2 Description: ACT_VALUE2 output

DATA_IN1 Description: READ_VALUE[1] output

DATA_IN2 Description: READ_VALUE[2] output

DATA_IN3 Description: READ_VALUE[3] output

DATA_IN4 Description: READ_VALUE[4] output

DATA_IN5 Description: READ_VALUE[5] output

DATA_IN6 Description: READ_VALUE[6] output

DATA_IN7 Description: READ_VALUE[7] output

DATA_IN8 Description: READ_VALUE[8] output

DATA_IN9 Description: READ_VALUE[9] output

DATA_IN10 Description: READ_VALUE[10] output

ACS_COM_MOD_RTU_GEN_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN

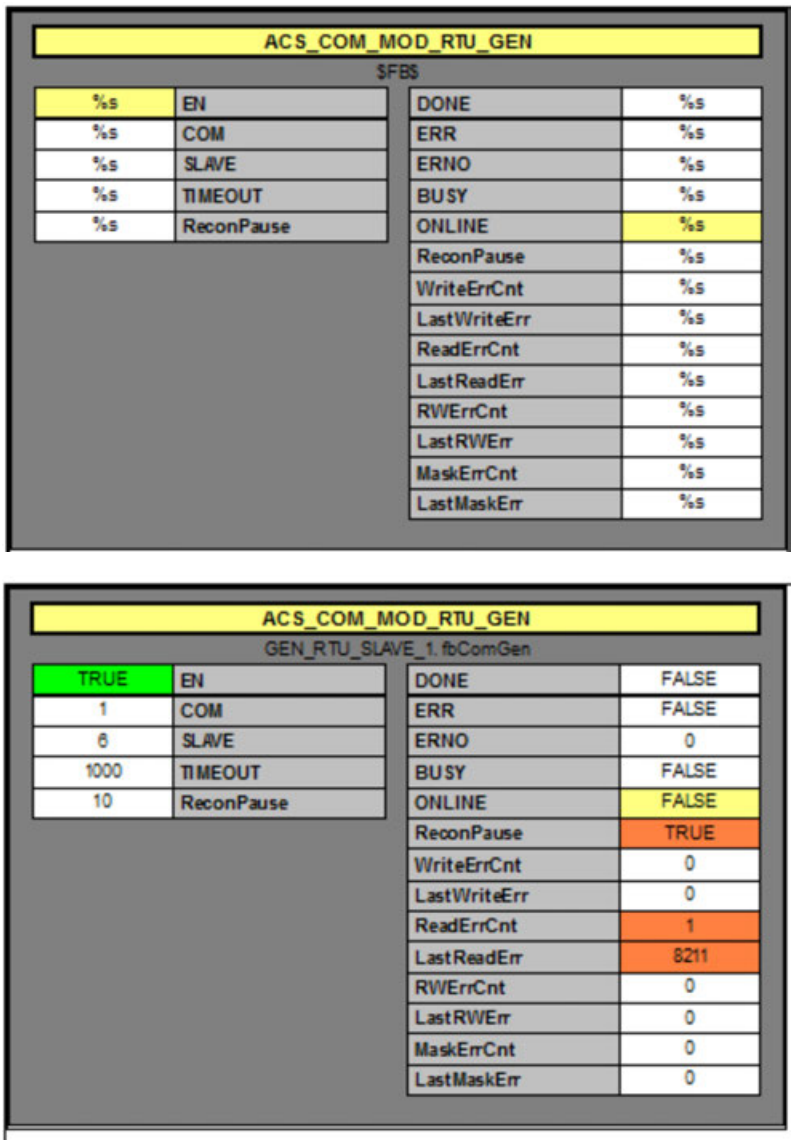


Fig. 163: Visualization offline / online mode

Visualization element ACS_COM_MOD_RTU_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN ↳ *Chapter 1.5.6.3.5.4 “ACS_COM_MOD_RTU_GEN communication for generic devices via Modbus RTU ” on page 2322* function block which instance is used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Visualization data

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib

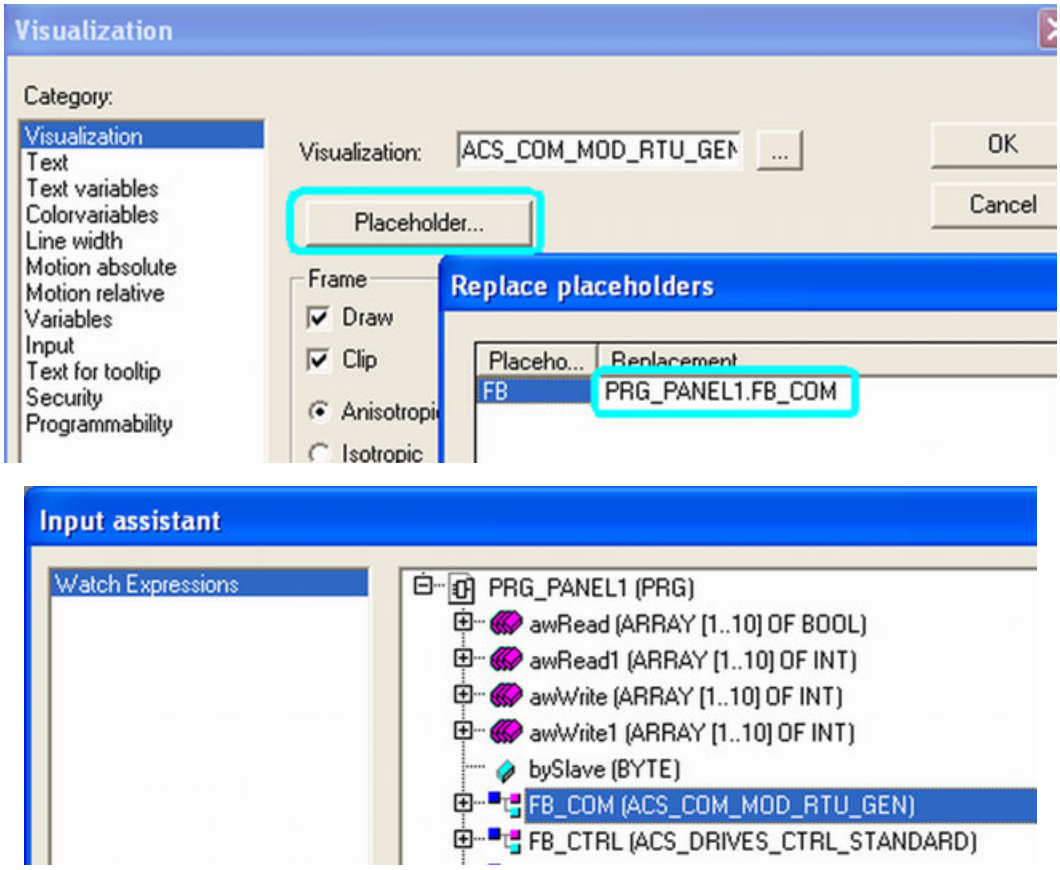
Description

Visualization element ACS_COM_MOD_RTU_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN function block, which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_RTU_GEN function block, which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.

Even if most of the inputs should not be change while the program is running (COM, SLAVE and TIMEOUT), the visualization is nevertheless helpful to check the ONLINE state and additional internal variables about error count and last error numbers.

The SLAVE_DATA variable must be connected to the function block.



Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

COM

Access via: Numpad 1..10

Description: COM. After a change the program must be reset (create boot project, reset and start).

SLAVE

Way of Access: Numpad 1..31

Description: SLAVE input

TIMEOUT

Access via: Numpad min. 1

Description: TIMEOUT input

ReconPause Access via: Numpad min. 0

Description: Reconnect Pause value in seconds.

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_COM

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Panel1.FB_Write

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

ONLINE Description: ONLINE output

ReconPause Timeout or other SLAVE error detected and reconnection pause is active.

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErr Description: Error number of the last read job. See error messages of COM_MOD_MAST function block.

RWErrCnt Description: error count of read/Write (FCT-23) errors.

LastRWErr Description: last error code of read/Write (FCT-23) error.

MaskErrCnt Description: error count of masked (FCT-22) errors.

LastMaskErr Description: Last error code for the masked (FCT-22) error.

Colors

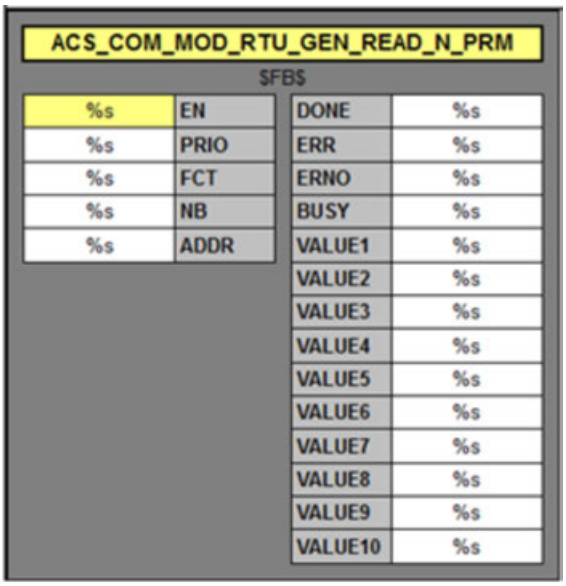
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_RTU_GEN_READ_N_PRM_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN_READ_N_PRM



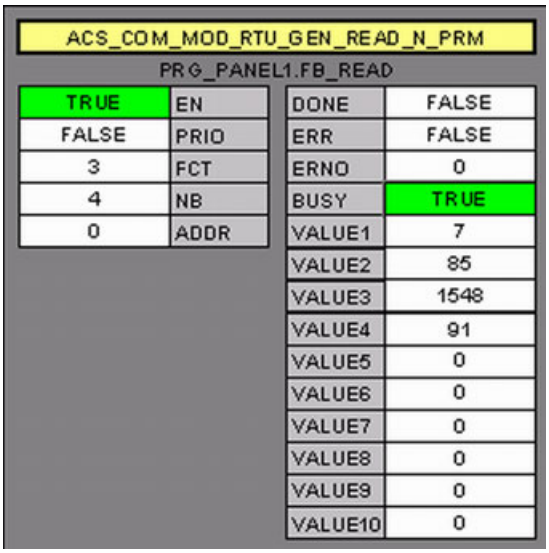


Fig. 164: Visualization offline / online mode

Visualization element ACS_COM_MOD_RTU_GEN_READ_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN_READ_N_PRM ↳ *Chapter 1.5.6.3.5.5 "ACS_COM_MOD_RTU_GEN_READ_N_PRM read N parameters from a generic Modbus RTU device " on page 2327* function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Visualization data

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib

Description

Visualization element ACS_COM_MOD_RTU_GEN_READ_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN_READ_N_PRM function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_RTU_GEN_READ_N_PRM function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

The values to where the read data is stored to in the PLC can not be displayed with this faceplate, because they have to be accessed by the ADR operator outside the function block ACS_COM_MOD_RTU_GEN_READ_N_PRM.

Nevertheless the first 10 values are also copied to an internal array of 15 elements. The first 10 elements are displayed in this visualization.

The input DATA is not shown as it has to be connected to the address of a variable and therefore can not be set in the visualization.

SLAVE_DATA input must be connected in the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

PRIO (priority) Data type: BOOL
Input PRIO is reserved for future usage. It can be left open.

FCT Access via: Numpad 5..16
Description: FCT input

NB Access via: Numpad 1..1968
Description: NB input

ADDR Access via: Numpad min. 0
Description: ADR input

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

VALUE1 Description: Value of 1st read parameter, written if read job was successfully.

VALUE2 Description: Value of 2nd read parameter, written if read job was successfully.

VALUE3 Description: Value of 3rd read parameter, written if read job was successfully.

VALUE4 Description: Value of 4th read parameter, written if read job was successfully.

VALUE5 Description: Value of 5th read parameter, written if read job was successfully.

VALUE6 Description: Value of 6th read parameter, written if read job was successfully.

VALUE7 Description: Value of 7th read parameter, written if read job was successfully.

VALUE8 Description: Value of 8th read parameter, written if read job was successfully

VALUE9 Description: Value of 9th read parameter, written if read job was successfully.

VALUE10 Description: Value of 10th read parameter, written if read job was successfully.

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Panel1.FB_Read

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_RTU_GEN_WRITE_N_PRM_VISU_PH faceplate for the function block ACS_COM_MOD_RTU_GEN_WRITE_N_PRM

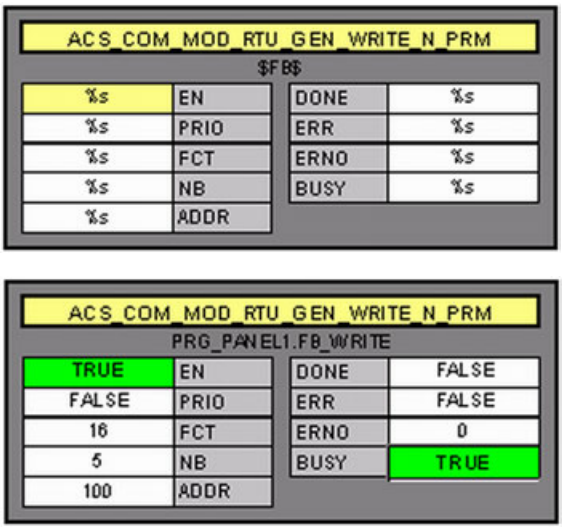


Fig. 165: Visualization offline / online mode

Visualization element ACS_COM_MOD_RTU_GEN_WRITE_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN_WRITE_N_PRM *Chapter 1.5.6.3.5.6 “ACS_COM_MOD_RTU_GEN_WRITE_N_PRM write N parameters to a generic Modbus RTU device ” on page 2330* function block which instance was used to replace the placeholder FB.

The visualization can also be used to control the function block by those inputs which are not connected inside the program.

Visualization data

Available as of runtime system:	V1.3.2
Included in library:	ACSDrivesComModRTU_AC500_V20.lib

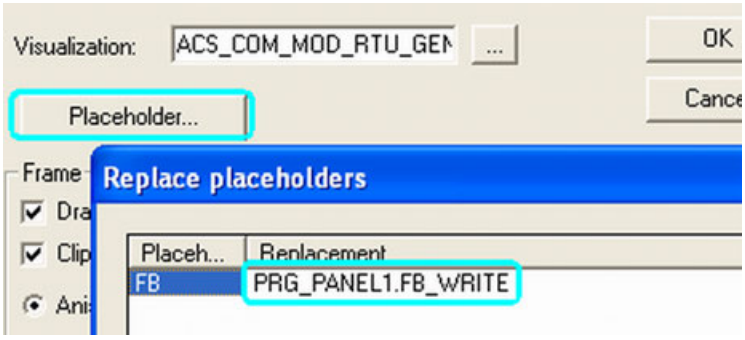
Description

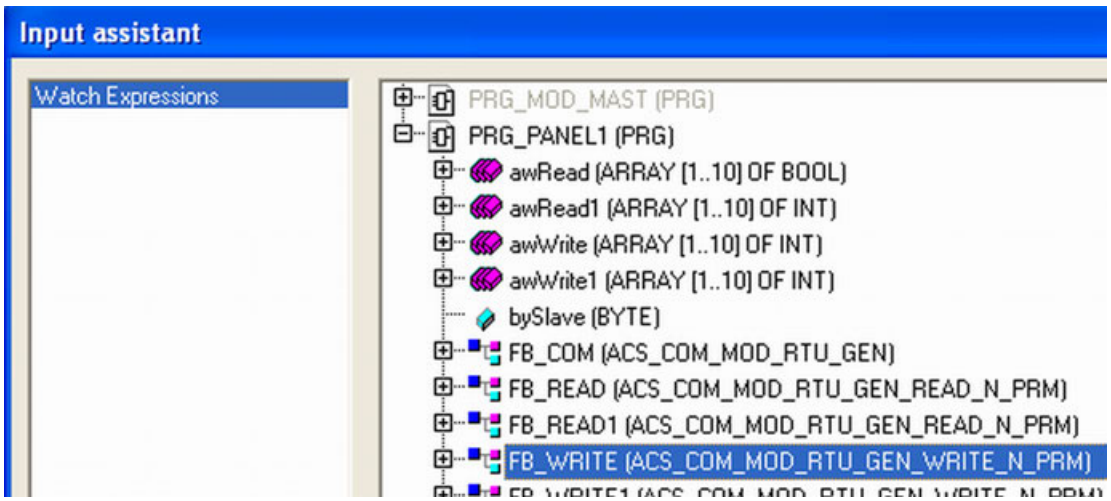
Visualization element ACS_COM_MOD_RTU_GEN_WRITE_N_PRM_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_RTU_GEN_WRITE_N_PRM function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.

The SLAVE_DATA variable must be connected to the function block.

The values to be written can not be displayed with this faceplate, because they have to be accessed by the ADR operator outside the function block ACS_COM_MOD_RTU_GEN_WRITE_N_PRM.





Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

PRIO

Access via: Text

Description: PRIO input

FCT Access via: Numpad 5..16

Description: FCT input

NB Access via: Numpad 1..1968

Description: NB input

ADDR Access via: Numpad min. 0

Description: ADR input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Panel1.FB_Write

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.4 ACS / DCS drives communication via Modbus TCP library

To establish the communication to an ACS / DCS drive one of the following two libraries can be used: ACSDrivesComModTCP_AC500_V22.lib or ACSDrivesComModTCP_Ext_AC500_V24.lib ↳ *Chapter 1.5.6.5 “ACS / DCS drives communication via Modbus TCP ext library” on page 2384* The ACSDrivesComModTCP_AC500_V22.lib can be used, as long as the CPU does not support more than one internal Ethernet Interface. And this library must be used, if the Firmwareversion of the CPU is less than V2.4.x.

The ACSDrivesComModTCP_Ext_AC500_V24.lib should be used for new project and if the Firmware Version of the CPU is at least V2.4.x. It must be used, if a CPU with more than one internal Ethernet Interface is used, e.g. PM595.

1.5.6.4.1 Preconditions for the use of the ACS / DCS drives communication via Modbus TCP library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks can only be used in combination with the ACSDrives-Base_AC500_V20 Library.

The library is released for the following products:

- **CPUs:** AC500 and AC500-eCo
- **Fieldbus:** Modbus TCP
- **Drives:**ACS800, ACSM1, ACS350, ACS355, ACS550, ACH550, ACQ810, ACS850, ACS880, ACS580, DCS550, DCS800
- **Modbus TCP configuration:**
  Prior to the use of the function blocks a Communication Module "Modbus_on_TCP_IP" has to be configured accordingly using Automation Builder, either at "Onboard_Ethernet" or at "CM597-ETH" module.
- **ACS_COM_MOD_TCP and ACS_COM_MOD_TCP_ENHANCED:**
  The communication function blocks are designed to be used each for one specific drive at run time. So it´s not recommended to change the COM or SLAVE inputs of the blocks while the program is running.
  There is no check in the function blocks if the maximum number of TCP/IP sockets is already reached. So the user has to take care about the number of used socket. Each ACS_COM_MOD_TCP or ACS_COM_MOD_TCP_ENHANCED function block will use one socket.

1.5.6.4.2 Components of the ACS / DCS drives communication via Modbus TCP library

Function blocks

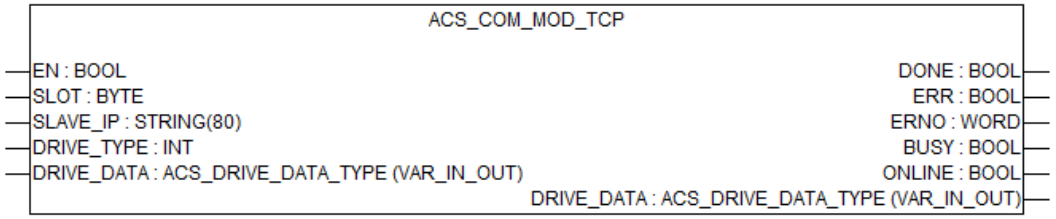
ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360*	Communication for ACS / DCS Drives via Modbus TCP
ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367*	Communication for ACS / DCSDrives via Modbus TCP

Visualizations

ACS_COM_MOD_TCP_VISU_PH *Chapter 1.5.6.4.4.1 "ACS_COM_MOD_TCP_VISU_PH faceplate for the function block ACS_COM_MOD_TCP" on page 2376*	Faceplate for the function block
ACS_COM_MOD_TCP_ENHANCED_VISU_PH *Chapter 1.5.6.4.4.2 "ACS_COM_MOD_TCP_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCP_ENHANCED" on page 2379*	Faceplate for the function block

1.5.6.4.3 Function blocks

ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP



Function block ACS_COM_MOD_TCP controls the Modbus TCP communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives profile.

Function block information

Available in runtime system:	V1.3.2
Included in library:	ACSDrivesComModTCP_AC500_V22_App.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_MOD_TCP controls the Modbus TCP communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives profile.

If the user changes drive profile while drive is online with PLC, function block output's may give wrong indication.

Reading Status Information from Drive

The function block continuously reads data from the drive starting at Modbus register 400004. So at least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable. These values are stored in DRIVE_DATA.MSW, ActValue1 and ActValue2.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	Mapping configuration in drive			Written to in AC500	Condition at function block
	ACS355, ACS850, ACQ810, ACSM1, ACS880	**ACS800**	**ACS550, ACH550**		
Communication module	FENA-01 / -11	RETA-01 / -02	RETA-01 / -02		
40004	Status Word (SW)	Status Word (SW) fix	Status Word 51.23 (SW) fix	DRIVE_DATA.sw	EN = TRUE
40005	Actual Value1	92.02 = Actual Value1, e.g. = 102 (Speed)	51.24 = Actual Value1 (fix)	DRIVE_DATA.actValue1	EN = TRUE
40006	Actual Value2	92.03 = Actual Value2 e.g. = 105 (Torque) *)	51.25 = Actual Value2, e.g.= 105 (Torque)	DRIVE_DATA.actValue2	EN = TRUE
*) If 51.19 .. 51.22 (Output 1 .. 4) are set to the actual values the Modbus response will be faster because those values are updated cyclically between RETA-01 and ACS800.					

Writing Control Word and Reference Values to Drive

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 40001.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	Communication module		Written to in AC500	Condition at function block
	ACS355, ACS850, ACQ810, ACSM1, ACS880, ACS580	**ACS800, ACS550, ACH550, DCS550, DCS800**		
Communication module	**FENA-01 / -11**	**RETA-01 / -02**		
40001	Control Word (CW)	Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
40002	Reference Value1	Reference Value1	DRIVE_DATA.ref Value1	EN = TRUE
40003	Reference Value2	Reference Value2	DRIVE_DATA.ref Value2	EN = TRUE

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore the drive parameters in FBA DATA OUT group have to be configured according to the used NVAR_WRITE input number.

If 32-bit parameters are mapped to DATA OUT,

- *The following field in DATA OUT has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. If e.g. FENA-x1 is used the configuration is done in Par.51.22.*
- *The original 32-bit value in AC500 has to be split up in HW and LW in the WRITE_VALUES array.*

Function block DATA OUT has to be configured in drive in the following groups see also FENA-x1 manual.

Drive	Parameter group
ACS355	55.01 .. 55.10
ACS850, ACQ810, ACSM1, ACS880	53.01 .. 53.12
ACS880	53.01 .. 53.02(Fieldbus A), 56.01 .. 56.12 (Fieldbus B)

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set. Please see drive manuals and following table which parameter has to be set.

Save valid parameters to permanent memory in drive	ACS3XX, ACX550, ACQ810, ACS850, ACSM1, ACS800	ACS880, ACS580	DCS550, DCS800
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1	Par 16.06 = 1

Read/Write Jobs Coming from Other Function Blocks

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/write function blocks e.g. ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* or ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215* of this drive.

Communication with Several ACS Drives

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_TCP or ACS_COM_MOD_TCP_ENHANCED function block must be programmed.

The function block provides the basic start/stop signals, basic diagnosis signals and the scaling of the SPEED_REF input and ACT_SPEED to the ACS fieldbus scaling range. -20000 .. +20000.

The AC500 CPU types provide different numbers of usable TCP/IP sockets. For each ACS Modbus TCP communication block (ACS_COM_MOD_TCP and ACS_COM_MOD_TCP_ENHANCED) one socket will be needed. The user has to check that the programmed number of ACS Modbus TCP communication blocks is not higher than the number of available free sockets for the used CPU.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>.
- Create a visualization element of the function block see ACS_COM_MOD_TCP_ENHANCED_VISU_PH.

The additional diagnosis variables are:

- iWriteErrCnt: number of errors in write jobs since EN = TRUE.
- wLastWriteErno: holds the error number of the last executed write job.
- iReadErrCnt: number of errors in read jobs since EN = TRUE.
- wLastReadErno: holds the error number of the last executed read job.

Preconditions

The function block is working with all ACS / DCS drives via Modbus TCP communication with field bus adapter FENA-X1 / RETA-X1.

The data transfer to other function blocks for this drive communication to the ACS / DCS drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to then ACS_COM_MOD_TCP even if no other function block is connected.

For ACS drive parameters must be set as follows:

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Setting according to AC500 configuration or function block input	**ACS355**	**ACS850, ACQ810, ACSM1**	**ACS580, ACS880**	**ACS800, ACS550, ACH550, DCS550, DCS800**
Communication module:	FENA-01	FENA-11	FENA-11	RETA-01 / RETA-02
Fieldbus activation = EXT FBA / ENABLE	98.02	50.01	50.01	98.02
COMM RATE = Auto (0)	51.03	51.03	51.03	51.02
IP CONFIGURATION = Static IP (0) ! not default ! Set 51.27 (Refresh) after first change to "Static IP".	51.04	51.04	51.04	51.03
IP ADDRESS1 .. IP ADDRESS4	51.05 .. 51.08	51.05 .. 51.08	51.05 .. 51.08	51.04 .. 51.07 **)

Setting according to AC500 configuration or function block input	ACS355	ACS850, ACQ810, ACSM1	ACS580, ACS880	ACS800, ACS550, ACH550, DCS550, DCS800
SUBNET CIDR = e.g. 255.255.255.0 = 24	51.09	51.09	51.09	51.08 .. 51.11
GateWay ADDRESS (normally = 0.0.0.0)	51.10 .. 51.13	51.10 .. 51.13	51.10 .. 51.13	51.12 .. 51.15
PROTOCOL / PROFILE = MB/TCP ABB E (ABB Drives Profile Classic) (0)	51.02	51.02	51.02	51.16
Word order for 32-bit parameter access	No 32-bit access	51.22	51.22	No 32-bit access
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as these values are only written after changes	51.21	51.21	Timeout mode	
Modbus Timeout. Depending on Timeout mode. Value in 100ms.	51.20	51.20	Modbus timeout	51.17
Refresh settings in drive	51.27	51.27	51.27	51.27
**) For RETA-01/-02 IP address could also be set via hardware DIP switches. If any switch is set (192.168.0.xxx) with xxx = DIP switches setting.				

For further settings, e.g. reaction of drive at communication error, please see related drive and fieldbus manual.

Input description

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS drive.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) after changes to the drive is started.

If EN is reset to FALSE while a Modbus TCP job is performed (BUSY = TRUE), the function block will be processed until the Modbus TCP job is terminated (DONE = TRUE for 1 cycle).

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero.

SLOT (slot) Data type: BYTE

At input SLOT, the Modbus interface number is specified:

SLOT = 0 ; SLOT0 (onboard Ethernet slot)

SLOT = 1 : SLOT1 (CM597-ETH)

SLOT = 2 : SLOT2 (CM597-ETH)

Default value = 0.

Maximum value = 4.

SLAVE_IP (slave IP-Address)

Data type: STRING

IP Address of the drive (slave) to which the connecton shall be established must be specified here as string. Used with 4 times 3 digits.

Default value: ‘192.168.005.003’

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

DRIVE_DATA (drive data)

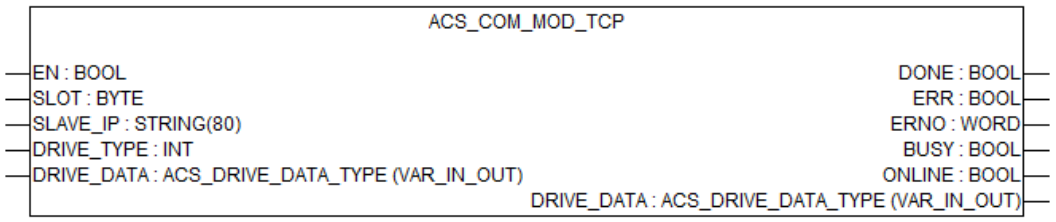
Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS / DCS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_TCP reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus TCP jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy) Data type: BOOL

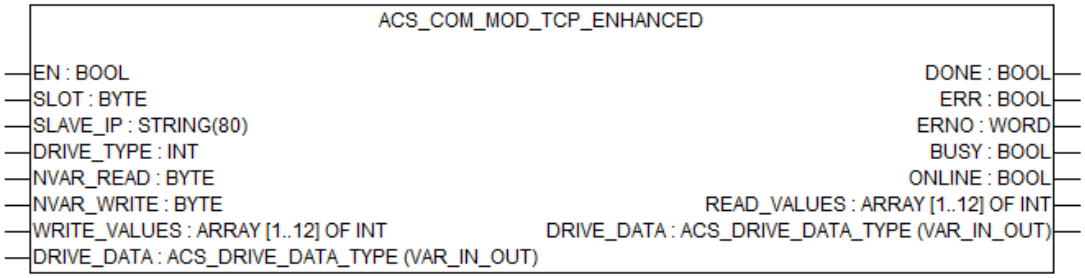
Output BUSY indicates, whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP



Function block ACS_COM_MOD_TCP_ENHANCED establishes the Modbus TCP communication to an ACS drive. This function block is used for the basic control of the ACS drive using ABB Drives Profile Enhanced. It also reads and writes the drive parameters which are mapped in the drive.

Function block information

Available in runtime system:	V2.4
Included in library:	ACSDrivesComModTCP_AC500_V22_App.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_MOD_TCP_ENHANCED establishes the Modbus TCP communication to an ACS drive and is used for the basic control of ACS drives with ABB Drives Profile Enhanced.

If the user changes drive profile while drive is online with PLC, function block output's may give wrong indication.

Reading Status Information from Drive

The function block continuously reads data from the drive starting at Modbus register 400051. So at least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA ↳ *Chapter 1.5.6.2.6.2 "ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive" on page 2253*variable. These values are stored in DRIVE_DATA.MSW, ActValue1 and ActValue2.

Apart from these three there is also an option to read 12 additional drive parameters. Using the input NVAR_READ the function block can be configured to read between 0 and 12 parameters from the drive. All read data is then written to the array at the READ_VALUE output. Configuration in ACS drive is depending of configured parameters in group FBD DATA IN.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	Mapping configuration in drive		Written to in AC500	Condition at function block
	ACS355	ACS850, ACQ810, ACSM1, ACS880, ACS580		
400051	Status Word (SW)	Status Word (SW)	DRIVE_DATA.sw	EN = TRUE
400052	Actual Value1	Actual Value1	DRIVE_DATA.act Value1	EN = TRUE
400053	Actual Value2	Actual Value2	DRIVE_DATA.act Value2	EN = TRUE
400054	FBA DATA IN 1	FBA DATA IN 1	READ_VALUES[1]	EN = TRUE and NVAR_READ >= 1
400055	FBA DATA IN 2	FBA DATA IN 2	READ_VALUES[2]	EN = TRUE and NVAR_READ >= 2
...	...	...	...	...
400063	FBA DATA IN 10	FBA DATA IN 10	READ_VALUES[10]	EN = TRUE and NVAR_READ >= 10
...		...	...	...
400065		FBA DATA IN 12	READ_VALUES[12]	EN = TRUE and NVAR_READ = 12

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information the job is aborted with an error.

Therefore the drive parameters in FBA DATA IN group have to be configured according to the used NVAR_READ input number.

If 32-bit parameters are mapped to DATA IN,

- *The following field in DATA IN has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. (using FENA-X1: Par. 51.22)*
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from READ_VALUES fields have to be recombined in the program.*

Function block DATA IN has to be configured in drive in the following groups see also FSCA-01 manual.

Drive	Parameter group
ACS355	54.01 .. 54.10
ACS850, ACQ810, ACSM1, ACS880, ACS580	52.01 .. 52.12 52.01 .. 52.12 if installed as adapter A

Writing Control Word and Reference Values to Drive

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 400001.

Apart from these three parameters there is also an option to write additional 12 more drive parameters in the same Modbus job. Using the input NVAR_WRITE the function block can be configured to write between 0 and 12 more parameters to the drive. The necessary values have to present in the array connected to WRITE_VALUES input.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area from where the data in the AC500 is taken.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Table 137: Writing control word and reference values to drive

Modbus register address in drive	**Mapping configuration in drive**		**Written to in AC500**	**Condition at function block**
	ACS355	ACS850, ACQ810, ACSM1, ACS880, ACS580		
400001	Control Word (CW)	Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
400002	Reference Value1	Reference Value1	DRIVE_DATA.ref Value1	EN = TRUE
400003	Reference Value2	Reference Value2	DRIVE_DATA.ref Value2	EN = TRU
400004	FBA DATA OUT 1	FBA DATA OUT 1	READ_VALUES[1]	EN = TRUE and NVAR_WRITE >= 1
400005	FBA DATA OUT 2	FBA DATA OUT 2	READ_VALUES[2]	EN = TRUE and NVAR_WRITE >= 2
...	...	...	...	...
400013	FBA DATA OUT 10	FBA DATA OUT 10	READ_VALUES[10]	EN = TRUE and NVAR_WRITE >= 10
...		...	...	...
400015		FBA DATA OUT 12	READ_VALUES[12]	EN = TRUE and NVAR_WRITE >= 12

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information the job is aborted with an error.

Therefore the drive parameters in FBA DATA OUT group have to be configured according to the used NVAR_WRITE input number.

If 32-bit parameters are mapped to DATA OUT,

- *The following field in DATA OUT has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. (using FENA-X1: Par. 51.22)*
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from WRITE_VALUES fields have to be recombined in the program.*

Function block DATA OUT has to be configured in drive in the following groups see also FENA-X1 manual.

Drive	Parameter group
ACS355	55.01 .. 55.10
ACS850, ACQ810, ACSM1, ACS580, ACS880	53.01 .. 53.12 53.01 .. 53.12 if installed as adapter A

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set.

Please see drive manuals and following table which parameter has to be set.

Save valid parameters to permanent memory in drive	ACS3XX, ACQ810, ACS850, ACSM1	ACS880, ACS580
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1

Read/Write Jobs Coming from Other Function Blocks

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/write function blocks e.g. ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* or ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215* of this drive.

Communication with several ACS Drives

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_TCP_ENHANCED or ACS_COM_MOD_TCP function block must be programmed.

The function block provides the basic start/stop signals, basic diagnosis signals and the scaling of the SPEED_REF input and ACT_SPEED to the ACS fieldbus scaling range -20000 .. +20000.

The AC500 CPU types provide different numbers of usable TCP/IP sockets. For each ACS Modbus TCP communication block (ACS_COM_MOD_TCP and ACS_COM_MOD_TCP_ENHANCED) one socket will be needed.

The user has to check that the programmed number of ACS_COM_MOD_TCP or ACS_COM_MOD_TCP_ENHANCED communication blocks is not higher than the number of available free sockets for the used CPU.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>.
- Create a visualization element of the function block see ACS_COM_MOD_TCP_ENHANCED_VISU_PH *Chapter 1.5.6.4.4.2 “ACS_COM_MOD_TCP_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCP_ENHANCED” on page 2379*.

The additional diagnosis variables are:

iWriteErrCnt:	number of errors in write jobs since EN = TRUE
wLastWriteErno:	holds the error number of the last executed write job
iReadErrCnt:	number of errors in read jobs since EN = TRUE
wLastReadErno:	holds the error number of the last executed read job

Preconditions

The function block is working with all ACS drives via Modbus TCP communication with field bus adapter FENA-X1.

The data transfer to other function blocks for this drive communication to the ACS drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to ACS_COM_MOD_TCP_ENHANCED even if no other function block is connected.

The following ACS drive parameters have to be set according to the configuration of the Modbus line and the inputs of the function block.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

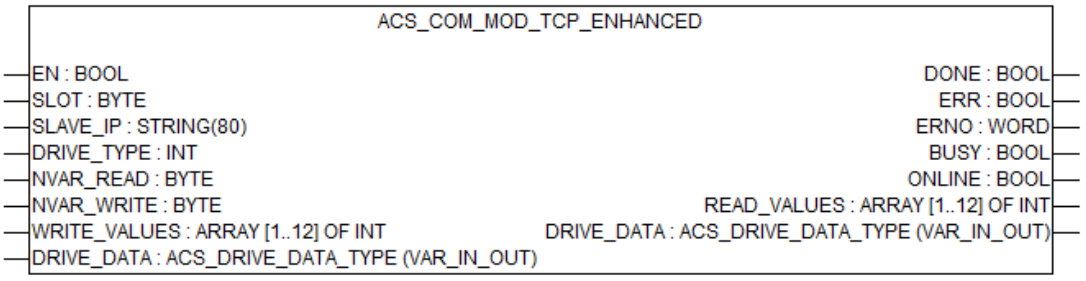
Setting according to AC500 configuration or function block input	**ACS355**	**ACS850, ACQ810, ACSM1,**	**ACS880, ACS580**	**ACX550, ACS800, DCS550, DCS800**
Communication module:	FENA-01	FENA-11	FENA-11	ENHANCED not possible with RETA-0X
Fieldbus activation = EXT FBA	98.02	50.01	50.01	
COMM RATE = Auto (0)	51.03	51.03	51.03	
IP CONFIGURATION = Static IP (0) ! not default ! Set 51.27 (Refresh) after first change to "Static IP".	51.04	51.04	51.04	
IP ADDRESS1 .. IP ADDRESS4	51.05 .. 51.08	51.05 .. 51.08	51.05 .. 51.08	
SUBNET CIDR = e.g. 255.255.255.0 = 24	51.09	51.09	51.09	

Setting according to AC500 configuration or function block input	ACS355	ACS850, ACQ810, ACSM1,	ACS880, ACS580	ACX550, ACS800, DCS550, DCS800
PROTOCOL / PROFILE = MB/TCP ABB E (ABB Drives Profile Enhanced) (1)	51.02	51.02	51.02	
Word order for 32-bit parameter access	No 32-bit access	51.22	51.22	
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as these values are only written after changes	51.21	51.21	Timeout mode	
Modbus Timeout. Depending on Timeout mode. Value in 100ms.	51.20	51.20	Modbus timeout	
Refresh settings in drive	51.27	51.27	51.27	

For further settings, e.g. reaction of drive at communication error, please see related drive and fieldbus manual.

Please refer FENA-X1 Ethernet adapter manual for more information.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS drive.

After successfully reading and writing of the ACS parameters, output ONLINE is set to TRUE.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) after changes to the drive is started.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero. The elements of the READ_VALUES array are also reset to zero.

SLOT (slot)

Data type: BYTE

At input SLOT, the Modbus interface number is specified:

SLOT = 0 ; SLOT0 (onboard Ethernet slot)

SLOT = 1 : SLOT1 (CM597-ETH)

SLOT = 2 : SLOT2 (CM597-ETH)

Default value = 0.

Maximum value = 4.

SLAVE_IP (slave IP-Address)

Data type: STRING

IP Address of the drive (slave) to which the connecton shall be established must be specified here as string. Used with 4 times 3 digits.

Default value: ‘192.168.005.003’

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

NVAR_READ (number of variables for reading)

Data type: BYTE

With the input NVAR_READ the function block can be configured to read between 1 and 12 signals from the drive. All read data is written to the array at the READ_VALUE output. Configuration in ACS drive is depending of configured parameters in group FBD DATA IN in drive

Default value = 0. Minimum 0, Maximum 12.

To read/write "one" 32-bit data, the NVAR should be equal to 2. Accordingly this has to be followed if we want to read/write more than one data. The user has to check that the programmed number of ACS_COM_MOD_TCP or ACS_COM_MOD_TCP_ENHANCED communication blocks is not higher than the number of available free sockets for the used CPU.

NVAR_WRITE (number of variables for writing)

Data type: BYTE

The internal Modbus write job writes 3 + NVAR_WRITE words starting from Modbus register address 400001 to the drive, every time a change in those variables is detected.

With the input NVAR_WRITE the function block can be configured to write between 0 and 12 variables more to the drive in addition to the 3 controls (CW, ref1 and ref2). These are always written to Modbus registers addresses 400001 .. 400003 if changed.

If NVAR_WRITE input is set to e.g. 5 the internal write job addresses the Modbus registers from 400001 .. 400008.

The first 3 signals are always the Control Word (CW), Reference value1 and Reference value2 and will be taken from the DRIVE_DATA variable.

The additional NVAR_WRITE data will be taken from the array at the WRITE_VALUES input.

Default value = 0. Minimum 0, Maximum 12.

WRITE_VALUES (write values to mapped parameters in drive group DATA_OUT)

Data type: ARRAY[1..12] OF INT

The values from the array at input WRITE_VALUE will be written to Modbus registers 400004 400015 in the drive. The number of data written to the drive is specified at the input NVAR_WRITE.

See table in chapter Writing Control Word and Reference Values to the drive for information about how to configure the drive.

DRIVE_DATA (drive data)

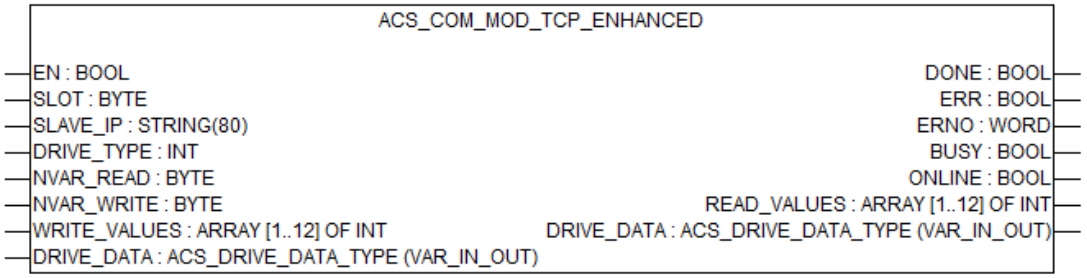
Data type: ACS_DRIVE_DATA_TYPE ↯ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_TCP_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy) Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

READ_VALUES (array of read values) Data type: ARRAY[1..12] OF INT

At output READ_VALUES the values of the array are updated after the read status information job was terminated successfully (DONE = TRUE, ERR = FALSE, BUSY = FALSE).

The read status information job is requested cyclically. It reads data from the ACS drive starting at Modbus register 400051 up to the number specified at input 3 + NVAR_READ

READ_VALUES contains the data as follows:

READ_VALUES[1] = <Modbus register 400054>

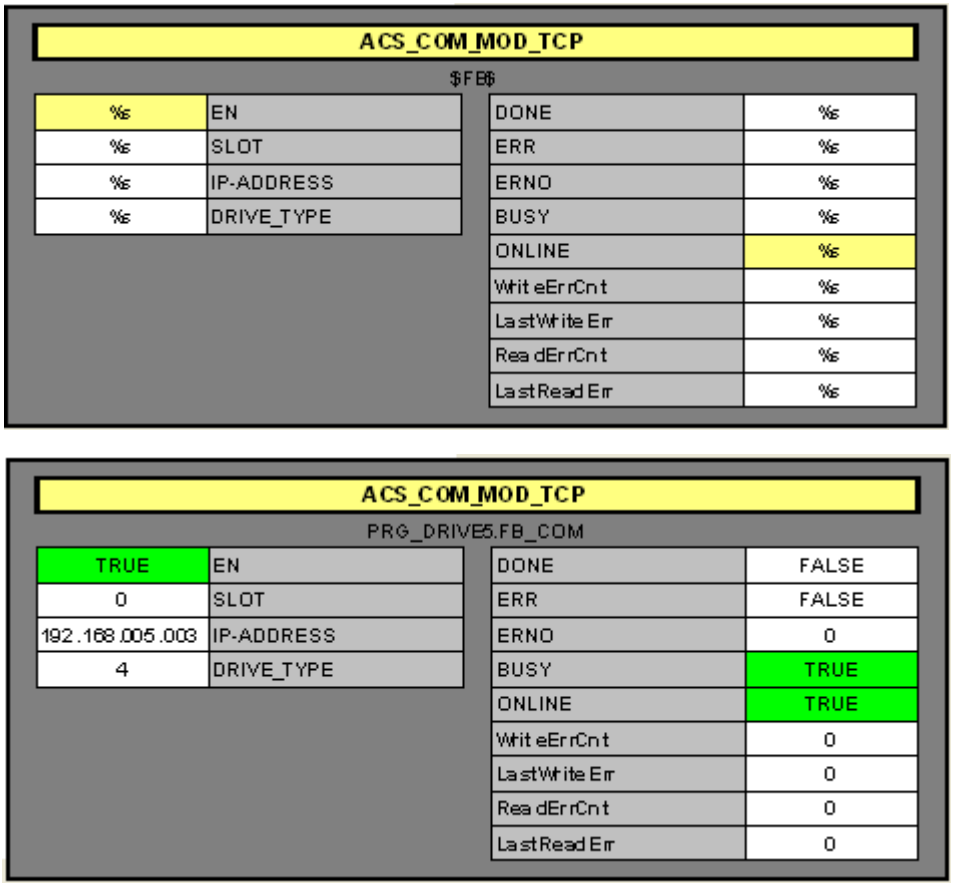
READ_VALUES[2] = <Modbus register 400055>

...

READ_VALUES[12] = <Modbus register 400065>.

1.5.6.4.4 Visualization

ACS_COM_MOD_TCP_VISU_PH faceplate for the function block ACS_COM_MOD_TCP



Visualization element ACS_COM_MOD_TCP_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCP ↬ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

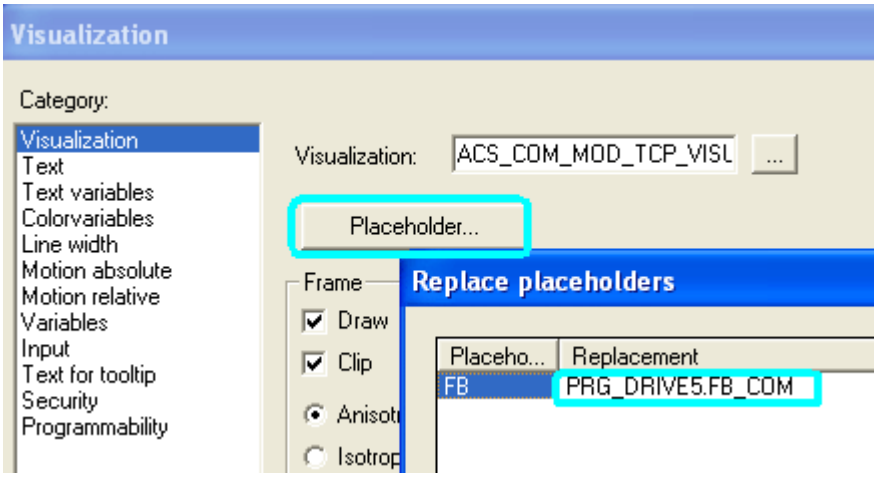
Visualization information

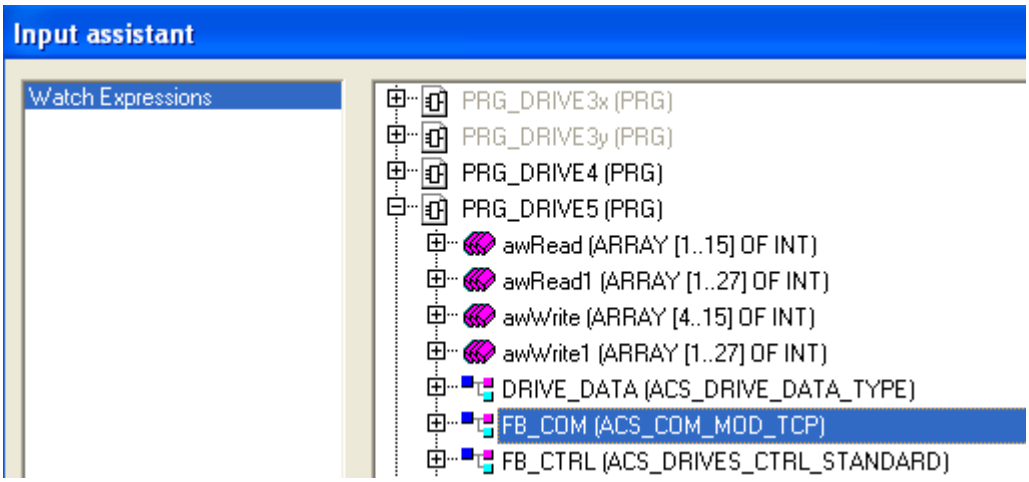
Available in runtime system:	V2.4
Included in library:	ACSDrivesComModTCP_Eth_AC500_V24.lib

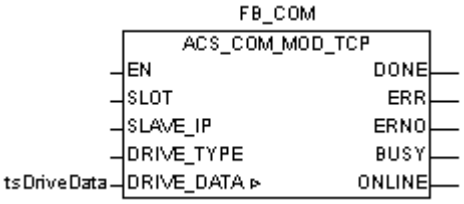
Visualization description

Visualization element ACS_COM_MOD_TCP_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCP ↬ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.







Parameters

Access R/W

EN Access via: Toggle

Description: EN input

SLOT Access via: Numpad 0 .. 4

Description: SLOT (module number) of the communication module

IP_ADDRESS Access via: Text

Description: Slave address

DRIVE_TYPE Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drivel.FB_COM

Access R

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of ETH_MOD_MAST function block.

Colors

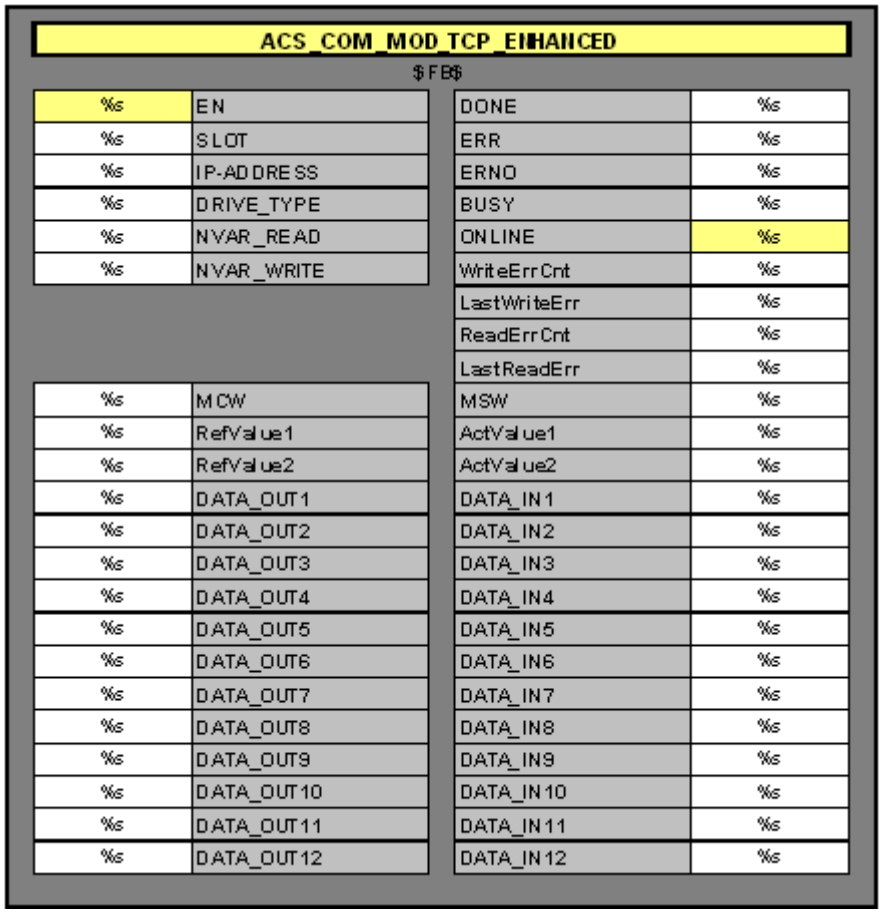
The color of the variables have the following meaning:

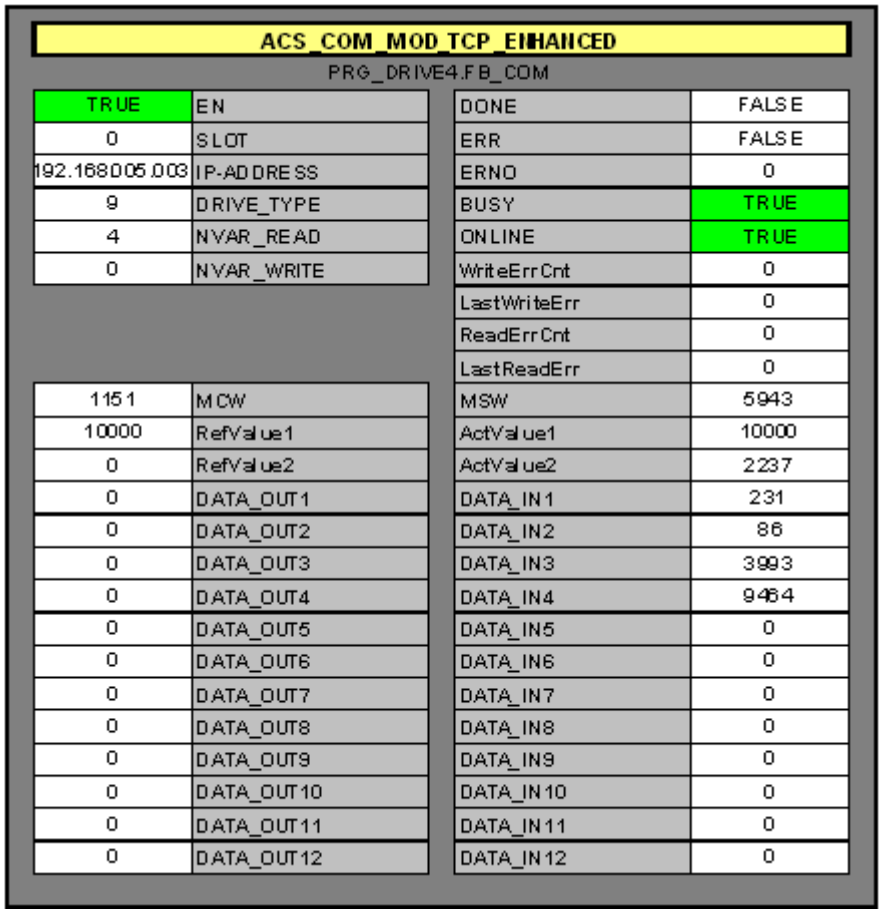
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_TCP_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCP_ENHANCED





Visualization element ACS_COM_MOD_TCP_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367* function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

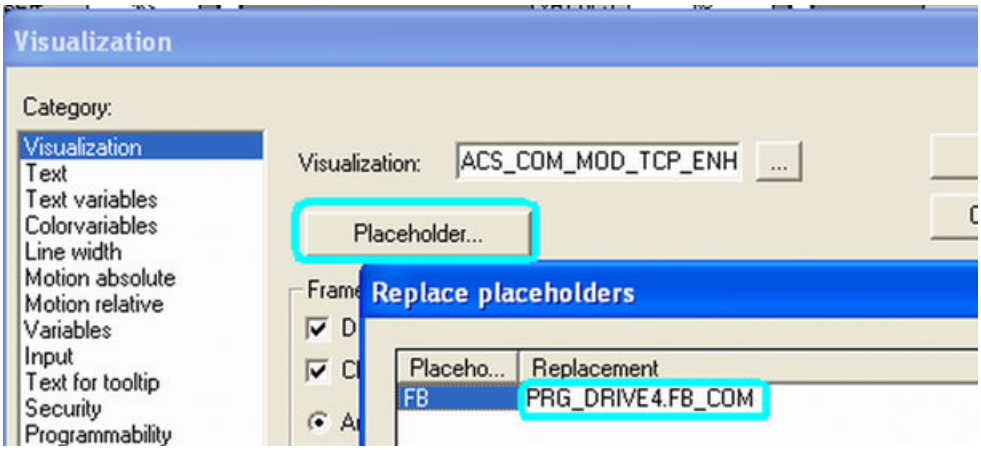
Visualization information

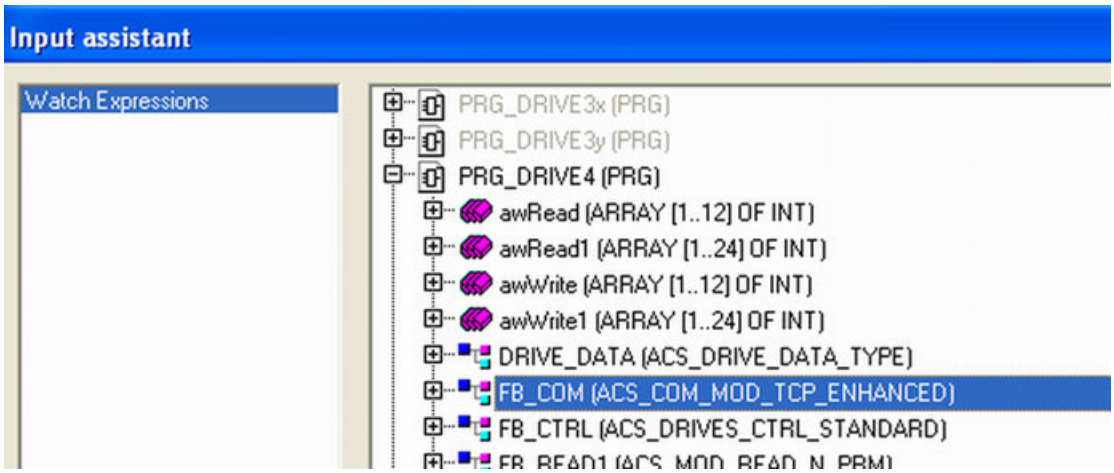
Available in runtime system:	V1.3.2
Included in library:	ACSDrivesComModTCP_AC500_V22_App.lib

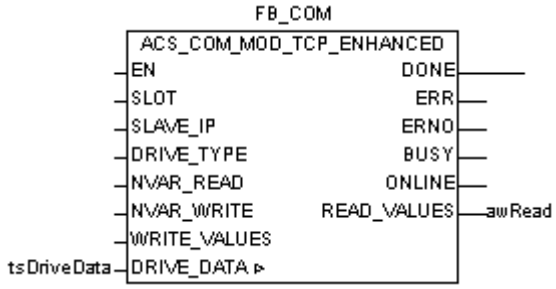
Visualization description

Visualization element ACS_COM_MOD_TCP_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCP_ENHANCED ↳ *Chapter 1.5.6.4.3.2 “ACS_COM_MOD_TCP_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2367* function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_TCP_ENHANCED function block, which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.







Parameters

Access R/W

EN
Access via: Toggle
Description: EN input

SLOT
Access via: Numpad 0 .. 4
Description: SLOT (module number) of the communication module

IP_ADDRESS
Access via: Text
Description: Slave address

DRIVE_TYPE
Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

NVAR_READ Access via: Numpad 0 .. 12
Description: NVAR_READ input

NVAR_WRITE Access via: Numpad 0 .. 12
Description: NVAR_WRITE input

DATA_OUT1 Description: WRITE_VALUE[1] input

DATA_OUT2 Description: WRITE_VALUE[2] input

DATA_OUT3 Description: WRITE_VALUE[3] input

DATA_OUT4 Description: WRITE_VALUE[4] input

DATA_OUT5 Description: WRITE_VALUE[5] input

DATA_OUT6 Description: WRITE_VALUE[6] input

DATA_OUT7 Description: WRITE_VALUE[7] input

DATA_OUT8 Description: WRITE_VALUE[8] input

DATA_OUT9 Description: WRITE_VALUE[9] input

DATA_OUT10 Description: WRITE_VALUE[10] input

DATA_OUT11 Description: WRITE_VALUE[11] input

DATA_OUT12 Description: WRITE_VALUE[12] input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drivel.FB_COM

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of ETH_MOD_MAST function block.

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

MSW Description: Status Word of drive, READ_VALUE[1] output.

ActValue1 Description: Actual Value 1 mapped in Par. 53.10 - READ_VALUE[2] output

ACT_VALUE2 Description: ACT_VALUE2 output

DATA_IN1 Description: READ_VALUE[1] output

DATA_IN2 Description: READ_VALUE[2] output

DATA_IN3 Description: READ_VALUE[3] output

DATA_IN4 Description: READ_VALUE[4] output

DATA_IN5 Description: READ_VALUE[5] output

DATA_IN6 Description: READ_VALUE[6] output

DATA_IN7 Description: READ_VALUE[7] output

DATA_IN8 Description: READ_VALUE[8] output

DATA_IN9 Description: READ_VALUE[9] output

DATA_IN10 Description: READ_VALUE[10] output

DATA_IN11 Description: READ_VALUE[11] output

DATA_IN12 Description: READ_VALUE[12] output

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.5 ACS / DCS drives communication via Modbus TCP ext library

To establish the communication to an ACS / DCS drive one of the following two libraries can be used: ACSDrivesComModTCP_AC500_V22.lib *Chapter 1.5.6.4 “ACS / DCS drives communication via Modbus TCP library” on page 2359* or ACSDrivesComModTCP_Ext_AC500_V24.lib The ACSDrivesComModTCP_AC500_V22.lib can be used, as long as the CPU does not support more than one internal Ethernet Interface. And this library must be used, if the Firmware-version of the CPU is less than V2.4.x.

The ACSDrivesComModTCP_Ext_AC500_V24.lib should be used for new project and if the Firmware Version of the CPU is at least V2.4.x. It must be used, if a CPU with more than one internal Ethernet Interface is used, e.g. PM595.

1.5.6.5.1 Preconditions for the use of the ACS / DCS drives communication via Modbus TCP ext library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks can only be used in combination with the ACSDrives-Base_AC500_V20 Library.

The library is released for the following products:

- **CPUs:** AC500 and AC500-eCo
- **Fieldbus:** Modbus TCP
- **Drives:**ACS800, ACSM1, ACS350, ACS355, ACS550, ACH550, ACQ810, ACS850, ACS880, ACS580, DCS550, DCS800
- **Modbus TCP configuration:**
  Prior to the use of the function blocks a Communication Module "Modbus_on_TCP_IP" has to be configured accordingly using Automation Builder, either at "Onboard_Ethernet" or at "CM597-ETH" module.
- **ACS_COM_MOD_TCPx and ACS_COM_MOD_TCPx_ENHANCED:**
  The communication function blocks are designed to be used each for one specific drive at runtime. So it´s not recommended to change the COM or SLAVE inputs of the blocks while the program is running.
  There is no check in the function blocks if the maximum number of TCP/IP sockets is already reached. So the user has to take care about the number of used socket. Each ACS_COM_MOD_TCP or ACS_COM_MOD_TCP_ENHANCED function block will use one socket.

1.5.6.5.2 Components of the ACS / DCS drives communication via Modbus TCP ext library

Function blocks

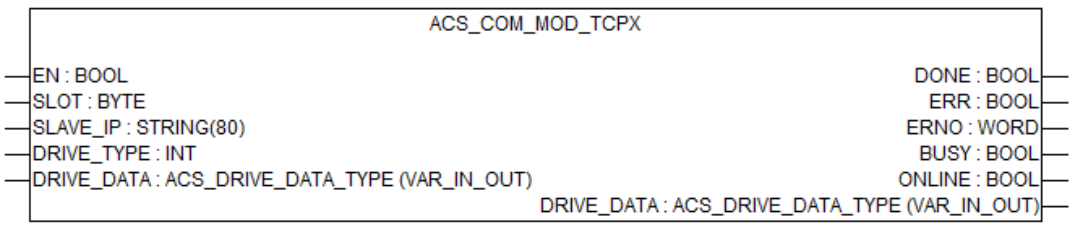
ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 "ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP" on page 2385*	Communication for ACS / DCS Drives via Modbus TCP
ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 "ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP" on page 2392*	Communication for ACS / DCSDrives via Modbus TCP

Visualizations

ACS_COM_MOD_TCPx_VISU_PH ↳ *Chapter 1.5.6.5.4.1 "ACS_COM_MOD_TCPx_VISU_PH faceplate for the function block ACS_COM_MOD_TCPx" on page 2402*	Faceplate for the function block
ACS_COM_MOD_TCPx_ENHANCED_VISU_PH ↳ *Chapter 1.5.6.5.4.2 "ACS_COM_MOD_TCPx_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCPx_ENHANCED" on page 2405*	Faceplate for the function block

1.5.6.5.3 Function blocks

ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP



Function block ACS_COM_MOD_TCPx controls the Modbus TCP communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives profile.

Function block information

Available in runtime system:	V2.4.x
Included in library:	ACSDrivesCom-ModTCP_Ext_AC500_V24_App.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_MOD_TCPx controls the Modbus TCP communication to an ACS / DCS drive and is used for the basic control of ACS / DCS drives with ABB Drives profile.

If the user changes drive profile while drive is online with PLC, function block output's may give wrong indication.

Reading Status Information from Drive

The function block continuously reads data from the drive starting at Modbus register 400004. So at least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable. These values are stored in DRIVE_DATA.MSW, ActValue1 and ActValue2.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	**Mapping configuration in drive**			**Written to in AC500**	**Condition at function block**
	ACS355, ACS850, ACQ810, ACSM1, ACS880	**ACS800**	**ACS550, ACH550**		
Communication module	FENA-01 / -11	RETA-01 / -02	RETA-01 / -02		
40004	Status Word (SW)	Status Word (SW) fix	Status Word 51.23 (SW) fix	DRIVE_DATA.sw	EN = TRUE
40005	Actual Value1	92.02 = Actual Value1, e.g. = 102 (Speed)	51.24 = Actual Value1 (fix)	DRIVE_DATA.actValue1	EN = TRUE
40006	Actual Value2	92.03 = Actual Value2 e.g. = 105 (Torque) *)	51.25 = Actual Value2, e.g.= 105 (Torque)	DRIVE_DATA.actValue2	EN = TRUE
*) If 51.19 .. 51.22 (Output 1 .. 4) are set to the actual values the Modbus response will be faster because those values are updated cyclically between RETA-01 and ACS800.					

Writing Control Word and Reference Values to Drive

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 40001.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	**Communication module**		**Written to in AC500**	**Condition at function block**
	ACS355, ACS850, ACQ810, ACSM1, ACS880, ACS580	**ACS800, ACS550, ACH550, DCS550, DCS800**		
Communication module	**FENA-01 / -11**	**RETA-01 / -02**		
40001	Control Word (CW)	Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
40002	Reference Value1	Reference Value1	DRIVE_DATA.ref Value1	EN = TRUE
40003	Reference Value2	Reference Value2	DRIVE_DATA.ref Value2	EN = TRUE

If a Modbus job tries to access a register in the drive which has no valid mapping information the job is aborted with an error. Therefore the drive parameters in FBA DATA OUT group have to be configured according to the used NVAR_WRITE input number.

If 32-bit parameters are mapped to DATA OUT,

- *The following field in DATA OUT has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. If e.g. FENA-x1 is used the configuration is done in Par.51.22.*
- *The original 32-bit value in AC500 has to be split up in HW and LW in the WRITE_VALUES array.*

Function block DATA OUT has to be configured in drive in the following groups see also FENA-x1 manual.

Drive	**Parameter group**
ACS355	55.01 .. 55.10
ACS850, ACQ810, ACSM1, ACS880	53.01 .. 53.12
ACS880	53.01 .. 53.02(Fieldbus A), 56.01 .. 56.12 (Fieldbus B)

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set. Please see drive manuals and following table which parameter has to be set.

Save valid parameters to permanent memory in drive	ACS3XX, ACX550, ACQ810, ACS850, ACSM1, ACS800	ACS880, ACS580	DCS550, DCS800
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1	Par 16.06 = 1

Read/Write Jobs Coming from Other Function Blocks

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/write function blocks e.g. ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* or ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215* of this drive.

Communication with Several ACS Drives

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_TCPx or ACS_COM_MOD_TCPx_ENHANCED function block must be programmed.

The function block provides the basic start/stop signals, basic diagnosis signals and the scaling of the SPEED_REF input and ACT_SPEED to the ACS fieldbus scaling range. -20000 .. +20000.

The AC500 CPU types provide different numbers of usable TCP/IP sockets. For each ACS Modbus TCP communication block (ACS_COM_MOD_TCPx and ACS_COM_MOD_TCPx_ENHANCED) one socket will be needed. The user has to check that the programmed number of ACS Modbus TCP communication blocks is not higher than the number of available free sockets for the used CPU.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>.
- Create a visualization element of the function block see ACS_COM_MOD_TCPx_VISU_PH.

The additional diagnosis variables are:

- iWriteErrCnt: number of errors in write jobs since EN = TRUE.
- wLastWriteErno: holds the error number of the last executed write job.
- iReadErrCnt: number of errors in read jobs since EN = TRUE.
- wLastReadErno: holds the error number of the last executed read job.

Preconditions

The function block is working with all ACS / DCS drives via Modbus TCP communication with field bus adapter FENA-X1 / RETA-X1.

The data transfer to other function blocks for this drive communication to the ACS / DCS drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to then ACS_COM_MOD_TCPx even if no other function block is connected.

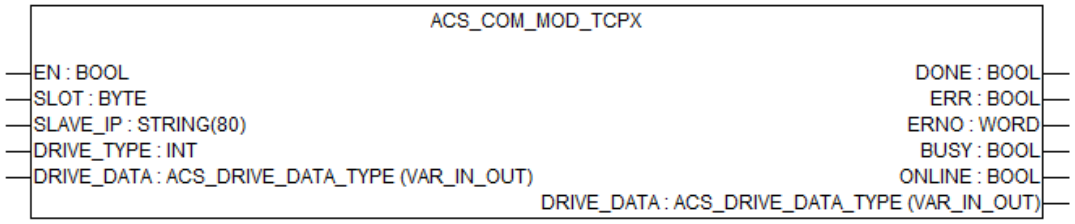
For ACS drive parameters must be set as follows:

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Setting according to AC500 configuration or function block input	**ACS355**	**ACS850, ACQ810, ACSM1**	**ACS580, ACS880**	**ACS800, ACS550, ACH550, DCS550, DCS800**
Communication module:	FENA-01	FENA-11	FENA-11	RETA-01 / RETA-02
Fieldbus activation = EXT FBA / ENABLE	98.02	50.01	50.01	98.02
COMM RATE = Auto (0)	51.03	51.03	51.03	51.02
IP CONFIGURATION = Static IP (0) ! not default ! Set 51.27 (Refresh) after first change to "Static IP".	51.04	51.04	51.04	51.03
IP ADDRESS1 .. IP ADDRESS4	51.05 .. 51.08	51.05 .. 51.08	51.05 .. 51.08	51.04 .. 51.07 **)
SUBNET CIDR = e.g. 255.255.255.0 = 24	51.09	51.09	51.09	51.08 .. 51.11
GateWay ADDRESS (normally = 0.0.0.0)	51.10 .. 51.13	51.10 .. 51.13	51.10 .. 51.13	51.12 .. 51.15
PROTOCOL / PROFILE = MB/TCP ABB E (ABB Drives Profile Classic) (0)	51.02	51.02	51.02	51.16
Word order for 32-bit parameter access	No 32-bit access	51.22	51.22	No 32-bit access
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as these values are only written after changes	51.21	51.21	Timeout mode	
Modbus Timeout. Depending on Timeout mode. Value in 100ms.	51.20	51.20	Modbus timeout	51.17
Refresh settings in drive	51.27	51.27	51.27	51.27
**) For RETA-01/-02 IP address could also be set via hardware DIP switches. If any switch is set (192.168.0.xxx) with xxx = DIP switches setting.				

For further settings, e.g. reaction of drive at communication error, please see related drive and fieldbus manual.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS drive.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) after changes to the drive is started.

If EN is reset to FALSE while a Modbus TCP job is performed (BUSY = TRUE), the function block will be processed until the Modbus TCP job is terminated (DONE = TRUE for 1 cycle).

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero.

SLOT (slot)

Data type: BYTE

At input SLOT, the Modbus interface number is specified:

SLOT = 1 : internal Ethernet communication module ETH1

SLOT = 2 : internal Ethernet communication module ETH2

SLOT = 11 : SLOT1 (CM597-ETH)

SLOT = 21 : SLOT2 (CM597-ETH)

SLOT = 31 : SLOT3 (CM597-ETH)

SLOT = 41 : SLOT4 (CM597-ETH)

Default value = 1. Valid values = 1, 2, 11, 21, 31, 41

SLAVE_IP (slave IP-Address)

Data type: STRING

IP Address of the drive (slave) to which the connecton shall be established must be specified here as string. Used with 4 times 3 digits.

Default value: ‘192.168.005.003’

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

DRIVE_DATA (drive data)

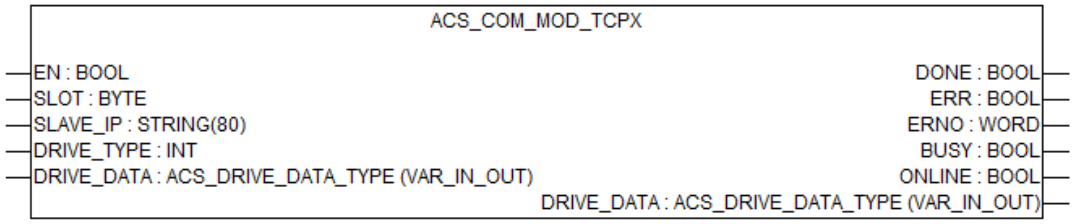
Data type: ACS_DRIVE_DATA_TYPE *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_TCPx reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy) Data type: BOOL

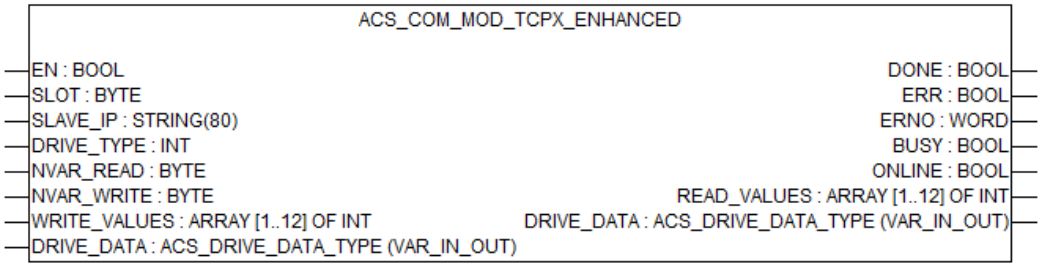
Output BUSY indicates, whenever there is a communication action performed.

ONLINE (online) Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP



Function block ACS_COM_MOD_TCPx_ENHANCED establishes the Modbus TCP communication to an ACS drive. This function block is used for the basic control of the ACS drive using ABB Drives Profile Enhanced .It also reads and writes the drive parameters which are mapped in the drive.

Function block information

Available in runtime system:	V2.4x
Included in library:	ACSDrivesCom-ModTCP_Ext_AC500_V24_App.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_MOD_TCPx_ENHANCED establishes the Modbus TCP communication to an ACS drive and is used for the basic control of ACS drives with ABB Drives Profile Enhanced.

If the user changes drive profile while drive is online with PLC, function block output's may give wrong indication.

Reading Status Information from Drive

The function block continuously reads data from the drive starting at Modbus register 400051. So at least the Status Word (SW), Actual Value 1 (SPEED_REF), Actual Value 2 (ACT_VALUE2) are continuously read from the drive and written to the DRIVE_DATA variable. These values are stored in DRIVE_DATA.MSW, ActValue1 and ActValue2.

Apart from these three there is also an option to read 12 additional drive parameters. Using the input NVAR_READ the function block can be configured to read between 0 and 12 parameters from the drive. All read data is then written to the array at the READ_VALUE output. Configuration in ACS drive is depending of configured parameters in group FBD DATA IN.

The following table shows the performed Modbus read job and the needed mapping in the drive as well as the area where the data is stored in the AC500.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	Mapping configuration in drive		Written to in AC500	Condition at function block
	ACS355	ACS850, ACQ810, ACSM1, ACS880, ACS580		
400051	Status Word (SW)	Status Word (SW)	DRIVE_DATA.sw	EN = TRUE
400052	Actual Value1	Actual Value1	DRIVE_DATA.act Value1	EN = TRUE
400053	Actual Value2	Actual Value2	DRIVE_DATA.act Value2	EN = TRUE
400054	FBA DATA IN 1	FBA DATA IN 1	READ_VALUES[1]	EN = TRUE and NVAR_READ >= 1

Modbus register address in drive	Mapping configuration in drive		Written to in AC500	Condition at function block
400055	FBA DATA IN 2	FBA DATA IN 2	READ_VALUES[2]	EN = TRUE and NVAR_READ >= 2
...	...	...	...	...
400063	FBA DATA IN 10	FBA DATA IN 10	READ_VALUES[10]	EN = TRUE and NVAR_READ >= 10
...		...	...	...
400065		FBA DATA IN 12	READ_VALUES[12]	EN = TRUE and NVAR_READ = 12

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information the job is aborted with an error.

Therefore the drive parameters in FBA DATA IN group have to be configured according to the used NVAR_READ input number.

If 32-bit parameters are mapped to DATA IN,

- *The following field in DATA IN has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. (using FENA-X1: Par. 51.22)*
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from READ_VALUES fields have to be recombined in the program.*

Function block DATA IN has to be configured in drive in the following groups see also FSCA-01 manual.

Drive	Parameter group
ACS355	54.01 .. 54.10
ACS850, ACQ810, ACSM1, ACS880, ACS580	52.01 .. 52.12 52.01 .. 52.12 if installed as adapter A

Writing Control Word and Reference Values to Drive

The function block checks if there are changes of the Control Word (CW), Reference Value 1 (SPEED_REF) or Reference Value 2 (REF_VALUE2) on the DRIVE_DATA variable. If there is a change a write job is requested to send these 3 values to the ACS drive starting at Modbus register 400001.

Apart from these three parameters there is also an option to write additional 12 more drive parameters in the same Modbus job. Using the input NVAR_WRITE the function block can be configured to write between 0 and 12 more parameters to the drive. The necessary values have to present in the array connected to WRITE_VALUES input.

The following table shows the performed Modbus write job and the needed mapping in the drive as well as the area from where the data in the AC500 is taken.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

Modbus register address in drive	Mapping configuration in drive		Written to in AC500	Condition at function block
	ACS355	ACS850, ACQ810, ACSM1, ACS880, ACS580		
400001	Control Word (CW)	Control Word (CW)	DRIVE_DATA.cw	EN = TRUE
400002	Reference Value1	Reference Value1	DRIVE_DATA.ref Value1	EN = TRUE
400003	Reference Value2	Reference Value2	DRIVE_DATA.ref Value2	EN = TRU
400004	FBA DATA OUT 1	FBA DATA OUT 1	READ_VALUES[1]	EN = TRUE and NVAR_WRITE >= 1
400005	FBA DATA OUT 2	FBA DATA OUT 2	READ_VALUES[2]	EN = TRUE and NVAR_WRITE >= 2
...	...	...	...	...
400013	FBA DATA OUT 10	FBA DATA OUT 10	READ_VALUES[10]	EN = TRUE and NVAR_WRITE >= 10
...		...	...	...
400015		FBA DATA OUT 12	READ_VALUES[12]	EN = TRUE and NVAR_WRITE >= 12

If a Modbus TCP job tries to access a register in the drive which has no valid mapping information the job is aborted with an error.

Therefore the drive parameters in FBA DATA OUT group have to be configured according to the used NVAR_WRITE input number.

If 32-bit parameters are mapped to DATA OUT,

- *The following field in DATA OUT has to be left open (= 0)*
- *The word order of the High-Word and Low-Word can be configured in the drive. (using FENA-X1: Par. 51.22)*
- *To retrieve the original 32-bit value from the drive in AC500 the HW and LW from WRITE_VALUES fields have to be recombined in the program.*

Function block DATA OUT has to be configured in drive in the following groups see also FENA-X1 manual.

Drive	Parameter group
ACS355	55.01 .. 55.10
ACS850, ACQ810, ACSM1, ACS580, ACS880	53.01 .. 53.12 53.01 .. 53.12 if installed as adapter A

ACS drive parameters are only saved temporarily, if changed via fieldbus. To make these changes permanent in the drive the special parameter "PARAMETER SAVE" has to be set.

Please see drive manuals and following table which parameter has to be set.

Save valid parameters to permanent memory in drive	ACS3XX, ACQ810, ACS850, ACSM1	ACS880, ACS580
1 = Saves the valid parameter values to permanent memory. 0 = Save completed.	Par 16.07 = 1	Par 96.07 = 1

Read/Write Jobs Coming from Other Function Blocks

The requests to process other read or write Modbus jobs is transferred via the DRIVE_DATA variable at the IN_OUT variable DRIVE_DATA which can be connected to several other read/write function blocks e.g. ACS_MOD_READ_N_PRM *Chapter 1.5.6.2.4.1 “ACS_MOD_READ_N_PRM” on page 2212* or ACS_MOD_WRITE_N_PRM *Chapter 1.5.6.2.4.2 “ACS_MOD_WRITE_N_PRM” on page 2215* of this drive.

Communication with several ACS Drives

If several drives are used, for each drive a communication function block such as ACS_COM_MOD_TCPx_ENHANCED or ACS_COM_MOD_TCPx function block must be programmed.

The function block provides the basic start/stop signals, basic diagnosis signals and the scaling of the SPEED_REF input and ACT_SPEED to the ACS fieldbus scaling range -20000 .. +20000.

The AC500 CPU types provide different numbers of usable TCP/IP sockets. For each ACS Modbus TCP communication block (ACS_COM_MOD_TCPx and ACS_COM_MOD_TCPx_ENHANCED) one socket will be needed.

The user has to check that the programmed number of ACS_COM_MOD_TCPx or ACS_COM_MOD_TCPx_ENHANCED communication blocks is not higher than the number of available free sockets for the used CPU.

Diagnosis

The output ERNO, which reflects an actual error number is only valid for one cycle if DONE and ERR output are set to TRUE.

To catch this error number an external function must be programmed.

However there are internal diagnosis variables available, which are not shown at any output, but can be accessed from the function block instance. This can be done in three ways:

- Opening the "+" sign of function block instance in the declaration part being online.
- Create an assignment in the code with <instance>.<diagnosis variable>.
- Create a visualization element of the function block see ACS_COM_MOD_TCPx_ENHANCED_VISU_PH *Chapter 1.5.6.5.4.2 “ACS_COM_MOD_TCPx_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCPx_ENHANCED” on page 2405*.

The additional diagnosis variables are:

iWriteErrCnt:	number of errors in write jobs since EN = TRUE
wLastWriteErno:	holds the error number of the last executed write job

iReadErrCnt:	number of errors in read jobs since EN = TRUE
wLastReadErno:	holds the error number of the last executed read job

Preconditions

The function block is working with all ACS drives via Modbus TCP communication with field bus adapter FENA-X1.

The data transfer to other function blocks for this drive communication to the ACS drive is realized via the IN_OUTPUT variable DRIVE_DATA, which must be connected to ACS_COM_MOD_TCPx_ENHANCED even if no other function block is connected.

The following ACS drive parameters have to be set according to the configuration of the Modbus line and the inputs of the function block.

Please refer the respective drives / fieldbus module manual for parameter setting, if the drive setting is not mentioned in below table.

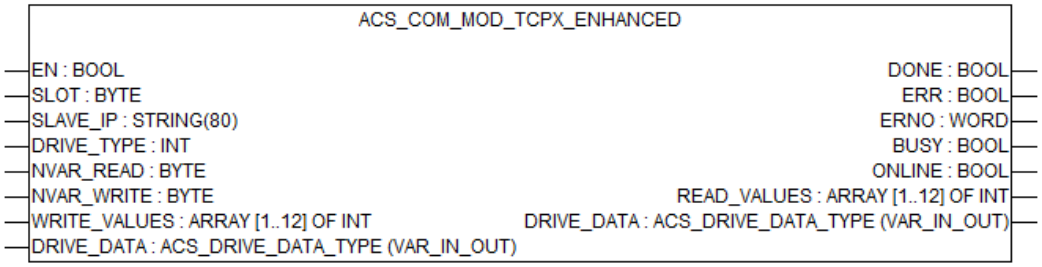
Setting according to AC500 configuration or function block input	**ACS355**	**ACS850, ACQ810, ACSM1,**	**ACS880, ACS580**	**ACX550, ACS800, DCS550, DCS800**
Communication module:	FENA-01	FENA-11	FENA-11	ENHANCED not possible with RETA-0X
Fieldbus activation = EXT FBA	98.02	50.01	50.01	
COMM RATE = Auto (0)	51.03	51.03	51.03	
IP CONFIGURATION = Static IP (0) ! not default ! Set 51.27 (Refresh) after first change to "Static IP".	51.04	51.04	51.04	
IP ADDRESS1 .. IP ADDRESS4	51.05 .. 51.08	51.05 .. 51.08	51.05 .. 51.08	
SUBNET CIDR = e.g. 255.255.255.0 = 24	51.09	51.09	51.09	
PROTOCOL / PROFILE = MB/TCP ABB E (ABB Drives Profile Enhanced) (1)	51.02	51.02	51.02	
Word order for 32-bit parameter access	No 32-bit access	51.22	51.22	
Timeout mode = None(0) or Any message(1), but not Ctrl write(2) as these values are only written after changes	51.21	51.21	Timeout mode	

Setting according to AC500 configuration or function block input	ACS355	ACS850, ACQ810, ACSM1,	ACS880, ACS580	ACX550, ACS800, DCS550, DCS800
Modbus Timeout. Depending on Timeout mode. Value in 100ms.	51.20	51.20	Modbus timeout	
Refresh settings in drive	51.27	51.27	51.27	

For further settings, e.g. reaction of drive at communication error, please see related drive and fieldbus manual.

Please refer FENA-X1 Ethernet adapter manual for more information.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the block is active, the current values are available at the outputs.

After a rising edge (FALSE -> TRUE) of input EN output ONLINE is set to FALSE and parameters are read from the ACS drive.

After successfully reading and writing of the ACS parameters, output ONLINE is set to TRUE.

The processing of continuously read of status information from the drive (SW, ACT_VALUE1 and ACT_VALUE2) and writing of Control Word and Reference Values (CW, SPEED_REF, REF_VALUE2) after changes to the drive is started.

If EN is reset to FALSE while a Modbus job is performed (BUSY = TRUE), the function block will be processed until the Modbus job is terminated (DONE = TRUE for 1 cycle).

If EN = FALSE the outputs ONLINE are reset to zero, as well as the data SW, actValue1 and actValue2 on the DRIVE_DATA variable are reset to zero. The elements of the READ_VALUES array are also reset to zero.

SLOT (slot) Data type: BYTE

At input SLOT, the Modbus interface number is specified:

SLOT = 1 : internal Ethernet communication module ETH1

SLOT = 2 : internal Ethernet communication module ETH2

SLOT = 11 : SLOT1 (CM597-ETH)

SLOT = 21 : SLOT2 (CM597-ETH)

SLOT = 31 : SLOT3 (CM597-ETH)

SLOT = 41 : SLOT4 (CM597-ETH)

Default value = 1. Valid values = 1, 2, 11, 21, 31, 41

SLAVE_IP (slave IP-Address)

Data type: STRING

IP Address of the drive (slave) to which the connecton shall be established must be specified here as string. Used with 4 times 3 digits.

Default value: ‘192.168.005.003’

DRIVE_TYPE (drive type)

Data type: INT, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM. The input can be set either by the value directly or by using the enum.

ENUM	Value
ACS_DRIVE_ACS800	1
ACS_DRIVE_ACSM1	2
ACS_DRIVE_ACS350	3
ACS_DRIVE_ACS355	4
ACS_DRIVE_ACS310	5
ACS_DRIVE_ACS550	6
ACS_DRIVE_ACH550	7
ACS_DRIVE_ACQ810	8
ACS_DRIVE_ACS850	9
ACS_DRIVE_ACS880	10
ACS_DRIVE_ACS580	11
ACS_DRIVE_DCS800	12
ACS_DRIVE_DCS550	13
ACS_DRIVE_ACH580	14
ACS_DRIVE_ACS380	15
ACS_DRIVE_ACS480	16
ACS_DRIVE_ACQ580	17

NVAR_READ (number of variables for reading)

Data type: BYTE

With the input NVAR_READ the function block can be configured to read between 1 and 12 signals from the drive. All read data is written to the array at the READ_VALUE output. Configuration in ACS drive is depending of configured parameters in group FBD DATA IN in drive

Default value = 0. Minimum 0, Maximum 12.

To read/write "one" 32-bit data, the NVAR should be equal to 2. Accordingly this has to be followed if we want to read/write more than one data. The user has to check that the programmed number of ACS_COM_MOD_TCPx or ACS_COM_MOD_TCPx_ENHANCED communication blocks is not higher than the number of available free sockets for the used CPU.

NVAR_WRITE (number of variables for writing)

Data type: BYTE

The internal Modbus write job writes 3 + NVAR_WRITE words starting from Modbus register address 400001 to the drive, every time a change in those variables is detected.

With the input NVAR_WRITE the function block can be configured to write between 0 and 12 variables more to the drive in addition to the 3 controls (CW, ref1 and ref2). These are always written to Modbus registers addresses 400001 .. 400003 if changed.

If NVAR_WRITE input is set to e.g. 5 the internal write job addresses the Modbus registers from 400001 .. 400008.

The first 3 signals are always the Control Word (CW), Reference value1 and Reference value2 and will be taken from the DRIVE_DATA variable.

The additional NVAR_WRITE data will be taken from the array at the WRITE_VALUES input.

Default value = 0. Minimum 0, Maximum 12.

WRITE_VALUES (write values to mapped parameters in drive group DATA_OUT)

Data type: ARRAY[1..12] OF INT

The values from the array at input WRITE_VALUE will be written to Modbus registers 400004 400015 in the drive. The number of data written to the drive is specified at the input NVAR_WRITE.

See table in chapter Writing Control Word and Reference Values to the drive for information about how to configure the drive.

DRIVE_DATA (drive data)

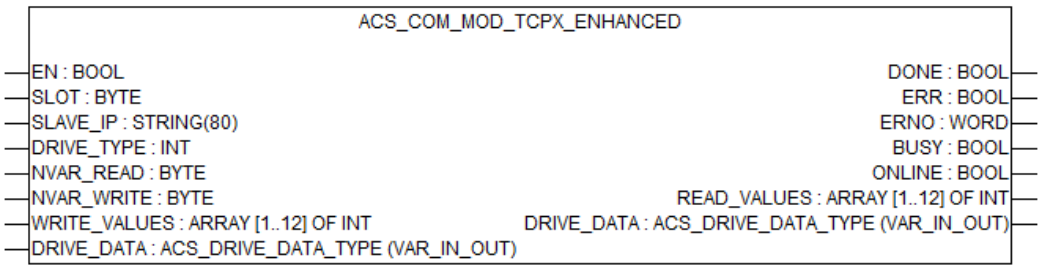
Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

The function block ACS_COM_MOD_TCPx_ENHANCED reads the Control Word and references (CW, SPEED_REF, REF_VALUE2) from the DRIVE_DATA variable and writes the status information (SW, ACT_SPEED, ACT-VALUE2) to the DRIVE_DATA variable. It also receives requests and data for Modbus TCP jobs from other function blocks e.g. ACS_MOD_READ_N_PRM or ACS_MOD_WRITE_N_PRM via the DRIVE_DATA variable.

Output description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

BUSY (busy)

Data type: BOOL

Output BUSY indicates whenever there is a communication action performed.

ONLINE (online)

Data type: BOOL

After the first time input EN is set to TRUE and at least one read job and one write job is performed successfully, output ONLINE is set to TRUE.

Output ONLINE is reset to FALSE after a rising edge of EN or if an error occurs while reading the status information or writing the Control Word and Reference Values.

READ_VALUES (array of read values)

Data type: ARRAY[1..12] OF INT

At output READ_VALUES the values of the array are updated after the read status information job was terminated successfully (DONE = TRUE, ERR = FALSE, BUSY = FALSE).

The read status information job is requested cyclically. It reads data from the ACS drive starting at Modbus register 400051 up to the number specified at input 3 + NVAR_READ

READ_VALUES contains the data as follows:

READ_VALUES[1] = <Modbus register 400054>

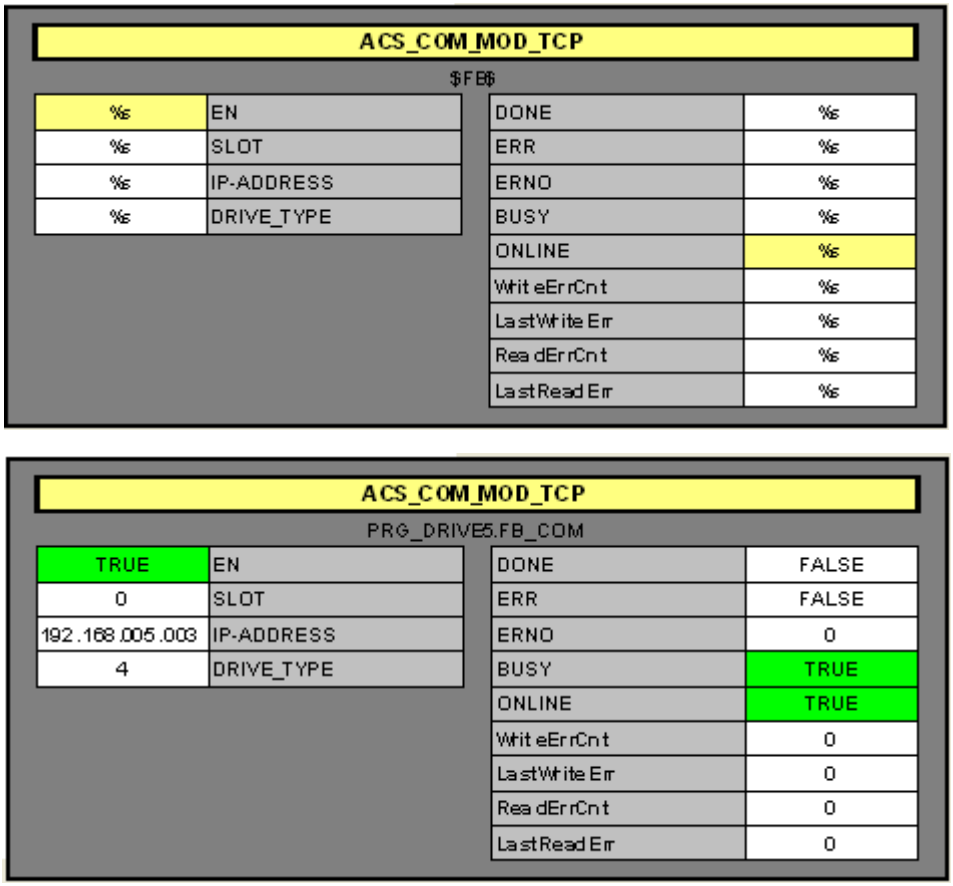
READ_VALUES[2] = <Modbus register 400055>

...

READ_VALUES[12] = <Modbus register 400065>.

1.5.6.5.4 Visualization

ACS_COM_MOD_TCPx_VISU_PH faceplate for the function block ACS_COM_MOD_TCPx



Visualization element ACS_COM_MOD_TCPx_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCPx *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385* function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

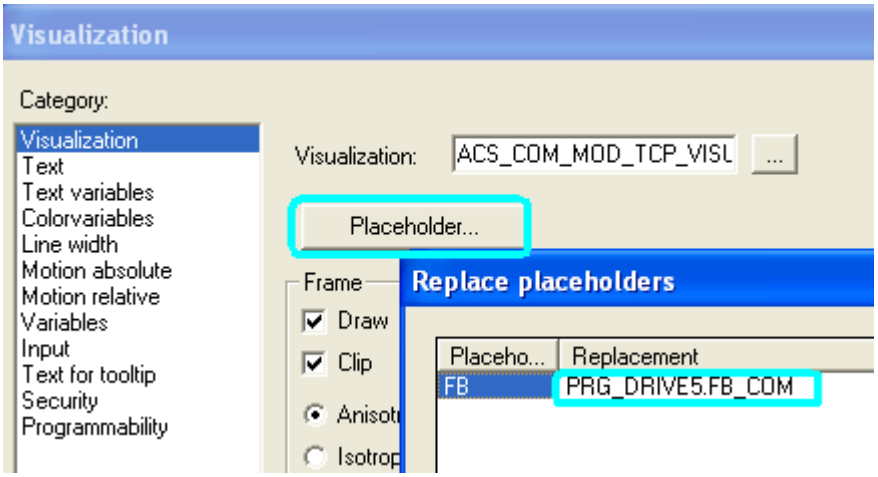
Visualization information

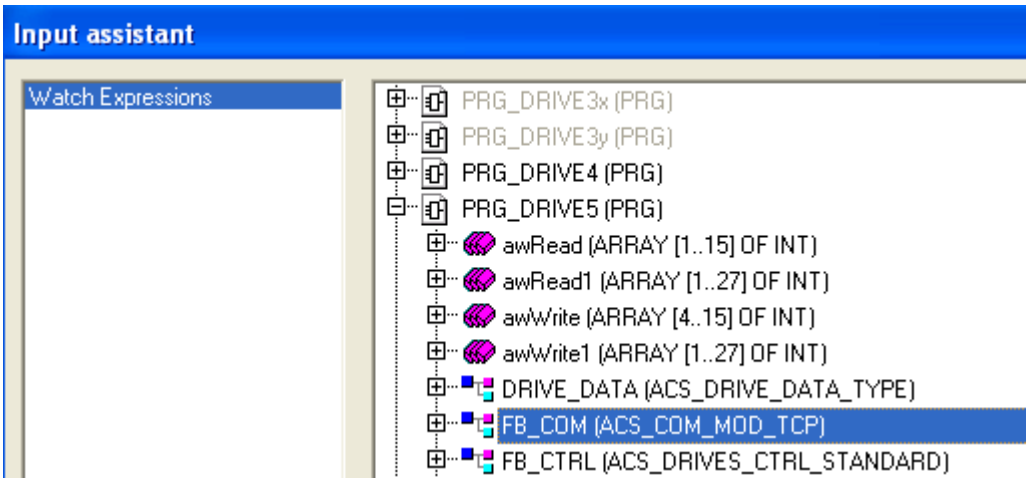
Available in runtime system:	V2.4
Included in library:	ACSDrivesComModTCP_Ext_AC500_V24.lib

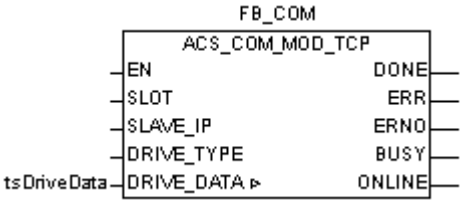
Visualization description

Visualization element ACS_COM_MOD_TCPx_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCPx *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385*function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_TCPx ↳ *Chapter 1.5.6.5.3.1 “ACS_COM_MOD_TCPx communication for ACS / DCS drives via Modbus TCP” on page 2385* function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

SLOT Access via: Numpad 1, 2, 11, 21, 31, 41
Description: SLOT (module number) of the communication module

IP_ADDRESS Access via: Text
Description: Slave address

DRIVE_TYPE Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drivel.FB_COM

Access R

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of ETH_MOD_MAST function block.

Colors

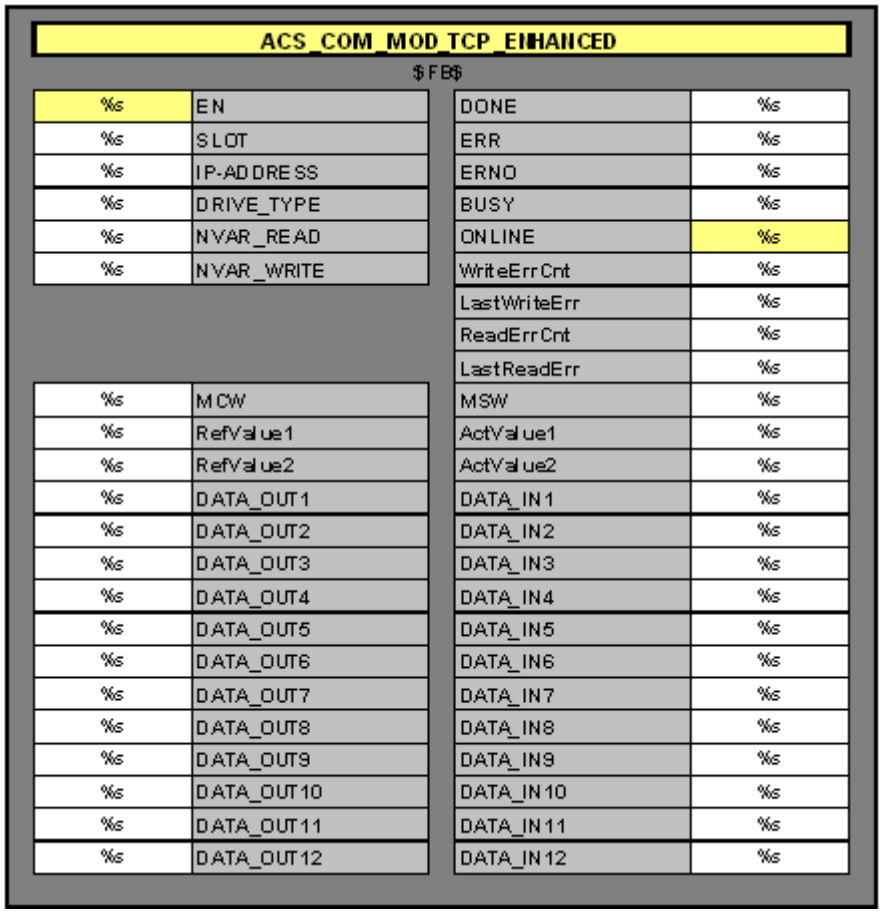
The color of the variables have the following meaning:

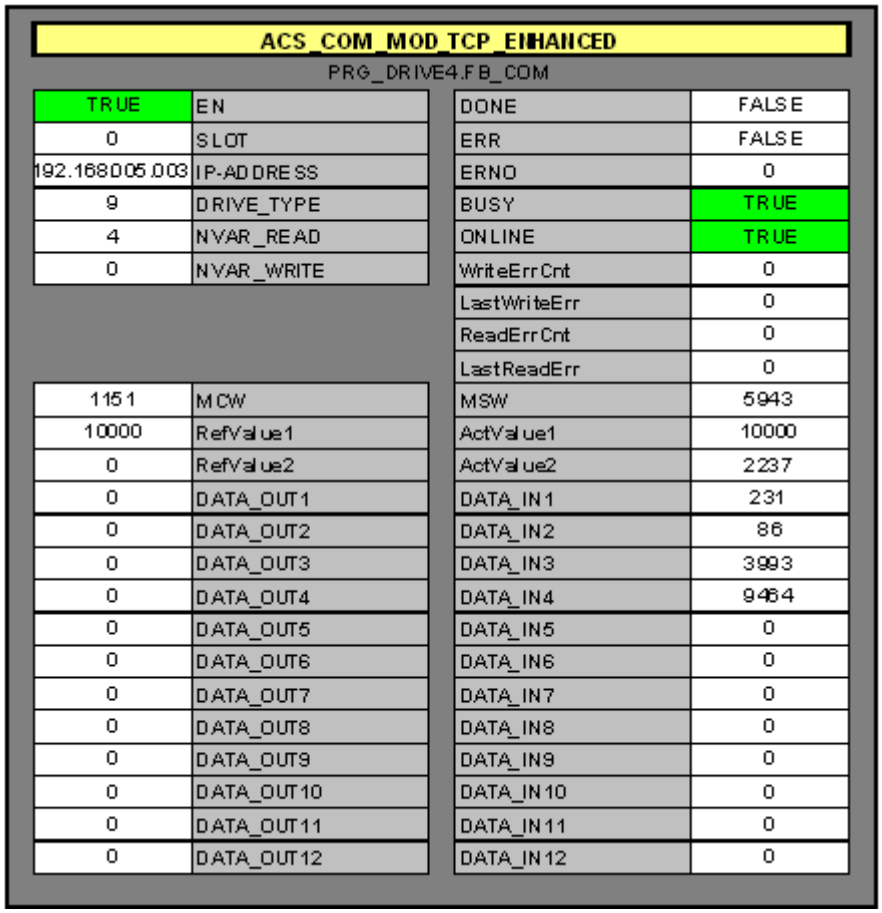
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_MOD_TCPx_ENHANCED_VISU_PH faceplate for the function block ACS_COM_MOD_TCPx_ENHANCED





Visualization element ACS_COM_MOD_TCPx_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392* function block which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

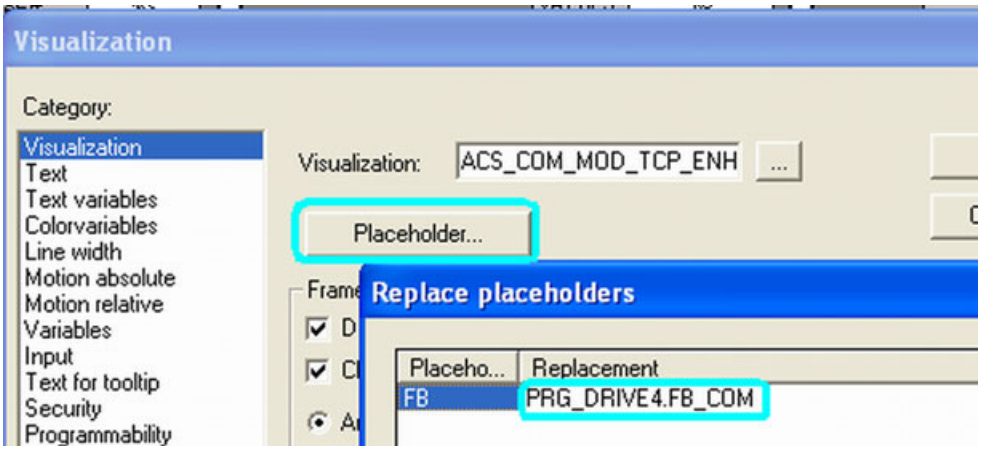
Visualization information

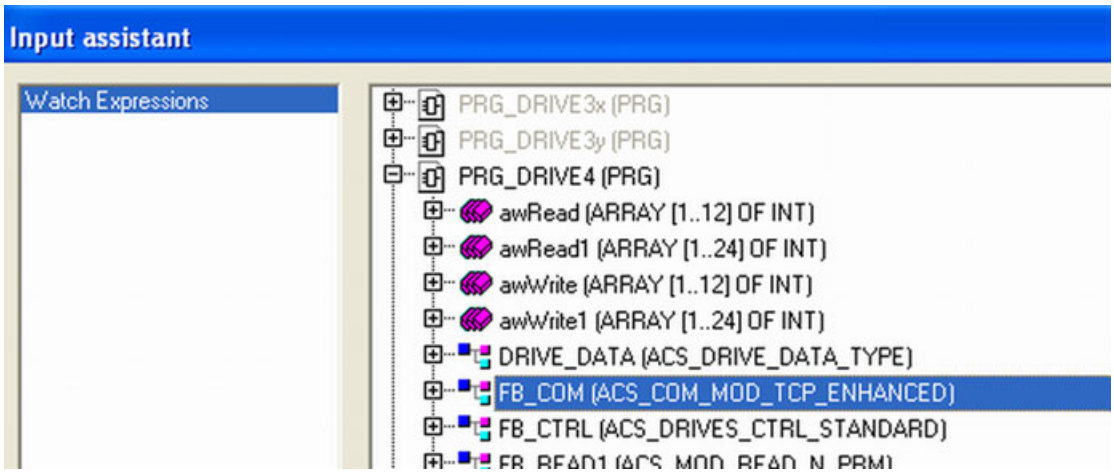
Available in runtime system:	V2.4.x
Included in library:	ACSDrivesCom-ModTCP_Ext_AC500_V24_App.lib

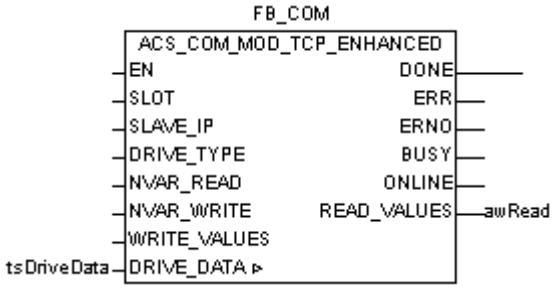
Visualization description

Visualization element ACS_COM_MOD_TCPx_ENHANCED_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an ACS_COM_MOD_TCPx_ENHANCED ↳ *Chapter 1.5.6.5.3.2 “ACS_COM_MOD_TCPx_ENHANCED communication for ACS / DCS drives via Modbus TCP” on page 2392* function block which instance was used to replace the placeholder FB.

All inputs of that ACS_COM_MOD_TCPx_ENHANCED function block, which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.







Parameters

Access R/W

EN
Access via: Toggle
Description: EN input

SLOT
Access via: Numpad 1, 2, 11, 21, 31, 41
Description: SLOT (module number) of the communication module

IP_ADDRESS
Access via: Text
Description: Slave address

DRIVE_TYPE
Access via: Numpad 1 ..17

Description: DRIVE_TYPE input

NVAR_READ Access via: Numpad 0 .. 12
Description: NVAR_READ input

NVAR_WRITE Access via: Numpad 0 .. 12
Description: NVAR_WRITE input

DATA_OUT1 Description: WRITE_VALUE[1] input

DATA_OUT2 Description: WRITE_VALUE[2] input

DATA_OUT3 Description: WRITE_VALUE[3] input

DATA_OUT4 Description: WRITE_VALUE[4] input

DATA_OUT5 Description: WRITE_VALUE[5] input

DATA_OUT6 Description: WRITE_VALUE[6] input

DATA_OUT7 Description: WRITE_VALUE[7] input

DATA_OUT8 Description: WRITE_VALUE[8] input

DATA_OUT9 Description: WRITE_VALUE[9] input

DATA_OUT10 Description: WRITE_VALUE[10] input

DATA_OUT11 Description: WRITE_VALUE[11] input

DATA_OUT12 Description: WRITE_VALUE[12] input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drivel.FB_COM

Access R

DONE Description: DONE output.

ERR Description: ERR output.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

ONLINE Description: ONLINE output

WriteErrCnt Description: Numbers of write errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastWriteErno Description: Error number of the last write job. See error messages of ETH_MOD_MAST function block.

ReadErrCnt Description: Numbers of read errors since EN = TRUE, is reset to 0 with rising edge of EN.

LastReadErno Description: Error number of the last read job. See error messages of ETH_MOD_MAST function block.

MCW Description: Control Word to the drive

RefValue1 Description: Reference value 1 to the drive

RefValue2 Description: Reference value 2 to the drive

MSW Description: Status Word of drive, READ_VALUE[1] output.

ActValue1 Description: Actual Value 1 mapped in Par. 53.10 - READ_VALUE[2] output

ACT_VALUE2 Description: ACT_VALUE2 output

DATA_IN1 Description: READ_VALUE[1] output

DATA_IN2 Description: READ_VALUE[2] output

DATA_IN3 Description: READ_VALUE[3] output

DATA_IN4 Description: READ_VALUE[4] output

DATA_IN5 Description: READ_VALUE[5] output

DATA_IN6 Description: READ_VALUE[6] output

DATA_IN7 Description: READ_VALUE[7] output

DATA_IN8 Description: READ_VALUE[8] output

DATA_IN9 Description: READ_VALUE[9] output

DATA_IN10 Description: READ_VALUE[10] output

DATA_IN11 Description: READ_VALUE[11] output

DATA_IN12 Description: READ_VALUE[12] output

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.6 ACS / DCS Drives communication via PROFIBUS

1.5.6.6.1 Preconditions for the use of the ACS / DCS drives communication via PROFIBUS library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks can only be used in combination with the ACSDrives-Base_AC500_V20 Library.

The library is released for the following products:

- CPUs: AC500
- Fieldbus: PROFIBUS
- Drives: ACS800, ACSM1, ACS355, ACS550, ACH550, ACQ810, ACS850, ACS880, ACS580, DCS550, DCS800
- PROFIBUS configuration:
  Prior to the use of the function blocks PROFIBUS Communication Module has to be configured accordingly using Automation Builder, at "Interfaces".

If the ACS_PB_READ_N_PRM_DPV1 function blocks are used at the same time the DriveManager is connecting, refreshing or writing variables, it might come to an error on either the ACS_PB_READ_N_PRM_DPV1 block or the DriveManager.

The DriveManager might be disconnected!

1.5.6.6.2 Components of the ACS / DCS drives communication via PROFIBUS library

Function blocks

ACS_COM_PB ↳ *Chapter 1.5.6.6.3.1 “ACS_COM_PB communication block via PROFIBUS” on page 2412*	Basic communication block for ABB / DCS drives via PROFIBUS DP using FBPA-01 or RPBA-01 communication modules including direct access for PZD1..3
ACS_COM_PB_PZD ↳ *Chapter 1.5.6.6.3.2 “ACS_COM_PB_PZD communication block for direct access to PZD4..12” on page 2415*	Communication block, which gives direct access for PZD4..12
ACS_PB_READ_PRM_DPV0 ↳ *Chapter 1.5.6.6.3.3 “ACS_PB_READ_PRM_DPV0 read parameters from ABB drives via PROFIBUS DPV0” on page 2420*	Read a parameter from drive via PROFIBUS DPV0 (using PKWs) - only for PPO-Types PPO 1, 2, 5 or 7
ACS_PB_WRITE_PRM_DPV0 ↳ *Chapter 1.5.6.6.3.4 “ACS_PB_WRITE_PRM_DPV0” on page 2423*	Write a parameter to drive via PROFIBUS DPV0 (using PKWs) - only for PPO-Types PPO 1, 2, 5 or 7
ACS_PB_READ_N_PRM_DPV1 ↳ *Chapter 1.5.6.6.3.5 “ACS_PB_N_READ_PRM_DPV1 read parameters from ABB drives via PROFIBUS DPV1” on page 2425*	This function block reads up to 37 parameters from an ACS / DCS drive via PROFIBUS DPV1 in a single query.
ACS_PB_WRITE_N_PRM_DPV1 ↳ *Chapter 1.5.6.6.3.6 “ACS_PB_N_WRITE_PRM_DPV1 write parameters from ABB drives via PROFIBUS DPV1” on page 2431*	This function block writes up to 37 parameters to an ACS / DCS drive via PROFIBUS DPV1 in a single query.

Visualizations

ACS_COM_PB_VISU_PH ↳ *Chapter 1.5.6.6.4.1 “ACS_COM_PB_VISU_PH visualization to run the ACS_COM_PB function block.” on page 2438*	Visualization to run the ACS_COM_PB function block.
ACS_COM_PB_PZD_VISU_PH ↳ *Chapter 1.5.6.6.4.2 “ACS_COM_PB_PZD_VISU_PH visualization to run the ACS_COM_PB_PZD function block.” on page 2440*	Visualization to run the ACS_COM_PB_PZD function block.
ACS_PB_READ_PRM_DPV0_VISU_PH ↳ *Chapter 1.5.6.6.4.3 “ACS_PB_READ_PRM_DPV0_VISU_PH visualization to run the ACS_PB_READ_PRM_DPV0 function block.” on page 2444*	Visualization to run the ACS_PB_READ_PRM_DPV0 function block.
ACS_PB_WRITE_PRM_DPV0_VISU_PH ↳ *Chapter 1.5.6.6.4.4 “ACS_PB_WRITE_PRM_DPV0_VISU_PH visualization to run the ACS_PB_WRITE_PRM_DPV0 function block.” on page 2445*	Visualization to run the ACS_PB_WRITE_PRM_DPV0 function block.

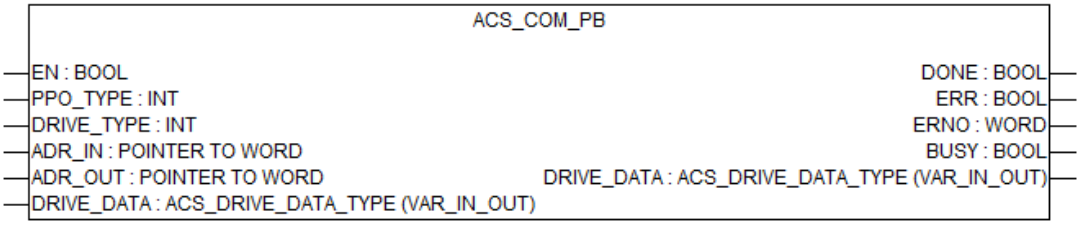
ACS_PB_READ_N_PRM_DPV1_VISU_PH *Chapter 1.5.6.6.4.5 "ACS_PB_READ_N_PRM_DPV1_VISU_PH visualization to run the ACS_PB_READ_N_PRM_DPV1 function block." on page 2447*	Visualization to run the ACS_PB_READ_N_PRM_DPV1 function block.
ACS_PB_WRITE_N_PRM_DPV1_VISU_PH *Chapter 1.5.6.6.4.6 "ACS_PB_WRITE_N_PRM_DPV1_VISU_PH visualization to run the ACS_PB_WRITE_N_PRM_DPV1 function block." on page 2449*	Visualization to run the ACS_PB_WRITE_N_PRM_DPV1 function block.

Global variables

ACS_VERSION_INFOR-MATION	Stores all the version information of the file along with the change log. No variable is declared inside this section.

1.5.6.6.3 Function blocks

ACS_COM_PB communication block via PROFIBUS



Function block ACS_COM_PB provides a PROFIBUS interface to the cyclic exchanged process data (PZD) of the PROFIBUS. It includes the handshake to READ/WRITE single parameters via PKWs variables for PPO-Types, that include PKWs (only in DPV0).

Function block information

Available in runtime system:	V2.4
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_PB provides a PROFIBUS interface to the cyclic exchanged process data (PZD) of the PROFIBUS. It includes the handshake to read / write single parameters via PKWs variables for PPO-Types that include PKWs (only in DPV0).

ACS and DCS drives offer different telegram types in their GSD file for cyclic data transmission. These different telegrams are so called Parameter Process Data Object (PPO). Each PPO contains a Process Data channel (PZD) and sometimes (only for DPV0) also a Parameter channel (PKW). (For more details refer to FPBA-01 or RPBA-01 PROFIBUS DP user manual.)

To READ / WRITE single parameters via PKWs, the separate blocks ACS_PB_READ_PRM_DPV0 and ACS_PB_WRITE_PRM_DPV0 can be used and therefore must be connected to the same DRIVE_DATA variable as the ACS_COM_PB.

For PROFIBUS the following function blocks can only be used together with this ACS_COM_PB block:

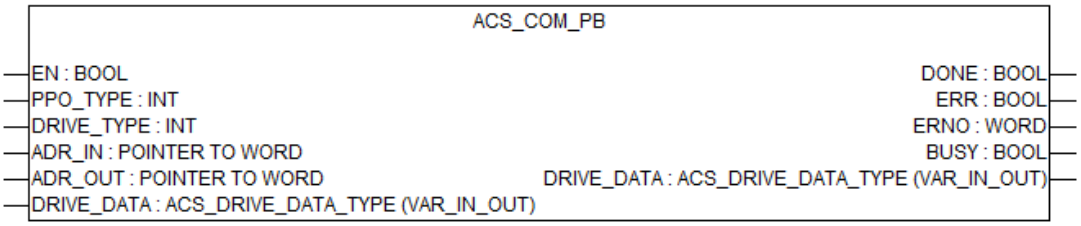
- ACS_COM_PB_PZD
- ACS_PB_READ_PRM_DPV0
- ACS_PB_WRITE_PRM_DPV0

- ACS_DRIVES_CTRL_STANDARD
- ACS_DRIVES_CTRL_ENG
- ACS3XX_DRIVES_CTRL_BASIC
- DCS_DRIVES_CTRL

In the Automation Builder configuration one of the PPO types 1..8 with predefined number of PKW and PZDs has to be selected. ADR_IN and ADR_OUT inputs of the function block need to be connected to the address of the first input process variable, and to the address of the first output variable respectively.

> *For parameter setting inside drive, please refer the respective drives / fieldbus module manual for parameter setting.*

Input description



> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN (enable)

Data type: BOOL

It enables function block processing.

The function block is not processed, if input EN = FALSE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

PPO_TYPE (parameter process data object type)

Data type: INT

At the input PPO_TYPE the configured PPO-Type must be set.

E.g., if PPO-Type 5 is configured the input PPO_TYPE must be set to 5.

DRIVE_TYPE (drive type)

Data type: ENUM, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM or INT. (Please refer ACS_DRIVE_ENUM for details.) The input can be set either by the value directly or by using the enum.

ADR_IN (ADR_IN)

Data type: POINTER TO WORD

Address of the first process input variable (PZD).

(PKW0 or PZD0 if no PKW is available in the PPO-Type).

With this address and the correct input PPO_TYPE, all other variables can be accessed directly via the function block ACS_COM_PB_PZD and a drives control block (e.g. ACS_DRIVES_CTRL_STANDARD, DCS_DRIVES_CTRL, ACS_DRIVES_CTRL_ENG, ACS3XX_DRIVES_CTRL_BASIC). So the configuration of input and output variables can be reduced to the first variable only.

ADR_OUT (ADR_OUT)

Data type: POINTER TO WORD

Address of the first process output variable (PZD).

(PKW0 or PZD0 if no PKW is available in the PPO-Type).

With this address and the correct input PPO_TYPE, all other variables can be accessed directly via the function block ACS_COM_PB_PZD and a drives control block (e.g. ACS_DRIVES_CTRL_STANDARD, DCS_DRIVES_CTRL, ACS_DRIVES_CTRL_ENG, ACS3XX_DRIVES_CTRL_BASIC). So the configuration of input and output variables can be reduced to the first variable only.

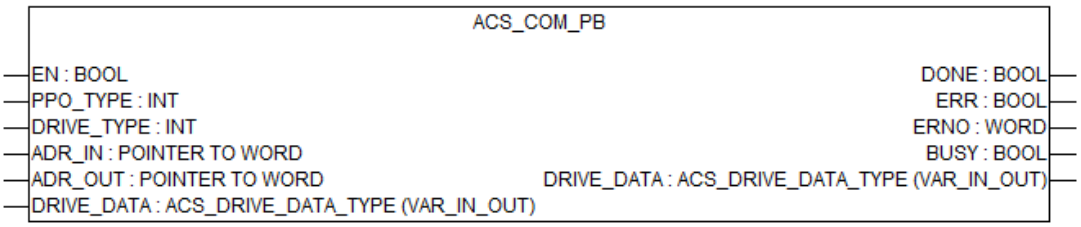
DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↳ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the processing state of the block.

If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY (busy) Data type: BOOL

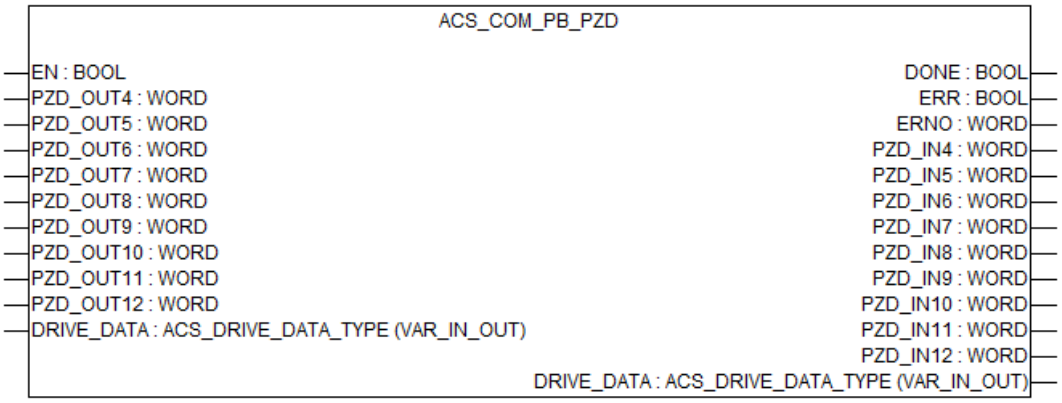
Output BUSY = TRUE is indicating that the function block is enabled.

In case of an error of this block, the BUSY output will be set to FALSE while ERR = TRUE. In case of a read / write error of a function block that is attached to the DRIVE_DATA variable, the BUSY will stay TRUE and ERR = TRUE.

Calling of ACS_COM_PB in ST

```
AcsComPb (EN                                    := xAcsComPb_EN,

PPO_TYPE              := iAcsComPb_Ppo_Type,

DRIVE_TYPE         := iAcsComPb_Drive_Type,

ADR_IN                          := ADR(pAcsComPb_AdrIn),

ADR_OUT                   := ADR(pAcsComPb_AdrOut),

                                                DRIVE_DATA      :=
AcsComPb_DriveData);

xAcsComPb_Done     := AcsComPb.DONE;
xAcsComPb_Err         := AcsComPb.ERR;
wAcsComPb_Erno     := AcsComPb.ERNO;
xAcsComPb_Busy  := AcsComPb.Busy
```

ACS_COM_PB_PZD communication block for direct access to PZD4..12



Function block ACS_COM_PB_PZD gives direct access to the PZD 4 .. 12. It must be used together with ACS_COM_PB function block.

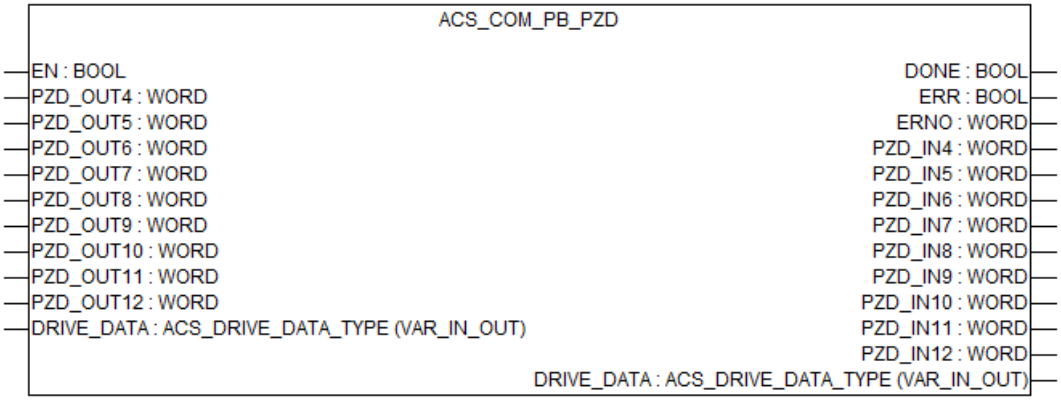
Function block information

Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_COM_PB_PZD gives direct access to the PZD 4 .. 12. User can use PZD 4 to 12 IN and OUT to read and write PZD parameters with ease. Therefore the DRIVE_DATA variable must be connected to the related ACS_COM_PB function block.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block is deactivated (EN = FALSE), then all outputs are reset to zero.

PZD_OUT4 (PZD output 4)

Data type: WORD

Process Data word to drive

Available with PPO_Type: 2,4,5,6,7,8

PZD_OUT5 (PZD output 5)

Data type: WORD

Process Data word to drive

Available with PPO_Type: 2,4,5,6,7,8

PZD_OUT6 (PZD output 6)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 2,4,5,6,7,8

PZD_OUT7 (PZD output 7)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_OUT8 (PZD output 8)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_OUT9 (PZD output 9)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_OUT10 (PZD output 10)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_OUT11 (PZD output 11)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 7,8

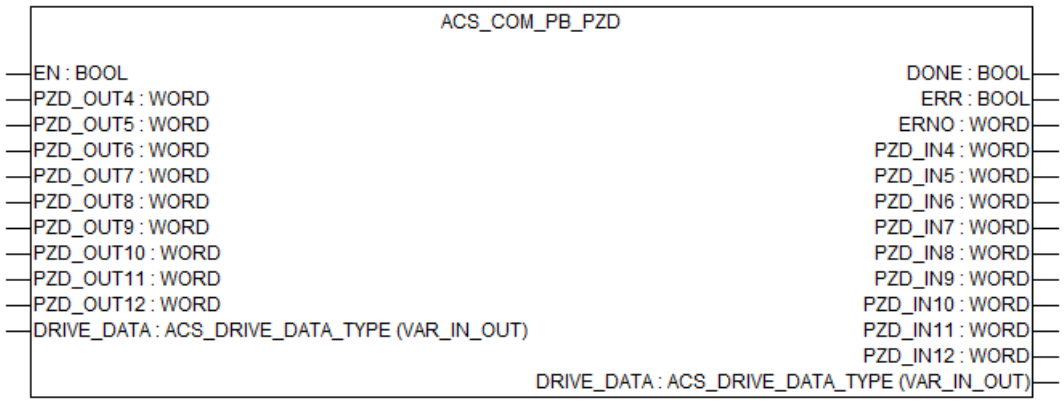
PZD_OUT12 (PZD output 12)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 7,8

DRIVE_DATA (drive data)
Data type: ACS_DRIVE_DATA_TYPE ↯ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the processing state of the function block.

The DONE remains TRUE as long as the EN is TRUE.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PZD_IN4 (PZD input 4)

Data type: WORD

Process Data word to drive

Available with PPO_Type: 2,4,5,6,7,8

PZD_IN5 (PZD input 5)

Data type: WORD

Process Data word to drive

Available with PPO_Type: 2,4,5,6,7,8

PZD_IN6 (PZD input 6)

Data type: WORD

Process Data word to drive

Available with PPO_Type: 2,4,5,6,7,8

PZD_IN7 (PZD input 7)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_IN8 (PZD input 8)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_IN9 (PZD input 9)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_IN10 (PZD input 10)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 5,6,7,8

PZD_IN11 (PZD input 11)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 7,8

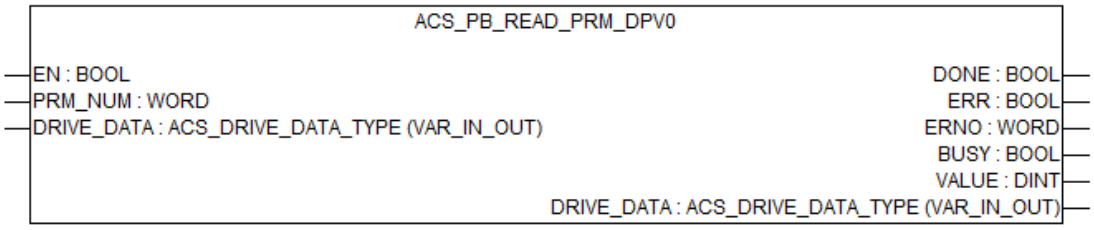
PZD_IN12 (PZD input 12)
Data type: WORD
Process Data word to drive
Available with PPO_Type: 7,8

Calling of ACS_COM_PB_PZD in ST

```
AcsComPbPzd  ( EN                                   := xAcsComPbPzd_En,

PZD_OUT4          := wAcsComPbPzd_PzdOut4,

PZD_OUT5          := wAcsComPbPzd_PzdOut5,

PZD_OUT6          := wAcsComPbPzd_PzdOut6,

PZD_OUT7          := wAcsComPbPzd_PzdOut7,

PZD_OUT8          := wAcsComPbPzd_PzdOut8,

PZD_OUT9          := wAcsComPbPzd_PzdOut9,

PZD_OUT10     := wAcsComPbPzd_PzdOut10,
```

```
PZD_OUT11      := wAcsComPbPzd_PzdOut11,

PZD_OUT12      := wAcsComPbPzd_PzdOut12,

DRIVE_DATA := AcsComPbPzd_DriveData);

xAcsComPbPzd_Done              := AcsComPbPzd.DONE;
xAcsComPbPzd_Err                     := AcsComPbPzd.ERR;
wAcsComPbPzd_Erno              := AcsComPbPzd.ERNO;
wAcsComPbPzd_PzdIn4          := AcsComPbPzd.PZDIN4 ;
wAcsComPbPzd_PzdIn5          := AcsComPbPzd.PZDIN5 ;
wAcsComPbPzd_PzdIn6          := AcsComPbPzd.PZDIN6 ;
wAcsComPbPzd_PzdIn7          := AcsComPbPzd.PZDIN7 ;
wAcsComPbPzd_PzdIn8          := AcsComPbPzd.PZDIN8 ;
wAcsComPbPzd_PzdIn9          := AcsComPbPzd.PZDIN9 ;
wAcsComPbPzd_PzdIn10     := AcsComPbPzd.PZDIN10 ;
wAcsComPbPzd_PzdIn11     := AcsComPbPzd.PZDIN11 ;
wAcsComPbPzd_PzdIn12     := AcsComPbPzd.PZDIN12 ;
```

ACS_PB_READ_PRM_DPV0 read parameters from ABB drives via PROFIBUS DPV0



Function block ACS_PB_READ_PRM_DPV0 is used to read parameters from ACS / DCS drive using PROFIBUS communication via PKW. It must be used together with ACS_COM_PB function block.

Function block information

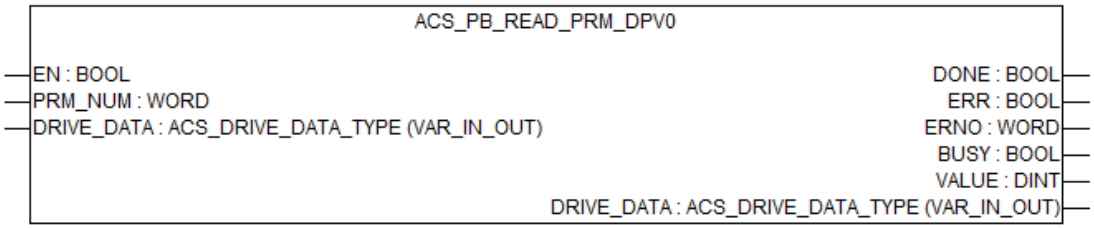
Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

If the ACS_PB_READ_N_PRM_DPV1 function block is used at the same time the DriveManager is connecting, refreshing or writing variables it might come to an error on either the ACS_PB_READ_N_PRM_DPV1 block or the DriveManager.

Block description

Function block ACS_PB_READ_PRM_DPV0 reads one drive parameter using PROFIBUS DPV0 communication via PKW especially for ABB Drives. This function block works only with PPO types that contains the PKW part (e.g. PPO 1, 2, 5 or 7.)

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

It enables function block processing.

The function block is not processed if input EN = FALSE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

PRM_NUM (parameter number)

Data type: WORD

Input PRM_NUM is the parameter number in the drive which will be read. The parameter number is calculated from group number and index.

Parameter: 2 digits = group, 2 digit = index, e.g. Par 20.06 = 2006

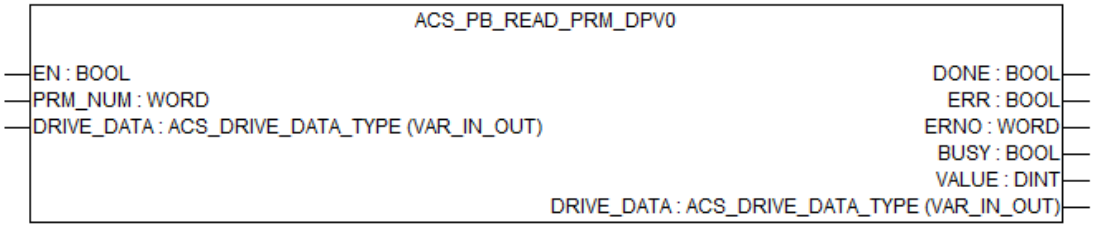
DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

Output description



DONE (done) Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY (busy) Data type: BOOL

Output BUSY = TRUE indicates, that the read job requested by the function block is being processed.

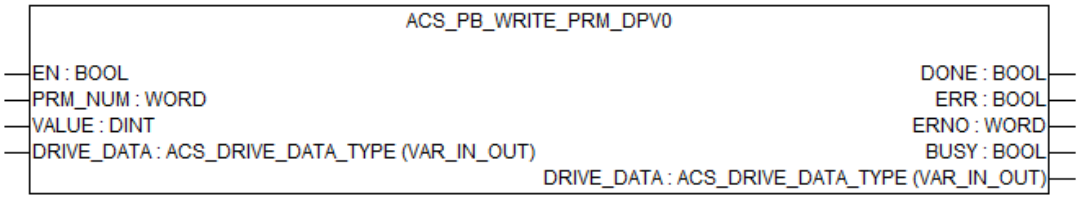
VALUE (value) Data type: DINT

Parameter value of the READ parameter

Calling of ACS_PB_READ_PRM_DPV0 in ST

```
AcsPBReadPrmDpv0     (EN                                  :=
xAcsPBReadPrmDpv0_En,


            PRM_NUM             := wAcsPBReadPrmDpv0_PrmNum,


            DRIVE_DATA := AcsPBReadPrmDpv0_DriveData);

xAcsPBReadPrmDpv0_Done            := AcsPBReadPrmDpv0.DONE;
xAcsPBReadPrmDpv0_Err                := AcsPBReadPrmDpv0.ERR;
xAcsPBReadPrmDpv0_Busy            := AcsPBReadPrmDpv0.BUSY;
diAcsPBReadPrmDpv0_Value    := AcsPBReadPrmDpv0.VALUE;
```

ACS_PB_WRITE_PRM_DPV0



Function block ACS_PB_WRITE_PRM_DPV0 is used to write parameters to ACS / DCS drive using PROFIBUS communication. It must be used together with ACS_COM_PB function block

Function block information

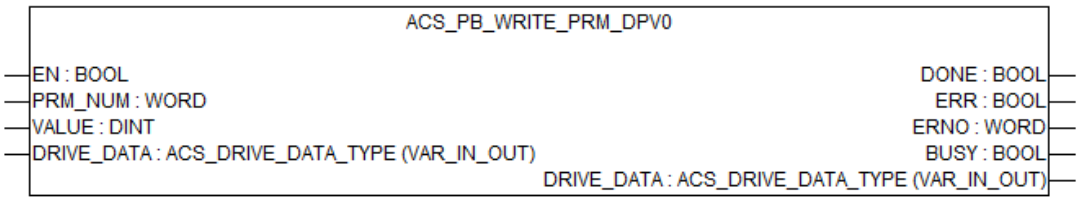
Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_PB_WRITES_PRM_DPV0 writes one drive parameter using PROFIBUS DPV0 communication via PKW especially for ABB Drives.

This function block works only with PPO types that contain the PKW part (e.g. PPO 1, 2, 5 or 7.)

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

It enables function block processing (edge sensitive).

The function block is not processed if input EN = FALSE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

PRM_NUM (parameter number)

Data type: WORD

Input PRM_NUM is the parameter number in the drive which will be read. The parameter number is calculated from group number and index.

Parameter: 2 digits = group, 2 digit = index, e.g. Par 20.06 = 2006

VALUE (value)

Data type: DINT

Parameter value of the READ parameter

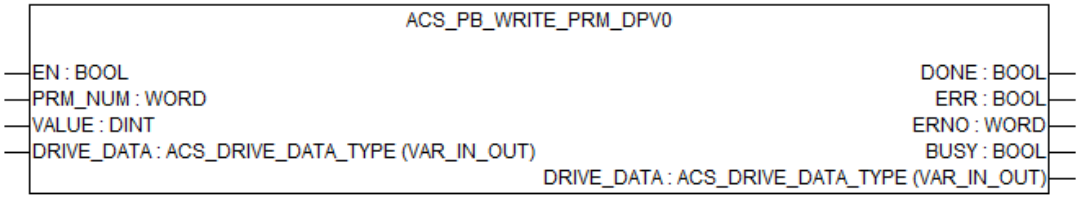
DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↬ *Chapter 1.5.6.2.6.2 "ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive" on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY (busy)

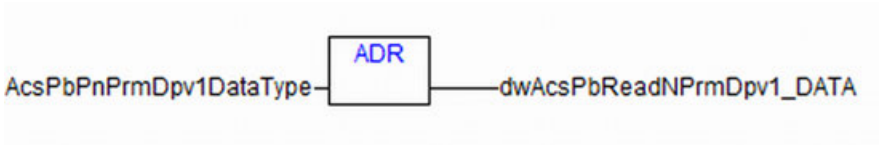
Data type: BOOL

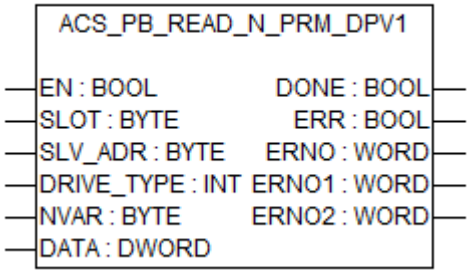
Output BUSY is not used at the moment.

Calling of ACS_PB_WRITE_PRM_DPV0 in ST

```
AcsPBWritePrmDpv0 (EN                                    :=
xAcsPBWritePrmDpv0_En,

            PRM_NUM              := wAcsPBWritePrmDpv0_PrmNum,

            VALUE                      := diAcsPBWritePrmDpv0_Value,

            DRIVE_DATA := AcsPBWritePrmDpv0_DriveData);

xAcsPBWritePrmDpv0_Done        := AcsPBWritePrmDpv0.DONE;
xAcsPBWritePrmDpv0_Err            := AcsPBWritePrmDpv0.ERR;
wAcsPBWritePrmDpv0_Erno     := AcsPBWritePrmDpv0.ERNO;
xAcsPBWritePrmDpv0_Busy        := AcsPBWritePrmDpv0.BUSY;
```

ACS_PB_N_READ_PRM_DPV1 read parameters from ABB drives via PROFIBUS DPV1





Function block ACS_PB_READ_N_PRM_DPV1 is used for reading maximum 37 parameters from a drive via PROFIBUS DPV1 in a single query.

Function block information

Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

> *If the ACS_PB_READ_N_PRM_DPV1 function block is used at the same time the DriveManager is connecting, refreshing or writing variables, it might come to an error on either the ACS_PB_READ_N_PRM_DPV1 block or the DriveManager.*

Block description

Function block ACS_PB_READ_N_PRM_DPV1 reads maximum 37 parameters from the drive in a single query. The number of parameters to be read is specified at the input NVAR. Parameters to read to the drive is specified at the DATA input.

ACS_PB_PN_PRM_DPV1_DATA_TYPE structure must be declared to a variable and connected to DATA input using ADR, which is to be entered with Group, Index .

Read parameter type and values are stored in the same variable.

ACS_PB_PN_PRM_DPV1_DATA_TYPE structure has the following array elements:

- abyPrmGroup : Array of 37 BYTE for specifying parameter Group.
- abyPrmIndex : Array of 37 BYTE for specifying parameter Index
- abyPrmType : Array of 37 BYTE
  READ parameter data type will be available here. For details refer to ACS_PB_PN_PRM_TYPE_ENUM *Chapter 1.5.6.2.5.2 “ACS_PB_PN_PRM_TYPE_ENUM” on page 2251*
- adwPrmValue: Array of 37 DWORD
  READ parameter value will be available here.

> *Currently user cannot use enumeration from ACS_PB_PN_PRM_TYPE_ENUM.*
>
> *Instead user need to use numerical values from ACS_PB_PN_PRM_TYPE_ENUM only.*

The values in the structure area are updated, when the READ job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Example

If the user need to read Parameter 10.01 and 11.05, then user need to enter the group and index number separately in input DATA as below:

DataStructure : ACS_PB_PN_PRM_DPV1_DATA_TYPE

Parameter Group : DataStruct.abyPrmGroup[1]:= 10; DataStruct.abyPrmGroup[2]:= 11;

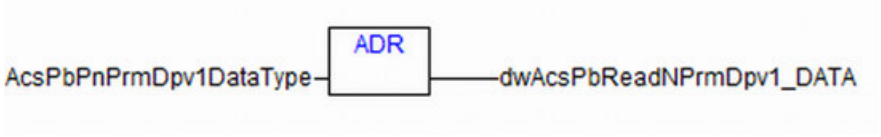
Parameter Index : DataStruct .abyPrmIndex [1]:= 01; DataStruct .abyPrmIndex [2]:= 05;

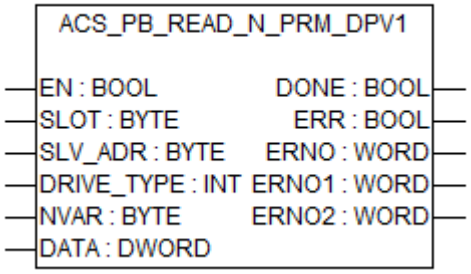
Parameter Type : Once read operation is complete parameter type will be stored inside DataStruct. abyPrmType [1]:= xx; DataStruct. abyPrmType [2]:= xx;

Parameter Value : Value will be stored inside DataStruct. adwPrmValue [1]:= xx; DataStruct. adwPrmValue [2]:= xx;

The values in the structure area are updated once the read job is performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Input description





EN (enable)

Data type: BOOL

In order to enable the function block processing, input EN has to be set from FALSE to TRUE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

Default value = FALSE.

> *If multiple ACS_PB_READ_N_PRM_DPV1 and / or multiple ACS_PB_WRITE_N_PRM_DPV1 functions blocks enabled at the same time it may cause for error Read / Write error.*

SLOT (slot)

Data type: BYTE, Default value: 1, Range: 1 to 6

At input SLOT the communication module SLOT (module number) is selected, which should be used by the function block. All external Communication Modules are serially numbered from right to left, starting with module number 1.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

If the SLOT number is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4020.

SLV_ADR (slave address)

Data type: BYTE, Default value: 1, Range: 1 to 126

At input SLV_ADR, the address of the drive (slave), from which the parameter value is to be read, must be specified.

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed, while the program is running. If changed, nevertheless the new value will become effective, only after the function block is enabled again.

If the SLV_ADR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4030.

DRIVE_TYPE (drive type)

Data type: INT, Default value = ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM (enumeration). The input can be set either by the value directly or by using the enum ↳ *Chapter 1.5.6.2.5.1 “ACS_DRIVE_ENUM enumerations to select the type of drive used” on page 2251*.

If the DRIVE_TYPE is given invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

NVAR (number of variables)

Data type: BYTE, Default value: 0, Range: 0 to 37

With input NVAR the function block can be configured to read between 0 to 37 drive parameter values.

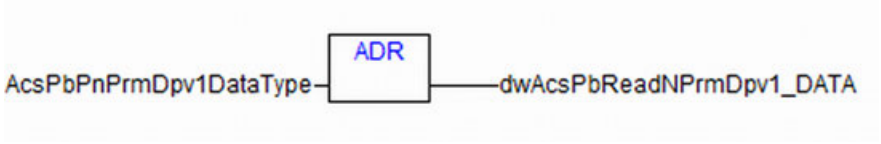
If the NVAR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

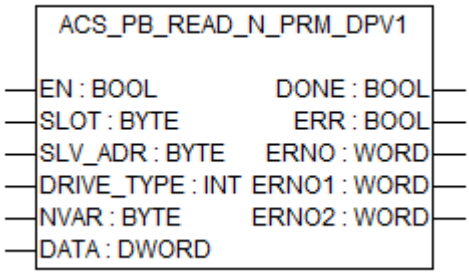
DATA (data)

Data type: DWORD

Input Data must be connected to the variable of type ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index. Each drive must have its own DATA variable.

Output description





DONE (done)

Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1)

Data type: WORD, Default value: 0, Range: ≥0

Output ERNO1 provides additional error information in case an error occurred during processing. ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

The encoding of the error messages output at ERNO is explained in a separate table "Error Messages" (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Encoding of ERNO1 Parameter error messages

ERNO1 of the DPV1 function blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

Error class				Error code			
Bit: 7	6	5	4	3	2	1	0
ERNO1							
DEC	HEX			Error class / Error code			
0	0			Reserved			
...	...						
159	9F			Reserved			
160	A0			10 Application / 0 Read error			
161	A1			10 Application / 1 Write error			
162	A2			10 Application / 2 Error module			
163	A3			Reserved			
...	...			...			
167	A7			Reserved			
168	A8			10 Application / 8 Version conflict			
169	A9			10 Application / 9 Function not supported			
170	AA			10 Application / 10 Manufacturer-specific			
...	...			...			
175	AF			10 Application / 15 Manufacturer-specific			
176	B0			11 Access / 0 Invalid index			
177	B1			11 Access / 1 Invalid length of data to be written			
178	B2			11 Access / 2 Invalid slot			
179	B3			11 Access / 3 Type conflict			
180	B4			11 Access / 4 Invalid range			
181	B5			11 Access / 5 Status conflict			
182	B6			11 Access / 6 Access denied			
183	B7			11 Access / 7 Invalid value range			
184	B8			11 Access / 8 Invalid parameter			
185	B9			11 Access / 9 Invalid type			
186	BA			11 Access / 10 Manufacturer-specific			
...	...			...			
191	BF			11 Access / 15 Manufacturer-specific			

192	C0			12 Resources / 0 Read conflict
193	C1			12 Resources / 1 Write conflict
194	C2			12 Resources / 2 Resource used
195	C3			12 Resources / 3 Resource not available
196	C4			Reserved
...	...			...
199	C7			Reserved
200	C8			12 Resources / 10 Manufacturer-specific
...	...			...
207	CF			12 Resources / 15 Manufacturer-specific
208	D0			Reserved
...	...			...
255	FF			Reserved

ERNO2 (error number 2)

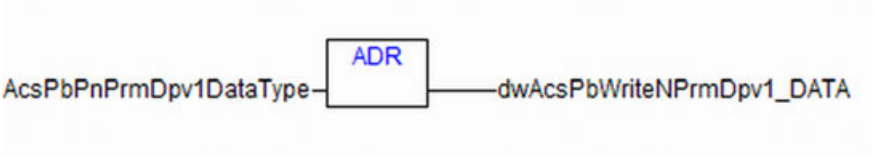
Data type: WORD, Default value: 0, Range: ≥0

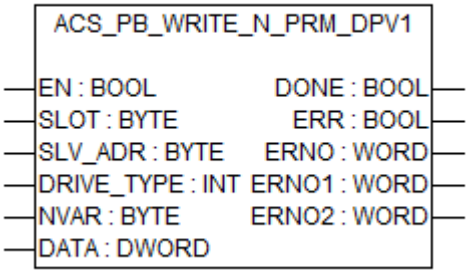
Output ERNO2 provides an additional DPV1-specific error information, if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC). The encoding of ERNO2 is completely manufacturer-specific.

Calling of ACS_PB_READ_N_PRM_DPV1 in ST

```
dwAcsPbReadNPrmDpv1_DATA:= ADR (AcsPbPnPrmDpv1DataType)
ACS_PB_READ_N_PRM_DPV1
(EN                              := xAcsPbReadNPrmDpv1_EN,


SLOT                           := byAcsPbReadNPrmDpv1_SLOT,

                                          SLV_ADR              :=
byAcsPbReadNPrmDpv1_SLV_ADR,

                                          DRIVE_TYPE :=
iAcsPbReadNPrmDpv1_DRIVE_TYPE,


NVAR                           := byAcsPbReadNPrmDpv1_NVAR,


DATA                           := dwAcsPbReadNPrmDpv1_DATA);

xAcsPbReadNPrmDpv1_DONE          := ACS_PB_READ_N_PRM_DPV1.DONE;
xAcsPbReadNPrmDpv1_ERR              := ACS_PB_READ_N_PRM_DPV1.ERR;
wAcsPbReadNPrmDpv1_ERNO          := ACS_PB_READ_N_PRM_DPV1.ERNO;
wAcsPbReadNPrmDpv1_ERNO1      := ACS_PB_READ_N_PRM_DPV1.ERNO1;
wAcsPbReadNPrmDpv1_ERNO2      := ACS_PB_READ_N_PRM_DPV1.ERNO2;
```

ACS_PB_N_WRITE_PRM_DPV1 write parameters from ABB drives via PROFIBUS DPV1





Function block ACS_PB_WRITE_N_PRM_DPV1 is used for writing maximum 37 parameters to a drive via PROFIBUS in a single query.

Function block information

Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPB_AC500_V24.lib
Function block type:	Function block with historical values.

If the ACS_PB_WRITE_N_PRM_DPV1 function block is used at the same time the DriveManager is connecting, refreshing or writing variables, it might come to an error on either the ACS_PB_WRITE_N_PRM_DPV1 block or the DriveManager.

Block description

Function block ACS_PB_WRITE_N_PRM_DPV1 writes maximum 37 parameters to drive in a single query. The number of parameters to be write is specified at the input NVAR.

Another limit while using the WRITE function block is, it can process only up to 240 byte data in one request or 37 drive parameters whichever is lower. If the WRITE data length is more than 240 byte, the function block generates error code 16#7012.

Parameters to write to the drive is specified at the DATA input.
ACS_PB_PN_PRM_DPV1_DATA_TYPE structure must be declared to a variable and connected to DATA input using ADR, which is to be entered with Group, Index, types and values.

ACS_PB_PN_PRM_DPV1_DATA_TYPE structure has the following array elements:

- abyPrmGroup : Array of 37 BYTE for specifying parameter Group.
- abyPrmIndex : Array of 37 BYTE for specifying parameter Index
- abyPrmType : Array of 37 BYTE for specifying parameter type, please refer the respective drives manual for parameter data type and enter the respective ENUM / VALUE . For details about ENUM / VALUE please refer ACS_PB_PN_PRM_TYPE_ENUM *Chapter 1.5.6.2.5.2 “ACS_PB_PN_PRM_TYPE_ENUM” on page 2251*
- adwPrmValue: Array of 37 BYTE for specifying parameter value

Currently user cannot use enumeration from ACS_PB_PN_PRM_TYPE_ENUM.

Instead user need to use numerical values from ACS_PB_PN_PRM_TYPE_ENUM only.

The values in the structure area are updated, when the WRITE job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Example

If the user need to read Parameter 10.01 and 11.05, then user need to enter the group and index number separately in input DATA as below:

DataStructure : ACS_PB_PN_PRM_DPV1_DATA_TYPE

Parameter Group : DataStruct.abyPrmGroup[1]:= 10; DataStruct.abyPrmGroup[2]:= 11;

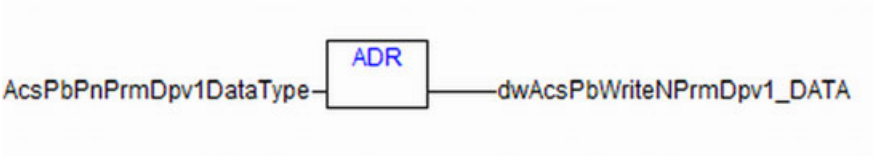
Parameter Index : DataStruct .abyPrmIndex [1]:= 01; DataStruct .abyPrmIndex [2]:= 05;

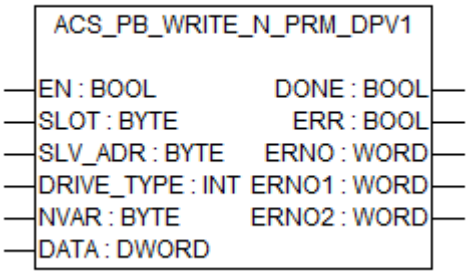
Parameter Type : DataStruct. abyPrmType [1]:= xx; DataStruct. abyPrmType [2]:= xx; . . DataStruct. abyPrmType [37]:= xx;

Parameter Value : And the value will be stored inside DataStruct. adwPrmValue [1]:= xx; DataStruct. adwPrmValue [2]:= xx;

The values in the structure area are written to drive when the write job is performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Input description





EN (enable)

Data type: BOOL

In order to enable the function block processing, input EN has to be set from FALSE to TRUE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

Default value = FALSE.

If multiple ACS_PB_READ_N_PRM_DPV1 and / or multiple ACS_PB_WRITE_N_PRM_DPV1 functions blocks enabled at the same time it may cause for error Read / Write error.

SLOT (slot)

Data type: BYTE, Default value: 1, Range: 1 to 6

At input SLOT the communication module SLOT (module number) is selected, which should be used by the function block. All external Communication Modules are serially numbered from right to left, starting with module number 1.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

If the SLOT number is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4020.

SLV_ADR (slave address)

Data type: BYTE, Default value: 1, Range: 1 to 126

At input SLV_ADR, the address of the drive (slave), from which the parameter value is to be read, must be specified.

The function block is designed to be used with a fix SLAVE device. The SLAVE input should not be changed, while the program is running. If changed, nevertheless the new value will become effective, only after the function block is enabled again.

If the SLV_ADR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4030.

DRIVE_TYPE (drive type)

Data type: INT, Default value = ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM (enumeration). The input can be set either by the value directly or by using the enum ↳ *Chapter 1.5.6.2.5.1 “ACS_DRIVE_ENUM enumerations to select the type of drive used” on page 2251*.

If the DRIVE_TYPE is given invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

NVAR (number of variables)

Data type: BYTE, Default value: 0, Range: 0 to 37

With input NVAR the function block can be configured to read between 0 to 37 drive parameter values.

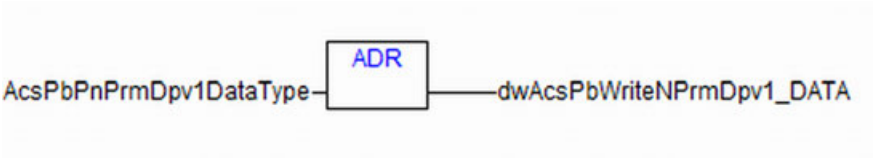
If the NVAR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

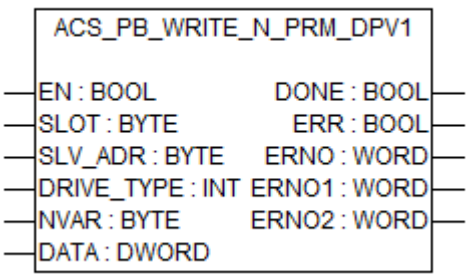
DATA (data)

Data type: DWORD

Input Data must be connected to the variable of type ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index. Each drive must have its own DATA variable.

Output description





DONE (done)

Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table of Error Messages (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1)

Data type: WORD, Default value: 0, Range: ≥0

Output ERNO1 provides additional error information in case an error occurred during processing. ERNO1 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

The encoding of the error messages output at ERNO is explained in a separate table "Error Messages" (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

Encoding of ERNO1 Parameter error messages

ERNO1 of the DPV1 function blocks is encoded as follows. The upper nibble (the higher significant 4 bits) describes the error class, the lower nibble represents the error cause.

Error class				Error code			
Bit: 7	6	5	4	3	2	1	0
ERNO1							
DEC	HEX			Error class / Error code			
0	0			Reserved			
...	...						
159	9F			Reserved			
160	A0			10 Application / 0 Read error			
161	A1			10 Application / 1 Write error			
162	A2			10 Application / 2 Error module			
163	A3			Reserved			
...	...			...			

167	A7			Reserved
168	A8			10 Application / 8 Version conflict
169	A9			10 Application / 9 Function not supported
170	AA			10 Application / 10 Manufacturer-specific
...	...			...
175	AF			10 Application / 15 Manufacturer-specific
176	B0			11 Access / 0 Invalid index
177	B1			11 Access / 1 Invalid length of data to be written
178	B2			11 Access / 2 Invalid slot
179	B3			11 Access / 3 Type conflict
180	B4			11 Access / 4 Invalid range
181	B5			11 Access / 5 Status conflict
182	B6			11 Access / 6 Access denied
183	B7			11 Access / 7 Invalid value range
184	B8			11 Access / 8 Invalid parameter
185	B9			11 Access / 9 Invalid type
186	BA			11 Access / 10 Manufacturer-specific
...	...			...
191	BF			11 Access / 15 Manufacturer-specific
192	C0			12 Resources / 0 Read conflict
193	C1			12 Resources / 1 Write conflict
194	C2			12 Resources / 2 Resource used
195	C3			12 Resources / 3 Resource not available
196	C4			Reserved
...	...			...
199	C7			Reserved
200	C8			12 Resources / 10 Manufacturer-specific
...	...			...
207	CF			12 Resources / 15 Manufacturer-specific
208	D0			Reserved
...	...			...
255	FF			Reserved

ERNO2 (error number 2)

Data type: WORD, Default value: 0, Range: ≥0

Output ERNO2 provides an additional DPV1-specific error information, if an error occurred during processing. ERNO2 always has to be considered together with the outputs DONE, ERR and ERNO. The value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC). The encoding of ERNO2 is completely manufacturer-specific.

Calling of ACS_PB_WRITE_N_PRM_DPV1 in ST

```
dwAcsPbWriteNPrmDpv1_DATA:= ADR (AcsPbPnPrmDpv1DataType)
ACS_PB_WRITE_N_PRM_DPV1                 (EN         :=
xAcsPbWriteNPrmDpv1_EN,


SLOT                          := byAcsPbWriteNPrmDpv1_SLOT,

                                          SLV_ADR             :=
byAcsPbWriteNPrmDpv1_SLV_ADR,

                                          DRIVE_TYPE :=
iAcsPbWriteNPrmDpv1_DRIVE_TYPE,


NVAR                          := byAcsPbWriteNPrmDpv1_NVAR,


DATA                          := dwAcsPbWriteNPrmDpv1_DATA);

xAcsPbWriteNPrmDpv1_DONE      := ACS_PB_WRITE_N_PRM_DPV1.DONE;
xAcsPbWriteNPrmDpv1_ERR          := ACS_PB_WRITE_N_PRM_DPV1.ERR;
wAcsPbWriteNPrmDpv1_ERNO      := ACS_PB_WRITE_N_PRM_DPV1.ERNO;
wAcsPbWriteNPrmDpv1_ERNO1  := ACS_PB_WRITE_N_PRM_DPV1.ERNO1;
wAcsPbWriteNPrmDpv1_ERNO2  := ACS_PB_WRITE_N_PRM_DPV1.ERNO2;
```

1.5.6.6.4 Visualization

In the application program, the user can add the visualization object in the project. In this object, the user needs to add faceplate of the desired function block, as shown in the figure.

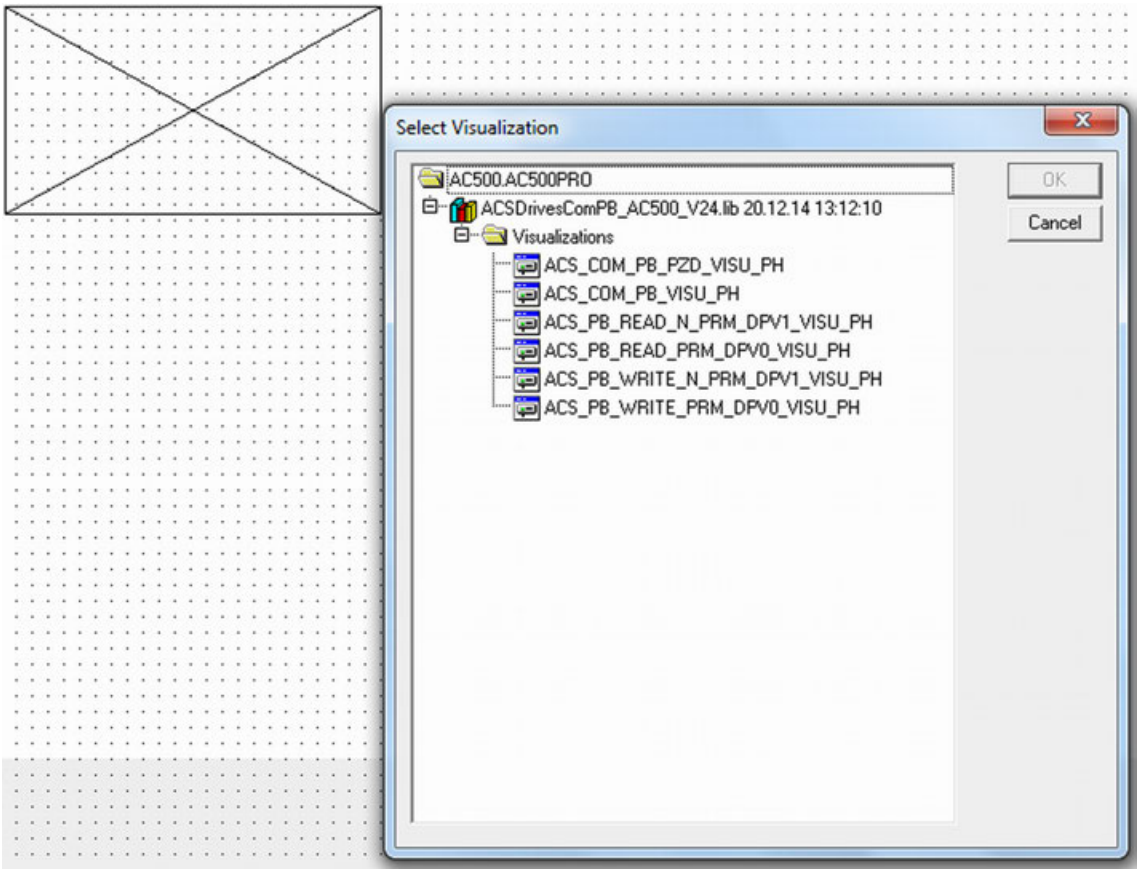


Fig. 166: Visualization

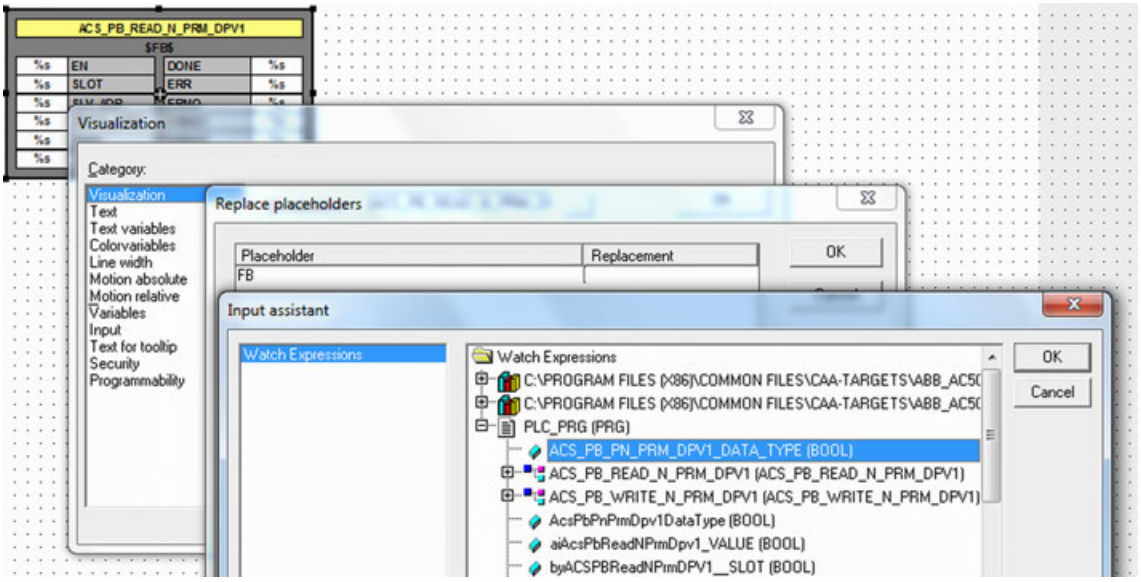


Fig. 167: Visualization: file selection

In the following chapter, overall visualization and visualization for each individual function block are discussed in detail.

ACS_COM_PB_VISU_PH visualization to run the ACS_COM_PB function block.

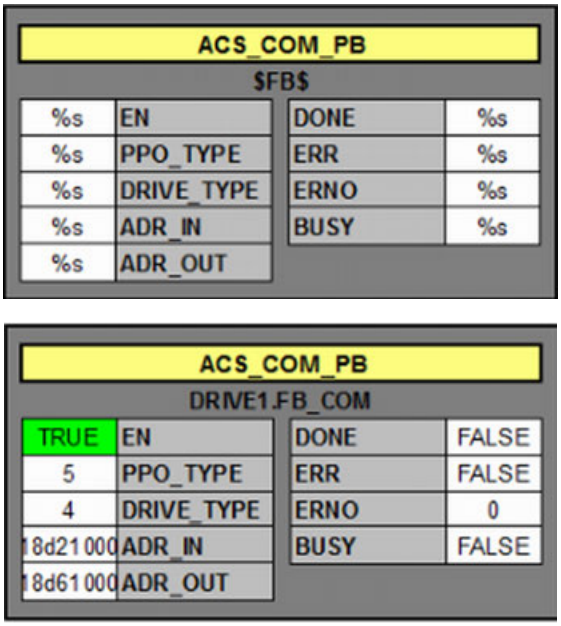


Fig. 168: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_COM_PB_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_COM_PB_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

PRO_TYPE

Access via: Numpad

Description: PRO_TYPE input 1, 2, 3, 4, 5, 6, 7 or 8

DRIVE_TYPE

Access via: Text

Description: DRIVE_TYPE input

ADR_IN

Access via: Text

Description: Address of first process input variable

ADR_OUT

Access via: Text

Description: Address of first process output variable

All inputs of ACS_COM_PB_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE Description: DONE output.

To indicate, the Function Block processing is completed.

ERR Description: ERR output.

Function Block processed with error.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

Colors

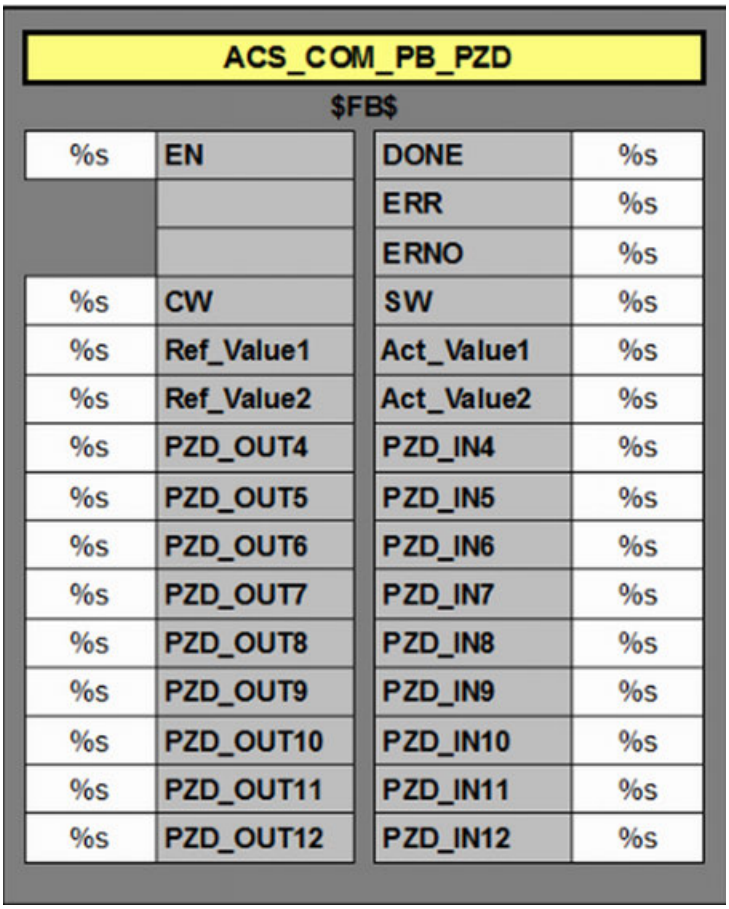
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_COM_PB_PZD_VISU_PH visualization to run the ACS_COM_PB_PZD function block.



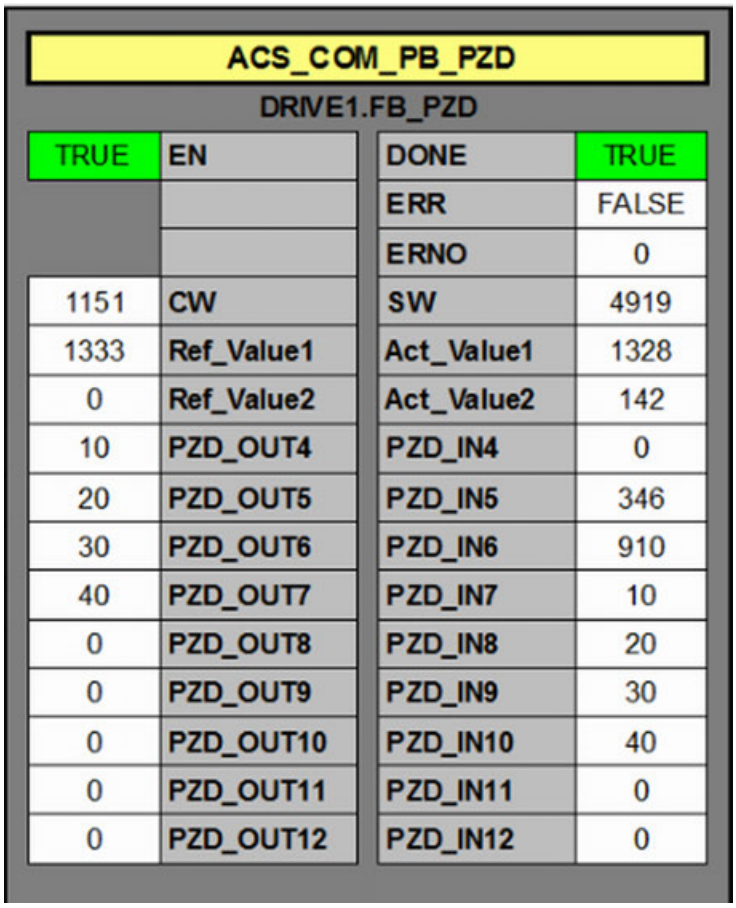


Fig. 169: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_COM_PB_PZD_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_COM_PB_PZD_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

PZD_OUT4

Access via: Text

Description: Process data word 4 to drive

PZD_OUT5

Access via: Text

Description: Process data word 5 to drive

PZD_OUT6 Access via: Text
Description: Process data word 6 to drive

PZD_OUT7 Access via: Text
Description: Process data word 7 to drive

PZD_OUT8 Access via: Text
Description: Process data word 8 to drive

PZD_OUT9 Access via: Text
Description: Process data word 9 to drive

PZD_OUT10 Access via: Text
Description: Process data word 10 to drive

PZD_OUT11 Access via: Text
Description: Process data word 11 to drive

PZD_OUT12 Access via: Text
Description: Process data word 12 to drive

All inputs of ACS_COM_PB_PZD_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE Description: DONE output.
To indicate, the Function Block processing is completed.

ERR Description: ERR output.
Function Block processed with error.

ERNO Description: ERNO output.

CW Drive Control Word

Ref1 Description: Reference 1

Ref2 Description: Reference 2

SW Drive Status Word

Act1 Actual value 1

Act2 Actual value 2

PZD_IN4 Description: Process data word 4 from drive

PZD_IN5 Description: Process data word 5 from drive

PZD_IN6 Description: Process data word 6 from drive

PZD_IN7 Description: Process data word 7 from drive

PZD_IN8 Description: Process data word 8 from drive

PZD_IN9 Description: Process data word 9 from drive

PZD_IN10 Description: Process data word 10 from drive

PZD_IN11 Description: Process data word 11 from drive

PZD_IN12 Description: Process data word 12 from drive

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" ↳ *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_PB_READ_PRM_DPV0_VISU_PH visualization to run the ACS_PB_READ_PRM_DPV0 function block.

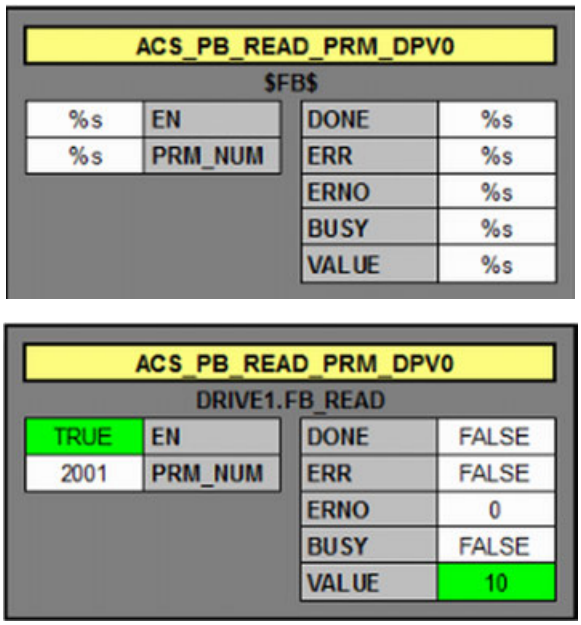


Fig. 170: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PB_READ_PRM_DPV0_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PB_READ_PRM_DPV0_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

PRM_NUM

Access via: Numpad

Description: Parameter number in drive.

All inputs of ACS_PB_READ_PRM_DPV0_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE

Description: DONE output.

To indicate, the Function Block processing is completed.

ERR Description: ERR output.
Function Block processed with error.

ERNO Description: ERNO output.

BUSY Description: BUSY output.

VALUE Parameter value

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_PB_WRITE_PRM_DPV0_VISU_PH visualization to run the ACS_PB_WRITE_PRM_DPV0 function block.

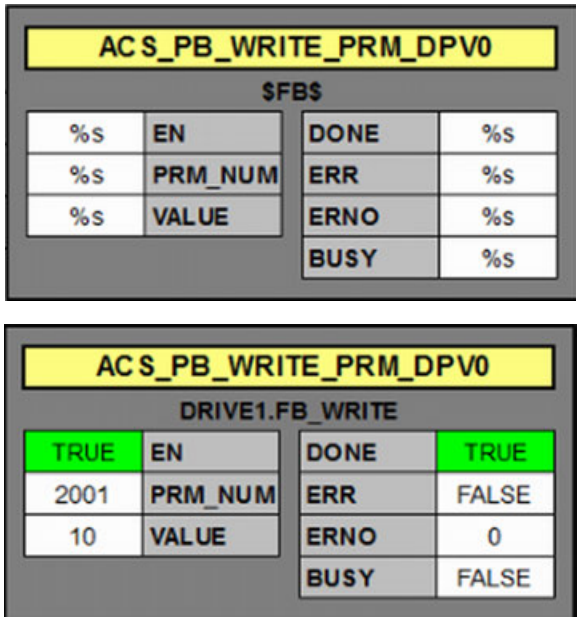


Fig. 171: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PB_WRITE_PRM_DPV0_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PB_WRITE_PRM_DPV0_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

PRM_NUM

Access via: Numpad

Description: Parameter number in drive.

VALUE

Access via: Numpad

Parameter value

All inputs of ACS_PB_WRITE_PRM_DPV0_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE

Description: DONE output.

To indicate, the Function Block processing is completed.

ERR

Description: ERR output.

Function Block processed with error.

ERNO

Description: ERNO output.

BUSY

Description: BUSY output.

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_PB_READ_N_PRM_DPV1_VISU_PH visualization to run the ACS_PB_READ_N_PRM_DPV1 function block.

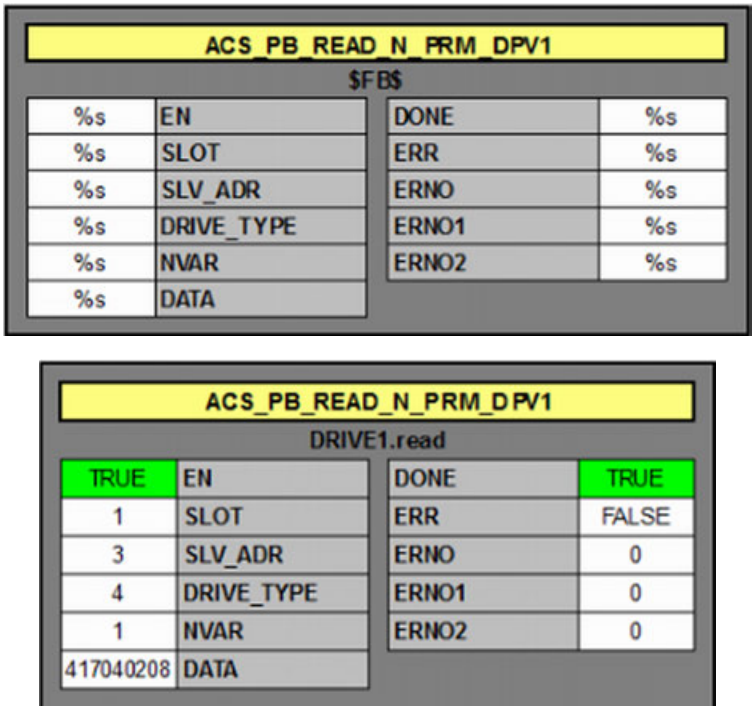


Fig. 172: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PB_READ_N_PRM_DPV1_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PB_READ_N_PRM_DPV1_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

SLOT Access via: Text

Description: SLOT, in which the PROFIBUS master communication module is mounted.

SLV_ADR Access via: Text

Description: PROFIBUS station address.

DRIVE_TYPE Access via: Text

Description: DRIVE_TYPE input

NVAR Access via: Text

Description: Number of drive parameters to be read from drive using PROFIBUS.

All inputs of ACS_PB_READ_N_PRM_DPV1_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE Description: DONE output.

To indicate, the Function Block processing is completed.

ERR Description: ERR output.

Function Block processed with error.

ERNO Description: ERNO output.

DATA Description: Data input to connect structure ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index.

ERNO1 Description: Additional error information.

The Value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO2 Description: Additional error information.

The Value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_PB_WRITE_N_PRM_DPV1_VISU_PH visualization to run the ACS_PB_WRITE_N_PRM_DPV1 function block.

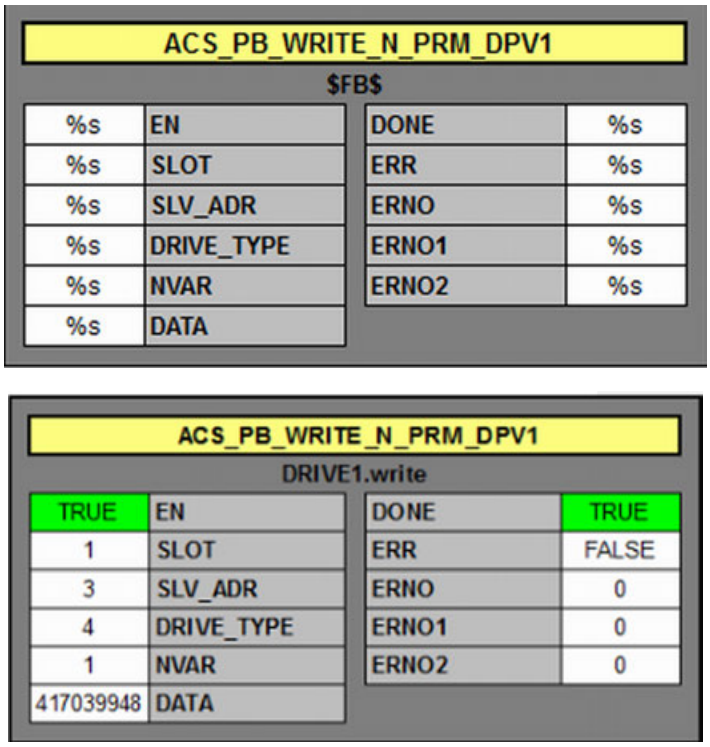


Fig. 173: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PB_WRITE_N_PRM_DPV1_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PB_WRITE_N_PRM_DPV1_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

SLOT Access via: Text

Description: SLOT, in which the PROFIBUS master communication module is mounted.

SLV_ADR Access via: Text

Description: PROFIBUS station address.

DRIVE_TYPE Access via: Text

Description: DRIVE_TYPE input

NVAR Access via: Text

Description: Number of drive parameters to be read from drive using PROFIBUS.

All inputs of ACS_PB_WRITE_N_PRM_DPV1_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE Description: DONE output.

To indicate, the Function Block processing is completed.

ERR Description: ERR output.

Function Block processed with error.

ERNO Description: ERNO output.

DATA Description: Data input to connect structure ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index.

ERNO1 Description: Additional error information.

The Value applied at ERNO1 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

ERNO2 Description: Additional error information.

The Value applied at ERNO2 is only valid, if DONE = TRUE, ERR = TRUE and ERNO = 6036 HEX (24630 DEC).

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.7 ACS / DCS Drives read / write parameter via PROFINET library

1.5.6.7.1 Preconditions for the use of the ACS / DCS drives read / write parameter via PROFINET library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

The blocks can only be used in combination with the ACSDrives-Base_AC500_V20 Library.

The library is released for the following products:

- CPUs: AC500
- Fieldbus: PROFINET
- Drives: ACS800, ACSM1, ACS355, ACS550, ACH550, ACQ810, ACS850, ACS880, ACS580, DCS550, DCS800
- PROFINET configuration:
  Prior to the use of the function blocks, PROFINET Communication Module has to be configured accordingly using Automation Builder, at "Interfaces".

If multiple ACS_PN_READ_N_PRM_DPV1 and / or multiple ACS_PN_WRITE_N_PRM_DPV1 function blocks are enabled at the same time, it may cause an READ / WRITE error.

1.5.6.7.2 Components of PROFINET read / write library

ACSDrivesComPN_AC500_V24 Library contains PROFINET DPV1 READ / WRITE function blocks and visualizations.

Function blocks

ACS_PN_READ_N_PRM_DPV1 ↳ *Chapter 1.5.6.7.4.1 “ACS_PN_READ_N_PRM_DPV1 read parameters from ABB drives via PROFINET DPV1” on page 2453*	This function block reads maximum 37 parameters from an ACS Drive via PROFINET.
ACS_PN_WRITE_N_PRM_DPV1 ↳ *Chapter 1.5.6.7.4.2 “ACS_PN_WRITE_N_PRM_DPV1 write parameters from ABB drives via PROFINET DPV1” on page 2457*	This function block writes maximum 37 parameters to an ACS Drive via PROFINET.

Visualizations

ACS_PN_READ_N_PRM_DPV1_VISU_PH ↳ *Chapter 1.5.6.7.5.1 “ACS_PN_READ_N_PRM_DPV1_VISU_PH faceplate of ACS_PN_READ_N_PRM_DPV1 function block.” on page 2463*	Visualization to run the ACS_PN_READ_N_PRM_DPV1 function block.
ACS_PN_WRITE_N_PRM_DPV1_VISU_PH ↳ *Chapter 1.5.6.7.5.2 “ACS_PN_WRITE_N_PRM_DPV1_VISU_PH visualization to run the ACS_PN_WRITE_N_PRM_DPV1 function block.” on page 2465*	Visualization to run the ACS_PN_WRITE_N_PRM_DPV1 function block.

Global variables

ACS_PN_VERSION_INFORMATION	Stores all the version information of the file along with the change log. No variable is declared inside this section.

1.5.6.7.3 Overview of the ACS / DCS drives read / write parameter via PROFINET function blocks

Used abbreviations:

FBhv	Function block with historical values
FBnohv	Function block without historical values
F	Function
Enum	Enumeration
Struct	Structure
Visu	Visualization

Function blocks

POU name	Function
ACS_PN_READ_N_PRM_DPV1 ↳ *Chapter 1.5.6.7.4.1 “ACS_PN_READ_N_PRM_DPV1 read parameters from ABB drives via PROFINET DPV1” on page 2453*	Function block ACS_PN_READ_N_PRM_DPV1 reads maximum 37 parameters from the drive
ACS_PN_WRITE_N_PRM_DPV1 ↳ *Chapter 1.5.6.7.4.2 “ACS_PN_WRITE_N_PRM_DPV1 write parameters from ABB drives via PROFINET DPV1” on page 2457*	Function block ACS_PN_WRITE_N_PRM_DPV1 writes maximum 37 parameters to drive

Visualizations

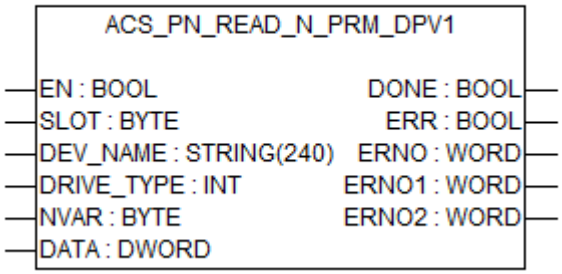
POU name	Function
ACS_PN_READ_N_PRM_DPV1_VISU_PH *Chapter 1.5.6.7.5.1 “ACS_PN_READ_N_PRM_DPV1_VISU_PH faceplate of ACS_PN_READ_N_PRM_DPV1 function block.” on page 2463*	Visualization ACS_PN_READ_N_PRM_DPV1_VISU_PH shows the actual values of all inputs and outputs of the instance of ACS_PN_READ_N_PRM_DPV1 function block
ACS_PN_WRITE_N_PRM_DPV1_VISU_PH *Chapter 1.5.6.7.5.2 “ACS_PN_WRITE_N_PRM_DPV1_VISU_PH visualization to run the ACS_PN_WRITE_N_PRM_DPV1 function block.” on page 2465*	Visualization ACS_PN_WRITE_N_PRM_DPV1_VISU_PH shows the actual values of all inputs and outputs of the instance of the ACS_PN_WRITE_N_PRM_DPV1 function block.

Global variables

POU name	Function
ACS_PN_VERSION_INFORMATION	Stores all the version information of the file along with the change log. No variable is declared inside this section.

1.5.6.7.4 Function blocks

ACS_PN_READ_N_PRM_DPV1 read parameters from ABB drives via PROFINET DPV1



Function block ACS_PN_READ_N_PRM_DPV1 reads maximum 37 parameters from the drive in a single query. The number of parameters to be read is specified at the input NVAR.

Function block information

Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPN_AC500_V24.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_PN_READ_N_PRM_DPV1 reads maximum 37 parameters from the drive in a single query. The number of parameters to be read is specified at the input NVAR.

Parameters to read to the drive is specified at the DATA input.
ACS_PB_PN_PRM_DPV1_DATA_TYPE structure must be declared to a variable and connected to DATA input using ADR, which is to be entered with Group, Index.

Read parameter type and values are stored in the same variable.

ACS_PB_PN_PRM_DPV1_DATA_TYPE structure has the following array elements:

- abyPrmGroup : Array of 37 BYTE for specifying parameter Group.
- abyPrmIndex : Array of 37 BYTE for specifying parameter Index

- abyPrmType : Array of 37 BYTE
  READ parameter data type will be available here. For details refer to ACS_PB_PN_PRM_TYPE_ENUM *Chapter 1.5.6.2.5.2 “ACS_PB_PN_PRM_TYPE_ENUM” on page 2251*
- adwPrmValue: Array of 37 DWORD
  READ parameter value will be available here.

> *Currently user cannot use enumeration from ACS_PB_PN_PRM_TYPE_ENUM.*
>
> *Instead user need to use numerical values from ACS_PB_PN_PRM_TYPE_ENUM only.*

The values in the structure area are updated, when the READ job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Example

If the user need to read Parameter 10.01 and 11.05, then user need to enter the group and index number separately in input DATA as below:

DataStructure : ACS_PB_PN_PRM_DPV1_DATA_TYPE

Parameter Group : DataStruct.abyPrmGroup[1]:= 10; DataStruct.abyPrmGroup[2]:= 11;

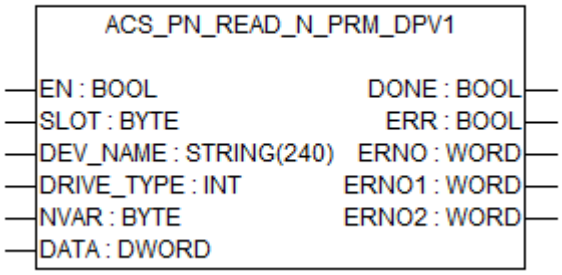
Parameter Index : DataStruct .abyPrmIndex [1]:= 01; DataStruct .abyPrmIndex [2]:= 05;

Parameter Type : Once read operation is complete parameter type will be stored inside DataStruct. abyPrmType [1]:= xx; DataStruct. abyPrmType [2]:= xx;

Parameter Value : And the value will be stored inside DataStruct. adwPrmValue [1]:= xx; DataStruct. adwPrmValue [2]:= xx;

The values in the structure area are updated once the read job is performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Input description



EN (enable)

Data type: BOOL

In order to enable the function block processing, input EN has to be set from FALSE to TRUE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

Default value = FALSE.

If multiple ACS_PN_READ_N_PRM_DPV1 and / or multiple ACS_PN_WRITE_N_PRM_DPV1 functions blocks enabled at the same time it may cause for error Read / Write error.

SLOT (slot)

Data type: BYTE, Default value: 1, Range: 1 to 6

At input SLOT the communication module SLOT (module number) is selected, which should be used by the Function Block. All external communication modules are serially numbered from right to left, starting with module number 1.

If the SLOT number is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4020.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME (device name)

Data type: STRING, Default value: "Not Available", Range: 1 to 240

At input DEV_NAME, the name of the drive (slave) from which the parameter value to be read must be specified.

The function block is designed to be used with a fix SLAVE device.

If the DEV_NAME is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4030.

The DEV_NAME input should not be changed, while the program is running. If changed, nevertheless the new value will become effective only after the function block is enabled again.

DRIVE_TYPE (drive type)

Data type: ENUM, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM or INT. (Please refer ACS_DRIVE_ENUM for details.) The input can be set either by the value directly or by using the enum.

NVAR (number of variables)

Data type: BYTE, Default value: 0, Range: 0 to 37

With input NVAR the function block can be configured to read between 0 to 37 drive parameter values.

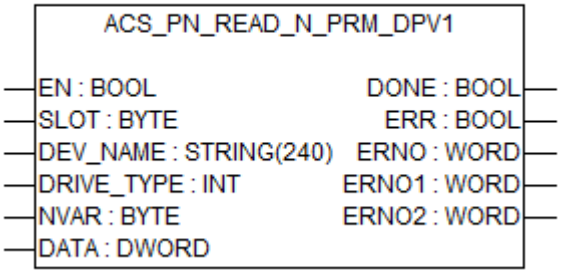
If the NVAR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

DATA (data)

Data type: DWORD

Input Data must be connected to the variable of type ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index. Each drive must have its own DATA variable.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table Error Messages of the ACS Drives function block libraries(see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1)

Data type: WORD, Default value: 0, Range: ≥0

Reserved output.

ERNO2 (error number 2)

Data type: WORD, Default value: 0, Range: ≥0

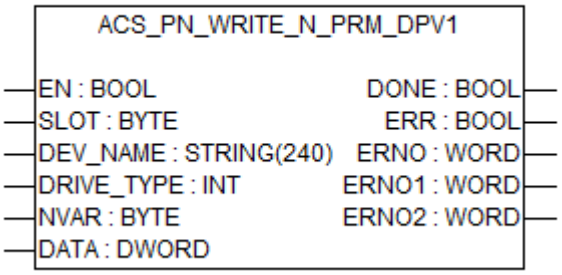
Reserved output.

Calling of ACS_PN_READ_N_PRM_DPV1 in ST

```
dwAcsPnReadNPrmDpv1_DATA:= ADR (AcsPbPnPrmDpv1DataType)
ACS_PN_READ_N_PRM_DPV1 (EN                                    :=
xAcsPnReadNPrmDpv1_EN,

                                SLOT                          :=
```

```
byAcsPnReadNPrmDpv1_SLOT,

                          DEV_NAME          :=
byAcsPnReadNPrmDpv1_DEV_NAME,

                          DRIVE_TYPE :=
iAcsPnReadNPrmDpv1_DRIVE_TYPE,

                          NVAR                          :=
byAcsPnReadNPrmDpv1_NVAR,

                          DATA                          :=
dwAcsPnReadNPrmDpv1_DATA);

xAcsPnReadNPrmDpv1_DONE     := ACS_PN_READ_N_PRM_DPV1.DONE;
xAcsPnReadNPrmDpv1_ERR         := ACS_PN_READ_N_PRM_DPV1.ERR;
wAcsPnReadNPrmDpv1_ERNO     := ACS_PN_READ_N_PRM_DPV1.ERNO;
wAcsPnReadNPrmDpv1_ERNO1 := ACS_PN_READ_N_PRM_DPV1.ERNO1;
wAcsPnReadNPrmDpv1_ERNO2 := ACS_PN_READ_N_PRM_DPV1.ERNO2;
```

ACS_PN_WRITE_N_PRM_DPV1 write parameters from ABB drives via PROFINET DPV1



Function block ACS_PN_WRITE_N_PRM_DPV1 writes maximum 37 parameters to the drive in a single query. The number of parameters to be write is specified at the input NVAR.

Function block information

Available in runtime system:	V2.4 and above
Included in library:	ACSDrivesComPN_AC500_V24.lib
Function block type:	Function block with historical values.

Block description

Function block ACS_PN_WRITE_N_PRM_DPV1 writes maximum 37 parameters to the drive in a single query. The number of parameters to be write is specified at the input NVAR.

Another limit while using the write function block is, it can process only up to 240 byte data in one request or 37 drive parameters whichever is lower. If the write data length is more than 240 byte, the function block generates an error code 16#7012.

Parameters to write to the drive are specified at the DATA input.
ACS_PB_PN_PRM_DPV1_DATA_TYPE structure must be declared to a variable and connected to DATA input using ADR, which is to be entered with Group, Index.

ACS_PB_PN_PRM_DPV1_DATA_TYPE structure has the following array elements:

- abyPrmGroup : Array of 37 BYTE for specifying parameter Group.
- abyPrmIndex : Array of 37 BYTE for specifying parameter Index

- abyPrmType : Array of 37 BYTE for specifying parameter type , Please refer the respective Drives manual for parameter data type and enter the respective ENUM / VALUE . For details about ENUM / VALUE please refer ACS_PB_PN_PRM_TYPE_ENUM *Chapter 1.5.6.2.5.2 “ACS_PB_PN_PRM_TYPE_ENUM” on page 2251*
- adwPrmValue: Array of 37 BYTE for specifying parameter value
  The values in the structure area are updated, when the WRITE job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

> *Currently user cannot use enumeration from ACS_PB_PN_PRM_TYPE_ENUM.*
>
> *Instead user need to use numerical values from ACS_PB_PN_PRM_TYPE_ENUM only.*

The values in the structure area are updated, when the WRITE job was performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Example

If the user need to read Parameter 10.01 and 11.05, then he need to enter the group and index number separately in input DATA as below:

DataStructure : ACS_PB_PN_PRM_DPV1_DATA_TYPE

Parameter Group : DataStruct.abyPrmGroup[1]:= 10; DataStruct.abyPrmGroup[2]:= 11;

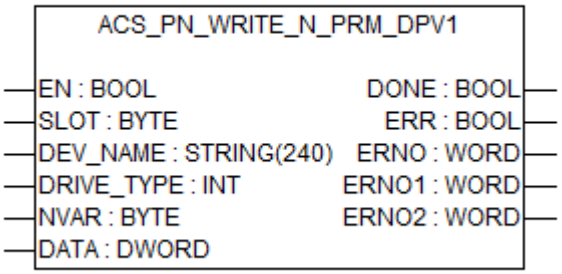
Parameter Index : DataStruct .abyPrmIndex [1]:= 01; DataStruct .abyPrmIndex [2]:= 05;

Parameter Type : DataStruct. abyPrmType [1]:= xx; DataStruct. abyPrmType [2]:= xx;

Parameter Value : DataStruct. adwPrmValue [1]:= xx; DataStruct. adwPrmValue [2]:= xx;

The values in the structure area are written to drive when the write job is performed without error. This is indicated by DONE=TRUE and ERR=FALSE.

Input description



EN (enable)

Data type: BOOL

In order to enable the function block processing, input EN has to be set from FALSE to TRUE.

While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

Default value = FALSE.

If multiple ACS_PN_READ_N_PRM_DPV1 and / or multiple ACS_PN_WRITE_N_PRM_DPV1 functions blocks enabled at the same time it may cause for error Read / Write error.

SLOT (slot)

Data type: BYTE, Default value: 1, Range: 1 to 6

At input SLOT the communication module SLOT (module number) is selected, which should be used by the Function Block. All external communication modules are serially numbered from right to left, starting with module number 1.

If the SLOT number is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4020.

For PM595 processor modules, internal PROFINET communication module with connector ETH3 is slot 5, ETH4 is slot 6.

DEV_NAME (device name)

Data type: STRING, Default value: "Not Available", Range: 1 to 240

At input DEV_NAME, the name of the drive (slave) from which the parameter value to be read must be specified.

The function block is designed to be used with a fix SLAVE device.

If the DEV_NAME is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4030.

The DEV_NAME input should not be changed, while the program is running. If changed, nevertheless the new value will become effective only after the function block is enabled again.

DRIVE_TYPE (drive type)

Data type: ENUM, Default value: ACS_DRIVE_ACS355

At the input DRIVE_TYPE the type of ACS drive is specified with an ENUM or INT. (Please refer ACS_DRIVE_ENUM for details.) The input can be set either by the value directly or by using the enum.

NVAR (number of variables)

Data type: BYTE, Default value: 0, Range: 0 to 37

With input NVAR the function block can be configured to read between 0 to 37 drive parameter values.

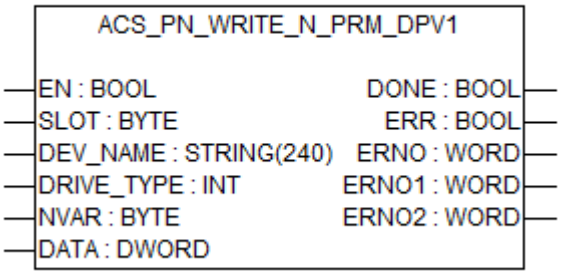
If the NVAR is given incorrect or invalid, then the function block will display an error. In such case ERR=TRUE and ERNO=16#4040.

DATA (data)

Data type: DWORD

Input Data must be connected to the variable of type ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index. Each drive must have its own DATA variable.

Output description



DONE (done) Data type: BOOL

Output DONE indicates the processing of the function block. If DONE = FALSE, the function block is not processed due to EN = FALSE and all outputs are set to 0.

For that reason, the other outputs always have to be considered together with output DONE. All other outputs are only valid, if DONE = TRUE.

DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in a separate table Error Messages of the ACS Drives function block libraries(see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ERNO1 (error number 1) Data type: WORD, Default value: 0, Range: ≥0

Reserved output.

ERNO2 (error number 2) Data type: WORD, Default value: 0, Range: ≥0

Reserved output.

Calling of ACS_PN_WRITE_N_PRM_DPV1 in ST

```
dwAcsPnWriteNPrmDpv1_DATA:= ADR (AcsPbPnPrmDpv1DataType)
ACS_PN_WRITE_N_PRM_DPV1 (EN                                  :=
xAcsPnWriteNPrmDpv1_EN,

                                SLOT                         :=
byAcsPnWriteNPrmDpv1_SLOT,

                                DEV_NAME          :=
byAcsPnWriteNPrmDpv1_DEV_NAME,

                                DRIVE_TYPE :=
iAcsPnWriteNPrmDpv1_DRIVE_TYPE,

                                NVAR                         :=
byAcsPnWriteNPrmDpv1_NVAR,

                                DATA                         :=
dwAcsPnWriteNPrmDpv1_DATA);

xAcsPnWriteNPrmDpv1_DONE     := ACS_PN_WRITE_N_PRM_DPV1.DONE;
xAcsPnWriteNPrmDpv1_ERR         := ACS_PN_WRITE_N_PRM_DPV1.ERR;
wAcsPnWriteNPrmDpv1_ERNO     := ACS_PN_WRITE_N_PRM_DPV1.ERNO;
wAcsPnWriteNPrmDpv1_ERNO1 := ACS_PN_WRITE_N_PRM_DPV1.ERNO1;
wAcsPnWriteNPrmDpv1_ERNO2 := ACS_PN_WRITE_N_PRM_DPV1.ERNO2;
```

1.5.6.7.5 Visualization

In the application program, the user can add the visualization object in the project. In this object, the user needs to add faceplate of the desired function block, as shown in the figure.

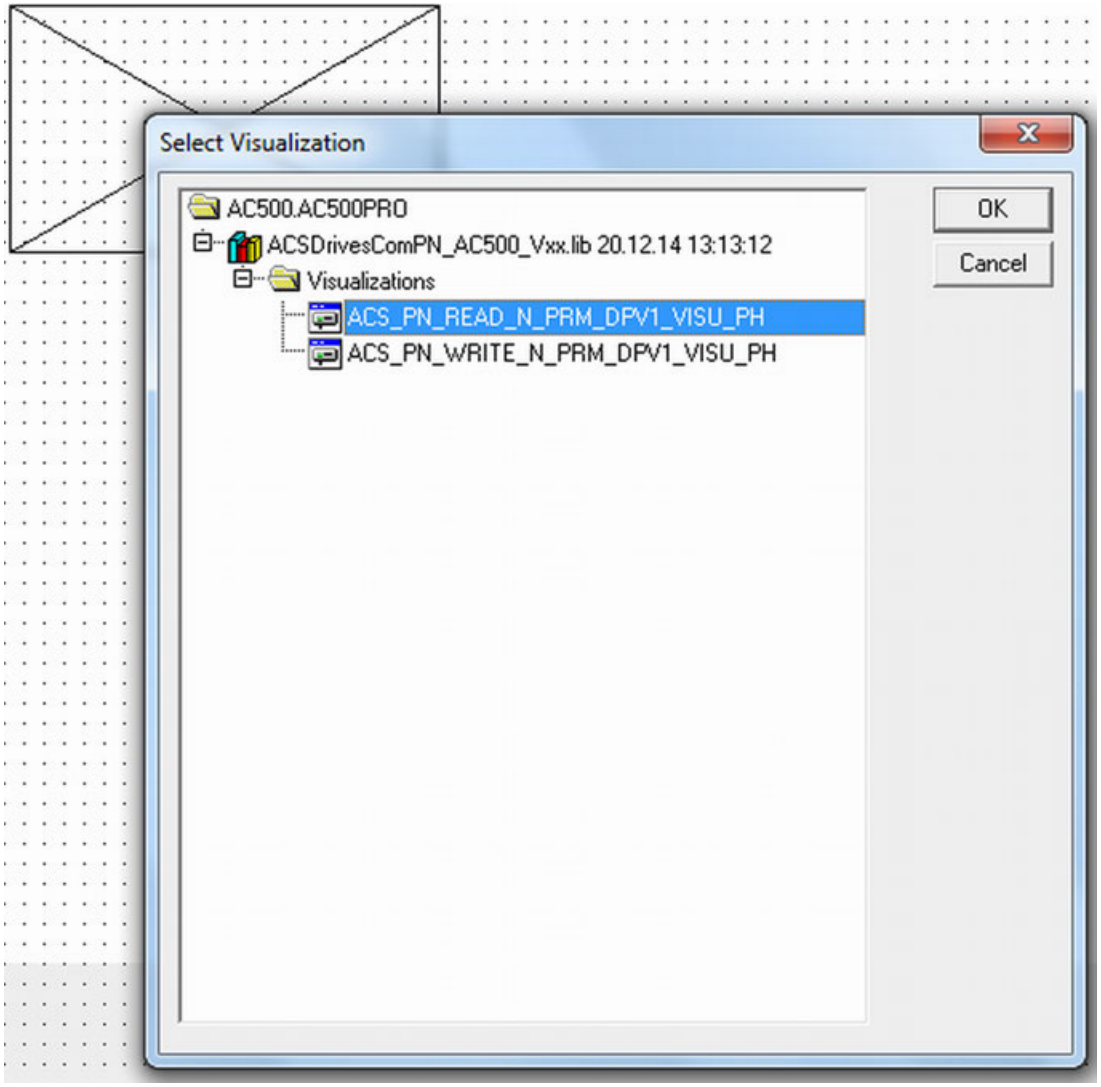


Fig. 174: Visualization

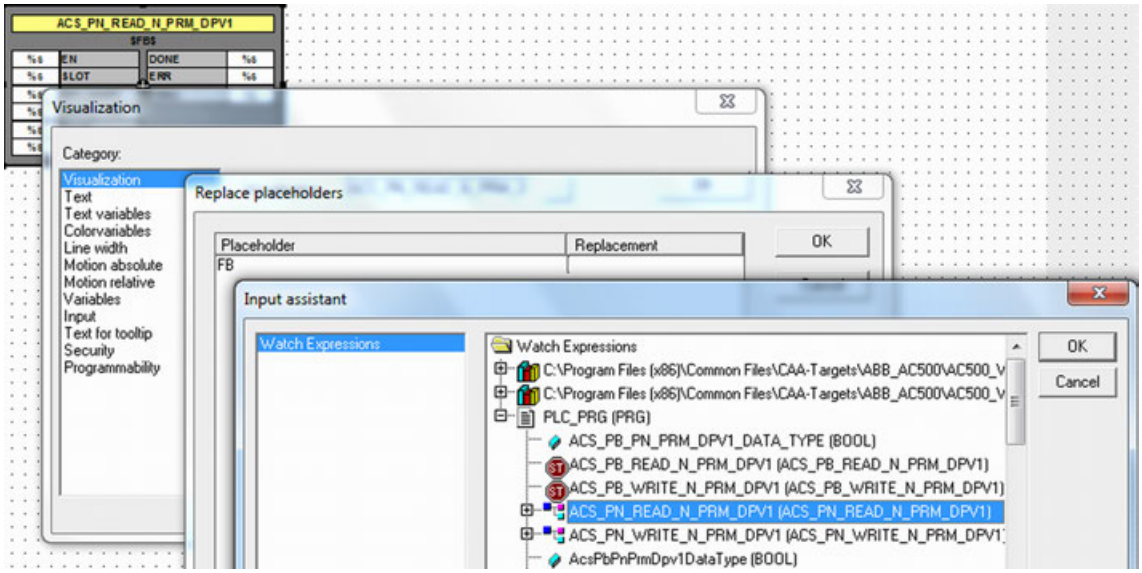


Fig. 175: Visualization: file selection

In the following chapter, overall visualization and visualization for each individual function block are discussed in detail.

ACS_PN_READ_N_PRM_DPV1_VISU_PH faceplate of ACS_PN_READ_N_PRM_DPV1 function block.

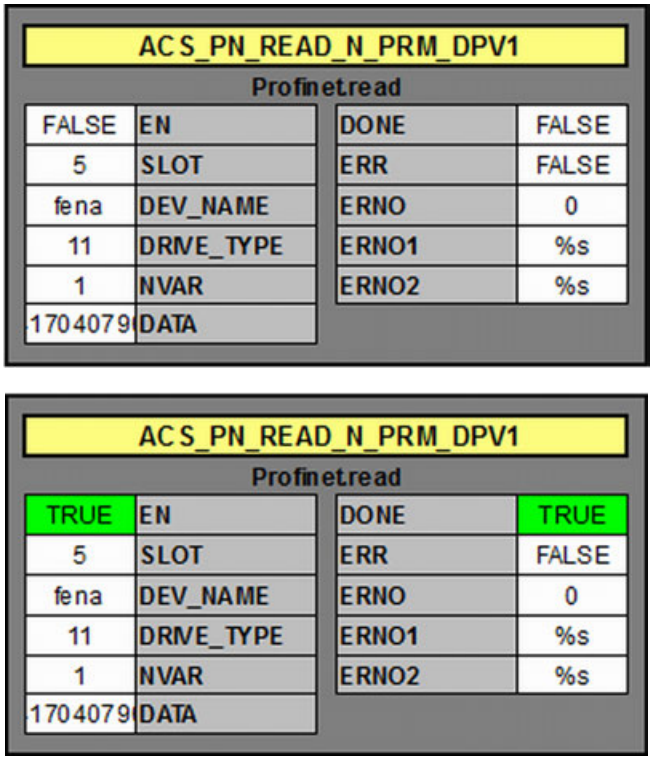


Fig. 176: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PN_READ_N_PRM_DPV1_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PN_READ_N_PRM_DPV1_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

SLOT

Access via: Text

Description: SLOT, in which the PROFIBUS master communication module is mounted.

DEV_NAME (device name)

Data type: STRING, Default value: "Not Available", Range: 1 to 240

At input DEV_NAME, the name of the drive (slave) from which the parameter value to be read must be specified.

The function block is designed to be used with a fix SLAVE device.

If the DEV_NAME is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4030.

The DEV_NAME input should not be changed, while the program is running. If changed, nevertheless the new value will become effective only after the function block is enabled again.

DRIVE_TYPE Access via: Text

Description: DRIVE_TYPE input

NVAR Access via: Text

Description: Number of drive parameters to be read from drive using PROFIBUS.

Access R

DONE Description: DONE output.

To indicate, the Function Block processing is completed.

ERR Description: ERR output.

Function Block processed with error.

ERNO Description: ERNO output.

DATA Description: Data input to connect structure ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index.

All inputs of ACS_PN_READ_N_PRM_DPV1_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

ACS_PN_WRITE_N_PRM_DPV1_VISU_PH visualization to run the ACS_PN_WRITE_N_PRM_DPV1 function block.

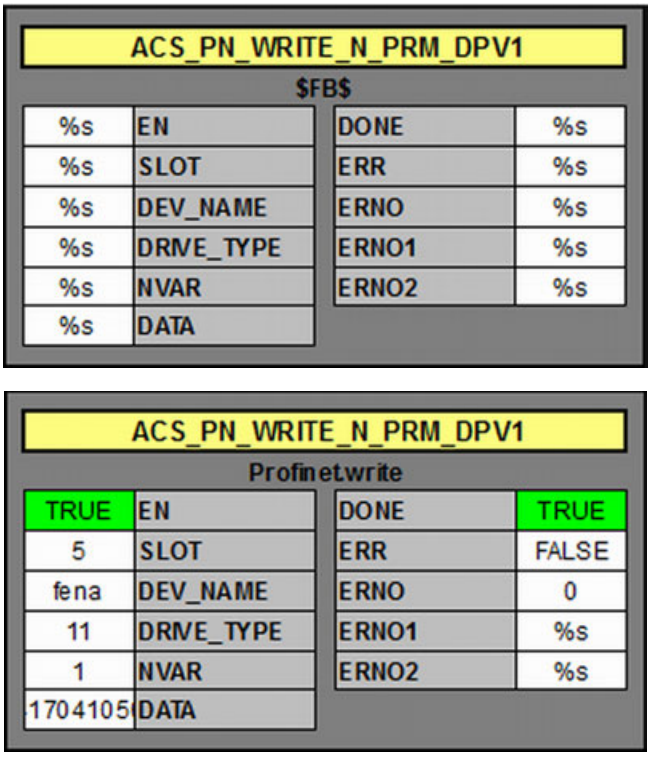


Fig. 177: Visualization in offline / online mode

Visualization information

Available in runtime system:	V2.4
Included in library:	ACSDrivesBase_AC500_V20.lib

Visualization description

Visualization element ACS_PN_WRITE_N_PRM_DPV1_VISU_PH is used to show the actual values of all inputs and outputs of the instance of ACS_PN_WRITE_N_PRM_DPV1_VISU_PH function block.

The visualization is also used to control the function block by those inputs, which are not connected inside the program.

Parameters

Access R/W

EN

Access via: Toggle

Description: EN input

Enables the function block by TRUE value of input EN.

SLOT

Access via: Text

Description: SLOT, in which the PROFIBUS master communication module is mounted.

DEV_NAME (device name)

Data type: STRING, Default value: "Not Available", Range: 1 to 240

At input DEV_NAME, the name of the drive (slave) from which the parameter value to be read must be specified.

The function block is designed to be used with a fix SLAVE device.

If the DEV_NAME is given incorrect or invalid, then the function block will displays an error. In such case ERR=TRUE and ERNO=16#4030.

The DEV_NAME input should not be changed, while the program is running. If changed, nevertheless the new value will become effective only after the function block is enabled again.

DRIVE_TYPE

Access via: Text

Description: DRIVE_TYPE input

NVAR

Access via: Text

Description: Number of drive parameters to be read from drive using PROFIBUS.

All inputs of ACS_PN_WRITE_N_PRM_DPV1_VISU_PH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

Access R

DONE

Description: DONE output.

To indicate, the Function Block processing is completed.

ERR

Description: ERR output.

Function Block processed with error.

ERNO

Description: ERNO output.

DATA (data)

Data type: DWORD

Input Data must be connected to the variable of type ACS_PB_PN_PRM_DPV1_DATA_TYPE for specifying READ parameter group and index. Each drive must have its own DATA variable.

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.8 DCS drives library

1.5.6.8.1 Preconditions for the use of the DCS drives library

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

Some of the blocks can only be used in combination with one of the ACSDrivesComXXX_AC500_Vyy libraries, e.g. the ACSDrivesComModRTU_AC500_V20.lib.

The library is released for the following products:

- CPUs: AC500 and AC500-eCo
  Connection of more drives depends on performance of used CPU, communication type and settings.
- Fieldbus: PROFIBUS DP, Modbus RTU, Modbus TCP, PROFINET, CANopen

Compatibility

To check the compatiblity of drives and their communication modules, please refer to the following table that shows the tested combinations:

Communication	PLC communication modules		PLC Fieldbus Adapter of drive			Drive		
	PLC communication module	Firmware Version	Fieldbus Adapter (FBA)	FBA comm sw ver	FBA appl sw ver	Drive	Firmware Version	Drive Rating ID
PROFIBUS	CM572-DP	V1.0.97	RPBA	132h	307h	DCS550	-1.3	
		V1.0.97	RPBA	132h	307h	DCS800	-3.6	

Communication	PLC communication modules		PLC Fieldbus Adapter of drive			Drive		
	PLC communication module	Firmware Version	Fieldbus Adapter (FBA)	FBA comm sw ver	FBA appl sw ver	Drive	Firmware Version	Drive Rating ID
		V01.097	RPBA-01	132 h	307h	DCS800	3.7	DCS800-S02-0050-05
PROFIBUS	CM592-DP	V1.1.1.21	RPBA-01	132h	308h	DCS800	3.7	DCS800-S02-0050-05
Modbus	Onboard		RMBA-01			DCS800	3.7	DCS800-S02-0050-05
Modbus TCP	Onboard		RETA-01	130h	306h	DCS800	3.7	DCS800-S02-0050-05
PROFINET	CM579-PNIO	V2.6.5	RETA-02	130h	305h	DCS800	3.7	DCS800-S02-0050-05
CANopen	CM578	V01.220	RCAN-01	131h	114h	DCS800	3.7	DCS800-S02-0050-05
CANopen	CM598-CN	V1.11.1.21	RCAN-01	131h	114h	DCS800	3.7	DCS800-S02-0050-05

1.5.6.8.2 Components of the DCS drives library

Function blocks

DCS_DRIVES_CTRL ↳ *Chapter 1.5.6.8.4.1 "DCS_DRIVES_CTRL Control of DCS Drives with ABB-Drives profile using a communication block" on page 2470*	Control of DCS Drives with ABB-Drives Profile using additional communication block
DCS_DRIVES_CTRL_GEN ↳ *Chapter 1.5.6.8.4.2 "DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus" on page 2477*	Control of DCS Drives with ABB-Drives Profile

Enumeration

DCS_DRIVE_ENUM ↳ *Chapter 1.5.6.8.5.1 "DCS_DRIVE_ENUM enumerations to be used at the input DRIVE_TYPE of ACS_COM_xxx function blocks" on page 2484*	Enumeration of DCS drive type

Visualizations

DCS_DRIVES_CTRL_VISU_PH ↳ *Chapter 1.5.6.8.6.1 “DCS_DRIVES_CTRL_VISU_PH faceplate of function block DCS_DRIVES_CTRL” on page 2484*	Faceplate for the function block
DCS_DRIVES_CTRL_GEN_VISU_PH ↳ *Chapter 1.5.6.8.6.2 “DCS_DRIVES_CTRL_GEN_VISU_PH faceplate of function block DCS_DRIVES_CTRL_GEN” on page 2489*	Faceplate for the function block

1.5.6.8.3 Overview of the DCS Drives library components according to their call names

Used abbreviations:

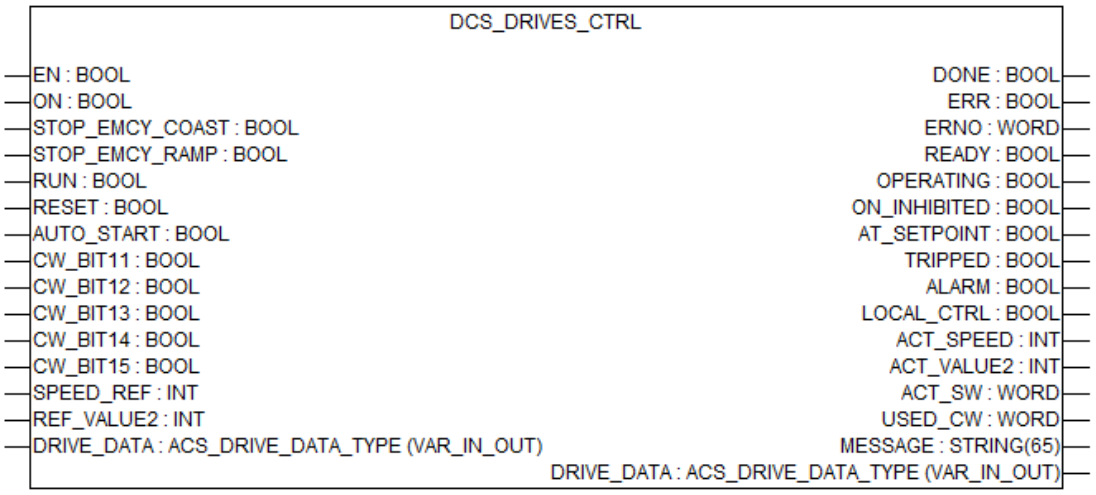
FBhv	Function block with historical values
FBnohv	Function block without historical values
F	Function
Enum	Enumeration
Struct	Structure
Visu	Visualization

VE name	Type	Function
DCS_DRIVES_CTRL ↳ *Chapter 1.5.6.8.4.1 “DCS_DRIVES_CTRL Control of DCS Drives with ABB-Drives profile using a communication block” on page 2470*	FBhv	DCS Drives Control via communication block
DCS_DRIVES_CTRL_GEN ↳ *Chapter 1.5.6.8.4.2 “DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus” on page 2477*	FBhv	DCS Drives Control direct via Status and Control word
DCS_DRIVES_CTRL_GEN_VISU_PH ↳ *Chapter 1.5.6.8.6.2 “DCS_DRIVES_CTRL_GEN_VISU_PH faceplate of function block DCS_DRIVES_CTRL_GEN” on page 2489*	Visu	Faceplate for DCS_DRIVES_CTRL_GEN

VE name	Type	Function
DCS_DRIVES_CTRL_VISU_PH ↳ *Chapter 1.5.6.8.6.1 "DCS_DRIVES_CTRL_VISU_PH faceplate of function block DCS_DRIVES_CTRL" on page 2484*	Visu	Faceplate for DCS_DRIVES_CTRL
DCS_DRIVE_ENUM ↳ *Chapter 1.5.6.8.5.1 "DCS_DRIVE_ENUM enumerations to be used at the input DRIVE_TYPE of ACS_COM_xxx function blocks" on page 2484*	Enum	Enumarations for DCS_DRIVES types

1.5.6.8.4 Function blocks

DCS_DRIVES_CTRL Control of DCS Drives with ABB-Drives profile using a communication block

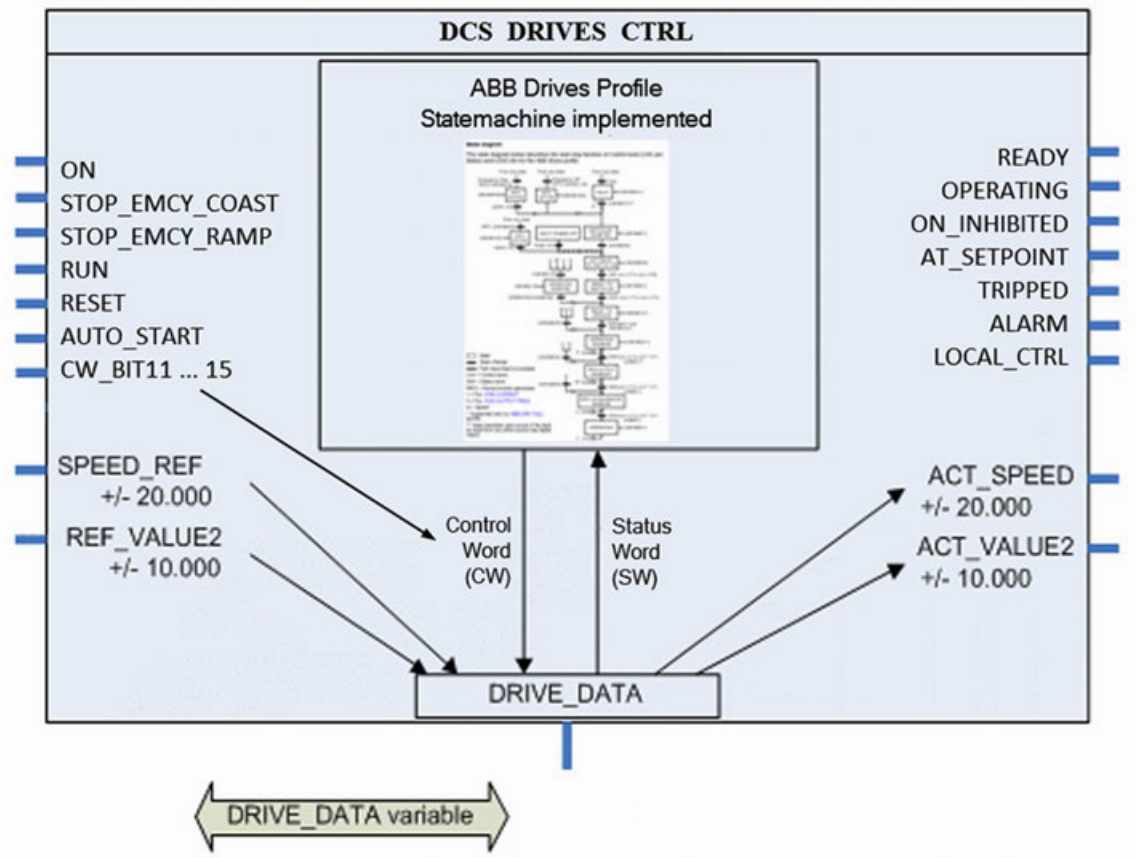


Function block DCS_DRIVES_CTRL is used for controlling DCS Drives with ABB Drives Profile connected to a communication block via a DRIVE_DATA variable.

Block data

Available as of runtime system:	V2.3.3
Included in library:	DCSDrives_AC500_V24.lib
Function block	with historical values

Description



The function block DCS_DRIVES_CTRL is used for controlling DCS Drives with ABB Drives profile.

The function block provides standard start/stop signals to control the drive and standard diagnosis signals read from the drive.

According to the input signals and the actual Status Word (SW), read from the DRIVE_DATA variable, the ABB drives profile state machine is executed. The Control Word (CW) is build and written to the DRIVE_DATA variable. For diagnosis purpose the CW is also written to the output USED_CW.

SPEED_REF input has to be given in the fieldbus range of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355).ACT_SPEED provides the actual speed in the fieldbus range of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355). ACS_REF_SCALING function block could be used to scale the fieldbus range to a physical value.

If the connected communication block (via DRIVE_DATA variable) is disabled or not parameterized correctly, all outputs except DONE, ERR and ERNO are reset to zero. This can be checked by the ERNO output.

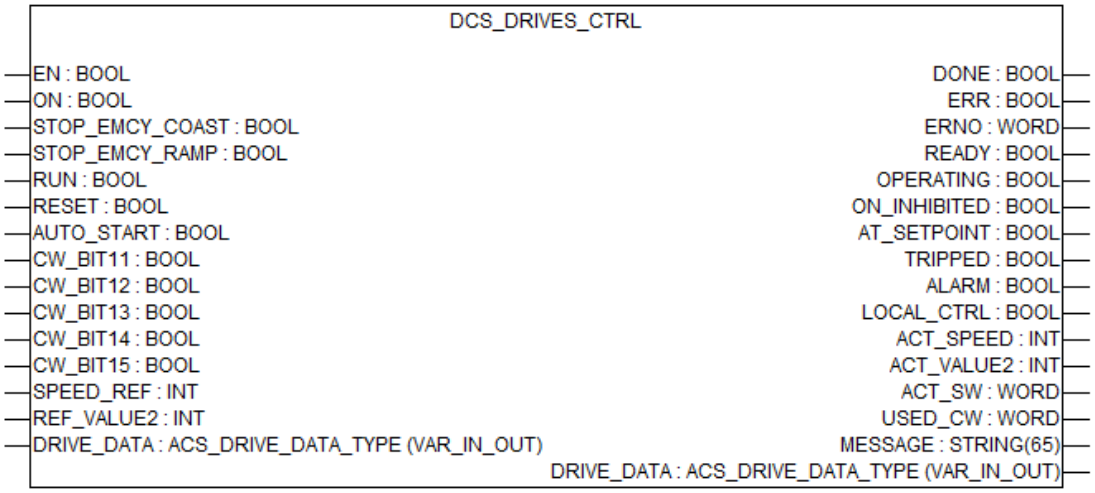
Preconditions

The function block is only working for DCS drives using ABB Drive profile. The data transfer to the drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to an ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301* or an ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* function block.

For the necessary configuration of parameters in the drive see table below.

Drive Parameter	DCS800	DCS550	Comment
EXT1 COM-MANDS	10.01 = ManCtrlWord	10.01 = ManCtrlWord	Fieldbus interface as source for start and stop
EXT1/EXT2 SEL	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bitt11 11.12 (Ref2Mux) Invert 11.02	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bitt11 11.12 (Ref2Mux) Invert 11.02	Fieldbus interface as source to switch to EXT2 control place. Tbd / tbc
REF1 SELECT	11.03 = SpeedRef2301	11.03 = Spee-dRef2301	Fieldbus interface as source for speed refer-ence
FAULT RESET SEL	NA	NA	Fieldbus interface as source for fault reset
PROFILE	NA	NA	Control Profile to ABB Drives Profile classic or enhanced.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0, with the exception of the USED_CW output to 1024 (hex 0400 - only remote bit).

ON (switch on)

Data type: BOOL

With a rising edge of the input ON (FALSE->TRUE) the DCS Drive Contactors are closed, field exciter and fans are started. (depending on the drive configuration, it might directly starts, if RUN = TRUE). (bit 0 of the Control Word On / OFF1N)

If ON = FALSE, the drive is stopped via Off1Mode (21.02). After reaching zero speed, the main Contactors will be switched off.

According to the ABB Drives Profile a new rising edge of input ON will be ignored, until zero speed was reached.

With the AUTO_START function an automatically internally rising edge of this input can be generated to avoid waiting for and detecting zero speed.

STOP_EMCY_COAST (stop emergency coast)

Data type: BOOL, Default value: TRUE

STOP_EMCY_COAST=TRUE enables normal operation of the drive.

Input STOP_EMCY_COAST=FALSE will coast the drive (bit 1 of the Control Word OFF2). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_RAMP (stop emergency ramp)

Data type: BOOL, Default value: TRUE

STOP_EMCY_RAMP = TRUE enables normal operation of the drive.

Input STOP_EMCY_RAMP = FALSE will stop the drive along the emergency ramp, defined in the drive (bit 2 of the Control Word OFF3). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_RAMP (stop emergency ramp)

Data type: BOOL, Default value: TRUE

STOP_EMCY_RAMP = TRUE enables normal operation of the drive.

Input STOP_EMCY_RAMP = FALSE will stop the drive along the emergency ramp, defined in the drive (bit 2 of the Control Word OFF3). A new rising edge of the START input is needed to start the drive again.

RUN (run)

Data type: BOOL

Input RUN = TRUE will start the drive (bit 3 of the Control Word RUN / INHIBIT_OP).

RUN = FALSE will stop the drive depending on the StopMode configuration (Par. 21.03)

Depending on the drive configuration a new rising edge of the input ON might be needed to restart the drive. With the AUTO_START function an automatically internally rising edge of this input can be generated each 1 sec.

RESET (reset)

Data type: BOOL

Input RESET is used to reset the drive (bit 7 in the Control Word RESET).

RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

For DCS drives to reset the drive the input ON and the input RUN must be reset to FALSE.

CW_BIT11 (Control Word bit 11)

Data type: BOOL

Input CW_BIT11 = TRUE sets the Control Word Bit11. The meaning is according to the user specific drive configuration.

CW_BIT12 (Control Word bit 12)

Data type: BOOL

Input CW_BIT12 = TRUE sets the Control Word Bit12. The meaning is according to the user specific drive configuration.

CW_BIT13 (Control Word bit 13)

Data type: BOOL

Input CW_BIT13 = TRUE sets the Control Word Bit13. The meaning is according to the user specific drive configuration.

CW_BIT14 (Control Word bit 14)

Data type: BOOL

Input CW_BIT14 = TRUE sets the Control Word Bit14. The meaning is according to the user specific drive configuration.

CW_BIT15 (Control Word bit 15)

Data type: BOOL

Input CW_BIT15 = TRUE sets the Control Word Bit15. The meaning is according to the user specific drive configuration.

AUTO_START (auto start)

Data type: BOOL

Input AUTO_START enables the auto start function of the function block.

The AUTO_START function internally creates cyclically switch on / start commands on the Control Word. So that the ON and RUN input of the function block can be used as level sensitive inputs.

They don´t need to be reset to FALSE and back to TRUE to restart the drive after a normal stop.

The AUTO_START function is not working, if the stop was caused by an emergency stop, e.g. STOP_EMCY_COAST or STOP_EMCY_RAMP input or STO. The time of cyclically retries is 1second.

WARNING!

Automatically start!

Be aware, that the drive starts automatically, if the AUTO_START function is enabled and additionally ON, RUN, STOP_EMCY_COAST and STOP_EMCY_RAMP inputs are set to TRUE.

Details of implementation:

The function will internally reset the ON or RUN bit of the Control Word to create a rising edge on one of these bits to the drive.

The ON bit (bit 0 of Control Word OFF1) is reset to FALSE in case that the ON input is set, but the RDY_ON (bit 0 of Status Word) feedback from drive was still missing after 1sec. Then the ON bit (bit 0 of Control Word OFF1) is set to the value of the ON input again. The RUN bit (bit 3 of Control Word INHIBIT_OP) is reset to FALSE in case that the ON and RUN inputs are set, but the RDY_REF (bit 2 of Status Word) feedback from drive is still missing. Then after 1 sec the RUN bit (bit 3 of Control Word INHIBIT_OP) is set to the value of the RUN input again.

Use Case:

This function is especially useful in e.g. the following situation:

- The drive is running and local control is enabled by the panel on the drive. There the drive is stopped and after a time the control is switched back to the PLC.
  Then the auto start function restarts the drive automatically without the need to give an external rising edge to the input ON and RUN.

SPEED_REF (speed reference)

Data type: INT

Input SPEED_REF must be given in fieldbus equivalent value between -20000 .. +20000.

20000 = the value, defined in the drive as Ref1 Max (e.g. Par.11.05 Ref1 Max for ACS3XX).

Tbc for DCS and RMBA: For drive types ACS3XX, ACS550 and ACH550 using a Modbus RTU connection the speed reference value send to the drive is limited to 20000 or +20000. If the input exceeds this limits, the ERR output is set to TRUE, output ERNO is set to the message that indicates that the input of SPEED_REF is out of range.

Nevertheless the function block is processed normally.

REF_VALUE2 (reference value 2)

Data type: INT

Input REF_VALUE2 must be given in fieldbus equivalent value between -10000 .. +10000.

10000 = the value defined in the drive as Ref2 Max (e.g. Par.11.08 Ref2 Max for ACS3XX).

Tbc for DCS and RMBA: For the drive types ACS3XX, ACS550 and ACH550 using a Modbus RTU connection the Reference Value2, send to the drive is limited to 10000 or + 10000. If the input exceeds this limits, the ERR output is set to TRUE, output ERNO is set to the message that indicates that the input of REF_VALUE2 is out of range.

Nevertheless the function block is processed normally.

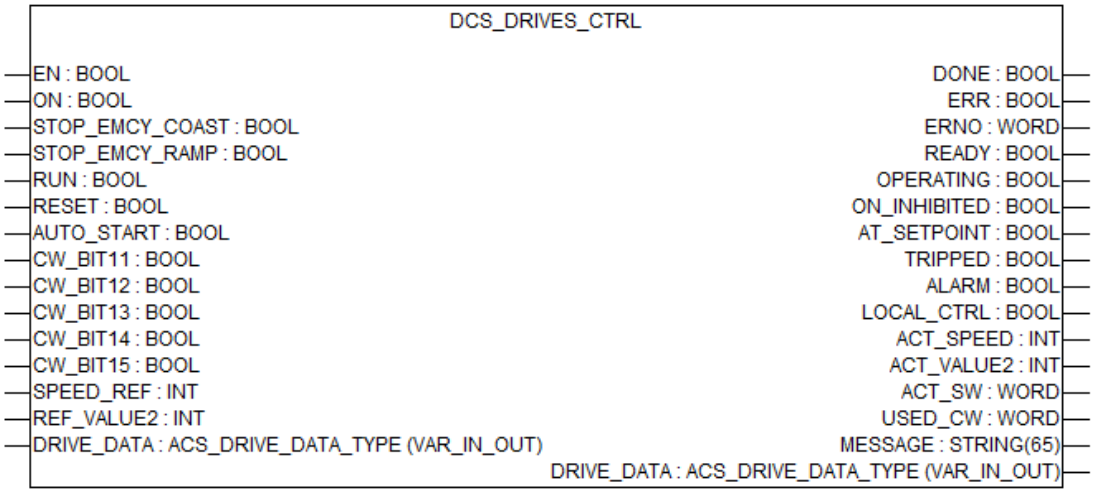
DRIVE_DATA (drive data)

Data type: ACS_DRIVE_DATA_TYPE ↳ *Chapter 1.5.6.2.6.2 “ACS_DRIVE_DATA_TYPE structure to exchange data between function blocks for 1 Drive” on page 2253*

The combined input/output DRIVE_DATA must be connected to the variable of type ACS_DRIVE_DATA_TYPE of the related ACS drive (slave). Each drive must have its own DRIVE_DATA variable.

The DRIVE_DATA variable contains the data of the drive and must be connected to all related function blocks of this drive.

Output description



DONE (done)

Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

READY (ready)

Data type: BOOL

Output READY=TRUE indicates that the drive is ready to switch on (Bit 0 in the status word from the drive).

OPERATING (operating)

Data type: BOOL

Output OPERATING=TRUE indicates, that the drive is modulating. The drive is enabled and running (Status Word of drive bit: RDY_REF = TRUE).

This indication works, even if the drive is controlled from another control place, e.g. local panel.

This is different to the ACS_DRIVES_CTRL_STANDARD or ACS_DRIVES_CTRL_STANDARD_GEN.

ON_INHIBIT (on is inhibited)

Data type: BOOL

Output ON_INHIBITED=TRUE indicates, that the OnInhibited state is active (Bit 6 in the Status Word from the drive).

This is the case after a fault or an Emergency Off / Coast Stopp (Off2) or an E-stop (Off3) or via digital input “Off2” (10.08) or E Stop (10.09).

AT_SETPOINT (actual value is at setpoint)

Data type: BOOL

Output AT_SETPOINT=TRUE indicates, that the actual value (MotSpeed (1.04)) and the set-point (SpeedRef4 (2.18)) are in the tolerance zone.

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates, that the drive is tripped (Bit 3 in the Status Word from the drive).

ALARM (alarm)

Data type: BOOL

Output ALARM=TRUE indicates, that the drive has an alarm (Bit 7 in the Status Word from the drive).

LOCAL_CTRL (local control)

Data type: BOOL

Output LOCAL_CTRL = TRUE indicates, that the drive is not controlled from remote control e.g. PLC (LOCAL_CTRL = inverted bit 9 in the Status Word from the drive - REMOTE). LOCAL_CTRL might be set to TRUE due to no connection to the drive or the drive was set to local control via the drive panel or a drive configuration tool from PC.

ACT_SPEED (actual speed)

Data type: INT

Output ACT_SPEED returns the actual speed value from the drive. The scaling depends on the drive settings.

ACT_VALUE2 (actual value 2)

Data type: INT

Output ACT_VALUE2 returns the value from the drive parameterized in the drive for Actual Value2 on the fieldbus.

ACT_SW (actual status word)

Data type: WORD

Output ACT_SW returns the actual Status Word read from the drive.

USED_CW (used control word)

Data type: WORD

Output USED_CW returns the Control Word, that was build in the function block due to the inputs and Status Word according the ABB Drives Profile state machine.

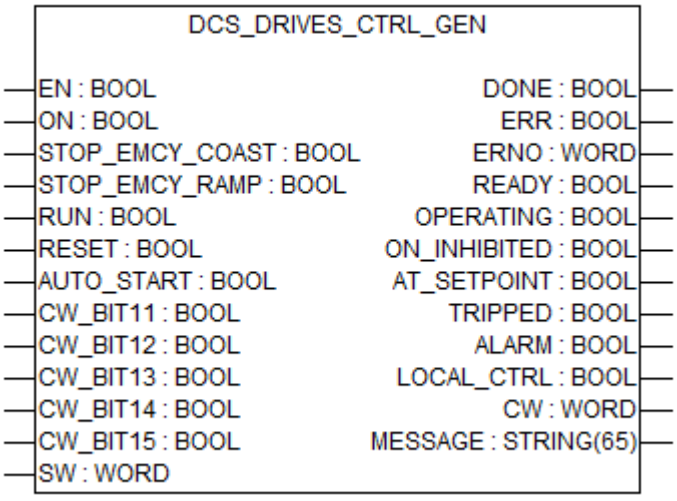
The USED_CW is sent to the drive as Control Word.

MESSAGE (message)

Data type: STRING

Output MESSAGE gives information about the actual state of the function block. This string also indicates what would be the next steps to continue to start the drive, or which signal from the drive is missing.

DCS_DRIVES_CTRL_GEN control of DCS drives with ABB-Drives profile via generic fieldbus

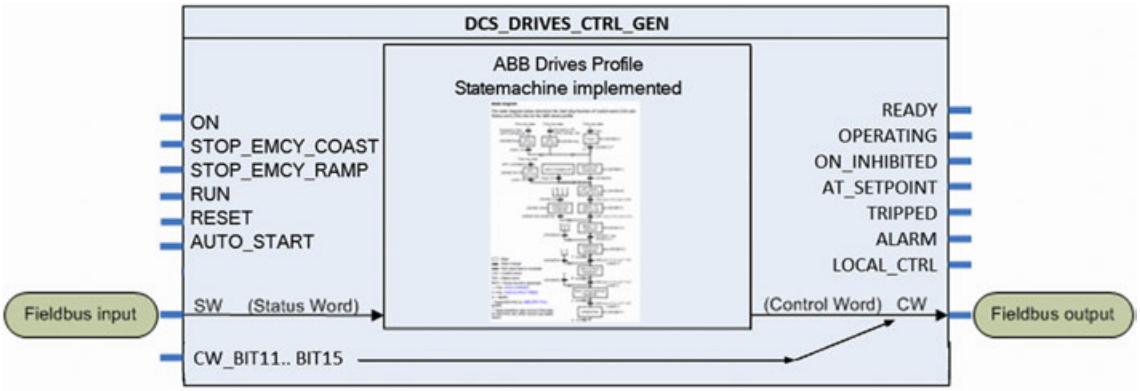


Function block DCS_DRIVES_CTRL is used for controlling DCS Drives with ABB Drives profile connected to a communication block via a DRIVE_DATA variable.

Block data

Available as of runtime system:	V2.3.3
Included in library:	DCSDrives_AC500_V24.lib
Function block	with historical values

Description



The function block DCS_DRIVES_CTRL is used for controlling DCS Drives with ABB Drives profile.

The function block provides standard start/stop signals to control the drive and standard diagnosis signals read from the drive.

According to the input signals and the actual Status Word (SW), read from the DRIVE_DATA variable, the ABB Drives profile state machine is executed. The Control Word (CW) is build and written to the DRIVE_DATA variable. For diagnosis purpose the CW is also written to the output USED_CW.

SPEED_REF input has to be given in the fieldbus range of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355).ACT_SPEED provides the actual speed in the fieldbus range of -20000 .. +20000, according to the scaling parameter in the drive (e.g. Par. 11.05 of ACS355). ACS_REF_SCALING function block could be used to scale the fieldbus range to a physical value.

If the connected communication block (via DRIVE_DATA variable) is disabled or not parameterized correctly, all outputs except DONE, ERR and ERNO are reset to zero. This can be checked by the ERNO output.

Preconditions

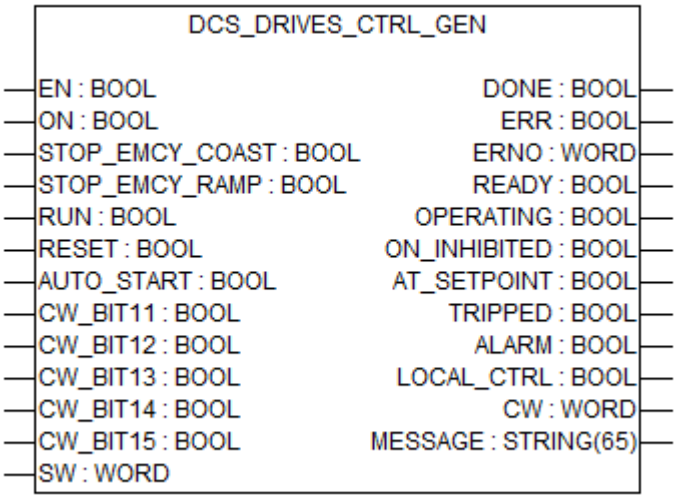
The function block is only working for DCS drives using ABB Drive profile. The data transfer to the drive is realized via the IN_OUT variable DRIVE_DATA, which must be connected to an ACS_COM_MOD_RTU ↳ *Chapter 1.5.6.3.5.2 “ACS_COM_MOD_RTU communication for ACS / DCS drives via Modbus RTU” on page 2301* or an ACS_COM_MOD_TCP ↳ *Chapter 1.5.6.4.3.1 “ACS_COM_MOD_TCP communication for ACS / DCS drives via Modbus TCP” on page 2360* function block.

Table 138: Necessary configuration of parameters in the drive

Drive Parameter	DCS800	DCS550	Comment
EXT1 COMMANDS	10.01 = ManCtrlWord	10.01 = ManCtrlWord	Fieldbus interface as source for start and stop
EXT1/EXT2 SEL	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bitt11 11.12 (Ref2Mux) Invert 11.02	10.07 (HandAuto) MCW: Bit11 11.02 (Ref1Mux) MCW: Bitt11 11.12 (Ref2Mux) Invert 11.02	Fieldbus interface as source to switch to EXT2 control place. Tbd / tbc

Drive Parameter	DCS800	DCS550	Comment
REF1 SELECT	11.03 = SpeedRef2301	11.03 = Spee-dRef2301	Fieldbus interface as source for speed refer-ence
FAULT RESET SEL	NA	NA	Fieldbus interface as source for fault reset
PROFILE	NA	NA	Control Profile to ABB Drives Profile classic or enhanced.

Input description



EN (enable) Data type: BOOL

The function block is activated (EN = TRUE) or deactivated (EN = FALSE) via input EN.

If the function block is active, the current values are available at the outputs.

If the function block has been deactivated, all outputs are set to 0, with the exception of the CW output to 1024 (hex 0400 - only remote bit).

ON (switch on) Data type: BOOL

With a rising edge of the input ON (FALSE->TRUE) the DCS Drive Contactors are closed, field exciter and fans are started. (depending on the drive configuration, it might directly starts, if RUN = TRUE). (bit 0 of the Control Word On / OFF1N)

If ON = FALSE, the drive is stopped via Off1Mode (21.02). After reaching zero speed, the main Contactors will be switched off.

According to the ABB Drives Profile a new rising edge of input ON will be ignored, until zero speed was reached.

With the AUTO_START function an automatically internally rising edge of this input can be generated to avoid waiting for and detecting zero speed.

STOP_EMCY_COAST (stop emergency coast)

Data type: BOOL, Default value: TRUE

STOP_EMCY_COAST=TRUE enables normal operation of the drive.

Input STOP_EMCY_COAST=FALSE will coast the drive (bit 1 of the Control Word OFF2). A new rising edge of the START input is needed to start the drive again.

STOP_EMCY_RAMP (stop emergency ramp)

Data type: BOOL, Default value: TRUE

STOP_EMCY_RAMP = TRUE enables normal operation of the drive.

Input STOP_EMCY_RAMP = FALSE will stop the drive along the emergency ramp, defined in the drive (bit 2 of the Control Word OFF3). A new rising edge of the START input is needed to start the drive again.

RUN (run)

Data type: BOOL

Input RUN = TRUE will start the drive (bit 3 of the Control Word RUN / INHIBIT_OP).

RUN = FALSE will stop the drive depending on the StopMode configuration (Par. 21.03)

Depending on the drive configuration a new rising edge of the input ON might be needed to restart the drive. With the AUTO_START function an automatically internally rising edge of this input can be generated each 1 sec.

RESET (reset)

Data type: BOOL

Input RESET is used to reset the drive (bit 7 in the Control Word RESET).

RESET = TRUE resets faults and warnings in the drive. It does not reset the function block itself.

For DCS drives to reset the drive the input ON and the input RUN must be reset to FALSE.

CW_BIT11 (Control Word bit 11)

Data type: BOOL

Input CW_BIT11 = TRUE sets the Control Word Bit11. The meaning is according to the user specific drive configuration.

CW_BIT12 (Control Word bit 12)

Data type: BOOL

Input CW_BIT12 = TRUE sets the Control Word Bit12. The meaning is according to the user specific drive configuration.

CW_BIT13 (Control Word bit 13)

Data type: BOOL

Input CW_BIT13 = TRUE sets the Control Word Bit13. The meaning is according to the user specific drive configuration.

CW_BIT14 (Control Word bit 14)

Data type: BOOL

Input CW_BIT14 = TRUE sets the Control Word Bit14. The meaning is according to the user specific drive configuration.

CW_BIT15 (Control Word bit 15)

Data type: BOOL

Input CW_BIT15 = TRUE sets the Control Word Bit15. The meaning is according to the user specific drive configuration.

AUTO_START (auto start)

Data type: BOOL

Input AUTO_START enables the auto start function of the function block.

The AUTO_START function internally creates cyclically switch on / start commands on the Control Word. So that the ON and RUN input of the function block can be used as level sensitive inputs.

They don´t need to be reset to FALSE and back to TRUE to restart the drive after a normal stop.

The AUTO_START function is not working, if the stop was caused by an emergency stop, e.g. STOP_EMCY_COAST or STOP_EMCY_RAMP input or STO. The time of cyclically retries is 1second.

WARNING!

Automatically start!

Be aware, that the drive starts automatically, if the AUTO_START function is enabled and additionally ON, RUN, STOP_EMCY_COAST and STOP_EMCY_RAMP inputs are set to TRUE.

Details of implementation:

The function will internally reset the ON or RUN bit of the Control Word to create a rising edge on one of these bits to the drive.

The ON bit (bit 0 of Control Word OFF1) is reset to FALSE in case that the ON input is set, but the RDY_ON (bit 0 of Status Word) feedback from drive was still missing after 1sec. Then the ON bit (bit 0 of Control Word OFF1) is set to the value of the ON input again. The RUN bit (bit 3 of Control Word INHIBIT_OP) is reset to FALSE in case that the ON and RUN inputs are set, but the RDY_REF (bit 2 of Status Word) feedback from drive is still missing. Then after 1 sec the RUN bit (bit 3 of Control Word INHIBIT_OP) is set to the value of the RUN input again.

Use Case:

This function is especially useful in e.g. the following situation:

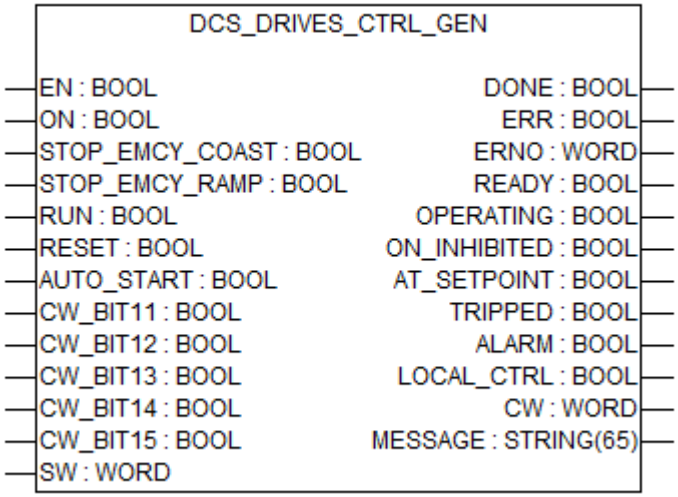
- The drive is running and local control is enabled by the panel on the drive. There the drive is stopped and after a time the control is switched back to the PLC.
  Then the auto start function restarts the drive automatically without the need to give an external rising edge to the input ON and RUN.

SW (Status Word)

Data type: WORD

Input Status Word from drive. Connect Status Word from fieldbus to this input.

Output description



DONE (done) Data type: BOOL

Output DONE indicates the state of the job processing. After completing or aborting the processing (due to an error), DONE is set to TRUE for one cycle. For that reason, the output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during block processing. This output always has to be considered together with output DONE. If DONE = TRUE and ERR = TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD

Output ERNO provides an error identifier, if an invalid value was applied to an input or if an error occurred during request processing. ERNO always has to be considered together with the outputs DONE and ERR. The value output at ERNO is only valid, if DONE is TRUE and ERR is TRUE.

The encoding of the error messages output at ERNO is explained in the chapter ↬ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*.

READY (ready) Data type: BOOL

Output READY=TRUE indicates that the drive is ready to switch on (Bit 0 in the status word from the drive).

OPERATING (operating) Data type: BOOL

Output OPERATING=TRUE indicates, that the drive is modulating. The drive is enabled and running (Status Word of drive bit: RDY_REF = TRUE).

This indication works, even if the drive is controlled from another control place, e.g. local panel.

This is different to the ACS_DRIVES_CTRL_STANDARD or ACS_DRIVES_CTRL_STANDARD_GEN.

ON_INHIBIT (on is inhibited)

Data type: BOOL

Output ON_INHIBITED=TRUE indicates, that the OnInhibited state is active (Bit 6 in the Status Word from the drive).

This is the case after a fault or an Emergency Off / Coast Stopp (Off2) or an E-stop (Off3) or via digital input “Off2” (10.08) or E Stop (10.09).

AT_SETPOINT (actual value is at setpoint)

Data type: BOOL

Output AT_SETPOINT=TRUE indicates, that the actual value (MotSpeed (1.04)) and the set-point (SpeedRef4 (2.18)) are in the tolerance zone.

TRIPPED (tripped)

Data type: BOOL

Output TRIPPED=TRUE indicates, that the drive is tripped (Bit 3 in the Status Word from the drive).

ALARM (alarm)

Data type: BOOL

Output ALARM=TRUE indicates, that the drive has an alarm (Bit 7 in the Status Word from the drive).

LOCAL_CTRL (local control)

Data type: BOOL

Output LOCAL_CTRL = TRUE indicates, that the drive is not controlled from remote control e.g. PLC (LOCAL_CTRL = inverted bit 9 in the Status Word from the drive - REMOTE). LOCAL_CTRL might be set to TRUE due to no connection to the drive or the drive was set to local control via the drive panel or a drive configuration tool from PC.

CW (Control Word)

Data type: WORD

Output CW returns the Control Word that was build in the function block using the inputs and Status Word according to the ABB Drives Profile state machine.

Connect CW directly to the fieldbus. The CW output will keep it´s value, even if the drive is not controlled from this block anymore. E.g., if local control or EXT2 is enabled.

This is different to the ACS_DRIVES_CTRL_STANDARD or ACS_DRIVES_CTRL_STANDARD_GEN.

MESSAGE (message)

Data type: STRING

Output MESSAGE gives information about the actual state of the function block. This string also indicates what would be the next steps to continue to start the drive, or which signal from the drive is missing.

1.5.6.8.5 Enumerations

DCS_DRIVE_ENUM enumerations to be used at the input DRIVE_TYPE of ACS_COM_xxx function blocks

Enumerations DCS_DRIVE_ENUM can be used at the input of any communication block ACS_COM_xxx e.g. ACS_COM_MOD_RTU, ACS_COM_MOD_TCP or ACS_COM_PB.

Table 139: Enumeration data

Available as of runtime system:	V1.3.2
Included in library:	DCSDrives_AC500_V24.lib

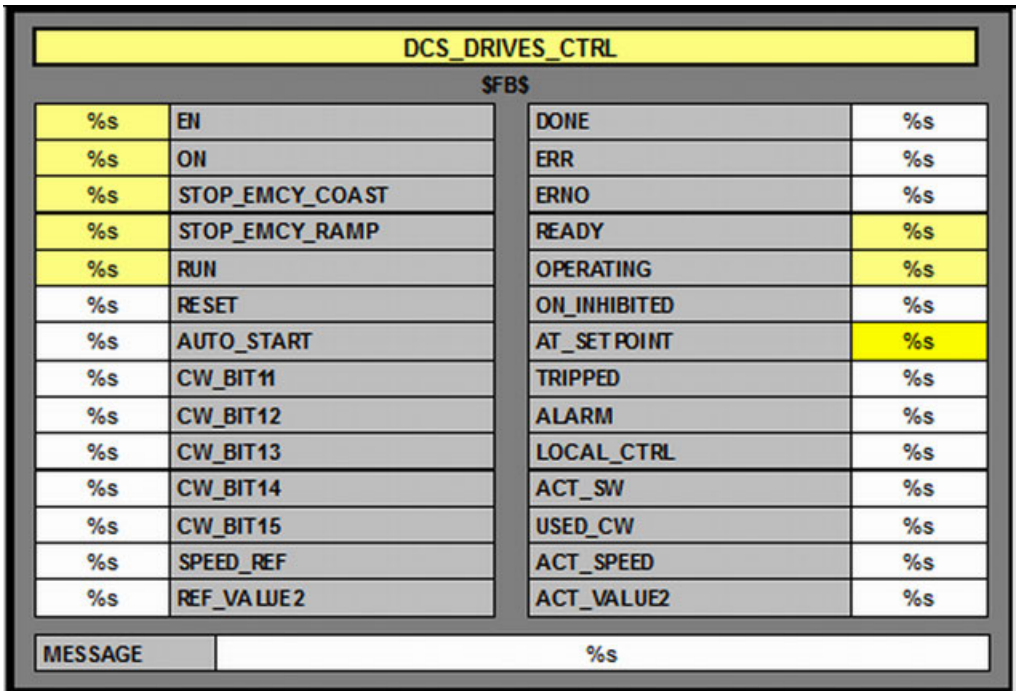
Table 140: ENUMs

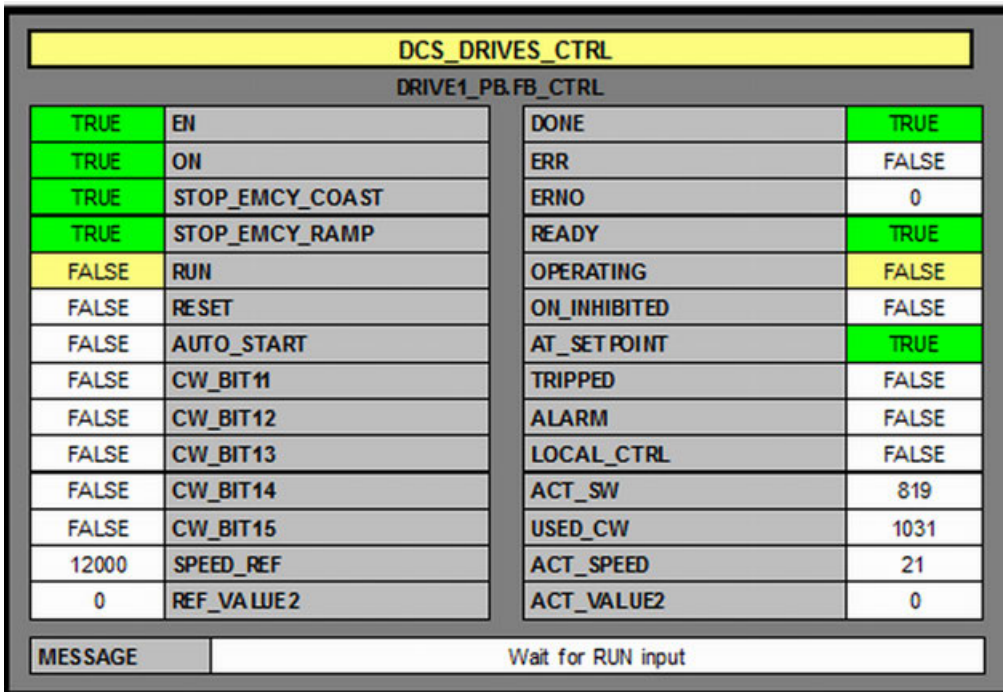
Enumeration	Type	Value	Description
DCS_DRIVE_DCS800	INT	12	
DCS_DRIVE_DCS550	INT	13	

These Enumerations can be used at the input DRIVE_TYPE of any communication block ACS_COM_xxx e.g. ACS_COM_MOD_RTU, ACS_COM_MOD_TCP or ACS_COM_PB, for a better reading.

1.5.6.8.6 Visualization

DCS_DRIVES_CTRL_VISU_PH faceplate of function block DCS_DRIVES_CTRL





Visualization element DCS_DRIVES_CTRL_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an DCS_DRIVES_CTRL function block, which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

Visualization information

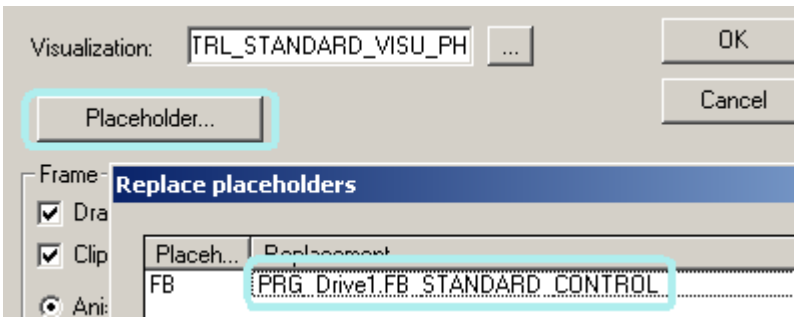
Available in runtime system:	V1.3.2
Included in library:	DCSDrivesBase_AC500_V24.lib

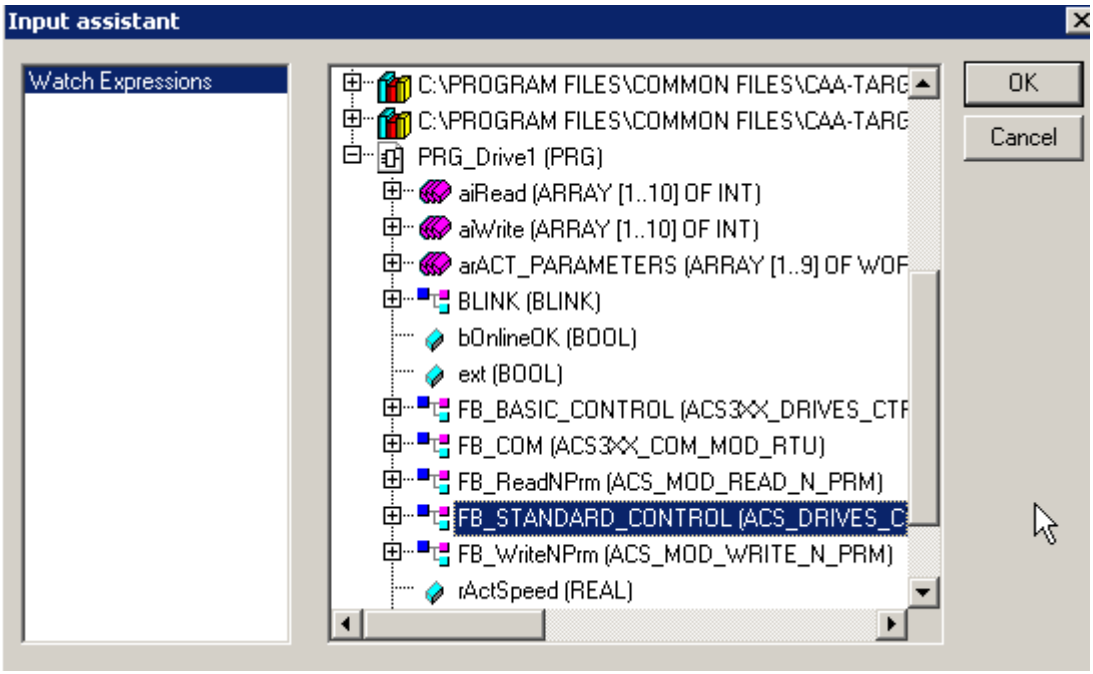
Visualization description

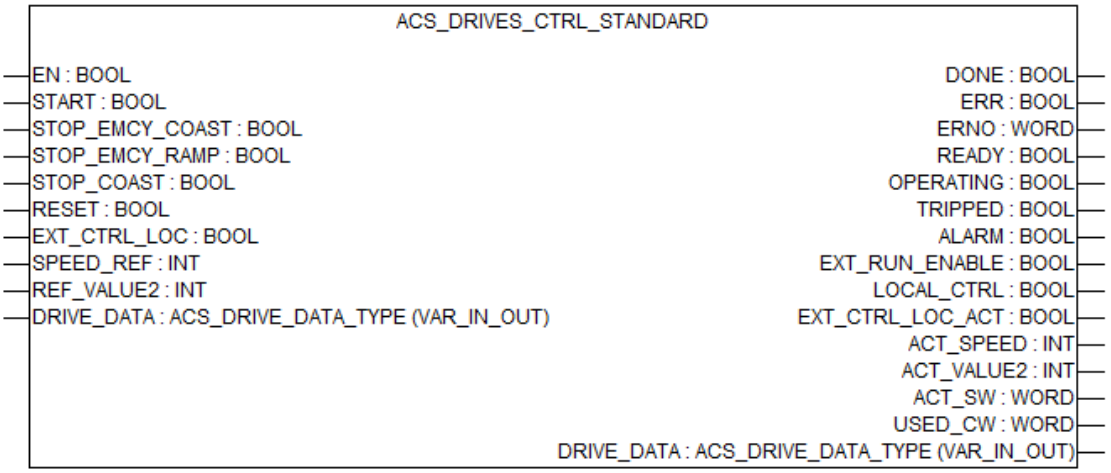
Visualization element DCS_DRIVES_CTRL_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an DCS_DRIVES_CTRL function block which instance was used to replace the placeholder FB.

All inputs of that DCS_DRIVES_CTRL function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

Additionally, a text message of the actual state, e.g. missing input or missing feedback from drive, is given by the variable MESSAGE, shown next to label MESSAGE.







Parameters

Access R/W

EN Access via: Toggle
Description: EN input

ON Access via: Toggle
Description: ON input

EMCY_COST Access via: Toggle
Description: EMCY_COAST input

EMCY_RAMP Access via: Toggle
Description: EMCY_RAMP input

RUN Access via: Toggle
Description: RUN input

RESET Access via: Toggle
Description: RESET input

CW_BIT 11 Access via: Toggle
Description: CW_BIT 11 input

CW_BIT 12 Access via: Toggle
Description: CW_BIT 12 input

CW_BIT 13 Access via: Toggle
Description: CW_BIT 13 input

CW_BIT 14 Access via: Toggle
Description: CW_BIT 14 input

CW_BIT 15 Access via: Toggle
Description: CW_BIT 15 input

AUTO_START Access via: Toggle
Description: AUTO_START input

SPEED_REF Access via: Text, -32768...+32768
Description: SPEED_REF input

REF_VALUE2 Access via: Text, -32768...+32768
Description: REF_VALUE 2 input

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_BASIC_CTRL

Access R

DONE Description: DONE output

ERR Description: ERR output

ERNO Description: ERNO output

READY Description: READY output

OPERATING Description: OPERATING output

ON_INHIBITED Description: ON_INHIBITED output

AT_SETPOINT Description: AT_SETPOINT output

TRIPPED Description: TRIPPED output

ALARM Description: ALARM output

LOC_CTRL Description: LOC_CTRL output

ACT_SPEED Description: ACT_SPEED output

ACT_VALUE2 Description: ACT_VALUE 2 output

MESSAGE Description: MESSAGE output

Colors

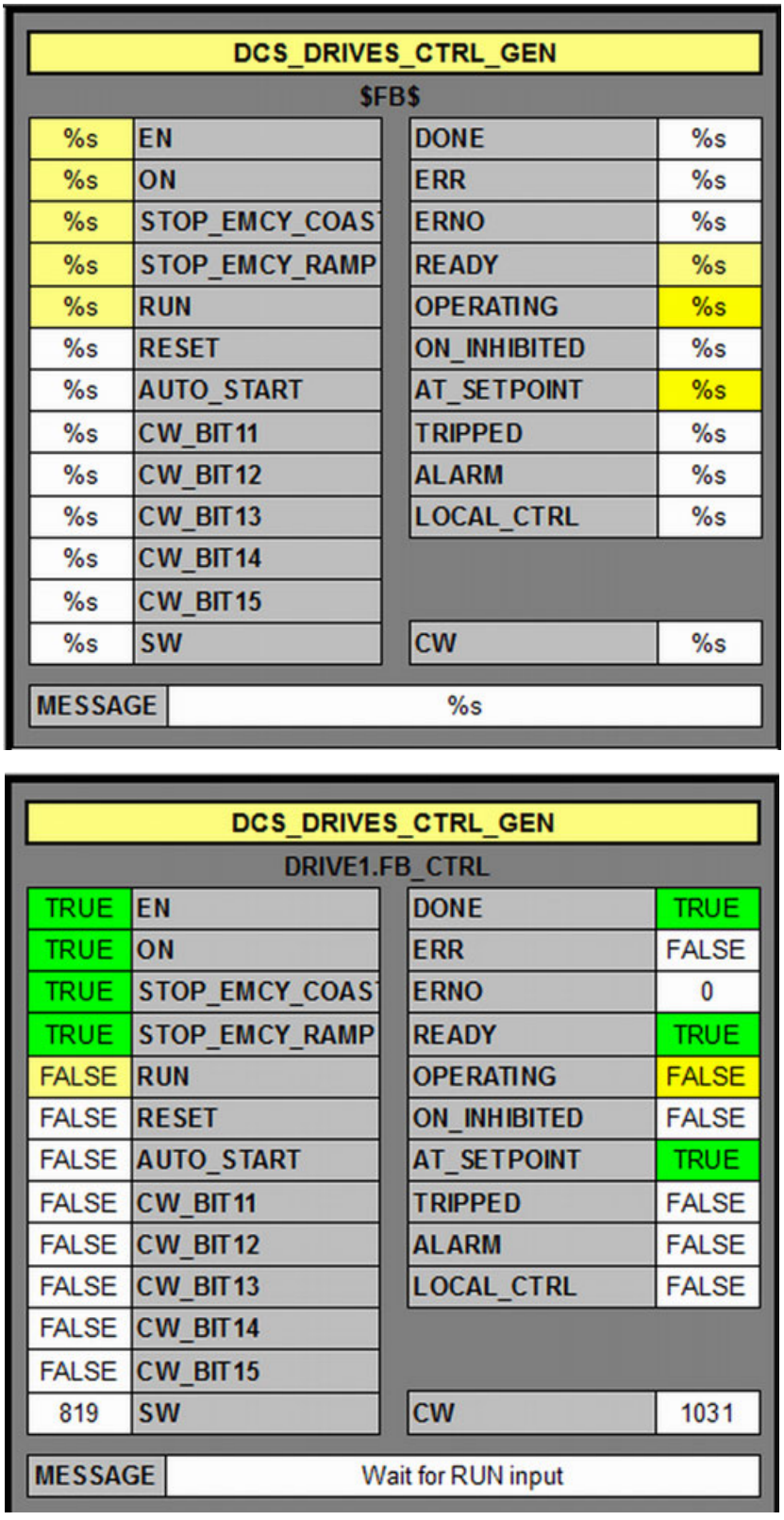
The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

DCS_DRIVES_CTRL_GEN_VISU_PH faceplate of function block DCS_DRIVES_CTRL_GEN



Visualization element DCS_DRIVES_CTRL_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an DCS_DRIVES_CTRL_GEN function block, which instance was used to replace the placeholder FB.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

Visualization information

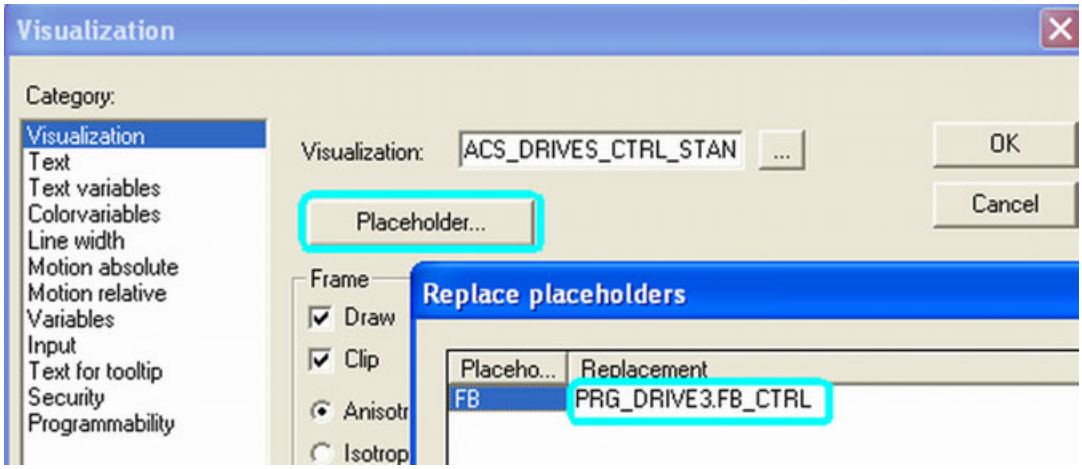
Available in runtime system:	V1.3.2
Included in library:	DCSDrivesBase_AC500_V24.lib

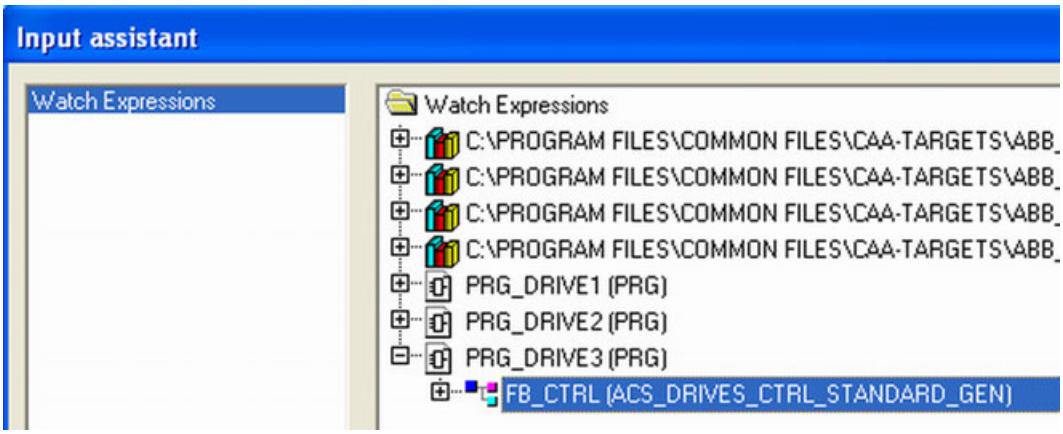
Visualization description

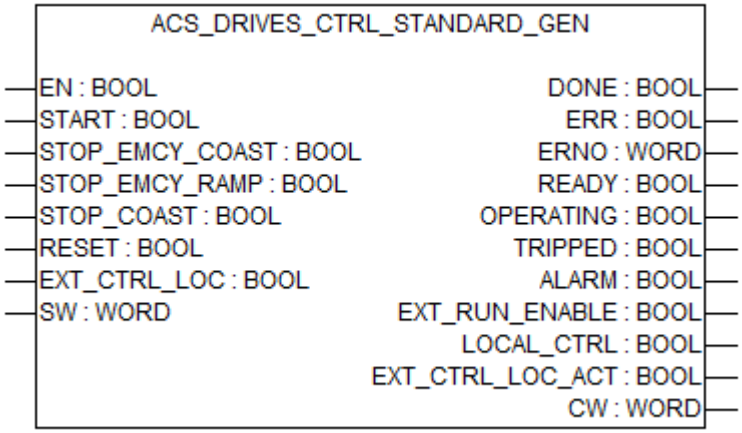
Visualization element DCS_DRIVES_CTRL_GEN_VISU_PH can be used to show the actual values of all inputs and outputs of the instance of an DCS_DRIVES_CTRL_GEN function block which instance was used to replace the placeholder FB.

All inputs of that DCS_DRIVES_CTRL_GEN function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open. The DRIVE_DATA variable must be connected to the function block.

Additionally, a text message of the actual state, e.g. missing input or missing feedback from drive, is given by the variable MESSAGE, shown next to label MESSAGE.







Parameters

Access R/W

EN	Access via: Toggle Description: EN input
ON	Access via: Toggle Description: ON input
EMCY_COST	Access via: Toggle Description: EMCY_COAST input
EMCY_RAMP	Access via: Toggle Description: EMCY_RAMP input
RUN	Access via: Toggle Description: RUN input
RESET	Access via: Toggle Description: RESET input
CW_BIT 11	Access via: Toggle Description: CW_BIT 11 input
CW_BIT 12	Access via: Toggle Description: CW_BIT 12 input
CW_BIT 13	Access via: Toggle Description: CW_BIT 13 input
CW_BIT 14	Access via: Toggle Description: CW_BIT 14 input
CW_BIT 15	Access via: Toggle Description: CW_BIT 15 input
AUTO_START	Access via: Toggle Description: AUTO_START input
SW	Access via: Text Description: SW input (Status Word)

Placeholder	Replacement	Example
FB	Instance name of the function block	PRG_Drive1.FB_BASIC_CTRL

Access R

DONE Description: DONE output

ERR Description: ERR output

ERNO Description: ERNO output

READY Description: READY output

OPERATING Description: OPERATING output

ON_INHIBITED Description: ON_INHIBITED output

AT_SETPOINT Description: AT_SETPOINT output

TRIPPED Description: TRIPPED output

ALARM Description: ALARM output

LOC_CTRL Description: LOC_CTRL output

CW Description: CW output (Control Word)

MESSAGE Description: MESSAGE output

Colors

The color of the variables have the following meaning:

white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

The color of the background can be changed by writing a value to the global variable "dwAcsVisuBackgroundColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

The color of the title can be changed by writing a value to the global variable "dwAcsVisuTitleColor" *Chapter 1.5.6.2.7.1 “dwAcsVisuBackgroundColor and dwAcsVisuTitleColor global variables to set the background and title colors for the visualization elements” on page 2255*.

1.5.6.9 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.7 BACnet B-ASC library

1.5.7.1 System technology

BACnet B-ASC library allows to integrate an AC500 processor module into a BACnet network and to exchange data between the processor module and other devices connected to BACnet network. The library provides the BACnet B-ASC profile which enables the processor module to act as a server. The processor module receives client requests, performs them and reports back the results.

You can use BACnet IP or MS/TP as communication protocol. For BACnet IP communication use a device with onboard Ethernet. For MS/TP communication use a device with onboard serial interface.

A proprietary BACnet protocol stack is embedded into some interface implementation which connects the protocol stack to system resources like operating system and communication interfaces.

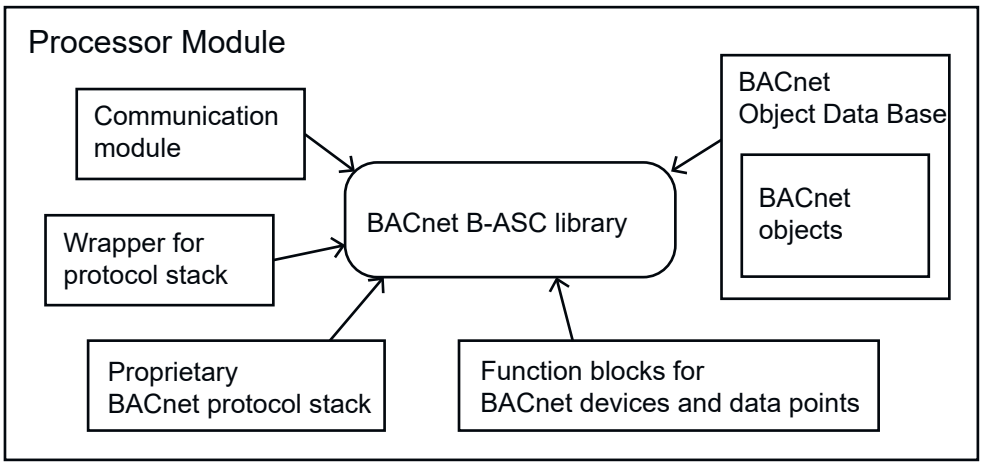
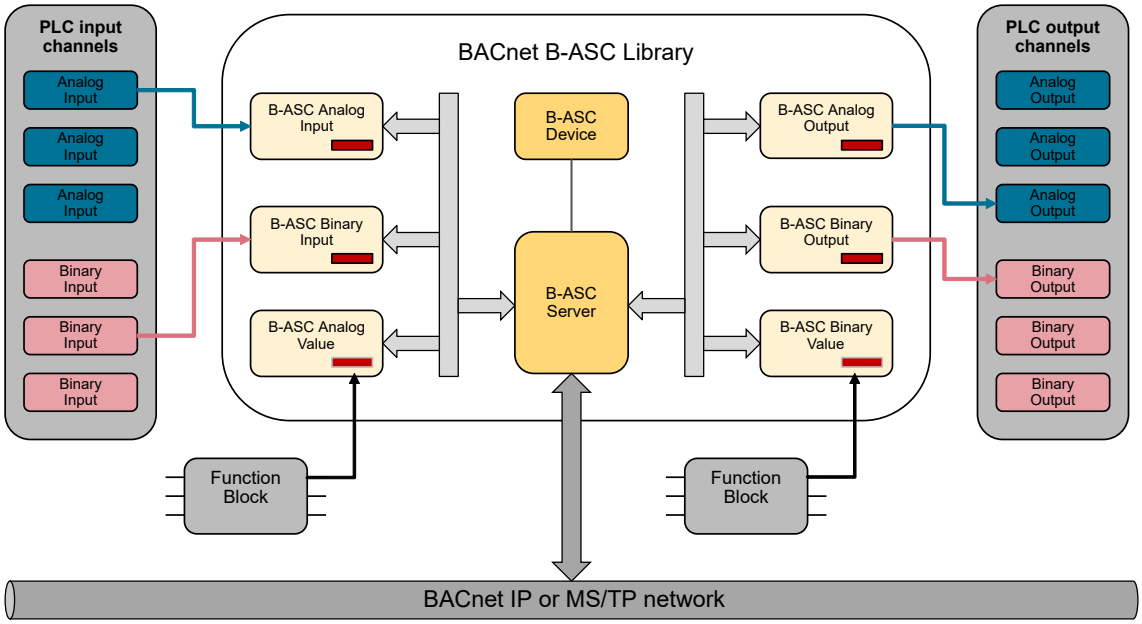


Fig. 178: BACnet B-ASC model

BACnet B-ASC library acts as follows.

- BACnet server protocol stack runs in AC500 system task context.
- BACnet request messages are received and responded via IP or serial interface.
- Execution of BACnet protocol state machine by protocol stack.
- Servicing received requests in detail by BACnet B-ASC library object implementation.

The BACnet B-ASC library provides a set of function blocks representing the server, device and data points. The function blocks are the programming interface to the user application.



Function blocks BASC_SERVER and BASC_DEVICE are mandatory. The other function blocks are optional *Chapter 1.5.7.2 “Function blocks” on page 2495*.

When initializing the function blocks via downloading to the processor module, the function blocks evaluate their inputs OBJ_ID and OBJ_NAME and register at the object database. Later changes at these inputs are not evaluated. With each further call by the application the function blocks synchronize the inputs and outputs with the associated BACnet property `Present_Value`. Received BACnet requests will be serviced by the protocol stack. If an object is addressed that is registered at the object database the request is routed to this object to be handled in detail.

The communication module provides a network layer which acts as interface between proprietary BACnet protocol stack and communication drivers.

Configuration in Automation Builder

For BACnet IP no special configuration is needed. The onboard Ethernet interface will be selected via function block input.

For BACnet MS/TP

1. Configure serial interface to use with protocol "COM1 - SysLibCom".
2. Adjust the settings as it is required for RS-485 communications. For further details see *Chapter 1.6.5.2.11 “Serial interfaces COM1 and COM2” on page 6098*.

Task configuration in CODESYS

All function blocks have to be called in tasks with cyclically processing.

You can use the function blocks with:

- PLC_PRG with automatic task configuration or manual task configuration.
- One single program or different programs.
- One single task or different tasks.

With different programs assigned to different tasks you can define different cycle times and priorities.

Executing BACnet protocol stack is decoupled from IEC task execution. BACnet protocol stack runs in a system task. IEC task cycle times and priorities do not influence receiving of requests and sending of responses. Function block BASC_SERVER does the initialization work only. Function block BASC_DEVICE provides static settings of the BACnet device only. Therefore it does not matter in which IEC task these function blocks are placed.

For the other function blocks the IEC task cycle time and priority are important. These function blocks update their present values while be called by the IEC task.

Limitations

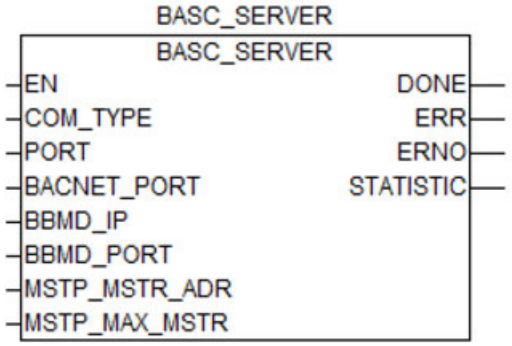
- No support of COV (change of value), alarms and events reporting.
- Supported string lengths: up to 40 characters for object names.
- Supports UTF-8 character set.
- Maximum number of object instances depends on available user memory of the processor module.
- The response of a `readProperty` service on the device object property `Object_List` can transfer maximum 93 object instances even if more objects are used. Due to limitations of used BACnet protocol stack.
- No support of message segmentation due to used BACnet protocol stack.
- The supported baud rates are 9600 Baud, 19200 Baud and 38400 Baud.

1.5.7.2 Function blocks

The function blocks in this library can only be executed in RUN mode of the processor module, not in simulation mode.

BACnet B-ASC library is based on BACnet data communication protocol according to ISO 16484-5.

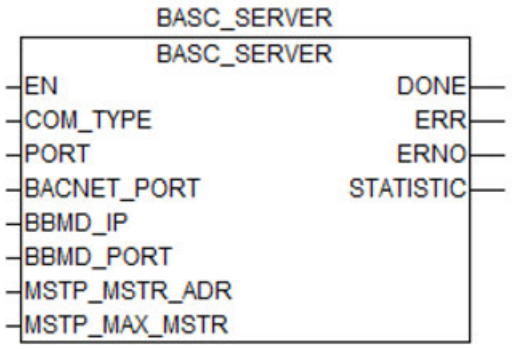
1.5.7.2.1 BASC_SERVER



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

Function block BASC_SERVER encapsulates the BACnet protocol stack. This function block can be instantiated only once. The input parameters of the function block are used to configure the BACnet protocol and the desired network type.

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. If input EN = TRUE BACnet protocol stack is initialized and system background task is started. As long as EN remains TRUE BACnet protocol stack execution is active. BACnet messages will be received and responded. The function block is not processed if input EN = FALSE.

COM_TYPE (communication type)

Data type	Default value	Range	Unit
BASC_COM_TYPE_ ENUM	BASC_IP_COM	BASC_IP_COM, BASC_MSTP_COM	-

Set communication type ↳ *"BASC_COM_TYPE_ENUM" on page 2517*.

PORT

Data type	Default value	Range	Unit
BYTE	1	Depends on the used processor module	-

Set port number.

If COM_TYPE = IP, 1 = ETH1, 2 = ETH2...

If COM_TYPE = MS/TP, 1 = COM1, 2 = COM2...

BACNET_PORT

Data type	Default value	Range	Unit
WORD	47808(16#BAC0)	0 ... 65535	-

Set BACnet port number to act at the BACnet network.

If COM_TYPE = IP, set the BACnet port number to be used.

If COM_TYPE = MS/TP, input will be ignored.

BBMD_IP (broadcast management device IP)

Data type	Default value	Range	Unit
STRING(15)	0.0.0.0	-	-

Set IP address of BACnet Brodcast Management Device to register as foreign device.

If COM_TYPE = IP, set the IP address to be used.

If COM_TYPE = MS/TP, input will be ignored.

BBMD_PORT (broadcast management device port)

Data type	Default value	Range	Unit
WORD	47808(16#BAC0)	0 ... 65535	-

Set BACnet port number of BACnet Brodcast Management Device to register as foreign device.

If COM_TYPE = IP, set the port number to be used.

If COM_TYPE = MS/TP, input will be ignored.

MSTP_MSTR_ADR (MS/TP master address)

Data type	Default value	Range	Unit
BYTE	1	1-127	-

Set BACnet master address for MS/TP protocol to act as master at the MS/TP network.

If COM_TYPE = IP, input will be ignored.

If COM_TYPE = MS/TP, set the master address to be used.

MSTP_MAX_MSTR (MS/TP maximum master address)

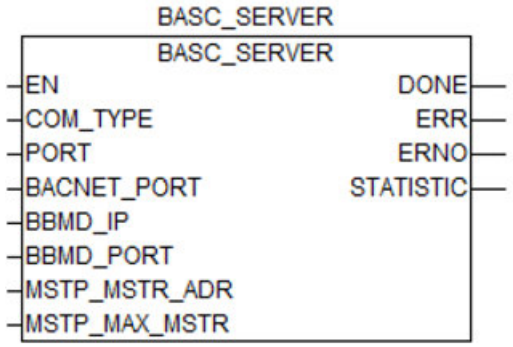
Data type	Default value	Range	Unit
BYTE	127	1-127	-

Set BACnet maximum master address for MS/TP protocol. Used when scanning the MS/TP network for other master devices.

If COM_TYPE = IP, input will be ignored.

If COM_TYPE = MS/TP, set the maximum master address to be used.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

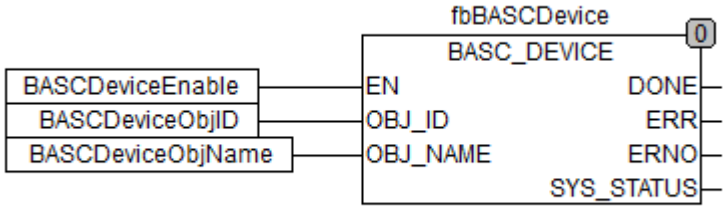
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATISTIC

Data type	Default value	Range	Unit
BASC_STA-TISTIC_COUNTER_TYPE	-	-	-

Statistic data *“BASC_STATISTIC_COUNTER_TYPE” on page 2518*.

1.5.7.2.2 BASC_DEVICE



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

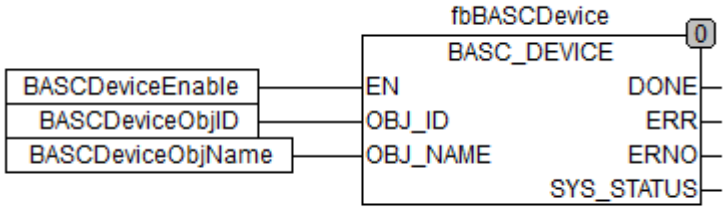
BASC_DEVICE represents the BACnet server device provided by BACnet B-ASC library. The inputs are used to configure corresponding BACnet device object properties. The outputs provide the BACnet device status.

Table 141: Device object properties provided at the BACnet network by BASC_DEVICE

Property	Characteristics	Value	BACnet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
System_Status	Updates output SYS_STATUS	-	R - read
Object_Type	Constant	DEVICE (8)	R - read
Vendor_Name	Constant	ABB	R - read
Vendor_Identifier	Constant	808	R - read
Model_Name	Constant	AC500-PM5xx	R - read
Firmware_Revision	Constant	B-ASC library V1.0.0	R - read
Application_Software_Version	Constant	0.0.0	R - read
Protocol_Version	Constant	1	R - read
Protocol_Revision	Constant	12	R - read
Protocol_Services_Supported	Constant	Who-Is / I-Am Who-Has / I-Have Read property Write property Device communication control	R - read

Property	Characteristics	Value	BACnet conformance code
Protocol_Object_Types_Supported	Constant	Analog input Analog output Analog value Binary input Binary output Binary value Device	R - read
Object_List	Depends on used objects	-	R - read
Max_APDU_Length_Accepted	Constant	480	R - read
Segmentation_Supported	Constant	No segmentation (3)	R - read
APDU_Timeout	Constant	10000	R - read
Number_Of_APDU_Retries	Constant	0	R - read
Device_Address_Binding	Constant	Empty list	R - read
Database_Revision	Constant	1	R - read
Max_Master	Depends on setting in BASC_SERVER	127	R - read
Max_Info_Frames	Constant	1	R - read

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

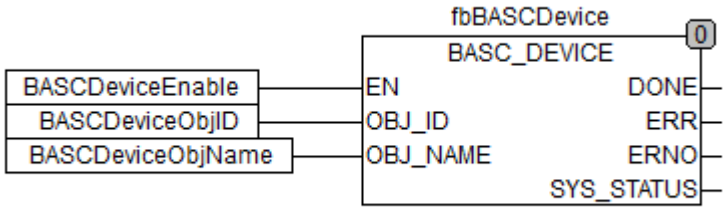
Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

Defines the BACnet object name. The object name has to be unique within BACnet network.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

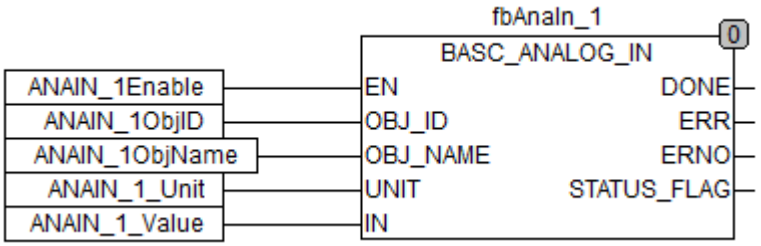
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

SYS_STATUS (system status)

Data type	Default value	Range	Unit
BASC_DEV_STATUS_ENUM	BASC_NON_OPERATIONAL	BASC_NON_OPERATIONAL, BASC_OPERATIONAL	-

Indicates the current state of the BACnet device ↳ *“BASC_DEV_STATUS_ENUM” on page 2518*.

1.5.7.2.3 BASC_ANALOG_IN



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

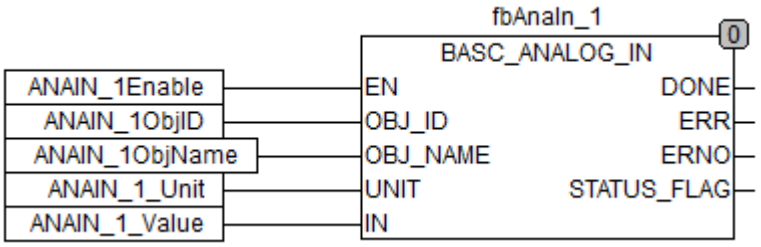
BASC_ANALOG_IN allows to connect an IEC variable with a BACnet object type "analog input". This object represents a physical analog input of the device. Connect the variables mapped to the physical analog inputs to this function block only.

Table 142: Analog input object properties provided at the BACnet network by BASC_ANALOG_IN

Property name	Characteristics	Value	BACnet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Units	Set via input UNIT	-	R - read
Present_Value	Set via input IN	-	R - read
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	ANALOG_INPUT (0)	R - read
Event_State	Constant	NORMAL (0)	R - read
Out_Of_Service	Set via BACnet write services	-	W - read/write

If property `Out_Of_service` = TRUE the property `Present_Value` becomes writable.

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE input IN is evaluated and used to update BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

Defines the BACnet object name. The object name has to be unique within BACnet network.

UNIT

Data type	Default value	Range	Unit
BASC_ENG_UNITS_ENUM	BASC_NO_UNITS	All units defined in BACnet standard.	-

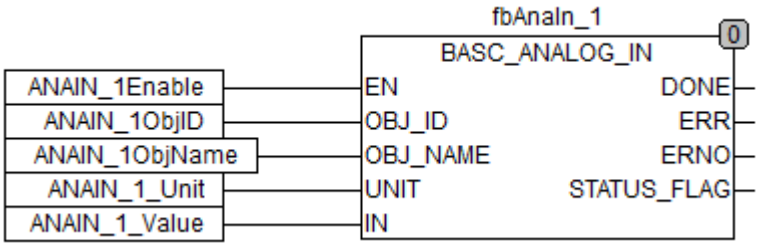
Indicates the engineering unit of the `Present_Value` property of the object. In BACnet network the value connected to the function block input IN will be interpreted as value of this engineering unit.

IN

Data type	Default value	Range	Unit
REAL	0.0	-3.40282347E38...3.40282347E38	-

To be connected to an IEC variable representing a physical input of the processor module. The value will be used to update BACnet property `Present_Value`.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

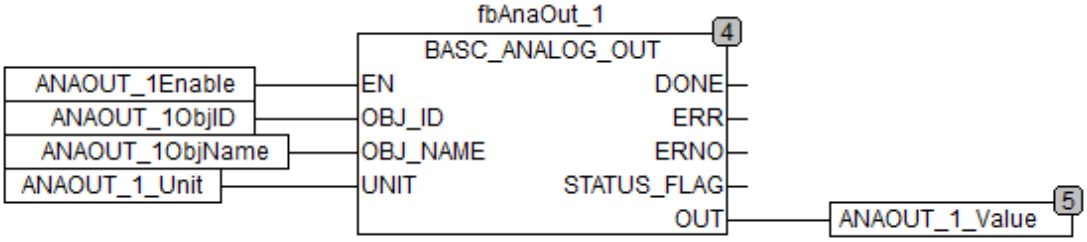
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

1.5.7.2.4 BASC_ANALOG_OUT



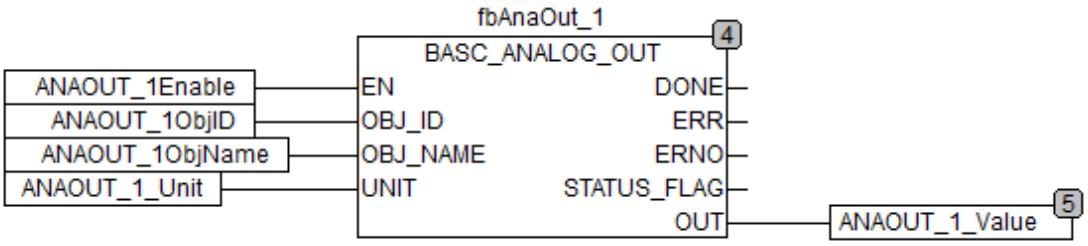
Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

BASC_ANALOG_OUT allows to connect an IEC variable with a BACnet object type "analog output". This object represents a physical analog output of the device. Connect the variables mapped to the physical analog outputs to this function block only.

Table 143: Analog output object properties provided at the BACnet network by BASC_ANALOG_OUT

Property	Characteristics	Value	BACNet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Units	Set via input UNIT	-	R - read
Present_Value	Updates output OUT	-	W - read/write
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	ANALOG_OUTPUT (1)	R - read
Event_State	Constant	NORMAL (0)	R - read
Out_Of_Service	Set via BACnet write services	-	W - read/write
Priority_Array	Set via BACnet write services	-	R - read
Relinquish_Default	Constant	0.0	R - read

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE output OUT is updated with current value of BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

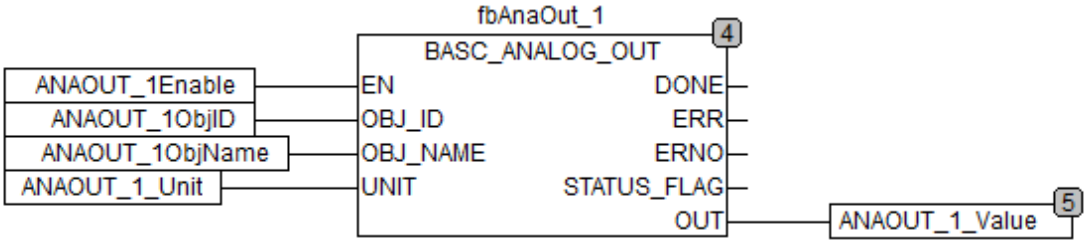
Defines the BACnet object name. The object name has to be unique within BACnet network.

UNIT

Data type	Default value	Range	Unit
BASC_ENG_UNITS_ ENUM	BASC_NO_UNITS	All units defined in BACnet standard.	-

Indicates the engineering unit of the `Present_Value` property of the object. In BACnet network the value connected to the function block input IN will be interpreted as value of this engineering unit.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

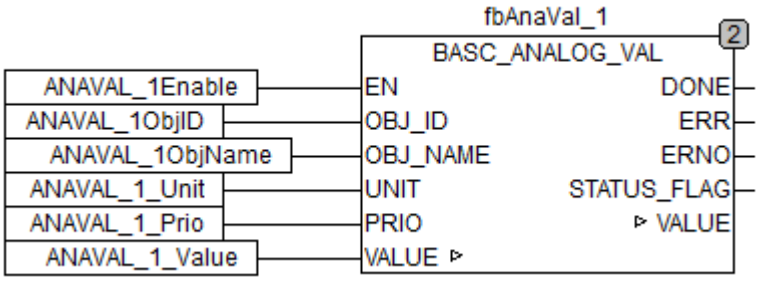
BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

OUT

Data type	Default value	Range	Unit
REAL	0.0	-3.40282347E38...3.40282347E38	-

To be connected to an IEC variable representing a physical output of the processor module. This output will be updated with the BACnet object property `Present_Value`.

1.5.7.2.5 BASC_ANALOG_VAL



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

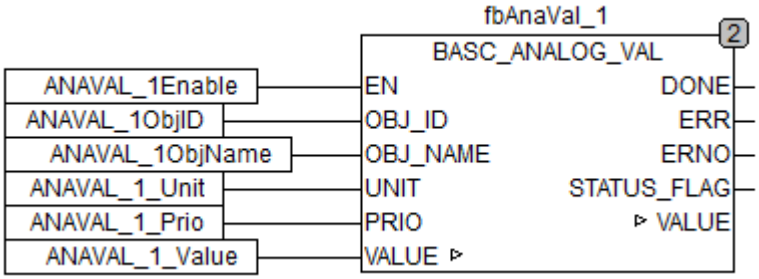
BASC_ANALOG_VAL allows to connect an IEC variable with a BACnet object type "analog value". This object represents an analog value that is neither physical input nor physical output of the device. Connect the internal IEC variables to this function block only.

Table 144: Analog value object properties provided at the BACnet network by BASC_ANALOG_VAL

Property	Characteristics	Value	BACNet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Units	Set via input UNIT	-	R - read

Property	Characteristics	Value	BACNet conformance code
Present_Value	Set via input VALUE, Updates output VALUE	-	W – read/write
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	ANALOG_VALUE (2)	R - read
Event_State	Constant	NORMAL (0)	R - read
Out_Of_Service	Set via BACnet write services	-	W - read/write
Priority_Array	Set via BACnet write services	-	R - read
Relinquish_Default	Constant	0.0	R - read

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE input VALUE is evaluated and used to update BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

Defines the BACnet object name. The object name has to be unique within BACnet network.

UNIT

Data type	Default value	Range	Unit
BASC_ENG_UNITS_ENUM	BASC_NO_UNITS	All units defined in BACnet standard.	-

Indicates the engineering unit of the `Present_Value` property of the object. In BACnet network the value connected to the function block input IN will be interpreted as value of this engineering unit.

PRIO

Data type	Default value	Range	Unit
BYTE	1	1...16	-

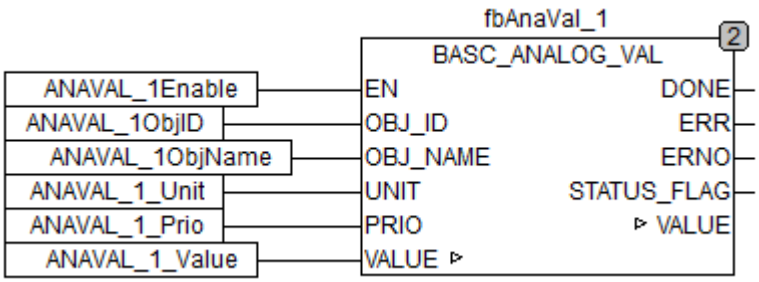
Defines the BACnet priority that will be used on updating the property `Present_Value` of the analog value object.

VALUE

Data type	Default value	Range	Unit
REAL	0.0	-3.40282347E38….3.40282347E38	-

To be connected to an IEC variable representing this object type. The value will be used to update BACnet property `Present_Value`.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

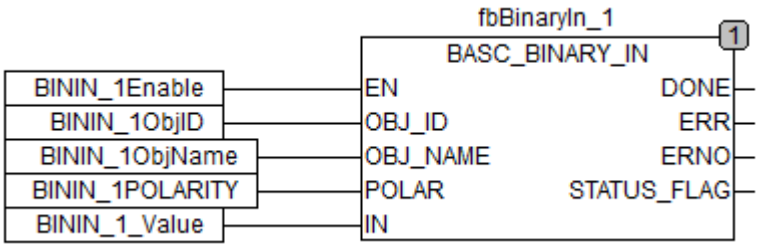
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

1.5.7.2.6 BASC_BINARY_IN



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

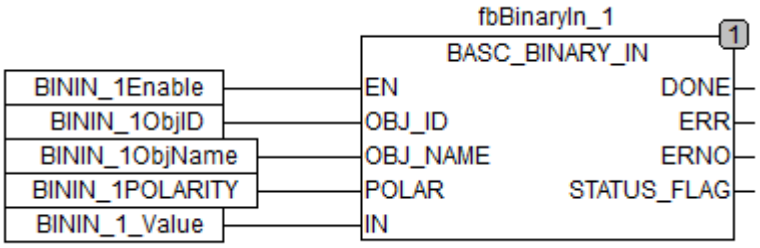
BASC_BINARY_IN allows to connect an IEC variable with a BACnet object type "binary input". This object represents a physical binary input of the device. Connect the variables mapped to physical binary inputs to this function block only.

Table 145: Binary input properties provided at the BACnet network by BASC_BINARY_IN

Property	Characteristics	Value	BACNet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Polarity	Set via input POLAR	-	R - read
Present_Value	Set via input IN	-	R - read
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	BINARY_INPUT (3)	R - read
Event_State	Constant	NORMAL (0)	R - read
Out_Of_Service	Set via BACnet write services	-	W - read/write

> *If property* `Out_Of_service` *= TRUE the property* `Present_Value` *becomes writable.*

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE input IN is evaluated and used to update BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

Defines the BACnet object name. The object name has to be unique within BACnet network.

POLAR

Data type	Default value	Range	Unit
BASC_POLAR_ENUM	BASC_NORMAL	BASC_NORMAL, BASC_REVERSE	-

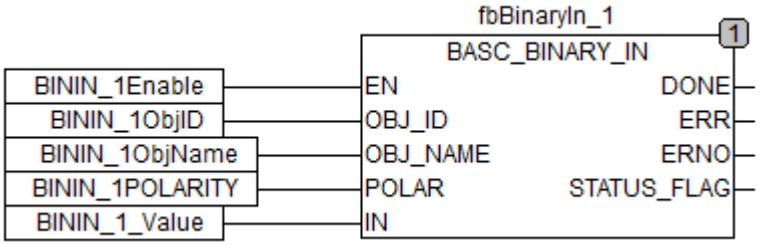
Defines the polarity of the value that is connected to the input IN. Polarity is defined according to BACnet standard ↳ *"BASC_POLAR_ENUM" on page 2518*.

IN

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

To be connected to an IEC variable representing a physical input of the processor module. The value will be used to update BACnet property `Present_Value`.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

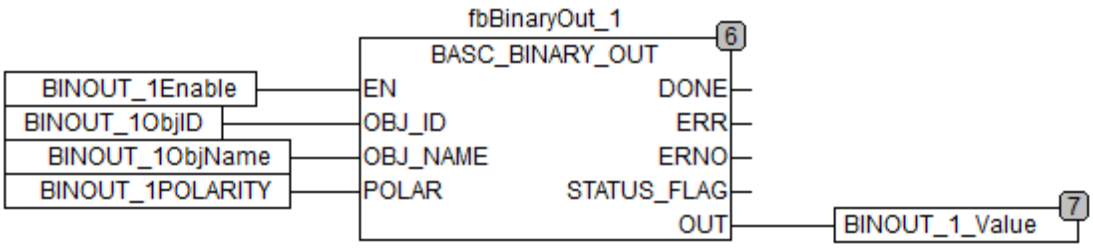
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

1.5.7.2.7 BASC_BINARY_OUT



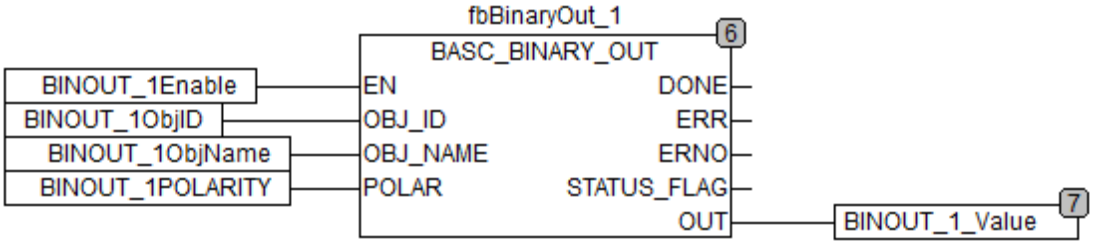
Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

BASC_BINARY_OUT allows to connect an IEC variable with a BACnet object type "binary output". This object represents a physical binary output of the device. Connect the variables mapped to physical binary outputs to this function block only.

Table 146: Binary output properties provided at the BACnet network by BASC_BINARY_OUT

Property name	Characteristics	Value	BACNet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Polarity	Set via input POLAR	-	R - read
Present_Value	Updates output OUT	-	W - read/write
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	BINARY_OUTPUT (4)	R - read
Event_State	Constant	NORMAL (0)	R - read
Out_Of_Service	Set via BACnet write services	-	W - read/write
Priority_Array	Set via BACnet write services	-	R - read
Relinquish_Default	Constant	INACTIVE	R - read

Input description



EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE output OUT is updated with current value of BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

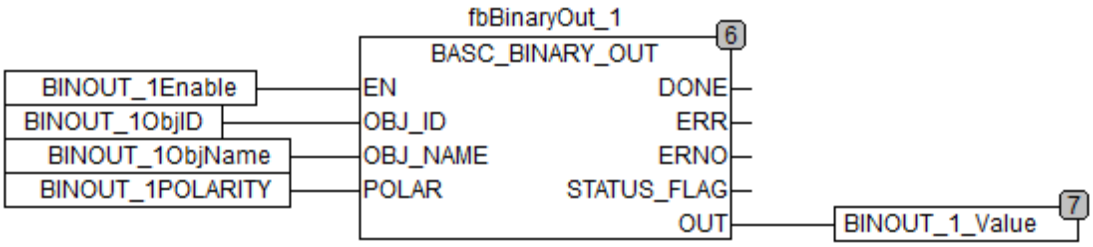
Defines the BACnet object name. The object name has to be unique within BACnet network.

POLAR

Data type	Default value	Range	Unit
BASC_POLAR_ENUM	BASC_NORMAL	BASC_NORMAL, BASC_REVERSE	-

Defines the polarity of the value that is connected to the input IN. Polarity is defined according to BACnet standard *"BASC_POLAR_ENUM" on page 2518*.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 "Error messages of the AC500 V2 function block libraries" on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

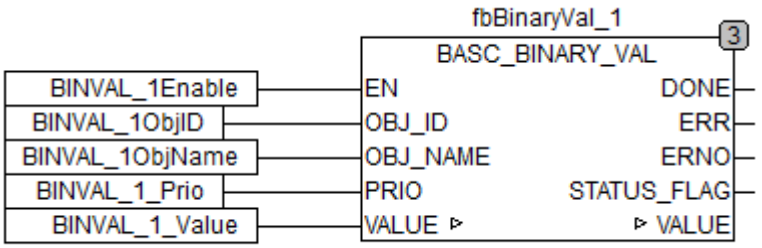
BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

OUT

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

To be connected to an IEC variable representing a physical output of the processor module. This output will be updated with the BACnet object property `Present_Value`.

1.5.7.2.8 BASC_BINARY_VAL



Parameter	Value
Included in library	BACnet_B-ASC_AC500_V25.lib
Available as of firmware	V2.5
Type	Function block with historical values
Group	C interface

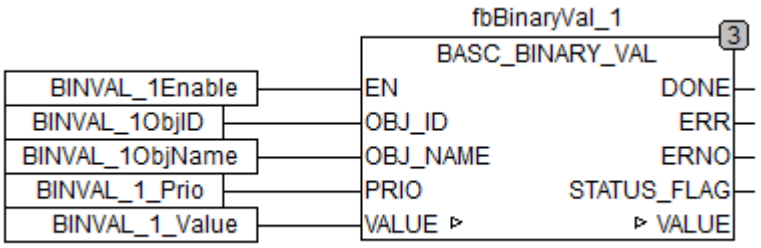
BASC_BINARY_VAL allows to connect an IEC variable with a BACnet object type "binary value". This object represents a binary value that is neither physical input nor physical output of the device. Connect the internal IEC variables to this function block only.

Table 147: Binary value object properties provided at the BACnet network by BASC_BINARY_VAL

Property name	Characteristics	Value	BACNet conformance code
Object_Identifier	Set via input OBJ_ID	-	R - read
Object_Name	Set via input OBJ_NAME	-	R - read
Present_Value	Set via input VALUE Updates output VALUE	-	W – read/write
Status_Flags	Updates output STATUS_FLAG	-	R - read
Object_Type	Constant	BINARY_VALUE (5)	R - read
Event_State	Constant	NORMAL (0)	R - read

Property name	Characteristics	Value	BACNet conformance code
Out_Of_Service	Set via BACnet write services	-	W - read/write
Priority_Array	Set via BACnet write services	-	R - read
Relinquish_Default	Constant	INACTIVE	R - read

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

In order to enable the function block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input EN is set to TRUE input VALUE is evaluated and used to update BACnet object property `Present_Value`.

OBJ_ID (object ident number)

Data type	Default value	Range	Unit
WORD	0	0...4194302	-

Defines the BACnet object ident number. The object ident number has to be unique within the BACnet network.

OBJ_NAME (object name)

Data type	Default value	Range	Unit
STRING(40)	Empty string	-	-

Defines the BACnet object name. The object name has to be unique within BACnet network.

PRIO

Data type	Default value	Range	Unit
BYTE	16	1...16	-

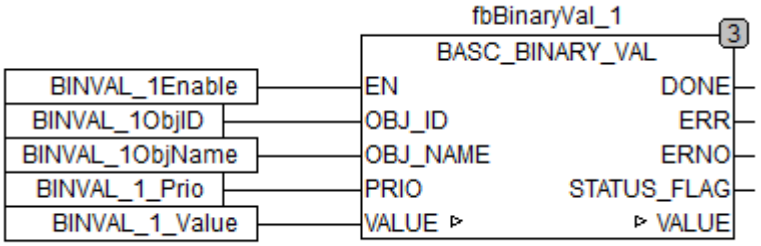
Defines the BACnet priority that will be used on updating the property `Present_Value` of the analog value object.

VALUE

Data type	Default value	Range	Unit
BOOL	FALSE	-	-

To be connected to an IEC variable representing this object type. The value will be stored in BACnet property `Present_Value`.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

STATUS_FLAG

Data type	Default value	Range	Unit
WORD	-	-	-

BACnet B-ASC library supports property `Status_Flags`. The current value of this property will be provided at output STATUS_FLAG.

1.5.7.3 Structures and enumerations

BASC_COM_TYPE_ENUM Enumeration for BACnet communication layers ↳ *“COM_TYPE (communication type)” on page 2496*.

Parameter	Value	Description
BASC_IP_COM	0	Use BACnet IP communication.
BASC_MSTP_COM	1	Use BACnet MS/TP communication.

BASC_DEV_STATUS_ENUM Enumeration for BACnet device status values *“SYS_STATUS (system status)” on page 2501*.

Parameter	Value	Description
BASC_OPERATIONAL	0	The device is initialized successfully and in operation.
BASC_NON_OPERATIONAL	4	The device is not in operation.

BASC_ENG_UNITS_ENUM Enumeration for BACnet units *“UNIT” on page 2509*. This enumeration lists all units defined in BACnet standard *Further information on page 2495*.

BASC_POLAR_ENUM Enumeration of BACnet values for polarity of `Present_Value` property *“POLAR” on page 2514*.

Parameter	Value	Description
BASC_NORMAL	0	Physical state relation. ACTIVE = physical on.
BASC_REVERSE	1	Physical state relation. ACTIVE = physical off.

BASC_STATISTIC_COUNTER_TYPE Structure of statistic data *“STATISTIC” on page 2498*.

Parameter	Type	Value	Description
dwNumReceivedRequests	DWORD	0	Number of received request.
dwNumRepliesSuccess	DWORD	0	Number of replies with success.
dwNumRepliesFailure	DWORD	0	Number of replies with failure.

1.5.7.4 Hardware

An AC500 processor module equipped with an integrated Ethernet or serial interface is required. Only one communication protocol at a time is supported.

1.5.7.5 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.8 FM502-CMS library

1.5.8.1 System technology

The WAV File and CMS-IO libraries are libraries to enable the work with the *Chapter 1.6.4.4.3 “FM502-CMS function module” on page 5723*. A measurement can be started using the CMS-IO library. The results are stored by the Function Module FM502-CMS on the processor module flash disk as WAV files. Data handling of the WAV files is possible with the WAV-File Library.

For immediate local analysis an application library for signal processing is available. SP_Library on *www.abb.com/PLC*. The application program then may use signal processing blocks to read and analyze the data, to calculate status and trends of the monitored equipment already locally on the PLC. Typically the measurement need to be labeled with the operation point details from a control program section and the signal processing needs further information, values or parameters also used in the control section (e.g. rated values of the equipment, speed if not measured with the FM502 counter functions).

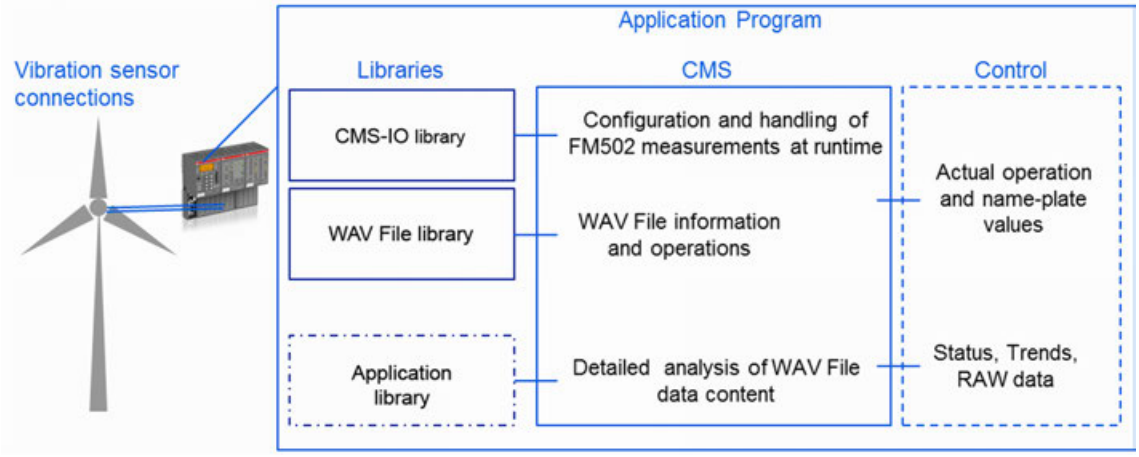


Fig. 179: CMS application and programming

1.5.8.1.1 CMS-IO library

The CMS-IO library contains all function blocks necessary for using the Function Module FM502-CMS.

In addition, it is possible to change the configuration of the analog channels during run time via function blocks.

Once a FM502-CMS has been added to the configuration, the CMS-IO library is automatically included with the next compilation of the project.

1.5.8.1.2 WAV File library

The WAV File library is used for reading, writing and handling the the stored WAV files.

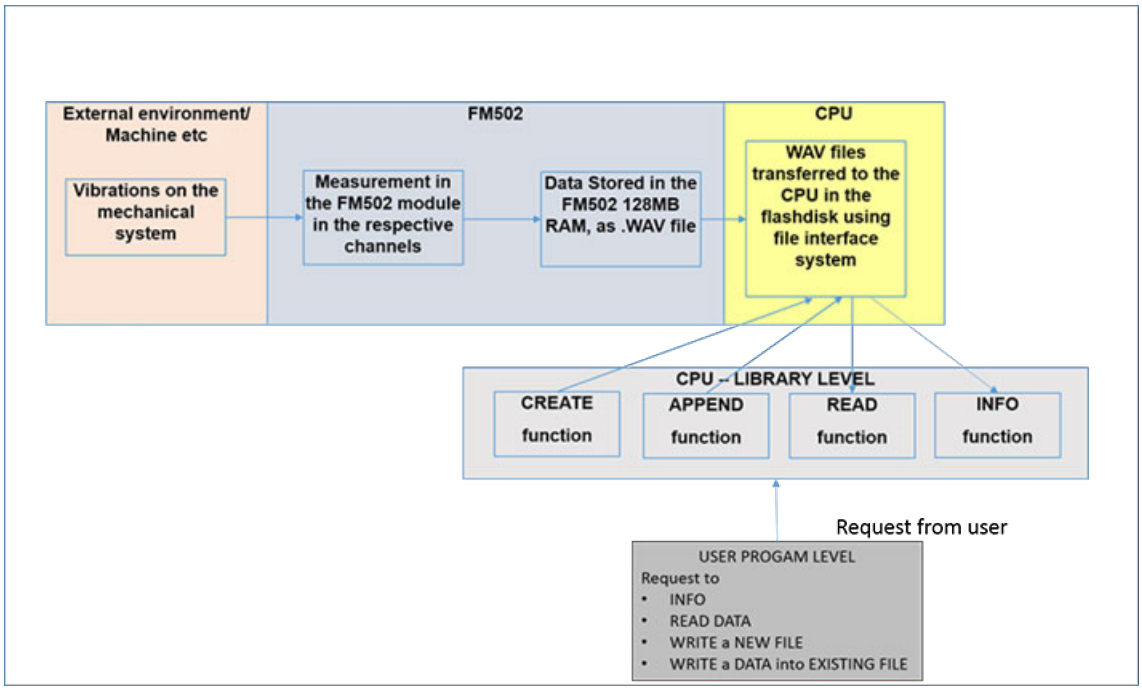


Fig. 180: Process map

To unzip the generated WAV files, the archive function blocks of the ↳ *Chapter 1.5.4.4 “CAA_File library” on page 789* are needed in addition. Example: Function block ↳ *Chapter 1.5.4.4.2.6 “FILE_ArchiveUnpack” on page 801*. The CAA_File libraries (CAA_...lib) are automatically included when configuring the FM502-CMS.

Architecture

Internally the WAV File library uses the four chunk structure model of the WAV files to handle the data in the read and write operations.

To perform the correct data handling the WAV File library uses a structure internally. This structure is a group of data stored in the following order:

- RIFF Chunk ID,
- File Size,
- RIFF Type,
- Format Chunk ID,
- Channel Information,
- Sampling Frequency,
- Block Align,
- Bits per Sample,
- Data Chunk ID,
- Data,
- Label Information.

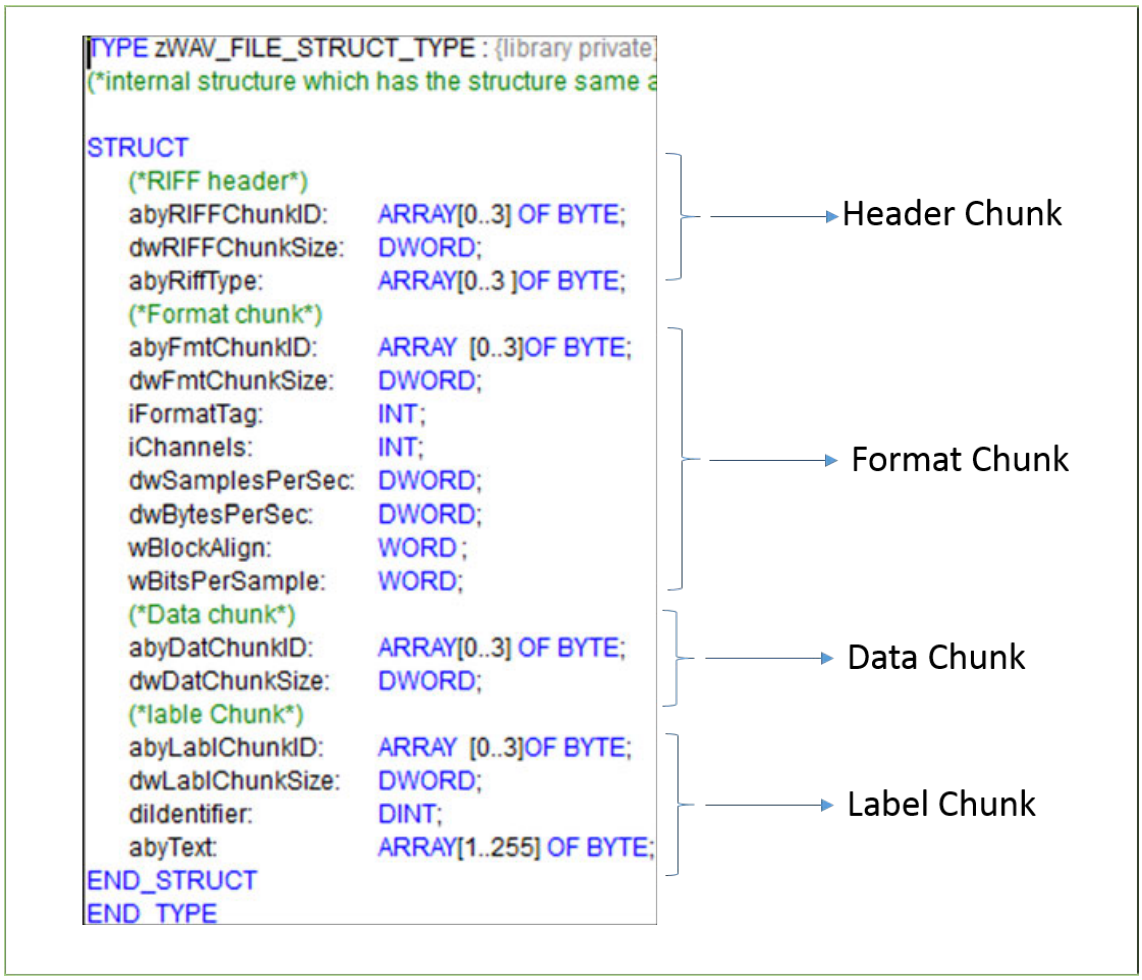


Fig. 181: Internal data type of the library

Read and write WAV files

A single READ or WRITE functionality in this library is executed in the following steps:

1. Open the file to create the handle.
   ⇨ Validates the file, if already present (except in function block CREATE).
2. Set the pointer to the file position from where the desired READ or WRITE operation is to be executed.
3. Assign the inputs of the function block to the internal structure.
   ⇨ Stores the data in the proper structure before writing (in function blocks CREATE and APPEND).
4. Perform the READ/WRITE operation.
   ⇨ Stores the data in the proper structure after reading (in function blocks INFO and READ)
5. Assign the outputs to the function block.
6. Close the file.

With the development of the WAV-File library the user can get the following benefits:

- Specific function blocks to perform the specific task.
- Either a function block or the visualization can be used.

- The function blocks internally take care of four important tasks
  - Opening the file and validating it.
  - Preparing the proper data structure at the time of reading or writing, according to the WAV file standards.
  - Performing the READ or WRITE operation.
  - Closing the file.
- Simple to understand and execute.

1.5.8.1.3 WAV file format

The wave audio file format or commonly known as the WAV files are the type of audio files which was developed by Microsoft and IBM. It stores the data in the structured format in the dedicated memory area called as chunks.

The measurement data will be stored in the WAV file format. One WAV file will be created for each active channel.

Table 148: RIFF header

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	0 (0x00)	bfChunkID	"RIFX"
DWORD	Little	4	4 (0x04)	dwChunkSize	Data length - 8
BYTE[4]	Big	4	8 (0x08)	bfRiffType	"WAVE"

Table 149: Format chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	12 (0x0C)	bfChunkID	"fmt"
DWORD	Little	4	16 (0x10)	dwChunkSize	Data length - 8
INT	Little	2	18 (0x12)	wFormatTag	0x0001 (PCM)
INT	Little	2	20 (0x14)	wChannels	0x0001 (1 ch.)
DWORD	Little	4	24 (0x18)	dwSamples-PerSec	100 Hz - 50.000 kHz
DWORD	Little	4	28 (0x1C)	dwBytes-PerSec	Sample rate * block align
WORD	Little	2	32 (0x1E)	wBlockAlign	4 byte
WORD	Little	2	34 (0x20)	wBitsPer-Sample	32 bit

Table 150: Data chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	36 (0x24)	bfChunkID	"data"
DWORD	Little	4	40 (0x28)	dwChunkSize	Data length - 8
BYTE[]	Big	Undefined	44 (0x2C)	bfData	Measurement data

Table 151: Label chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	44+sz(bfData)	bfChunkID	“labl”
DWORD	Little	4	48+sz(bfData)	dwChunkSize	Data length -8
DINT	Little	4	52+sz(bfData)	dwIdentifier	Identifier
BYTE[256]	Little	255	56+sz(bfData)	bfText	„Label Text“

Use of the WAV file

The unique and structured architecture is of a great help in the condition monitoring systems (CMS). The CMS monitors the stability of the mechanical system. The vibration data of the machine/system can be recorded using the sensors. These sensors will convert the physical data into the analog inputs to the CPU and its modules. These data is stored in the WAV file structure and can be re-read for the analysis purpose using the mathematical tools.

Main purpose of the WAV file

The main purpose of the WAV files is to do the analysis in the CMS. For this purpose the WAV File Library is developed. This would help the user to perform the READ/WRITE operations on the WAV file using the AC500 PLC and its programming tool.

1.5.8.1.4 Limitations

Memory

Although the file size can be 128 MB, however the amount of data which the user wants to read or write is limited due to the user data capacity of the processor module.

The maximum user data capacity of the processor module is up to 4 MB which also includes the application program. In any case the WAV file will be able to handle less than 4 MB data.

WAV files (Single/Multi Channel)

The WAV File library can only READ/APPEND a single channel WAV file. In case the user tries to READ/APPEND the multi-channel WAV file, the function block will throw an error message.

Measured data

All measured data is read and stored in the buffer as bytes and not as real values. In the same way while writing, the library accepts the data in the buffer as bytes.

Data loss when using function block WAV_FILE_CREATE

No error occurs, when the user creates a new file with the same name of an already existing file. This is due to the limitation of the CAA_File library WRITE functionality. The user needs to take care, otherwise the existing file might get overwritten with the new data.

Usage of function blocks

The CAA_File library restricts the usage of more than 20 CAA functions at a time. Since the WAV File library function blocks consists of CAA functions internally, the user might get an error (error code 5803) from the library. Also simultaneous enabling of the WAV File function blocks on the same file may generate the error from the CAA_File library.

1.5.8.2 CMS-IO library for modul handling

Preconditions

The function blocks of the CMS-IO library are only working in the RUN mode of the processor module. Using the library in the SIMULATION mode will not provide any valid or usable diagnosis information.

The encoder/counter interface inside the FM502-CMS can be used in different configurable operation modes. The operation mode is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when the Function Module is added into the configuration.

Operation modes

Inputs and outputs which are not used by the counters are available for other tasks.

Table legend: A = input channel A, B = input channel B, Z = output channel Z.

Operation Mode	Function	Used inputs	Description	Function block
0-1	No counter	None	This operating mode is selected, if the integrated high-speed counter is not needed.	-
1-1	Up/down counter (A)	A = Counting input	1 bidirectional 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT
2-1	Up/down with release input (B)	A = Counting input B = Enable input	1 bidirectional 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT
3-2	Up/down counters (A,B)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	CMS_IO_16BIT_2CNT
4-2	Up/down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and overflow flag.	CMS_IO_16BIT_2CNT
5-1	Up/down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge sets START_VALUE.	CMS_IO_32BIT_CNT

Opera-tion Mode	Function	Used inputs	Description	Function block
6-1	Up/down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge sets START_VALUE.	CMS_IO_32BIT_CNT
7-1	Reserved	None	-	-
8-1	Up/down with release (B), 0 cross detection	A = Counting input B = Enable input	1 bidirectional 16-bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	CMS_IO_16BIT_CNT
9-1	Reserved	None	-	-
10-1	Reserved	None	-	-
11-1	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for encoder x1 count, touch/ catch value, RPI function, reset and set Function block counts rising edges at input A.	CMS_IO_32BIT_ENCODER
12-1	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x2 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A.	CMS_IO_32BIT_ENCODER
13-1	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x4 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A and B.	CMS_IO_32BIT_ENCODER
14-1	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface	CMS_IO_SSI_CNT
15-1	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation	CMS_IO_FREQ_SCAN

Identifying a FM502-CMS at the function block

The function blocks have to identify every FM502-CMS connected to the processor module. Therefore the Automation Builder automatically creates an instance description of the module. This instance description is available as a global variable in CODESYS. Use the input INST to connect the address of the instance *“INST (instance)” on page 2527*.

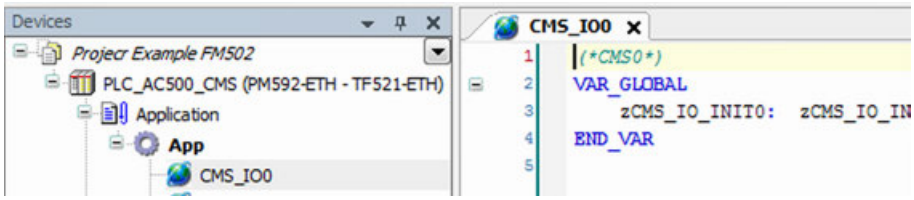


Fig. 182: Global variable in Automation Builder

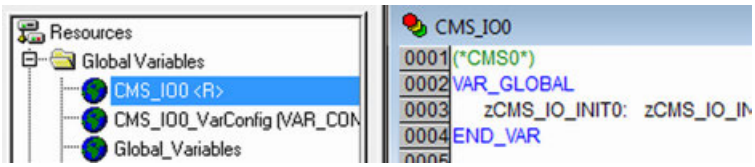
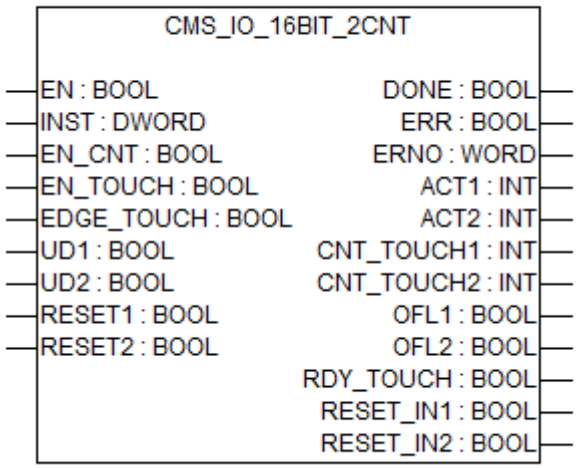


Fig. 183: Global variable in CODESYS

1.5.8.2.1 Function blocks

CMS_IO_16BIT_2CNT

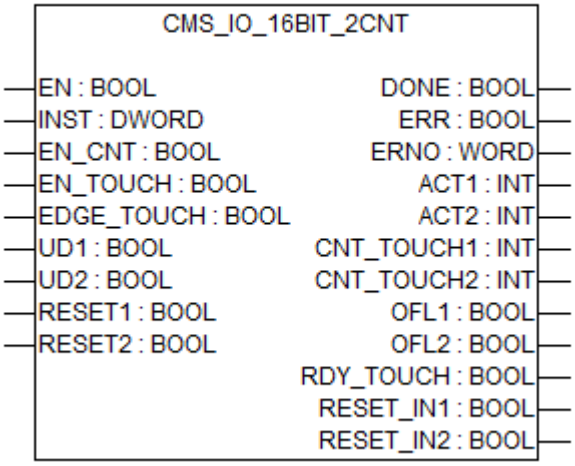


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counters

The module FM502-CMS provides *Chapter 1.6.4.4.3.4.3 “Two 16 bit up/down counters” on page 5738* functions.

Possible operation modes: 3-2, 4-2 *“Operation modes” on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

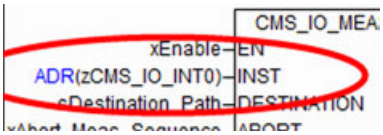


Fig. 184: Example

EN_CNT (enable counter)

Data type: BOOL

If EN_CNT=TRUE , pulse counting of counter is enabled. If EN_CNT=FALSE, no pulse counting is performed and the pulses are lost. If counting has already started and EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_TOUCH (enable touch)

Data type: BOOL

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counters A and B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

EDGE_TOUCH

Data type: BOOL

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of the configured digital input (for counters A and B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of the configured digital input (for counters A and B).

UD1 (up/down mode for counter 1)

Data type: BOOL

At input UD1, the counting selection is set for up/down counting mode for counter A:

UD1=FALSE: count up

UD1=TRUE: count down

UD2 (up/down mode for counter 2)

Data type: BOOL

At input UD2, the counting selection is set for up/down counting mode for counter B:

UD2=FALSE: count up

UD2=TRUE: count down

RESET1 (reset counter 1)

Data type: BOOL

If input RESET1 = TRUE, the counter value (ACT1) is reset to 0. As long as input RESET1 = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input DI0, DI1, DC2, DC3 (for counter A) causes the function block to reset the value at output ACT1.

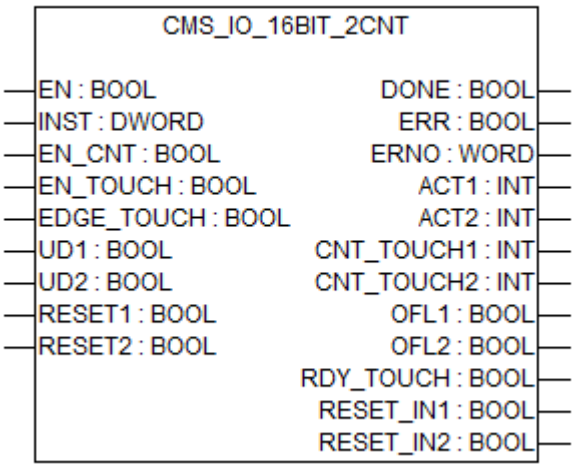
RESET2 (reset counter 2)

Data type: BOOL

If input RESET2 = TRUE, the counter value (ACT2) is set to 0.As long as input RESET2 = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at input DI0, DI1, DC2, DC3 (for counter B) causes the function block to reset the value at output ACT2.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT1 (actual value 1)

Data type: INT

The current counter value (actual value) from counter A can be retrieved at any time using the output ACT1 of the function block.

ACT2 (actual value 2)

Data type: INT

The current counter value (actual value) from counter B can be retrieved at any time using the output ACT2 of the function block.

CNT_TOUCH1

Data type: INT

The output CNT_TOUCH1 (counter touch value 1) displays the result of the catch/touch trigger measurement for counter A.

CNT_TOUCH2 (counter touch value 2)

Data type: INT

The output CNT_TOUCH2 displays the result of the catch/touch trigger measurement for counter B.

OFL1 (overflow counter 1)

Data type: BOOL

The overflow from counter A is specified at the output OFL1.

The counter operates as infinite counter. It is set to TRUE, when an overflow occurs, i. e. the counter value ACT1 rises to value 16#FFFF= -1. Any exceeding or falling below this value (depending on use up and use down) will set OFL1=TRUE. The output OFL1 is reset when the configuration is changed, and if counter value ACT1 is set or reset.

OFL2 (overflow counter 2)

Data type: BOOL

The overflow from counter B is specified at the output OFL2.

The counter operates as infinite counter. It is set to TRUE when an overflow occurs, i. e. the counter value ACT2 rises to value 16#FFFF= -1. Any exceeding or falling below this value (depending to use up and use down) will set OFL2=TRUE. The output OFL2 is reset when the configuration is changed, and if counter value ACT2 is set or reset.

RDY_TOUCH (ready catch/ touch value)

Data type: BOOL

The output RDY_TOUCH is set to TRUE, when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

RESET_IN1 (reset input counter 1)

Data type: BOOL

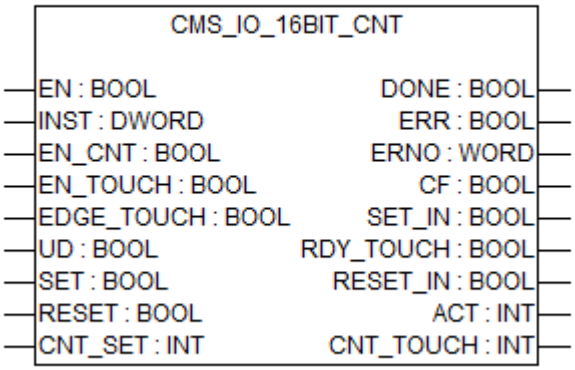
The output RESET_IN1 is set to TRUE, if one of the digital inputs DI0, DI1, DC2, DC3 is configured as RESET input for counter A and has a TRUE signal or the input RESET of CMS_IO_16BIT_2CNT is set to TRUE.

RESET_IN2 (reset input counter 2)

Data type: BOOL

The output RESET_IN2 is set to TRUE, if one of the digital inputs DI0, DI1, DC2, DC3 is configured as RESET input for counter B and has a TRUE signal or the input RESET of CMS_IO_16BIT_2CNT is set to TRUE.

CMS_IO_16BIT_CNT

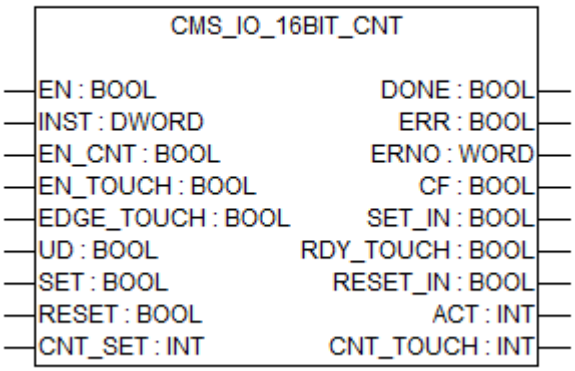


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counter & Encoder

The FM502-CMS provides one *Chapter 1.6.4.4.3.4.2 “16-bit bidirectional counter” on page 5737* function.

Possible operation modes: 8-1 *“Operation modes” on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

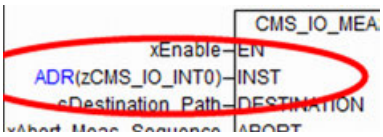


Fig. 185: Example

EN_CNT (enable counter)

Data type: BOOL

If EN_CNT=TRUE , pulse counting of counter is enabled. If EN_CNT=FALSE, no pulse counting is performed and the pulses are lost. If counting has already started and EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_TOUCH (enable touch)

Data type: BOOL

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counters A and B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

EDGE_TOUCH

Data type: BOOL

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of the configured digital input (for counters A and B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of the configured digital input (for counters A and B).

UD (up/down mode)

Data type: BOOL

At input UD, the counting selection is set for up/down counting mode:

UD=FALSE: count up

UD=TRUE: count down

SET (set counter)

Data type: BOOL

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET (reset counter)

Data type: BOOL

If input RESET = TRUE, the counter value (ACT) is reset to 0. As long as input RESET = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at digital input DI0, DI1, DC2, DC3 causes the function block to reset the value at output ACT.

CNT_SET

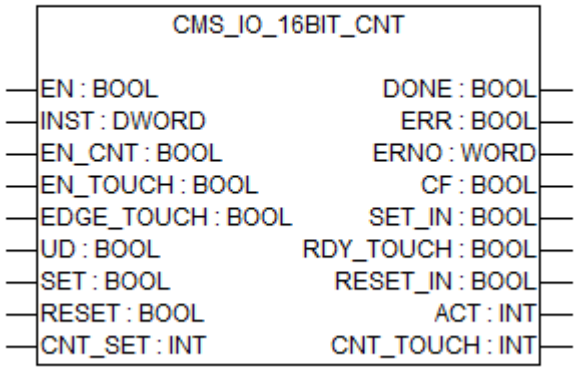
Data type	Default value	Range	Unit
INT	-	-	-

The counter can be set to a start value. This value must be applied to the input CNT_SET.

If input SET=TRUE, counter takes this value.

Input START_VALUE corresponds to output low word in "counter settings".

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

CF (carry flag)

Data type: BOOL

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE, when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

SET_IN (set input counter)

Data type: BOOL

The output SET_IN is set to TRUE, if one of the inputs is configured as SET input.

RDY_TOUCH (ready catch/ touch value)

Data type: BOOL

The output RDY_TOUCH is set to TRUE, when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

RESET (reset counter)

Data type: BOOL

If input RESET = TRUE, the counter value (ACT) is reset to 0. As long as input RESET = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at digital input DI0, DI1, DC2, DC3 causes the function block to reset the value at output ACT.

ACT (actual value)

Data type: INT

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

CNT_TOUCH (counter touch value)

Data type: INT

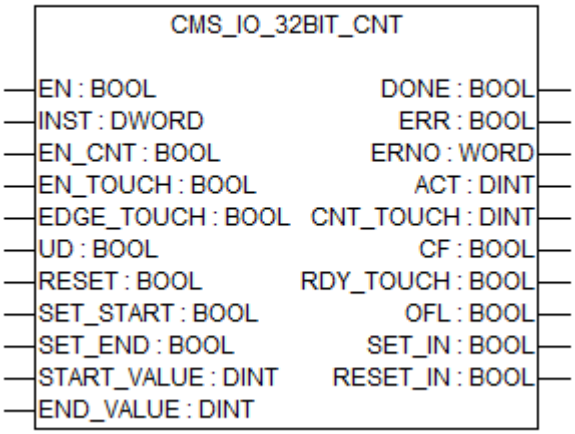
The output CNT_TOUCH displays the result of the catch/touch trigger action.

CNT_TOUCH (counter touch value)

Data type: DINT

The output CNT_TOUCH displays the result of the catch/touch trigger action.

CMS_IO_32BIT_CNT 32-bit counter

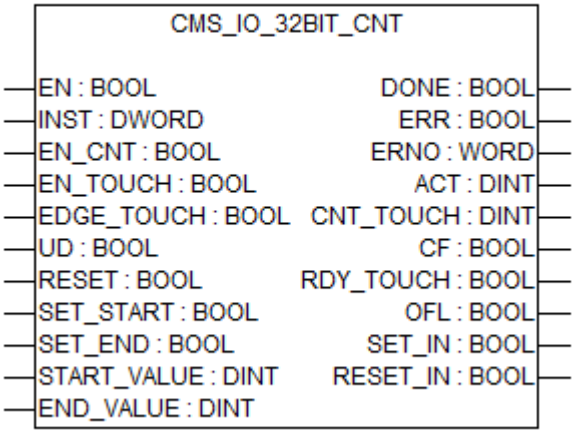


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counter & Encoder

The FM502-CMS provides one *Chapter 1.6.4.4.3.4.1 “32-bit bidirectional counter” on page 5736* functions.

Possible operation modes: 1-1, 2-1, 5-1, 6-1 *“Operation modes” on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

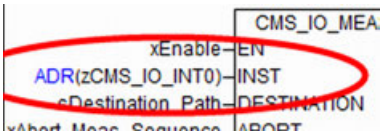


Fig. 186: Example

EN_CNT (enable counter)

Data type: BOOL

If EN_CNT=TRUE , pulse counting of counter is enabled. If EN_CNT=FALSE, no pulse counting is performed and the pulses are lost. If counting has already started and EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_TOUCH (enable touch)

Data type: BOOL

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counters A and B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

EDGE_TOUCH

Data type: BOOL

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of the configured digital input (for counters A and B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of the configured digital input (for counters A and B).

UD (up/down mode)

Data type: BOOL

At input UD, the counting selection is set for up/down counting mode:

UD=FALSE: count up

UD=TRUE: count down

RESET (reset counter)

Data type: BOOL

If input RESET = TRUE, the counter value (ACT) is reset to 0. As long as input RESET = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at digital input DI0, DI1, DC2, DC3 causes the function block to reset the value at output ACT.

SET_START (set start value)

Data type: BOOL

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input causes the function block to store the START_VALUE value and to display this value at output ACT.

SET_END (set end value)

Data type: BOOL

If input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

START_VALUE

Data type: DINT

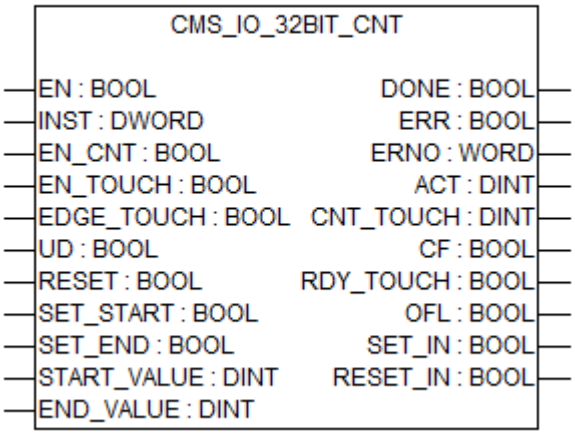
The counter can be set to a start value. This value must be applied to the input START_VALUE.

END_VALUE

Data type: DINT

If the counter reaches the planned input END_VALUE, the binary output CF is set to TRUE and the value is stored.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT (actual value)

Data type: DINT

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

CNT_TOUCH (counter touch value)

Data type: DINT

The output CNT_TOUCH displays the result of the catch/touch trigger action.

CF (carry flag)

Data type: BOOL

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE, when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

RDY_TOUCH (ready catch/ touch value)

Data type: BOOL

The output RDY_TOUCH is set to TRUE, when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

OFL (overflow)

Data type: BOOL

The overflow is specified at the output OFL.

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648. Any exceeding or falling below of this value (depending to up and down use) will set OFL to TRUE.

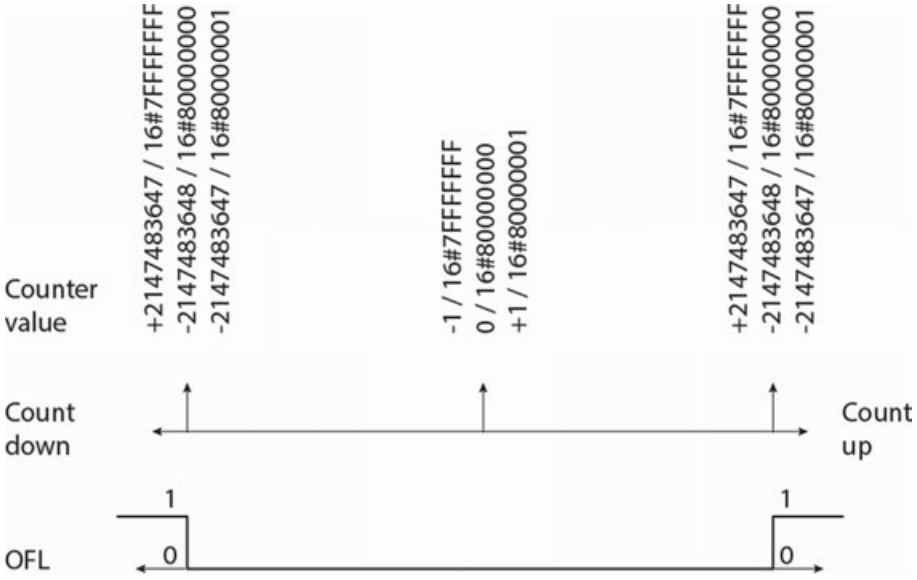


Fig. 187: Detection for output OFL

SET_IN (set input counter)

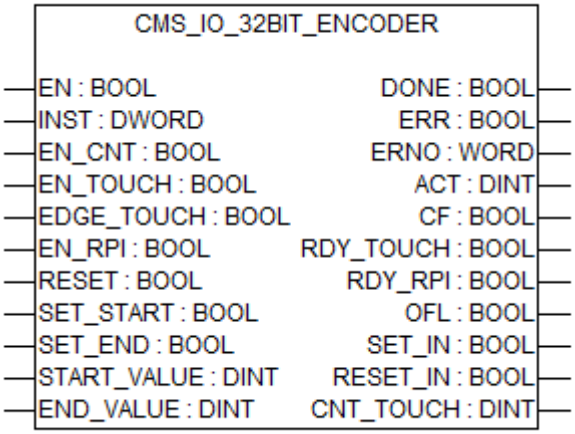
Data type: BOOL

The output SET_IN is set to TRUE, if one of the inputs is configured as SET input.

RESET_IN (reset input counter) Data type: BOOL

The output RESET_IN is set to TRUE, if one of the inputs is configured as RESET input.

CMS_IO_32BIT_ENCODER

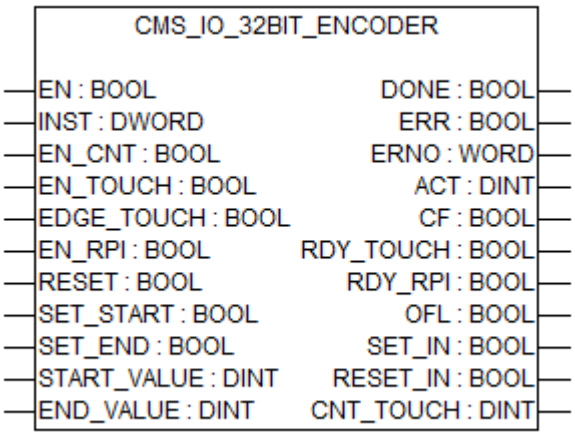


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counter & Encoder

The FM502-CMS provides one ↳ *Chapter 1.6.4.4.3.5.1 “Incremental encoder” on page 5740* function for relative positioning with 3 signals.

Possible operation modes: 11-1, 12-1, 13-1 ↳ *“Operation modes” on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

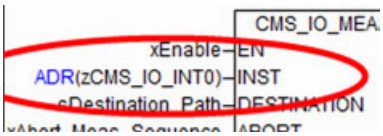


Fig. 188: Example

EN_CNT (enable counter)

Data type: BOOL

If EN_CNT=TRUE , pulse counting of counter is enabled. If EN_CNT=FALSE, no pulse counting is performed and the pulses are lost. If counting has already started and EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_TOUCH (enable touch)

Data type: BOOL

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counters A and B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

EDGE_TOUCH

Data type: BOOL

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of the configured digital input (for counters A and B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of the configured digital input (for counters A and B).

EN_RPI (enable reference point initiator)

Data type: BOOL

A rising edge at input EN_RPI enables a reference point initiator measurement. If input EN_RPI = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counter 0) validates the counter value capture and the counter reset during the capture.

Only one function may be enabled at a time, either the RPI (reference point indicator) or TOUCH (touch trigger measurement). If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, a RPI function will have a higher priority than TOUCH.

RESET (reset counter)

Data type: BOOL

If input RESET = TRUE, the counter value (ACT) is reset to 0. As long as input RESET = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at digital input DI0, DI1, DC2, DC3 causes the function block to reset the value at output ACT.

SET_START (set start value)

Data type: BOOL

If set input SET=TRUE, the counter takes the values from input START_VALUE to transfer it to output ACT. As long as input SET=TRUE, no pulses are counted because the counter is always overwritten by the input START_VALUE.

A rising edge at input causes the function block to store the START_VALUE value and to display this value at output ACT.

SET_END (set end value)

Data type: BOOL

If input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

START_VALUE

Data type: DINT

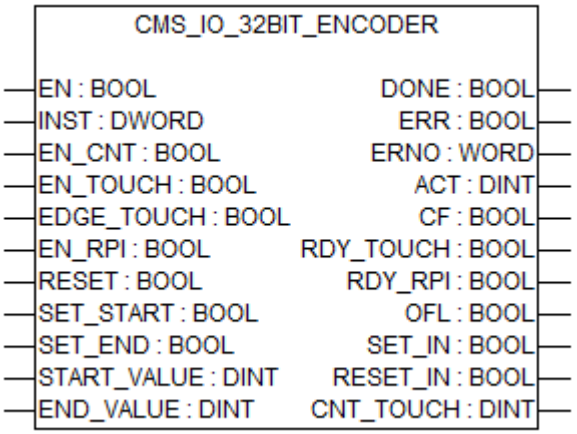
The counter can be set to a start value. This value must be applied to the input START_VALUE.

END_VALUE

Data type: DINT

If the counter reaches the planned input END_VALUE, the binary output CF is set to TRUE and the value is stored.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT (actual value)

Data type: DINT

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

CF (carry flag)

Data type: BOOL

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE, when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

RDY_TOUCH (ready catch/ touch value)

Data type: BOOL

The output RDY_TOUCH is set to TRUE, when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

RDY_RPI (ready reference point initiator)

Data type: BOOL

The output RDY_RPI is set to TRUE, when the RPI operation is done. If input EN_RPI is set to FALSE, the output RDY_RPI is set to FALSE.

OFL (overflow)

Data type: BOOL

The overflow is specified at the output OFL.

The output OFL is set to TRUE, when the counter value ACT passes from -1 to 0 or from 0 to -1.

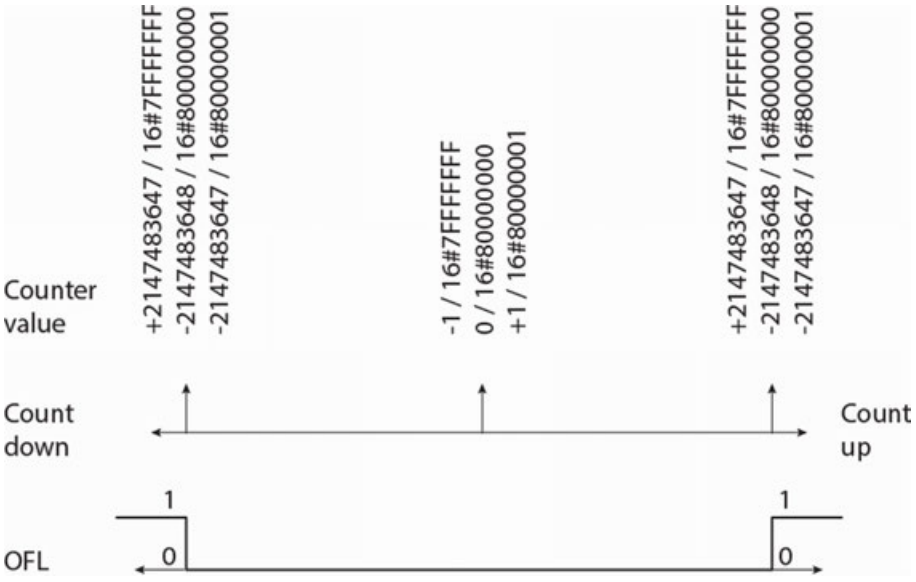


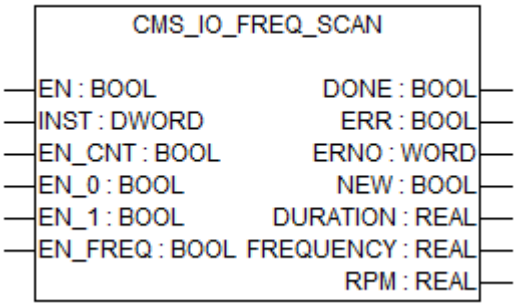
Fig. 189: Detection for output OFL (overflow)

SET_IN (set input counter)
Data type: BOOL
The output SET_IN is set to TRUE, if one of the inputs is configured as SET input.

RESET_IN (reset input counter)
Data type: BOOL
The output RESET_IN is set to TRUE, if one of the inputs is configured as RESET input.

CNT_TOUCH (counter touch value)
Data type: DINT
The output CNT_TOUCH displays the result of the catch/touch trigger action.

CMS_IO_FREQ_SCAN

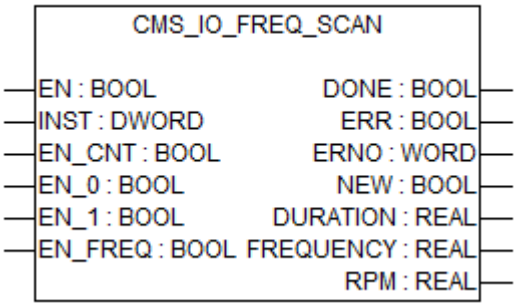


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counter & Encoder

The FM502-CMS provides one channel Z which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs. The function block should be used to control with input EN_CNT, configure the capture on falling edge with input EN_0 or rising edge with input EN_1 of signal, and the specification of the mode of the measurement (*Chapter 1.6.4.4.3.6 "FM502-CMS used as time frequency meter" on page 5747*) with input EN_FREQ.

Possible operation modes: 15-1 *"Operation modes" on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

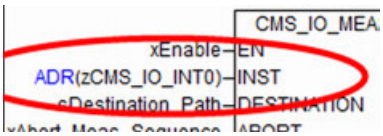


Fig. 190: Example

EN_CNT (enable counter)

Data type: BOOL

If EN_CNT=TRUE , pulse counting of counter is enabled. If EN_CNT=FALSE, no pulse counting is performed and the pulses are lost. If counting has already started and EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_0 (enable 0)

Data type: BOOL

If EN_0=TRUE, the time frequency measurement will be captured on the falling edge of signal.

EN_1 (enable 1)

Data type: BOOL

If EN_1=TRUE, the time frequency measurement will be capture on rising edge of signal.

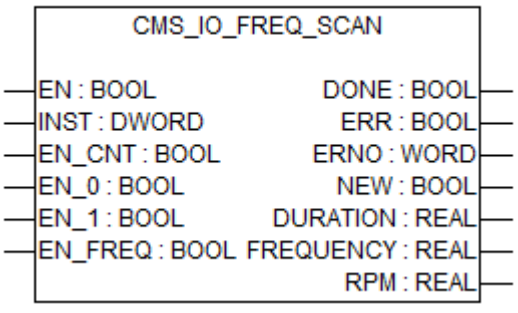
EN_FREQ

Data type: BOOL

If EN_FREQ=FALSE, the time frequency measurement will be specified in time mode and displayed on output DUR (in µs).

If EN_FREQ= TRUE, the time frequency measurement will be specified in frequency and rpm modes and displayed on output FREQ (in Hz) and RPM (in rotation per minute).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

NEW

Data type: BOOL

If output NEW=TRUE, a new timing value is available.

DUR

Data type: REAL

The output DUR (duration) is used for display the result of timing measurement. If the input EN_FREQ=FALSE, measured time is in µs.

FREQ (frequency)

Data type: REAL, unit: Hz

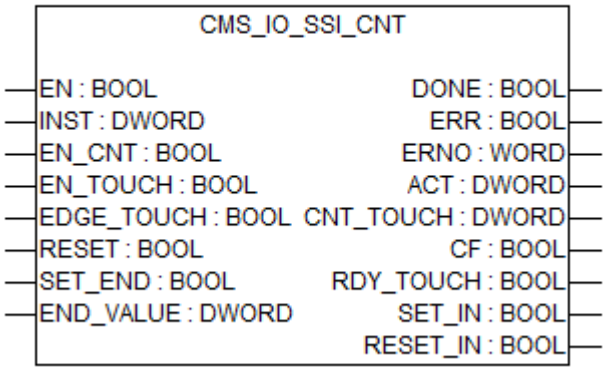
The output FREQ is used to display the result of time measurement. If the input EN_FREQ = TRUE, measured frequency is shown.

RPM (rotations per minute)

Data type: REAL, unit: rpm

The output RPM is used to display the result of timing measurement. If the input EN_FREQ = TRUE, measured speed of rotation is shown.

CMS_IO_SSI_ENC

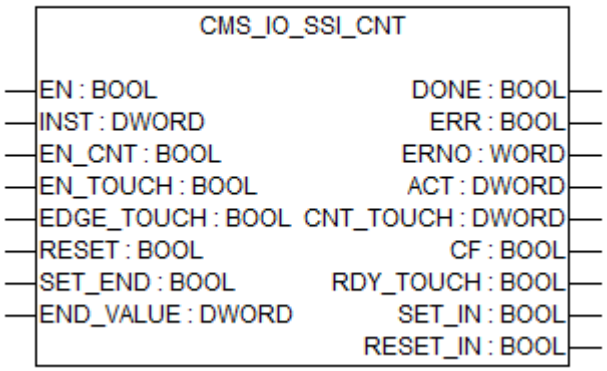


Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Counter & Encoder

The FM502-CMS *Chapter 1.6.4.4.3.5.2 “Absolute SSI encoder” on page 5744*function.

Possible operation modes: 14-1 *“Operation modes” on page 2524*

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance) Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

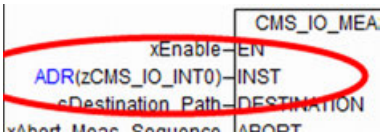


Fig. 191: Example

EN_COUNT (enable frequency output) Data type: BOOL

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

EN_TOUCH (enable touch) Data type: BOOL

A rising edge at input EN_TOUCH enables a catch/touch trigger measurement. If input EN_TOUCH = TRUE, a rising edge at digital inputs DI0, DI1, DC2, DC3 (for counters A and B) causes the function block to store the actual counter value ACT1 and ACT2 and to display this value at output CNT_TOUCH1 and CNT_TOUCH2.

If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

EDGE_TOUCH Data type: BOOL

If EDGE_TOUCH = FALSE, a catch/touch value will be operate on falling edge of the configured digital input (for counters A and B).

If EDGE_TOUCH = TRUE, a catch/touch value will be operate on rising edge of the configured digital input (for counters A and B).

RESET (reset counter) Data type: BOOL

If input RESET = TRUE, the counter value (ACT) is reset to 0. As long as input RESET = TRUE, no pulses are counted because the counter is always overwritten by the value 0.

A rising edge at digital input DI0, DI1, DC2, DC3 causes the function block to reset the value at output ACT.

SET_END (set end value) Data type: BOOL

If input SET_END=TRUE, the counter is set to the value specified at input END_VALUE.

END_VALUE Data type: DWORD

If the counter reaches the planned input END_VALUE, the binary output CF is set to TRUE and the value is stored.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

ACT (actual value)

Data type: DWORD

The current counter value (actual value) can be retrieved at any time using the output ACT of the function block.

CNT_TOUCH

Data type: DWORD

The output CNT_TOUCH (counter touch value) displays the result of the catch/touch trigger action.

CF (carry flag)

Data type: BOOL

If the zero crossover indicator CF=TRUE, this output indicates the sign of the actual counter value ACT. It is set to FALSE, when counter value ACT is less than or equal to zero. It is set to TRUE otherwise.

RDY_TOUCH (ready catch/ touch value)

Data type: BOOL

The output RDY_TOUCH is set to TRUE, when a new catch/touch value is available. If input EN_TOUCH is set to FALSE, the output RDY_TOUCH is set to FALSE.

SET_IN (set input counter)

Data type: BOOL

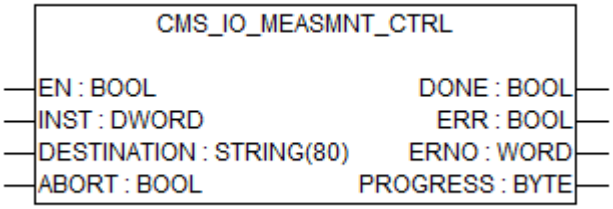
The output SET_IN is set to TRUE, if one of the inputs is configured as SET input.

RESET_IN (reset input counter)

Data type: BOOL

The output RESET_IN is set to TRUE, if one of the inputs is configured as RESET input.

CMS_IO_MEASMNT_CTRL



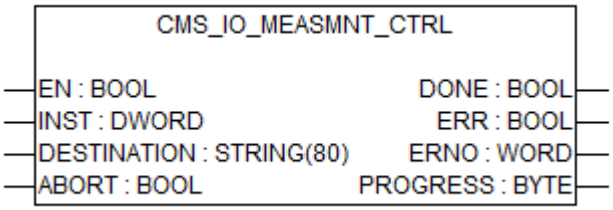
Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0, FM502-CMS firmware: V1.0
Type	Function block with historical values
Group	Measurement

Possible operation modes: 1-1, 2-1 *“Operation modes” on page 2524*

Control for *Chapter 1.6.4.4.3.3 “FM502-CMS analog measurement” on page 5730*.

CMS_IO_MEASMNT_CTRL needs the destination path where to store the measurement files. When EN = TRUE the configured measurement starts and the files are stored at destination path as ZIP file. The ZIP file includes the WAV files. Each WAV file per activated channel and encoder. When there are activated channels with different sample rate, there will be another encoderWAV file for every channel with different sample rate. The time for measurement and transmitting depends on the number of active channels, the sample rate, the record length value.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

INST (instance) Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

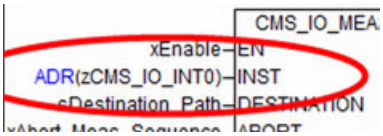


Fig. 192: Example

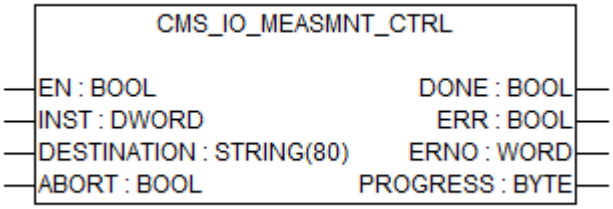
DESTINATION Data type: STRING

To store the measurement files after the measurement progress. Example: flashdisk/Meas.zip. Filename in 8.3 characters.

ABORT Data Type: BOOL

A rising edge at input stops the measurement or transmitting. When ABORT = TRUE, measurement data will be deleted.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

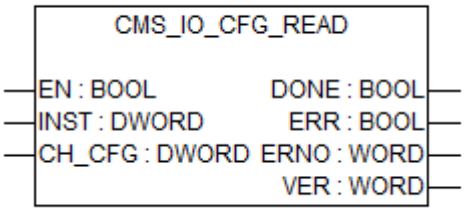
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

PROGRESS Data type: BYTE

Percentage of transmitting the measurement files from Function Module to processor module.

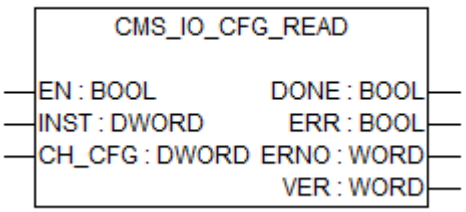
CMS_IO_CFG_READ



Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0
Available as of device firmware	V1.0
Type	Function block with historical values
Group	Configuration

Read configuration of analog channels from FM502-CMS.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance) Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

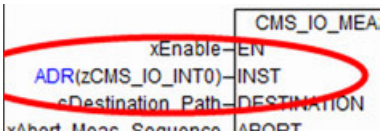


Fig. 193: Example

CH_CFG (channel configuration)

Data type: DWORD (address of CMS_IO_CFG_FM502_TYPE)

Address where channel configuration data will be read from and written into the function module.

↳ *Chapter 1.6.4.4.3.2.2.1 “Parameter set” on page 5726*

Output description

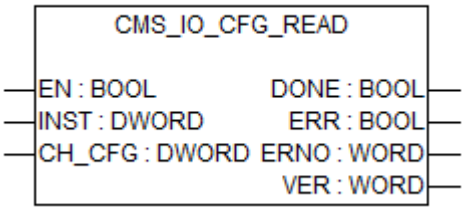


Fig. 194: Function block CMS_IO_CONFIG_READ

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

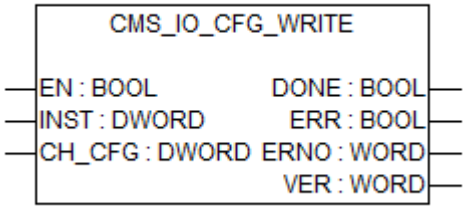
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

VER (version index)

Data type: WORD

Version index of configuration data.

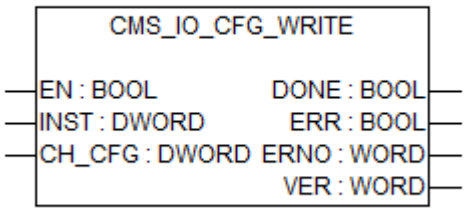
CMS_IO_CFG_WRITE



Parameter	Value
Included in library	CMS_IO_AC500_V24.lib
Available as of firmware	V2.4.0
Available as of device firmware	V1.0
Type	Function block with historical values
Group	Configuration

With this function block you can change the configuration of the function module also during the run time of the CODESYS. Configuration cannot be written while function block CMS_IO_MEASMNT_CTRL is active. When configuration data has an error (plausibility will be checked in function module) or will be sent while CMS_IO_MEASMNT_CTRL is active, configuration data will be deleted and the old one will be used. If the function module gets an error while configuration data is written to the function module, the function block CMS_IO_CFG_READ output DONE will be TRUE, but errors can be seen in diagnosis system of processor module. Encoder/counter function blocks will be reset, when the configuration is written.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

INST (instance)

Instance description of the Function Module that should be controlled via this function block. The variable is automatically generated during the configuration with the Automation Builder. Use the operator ADR to get the address of the variable.

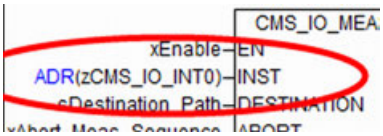


Fig. 195: Example

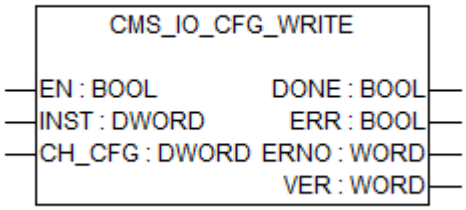
CH_CFG (channel configuration)

Data type: DWORD (address of CMS_IO_CFG_FM502_TYPE)

Address where channel configuration data will be read from and written into the function module.

↳ *Chapter 1.6.4.4.3.2.2.1 “Parameter set” on page 5726*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

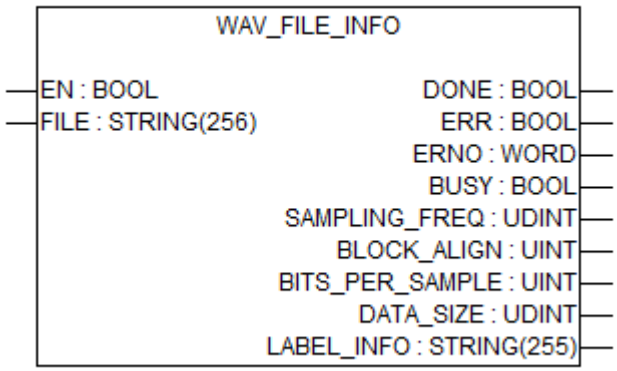
VER (version index) Data type: WORD

Version index of configuration data.

1.5.8.3 WAV file library for data handling

1.5.8.3.1 Function blocks

WAV_FILE_INFO

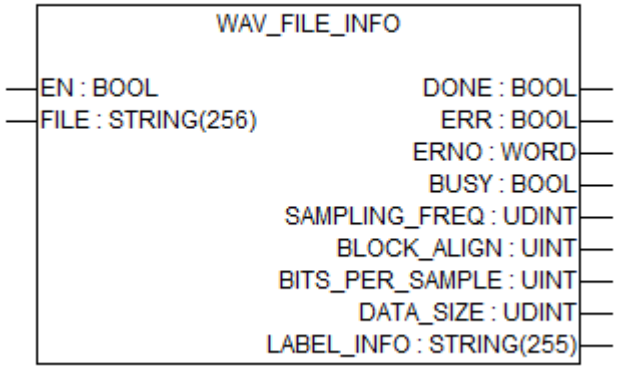


Parameter	Value
Included in library	WAV_FILE_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	-

Function block WAV_FILE_INFO is used to retrieve the WAV file properties. This function block does not read the measured data values present in the file. The user can get the file information on the WAV file which may be needed before performing the read and write operations.

Function block WAV_FILE_INFO works with the file in the READ mode.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

FILE (file path and name)

Data type: STRING

Input is used to enter the absolute path and name of the file whose information is required.

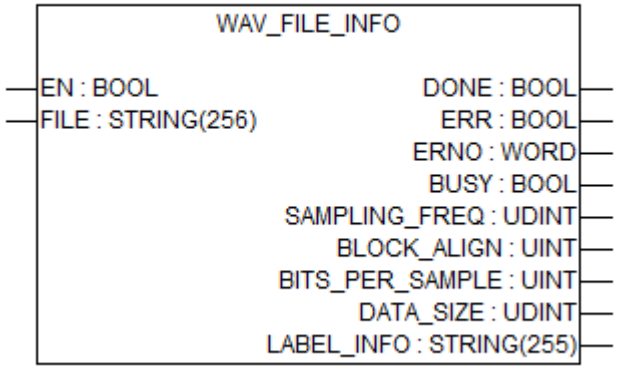
If the path or the file name given is incorrect or invalid, an error occurs at output ERNO and output ERR = TRUE.

Example of the value of FILE

File is present on the flash: flashdisk\Sample.wav

File is present on the memory card: SDCard\Sample.wav

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY

Data type: BOOL

Output BUSY stays TRUE, till the processing of the function block is in progress. Once the processing is done, output BUSY becomes FALSE and output DONE becomes TRUE. In the course of output ERR=TRUE, or the input EN=FALSE, output BUSY becomes FALSE.

SAMPLING_FREQ (sampling frequency)

Data type: UDINT

Output SAMLING_FREQ captures the sampling frequency of the WAV file. The sampling frequency is the number of samples of data, stored in one second.

BLOCK_ALIGN

Data type: UDINT

Output BLOCK_ALIGN captures the respective value of the WAV file. The value gives the number of bytes present in one sample of data recorded in the WAV file.

The BLOCK_ALIGN value is a function of bits per sample, number of channels and sampling frequency *Chapter 1.5.8.1 “System technology” on page 2519*.

BITS_PER_SAMPLE

Data type: UINT

Output BITS_PER_SAMPLE stores the number of bits present in one sample of the data.

DATA_SIZE

Data type: UDINT

Output DATA_SIZE stores the value of number of bytes from the measured data which is present in the WAV file.

LABEL_INFO

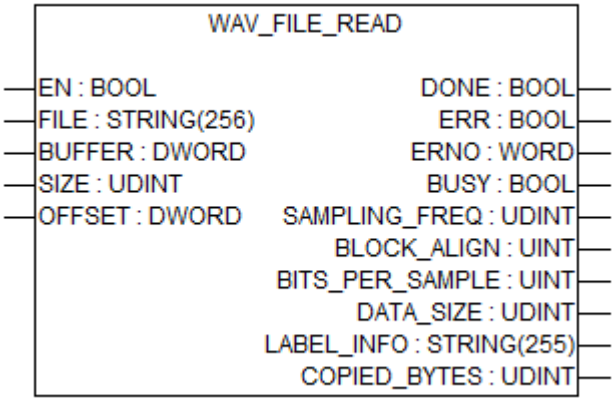
Data type: STRING[255]

Output LABEL_INFO captures the label information, which stores the channel and encoder information.

Function call in ST

```
Inst_File_Info(
        EN:=TRUE,
        FILE:=:= 'flashdisk\Sample.WAV',
        xInfoDone:= Inst_File_Info.DONE,
        xInfoError:= Inst_File_Info.ERR,
        wInfoErNo:= Inst_File_Info.ERNO
        dwInfoSamplingFreqInst_File_Info.SAMPLING_FREQ
        wInfoBlockAlign:= Inst_File_Info.BLOCK_ALIGN
        wInfoResolution:= Inst_File_Info.BITS_PER_SAMPLE
        dwInfoDataSize:= Inst_File_Info.DATA_SIZE
        sInfoLable:= Inst_File_Info.LABLE_INFO);
```

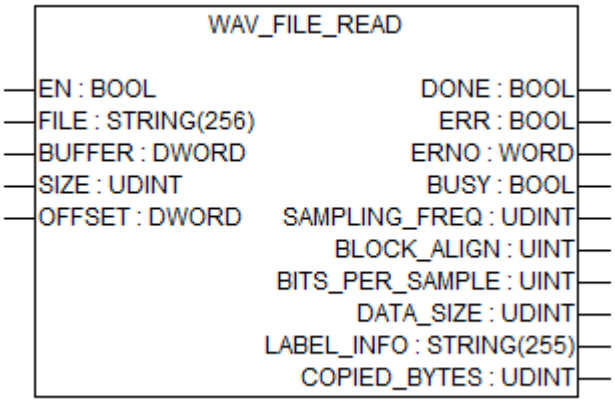
WAV_FILE_READ



Parameter	Value
Included in library	WAV_FILE_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	-

Function block WAV_FILE_READ is used to retrieve the WAV file properties and to read the measured data stored in the file.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

FILE (file path and name)

Data type: STRING

Input is used to enter the absolute path and name of the file whose information is required.

If the path or the file name given is incorrect or invalid, an error occurs at output ERNO and output ERR = TRUE.

Example of the value of FILE

File is present on the flash: flashdisk\Sample.wav

File is present on the memory card: SDCard\Sample.wav

BUFFER (memory buffer)

Data type: DWORD, Default value = 0

Input BUFFER contains the memory address of the array where all the measured data is stored after reading.

Example

If the measured data is stored in the array 'abyReadData[1..100]', then BUFFER=ADR(abyReadData).

SIZE (size of data to be read)

Data type: UDINT, Default value = 1, Range ≥ 1 (block align) and ≤ data size of the file.

Input SIZE gives the number of bytes of the measured data to be read and stored in the buffer. The function block would throw an error, if value of SIZE given as 0. For a given file, the input SIZE value must always be the integral multiple of its block align value.

Example

With data size = 1000 and block align = 4, the valid value of SIZE is in the range from 4 to 1000.

SIZE must not exceed the data size of the WAV file.

OFFSET (memory offset)

Data type: UDINT, Default value = 0, Range ≥ 0 and ≤ data size of the file.

Input OFFSET is an offset in terms of number of bytes. It points to the byte from where the reading should start.

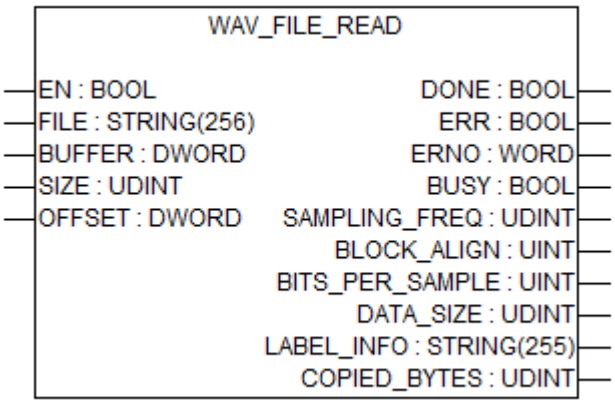
Example

If the value of OFFSET = 10, the function block will read from the 11th measured data present inside the file.

OFFSET is an integral multiple of the block align

OFFSET must not exceed the data size of the WAV file.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY

Data type: BOOL

Output BUSY stays TRUE, till the processing of the function block is in progress. Once the processing is done, output BUSY becomes FALSE and output DONE becomes TRUE. In the course of output ERR=TRUE, or the input EN=FALSE, output BUSY becomes FALSE.

SAMPLING_FREQ (sampling frequency)

Data type: UDINT

Output SAMLING_FREQ captures the sampling frequency of the WAV file. The sampling frequency is the number of samples of data, stored in one second.

BLOCK_ALIGN

Data type: UDINT

Output BLOCK_ALIGN captures the respective value of the WAV file. The value gives the number of bytes present in one sample of data recorded in the WAV file.

The BLOCK_ALIGN value is a function of bits per sample, number of channels and sampling frequency *Chapter 1.5.8.1 “System technology” on page 2519*.

BITS_PER_SAMPLE

Data type: UINT

Output BITS_PER_SAMPLE stores the number of bits present in one sample of the data.

DATA_SIZE

Data type: UDINT

Output DATA_SIZE stores the value of number of bytes from the measured data which is present in the WAV file.

LABEL_INFO

Data type: STRING[255]

Output LABEL_INFO captures the label information, which stores the channel and encoder information.

COPIED_BYTES

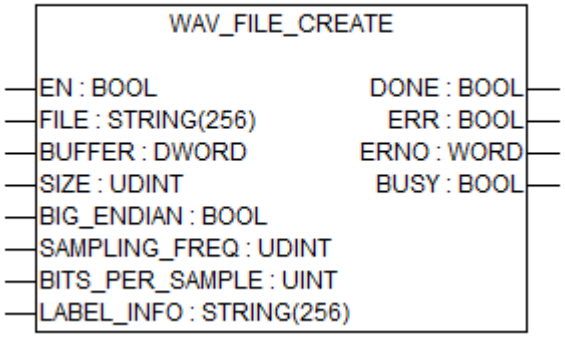
Data type: UDINT

Output COPIED_BYTES stores the number of bytes, the function block read and stored in the buffer area.

Function call in ST

```
Inst_File_READ(
        EN:=TRUE,
        FILE:=:= 'flashdisk\Sample.WAV',
        BUFFER:= ADR(abyReadData),
        SIZE:= 100,
        OFFSET:= 0,
        xReadDone:= Inst_File_Read.DONE,
        xReadError:= Inst_File_Read.ERR,
        wReadErNo:= Inst_File_Read.ERNO,
        dwReadSamplingFreq:= Inst_File_Read.SAMPLING_FREQ,
        wReadBlockAlign:= Inst_File_Read.BLOCK_ALIGN,
        wReadResolution:= Inst_File_Read.BITS_PER_SAMPLE,
        dwReadDataSize:= Inst_File_Read.DATA_SIZE,
        ReadLabel:= Inst_File_Read.LABEL_INFO,
        udiReadBytes:= Inst_File_Read.COPIED_BYTES);
```

WAV_FILE_CREATE

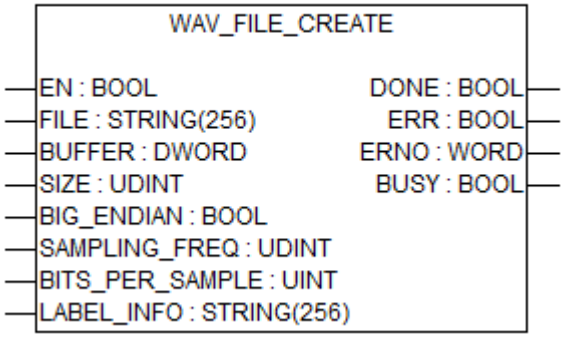


Parameter	Value
Included in library	WAV_FILE_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	-

Function block WAV_FILE_CREATE is used to create a new file by giving the required properties of the file.

This function block creates a new WAV file by handling the file in the WRITE mode. It is possible to write the data in the big or the little endian format.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

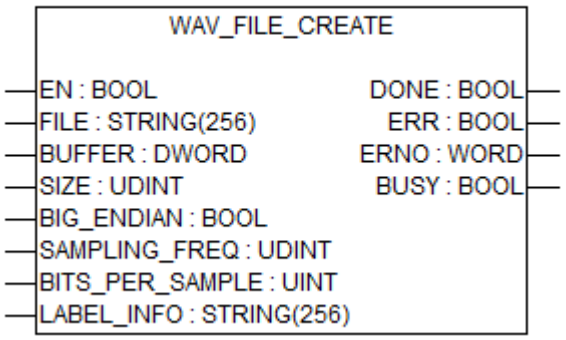
The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

FILE (file path and name)	Data type: STRING Input is used to enter the absolute path and name of the file whose information is required. If the path or the file name given is incorrect or invalid, an error occurs at output ERNO and output ERR = TRUE.
Example of the value of FILE	File is present on the flash: flashdisk\Sample.wav File is present on the memory card: SDCard\Sample.wav
BUFFER (memory buffer)	Data type: DWORD, Default value = 0 Input BUFFER contains the memory address of the array where all the measured data is stored after reading.
Example	If the measured data is stored in the array 'abyReadData[1..100]', then BUFFER=ADR(abyReadData).
SIZE (size of data to be read)	Data type: UDINT, Default value = 1, Range ≥ 1 (block align) and ≤ data size of the file. Input SIZE gives the number of bytes of the measured data to be read and stored in the buffer. The function block would throw an error, if value of SIZE given as 0. For a given file, the input SIZE value must always be the integral multiple of its block align value.
Example	With data size = 1000 and block align = 4, the valid value of SIZE is in the range from 4 to 1000.
	SIZE must not exceed the data size of the WAV file.
BIG_ENDIAN	Data type: BOOL, default value: FALSE = little endian Input BIG_ENDIAN decides if the file to be created will be a big endian or a little endian. BIG_ENDIAN = TRUE = big endian data file.
SAMPLING_FREQ (sampling frequency)	Data type: UDINT, default value: 1, range: > 0. Input SAMPLING_FREQ enters the sampling frequency of the WAV file to be created. The sampling frequency is the number of samples of data stored in one second.
BITS_PER_SAMPLE	Data type: UINT Input BITS_PER_SAMPLE enters the number of bits present in one sample of the data. ● 16, 24, 32 or 64 are the valid values for this input.
LABEL_INFO	Data type: STRING[255] Input LABEL_INFO enters the label information. The label information normally stores the channel and encoder information.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY

Data type: BOOL

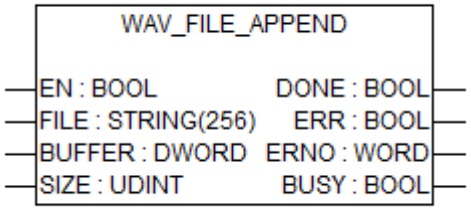
Output BUSY stays TRUE, till the processing of the function block is in progress. Once the processing is done, output BUSY becomes FALSE and output DONE becomes TRUE. In the course of output ERR=TRUE, or the input EN=FALSE, output BUSY becomes FALSE.

Function call in ST

```
Inst_File_CREATE(
         EN:=TRUE,
         FILE:=:= 'flashdisk\Sample.WAV',
         BUFFER:= ADR(abyCreateData),
         SIZE:= 100,
         BIG_ENDIAN:= FALSE,
         SAMPLING_FREQ:= 50000,
```

```
BITS_PER_SAMPLE:= 16,
xCreateDone:= Inst_File_Info.DONE
xCreateError:= Inst_File_Info.ERR
wCreateErNo:= Inst_File_Info.ERNO
dwCreateBusy:= Inst_File_Create.BUSY);
```

WAV_FILE_APPEND

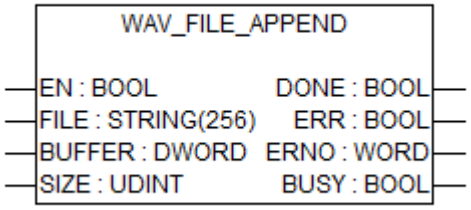


Parameter	Value
Included in library	WAV_FILE_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	-

Function block WAV_FILE_APPEND is used if you want to add more data samples to an existing WAV file.

This function block appends the WAV file by handling the file in the WRITE mode.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

FILE (file path and name)

Data type: STRING

Input is used to enter the absolute path and name of the file whose information is required.

If the path or the file name given is incorrect or invalid, an error occurs at output ERNO and output ERR = TRUE.

Example of the value of FILE	File is present on the flash: flashdisk\Sample.wav File is present on the memory card: SDCard\Sample.wav

BUFFER (memory buffer)

Data type: DWORD, Default value = 0

Input BUFFER contains the memory address of the array where all the measured data is stored after reading.

Example	If the measured data is stored in the array 'abyReadData[1..100]', then BUFFER=ADR(abyReadData).

SIZE (size of data to be read)

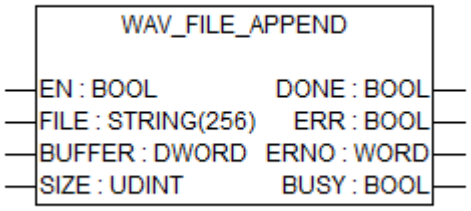
Data type: UDINT, Default value = 1, Range ≥ 1 (block align) and ≤ data size of the file.

Input SIZE gives the number of bytes of the measured data to be read and stored in the buffer. The function block would throw an error, if value of SIZE given as 0. For a given file, the input SIZE value must always be the integral multiple of its block align value.

Example	With data size = 1000 and block align = 4, the valid value of SIZE is in the range from 4 to 1000.

SIZE must not exceed the data size of the WAV file.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.5.3 “Error messages of the AC500 V2 function block libraries” on page 735*).

BUSY

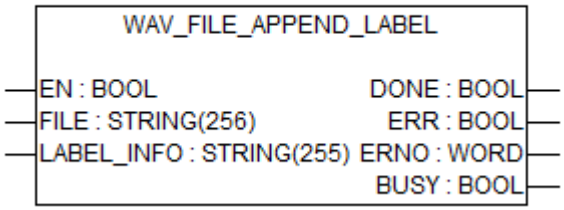
Data type: BOOL

Output BUSY stays TRUE, till the processing of the function block is in progress. Once the processing is done, output BUSY becomes FALSE and output DONE becomes TRUE. In the course of output ERR=TRUE, or the input EN=FALSE, output BUSY becomes FALSE.

Function call in ST

```
Inst_File_APPEND(
         EN:=TRUE,
         FILE:=:= 'flashdisk\Sample.WAV',
         BUFFER:= ADR(abyAppendData),
         SIZE:= 100,
         xAppendDone:= Inst_File_Append.DONE
         xAppendError:= Inst_File_Append.ERR
         wAppendErNo:= Inst_File_Append.ERNO
         dwAppendBusy:= Inst_File_Append.BUSY);
```

WAV_FILE_APPEND_LABEL

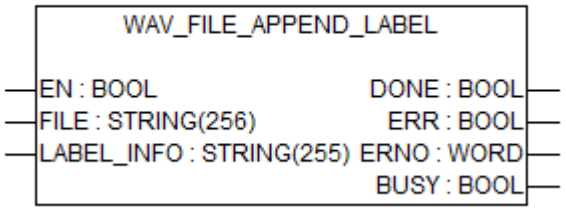


Parameter	Value
Included in library	WAV_FILE_AC500_V24.lib
Available as of firmware	V2.4.0
Type	Function block with historical values
Group	-

Function block WAV_FILE_APPEND_LABEL is used if you want to add a label to an existing WAV file.

This function block appends the WAV file by handling the file in the WRITE mode.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The function block is activated by a TRUE at the input EN.

A FALSE keeps the function block deactivated.

Is the function block activated, the values being present at the inputs are processed and the output values are delivered.

FILE (file path and name)

Data type: STRING

Input is used to enter the absolute path and name of the file whose information is required.

If the path or the file name given is incorrect or invalid, an error occurs at output ERNO and output ERR = TRUE.

Example of the value of FILE

File is present on the flash: flashdisk\Sample.wav

File is present on the memory card: SDCard\Sample.wav

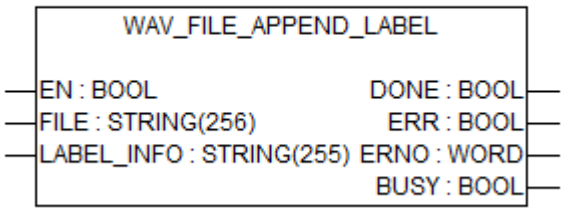
LABEL_INFO

Data type: STRING

Input is used to append the information as a label to the WAV file.

If the total length of the label information is bigger than 256 signs, an error occurs at output ERNO and output ERR = TRUE.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Output DONE indicates the processing state of the function block.

After completion or abortion of processing (due to an error), DONE is set to TRUE, for one cycle, as long as input EN = TRUE.

This output always has to be considered together with output ERR. If ERR is TRUE, then the corresponding error code is displayed by output ERNO.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE / FALSE	-

Output ERR indicates, whether an error occurred during control processing.

In case of error, the error number can be read at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

Output ERNO provides an error identifier, if an invalid value was applied to an input, or if an error occurred during request processing. Then the error number is displayed at the output (see, error messages).

Output ERNO always has to be considered together with output ERR. The value output at ERNO is only valid, if output ERR is TRUE.

BUSY

Data type: BOOL

Output BUSY stays TRUE, till the processing of the function block is in progress. Once the processing is done, output BUSY becomes FALSE and output DONE becomes TRUE. In the course of output ERR=TRUE, or the input EN=FALSE, output BUSY becomes FALSE.

Function call in ST

```
Inst_File_APPEND_LABEL(
         EN:=TRUE,
         FILE:= 'flashdisk\Sample.WAV',
         LABEL INFO:= 'Label information',
         xAppendDone:= Inst_File_Append_Label.DONE
         xAppendError:= Inst_File_Append_Label.ERR
         wAppendErNo:= Inst_File_Append_Label.ERNO
         dwAppendBusy:= Inst_File_Append_Label.BUSY);
```

zWAV_FILE_BYTES_TO_STRING

Internal function block to retrieve the string of label section of the WAV-File. The data stored in terms of bytes is converted into the string equivalent.

1.5.8.3.2 Structures

Structure	Function
zWAV_FILE_LOOKUP_TYPE	Internal structure not for the user access. To create a look up table of ASCII format and characters.
zWAV_FILE_STRING_TO_ASCII_TYPE	Internal structure not for the user access. Has the elements for ASCII format and character of a string.
zWAV_FILE_STRUCT_TYPE	Internal structure not for the user access. Has the structure same as the WAV-File standards

1.5.8.3.3 Variables

Table 152: Global variables

Variable	Function
WAV_FILE_VERSION_INFORMATION	Stores all the version information of the file along with the change log. It has no variable declared inside this section.
dwWAV_FILE_VisuBackgroundColor Group: WAV_FILE_VISU_COLOR_INFO	Visualization elements: background color 16#00<B><G><R>
dwWAV_FILE_VisuTitleColor Group: WAV_FILE_VISU_COLOR_INFO	Visualization elements: title background color 16#00<B><G><R>

1.5.8.3.4 Visualization

A visualization object can be used to show the actual values of all inputs and outputs of the instance of the function block. The visualization can also be used to control the function block by those inputs which are not connected inside the program.

In CODESYS, you can add a visualization object in the project and select the required function block visualization.

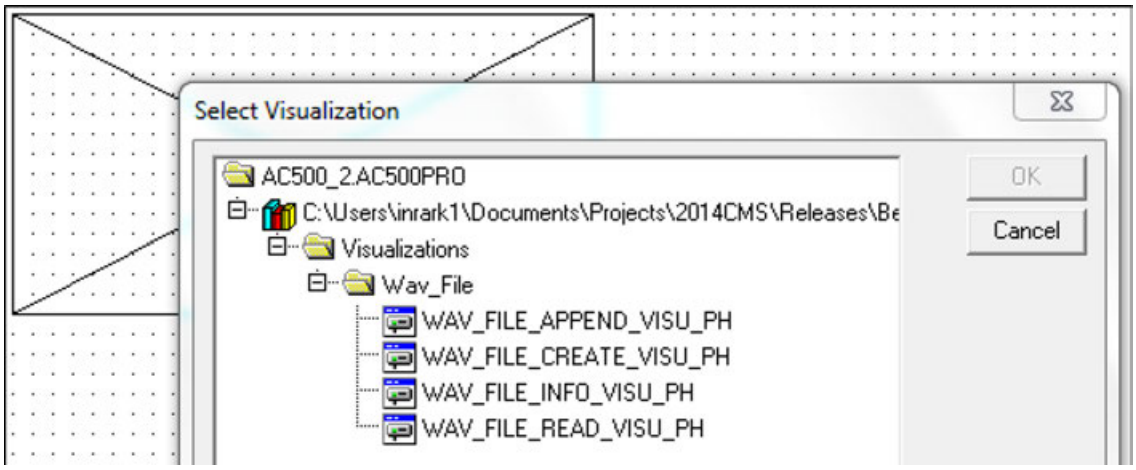


Fig. 196: Visualization: Selection of function blocks

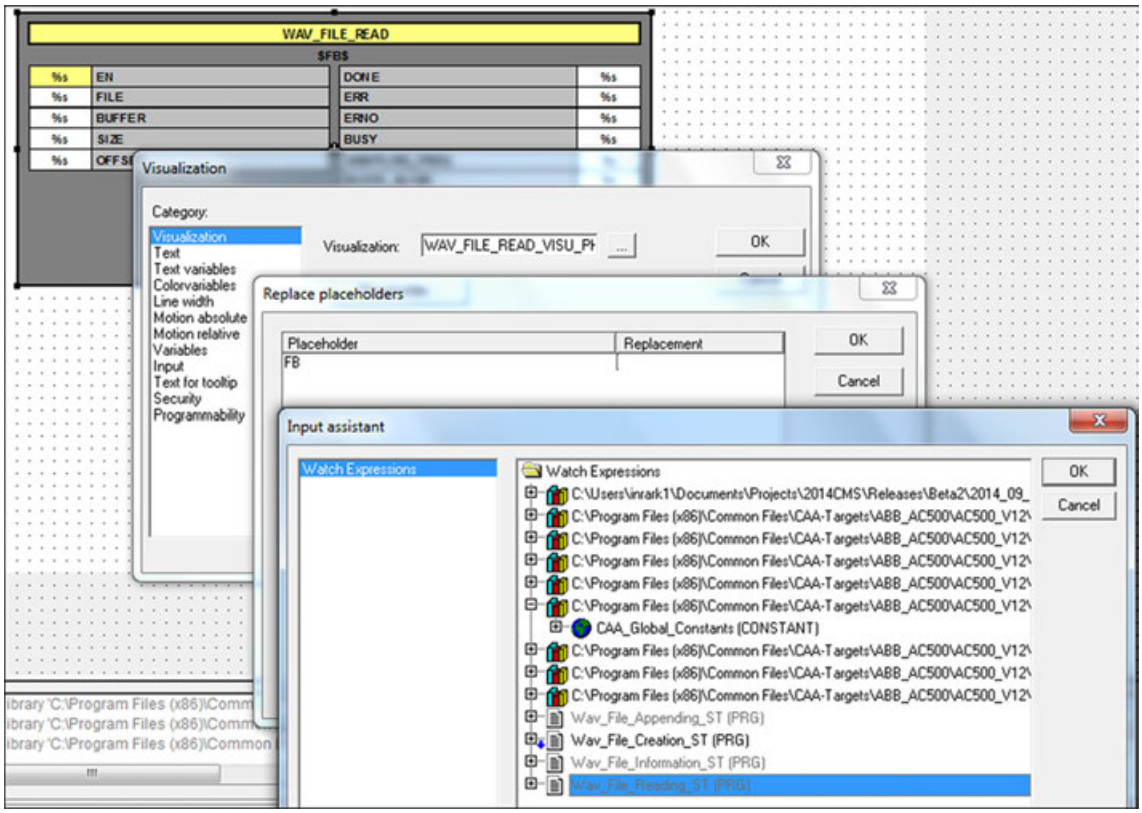


Fig. 197: Example: WAV_FILE_READ_VISU_PH

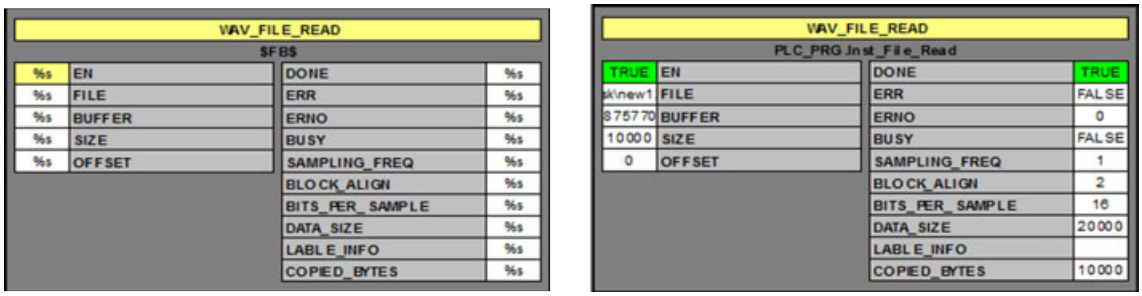


Fig. 198: Example: WAV_FILE_READ_VISU_PH in offline and online mode

Colors

- WHITE: Current FALSE and should be FALSE in normal operation.
- GREEN: Current TRUE and should be TRUE in normal operation.
- YELLOW: Current FALSE but should be TRUE in normal operation.
- RED: Current TRUE but should be FALSE in normal operation.

Color of the background can be changed by writing a value to the global variable dwWAV_FILE_VisuBackgroundColor.

Color of the title can be changed by writing a value to the global variable dw WAV_FILE_VisuTitleColor.

1.5.8.4 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.9 Motion control library

Safety instructions

- All pertinent state, regional, and local safety regulations must be observed when installing and using this product. When functions or devices are used for applications with technical safety requirements, the relevant instructions must be followed.
- Read the complete safety instructions of the user's manuals for the drives you are using, before installation and commissioning.
- Read all safety instructions of the AC500 PLC. See System description AC500 or chapter *Chapter 1.6.1.4 “Regulations” on page 3709* in the online help.
- Read the Important user Information. See chapter *Chapter 1.6.1.1 “Safety instructions” on page 3697* in the online help.

1.5.9.1 Preconditions for the use of the libraries

The user has to read the following instructions and documents before using the libraries:

The library package has been released for the software and firmware versions listed in the readme file of Automation Builder only (see “Help ➔ Automation Builder Release Notes”) . In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product, software or firmware versions. The error-free operation of the HA library with other devices, software or firmware versions should be possible but cannot be guaranteed and may need adaptations e. g. of example programs.

The first version of Motion Control Library Package PS5611-Motion has been released with Automation Builder 2.4.0.There after the package is updated with several changes. For details on all changes please refer PS5611-Motion release note area from Automation Builder release notes.

The Motion control package contains follows libraries:

Library	Automation Builder	PLC firmware
ABB_MotionControl_AC500	AB 2.4.0 or higher	AC500 V3 firmware version 3.3.1 or higher AC500-eCo V3 firmware version 3.4.0 or higher
ABB_Ecat_CiA402_AC500		
ABB_MathFunctions_AC500		
ABB_MotionContro-lEco_AC500 (kernel blocks for Eco V3 PLCs)		
ABB_MotionControl-Load_AC500	AB 2.5.0 or higher	AC500 V3 firmware version 3.5.0 or higher

The version 3.0.0 of the Motion Control Library Package PS552-MC-E has been released for:

- AC500 and AC500-eCo, AC500 Firmware version 2.3
- Control Builder Plus V2.3
- CODESYS V2.3.9.x

The PS552-MC libraries V3.0.0 have been tested with the following product / firmware / software versions:

- CM579-ETH EtherCAT Communication Module firmware 2.6.7
- Bosch Indra Drive Cs FW MPB-16V20-D5-1-NNN-NN
- ACSM1 FW 1510 + FECA-01 FW 109
- ACS355 FW 5040 + FECA-01 FW 109
- ACS355 FW 4050 + FPBA-01 FW 201B

- ACSM1 Motion FW 1510 + FPBA-01 FW 201B
- ACSM1 FW v1.8.2

In no event will ABB or its representatives be liable for loss of data, profits, revenue or consequential, incidental or other damage that may result from the use of other versions of product / software / firmware versions. The error-free operation of the PS552-MC-E V3.0.0 with other devices / software / firmware versions should be possible but can not be guaranteed and may need adaptations e.g. of example programs.

CAUTION!
Generally, the user in all applications is fully and alone responsible for checking all functions carefully, especially for safe and reliable operation.

The function blocks contained in the library can only be executed in RUN mode of the PLC, but not in simulation mode.

Limits on number of synchronized axis

There are limits on the minimum EtherCAT cycle time, user can configure in each PLC type.

Table 153: Details on the limits on the minimum EtherCAT cycle time

PLC type	PM5630	PM5650	PM5670
Min. EtherCAT master cycle time	2 ms	1 ms	0,5 ms

Other than the above limits, there is also limits on configuring the number of synchronized axis in each PLC type. This limits is based on the EtherCAT master cycle time configured under EtherCAT master.

Table 154: Details on the limits for each PLC type

PLC type	PM5630	PM5650	PM5670
Number of synchronized axis in 1 ms	-	8	16
Number of synchronized axis in 2 ms	4	16	32
Number of synchronized axis in 4 ms	8	32	64

“Number of axis” is counted in Automation Builder is based on the number of Kernel function block instance declared in the IEC application. In this way, it is made sure all real and virtual axis are counted.

User can increase the EtherCAT cycle time to accommodate more “Number of axis” in the same PLC type.

User can use the *[Statistics]* tab from Automation Builder to see how many axis are supported for the particular PLC type and for the EtherCAT master cycle time configured. Once the axis is configured user need to update the *[Statistics]* tab by *“Generate Code”* to get the updated information.

Automation Builder allows an additional axis than what is mentioned in the above table to support one virtual axis additionally.

> *Please remove any Kernel function block instance which is declared but not used in the application to get the correct number of axis calculated by Automation Builder under the [Statistics] tab.*

1.5.9.2 Overview

The PS552-MC is a Motion Control software to create Motion Control applications based on function blocks according to the standard of PLCopen Motion Control ↳ *Chapter 1.5.9.3 “PLCopen” on page 2587*.These function blocks can be used for Central Motion Control as well as for drive-based Motion Control axis implementations.

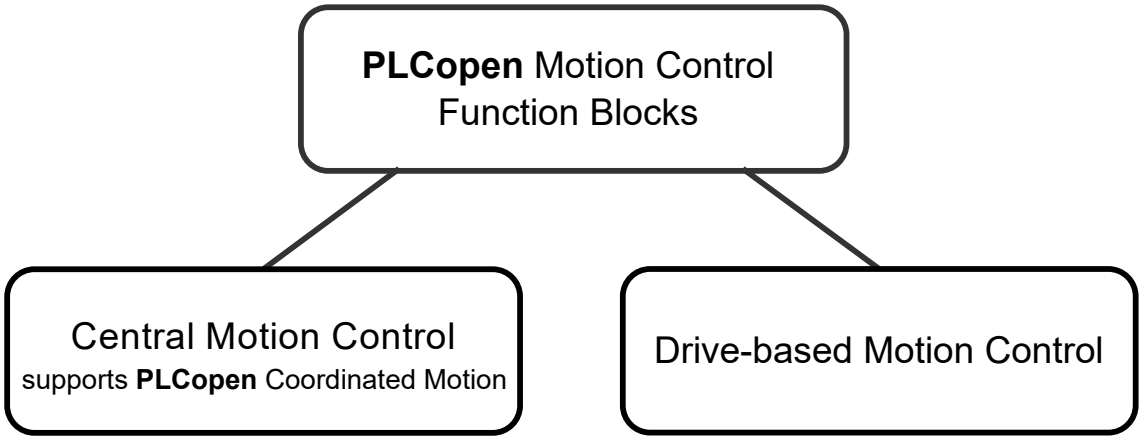


Fig. 199: Use of PLCopen function blocks for different axis implementations provided by PS552-MC

The Central Motion Control axis implementation covers a wide range of possible Motion Control functionalities starting from single axis movements to master-follower axes to perform electronic gearing and CAM functions up to coordinated Motion Control in cartesian coordinates with optional kinematic transformations to realize even a portal or robotic application.

There are different drive-based Motion Control implementations for specific ABB drives. PLCopen function blocks can be used to control the motion capabilities of the drives from AC500 PLC.

This documentation contains the following chapters:

- Overview
  In the subsequent chapters general information are provided for a better understanding of Motion Control with AC500 PLC and PS552-MC. There is also an overview of the available PLCopen function blocks and their compatibility with Central Motion Control and the provided drive-based Motion Control axis implementations.
- PLCopen
  The principle of the PLCopen Motion Control standard is explained as well as how PLCopen function blocks can be used to create PLC Motion Control application programs.
- Central Motion Control (PLC-Based)
  This chapter explains how PLC-based Motion Control with AC500 can be realized and how it can be used in combination with the available PLCopen function blocks.
- PLC-based Motion Control Fluid Power Extension or Load Control
  This chapter explains how the PLCopen part 6 Fluid Power - extension also called “Load Control” can be used to practically realize also a form of Torque control (or -profiling) and how it can be used in combination with the available PLCopen function blocks and switch between Torque/Load control and position control.

- Drive-Based Motion
  Control Different realizations for specific drives and modules are introduced and explained. Drive or module dependent restictions as well as device dependent commissioning instruction are given.
- PLCopen function blocks
  This chapter covers the documentation of all included PLCopen Motion Control function blocks of this product.
- Glossary
  All abbreviations used in this documentation are listed here.

1.5.9.2.1 Motion Control with PS552-MC

With PS552-MC different Motion Control system structures are possible. Independently of the system structure a typical Motion Control application consists of the following system elements:

- An application program which contains PLCopen function blocks that defines the general application behavior and logics.
- A profile generator which generates a position profile based on the dynamic specifications of the application program to guide the axis to the desired positions.
- A position control loop which outputs a speed reference signal to minimize the following error.

To achieve the best system structure for an application these components can be separated into different devices. Each type of structure has its own kind of interface and type of signals which need to be transferred between the interacting devices.

All shown Motion Control system structures (drive based or central Motion Control with or without position control loop) can be combined together in the same application program for a Motion Control project.

1.5.9.2.2 PLC-based motion control

With central Motion Control based on an AC500 PLC and PS552-MC the application program is done in the PLC but also the profile generator. The implementation of the profile generator is based on a set of function blocks which are named Compact Motion (CMC).

The profile generator of many possible axes is centrally placed inside the AC500 PLC. Therefore multiaxis motion functionalities become easily available and can be accessed by PLCopen function blocks. As a result, Motion Control functionalities are almost drive independent.

Available motion control functionalities:

- Simple axis Movements
- Electronic Gearin
- Electronic CAMs
- Position Profiles
- Velocity Profiles
- Acceleration Profiles
- Load control (Torque profilling)
- Simple Movements in 3D
- Path Movements in 3D
- Kinematic Transformations for Roboitc Applications

Then the output is a position reference signal which the drive will follow. A new position reference value will be calculated with every cycle of the PLC and has to be transferred to the drive, which demands real time capabilities to the PLC and to the communication channel. A real time fieldbus like EtherCAT is needed. The feedback of the actual position can be used for supervision purposes during operation and is needed to adjust the value of the position reference before the drive will be enabled.

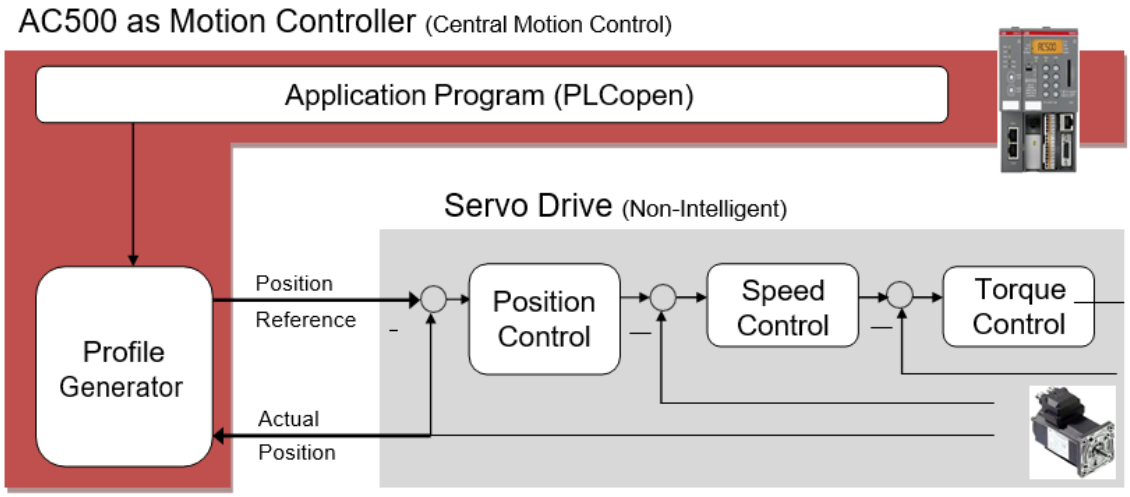


Fig. 200: System structure of Central Motion Control with AC500 PLC and PS552-MC

With central Motion Control it is also possible to include the position control loop to the AC500 PLC. In this case a speed reference signal will be transferred to the drive which makes it possible to perform the full range of motion functionalities with standard drives. To close the position control loop a feedback of the actual position is mandatory.

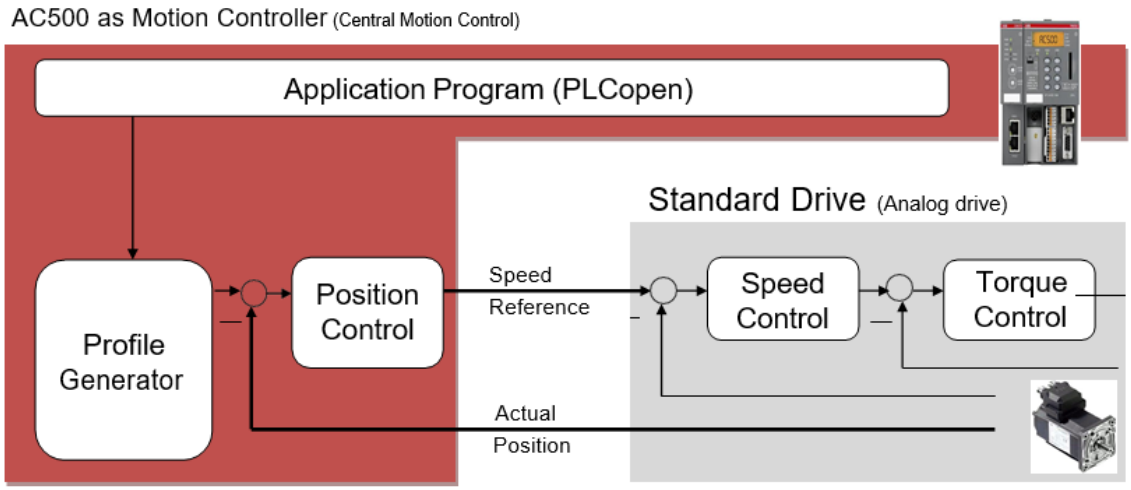


Fig. 201: Central Motion Control with AC500 PLC and PS552-MC, closed position control loop

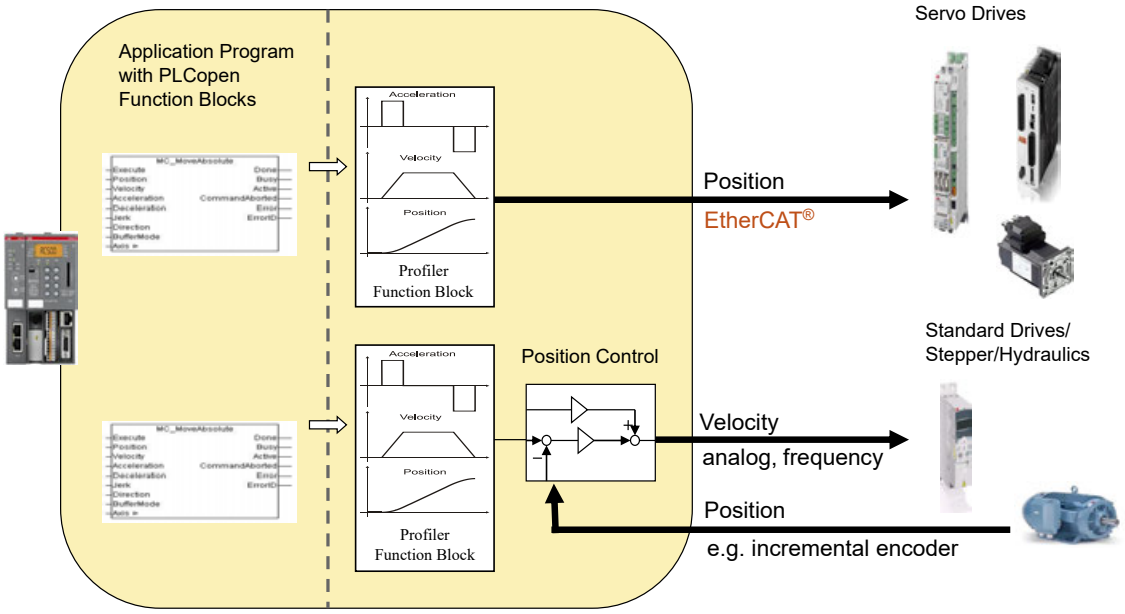


Fig. 202: PLC-based Motion Control with AC500 PLC and PS552-Motion, different axis implementations at the same time

1.5.9.2.3 Drive-Based Motion Control

For drive-based Motion Control the application program is done in the AC500 PLC with PLCopen function blocks included in PS552-MC. Based on this program commands and parameters will be transferred to the drive. Then the drive will act on its own during operation. Actual values and parameters can be read by the PLC. The commanded movement will be performed by the drive itself using its own profile generator for positioning drives or speed ramps in case of standard drives.

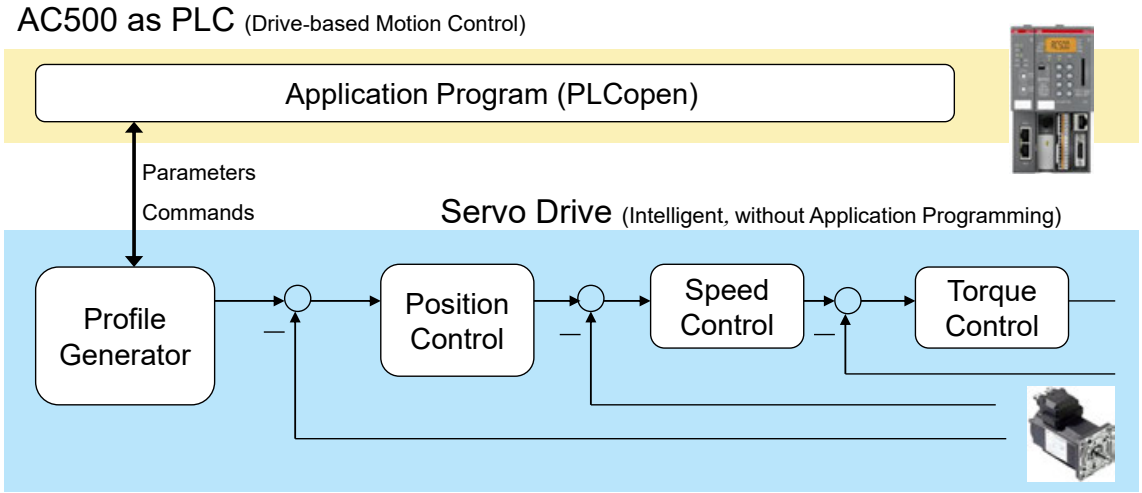


Fig. 203: System structure of drive-based Motion Control with AC500 PLC and PS552-MC

The available Motion Control functionalities depend on the used drive system.

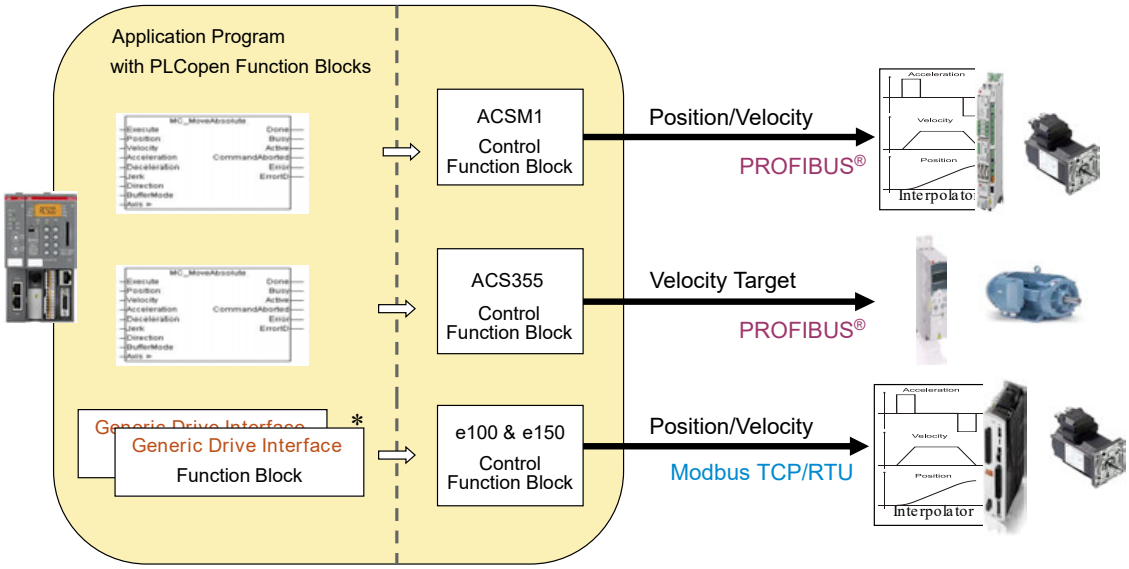


Fig. 204: Drive-based Motion Control with AC500 PLC and PS552-MC, different axis implementations at the same time

The generic drive interface is an implementation which is not part of PS552-MC, but is available as application note for MicroFlexTM e100 and MicroFlex e150 drives.

1.5.9.2.4 Overview of PLCopen function blocks

The following tables give an overview of the defined function blocks, divided into administrative (not driving motion) and motion related sets. They give an overview which function block could be used for the different possible configurations.

The function blocks are part of the library MC_Block_AC500_V11.

> *Function blocks for PLCopen Coordinated Motion are compabible with PLC-based central Motion Control only.* *Chapter 1.5.9.4.9 “PLCopen coordinated motion” on page 2679*

If there are restrictions concerning a certain drive ("XXX") which lead to a different or limited behavior compared to the standard the respective chapter is supplemented with an additional paragraph "Notes for XXX".

Table 155: Administrative function blocks

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MC_Power *Chapter 1.5.9.6.3.2 “MC_Power” on page 2835*	X	X	X	X	X	X
MC_ReadStatus *Chapter 1.5.9.6.3.3 “MC_ReadStatus” on page 2837*	X	X	X	X	X	X

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MC_ReadAxisError ↳ *Chapter 1.5.9.6.3.4 “MC_ReadAxisError” on page 2840*	-	-	X	X	X	-
MC_ReadParameter ↳ *Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*	X	X	X	X	X	-
MC_ReadBoolParameter ↳ *Chapter 1.5.9.6.3.7 “MC_ReadBoolParameter” on page 2846*	X	X	X	X	X	-
MC_WriteParameter ↳ *Chapter 1.5.9.6.3.8 “MC_WriteParameter” on page 2848*	X	X	X	X	X	-
MC_WriteBoolParameter ↳ *Chapter 1.5.9.6.3.9 “MC_WriteBoolParameter” on page 2850*	X	X	X	X	X	-
MC_Reset ↳ *Chapter 1.5.9.6.3.5 “MC_Reset” on page 2842*	X	X	X	X	X	X
MC_ReadActualPosition ↳ *Chapter 1.5.9.6.3.10 “MC_ReadActualPosition” on page 2852*	X	X	X	X	X	X
MC_ReadActualVelocity ↳ *Chapter 1.5.9.6.3.11 “MC_ReadActualVelocity” on page 2854*	X	X	X	X	X	-
MC_SetOverride ↳ *Chapter 1.5.9.6.3.12 “MC_SetOverride” on page 2856*	X	X	X	X	X	-
MC_SetPosition ↳ *Chapter 1.5.9.6.3.13 “MC_SetPosition” on page 2858*	X	X	-	-	-	X
MC_CamTableSelect ↳ *Chapter 1.5.9.6.3.1 “MC_CamTableSelect” on page 2832*	X	-	-	-	-	-

Table 156: Single-Axis function blocks

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MC_MoveAbsolute ↳ *Chapter 1.5.9.6.1.1 “MC_MoveAbsolute” on page 2747*	X	X	-	X	X	X
MC_MoveRelative ↳ *Chapter 1.5.9.6.1.2 “MC_MoveRelative” on page 2751*	X	X	-	X	X	X
MC_MoveAdditive ↳ *Chapter 1.5.9.6.1.3 “MC_MoveAdditive” on page 2756*	-	-	-	X	X	-
MC_MoveSuperimposed ↳ *Chapter 1.5.9.6.1.4 “MC_MoveSuperImposed” on page 2760*	X	X	-	-	-	-
MC_HaltSuperimposed ↳ *Chapter 1.5.9.6.1.5 “MC_HaltSuperimposed” on page 2764*	X	X	-	-	-	-
MC_MoveVelocity ↳ *Chapter 1.5.9.6.1.6 “MC_MoveVelocity” on page 2767*	X	X	X	X	X	X
MC_MoveContinuousAbsolute ↳ *Chapter 1.5.9.6.1.7 “MC_MoveContinuousAbsolute” on page 2771*	X	-	-	-	-	-
MC_MoveContinuousRelative ↳ *Chapter 1.5.9.6.1.8 “MC_MoveContinuousRelative” on page 2776*	X	-	-	-	-	-
MC_Stop ↳ *Chapter 1.5.9.6.1.9 “MC_Stop” on page 2781*	X	X	X	X	X	X
MC_PositionProfile ↳ *Chapter 1.5.9.6.1.11 “MC_PositionProfile” on page 2787*	X	-	-	-	-	-

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MC_VelocityProfile ↳ *Chapter 1.5.9.6.1.12 “MC_VelocityProfile” on page 2791*	X	-	-	-	-	-
MC_AccelerationProfile ↳ *Chapter 1.5.9.6.1.13 “MC_AccelerationProfile” on page 2794*	X	-	-	-	-	-
MC_Halt ↳ *Chapter 1.5.9.6.1.10 “MC_Halt” on page 2784*	X	X	X	X	X	-

Table 157: Multi-Axis function blocks

Function block	Central Motion Control (PLC-based)	
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT
MC_CamIn ↳ *Chapter 1.5.9.6.2.1 “MC_CamIn” on page 2799*	X	-
MC_CamOut ↳ *Chapter 1.5.9.6.2.2 “MC_CamOut” on page 2803*	X	-
MC_GearIn ↳ *Chapter 1.5.9.6.2.3 “MC_GearIn” on page 2805*	X	X
MC_GearInPos ↳ *Chapter 1.5.9.6.2.4 “MC_GearInPos” on page 2809*	X	X
MC_GearOut ↳ *Chapter 1.5.9.6.2.5 “MC_GearOut” on page 2814*	X	X
MC_PhasingAbsolute ↳ *Chapter 1.5.9.6.2.6 “MC_PhasingAbsolute” on page 2816*	X	-
MC_PhasingRelative ↳ *Chapter 1.5.9.6.2.7 “MC_PhasingRelative” on page 2821*	X	-
MC_CombineAxes ↳ *Chapter 1.5.9.6.2.8 “MC_CombineAxes” on page 2826*	X	X

Table 158: Homing function blocks

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MC_Home ↳ *Chapter 1.5.9.6.4.5 “MC_Home” on page 2876*	-	-	-	X	X	X
MC_StepAbsSwitch ↳ *Chapter 1.5.9.6.4.1 “MC_StepAbsSwitch” on page 2860*	X	X	-	-	-	-
MC_StepLimitSwitch ↳ *Chapter 1.5.9.6.4.3 “MC_StepLimitSwitch” on page 2867*	X	X	-	-	-	-
MC_StepRefPulse ↳ *Chapter 1.5.9.6.4.4 “MC_StepRefPulse” on page 2871*	X	X	-	-	-	-
MC_StepDirect ↳ *Chapter 1.5.9.6.4.2 “MC_StepDirect” on page 2865*	X	X	-	-	-	-

Homing with FM562 can be realized by the application program. An example is provided in http://www.abb.com/plc under “Application Examples" (select English Langauge for page!).

Table 159: ABB specific function blocks

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MCA_CAM_EXTRA ↳ *Chapter 1.5.9.6.5.1 “MCA_CAM_EXTRA” on page 2878*	X	-	-	-	-	-
MCA_Home ↳ *Chapter 1.5.9.6.5.6 “MCA_Home” on page 2887*	-	-	-	X	X	-
MCA_Indexing ↳ *Chapter 1.5.9.6.5.8 “MCA_Indexing” on page 2892*	X	X	-	X	X	-

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MCA_JogAxis *Chapter 1.5.9.6.5.9 "MCA_JogAxis" on page 2895*	X	X	X	X	X	-
MCA_MoveByExternalReference *Chapter 1.5.9.6.5.5 "MCA_MoveByExternalReference" on page 2885*	X	X	-	-	-	-
MCA_MoveVelocityContinuous *Chapter 1.5.9.6.5.4 "MCA_MoveVelocityContinuous" on page 2882*	X	X	-	-	-	X
MCA_Parameter *Chapter 1.5.9.6.5.2 "MCA_Parameter" on page 2879*	X	X	X	X	X	-
MCA_Power *Chapter 1.5.9.6.5.3 "MCA_Power" on page 2880*	X	X	X	X	X	-
MCA_ReadParameterList *Chapter 1.5.9.6.5.11 "MCA_ReadParameterList" on page 2900*	X	X	X	X	X	-
MCA_WriteParameterList *Chapter 1.5.9.6.5.10 "MCA_WriteParameterList" on page 2898*	X	X	X	X	X	-
MCA_SetPositionContinuous *Chapter 1.5.9.6.5.12 "MCA_SetPositionContinuous" on page 2903*	X	-	-	-	-	-
MCA_GearInDirect *Chapter 1.5.9.6.5.14 "MCA_GearInDirect" on page 2909*	X	-	-	-	-	-

Function block	Central Motion Control (PLC-based)		Decentralized Motion Control (drive-based)			
	CMC_MOTION_KERNEL_REAL	CMC_MOTION_KERNEL_INT	ACS35x	ACS800	ACSM1	FM562
MCA_CamInDirect ↳ *Chapter 1.5.9.6.5.15 "MCA_CamInDirect" on page 2913*	X	-	-	-	-	-
MCA_SetOperatingMode ↳ *Chapter 1.5.9.6.5.16 "MCA_SetOperatingMode" on page 2917*	X	-	-	-	-	-

1.5.9.2.5 Overview of libraries

▷ Add the following libraries for the listed applications.

⇨ In some cases by adding a library, there will be other libraries added automatically.

Application	Library to be added manually	Libraries added automatically
Central Motion Control	MC_Blocks_AC500_V11. lib	-
	CompactMotionControl_AC500_V12.lib	MC_Base_AC500_V11.lib
Central Motion Control, optional for EtherCAT	ECAT_AC500_APPL_V21.lib	-
Central Motion Control, optional auxiliary	MathFunctions_AC500_V23.lib	-
Central Motion Control - Coordinated Motion Control	MC_Blocks_AC500_V11.lib	-
	MC_CoBlocks_AC500_V11.lib	CompactMotionControl_AC500_V12.lib MC_Base_AC500_V11.lib CoordinatedMotion_AC500_V23.lib MathFunctions_AC500_V23.lib CMC_Ext_AC500_V23.lib
Central Motion Control - Coordinated Motion Control, optional for EtherCAT	ECAT_AC500_APPL_V21.lib	-
Central Motion Control - Coordinated Motion Control, optional auxiliary	CMC_Transformationen_AC500_V23.lib	-
Drive-based Motion Control, ACSM1 via PROFIBUS	MC_Blocks_AC500_V11.lib	-
	ACSM1_MC_support_AC500_V11.lib	ACSM1_AC500_V11.lib MC_Base_AC500_V11.lib
Drive-based Motion Control, ACS355 via PROFIBUS	MC_Blocks_AC500_V11.lib	-

Application	Library to be added manually	Libraries added automatically
	ACS350_MC_support_AC500_V11.lib	ACS350_AC500_V11.lib MC_Base_AC500_V11.lib
Drive-based Motion Control, FM562 via AC500 I/O bus	MC_Blocks_AC500_V11.lib	-
	MC_Base_AC500_V11.lib	-
	PTO_FM562_MC_support_V22.lib	-

The features of the function blocks provided with PS552-MC can be used from the PLC program according to PLCopen standard. Different drives and different Motion Control realizations could be used and can be combined with each other as well as different fieldbusses.

Table 160: Libraries for all types of Motion Control

Application	Library	Version	Drive	Fieldbus	Comment
Central Motion Control - Coordinated Motion or Drive based Motion Control	MC_Base_AC500_V11.lib	1.1	Any	Any	
	MC_Blocks_AC500_V11.lib	1.1	Any	Any	

Application	Library	Version	Drive	Fieldbus	Comment
Central Motion Control - Coordinated Motion	CMC_Transformationen_AC500_V23.lib	2.3	Any	Any	
	CoordinatedMotion_AC500_V23.lib	2.3	Any	Any	
	MC_CoBlocks_AC500_V23.lib	2.3	Any	Any	

Application	Library	Version	Drive	Fieldbus	Comment
Central Motion Control	CompactMotionControl_AC500_V12.lib	1.2	Any	Any	
	CMC_Ext_AC500_V23.lib	2.3	Any	Any	
	ECAT_AC500_APPL_V21.lib	2.3	Any	Any	
	MathFunctions_AC500_V23.lib	2.3	Any	Any	

Application	Library	Version	Drive	Fieldbus	Comment
Drive based Motion Control	ACSM1_AC500_V11.lib	1.1	ACSM1	PROFIBUS	
	ACSM1_MC_support_AC500_V11.lib	1.1	ACSM1	PROFIBUS	
	ACS350_AC500_V11.lib	1.1	ACS350, ACS355	PROFIBUS	
	ACS350_MC_support_AC500_V11.lib	1.1	ACS35x	PROFIBUS	Can also be used for ACS355
	ACS800_AC500_V11.lib	1.1	ACS800	PROFIBUS	
	ACS800_MC_support_AC500_V11.lib	1.1	ACS800	PROFIBUS	
	PTO_FM562_MC_support_V22.lib	2.2	FM562	AC500 I/O bus	

1.5.9.2.6 Overview of data types

The following data types are used for the PS552-MC Motion Control library. The data types are defined in the library file MC_BASE_AC500_V11. The corresponding elements can be used for the function blocks inputs.

Table 161: Structures

Data type	Elements	Element data type
CMC_AXIS_IO	limitSwitchPos	BOOL
	limitSwitchNeg	BOOL
	absRefSwitch	BOOL
MC_PPROFILE *Chapter 1.5.9.4.6.1 "PositionPositionProfile" on page 2653*	master_position	LREAL
	interpolation_point	LREAL
	velocity_ratio	LREAL
	acceleration_ratio	LREAL
MC_TPROFILE *Chapter 1.5.9.4.6.2 "PositionTimeProfile" on page 2654*	interpolation_point	LREAL
	first_derivative	LREAL
	second_derivative	LREAL
	delta_time	TIME

Table 162: Enum

Data type	Possible values
MC_ABB_iTYPES_ENUM *Chapter 1.5.9.4.6.3 “Interpolation types for profiles” on page 2654*	MCA_SPLINE_COMPLETE
	MCA_SPLINE_NATURAL
	MCA_POLY5
	MCA_POLY3
	MCA_LINEAR
MC_BUFFERMODE	mcABORTING
	mcBUFFERED
	mcBLENDINGlow
	mcBLENDINGprevious
	mcBLENDINGnext
	mcBLENDINGhigh
MC_DIRECTION	DEFAULT
	POSITIVE
	SHORTEST
	NEGATIVE
	CURRENT
MC_HOMING_DIRECTION	MC_SwitchNegative
	MC_SwitchPositive
	MC_Positive
	MC_Negative
MC_HOMING_EDGE	MC_EdgeOn
	MC_EdgeOff
	MC_On
	MC_Off
MC_HOMING_MODE	MC_REFPULSE
	MC_DIRECT
MC_SOURCE	mcActualValue
	mcSetValue

Data types of PLCopen Coordinated Motion Control: *Chapter 1.5.9.4.9.2.10 “ABB specific data structures” on page 2701*

1.5.9.2.7 Naming of function blocks and data structures

PLCopen

All function blocks and data types named MC_xxx are implemented according to PLCopen definition and follow the PLCopen documentation. They may have additional inputs but according to PLCopen rules.

All function blocks and data types named MCA_xxx are implemented corresponding to PLCopen rules with adaptations specific to AC500. They are AC500 specific extensions to the PLCopen library.

Central Motion Control

All function blocks named CMC_xxx belong to the implementation of Central Motion Control.

All data types named CMC_xxx belong to the implementation of Central Motion Control.

All data types named MC_xxx are implemented according PLCopen definition and follow the PLCopen documentation.

All data types named AXIS_xxx exist according to PLCopen definition. The content is ABB specific and not documented.

All function blocks named zCMC_xxx belong to the implementation of Central Motion Control. These are not documented and not intended for customer use.

Drive-based Motion Control

All function blocks starting with a name of a specific product (e.g. PTO_FM562_xxx, ACSM1_xxx, ACS800_xxx or ACS350_xxx) are intendend to be used with this product.

1.5.9.3 PLCopen

Based on application requirements and project specifications engineers are required to use or select a wide range of Motion Control hardware. In the past this required unique software to be created for each application even though the functions are the same. PLCopen motion standard provide a way to have standard application libraries that are reusable for multiple hardware platforms. This lowers development, maintenance and support costs while eliminating confusion. In addition, engineering becomes easier, training costs decrease, and the software is reusable across platforms. Effectively, this standardization is done by defining libraries of reusable components. In this way the programming is less hardware dependent, the reusability of the application software increased, the cost involved in training and support reduced, and the application becomes scalable across different control solutions. Due to the data hiding and encapsulation, it is usable on different architectures, for instance ranging from centralized to distributed or integrated to networked control. It is not specifically designed for one application, but will serve as a basic layer for ongoing definitions in different areas. As such it is open to existing and future technologies.

ABB is a member of the PLCopen organization. More Information about PLCopen can be read on the *PLCopen website*.

Fig. 205: PLCopen Motion Control logo

Function blocks according to PLCopen are designed for controlling axes via the language elements consistent with those defined in the IEC 61131-3 standard. It was decided by the task force that it would not be practical to encapsulate all the aspects of one axis into only one function block. The retained solution is to provide a set of command-oriented function blocks that have a reference to the axis, e.g. the abstract data type Axis, which offers flexibility, ease of use and reusability.

Implementations based on IEC 61131-3 (for instance via function blocks and SFC) will be focused towards the interface (look-and-feel/proxy) of the function blocks. This specification does not define the internal operation of the function blocks.

PLCopen Motion Control function blocks can be used in any IEC 61131-3 programming language. The following picture shows an example of a function block used in Function Block Diagram (FBD) language.

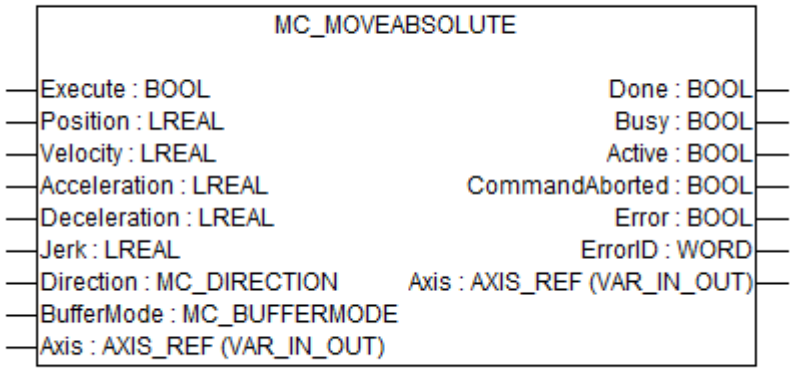


Fig. 206: Command for absolute positioning according to PLCopen standard

Application programs which use the manufacturer independent function blocks according to PLCopen will lead to the following advantages:

- Reusable software structure for different platforms.
- Programming based on function blocks.
- Function blocks can be used in any IEC 61131-3 language.

All function blocks which are defined by PLCopen will have the following qualities independently to the manufacturer of the motion control system:

- Same inputs/outputs
- Same functional behavior
- Same name

The following parts of the PLCopen motion control definition are completely or partly included in this product:

- Part 1: Function blocks for motion control
- Part 2: Extensions
- Part 3: User Guidelines
- Part 4: Coordinated Motion
- Part 5: Homing Procedures

1.5.9.3.1 Programming guidelines

This chapter explains some rules on the usage of the libraries and the structure Axis_Ref.

- In general, the kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.
- If Axis_Ref is used as input on a user defined function block or program or function, always use it as VAR_IN_OUT and never use it as VAR_INPUT or VAR_OUTPUT. The reason is that this would
  - Break the consistency and destroy data.
  - Consume a lot of computing power by copying data.
- Any instance of a function block should be called only once per cycle and in only one specific task.

  If the instance is used in several tasks, it has to be checked that is not called several times.

  Because this could corrupt the handshake from function block to Axis_Ref to CMC_MOTION_KERNEL_REAL and vice versa.

- Some PLCopen function blocks are only allowed to be called within the same task as the CMC_MOTION_KERNEL_REAL function block. This is mentioned in the function block descriptions.
- If PLCopen function blocks are called from a different task, they cycle time should be at least 2 times the cycle time for CMC_MOTION_KERNEL_REAL function block is.

Axis data type Axis_Ref

The Axis_Ref is a structure that contains information on the corresponding axis. It is used as a VAR_IN_OUT in all Motion Control function blocks defined in this document. The content of this structure is implementation dependent and can ultimately be empty. If there are elements in this structure, the supplier shall support the access to them, but this is outside of the scope of this document. The refresh rate of this structure is also implementation dependent. According to IEC 61131-3 it is allowed to switch the Axis_Ref for an active function block, for instance from Axis1 to Axis2. However, the behavior of this can vary across different platforms, and is not encouraged to do.

Axis_Ref data type declaration:

TYPE Axis_Ref : STRUCT

(Content is implementation dependent)

END_STRUCT

Example:

TYPE Axis_Ref : STRUCT

AxisNo: UINT; AxisName: STRING (255);

…….

END_STRUCT

1.5.9.3.2 The single axis state diagram

The following diagram normatively defines the behavior of the axis at a high level when multiple motion control function blocks are simultaneously activated. This combination of motion profiles is useful in building a more complicated profile or to handle exceptions within a program. (In real implementations there may be additional states at a lower level defined). The basic rule is that motion commands are always taken sequentially, even if the PLC had the capability of real parallel processing. These commands act on the axis' state diagram.

The axis is always in one of the defined states (see diagram below). Any motion command that causes a transition changes the state of the axis and, as a consequence, modifies the way the current motion is computed. The single axis state diagram is an abstraction layer of what the real state of the axis is, comparable to the image of the I/O points within a cyclic (PLC) program. A change of state is reflected immediately when issuing the corresponding motion command.

The response time of immediately is system dependent, coupled to the state of the axis, or an abstraction layer in the software.

The diagram is focused on a single axis. The multiple axis function blocks, MC_CamIn, MC_GearIn and MC_Phasing, can be looked at, from a single axis state diagram point of view, as multiple single-axes all in specific states. For instance, the CAM-master can be in the state Continuous Motion. The corresponding slave is in the state Synchronized Motion. Connecting a slave axis to a master axis has no influence on the master axis.

The state Disabled describes the initial state of the axis. In this state the movement of the axis is not influenced by the function blocks. The axis feedback is operational. If the MC_Power function block is called with Enable=TRUE while being in state Disabled, this either leads to Standstill if there is no error inside the axis, or to ErrorStop if an error exists.

Calling MC_Power with Enable=FALSE in any state, the axis goes to the state Disabled, either directly or via any other state. If a motion generating function block controls an axis, while the MC_Power function block with Enable=FALSE is called, the motion generating function block is aborted (CommandAborted).

The intention of the state ErrorStop is that the axis goes to a stop, if possible. There are no further inputs from function blocks accepted until a reset has been done from the ErrorStop state.

The transition Error refers to errors from the axis and axis control, and not from the function block instances. These axis errors may also be reflected in the output of the function blocks instances errors.

Issuing MC_Home in any other state than StandStill will go to ErrorStop, even if MC_Home is issued from the state Homing itself.

Function blocks which are not listed in the single axis state diagram do not affect the state of the axis, meaning that whenever they are called the state does not change. They are:

MC_ReadStatus; MC_ReadAxisError; MC_ReadParameter; MC_ReadBoolParameter; MC_WriteParameter; MC_WriteBoolParameter; MC_ReadActualPosition and MC_CamTable-Select.

Calling the function block MC_Stop in state StandStill changes the state to Stopping and back to Standstill when Execute = FALSE. The state Stopping is kept as long as the input Execute is TRUE. The output Done is set when the stop ramp is finished.

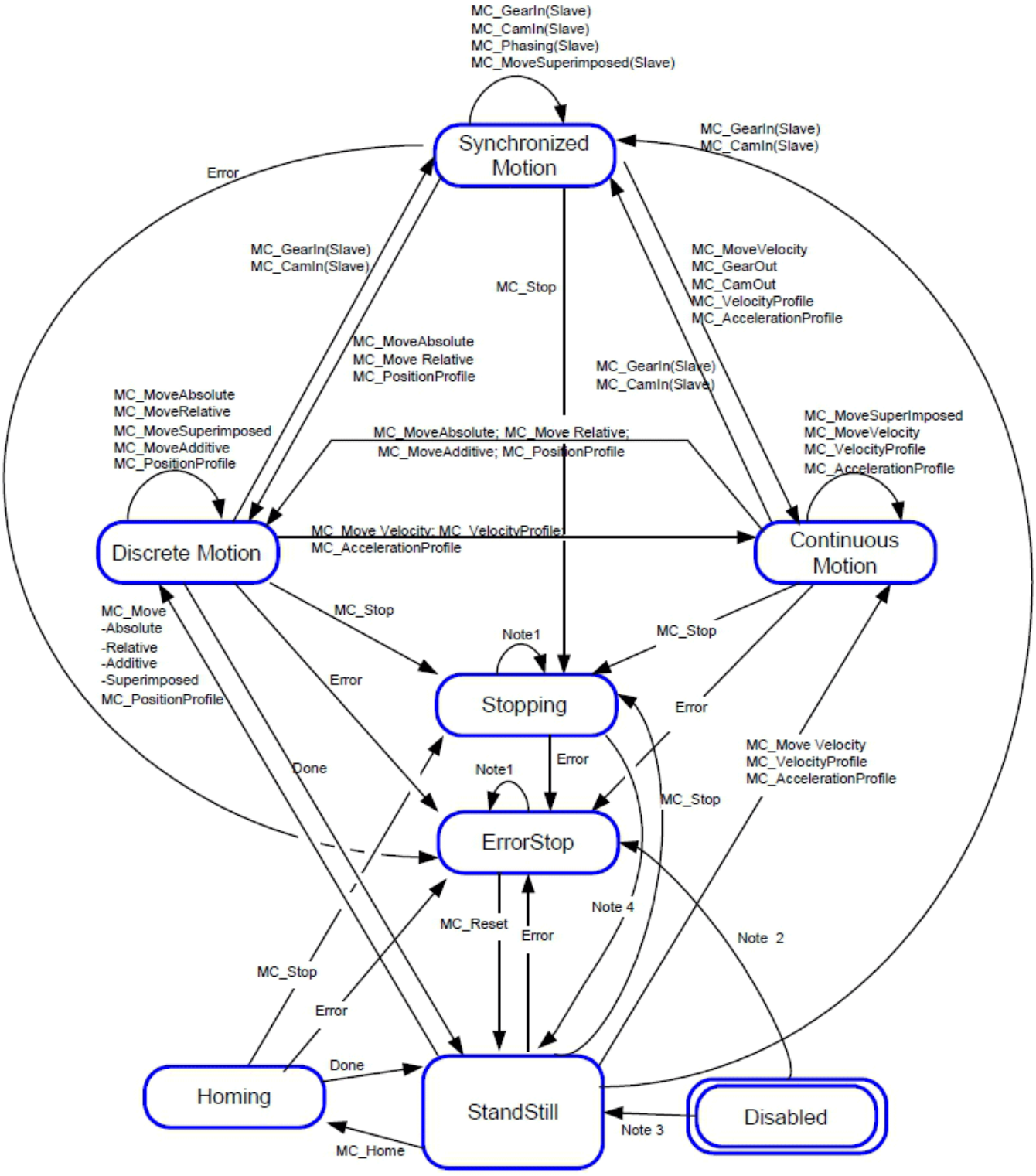


Fig. 207: Function block state behavior

1. In this state ErrorStop or Stopping, all function blocks can be called, although they will not be executed, except MC_Reset and Error – they will generate the transition to StandStill or ErrorStop respectively.

2. Power.Enable=TRUE and there is an error in the Axis.

3. Power.Enable=TRUE and there is no error in the Axis.

4. MC_Stop.Done AND NOT MC_Stop.Execute.

1.5.9.3.3 Visualizations

For usage with the PLCopen Library, a set of visualization objects is defined. These visualizations use the placeholder concept, which means that they could be used in an actual visualization several times and be instantiated by replacing the “placeholder” with an effective data-structure.

Two types of visualizations exist:

- As placeholder, an instance of Axis_Ref should be used. These are named: MC_VISU_Axis_name.
- As placeholder, an instance of the respective PLCopen function block should be used. These visualizations are named MC_VISU_FB_name where "name" could be MoveAbsolute or MoveVelocity, so the complete element is named MC_VISU_FB_MoveAbsolute or MC_VISU_FB_MoveVelocity.

The background colour and the colour for the title of each element could be changed. The colours are defined in some global predefined variables in MC_VISU_COLOR_INFORMATION. By changing these values, different colours will be used.

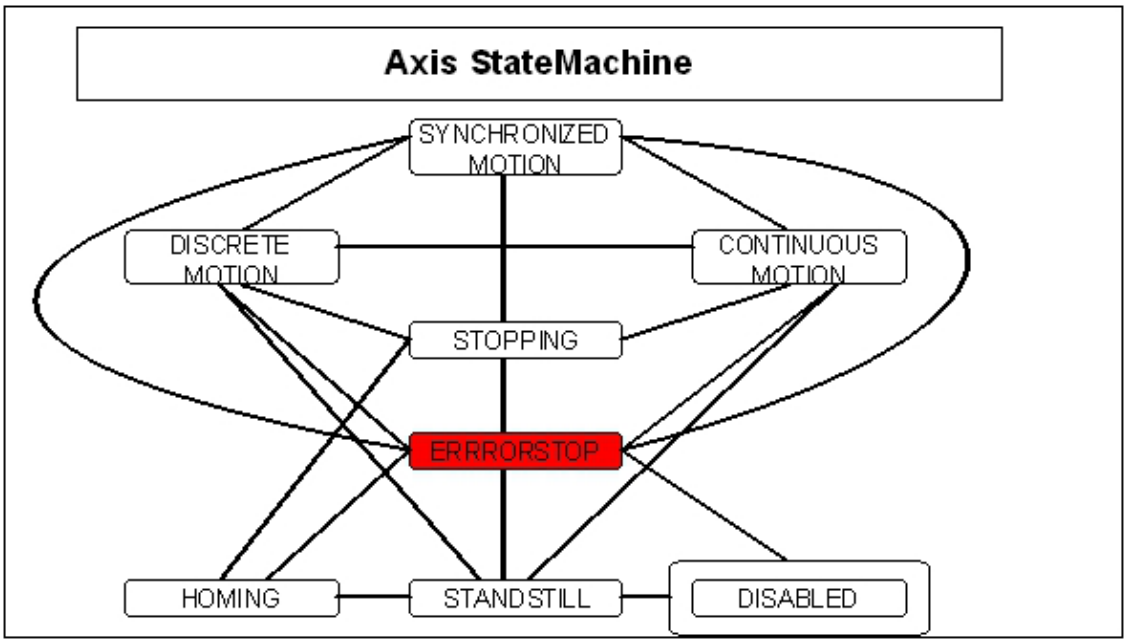
```
VAR_GLOBAL
    MC_VISU_BACKGROUND_COLOR:DWORD:=16#8080;
    MC_VISU_TITLE_COLOR:DWORD:=16#80FFFF;
END_VAR
```

Below, some existing visualizations are shown.

MC_VISU_Axis_StateMachine

This shows the state machine of the axis according to PLCopen definition. The active state is shown green except the ErrorStop which is shown red. Usually, it starts with Disabled. When no remote connection to the drive is available, it will switch to ErrorStop immediately.

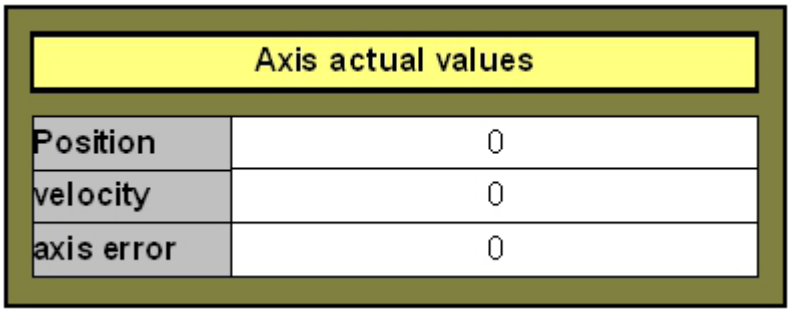
The placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



MC_VISU_Axis_actual

This object shows some actual values.

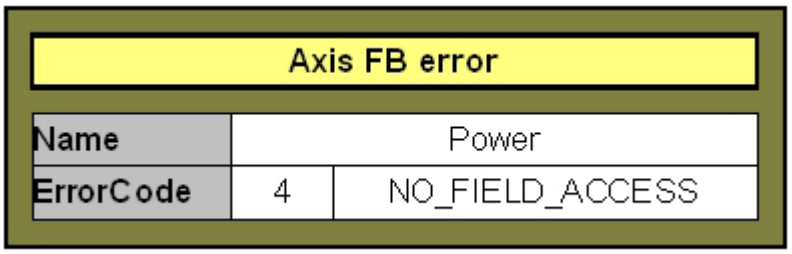
The Placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



MC_VISU_Axis_FB_error

This object shows the error information connected to the PLCopen function blocks. This is NOT a drive error. If no error occurs in the execution of a function block, just the name is shown. If an error occurred, it shows the name of the function block as well as the error number and a short description. In the example below, the MC_Power function block recognized that no fieldbus connection to the drive was available.

The Placeholder of this visualization has to be connected to an instance of the data type Axis_Ref.



1.5.9.3.4 Error codes

Besides the diagnosis information of the drive which is described in the respective drive documentation, there are a number of error codes directly related to the function blocks. These error codes are displayed at the output “ErrorID” of the function block.

Error Code	Mnemonic	Explanation
0	MC_Ok	No Error
1	WRONG_STATE	A function block was activated not according to the state machine, e.g. tried to start a movement while in state Disabled.
2	DRIVE_PROBLEM	The drive indicates an error, e.g. tripped.
3	PARAM-ETER_EXCEEDS_LIMIT	A parameter at the function block is outside the possible range. This does not refer to the parameter range which is allowed for the drive but just to the 32-Bit Integer which is used for internal calculation.
4	NO_FIELD_ACCESS	No fieldbus connection to the drive.
5	BUS_PROBLEM	Not used
6	ABS_SWITCH_ERROR	During Homing, (when done by function blocks) limit switch not according to moving direction e.g. the positive switch occurred when moving in negative direction.
7	TIMEOUT	Timeout in block execution.

Error Code	Mnemonic	Explanation
8	NAK	Parameter access not applicable
9	MC_TimeLimitExceeded	Used by function blocks with TimeLimit.
10	MC_DistanceLimitExceeded	Used by function blocks with DistanceLimit.
11	MC_TorqueLimitExceeded	Used by function blocks with TorqueLimit.
12	NOT_IMPLEMENTED	Functionality not implemented for certain axis type.

1.5.9.3.5 Error handling

All access to the drive/motion control is via function blocks. Internally these function blocks provide basic error checking on the input data. Exactly, how this is done is implementation dependent. For instance, if MaxVelocity is set to 6000, and the Velocity input to a function block is set to 10,000, a basic error report is generated. In the case where an intelligent drive is coupled via a network to the system, the MaxVelocity parameter is probably stored on the drive. The function block must take care of the errors generated by the drive internally. With another implementation, the MaxVelocity value could be stored locally. In this case the function block will generate the error locally.

Both centralized and decentralized error handling methods are possible when using the motion control function blocks.

Centralized error handling is used to simplify programming of the function block. Error reaction is the same independent of the instance in which the error has occurred.

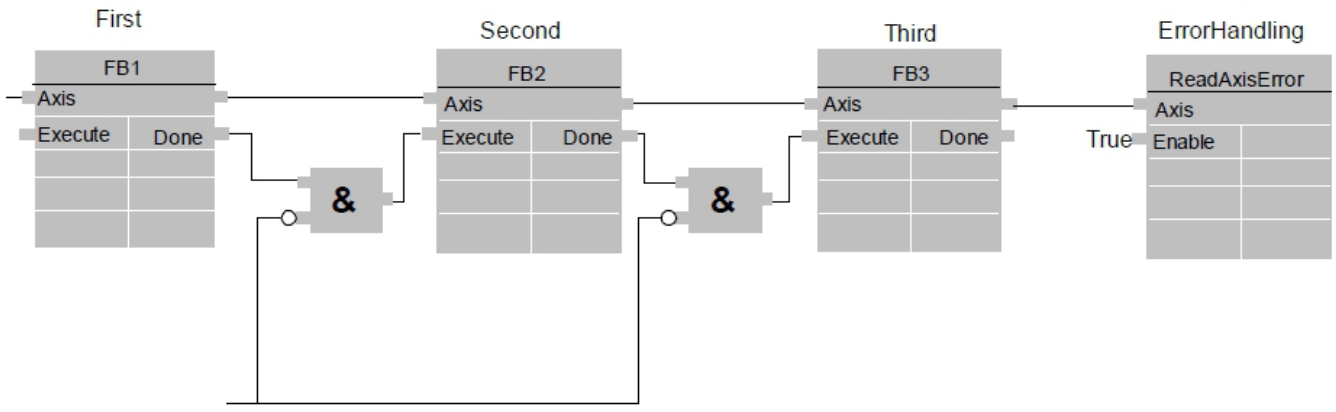


Fig. 208: Function blocks with centralized error handling

Decentralized error handling gives the possibility of different reactions depending on the function block in which an error occurred.

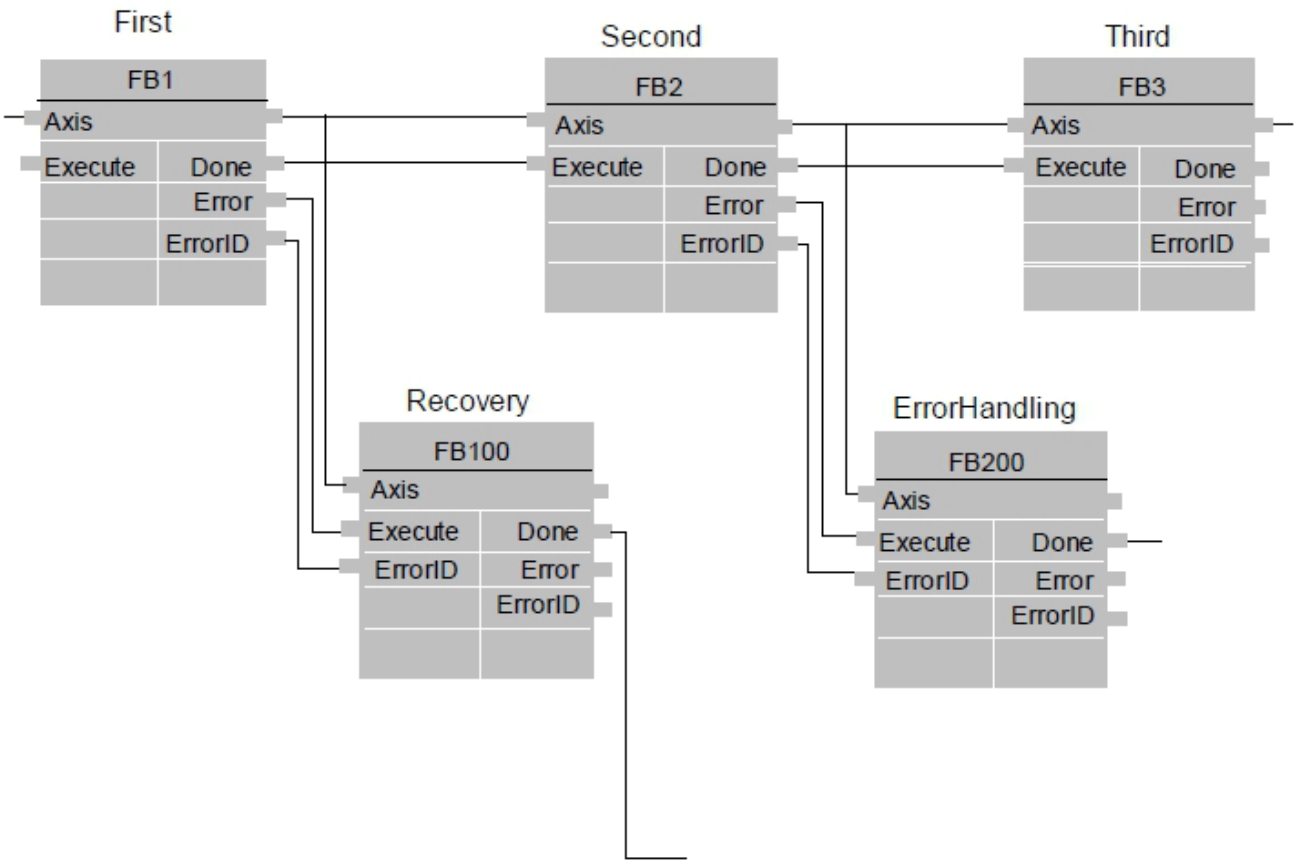


Fig. 209: function blocks with decentralized error handling

1.5.9.3.6 PLCopen parameter

Additional parameters are available by ReadParameter and WriteParameter function blocks.

Please see the following function blocks to read and write parameters:

- *MC_ReadParameter*
  Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844
- *MC_WriteParameter*
  Chapter 1.5.9.6.3.8 “MC_WriteParameter” on page 2848
- *MC_ReadBoolParameter*
  Chapter 1.5.9.6.3.7 “MC_ReadBoolParameter” on page 2846
- *MC_WriteBoolParameter*
  Chapter 1.5.9.6.3.9 “MC_WriteBoolParameter” on page 2850

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
1	CommandedPosition	DINT				R	Commanded position.
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive Software limit switch position.
3	SWLimitNeg	DINT	-2147483647	2147483647	-2147483647	R/W	Negative Software limit switch position.
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch.
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch.
6	EnablePosLagMonitoring	BOOL	FALSE	TRUE	TRUE	R/W	Enable monitoring of position lag (following error).
7	MaxPositionLag	DINT	1	2147483647 ***		R	Maximal position lag.
8	MaxVelocitySystem	DINT			32767	R	Maximal allowed velocity of the axis in the motion system.
9	MaxVelocityAppl	DINT	0**	32767	32767	R/W	Maximal allowed velocity of the axis in the application.
10	ActualVelocity	DINT	-32767	32767		R	Actual velocity.
11	CommandedVelocity	DINT	-32767	32767		R	Commanded velocity.
12	MaxAccelerationSystem	DINT			32767	R	Maximal allowed acceleration of the axis in the motion system.

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
13	MaxAccelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed acceleration of the axis in the application.
14	MaxDecelerationSystem	DINT			32767	R	Maximal allowed deceleration of the axis.
15	MaxDecelerationAppl	DINT	10	32767	32767	R/W	Maximal allowed deceleration of the axis.
16	MaxJerk	DINT	0*	2147483647	2147483647	R/W	Maximal allowed jerk of the axis.
2001	MODULO_NOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2002	MODULO_DENOMINATOR	DINT	1	2147483647	1	R/W	ABB specific parameter. Used for Central Motion Control implementation: Gearbox modifier to MODULO_RANGE
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate

Parameter number (PN)	Name	Datatype	Min.	Max.	Default	R/W	Comments
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
2005	EnableLimtVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGapLL	LREAL	0	214748364700	0	R/W	Used to stop the ongoing movement if position is behind

0* means: no limitation of jerk is performed.

**Axis will stay in stop.

***is modified by CMC_Axis_Control_Parameter, the max. Value is calculated in increments, the value which is delivered by ReadParameter will be given in [u].

1.5.9.3.7 Limits

Table 163: Limitations for the inputs of PLCopen function blocks when used with CMC_MOTION_KERNEL_INT

Parameter	Min.	Max.	Comment
Velocity	1	32767	-
Acceleration, Deceleration	1	32767	-

Parameter	Min.	Max.	Comment
Position	-2147483647	2147483647	-
INC_PER_R * U_PER__REV_DENOMINATOR /U_PER_REV_NOMINATOR	xxx	3276	The resolution could not be higher than 3276 INC/u.

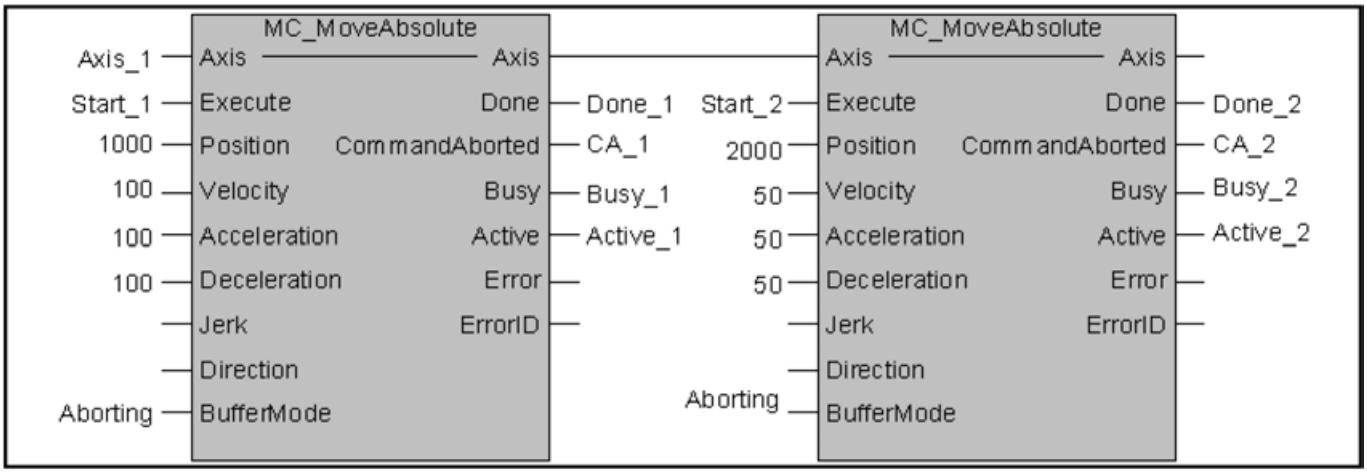
Table 164: Limitations for the inputs of PLCopen function blocks when used with CMC_MOTION_KERNEL_REAL

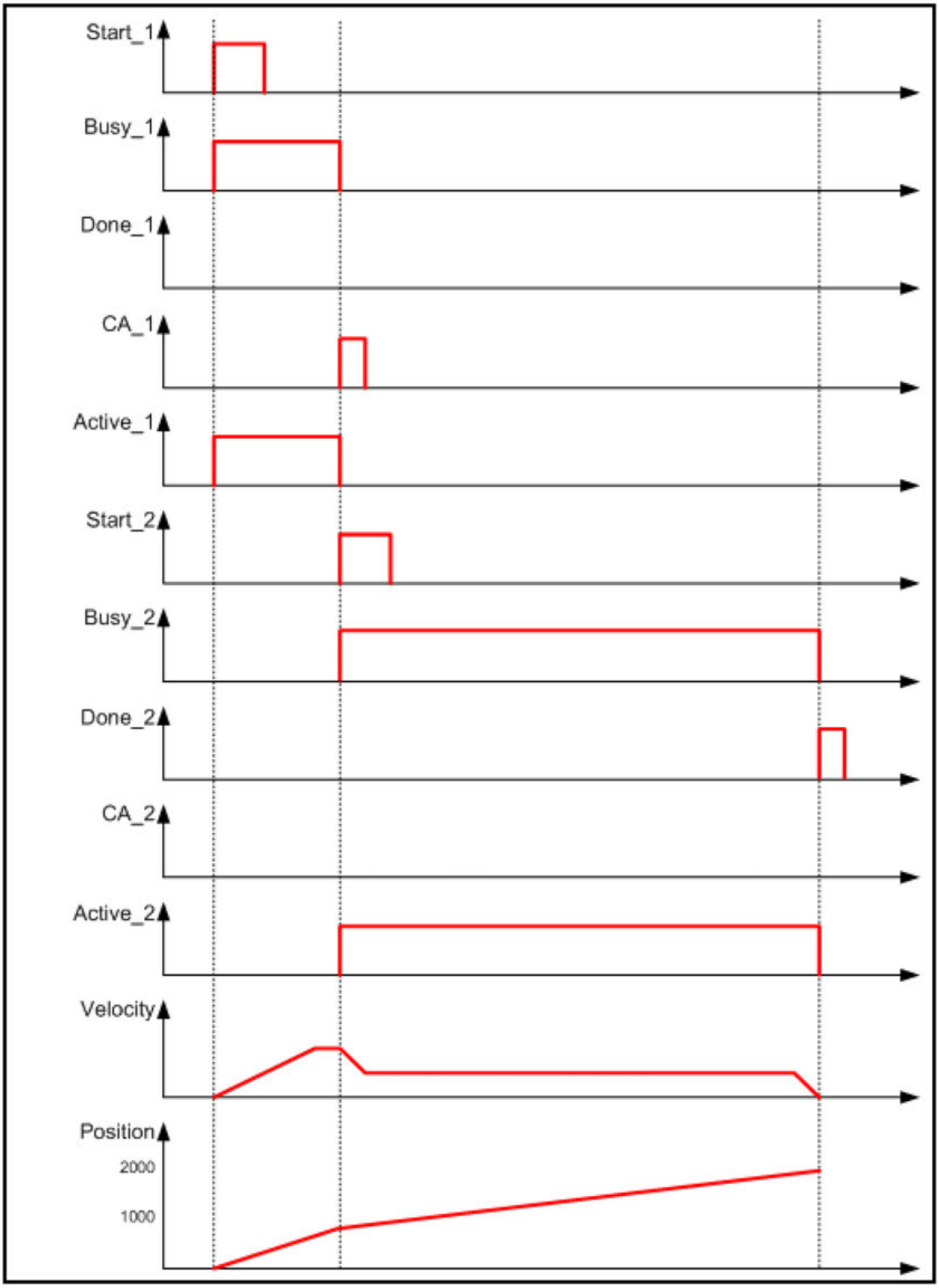
Parameter	Min.	Max.
Velocity	x	x
Acceleration, Deceleration	1	x
Position	-2147483647	2147483647

1.5.9.3.8 General restrictions

Restrictions for the available function blocks

- As buffered mode, MC_Aborting is realized as a default. This does NOT mean that the axis stops when another movement is started while an ongoing movement is still active. It means instead that the new movement will take control immediately and change the velocity to its own velocity by using its own acceleration or deceleration.
- The buffered mode MC_Buffered could be reached with using the axis state StandStill as enable signal for the Execute of the next block.
- From the Extended Inputs and Outputs at the function blocks, the following are not realized:
  - BufferedMode: The realization just supports the MC_Aborting mode.
  - The following Outputs at ReadStatus are not supported: ConstantVelocity, Accelerating and Decelerating.
  - TorqueLimit for Homing function blocks.




Start_1
Busy_1
Done_1
CA_1
Active_1
Start_2
Busy_2
Done_2
CA_2
Active_2
Velocity
Position
2000
1000

MC_Aborting Mode The diagram shows the behavior with BufferMode MC_Aborting, which is the only available BufferMode. When the second Block is activated, it will take control and will continue on its own velocity. The velocity is changed by using the acceleration value from the second function block. The movement will not be stopped in between. The first function block shows CommandAborted when the second function block is activated.

MC_Buffered A behavior according to BufferMode MC_Buffered could be reached by using the Done output from the first function block to enable the Execute of the second function block.

1.5.9.3.9 Behavior of the function block inputs and outputs

General rules

Table 165: General rules

Output exclusivity	The outputs Busy, Done, Error, and CommandAborted are mutually exclusive: Only one of them can be TRUE on one function block. If Execute is TRUE, one of these outputs has to be TRUE. Only one of the outputs Active, Error, Done and CommandAborted is set at the same time.
Output status	The outputs Done, InGear, InSync, InVelocity, Error, ErrorID and CommandAborted are reset with the falling edge of Execute. However, the falling edge of Execute does not stop or even influence the execution of the actual function block. It must be guaranteed that the corresponding outputs are set for at least one cycle if the situation occurs, even if execute was reset before the function block completed. If an instance of a function block receives a new execute before it has finished (as a series of commands on the same instance), the function block will not return any feedback, like Done or CommandAborted, for the previous action.
Input parameters	The parameters are used with the rising edge of the execute input. To modify any parameter, it is necessary to change the input parameter(s) and to trigger the motion again.
Missing input parameters	According to IEC 61131-3, if any parameter of a function block input is missing (open) then the value from the previous invocation of this instance will be used. In the first invocation the initial value is applied.
Position versus distance	Position is a value defined within a coordinate system. Distance is a relative measure related to technical units. Distance is the difference between two positions.
Sign rules	Velocity, Acceleration, Deceleration and Jerk are always positive values. Position and Distance can be both positive and negative.
Error Handling Behavior	All function blocks have two outputs, which deal with errors that can occur while executing that function block. These outputs are defined as follow:
	Error Rising edge of Error informs that an error occurred during the execution of the function block. ErrorID: Error number

	The outputs Done, InVelocity, InGear, and InSync mean successful completion so these signals are logically exclusive to Error. Types of errors: ● Function blocks (e.g. parameters out of range, state machine violation attempted), ● Communication, ● Drive Instance errors do not always result in an axis error (bringing the axis to StandStill). The error outputs of the relevant function block are reset with falling edge of Execute.
Function block naming	In case of multiple libraries within one system (to support multiple drive/ motion control systems), the function block naming may be changed to MC_FunctionBlockName_SupplierID.
Behavior of Done output	The outputs Done, InGear, InSync... are set when the commanded action has been completed successfully. With multiple function blocks working on the same axis in a sequence, the following applies: When one movement on an axis is interrupted with another movement on the same axis without having reached the final goal, Done of the first function block will not be set.
Behavior of CommandAborted output	CommandAborted is set, when a commanded motion is interrupted by another motion command. The reset-behavior of CommandAborted is like that of Done. When CommandAborted occurs, the other output-signals such as **InVelocity** are reset.
Inputs exceeding application limits	If a function block is commanded with parameters which result in a violation of application limits, the instance of the function block generates an error. The consequences of this error for the axis are application specific and thus should be handled by the application program.
Behavior of Busy output	Every function block can have an output Busy, reflecting that the function block is not finished. Busy is SET at the rising edge of Execute and RESET when one of the outputs Done, Aborted, or Error is set. It is recommended that this function block should be kept in the active loop of the application program for at least as long as Busy is true, because the outputs may still change. For one axis, several function blocks might be busy, but only one can be active at a time. Exceptions are MC_SuperImposed and MC_Phasing, where more than one function block related to one axis can be active.
Output **Active**	The output Active is required on buffered function blocks. This output is set at the moment the function block takes control of the motion of the according axis. For un-buffered mode the outputs Active and Busy can have the same value.
Enable and Valid/Status	The input Enable is coupled to output Valid. Enable is level sensitive, and Valid shows that a valid set of outputs is available at the function block. The output Valid is TRUE as long as an output value of Valid is available and the input Enable is TRUE. The relevant output value can be refreshed as long as the input Enable is TRUE. If there is a function block error, the output is not Valid (Valid set to FALSE). When the error condition disappears, the values will reappear and output Valid will be set again.

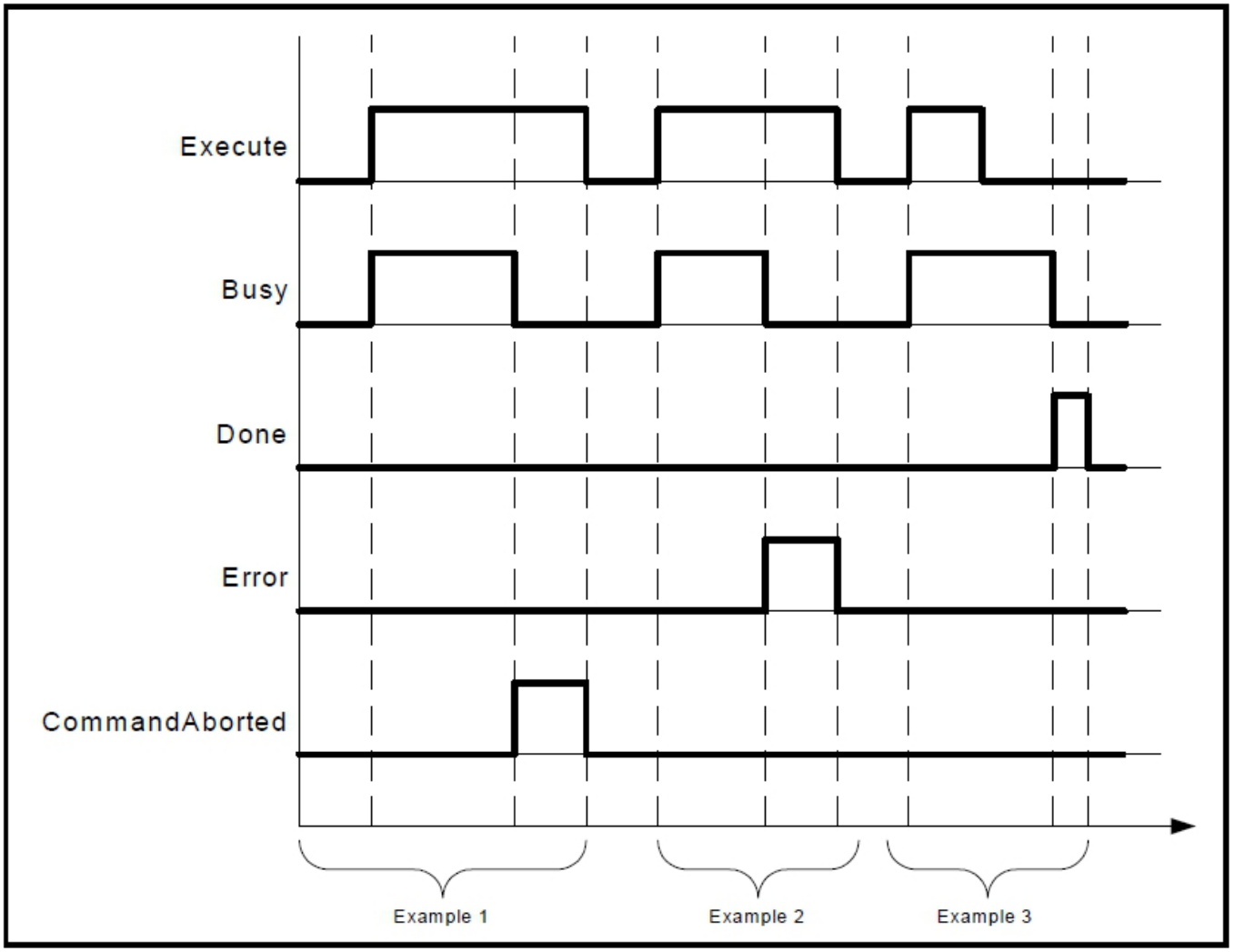


Fig. 210: Behavior of the Execute/Done style function blocks.

Why is the command input edge sensitive?

The input Execute for the different function blocks described in this document always triggers the function with its rising edge. The reason for this is that with edge triggered Execute new input values may be commanded during execution of a previous command. The advantage of this method is a precise management of the instant a motion command is performed. Combining different function blocks is then easier in both centralized and decentralized models of axis management. The output Done can be used to trigger the next part of the movement. The example given below is intended to explain the behavior of the function block execution.

The following figure illustrates the sequence of three function blocks First, Second and Third controlling the same axis. These three function blocks could be for instance various absolute or relative move commands. When First is completed the motion its rising output First.Done triggers Second.Execute. The output Second.Done AND In13 triggers the Third.Execute.

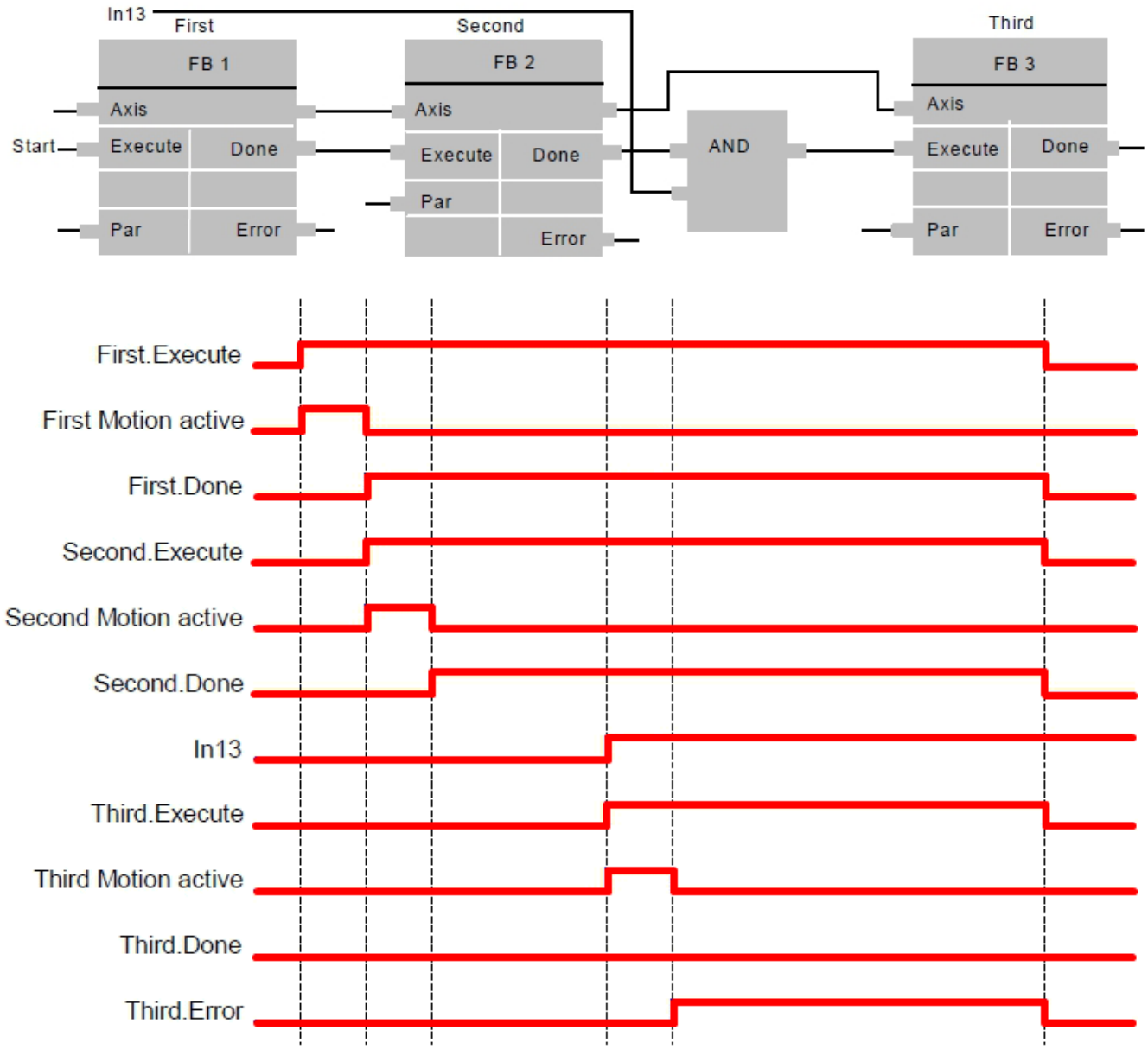


Fig. 211: Function blocks to perform a complex movement

The input ContinuousUpdate

Like described in the previous chapter, the input Execute triggers a new movement. With a rising edge of this input the values of the other function block inputs are defining the movement. Until version 1.1 of PLCopen there was the general rule that a later change in these input parameters does not affect the ongoing motion.

Nevertheless, there are numerous application examples, where a continuous change of the parameters is needed. The user could retrigger the input Execute of the function block, but this complicated the application.

Therefore, the input ContinuousUpdate has been introduced. It is an extended input to all applicable function blocks. If it is TRUE, when the function block is triggered (rising Execute), it will - as long as it stays TRUE – make the function block use the current values of the input variables and apply it to the ongoing movement. This does not influence the general behavior of the function block nor does it impact the single axis state diagram. In other words it only influences the ongoing movement and its impact ends as soon as the function block is no longer Busy or the input ContinuousUpdate is set to FALSE.

It can be that certain inputs like BufferMode are not really intended to change every cycle. However, this has to be dealt with in the application, and is not forbidden in the specification

If ContinuousUpdate is FALSE with the rising edge of the input Execute, a change in the input parameters is ignored during the whole movement and the original behavior of previous versions is applicable. The ContinuousUpdate is not a retriggering of the input Execute of the function block. A retriggering of a function block which was previously aborted, stopped, or completed, would regain control on the axis and modify its single axis state diagram. Opposite to this, the ContinuousUpdate only effects an ongoing movement. Also, a ContinuousUpdate of relative inputs (e.g. Distance in MC_MoveRelative) always refers to the initial condition (at rising edge of Execute).

Example

- MC_MoveContinuousRelative is started at Position 0 with Distance 100, Velocity 10 and ContinuousUpdate set TRUE. Execute is Set and so the movement is started to position 100.
- While the movement is executed (let the drive be at position 50), the input Distance is changed to 130, Velocity 20.
- The axis will accelerate (to the new Velocity 20) and stop at Position 130 and set the output Done and does not accept any new values.

1.5.9.3.10 Unit of length

The only specification for physical quantities is made on the unit of length (noted as [u]) that is to be coherent with its derivatives i.e. (velocity [u/s]; acceleration [u/s^2]; jerk [u/s^3]). Nevertheless, the unit [u] is not specified (manufacturer dependent). Only its relations with others are specified.

1.5.9.3.11 Aborting versus buffered modes

Some of the function blocks have an input called BufferMode. With this input, the function block can either work in a Non-buffered mode (default behavior) or in a Buffered mode. The difference between those modes is when they should start their action:

- A command in a non-buffered mode acts immediately, even if this interrupts another motion,
- A command in a buffered mode waits till the current function block sets its output Done (or InPosition, InVelocity...).
- The library just supports the mode "aborting" (MCAborting)

The following examples describe the different behavior of these modes:

Example 1: Standard behavior of two following absolute movements

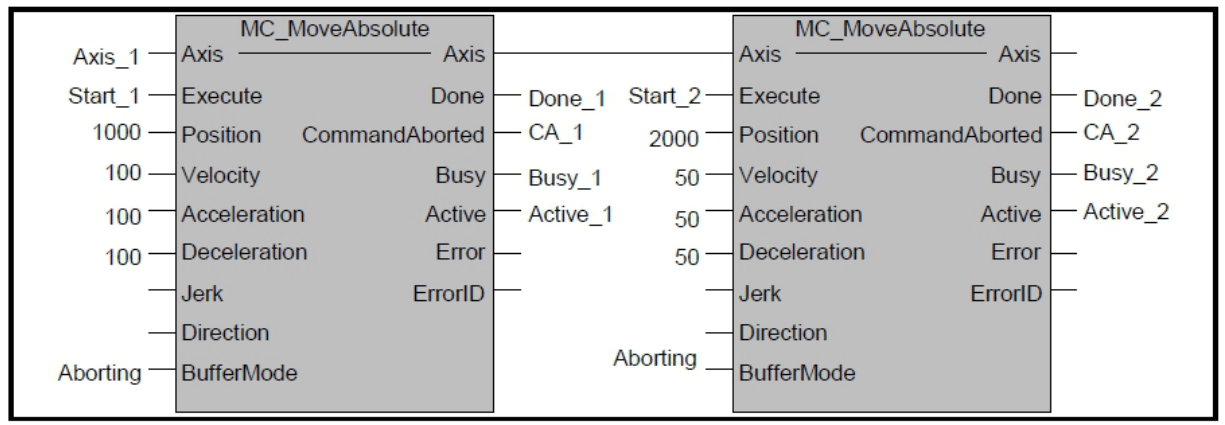


Fig. 212: Basic example with two MC_MoveAbsolute on same axis

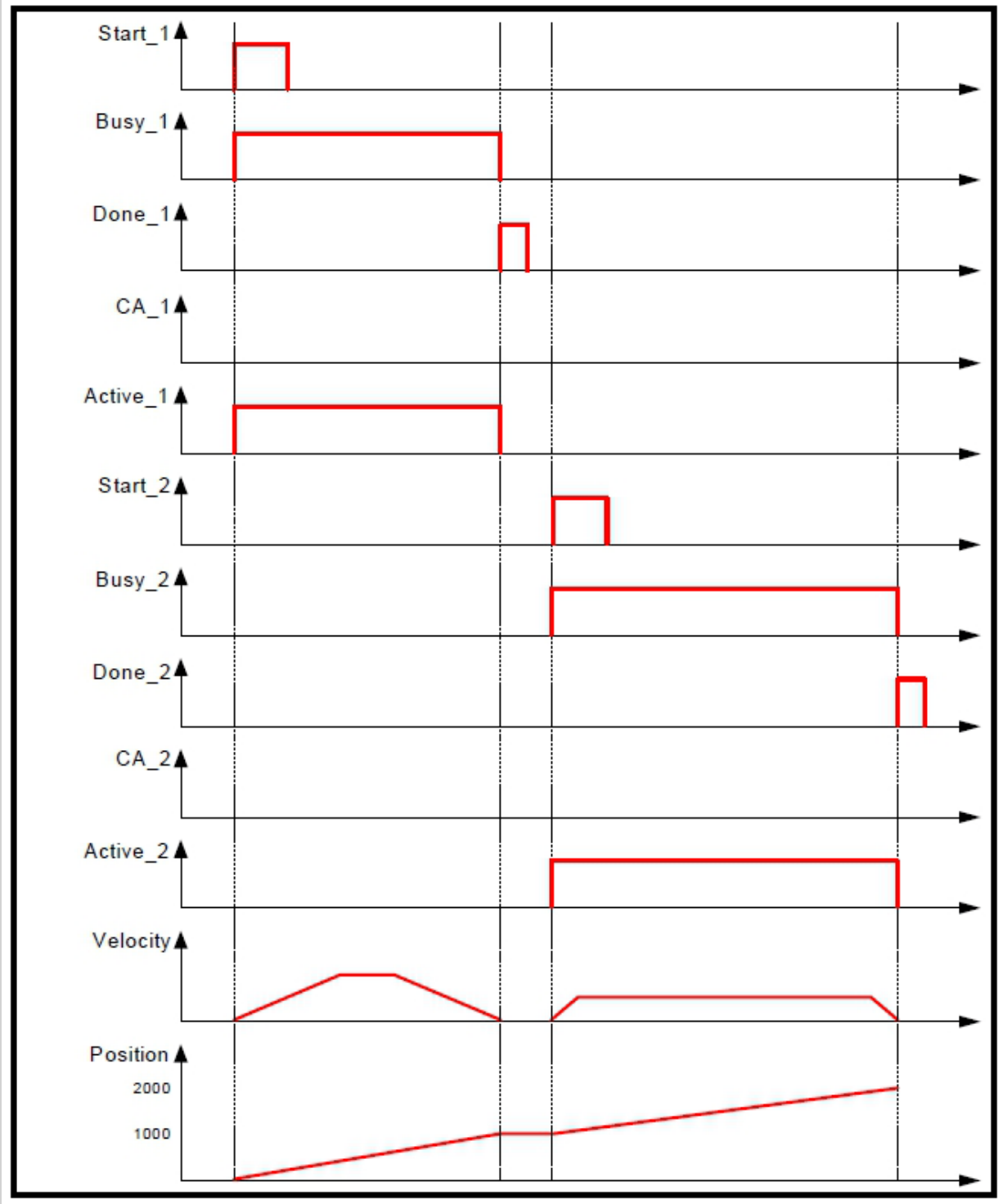
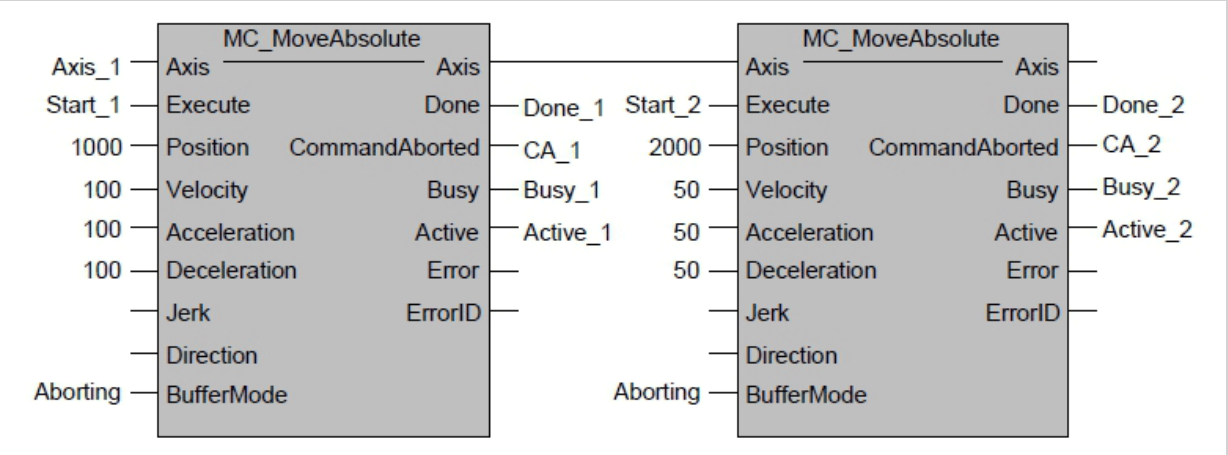


Fig. 213: Timing diagram for example above without interference between function block 1 and function block 2

Example 2: Aborting motion



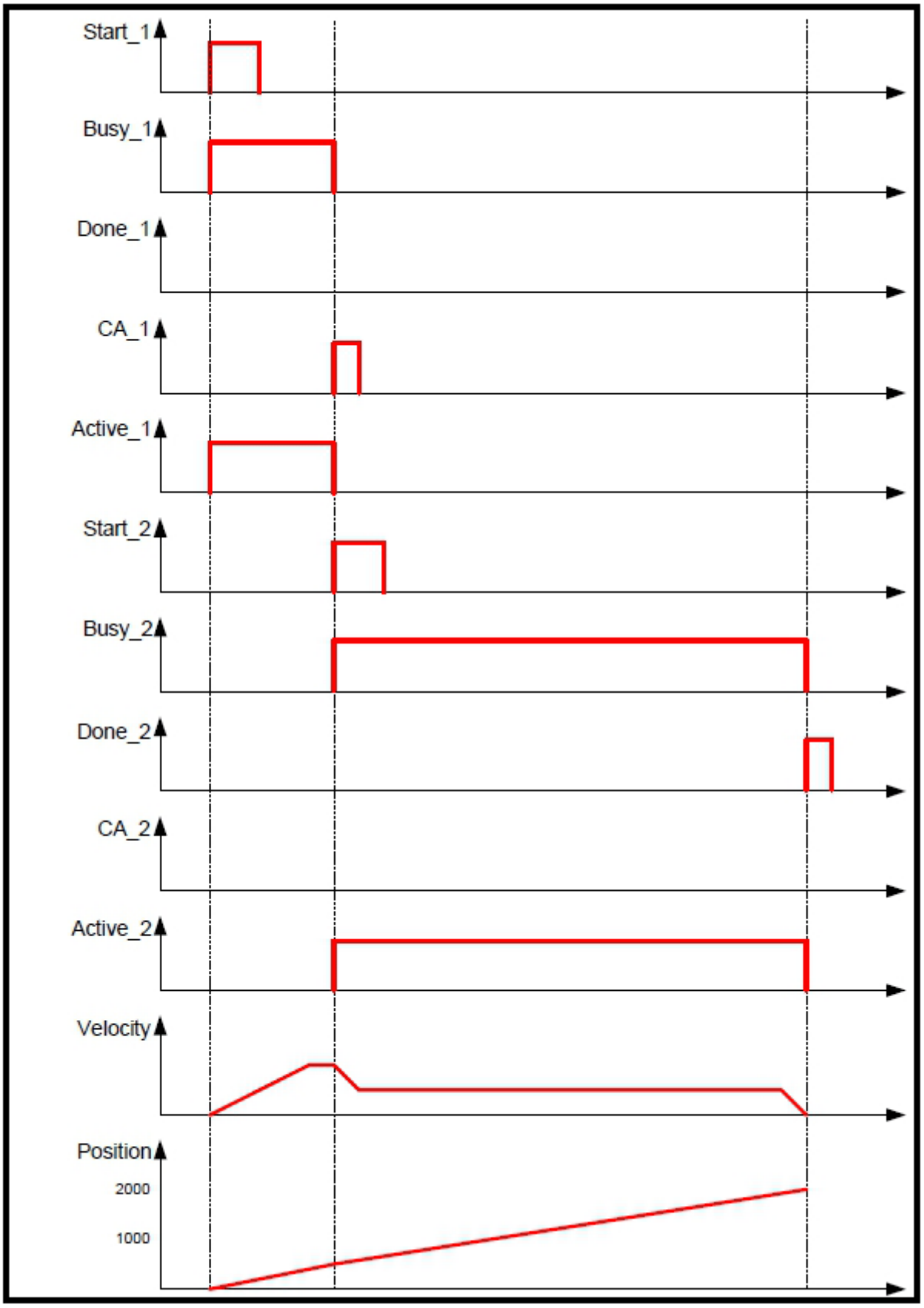


Fig. 214: Timing diagram for example above with function block 2 interrupting function block 1 (McAborting Mode)

If an on-going motion is aborted by another movement, it can occur that the braking distance is not sufficient due to deceleration limits.

In rotary axis, a modulo can be added. A modulo axis could go to the earliest repetition of the absolute position specified, in cases where the axis should not change direction and reverse to attain the target position.

In linear systems, the resulting overshoot can be resolved by reversing, as each position is unique and therefore there is no need to add a modulo to reach the correct position..

1.5.9.3.12 PLCopen examples

Example: A function block instance controls different motions of an axis

The following figure shows an example where the function block (MC_MoveVelocity) is used to control AxisX with three different values of Velocity. In a Sequential Function Chart (SFC) the velocity 10, 20, and 0 is assigned to V. To trigger the input Execute with a rising edge the variable E is stepwise set and reset.

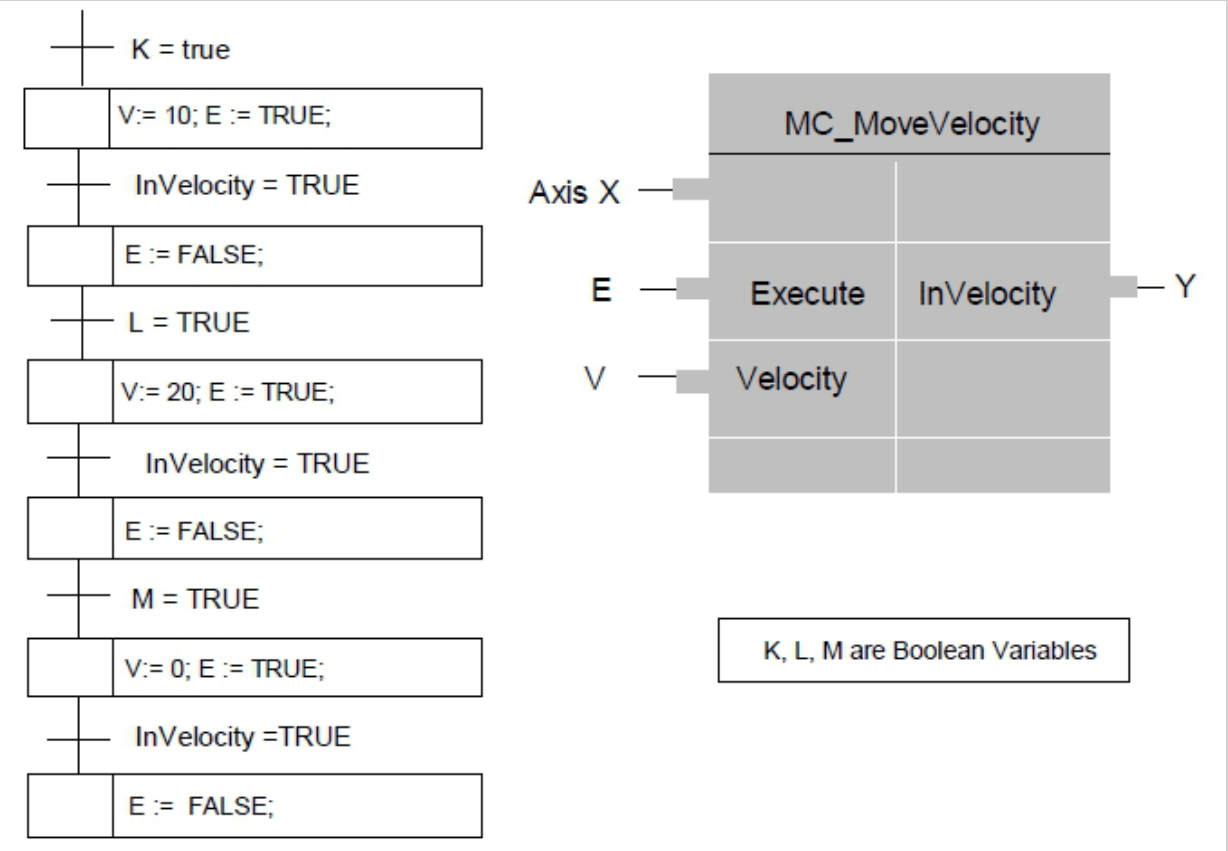


Fig. 215: Single function block with SFC

The following timing diagram explains how it works:

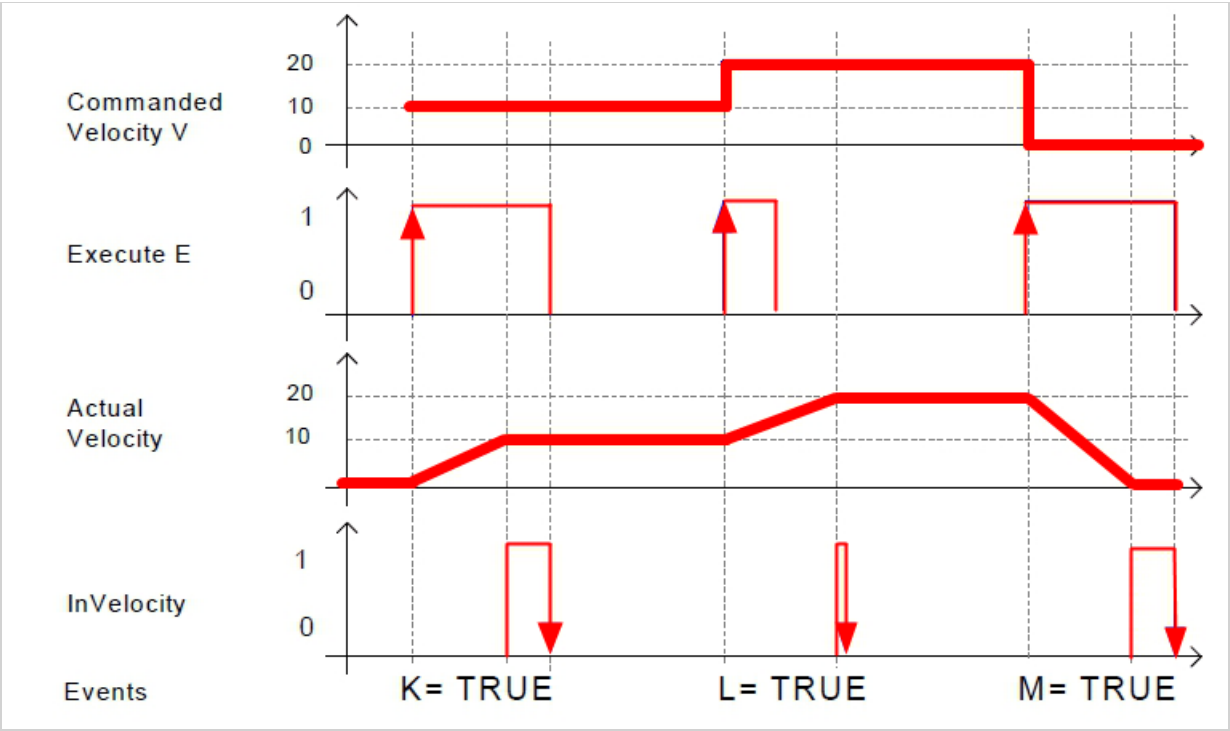


Fig. 216: Timing diagram for a usage of single function block

The second InVelocity is set for only one cycle because the Execute has gone low before the ActualVelocity equals CommandedVelocity.

Example: Different function blocks instances control the motions of an axis

Different instances related to the same axis can control the motions on an axis. Each instance will then be responsible for one part of the global profile.

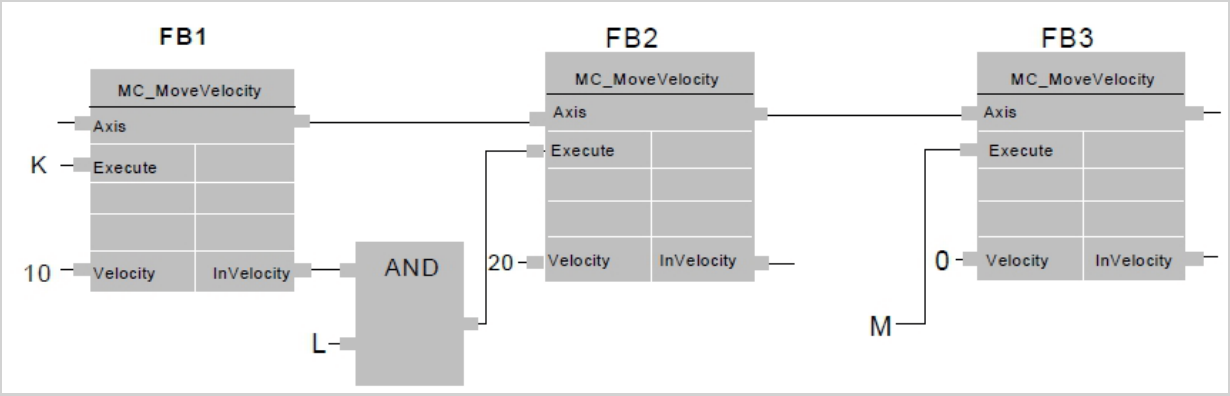


Fig. 217: Cascaded function blocks

The timing diagram:

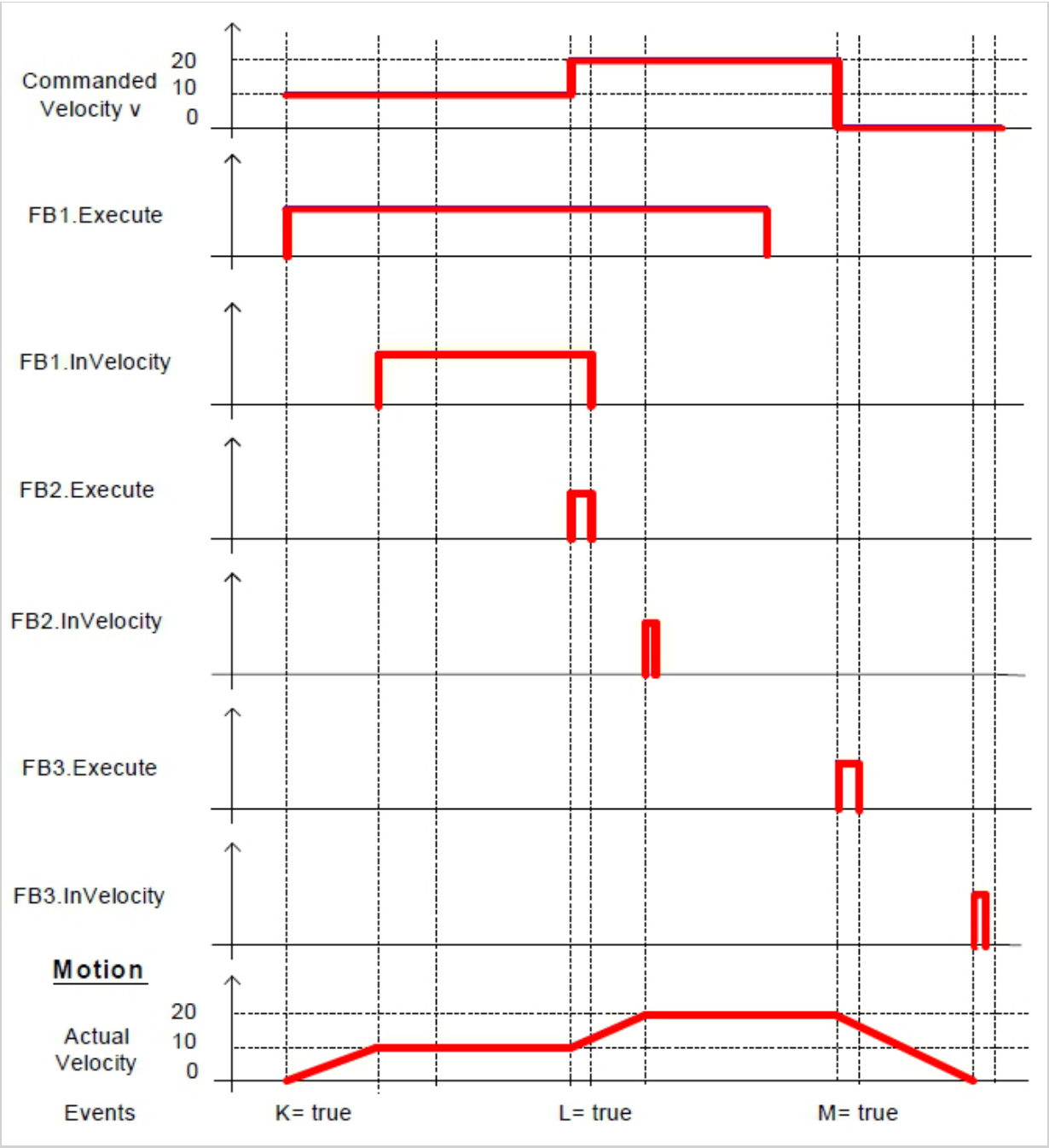


Fig. 218: Cascaded function blocks timing diagram

A corresponding solution written in LD looks like:

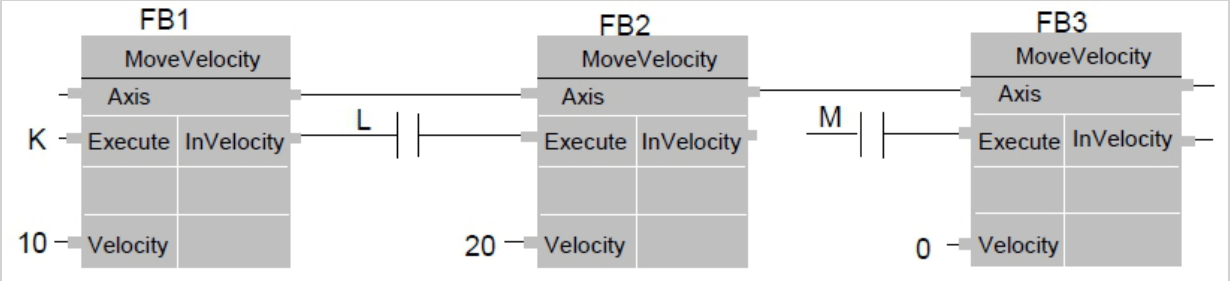


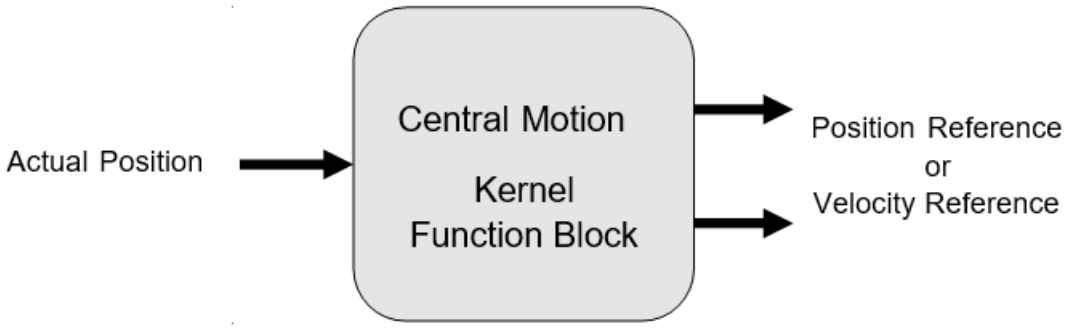
Fig. 219: Cascaded function blocks with LD

1.5.9.4 PLC-based motion control

1.5.9.4.1 Central Motion Control Architecture

The implementation of an axis for central Motion Control in the AC500 PLC is done by the use of the function blocks from the Compact Motion library file (CompactMotionControl_AC500_V12.lib).

With the function blocks of Compact Motion Control a Motion Control profiler can be used inside the PLC. As shown in the following figure it is needed to provide the actual position of the drive. The output can be either a position or a velocity reference signal. The used output signal will then be used to move the axis in the desired way.



There are 2 possibilities to send a reference value to the drive:

- When the position control loop is closed by the PLC by a CMC_MOTION_KERNEL_x function block, the output Speed_Reference should be connected to the drive. The value of Speed_Reference can be scaled with the axis parameters Max_Rpm and MAX_REFERENCE.
- When the position control loop is closed by the drive, the output Position_Reference should be connected to the drive. The unit for the output Position_Reference is incremented as well as the input Drive_ActualPosition.

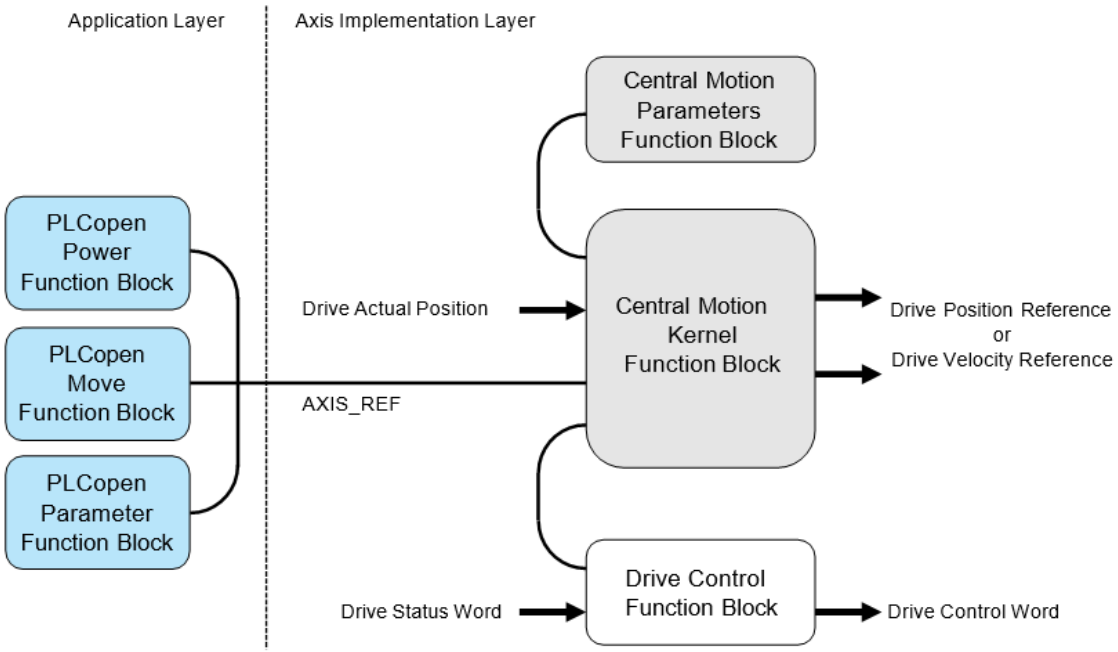


Fig. 220: Architecture for centralized motion control

In general the programming of a machine consists of two layers as shown in the figure above.

In the application layer function blocks according to PLCopen Motion Control are used to program the application sequences with all necessary types of movements and administrational commands. Due to the standard PLCopen Motion Control this can be reused in any other machine programs that used PLCopen function blocks.

The axis implementation layer is responsible for the execution of the commands from the application layer and can be programmed for each axis in a different way depending on the used hardware components.

Table 166: Needed function blocks for an application with central Motion Control

Library	Content
CompactMotionControl_AC500_V12.lib	Kernel function block, Parameters function block, Axis Simulation function block
MC_Base_AC500_V11.lib	Data types for AC500 Motion Control
MC_Blocks_AC500_V11.lib	Motion Control function blocks according to PLCopen

For a central motion axis implementation the use of the function blocks CMC_Motion_Kernel_Real and CMC_Axis_Control_Parameter_Real or CMC_Motion_Kernel_Int and CMC_Axis_Control_Parameter_Int is mandatory.

The Compact Motion function block design is independent from any bus architecture or any specific drive features.

Example for a possible system architecture

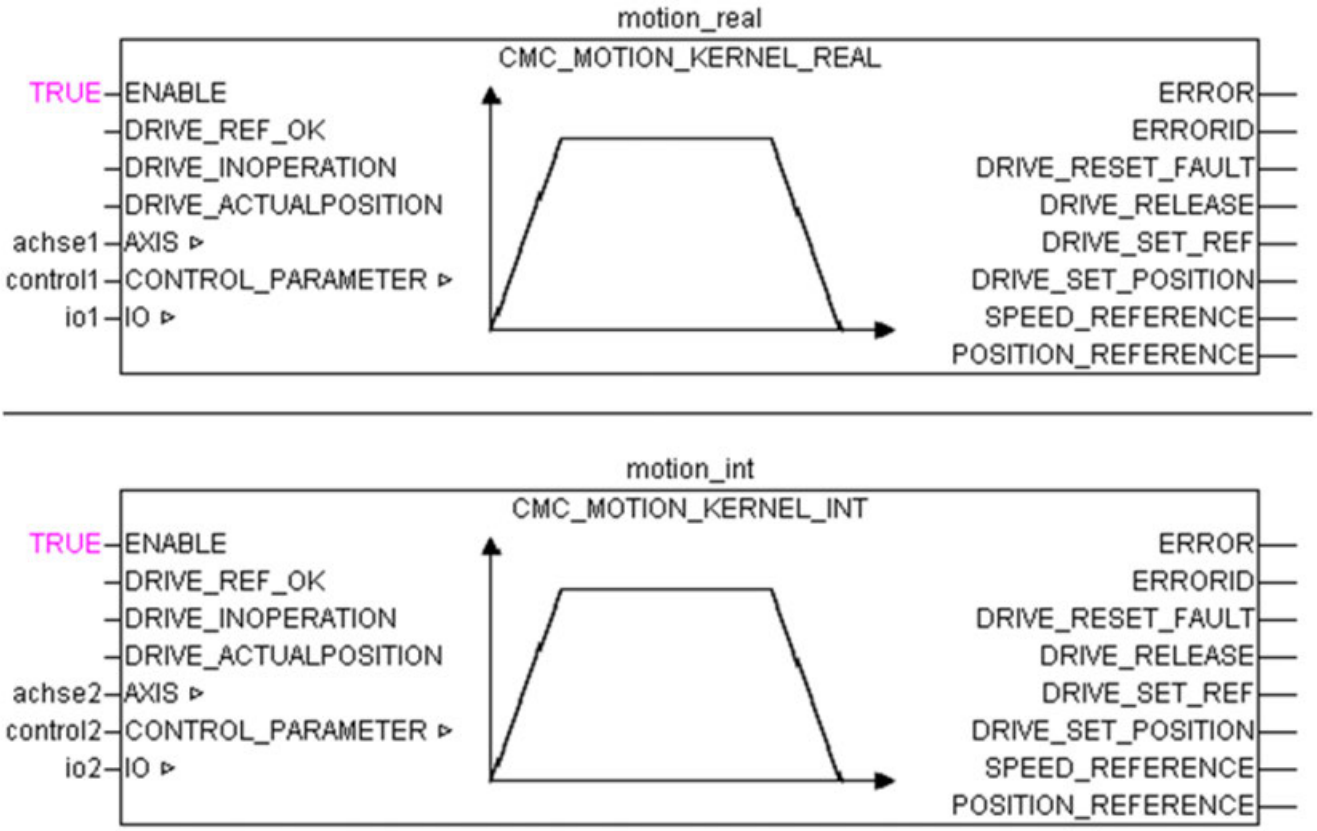
System	Velocity reference	Position feedback
System A	Output via analog output channel as voltage or current	From incremental encoder connected to CD522 I/O module
System B	Output via EtherCAT network	Input via EtherCAT network
System C	Output as frequency signal of CD522 I/O module	From incremental encoder connected to CD522 I/O module
System D	Output via PROFINET IO network	Input via PROFINET IO network

In case the velocity reference value is used from the kernel function block the position control loop is closed inside the drive. In this case, it is necessary to adjust the related parameters from the parameters function block. When the position reference will be used the position control loop is closed inside the drive. In this case, the internal control loop is just used to monitor the position and velocity.

When the position reference is used for the drive the following aspects have to be taken care of:

- *It is necessary to use a real time fieldbus, like EtherCAT.*
- *The PLC cycle has to be synchronized to the fieldbus cycle.*
- *The task calculation times may not exceed the used cycle time.*

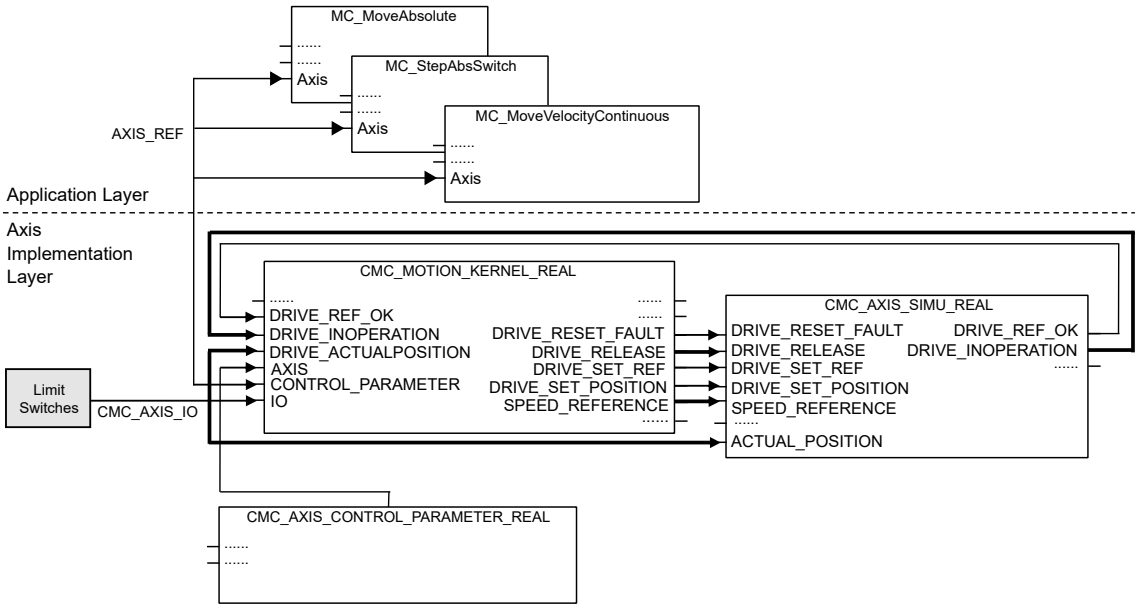
The drive's status should be managed by a specialized function block that supports the used type of drive as shown in the figure above. The kernel function block is the main function block which is needed to operate an axis with central Motion Control. It must be used with the parameter function block which is the interface to input parameters which are used to setup the axis.



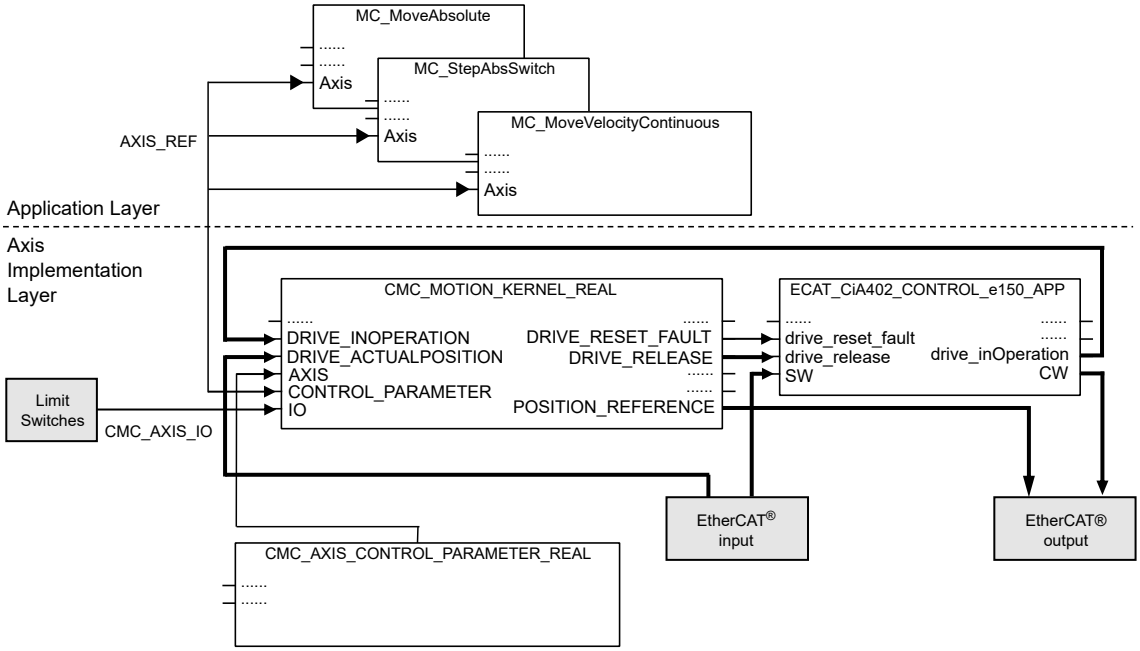
The corresponding function blocks for drive-based motion control are the function blocks <DriveName>_ACCESS.

The drive has to be accessed outside the CMC_MOTION_KERNEL function block. Actual values and reference values might be transferred by a synchronized fieldbus or by I/Os. The function block CMC_Motion_Kernel has to run inside the same task as the function blocks MC… and to be called every cycle and at least once before any function block MC… is activated.

The following figure shows an example with an axis simulation. The main data signals are drawn in bold lines. Here, the drive will receive a speed reference signal which means that the position control loop is closed inside the PLC by the Compact Motion function blocks. The time behavior of the simulated drive can be set by the parameter T1 at the axis simulation function block. If the time constant is to slow and the axis parameter Control_Time is too short the simulationed axis will run into instability – like a real drive. Sample values: *Chapter 1.5.9.4.2.3 “How to use the axis simulation” on page 2625*



The following figure shows an example with a MicroFlex e150 on an EtherCAT network. The main data signals are drawn in bold lines. Here, the drive will receive a position reference signal which means that the position control loop is closed inside the drive.



In the example the main signals are to be transferred via EtherCAT network. The drive control function block for the MicroFlex e150 can be found in the ECAT_AC500_APPL_V21 library.

Kernel function block

Kernel Arithmetic

The Compact Motion Control function blocks are available in two variants. Function blocks with the ending REAL use floating point arithmetic which results in a higher precision. These function blocks are intended to be used with PM59x PLCs.

Function blocks with the ending INT use integer arithmetic which uses less calculation time on PLCs without floating point CPUs. There are some restrictions connected with the use of these function blocks as shown in the following table. Also the use of some PLCopen function blocks will not be supported. ↯ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

Function blocks with the prefix CMC always have to be used with others of the same kind.
Function blocks with the ending REAL should not be compined with those with the ending INT.

Topic	CMC_Motion_Kernel_Int	CMC_Motion_Kernel_Real
Arithmetic	Integer arithmetic for interpolation and control loop.	Real arithmetic for interpolation and control loop.
Recommended PLC		Recommended to use in PLC with floating-point unit, e.g. PM59x.
Jerk	Acceleration and deceleration will be changed in a single step to the required value.	The jerk input of MC_xxx will be used in positioning, ● Jerk=0 "Acceleration and deceleration will be changed in a single step to the required value. ● Jerk >0 "Acceleration and deceleration will be changed regarding the maximum jerk. ● Every ramp will need a certain time longer in case jerk>0. This time is: (acceleration / jerk). ● The given acceleration at MC_xxx equates the maximum acceleration which is reached during the ramp.

How does the parameter for jerk influence the axis movements

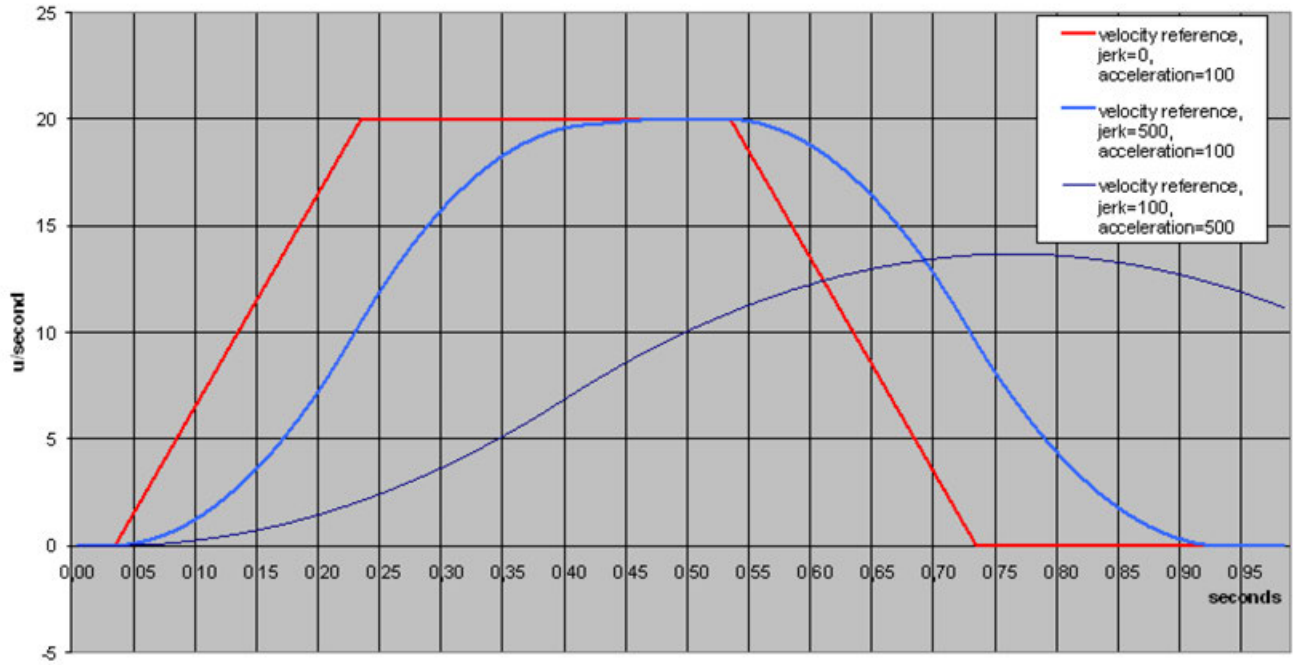


Fig. 221: Velocity reference with different jerk values

The diagram shows the result with different jerk values and the same velocity and acceleration.
The time needed for acceleration with jerk=0 is:

Time1=velocity/acceleration=(20/100)s=0.2s

The additional time with jerk=500 will be:

Time2=acceleration/jerk=(100/500)s = 0.2s

So the total time is:

Time=Time1 + Time2=0.2s + 0.2s=0.4s

In the last example with jerk=100, the velocity and acceleration values are not reached.

1.5.9.4.2 Basic functionalities

How to connect a drive

The connection to a drive must be done with the inputs and outputs of the function block CMC_Motion_Kernel_Real or CMC_Motion_Kernel_Int. All inputs and outputs of the kernel function block with the prefix “Drive_” are intended to be used with a drive, but in some cases not all of them are needed. In all cases the input Drive_ActualPosition has to be connected with the actual position of the axis. This value can be received by an I/O module of the PLC or via a fieldbus.

Depending on which device closes the position control loop either the output Speed_Reference or Position_Reference output has to be used. The value of Speed_Reference can be connected to an analog output module or be transferred via a fieldbus. The value of Position_Reference should be exclusively sent via a real-time fieldbus like EtherCAT.

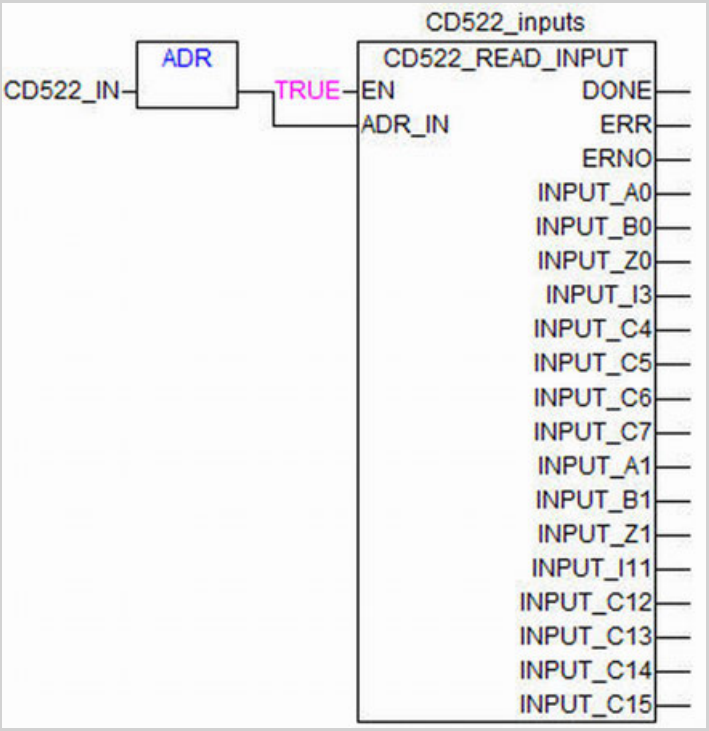
Example 1: Analog drive - Motor with incremental encoder

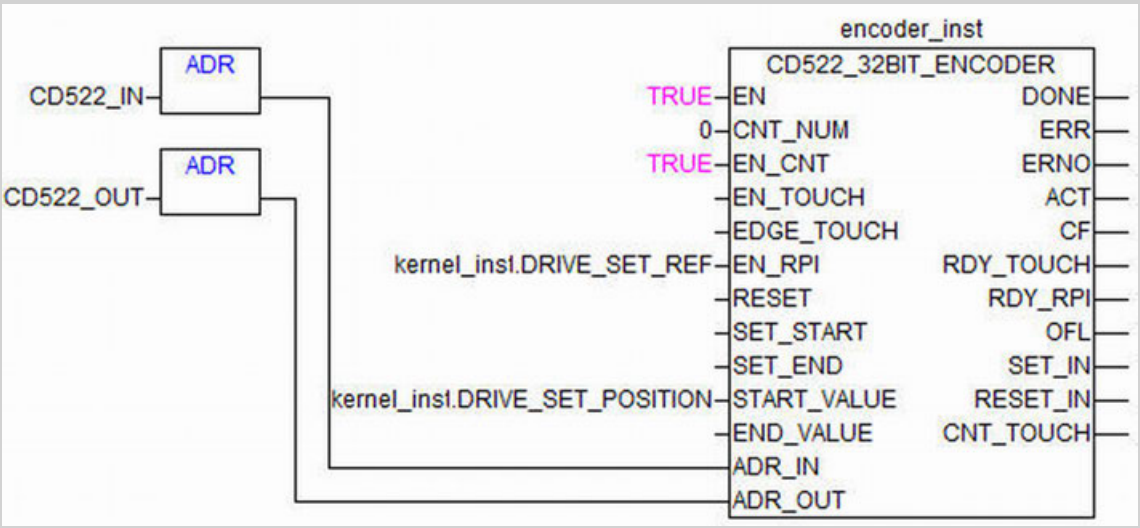
In the example the position control loop will be closed by the PLC, therefore the input Drive_ActualPosition and the output Speed_Reference are to be used.

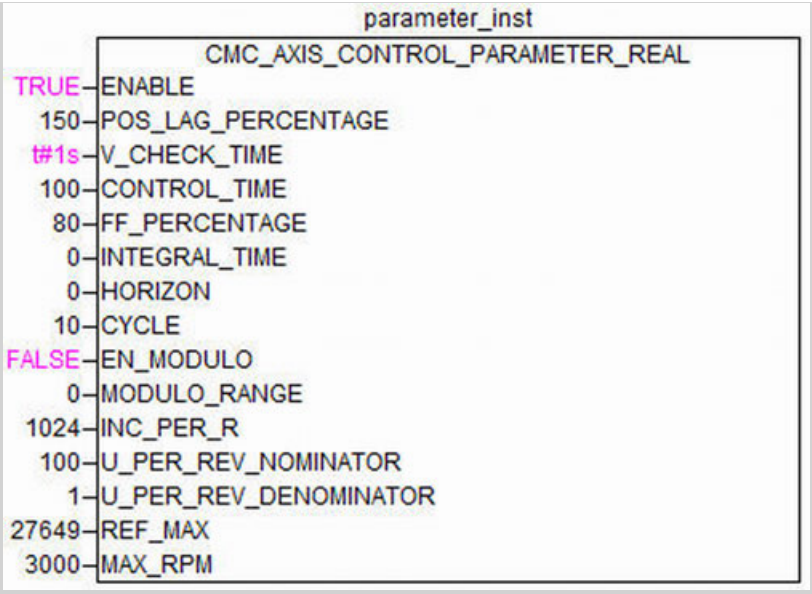
In combination with the I/O module CD522 and the corresponding function block CD522Encoder32Bit the position of the encoder can be used. For the effective resolution of the encoder parameter Inc_Per_R of the parameter function block has to be used.

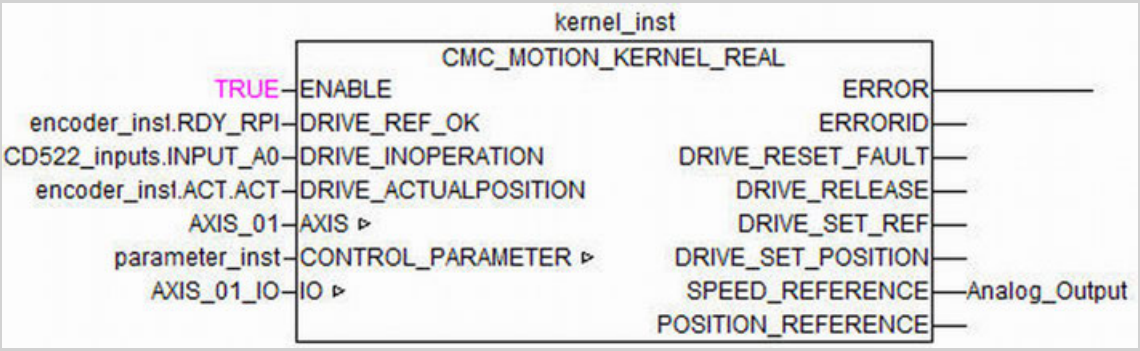
The output Speed_Reference can be written directly to the global variable of an output channel of an analog module but can also be transferred via a fieldbus. The scaling of this output value can be done with the parameters Ref_Max and Max_Rpm of the function block CMC_Axis_Control_Parameter_Real or CMC_Axis_Control_Parameter_Int.

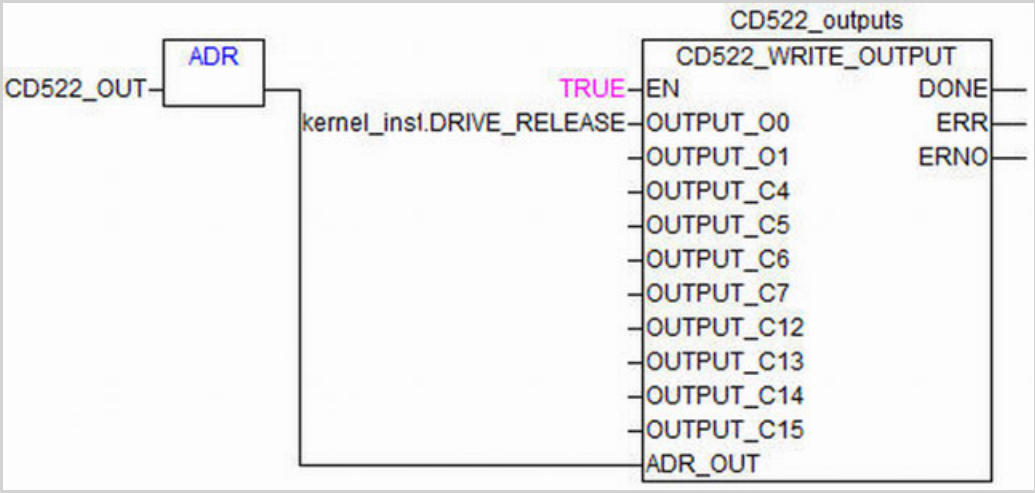
The scaling of the Speed_Reference value can be set with the inputs Ref_Max and Max_Rpm of the parameter function block.









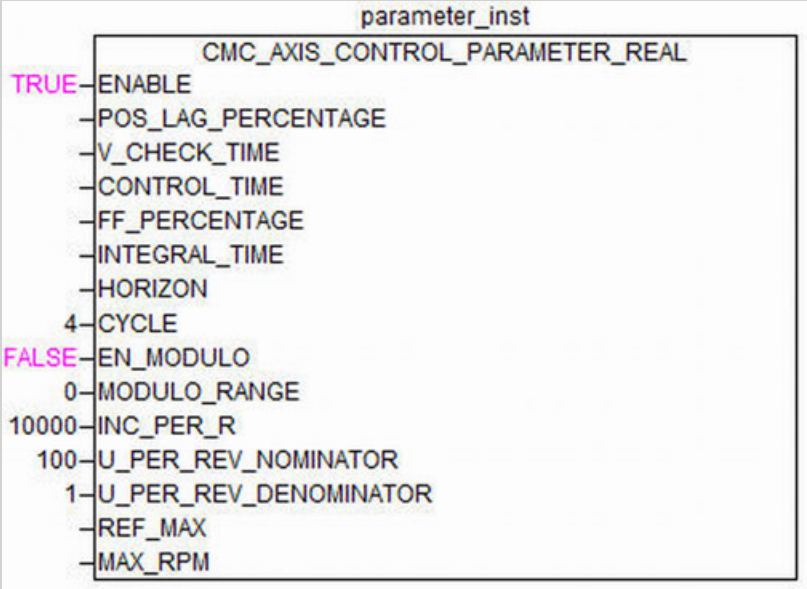


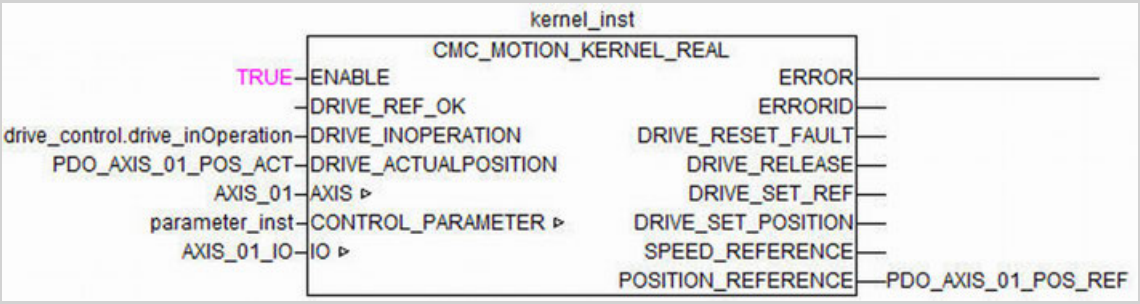
In order to finish a homing sequence which is done by the function block MC_StepRefPulse the outputs Drive_Set_Ref and Drive_Set_Position from the kernel function block have to be connected with the inputs EN_RPI and START_VALUE of the CD552 I/O module function block. Also the output RdyRpi of the CD552 I/O module function block has to be connected with Drive_Ref_Ok from the kernel function block.

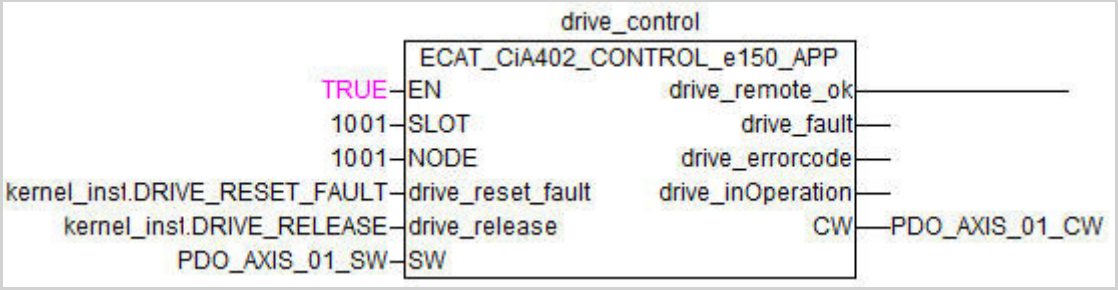
To enable and disable the drive Drive_Release could be connected to a binary output to activate the drive. Drive_InOperation could be connected to a binary input to get the information that Drive_Release was successful.

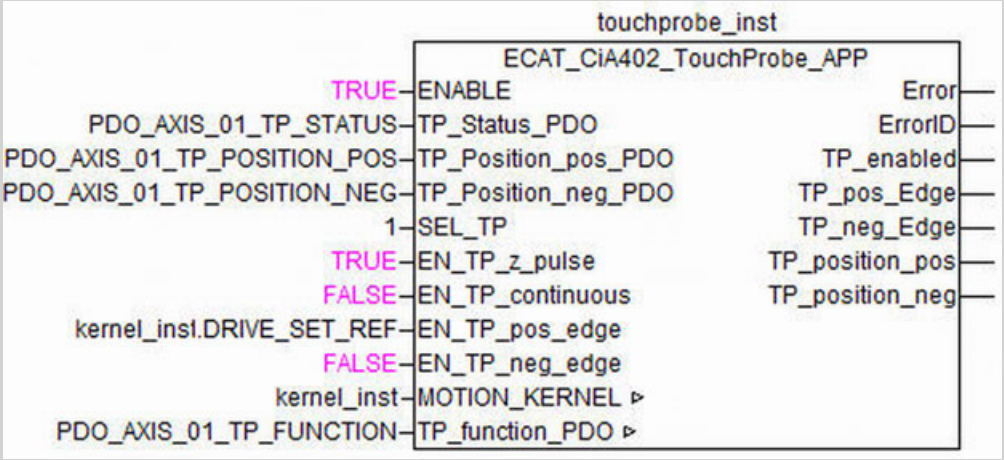
Example 2: Servo Drive - Microflex e150 via EtherCAT in contiuous positioning mode (csp)

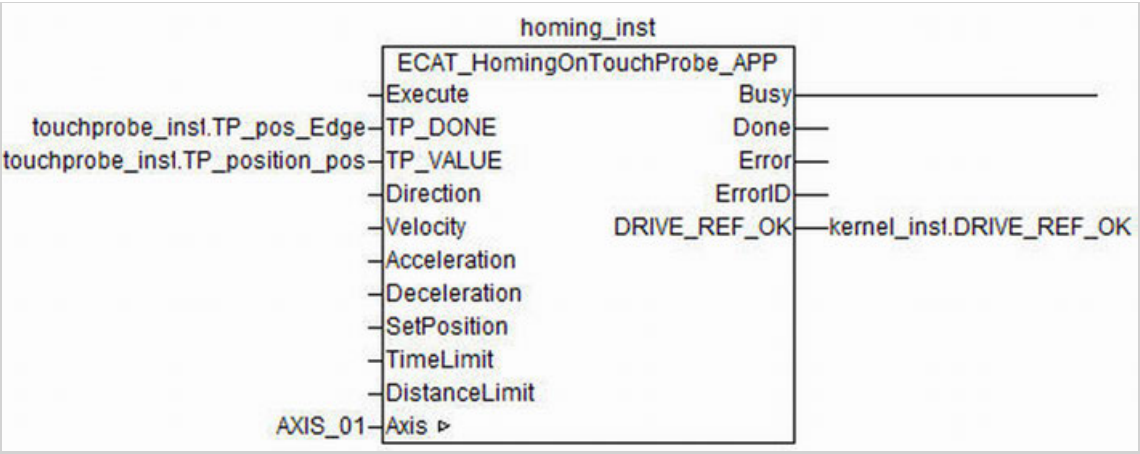
In the example the position control loop will be closed by the drive, therefore the input Drive_ActualPosition and the output Position_Reference are to be used. The inputs refering to the position control loop of the parameter function block do not have to be set.











To enable and disable the drive Drive_Release and Drive_Inoperation have to be connected to the control function block ECAT_CiA402_Control_App of the library ABB_Ecat_CiA402_AC500.library, which controls the status and control word of the drive.

All function blocks from this library are not password protected and free to be changed in order to be adapted for different drives. The library and the function blocks are marked with the ending _APP.

For a precise homing finish, a position can be latched from the MicroFlex e150 drive, by the use of the Touch Probe objects, which are available in the MicroFlex e150 EtherCAT configuration. To use these Touch Probe objects the function block ECAT_CiA402_TouchProbe_APP can be used.

In addition to the Touch Probe function block the homing can be executed with the function block ECAT_HomingOnTouchProbe_APP, which uses the latched position from the Touch Probe objects.

How to enable and disable a drive

In order to enable a drive the function block MC_Power has to be used within the applicational layer. The kernel function block will then, if possible, output a rising edge on the output Drive_Release which can be connected to the drive-control function block which performs the needed actions on the drives control word to enable the drive. As soon the drive states enabled, this signal can be connected to the input Drive_In_Operation of the kernel function block. The axis state according to the single axis state diagram of PLCopen will then switch from Disabled to Standstill.

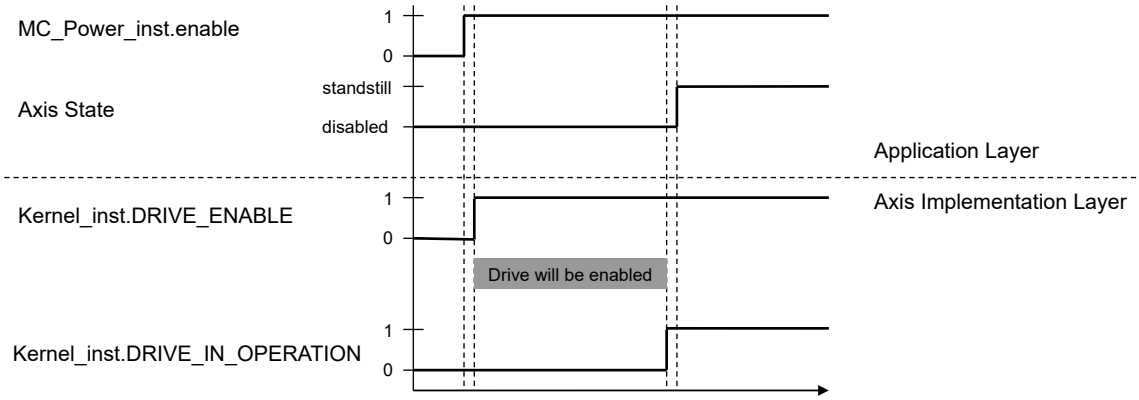


Fig. 222: Enabling sequence of a drive

As long as the drive is in state Disabled or ErrorStop the input Drive_Actual_Position will be copied to the output Position_Reference of the kernel function block. The output Speed_Reference will be zero.

When the axis is in operation, which means it is not in state Disabled or ErrorStop, then the output Position_Reference will be calculated by the kernel function block and the position control loop will be closed, which outputs non zero value for the output Speed_Reference in case of a following error. The input Actual_Position should then follow the position reference. The difference of both values is the following error and will be supervised by the kernel function block.

In case of drive problem, Drive_InOperation should be reset. The function block will open the position control loop and Speed_Reference will be set to zero.

For the most drives the status is control by the drives control word whereas the drives status word represents its actual status. In order to enable the drive it might be necessary to pass through several drives states according a defined scheme which depends on the used drive. Therefore the library ECAT_AC500_APPL_V21.lib is added to PS552-MC which contains function blocks to operate with different drives on an EtherCAT network. There is also the PS553-DRIVES software which can be used to control the state of a drive.

How to use the axis simulation

It is possible to use a simulated axis instead of a real drive.

The axis simulation can be used in the following use cases:

- When the real drive is not available the simulation can be used to test all available motion functionalities to verify the application program.
- The simulation can be used to create a virtual master axis and synchronize other axes to it.

The simulation is realized by the function block CMC_Axis_Simu_Real or CMC_Axis_Simu_Int. Depending on the used version of the kernel function block the corresponding version of the simulation function block has to be used.

Homing will be possible if the limit-switches (data type CMC_Axis_IO) are simulated also. This is not done by CMC_Axis_Simu but could be realized in the PLC program.

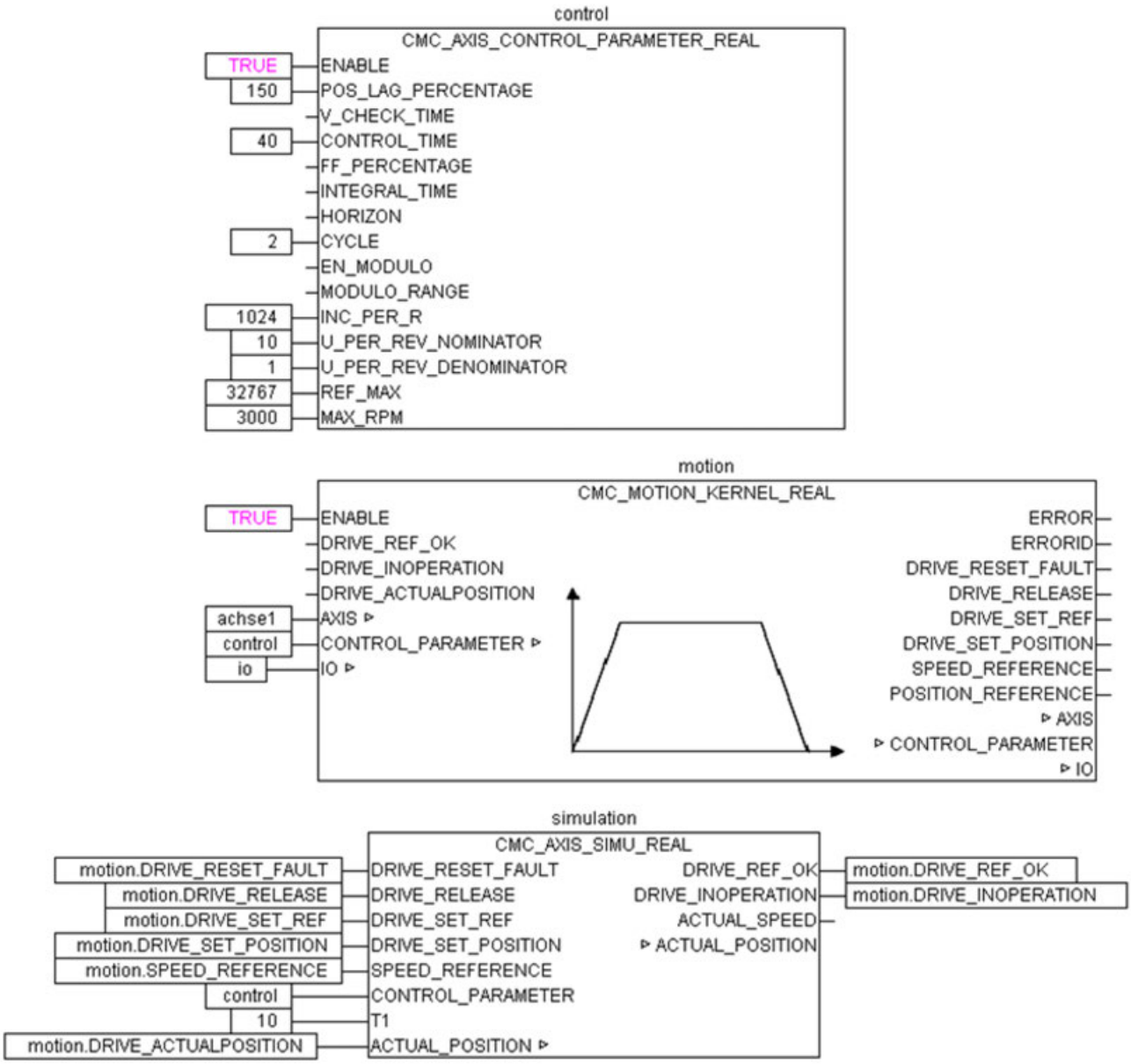


Fig. 223: Example for Simulation

The drive velocity is simulated by PT1-Characteristic. The input T1 gives the time constant for this PT1 as multiple of the cycle time. All other properties are simulated according to the CMC_Axis_Control_Parameter.

> *The value of the time behavior from the axis simulation function block set by the input T1 has to be at least four times smaller than the value of the axis parameter Control_Time from the parameter function block.*

How to perform a homing

The homing of an axis is a procedure which consists of up to two phases. For each phase there are different function blocks available. The available function blocks are according to PLCopen and belong to the application layer.

Table 167: Overview of the available homing function blocks

	Phase 1	Phase 2/Finish Homing
MC_StepAbsSwitch	X	
MC_StepDirect		X
MC_StepLimitSwitch	X	
MC_StepRefPulse		X

In order to create a complete homing sequence one function block of each phase can be used.

First phase

The used function blocks will change the axis state to Homing and will move the axis to approach installed limit switches or a dedicated absolute switch in the desired directions. No manipulation of a position value will be done in this phase. The use of function blocks of this phase is optional for a homing.

The signals of the installed limit switches have to be written to a variable of the data type CMC_Axis_IO.

Second phase

Function blocks from this phase will also change the axis state to Homing if this has not already happen and will finish the homing. Therefore a new position will be set to the axis. The axis state will then switch back to Standstill.

The use of a function block of the second phase is mandatory for a homing.

In general with AC500 Central Motion Control there are two position values: One position value will represent the encoder counts of a drive or the CD522 module which is connected to the input Drive_ActualPosition of the kernel function block. The other position is a user defined scaled unit which is used for PLCopen function blocks.

There are different ways to finish the homing by manipulate and adjust a position value. Which value should be manipulated depends on the used drive or module and its capabilities. See the following types A, B and C.

Type A

The user defined position unit will be changed only. The function block MC_StepDirect must be used here. This type of homing is less complex than the other types but also less precise.

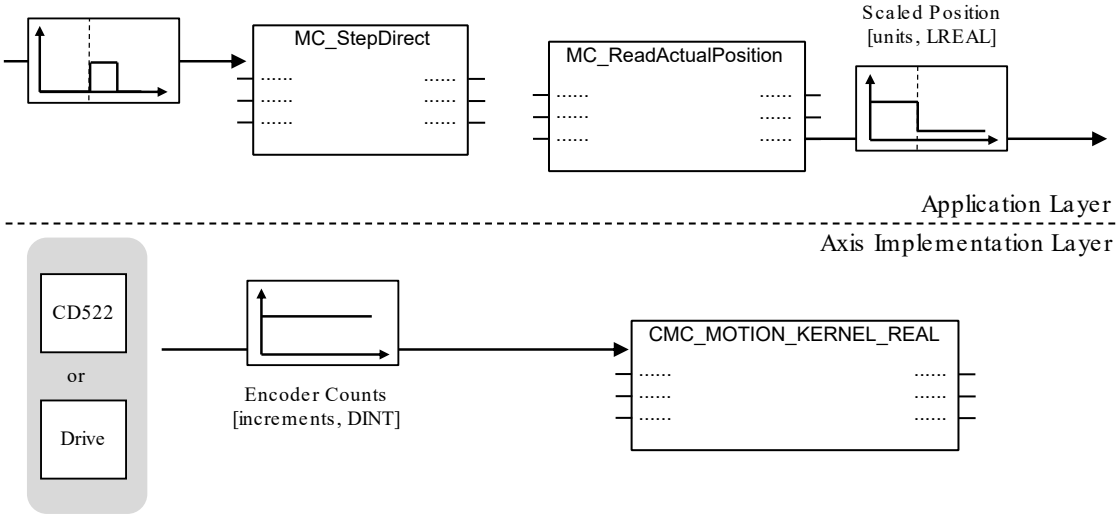


Fig. 224: Homing Type A

Type B

The Drive or the CD522 module will change its own position value, the encoder counts.

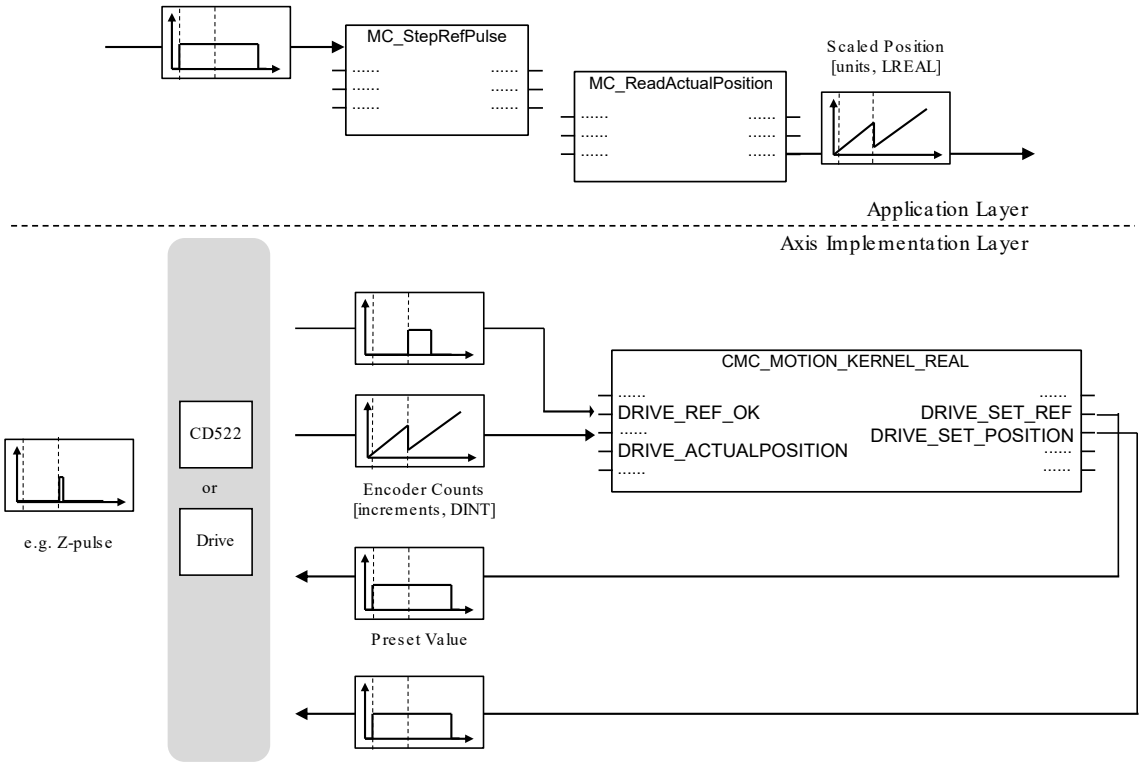


Fig. 225: Homing Type B

The process will be started by the execution of the function block MC_StepRefPulse.

The axis will start to move.

The output Drive_Set_Ref of the kernel function block will then set the drive to sense for a digital signal. At the same time the kernel function block outputs a preset value which will replace the actual encoder count value at the moment the digital signal occurs.

This signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module in order to be used.

In the same cycle when the new position value is set there also has to be a boolean signal stating a new position value at the input Drive_Ref_Ok of the kernel function block. The user defined position value will then be shifted accordingly.

Example of type B for phase 2: *Chapter 1.5.9.4.2.1 “How to connect a drive” on page 2620*

Type C

The encoder count position value will not be changed but involves registration capabilities of a drive or the CD522 module.

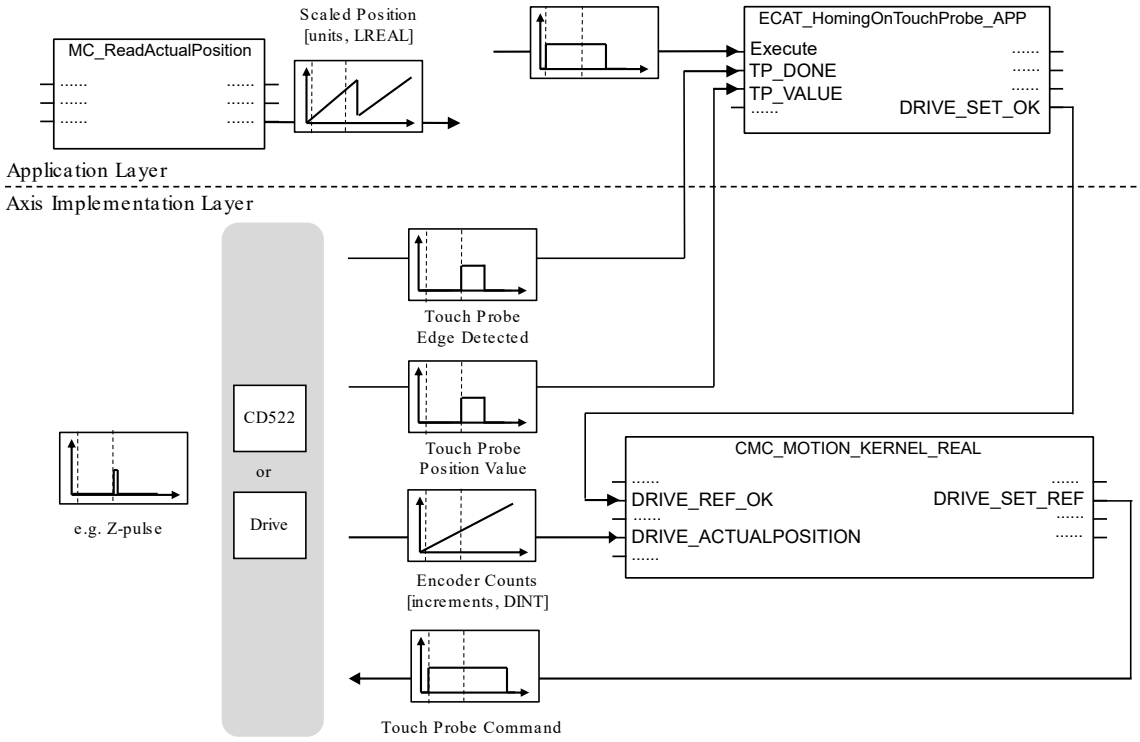


Fig. 226: Homing Type C

The process will be started by the execution of the function block ECAT_HomingOnTouchProbe_APP (ABB_Ecat_CiA402_AC500.library).

The axis will start to move.

The output Drive_Set_Ref of the kernel function block will then command the drive or the CD522 module to activate the Touch Probe functionality. This will configure the drive to latch a position at the moment a digital signal occurs. The digital signal can be a Z-pulse of an incremental encoder but also any other signal from a sensor. This functionality may require a configuration of the drive or the CD522 module in order to be used.

In combination with the latched position value there is a boolean signal which states that a new latch value has been received. In case of the module CD522 this encoder count position value has to be converted from encoder counts to equivalent user scaled units by the use of the function “CMC_Get_Units_From_Inc” (CompactMotionContral_AC500_V12.lib) before it can be connected to the function block ECAT_HomingOnTouchProbe_APP.

To manage the Touch Probe objects of a drive within the CiA402 profile (e.g. MicroFlex e150) the function block ECAT_HomingOnTouchProbe_APP (ECAT_AC500_V12.lib) can be used. This will also cover the conversion from encoder counts to user scaled units.

At the end of the process the function block ECAT_HomingOnTouchProbe_APP will manipulate the user scaled position value according to the latched position from the drive and the users settings.

Example of type C for phase 2: ↳ *Chapter 1.5.9.4.2.1 “How to connect a drive” on page 2620*

For further information see: *AN00220-001 - AC500 and MicroFlex e150 - EtherCAT Homing Methods*

How to Use a CAM curve

The CAM functionality is only available in combination with the kernel function block CMC_Motion_Kernel_Real ↳ *Chapter 1.5.9.4.8.1 “CMC_MOTION_KERNEL_REAL” on page 2659*.

General usage

The usage of a CAM function is based on the following elements:

- CAM table defined with the data type MC_PProfile.
- An instance of the function block MC_CamTableSelect
  ↳ *Chapter 1.5.9.6.3.1 “MC_CamTableSelect” on page 2832*.
- An instance of the function block MCA_Cam_Extra (optional)
  ↳ *Chapter 1.5.9.6.5.1 “MCA_CAM_EXTRA” on page 2878*.
- An instance of function block MC_CamIn
  ↳ *Chapter 1.5.9.6.2.1 “MC_CamIn” on page 2799*.
- An instance of function block MC_CamOut
  ↳ *Chapter 1.5.9.6.2.2 “MC_CamOut” on page 2803*.

The following steps are necessary to use a CAM table

1. Declare a CAM table as an array of the data type MC_PProfile in the program.
2. Write data to this array.
3. Use the address of the CAM table at the input CamTable of the function block MC_CamTableSelect.
4. Execute the function block MC_CamTableSelect to process the data of the CAM table with the function block’s input parameters
5. Additionally you can execute the function block MCA_Cam_Extra for optional parameters after the processing of the function block MC_CamTableSelect.
6. Execute the function block MC_CamIn to start the slave axis movement according to the CAM table data and parameters.
   ⇨ The axis will operate in the axis state Synchronized Motion.
7. To leave the axis state you can execute the function block MC_CamOut.
   ⇨ The axis state will switch to state Continuous Motion and maintains its last velocity as long as there is no other command.
8. You can also use any other motion command interrupt the Synchronized Motion.

CAM table

CAM data is done with one table (two dimensional – describing master and slave positions together).

The data of the elements (array of data type MC_PProfile) can either be assigned within the declaration or can be assigned during run time before the execution of the function block MC_CamTableSelect.

It can be filled with data in the following ways:

- To use a predefined variable list.
- To calculate the values within the program (before using the MC_CamTableSelect).
- To send values by any communication access to the PLC.

In order to use the new data it is necessary to execute the function block MC_CamTableSelect again. In case the CAM table is executed the function block MC_CamTableSelect may not be executed.

Elements of the data type MC_PProfile: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

The inputs MasterSyncPosition and MasterSyncDistance of the function block MC_CamIn can be used to define a distance to synchronize the slave axis onto the CAM table during the start. In case master axis moves with negative velocity the parameter MasterSyncDistance can be negative. The MasterSyncPosition should always be within the range of the CAM table master position.

MasterSyncDistance = 0 will deactivate the synchronization. In this case the slave axis should be moved on the CAM curve before MC_CamIn is executed, otherwise a following error can occure.

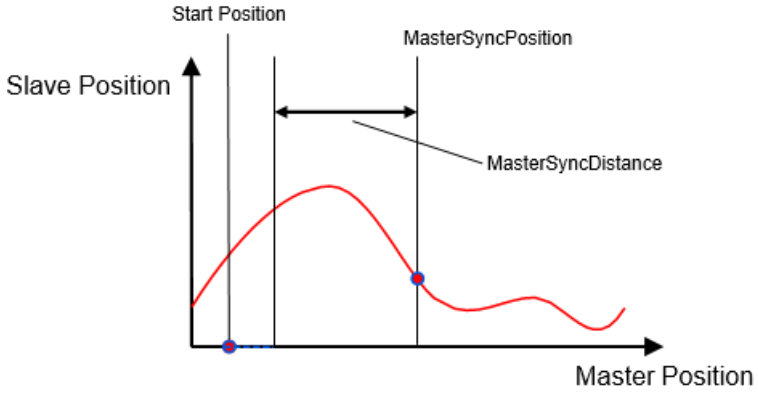


Fig. 227: CAM profile figure

The master position in the CAM table must be strictly monotonic rising.

The length of a CAM table is just restricted by the memory size of the PLC. When long tables are used, it is recommended to call CamTableSelect in a task with lower priority as it will need a considerable computing time.

It is possible to hold several CamTables as a pool and to switch from one to another. This has to be done at matching positions as no means for synchronization are available.

The offset and scaling values (except the time-scale) are transferred continuously. This will allow to follow a "Moving Target" by adjusting these values.

The parameters at MC_CamTableSelect, MC_CamIn and function and MCA_Cam_Extra also modify the behavior:

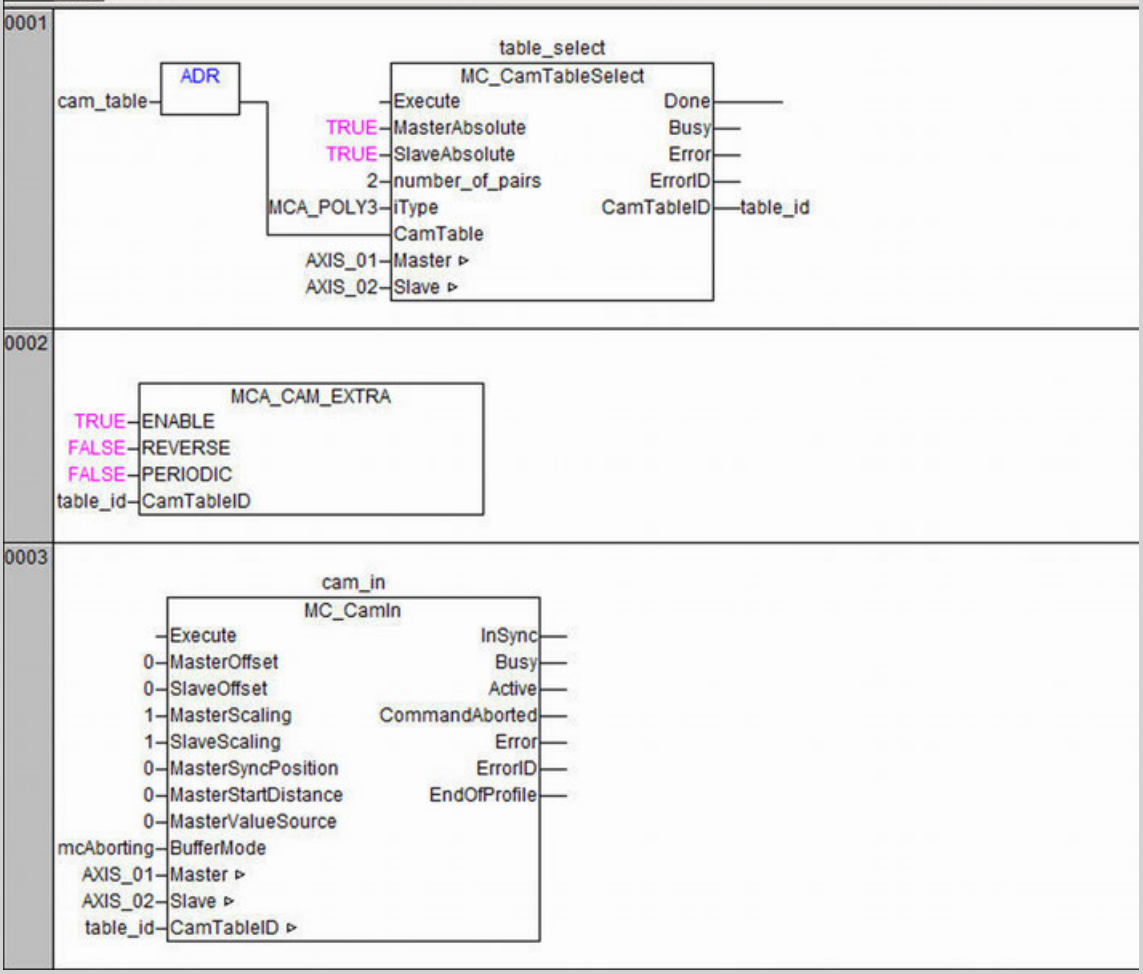
Parameter MC_CamTableSelect	Type	Default value	Comment
MasterAbsolute	BOOL	FALSE	TRUE=Master_position from MC_PProfile equals the master axis absolute position. FALSE=CAM is executed relative to the master axis actual position at start.
SlaveAbsolute	BOOL	FALSE	TRUE=interpolation_point from MC_PProfile equals the slave axis absolute position. FALSE=CAM is started from actual slave position. The values "interpolation_point" are relative to the slave axis position at start.
iType	MC_ABB_iTypes_ENUM		Interpolationtype.
Number_of_pairs	INT		Number of points used in TimePosition Array.

Parameter MC_CamIn	Type	Default value	Comment
MasterOffset	LREAL	0	Just used with MasterAbsolute=TRUE, ignored otherwise. Used position for cam-table is: Master axis position-Masteroffset.
SlaveOffset	LREAL	0	Just used with SlaveAbsolute=TRUE, ignored otherwise. Used position is slave axis position=interpolation_point+Slaveoffset.

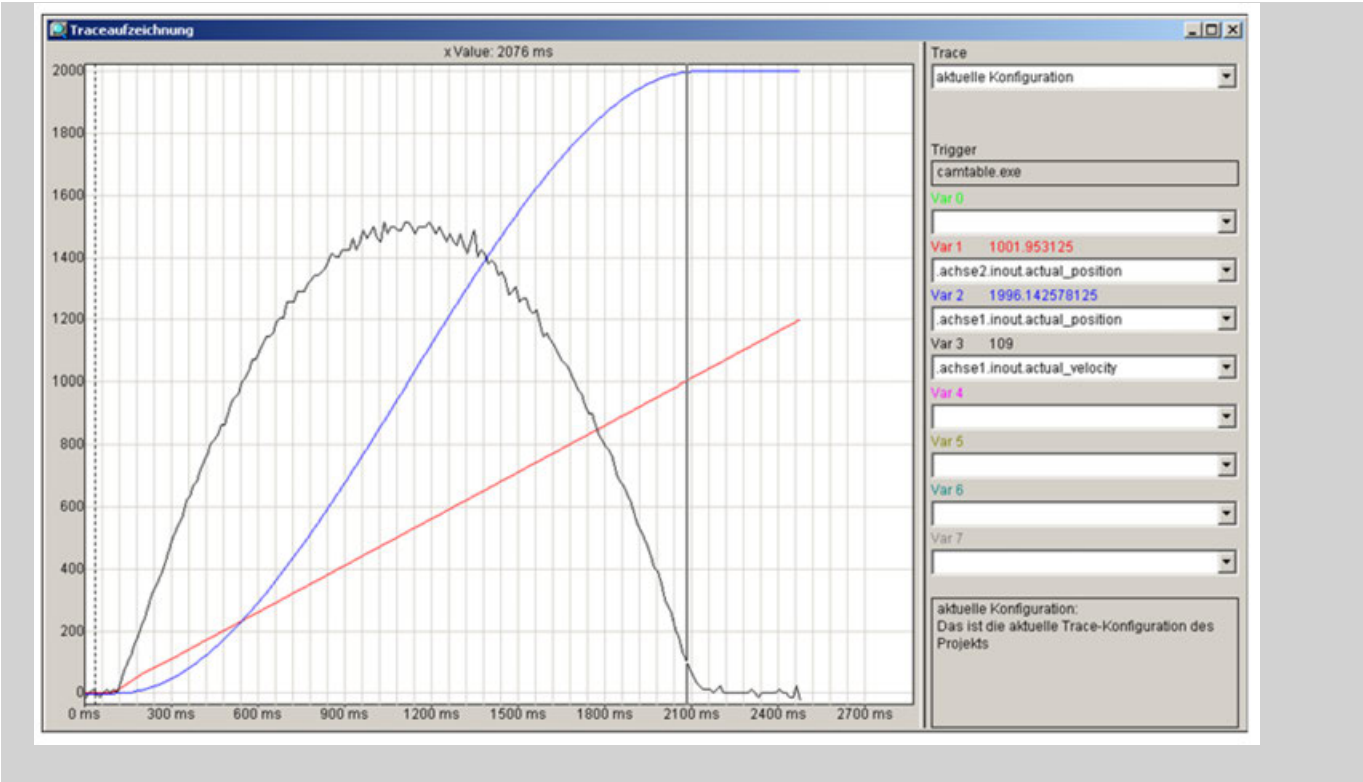
Parameter MC_CamIn	Type	Default value	Comment
MasterScaling	LREAL	1	The position used for interpolation is multiplied by MasterScaling, e.g MasterScaling=2, the scaled master will pass the position range with double velocity and within the half distance compared to its real velocity and position.
SlaveScaling	LREAL	1	Interpolation result is multiplied by Slave-Scaling, e.g SlaveScaling=2: Slave axis will run twice the distance.
MasterSyncPosition	LREAL	0	Start synchronization at master axis position=MasterSyncPosition-Master-StartDistance+MasterOffset, meet the CamTable at master axis position=MasterSyncPosition. In case of MasterAbsolute=FALSE: start at "actualPosition+MasterSyncPosition-MasterStartDistance", meet the CamTable at "actualPosition+MasterSyncPosition"!!! It is just possible to use the "sync" mechanism when the axis is in StandStill on start.
MasterStartDistance	LREAL	0	A negative value will create a reverse synchronization mode, which means the master should move in negative direction to synchronize. It is independent from the Reverse-Bit which indicates how to end the movement.
These 2 parameters are "extras" to be written with the MCA_Cam_Extra function. When the parameters are used, the MCA_Cam_Extra has to be called after the MC_CamTableSelect.			
Periodic	BOOL	TRUE for master “Modulo”, FALSE for master linear axis	CamTable will not reach "EndOfProfile" but will be repeated periodically. When the master is a linear axis, it has to move forward and backward within the CamTable position range, but even when it leaves this position range, the CamTable will stay active.
Reverse	BOOL	FALSE	Just necessary when a CamTable is NOT "periodic" and will run in reverse direction (master with negative velocity) Reverse=FALSE, the CamTable is ready when the master leaves the position range in positive direction, e.g. when it moves from 359° to 0° on a rollover axes Reverse=TRUE, the CamTable is ready when the master leaves the position range in negative direction.

Example for CAM curve

```
0002 VAR
0003     cam_table     : ARRAY [0..1] OF MC_PProfile :=
0004                   (Master_position  := 0     ,interpolation_point:= 0     ,Velocity_ratio:=  0  ,Acceleration_ratio:= 0 ),
0005                   (Master_position  := 1000  ,interpolation_point:= 2000  ,Velocity_ratio:=  0  ,Acceleration_ratio:= 0 );
0006     cam_in        : MC_CamIn;
0007     table_id      : MC_CAM_ID;
0008     table_select  : MC_CamTableSelect;
0009     AXIS_01       : AXIS_REF;
0010     AXIS_02       : AXIS_REF;
0011 END_VAR
0012
```



In the example, the slave will run from 0 to 2000 while the master runs from 0 to 1000. The slave will start and end with velocity=0, no matter which velocity the master has during start. The slave will reach the maximum velocity when it is at position 1000 and the master is at position 500.



How to use an external axis

To use multiaxis PLCopen function blocks with an externally sensed axis as master axis the following structure can be used for the axis implementation:

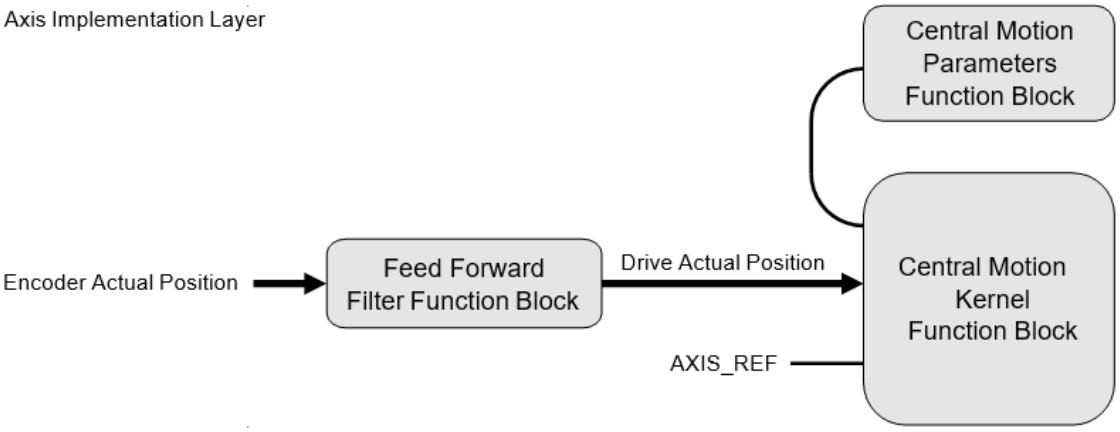


Fig. 228: Structure synchronization to an external axis

The use of a feed forward filter function block is needed if the slave axis has to follow the position of the external axis. In this case there will be a time delay between sensing the position of the external axis and moving the follower axis along the sensed position. The filter function block will then add a certain distance to the external axis' position depending of its speed.

The filter function block MATH_LINEAR_REGRESSION from the library MathFunctions_AC500_V23 can be used here.

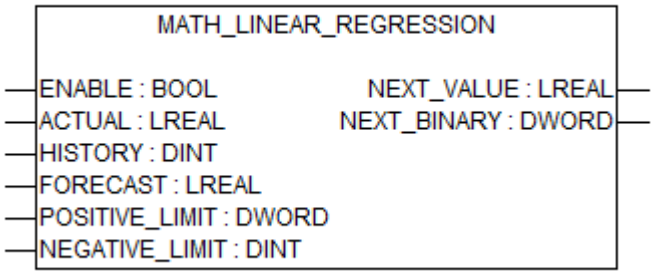


Fig. 229: Filter function block to feed forward an externally sensed position

For an axis which is following the external axis, the value “mcActualValue” (from MC_Source enumeration) for the input “MasterValueSource” for multi-axis PLCopen function blocks has to be used.

When the filter function block MATH_LINEAR_REGRESSION is used to process an actual position, 2 different purposes are fulfilled:

- A jitter or noise can be compensated
- It is possible to calculate a forecast-position to compensate for a delay in position measurement

> *Process the actual position or any other master axis always before the slave axis.*
>
> *Otherwise, an additional one cycle-delay is introduced.*

The MATH_LINEAR_REGRESSION function block calculates the progress for a variable which is captured in equidistant periods of time and is assumed to follow a linear curve. It uses the Gauss “least squares” -algorithm to do so. The line is calculated in a way that the sum of squares for the distances from the measured points to the assumed straight line is minimized.

A noise or jitter influence of the value is compensated and a predictive value for the variable with an adjustable forecast horizon can be calculated.

Linear equation:

$$\mathrm{Line[i]} = \mathrm{gradient} * \mathrm{i} + \mathrm{offset}$$

Sum of squares:

$$sum = \sum_{i=1}^{\infty history} (\mathrm{x[i]} - \mathrm{line[i]})^2$$

The gradient and offset for the line are calculated in a way that “sum” is minimized. Then these 2 values are used to calculate the forecast value:

$$\mathrm{NEXT\_VALUE} := \mathrm{gradient} * \mathrm{FORECAST} + \mathrm{offset}$$

FORECAST=0 would mean: value right now, no future or past considered.

When the ACTUAL value is a modulo value, for example a single turn encoder or a rollover axis, this has to be considered in the calculation. The 2 input values POSITIVE_LIMIT and NEGATIVE_LIMIT can be used to configure this. They define the upper and lower limit for ACTUAL. Also, the NEXT_BINARY will as a result be limited to these borders.

Example

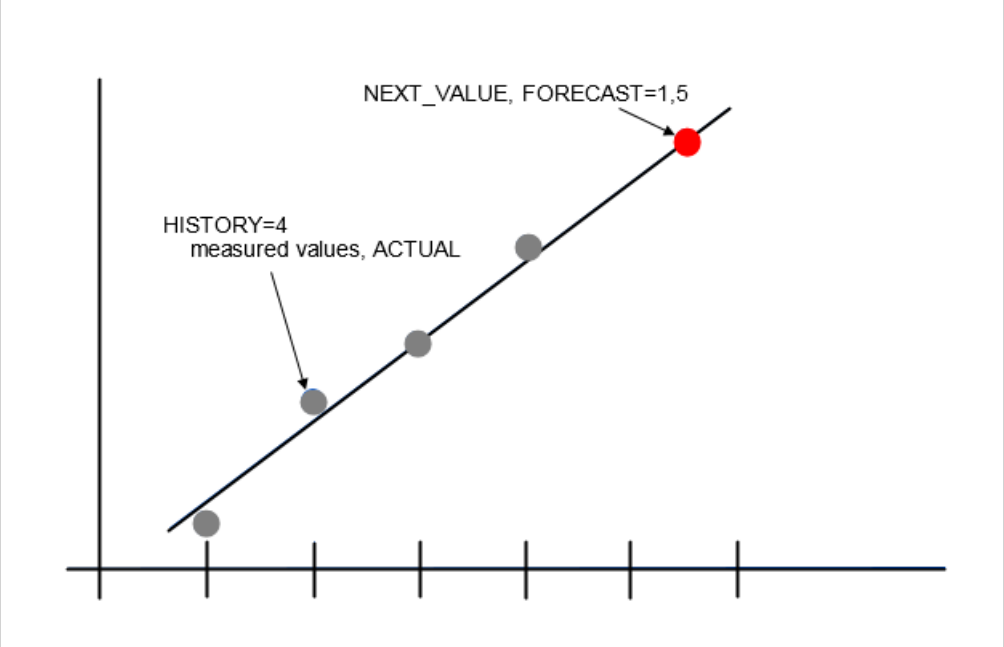


Fig. 230: Next Value_Forecast

How to use an encoder/drive with <> 32-bit position overrun

The incremental position as actual position at the function block CMC_Motion_Kernel_Real is usually assumed as position with a 32-bit position overrun. As well as it is the reference position which is sent to the drive.

Any modulo-axis configuration should be done inside the PLC.

Some drives are requested to correct their positions themselves for a non-linear axis which should constantly run into the same direction.

In this case, the drive has to be configured as a modulo-axis and the function block CMC_Motion_Kernel_Real needs some additional function blocks to create the 32-bit value ↳ *Chapter 1.5.9.4.3.4 “Roll-Over axis” on page 2646*.

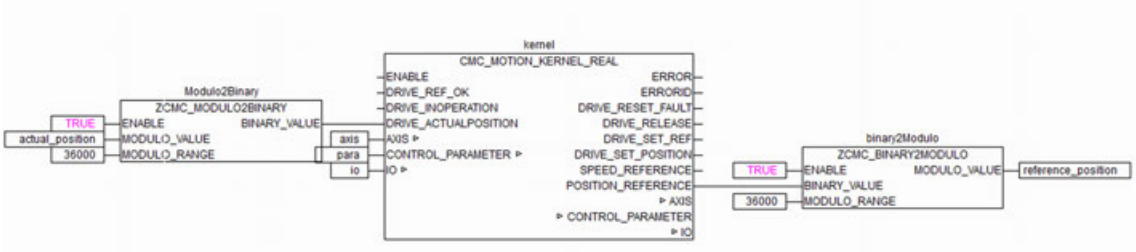


Fig. 231: Kernel

The function block CMC_Modulo2Binary will convert any position with any Modulo_Range to a 32-bit binary position.

The actual_position is assumed to run between 0 to Modulo_Range.

The actual_position should not change > 1/4 Modulo_Range between two scan cycles.

The function block ZCMC_Modulo2Binary will convert the 32-bit binary position reference from CMC_Motion_Kernel_Real to a position reference which runs from 0 to Modulo_Range.

How to do position correction “on the fly”

Sometimes it is required to have a position correction "on the fly". For example, it can happen that a position is wrong due to mechanical slip and that a switch which is passed by during the movement is used to capture a position value.

In other cases, it is required to synchronize the position to a print mark, so an actual_position has to be corrected, but not the movement of the printed material.

For both applications, the function block MCA_SetPositionContinuous can be used. It will use ramps and a limited velocity for the correction, so it will be tolerable to execute it during an ongoing movement and while the axis is activated in a multi-axis movement.

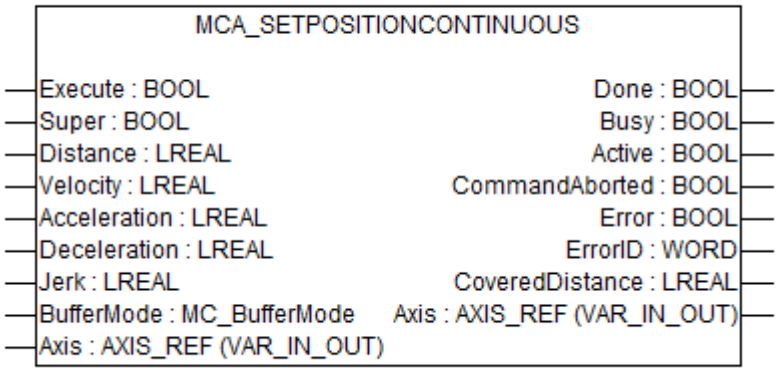


Fig. 232: MCA_Set PositionsContinuous

The block can be used in any axis state except ERRORSTOP and HOMING.

Two different operation modes are possible:

1. SuperImp=FALSE
   - The actual_position will be modified.
   - The block will not cause any movement.
   - If a PLCopen block in DISCRETE_MOTION (positioning) is active during the execution, this block will not reach Done as the actual_position is modified.
   - If a slave axis is coupled to an axis while MCA_SetPositionContinuous is executed (with SuperImp=FALSE) it will follow.
   - This mode is possible while the axis is in state DISABLED.
2. SuperImp=TRUE
   - The actual_position will stay constant.
   - A mechanical movement is executed (without changing the axis state machine).
   - A slave axis will not follow.
   - This behavior is similar to a superimposed movement.
   - It is not possible when the axis is in state DISABLED.

The block can just be aborted by another MCA_SetPositionContinuous.

How to limit the movement

It is possible to limit the movement by position (software limit switches) and by velocity. By default, no software limit switches are activated in PS552. It is possible to activate them by accessing some PLCopen parameter.

The functionality described below is just available with linear axes.

	Parameter	Data type	Minimum	Maximum	Default	R/W	Description
2	SWLimitPos	DINT	-2147483647	2147483647	2147483647	R/W	Positive software limit switch position.
3	SWLimitNeg	DINT	-2147483647	2147483647	2147483647	R/W	Negative software limit switch position.
4	EnableLimitPos	BOOL	FALSE	TRUE	FALSE	R/W	Enable positive software limit switch.
5	EnableLimitNeg	BOOL	FALSE	TRUE	FALSE	R/W	Enable negative software limit switch.
2003	EnableLimit2Decelerate	BOOL	FALSE	TRUE	FALSE	R/W	Enable software limit switches to decelerate
2004	EnableLimitAbort	BOOL	FALSE	TRUE	FALSE	R/W	Enable that software limit switches will abort ongoing movement FALSE = Limits position and velocity, decelerates and shows a warning until the position limit is reached, then ERROR STOP TRUE = Switches off any ongoing motion and decelerates to the position limit, then ERROR STOP
2005	EnableLimtVelocity	BOOL	FALSE	TRUE	FALSE	R/W	If the velocity is limited the unmoved position will be covered whenever possible
2006	SWLimit2DecPos	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2007	SWLimit2DecNeg	LREAL	-2147483647	2147483647	2147483647	R/W	Used as end position for EnableLimit2Decelerate
2008	MaxPositionGap	LREAL	0	21474836 4700	0	R/W	Used to stop the ongoing movement if position is behind

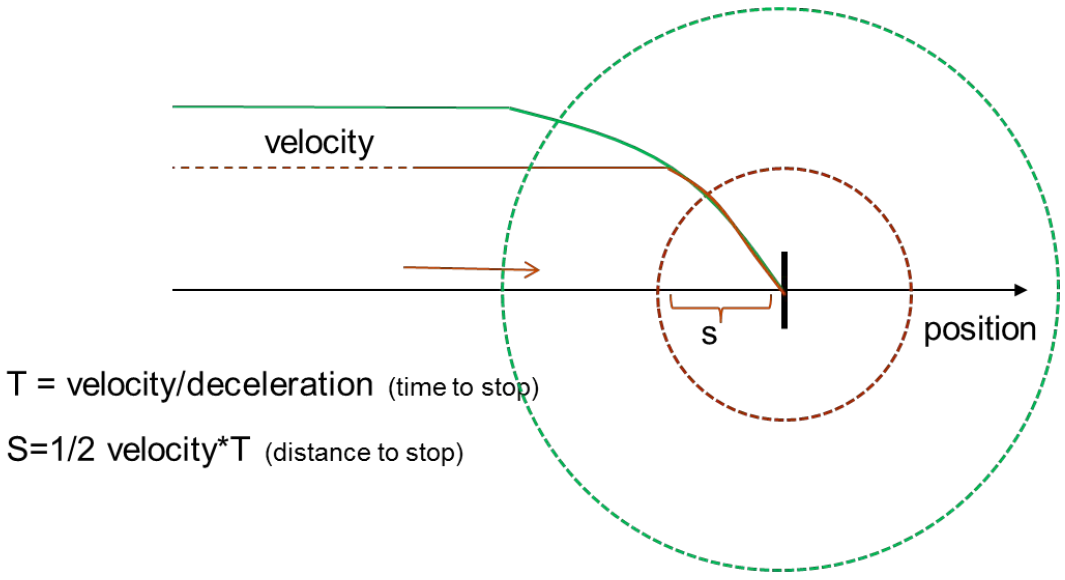
The following different behavior is possible:

- No limitation at all (default)
- Limit position with ERRORSTOP:
  - Limit position between SWLimitNeg to SWLimitPos, axis to state ERRORSTOP in case the position range is left.
- Limit velocity and acceleration:
  - Limit velocity to paraMaxVelocityAppl and acceleration/deceleration to paraMaxDecelerationAppl, create WARNING_VELOCITY, not state changes for axis, abort movement is optional when MaxPositionGap is reached due to limitation.
- Limit Position with ramp-down:
  - In addition, it is possible to limit the position between SWLimit2DecNeg and SWLimit2DecPos. paraMaxDecelerationAppl is used to ramp down.

When activated with EnableLimitPos or EnableLimitNeg, the reaction will be as follows:

- When the control position reaches the respective limit switch, the axis will go to state ERRORSTOP, and Drive_Release will be switched off. The actual_position might be behind, depending on the following error. It is assumed that a drive or application specific braking is performed. The axis will be stopped behind the limit.
- The axis could be switched on again by MC_Power. A movement in the opposite direction will be possible.
- The functionality of EnableLimitPos and EnableLimitNegis unchanged.

You can use the limitation of movement to achieve a soft or adjustable braking in advance before reaching the software limit switch. The limitation is activated by three Boolean parameter and will calculate a position distance to the limit switch, which depends on the actual velocity and given deceleration ramp. “paraMaxDecelerationAppl” is used for deceleration. It will decelerate the axis by the given deceleration ramp when the calculated position is reached and stop at the software limit switch. The original behavior is not modified, so if also these software limit-switches are activated, the axis might be set to state ERRORSTOP.



There are 2 different modes:

- EnableLimitAbort = TRUE
  Any ongoing motion will be aborted immediately (when the distance to stop is reached, as shown in the above diagram), a warning is shown
  The axis will be decelerated to reach the software limit switch.
- EnableLimitAbort =FALSE, EnableLimitDecelerate=TRUE
  A warning is shown and the velocity is reduced, with respect to the given deceleration and position limit.
  The ongoing motion is not aborted. If it was just a “tight fit”, e.g. in a master slave movement and the direction is turned soon enough, it might be possible to continue the movement.
  As the ongoing movement is not interrupted, an activated movement might not be completed, for example a MC_MoveAbsolute will never reach its target position. A warning is shown at function block CMC_Motion_Kernel_Real.

When EnableLimitPos = TRUE or EnableLimitNeg = TRUE, and the values for SWLimitPos or SWLimitNeg are set, the axis will be set to state ERRORSTOP when these position limits are reached.

In addition, the function block will allow to limit the velocity. With EnableLimitVelocity = TRUE, it will monitor the velocity demand from the position reference and limit the position reference, so the given velocity limit will not be exceeded. A warning will be shown. The velocity used for limitation is MaxVelocityAppl.

The velocity limitation can be used to prevent short-term velocity peeks. The limited position will be caught up later, whenever possible. This can result in not-expected behavior. The WARNING issued by CMC_Motion_Kernel_Real can be checked and used to stop a movement. The movement will be aborted automatically when the position is by MaxPositionGap behind.

- *For a single axis movement, the commanded velocity is limited at the beginning. No position gap will occur.*
- *In a multi-axis movement, the slave axis follows a master. This can result in a position gap. A velocity peek from the master axis can be reduced by using the limitation. If the master is too fast because of the value for MaxPositionGap, the movement will be aborted.*

When EnableLimit2Decelerate or EnableLimitAbort are used, the velocity is limited to MaxVelocitySystem with EnableLimitVelocity = FALSE. The function modifies the position reference. This modified position reference is used to control the drive. Whenever the limitation interferes the kernel will show a warning or an error. The warning or error message will disappear when the situation is cleared.

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	TRUE	ERRORSTOP when positions exceed, no previous warning or deceleration.
5	EnableLimitNeg	TRUE	
2003	EnableLimit2Decelerate	FALSE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Motion_Kernel_Real. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	TRUE	
2004	EnableLimitAbort	FALSE	
2005	EnableLimtVelocity	FALSE	

Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	FALSE/TRUE	Reduce the velocity when reaching a position limit within the deceleration distance calculated by using MaxDecelerationAppl. Display a warning at CMC_Motion_Kernel_Real. The underlying movement stays active. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP also if EnableLimitPos or EnableLimitNeg are used. Otherwise, just the movement is limited, without affecting the state machine. An activated positioning movement will not reach its target. Velocity is limited to MaxVelocitySystem. The active PLCopen function block is aborted as soon as the warning is issued. With EnableLimitPos = TRUE or EnableLimitNeg = TRUE: When the Position limit is reached, the axis is set to mode ERRORSTOP.
5	EnableLimitNeg	FALSE/TRUE	
2003	EnableLimit2Decelerate	---	
2004	EnableLimitAbort	TRUE	
2005	EnableLimtVelocity	FALSE	

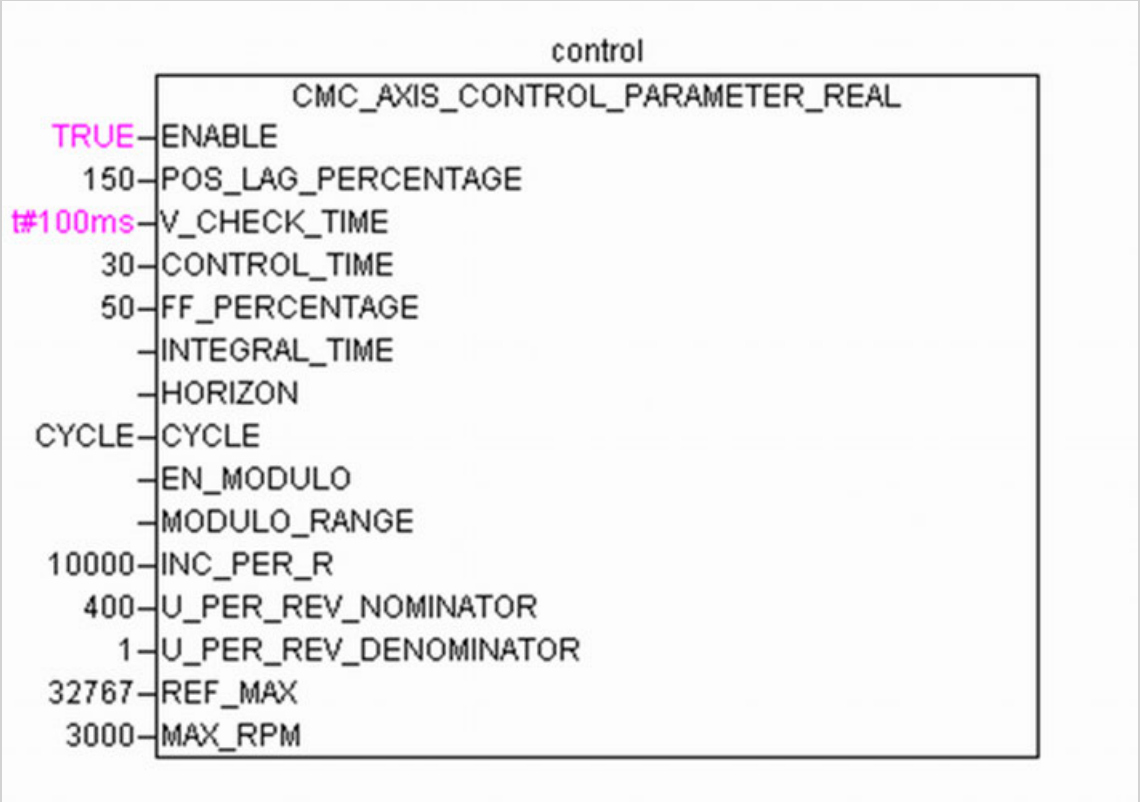
Parameter Number	Parameter Name	Value	Comments
4	EnableLimitPos	---	The velocity is checked and also limited to the value MaxVelocityAppl. A warning is shown. The active movement is not aborted. This functionality works independent from software limit switches.
5	EnableLimitNeg	---	
2003	EnableLimitDecelerate	---	
2004	EnableLimitAbort	---	
2005	EnableLimtVelocity	TRUE	

1.5.9.4.3 Axis parameters

The parameters for axis configuration and adjustment are set by the function blocks CMC_AXIS_CONTROL_PARAMETRS_REAL ↳ *Chapter 1.5.9.4.8.3 “CMC_AXIS_CONTROL_PARAMETER_REAL” on page 2666* or CMC_AXIS_CONTROL_PARAMETRS_INT ↳ *Chapter 1.5.9.4.8.4 “CMC_AXIS_CONTROL_PARAMETER_INT” on page 2669*.

Depending on the version of the kernel function block the corresponding version of the parameters function block has to be used. The instance will then be connected to the kernel function block by its instance name.

Example



- FF_PERCENTAGE=0
- INTEGRAL_PART=0
- HORIZON=0

In the example the control structure is a simple position control loop with just proportional gain. When the application does not require minimized position following error it should be used this way as it is simple to adjust, robust and requires minimal performance. The proportional gain is then adjusted by Control_Time. Just change values at CMC_Axis_Control_Parameter when the position control loop is open (Drive_Release=FALSE, the axis state is Disabled). The values are sending to the control loop whit a positive edge at "Enable". The CMC_Motion_Kernel block needs to be already enabled.

Supervision

Pos_Lag_Percentage

This parameter configures the position window for the supervision of the following error.

The default value is 150[%]. A value of 0[%] will deactivate the supervision function.

The size of the position window depends on the setting of the parameters Control_Time and Max_Rpm *“Control_Time” on page 2643*.

Position Window [Increments] = (Inc_Per_R) * (Max_Rpm/60) * (Control_Time/1000)

Position Window [Units] = (U_Per_Rev_Nominator/ U_Per_Rev_Denominator) * (Max_Rpm/60) * (Control_Time/1000)

Example

Position Window [Increments] = (10000) * (6000/60) * (50/1000) = 50000 [Increments]

Position Window [Units] = (1/1) * (6000/60) * (50/1000) = 5 [Units]

A value of 100% will result in a position window which corresponds to the expected following error with the giving Control_Time at Max_Rpm. Therefore it is recommended to use values higher than 100[%]. In case the parameter FF_Percentage is used smaller values can be used.

If the supervised position window is exceeded the axis state will change to ERRORSTOP.

V_Check_Time

After the configured time the drive’s actual velocity has to be at least 50 % of the commanded velocity. This function can also be used in case the Position Reference is transferred to the drive.

A value of 0 will deactivate this supervision function.

If the supervised velocity window is exceeded the axis state will change to ERRORSTOP.

Position control loop

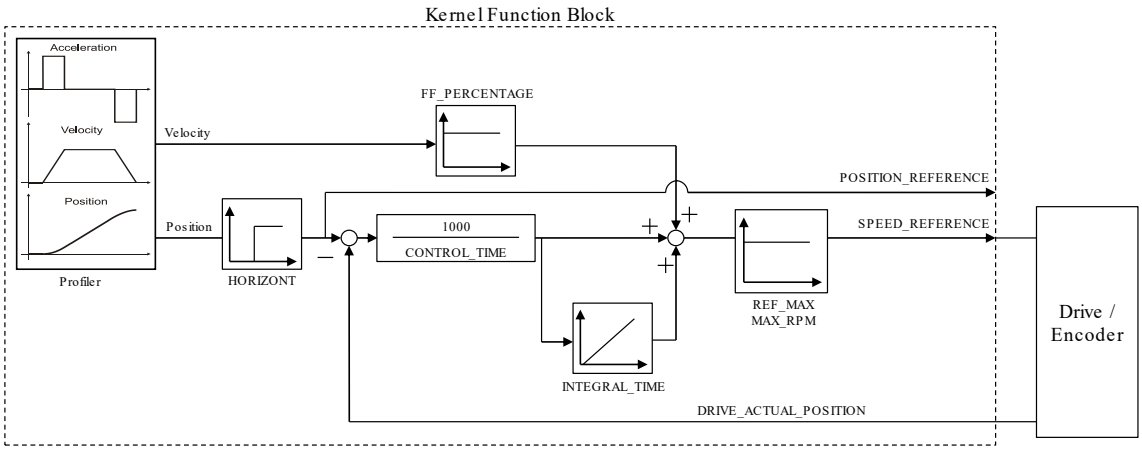


Fig. 233: Basic structure of position control loop

Control_Time

The default value is 100 which leads to a proportional gain of 10.

> *In case the value of Control Time is too short the position control loop will run into instability.*

In case the position control loop is not used this parameter must not be set to 0.

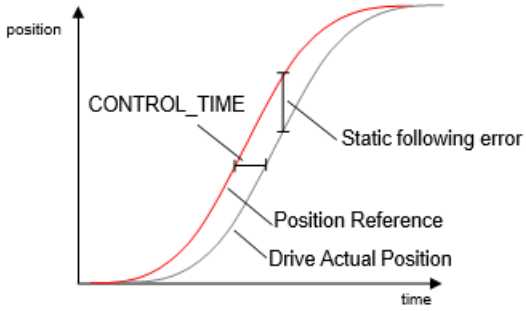
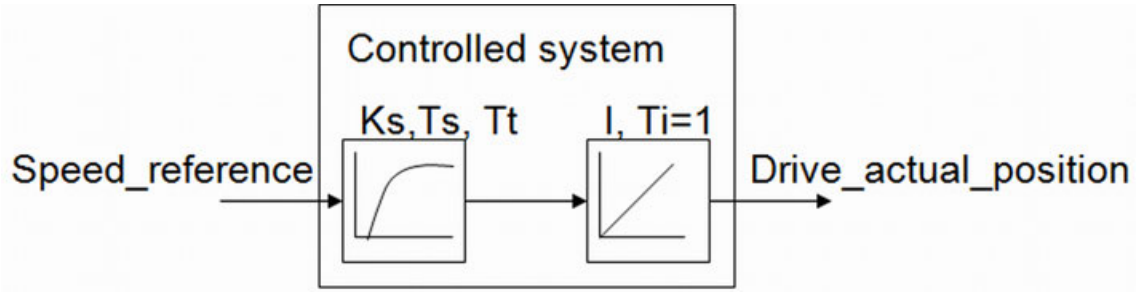


Fig. 234: Control Time and static following error in case the feed forward of velocity and the integrational part of the position control loop is not used.

The static following error depends on the axis velocity and can be calculated easily: Control Time multiplied by the axis velocity (p_error = v * CT).

In general it should be aimed to reach a high position control loop gain with a short Control Time to achieve a small following error. As the reaction times take account in the possible Control Time of the complete system (parameters of the drive control loop, PLC cycle time as well as the communication fieldbus) should be considered.

As a basic rule the Control Time should be at least four times longer than the reaction time between the output of the Speed Reference and the input of actual position.

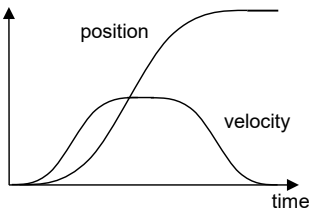


When the time Ts and Tt is measured, a control_time of 4 * (Ts + Tt) will result in an aperiodic damping of the position control loop. It is important to measure the values from inside the PLC (e.g. Trace) to have the complete reaction times included. Practical values for Control_Time might be from 50 - 500ms. The PLC cycle time as well as bus cycle times and mechanical reaction will influence the value.

FF_Percentage The default value is 0.

In case a velocity feedforward has to be configured a value of up to 80 is recommended. For larger values than 80 the parameter Horizon needs to be used as the resulted position will overshoot otherwise.

A value of 100 adds a velocity to the Speed Reference output which corresponds exactly to the ongoing Position Reference value.



Integral_Part

The integral part of the position control loop can be used to eliminate a permanent positioning error, e.g. in case of hanging loads.

The time value can be regarded as the time the integrator needs to sum up the input value to reach the same value for its output.

> *In case the Integral Part Time is too short the position control loop will run into instability.*

Horizon

A communication delay of the Speed Reference value to the drive system can cause an overshoot during positioning caused by the velocity feedforward gain.

This function will compensate this communication delay to prevent an overshoot by time shifting the signals Velocity Feed Forward and Position Reference relatively to each other.

The value of Horizon can be approximately assumed to be the time delay of the communication delay.

The delay time might be caused by the cycle time of the control loop and by any delay in sending the speed reference, delay in the drive to build up the torque and delay to receive the actual position. To overcome this delay, a Horizon > 0 might be used. The feed forward reference will be created in advance, while the proportional gain is applied to the original motion profile. The delay is then compensated.

This function should not be used if the feed forward parameter FF_Percentage is 0.

A value of 0 will deactivate this function, which is the default value.

While this function is used, it will increase the needed PLC calculation time for this axis.

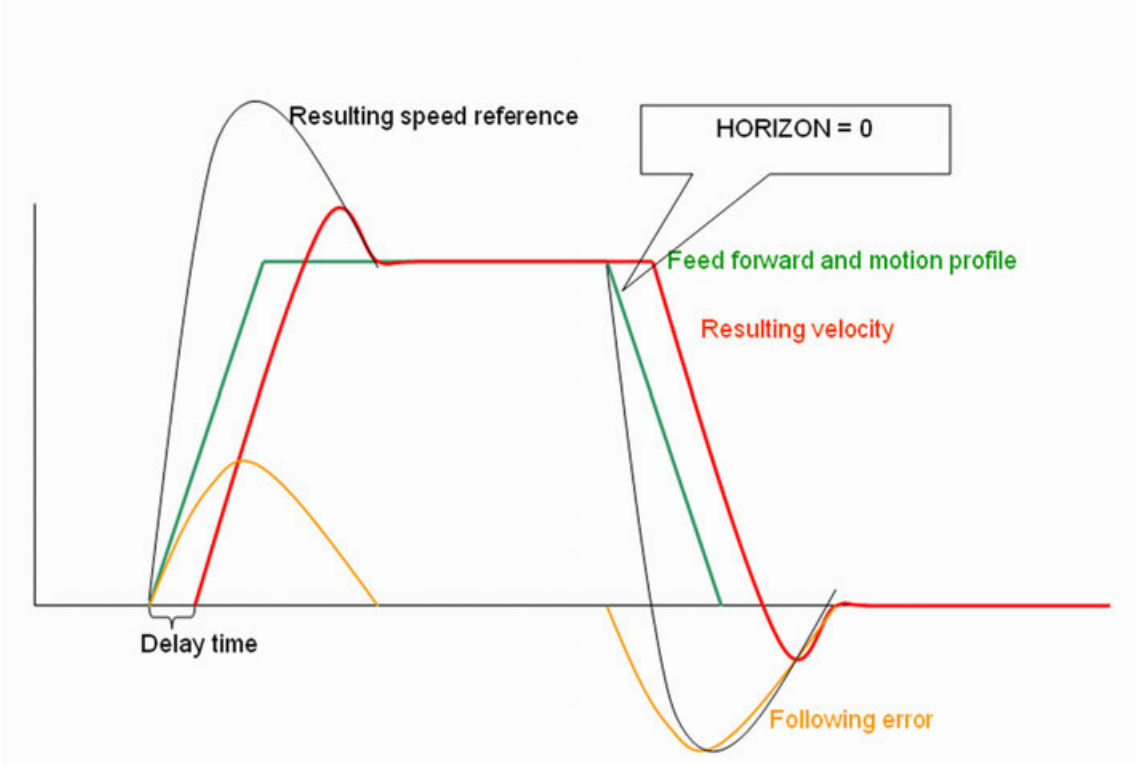


Fig. 235: Result with Horizon=0

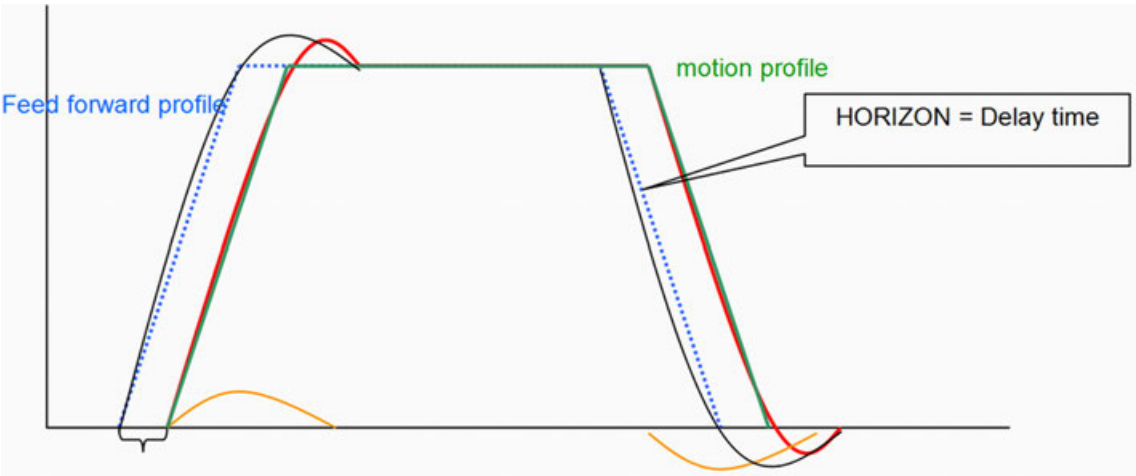


Fig. 236: Result with Horizon>0

PLC cycle time

Cycle

This parameter represents the cycle time in which the kernel function block of the axis is called. If the configured cycle time is not correct the resulting acceleration and speed of an axis will be not correct also.

In case the task execution of the axis is synchronized to a fieldbus (e.g. EtherCAT) the cycle time of the fieldbus has to be used.

Roll-Over axis

If the Position Reference value is used, the drive must able to perform a position over-run after 32 bit. If the drive's position over-run is different, it can be adapted with the function blocks CMC_Binary2Modulo and CMC_Modulo2Binary from the library ABB_MotionControl_AC500.library. Incompatibility can cause an axis to trip after hours of operation.

The possible position following error has to be smaller the ½ Modulo_Range. Make sure that the modulo range is large enough.

Position following error = (100 - FF_Percentage) * Max_Rpm * Inc_Per_R * Control_Time/6000000. This is the maximum value at constant velocity.

En_Modulo

With this parameter the axis can be configured as a roll-over axis.

Modulo_Range

The modulo range will be defined in drive position counts (DINT). It will result that the scaled unit position which is used by the PLCopen function blocks will stay within the defined range.

Example

```
En_Modulo             = TRUE
Modulo_Range          = 20000
Inc_Per_Rev             = 10000
U_Per_Rev_Nominator     = 360      (e.g. degree)
U_Per_Rev_Denominator     = 1
```

The scaled unit's position will cover the range from 0 to 720 (degrees).

In some cases it is not suitable to set the modulo range of an application with the DINT value of the parameter Modulo_Range only. In such cases the parameters 2001 Modulo_Nominator and 2002 Modulo_Denominator can be used to scale the parameter Modulo_Range to a more precise value.

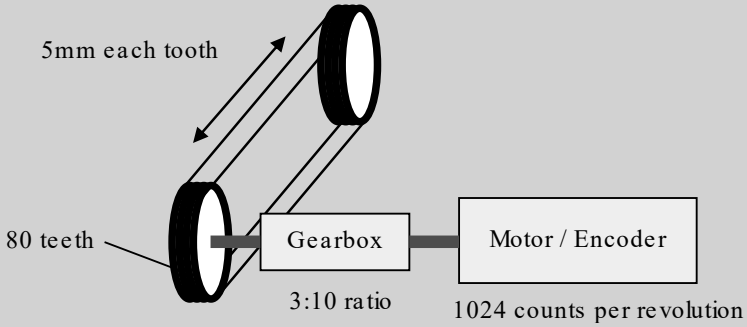
Parameter Modulo_Nominator and Modulo_Denominator (supported with CMC_Motion_Kernel_Real)

These parameters can be used to modify the Modulo_Range in a way that fractions of an increment could be used for 1 modulo (=rollover) distance

- Default: Modulo_Nominator=1 and Modulo_Denominator=1: the actual position for an axis is limited between 0 and Modulo_Range increments.
- Limitations: Modulo_Range*Modulo_Nominator < 2147483647. Otherwise: default values will be used.
- When modifying these parameters, the position control loop should be opened.

Example

```
En_Modulo             = TRUE
Modulo_Range          = 1024
Modulo_Nominator        = 10
Modulo_Denominator        = 3
Inc_Per_R             = 1024
U_Per_Rev_Nominator     = 80*5*3
U_Per_Rev_Denominator     = 10
```



Result of parameters Modulo_Range, Modulo_Nominator and Modulo_Denominator: The modulo range will cover one revolution of the toothed-belt wheel.

Result of parameters U_Per_Rev_Nominator and U_Per_Rev_Denominator: One scaled unit corresponds to one mm of the tooth belt.

Example: Gearbox 10.1

	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	10240	10240
Modulo_Nominator	1	1
Modulo_Denominator	1	1
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	36	360
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The Modulo_Range is equivalent to 10 motor revolutions and is 10240 increments. For the position, 1u means 1° and the resolution is 360°/10240inc = 0,035°/Inc = 1°/28,44 Inc.

Example: Gearbox 10.3

	Option1	Option2
En_Modulo	TRUE	TRUE
Modulo_Range	1024	10240
Modulo_Nominator	10	1
Modulo_Denominator	3	3
Inc_Per_R	1024	10240
U_Per_Rev_Nominator	108	1080
U_Per_Rev_Denominator	1	1
Max_Rpm	3000	300

The two options above describe exactly the same configuration. The gearbox is 10:3, so the Modulo_Range is equivalent to 1024*10/3 = 3413 + 1/3 increments. For the first option, the resulting modulo range is calculated 1024*10/3, for option2, it is 10240*1/3. For the position, 1u means 1° and the resolution is 108°/1024inc = 0,105°/Inc = 1°/9.481 Inc.

Scaling of the unit of length

Inc_Per_R With this parameter the number of the drive position counts each revolution of the motor (DINT) have to be entered.

U_Per_Rev_Denominator & U_Per_Rev_Nominator With these two parameters the number of units which correspond to one revolution of the motor have to be entered.

The units of length can be scaled to values like: mm, inch, degree, …

All dynamic parameters of the PLCopen function blocks like velocity, acceleration and jerk are based on seconds. Velocity [units/s], acceleration [units/s²], jerk [units/s³]

Example 1

```
Inc_Per_Rev            = 10000
U_Per_Rev_Nominator     = 360
U_Per_Rev_Denominator     = 1
```

This will scale one unit to one degrees of the motor shaft. Correspondingly a velocity [units/s] of 360 will turn the motor shaft one revolution per second.

Example 2 In the example one unit will be scaled to one millimeter of the conveyor.

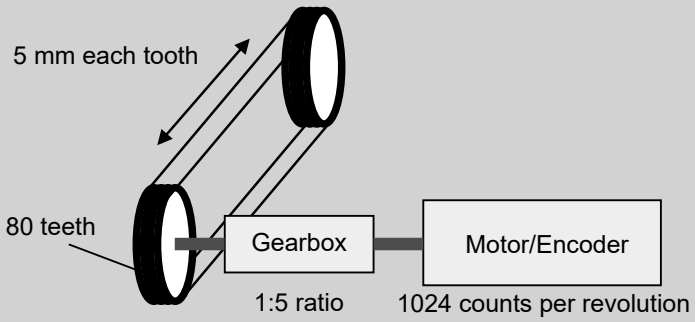


Fig. 237: Scaling units

How many units will pass after one revolution of the motor? (80*5mm) / 5 = 80

```
Inc_Per_Rev            = 1024
U_Per_Rev_Nominator     = 80
U_Per_Rev_Denominator     = 1
```

Example 3 In the example one unit will be scaled to one millimeter of the conveyor.

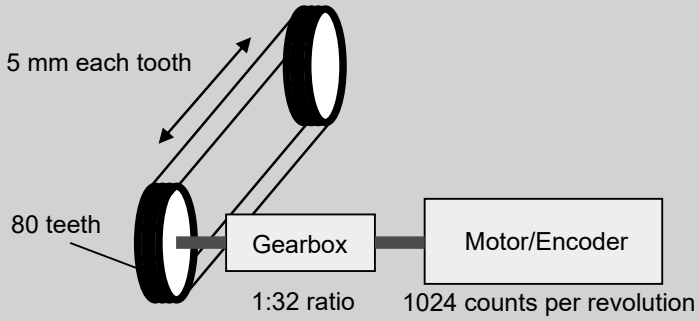


Fig. 238: Scaling units

How many units will pass after one revolution of the motor? (80*5mm) / 32 = 12,5 = 125 / 10

```
Inc_Per_Rev             = 1024
U_Per_Rev_Nominator      = 125
U_Per_Rev_Denominator      = 10
```

Scaling of the speed reference output

These two parameters are used to scale Speed Reference output of the kernel FB in order to reach the intended velocity by the output value and to limit the highest possible output value.

Ref_Max Highest possible output value of the Speed Reference output. The Speed Reference value that corresponds to the parameter Max_Rpm should be used.

Max_Rpm Maximum speed of the motor in revolutions per minute.

Example

- Analog Drive: 1000 rpm at 2 Volts, 3200 rpm at 6,4 Volts (max.)
- Analog output module: 10 Volts output at digital value 27648
- Ref_Max = 17695 (= 27648 / 10 * 6,4)
- Max_Rpm = 3200

Access and modify parameters

All modifications will be effective immediately. There is no extra plausibility check and values are not checked for limitations.

Use this functionality with care.

Some parameters are collected inside a structure in Axis_Ref, and can be accessed and modified immediately. They are the same parameters as used with function blocks MC_Write-Parameter and MC_ReadParameter *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*.

The differences are:

- Only available with CMC_Motion_Kernel_Real
- The parameter values are LREAL instead of DINT and can be used with decimals.
- The parameters will be effective immediately.
- There is no check for consistency or limits.
- The parameters for position control can be checked and modified by accessing the structure Axis_Parameter.CMC_Pos_Control in addition.

Parameter for position control	Description
KP	Proportional gain in positive direction. Used directly to multiply the following error and create the Reference_Prop.
KF	Feed forward in positive direction. Used directly to multiply the speed reference and create the Reference_FF.
KP_BACK	Proportional gain in negative direction. Used directly to multiply the following error and create the Reference_Prop.
KF_BACK	Feed forward in negative direction. Used directly to multiply the speed reference and create the Reference_FF.
TI	Integration time. When parameter is used the position control loop has an additional integral part. In TI cycle, the Reference_ITG will reach the value of Reference_Prop, when KI=100*KP.
KI	Proportional gain, used for integral part of position control loop.
KF_100	Value for feed forward gain, if 100% would be used.
Max_Time	Delay time used for supervision of velocity. With Max_Time=0, no supervision is executed.
D_XS_Max	Maximum possible velocity in [u/cycle]. The maximum allowed following error is part of the parameter structure, PLCopen parameter paraMaxPositionLag.
Ref_Max	Limit for Speed_Reference.

Element actual of Axis_Ref

The element `actual` represents actual values from inside the position control loop.

Value	Description
Position	Actual position in [u] to control the axis.
Control_Position	Reference position in [u] which is actually used for control loop.
D_XS	Distance in [u] to be moved per cycle.
D_XSS	Following error in [u].
Reference_Prop	Proportional part for Speed_Reference.
Reference_FF	Feed forward part for Speed_Reference.
Reference_ITG	Integral part for Speed_Reference.

Possible to use different gain for forward/ backward movement , possible improvement for hydraulic axis or vertical movement

See parameter KP/KP_BACK and KF/KF_BACK.

Limitation for velocity and acceleration and deceleration

From library version 3.1 on, these values are not limited to the 16-bit range of values (32767). The limit for velocity is calculated by the values given at CMC_Axis_Control_Parameter_Real and the acceleration is limited such that this velocity can not be reached faster than 1 cycle.

1.5.9.4.4 Programming guidelines

To achieve the best results for Motion Control the actual position has to be transferred in best possible quality (with minimal jitter) to the PLC. The position feedback is expected to be in increments as the data type is a DINT.

The kernel function block (CMC_Motion_Kernel_Real or CMC_Motion_Kernel_Int) has to be called every cycle and its task requires a fixed cycle time.

A variable of type Axis_Ref is used to connect to the PLCopen function blocks and their kernel function block.

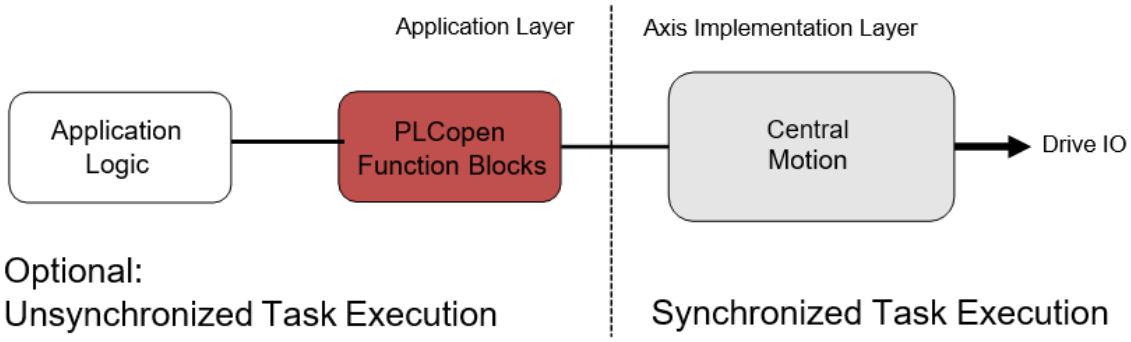
The function block CMC_Axis_Control_Parameter_Real or CMC_Axis_Control_Parameter_Int has be used for the axis configuration. *Chapter 1.5.9.4.3 “Axis parameters” on page 2642*

The signal of the limits switches and the absolute switch should be connected to the elements of the data type CMC_Axis_IO. The signal of the absolute switch must be TRUE in case the axis hits the sensor. The signal of a corresponding limit switch has to be true when the axis leaves the area surrounded by the limit switches. If needed the signal has to be inverted before it is connected to the elements of the data type.

Task configuration

The kernel function block and the transfer of axis IO data should be processed in a cyclic task. This task should be as short and real-time as possible to achieve the best motion control performance. Always make sure Kernel function block is called at the highest priority task and other applications must be at a lower priority task.

In order to save PLC processing time the most PLCopen function blocks as well as the application logic can also be processed in a task which runs on a lower priority than the real-time task with the axis implementation as shown in the figure below.



All PLCopen function blocks which must be called in the same task than the kernel function block:

- MC_CombineAxes
- MCA_MoveByExternalReference
- MCA_SetCoordinateTransformation
- MCA_SetDynamicFollower
- MCA_SyncInfeedToPath
- MCA_SyncCamToPath
- MC_SetCoordinateTransform
- MC_SetCartesianTransform (only if the transformation will be changed during run time)
- MC_SyncAxisToGroup

In case the position reference is transferred to the drive the task of the axis implementation should be synchronized to the fieldbus cycle. The following figures show an example for EtherCAT:

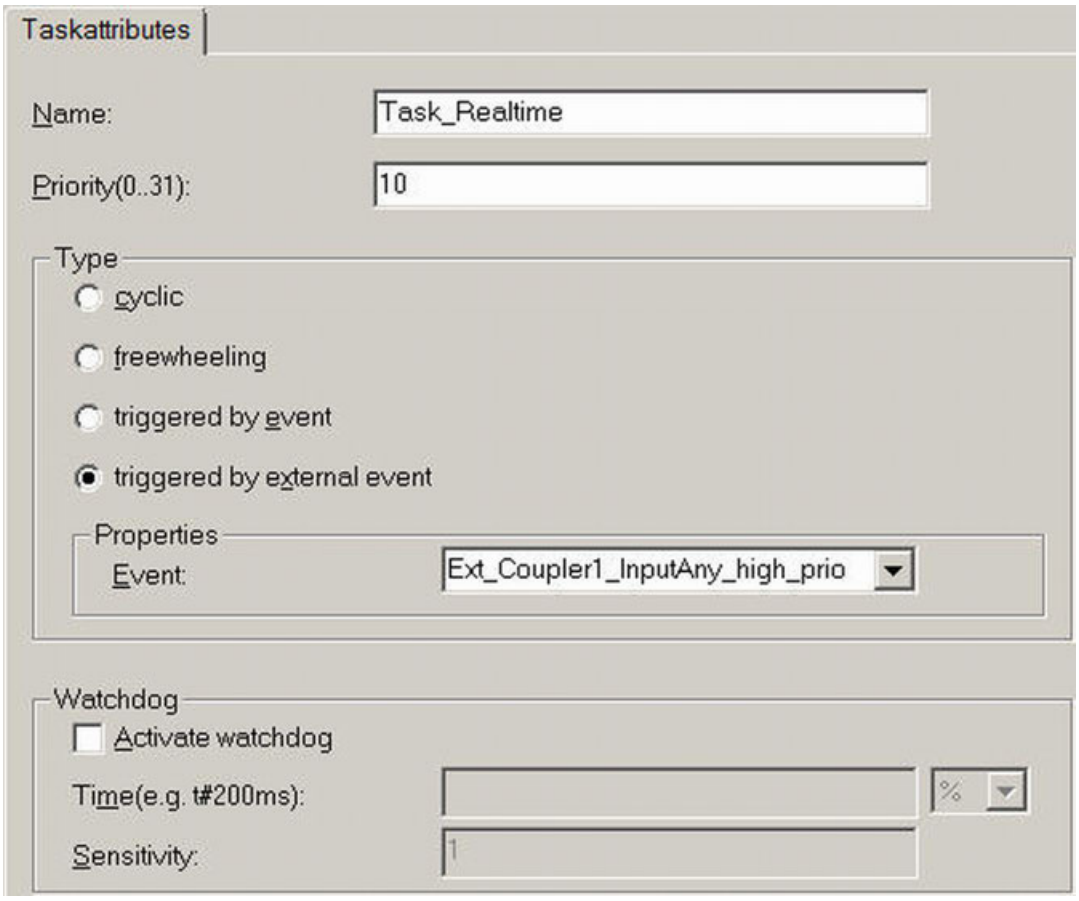


Fig. 239: Task of axis layer

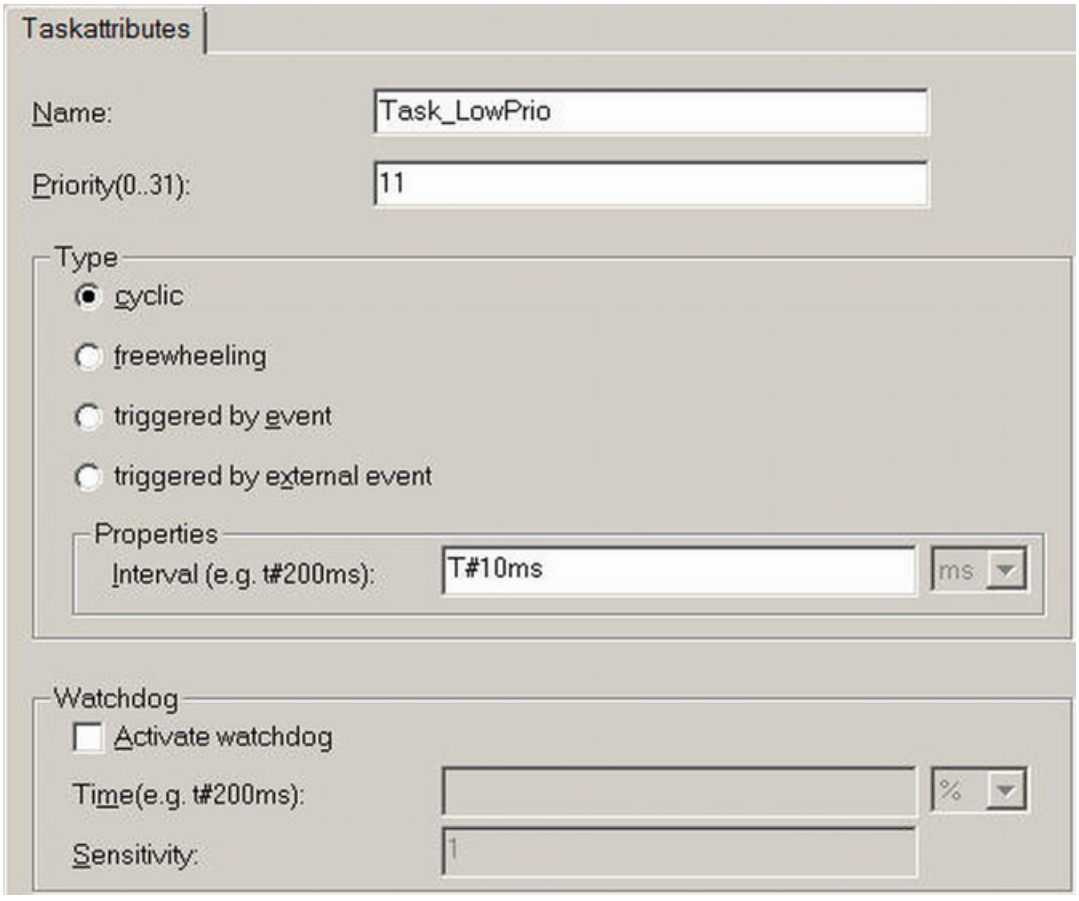
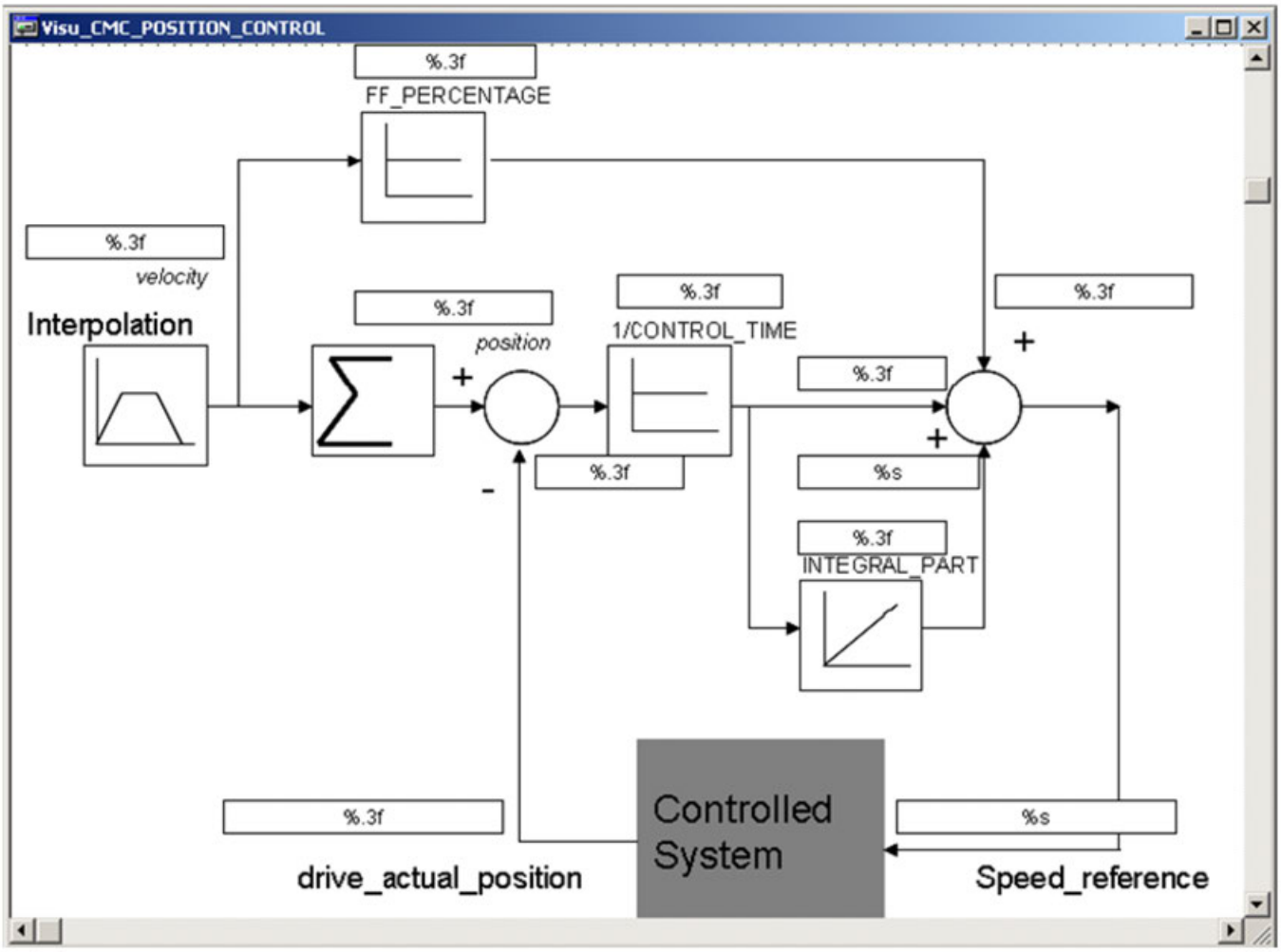


Fig. 240: Task of application implementation

1.5.9.4.5 Visualization

The structure of the position control loop is also as visualization element Visu_CMC_POSITION_CONTROL. included in CompactMotionControl_AC500_V21.lib. As placeholder, an instance of CMC_Motion_Kernel_Real or CMC_Motion_Kernel_Int has to be used. The visualization shows all numbers as they are really used inside the block, the adjustment for different resolution or cycle times is already included.



1.5.9.4.6 ABB specific data structures

Not all data structures are defined by PLCopen. Some specific structures are described in the following chapter. In addition to the data in these arrays, the movement is modified by offset and scaling values at the respective function block. These offset and scaling values (except the time-scale) are transferred continuously. This will allow to follow a "Moving Target" by adjusting these values.

PositionPositionProfile

The data type MC_PProfile is used for CamTable. An array has to be defined and provided at MC_CamTableSelect. Several CamTables could be defined and the axis could change between them on the fly. There is no routine of smooth movement from one table to the next so the user has to take care just to switch on appropriate positions. Details are described in the documentation included with the library.

Declaration example CAM_table

```
ARRAY[1..3] OF MC_PProfile:=
    (Master_position:=    0    ,interpolation_point :=
0    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 ),
(Master_position:=    50    ,interpolation_point :=
25    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 ),
(Master_position:=    100    ,interpolation_point :=
0    ,Velocity_ratio:= 0    ,Acceleration_ratio:= 0 );
```

PositionTimeProfile

This structure is used for time based profiles, e.g. MC_PositionProfile:

Interpolation types for profiles

The curves defined by an array of MC_PProfile hold master position points and according slave positions. When the master position is between 2 points, the according position for the slave is interpolated. Different types of interpolation are possible. The type is defined in MC_ABB_iTypes_Enum . The master could be a real axis or some virtual axis which could be created by just writing values for position and velocity to the Axis_Master variable as shown in the example. The same interpolation types could be used on MC_TProfile.

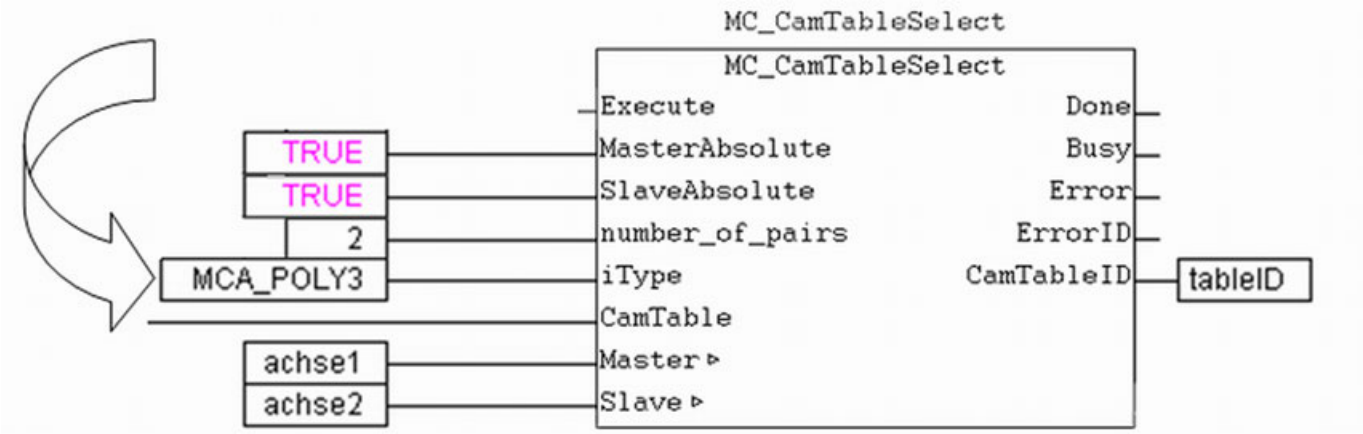


Table 168: Overview of different interpolations

Interpolation Types	Results in	Requires
MCA_LINEAR	Linear interpolation with constant velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_SPLINE_NATURAL	Cubic spline interpolation without jerk.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point
MCA_SPLINE_COMPLETE	Cubic spline interpolation without jerk, start and end of profile with velocity=0.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point

Interpolation Types	Results in	Requires
MCA_POLY3	Polynomial interpolation with linear velocity between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio
MCA_POLY5	Polynomial interpolation with linear acceleration between interpolation points.	profile.MC_PProfile_Array[x].master_position, profile.MC_PProfile_Array[x].interpolation_point, profile.MC_PProfile_Array[x].velocity_ratio, profile.MC_PProfile_Array[x].acceleration_ratio

The interpolations allow to run on smooth curves without the need to define a large number of points. The following chapter shows the results with different interpolation modes for a sinus-curve with 10 interpolation points. The following table gives the mean deviation.

Interpolation Type	Mean deviation [ppm]
MCA_LINEAR	19686 =1.9%
MCA_SPLINE_NATURAL	151=0.0151%
MCA_SPLINE_COMPLETE	25510=2.5%
MCA_POLY3	131=0.0131%
MCA_POLY5	0.37

The original curve is represented by y_sinus for position and v_sinus for velocity. The diagrams show the result which is achieved by different interpolation types.

MCA_LINEAR

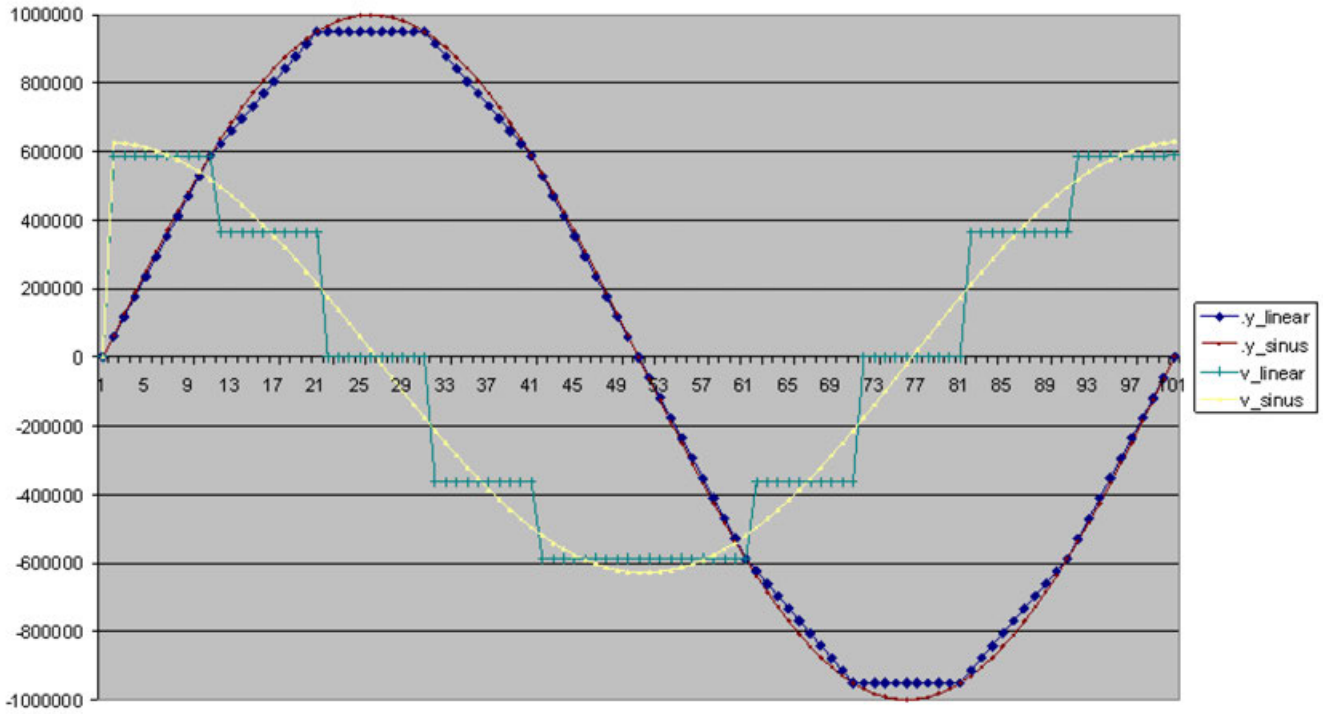


Fig. 241: Results from linear interpolation

The velocity is constant between the interpolation points.

MCA_POLY3

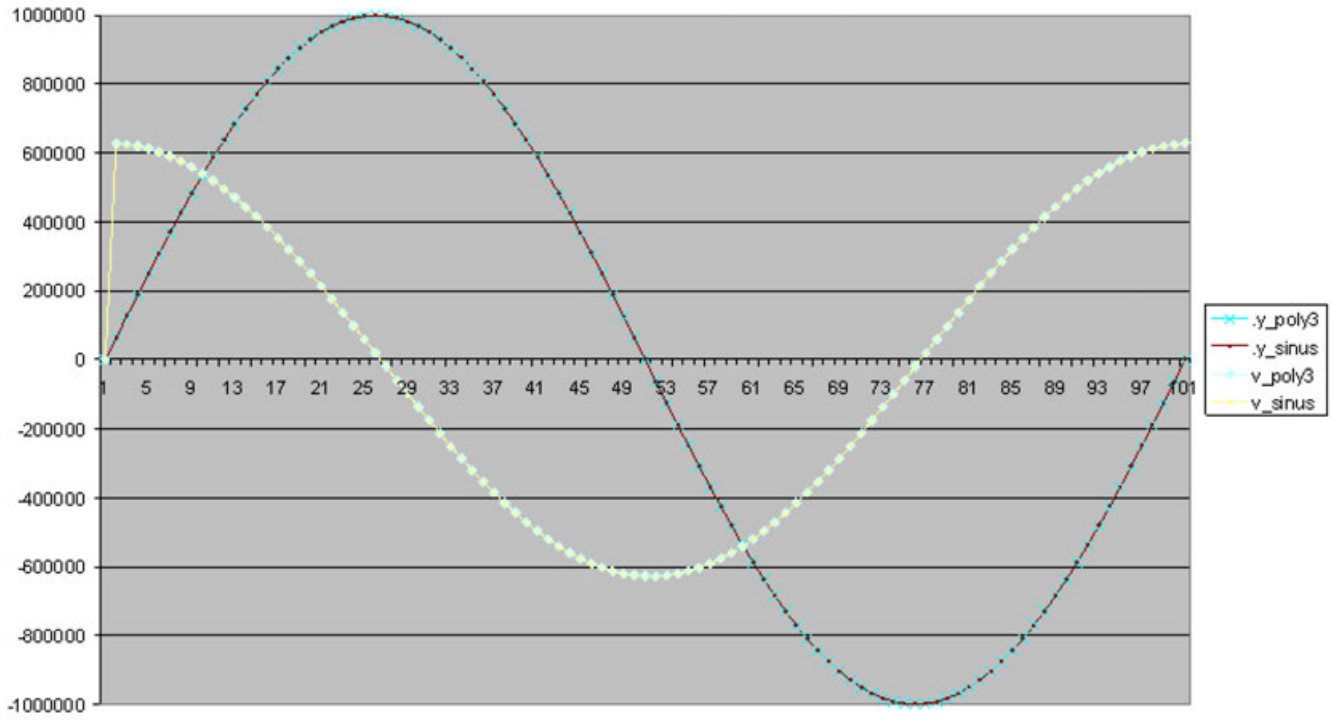


Fig. 242: Results from polynomial interpolation

The result looks almost identical to the original curve. The mean deviation shows that MCA_POLY3, MCA_POLY5 and MCA_SPLINE_NATURAL produce results which follow the original curve really good and are almost identical. The spline interpolation produces a jerk-free curve without the need of providing velocity values and acceleration values in advance.

MCA_COMPLETE

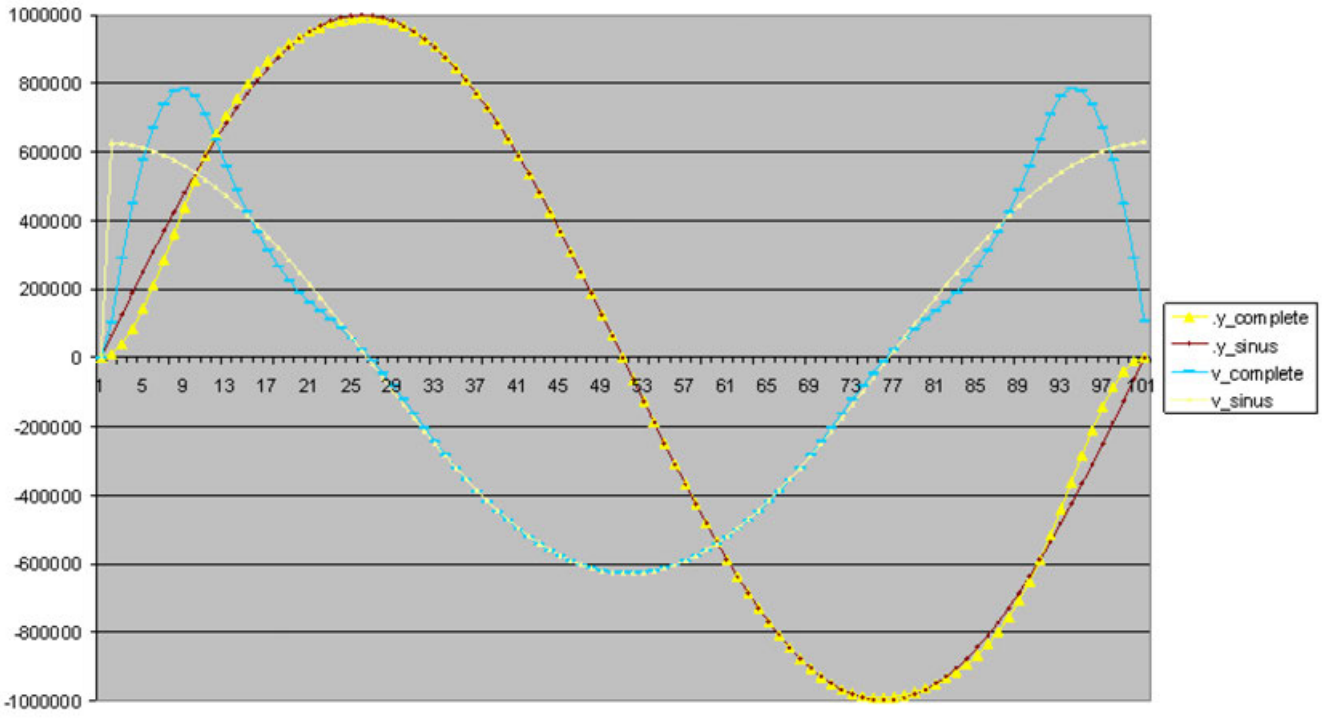


Fig. 243: Results from complete spline interpolation

In the beginning and the end, the curve does not follow the original curve. The reason is that it starts with velocity=0 and produces a jerk free result.

So the favoured result has to be considered in advance to choose the right interpolation method. With these different methods it is not necessary to provide a large number of interpolation points to get good results and smooth acceleration and deceleration ramps.

1.5.9.4.7 Appendix

List of all PLCopen and ABB specific function blocks in PS552-MC (for V2 PLC) and PS5611-Motion (for V3)

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
1	PLCopen	MC_Power	x	x
2	PLCopen	MC_Home	x	-
3	PLCopen	MC_Stop	x	x
4	PLCopen	MC_Halt	x	x
5	PLCopen	MC_MoveAbsolute	x	x
6	PLCopen	MC_MoveRelative	x	x
7	PLCopen	MC_MoveAdditive	x	x
8	PLCopen	MC_MoveSuperImposed	x	x
9	PLCopen	MC_HaltSuperImposed	x	x
10	PLCopen	MC_MoveVelocity	x	x
11	PLCopen	MC_MoveContinuousAbsolute	x	x
12	PLCopen	MC_MoveContinuousRelative	x	x
13	PLCopen	MC_PositionProfile	x	x
14	PLCopen	MC_VelocityProfile	x	x
15	PLCopen	MC_AccelerationProfile	x	x
16	PLCopen	MC_SetPosition	x	x
17	PLCopen	MC_SetOverride	x	x
18	PLCopen	MC_ReadParameter	x	x
19	PLCopen	MC_ReadBoolParameter	x	x
20	PLCopen	MC_WriteBoolParameter	x	x
21	PLCopen	MC_WriteParameter	x	x
22	PLCopen	MC_ReadActualPosition	x	x
23	PLCopen	MC_ReadActualVelocity	x	x
24	PLCopen	MC_ReadStatus	x	x
25	PLCopen	MC_ReadAxisError	x	x
26	PLCopen	MC_Reset	x	x
27	PLCopen	MC_CamTableSelect	x	x

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
28	PLCopen	MC_CamIn	x	x
29	PLCopen	MC_CamOut	x	x
30	PLCopen	MC_GearIn	x	x
31	PLCopen	MC_GearOut	x	x
32	PLCopen	MC_GearInPos	x	x
33	PLCopen	MC_PhasingAbsolute	x	x
34	PLCopen	MC_PhasingRelative	x	x
35	PLCopen	MC_HaltPhasing	-	x
36	PLCopen	MC_LoadControl	-	x
37	PLCopen	MC_LimitLoad	-	x
38	PLCopen	MC_LimitMotion	-	x
39	PLCopen	MC_LoadSuperImposed	-	x
40	PLCopen	MC_LoadProfile	-	x
41	PLCopen	MC_TorqueControl	-	x
42	ABB	MCA_CamInDirect	x	x
43	ABB	MCA_CamInfo	-	x
44	ABB	MCA_Cam_Extra	x	x
45	ABB	MCA_DriveBasedHome	x	x
46	ABB	MCA_GearInDirect M	x	x
47	ABB	CA_Indexing	x	x
48	ABB	MCA_JogAxis	x	x
49	ABB	MCA_MoveByExternalReference	x	x
50	ABB	MCA_MoveVelocityContinuous	x	x
51	ABB	MCA_MoveRelativeOpti	x	x
52	ABB	MCA_Parameter	x	x
53	ABB	MCA_PhasingbyMaster	-	x
54	ABB	MCA_ReadParameterList	x	x
55	ABB	MCA_SetOperatingMode	x	x
56	ABB	MCA_SetPositionContinuous	x	x
57	ABB	MCA_WriteParameterList	x	x
58	ABB	MCA_CamGetInterpolationPosition	-	x
59	ABB	MCA_Home	x	-
60	ABB	MCA_Power	x	-
61	ABB	ECAT_402Parameter-Homing_APP	x	x

SNo	Funktion block type	Funktion block name	Motion Library V2 (PS552-MC)	Motion Library V3 (PS5611-Motion)
62	ABB	ECAT_HomingOnTouchP-robe_APP	x	x
63	ABB	ECAT_CiA402_TouchP-robe_App	x	x

PLCopen Part 4 –Coordinated Motion is only available for V2 PLC and not yet available for V3 PLC.

1.5.9.4.8 Function blocks for central motion control implementation

CMC_MOTION_KERNEL_REAL

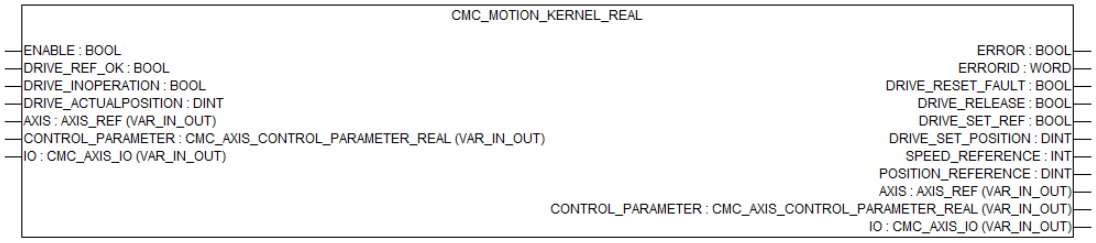


Fig. 244: Function block CMC_MOTION_KERNEL_REAL

Table 169: General information

Available as of runtime system	V1.2
Included in library	CompactMotionControl_AC500_V21.LIB
Type	Function block with historical values

The kernel function block is the fundamental part of the Central Motion Control axis implementation named Compact Motion. It performs floating point artithmetic for all calculations. More detailed information about this function block and the use of it: *Chapter 1.5.9.4 “PLC-based motion control” on page 2615*

Input description

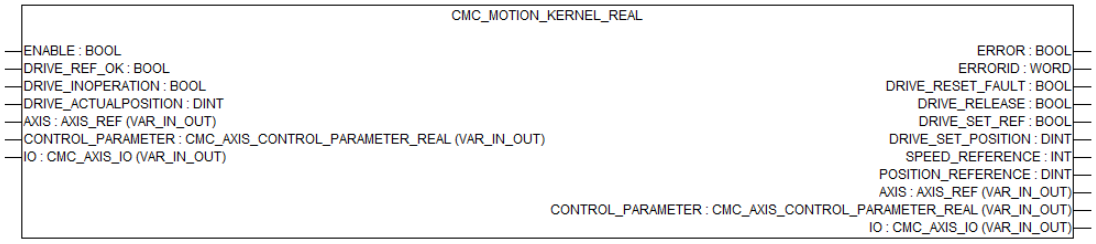


Fig. 245: Function block CMC_MOTION_KERNEL_REAL

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

If the function block is stored in a RETAIN memory area, the connected AXIS has to be RETAIN.

ENABLE Data type: BOOL

Release of function block. Enable has to be set before new control parameters are released by CMC_AXIS_CONTROL_PARAMETER.

DRIVE_REF_OK Data type: BOOL

Indication for homing.

DRIVE_INOPERATION Data type: BOOL

Indication that drive is running.

DRIVE_ACTUALPOSITION Data type: DINT

Actual position in increments.

Axis Data type: AXIS_REF

Reference to the axis.

CONTROL_PARAMETER Data type: CMC_AXIS_CONTROL_PARAMETER

Parameters for configuration and adjustment of the control loop.Fig. 233

IO Data type: CMC_AXIS_IO

By the structure IO (CMC_AXIS_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_AXIS_IO and to assign the inputs.

Output description

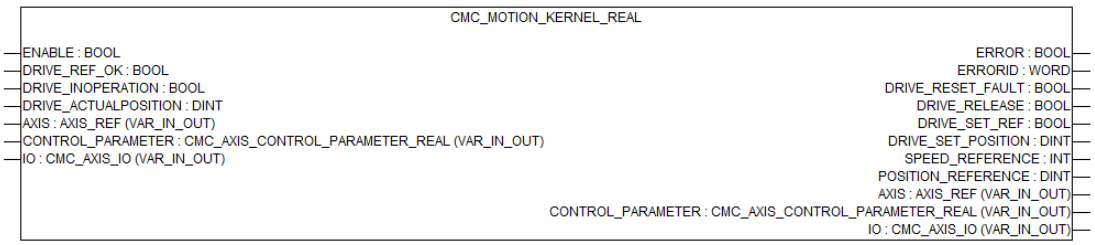


Fig. 246: Function block CMC_MOTION_KERNEL_REAL

Error Data type: BOOL

Signals that an error has occurred within the function block.

ERRORID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

The error codes ERRORID also sets the output-bit ERROR=TRUE and sets the axis in state ERROR_STOP. To allow a new movement the error codes ERRORID require that either the axis is disabled/enabled by MC_Power or the error reset is disabled/enabled by MC_Reset.

The error codes ErrorID_WARNING will not set ERROR=TRUE, and will not set the axis to ERROR_STOP. The error codes ErrorID_WARNING do not require the MC_Reset or MC_Power. It is possible the axis is stopped and ongoing motion is aborted by a WARNING.

The value will be shown until:

- An other error or warning occurs
- MC_Reset or MC_Power is used

ErrorCode	Value	Description
ErrorID_POSITION_FOLLOW	1	The position lag was to large (parameter POS_LAG_PERCENTAGE) or the velocity had a wrong value by 50% for a certain time (parameter V_CHECK_TIME).
ErrorID_POSSW	2	The actual position did exceed the positive Software limit switch position.This supervision has to be activated with MC_WriteParameter.
ErrorID_NEGSW	3	The actual position did exceed the negative Software limit switch position. This supervision has to be activated with MC_WriteParameter.
ErrorID_VELOCITY_FAULT	4	The measured velocity and commanded velocity are > 50% (related to maximum velocity) apart.
ErrorID_INTERPOLATION_FAULT	5	Position following error occurred, but reason most likely a interpolation problem, not drive problem (e.g. CAM Table, position step).
ErrorID_WARNING_POSITIONOVERRUN	13	A linear axis has a 32bit position overrun (configure modulo instead).
ErrorID_WARNING_VELOCITYLIMIT	10	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity.
ErrorID_WARNING_POSITIONLIMITPOS	11	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity (MaxVelocityAppl, MaxDecelerationAppl) Position is in limitation towards position limit (SWLimit2DecPos) .
ErrorID_WARNING_POSITIONLIMITNEG	12	Position is in limitation towards position limit (SWLimit2DecNeg).
ErrorID_WARNING_ABORT	14	Axis has been aborted due to too large position gap in velocity limit.

DRIVE_RESET_FAULT Data type: BOOL

Binary signal to be used for resetting the drive error, if applicable.

DRIVE_RELEASE Data type: BOOL
Activate the drive.

DRIVE_SET_REF Data type: BOOL
Activate homing.

DRIVE_SET_POSITION Data type: DINT
Position to be used at homing.

SPEED_REFERENCE Data type: DINT
Reference value for the drive.

POSITION_REFERENCE Data type: DINT
Position reference value for the drive in increments.

CMC_MOTION_KERNEL_INT

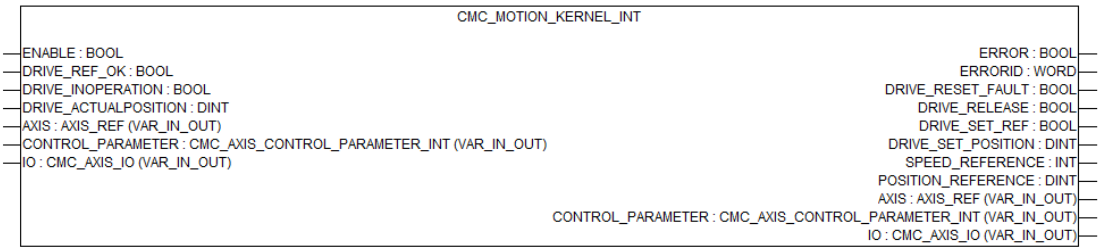


Fig. 247: Function block CMC_MOTION_KERNEL_INT

Table 170: General information

Available as of runtime system	V1.2
Included in library	CompactMotionControl_AC500_V21.LIB
Type	Function block with historical values

The kernel function block is the fundamental part of the Central Motion Control axis implementation named Compact Motion. It performs integer based calculations which use less CPU processing time on the following PLCs: eCo, PM57x, PM58x. As a result of the integer based calculations the available motion control functions are limited. More detailed information about this function block and the use of it: *Chapter 1.5.9.4 “PLC-based motion control” on page 2615*

Input description

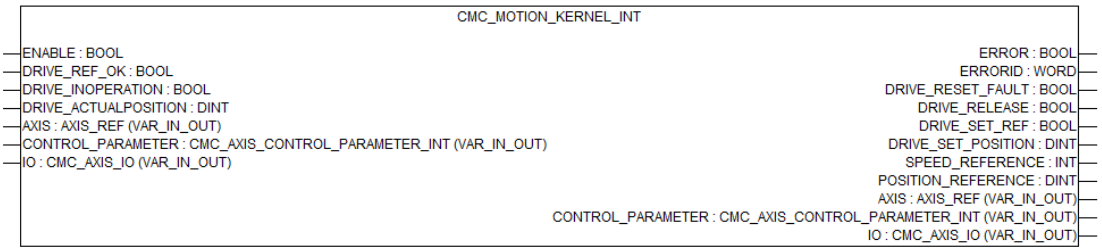


Fig. 248: Function block CMC_MOTION_KERNEL_INT

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

ENABLE

Data type: BOOL

Release of function block. Enable has to be set before new control parameters are released by CMC_AXIS_CONTROL_PARAMETER.

DRIVE_REF_OK

Data type: BOOL

Indication for homing.

DRIVE_INOPERATION

Data type: BOOL

Indication that drive is running.

DRIVE_ACTUAL POSITION

Data type: DINT

Actual position in increments.

Axis

Data type: AXIS_REF

Reference to the axis.

CONTROL_PARAMETER

Data type: CMC_AXIS_CONTROL_PARAMETER

Parameters for configuration and adjustment of the control loop.Fig. 233

IO

Data type: CMC_AXIS_IO

By the structure IO (CMC_AXIS_IO), some binary inputs are provided. The PLC program has to define a variable of type CMC_AXIS_IO and to assign the inputs.

Output description

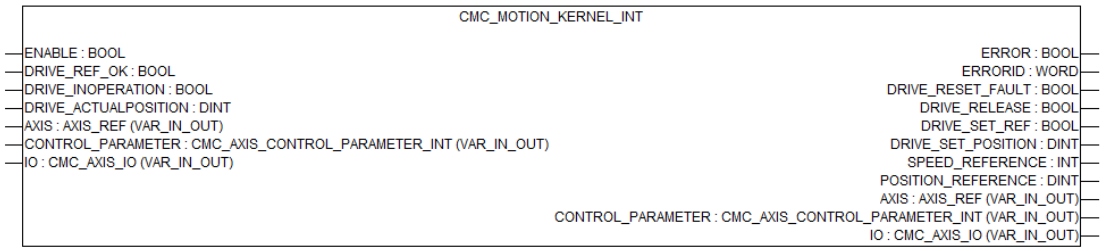


Fig. 249: Function block CMC_MOTION_KERNEL_INT

Error Data type: BOOL

Signals that an error has occurred within the function block.

ERRORID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

The error codes ERRORID also sets the output-bit ERROR=TRUE and sets the axis in state ERROR_STOP. To allow a new movement the error codes ERRORID require that either the axis is disabled/enabled by MC_Power or the error reset is disabled/enabled by MC_Reset.

The error codes ErrorID_WARNING will not set ERROR=TRUE, and will not set the axis to ERROR_STOP. The error codes ErrorID_WARNING do not require the MC_Reset or MC_Power. It is possible the axis is stopped and ongoing motion is aborted by a WARNING.

The value will be shown until:

- An other error or warning occurs
- MC_Reset or MC_Power is used

ErrorCode	Value	Description
ErrorID_POSITION_FOLLOW	1	The position lag was to large (parameter POS_LAG_PERCENTAGE) or the velocity had a wrong value by 50% for a certain time (parameter V_CHECK_TIME).
ErrorID_POSSW	2	The actual position did exceed the positive Software limit switch position.This supervision has to be activated with MC_WriteParameter.
ErrorID_NEGSW	3	The actual position did exceed the negative Software limit switch position. This supervision has to be activated with MC_WriteParameter.
ErrorID_VELOCITY_FAULT	4	The measured velocity and commanded velocity are > 50% (related to maximum velocity) apart.
ErrorID_INTERPOLATION_FAULT	5	Position following error occurred, but reason most likely a interpolation problem, not drive problem (e.g. CAM Table, position step).
ErrorID_WARNING_POSITIONOVERRUN	13	A linear axis has a 32bit position overrun (configure modulo instead).
ErrorID_WARNING_VELOCITYLIMIT	10	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity.
ErrorID_WARNING_POSITIONLIMITPOS	11	Velocity or acceleration/deceleration are in limitation, set by parameter EnableLimitVelocity (MaxVelocityAppl, MaxDecelerationAppl) Position is in limitation towards position limit (SWLimit2DecPos) .
ErrorID_WARNING_POSITIONLIMITNEG	12	Position is in limitation towards position limit (SWLimit2DecNeg).
ErrorID_WARNING_ABORT	14	Axis has been aborted due to too large position gap in velocity limit.

DRIVE_RESET_FAULT Data type: BOOL

Binary signal to be used for resetting the drive error, if applicable.

DRIVE_RELEASE Data type: BOOL

Activate the drive.

DRIVE_SET_REF Data type: BOOL
Activate homing.

DRIVE_SET_POSITION Data type: DINT
Position to be used at homing.

SPEED_REFERENCE Data type: DINT
Reference value for the drive.

POSITION_REFERENCE Data type: DINT
Position reference value for the drive in increments.

CMC_AXIS_CONTROL_PARAMETER_REAL

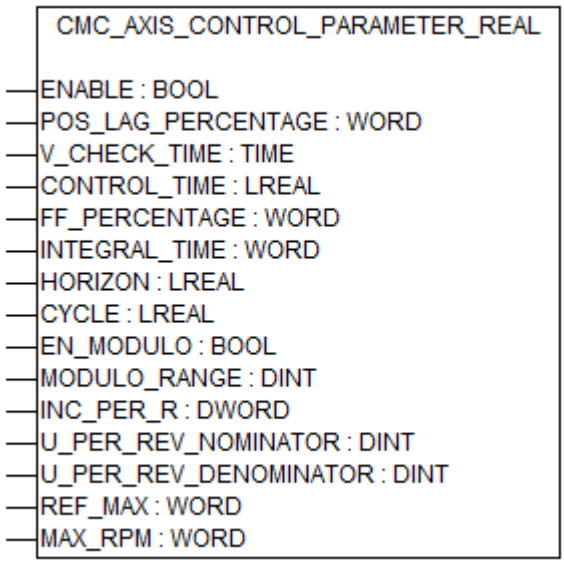


Fig. 250: Function block CMC_AXIS_CONTROL_PARAMETER_REAL

Interface function block for axis configuration and adjustment of the control loop.

> *Enable the function block before any PLCopen-block is used.*

To change parameters follow these steps:

1. MC_Power.ENABLE=FALSE, this will disable the axis.
2. CMC_AXIS_CONTROL_PARAMETER_REAL.ENABLE=FALSE.
3. Modify the parameter.
4. CMC_AXIS_CONTROL_PARAMETER_REAL.ENABLE=TRUE.

> *Some of these parameters result in position control loop parameters. These can also be modified directly (while CMC_AXIS_CONTROL_PARAMETER_REAL.ENABLE = TRUE) by using the axis parameter structure.*

This function block must be used in combination with the function block CMC_MOTION_KERNEL_REAL ↳ *Chapter 1.5.9.4.8.1 “CMC_MOTION_KERNEL_REAL” on page 2659*.

Functional group	Parameter
Axis supervision	POS_LAG_PERCENTAGE
	V_CHECK_TIME
Position Control Loop	CONTROL_TIME
	FF_PERCENTAGE
	INTEGRAL_TIME
	HORIZON
PLC system	CYCLE
Rollover axis	EN_MODULO
	MODULO_RANGE
Scaling units of length	INC_PER_R
	U_PER_REV_NOMINATOR
	U_PER_REV_DENOMINATOR
Scaling the Speed Reference output	REF_MAX
	MAX_RPM

Input description

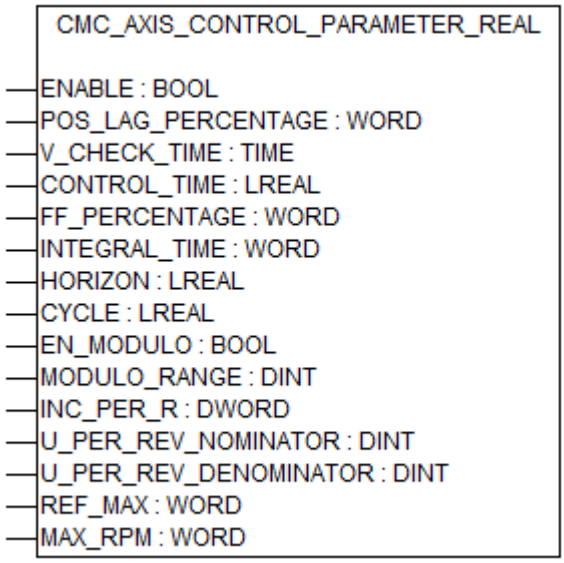


Fig. 251: Function block CMC_AXIS_CONTROL_PARAMETER_REAL

ENABLE Data type: BOOL

Enable new parameters with a positive edge.

POS_LAG_PER-CENTAGE Data type: WORD, default: 150, unit: %

Value for supervision of position difference. 100 is at least necessary to reach the maximum velocity with FF_PERCENTAGE = 0.

V_CHECK_TIME Data type: TIME, default: 100, unit: ms

Delay time for the supervision of actual velocity. With V_CHECK_TIME = 0, this supervision will be disabled.

CONTROL_TIME Data type: LREAL, default: 100, unit: ms

Determines the gain for position control loop. A lower time means a larger proportional gain for position control loop. The value means: In case FF_PERCENTAGE = 0, the drive will run CONTROL_TIME ms behind its position reference.

FF_PER-CENTAGE Data type: WORD, default: 0, unit: %

Feed-forward gain, usually should be <80%. For larger values, the parameter HORIZON needs to be used as the position will overshoot otherwise.

INTE-GRAL_TIME Data type: WORD, default: 0, unit: ms

Integration time for position control loop, 0 means no integral part is used.

HORIZON Data type: LREAL, default: 0, unit: ms

Gives a time in advance for the feed-forward. This could compensate reaction times. HORIZON > 0 requires more computing power.

CYCLE Data type: LREAL, default: 10, unit: ms

Cycle time of the PLC program.

EN_MODULO Data type: BOOL, default: FALSE

Enable rollover axis.

MODULO_RAN GE Data type: DINT, unit: increment

Distance for rollover, maximum value is 0x3FFFFFFF.

INC_PER_R Data type: DWORD, default: 1024

Increments per revolution for actual position.

U_PER_REV_N OMINATOR Data type: DINT, default: 1024

Units per revolution.

U_PER_REV_DENOMINATOR Data type: DINT, default: 1

Units per revolution.

REF_MAX Data type: WORD, default: 32767

Maximum value for SPEED_REFERENCE.

MAX_RPM Data type: WORD, default: 1500, unit: 1/m

Maximum value for rotations per minute, has to be the value which is reached at SPEED_REFERENCE = REF_MAX.

CMC_AXIS_CONTROL_PARAMETER_INT

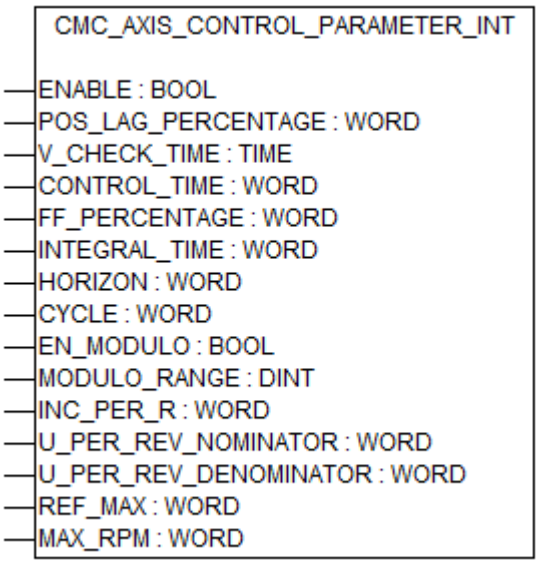


Fig. 252: Function block CMC_AXIS_CONTROL_PARAMETER_INT

Interface function block for axis configuration and adjustment of the control loop.

This function block must be used in combination with the function block CMC_MOTION_KERNEL_INT ↳ *Chapter 1.5.9.4.8.2 “CMC_MOTION_KERNEL_INT” on page 2662*.

Functional group	Parameter
Axis supervision	POS_LAG_PERCENTAGE
	V_CHECK_TIME
Position Control Loop	CONTROL_TIME
	FF_PERCENTAGE
	INTEGRAL_TIME
	HORIZON
PLC system	CYCLE
Rollover axis	EN_MODULO
	MODULO_RANGE

Functional group	Parameter
Scaling units of length	INC_PER_R
	U_PER_REV_NOMINATOR
	U_PER_REV_DENOMINATOR
Scaling the Speed Reference output	REF_MAX
	MAX_RPM

Input description

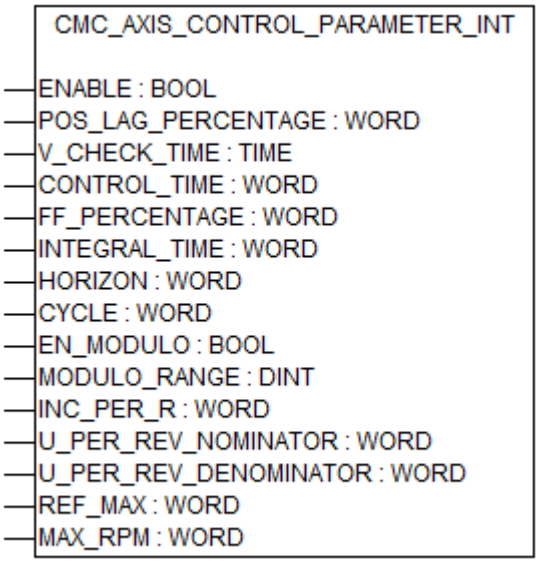


Fig. 253: Function block CMC_AXIS_CONTROL_PARAMETER_INT

ENABLE Data type: BOOL

Enable new parameters with a positive edge.

POS_LAG_PERCENTAGE Data type: WORD, default: 150, unit: %

Value for supervision of position difference. 100 is at least necessary to reach the maximum velocity with FF_PERCENTAGE = 0.

V_CHECK_TIME Data type: TIME, default: 100, unit: ms

Delay time for the supervision of actual velocity. With V_CHECK_TIME = 0, this supervision will be disabled.

CONTROL_TIME Data type: LREAL, default: 100, unit: ms

Determines the gain for position control loop. A lower time means a larger proportional gain for position control loop. The value means: In case FF_PERCENTAGE = 0, the drive will run CONTROL_TIME ms behind its position reference.

FF_PERCENTAGE Data type: WORD, default: 0, unit: %

Feed-forward gain, usually should be <80%. For larger values, the parameter HORIZON needs to be used as the position will overshoot otherwise.

INTEGRAL_TIME Data type: WORD, default: 0, unit: ms

Integration time for position control loop, 0 means no integral part is used.

HORIZON Data type: LREAL, default: 0, unit: ms

Gives a time in advance for the feed-forward. This could compensate reaction times. HORIZON > 0 requires more computing power.

CYCLE Data type: LREAL, default: 10, unit: ms

Cycle time of the PLC program.

EN_MODULO Data type: BOOL, default: FALSE

Enable rollover axis.

MODULO_RANGE Data type: DINT, unit: increment

Distance for rollover, maximum value is 0x3FFFFFFF.

INC_PER_R Data type: DWORD, default: 1024

Increments per revolution for actual position.

U_PER_REV_NOMINATOR Data type: WORD, default: 1024

Units per revolution.

U_PER_REV_DENOMINATOR Data type: WORD, default: 1

Units per revolution.

REF_MAX Data type: WORD, default: 32767

Maximum value for SPEED_REFERENCE.

MAX_RPM Data type: WORD, default: 1500, unit: 1/m

Maximum value for rotations per minute, has to be the value which is reached at SPEED_REFERENCE = REF_MAX.

CMC_AXIS_SIMU_REAL

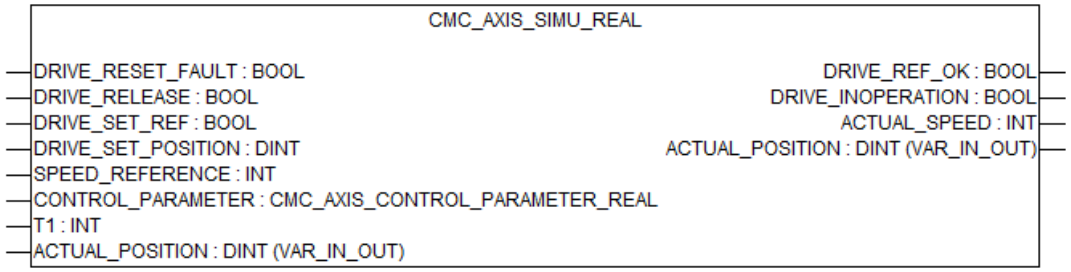


Fig. 254: Function block CMC_AXIS_SIMU_REAL

Function block to simulate a drive. This function block can be used in combination with the function block CMC_MOTION_KERNEL_REAL ↳ *Chapter 1.5.9.4.8.1 “CMC_MOTION_KERNEL_REAL” on page 2659*. Detailed information: ↳ *Chapter 1.5.9.4.2.3 “How to use the axis simulation” on page 2625*

CMC_AXIS_SIMU_INT

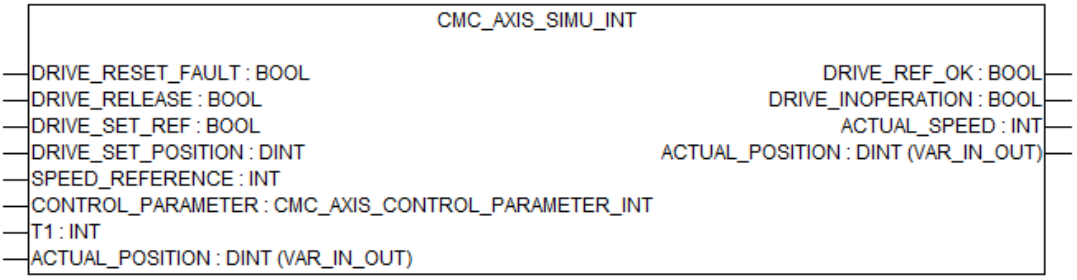
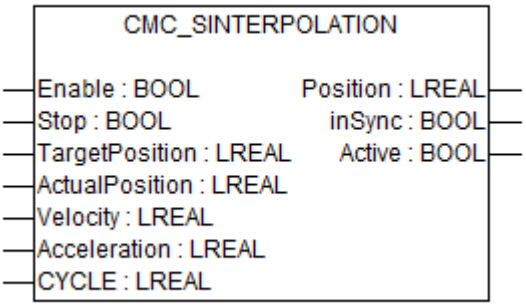


Fig. 255: Function block CMC_AXIS_SIMU_INT

Function block to simulate a drive. This function block can be used in combination with the function block CMC_MOTION_KERNEL_INT ↳ *Chapter 1.5.9.4.8.2 “CMC_MOTION_KERNEL_INT” on page 2662*. Detailed information: ↳ *Chapter 1.5.9.4.2.3 “How to use the axis simulation” on page 2625*

CMC_SInterpolation



The function block can be used for a simple interpolation. Alternatively, the function block CMC_SIPosi_Loop can be used ↳ *Chapter 1.5.9.4.8.8 “CMC_SIPosiLoop” on page 2676*.

The function block allows to create a very simple, basic axis for linear movement. The function block can be either used independent to create a basic axis with/without position control loop or it can be used in combination with PLCopen function blocks.

The function block creates a positioning interpolation towards TargetPosition and uses the given velocity and acceleration values. The funciton block has to be used within the real-time cycle, same as CMC_MOTION_KERNEL_REAL. The result is given to output Position. The TargetPosition and also Velocity and Acceleration can be changed anytime and will be used at once. With Enable = FALSE, the funciton block sets the output Position to ActualPosition, this is similar to an open loop.

Table 171: Behavior of inputs

Enable	Stop	Behavior
FALSE	-	Output Position = ActualPosition
TRUE	FALSE	Interpolates output Position to reach Target-Position with the given velocity and accelera-tion.
TRUE	TRUE	Ramps down to velocity = 0.

Table 172: Behavior of outputs

InSync	Active	Behavior
FALSE	TRUE	Function block is activated, position and velocity is not yet reached.
TRUE	FALSE	Funciton block is activated, position is reached, output Position = TargetPosition.
FALSE	FALSE	Funciton block is either disabled or stopped, velocity = 0.

Use in combination with MC_MoveByExternalReference

This way, a positioning axis can be created with modifying the parameters for positioning “on the fly”, without the need of a certain state machine to follow.

Use in combination with MCA_SetDynamicFollower

The function block will smooth the movement when a group has to follow a conveyor. It will create defined ramps to accelerate to the conveyors position and also can be used to prevent a position jump when switching between 2 conveyors or before switching of the follower.

Use to create a simple, basic positioning axis

A very basic, simple positioning axis can be created this way. Be aware that there is not additional check if the axis really follows, also no scaling for the position is included. If function block CMC_SIPosiLoop is used, it will check for position or velocity following error.

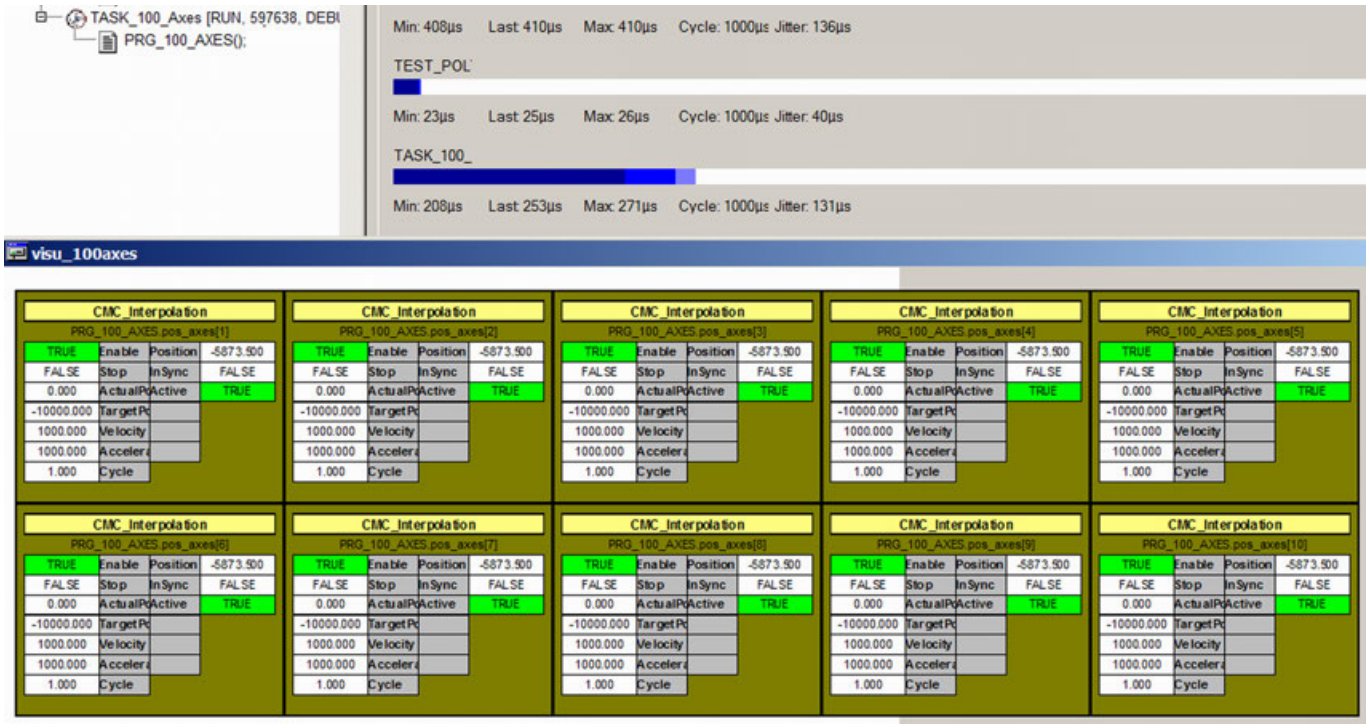
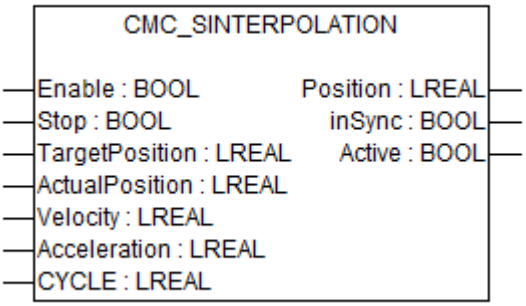


Fig. 256: Cycle time which was used for a PM590 when calculating 100 simple axes in a PLC task and priority 10.

Input description



Enable Data type: BOOL

Enables the interpolation.

Stop Data type: BOOL

TargetPosition Data type: LREAL

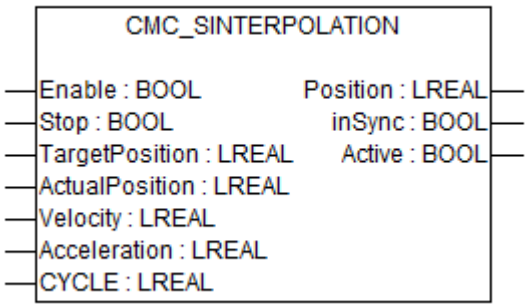
ActualPosition Data type: LREAL

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

CYCLE Data type: LREAL, default: 10, unit: ms
Cycle time of the PLC program.

Output description

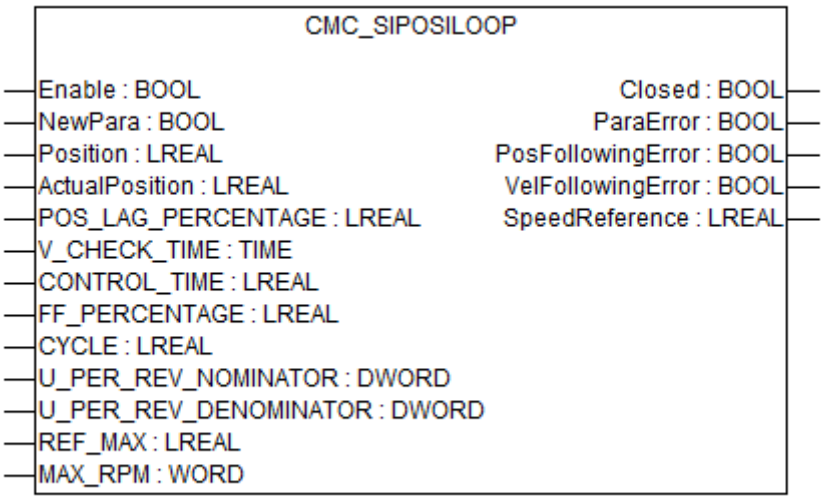


Position Data type: LREAL, unit: u
New absolute position.

InSync Data type: BOOL
Iindicates that Position reached TargetPosition.

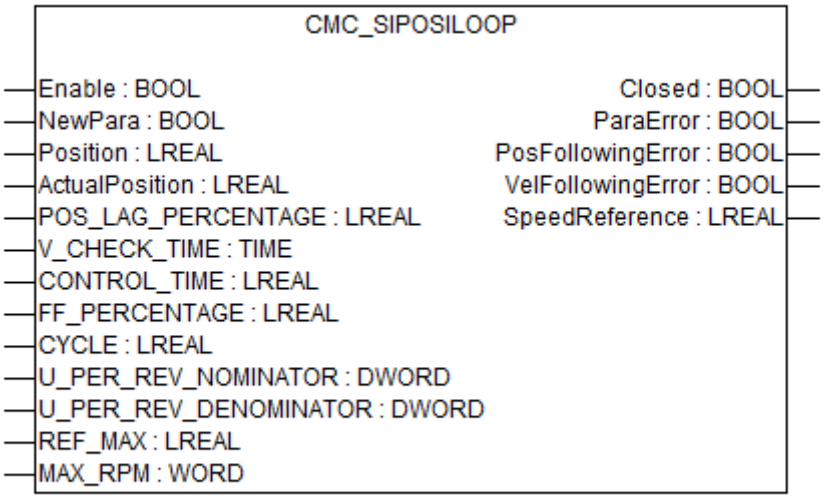
Active Data type: BOOL
Interpolation is activated.

CMC_SIPosiLoop



The function block can be used as a position control loop and can create a simple positioning axis if combined with CMC_SInterpolation. The CMC_SIPosiLoop uses same parameter setup as the parameter block CMC_AXIS_CONTROLPARAMETER_REAL.

Input description



Enable Data type: BOOL

Enable the block while TRUE, will close the position control loop.

NewPara Data type: BOOL

The block will use new input parameter with a positive edge.

Position Data type: LREAL

Reference position

ActualPosition Data type: LREAL

Actual position

POS_LAG_PER-CENTAGE Data type: WORD, default: 150, unit: %

Value for supervision of position difference. 100 is at least necessary to reach the maximum velocity with FF_PERCENTAGE = 0.

V_CHECK_TIME Data type: TIME, default: 100, unit: ms

Delay time for the supervision of actual velocity. With V_CHECK_TIME = 0, this supervision will be disabled.

CONTROL_TIME Data type: LREAL, default: 100, unit: ms

Determines the gain for position control loop. A lower time means a larger proportional gain for position con- trol loop. The value means: In case FF_PERCENTAGE = 0, the drive will run CONTROL_TIME ms behind its position reference.

FF_PER-CENTAGE Data type: WORD, default: 0, unit: %

Feed forward gain, usuallz shoul be <80%. For larger values, the paramter HORIZON needs to be used as the position will overshoot otherwise.

CYCLE Data type: LREAL, default: 10, unit: ms

Cycle time of the PLC program.

U_PER_REV_NOMINATOR Data type: DINT, default: 1024

Units per revolution.

U_PER_REV_DENOMINATOR Data type: DINT, default: 1

Units per revolution.

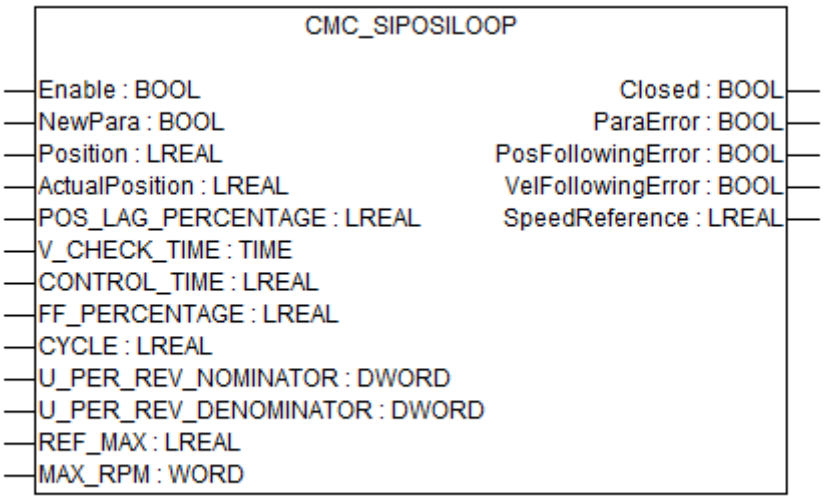
REF_MAX Data type: WORD, default: 32767

Maximum value for SPEED_REFERENCE.

MAX_RPM Data type: WORD, default: 1500, unit: 1/m

Maximum value for rotations per minute, has to be the value which is reached at SPEED_REFERENCE = REF_MAX.

Output description



Closed Data type: BOOL

Position control loop is closed, no active error.

ParaError Data type: BOOL

Indicates wrong input parameter.

PosFollowingError Data type: BOOL

Indicates position following error.

VelFollowingError Data type: BOOL

Indicates velocity following error.

SpeedReference Data type: LREAL

Speed Reference value which should be send to the drive. This value is scaled with MAX_RPM and REF_MAX: SpeedReference=REF_MAX => the axis should move with MAX_RPM.

CMC_GET_UNITS_FROM_INC

This function converts the drive's position value (DINT) which is exchanged between drive and PLC to the corresponding scaled position unit (LREAL) which is used by the PLCopen function blocks.

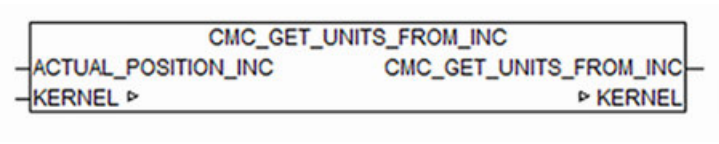


Fig. 257: CMC_GET_UNITS_FROM

Use case:

- The drive or an I/O module is used to capture an axis encoder position in relation to a binary signal (touch trigger). This position is delivered in [increments].
  If then the position is to be used in the PLCopen context, a unit position is required. It can be difficult to calculate this unit-position. Not just the scaling position units has to be considered but also the position might have experienced several correction measures. Measures like “SetPositionContinuous” or corrections due to modulo position overrun. To create the unit-value which matches a certain increment-value, the function CMC_GET_UNITS_FROM_INC has to be used.

Input:

ACTUAL_POSITION_INC Data type: DINT
A position [increments], for example captured by the drive as result for a touch trigger

KERNEL Data type: CMC_MOTION_KERNEL_REAL
Kernel block instance which belongs to the specific axis.

Result: The position [u], which describes exactly the same position as ACTUAL_POSIITON_INC, just transferred in to the axis coordinate system and delivered in [u].

1.5.9.4.9 PLCopen coordinated motion

Principles of coordinated motion

Coordinate system and kinematic transformation

The essence of a trajectory is the coordinated motion of two or more axes from a starting point to a target point via a defined path with a specified path velocity. As path one can think of a straight line, a circular movement, or via a spline function. The definition of a path -or any position information - in space requires a coordinate system. Within this specification three coordinate systems are defined:

ACS - Axis related

MCS - Machine related

PCS - Product or workpiece related

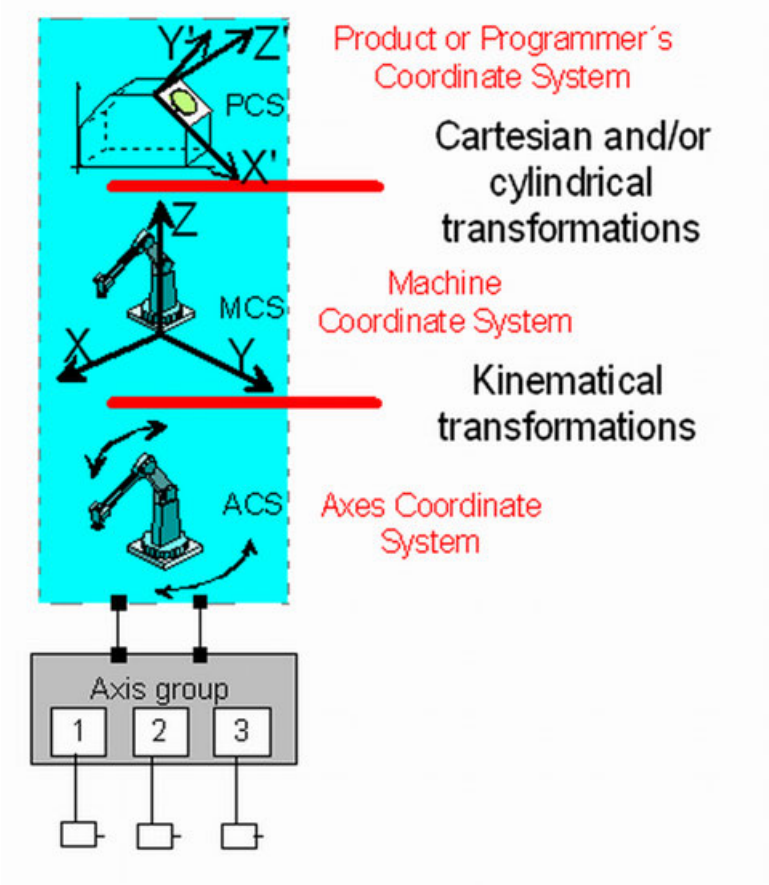


Fig. 258: Overview of the coordinate systems and transformations

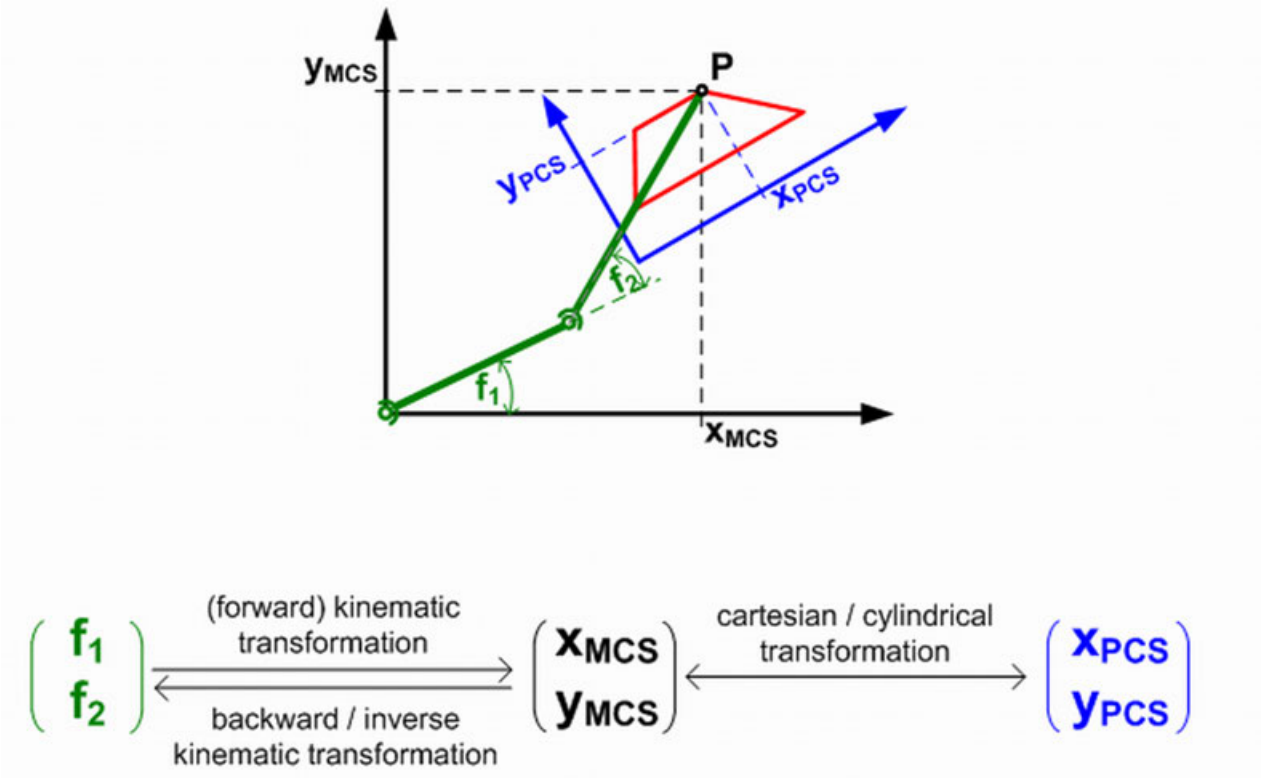


Fig. 259: Example for specifying point P in PCS, MCS or ACS assuming a SCARA robot with two rotary axes

ACS: Axes Coordinate System

Actual position of the physical axis (after homing).

MCS: Machine Coordinate System

Cartesian coordinate system with the origin is a fixed position relative to the machine. (Sometimes called “World Coordinate System” or “Base Coordinate System”). (Note: with Cartesian build machines, MCS may be identical to ACS, or mapped via a trivial transformation). The coordinate system from the physical multiple axes ACS is linked to the MCS via a kinematic transformation (forward and backward conversion).

PCS: Product Coordinate System

The real work piece can have a rotation or shift to the MCS or even might be moving relative to the MCS, and often one wants to describe the trajectory independent from the machine situation. To map these two worlds (MCS to PCS and vice versa), a cartesian or cylindrical transformation is normally done. The coordinate system of the product can be called PCS: "Product Coordinate System", or "Program Coordinate System" in CNC world. There can be more than one PCS transformation applicable at the same time. In this case the ENUM to specify the coordinate system (CS) has to be extended. A PCS can be a static or a dynamic transformation.

In order to specify a point or orientation in space a position always has to be related to a coordinate system. By means of transformations this position can be transformed to other coordinate systems. Within this specification, function blocks are defined for these transformations, hiding the complexity of these transformations to the programmer in its day to day use. All multi axes motion commands are related to only one of the coordinate systems at the same time.

Example for specifying point P in PCS, MCS or ACS assuming a SCARA robot with two rotary axes

Point P is situated on a 2D workpiece. It can be described equivalent in PCS, MCS and ACS. Point P could be specified by referring to PCS resulting in the position PPCS = (x_{PCS}, y_{PCS}). Given the shift and orientation of PCS relative to MCS, point P equivalently could be speci-fied by PMCS = (x_{MCS}, y_{MCS}). Assuming a SCARA robot with two rotary axes point P also could be described by the angles of the axes P_{ACS} = (ϕ1, ϕ2).

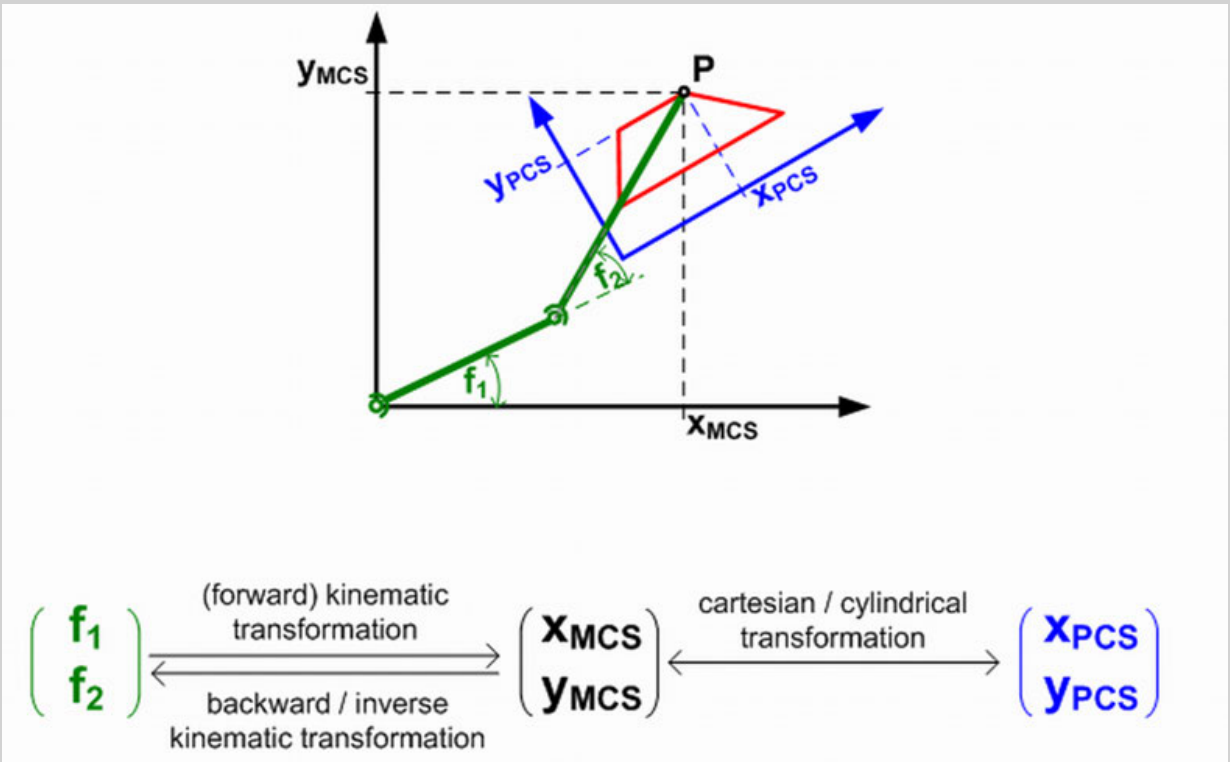


Fig. 260: Specifying point P in PCS, MCS or ACS assuming a SCARA robot with two rotary axes

Red trapezoid	Point P, situated on a 2 D workpiece
Blue	PCS
Black	MCS
Green	ACS

Kinematic transformation

Axes are connected via mechanical links providing movements of the ‘Tool Center Point’, TCP in space. TCP is a dis-tinguished point of the machine, sometimes also called ‘Point of Interest’, POI, or ‘effector’. The physical assembly of the axes and therefore the position of the TCP in MCS is described by a so called kinematic transformation. The kinematic transformation connects ACS to MCS (forward conversion). By applying the kinematic transformation on a position related to ACS, this position can be transformed into a position in MCS. The other way round, applying the inverse kinematic transformation, a position related to MCS can be transformed into a position in ACS (backward conversion).

With simple cartesian machine constructions, in which axes are directly oriented in X-, Y-, and Z-directions of MCS, the kinematic transformation can easily be specified. One just has to define which axis is in the X-direction, which in Y, and which in the Z-direction. In the simplest case ACS is identically to MCS and one needn’t distinguish between both. But in praxis there are many non-cartesian structures, like SCARA robots or Tripods, where the kinematic transformation is more complex.

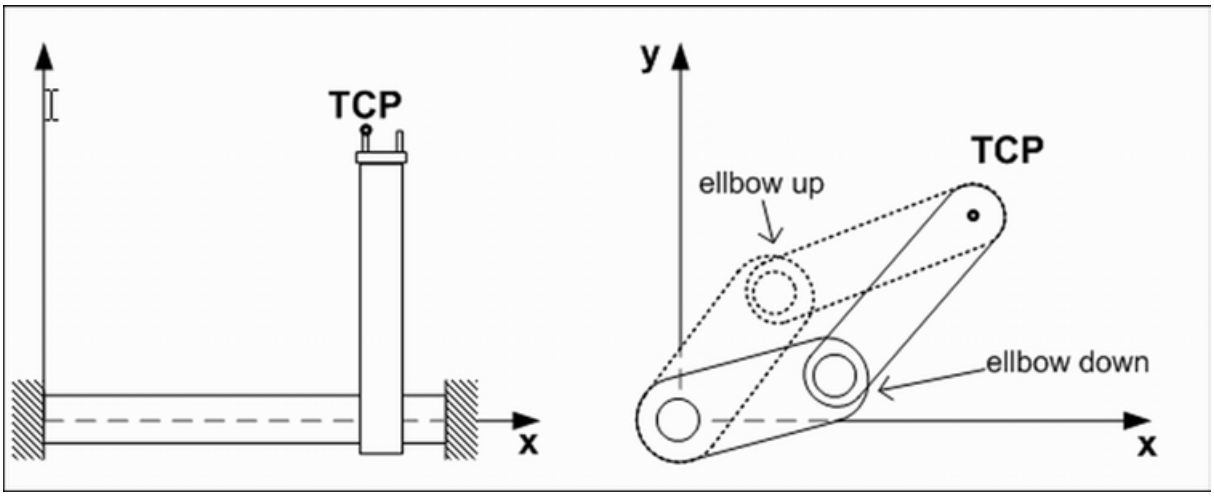


Fig. 261: Example for reaching the same position in space with a) a cartesian handling (2 linear axes) and b) a SCARA (2 rotary axes) with two possible configurations (elbow down and elbow up). (Note: the orientation is fixed in both examples)

Above example demonstrates how a position in space could be reached by a cartesian handling or a SCARA. Whereas the positions of the linear axes are more or less identical to the coordinates of the position in MCS, the positions of the axes of the SCARA are not that easy to calculate. Additionally there are two possible solutions of the backward kinematic transformation, different configurations of the machine: elbow down and elbow up.

How do commands behave in dynamic coordinate systems?

If the TCP should follow a moving target, this can be achieved by a dynamic coordinate transformation, leading to a PCS which is moving in relation to the MCS.

The activation of a dynamic transformation is done by activating MCA_SetDynamicFollower (ABB specific function block).

If there is a dynamic transformation active, the axis may follow the dynamic transformation or stay in the static ACS or MCS. The following example is showing the behavior. The example describes a robot fetching a screw from a fixed position and mounting it on a product that is moving on a belt.

Step	Move command	Axes (group) behavior	Application example
1	Activating Transformation ACS to MCS	Group is staying still (not moving)	Initialization, MCS is static
2	MC_MoveAbsolute in MCS	Group moves to the commanded position in MCS and stays in static MCS (not moving)	Moving to standby position and waiting for products
3	Motion command in static MCS	Group moves to the commanded position in MCS and stays in static MCS (not moving)	Moving to a fixed box of screws
4	Motion command in static MCS	Picking command	Picking up a screw
5	Activating a dynamic PCS	PCS is active and moves synchronized with the belt	PCS is ready for use
6	Motion command in dynamic PCS	Group moves to commanded position in PCS and is moving together with the dynamic PCS	Placing the screw and following the product on the belt

Step	Move command	Axes (group) behavior	Application example
7	Screwing command	Group is still following the product on the belt	Screw is being screwed into the product
8	Motion command in static MCS	Group moves to commanded position in MCS at the fixed screw box	Moving to the fixed box of screws and waiting for the next product on the belt
9	Motion command in dynamic PCS	Group moves to commanded position in PCS and is moving together with the dynamic PCS	Placing the screw to the new product and following the product on the belt

Rule: An axis group stays in the coordinate system which is specified with the last motion command. If this is a PCS with dynamic transformation, it will follow the PCS (keeping the same position in this PCS).

Movements

Applying a movement on a machine via a function block causes the TCP to move towards the new commanded position. The kind of function block applied specifies the path via which the new target position is reached. (Note: the coordinate system in which the new commanded position is specified does not have an influence on the path.)

Basically there are two types of movements which have to be distinguished:

- Point - to - Point movements, PTP (also referred to as Joint Interpolated Movements)
  With this type the essence is to reach the commanded position as fast as possible. This can be achieved by moving each axis on the shortest way from its starting position to its target position. Usually this kind of movement is the fastest way to reach a new commanded position, because at any time at least one axis moving at it’s dynamic limit. The path and the path velocity of the TCP are not important. They are determined by the process of the positions of the axes and the kinematic transformation of the machine. Therefore this kind of movement is applicable for handlings and whenever the path of the TCP is not crucial. It is recommended that all axes will arrive at the commanded position at the same point in time (synchronized).
  The applicable function blocks as specified herein are:
  – MC_MoveDirectAbsolute
  – MC_MoveDirectRelative
- Cartesian Path movements, CP (also referred to as Continuous Path movements):
  CP movements cause the TCP to move along a defined path in Cartesian space. A path can be (a set of) a straight line, a circular movement, or a spline function. The path via which the new commanded position is reached is important. For example, this is essential if a workpiece is being processed. Further, the path velocity of the TCP can be controlled directly. Contrary to point-to-point movements the process of the position of each axis is determined by the desired path and the inverse kinematic transformation.
  The applicable function blocks as specified herein are:
  – MC_MoveLinearAbsolute
  – MC_MoveLinearRelative
  – MC_MoveCircularAbsolute
  – MC_MoveCircularRelative
  – MC_MovePath

The figure below illustrates the differences between different types of movement by means of a theoretical machine.

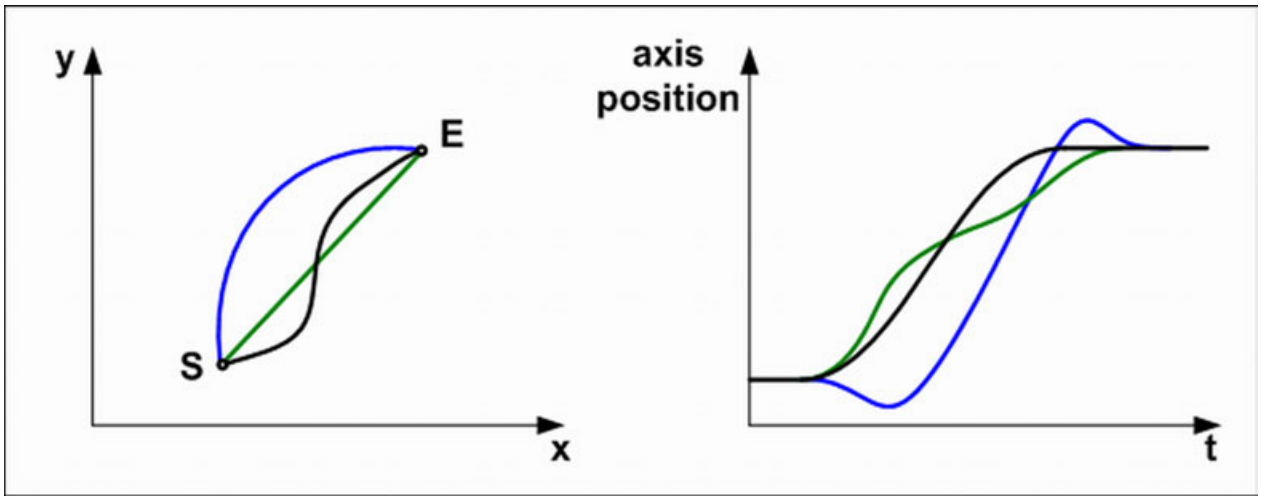


Fig. 262: Different types of movements MC_MoveDirect (black), MC_MoveLinear (green) and MC_MoveCircular (blue) and typical positions of one of the axis of the machine participating in the movement

Blending and buffering of movements

General information

A fundamental part of interpolated motion control is blending of (buffered) consecutive motion commands on an axes group. Without blending the TCP of an axes group moves towards the commanded position, decelerates and comes to standstill exactly at the commanded position. The following buffered motion command doesn’t become active until now. Obviously the axes group has to accelerate again. In many applications a different behavior of the TCP is desired and one wants to concatenate movements without stopping.

Reasons for this are:

- Reduction of the process cycle time (e.g. pick and place)
- Generate a smoother movement in order to reduce the mechanical stress
- Some applications demand a constant Velocity of the TCP (e.g. applying glue, painting, welding, etc.)

All this can be achieved by different types of blending. Common to all types of blending is a modification of the origi-nal path, resulting in a smooth trajectory without corners.

Blending of motion commands in interpolated motion control differs from blending of motion commands on single axes. With single axes the commanded position is always reached. Just the velocity at the time when the commanded position is reached (or passed) can be changed according to the input parameter BufferMode.

With interpolated motion control several types of blending can be thought of, depending on the application and proc-ess. Therefore new types of blending have to be introduced for interpo-lated motion control.

The input parameter for blending might vary due to the kind of interpolation method applied. So this input is ABB specific.

The type of inserted curve that modifies the original path (the ‘contour curve’) is not part of this specification and can be defined by the ABB specific input parameter for blending.

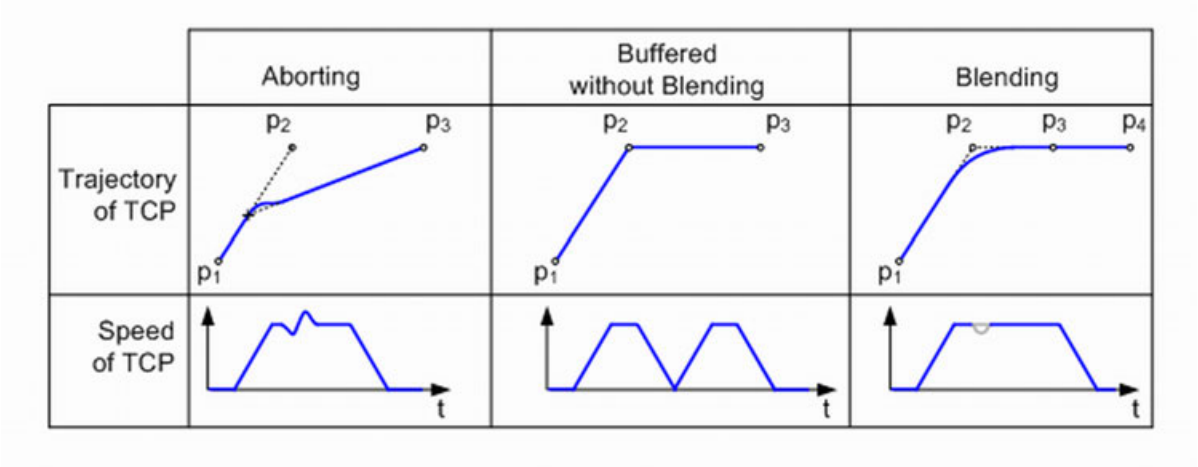


Fig. 263: Trajectories and process of Velocity in principle of two consecutive motion commands in three modes

AC500 realization

No buffered modes are realized. As no buffered mode is realized, the Transition modes are unavailable as well.

- The default mode which is used is “Aborting”. The TCP will move directly towards the next commanded position and use the next commanded velocity/acceleration/deceleration.
- Behavior according to “Buffered without Blending” can be reached by using the output DONE of the previous block to activate the next movement.
- A behavior as shown above in “Blending” could be achieved by using MC_MovePath and a dedicated corner distance.

Realization in AC500

Overview

This chapter describes the usage of PLCopen function blocks for motion control with ABB-PLC AC500. These function blocks are based on **Part4_CoordinatenMotion_V10.pdf, V. 1.1**.

The features could be used from the PLC program according to PLCopen standard. Different drives could be used and could be combined with each other as well as different fieldbusses. The following table gives an overview about the available features.

PLCopen Coordinated Motion and Principles of Coordinated Motion

Some information in this chapter require knowledge from the following chapters:

- *Chapter 1.5.9.4.9.1 “Principles of coordinated motion” on page 2679*

The library is not restricted to the use with a specific fieldbus, but it is recommended to use a synchronized bus.

Library	Version	Drive	Fieldbus	Description
MC_Base_AC500	1.1	Any	Any	Data types
MC_Blocks_AC500	1.1	Any	Any	PLCopen function blocks for single and multi axis
MC_CoBlocks_AC500	2.1	Any	Any	PLCopen function blocks for coordinated motion

Library	Version	Drive	Fieldbus	Description
CompactMotion_AC500	1.2	Any	Any	Motion control for single and multi axis
CoordinatedMotion_AC500	2.1	Any	Any	Motion control for coordinated motion
CMC_Ext_AC500	2.1	Any	Any	External libray, just used by other libraries
CMC_Transforma-tionen_AC500_APPL	2.1	Any	Any	Exemplary implementation for transformations
MathFunctions_AC500	2.1	Any	Any	Vector and matrix mathematic used by other libraries

The CMC_Transformation_AC500_APPL_V21.lib holds some kinematical transformations which are used to connect the MCS and ACS axes. This library is open, the integrated transformations might be copied and modified to match the specific requirements.

General rules

The following chapter explains some rules on the usage of the libraries.

- A general rule is that ALL PLCOpen function blocks which are used for a specific drive are to be used in the same PLC-task. When multitasking is used for the PLC, it is allowed to have different drives in different tasks, but all Funciton Blocks belonging to a specific drive need to be in the same task. There is no multithreading protection for the AXIS_REF instance.
- The CMC_MotionKernel… and the COMC_Group… function blocks might be in a different task then the PLCopen Blocks (MC…). All function blocks belonging to the same fieldbus Communication Module have to be called from the same task.
- When AXIS_REF or AXES_GROUP_REF is used as input on a user defined FUNCTION_BLOCK or PROGRAM or FUNCTION, then ALLWAYS use it as VAR_IN_OUT and NEVER use it as VAR_INPUT or VAR_OUTPUT. The reason is that this would
  - Break the consistency and destroy data
  - Consume a lot of computing power by copying data.
- The "Min update time" update time for the fieldbus, defined under *“PLC Configuration ➔ Communication Modules[FIX] ➔ <fieldbus master type>”* must not exceed the half of the scantime of the PLC-task. E.g. scantime of PLC-task is 5ms, then "Min update time" should not be greater than 2ms.
- The functionality of CompactMotion_AC500 library may be combined with the CoordinatedMotion_AC500 library. All single axis functions could still be used for an axis, even when it is combined with others to a group .

Create a group

The basic functionality in coordinated motion is to combine several axes in a movement in 3D-cartesian space. The combination of axes is named “group” and the movement is executed by the group.

Several axes could be combined to realize a group which performs coordinated motions in a specific coordinate sys-tem. The single axis block from CompactMotion_AC500 Library will be available too, and have to be combined with the group-block to realize the complete functionality. The Group-Block does not realize a position control or direct access to an axis, therefore the CompactMotion_AC500 library is used for this.

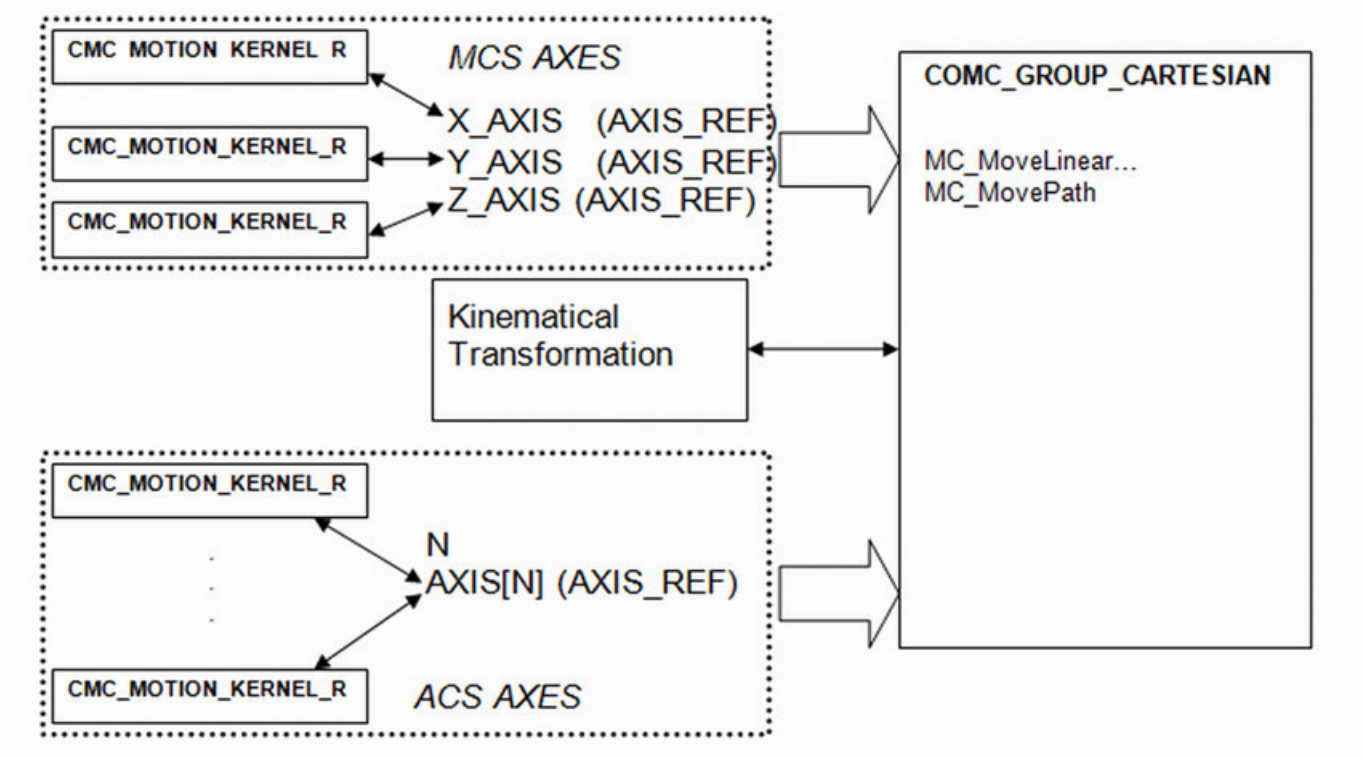


Fig. 264: architecture

The COMC_GROUP_CARTESIAN supports interpolation in a 3-dimensional Cartesian space. The MCS AXES repre-sent the 3 axes for this system. Depending on the mechanical construction, 3 or more real axes are necessary to realize the 3 dimensional movement. These are named ACS AXES and linked by a kinematical transformation to the MCS AXES.

State transitions

As the single axis and the group are related, the state machines for both are connected. The following description gives an overview:

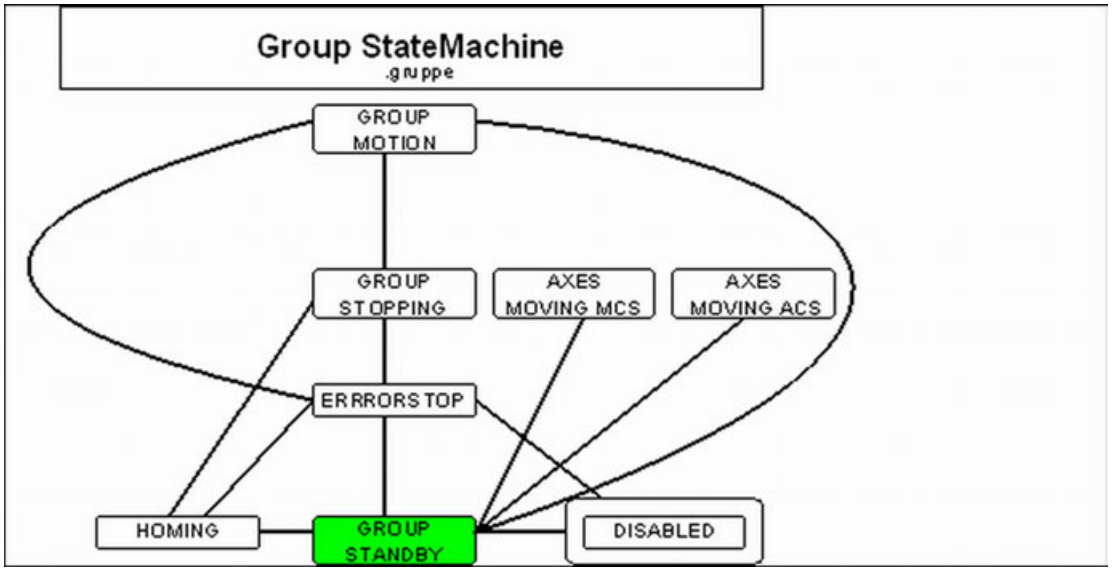


Fig. 265: state machine for group

MC_GroupEnable

- Precondition: The ACS axes are in state StandStill
- Postcondition:
  - The group changes to state GroupStandby.
  - The ACS axes and MCS axes changed to Synchronized Motion. All axes could be moved by the group.

MC_GroupDisable

- Precondition: None
- Postcondition:
  - The group changes to state GroupDisabled.
  - The ACS axes change from Synchronized Motion to StandStill or keep their state if not in Synchronized Motion.
  - The MCS axes change to state Disabled.

MC_Move… applied to GROUP

- Precondition: The group is in state Standby or GroupMotion.
- The group changes to state GroupMotion.
- When a MCS or ACS movement was executed, the group state has to be checked before a group block can be started.

Any axis changes its state (other then Disabled or ErrorStop)

- Precondition: The group was in a state other then Disabled or ErrorStop.
- Postcondition: The group is changed to AXES_MOVING_MCS or AXES_MOVING_ACS, as soon as ALL axes are in StandStill again, they will automatically be changed to Synchronized Motion and the group will change to GroupStandby.

Any ACS axis changes its state to ErrorStop

- Precondition: None
- Postcondition: The group state is set to Disabled. The MCS axes are set to Disabled.

Any MCS axis changes its state to ErrorStop

- Precondition: None
- Postcondition: The group state is set to ErrorStop.

Any axis changes its state to Disabled

- Precondition: None
- Postcondition: The group state is set to Disabled. The MCS axes are set to Disabled.

General restrictions

Table 173: Function blocks realized in CoordinatedMotion_AC500

Administrative	**Motion**	
Coordinated	**Coordinated**	**Synchronized**
MC_GroupEnable	MC_GroupStop	MC_SyncAxisToGroup
MC_GroupDisable	MC_GroupHalt	MC_SyncGroupToAxis
MC_SetCartesianTransform	MC_GroupInterrupt	
MC_SetCoordinateTransform	MC_GroupContinue	
MC_ReadCartesianTransform	MC_MoveLinearAbsolute	
MC_ReadCoordinateTransform	MC_MoveLinearRelative	
MC_GroupReadActualPosition	MC_MoveCircularAbsolute	
MC_GroupReadActualVelocity	MC_MoveCircularRelative	

Administrative	Motion	
Coordinated	**Coordinated**	**Synchronized**
MC_GroupReadStatus	MC_MoveDirectAbsolute	
MC_PathSelect	MC_MoveDirectRelative	
	MC_MovePath	

Table 174: Additional ABB specific function blocks

Function block	Comment
MCA_SetCoordinateTransformation *Chapter 1.5.9.7.3.3 “MCA_SetCoordinateTransformation” on page 3016*	Replaces MC_SetCartesianTransform
MCA_SetDynamicFollower *Chapter 1.5.9.7.3.4 “MCA_SetDynamicFollower” on page 3018*	Replaces MC_TrackRotaryTable, MC_TrackConveyorBelt and MC_SetDynCoordTransform
MCA_PathEvent *Chapter 1.5.9.7.3.2 “MCA_PathEvent” on page 3015*	Set a binary signal output related to path positions
MCA_MoveHelixRelative *Chapter 1.5.9.7.3.1 “MCA_MoveHelixRelative” on page 3008*	Allows a circular movement > 2Pi and a coordinated vertical movement
MCA_MovePathPos *Chapter 1.5.9.7.3.5 “MCA_MovePathPos” on page 3022*	A path movement with an included previous positioning to start position

Restrictions to be considered

The following extended inputs and outputs at the function blocks are not realized:

- BufferedMode: The realization just supports by default the “Aborting” mode.
- TransitionMode: The realization just supports by default a transition starting with the actual velocity.
- TransitionParameter: Not supported.
- CoordSystem: Set by default when using the specific group or function block.

When CoordSystem is available as an input, the type is MC_COORD_SYSTEM with values:

- MC_DEFAULT_COORD = the last activated coordinate system, MCS or PCS
- MC_MCS_COORD
- MC_PSC_COORD

A movement which is executed in a specific coordinate system also switches the group to the specified system.

To access the axes in ACS system, specific function blocks are used or the information has to be retrieved from the ACS axes directly.

Usage of function blocks

The basic function block is COMC_GROUP_CARTESIAN. *Chapter 1.5.9.7.1.20 “COMC_GROUP_CARTESIAN” on page 2994* It has to be called every cycle and at least once before any MC… function block is activated.

Coordinate transformations

Different transformations may be used to transfer the data from MCS to ACS (axes coordinate system) and back. Just the actual positions and reference positions are transferred, as shown in the diagram below.

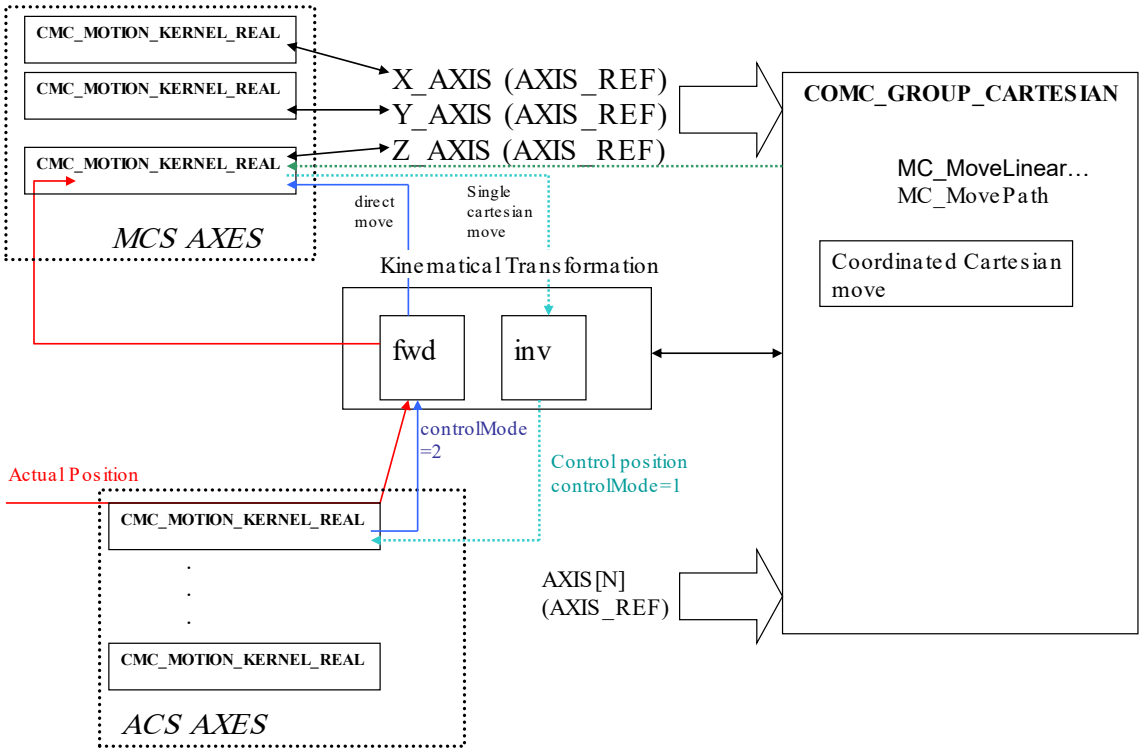


Fig. 266: Data flow for actual and reference position

The controlMode is switched internally and creates a different flow for control position, depending on the required movement, e.g if a direct movement on joints or a cartesian movement is required.

The actual position is always received from the ACS axes and transferred to the MCS axes by the forward transformation.

In a coordinated group movement or in a movement of cartesian axes, the reference position is transferred from the MCS axes to the ACS axes by the inverse transformation.

In case of a direct movement or a movement of ACS axes, the reference position is transferred by the forward transformation from ACS axes to MCS axes.

The calculation of actual positions is executed as follows:

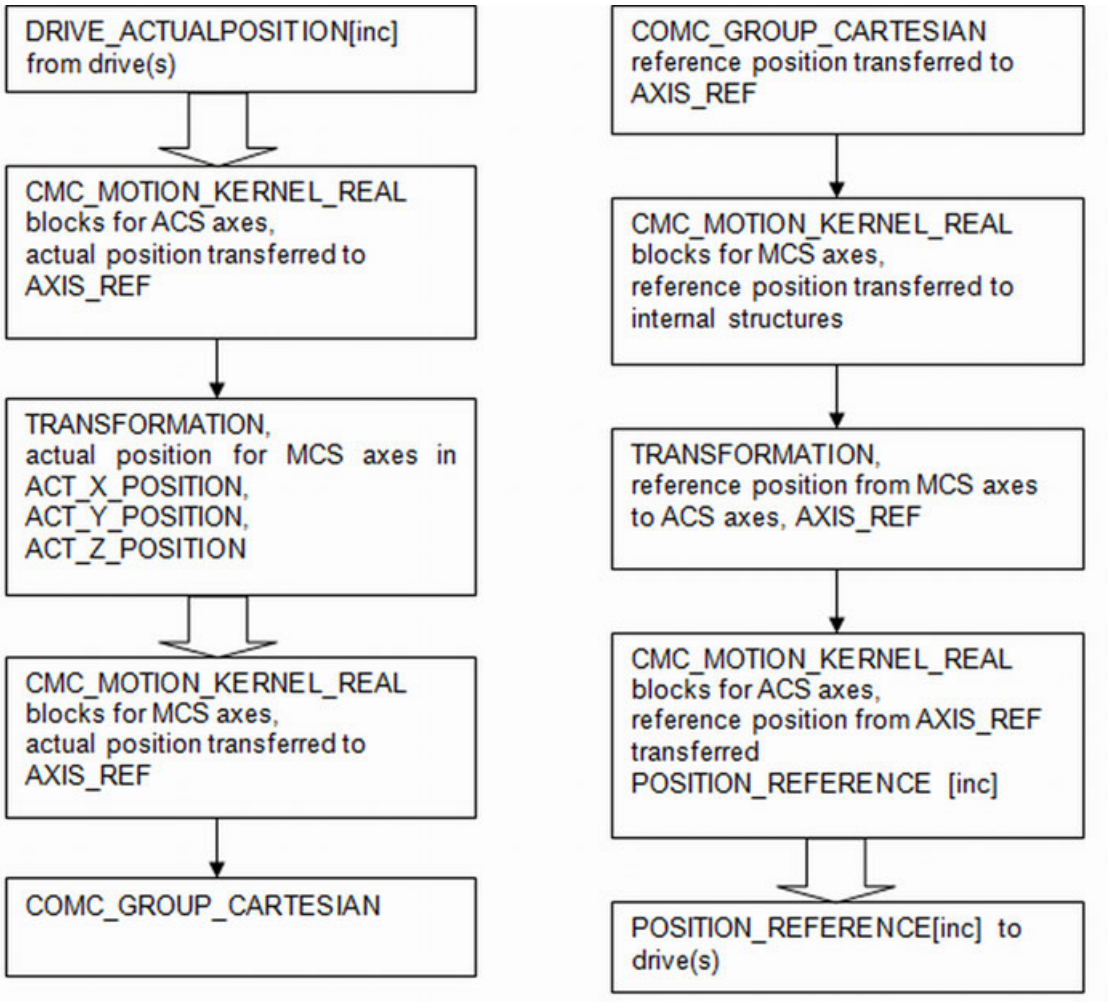


Fig. 267: Calculation of actual positions and reference positions for coordinated axes

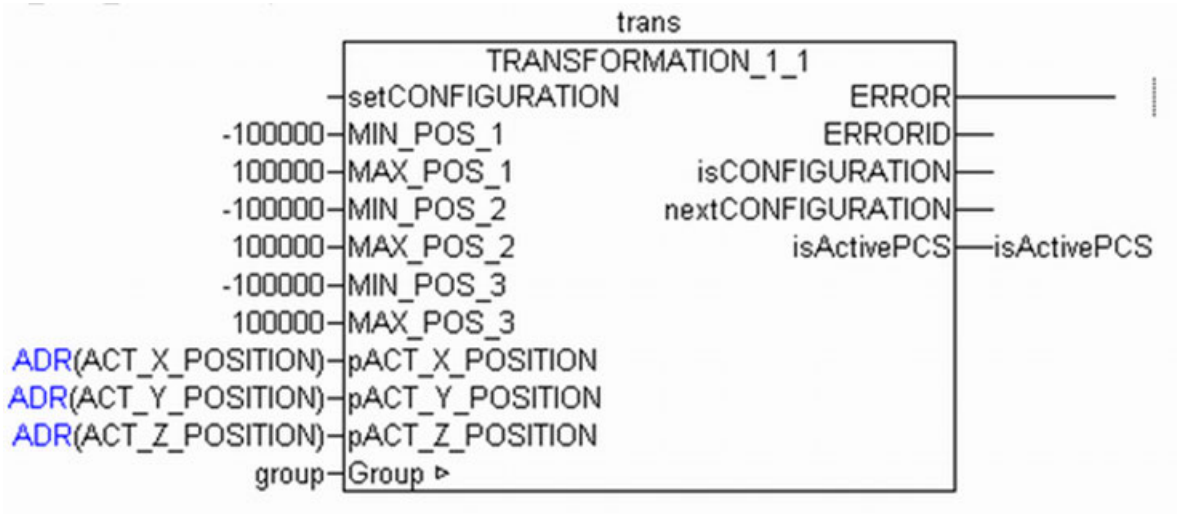
The user is free to define an own algorithm for transformation of coordinates.

The basic structure is given by the function block TRANSFORMATION_1_1. It does a 1:1 transformation. Any other transformation has to be created following the same structure.

For 1_1 Transformation, no different configurations are possible, nextCONFIGURATION = 0.

Structure of TRANSFORMATION_1_1

As an example, the TRANSFORMATION_1_1 is shown. To create an own transformation, this function block could be used and modified. The modified function block may have additional inputs and outputs. The functionality is realized by using different actions. Actions named ACTION_GENERIC_xxx are not allowed to change. Actions named ACTION_APP_xxx have to be changed according to the required transformation.



Input description

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

setConfiguration
Data type: WORD
Required configuration, if more than 1 solution for forward transformation is possible.

MIN_POS_1
Data type: LREAL
Minimal position for Cartesian axis 1.

MAX_POS_1
Data type: LREAL
Maximal position for Cartesian axis 1.

MIN_POS_2
Data type: LREAL
Minimal position for Cartesian axis 2.

MAX_POS_2
Data type: LREAL
Maximal position for Cartesian axis 2.

MIN_POS_3
Data type: LREAL
Minimal position for Cartesian axis 3.

MAX_POS_3
Data type: LREAL
Maximal position for Cartesian axis 3.

pACT_X_POSITION Data type: POINTER TO LREAL

Actual position, to be used at the corresponding CMC_MOTION_KERNEL… function block.

pACT_Y_POSITION Data type: POINTER TO LREAL

Actual position, to be used at the corresponding CMC_MOTION_KERNEL… function block.

pACT_Z_POSITION Data type: POINTER TO LREAL

Actual position, to be used at the corresponding CMC_MOTION_KERNEL… function block.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

isCONFIGURATION Data type: WORD

Active configuration.

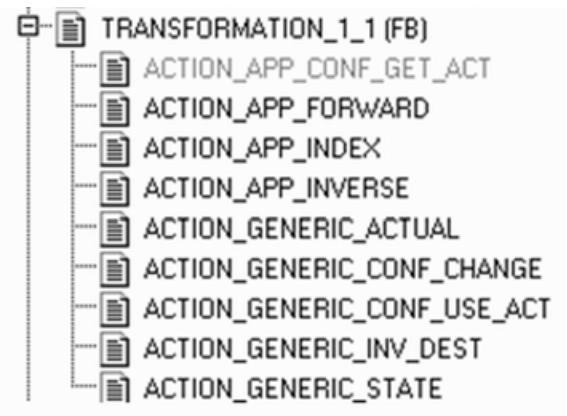
nextCONFIGURATION Data type: WORD

Configuration which will be reached by the next direct movement, if more than one solution for forward transformation is possible and isConfiguration differs from setConfiguration.

isActivePCS Data type: BOOL

PCS (Product Coordinate System) is activated, group movements are executed related to PCS instead MCS and tracking (e.g. Conveyor tracking) is possible.

Structure of actions within a transformation



The functionality is realized in different actions. All actions named “ACTION_GENEREC_xxx” don´t need to be changed for different transformations. The actions “ACTION_APP_xxx” need to be changed according to the transformation needed. A constant is needed to define the number of ACS axes. This has to be modified according to the transformation.

```
VAR CONSTANT
  (**NUMBER*)
     N_ACS_AXES:WORD:=3;
END
```

The minimal and maximal positions are not mandatory. It depends on the transformation if these are necessary or if additional limits have to be considered.

Responsibilities of different actions

- ACTION_APP_INDEX calculates the index and address for the used TRANSFORMATION-Block. The name of the actual block has to be used
  - TransformationIndex:=INDEXOF(Transformation_1_1).
  - Use the name of your own transformation block here.
- ACTION_APP_CONF_GET_ACT: Take the actual positions of ACS axes and determine the mechanical configuration actually used, write the result to isCONFIGURATION.
- ACTION_APP_FORWARD: Hold the forward transformation and calculates the actual position by using the actual positions from ACS axes and creating X_FORWARD, Y_FORWARD and Z_FORWARD.
- ACTION_APP_INVERSE: Holds the inverse transformation and calculates the positions POS1, POS2 and POS3 from the input in X_FORWARD, Y_FORWARD and Z_FORWARD. POS1, POS2 and POS3 are used as reference positions to the ACS axes or as destination positions to the ACS axes in case of a direct movement.
- ACTION_GENERIC_ACTUAL: Not to be modified. Uses X_FORWARD, Y_FORWARD and Z_FORWARD and calculates the actual positions ACT_X_POSITION, ACT_Y_POSITION and ACT_Z_POSITION by involving the dynamic transformation, if activated. These position values are calculated in [u]. They have to be used at position input for the MCS axes, instead of an actual position directly received from the drive.
- ACTION_GENERIC_CONF_CHANGE: Not to be modified.
- ACTION_GENERIC_CONF_USE_ACT: Not to be modified
- ACTION_GENERIC_FWD_DEST: Not to be modified.
- ACTION_GENERIC_STATE: Not to be modified.

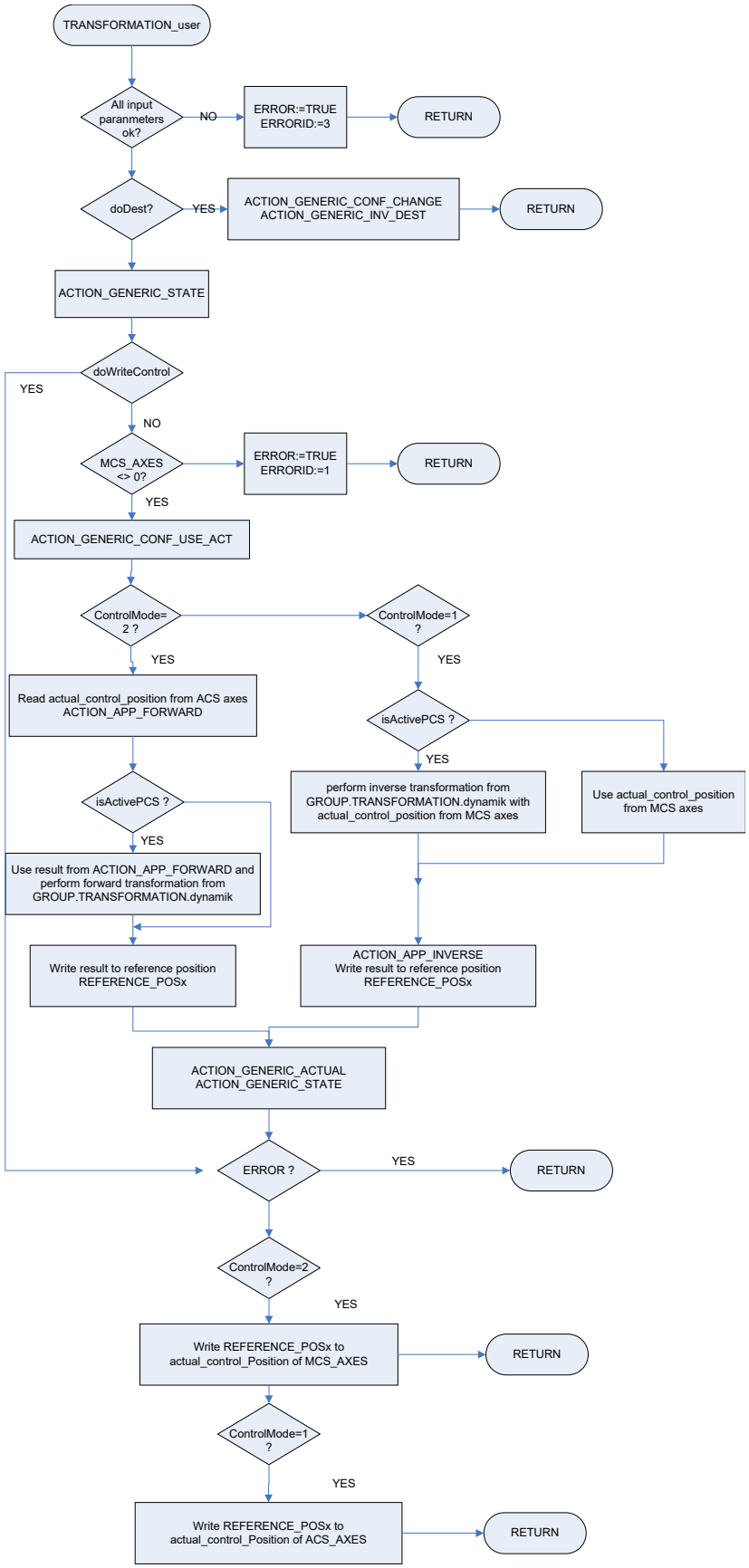


Fig. 268: Flowchart for the transformations implementation

Change of configurations

It might be possible to reach the same target position by using different mechanical configurations, e.g. elbow_up and elbow_down in a SCARA robot.

Sequence in changing the configuration

For changing the configuration, a direct movement has to be executed.

When different configurations are possible, the following situations have to be considered:

- No change in configuration is required: setConfiguration=nextConfiguration=isConfiguration
- The configuration should be changed: setConfiguration<>nextConfigurtion=isConfiguration
  start a direct movement: setConfiguration=nextConfiguration<> isConfiguration
  movement to required position is ready: setConfiguration=nextConfiguration=isConfiguration

Different actions have to consider the configuration:

- ACTION_APP_CONF_GET_ACT: generate a value for the actual active configuration form the ACS axes actual positions. Write the result to isCONFIGURATION.
- A new desired configuration could be written to the input setCONFIGURATION. This will be used when a di-rect movement is performed and will be written to nextCONFIGURATION.
- When no change in configuration is performed, the values setCONFIGURATION, nextCONFIGURATION and isCONFIGURATION will be identical.
- The action ACTION_APP_FORWARD should use the value from nextCONFIGURATION to calculate its result.

PCS (Product Coordinate System)

Basically, the MCS axes represent a 3 dimensional Cartesian space.

It is linked with the used “TRANSFORMATION” function block to the ACS axes which are the real movement axes.

The MCS has a fixed origin and the 3 axes X,Y,Z follow the right hand rule.

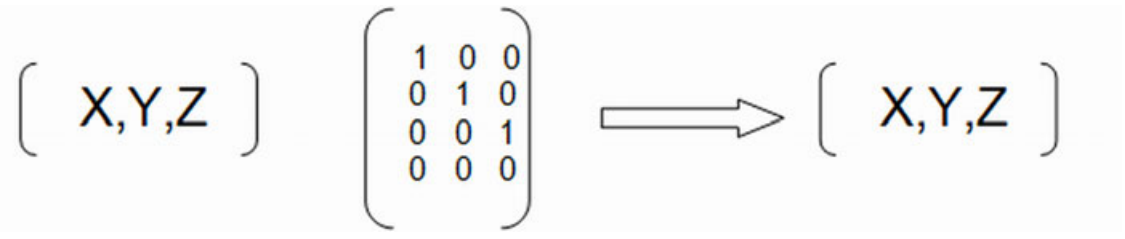
It is possible to use a PCS for all movements of the cartesian axes.

The PCS is activated with MCA_SetCoordinateTransformation.

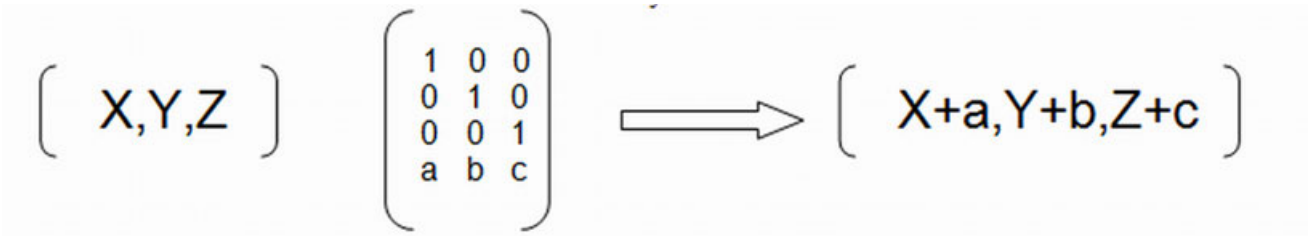
The calculation is implemented by a homogeneous transformation using a FORWARD matrix to transform MCS to PCS and an INVERSE matrix to transform PCS to MCS coordinates.

The PCS is a Cartesian system as well and might be turned and shifted and use different scaling compared to the MCS system.

A matrix without modification would have the following values:



A matrix which does a shift of the coordinate system will look as follows:



For transformation from MCS to PCS, the forward matrix means:

- The first line points the direction of PCS X-Axis, with respect to MCS
- The second line points the direction of PCS Y-Axis, with respect to MCS
- The third line points direction of PCS Z-axis, with respect to MCS

So the lines represent the vector for the new coordination systems axes. If the length for a line is different from "1", it means the new system has also a different scaling.

The matrix might be created by using the function “COMC_TeachCartesianTransformation”. This block allows teaching the relation between MCS and PCS.

How to switch the coordinate system

The PCS coordinate system allows to move the group in relation to a certain product workspace and not just in relation to the machines coordinates. It is also possible to follow a moving product by utilizing the PCS coordinate system. And move it continuously.

Table 175: Available function blocks for doing so

Function block	Step	Activation time	Used data format	Dynamic	From → To
MCA_Set-Coordinate-Transforma-tion	Step 1	Enable	Matrix	Yes	MCS → PCS
MC_SetCoor-dinateTrans-form	Step 1	Execute	Matrix	While Exe-cute=true	MCS → PCS
MCA_SetDy-namicFol-lower	Step 2	Enable	Translation/ Rotation	Yes	PCS → PCS'
MC_SetCarte-sianTransform	Step 2	Execute	Translation/ Rotation	While Exe-cute=true	PCS → PCS'

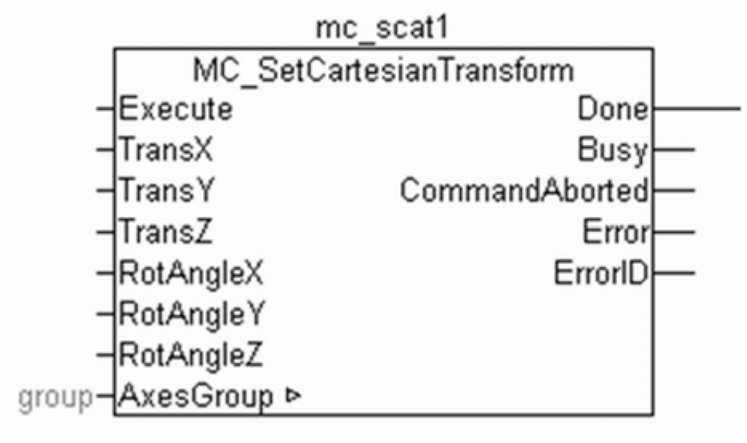
Activation time The function blocks with input Enable will be activated immediately with Enable=TRUE. The function blocks with “Execute” will be enabled with the next movement after Execute=TRUE. The function blocks with “Enable” will be deactivated with Enable=FALSE. The function blocks with “Execute” will be deactivated when an other transformation is activated.

Step Any step 2 transformation can just be active while a step 1 transformation is in place. The step 1 transformation function blocks will allow dynamic transformation while Execute=TRUE. “Frozen” values will be used with Execute=FALSE after once activated.

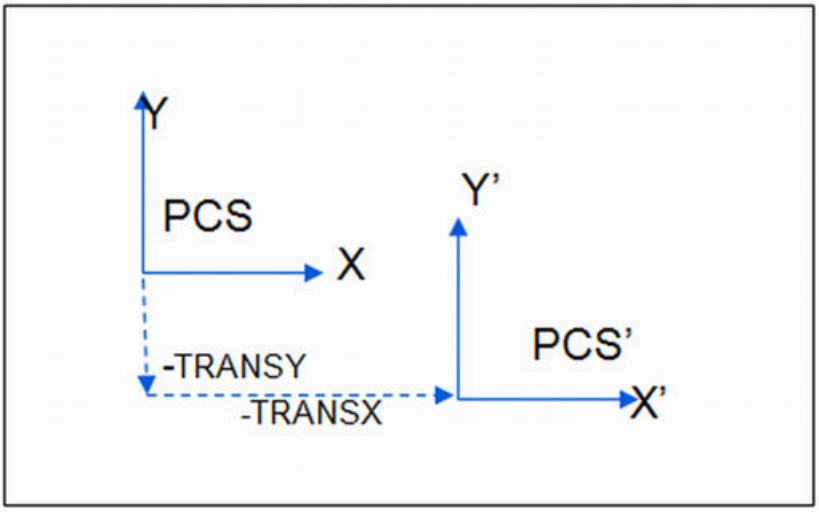
It is possible to do the transformation in 2 steps.

Step 1 is a precondition for any of the step 2 Funciton Blocks. The matrix used in step 1 is a more complete way to setup a PCS coordinate system, as it would also allow modifying the scaling for any axis in addition to translation and rotation. The forward and inverse matrix can be gained by using the COMC_TeachCartesianTransformation function block.

It is also possible to use the matrix in a dynamic way, e.g. modifying the values on the fly, but as usually the dynamic requirement is restricted to a specific translation or rotation, a step 2 function block can be used for this, once the PCS system has been set up.



The step 2 function blocks allow to use translation vectors in X, Y, Z direction and rotation angles for a rotation around X, Y, Z coordinate axis.



TransX	-18
TransY	10
RotangleZ	0

This transformation would modify the actual position as follows, when it is applied to the group in StandStill with the given parameter.

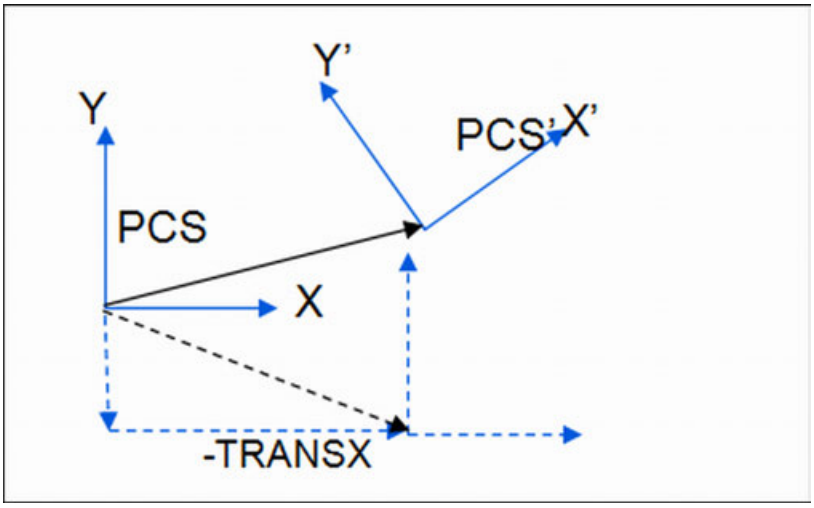
	PCS, PCS'	**PCS, PCS'**
X	0, 0	0, -18
Y	0, 0	0, 10
Z	0, 0	0, 0

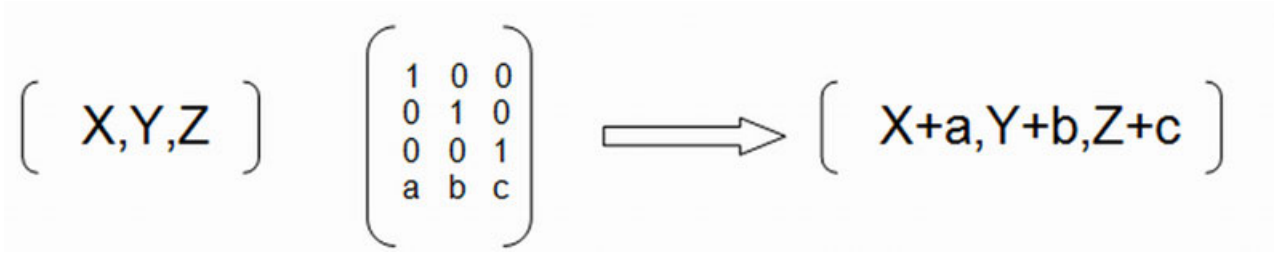
A group-movement to position 0,0,0 will move the group to the origin of the new PCS coordinate system.

When the function block MCA_SetDynamicFollower is enabled with TransX=0 and TransY=0 and the values are applied to the activated function block, the group will keep its PCS position constant and follow the translation. This will result in a real movement and modify the ACS and MCS positions.

	PCS, PCS'	PCS, PCS'
X	0, 0	18, 0
Y	0, 0	-10, 0
Z	0, 0	0, 0

When an additional rotation is applied (in this case RotAngleZ), this rotation is done with respect to the original PCS coordinate system. When a rotation should be done related to the “new” PCS’ system, the translation vector has to be applied to the matrix which builds the PCS system.





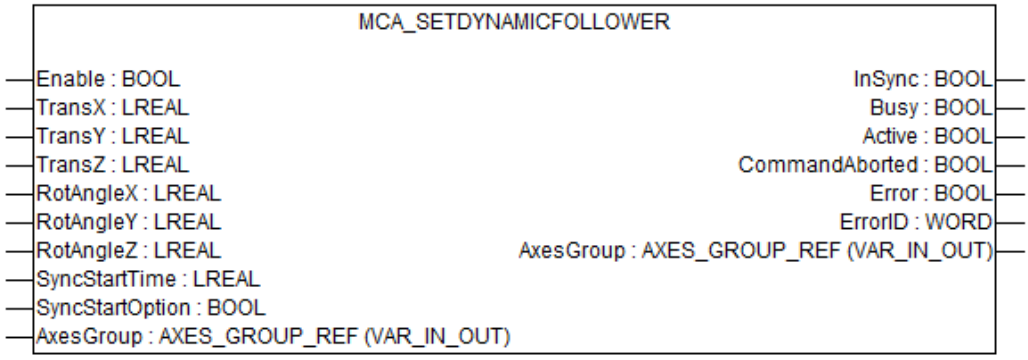
In the forward matrix, the values a,b,c match –TransX, -TransY and –TransZ. In the inverse matrix, the values a,b,c match TransX,TransY and TransZ.

How to teach the conversion matrix

When the group is activated in the MCS coordinate system, it is possible to teach the matrix (inverse and forward) which is necessary to reach the PCS coordinate system. 4 points are needed to do so. They should not be in a line or on the same level, they need to be 3-dimensional independent. ↳ *Chapter 1.5.9.7.1.21 “COMC_TeachCartesianTransformation” on page 2996*

Dynamic coordinate transformation

When a PCS is active, it is also possible to activate a dynamic coordinate transformation in addition which would allow move the axes in relation to a moving product. To do this, the function block “MCA_SetDynamicFollower” has to be used (also MC_SetCartesianTransform would be possible). With this function block, a movement in X and Y direction (related to PCS coordinates) and a rotation around the z-axis is possible.



Related to the FORWARD_MATRIX, this would mean:

$$\begin{pmatrix} c & -s & 0 \\ s & c & 0 \\ 0 & 0 & 1 \\ t_x & t_y & 0 \end{pmatrix}$$

With c = cos(alpha) and s = sin(alpha). Alpha is the angle used to turn around the z-axis. When transformations beyond the “MCA_SetDynamicFollower” options are needed, the matrix could be directly modified.

1. To move the product in the PCS X/Y plane, use MCA_SetDynamicFollower.
2. To turn the product around the Z-axis of PCS (e.g. rotary table), use MCA_SetDynamic-Follower.
3. To do both above movements, use MCA_SetDynamicFollower.

☑ The product is moved in 3 dimensions.

▷ Use MC_SetCartesianTransform or manipulate FORWARD_MATRIX and INVERSE_MATRIX accordingly.

ABB specific data structures

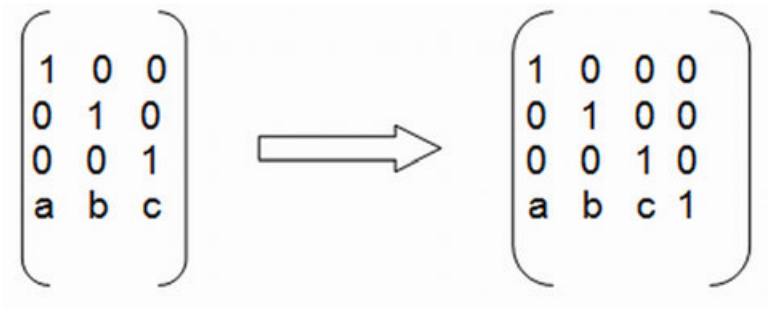
Not all data structures are defined by PLCopen. Some specific structures are described in the following chapter.

Data structures to be used for MCS-PCS transformation

The data type MC_COORD_REF is foreseen for the MCS->PCS transformation in forward (MCS to PCS) and inverse (PCS to MCS) direction. The data type is defined as follows:

```
TYPE MC_COORD_REF :
STRUCT
    INVERSE_MATRIX:     MATRIX_4_3;
    FORWARD_MATRIX:     MATRIX_4_3;
END_STRUCT
END_TYPE
```

The INVERSE_MATRIX should be the inverted matrix to FORWARD_MATRIX. It is not checked if this is really true, but when this rule is not followed it will not be possible to control the MCS axes as actual positions and reference positions would differ. According to the formal rules of matrices math, a square matrix would be necessary to allow inversion. The homogeneous transformation which is used for coordinate transformations expands the matrices to 4x4, so the rules are fulfilled.



Declaration example

```
CoordTransform_neutral:MC_COORD_REF:=
(INVERSE_MATRIX    :=(m:=1,0,0,0,1,0,0,0,1,0,0,0),
 FORWARD_MATRIX    :=(m:=1,0,0,0,1,0,0,0,1,0,0,0));
```

Data structures to be used for moving the group on a path

MC_PATH_REF

This structure is used for moving a Cartesian group on a path. It is evaluated by MC_PathSelect, which creates the structure MC_PATH_DATA_REF to be used for the path movement.

A path movement can be done with the following function blocks:

- MC_MovePath: Move on a path with a defined velocity/in a defined time.
- MC_MovePathPos: Move on a path with a defined velocity/in a defined time including the movement to the path start position.
- MC_SyncGroupToAxis: Move on a path following a master axis.

```
TYPE MC_PATH_REF :
(*This structure is used for moving a Cartesian group on a path*)
STRUCT
    CORNER_DISTANCE:LREAL;    (*u *)
    ACCELERATION_TIME:LREAL;  (*s*)
    pPATHPOINTS:POINTER TO MC_PATH_POINT;
    NUMBER_OF_POINTS:DINT;(*number of points*)
    CORNER_MODE:DWORD;        (*0 = "DISTANCE", 1 = "TANGENTE"*)
    TRANSITION_MODE:MC_TRANSITION_MODE;(*TmConstantVelocity possible with CORNER_MODE=0*)
END_STRUCT
END_TYPE
```

For PathMovement and BufferedMovement, the TMConstantVelocity can be used as TransitionMode and will influence the path-movement in a way that the velocity is kept constant, no matter the angle, as described with buffered movement. It is applied in combination with mode CORNER_MODE=0 (use CORNER_DISTANCE). This special TransitionMode is added to MC_PATH_REF. Any other value at TRANSITION_MODE will result in default behavior, which means a deceleration during a corner, depending on the angle *Chapter 1.5.9.7.3.9 “MCA_MoveBuffered” on page 3030*.

CORNER_DISTANCE Data type: LREAL, unit: u

Distance at which the velocity vector start to leave the actual direction and turning towards the next point. The value has to be >=0.

ACCELERATION_TIME Data type: LREAL, unit: ms

Time which is used to reach the path-velocity.

pPATHPOINTS Data type: POINTER TO MC_PATH _POINT

Pointer to an array which hold the points on the path.

NUMBER_OF_POINTS Data type: DINT

Number of points.

CORNER_MODE Data type: DWORD

=0 => use CORNER_DISTANCE.

=1 => use CORNER Interpolation.

=0x100 => use CORNER_DISTANCE, but related to MASTER_POSITION (ignore V_PATH).

=0x101 => use CORNER Interpolation, but related to MASTER_POSITION (ignore V_PATH).

Table 176: Available options

CORNER_MODE	MC_MovePath/MCA_MovePathPos	MC_SyncGroupToAxis
0,1	The movement is executed by time, calculated based on V_PATH.	The movement is executed by synchronizing to a master axis. The master positions are calculated based on V_PATH which means V_PATH gives the relation from slave/master velocity [u(slave)/u(master)] instead of [u/s].
0x100, 0x101	The movement is executed by time, calculated based on MASTER_POSITION. The given values are interpreted as [s].	The movement is executed by synchronizing to a master axis. MASTER_POSITION is used. The group will reach the given X/Y/Z point when the master axis reaches MASTER_POSITION. The points are not reached exactly but modified by CORNER_DISTANCE or the CORNER algorithm.

MC_PATH_POINT

V_PATH Data type: LREAL, unit: u/s

Path velocity which should be reached from this point to the following point.

The velocity V_PATH is used when the movement is executed by MC_MovePath or MCA_MovePathPos. The interpolation aims to reach V_PATH as path velocity, so V_PATH determines the interpolation progress. When using a path movement controlled by a master axis (MC_SyncGroupToAxis), the MASTER_POSITION is used instead, according to a CAM Table movement. V_PATH is then ignored.

X Data type: LREAL, unit: u

Position for the X-Axis.

Y Data type: LREAL, unit: u

Position for the Y-Axis.

Z Data type: LREAL, unit: u

Position for the Z-Axis.

MASTER_POSITION Data type: LREAL

Position for master axis used by MC_SyncGroupToAxis, unit: u.

Time when used with MC_MovePath or MCA_MovePathPos, unit: ms.

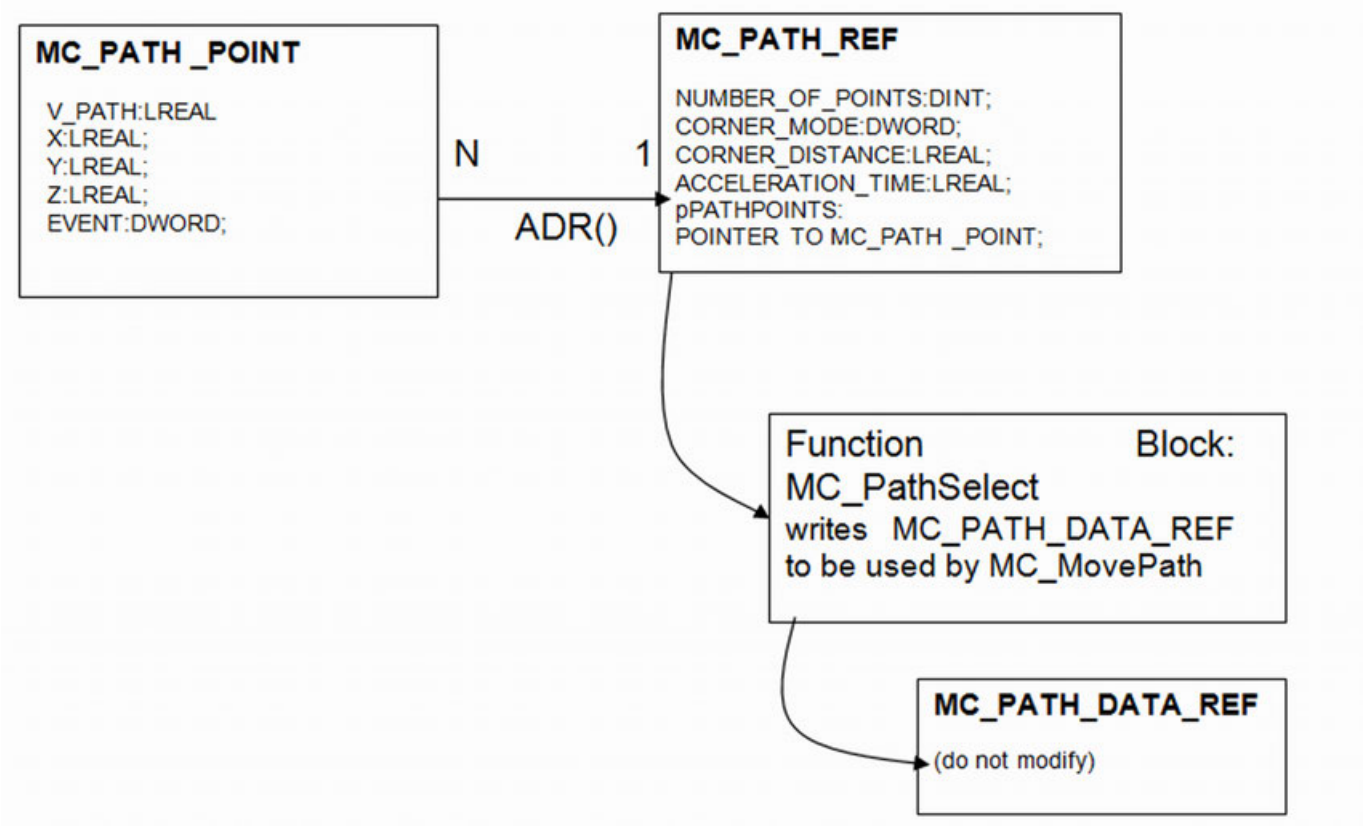
EVENT Data type: DWORD

Binary pattern which would be displayed by function block MCA_PathEvent.

MC_PATH_DATA_REF

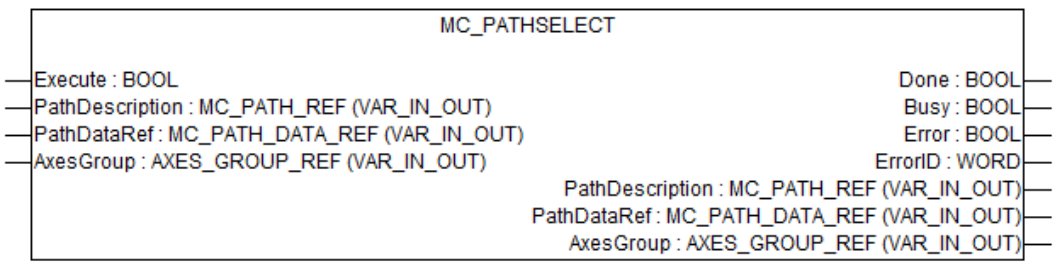
Filled by MC_PATH_SELECT, not to be modified.

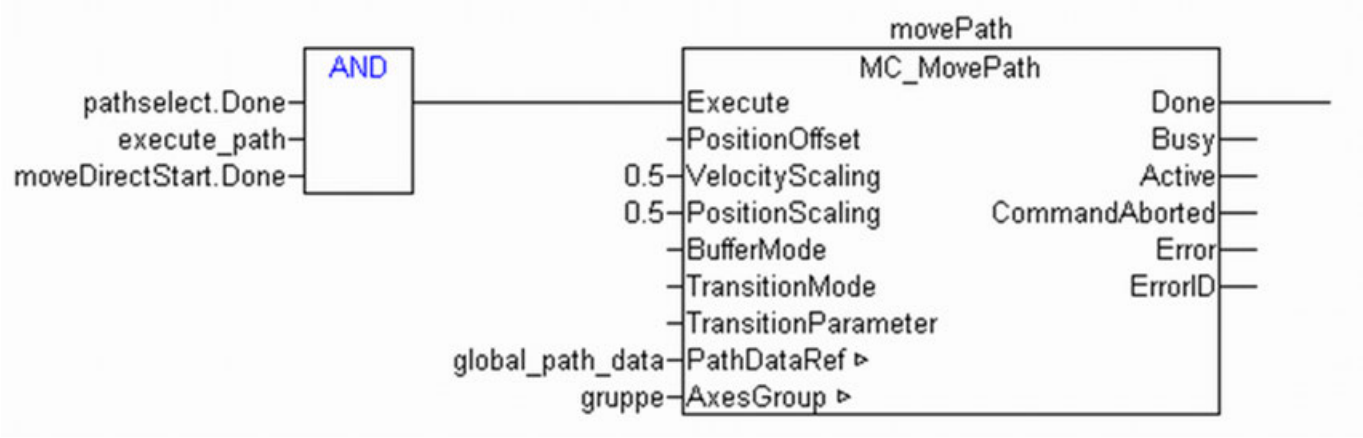
The structure MC_PATH_DATA_REF is completely filled out by the function block MC_PATH_SELECT. The basic data which has to be provided in an array of MC_PATH_POINT is prepared to be used by the function block MC_MOVE_PATH.

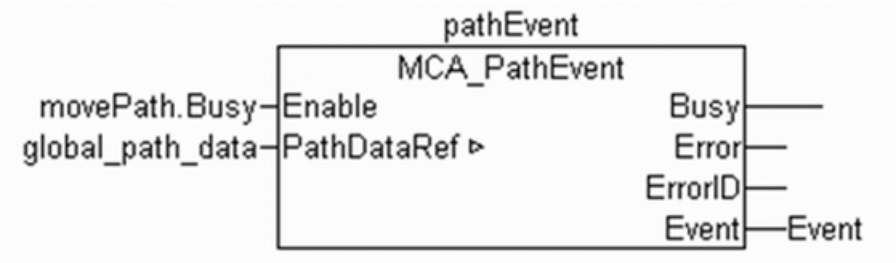


Example for using the path data

1. Create an element of type MC_PATH_REF.
   path_description: MC_PATH_REF;
2. Create an array of type MC_PATH _POINT.
   path_array: ARRAY[1..7] OF MC_PATH _POINT;
3. Fill the MC_PATH_REF with data.
   path_description.ACCELERATION_TIME:=100;(*ms*)
   path_description.corner_mode:=0;
   path_description.CORNER_DISTANCE:=50;(*mm*)
   path_description.N:=7;
   path_description.Path:=ADR(path_array);
4. Initialize the path_array itself.







The PathDescription has to be evaluated by MC_PathSelect. The resulting information is transferred by PathDataRef to the function block MC_MovePath which starts the movement. The MC_PathSelect just needs to be called once when the PathDescription has been changed. When the same data is used several times, it is enough to just call the MC_MovePath.

The MCA_PathEvent is an additional ABB specific function block which allows writing an “Event” value (32Bit) according to the position which is actually reached by the path interpolation. “Event” matches the respective value of EVENT as element of MC_PATH_POINT, when the group reaches this point.

Interpolation modes available for MC_MovePath

Two different interpolation modes are available:

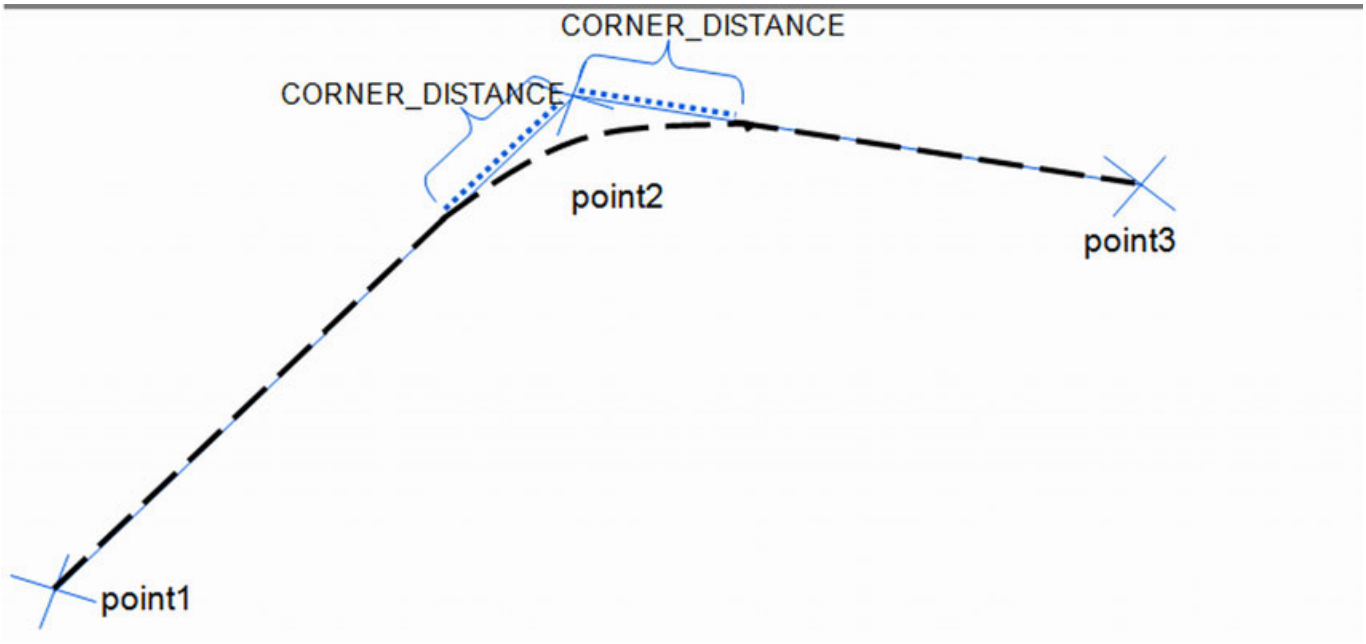


Fig. 269: Behavior with CORNER_MODE=0

When CORNER_MODE in MC_PATH_REF is =0, the movement will be executed on the given path between 2 points until the given CORNER_DISTANCE is reached. Then the direction will be changed and the TCP will again reach the path from point 2 to point 3 at CORNER_DISTANCE behind point 2.

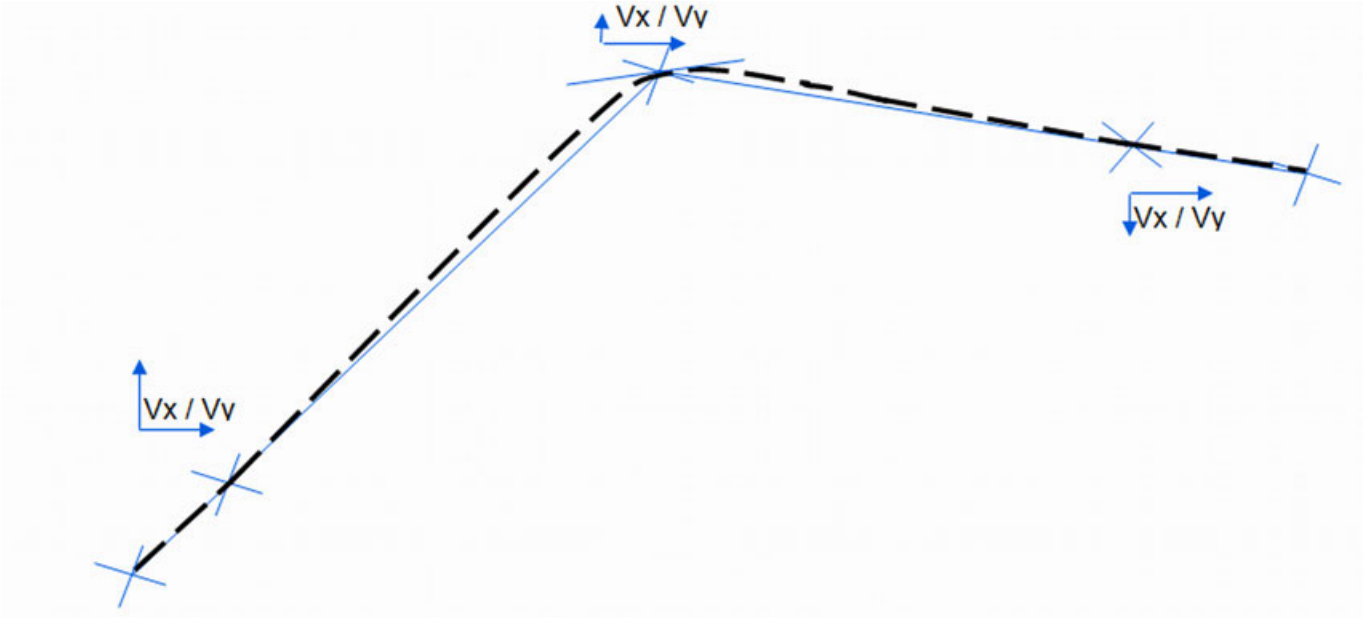


Fig. 270: Behavior with CORNER_MODE=1

When CORNER_MODE=1 in MC_PATH_REF, the interpolation will meet every given point but will modify the velocity before the corner to direct the movement towards the next point and to reach a continuous ve-locity profile while doing so. A cubic interpolation is used from point-to-point in a way that a smooth velocity curve is achieved in every single direction (X/Y/Z). The acceleration is not taken in to account and just 3 consecutive points influence every specific movement.

The following diagram shows the 2 different interpolations for a very simple path.

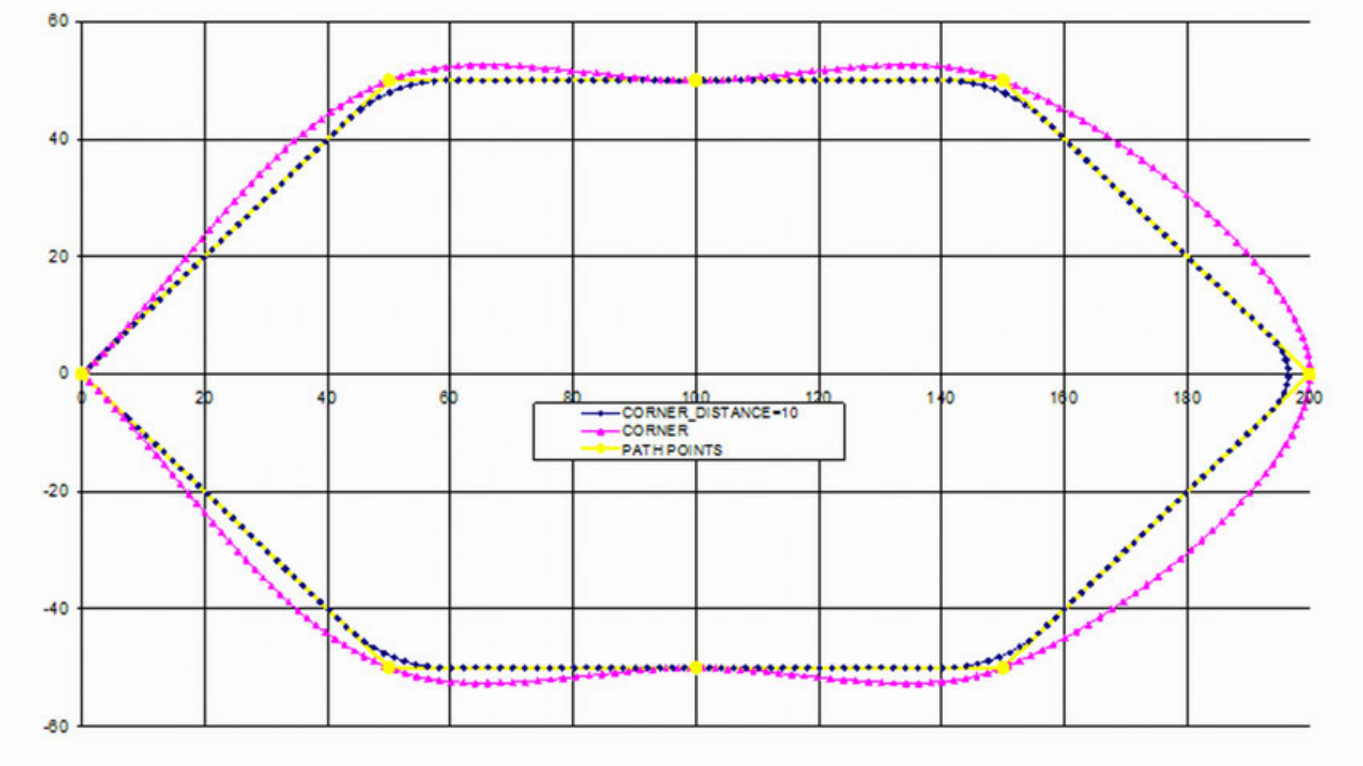


Fig. 271: POSITION comparison

Table 177: The points are:

X	Y
0	0
50	50
100	50
150	50
200	0
150	-50
100	-50
50	-50
0	0

The yellow curve shows the original points, the pink curve the interpolation result for CORNER_MODE=1 and the blue curve the result for CORNER_MODE=0 with CORNER_DISTANCE=10.

-CORNER_MODE=0: the given path, as a linear movement from point-to-point, is followed, just at the given distance from the corner (predefined X/Y-point) this path is left and the movement turned into the direction of the next point.

-CORNER_MODE=1: the interpolation meets the given points, but the linear path is left as a cubic interpolation is performed to achieve a smooth movement.

A second example is shown with far more points to define the curve. In this case, it seems almost as if there was no difference between the 2 modes:

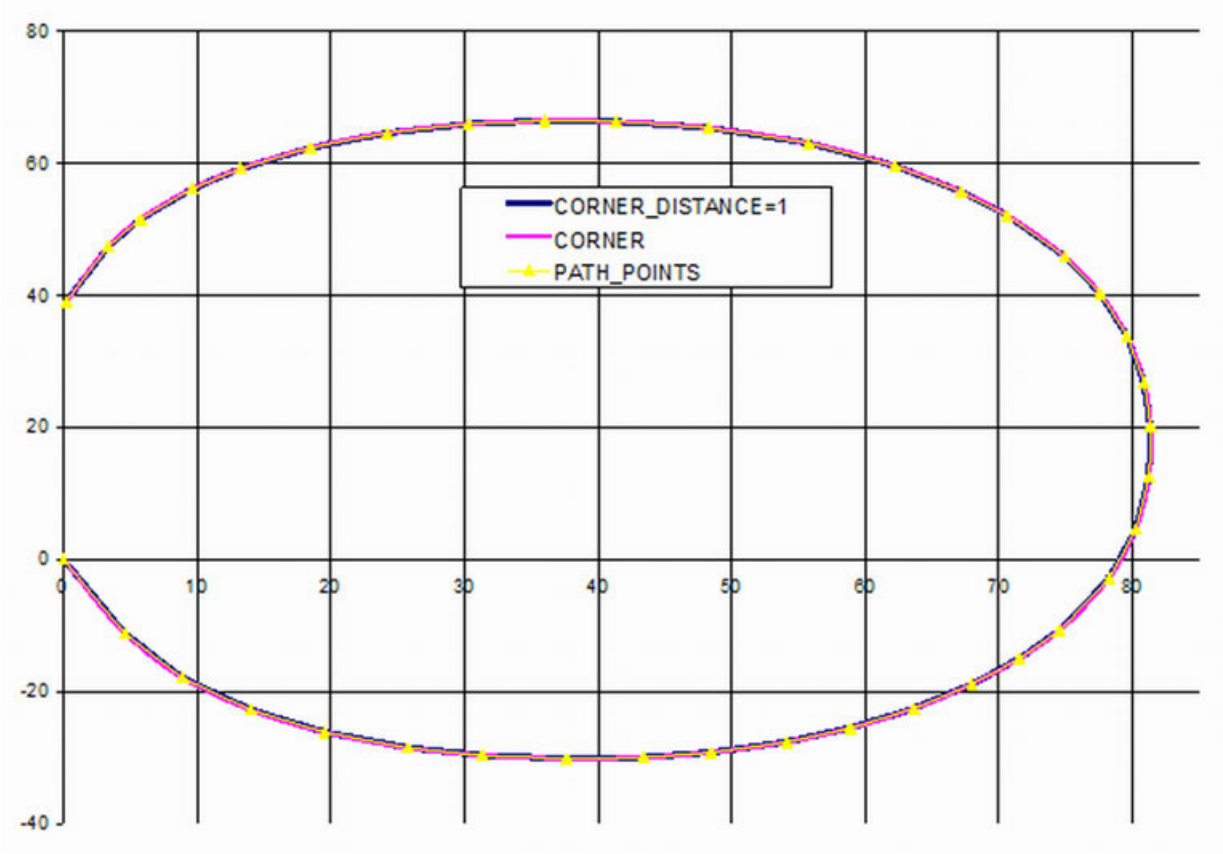


Fig. 272: POSITION comparison

The difference is to be seen when exploring the curve in more detail. In this case, a CORNER_DISTANCE=1 is used, so the interpolation follows the path somehow “edgy”, while CORNER_MODE=1 creates a rounded curve.

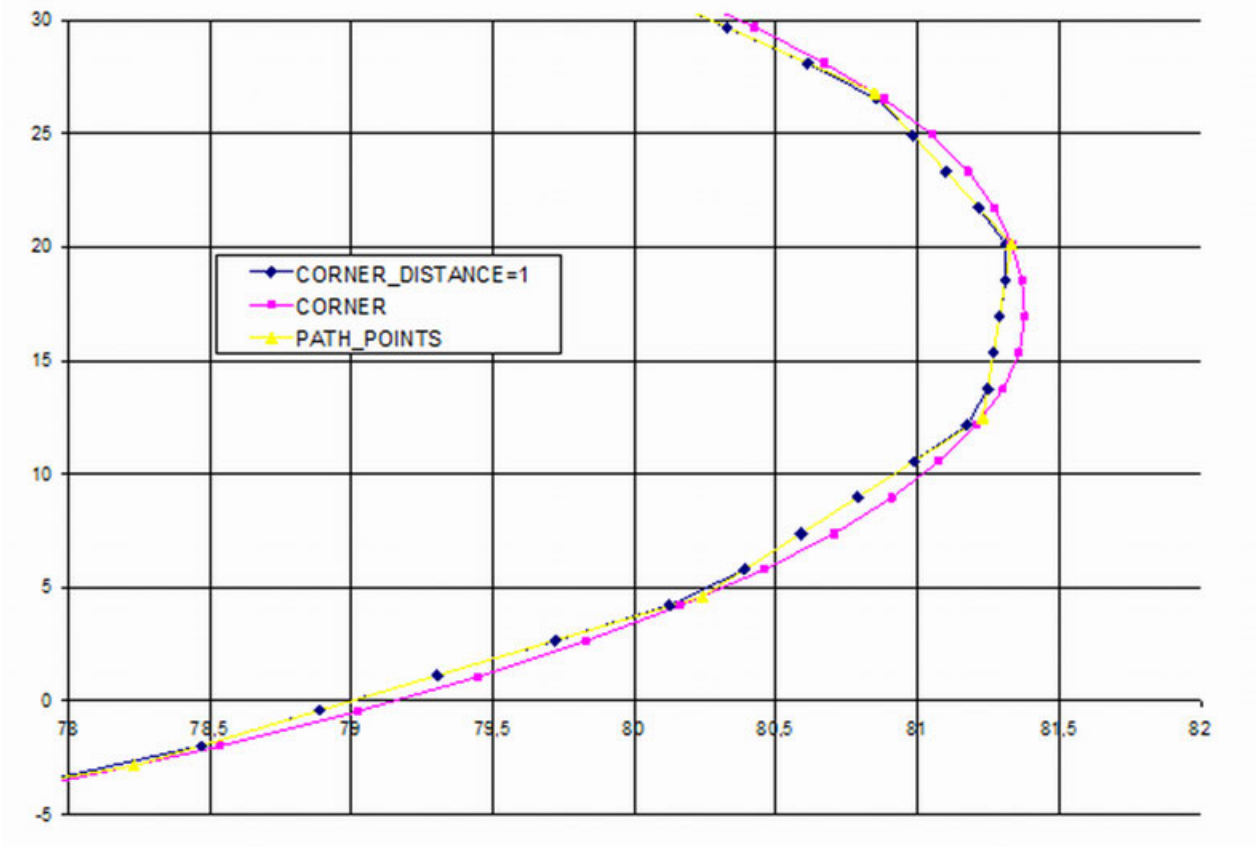


Fig. 273: POSITION comparison in detail

An even clearer difference is to be seen in the velocity curve:

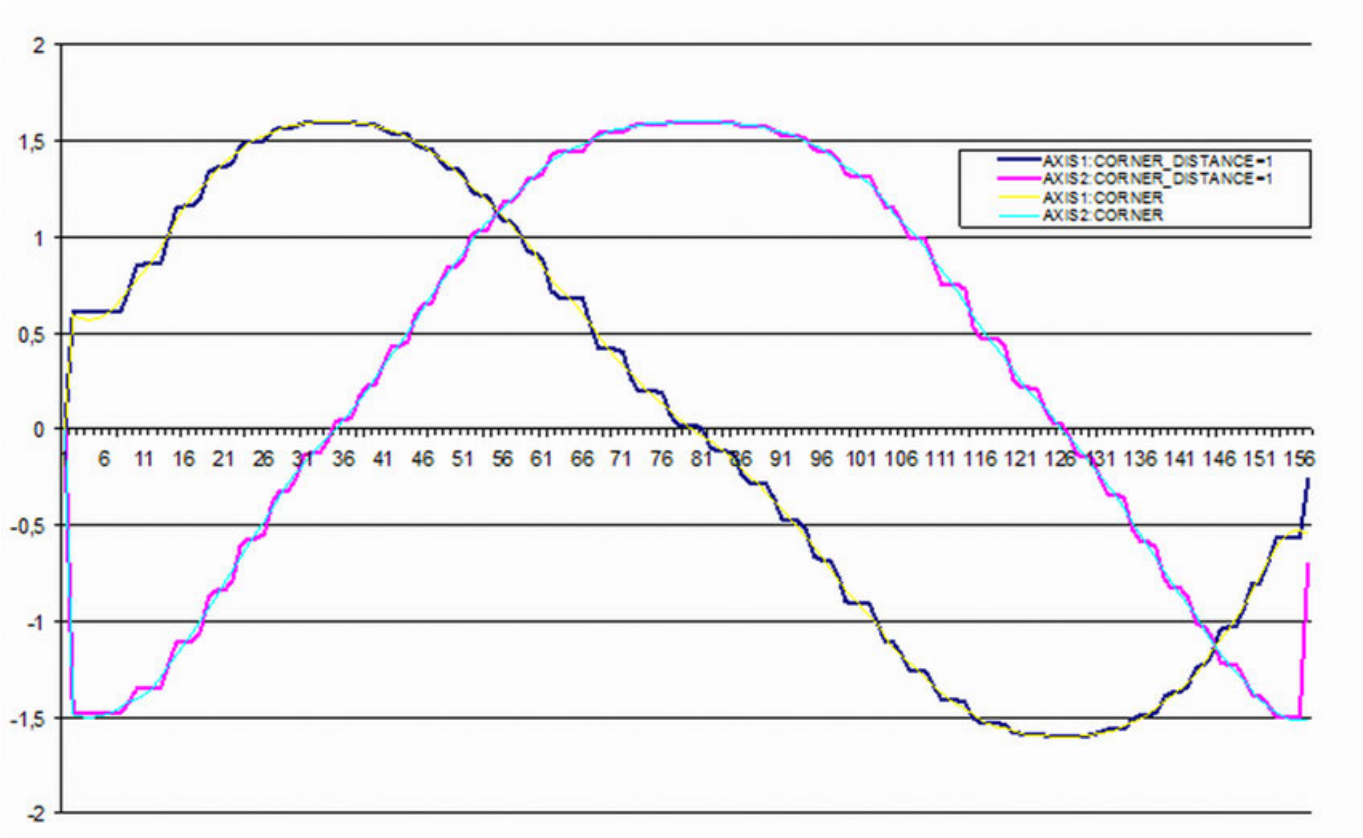


Fig. 274: VELOCITY comparison

-with CORNER_MODE=0, a constant velocity is maintained during the linear parts of the path. Just the section de-scribed by CORNER_DISTANCE (in this case, CORNER_DISTANCE=1) is used to change direction and velocity.

-with CORNER_MODE=1, a continuous velocity transition is achieved.

Model

Group state diagram

The group state diagram describes the commanded state of the group of axes. It is on top of the single axis state diagram *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*. While axes are in a group state, the single axis state diagram is also active per axis. Therefore interdependencies between the two types of state diagrams exist.

GroupDisabled is the initial state at power up where a group can be created. Issuing MC_GroupEnable leaves this state.

The next state is GroupStandby. In this state the group is enabled and no function block has control on one of the axes in the group. In this state the group can additionally be altered and homed if needed (State GroupHoming).

In the state GroupHoming a homing sequence can be defined for a group of axis. This can be applicable due to the mechanical constraints of multiple motors. For example in an mechanical construct looking like the letter “I“ with two motor mechanically coupled via one band or belt moving over the form of the letter I, need to be homed differently.

If a function block has control on (one of the axis of) the group, the state changes to Group-Moving.

GroupStopping is a special state that deals with the MC_GroupStop command, which automat-ically tranfers to the state GroupStandby as soon as Done = SET and Execute = FALSE in MC_GroupStop.

In case an error arises (in one of the axis) the state changes to GroupErrorStop, which can only be left via issuing MC_ResetGroup.

Explanations

Group motion commands will always lead to a Synchronized Motion state in the single axis state diagram. In case of a GroupStandby all axes of the group are also in single axis state StandStill.

A GroupErrorStop will not lead to ErrorStops of the grouped axes as the error may only affect the group. In case of a single axis ErrorStop the Group will also change to GroupErrorStop as the single error effects the group.

The group state diagram reflects the state of the group and the issued function blocks.

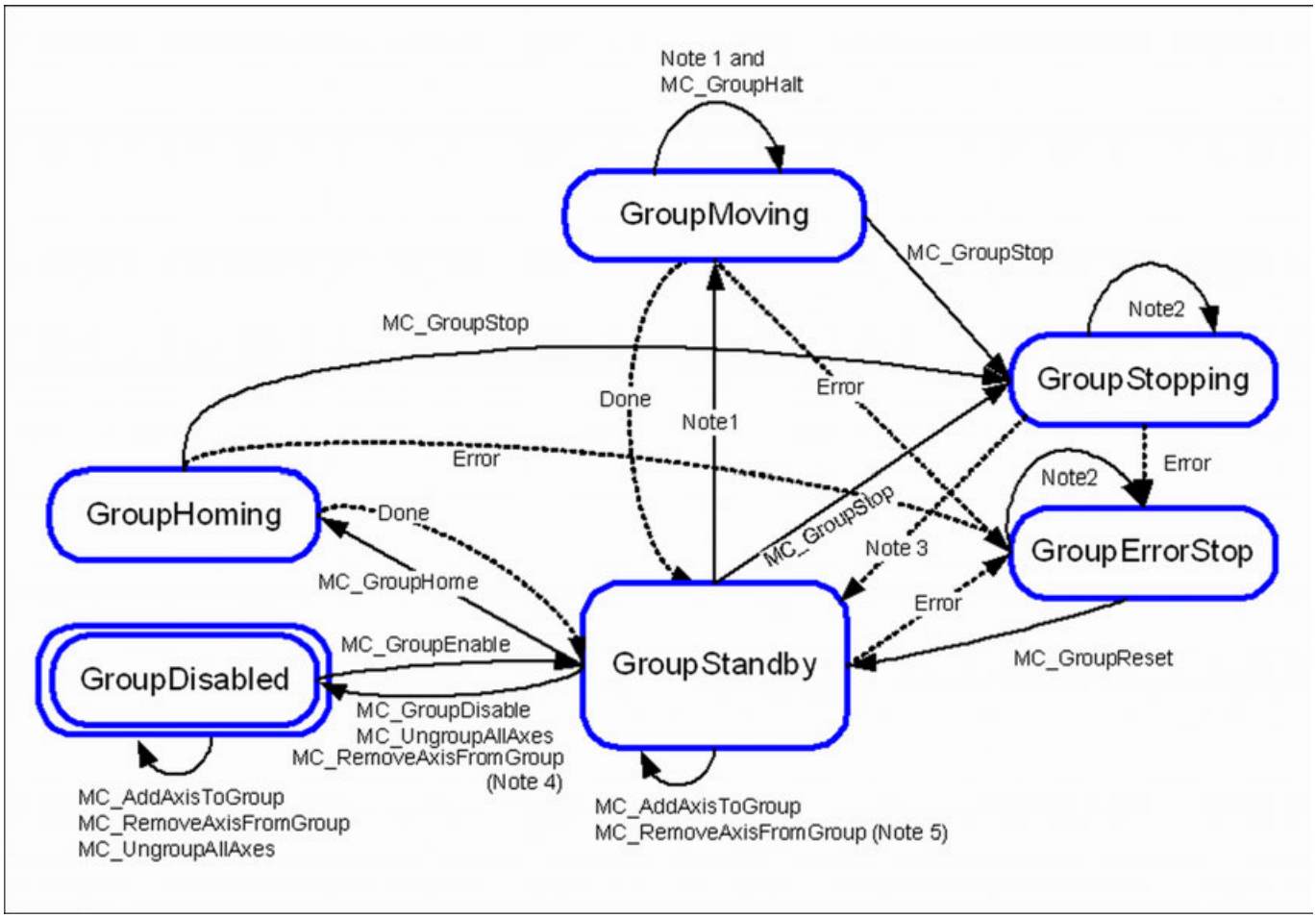


Fig. 275: The group state diagram

Relationship single axis and group state diagrams

Example of the relationship between three single axes combined in an axes group.

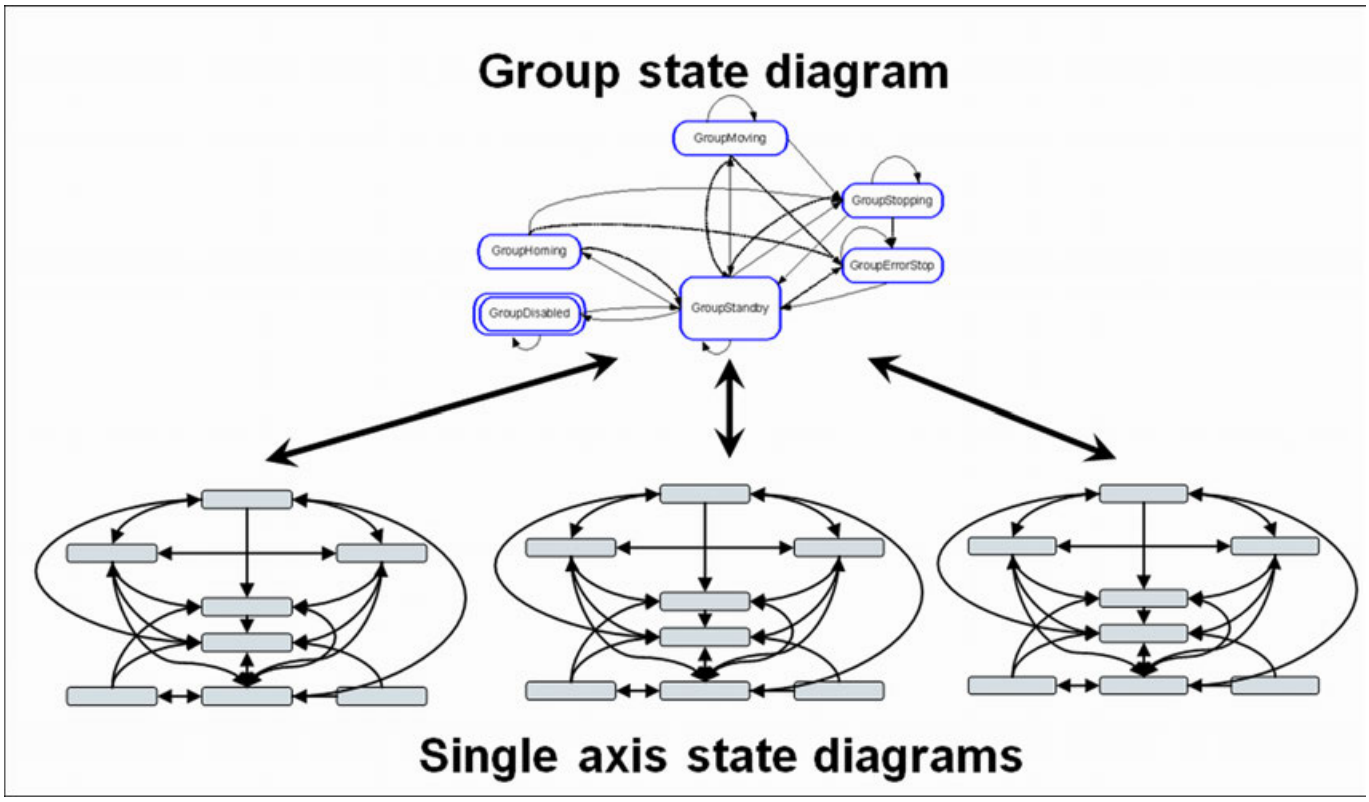


Fig. 276: Relationship single axis and group state diagrams

When a number of axes are grouped, and a single axis command, like MC_MoveAbsolute, is issued to an axis in this group, there are basically three options:

- Not allowed. Issuing a single axis command is not accepted and not performed: it signals this by setting the er-ror output of the applicable (issued) single axis function block. There is no change to the group, and as such continues their movements.
- Aborting the current group command(s), as well as following group commands, and continue with the single axis command only. The remaining axes of the group move to the state StandStill (via an implicit MC_Halt per axis). The original trajectory will not be finalized.
- Superimpose the single axis commands to the group commands.

This specification does not restrict to any of these options. This means that different implementations of this behavior will exists, and the supplier of the system has to specify what their system does support.

General rules for the interaction between a single axis towards its groups (for all 3 options above):

- If at least one axis in the group is moved by a command then the group is in the state GroupMoving.
- If all axes are in StandStill, the group can be in the state GroupStandby, GroupDisabled or GroupErrorStop.
- If one axis in a group is in ErrorStop, the whole group is in GroupErrorStop.
- If a single axis MC_Home is issued the group is in state GroupMoving.
- If a single axis MC_Stop is issued the group is in state GroupMoving.
- If supported by the system, it is allowed to disable a single axis of the axis group without influencing the axes group state. This can be useful to save energy or to apply a mechanical brake for a single axis not involved in the on-going motion.

General rules for the interaction between a group and the single axis in it (for all 3 options above):

- If the group is commanded by a group moving command, all the single axes in the group are in the state Synchronized Motion.
- If the group is in the state GroupStandby, the states of the single axes do not have to be all in StandStill.
- If the group is in the state GroupErrorStop the state of the single axis is not affected.

Table 178: Overview of the influence of group motion commands on a single axis state

Command	Group State	Axis State
MC_MoveLinearXxx MC_MoveCircularXxx MC_MoveDirectXxx MC_MovePath MC_GroupHalt MC_TrackConveyorBelt MC_TrackRotaryTable	GroupMoving	Synchronized Motion
MC_GroupStop	GroupStopping / Group-Standby	Synchronized Motion / Stand-Still
MC_GroupReset	GroupErrorStop / Group-Standby	Not relevant for Axis
MC_GroupHome	GroupHoming	Synchronized Motion

Explanation: A stopping group leaves the single axis in Synchronized Motion as none of the single axis performs a single axis stop.

Input execution mode

The input MC_EXECUTION_MODE is an ENUM providing information on the behavior of administrative function blocks.

The modes are:

- Immediately - the functionality is immediately valid and may influence the on-going motion but not the state.
- Delayed - The functionality is valid when the ongoing motion command sets one of the following output parame-ters: Done, Aborted or Error. This also implies that the output parameter Busy is set to FALSE.
- Queued - The new functionality becomes valid when all previous motion commands sets one of the following output parameters: Done, Aborted or Error. This also implies that the output parameter Busy is set to FALSE.

General rules

The following chapter explains some rules on the usage of the libraries.

- A general rule is that ALL PLCOpen function blocks which are used for a specific drive are to be used in the same PLC-task. When multitasking is used for the PLC, it is allowed to have different drives in different tasks, but all Funciton Blocks belonging to a specific drive need to be in the same task. There is no multithreading protection for the AXIS_REF instance.
- The CMC_MotionKernel… and the COMC_Group… function blocks might be in a different task then the PLCopen Blocks (MC…). All function blocks belonging to the same fieldbus Communication Module have to be called from the same task.

- When AXIS_REF or AXES_GROUP_REF is used as input on a user defined FUNCTION_BLOCK or PROGRAM or FUNCTION, then ALLWAYS use it as VAR_IN_OUT and NEVER use it as VAR_INPUT or VAR_OUTPUT. The reason is that this would
  - Break the consistency and destroy data
  - Consume a lot of computing power by copying data.
- The "Min update time" update time for the fieldbus, defined under *“PLC Configuration ➔ Communication Modules[FIX] ➔ <fieldbus master type>”* must not exceed the half of the scantime of the PLC-task. E.g. scantime of PLC-task is 5ms, then "Min update time" should not be greater than 2ms.
- The functionality of CompactMotion_AC500 library may be combined with the CoordinatedMotion_AC500 library. All single axis functions could still be used for an axis, even when it is combined with others to a group .

Axes grouping

Within this specification for interpolation, the related axes are grouped in an “AxesGroup”, and can be accessed via the type AXES_GROUP_REF. The relationship between the different axis levels and groups is shown hereunder.

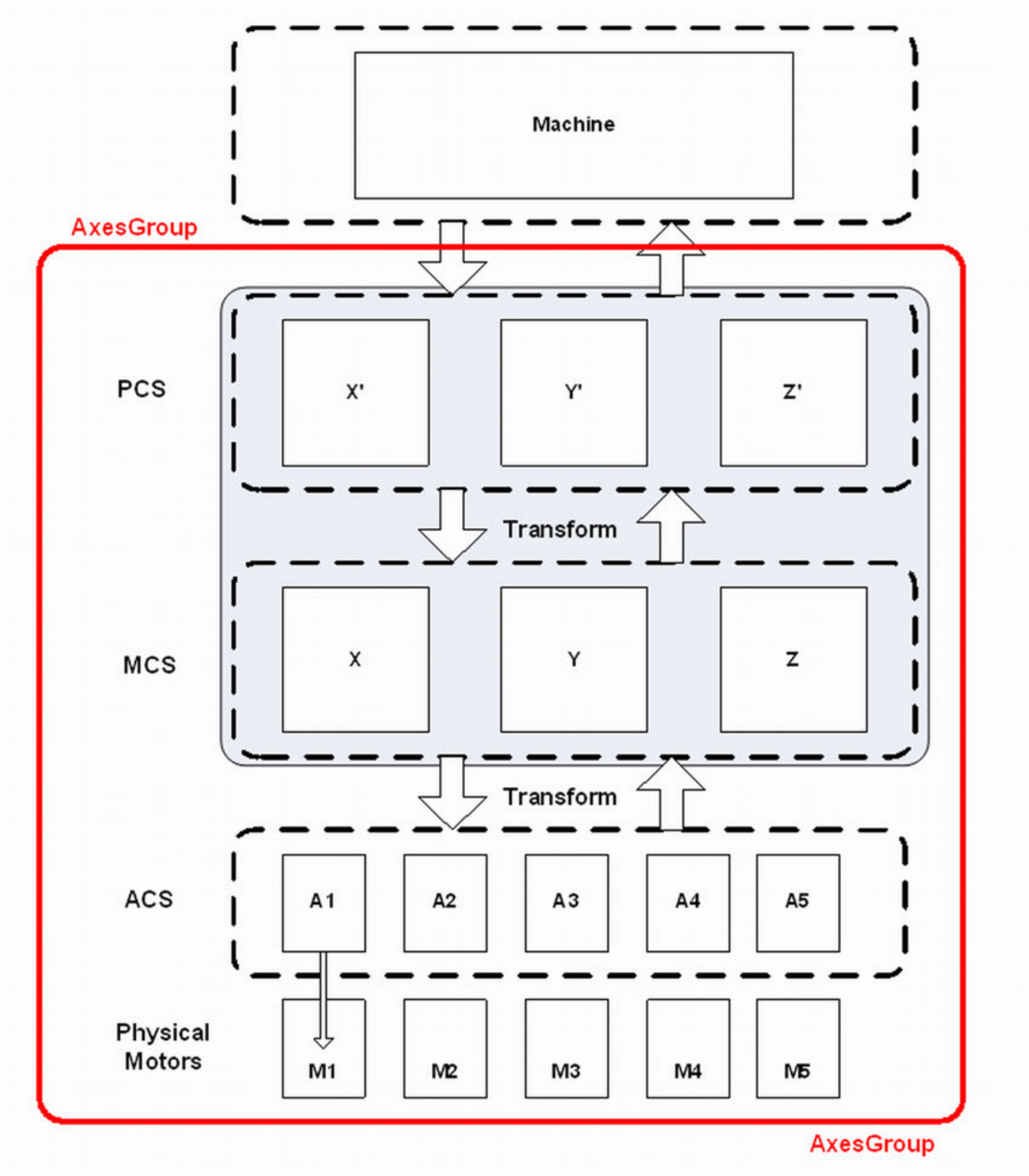


Fig. 277: Overview AxesGroup

The AxesGroup shown in red above provides the interface to the user of the group of axes.

AC500: All group movement is applied to the MCS or PCS axes. The output "isActivePCS" from the transfor-mation block shows if a PCS system is active.

- *isActivePCS=true => all group movement is done in the PCS (product coor-dinate system).*
- *isActivePCS=false => all group movement is done in the MCS (machine coordinate system).*

A movement in ACS coordinates could be done by using single axis blocks and the respective ACS axis as AXIS_REF.

Parameters in the AxesGroupRef can include remaining time and remaining distance before target position (or velocity or equal) is reached.

↳ *Chapter 1.5.9.7.1.20 "COMC_GROUP_CARTESIAN" on page 2994*

Axes group synchronized motion

The function blocks MC_SyncGroupToAxis ↳ *Chapter 1.5.9.7.1.18 "MC_SyncGroupToAxis" on page 2988* and MC_SyncAxisToGroup ↳ *Chapter 1.5.9.7.1.19 "MC_SyncAxisToGroup " on page 2991* deal with a master/slave relationship between a single or group of axes and a single or group of axes for coordination purposes.

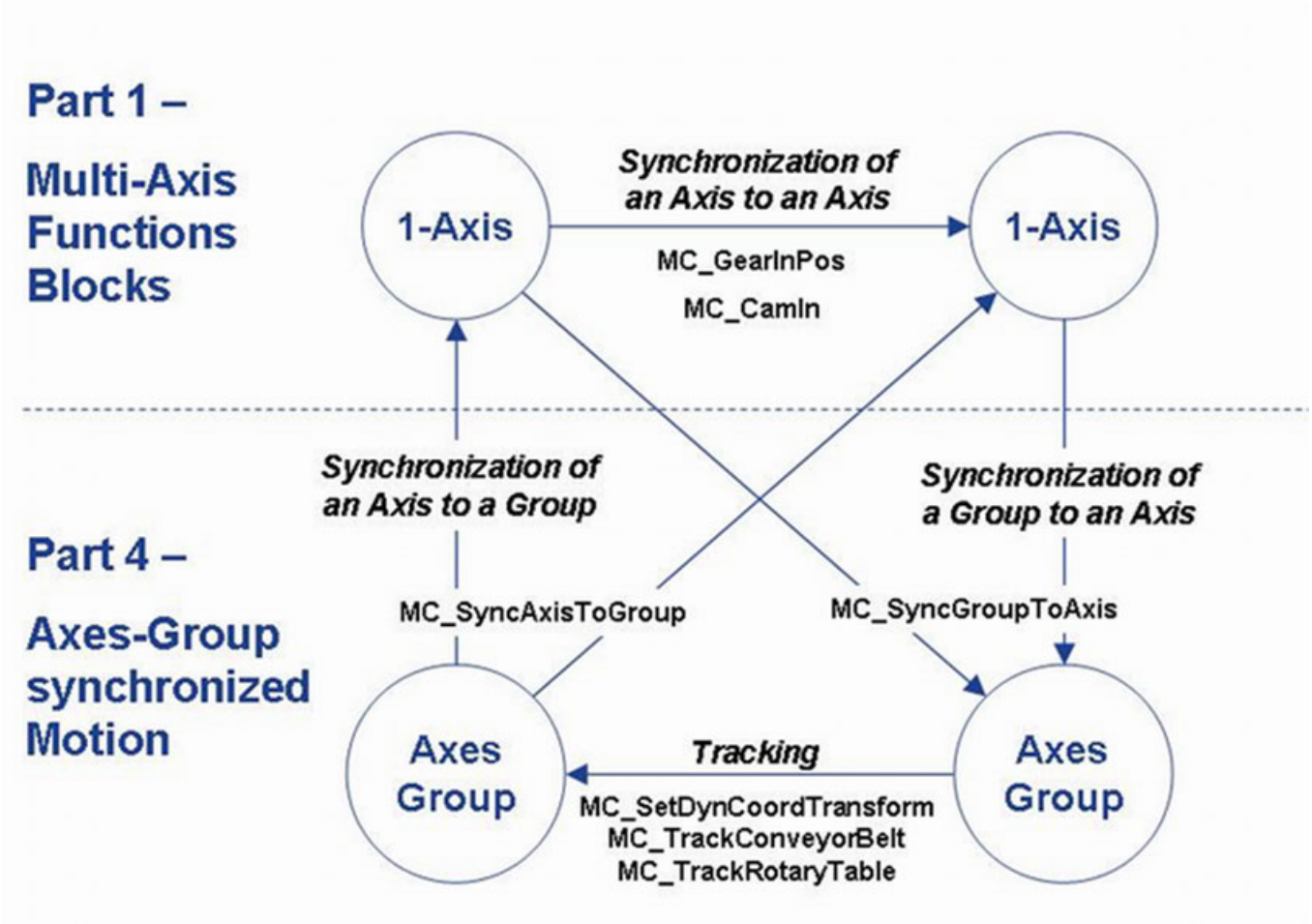


Fig. 278: Graphical explanation of coordination

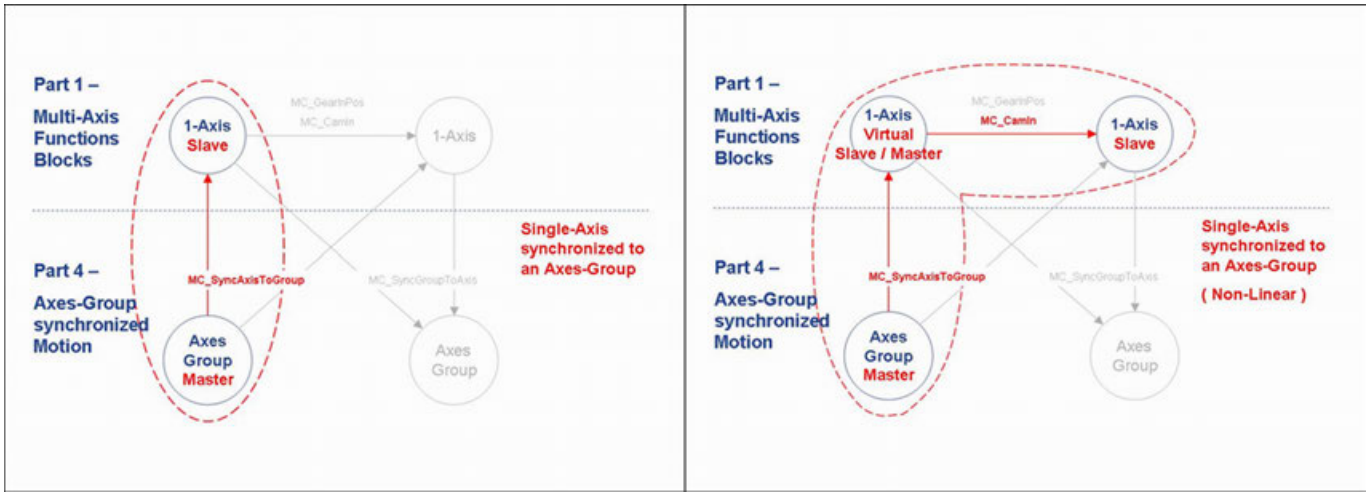
There are two kinds of coordinated motion that have to be distinguished from a programming point of view and in the realization of the motion control itself. These two modes are identified here through their names:

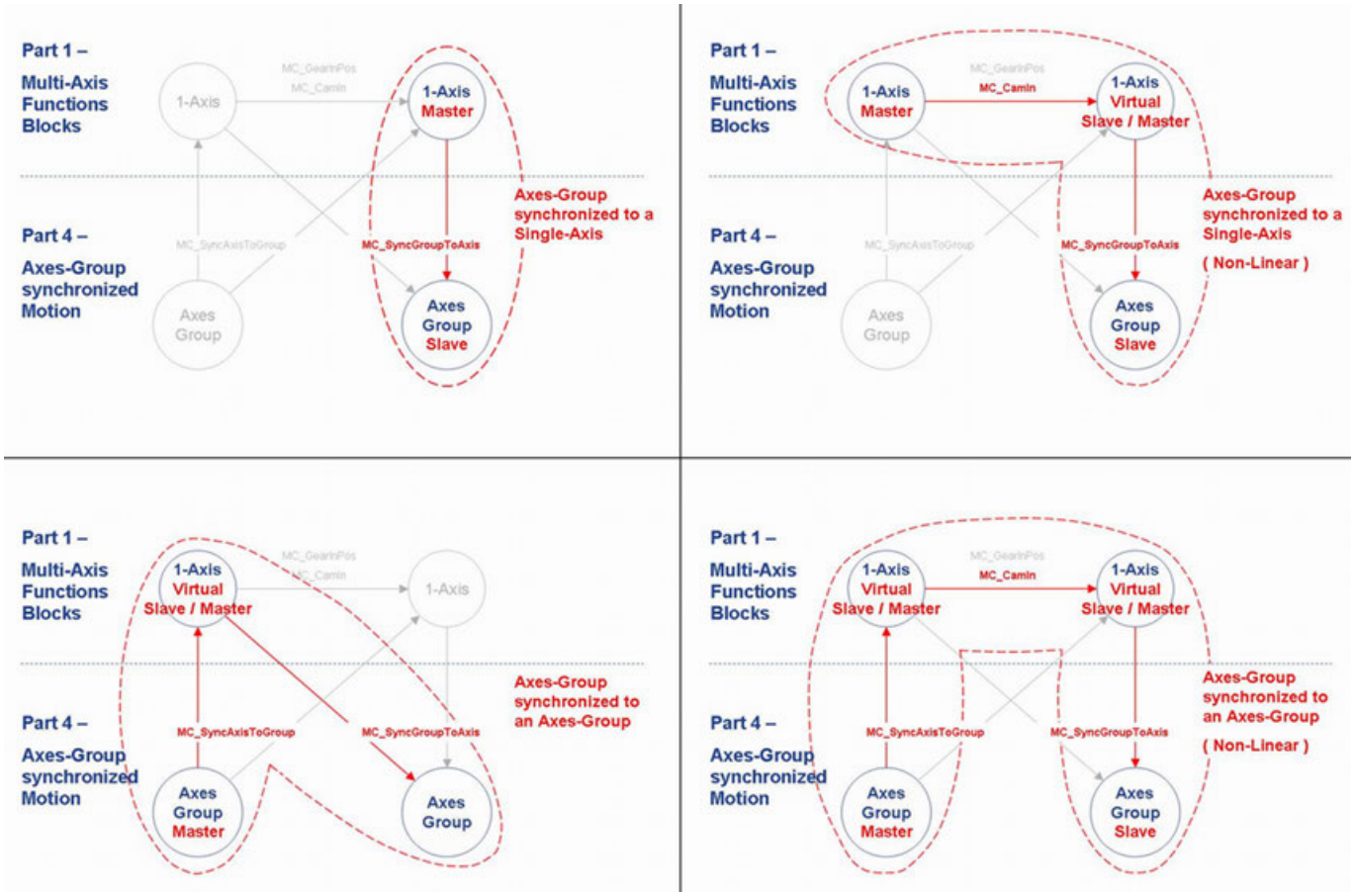
- Synchronization
- Tracking

Synchronization

The relationship between single axis commands and synchronized motion is shown here. There are 6 possibilities of associations:

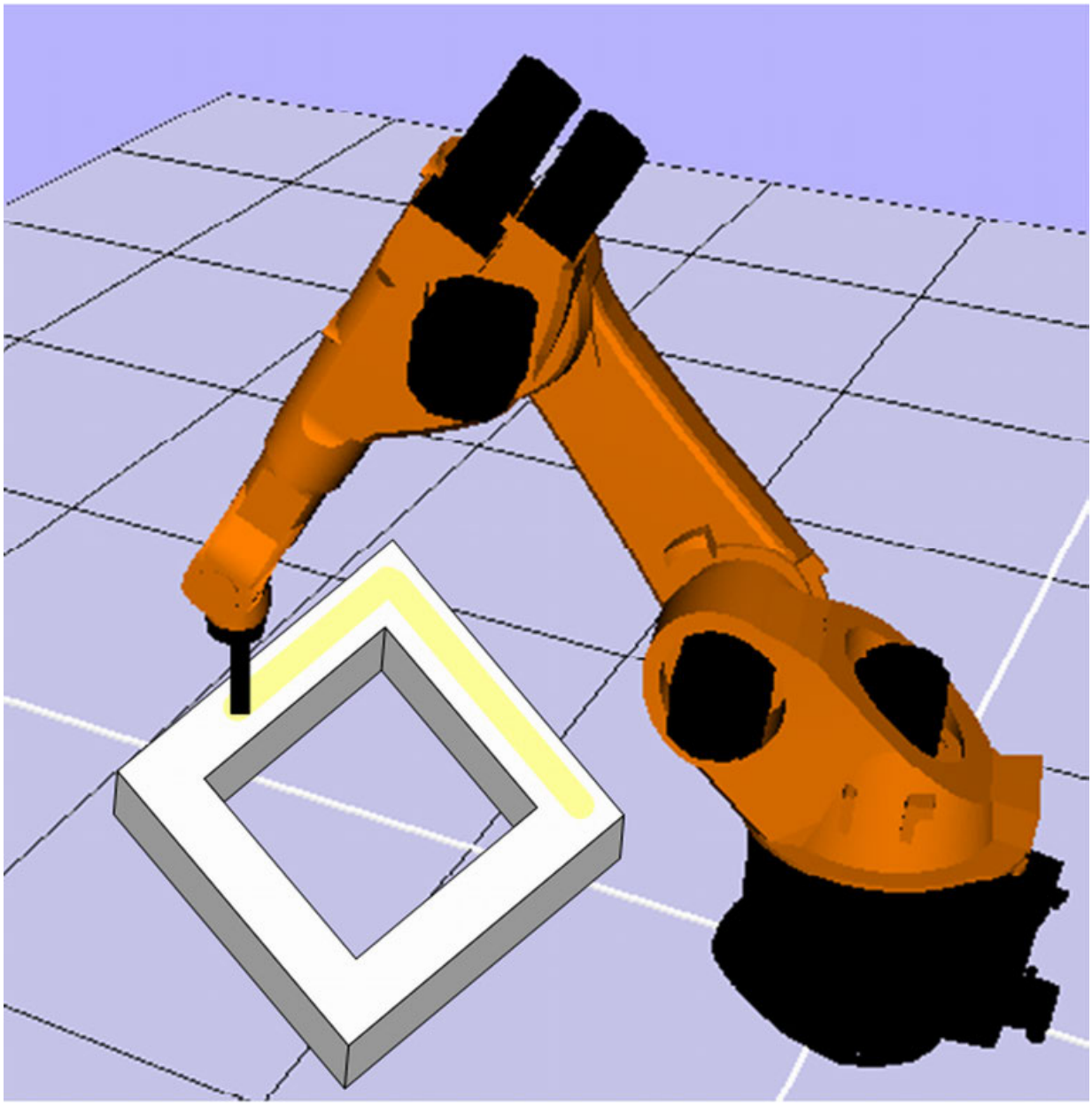
1. Single-axis synchronized to an Axes-Group, Linear synchronization,
2. Single-axis synchronized to an Axes-Group, Non-Linear synchronization (using MC_CamIn),
3. Axes-Group axis synchronized to a Single-axis, Linear synchronization,
4. Axes-Group axis synchronized to a Single-axis, Non-Linear synchronization (using MC_CamIn)
5. Axes-Group axis synchronized to an Axes-Group, Linear synchronization,
6. Axes-Group axis synchronized to an Axes-Group, Non-Linear synchronization (using MC_CamIn).





Synchronization of single axis to an axes group

This is an example of a single axis (as slave) synchronized to an axes group (master). The master follows its path and the slave is linked to the position, velocity, acceleration, or any other magnitude of the master. An example is glue dispensing, where the amount of glue to be dispensed is coupled to the velocity of the TCP of the robot. The single axis slave motor movement of the glue dispenser is coupled to the trajectory of the TCP of the group over the surface of the object via MC_AxisFollowGroup. Alternatively, if the position information is not critical, one can use MC_GroupReadActualVelocity, perhaps combined with a gearing factor, and thus providing the input to the motor of the glue dispenser.



Synchronization of an axes group to a single axis

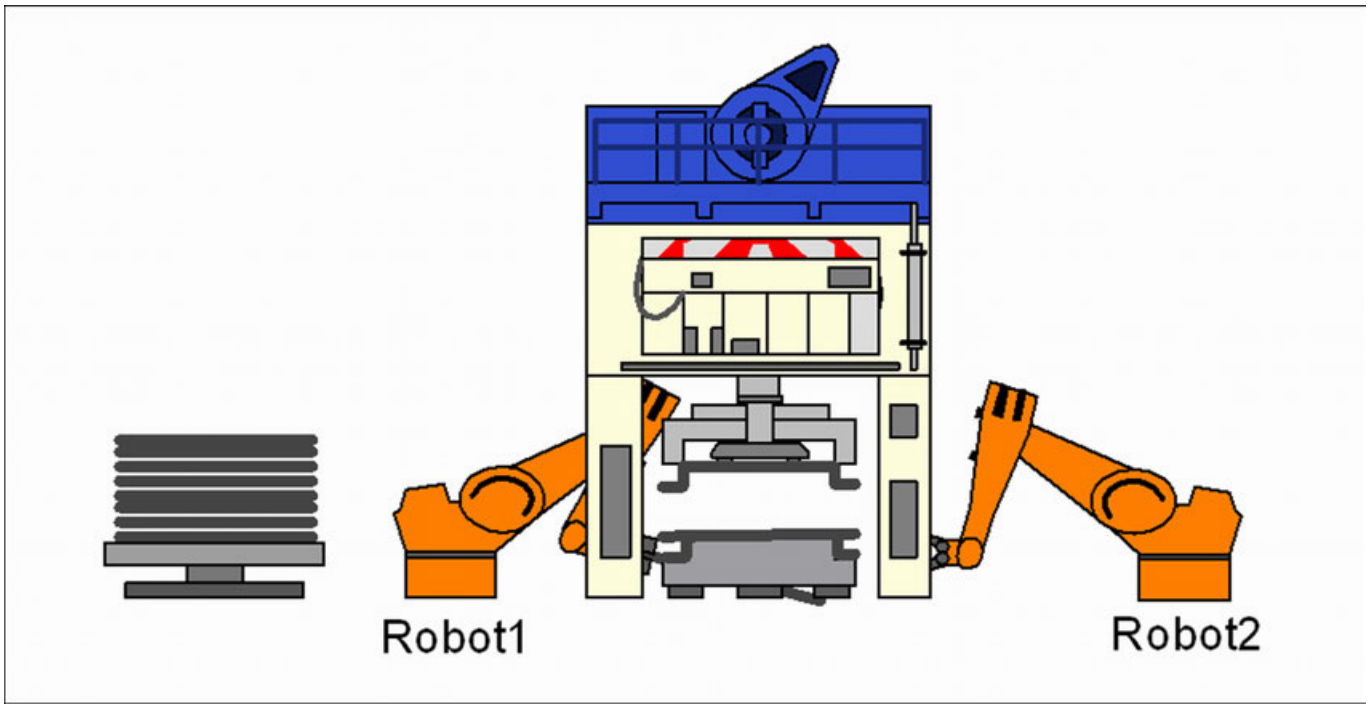
This mode combines an axes group (as slave) with an axis as master in order that the slave executes its path with synchronization to the progress of the master, meaning linked to a 1-dimensional source for synchronization.

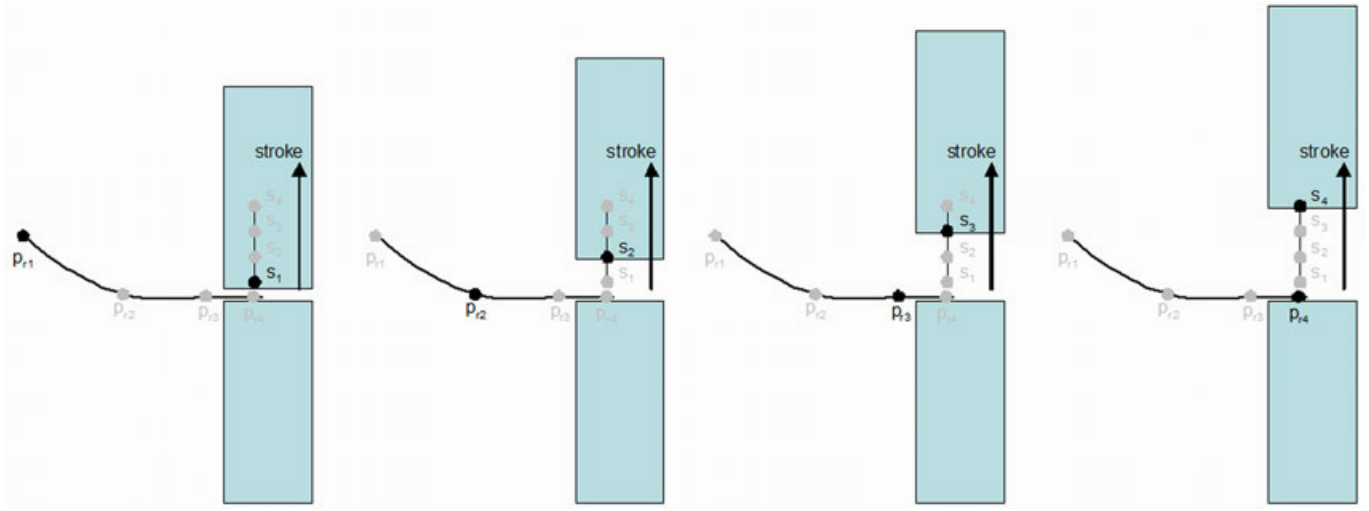
Examples here include press synchronization. As an example of synchronization between a master single axis and a group can be the robot which places material in a press machine:

The robot has to synchronize to the opening phase of the press. A second robot also synchronizes to take the material out. The master is the (single or virtual) axis controlling the press. The axes group for both robot 1 and robot 2 has to follow the press in a certain area of the master movement:

Opening for robot 2 to take the product out and closing for robot 1 to add the new product. For this synchronization the function block MC_SyncGroupToAxis is defined.

In case the slave is an axes group, a transformation to a virtual axis can be applicable to generate the 1-dimensional synchronization data. In case both the master and the slave are axes groups, a virtual master axis on the slave side is applicable (linked via cam profiles to the different axes) to use the 1-dimensional synchronization data of the master side.



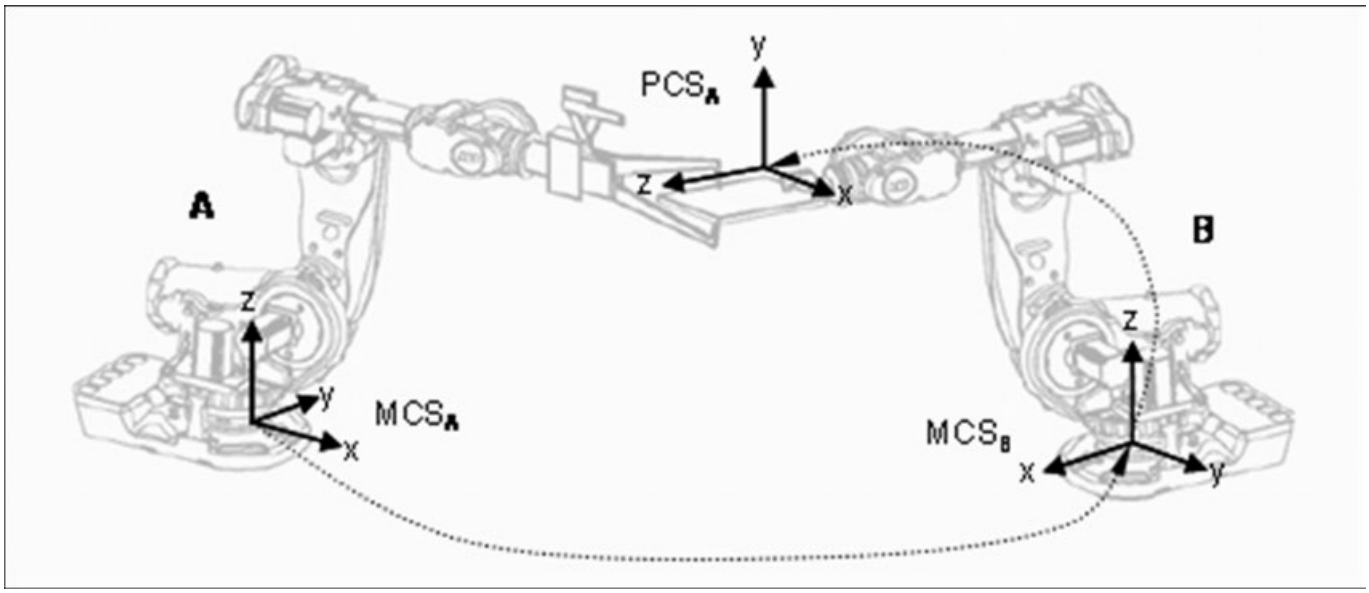


Tracking

Tracking is characterized by an axis group (A) that follows with its movement the movement of a single axis or another axis group (B). During the coordinated following A is performing a movement/task relative to the movement of B. The tracking data is a multidimensional source incl. position and orientation. Solutions can include a moving coordinate system or a multidimensional gear functionality. Tracking can be seen as a superposition of two movements, although these movements are independent. One, which is the movement of the product (moving PCS) and the second one, which describes the path of the TCP that would be executed if the product is standing still (Positions have to be defined in PCS). The Position of the PCS and therefore also the movement of the PCS relative to MCS is described by the coordinate transformation MCS to PCS. For tracking the following function blocks are defined here: MC_SetDynCoordTransform as a general one, and MC_TrackConveyorBelt plus MC_TrackRotaryTable for specific applications.

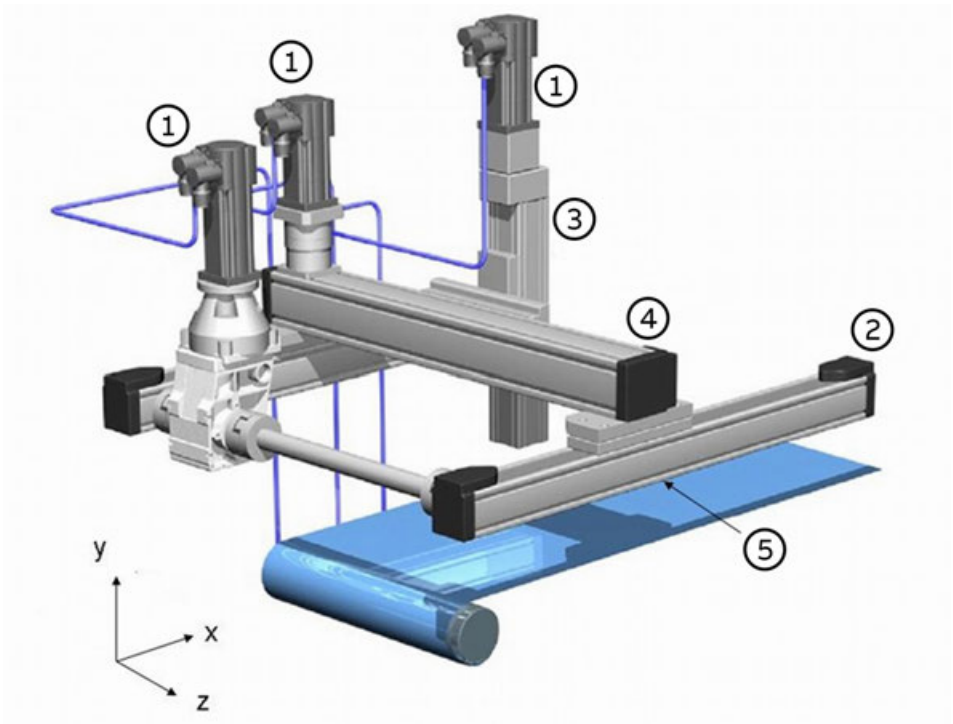
> *The considerations on the limitations of the dynamics or mechanics are implementation specific.*

Basic example: The basic example for the tracking of an axis group and a single axis is conveyor tracking, where the robot picks or places parts on the moving conveyor or is putting some crème on a cake moving on the belt. An example for the tracking of another axes group is having two robots, where robot B is holding a work piece, and robot A is performing some welding on the part at the same time B is moving the work piece (see figure.



Generally there is no difference if A is tracking a single axis or an axis group, when thinking of a single axis as an axis group having only one axis but also a kinematic (even if it is very simple). Then concepts of the motion planning as well as of programming are the same.

Example: A second tracking example deals with synchronization of a group and a transportation belt. The group synchronizes to the belt, which is the master. We have a (simple) Cartesian robot, consisting of 3 motors moving 3 axis. Application examples are to pick something from the belt (with a correction in the Z-position), or to put some cream on a cookie that is lying on the belt.



1 Motor
2 X-axis
3 Y-axis
4 Z-axis
5 TCP

1.5.9.5 Drive-Based motion control

1.5.9.5.1 Drive-based motion architecture

To connect a real axis to the PLCopen function blocks, a certain driver function block has to be used for every axis type. These function blocks are named XXX_ACCESS and are realized for a variety of drives. By creating new driver function blocks, additional drives or additional types of motion control software might be included.

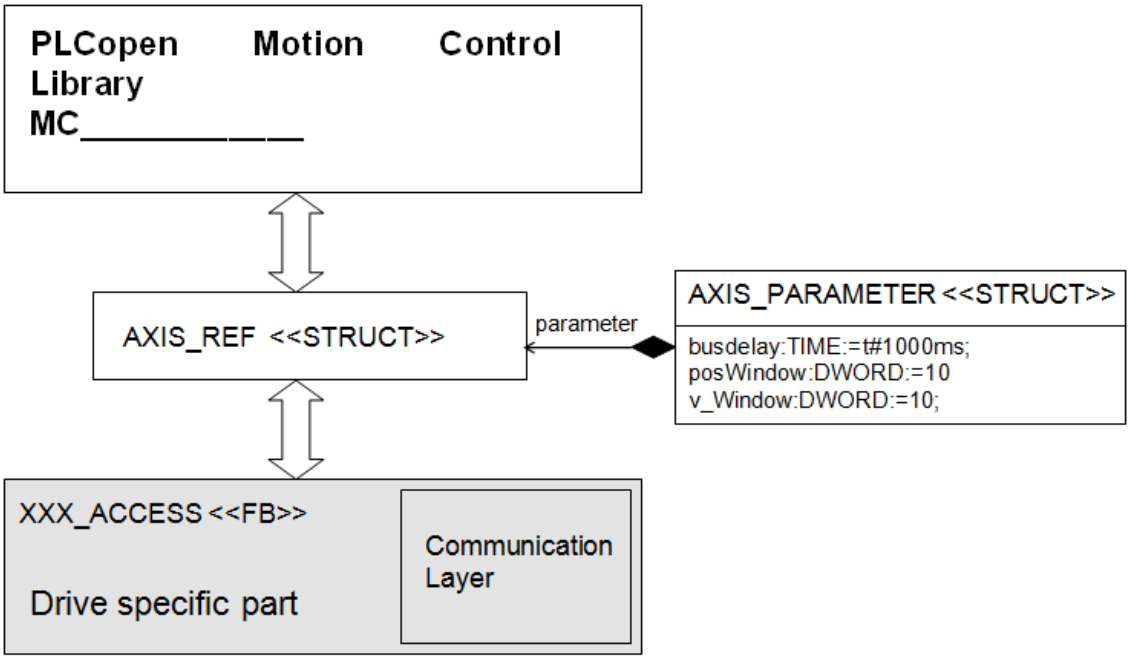


Fig. 279: Architecture

The usage of the “XXX_ACCESS” is dependent on the different drive types. For all drives, the structure AXIS_REF holds an element parameter from type AXIS_PARAMETER. This is initialized with some default values but might be adjusted to the specific application. ↳ *Chapter 1.5.9.5.2.3 “ACSM1_ACCESS_dc driver unit in decentralized motion control” on page 2725*
↳ *Chapter 1.5.9.5.3.3 “ACS350_ACCESS_dc” on page 2736*

1.5.9.5.2 Realization with ACSM1 on PROFIBUS DP network

General restrictions

Restrictions for the available function blocks

- As buffered mode, MCAborting is realized as a default.
- The “Jerk” is not available at the function block. When it is used, it is possible to send the respective parameter separately to the drive or to adjust it with the drives configuration software.
- From the Extended inputs and outputs at the function blocks, the following are not realized:
  - BufferedMode: The realization for ACSM1 just supports by default the MCAborting mode.
  - Jerk: A maximum jerk could be adjusted by parameters of the drive.
- For the parameter number (WriteParameter, ReadParameter), the following options are available:
  - The PROFIDRIVE parameter number.
  - The ACSM1 parameter number + 40000
    This manipulation of the parameter number is necessary to distinguish between the different types of parameter number.

Preconditions

- An instance of the AXIS_REF structure has to be created.
- A connection to the drives fieldbus interface has to be established.
- The drives parameter need to be adjusted to match the requirements.

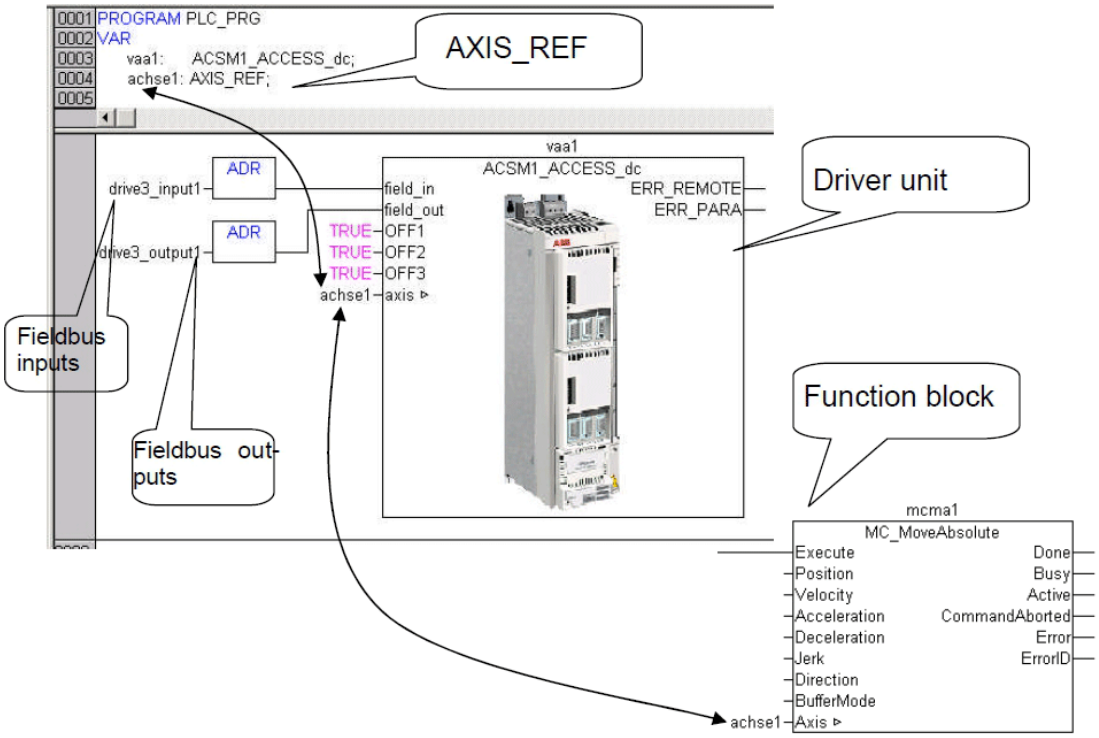


Fig. 280: Usage of function blocks

ACSM1_ACCESS_dc driver unit in decentralized motion control

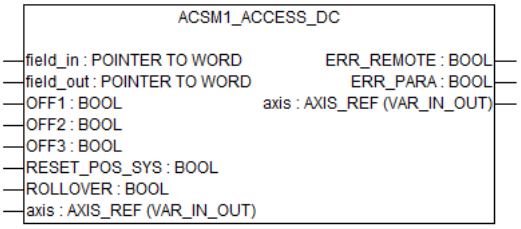


Fig. 281: Function block ACSM1_ACCESS_dc

To make this function block work, the following additional information should be considered:

↳ *Chapter 1.5.9.5.3.5 “Adjustment of parameters for drive” on page 2739*

↳ *Chapter 1.5.9.5.3.6 “MC_VISU_ACS350_mcw, MC_VISU_ACS350_msw” on page 2740*

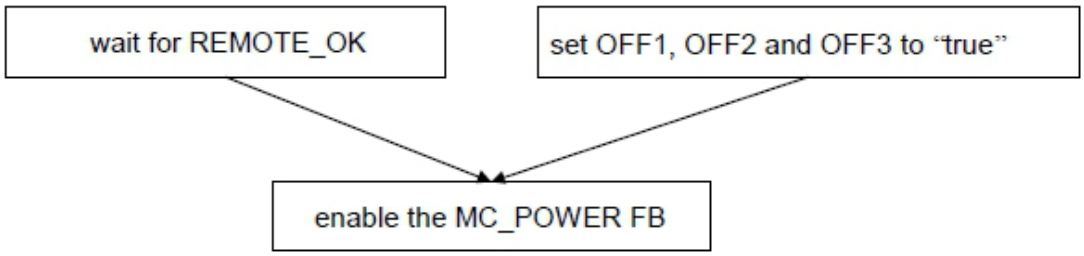
This function block is used as driver unit for PLCopen function blocks with ACSM1 in decentralized Motion Control.

During normal operation, the drive will be completely controlled by the fieldbus interface. For safety reasons, this could be combined with additional measures through binary inputs on the drive. So the drive would be in defined state when the fieldbus is not yet initialized or in case of communication problems. For coordination, the inputs OFF1, OFF2 and OFF3 are available at the driver unit function block ACSM1_ACCESS_dc. Adjust parameters in Group 10 of the drive when this additional control should be used. Then connect the binary values OFF1, OFF2 and OFF3 to the respective inputs.

When the fieldbus interface is ok (shown by REMOTE_OK) OFF1, OFF2 and OFF3 should be switched to TRUE before the MC_Power function block is enabled. When an emergency stop should be executed, the sequence depends on the required Stopping mode. (Details can be found in the drive manual.)

The priority of the signals is as follows: OFF2 > OFF3 > OFF1

CAUTION!
When switching off the drive ACSM1 via the MC_Power and ACSM1_ACCSESS_dc blocks, the drive is completely switched off. This results in limited use of the “brake holding function” of the drive, as for example the drive parameter 35.04 BRAKE CLOSE DELAY is no longer effective. This can result in increased wear of the holding brake depending on the application, in particular with suspended loads.



1. Use this sequence to power on the drive.
2. Use the following sequence to power on the drive:

   Set OFF1 or OFF2 or OFF3 or a combination to FALSE, depending on required stop mode. This will already reset the internal power-on state machine to the respective state and change the signals in the fieldbus CW (control word). In addition, disable the MC_Power function block. A sequence for these two actions is not required, but the MC_Power will show an error message when it is not disabled while the drive is already stopped.

If the Enable input at MC_Power is set to FALSE, bit 3 (ENABLE) in the control word is set to zero.

- *If the input OFF1 at the block ACSM1_ACCSESS_dc is set to FALSE: Bit 3 (ENABLE) and Bit 0 (OFF1) in the control word are set to zero.*
- *If the input OFF2 at the block ACSM1_ACCSESS_dc is set to FALSE: Bit 3 (ENABLE) Bit 0 (OFF1) and Bit 1 (OFF2) in the control word are set to zero.*
- *If the input OFF3 at the block ACSM1_ACCSESS_dc is set to FALSE: Bit 3 (ENABLE), Bit 0 (OFF1), Bit 1 (OFF2) and Bit 2 (OFF3) in the control word are set to zero.*

Input description

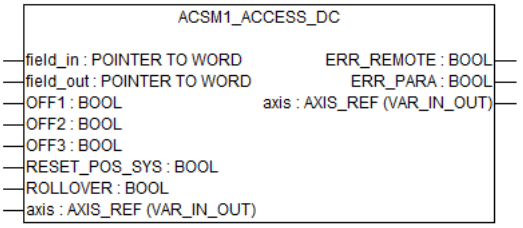


Fig. 282: Function block ACSM1_ACCESS_dc

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

field_in Data type: POINTER TO WORD

At this input, the address of the input data from drive to PLC should be connected.

field_out Data type: POINTER TO WORD

At this input, the address of the output data from PLC to drive should be connected.

OFF1 Data type: BOOL

Stop along the currently active deceleration ramp.

OFF2 Data type: BOOL

Power off the motor.

RESET_POS_SYS Data type: BOOL

Matches Bit3 in parameter POS_STYLE1 (62.09), refer to ACSM1 documentation. Just use it combined with absolute positioning.

ROLLOVER Data type: BOOL

Matches Bit5 in parameter POS_STYLE1 (62.09), refer to ACSM1 documentation. Just use it combined with absolute positioning.

> *In case of a linear positioning axis, use input-parameter Direction=SHORTEST for absolute positioning.*

Axis Data type: AXIS_REF

This data structure is needed to connect this driver unit to the PLCopen function blocks.

Output description

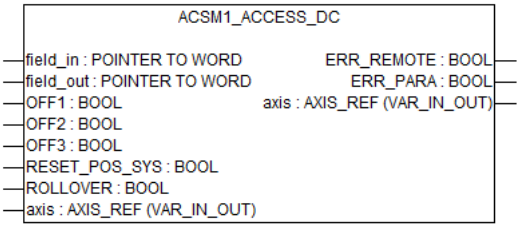


Fig. 283: Function block ACSM1_ACCESS_dc

ERR_REMOTE Data type: BOOL

This output indicates that no fieldbus connection to the drive is available. When this is shown, the PLC is not able to control the drive via fieldbus. This has to be considered in the PLC application to switch off the drive by other measures.

ERR_PARA Data type: BOOL

This output indicates that a parameter access was not successful. The reason could be a wrong parameter number at function block or the drive configuration.

Adjustment of parameters for drive

The drive has to be adjusted to use the PROFIDRIVE profile. To do this and to establish the PROFIBUS connection to the PLC follow the according documentation. In addition, some parameters have to be adjusted. These are listed in the following tables. For further commissioning, follow the chapter “Start-Up” in the drives manual (Link setzen auf?).

Parameter	Name	Value	Mnemonic	Comment
START/STOP				
10.1	EXT1 START FUNC	3	FBA	FBA, 2.12 FBA MAIN CW.
10.4	EXT2 START FUNC	3	FBA	FBA, 2.12 FBA MAIN CW.

REFERENCE CTRL				
34.01	Ext1/Ext2 Sel	0	False	Select External 1.
34.02	Ext1 Mode 1 or 2 Sel	0	False	Model

PROFILE REFERENCE				
65.01	PROF REF SOURCE	2	Fieldbus	Means Fieldbus for 65.02.. 65.05, the others are configured.
65.22	PROF VEL REF SEL	4	FBA REF2	Take fieldbus reference for velocity in PROF VEL MODE.

PROFILE GENERATOR				
66.05	POS ENABLE	0	False	Logical OR or with CW.

POS REF LIMIT				
77.03	POS REF ENA	0	False	Logical OR with reject traversing task bit.
77.06	POS DECEL LIM			Value for deceleration on reject trav-ersing task.

FIELDBUS				
51.01	FBA TYPE	1	Enable	
50.02	COMM LOSS FUNC	1	Fault	Trips the drive.
50.03	COMM LOSS TIMEOUT	X		Any timeout.
50.04	FBA REF1 MODESEL	3	Position	Ref1 and Act1 as Position.
50.05	FBA REF2 MODESEL	4	Velocity	Ref2 and Act2 as Velocity.
50.06, 50.07		x		anyway?
50.8,50.9,50.10 ,50.11				to be defined

FBA Settings				
51.01	FBA TYPE	1		=PROFIBUS DP Displays the type of the fieldbus adapter module.
51.02	NODE ADDRESS			According to PLC configuration.
51.03	BAUDRATE			According to PLC configuration.
51.04	TELEGRAM TYPE	5		(= PPO5) Displays telegram type selected by PLC configuration tool.
51.05	PROFILE	4		(= PROFIdrive Positioning)Con-trol word according to the PRO-FIdrive Positioning mode.
51.27	FPBA PARAM-ETER REFRESH	1		Refreshes the fieldbus parame-ters. Will be set to 0 (done) by drive.

FBA Data In				
52.01	FBA DATA IN1	4		Status word.
52.02	FBA DATA IN2	15		Actual value 1 Position(HW).
				(LW)
52.04	FBA DATA IN4	16		Actual value 2 Velocity(HW)
				(LW)
52.06	FBA DATA IN6	413		POS REF PROF GEN
52.08	FBA DATA IN8	801		ACTIVE FAULT
52.09				

FBA Data Out				
53.01	FBA DATA OUT2	1		Control word.
53.02	FBA DATA OUT2	12		Reference 1 Position (HW).
				(LW)
53.04	FBA DATA OUT4	13		Reference 2 Velocity (HW)
				(LW)
53.06	FBA DATA OUT6	65.06		Acceleration (HW)
53.07	FBA DATA OUT7			Acceleration (LW)
53.08	FBA DATA OUT8	65.07		Deceleration (HW)
53.09	FBA DATA OUT9			Deceleration (LW)
53.10	FBA DATA OUT10	65.09		POS STYLE

POS CORRECTION				
62.02	HOMING STARTFUNC	0	Normal	

Start and stop the drive

The following table shows the additional control bits which could be linked to OFF1 and OFF3 at the function block ACSM1_ACCESS_dc. These additional control bits could be set to "TRUE" to have full control through Fieldbus interface or could be mapped to binary inputs.

Parameter	Name	Value	Mnemonic	Function
START/STOP				
10.10	EM STOP OFF3	1	True Bit pointer	Default, controlled by Fieldbus. The drive is stopped along the emergency stop ramp time, 25.11 EM STOP TIME.
10.11	EM STOP OFF1	1	True Bit pointer	Default, controlled by Fieldbus. Stop along the currently active deceleration ramp.

For example: The drive will execute an OR-Connection from 10.10 and the OFF3-Bit at the ACSM1_ACCESS_dc Block to stop along the emergency ramp. With 10.10 = Bit pointer, the drive might be forced to stop along an emergency ramp without using the PLC to initialize this. This measure allows managing emergency situations without the PLC.

Units for position, velocity, acceleration and deceleration

The PLCopen definition uses as unit for position an abstract unit u, for velocity u/s (u per second) and for acceleration u/s^2 (u per squared second). So it does not matter which unit is selected, but they need to be in certain relations. To get more information, Technical Units (Link setzen). The ACSM1 supports this strategy. Adjustments have to be made in group 60. As a constraint, it has to be considered that although the values are in LREAL-Format at the function block, they will be transferred as 32-Bit Integer so the 32-Bit Integer defines the limits.

Table 179: Parameters for unit adjustment

Parameter	Name	Value	Mnemonic	Comment
POS FEEDBACK				
60.01	POS ACT SEL	0, 1		Depends on the drive configuration.
60.02	POS AXIS MODE	0, 1		Depends on the drive configuration.
60.03	LOAD GEAR MUL			*Chapter 1.5.9.5.2.7 “Gear functions” on page 2732*
60.04	LOAD GEAR DIV			*Chapter 1.5.9.5.2.7 “Gear functions” on page 2732*
60.05	POS UNIT	0, 1, 2, 3	REVOLUTION, DEGREE, METER, INCH	Determines the abstract unit **u**.
60.06	FEED CONST NUM			Scaling for speed.
60.07	FEED CONST DEN			
60.08	POS2INT SCALE		Power of ten	Scaling for position (u/s)
60.09	POS RESOLUTION			Determines the number of bits used for one revolution. Should be at least 16. The Bits remaining of the 32-bit integer is used for the count of whole revolutions.
60.10	POS SPEED UNIT	0	u/s	Speed in units per second or minute or hour.
60.11	POS SPEED2INT		Power of ten	Scaling for speed (rev/s)
60.12	POS SPEED SCALE	1		Scaling for speed, acceleration and deceleration.
60.13	MAXIMUM POS			Depends on the drive configuration.
60.14	MINIMUM POS			Depends on the drive configuration.
60.15	POS THRESHOLD			Depends on the drive configuration.

Some examples:

- The encoder has 1024 Bits per evolution, that are 10 bits (not relevant for PLCopen),
- The load moves 40 mm per revolution,
- The abstract unit **u** should be mm.

The following table shows the adjustment which are relevant for this configuration:

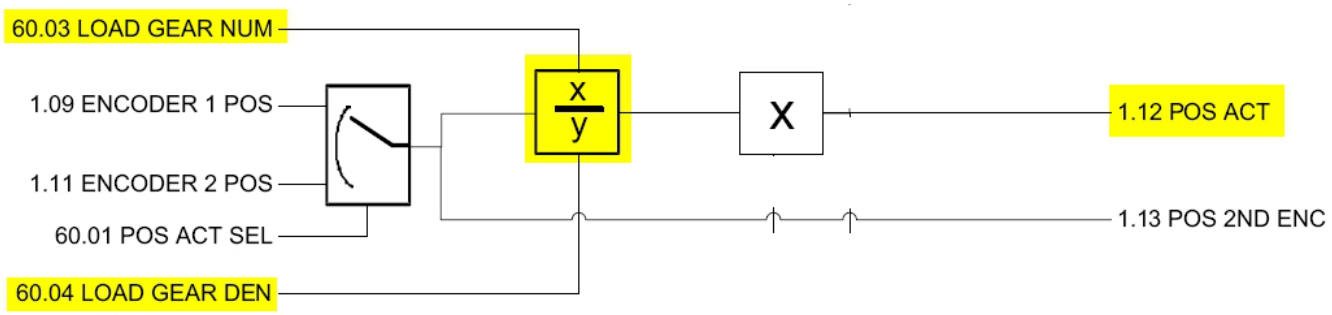
Table 180: Example for unit adjustment

Parameter	Name	Value	Mnemonic	Comment
60.05	POS UNIT	2	METER	Determines the abstract unit **u**.
60.06	FEED CONST NUM	40		Scaling for speed.
60.07	FEED CONST DEN	1000		
60.08	POS2INT SCALE	1000	1000	(Power of Ten). Scaling for position.
60.09	POS RESO-LUTION	16	18	Determines the number of Bits used per revolution. (Should be at least 16). Bits used for number of whole revolutions is (32 -18 =14) No influence on PLCopen.
60.10	POS SPEED UNIT	0	u/s	Units per Seconde (per minute, per hour).
60.11	POS SPEED2INT	1000	1000	(Power of Ten) Scaling for speed.

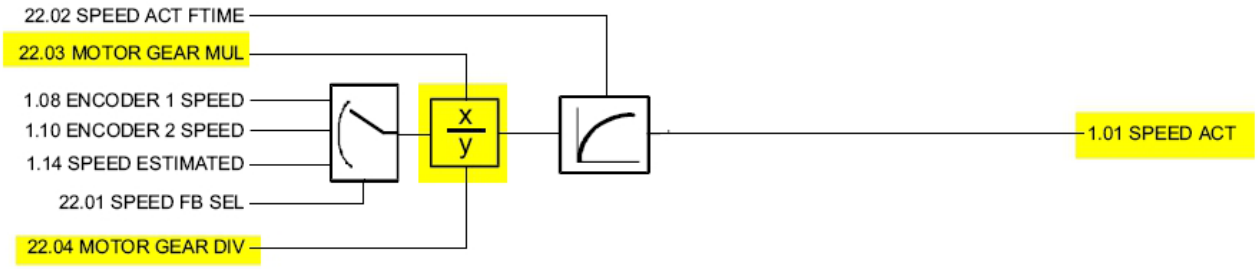
Gear functions

In use with a load gear and encoder signals directly from the motor some adjustment of drive parameters have to be done. The actual position POS ACT (Par 1.12) is calculated as shown, depending on the load gear parameters 60.03 and 60.04.

Example of gear 1:7 , 7 rev on motor side leads to 1 rev. on load side: Par.60.03 = 1, Par. 60.04 = 7.



In use with a load gear and encoder signals from the load side the actual speed is calculated as the load speed. This should be compensated with the motor gear function, Par. 22.03 and 22.04 for the speed control.



Example of gear 1:7 , 7 rev on motor side leads to 1 rev. on load side: Par.22.03 = 1, Par. 22.04 = 7.

As POS ACT, Par 1.12 (set in Par 71.01) is used as position input for speed reference the gear ratio function has to be used to adapt the position for internal speed reference. Par 71.07 = 60.04 x 22.03 and Par. 71.08 = 60.03. x 22.04.

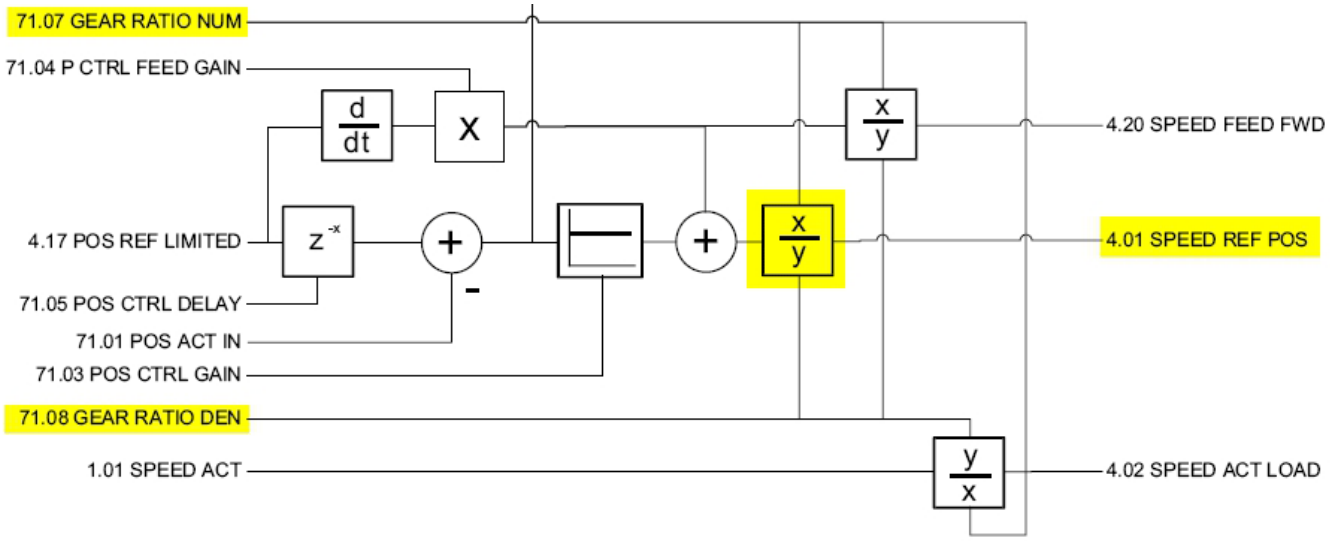


Table 181: Parameters for gear functions

Parameter	Name	Value	Mnemonic	Comment
SPEED FEEDBACK				
22.03	MOTOR GEAR MUL			Numerator for load gear if load encoder is used for speed feedback.
22.04	MOTOR GEAR DIV			Denominator for load gear function if load encoder is used for speed feedback.
POS FEEDBACK				
60.03				Numerator for load gear function if mo-tor encoder is used for pos. feedback.
60.04				Denominator for load gear function if motor encoder is used for pos. feedback.
POSITION CTRL				
71.07	GEAR RATIO MUL	=60.04 x 22.03		Gear ration function to compensate gear calculation for speed and/or position 71.07 =! (60.04 * 22.03).
71.08	GEAR RATIO DIV	=60.03 x 22.04		Scaling ratio for load gear 71.08 =! (60.03*22.04)

Some examples:

- The load gear has 1:7 (7rev of motor =1 rev. of load),
- Encoder on motor side is used for position and speed control.

Just the adjustments which are relevant for this configuration are shown below:

Table 182: Example for gear functions

Param-eter	Name	Value	Mnemonic	Comment
22.03	MOTOR GEAR MUL	1		Motor gear function if encoder on load side is used.
22.04	MOTOR GEAR NOM	1		
60.03	LOAD GEAR NOM	1		Scaling for mech. gear for position 1:7 (7 rev. motor=1 rev. load).
60.04	LOAD GEAR DIV	7		
71.07	GEAR RATIO MUL	7		Scaling for mech. Gear 1:7 (7 rev. motor=1 rev. load).
71.08	GEAR RATIO DIV	1		

MC_VISU_ACSM1_msw, MC_VISU_ACSM1_mcw

These objects show the main status word and main control word of the drive as it is defined for Profidrive Profile.

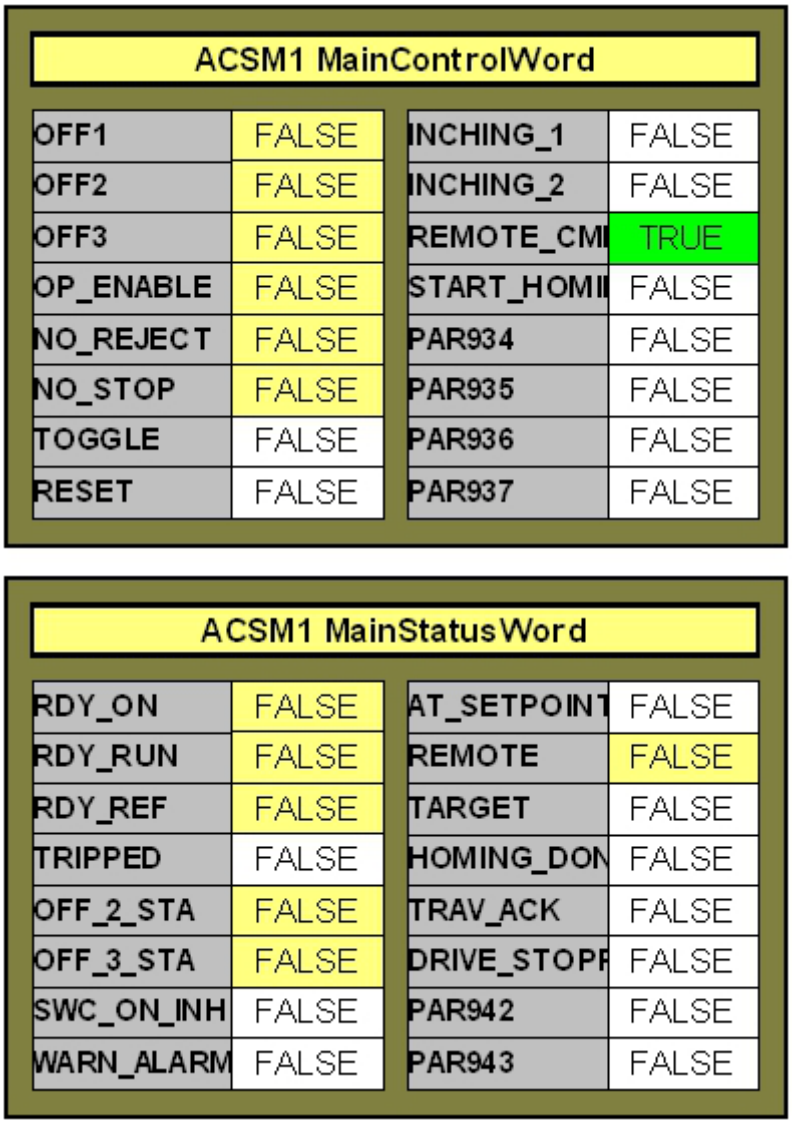


Fig. 284: MC_VISU_ACSM1_mcw, MC_VISU_ACSM1_msw

The colour coding is as follows:

- When the state differs from the expected state (for an active drive) it is shown yellow.
- When the state is equivalent to the expected state (for an active drive) it is shown green.
- An error is shown red.

1.5.9.5.3 Realization with ACS35x on PROFIBUS-DP network

General restrictions

Restrictions for the available function blocks

- As buffered mode, MC_Aborting is realized as a default.
- From the Extended Inputs and Outputs at the function blocks, the following are not realized:
- BufferMode: The realization for ACS35x just supports by default the MC_Aborting mode.
- For all supported PLCopen function blocks the input parameter Jerk will not be used.

- Active
- For the parameter number (WriteParameter, ReadParameter), the following options are available:
  – The PLCopen parameter number
  – The ACS35x parameter number: + 40000
    This manipulation of the parameter number is necessary to distinguish between the different types of parameter number.

Preconditions

The usage of function blocks is like the usage for ACSM1 ↳ *Chapter 1.5.9.5.2.2 “Preconditions” on page 2725*. The corresponding function block ACS350_ACCESS_DC is used to connect AXIS to the drive.

ACS350_ACCESS_dc

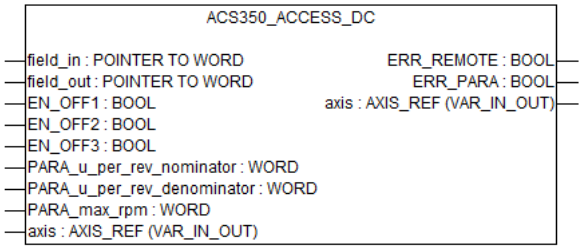


Fig. 285: Function block ACS350_ACCESS_dc

Table 183: General information

Available as of runtime system	V1.2
Included in library	ACS350_MC_support_AC500_V11.lib
Type	Function block without historical values

To make this function block work, the following additional information should be considered:

↳ *Chapter 1.5.9.5.3.5 “Adjustment of parameters for drive” on page 2739*

↳ *Chapter 1.5.9.5.3.6 “MC_VISU_ACS350_mcw, MC_VISU_ACS350_msw” on page 2740*

The function block ACS350_ACCESS_dc is used as driver unit for PLCopen function blocks with ACS35x in decentralized motion control.

During normal operation, the drive will be completely controlled by the fieldbus interface. For safety reasons, this could be combined with additional measures through binary inputs on the drive. So the drive would be in defined state when the fieldbus is not yet initialized or in case of communication problems. For coordination, the inputs EN_OFF1, EN_OFF2 and EN_OFF3 are available at the driver unit function block ACS350_ACCESS_dc. When the fieldbus interface is ok (shown by ERR_REMOTE=FALSE) EN_OFF1, EN_OFF2 and EN_OFF3 should be switched to "true" before the MC_Power function block is enabled. When an emergency stop should be executed, the sequence depends on the required Stopping mode. (Details could be found in the drive manual.) The priority of the signals is as follows:

EN_OFF2 >EN_OFF3> EN_OFF1

Do the following to power off:

Set EN_OFF1 or EN_OFF2 or EN_OFF3 or a combination to "false", depending on required stop mode. This will already reset the internal power-on state machine to the respective state and change the signals in the fieldbus CW. In addition, disable the MC_POWER function block. A sequence for these 2 actions is not required, but the MC_POWER will show an error message when it is not disabled while the drive is already stopped.

Input description

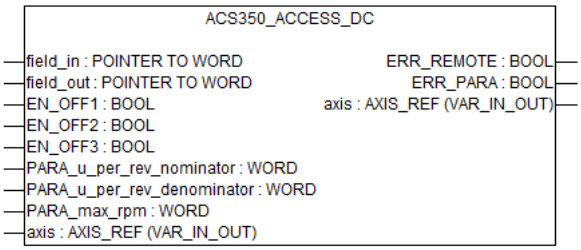


Fig. 286: Function block ACS350_ACCESS_dc

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

field_in Data type: POINTER TO WORD

At this input, the address of the input data from drive to PLC should be connected.

field_out Data type: POINTER TO WORD

At this input, the address of the output data from PLC to drive should be connected.

EN_OFF1 Data type: BOOL

Stop along the currently active deceleration ramp.

EN_OFF2 Data type: BOOL

Power off the motor.

EN_OFF3 Data type: BOOL

The drive is stopped along the emergency stop ramp time.

PARA_u_per_rev_nominator Data type: WORD

Units per revolution.

PARA_u_per_rev_denominator Data type: WORD

Units per revolution.

PARA_max_rpm Data type:WORD

Maximum value for rotations per minute.

Axis Data type: AXIS_REF

This data structure is needed to connect this driver unit to the PLCopen function blocks.

Output description

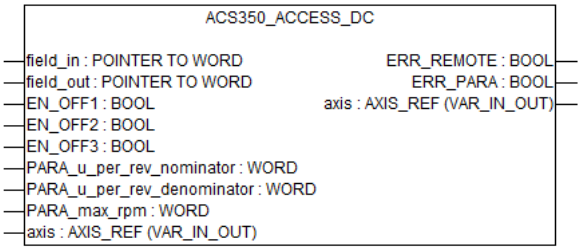


Fig. 287: Function block ACS350_ACCESS_dc

ERR_REMOTE Data type: BOOL

This output indicates that no fieldbus connection to the drive is available. When this is shown, the PLC is not able to control the drive via fieldbus. This has to be considered in the PLC application to switch off the drive by other measures.

ERR_PARA Data type: BOOL

This output indicates that a parameter access was not successful. The reason could be a wrong parameter number at function block or the drive configuration.

Units for position, velocity, acceleration and deceleration

The PLCopen definition uses as unit for position an abstract unit [u], for velocity [u/s](u per second) and for acceleration [u/s^2](u per squared second). So it does not matter which unit is selected, but they need to be in certain relations. The ACS35x follows a different this strategy. To establish the conversion of different units, the parameters:

- PARA_MAX_RPM
- PARA_U_PER_REV_NOMINATOROR
- PARA_U_PER_REV_DENOMINATOR
  Have to be adjusted at the function block ACS350_ACCESS_DC.

Some example:
- The motor moves at maximum with 1500 rpm.
- It is planned to use revolutions as unit, so the velocity will be in revolutions/second and the acceleration and deceleration will be in revolutions/s^2
  - PARA_MAX_RPM=1500
  - PARA_U_PER_REV_NOMINATOR=1
  - PARA_U_PER_REV_DENOMINATOR=1
- It is planned to use degrees as unit, so the velocity will be in degrees/second and the acceleration and deceleration will be in degrees/s^2
  - PARA_MAX_RPM=1500
  - PARA_U_PER_REV_NOMINATOR=360
  - PARA_U_PER_REV_DENOMINATOR=1

Adjustment of parameters for drive

The drive has to be adjusted to use the ABBdrive profile. To do this, and to establish the PROFIBUS connection to the PLC you have to follow the according documentation. In addition, some parameters have to be adjusted. These are listed in the following tables. For further commissioning, follow the chapter "Start-Up" in the drives manual.

Table 184: Parameter adjustment for ACS35x

START/STOP				
Parameter	**Name**	**Value**	**Mnemonic**	**Comment**
10.1	EXT1 START FUNC	10	KOMM	Fieldbus Interface
11.2	EXT1/EXT2	0	EXT1	FBA, 2.12 FBA MAIN CW
11.3	AUSW.EXT SOLLW1	8	KOMM	
16.1	FREIGABE	8	KOMM	
16.4	FEHL QUIT	8	KOMM	
22.1	BE/VERZ 1/2 AUSW	0	False	Select 22.02..22.04

FIELDBUS				
FBA Settings				
Parameter	**Name**	**Value**	**Mnemonic**	
51.01	FBA TYPE	1		=PROFIBUS DP Displays the type of the fieldbus adapter module.
51.02	NODE ADDRESS			According to PLC configuration.
51.03	BAUDRATE			According to PLC configuration.
51.04	TELEGRAM TYPE	2		(= PPO2) Displays telegram type selected by PLC configuration tool.
51.05	PROFILE	1		(= ABB drive)

FIELDBUS				
FBA Data In				
Parameter	**Name**	**Value**	**Mnemonic**	
54.01	FBA DATA IN1	4		Status word
54.02	FBA DATA IN2	5		Actual value 1
54.03	FBA DATA IN3	401		Actual fault
54.04	FBA DATA IN4			
54.05	FBA DATA IN5			
54.06	FBA DATA IN6			

FIELDBUS				
FBA Data Out				
Parameter	**Name**	**Value**	**Mnemonic**	
55.01	FBA DATA OUT2	1		Control word
55.02	FBA DATA OUT3	2		Reference 1
55.03	FBA DATA OUT4	2202		Acceleration
55.04	FBA DATA OUT5	2203		Deceleration
55.05	FBA DATA OUT6			
55.06				

FIELDBUS				
Options				
Parameter	**Name**	**Value**	**Mnemonic**	**Comment**
98.02	Comm Prot Sel	2	EXT FBA	Select of external fieldbus adapter

MC_VISU_ACS350_mcw, MC_VISU_ACS350_msw

These objects show the main status word and main control word of the drive as it is defined for Profidrive Profile.

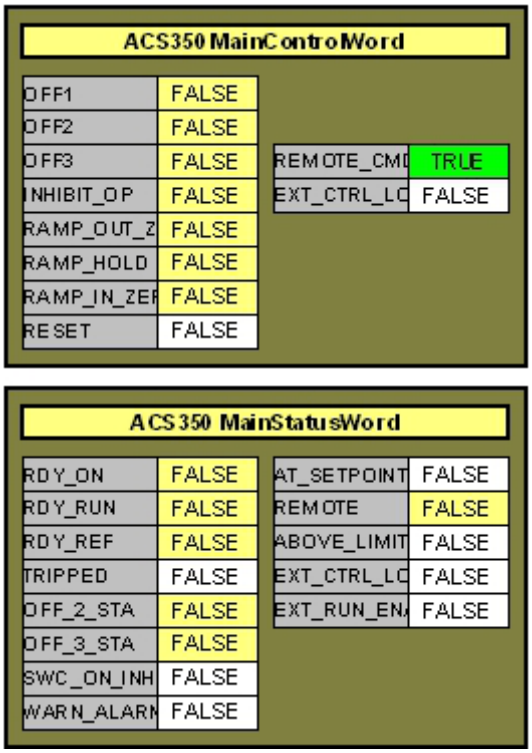


Fig. 288: MC_VISU_ACS350_mcw, MC_VISU_ACS350_msw

The colour coding is as follows:

- When the state differs from the expected state (for an active drive) it is shown yellow.
- When the state is equivalent to the expected state (for an active drive) it is shown green.
- An error is shown red.

1.5.9.5.4 Realization with FM562

Special function blocks are available to manage and control the function of the FM562 module. These function blocks are contained in PTO Library PTO_FM562_MC_support_V22.lib and PLCopen Libraries MC_Base_AC500_V11.lib and MC_Blocks_AC500_V11.lib.

PLCopen Library contains modular function blocks based on PLCopen motion control standard and can be used for various motion devices e.g: ACSM1, ACS35x, ACS800, E100, E150 and FM562.

PTO Library provides function blocks that adapts specific PLCopen function blocks to FM562 PTO module and also adds other specialized function for FM562.

General restrictions

Restrictions for the available function blocks

- Buffered mode is not implemented
- The Jerk is defined as a fixed value. When it is to be used, set “0” as “Jerk off” and “1” as Jerk on.
- State cannot be changed between Discrete Motion and Continuous Motion directly, state error will be generated.
- By executing function block MC_Reset, the state will transfer from ErrorStop to StandStill as well as resetting the current axis position to 0.

PTO_FM562_ACCESS adapts specific PLCopen blocks to FM562

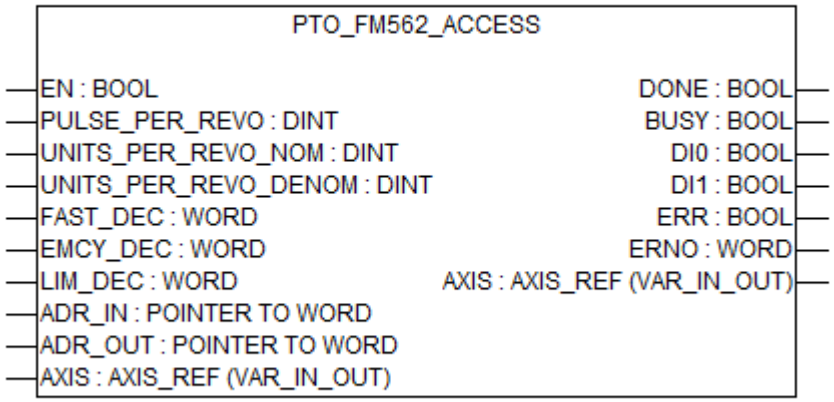


Table 185: General information

Available as of runtime system	V2.2
Included in library	PTO_FM562_MC_support_V22.lib
Type	Function block without historical values

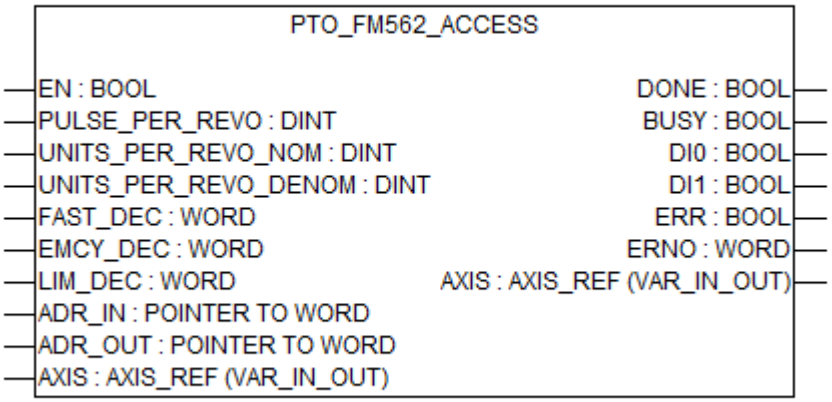
Description

The function block PTO_FM562_ACCESS is used to adapt specific PLCopen blocks to FM562 module and also adds other specialized function for FM562.

PLCopen function blocks must be used with function block PTO_FM562_ACCESS for realizing positioning and speed control with FM562. PTO_FM562_ACCESS supported PLCopen function blocks are:

- MC_MoveAbsolute *Chapter 1.5.9.6.1.1 “MC_MoveAbsolute” on page 2747*
- MC_MoveRelative *Chapter 1.5.9.6.1.2 “MC_MoveRelative” on page 2751*
- MC_MoveVelocity *Chapter 1.5.9.6.1.6 “MC_MoveVelocity” on page 2767*
- MC_Power *Chapter 1.5.9.6.3.2 “MC_Power” on page 2835*
- MC_Reset *Chapter 1.5.9.6.3.5 “MC_Reset” on page 2842*
- MC_Stop *Chapter 1.5.9.6.1.9 “MC_Stop” on page 2781*
- MC_SetPosition *Chapter 1.5.9.6.3.13 “MC_SetPosition” on page 2858*
- MC_ReadStatus *Chapter 1.5.9.6.3.3 “MC_ReadStatus” on page 2837*
- MC_ReadActualPosition *Chapter 1.5.9.6.3.10 “MC_ReadActualPosition” on page 2852*
- MCA_MoveVelocityContinuous *Chapter 1.5.9.6.5.4 “MCA_MoveVelocityContinuous” on page 2882*

Input description



EN (enable)

Data type: BOOL

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The function block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility.

PULSE_PER_REVO (pulse per revolution), UNITS_PER_REVO_NOM DINT (units per revolution nominator), UNITS_PER_REVO_DENOM DINT (units per revolution denominator)

Data type: DINT, range: 0001 to FFFF hex (1 to 2147483647)

These three parameters shall be used together for scaling, which are only be changed when PLCopen state machine is in “Disable” or when MC_Power is OFF.

Pulses per revolution depend on specific stepper drive/motor, how many pulses need to be created to turn 1 revolution.

When 1 revolution move e.g 40 mm, 1 mm is used as unit: UNITS_PER_REV_NOM=40 and UNITS_PER_REV_DENOM=1

When using additional gearbox 1:3: UNITS_PER_REV_NOM=40 and UNITS_PER_REV_DENOM=3

When using angle as unit: U_PER_REV_NOMINATOR=360, UNITS_PER_REV_DENOM=1

When working with revolutions 1 u = 1: UNITS_PER_REV_NOM=1, UNITS_PER_REV_DENOM= 1

FAST_DEC (fast deceleration)

Data type: WORD, range: 0001 to FFFF hex (1 to 65,535 u/s^2)

Fast deceleration for stopping CPU, set before the operation starts. The fast deceleration ramp is "FAST_DEC" multiples "increments per unit" (scaling).

EMCY_DEC (emergency deceleration)

Data type: WORD, range: 0001 to FFFF hex (1 to 65,535 u/s^2)

Emergency deceleration: for configurable digital input 1 "DI1" is configured as "Stop". When "emergency stop" DI1 is triggered (DI1=TRUE), PTO will decelerate with "EMCY_DEC" deceleration. The emergency deceleration ramp is "EMCY_DEC" multiples "increments per unit" (scaling).

LIM_DEC (limit deceleration)

Data type: WORD, range: 0001 to FFFF hex (1 to 65,535 u/s^2)

Limit deceleration: for configurable digital input 0 "DI0" is configured as "Enable (external hardware enable)". During operation digital input DI0=TRUE which shows external 24 V digital input. When "right/left limit stop" is triggered which means the external 24 V digital input is disabled (DI0=FALSE), PTO will decelerate with "LIM_DEC" deceleration rate. The limit deceleration ramp is "LIM_DEC" multiples "increments per unit" (scaling).

ADR_IN (address of process image input)

Data type: POINTER TO WORD

Pointer to structure of FM562 inputs. At this input, the address of the first input data from the structure of FM562 input should be connected. The use of an ADR operator is needed.

ADR_OUT (address of process image output)

Data type: POINTER TO WORD

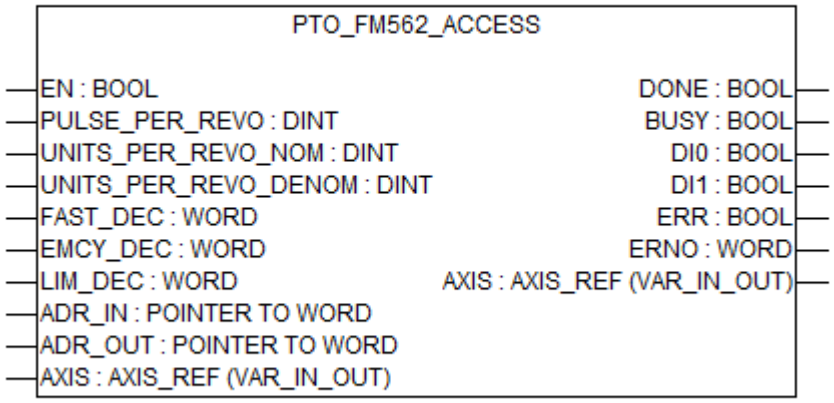
Pointer to structure of FM562 outputs. At this input, the address of the first output data from the structure of FM562 output should be connected. The use of an ADR operator is needed. The operand address of the first input and output data of each axis of FM562 must be mapped to input ADR_IN and ADR_OUT. Define and create configuration data will be emerged automatically as global variable after creating configuration data.

Axis

Data type: AXIS_REF

Reference to the axis.

Output description



DONE (done)

Data type: BOOL

The output DONE signals the completion of the process triggered with the EN input. After finishing of the process (pulse output finished or pulse output stopped), DONE is TRUE.

BUSY (busy)

Data type: BOOL

Output BUSY shows pulse output status, after pulse output start and before it finished/stopped/error occurred, the BUSY output will continuously be TRUE.

DI0 (digital input 0 or digital input 2)

Data type: BOOL

Output DI0 indicates the status of digital input0 or input2. The digital input0 or input2 can be configured by FM562 configuration in CBP. If input0 or input2 is configured as "Axis enable/ Limit switch" (external hardware enable), then DI0=TRUE or DI2=TRUE indicates that input0 or input2 is connected with 24 V input; DI0=FALSE or DI2=FALSE indicates input0 or input2 is not connected with 24 V, so PTO will stop output. If input0 or input2 is configured as "No function", PTO will be in operation without reading status of input0 or input2.

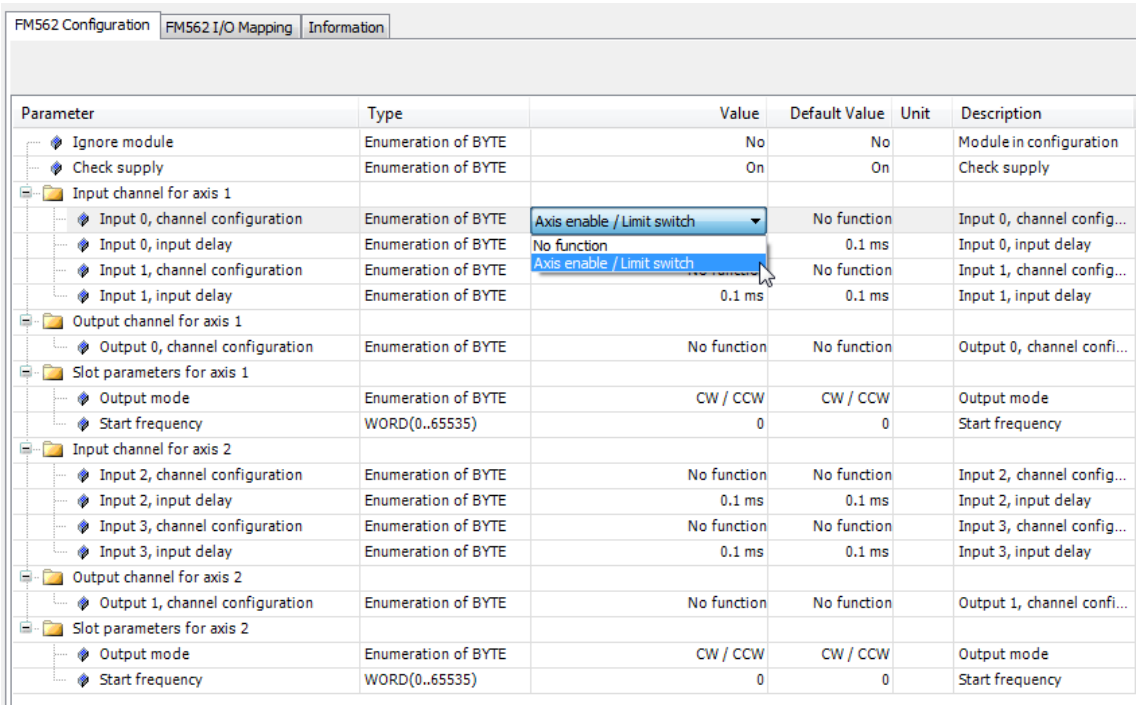


Fig. 289: Output DI0

DI1 (digital input 1 or digital input 3)

Data type: BOOL

Output DI1 indicates the status of digital input1 or digital input3. The digital input1 or input3 can be configured by FM562 configuration in CBP. If DI1 is configured as “Stop”, then if DI1=TRUE, PTO output is emergently stopped. If DI1 is configured as “No function”, PTO will be in operation without reading states of DI1. If DI1 is configured as “Registration”, current pulse number is recorded and can be checked on FM562 IO mapping when DI1=TRUE.

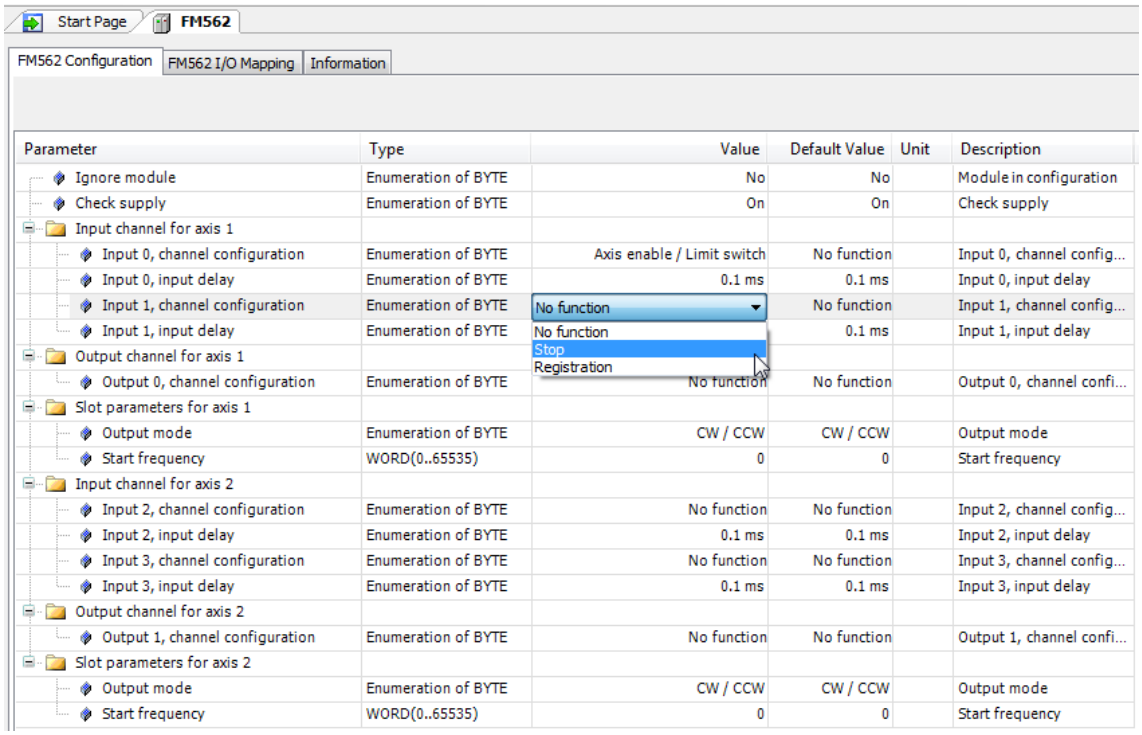


Fig. 290: Output DI1

ERR (error)

Data type: BOOL

Output ERR indicates whether an error occurred during data reception. If ERR is TRUE, an error occurred, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD

ERNO always has to be considered together with the output ERR. The value output at ERNO is only valid if ERR is TRUE.

Quick start documentation for FM562

A quick start documentation for FM562 can be found in the folder C:\Users\Public\Documents\AutomationBuilder\Examples.

1.5.9.6 PLCopen function blocks (Single and multi axis)

1.5.9.6.1 Single-Axis function blocks

MC_MoveAbsolute

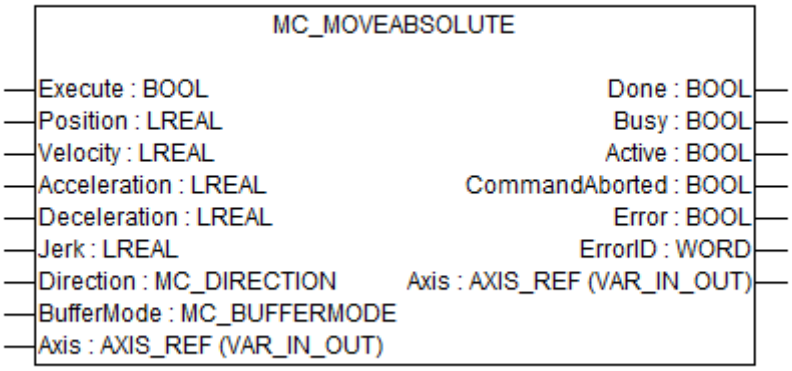


Fig. 291: Function block MC_MoveAbsolute

Table 186: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

This function block commands a controlled motion to a specified absolute position.

- *This action completes with velocity zero if no further action are pending,*
- *If there is only one mathematical solution to reach the commanded position (like in linear systems), the value of the input direction is ignored,*
- *For modulo axis - valid absolute position values are in the range of [0, [360, (360 is excluded), or corresponding range. The application, however, may shift the commanded position of MC_MoveAbsolute into the corresponding modulo range. For relative positions, modulo 360 is applicable.*
- *The Enum type "shortest_way" is focused to a trajectory which will go through the shortest route. The decision which direction to go is based on the current position where the command is issued.*

Example

The following figure shows two examples of the combination of two function blocks MC_MoveAbsolute.

The left part of timing diagram illustrates the case if the second function block is called after the first one. If first reaches the commanded position of 6000 (and the velocity is 0) then the output Done causes the second Functino Block to move to the position 10000.

The right part of the timing diagram illustrates the case if the second move function block starts the execution while the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first function block. The second function block moves directly to the position 10000 although the position of 6000 is not yet reached.

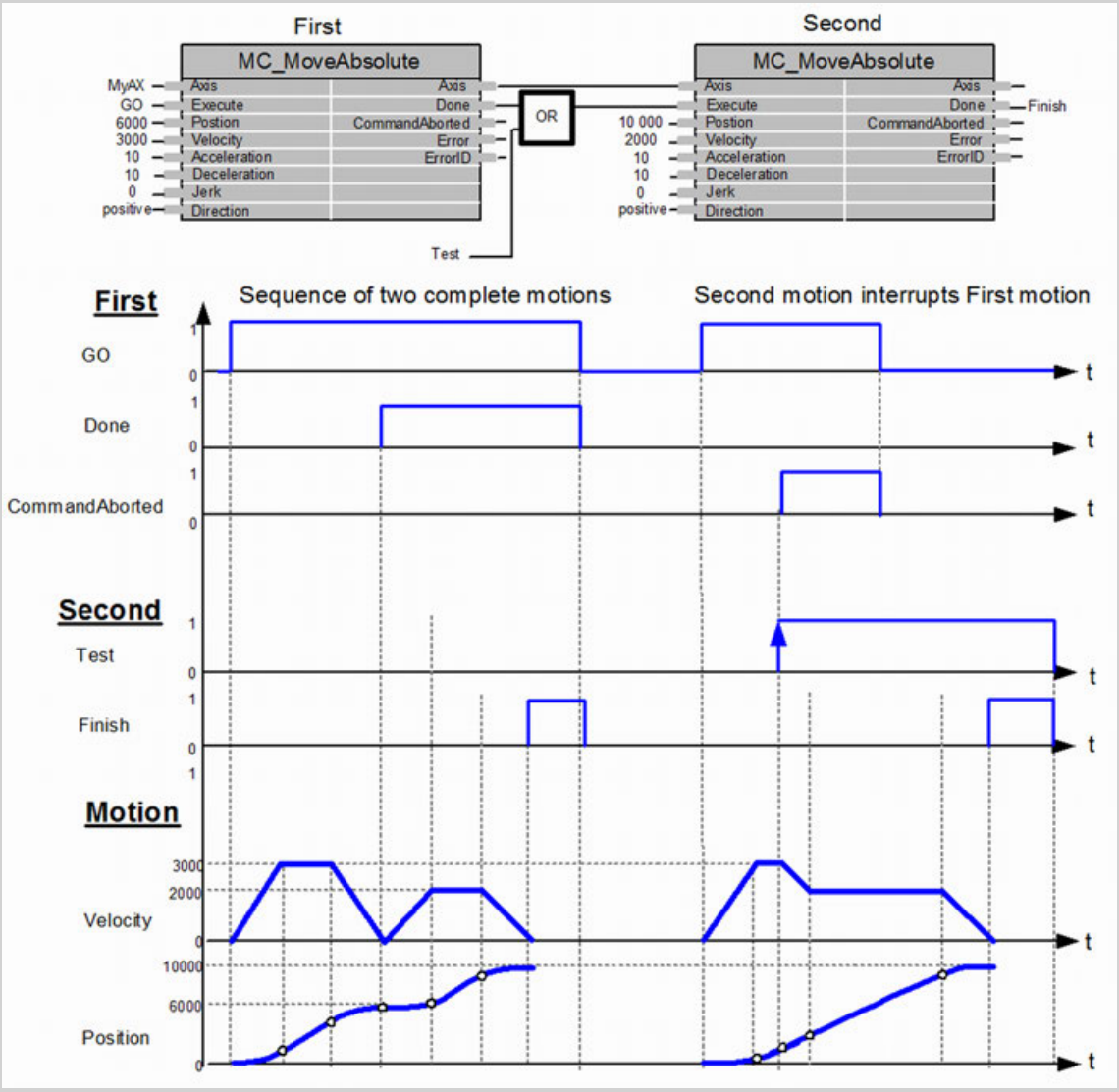


Fig. 292: These examples are based on two instances of the function block: Instance ”First“ and ”Second“.

Using this function block with FM562

Input Jerk: 1 = jerk on, 2 = jerk off

Input Direction: 0 = DEFAULT, 1 = POSITIVE, 3 = NEGATIVE

Input BufferMode: Not implemented

Input description

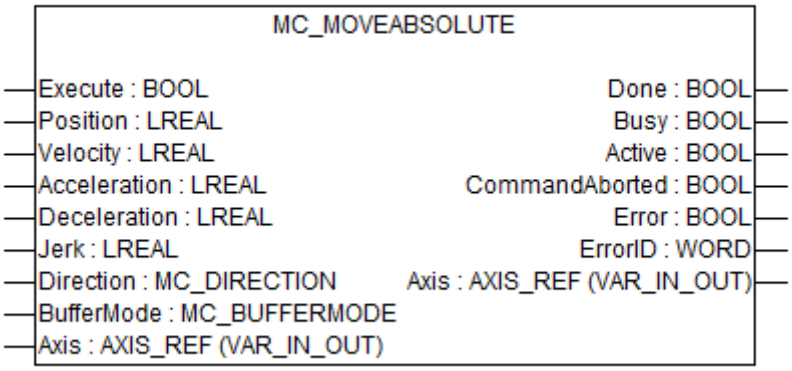


Fig. 293: Function block MC_MoveAbsolute

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Position Data type: LREAL, unit: u
Reference position.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

Direction Data type: MC_Direction, range: DEFAULT, POSITIVE, SHORTEST, NEGATIVE, CURRENT
Enum type.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

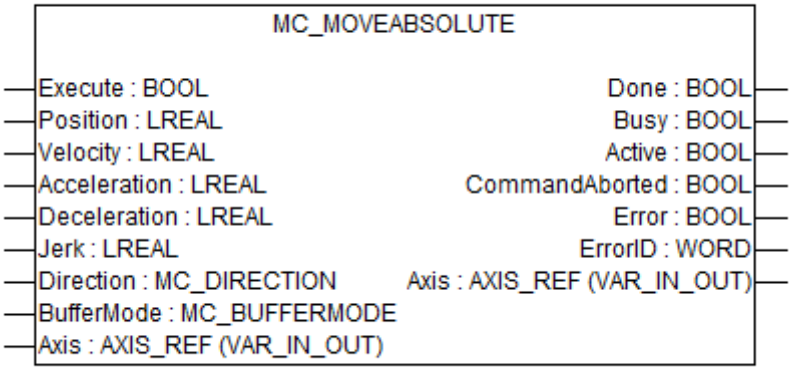


Fig. 294: Function block MC_MoveAbsolute

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveRelative

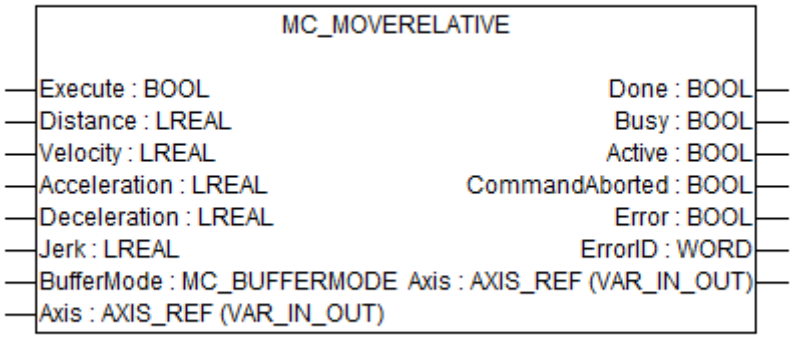


Fig. 295: Function block MC_MoveRelative

Table 187: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

This function block commands a controlled motion of a specified distance relative to the actual position at the time of the execution.

This action completes with velocity zero if no further action are pending.

The following figure shows the example of the combination of two MC_MoveRelative function blocks:	1.) The left part of timing diagram illustrates the case if the second function block is called **after** the first one. If the first one reaches the commanded distance 6000 (and the velocity is 0) then the output Done causes the second function block to move to the distance 10000. 2.) The right part of the timing diagram illustrates the case if the second move function blocks starts the execution **while** the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first function block. The second function block **adds on the actual position** of 3250 the distance 4000 and moves the axis to the resulting position of 7250.

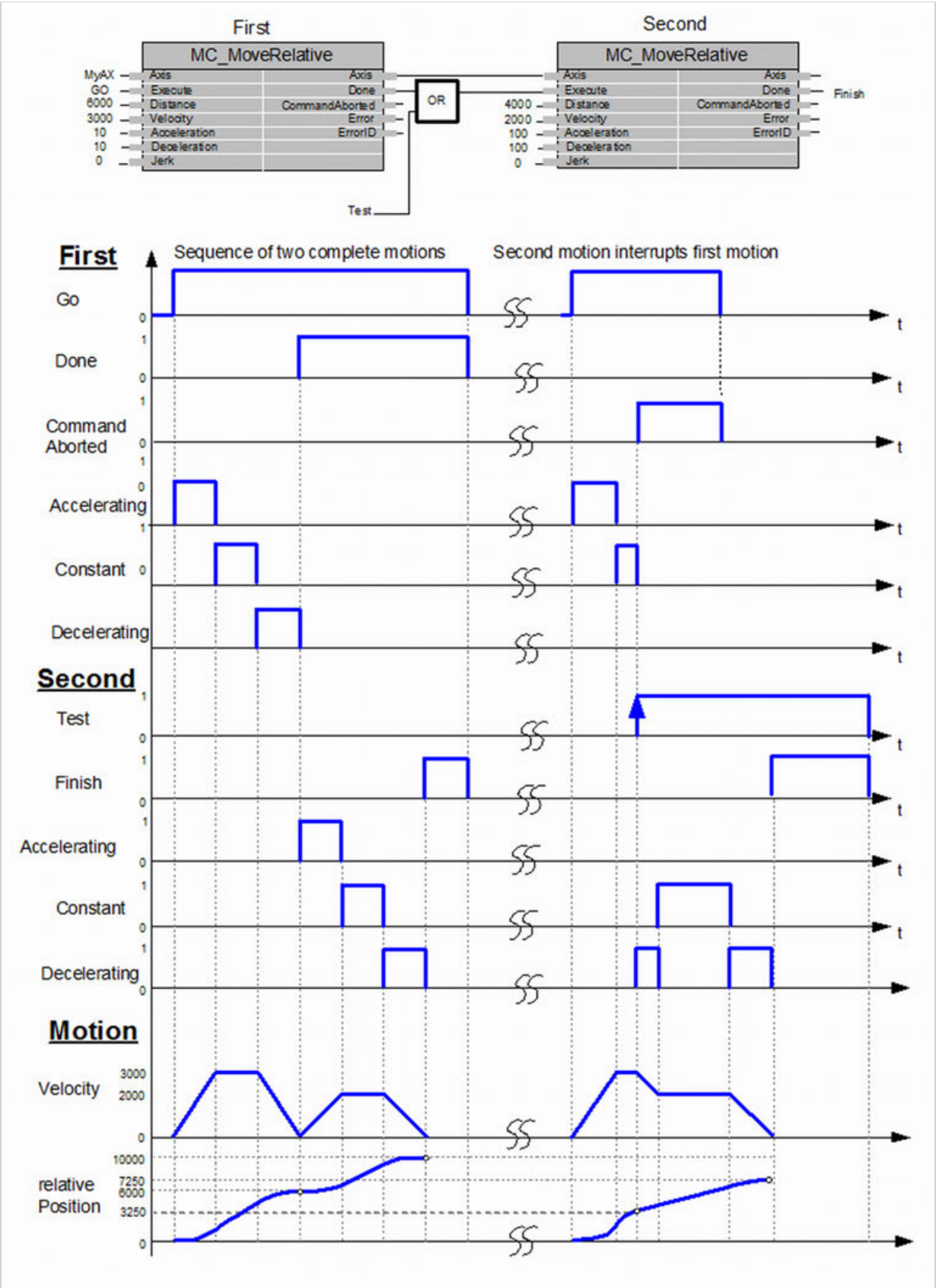


Fig. 296: Timing diagram for MC_MoveRelative

Using this function block with FM562

Input Jerk: 1 = jerk on, 2 = jerk off

Input BufferMode: Not implemented

Input description

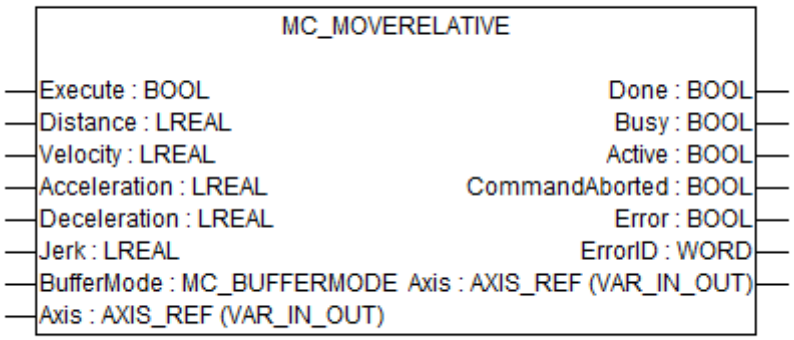


Fig. 297: Function block MC_MoveRelative

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

Distance

Data type: LREAL, unit: u

Relative distance for the motion.

Velocity

Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk

Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

BufferMode

Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

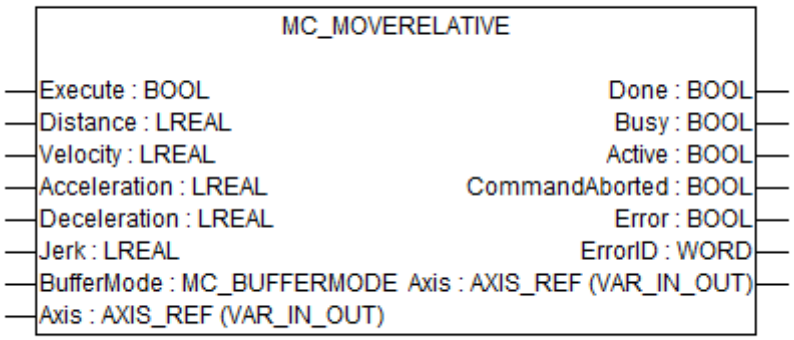


Fig. 298: Function block MC_MoveRelative

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandA-borted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveAdditive

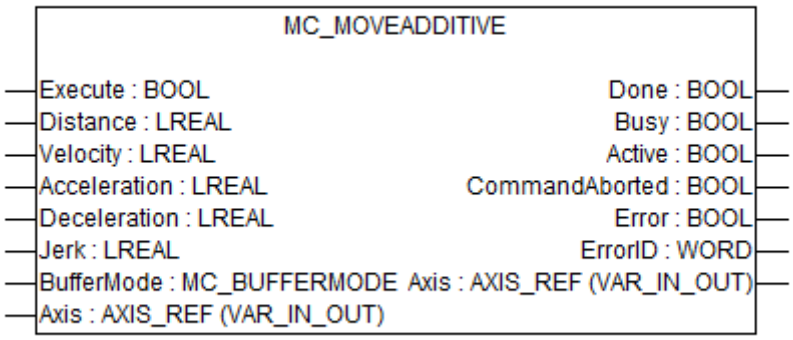


Fig. 299: Function block MC_MoveAdditive

This function block commands a controlled motion of a specified relative distance additional to the most recent commanded position in the state Discrete Motion. The most recent commanded position may be the result of a previous MC_MoveAdditive motion which was aborted. If the function block is activated in the Continuous Mode the specified relative distance is added to the actual position at the time of the execution.

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Examples of the combination of two function blocks while the axis is in state discrete motion

1.) The left part of timing diagram illustrates the case if the second function block is called **after** the first one. If the first one reaches the commanded distance 6000 (and the velocity is 0) then the output "Done" causes the second function block to move to the distance 10000.

2.) The right part of the timing diagram illustrates the case if the second move function blocks starts the execution **while** the first function block is still executing. In this case the first motion is interrupted and aborted by the test signal during the constant velocity of the first function block. The second function block **adds on the previous commanded position** of 6000 the distance 4000 and moves the axis to the resulting position of 10000.

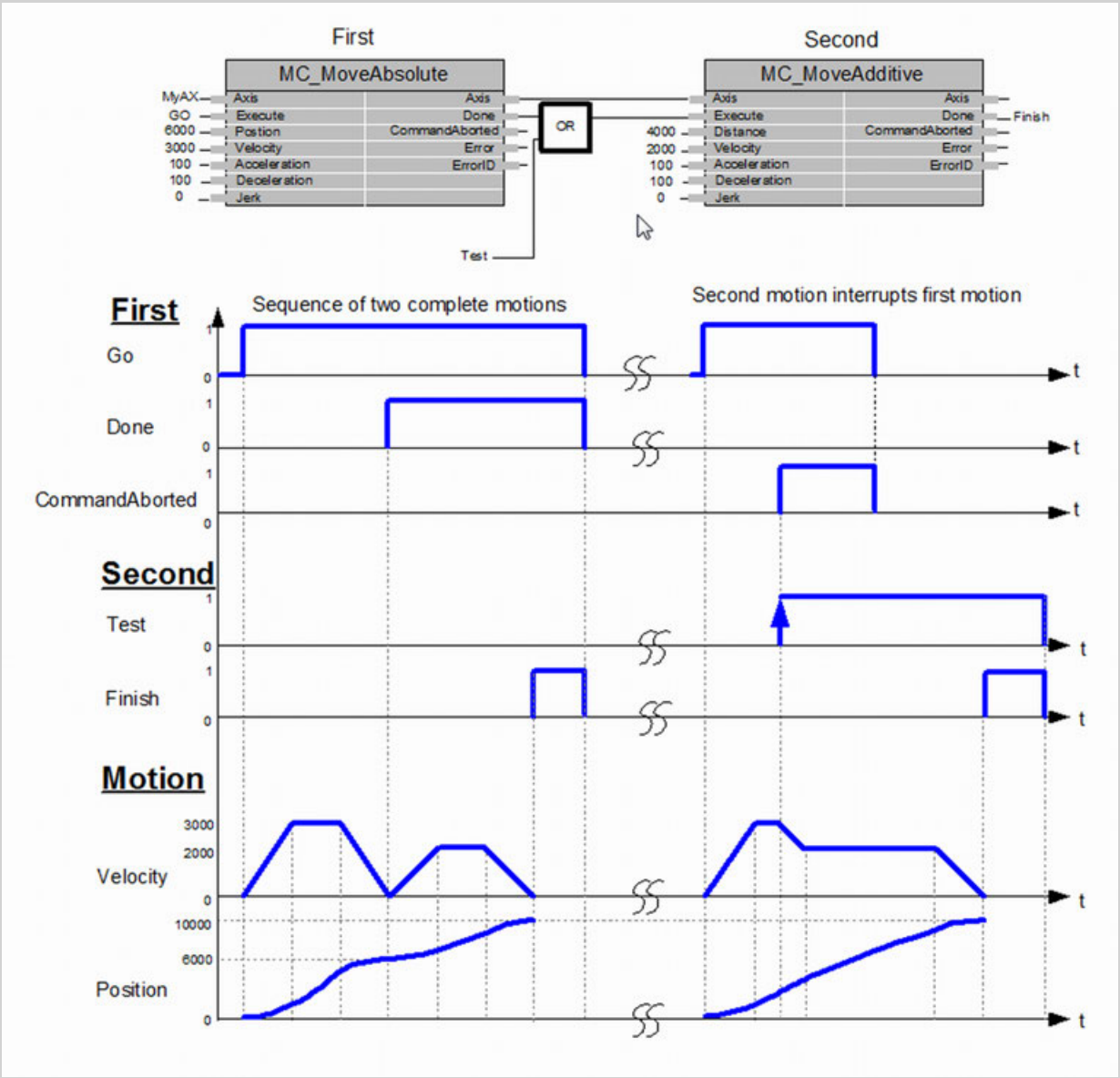


Fig. 300: Timing diagram for MC_MoveAdditive

Input description

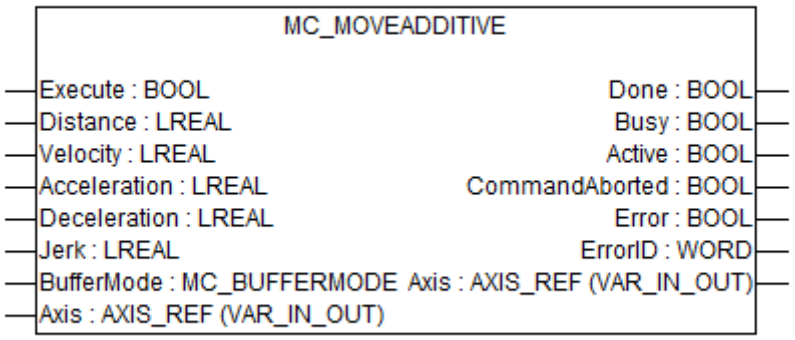


Fig. 301: Function block MC_MoveAdditive

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Distance Data type: LREAL, unit: u
Relative distance for the motion.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

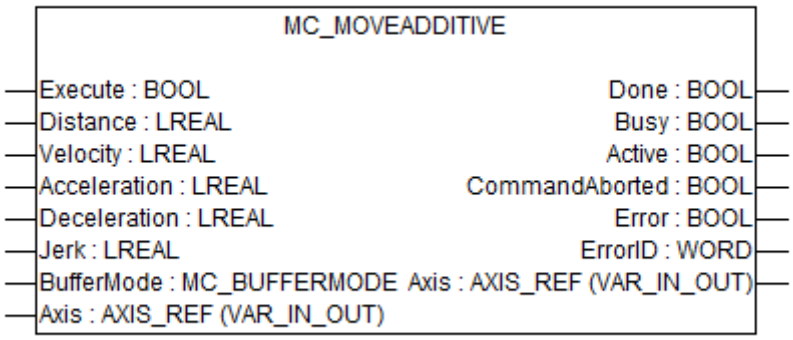


Fig. 302: Function block MC_MoveAdditive

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveSuperImposed

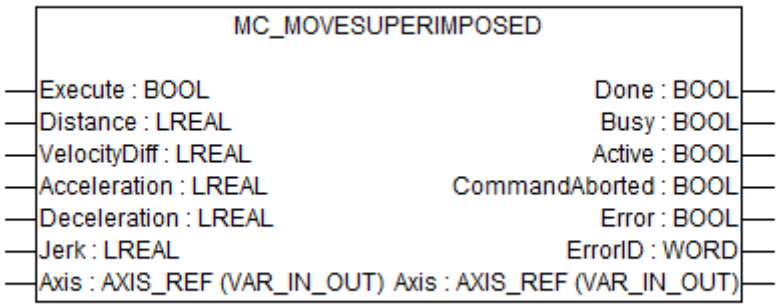


Fig. 303: Function block MC_MoveSuperimposed

This function block commands a controlled motion of a specified relative distance additional to an existing motion. The existing motion is not interrupted, but is superimposed by the additional motion.

- *If MC_MoveSuperImposed is active, then any other command in aborting mode except MC_MoveSuperImposed will abort both motion commands: Both the MC_MoveSuperImposed and the underlying motion command. In any other mode, the underlying motion command is not aborted*
- *If MC_MoveSuperImposed is active and another MC_MoveSuperImposed is commanded, only the on-going MC_MoveSuperImposed command is aborted, and replaced by the new MC_MoveSuperImposed, but not the underlying motion command*
- *The function block MC_MoveSuperimposed causes a change of the velocity and, if applicable, the commanded position of an on-going motion in all relevant states*
- *In the state StandStill the function block MC_MoveSuperimposed acts like MC_MoveRelative*
- *The values of Acceleration, Deceleration, and Jerk are additional values to the on-going motion, and not absolute ones. With this, the underlying function block always finishes its job in the same period of time regardless of whether a MC_MoveSuperimposed function block takes place concurrently*
- *MC_MoveSuperimposed acts on the slave axis, while MC_Phasing acts on the master side, as seen from the slave*
- *•The output "Active" has a different behavior as in buffered function blocks*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

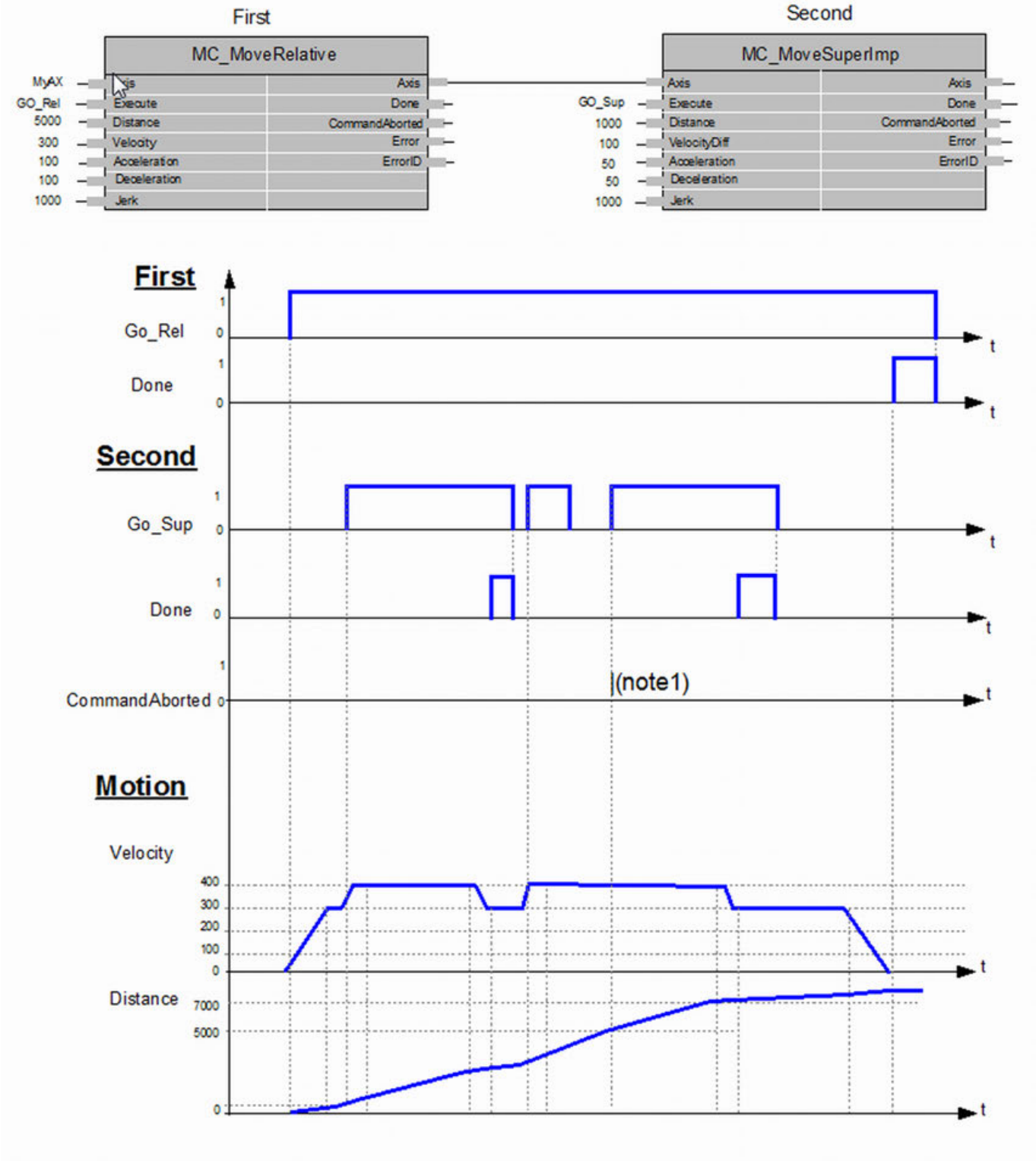


Fig. 304: Timing diagram for MC_MoveSuperimposed

- *The CommandAborted is not visible here, because the new command works on the same instance. ↬ Chapter 1.5.9.3.9.1 “General rules” on page 2601*
- *The end position is between 7000 and 8000, depending on the timing of the aborting of the second command set for the MC_MoveSuperimposed.*

Example of MC_MoveSuperImposed during camming:

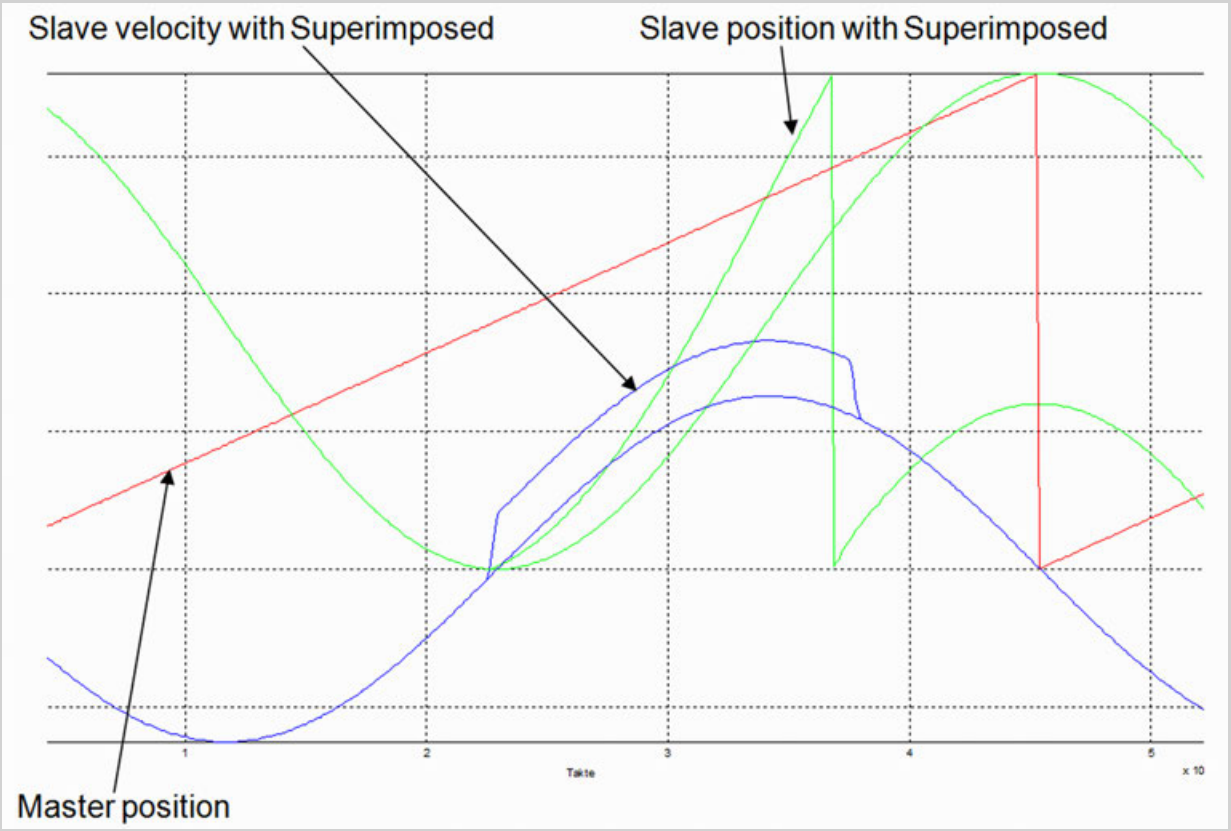


Fig. 305: Timing diagram of effect of MC_MoveSuperImposed on same axis

At Slave velocity, the double line shows the effect of MoveSuperimposed while in Synchronized Motion during Camming. The same is valid for the related slave position.

Input description

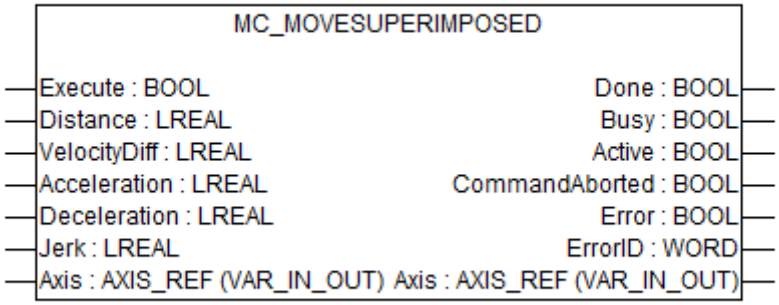


Fig. 306: Function block MC_MoveSuperimposed

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Distance Data type: LREAL, unit: u
Relative distance for the motion.

VelocityDiff Data type: LREAL, unit: u/s
Value of the maximum velocity difference to the ongoing motion (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

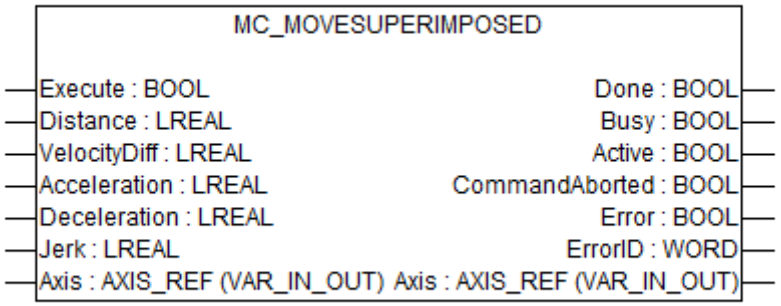


Fig. 307: Function block MC_MoveSuperimposed

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_HaltSuperimposed

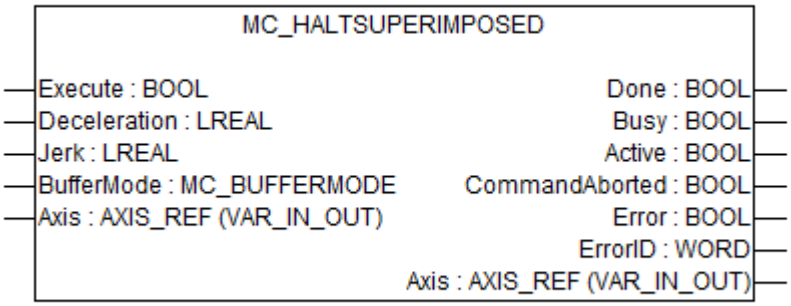


Fig. 308: Function block MC_HaltSuperimposed

This function block commands a halt to all superimposed motions of the axis. The underlying motion is not interrupted.

Input description

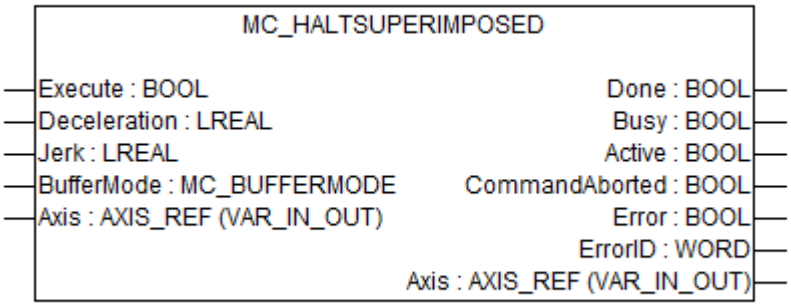


Fig. 309: Function block MC_HaltSuperimposed

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis
Data type: AXIS_REF
Reference to the axis.

Output description

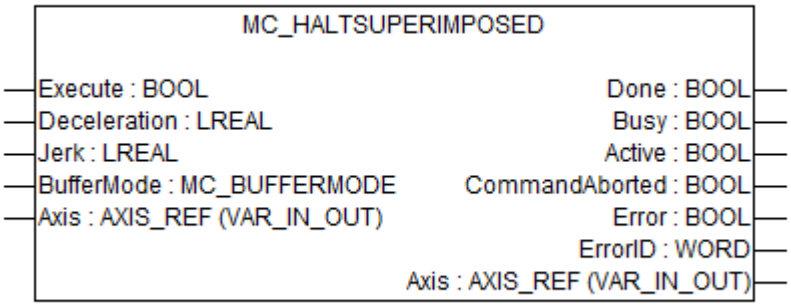


Fig. 310: Function block MC_HaltSuperimposed

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished and new output values are to be expected.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveVelocity

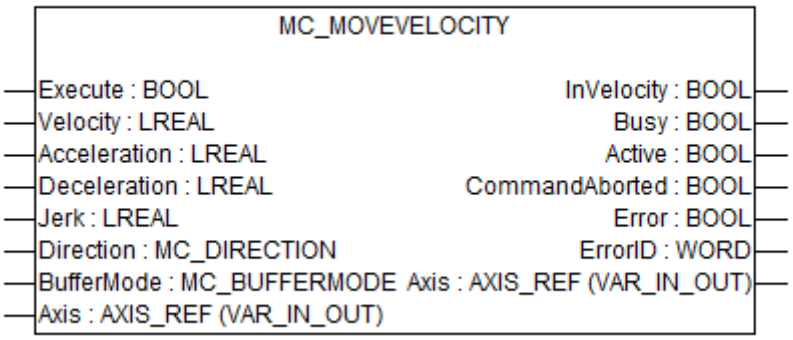


Fig. 311: Function block MC_MoveVelocity

Table 188: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block commands a never ending controlled motion at a specified velocity.

- *To stop the motion, the function block has to be interrupted by another function block issuing a new command.*
- *The signal "InVelocity" has to be reset when the block is aborted by another block or at the falling edge of "Execute".*
- *In combination with MC_MoveSuperimposed, the output “"nVelocity" stays TRUE once the velocity setpoint of the axis has reached the commanded velocity.*

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Examples of the combination of two function blocks MC_MoveVelocity

The left part of timing diagram illustrates the case if the second function block is called after the first one is completed. If first reaches the commanded velocity 3000 then the output First.InVelocity AND the signal Next causes the second function block to move to the velocity 2000.

The right part of the timing diagram illustrates the case if the second function block starts the execution while the first function block is not yet InVelocity. The following sequence is shown: The first motion is started again by Go at the input First.Execute. While the first function block is still accelerating to reach the velocity 3000 the first function block will be interrupted and aborted because the test signal starts the Run of the second function block. Now the second function block runs and decelerates the velocity to 2000.

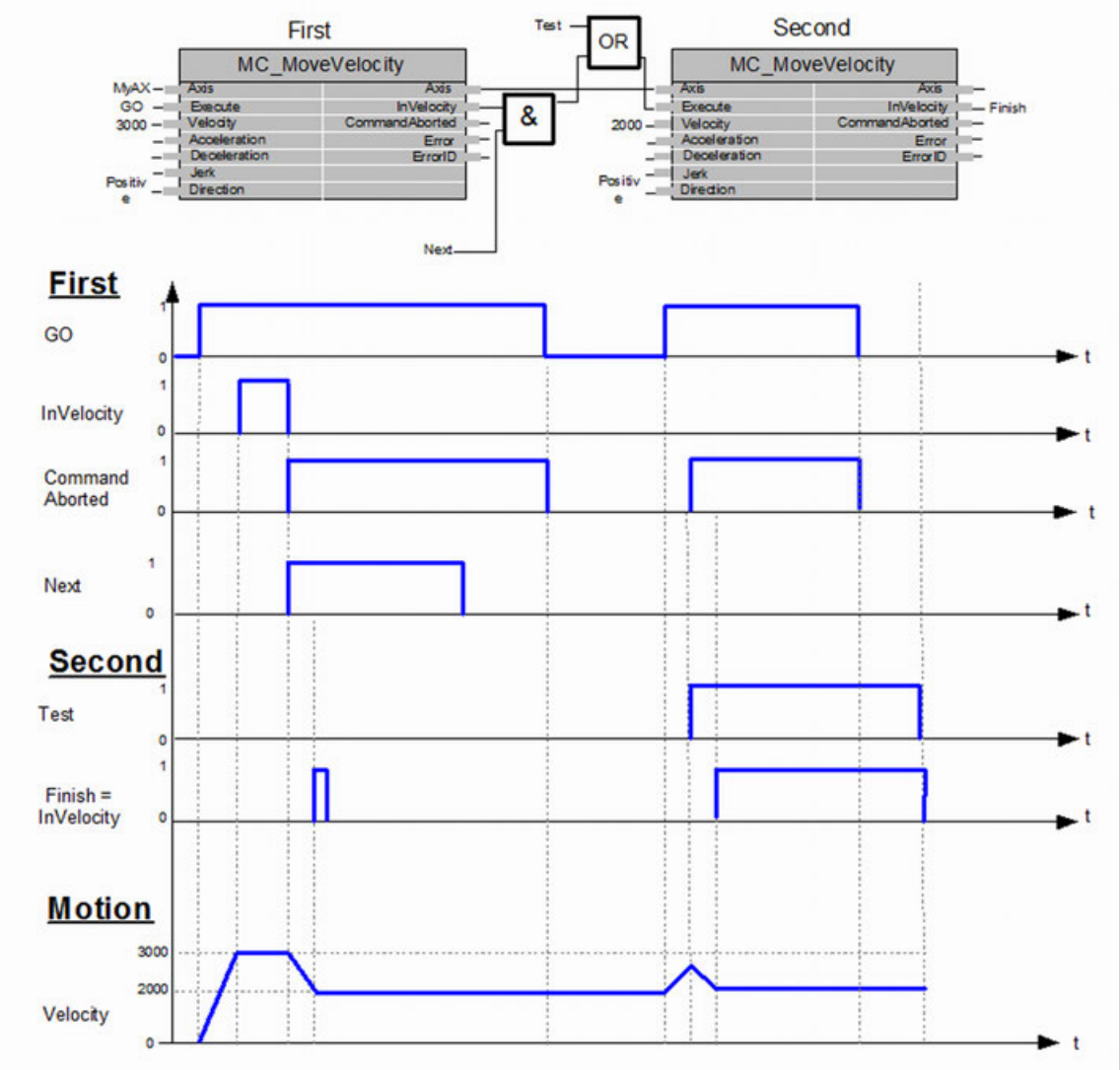


Fig. 312: MC_MoveVelocity timing diagram

Using this function block with FM562

Input Jerk: 1 = jerk on, 2 = jerk off

Input Direction: Enum type: 0 = DEFAULT, 1 = POSITIVE, 3 = NEGATIVE

Input BufferMode: Not implemented

Input description

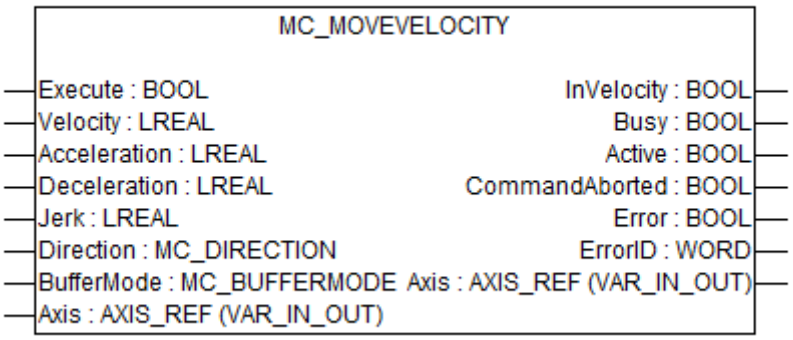


Fig. 313: Function block MC_MoveVelocity

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

Velocity

Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk

Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

Direction

Data type: MC_Direction, range: DEFAULT, POSITIVE, SHORTEST, NEGATIVE, CURRENT

Enum type.

> *Shortest way not applicable, DEFAULT is equivalent with CURRENT*

BufferMode

Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

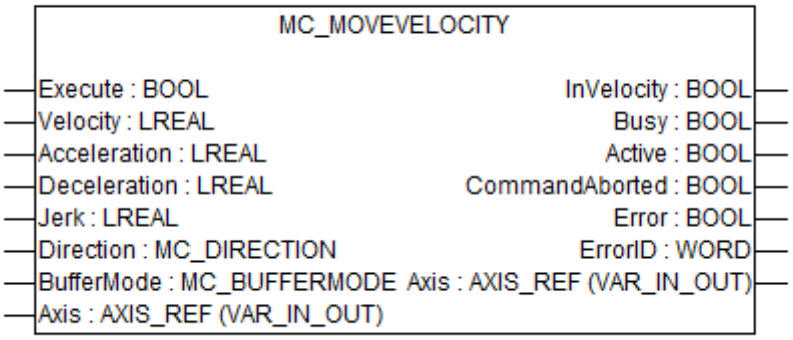


Fig. 314: Function block MC_MoveVelocity

InVelocity Data type: BOOL
Commanded velocity reached (first time reached).

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveContinuousAbsolute

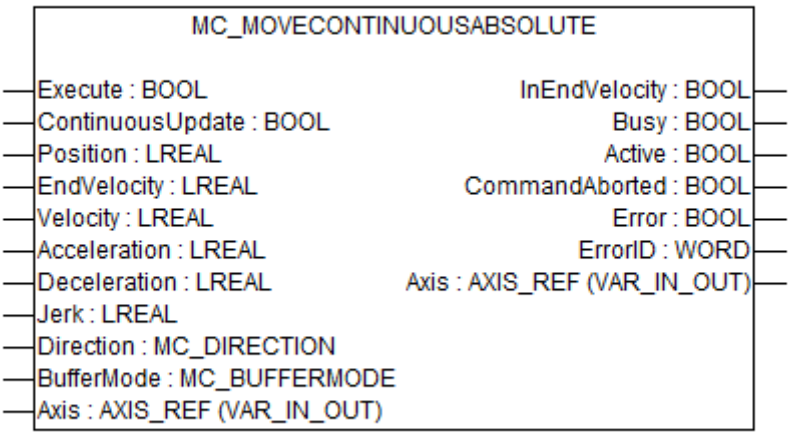


Fig. 315: Function block MC_MoveContinuousAbsolute

This function block commands a controlled motion to a specified absolute position ending with the specified velocity.

- *If the commanded position is reached and no new motion command is put into the buffer, the axis continues to run with the specified "EndVelocity".*
- *The function block will start the axis with state DiscreteMotion, while positioning.*
- *It will change to state Continuous Motion (meaning: it will not stop by itself) with EndVelocity <> 0.*
- *It will change to standstill with EndVelocity=0.*

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

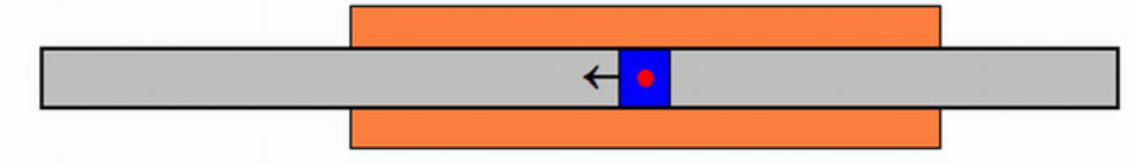
One use case for MC_MoveContinuousAbsolute is a linear cutter. One linear axis that is carrying a laser device that is used to cut a workpiece.

☑ Start from lrIdlePos.

1. Move the laser with fast velocity over the position lrStartCutPos. The laser is off during this movement.

2. Turn back and make sure to have the speed lrCutVelocity when at lrStartCutPos. At this position, switch the laser on.

3. Travel over the work piece with this constant speed while the laser is on.



4. When reaching lrEndCutPos switch off the laser and move back to idle position with fast velocity.

During the cutting process the laser must be moved with a fix velocity, no acceleration or deceleration phase can be tolerated. The laser must be moved to its waiting position after the cutting was done.

Example MC_MoveContinuousAbsolute

The explained movement can be achieved with the function block MC_MoveContinuousAbsolute in the following way:

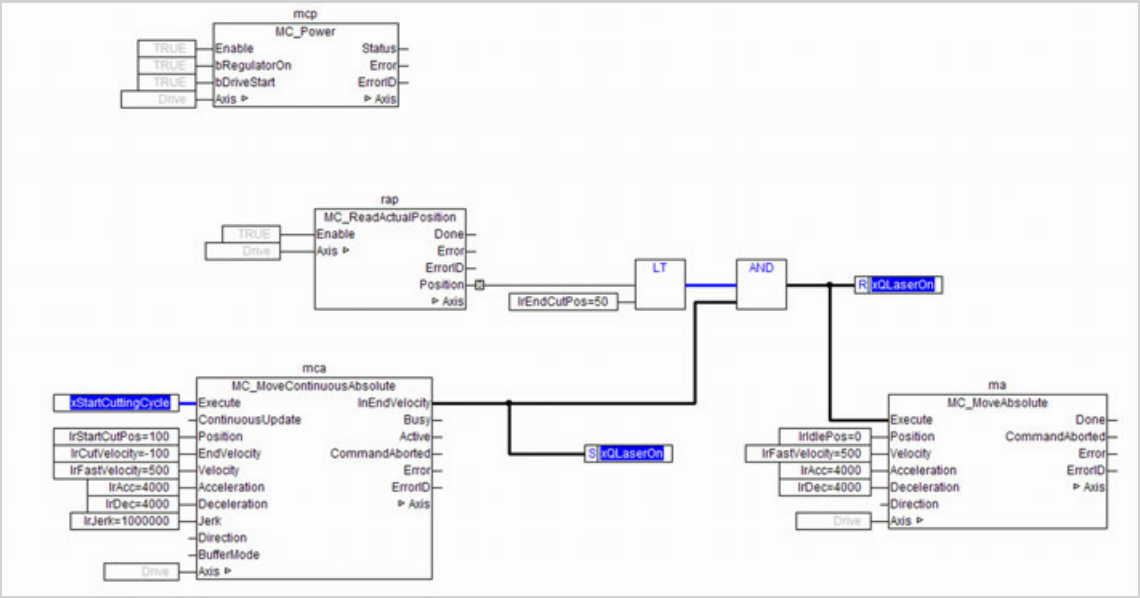


Fig. 316: Example MC_MoveContinuousAbsolute

Started with a rising edge of xStartCuttingCycle, the instance `mca` of MC_MoveContinuousAbsolute will move the axis with lrFastVelocity over lrStartCutPos, turn back and have the speed lrCutVelocity when reaching lrStartCutPos again in negative direction. In this point in time, `InEndVelocity` is set, and the laser is switched on. As no other motion function block interrupts this movement, MC_MoveContinuousAbsolute will keep travelling in negative direction with the current speed. After the axis has overstepped the position lrEndPos, where the laser is switched off, the MC_MoveAbsolute instance `ma` moves the axis with high speed to its idle position.

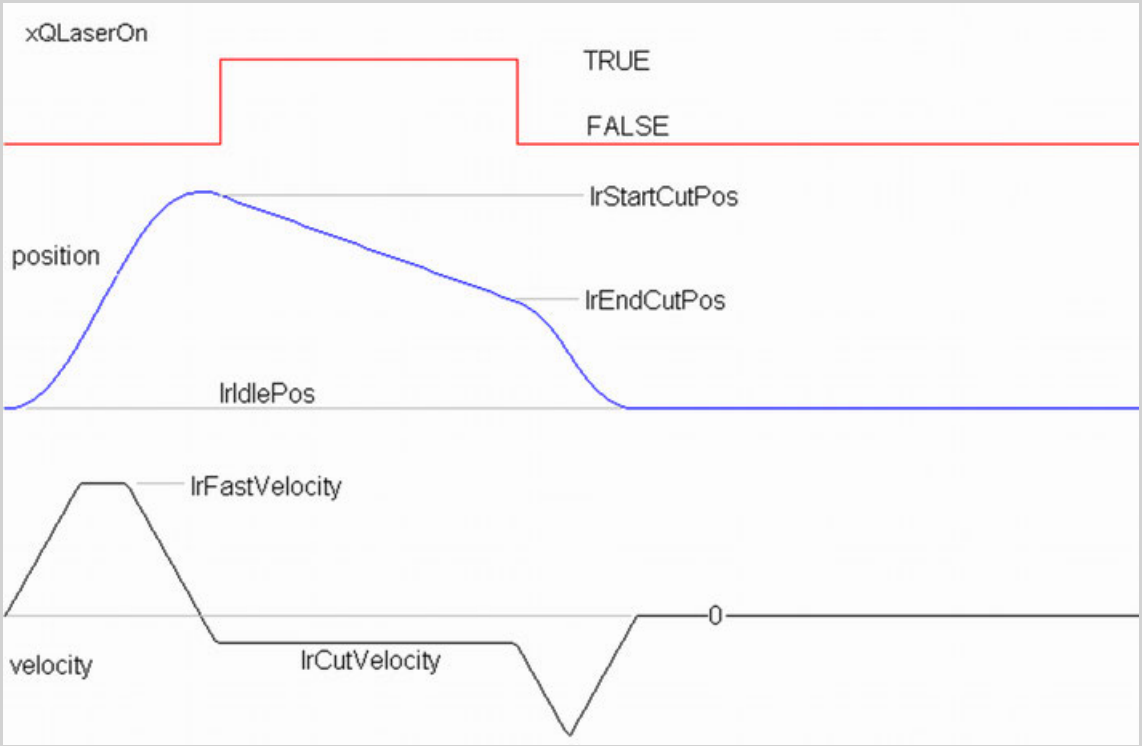


Fig. 317: Example MC_MoveContinuousAbsolute timing diagram

Input description

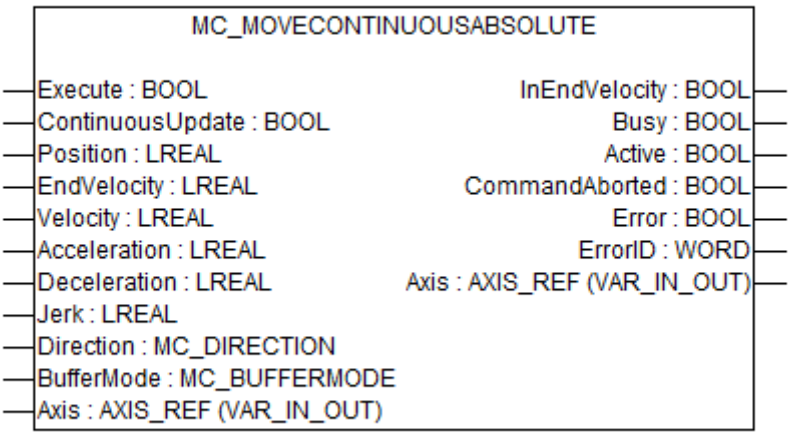


Fig. 318: Function block MC_MoveContinuousAbsolute

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

ContinousUpdate
Data type: BOOL
Chapter 1.5.9.3.9.3 “The input ContinuousUpdate” on page 2604

Position
Data type: LREAL, unit: u
Reference position.

EndVelocity
Data type: LREAL
Value of the end velocity [u/s]. Signed value.

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk

Jerk is not supported with this function block.

Direction Data type: MC_Direction, range: DEFAULT, POSITIVE, SHORTEST, NEGATIVE, CURRENT
Enum type.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

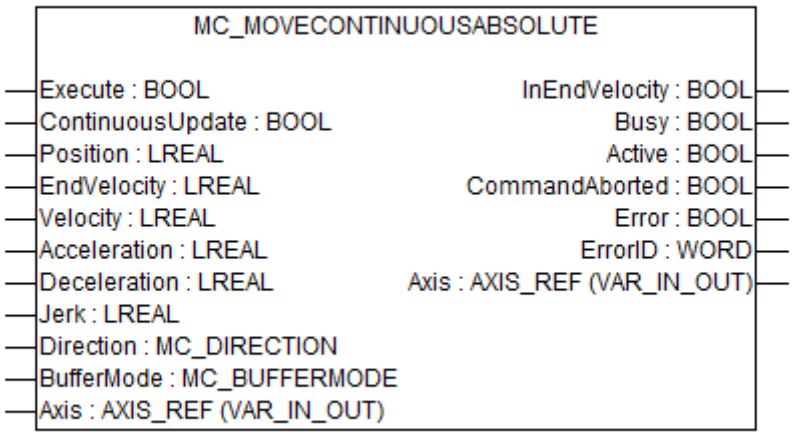


Fig. 319: Function block MC_MoveContinuousAbsolute

InEndVelocity Data type: BOOL
Commanded distance reached and running at requested end velocity .

Busy Data type: BOOL
The function block is not finished and new output values are to be expected.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveContinuousRelative

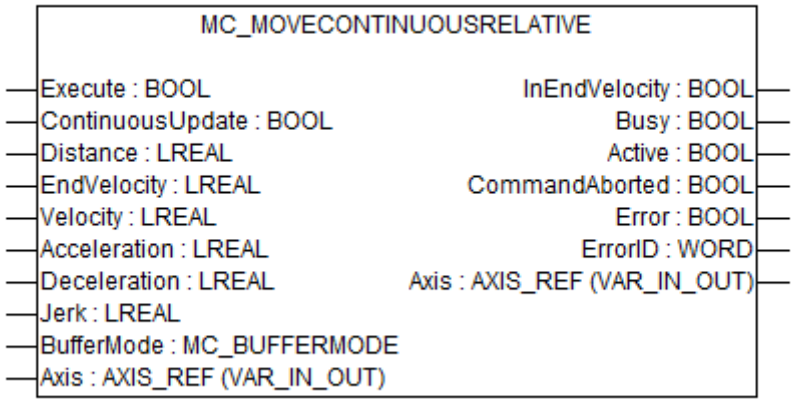


Fig. 320: MC_MoveContinuousRelative

This function block commands a controlled motion of a specified relative distance ending with the specified velocity.

- *If the commanded position is reached and no new motion command is put into the buffer, the axis continues to run with the specified "EndVelocity".*
- *The function block will start the axis with state DiscreteMotion, while positioning.*
- *It will change to state Continuous Motion (meaning: it will not stop by itself) with EndVelocity <> 0.*
- *It will change to standstill with EndVelocity=0.*
- *This function block is specified here for systems without the support for the "BufferMode".*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Sampling traces showing the effect of the sign of the value of the input EndVelocity

Input EndVelocity with positive direction

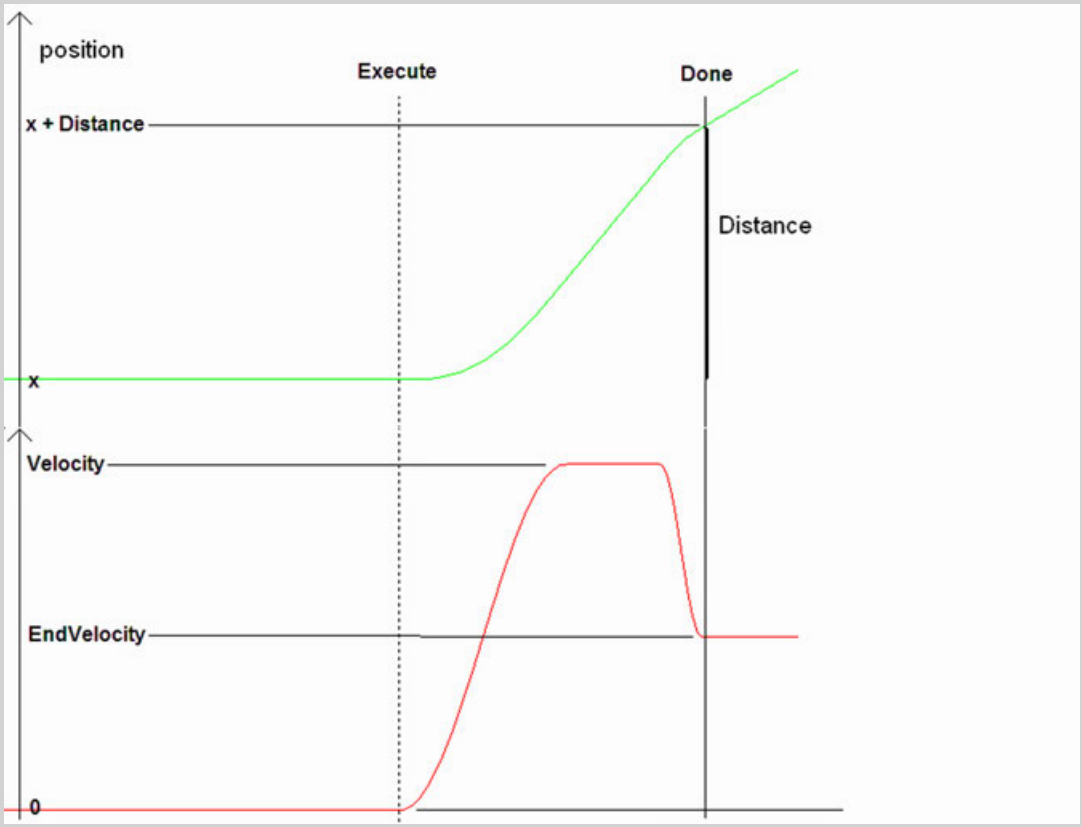


Fig. 321: Timing diagram

Input EndVelocity with negative direction

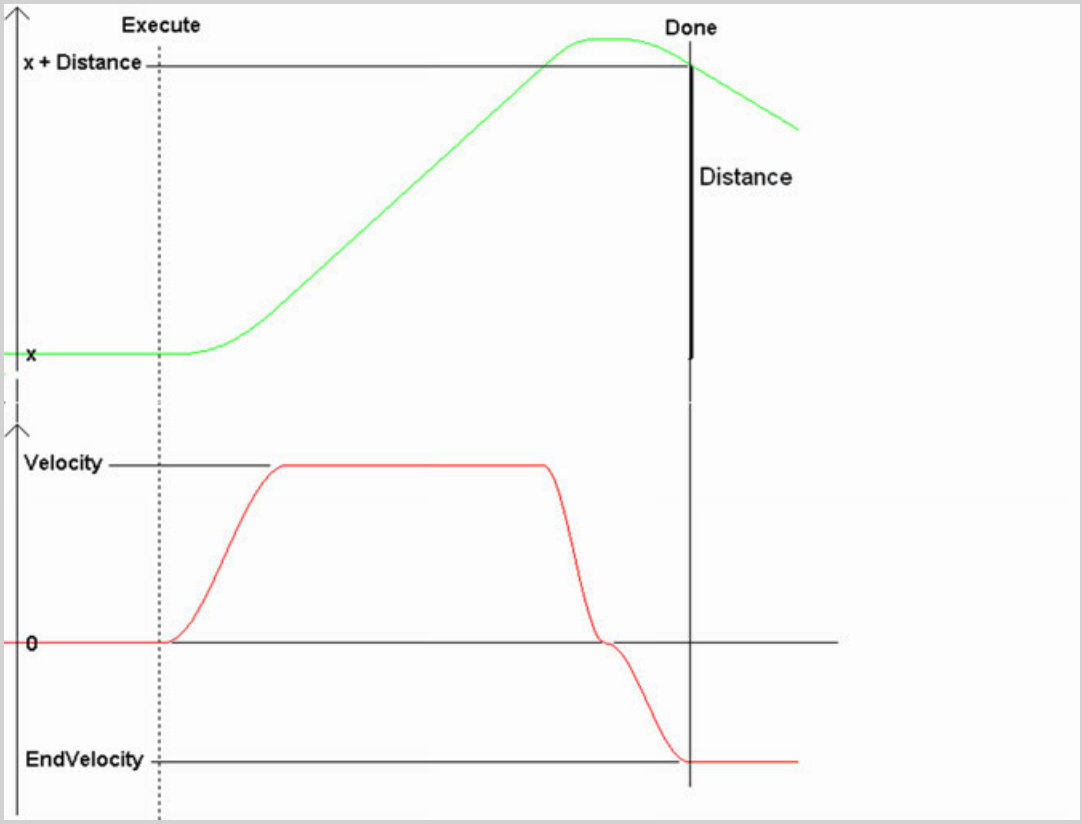
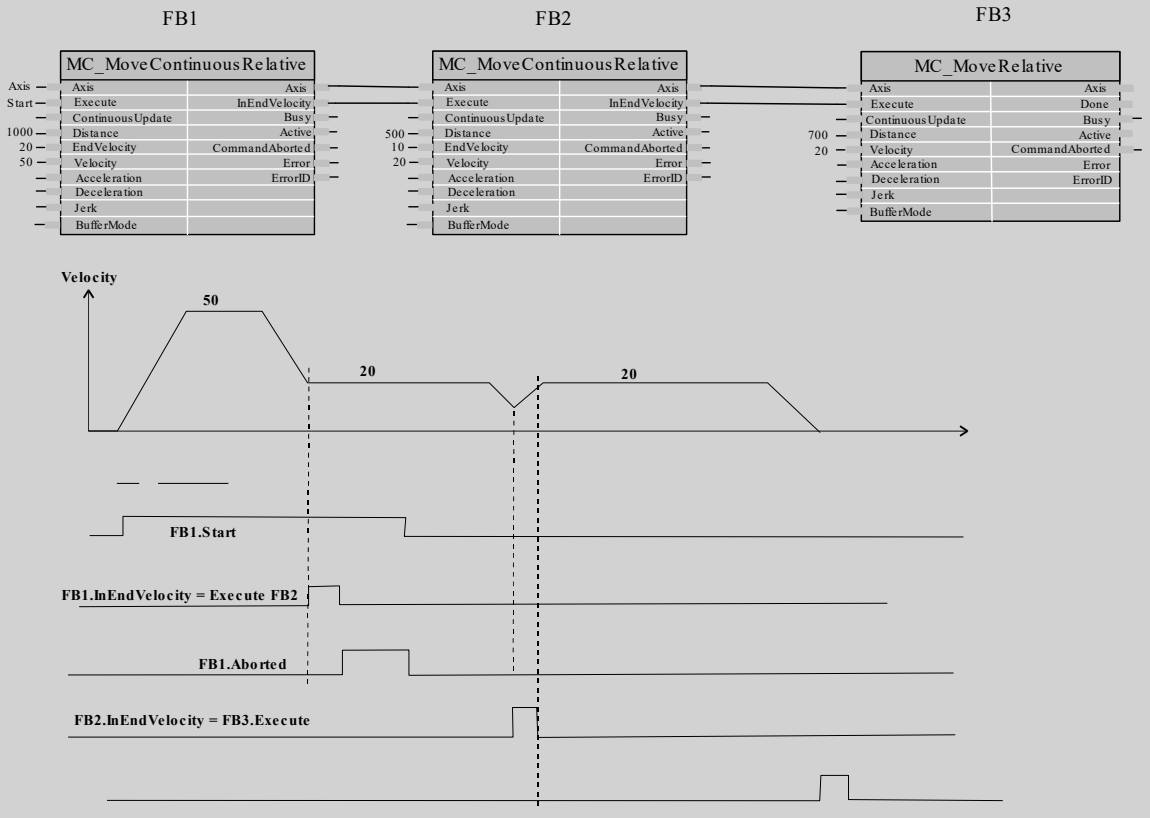


Fig. 322: Timing diagram

Example for MC_MoveContinuousRelative



Input description

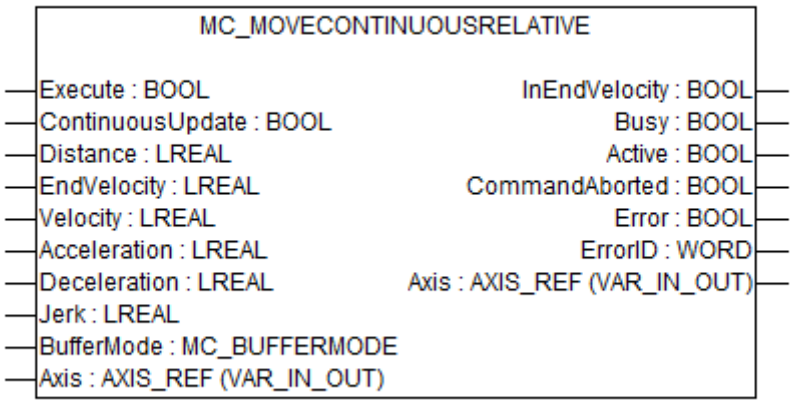


Fig. 323: MC_MoveContinuousRelative

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

ContinousUpdate
Data type: BOOL
Chapter 1.5.9.3.9.3 “The input ContinuousUpdate” on page 2604

Distance
Data type: LREAL, unit: u
Relative distance for the motion.

EndVelocity
Data type: LREAL
Value of the end velocity [u/s]. Signed value.

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk

Jerk is not supported with this function block.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

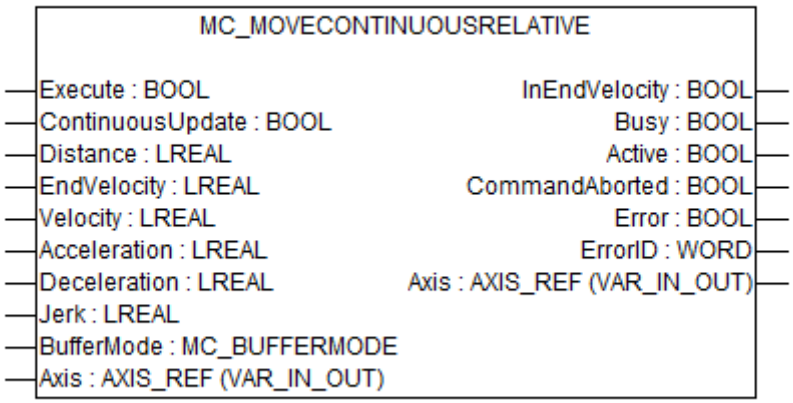


Fig. 324: MC_MoveContinuousRelative

InEndVelocity Data type: BOOL
Commanded distance reached and running at requested end velocity .

Busy Data type: BOOL
The function block is not finished and new output values are to be expected.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_Stop

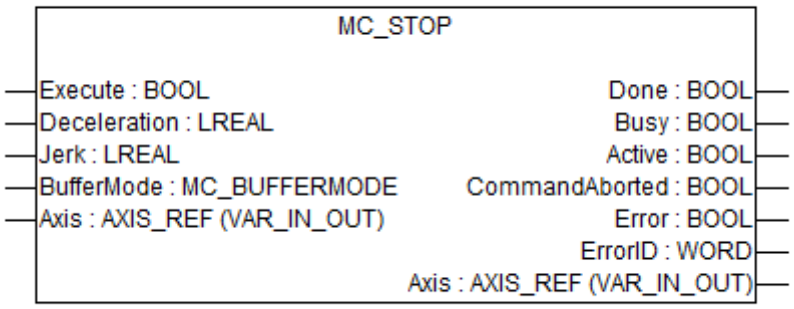


Fig. 325: Function block MC_Stop

Table 189: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block commands a controlled motion stop and transfers the axis to the state Stopping. It aborts any ongoing function block execution. While the axis is in state Stopping, no other function block can perform any motion on the same axis. After the axis has reached velocity zero, the Done output is set to TRUE immediately. The axis remains in the state Stopping as long as Execute is still TRUE or velocity zero is not yet reached. As soon as Done is SET and Execute is FALSE the axis goes to state StandStill.

As long as Execute is high, the axis remains in the state Stopping and may not be executing any other command.

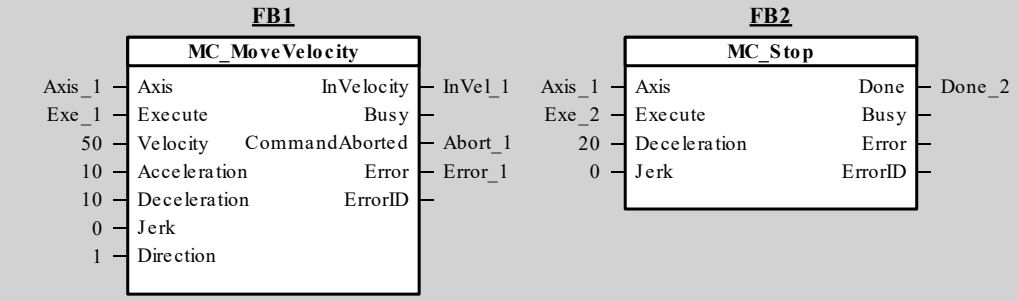
See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

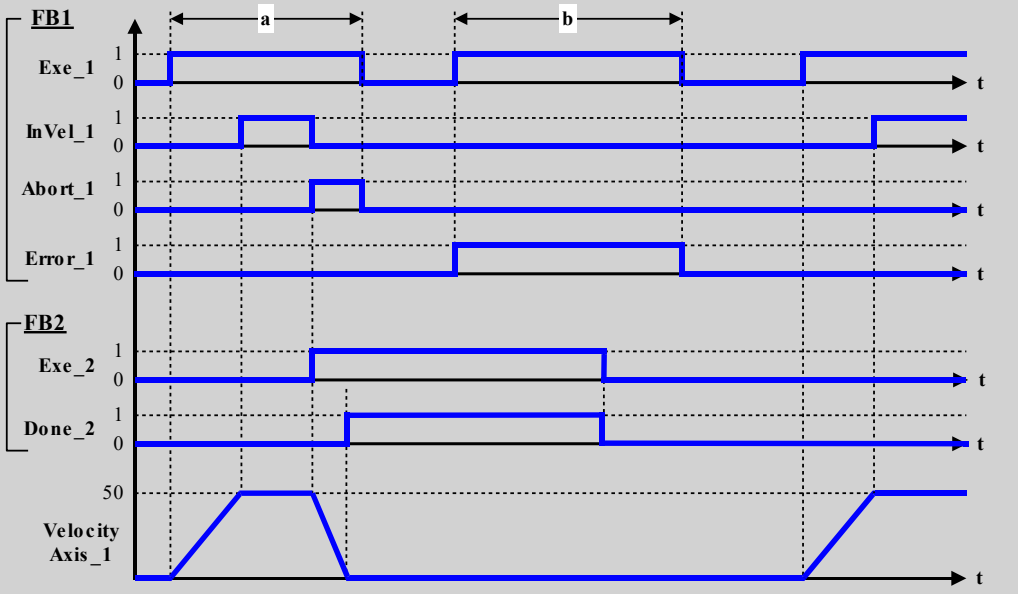
For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Example: Behavior of MC_Stop in combination with MC_MoveVelocity

- A rotating axis is ramped down with function block MC_Stop.
- The axis rejects motion commands as long as MC_Stop parameter Execute = TRUE. Function block MC_MoveVelocity reports an error indicating the busy MC_Stop command.





Using this function block with FM562

Input Jerk: Not implemented

Input BufferMode: Not implemented

Input description

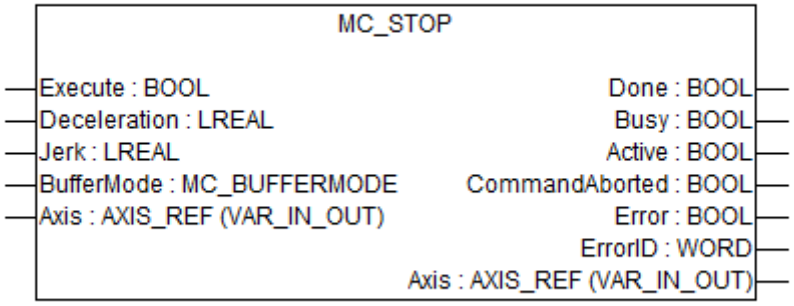


Fig. 326: Function block MC_Stop

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute Data type: BOOL
Starts the function block at rising edge.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

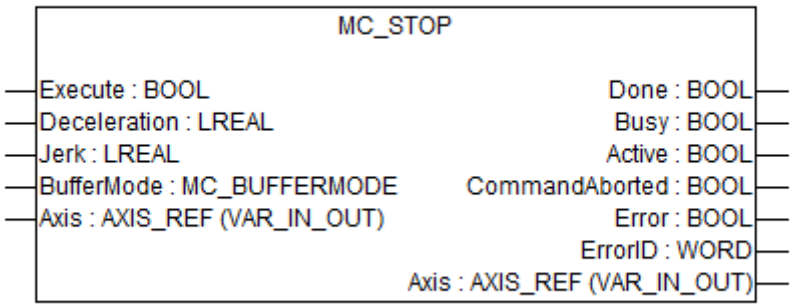


Fig. 327: Function block MC_Stop

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_Halt

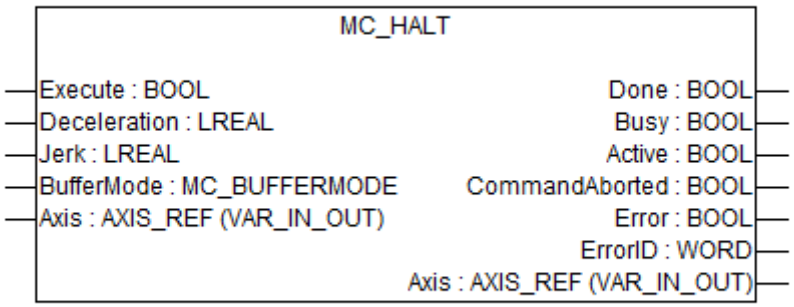


Fig. 328: Function block MC_Halt

This function block commands a controlled motion stop. The axis is moved to the state Discrete Motion, until the velocity is zero. With the Done output set, the state is transferred to StandStill.

- *MC_Halt is used to stop the axis under normal operation conditions. In non-buffered mode it is possible to set another motion command during deceleration of the axis, which will abort the MC_Halt and will be executed immediately.*
- *If this command is active the next command can be issued. E.g. a driverless vehicle detects an obstacle and needs to stop. MC_Halt is issued. Before the standstill is reached the obstacle is removed and the motion can be continued by setting another motion command, so the vehicle does not stop.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Example: Behavior of MC_Halt in combination with MC_Move-Velocity

- A rotating axis is ramped down with function block MC_Halt.
- Another motion command overrides the MC_Halt command. MC_Halt allows this, in contrast to MC_Stop. The axis can accelerate again without reaching standstill.

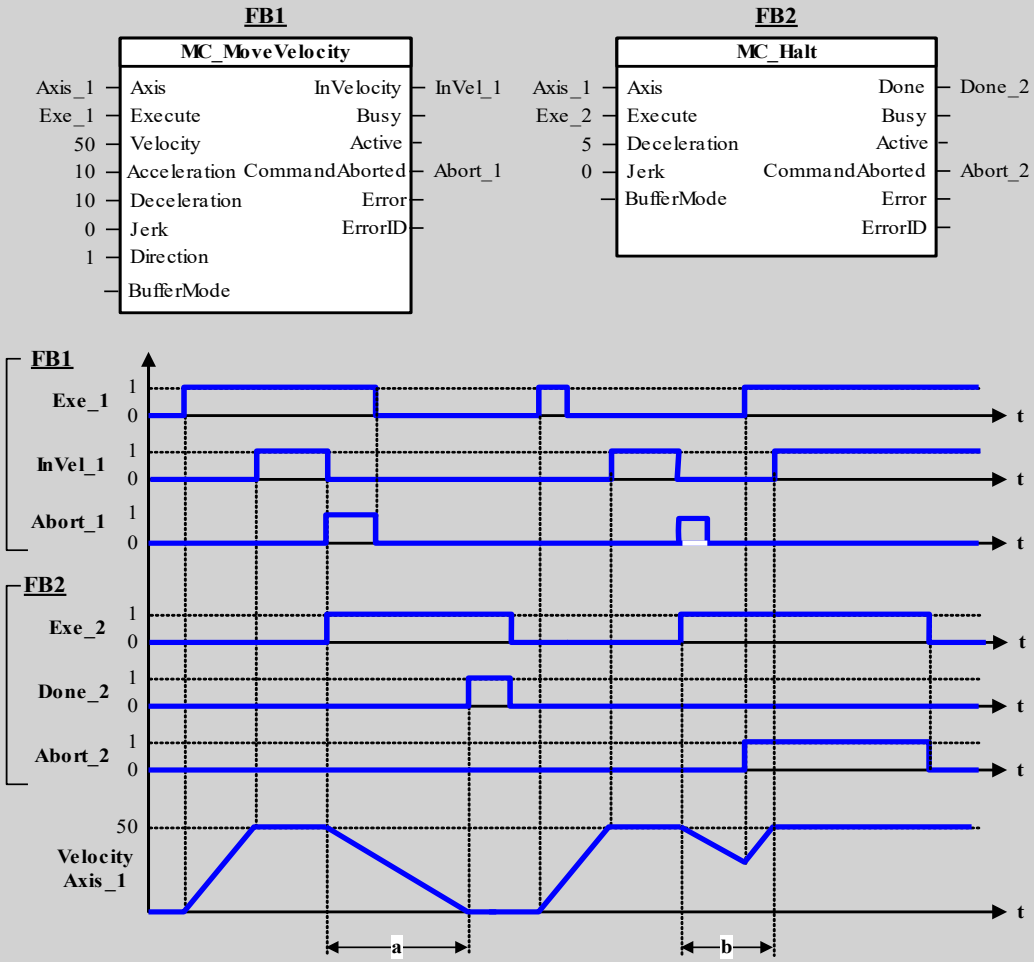


Fig. 329: Example MC_Halt

Input description

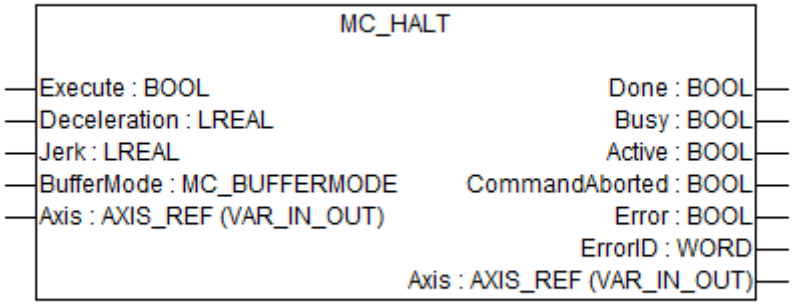


Fig. 330: Function block MC_Halt

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

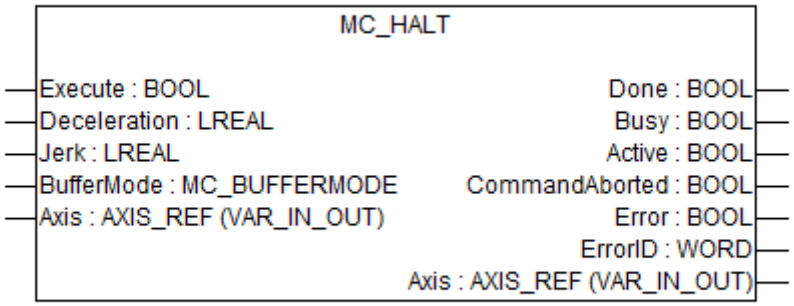


Fig. 331: Function block MC_Halt

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_PositionProfile

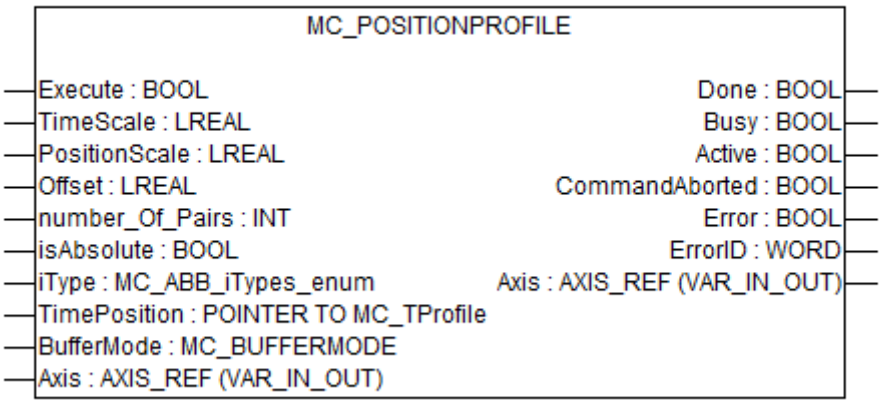
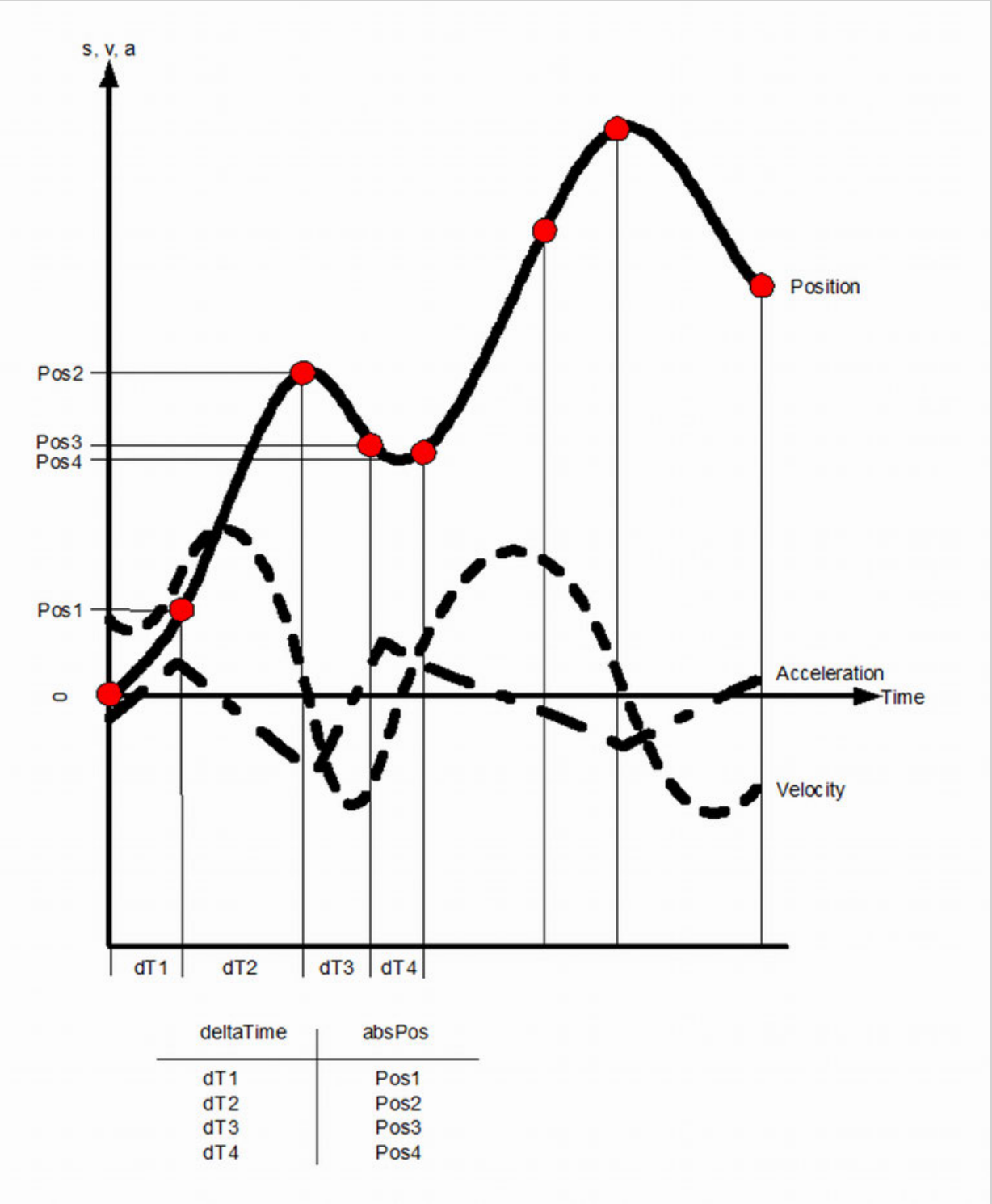


Fig. 332: Function block MC_PositionProfile

This function block commands a time-position locked motion profile .

- *MC_TPROFILE is an ABB specific data type.*
- *This functionality does not mean it runs one profile over and over again: It can shift between different profiles.*
- *Alternatively to this function block, the CAM function block coupled to a virtual master can be used.*

Example of time/position profile



deltaTime	absPos
dT1	Pos1
dT2	Pos2
dT3	Pos3
dT4	Pos4

The Time / Velocity and Time / Acceleration Profiles are similar to the Position Profile, with sampling points on the Velocity or Acceleration lines.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

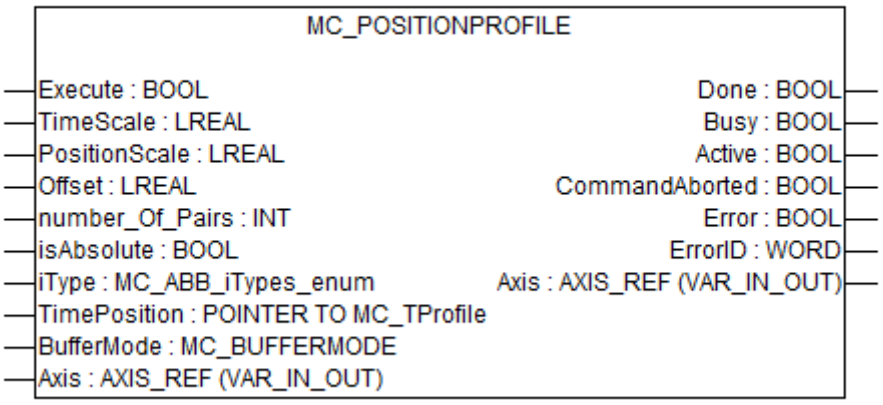


Fig. 333: Function block MC_PositionProfile

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

TimePosition
Data type: MC_TP_REF
Reference to Time/Position. Description: *Further information on page 2787*

Execute
Data type: BOOL
Starts the function block at rising edge.

TimeScale
Data type: LREAL, unit: t.u.
Overall time scaling factor of the profile.

PositionScale
Data type: LREAL, unit: t.u.
Overall Position scaling factor.

Offset
Data type: LREAL, unit: u/s
Overall offset for profile.

Number_of_pairs
Data type: INT
Number of points used in TimePosition Array.

isAbsolute Data type: BOOL

TRUE → Profile holds absolute position values.

iTYPE Data type: MC_ABB_iTypes_ENUM

Interpolationtype, possible values are:

- MCA_SPLINE_COMPLETE
- MCA_SPLINE_NATURAL
- MCA_POLY5
- MCA_POLY3
- MCA_LINEAR

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

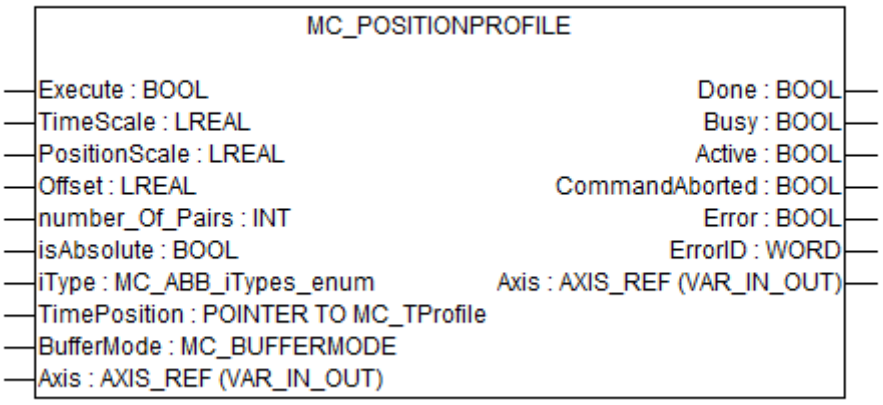


Fig. 334: Function block MC_PositionProfile

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted
Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error
Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID
Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_VelocityProfile

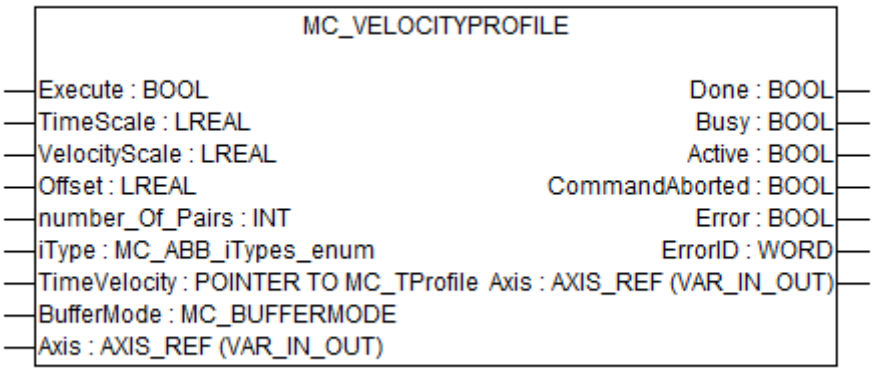


Fig. 335: MC_VelocityProfile

This function block commands a time-velocity locked motion profile.

- *MC_TPROFILE is an ABB specific data type.*
- *This functionality does not mean it runs one profile over and over again: It can shift between different profiles.*
- *Alternatively to this function block, the CAM function block coupled to a virtual master can be used.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

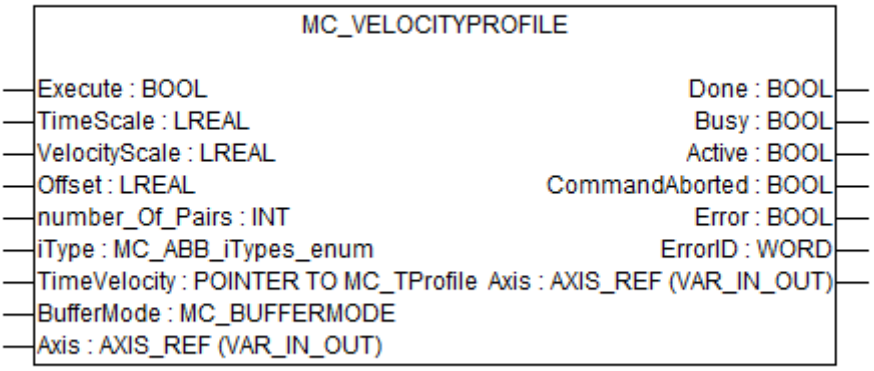


Fig. 336: Function block MC_VelocityProfile

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

TimeScale Data type: LREAL, unit: t.u.

Overall time scaling factor of the profile.

VelocityScale Data type: LREAL

Overall velocity scaling factor of the profile.

Offset Data type: LREAL, unit: u/s

Overall offset for profile.

Number_of_pairs Data type: INT

Number of points used in TimeVelocity Array.

iTYPE Data type: MC_ABB_iTypes_ENUM

Interpolationtype, possible values are:

- MCA_SPLINE_COMPLETE
- MCA_SPLINE_NATURAL
- MCA_POLY5
- MCA_POLY3
- MCA_LINEAR

TimeVelocity Data type: POINTER TO MC_TPROFILE
Reference to time/velocity. MC_TPROFILE is an ABB specific data type.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

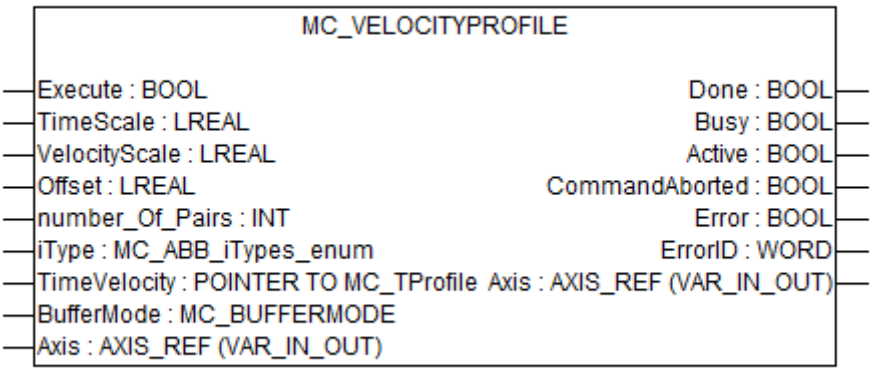


Fig. 337: Function block MC_VelocityProfile

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_AccelerationProfile

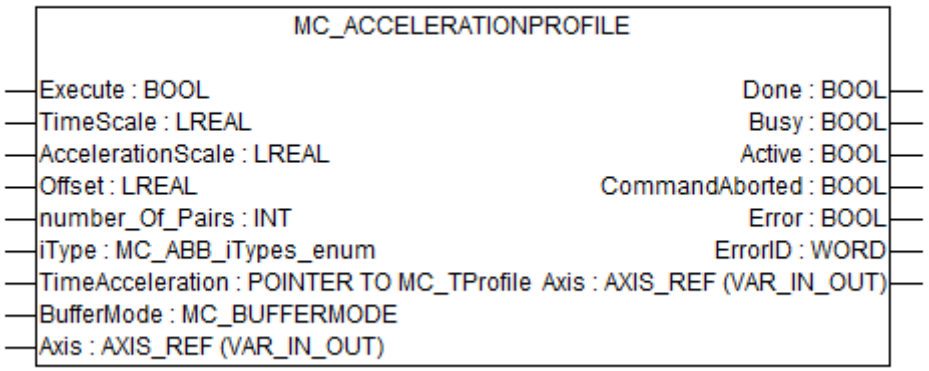


Fig. 338: Function block MC_AccelerationProfile

This function block commands a time-acceleration locked motion profile

- *MC_TPROFILE is an ABB specific datatype.*
- *Alternatively to this function block, the CAM function block coupled to a virtual master can be used.*

Example of an acceleration profile

A profile is made from a number of sequential "A to B" positioning points. It is simple to visualize, but requires a lot of sequences for a smooth profile. These requirements are often beyond the capability of low-end servos. Alternatively, by using a modest amount of constant acceleration segments it is possible to define a well-matching motion profile. With this method the capability range of low-end servos can be extended. It is possible to make matching to either:

1.) Position versus time profile

2.) Master versus slave axis

Advantages:

- Compact description of a profile.
- Smooth profile properties by nature.
- Low processor power requirements.

Disadvantages:

- Higher programming abstraction level with existing tools.

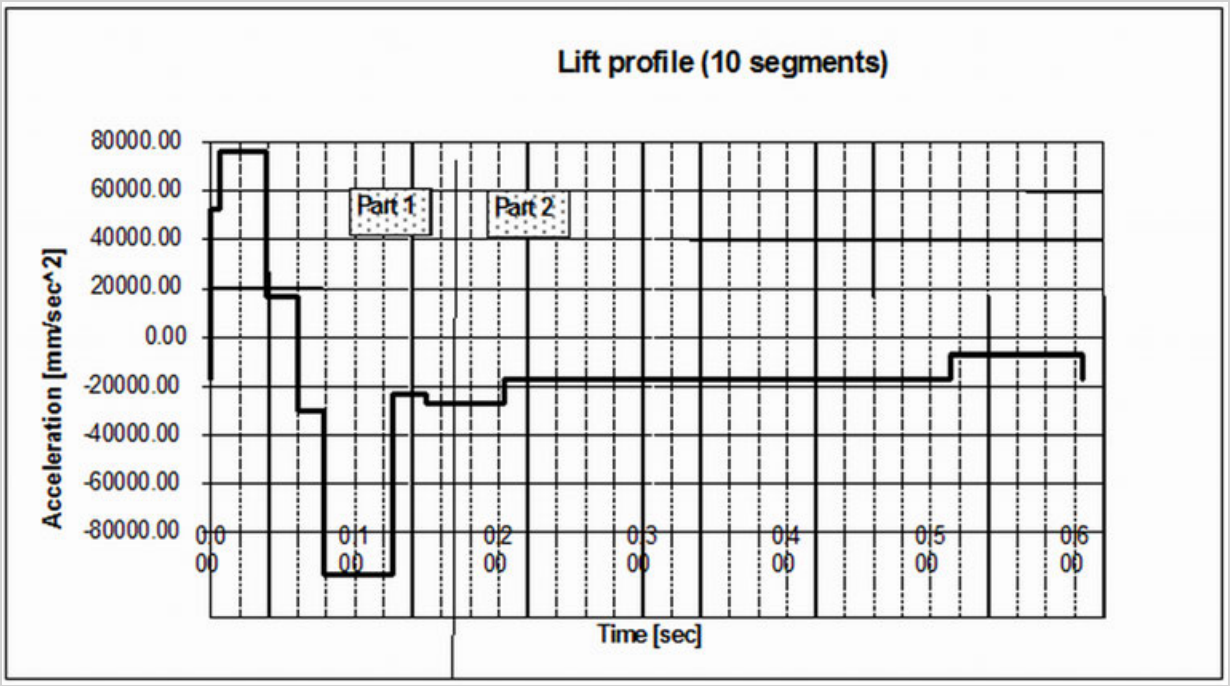


Fig. 339: Acceleration Profile, 10 segments only

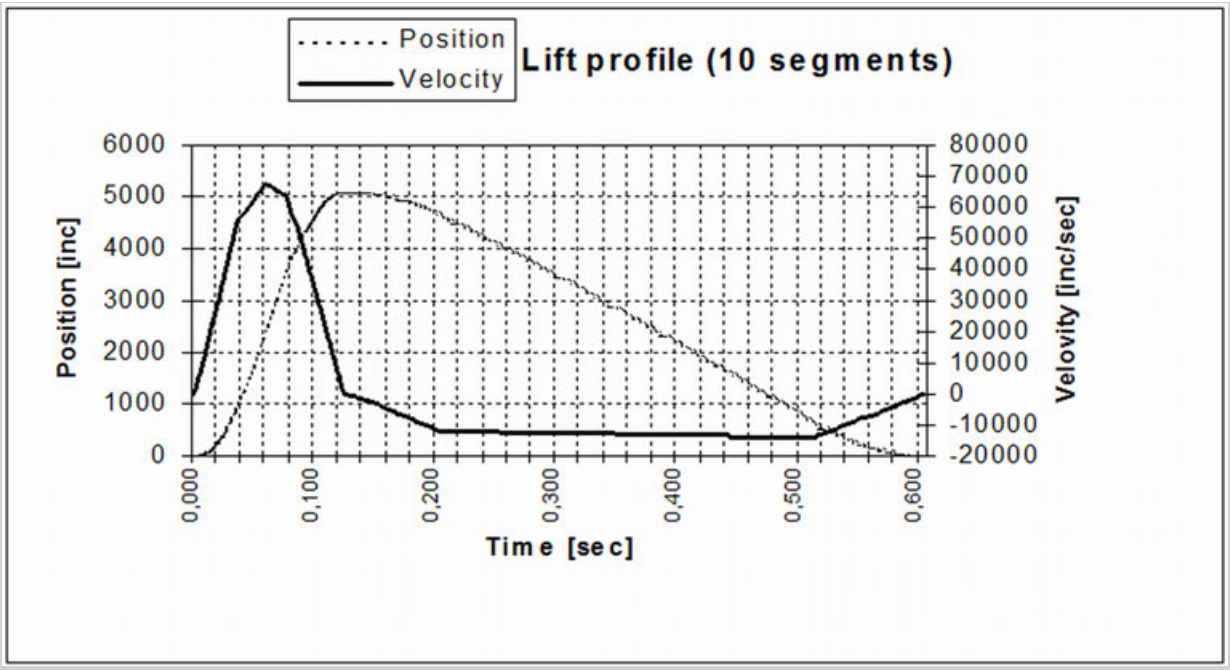


Fig. 340: Resulting Position Profile

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

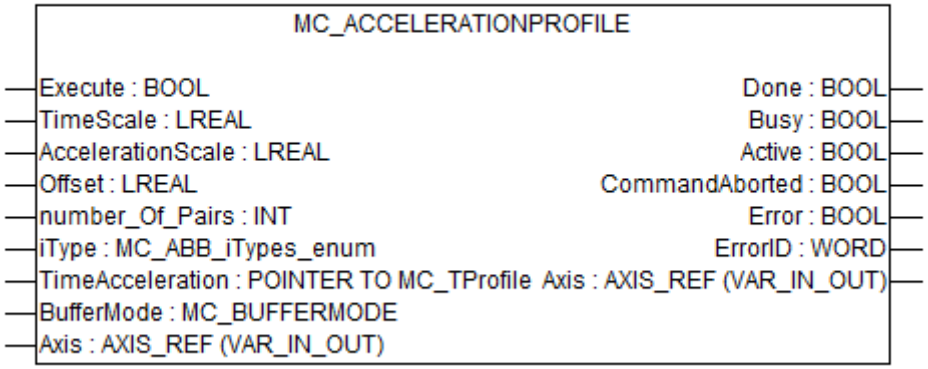


Fig. 341: Function block MC_AccelerationProfile

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute
Data type: BOOL
Starts the function block at rising edge.

TimeScale
Data type: LREAL, unit: t.u.
Overall time scaling factor of the profile.

Acceleration-Scale
Data type: LREAL
Scale factor for acceleration amplitude.

Offset
Data type: LREAL, unit: u/s
Overall offset for profile.

Number_of_pairs
Data type: INT
Number of points used in TimeAcceleration Array.

iTYPE
Data type: MC_ABB_iTypes_ENUM
Interpolationtype, possible values are:

- MCA_SPLINE_COMPLETE
- MCA_SPLINE_NATURAL
- MCA_POLY5
- MCA_POLY3
- MCA_LINEAR

TimeAcceleration
Data type: MC_TA_REF
Reference to Time / Acceleration. Description: *“Example of an acceleration profile” on page 2795*

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis
Data type: AXIS_REF
Reference to the axis.

Output description

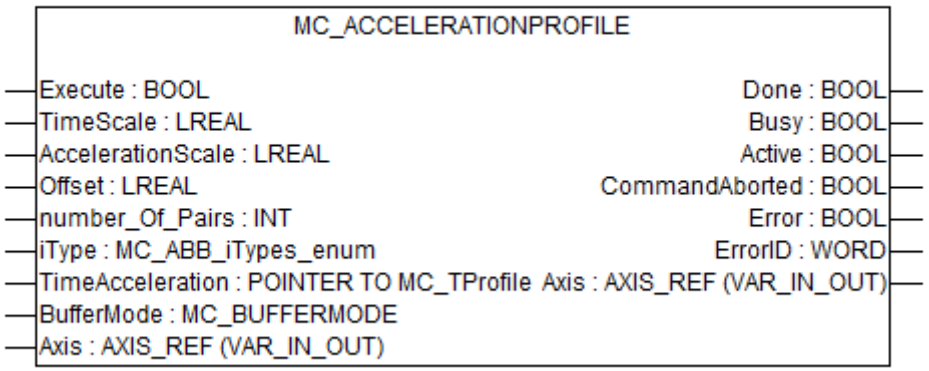


Fig. 342: Function block MC_AccelerationProfile

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

1.5.9.6.2 Multi-Axis function blocks

With Multi-Axis function blocks a synchronized relationship exists between two or more axes. The synchronization can be related to time or position. Often this relationship is between a master axis and one or more slave axes. A master axis can be a virtual axis. From the single axis state diagram point of view, the multi-axis function blocks related to camming and gearing can be looked at as a master axis in one state (for instance: MC_MoveContinuous) and the slave axis in a specific synchronized state, called SychronizedMotion. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

MC_CamIn

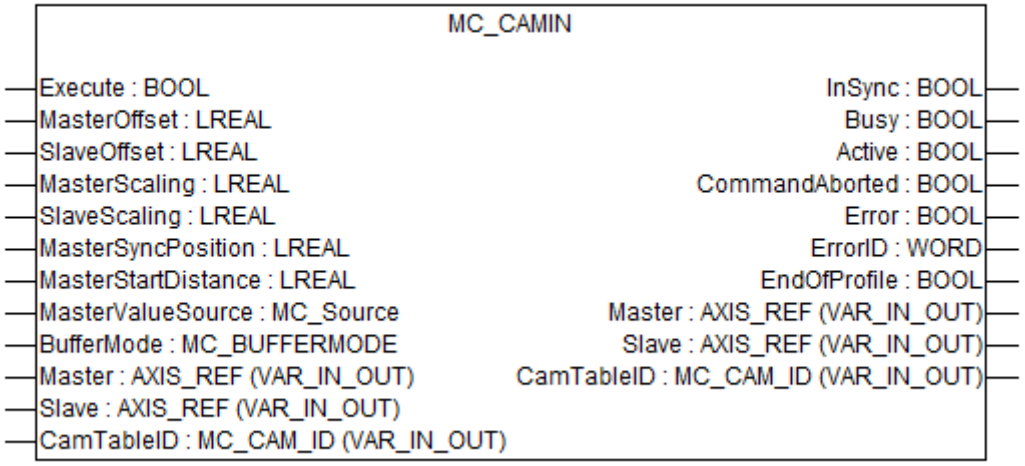


Fig. 343: Function block MC_CamIn

This function block engages the CAM.

- *It is not required that the master is stationary.*
- *If the actual master and slave positions do not correspond to the offset values when MC_CamIn is executed, either an error occurs or the system deals with the difference automatically.*
- *The Cam is placed either absolute or relative to the current master and slave positions.*
  Absolute: The profile between master and slave is seen as an absolute relationship.
  Relative: The relationship between master and slave is in a relative mode.
- *If a cam-table is to be used "relative", the first position has to be =0.*
- *This function block is not merged with the MC_CamTableSelect function block because this separation enables changes on the fly.*
- *A mechanical analogy to a slave offset is a cam welded with additional constant layer thickness. Because of this the slave positions have a constant offset and the offset could be interpreted as axis offset of the master shaft, if linear guided slave tappets are assumed.*

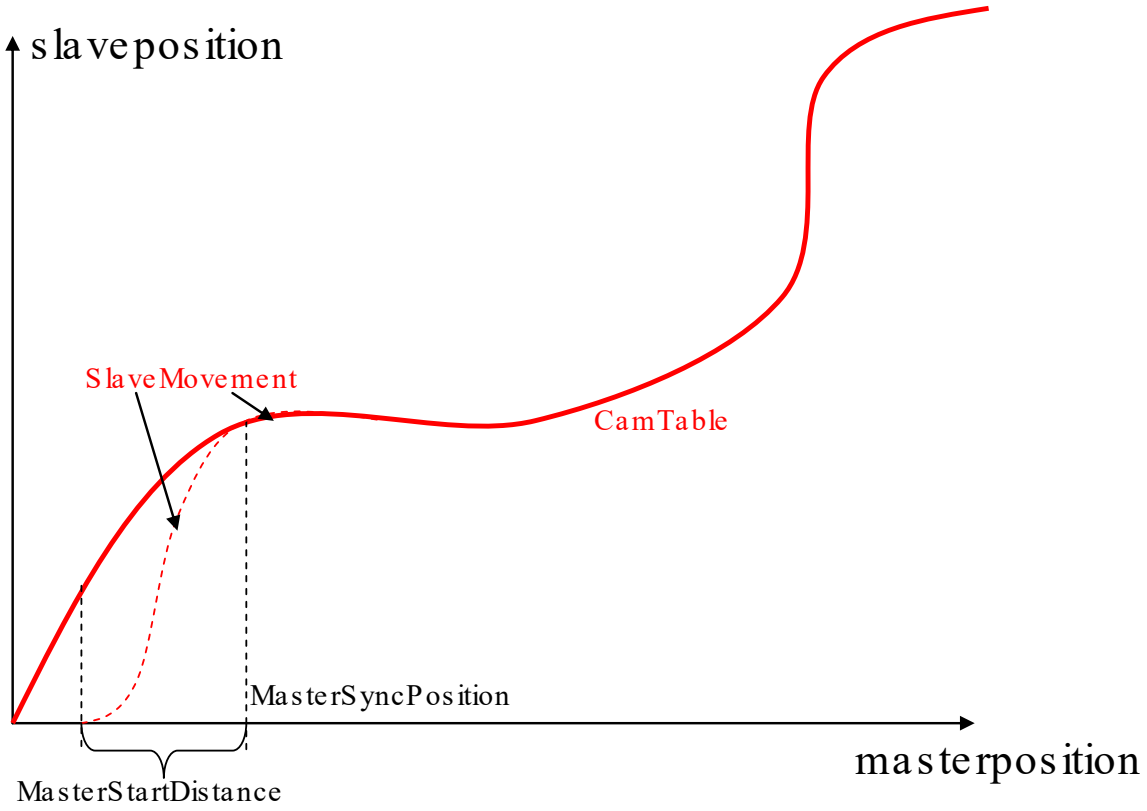
See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

In case MasterSyncPosition and MasterStartDistance are = 0 (default value), the Cam-Function is started with the rising edge of “Execute”. When the master did not yet reach a position in the cam area, the slave axis is stopped.

With MasterStartDistance <> 0 the slave axis needs to be in state StandStill when activating MC_CamIn. The function block will wait until the master axis reaches the position MasterSyncPosition-MasterStartDistance. The slave will then be started and be synchronized to the CAM table, to the position and velocity which is indicated by the master position MasterSyncPosition.



The MC_CamIn has parameters to scale the cam-table values (MasterScaling, SlaveScaling). It has to be considered that MasterOffset and SlaveOffset are scaled exactly like the corresponding cam-table values. The MasterSyncPosition and MasterStartDistance are not scaled at all, these positions are related to the actual master position whereas the MasterOffset and SlaveOffset are related to the cam-table.

Input description

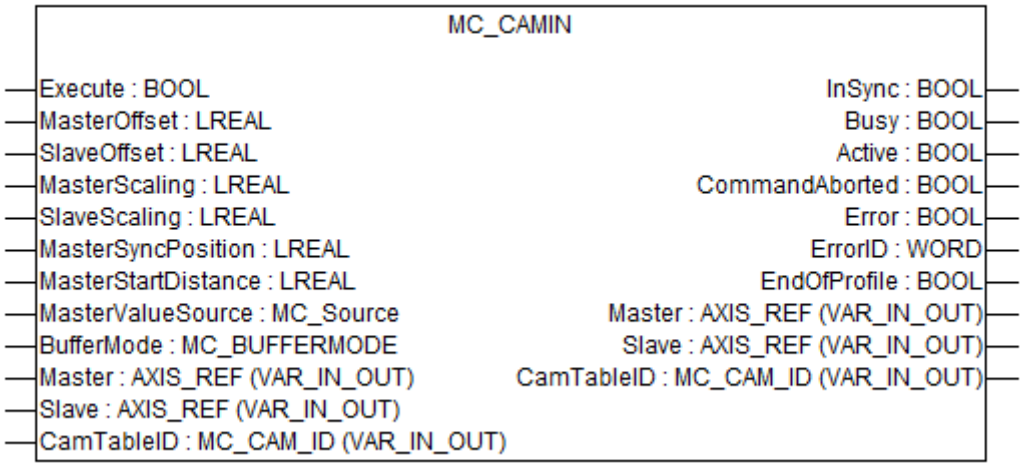


Fig. 344: Function block MC_CamIn

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute

Data type: BOOL

Starts the function block at rising edge.

MasterOffset

Data type: LREAL

Offset of master table. Angular offset of the master shaft to cam.

SlaveOffset

Data type: LREAL, default: 0

Offset of slave table. Sharpened cam (i.e higher elevation and deeper depression).

MasterScaling

Data type: LREAL, default: 1.0

Factor for the master profile. From the slave point of view the master overall profile is multiplied by this factor.

SlaveScaling

Data type: LREAL, default: 1.0

Factor for the slave profile. The overall slave profile is multiplied by this factor.

MasterSyncPosition

Data type: LREAL

The position of the master in the path where the group is insync with the master. (If the ‘MasterSyncPosition’ does not exist, at the first point of the path the master and slave are synchronized).

The inputs acceleration, deceleration and jerk are not added here.

MasterStartDistance

Data type: LREAL

The master distance for the slave to start to synchronize to the master.

MasterValueSource

Data type: MC_SOURCE

Defines the source for synchronization:

mcSetValue - Synchronization on master set value.

mcActualValue - Synchronization on master actual value.

BufferMode

Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Master

Data type: AXIS_REF

Reference to the master axis.

Slave Data type: AXIS_REF
Reference to the slave axis.

CamTableID Data type: MC_CAM_ID
Identifier of CAM Table to be used in the MC_CamIn function block.

Output description

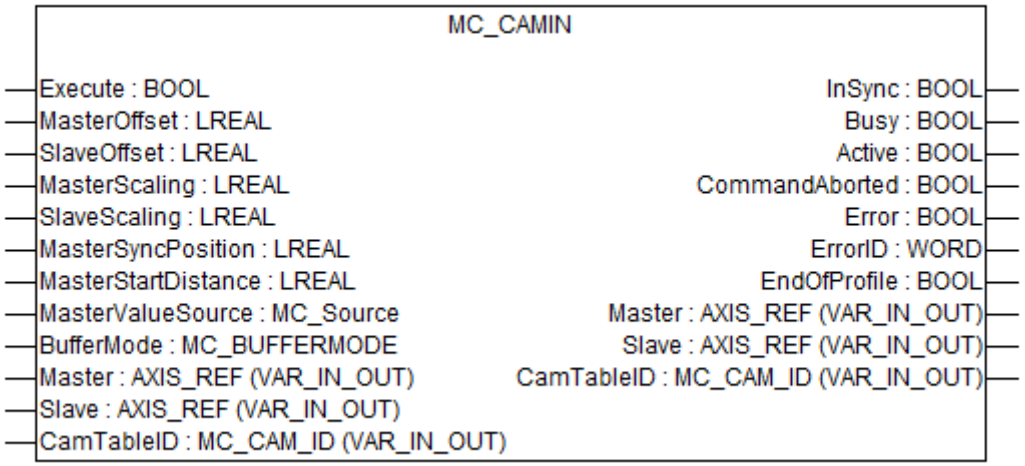


Fig. 345: Function block MC_CamIn

InSync Data type: BOOL
Cam is engaged for the first time.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

EndOfProfile Data type: BOOL

Pulsed output signaling the cyclic end of the CAM Profile.

MC_CamOut

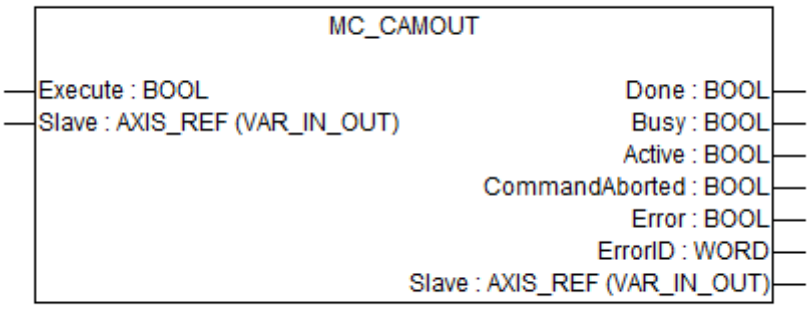


Fig. 346: Function block MC_CamOut

This function block disengages the Slave axis from the Master axis immediately.

> *It is assumed that this command is followed by another command, for instance MC_Stop, MC_GearIn, or any other command. If there is no new command, the default condition should be: Maintain last velocity. If there is no new command the axis will maintain its last velocity.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

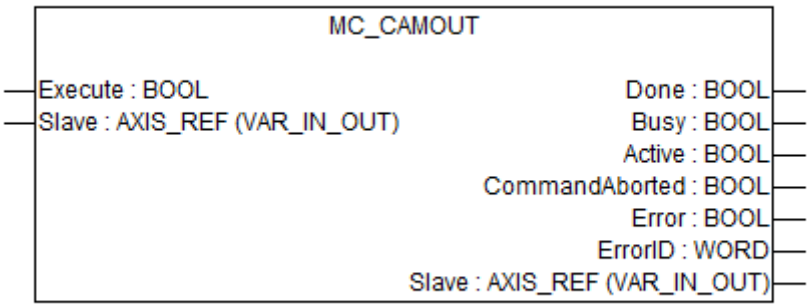


Fig. 347: Function block MC_CamOut

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output description

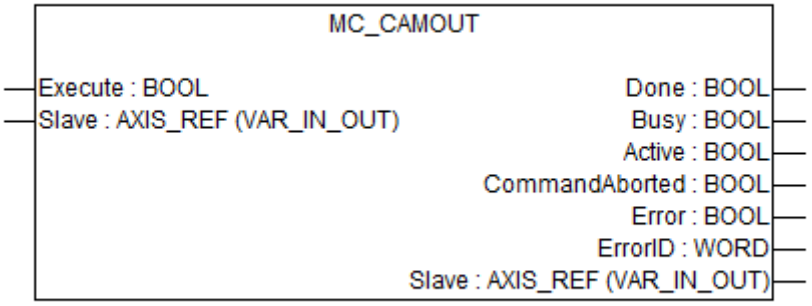


Fig. 348: Function block MC_CamOut

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GearIn

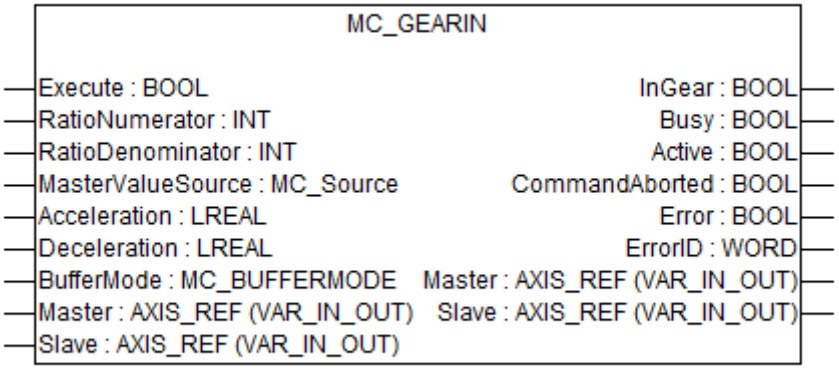
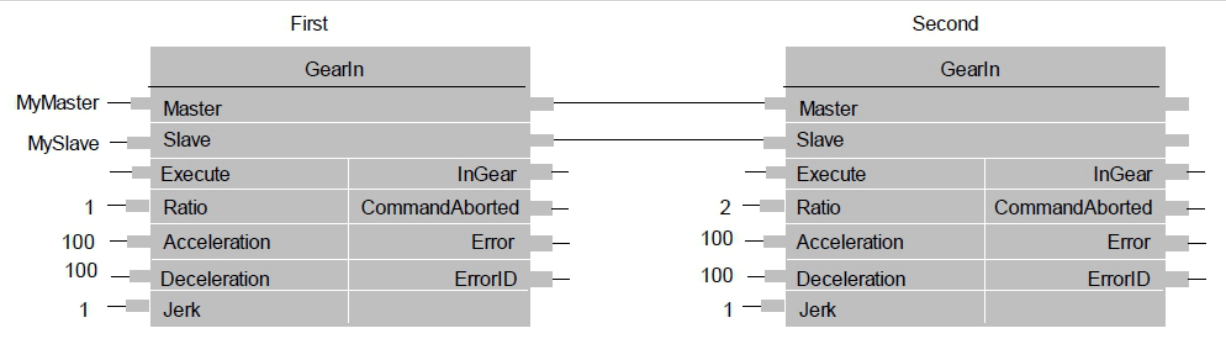


Fig. 349: Function block MC_GearIn

This function block commands a ratio between the VELOCITY of the slave and master axis.

- *The slave ramps up to the ratio of the master velocity and locks in when this is reached. Any lost distance during synchronization is not caught up.*
- *The gearing ratio can be changed while MC_GearIn is running, using a consecutive MC_GearIn command without the necessity to MC_GearOut first*
- *InGear is set the first time the ratio is reached.*
- *After being InGear, a position locking is performed.*

Example



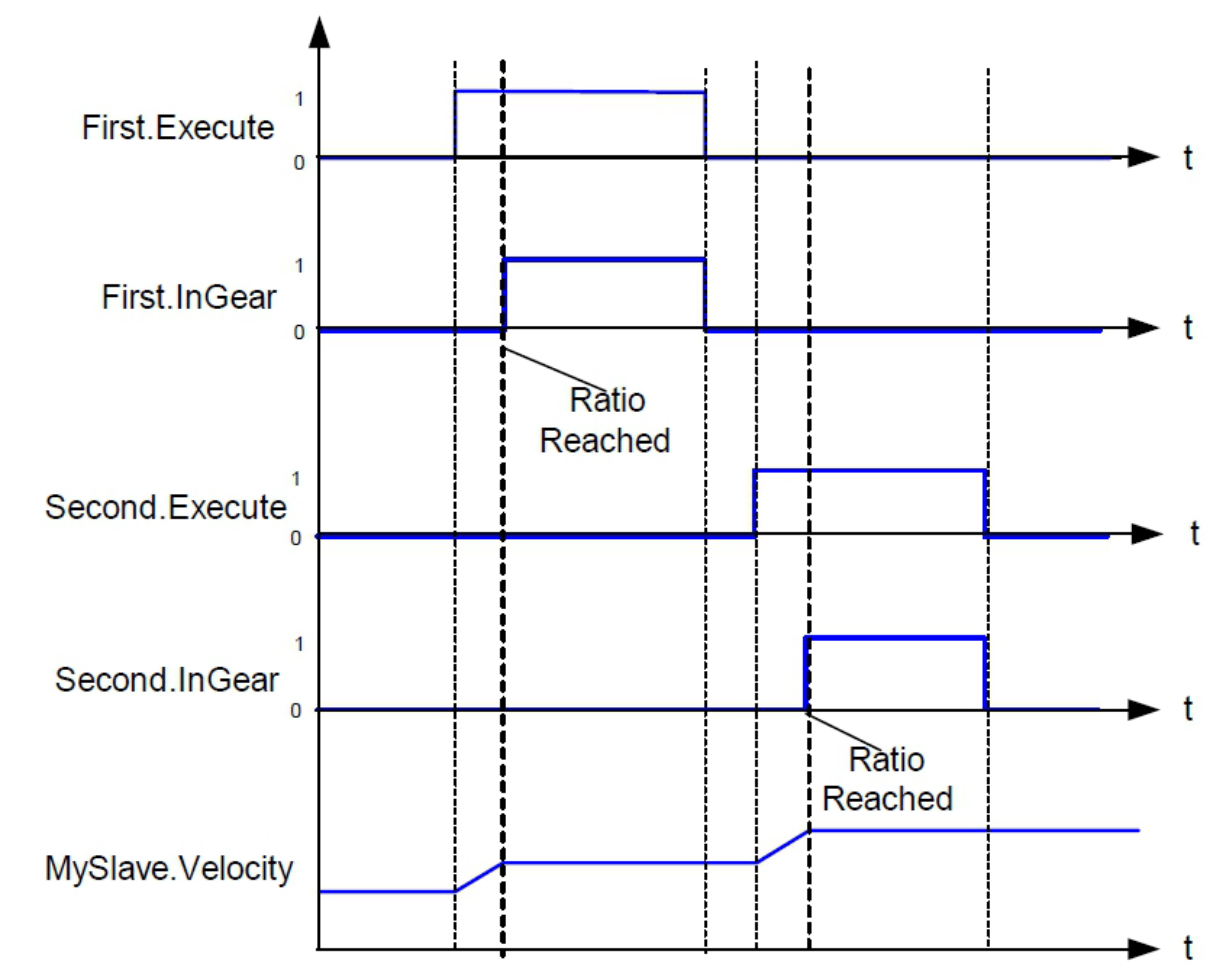


Fig. 350: Gear timing diagram

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

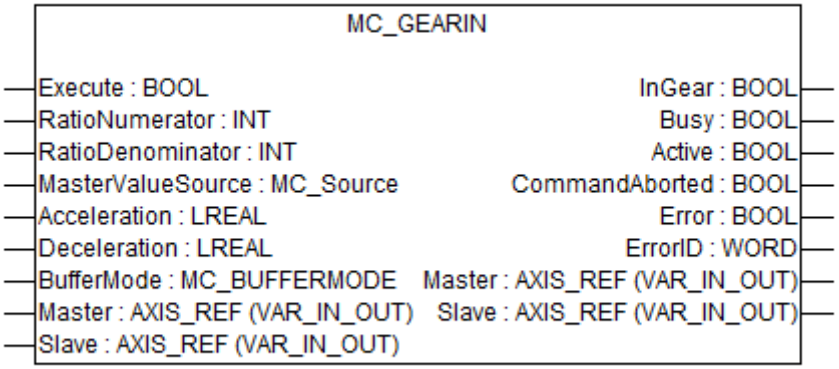


Fig. 351: Function block MC_GearIn

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

RatioNumerator
Data type: INT
Gear Ratio Numerator.

RatioDenominator
Data type: INT
Gear Ratio Denominator.

MasterValueSource
Data type: MC_SOURCE
Defines the source for synchronization:
mcSetValue - Synchronization on master set value.
mcActualValue - Synchronization on master actual value.

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Master Data type: AXIS_REF
Reference to the master axis.

Slave Data type: AXIS_REF
Reference to the slave axis.

Output description

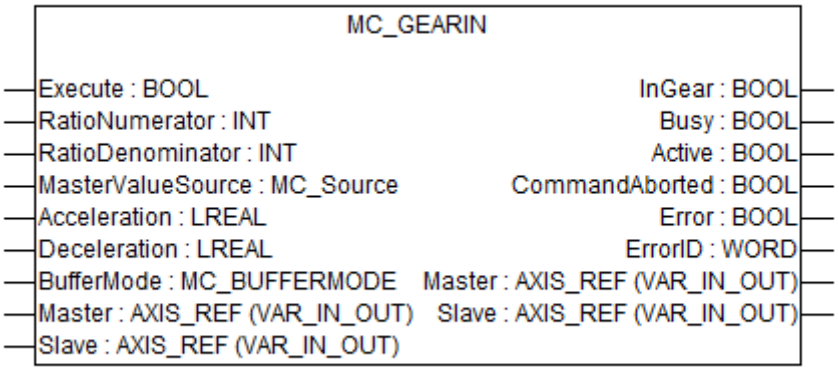


Fig. 352: Function block MC_GearIn

InGear Data type: BOOL
Commanded gearing completed.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GearInPos

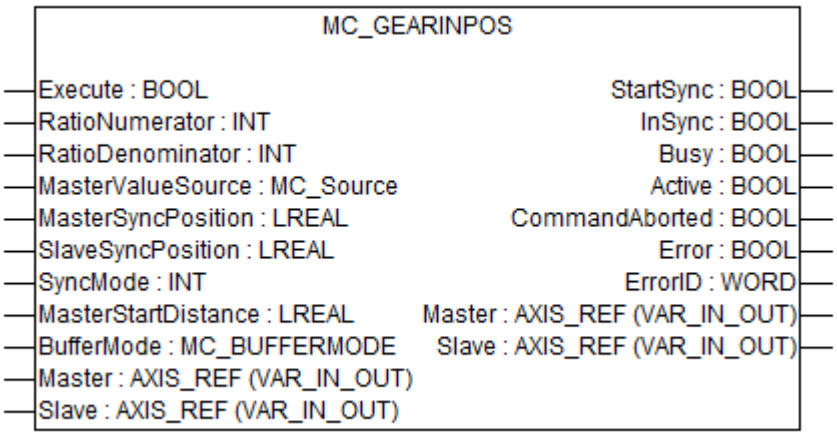


Fig. 353: Function block MC_GearInPos

This function block commands a gear ratio between the position of the slave and master axes from the synchronization point onwards.

- *If MasterStartDistance is implemented, any previous motion is continued until master crosses "MasterSyncPosition – MasterStartDistance" in the correct direction (according to the sign of MasterStartDistance). At that point in time the output StartSync is set. When a "Stop" command is executed on the "Slave" axis before the synchronization has happened, it inhibits the synchronization and the Function Bock issues "CommandAborted".*
- *If the MasterStartDistance is not specified, the system itself could calculate the set point for StartSync based on the other relevant inputs.*

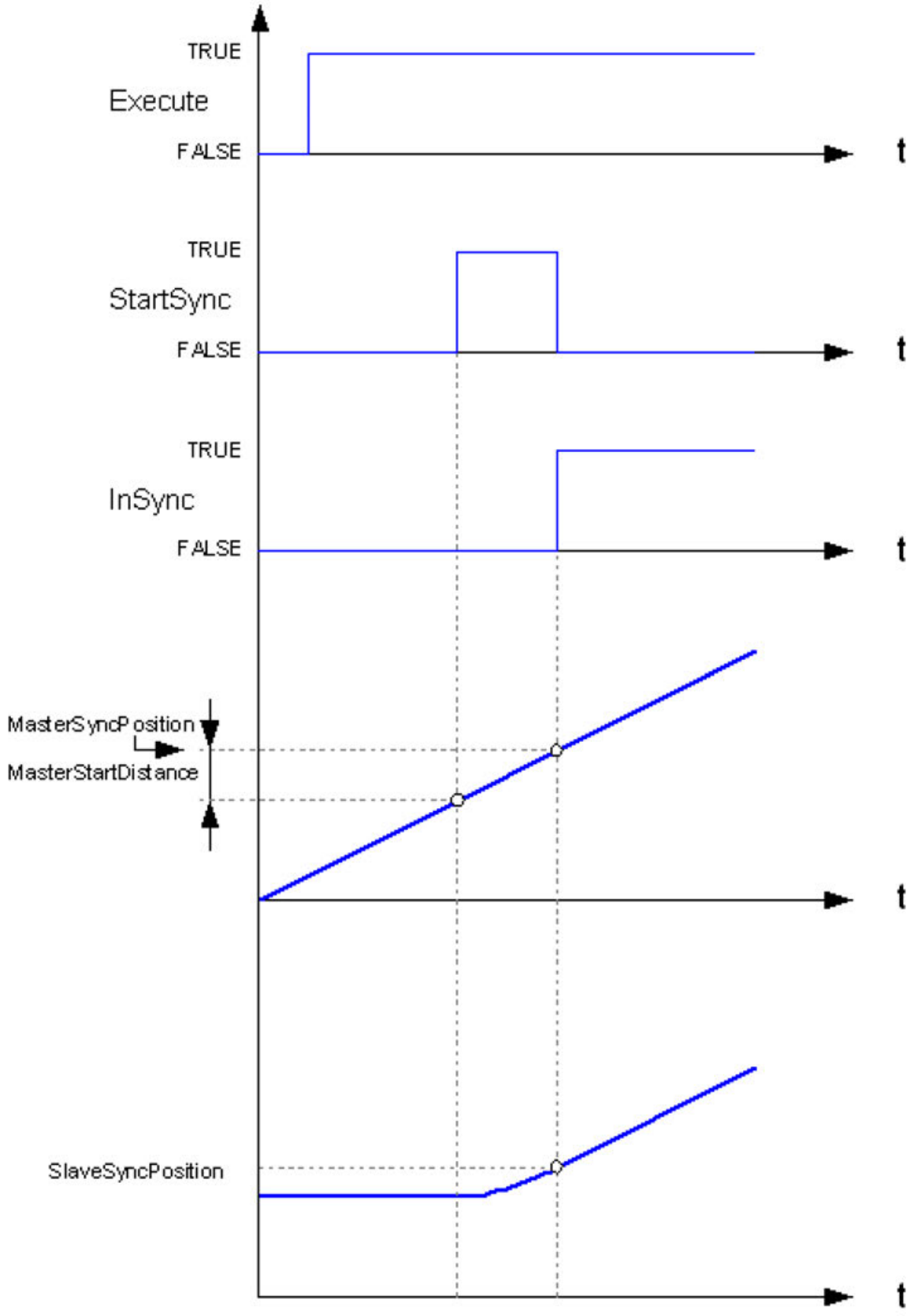
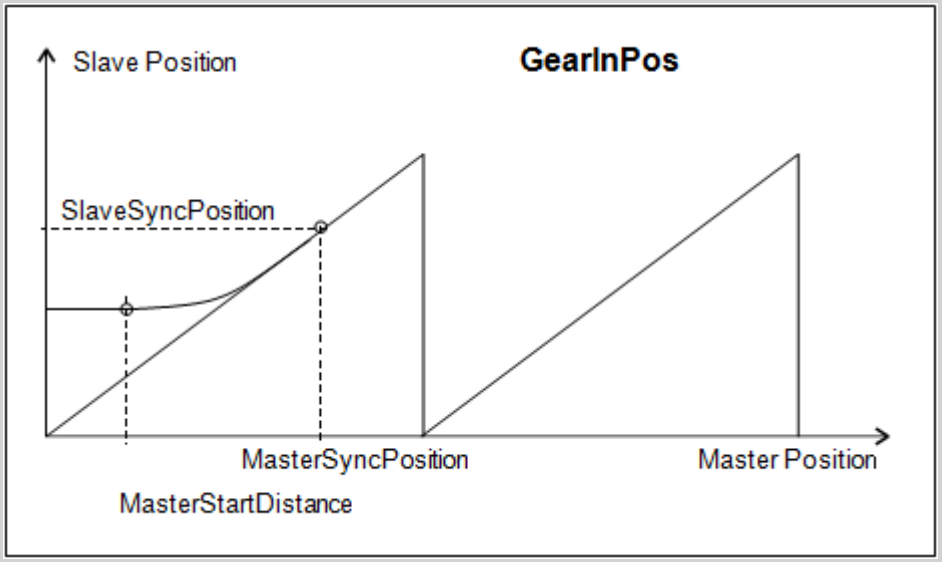
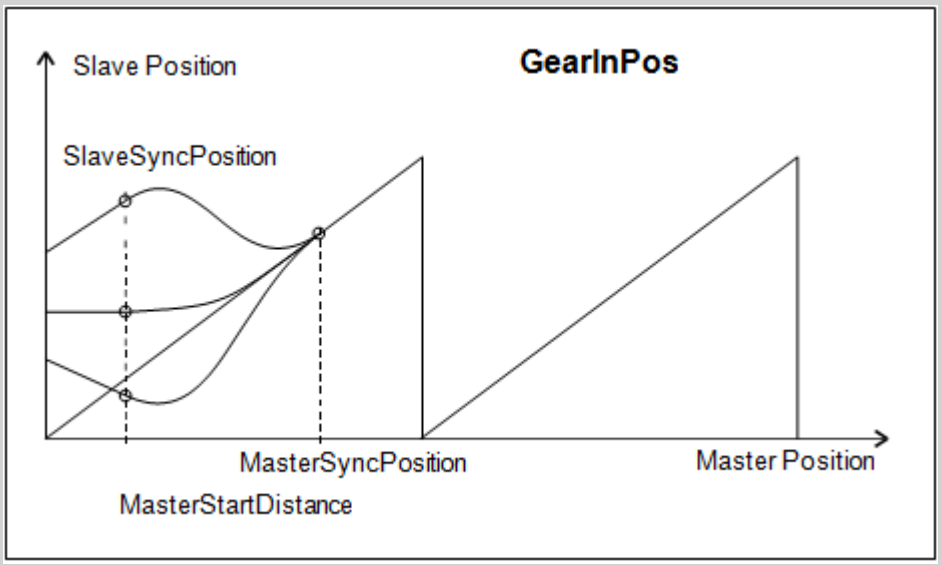
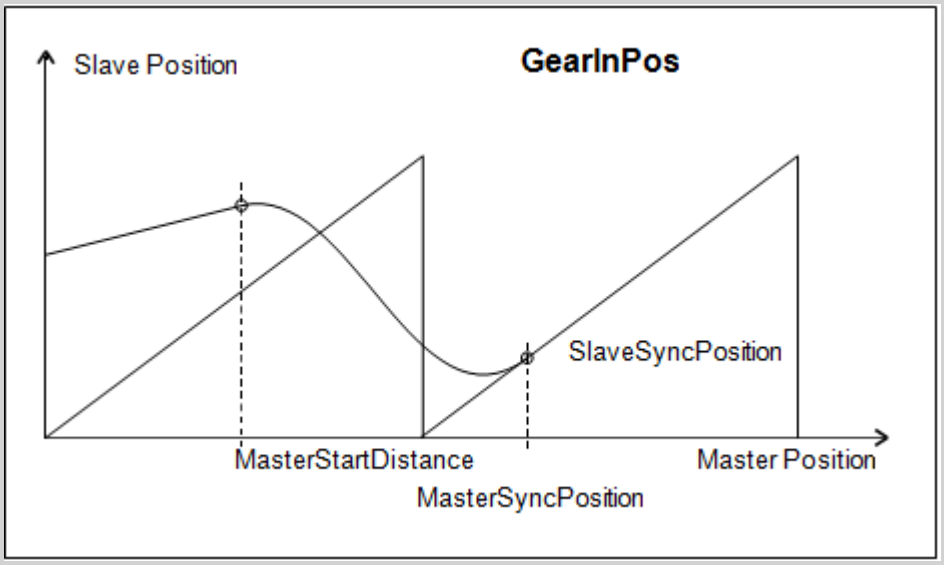


Fig. 354: Timing Diagram of MC_GearInPos

Different examples of MC_GearInPos







Input description

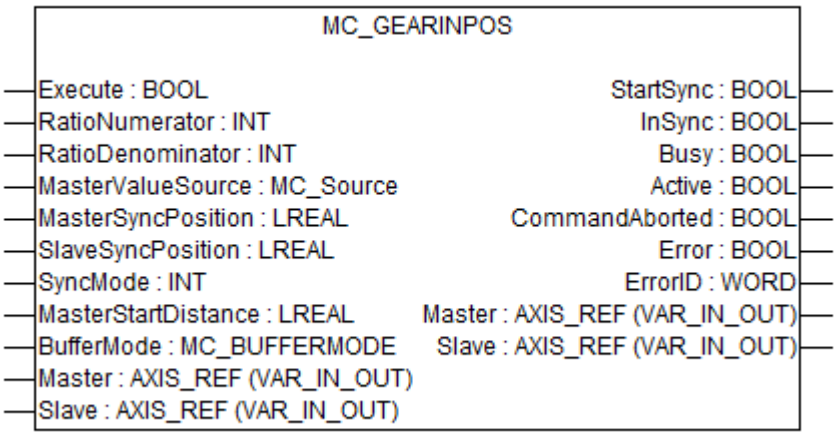


Fig. 355: Function block MC_GearInPos

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

RatioNumerator

Data type: INT

Gear Ratio Numerator.

RatioDenominator

Data type: INT

Gear Ratio Denominator.

MasterValueSource

Data type: MC_SOURCE

Defines the source for synchronization:

mcSetValue - Synchronization on master set value.

mcActualValue - Synchronization on master actual value.

MasterSyncPosition

Data type: LREAL

The position of the master in the path where the group is insync with the master. (If the ‘MasterSyncPosition’ does not exist, at the first point of the path the master and slave are synchronized).

> *The inputs acceleration, deceleration and jerk are not added here.*

SlaveSyncPosition Data type: LREAL

Slave Position at which the axes are running in sync.

SyncMode Data type: INT

This function block does not support different modes. Synchronization direction is determined by the sign of MasterStartDistance.

MasterStartDistance Data type: LREAL

The master distance for the slave to start to synchronize to the master.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Master Data type: AXIS_REF

Reference to the master axis.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output description

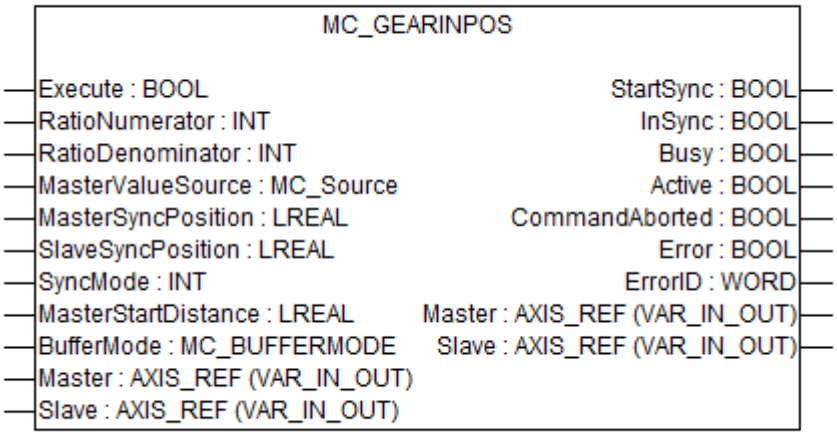


Fig. 356: Function block MC_GearInPos

StartSync Data type: BOOL

Commanded gearing starts.

InSync Data type: BOOL

Commanded gearing completed.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GearOut

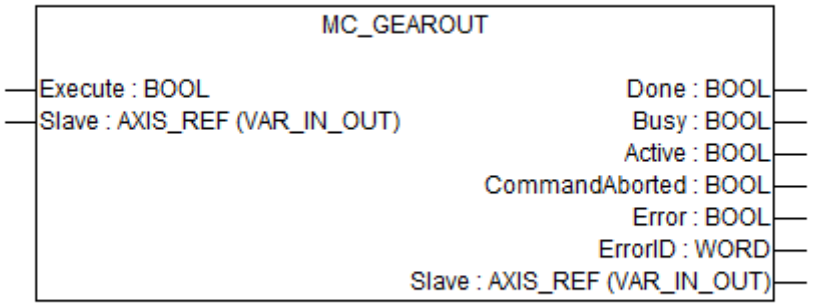


Fig. 357: Function block MC_GearOut

This function block disengages the Slave axis from the Master axis.

- *It is assumed that this command is followed by another command, for instance MC_Stop, MC_GearIn, or any other command. If there is no new command, the default condition should be: maintain last velocity.*
- *After issuing the function block there is no function block active on the slave axis till the next function block is issued (what can result in problems because no motion command is controlling the axis). Alternatively, one can read the actual velocity via MC_ReadActualVelocity and issue MC_MoveVelocity on the slave axis with the actual velocity as input. The function block is here because of compatibility reasons.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

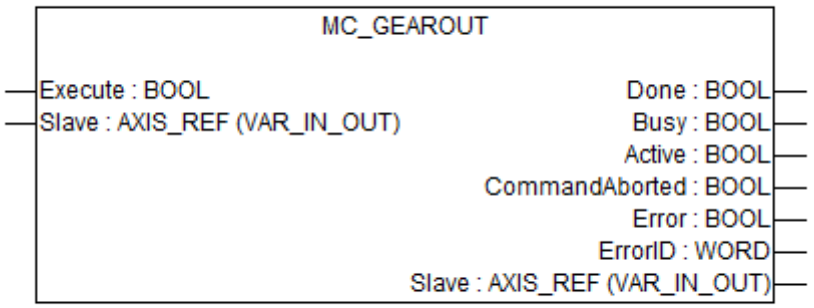


Fig. 358: Function block MC_GearOut

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output description

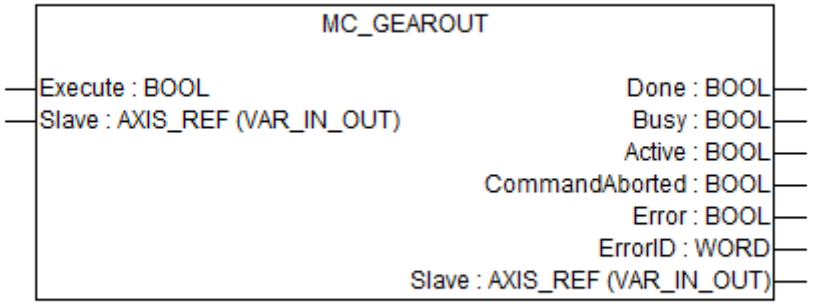


Fig. 359: Function block MC_GearOut

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_PhasingAbsolute

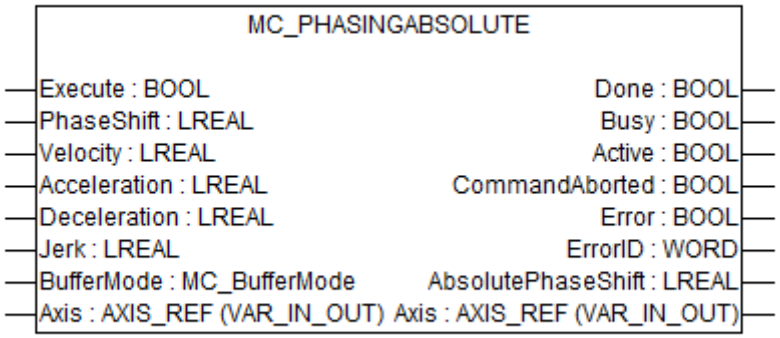
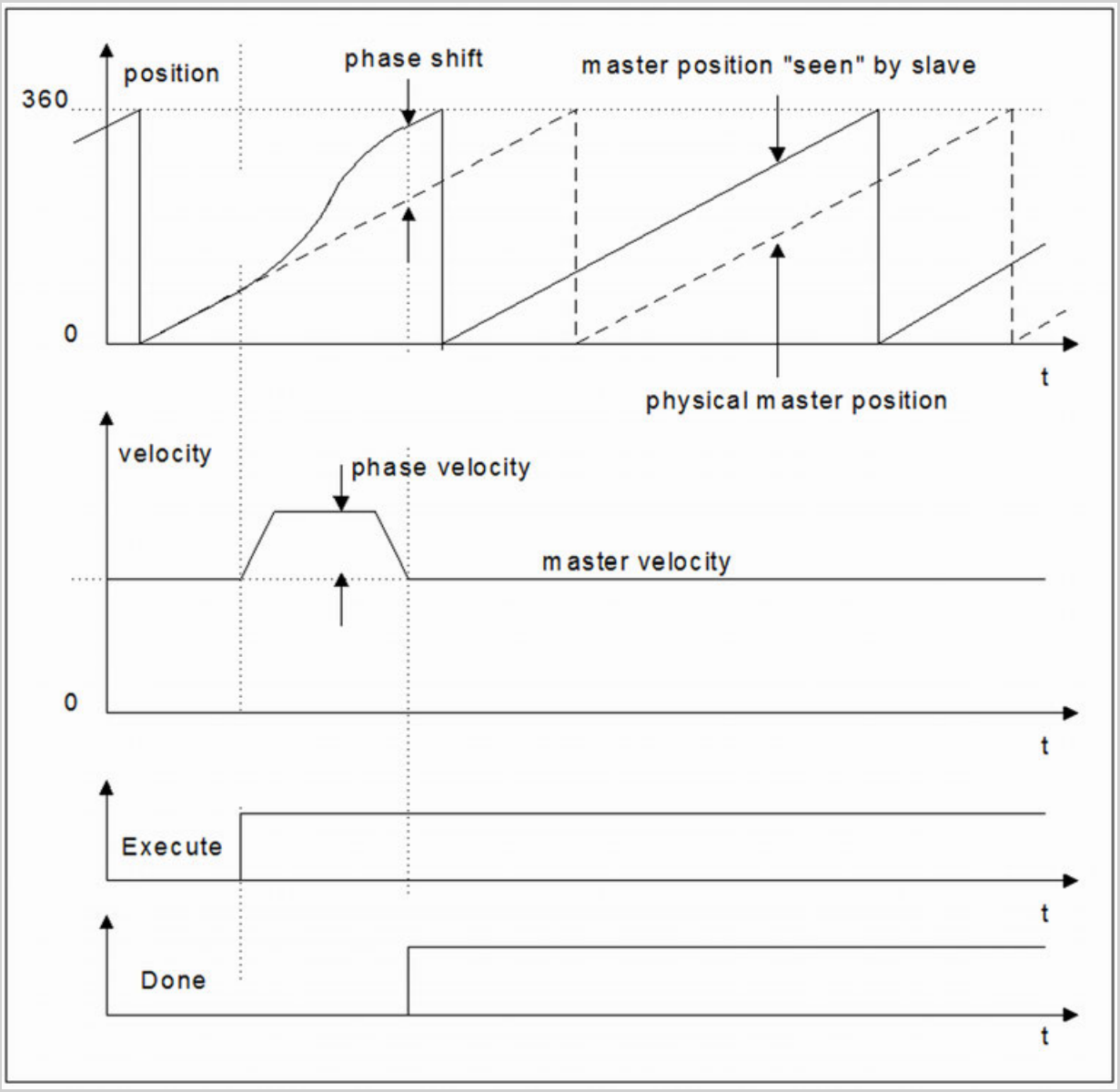


Fig. 360: Function block MC_PhasingAbsolute

This function block creates an absolute phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment, and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains until another "Phasing" command changes it again.

Phase, Velocity, Acceleration, Deceleration and Jerk of a phase shift are controlled by the function block.

Timing example of MC_Phasing



The following graph shows the effect of "Phasing".

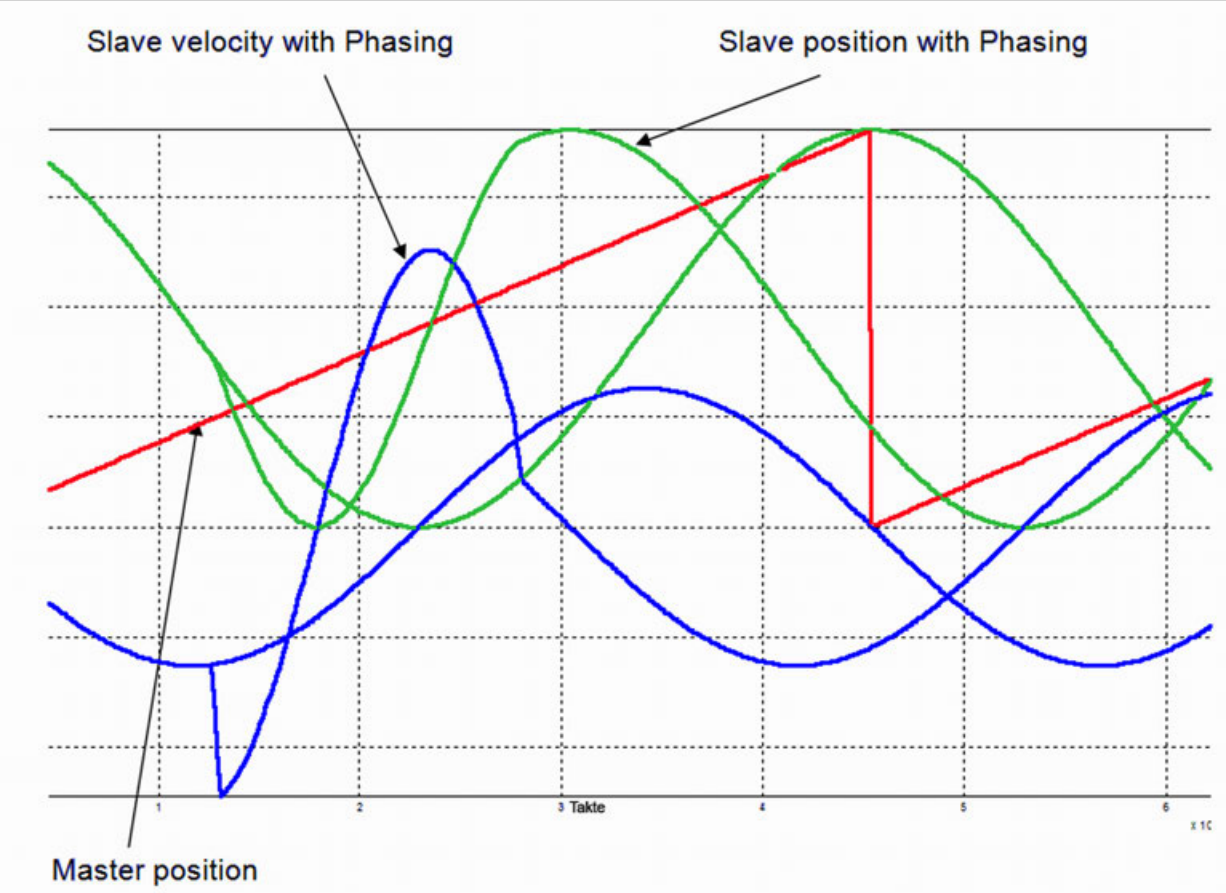


Fig. 361: Example of MC_Phasing

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

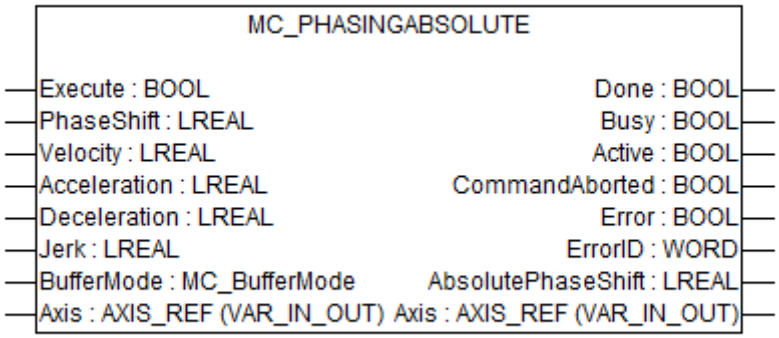


Fig. 362: Function block MC_PhasingAbsolute

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute Data type: BOOL
Starts the function block at rising edge.

PhaseShift Data type: LREAL, unit: u
Absolut phase difference in master.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

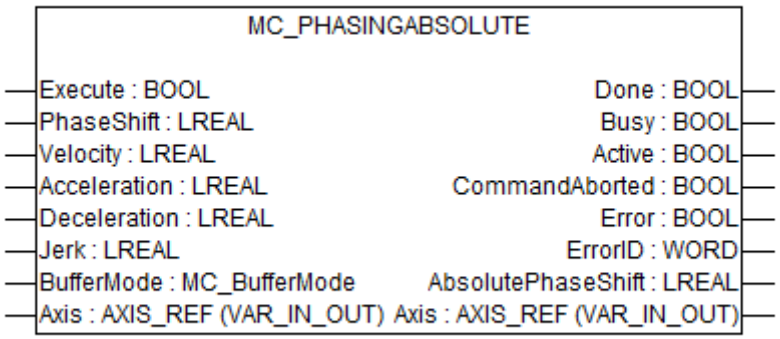


Fig. 363: Function block MC_PhasingAbsolute

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

AbsolutePhaseShift Data type: LREAL
Displays the covered distance since it was started continuously.

MC_PhasingRelative

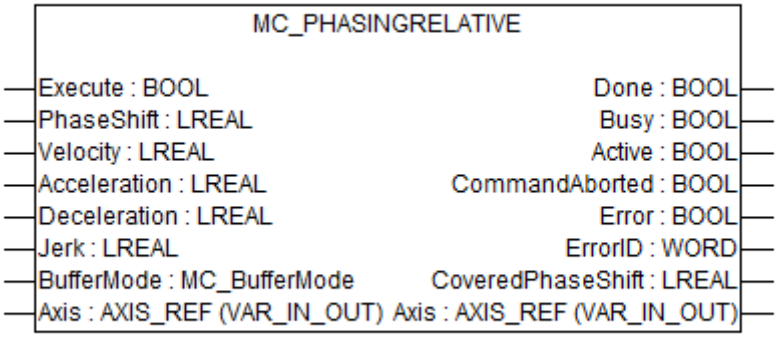
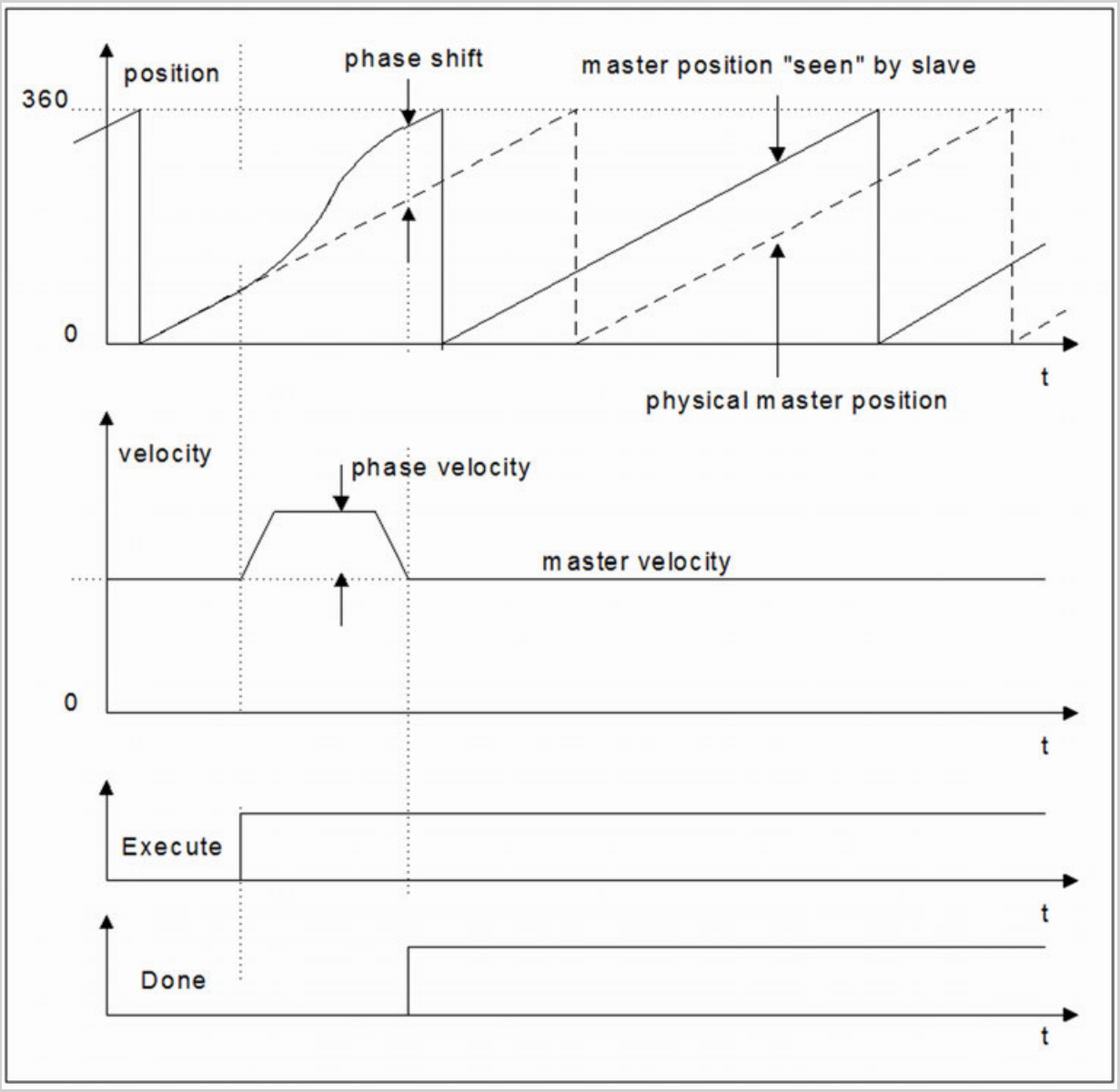


Fig. 364: Function block MC_PhasingRelative

This function block creates a relative phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment, and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains until another "Phasing" command changes it again.

> *Phase, Velocity, Acceleration, Deceleration and Jerk of a phase shift are controlled by the function block.*

Timing example of MC_Phasing



The following graph shows the effect of "Phasing".

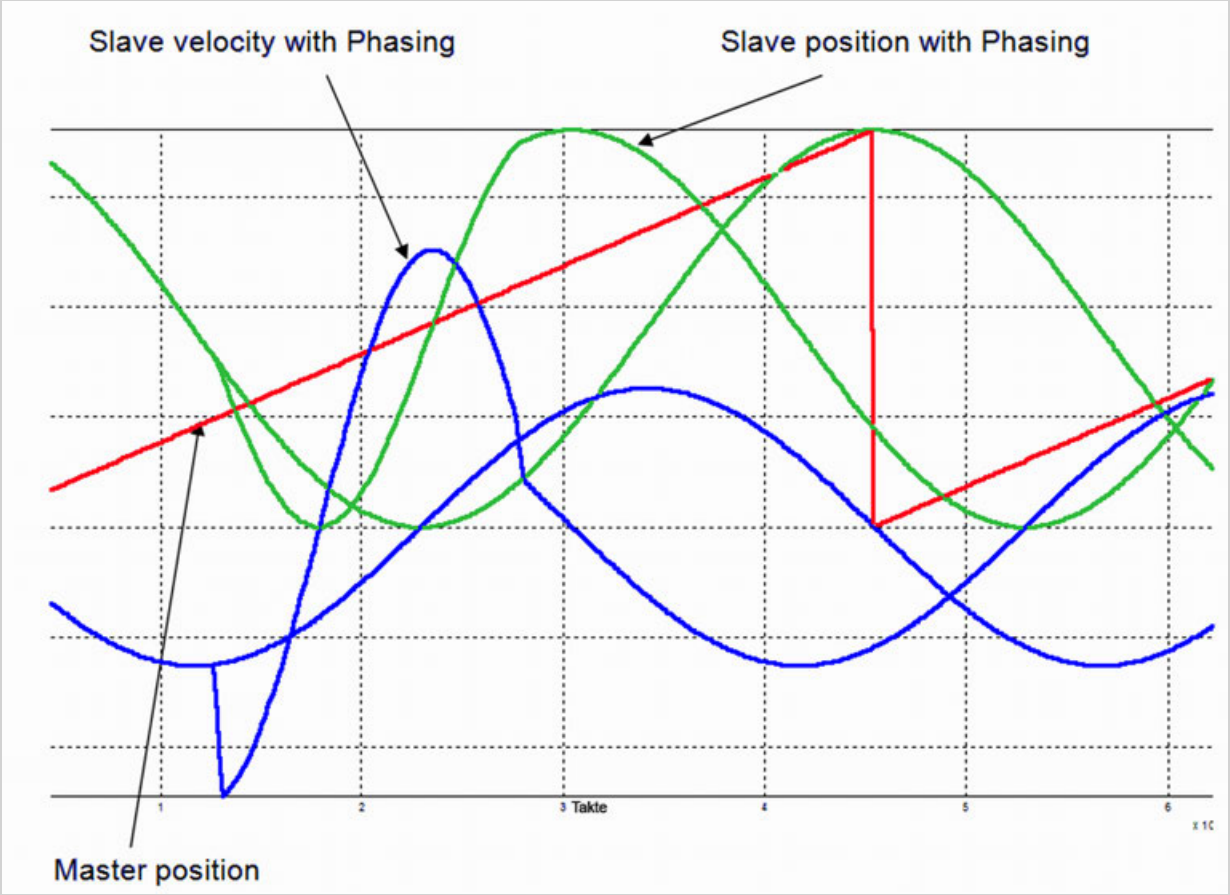


Fig. 365: Example of MC_Phasing

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

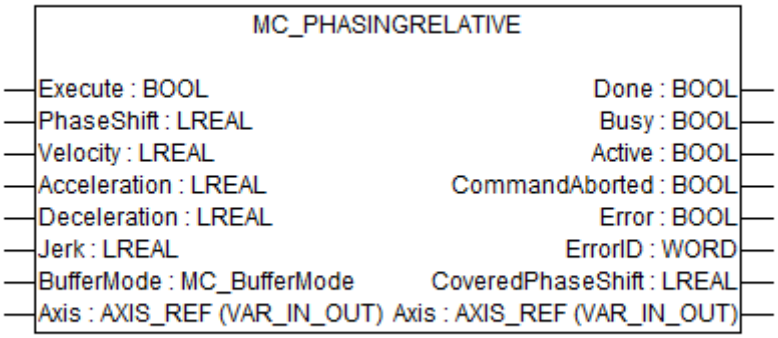


Fig. 366: Function block MC_PhasingRelative

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute Data type: BOOL
Starts the function block at rising edge.

PhaseShift Data type: LREAL, unit: u
Absolut phase difference in master.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

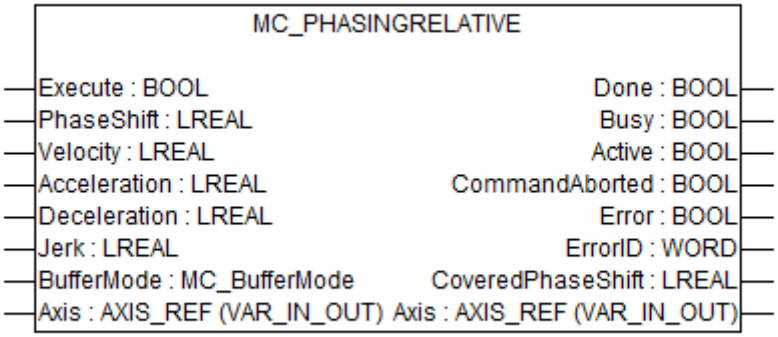


Fig. 367: Function block MC_PhasingRelative

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

CoveredPhaseShift Data type: LREAL
Displays the covered distance since it was started continuously.

MC_CombineAxes

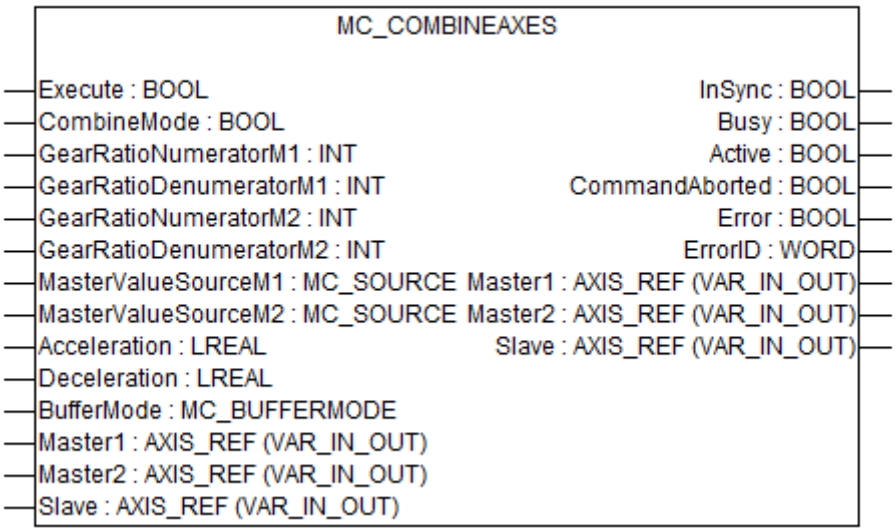


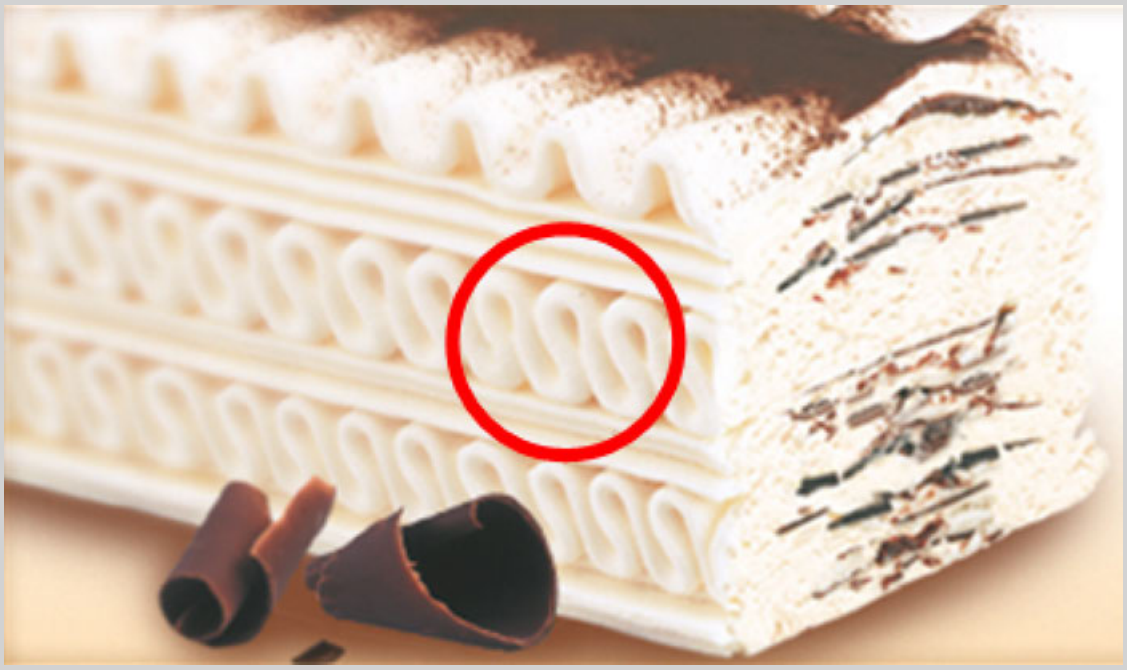
Fig. 368: Function block MC_CombineAxes

This function block combines the motion of 2 axes into a third axis with selectable combination method. Basically, it is a calculation of a new position setpoint based on the 2 position setpoints of the input axes. This function block is reflected in the single axis state diagram like a Synchronized Motion type. As application example one can work with a separate profile synchronized to an object on a moving belt, or a rotating knife with flexible covered distance to be cut.

To stop the motion, the function block has to be interrupted by another function block issuing a new command.

This block has to be called from the same task as CMC_MOTION_KERNEL_REAL.

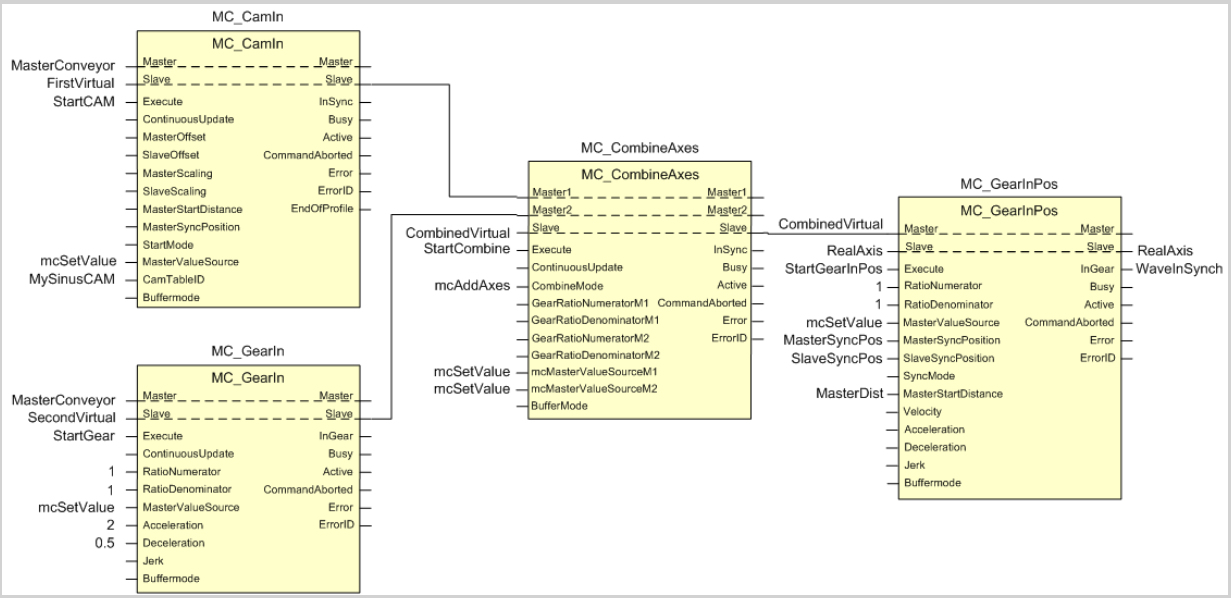
Example ice cream



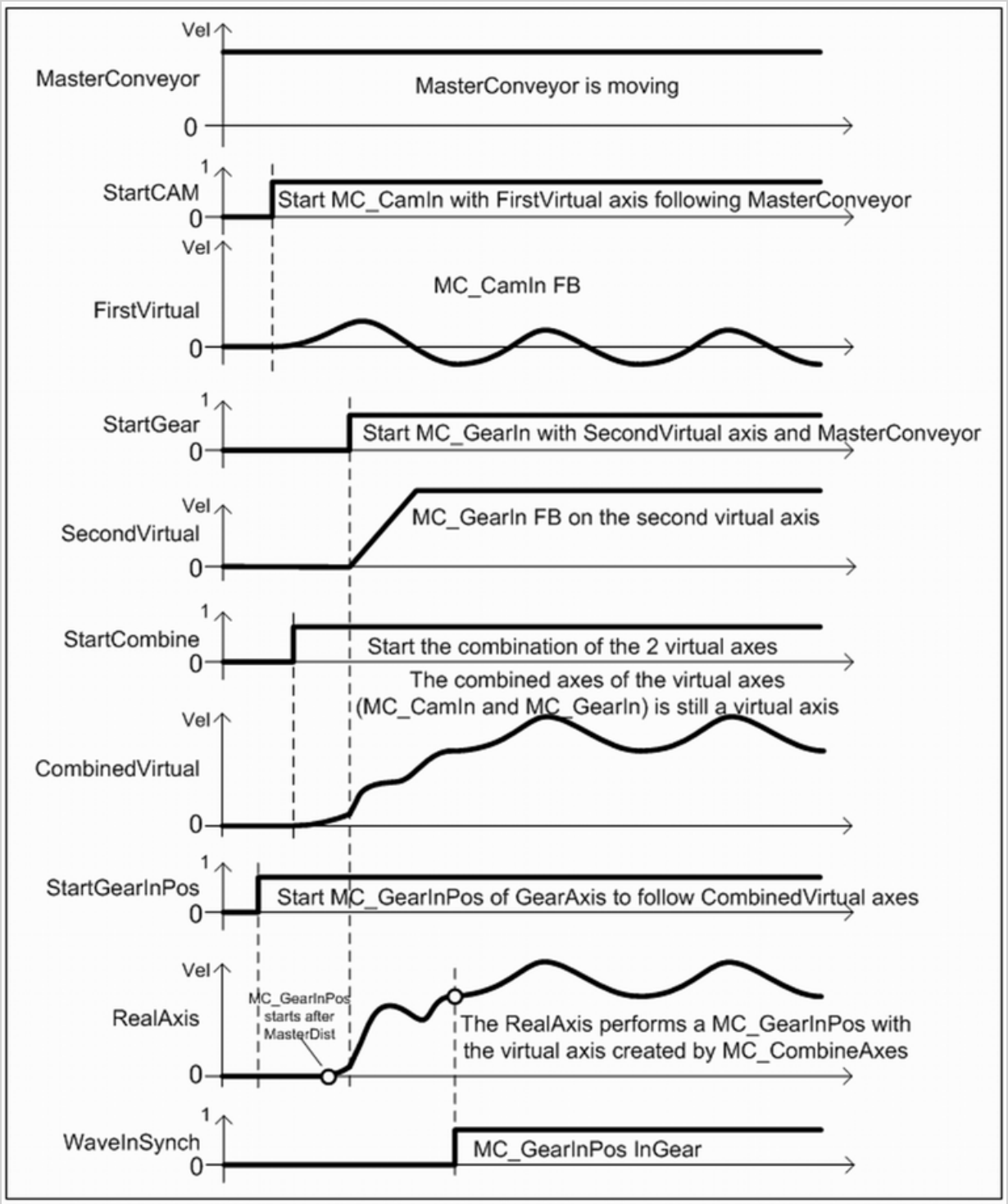
MC_CombineAxes can generate special synchronized movements that are not possible or complex to generate in other ways. In the following example, a CAM function block and the result of a Gear function block are both synchronized to a conveyor master, are added to generate a virtual master for a MC_GearInPos function of the final axis that will execute the movement. The particular application of this example could be a machine to deposit the icecream waving layers on top of the icecream base travelling through the freezer line in icecream factory. The dosing axis has to synchronize with a waving manner to the conveyor carrying the icecream base block. And it has to do this in a particular starting position and wave phase to achieve the expected result (therefore the GearInPos). With the CAM function block one can define different wave patterns easily (like the one longer in the top of icecream).

Another case application can be chocolate bars with decoration (individual bars in mouldings). The dosificator makes the wave synchronized with conveyor and returns for the next.

Application example of MC_CombineAxes



The corresponding timing diagram for MC_CombineAxes example



See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

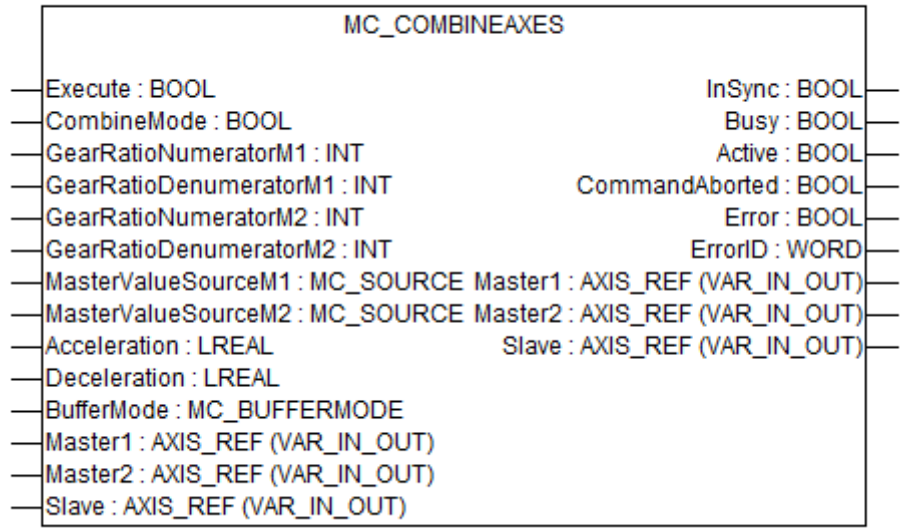


Fig. 369: Function block MC_CombineAxes

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

CombineMode
Data type: BOOL
Defines the type of combination applied to AxisOut:
FALSE: Addition of the 2 input axes positions
TRUE: Substraction of the 2 input axes positions.

GearRatioNumeratorM1
Data type: INT
Numerator for the gear factor for master axis 1 towards the slave.

GearRatioDenominatorM1
Data type: INT
Corresponding denominator for master axis 1.

GearRatioNumeratorM2
Data type: INT
Numerator for the gear factor for master axis 2 towards the slave.

GearRatioDenominatorM2
Data type: INT
Corresponding denominator for master axis 2.

MasterValueSourceM1
Data type: MC_SOURCE
Defines the source for synchronization for master axis 1:
mcSetValue: Synchronization on master set value.
mcActualValue. Synchronization on master actual value .

MasterValueSourceM2
Data type: MC_SOURCE
Defines the source for synchronization for master axis 2:
mcSetValue: Synchronization on master set value.
mcActualValue: Synchronization on master actual value .

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Master1
Data type: AXIS_REF
Reference to the first master axis.

Master2
Data type: AXIS_REF
Reference to the second master axis.

Slave
Data type: AXIS_REF
Reference to the resulting combined axis. Can be a virtual axis or linked directly to a real axis.

Output description

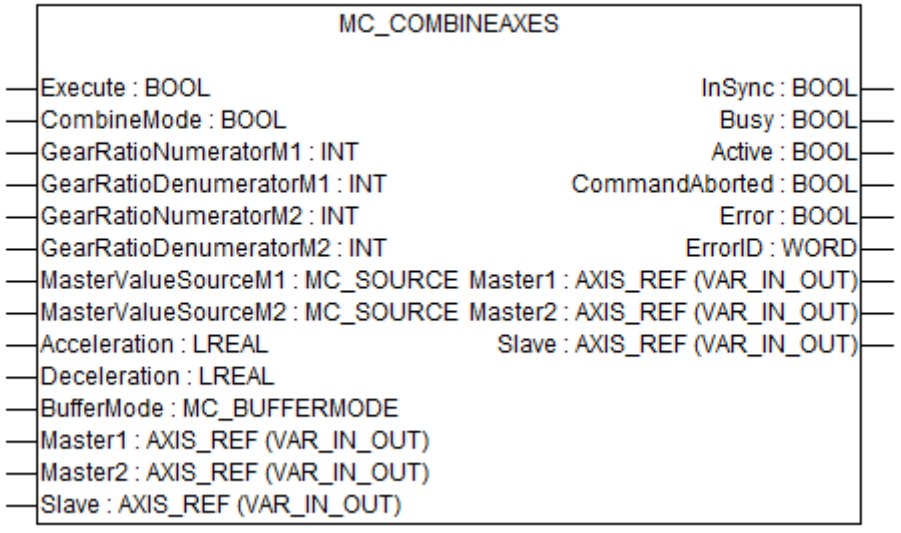


Fig. 370: Function block MC_CombineAxes

InSync Data type: BOOL
Is TRUE if the set value = the commanded value.

Busy Data type: BOOL
The function block is not finished and new output values are to be expected.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

1.5.9.6.3 Administrative function blocks

MC_CamTableSelect

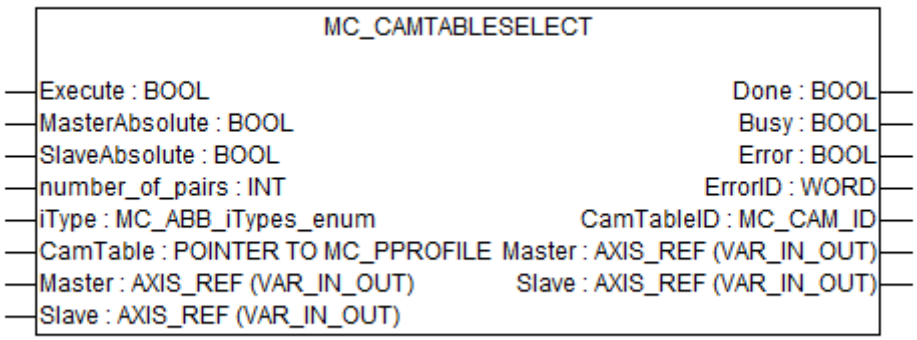


Fig. 371: Function block MC_CamTableSelect

This function block selects the CAM tables by setting the connections to the relevant tables.

- *A virtual axis can be used as master axis.*
- *MC_PPROFILE is an ABB specific data type.*
- ***CamTableSelect makes data available. This can include:***
  - *Starting point of a download of a profile.*
  - *Start to generate a CAM profile.*
  - *PC – based : No function. It is referenced by a pointer.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

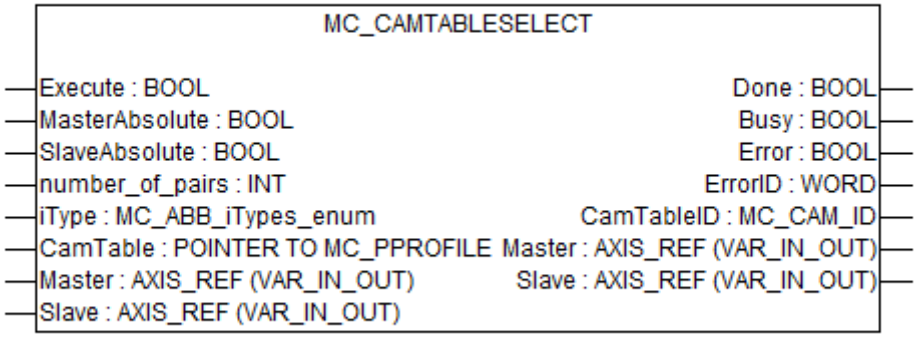


Fig. 372: Function block MC_CamTableSelect

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

MasterAbsolute Data type: BOOL

1=absolute

0 = relative coordinates

SlaveAbsolute Data type: BOOL

1=absolute

0 = relative coordinates

Number_of_pairs Data type: INT

Number of points used in CamTableTimeAcceleration array.

iTYPE Data type: MC_ABB_iTypes_ENUM

Interpolationtype, possible values are:

- MCA_SPLINE_COMPLETE
- MCA_SPLINE_NATURAL
- MCA_POLY5
- MCA_POLY3
- MCA_LINEAR

CamTable Data type: POINTER TO MC_PPROFILE

Reference to CAM description.

Master Data type: AXIS_REF

Reference to the master axis.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output description

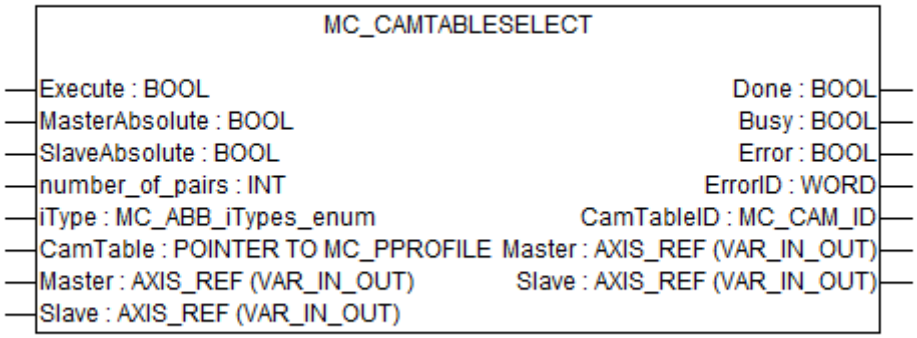


Fig. 373: Function block MC_CamTableSelect

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

CamTableID Data type: MC_CAM_ID

Identifier of CAM Table to be used in the MC_CamIn function block.

MC_Power

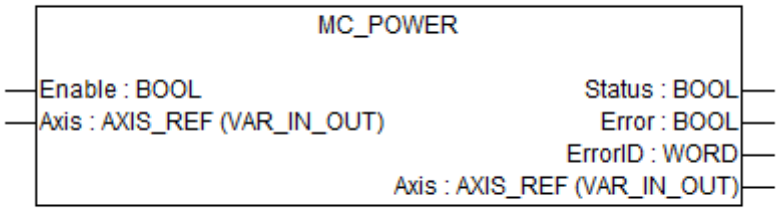


Fig. 374: Function block MC_Power

Table 190: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block controls the power stage (on or off). If this function block is called with Enable=TRUE while being in Disabled, this either leads to StandStill if there is no error inside the axis or to ErrorStop if an error exists.

- *If the function block MC_Power is called with the Enable true while being in Disabled, this either leads to Standstill if there is no error in the axis, or to ErrorStop if an Error exists.*
- *It is possible to set an error variable when the Command is TRUE for a while and the Status remains FALSE with a Timer function block and an AND Function (with inverted Status input). It indicates that there is a hardware problem with the power stage.*
- *If power fails (also during operation) it will generate a transition to the state ErrorStop.*
- *When MC_Power is called with Enable FALSE the axis goes to state Disabled for every state including ErrorStop.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Using this function block with FM562 Input variable AXIS of MC_Power must be the same with input variable AXIS of PTO_FM562_ACCESS. Therefore, the specific axis information contained in AXIS_REF structure can be transferred from PLCopen Library to PTO Library and then to FM562 IO mapping.

Input description

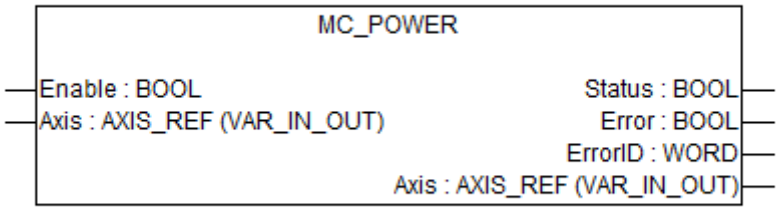


Fig. 375: Function block MC_Power

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

As long as Enable = TRUE, power is on.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

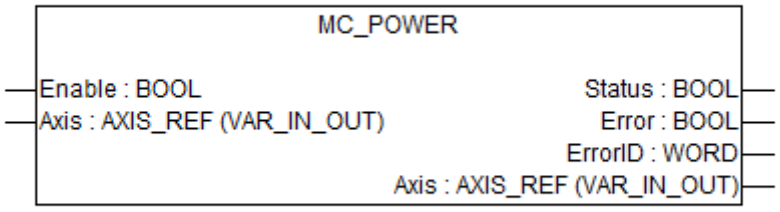


Fig. 376: Function block MC_Power

Status Data type: BOOL

Effective state of the power stage.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_ReadStatus

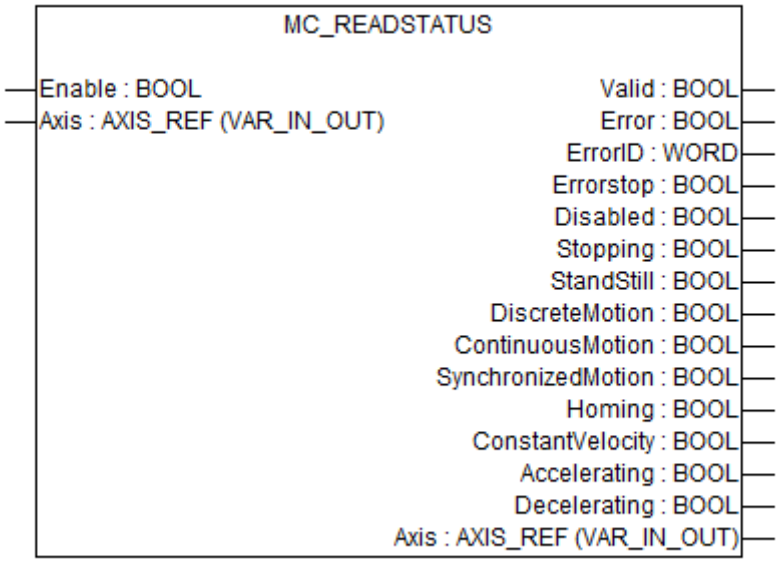


Fig. 377: MC_ReadStatus

This function block returns in detail the status of the axis with respect to the motion currently in progress.

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Using this function block with FM562

Output SynchronizedMotion: Not implemented

Output Homing: Not implemented

Output ConstantVelocity: Not implemented

Output Accelerating: Not implemented

Output Decelerating: Not implemented

Input description

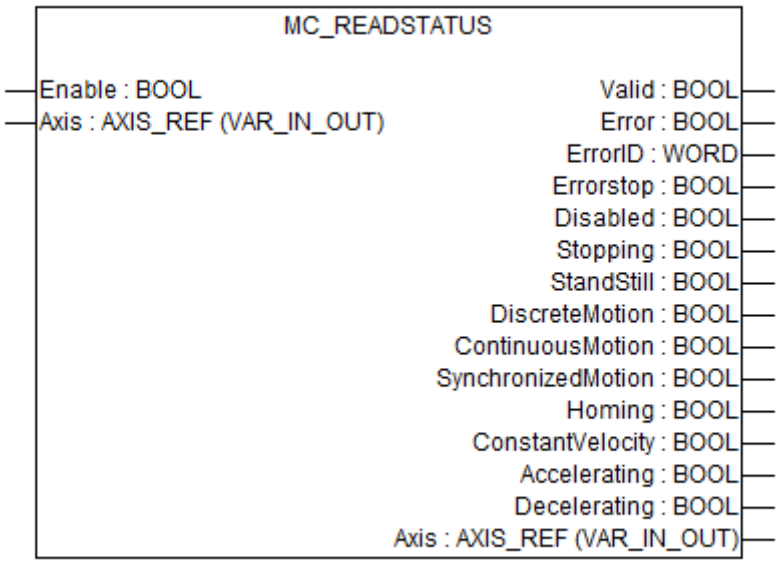


Fig. 378: Function block MC_ReadStatus

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the value of the parameter continuously while enabled.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

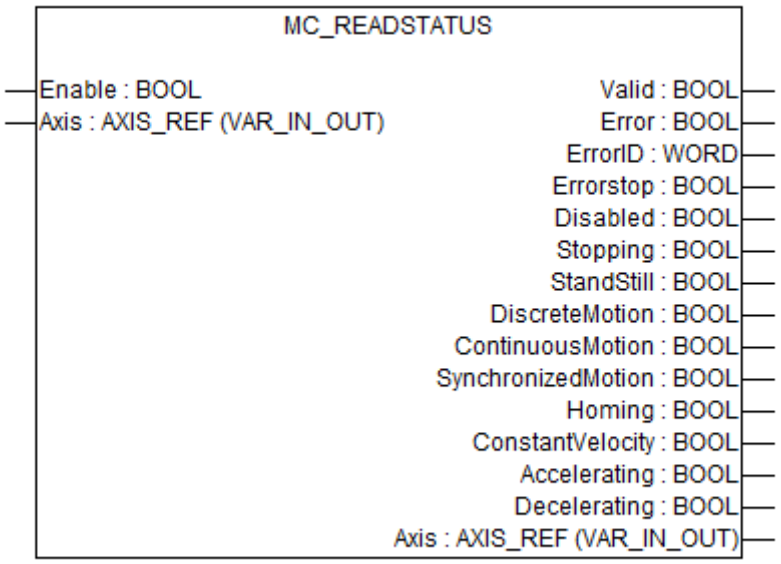


Fig. 379: Function block MC_ReadStatus

Valid Data type: BOOL

Parameter available.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

ErrorStop Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

Disabled Data type: BOOL

Is SET if the axis is in the Disabled state.

Stopping Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

StandStill Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

DiscreteMotion Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

ContinuousMotion Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

SynchronizedMotion Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

Homing Data type: BOOL

See single axis state diagram. *Chapter 1.5.9.3.2 “The single axis state diagram” on page 2589*

ConstantVelocity Data type: BOOL

Motor moves with constant velocity.

Accelerating Data type: BOOL

Increasing energy of the motor.

Decelerating Data type: BOOL

Decreasing energy of the motor.

MC_ReadAxisError

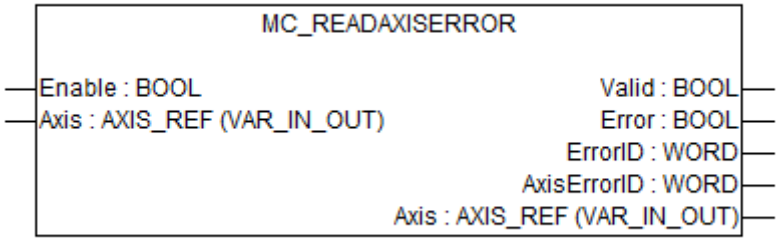


Fig. 380: Function block MC_ReadAxisError

This function block describes general axis errors not relating to the function blocks.

> *This function block is the equivalent to read the AxisErrorID parameter using MC_ReadParameter.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

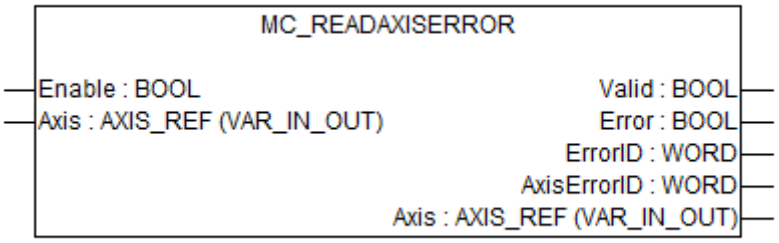


Fig. 381: Function block MC_ReadAxisError

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

As long as Enable = TRUE, power is on.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

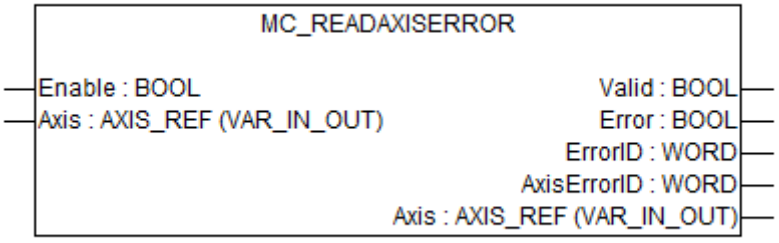


Fig. 382: Function block MC_ReadAxisError

Valid Data type: BOOL

Parameter available.

Error Data type: BOOL

Error flag.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

AxisErrorID Data type: WORD

The value of the axis error. These values are drive specific and only valid when it is possible to read an error from the drive (e.g. by fieldbus).

In Drive-based motion, the AxisErrorID shows the value which is configured for the process image.

In PLC-based motion, the AxisErrorID shows the CMC_MOTION_KERNEL… ErrorID.

MC_Reset

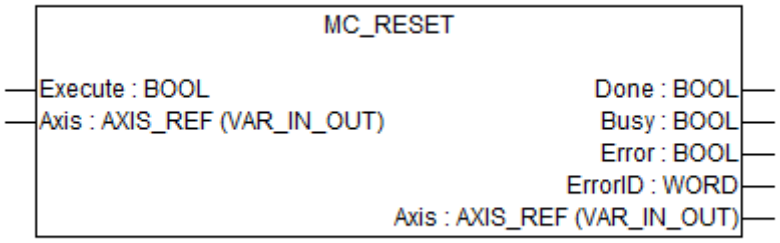


Fig. 383: Function block MC_Reset

Table 191: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block makes the transition from the state ErrorStop to StandStill by resetting all internal axis-related errors. It does not affect the output of the function block instances.

In addition, a reset message is sent to the drive (e.g. output DRIVE_RESET_FAULT at CMC_MOTION_KERNEL... or by ACCESS function block).

If used with FM562, MC_Reset also gives a command to reset position data on FM562 IO mapping and offset position set by MC_SetPosition.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

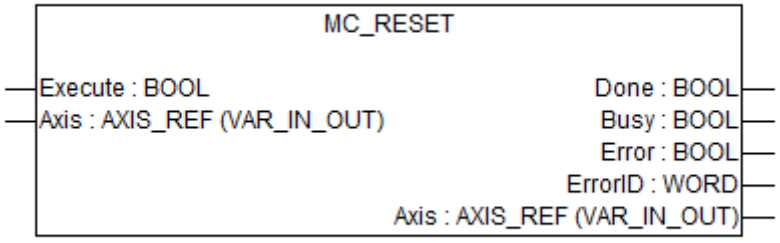


Fig. 384: Function block MC_Reset

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

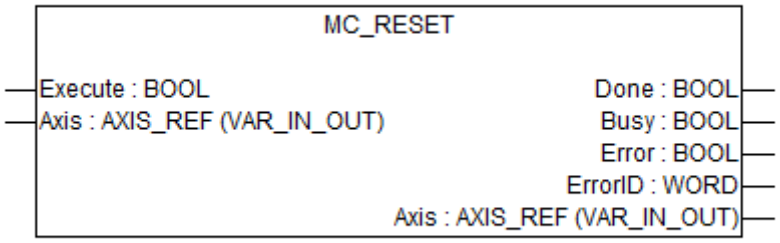


Fig. 385: Function block MC_Reset

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Error flag.

ErrorID Data type: WORD

Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_ReadParameter

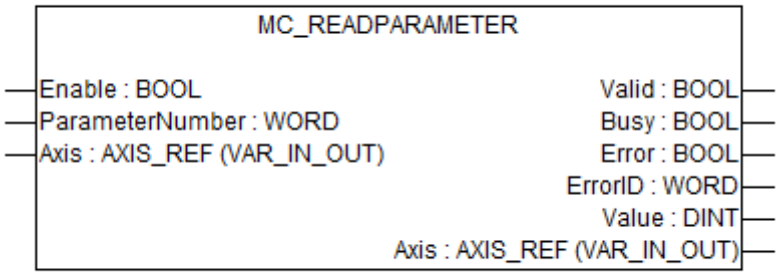


Fig. 386: Function block MC_ReadParameter

When a drive based axis implementation is used, the function block returns the value of a drive specific parameter or a PLCopen parameter. When the Central Motion Control axis implementation is used, a parameter according to the list PLCopen parameter is returned. ↳ *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

> *All available parameters are listed in this table: ↳ Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

> *See also:*
> – *MC_WriteParameter to write parameters ↳ Chapter 1.5.9.6.3.8 “MC_WriteParameter” on page 2848*
> – *MC_ReadBoolParameter to read Boolean parameter ↳ Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*
> – *MC_WriteBoolParameter to write Boolean parameter ↳ Chapter 1.5.9.6.3.9 “MC_WriteBoolParameter” on page 2850*

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

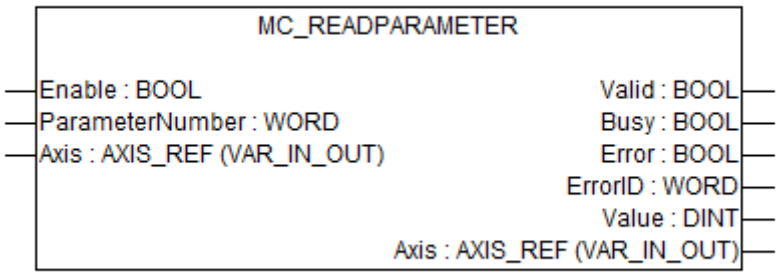


Fig. 387: Function block MC_ReadParameter

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable

Data type: BOOL

Get the value of the parameter continuously while enabled.

Parameter-Number

Data type: INT

Number of the parameter. One can also use as symbolic parameter names which are declared as VAR CONST.

Axis

Data type: AXIS_REF

Reference to the axis.

Output description

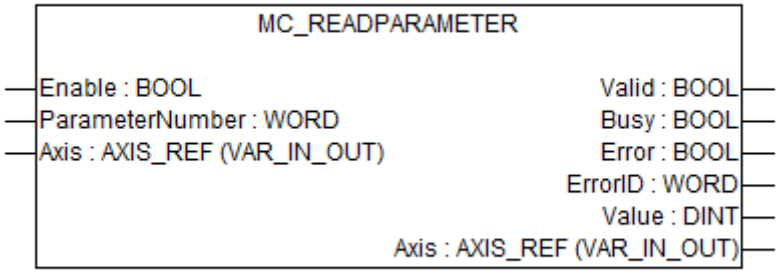


Fig. 388: Function block MC_ReadParameter

Valid

Data type: BOOL

Parameter available.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Value Data type: DINT

Value of the specified parameter in the datatype.

MC_ReadBoolParameter

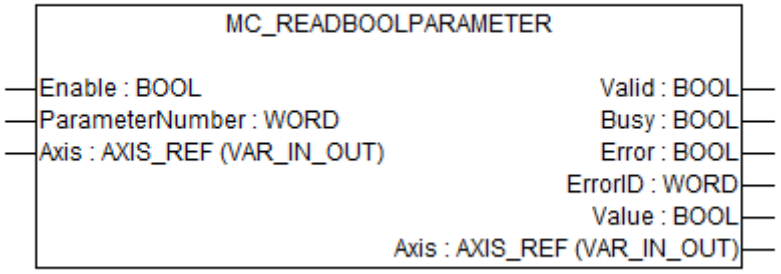


Fig. 389: Function block MC_ReadBoolParameter

When a drive based axis implementation is used, the function block returns the value of a drive specific BOOL parameter or a PLCopen BOOL parameter. When the Central Motion Control axis implementation is used, a BOOL parameter according to the list PLCopen parameter is returned. ↳ *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

All available parameters are listed in this table: ↳ Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595

See also:

- *MC_ReadParameter to read parameters ↳ Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*
- *MC_WriteParameter to write parameters ↳ Chapter 1.5.9.6.3.8 “MC_WriteParameter” on page 2848*
- *MC_WriteBoolParameter to write Boolean parameters ↳ Chapter 1.5.9.6.3.9 “MC_WriteBoolParameter” on page 2850*

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

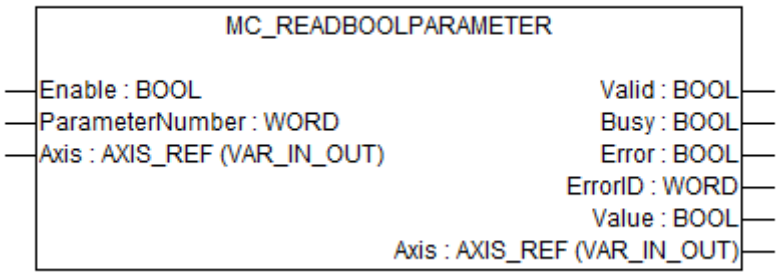


Fig. 390: Function block MC_ReadBoolParameter

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Axis
Data type: AXIS_REF
Reference to the axis.

Enable
Data type: BOOL
Get the value of the parameter continuously while enabled.

Parameter-Number
Data type: INT
Number of the parameter. One can also use as symbolic parameter names which are declared as VAR CONST.

Output description

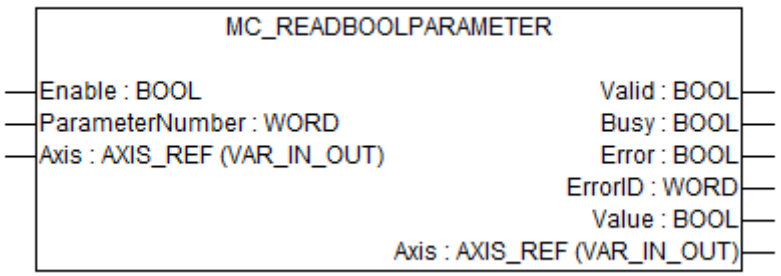


Fig. 391: Function block MC_ReadBoolParameter

Valid Data type: BOOL

Parameter available.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Value Data type: DINT

Value of the specified parameter in the datatype.

MC_WriteParameter

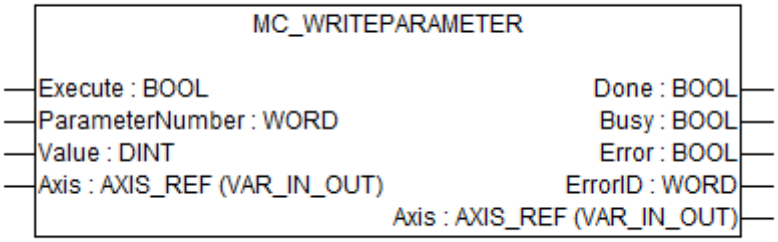


Fig. 392: Function block MC_WriteParameter

When a drive based axis implementation is used, the function block writes the value of a drive specific parameter or a PLCopen parameter. When the Central Motion Control axis implementation is used, a parameter according to the list PLCopen parameter is written. *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

All available parameters are listed in this table: *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

See also:

- *MC_ReadParameter to read parameters* *Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*
- *MC_ReadBoolParameter to read Boolean parameters* *Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*
- *MC_WriteBoolParameter to write Boolean parameters* *Chapter 1.5.9.6.3.9 “MC_WriteBoolParameter” on page 2850*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

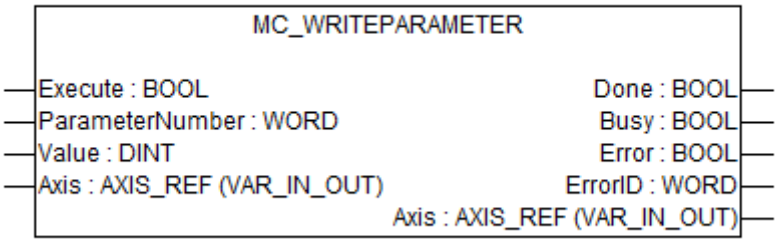


Fig. 393: Function block MC_WriteParameter

Execute
Data type: BOOL
Starts the function block at rising edge.

Parameter-Number
Data type: INT
Number of the parameter (correspondence between number and parameter is to be specified later).

Value
Data type: LREAL
New value of the specified parameter.

Axis
Data type: AXIS_REF
Reference to the axis.

Output description

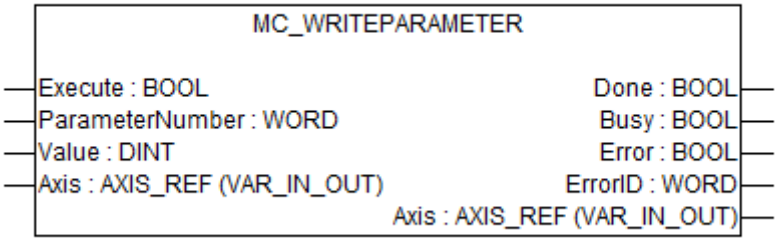


Fig. 394: Function block MC_WriteParameter

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_WriteBoolParameter

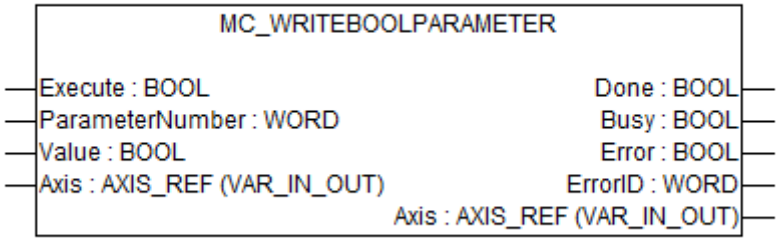


Fig. 395: Function block MC_WriteBoolParameter

When a drive based axis implementation is used, the function block writes the value of a drive specific BOOL parameter or a PLCopen BOOL parameter. When the Central Motion Control axis implementation is used, a BOOL parameter according to the list PLCopen parameter is written. ↳ *Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595*

All available parameters are listed in this table: ↳ Chapter 1.5.9.3.6 “PLCopen parameter” on page 2595

See also:

- *MC_ReadParameter to read parameters ↳ Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*
- *MC_WriteParameter to write parameters ↳ Chapter 1.5.9.6.3.8 “MC_WriteParameter” on page 2848*
- *MC_ReadBoolParameter to read Boolean parameters ↳ Chapter 1.5.9.6.3.6 “MC_ReadParameter” on page 2844*

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

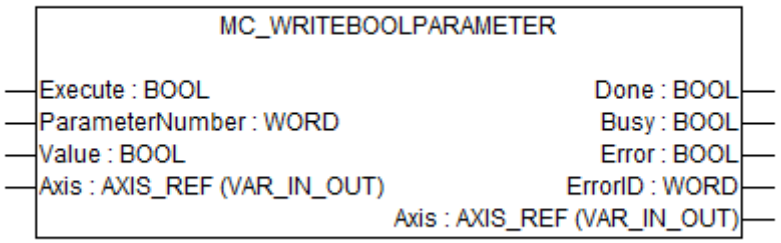


Fig. 396: Function block MC_WriteBoolParameter

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

Parameter-Number

Data type: INT

Number of the parameter (correspondence between number and parameter is to be specified later).

Value

Data type: LREAL

New value of the specified parameter.

Axis

Data type: AXIS_REF

Reference to the axis.

Output description

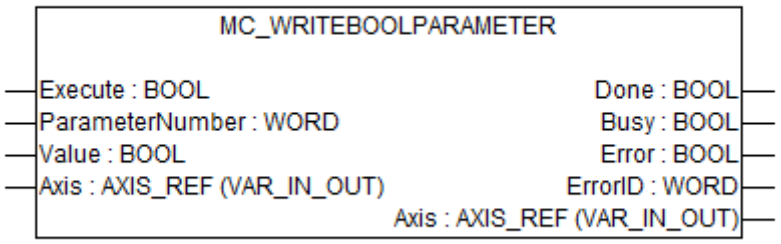


Fig. 397: Function block MC_WriteBoolParameter

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_ReadActualPosition

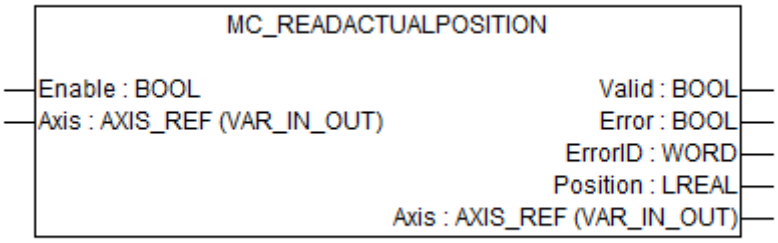


Fig. 398: MC_ReadActualPosition

Table 192: General information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block returns the actual position.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

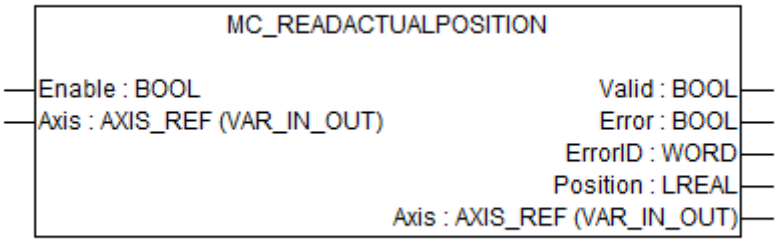


Fig. 399: MC_ReadActualPosition

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the value of the parameter continuously while enabled.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

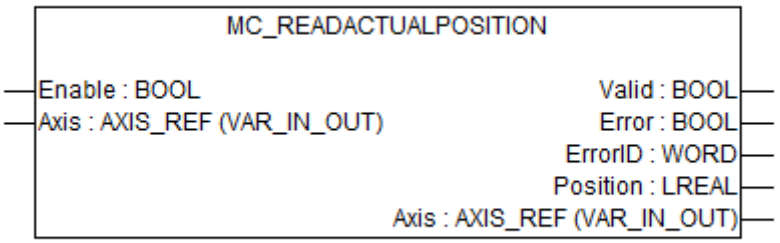


Fig. 400: MC_ReadActualPosition

Valid Data type: BOOL

Parameter available.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Position Data type: LREAL, unit: u

New absolute position.

MC_ReadActualVelocity

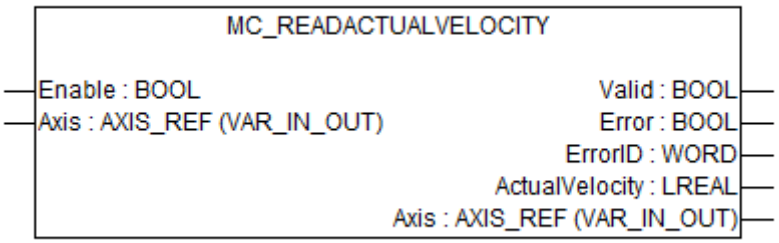


Fig. 401: MC_ReadActualVelocity

This function block returns the value of the actual velocity as long as Enable is set. Valid is true when the data-output “Velocity” is valid. If Enable is reset, the data loses its validity, and all outputs are reset, no matter if new data is available.

The output ActualVelocity can be a signed value.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

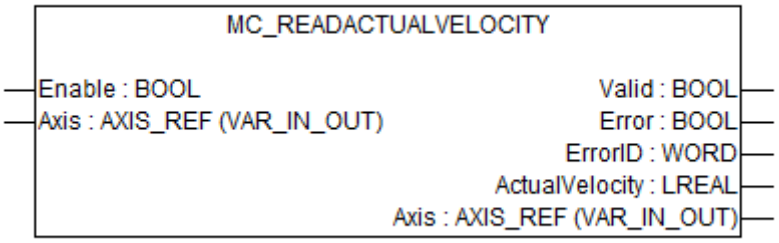


Fig. 402: MC_ReadActualVelocity

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the value of the parameter continuously while enabled.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

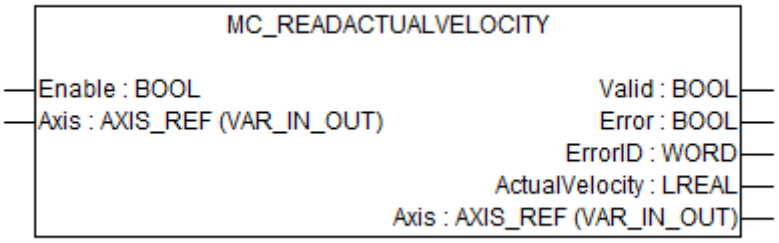


Fig. 403: MC_ReadActualVelocity

Valid Data type: BOOL

Parameter available.

Busy Data type: BOOL

The function block is not finished and new output values are to be expected.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

ActualVelocity Data type: LREAL, unit: u/s

The value of the actual velocity.

MC_SetOverride

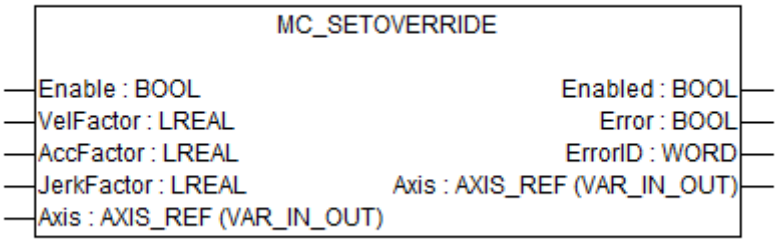


Fig. 404: Function block MC_SetOverride

This function block sets the values of override for the whole axis. The override parameters are applied as a factor that is multiplied to the set velocity, acceleration, deceleration and jerk during the executing of a motion function block. Ongoing motion commands are not affected when override factors are changed.

- The Input AccFactor acts on positive and negative acceleration (deceleration).
- This function block sets the factor. The override factor is valid until a new override is set.
- The default values of the override factors are 1.0.
- The value of the overrides can be between 0.0 and 1.0. Values > 1.0 and values < 0.0 are not allowed. The value 0.0 is not allowed for AccFactor and JerkFactor.

- *Override does not act on slave axes. (Axes in the state Synchronized Motion).*
- *The function block does not influence the single axis state diagram.*
- *The override factors are just effective to modify the velocity, acceleration, deceleration and jerk which are provided as explicit values by PLCopen function blocks. They do not modify a movement which is commanded by other means as camming, gearing, profiling, where no explicit velocity, acceleration,deceleration and jerk is in place.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

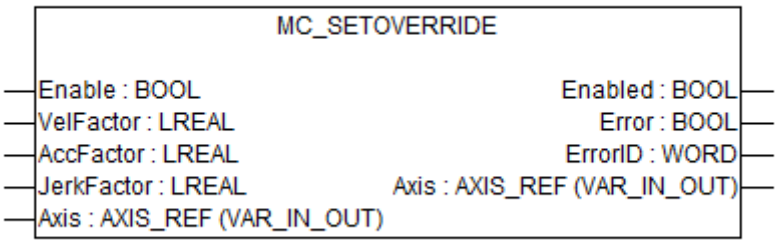


Fig. 405: Function block MC_SetOverride

Enable Data type: BOOL

If SET, it writes the value of the override factor continuously. If RESET it should keep the last value.

VelFactor Data type: LREAL

New override factor for the velocity.

AccFactor Data type: LREAL

New override factor for the acceleration or deceleration.

JerkFactor Data type: LREAL

New override factor for the velocity.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

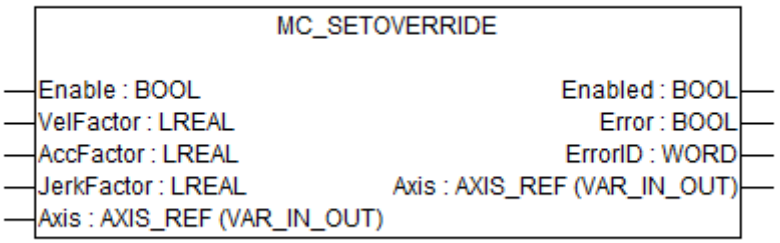


Fig. 406: Function block MC_SetOverride

Enabled Data type: BOOL

Signals that the override factor(s) is (are) set successfully.

Busy Data type: BOOL

The function block is not finished and new output values are to be expected.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_SetPosition

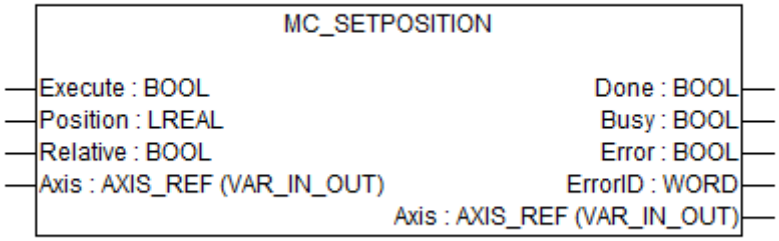


Fig. 407: Function block MC_SetPosition

This function block shifts the coordinate system of an axis by manipulating both the set-point position as well as the actual position of an axis with the same value without any movement caused. (Re-calibration with same following error). This can be used for instance for a reference situation. This function block can also be used during motion without changing the commanded position, which is now positioned in the shifted coordinate system.

The function block may just be called in: StandStill, Continuous Motion, ErrorStop or Disabled.

- *RELATIVE means that position is added to the actual position value of the axis at the time of execution. This results in a recalibration by a specified distance.*
- *ABSOLUTE means that the actual position value of the axis is set to the value specified in the position parameter.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

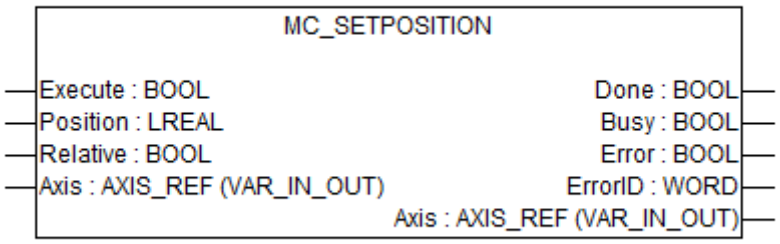


Fig. 408: Function block MC_SetPosition

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Position Data type: LREAL, unit: u

Position unit (Means ‘Distance’ if Relative= TRUE).

Relative Data type: BOOL, default: FALSE

Relative distance if TRUE. Absolute position if FALSE.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

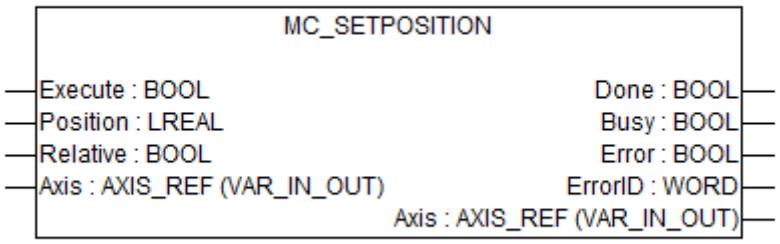


Fig. 409: Function block MC_SetPosition

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

1.5.9.6.4 Homing function blocks

For further information see: *AN00220-001 - AC500 and MicroFlex e150 - EtherCAT Homing Methods*

MC_StepAbsSwitch

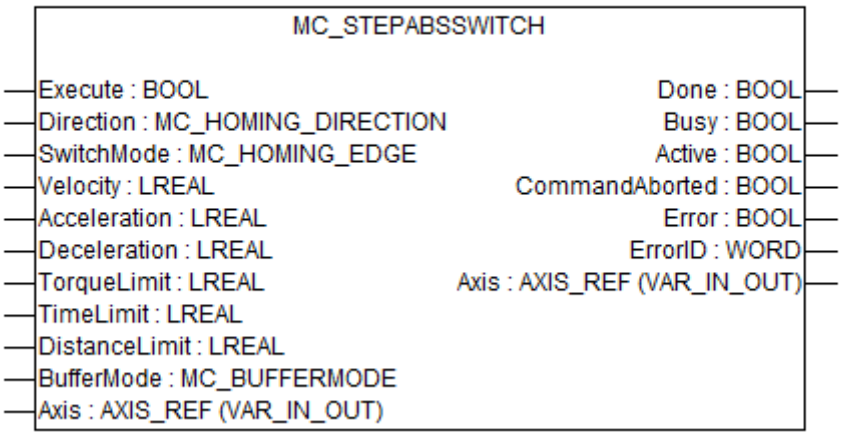
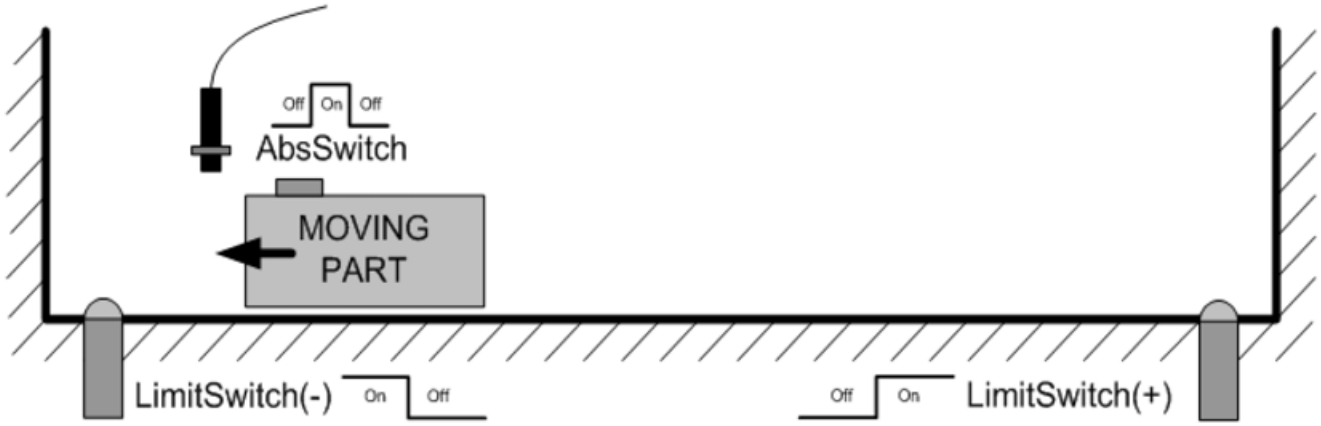


Fig. 410: Function block MC_StepAbsSwitch

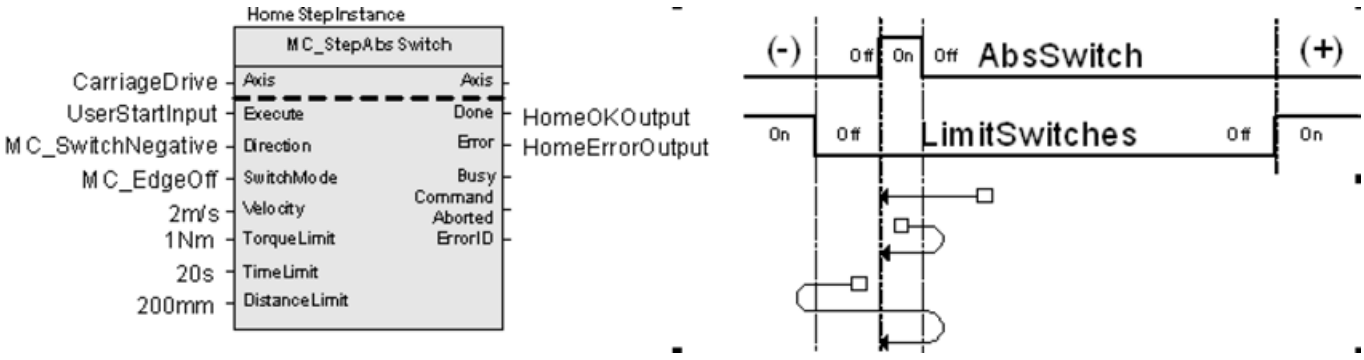
This function block performs a homing function by searching for an absolute positioned external physical switch. (An Absolute Switch has two “Off” (or “On”) areas – see example). For central Motion Control implementation: The signal of the Absolute Switch has to be written to the variable “absRefSwitch” of the data type CMC_AXIS_IO.

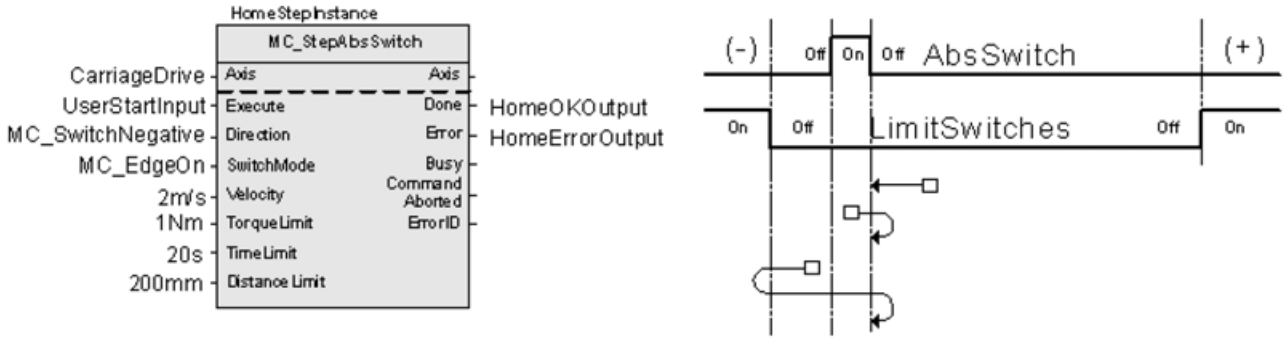
> *Inside the operation area the limit switches have to be logically FALSE and outside the borders the signal of the corresponding limit switch has to be logically TRUE.*
>
> *If needed the signal from the sensor must be inverted before it is connected to an element the AXIS_IO data type.*



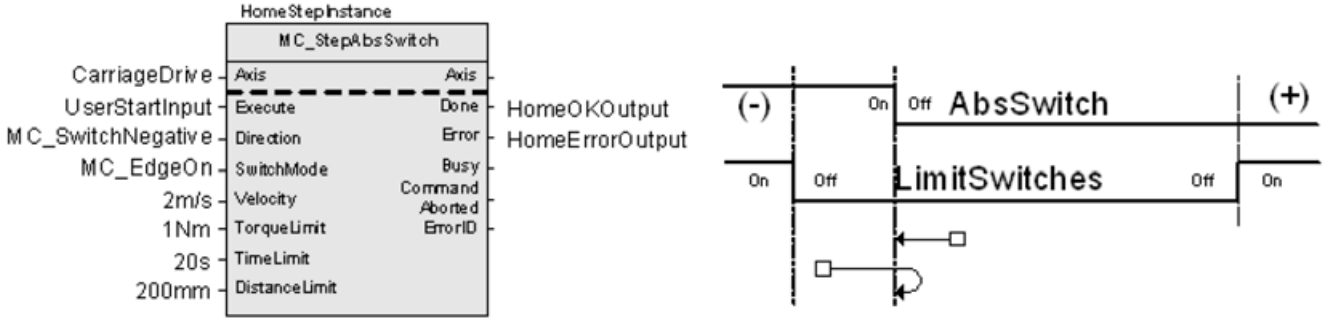
This physical layout has the risk that homing is started in the wrong direction (escaping the switch). To support such case, it implements a special behavior when Limit Switches are found (or the AbsSwitch itself is "On" at Execute):

- Axis State is set to Homing,
- The homing is commanded in the most likely direction were the sensor can be found. In the example (-),
- The velocity is defined by the input,
- Both time and distance limits can cause an error if exceeded,
- If any LimitSwitch is found during Homing (any of them), then a special process is started in the opposite direction, the AbsSwitch is searched to switch off (or "On" depending on SwitchMode setting). The Edge (passed by), and homing process is restarted in the original direction and with the same conditions. This ensures that the end conditions are always same,
- If the SwitchMode is either MC_SwitchNegative or MC_SwitchPositive, then the special process is also started in opposite direction depending from the switch state at "Execute",
- The direction changes only when the specified Velocity is reached (InVelocity),
- This function block does not modify the actual position,
- This function block does not leave the Homing State when done.
- This function block can only be used once for a homing sequence.

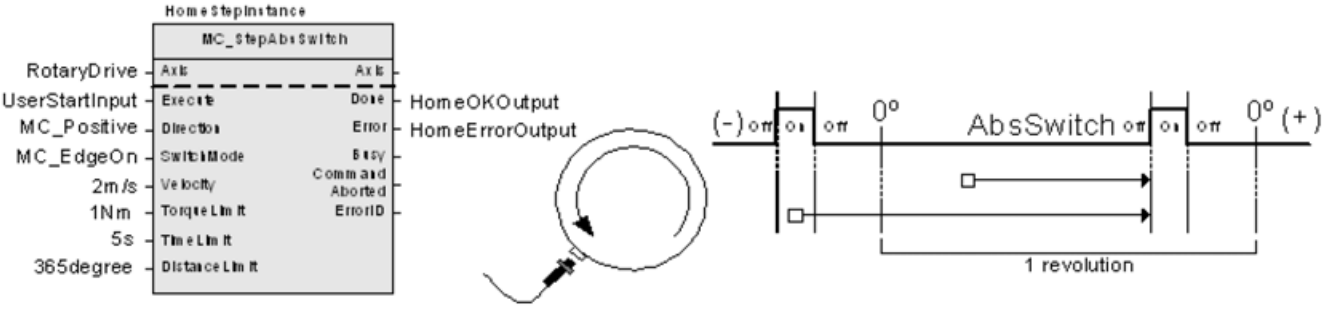




An overlapping switch configuration is also possible. This has same the behavior as working on the limit switches:



If the input direction is set to a fixed direction (MC_Positive or MC_Negative), then the initial switch state is ignored (used for example in rotary axis where only one sense of rotation is allowed):



With an overlapping switch configuration either MC_EdgeOn or MC_EdgeOff can be used for the input SwitchMode. This depends on the switching behavior of the absolute switch and the used option for the input Direction.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

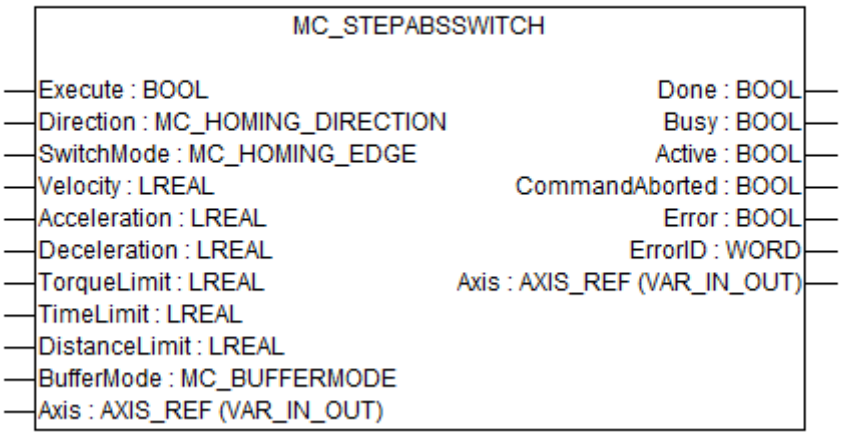


Fig. 411: Function block MC_StepAbsSwitch

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Direction Data type: MC_HOMING_DIRECTION

Specifies the direction of the motion if any:

- MC_Positive = Starts in positive direction always.
- MC_Negative = Starts in negative direction always.
- MC_SwitchPositive = Depends on Switch status at Execute edge. If Switch is "Off", direction is positive, if "On" it is negative.
- MC_SwitchNegative = Like previous, but opposite.

SwitchMode Data type: MC_HOMING_SWITCH

Sensor condition to finalize this function block in any switch mode:

- MC_On = When sensor is ON.
- MC_Off = When sensor is OFF.
- MC_EdgeOn = When Off to On transition in sensor.
- MC_EdgeOff = When On to Off transition in sensor.

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

TorqueLimit Data type: LREAL, unit: t.u.

Maximum torque or force. 0 = No torque limit.

Central Motion Control implementation (Compact Motion Control): TorqueLimit value will not be used.

TimeLimit Data type: LREAL, unit: s
If the function block condition is not met in the TimeLimit, an error is issued.
0 = No time limit.

DistanceLimit Data type: LREAL
If the Funciton Block condition is not met within a DistanceLimit travel, an error is issued.
0 = No distance limit.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

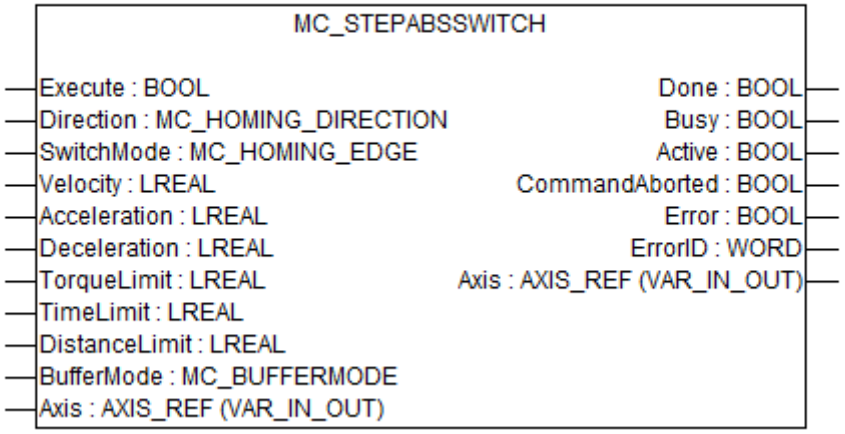


Fig. 412: Function block MC_StepAbsSwitch

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Specific error numbers:

- MC_TimeLimitExceeded
- MC_DistanceLimitExceeded
- MC_TorqueLimitExceeded

MC_StepDirect

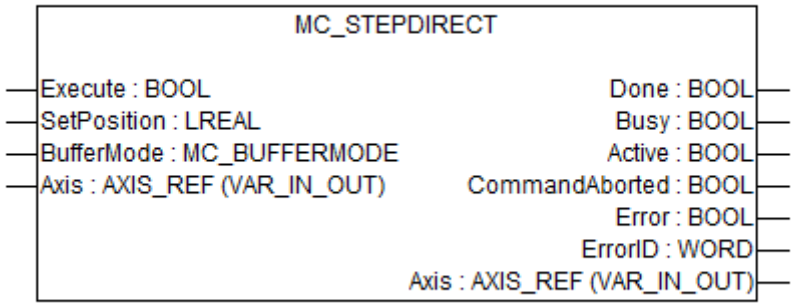


Fig. 413: Function block MC_StepDirect

This function block performs a static homing by directly forcing an actual position. No physical motion is performed in this mode. This is equivalent to a MC_SetPosition action, but clears the Homing State.

- *This function block modifies actual position and sets to the "SetPosition" input value at the end.*
- *This function block clears the Homing State when Done.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

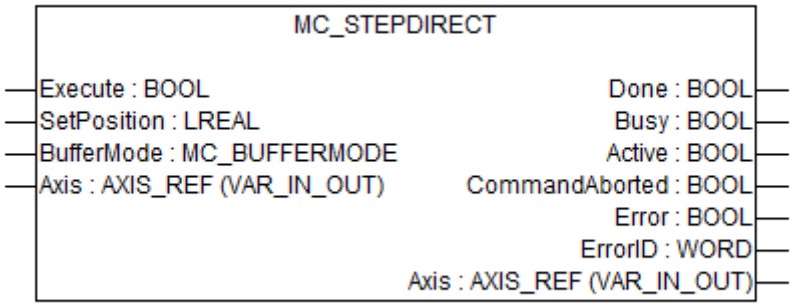


Fig. 414: Function block MC_StepDirect

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

SetPosition Data type: LREAL, unit: u

Value of the absolute position to be set when homing is done.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

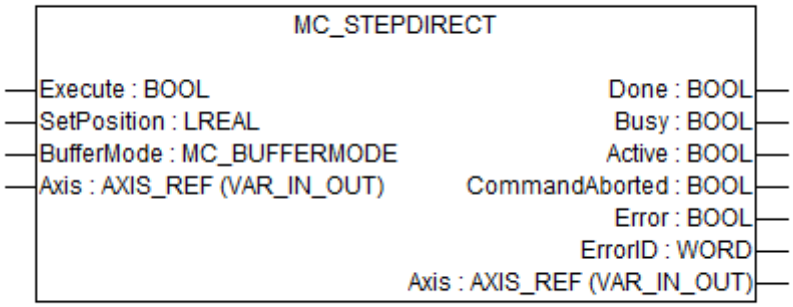


Fig. 415: Function block MC_StepDirect

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_StepLimitSwitch

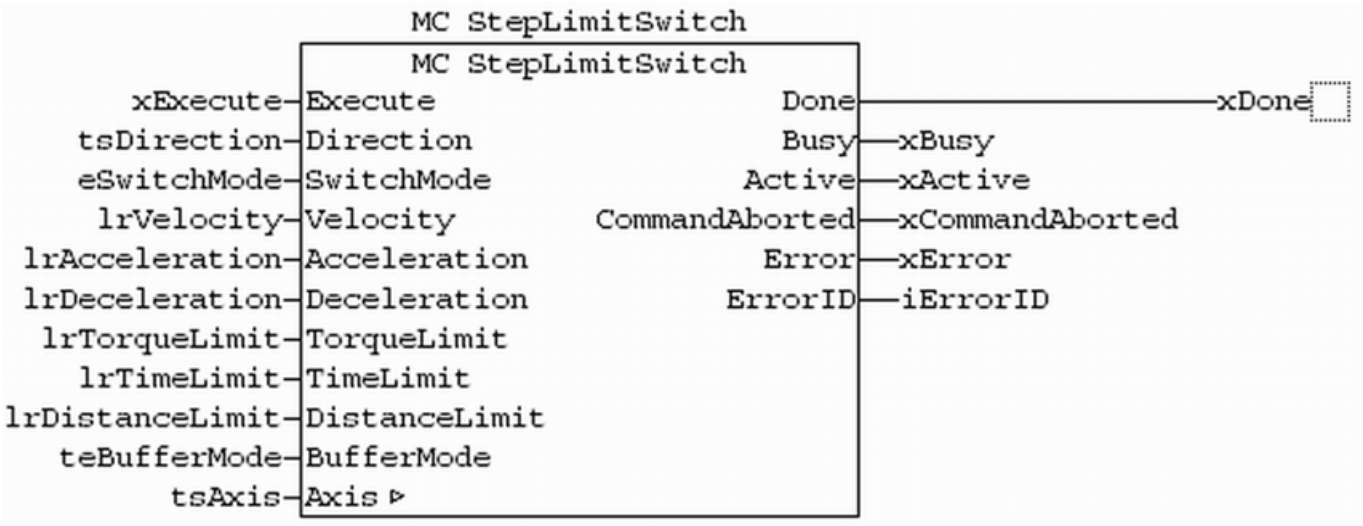
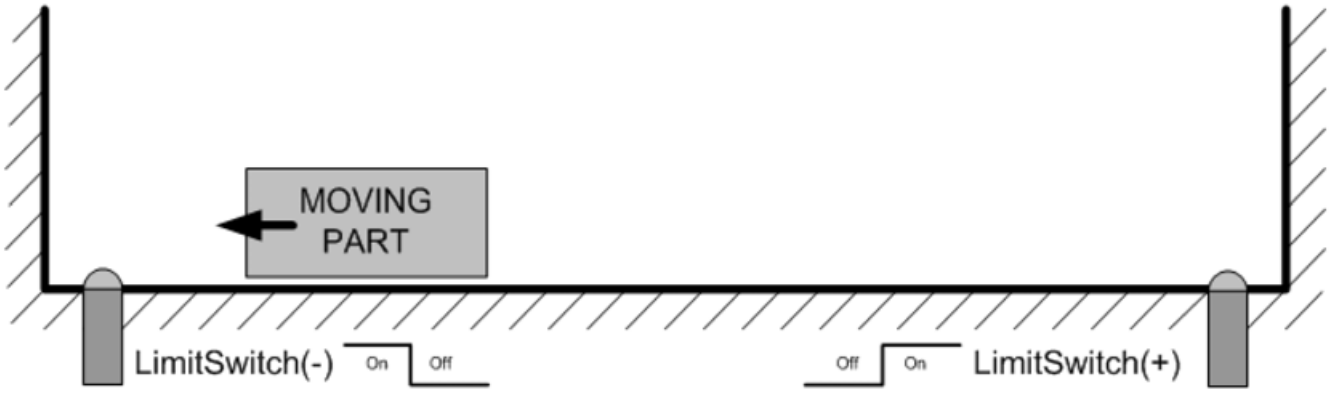


Fig. 416: Function block MC_StepLimitSwitch

This function block performs a homing function by searching for sensor using only limit switches. (A limit switch has 1 "Off" (or "On") area). The signal of the Limit Switches have to be written to the variables “limitSwitchPos” and “limitSwitchNeg” of the data type CMC_AXIS_IO.

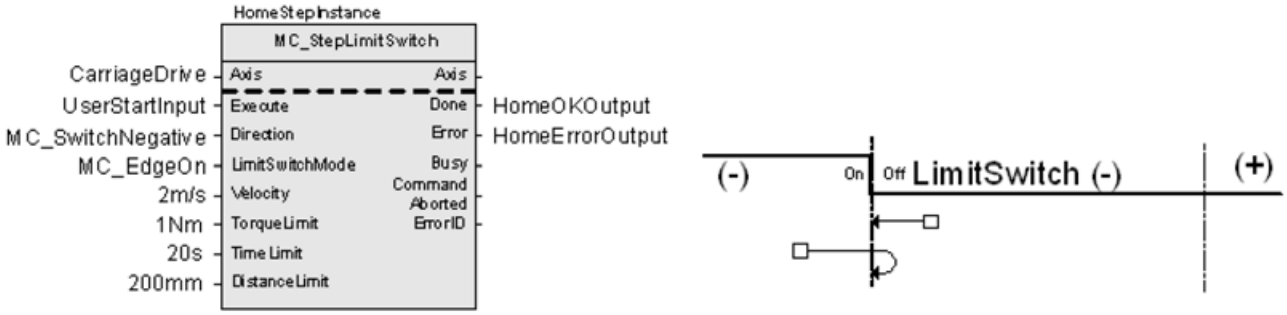
Inside the operation area the limit switches have to be logically FALSE and outside the borders the signal of the corresponding limit switch has to be logically TRUE.

If needed the signal from the sensor must be inverted before it is connected to an element the AXIS_IO data type



In this case the limit switches (always active once moving part working area has been surpassed) are used for homing procedure.

- The axis State is changed to Homing.
- Home is commanded by user in the desired homing direction at the selected Velocity.
- If LimitSwitch is found "On" on rising "Execute", then the process is started in the opposite direction as specified, LimitSwitch is search for "Off" (or On depending in LimitSwitchMode setting) Edge (released), and process is restarted again in original direction. This ensures that the end conditions are always the same.
- The time and distance limits can cause error if exceeded.
- The direction changes only when the specified velocity is reached, this ensures acceleration and deceleration spaces are fixed.
- This function block does not modify actual position.
- This function block does not leave the Homing State when done.
- This function block can only be used once for a homing sequence.



See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

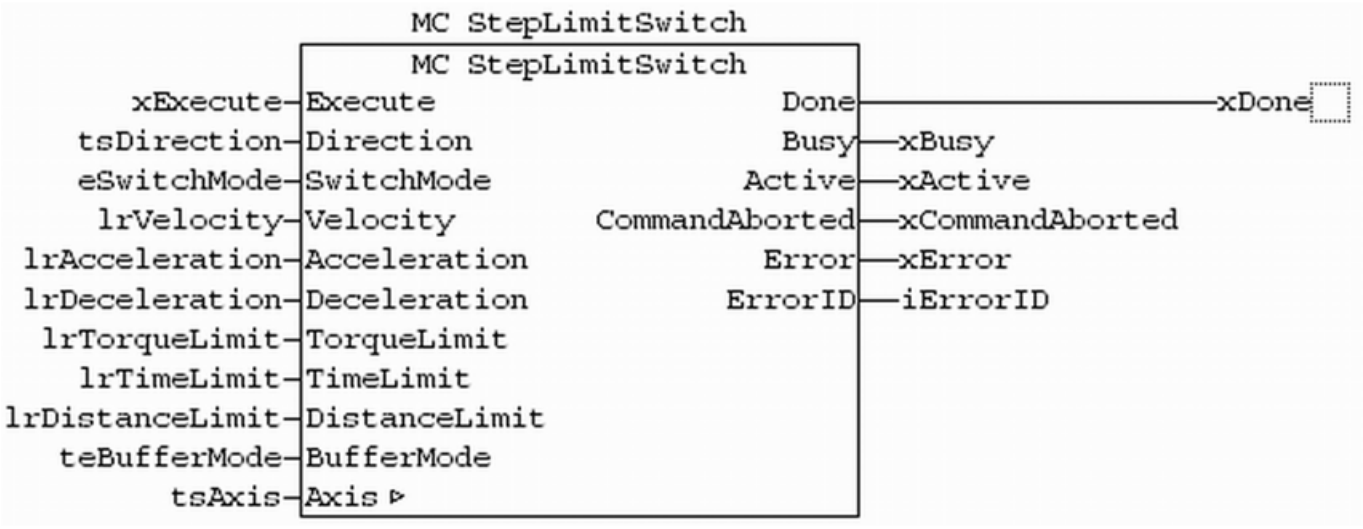


Fig. 417: Function block MC_StepLimitSwitch

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

Direction

Data type: MC_HOMING_DIRECTION

Specifies the direction of the motion and corresponding LimitSwitch to search for, just MC_Positive and MC_Negative are possible:

- MC_Positive = Positive direction searching positive LimitSwitch.
- MC_Negative = Negative direction searching negative LimitSwitch.

SwitchMode

Data type: MC_HOMING_EDGE

Sensor condition to finalize this function block:

- MC_On = When sensor is ON.
- MC_EdgeOn = When Off to On transition in sensor.

Velocity

Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

TorqueLimit Data type: LREAL, unit: t.u.

Maximum torque or force. 0 = No torque limit.

Central Motion Control implementation (Compact Motion Control): TorqueLimit value will not be used.

TimeLimit Data type: LREAL, unit: s

If the function block condition is not met in the TimeLimit, an error is issued.

0 = No time limit.

DistanceLimit Data type: LREAL

If the Funciton Block condition is not met within a DistanceLimit travel, an error is issued.

0 = No distance limit.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

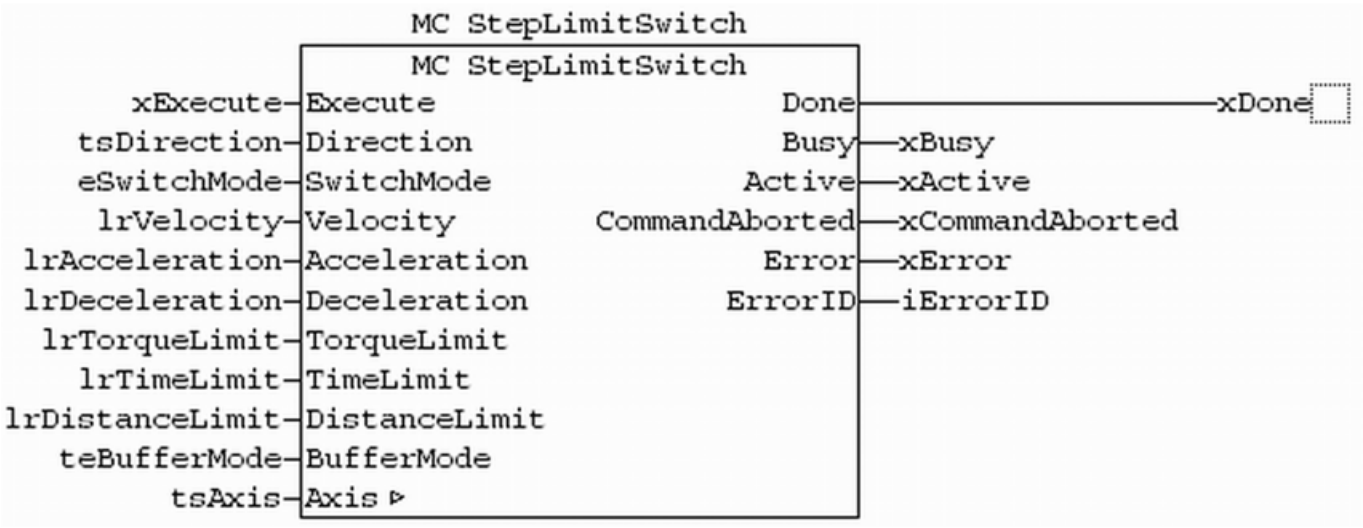


Fig. 418: Function block MC_StepLimitSwitch

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Specific error numbers:

- MC_TimeLimitExceeded
- MC_DistanceLimitExceeded
- MC_TorqueLimitExceeded

MC_StepRefPulse

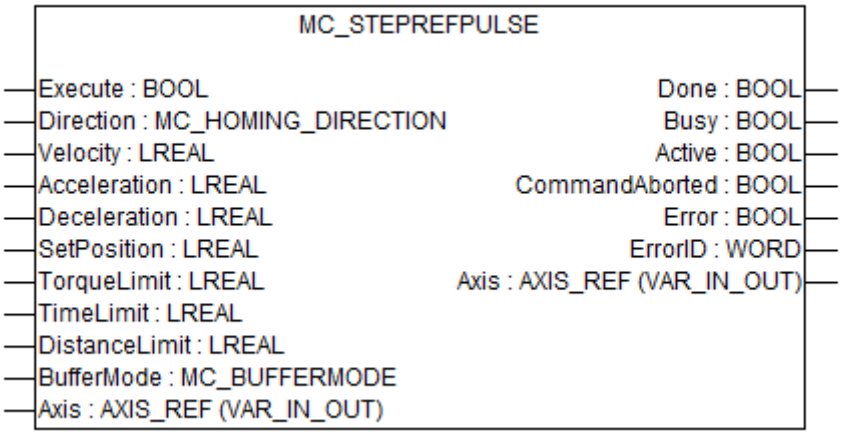
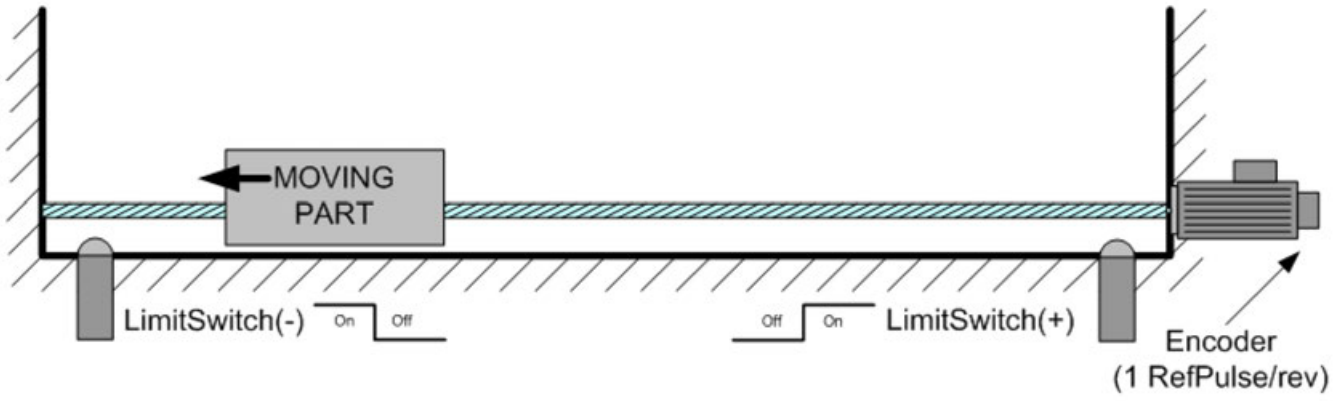


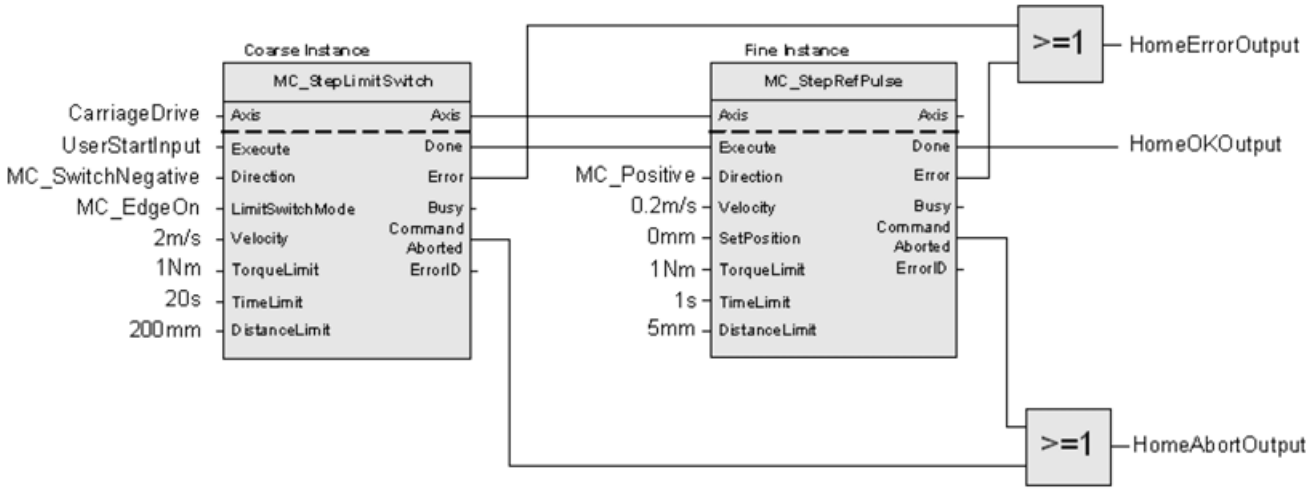
Fig. 419: Function block MC_StepRefPulse

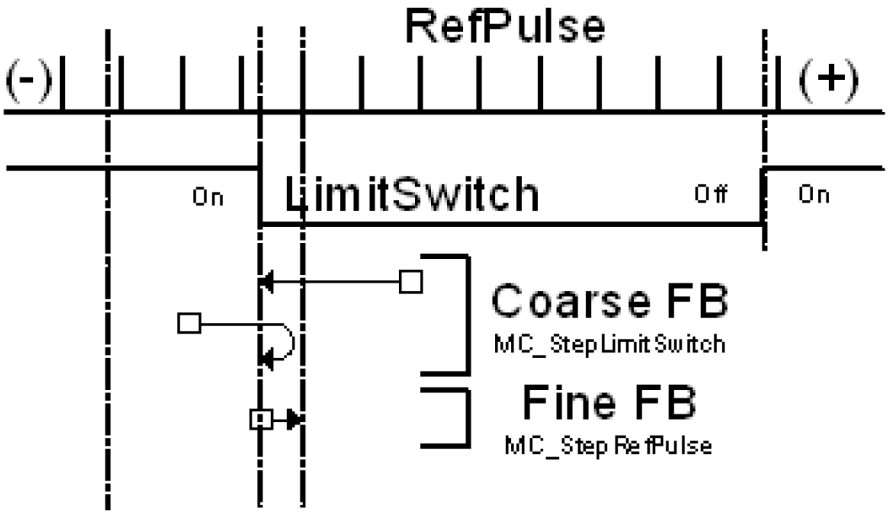
This function block performs homing by searching for zero pulse (also called Marker or reference pulse) in encoder. The reference pulse appears once per encoder revolution. The advantage in using Reference Pulse for homing is the higher accuracy and precision that can be achieved compared to traditional optical, mechanical or magnetic sensors.



- The axis state is changed to Homing if not already in.
- Home is commanded by user in the desired homing direction at the programmed velocity.
- First occurrence of the Reference Pulse, Homing is finished.
- Torque is limited. Time and Distance Limits can cause error if exceeded.
- This Function modifies actual position and sets to the "SetPosition" input value at the end
- This function block clears the Homing State when Done.

It is common that a first approach is performed against a mechanical sensor at higher velocity, and after a Reference Pulse, at a lower velocity. This is a traditional 2-Step homing (Coarse by external Switch in reverse and Fine by Reference Pulse in forward). For ease of use both functions could be grouped together in single function block. Advantage having the function blocks separate is that any combination is possible (MC_Block and after MC_RefPulse, etc.), stating different velocity and conditions for each Step (higly flexible), without increasing homing function block complexity too much.





See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

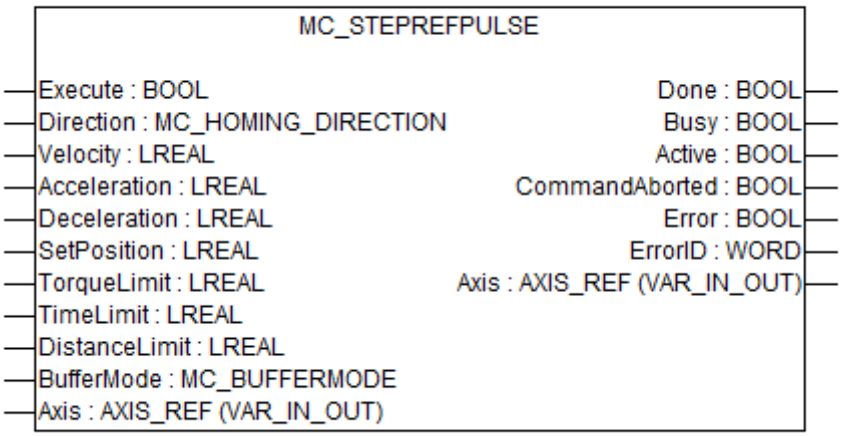


Fig. 420: Function block MC_StepRefPulse

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Direction Data type: MC_HOMING_DIRECTION

Specifies the direction to start the motion, just MC_Positive and MC_Negative are possible to use:

- MC_Positive = Starts in positive direction always
- MC_Negative = Starts in negative direction always

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

SetPosition Data type: LREAL, unit: u

Value of the absolute position to be set when homing is done.

TorqueLimit Data type: LREAL, unit: t.u.

Maximum torque or force. 0 = No torque limit.

Central Motion Control implementation (Compact Motion Control): TorqueLimit value will not be used.

TimeLimit Data type: LREAL, unit: s

If the function block condition is not met in the TimeLimit, an error is issued.

0 = No time limit.

DistanceLimit Data type: LREAL

If the Funciton Block condition is not met within a DistanceLimit travel, an error is issued.

0 = No distance limit.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output description

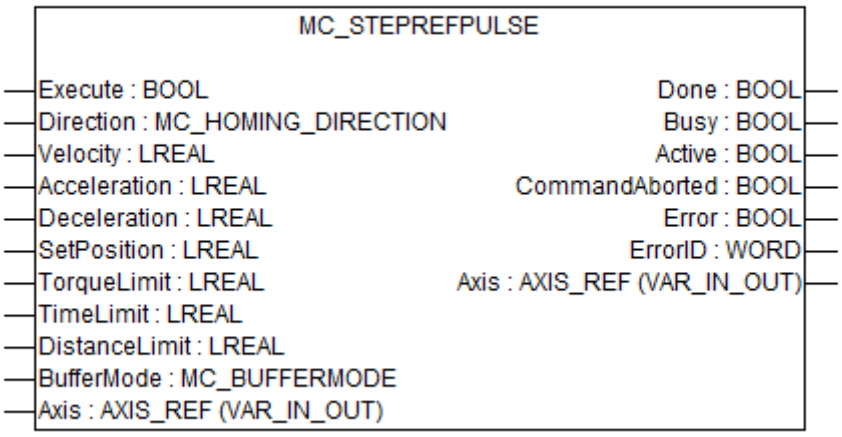


Fig. 421: Function block MC_StepRefPulse

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Specific error numbers:

- MC_TimeLimitExceeded
- MC_DistanceLimitExceeded
- MC_TorqueLimitExceeded

MC_Home

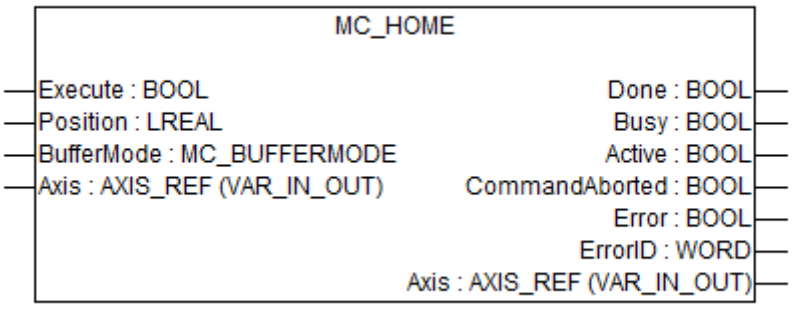


Fig. 422: Function block MC_Home

This function block commands the axis to perform the search home sequence. The details of this sequence are manufacturer dependent and can be set by the axis' parameters. The input Position is used to set the absolute position when reference signal is detected. This Function Bock completes at StandStill.

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input description

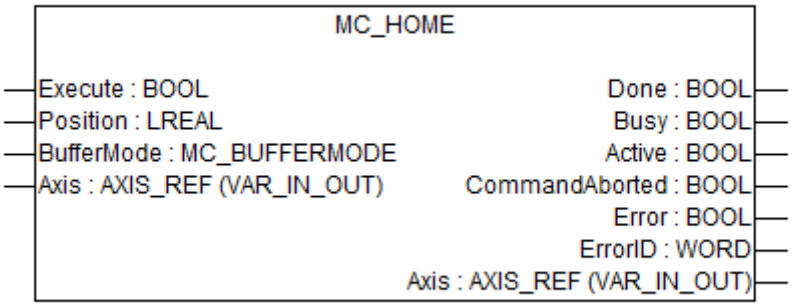


Fig. 423: Function block MC_Home

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Position Data type: LREAL, unit: u
New absolute position.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output description

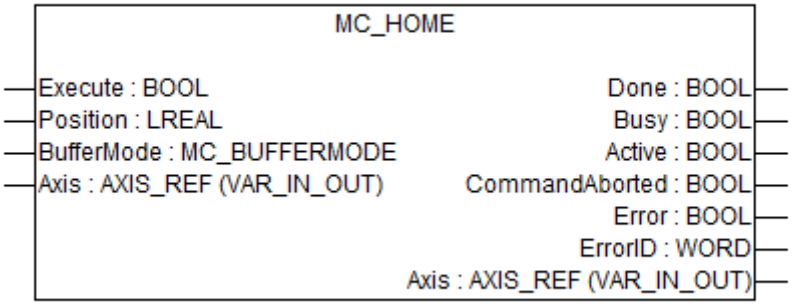


Fig. 424: Function block MC_Home

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

1.5.9.6.5 ABB specific function blocks

The ABB specific function blocks follow the general rules defined by PLCopen and implement some additional features.

MCA_CAM_EXTRA

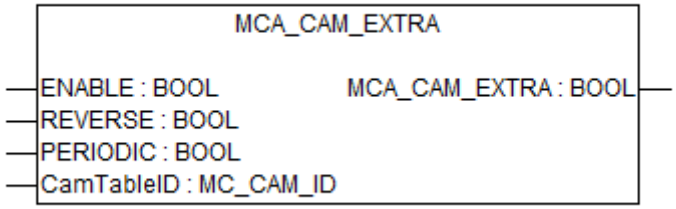


Fig. 425: Function MCA_CAM_EXTRA

This function is just usable together with MC_CamTableSelect and should be called right after MC_CamTableSelect to modify 2 mode-bits which define the behavior for the MC_CamIn more precise. Without this function, the default values will be used instead.

This Function modifies the CAM Table behavior.

- *With ENABLE = TRUE, the 2 bits will be written all the time and will be effective. So a cam table could be used in PERIODIC=TRUE mode and will come to a stop when the master leaves its position range when PERIODIC = FALSE is used.*
- *With MODULO-AXIS:*
  Uusage with PERIODIC = FALSE and position range for master equals MODULORANGE: When the master reaches 360°, the movement will be ready, even when it was started just at 359°.

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

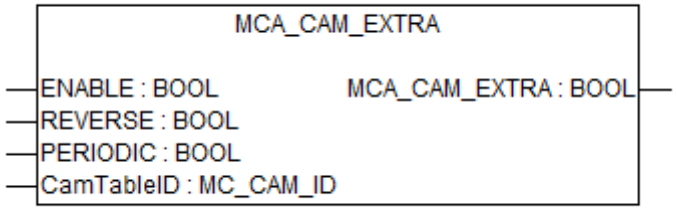


Fig. 426: Function MCA_CAM_EXTRA

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

ENABLE Data type: BOOL

Writes the values while ENABLE = TRUE.

REVERSE Data type: BOOL

Default value=FALSE, just relevant with PERIODIC = FALSE, the cam-table will reach "EndOf-Profile" when the master axis leaves the position range in the given direction

REVERSE = FALSE: positive direction

REVERSE= TRUE: negative direction.

PERIODIC Data type: BOOL

Default value = TRUE for master axis= MODULO, FALSE otherwise PERIODIC= TRUE, the CAM table will not reach EndOfProfile.

CamTableID Data type: MC_CAM_ID

Identifier of CAM Table to be used in the MC_CamIn function block.

MCA_Parameter

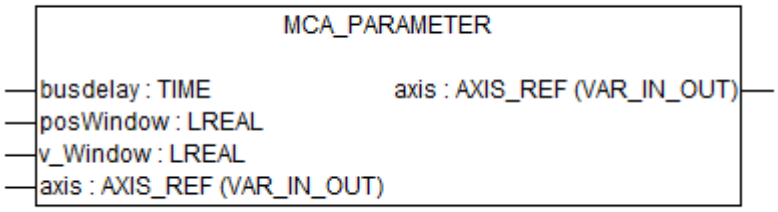


Fig. 427: Function MCA_Parameter

This function can be used to change the default values of the following parameters:

- Target position window. The default value is 10 units.
- Target velocity window. The default vaule is 10 units.
- Maximum fieldbus delay. If this value will be exceeded then it will be assumed that there is a communication error.

The block MCA_Parameter has to be used to adjust the velocity limit when velocities < 10 u/s should be used (10 u/s: default value).

The parameter v_Window defines the limit for the axis to reach its target velocity or standstill.

CAUTION!
This detection will not work properly when smaller velocities are used, especially the block MC_StepLimitSwitch will not stop the axis when reaching the switch!

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

busdelay Data type: TIME

A delay time to wait for fieldbus data. When the delay time is to long, the reaction time of function blocks might be increased, while when it is to short an error might be indicated although everything is ok.

posWindow Data type: DWORD

A position window to indicate that the movement is ready. It is a supervision additional to the drives control. When this value is too small, e.g. 1 increment, it might happen that a movement never indicates to be ready.

v_Window Data type: DWORD

A velocity window to indicate that the axis reached the commanded velocity.

MCA_Power

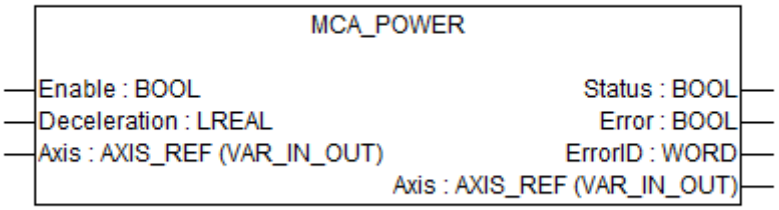


Fig. 428: Function block MCA_Power

This function block controls the power stage (on or off).

- *The basic behavior is the same as MC_Power.*
- *The additional input "Deceleration" is used to ramp the axis down to velocity = 0 in case it was moving when the MCA_Power was activated by a positive edge on "Enable"*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

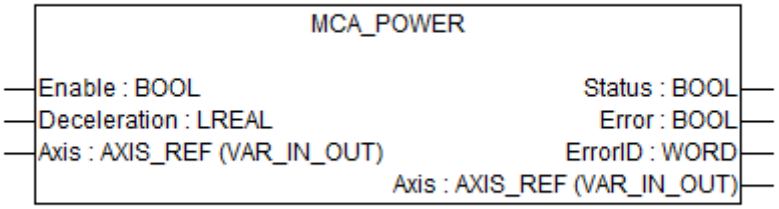


Fig. 429: Function block MCA_Power

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

As long as Enable = TRUE, power is on.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

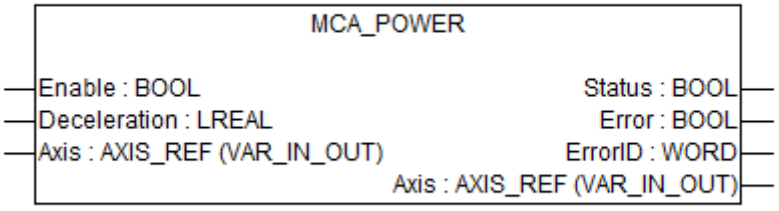


Fig. 430: Function block MCA_Power

Status Data type: BOOL

Effective state of the power stage.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_MoveVelocityContinuous

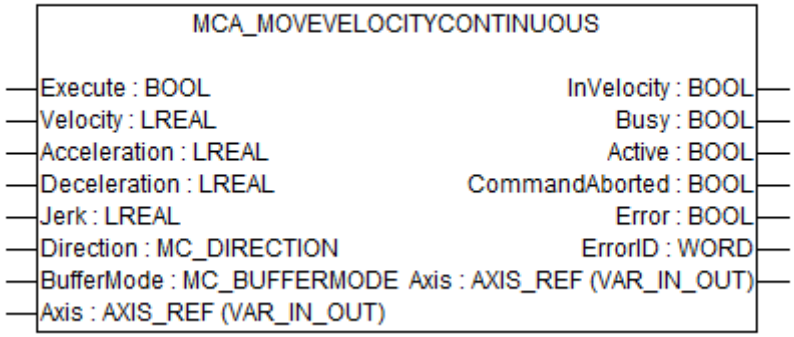


Fig. 431: Function block MCA_MoveVelocityContinuous

Table 193: General Information

Available as of runtime system	V2.2
Included in library	MC_Block_AC500_V11.lib
Type	Function block with historical values

This function block commands a never ending controlled motion at a specified velocity.
The difference to function block MC_MoveVelocity *Chapter 1.5.9.6.1.6 “MC_MoveVelocity” on page 2767* is that the values for Velocity, Acceleration and Deceleration can be modified continuously. If there is a change of the velocity, the reaction on the signal InVelocity will be delayed for 1 cycle.

- *To stop the motion, the function block has to be interrupted by another function block issuing a new command.*
- *The signal "InVelocity" has to be reset when the block is aborted by another block or at the falling edge of "Execute".*
- *In combination with MC_MoveSuperimposed, the output “"nVelocity" stays TRUE once the velocity setpoint of the axis has reached the commanded velocity.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Using this function block with FM562

Input Jerk: 1 = jerk on, 2 = jerk off

Input Direction: 0 = DEFAULT, 1 = POSITIVE, 3 = NEGATIVE

Input BufferMode: Not implemented

Input Description

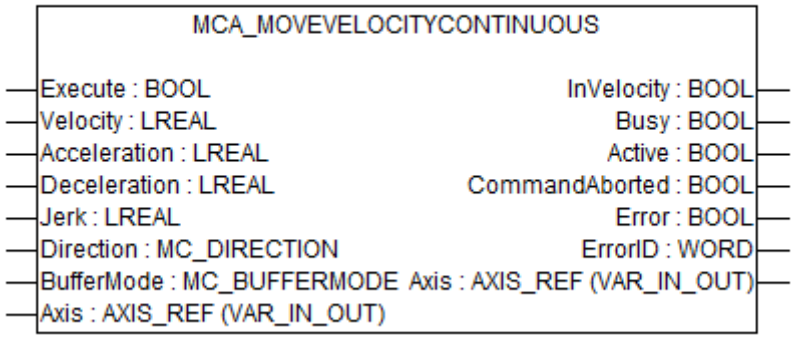


Fig. 432: Function block MCA_MoveVelocityContinuous

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

Velocity

Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk

Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

Direction Data type: MC_Direction, range: DEFAULT, POSITIVE, SHORTEST, NEGATIVE, CURRENT
Enum type.

> *Shortest way not applicable, DEFAULT is equivalent with CURRENT*

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output Description

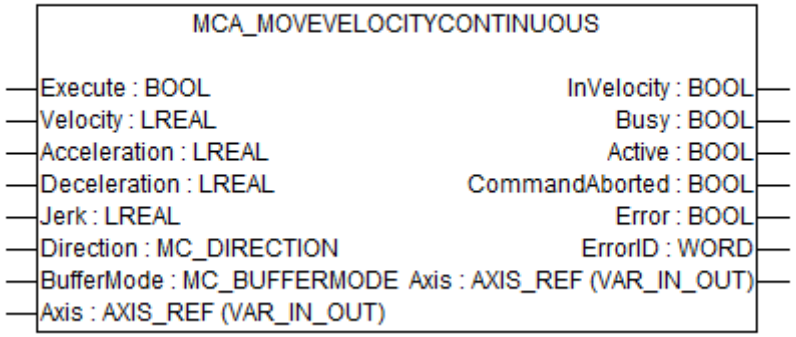


Fig. 433: Function block MCA_MoveVelocityContinuous

InVelocity Data type: BOOL
Commanded velocity reached (first time reached).

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_MoveByExternalReference

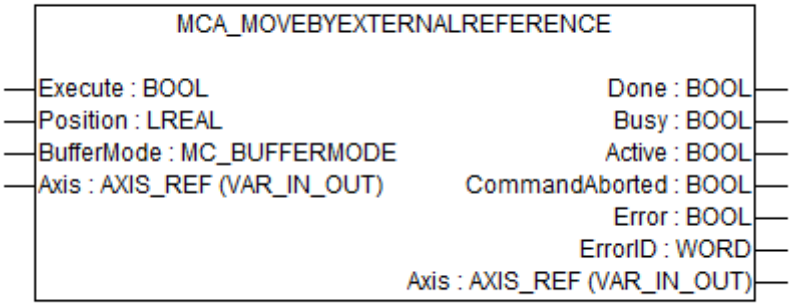


Fig. 434: Function block MCA_MoveByExternalReference

This function block gives a reference position to the axis which is directly passed to the position control loop. The axis will follow the given position without a ramp but immediately. The reference position is evaluated continuously.

> *To stop the motion, the function block has to be interrupted by another function block issuing a new command.*
>
> *This block has to be called from the same task as CMC_MOTION_KERNEL_REAL.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

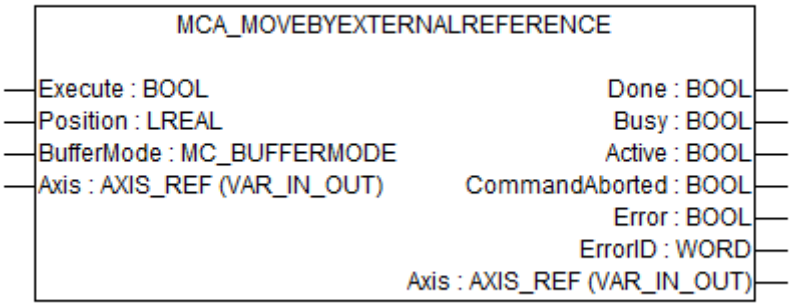


Fig. 435: Function block MCA_MoveByExternalReference

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute Data type: BOOL
Starts the function block at rising edge.

Position Data type: LREAL, unit: u
Reference position.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF
Reference to the axis.

Output Description

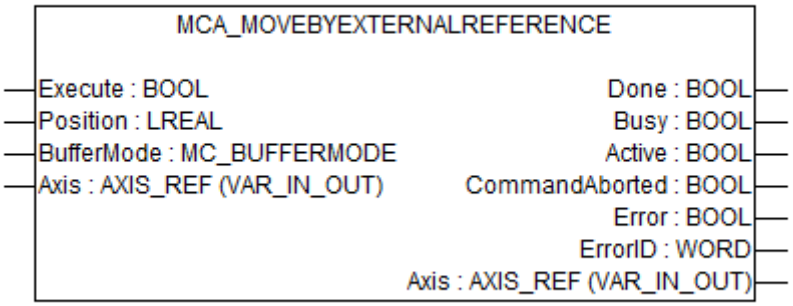


Fig. 436: Function block MCA_MoveByExternalReference

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_Home

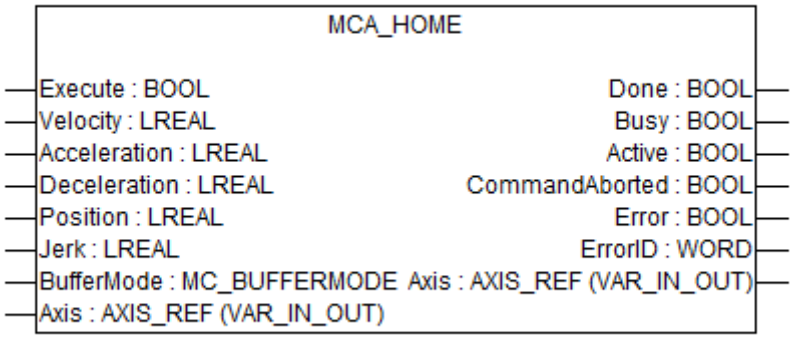


Fig. 437: Function block MCA_Home

The difference to the MC_Home is that the values for velocity, acceleration and deceleration can be set as inputs. The velocity is normally set by a parameter in the drive, so the input Velocity often does not influence the homing procedure. Acceleration and Deceleration are at start of the system zero as no other movement was done before. To performe a homing with defined Acceleration and Deceleration the inputs Acceleration and Decelerationcan be used.

This function block commands the axis to perform the «search home» sequence. The details of this sequence are manufacturer dependent and can be set by the axis’ parameters. The input Position is used to set the absolute position when reference signal is detected. The inputs Velocity, Acceleration, Deceleration are to define the homing motion start parameters. This Function Bock completes at StandStill.

The Homing procedure is executed according the definition in the drive.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

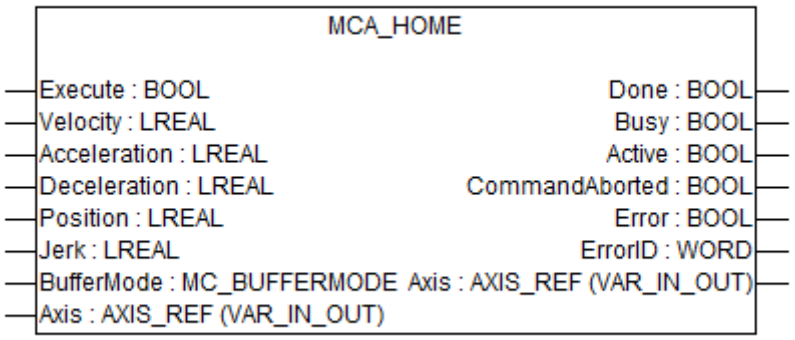


Fig. 438: Function block MCA_Home

Execute Data type: BOOL

Starts the function block at rising edge.

Velocity Data type: LREAL, unit: u/s

Value of the maximum velocity (not necessarily reached). The homing sequence in some drives uses Velocity from an internal parameter, so this input than has no influence .

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Position Data type: LREAL, unit: u

Absolute position when the reference signal is detected.

Jerk Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

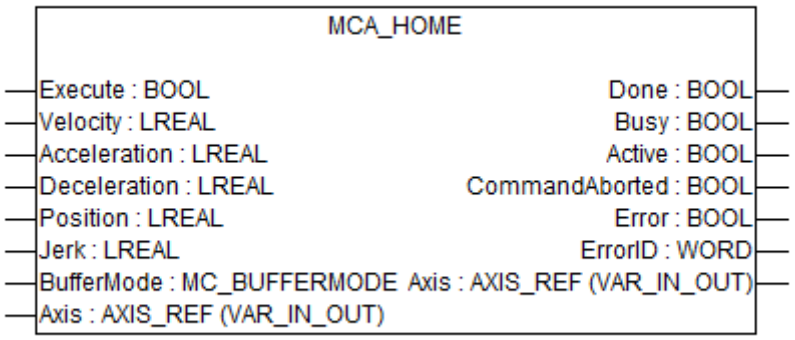


Fig. 439: Function block MCA_Home

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandA-borted Data type: BOOL
Command is aborted by another command.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_DriveBasedHome

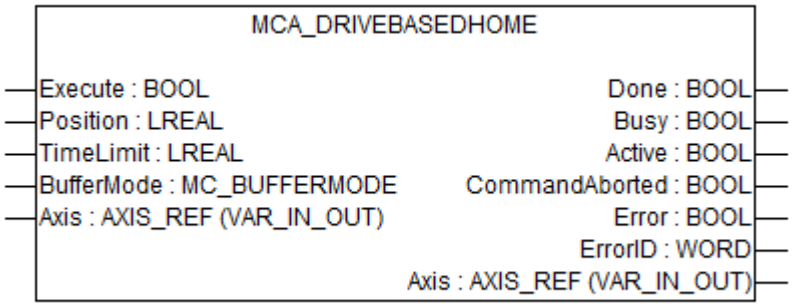


Fig. 440: Function block MCA_DriveBasedHome

This function block can be used to execute a homing procedure directly in the drive. It requires the drive supports 402-profile specific homing sequences.

The function block can be used in combination with:

- *ECAT_402ParameterHoming_APP* to send parameters
- *ECAT_CiA402_CONTROL_APP* to control the drive state machine and to set it to the appropriate operating mode

Input Description

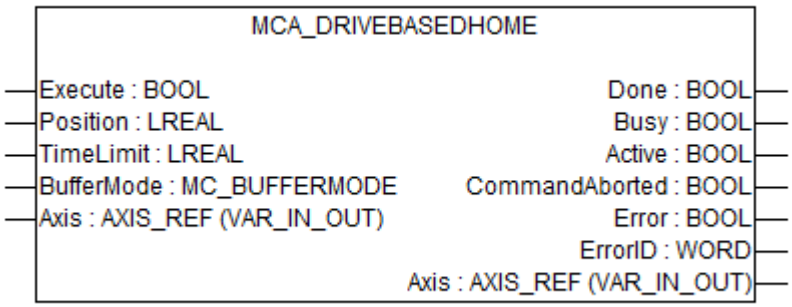


Fig. 441: Function block MCA_DriveBasedHome

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Position Data type: LREAL, default: 0

Position value in units, which will be used as home offset for the drive.

TimeLimit Data type: LREAL, default: 0
A time in seconds, which will be used as an upper limit for the time available to do the homing.
If the time is exceeded, the function block will show an Error.
With *TimeLimit = 0*, the limit is ignored.

Axis Data type: AXIS_REF
Reference to the axis.

Output Description

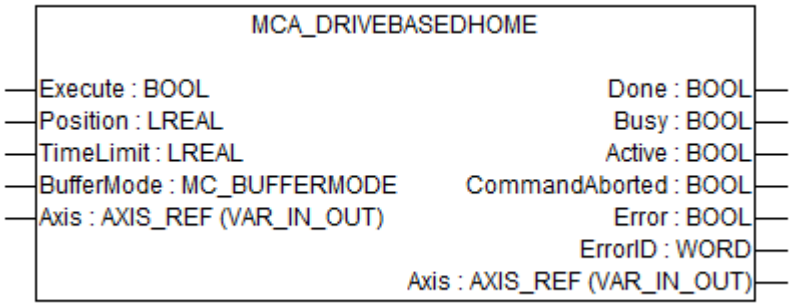


Fig. 442: Function block MCA_DriveBasedHome

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_Indexing

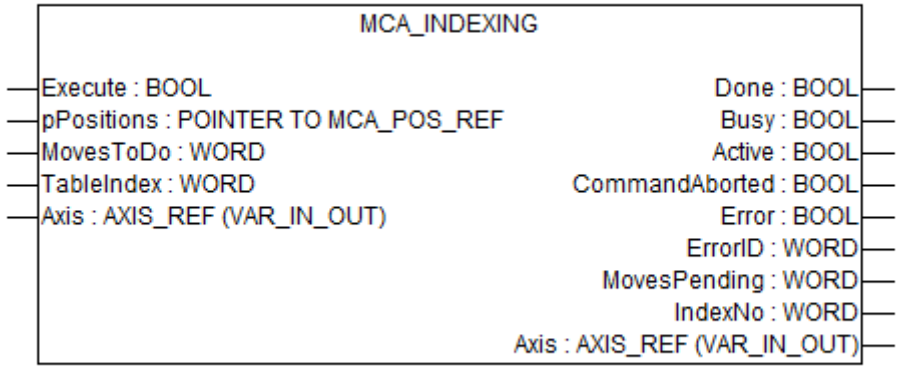
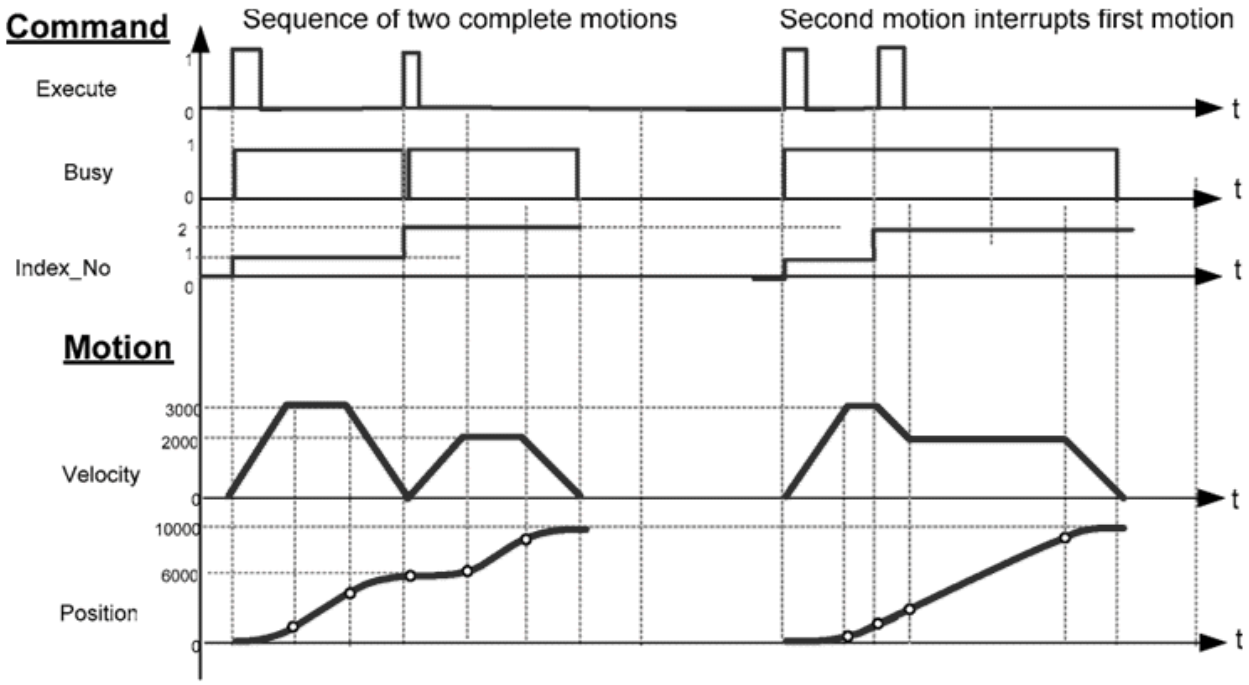


Fig. 443: MCA_Indexing

This function block will – upon R_Trig on Execute – do a number of relative or absolute moves, listed in a table (Array of MCA_POS_REF). The function block will position the axis to a complete stop at target position and continue with the next move from the table upon next R_Trig signal or automatically.



See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

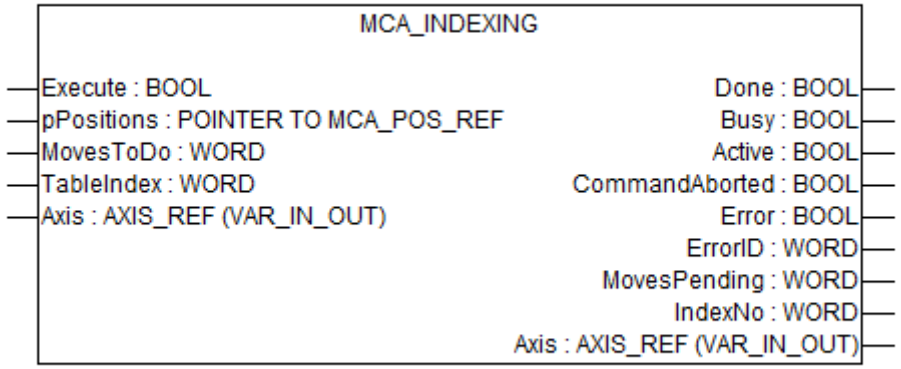


Fig. 444: MCA_Indexing

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

pPositions Data type: POINTER

Pointer to an array of MCA_POS_REF. The array needs to have at least (TableIndex + MovesToDo-1) elements.

MovesToDo Data type: WORD

Number of moves to be performed one after another.

TableIndex Data type: WORD

Index to an array of MCA_POS_REF, points to the movement to be performed on rising edge of **Execute**, start with **1** for the first entry.

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

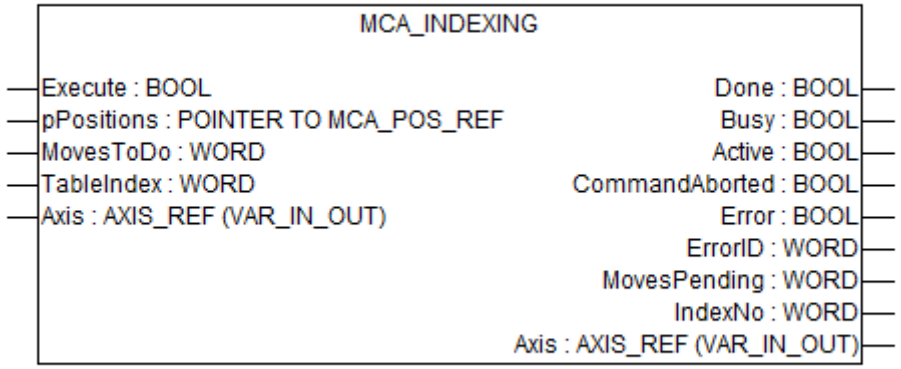


Fig. 445: MCA_Indexing

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block with Busy = TRUE has control on the axis.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MovesPending Data type: WORD
Indicates the number of moves still to execute.

IndexNo Data type: WORD
Index executing or last index completed, starts with **1**.

MCA_POS_REF

This structure is used to define the movement performed by MCA_Indexing. An array of this structure has to be created. Every single element holds the data for a complete movement. The address of this array has to be connected to input **pPositions** of MCA_Indexing.

MCA_POS_REF	Type	Comment
Position	LREAL	
Velocity	LREAL	
Acceleration	LREAL	
Deceleration	LREAL	
Mode	BOOL	TRUE = Use absolute positoning. FALSE = Use relative positoning.

MCA_JogAxis

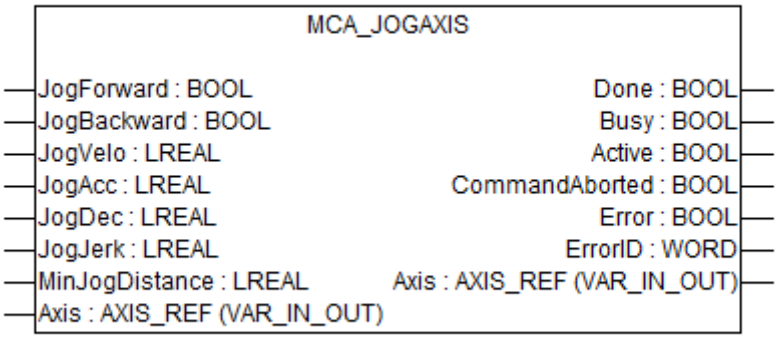


Fig. 446: Function block MCA_JogAxis

This function block jogs an axis for at least a given distance forward or backward with the selected jog velocity and acceleration.

This function block will, after rising edge on JogForward or JogBackward, start a continuous move (at least for the minimum distance) and continue upon high-level on these inputs with a continuous motion, until they are FALSE, then on their falling edge, the axis is regularly decelerated to stop. The movement is carried out on Jog velocity for the minimum distance or longer.

- *In case of both Enable signals are high, the function block will assume the result to be low as in an EXOR conjunction.*
- *In case of MinJogDistance= 0, no specified distance is moved and the movement will stop as soon as the JogForward and JogBackward= FALSE.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

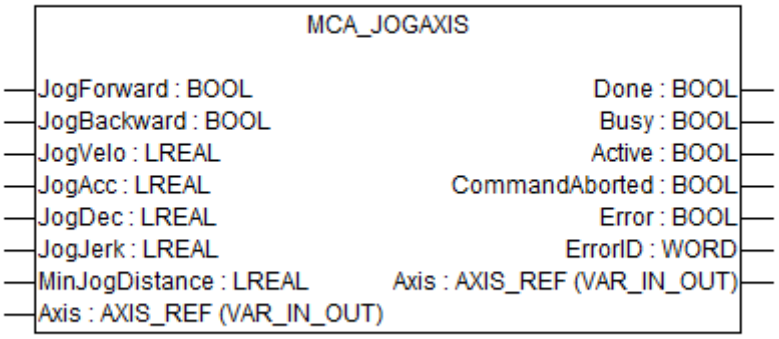


Fig. 447: Function block MCA_JogAxis

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

JogForward Data type: BOOL
Moves forward while “JogForward” =TRUE.

JogBackward Data type: BOOL
Moves backward while “JogBackward” =TRUE.

JogVelo Data type: LREAL
Velocity to jog .

JogAcc Data type: LREAL
Acceleration.

JogDec Data type: LREAL
Deceleration.

JogJerk Data type: LREAL
Jerk.

MinJogDistance Data type: LREAL
Minimum distance to jog (in t.u. [u]).

Axis Data type: AXIS_REF
Reference to the axis.

Output Description

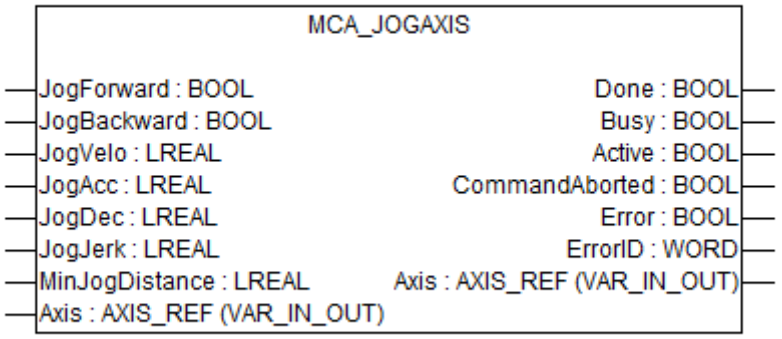


Fig. 448: Function block MCA_JogAxis

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_WriteParameterList

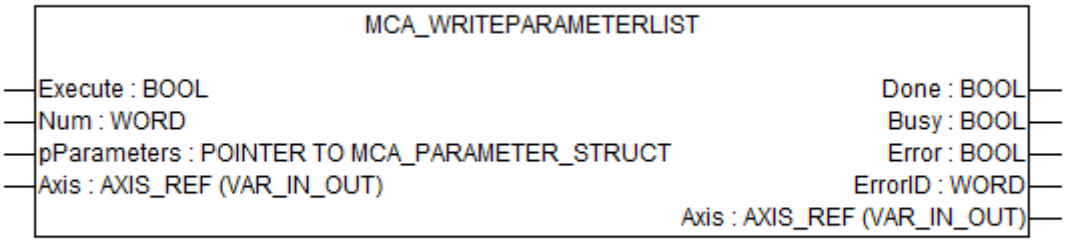


Fig. 449: Function block MCA_WriteParameterList

The function block writes a list of parameters by using the MC_WriteParameter. The rules for utilizing the function block correspond to MC_WriteParameter as well as the ErrorIds. All parameters and parameter numbers have to be stored in an array of type MCA_PARAMETER_STRUCT. The address of the first element is to be given to the function block's input “parameters”. The number of elements to be written is declared at input Num.

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

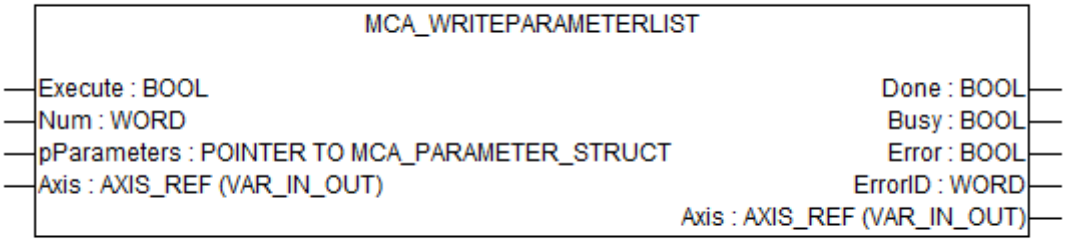


Fig. 450: Function block MCA_WriteParameterList

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Num Data type: WORD

Number of the parameters to be written.

pParameters Data type: POINTER

Points to an array of type MCA_Parameter_STRUCT which holds the parameter numbers and values.

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

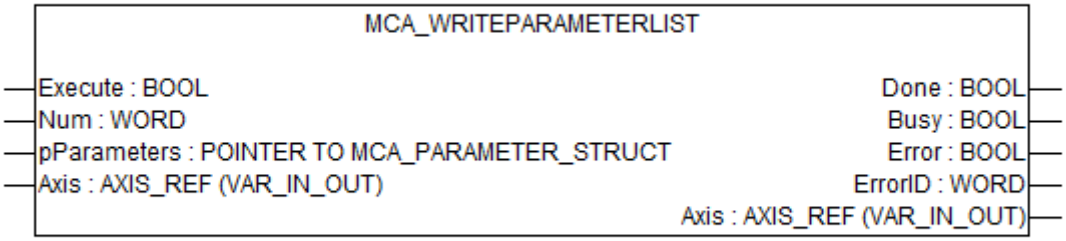


Fig. 451: Function block MCA_WriteParameterList

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_ReadParameterList

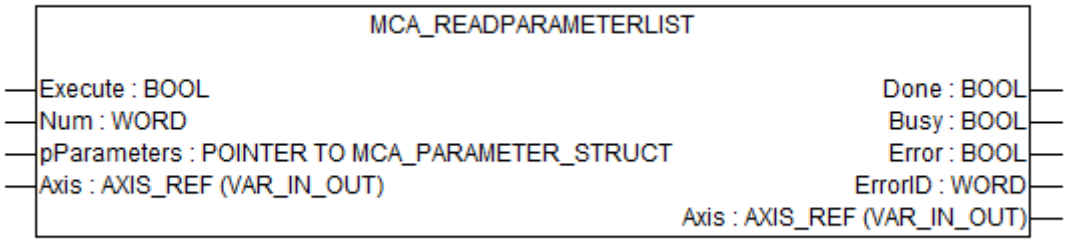


Fig. 452: Function block MCA_ReadParameterList

The function block reads a list of parameters by using the MC_ReadParameter. The rules for utilizing the function block correspond to MC_ReadParameter as well as the ErrorIds. All parameters and parameter numbers have to be stored in an array of type MCA_PARAMETER_STRUCT. The address of the first element is to be given to the function block's input parameters. The number of elements to be read is declared at input Num.

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input Description

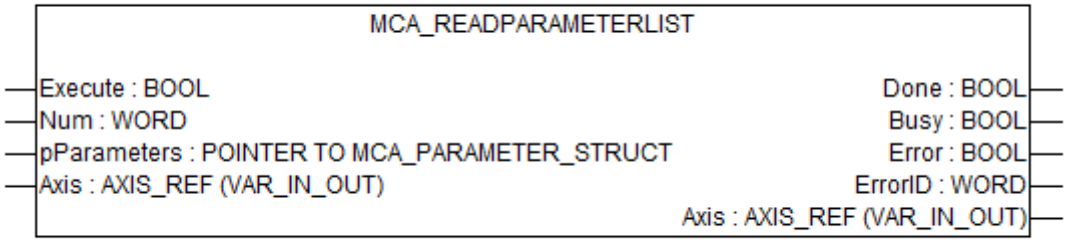


Fig. 453: Function block MCA_ReadParameterList

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

Num Data type: WORD

Number of the parameters to be written.

pParameters Data type: POINTER

Points to an array of type MCA_Parameter_STRUCT which holds the parameter numbers and values.

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

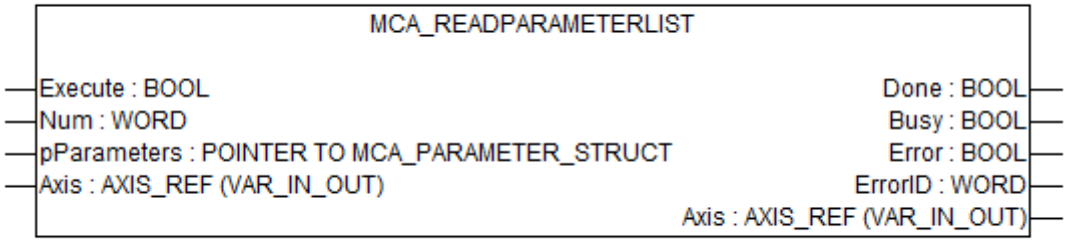


Fig. 454: Function block MCA_ReadParameterList

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block with Busy = TRUE has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

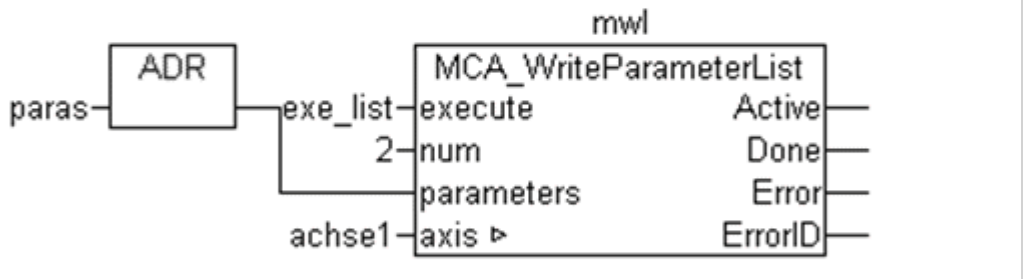
MCA_PARAMETER_STRUCT

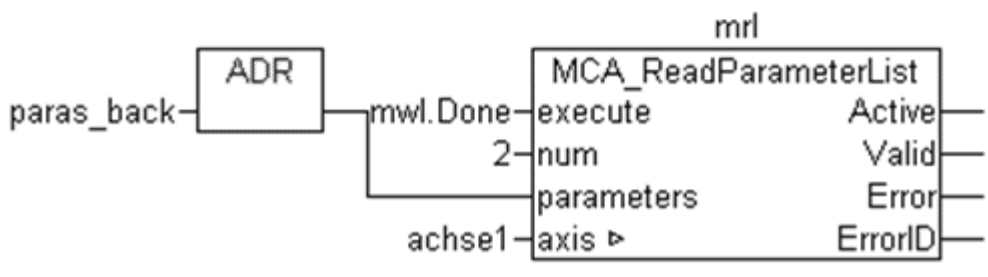
This structure is used to define the parameter list used by MCA_ReadParameterList or MCA_WriteParameterList. An array of this structure has to be created. The address of this array has to be connected to input **Parameters** of MCA_ReadParameterList or MCA_WriteParameterList

MCA_PARAM-ETER_STRUCT	Type	Comment
ParameterNumber	WORD	
Value	DINT	In case of WriteParameterList, **Value** will be written to the Parameter indicated by ParameterNumber. In case of ReadParameterList, the function block will read the Parameter indicated by ParameterNumber and write it to **Value**.

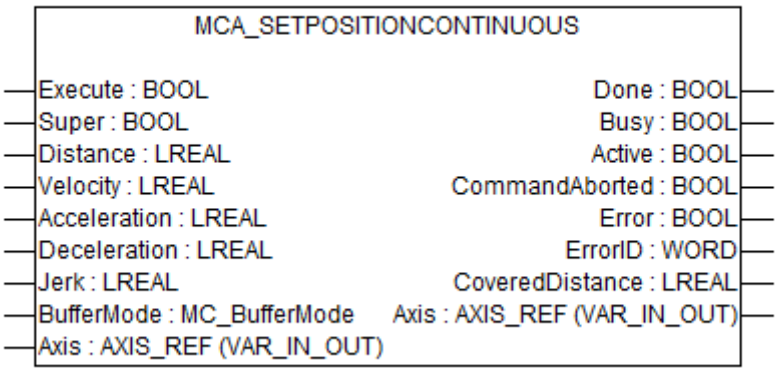
This example write 2 parameters to “achse1” and reads them back afterwards:

```
PROGRAM ParameterReadWrite
VAR
    paras_back:ARRAY[0..1]OF MCA_Parameter_STRUCT:=
        (ParameterNumber:=41001,value:=0),
        (ParameterNumber:=43401,value:=0);
    paras:ARRAY[0..1]OF MCA_Parameter_STRUCT:=
        (ParameterNumber:=41001,value:=3),
        (ParameterNumber:=43401,value:=0);
    mrl: MCA_ReadParameterList;
    mrw:MCA_WriteParameterList;
END_VAR
```





MCA_SetPositionContinuous



This function block shifts the coordinate system of an axis by manipulating either the set-point position or the actual position of an axis. This can be used for instance for a reference situation “on the fly” where no abrupt position change is allowed, eg when a slave axis is linked to the modified axis. This function block can also be used during motion without changing the commanded position, which is now positioned in the shifted coordinate system. A continuous position correction will be achieved, with a defined profile.

- *The function block is allowed in any state except ErrorStop or Homing. In Discrete Motion, just mode SUPER = TRUE is possible.*
- *The block will not change the axis state even when it results in a movement.*
- *With Super = TRUE, the axis will hold the setpoint position while an offset is applied to the actual position. This will result in a movement as the position control loop will keep the distance between setpoint- and actual position constant. A slave axis will not see this movement and will not follow. When the block is ready, the axis will have moved physically by -Distance but the positions in AXIS_REF will not have been changed.*
- *With Super = FALSE, the behavior equals MC_SetPosition, but executed continuously. The axis will physically stay where it is but the actual position and setpoint-position are modified. A slave axis will follow.*
- *With Super = FALSE: When it is acceptable and required to correct the position with a jump, use Acceleration = -1.*

See the following chapter to check if this function block is supported by the used axis implementation: ↳ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↳ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

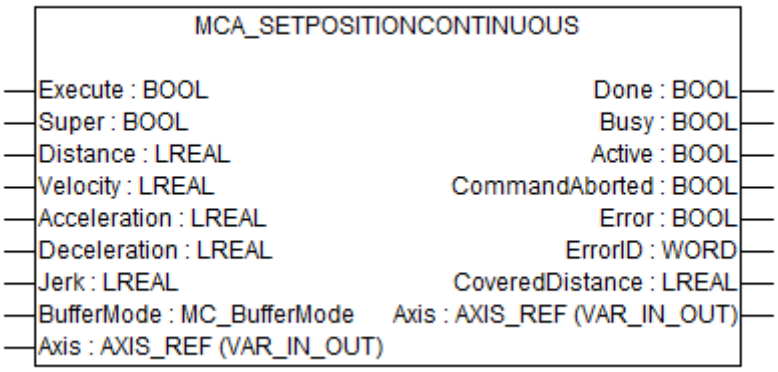


Fig. 455: Function block MCA_SetPositionContinuous

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Super Data type: BOOL
Defines 2 different modes.

Distance Data type: LREAL, unit: u
Relative distance for the motion.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis Data type: AXIS_REF

Reference to the axis.

Output Description

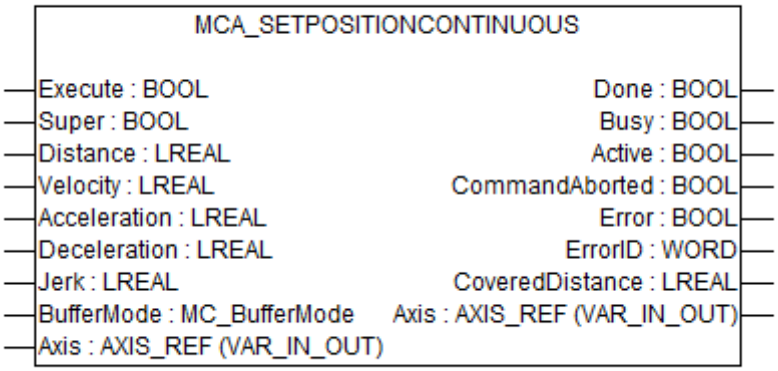


Fig. 456: Function block MCA_SetPositionContinuous

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

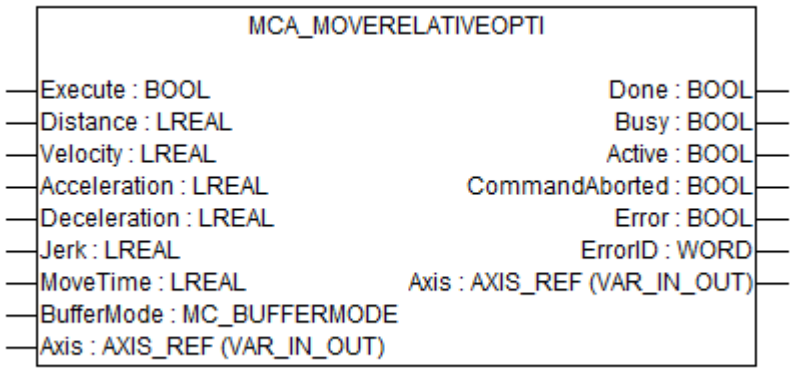
ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

CoveredDistance Data type: LREAL

Shows the progress, starts with 0 and ends with Covered-Distance= Distance. The value is just valid while the function block is active.

MCA_MoveRelativeOpti



This function block is designed to allow an easier setup for positioning movement. The input MoveTime which receives the allowed time to move the given distance. The other inputs, as Velocity, Acceleration, Deceleration and Jerk can be left "0", then the movement will use the system limits and perform an as “soft” as possible positioning.

- It will always use a Jerk (when possible).
- To switch off the usage of Jerk, use Jerk = -1 as input parameter.
- Use the smallest possible acceleration/deceleration.
- Use the smallest possible velocity.

If it is not required to create a movement with limited Jerk, the input Jerk = -1 should be used. The acceleration and deceleration is applied with a "jump" but a smaller maximum value is reached.

If parameters as Velocity, Acceleration and Deceleration are set, these will be considered to be the upper limits during the movement. If it is not possible to execute the movement in the given time, an error will be shown. The function block gives also a suggestion which values could be used to execute the movement within the time limitations.

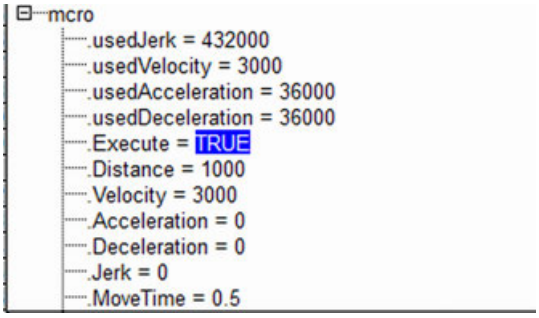
If the velocity was too small, the internal variable usedVelocity will hold the smallest possible value to execute the movement in the given time. This would mean execute it without any ramps. usedAcceleration and usedDeceleration will be "0" in this case.

If the velocity was ok, but acceleration and deceleration too small, usedVelocity will show the value from input Velocity and variables usedAcceleration and usedDeceleration will hold the necessary values to execute the movement in the given time.

With move_Time = 0, the behavior for the function block is identical with MC_MoveRelative ↬ Chapter 1.5.9.6.1.2 “MC_MoveRelative” on page 2751.

It is possible to use just Distance and MoveTime as input parameters. In this case, the function block will take the axis configuration parameters as limitations for the movement and always create the smoothest possible interpolation.

If parameters are given which are not possible to use, e.g. velocity is too small to reach the target in the given time, the function block´s internal variables usedJerk, usedVelocity, usedAcceleration and usedDeceleration will show a possible solution.

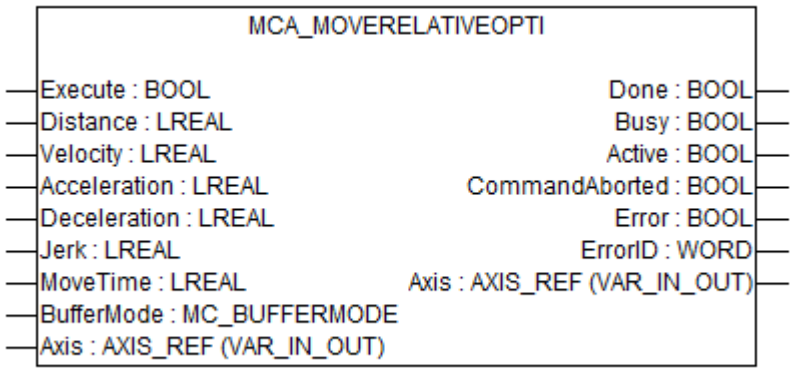


See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 "Overview of PLCopen function blocks" on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 "Overview of data types" on page 2585*

Input Description



Execute Data type: BOOL

Starts the function block at rising edge.

Distance Data type: LREAL, unit: u

Relative distance for the motion.

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

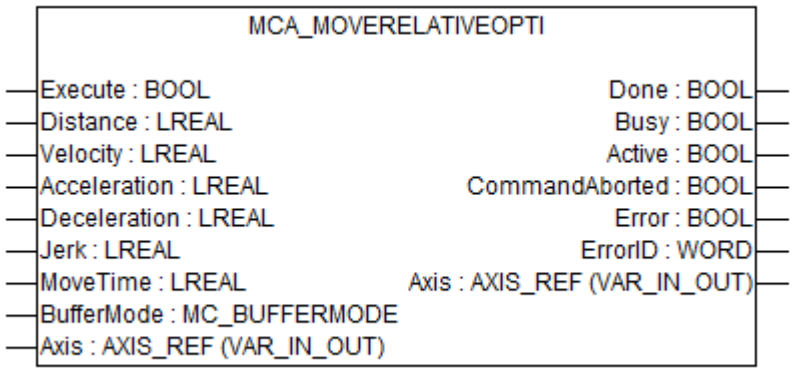
Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

MoveTime
Data type: LREAL, range: > 0, unit: s
Time to be used for the movement.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

Axis
Data type: AXIS_REF
Reference to the axis.

Output Description



Done
Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy
Data type: BOOL
The function block is not finished.

Active
Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted
Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error
Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_GearInDirect

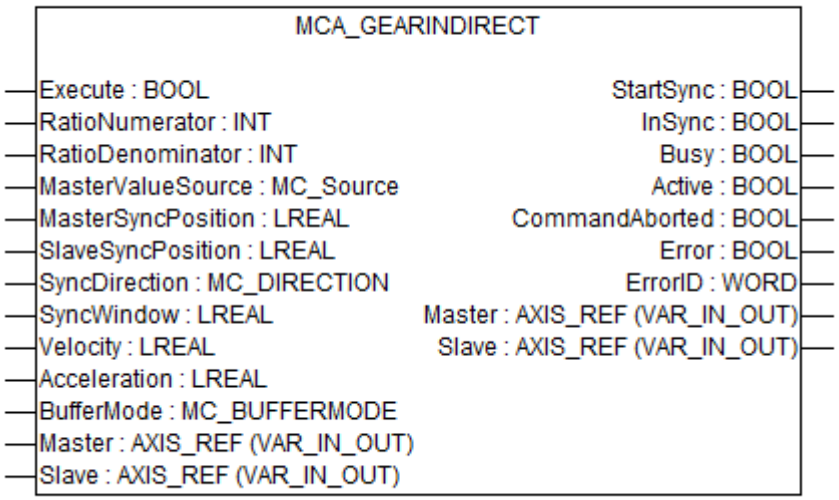


Fig. 457: MCA_GearInDirect

This function block commands a gear ratio between the position of the slave and master axes from the synchronization point onwards.

The function block behaves as follows:

- Synchronization starts right away, no matter if the master moves or is in standstill.
- The synchronization is limited by the given velocity and acceleration, and achieved as fast as possible, so it can happen that:
  - The 2 axes are synchronized earlier then the 2 given positions
  - The 2 axes are synchronized later then the 2 given positions
- Following formula is used:
  - slavePosition = (masterPosition-MasterSyncPosition)*RatioNumerator/RatioDenumerator+SlaveSyncPosition
- In a modulo-axis, it is possible to reach the synchronization point in different directions.
  The input parameter SyncDirection with its possible values: POSITIVE, NEGATIVE or SHORTEST can be used to set this direction.
  Inside the SyncWindow, automatically the direction SHORTEST will be used. It is important to set a SyncWindow>0 for a modulo axis, because otherwise, slightest deviations could result in moving a complete modulo distance.
  - Inside SyncWindow, the slave axis will move SHORTEST to reach the SlaveSyncPosition
  - Outside SyncWIndow, it will move the given SyncDirection, which can be POSITIVE or NEGATIVE
- If a direction POSITIVE or NEGATIVE is used in a linear axis, the slave will wait until the master reaches a position which allows the slave to move the required direction.

Input Description

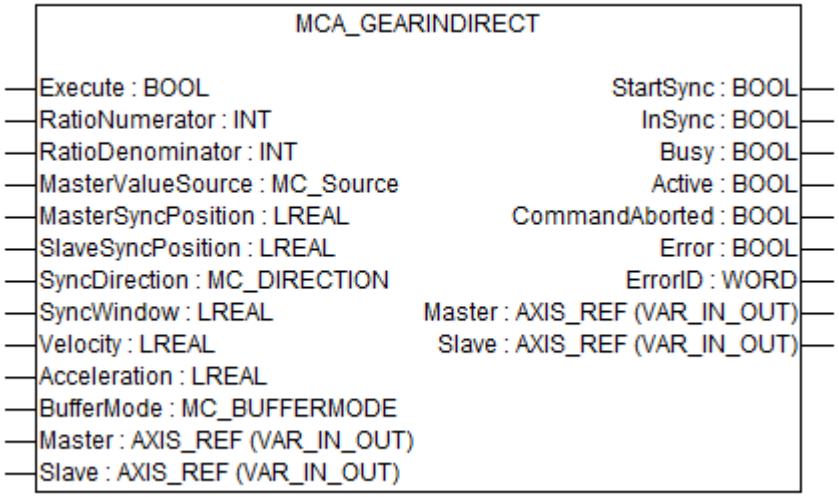


Fig. 458: MCA_GearInDirect

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

RatioNumerator
Data type: INT
Gear Ratio Numerator.

RatioDenominator
Data type: INT
Gear Ratio Denominator.

MasterValueSource
Data type: MC_SOURCE
Defines the source for synchronization:
mcSetValue - Synchronization on master set value.
mcActualValue - Synchronization on master actual value.

MasterSyncPosition
Data type: LREAL
The position of the master where the slave is insync with the master.

SlaveSyncPosition
Data type: LREAL
Slave Position at which the axes are running in sync.

SyncDirection
Data type: MC_DIRECTION

Moving direction for the slave to start the movement.

POSITIVE, NEGATIVE or SHORTEST are applicable.

SyncWindow Data type: LREAL

- when the slave is outside the SyncWindow, it will move the direction which is given in SyncDirection
- when the slave is inside the SyncWindow, it will move SHORTEST to meet the SlaveSync-Position.

Velocity Data type: LREAL

Velocity which limits the synchronization movement.

The slave has to be able to move faster than the master axis, otherwise it is possible the SlaveSyncPosition is never reached when the master starts to move.

Acceleration Data type: LREAL

Acceleration which limits the synchronization movement.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Master Data type: AXIS_REF

Reference to the master axis.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output Description

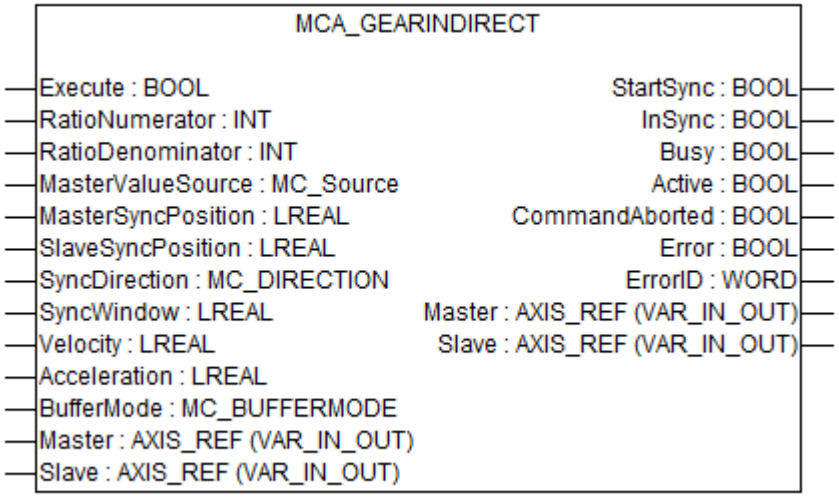


Fig. 459: MCA_GearInDirect

StartSync Data type: BOOL
Commanded gearing starts.

InSync Data type: BOOL
Commanded gearing completed.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_CamInDirect

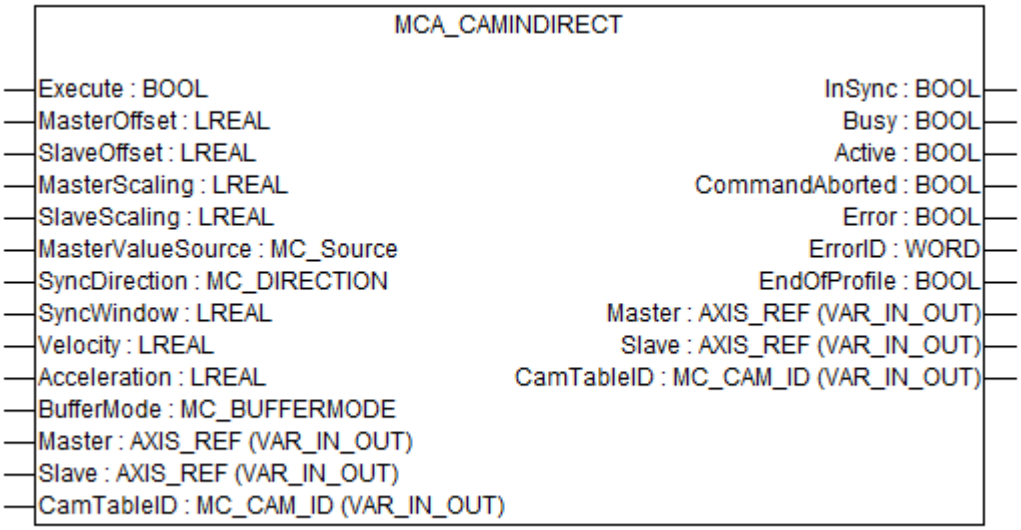


Fig. 460: MCA_CamInDirect

This function block engages the Cam.

- *It is not required that the master is stationary.*
- *If the actual master and slave positions do not correspond to the offset values when MC_CamIn is executed, either an error occurs or the system deals with the difference automatically.*
- *The Cam is placed either absolute or relative to the current master and slave positions.*

  Absolute: The profile between master and slave is seen as an absolute relationship.

  Relative: The relationship between master and slave is in a relative mode.
- *This function block is not merged with the MC_CamTableSelect function block because this separation enables changes on the fly.*
- *A mechanical analogy to a slave offset is a cam welded with additional constant layer thickness. Because of this the slave positions have a constant offset and the offset could be interpreted as axis offset of the master shaft, if linear guided slave tappets are assumed.*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

The function block behaves as follows:

- If the master is inside the position range which is described in the cam-table data, synchronization starts right away, no matter if the master moves or is in standstill.
- If the master is outside the position range which is described in the cam-table data, the slave position is not modifed.

- The synchronization is limited by the given Velocity and Acceleration, and achieved as fast as possible.
  The function block will show InSync when synchronization process is completed and the slave axes reference position matches the cam-table data for the current master position.
- In a modulo-axis, it is possible to reach the synchronization point in different directions.
  The input parameter SyncDirection with its possible values: POSITIVE, NEGATIVE or SHORTEST can be used to set this direction.
  Inside the SyncWindow, automatically the direction SHORTEST will be used.
  It is important to set a SyncWindow>0 for a modulo axis, because otherwise, slightest deviations could result in moving a complete modulo distance.
  – Inside SyncWindow, the slave axis will move SHORTEST to reach the SlaveSyncPosition
  – Outside SyncWIndow, it will move the given SyncDirection, which can be POSITIVE or NEGATIVE
- If a direction POSITIVE or NEGATIVE is used in a linear axis, the slave will wait until the master reaches a position which allows the slave to move the required direction.

> *The MC_CamIn has parameters to scale the cam-table values (MasterScaling, SlaveScaling).*
>
> *It has to be considered that MasterOffset and SlaveOffset are scaled exactly like the corresponding cam-table values.*

Input Description

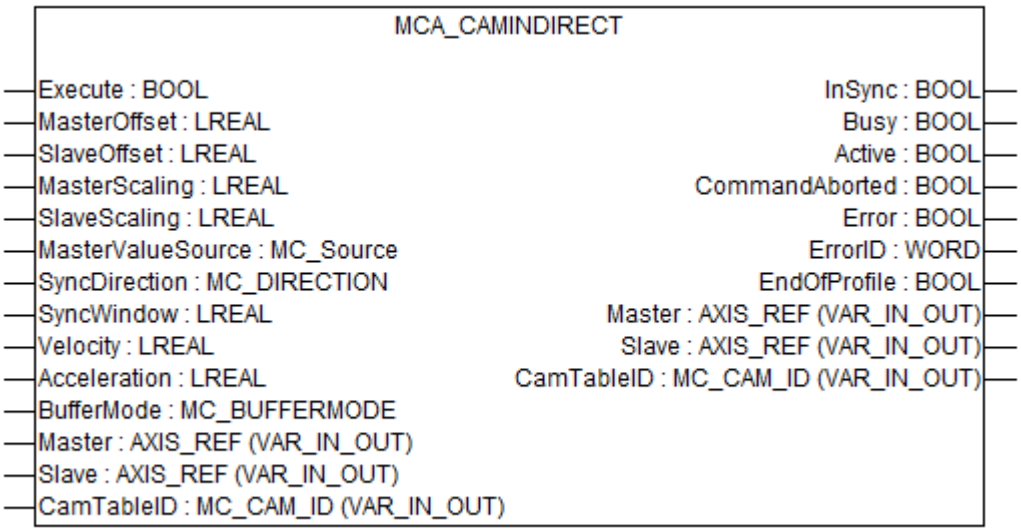


Fig. 461: MCA_CamInDirect

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

MasterOffset Data type: LREAL

Offset of master table. Angular offset of the master shaft to cam.

SlaveOffset Data type: LREAL, default: 0

Offset of slave table. Sharpened cam (i.e higher elevation and deeper depression).

MasterScaling Data type: LREAL, default: 1.0

Factor for the master profile. From the slave point of view the master overall profile is multiplied by this factor.

SlaveScaling Data type: LREAL, default: 1.0

Factor for the slave profile. The overall slave profile is multiplied by this factor.

MasterValue-Source Data type: MC_SOURCE

Defines the source for synchronization:

mcSetValue - Synchronization on master set value.

mcActualValue - Synchronization on master actual value.

SyncDirection Data type: MC_DIRECTION

Moving direction for the slave to start the movement.

POSITIVE, NEGATIVE or SHORTEST are applicable.

SyncWindow Data type: LREAL

- when the slave is outside the SyncWindow, it will move the direction which is given in SyncDirection
- when the slave is inside the SyncWindow, it will move SHORTEST to meet the SlaveSync-Position.

Velocity Data type: LREAL

Velocity which limits the synchronization movement.

The slave has to be able to move faster than the master axis, otherwise it is possible the SlaveSyncPosition is never reached when the master starts to move.

Acceleration Data type: LREAL

Acceleration which limits the synchronization movement.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Master Data type: AXIS_REF

Reference to the master axis.

Slave Data type: AXIS_REF
Reference to the slave axis.

CamTableID Data type: MC_CAM_ID
Identifier of CAM Table to be used in the MC_CamIn function block.

Output Description

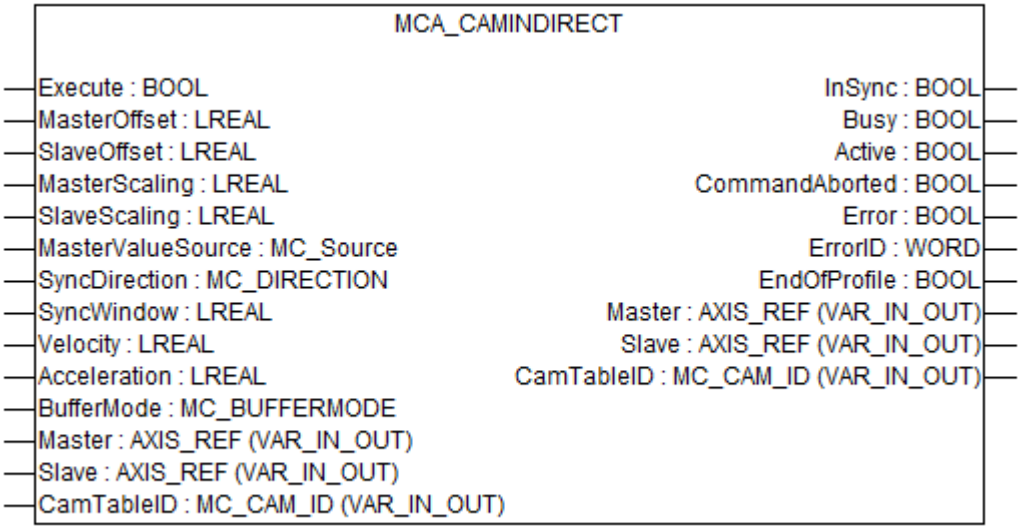


Fig. 462: MCA_CamInDirect

InSync Data type: BOOL
Cam is engaged for the first time.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandA-borted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

EndOfProfile Data type: BOOL

Pulsed output signaling the cyclic end of the CAM Profile.

MCA_SetOperatingMode

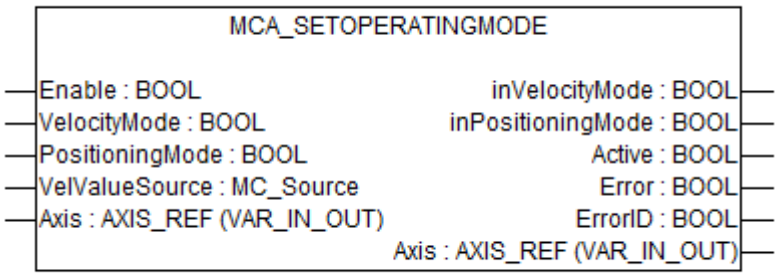


Fig. 463: Function block MCA_SetOperatingMode

This function block changes the axis mode from positioning to velocity mode and vice versa.

By default, an axis is always a positioning axis which has to follow either the drives or the PLCs position control loop. In some applications, the movement is limited (e.g. by torque restrictions) so the position can´t be reached.

A position controlled axis would first speed up, and then create a following error, both caused by the increasing position lag.

The function block MCA_SetOperatingMode can be used to prevent this behavior and will switch between velocity- and position controlled behavior “on the fly”.

> *To use the function block MCA_SetOperatingMode, the drive has to be used in CSV (ContinuousSyncronuousVelocity), or an analog drive has to be used, which means it has to move controlled by SPEED_REFERENCE (Kernel).*

If the function block is called in StandStill, ErrorStop or Disabled, it will be effective immediately.

In any other mode, it will be effective with the next “Execute” rising edge on a function block which activates a movement. It can be called while the axis is moving and will create a bump-less transition between the velocity- and position controlled mode.

In velocity mode:

- The SPEED_REFERENCE is created by feed-forward, in an open loop. (It is not required to set FF_PERCENTAGE parameter) REFERENCE_POSITION will follow the DRIVE_ACTUAL_POSITION Position following error is not supervised.
- REFERENCE_POSITION will follow the DRIVE_ACTUAL_POSITION.
- Position following error is not supervised.

See the following chapter to check if this function block is supported by the used axis implementation: ↬ *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

For this function block there is a visualization in the Library MC_Blocks_AC500_V11.

See the following chapter for a list of available data types: ↬ *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input Description

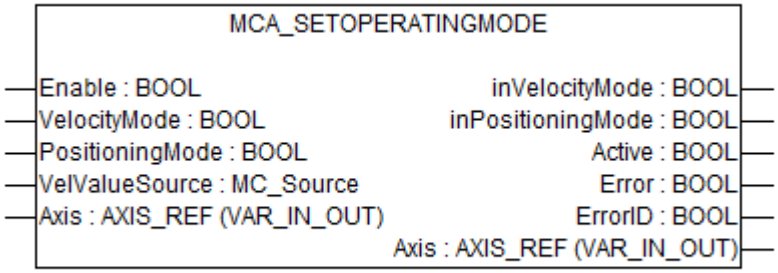


Fig. 464: Function block MCA_SetOperatingMode

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable

Data type: BOOL

Enables the function block to switch the operating mode. A rising edge is not required, the block will be operate if Enable=TRUE and will react to the VelocitiyMode/PositioningMode inputs.

VelocityMode=TRUE/PositioningMode=FALSE=> switch axis to velocity mode.

VelocityMode=FALSE/PositioningMode=TRUE=> switch axis to positioning mode.

VelocityMode=PositioningMode => no change.

VelocityMode

Data type: BOOL, default: FALSE

Demands the axis to be set in VelocityMode.

Positioning-Mode

Data type: BOOL, default: FALSE

Demands the axis to be set in PositioningMode.

Axis

Data type: AXIS_REF

Reference to the axis.

Output Description

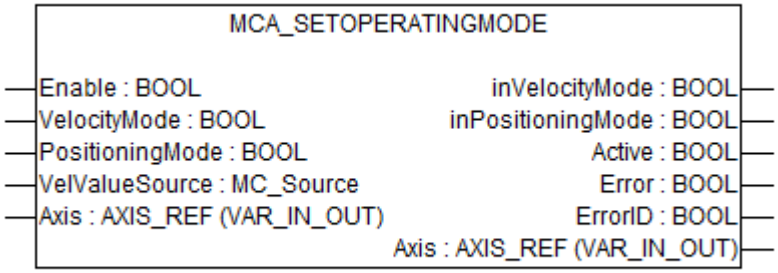


Fig. 465: Function block MCA_SetOperatingMode

InVelocityMode Data type: BOOL

Shows the axis state.

InPositioning-Mode Data type: BOOL

Shows the axis state.

Active Data type: BOOL

Indicates that the function block has control on the axis.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_PhasingByMaster

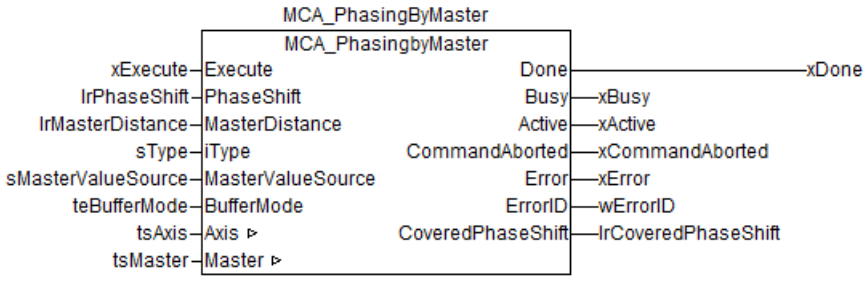


Fig. 466: Function block MCA_PhasingByMaster

This function block performs a movement for the relation to the master axis of the specified axis. A real movement is just performed in case the axis is in synchronized motion.

This function block creates a relative phase shift in the master position of a slave axis. The master position is shifted in relation to the real physical position. This is analogous to opening a coupling on the master shaft for a moment and is used to delay or advance an axis to its master. The phase shift is seen from the slave. The master does not know that there is a phase shift experienced by the slave. The phase shift remains, until another “Phasing” command changes it again.

> *The phasing is executed with respect to a master movement and will use a polynomial interpolation*

See the following chapter to check if this function block is supported by the used axis implementation: *Chapter 1.5.9.2.4 “Overview of PLCopen function blocks” on page 2577*

See the following chapter for a list of available data types: *Chapter 1.5.9.2.6 “Overview of data types” on page 2585*

Input description

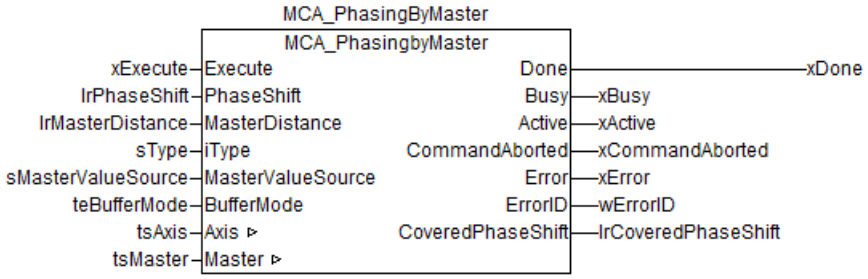


Fig. 467: Function block MCA_PhasingByMaster

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge

Phaseshift Data type: LREAL

[u] = Technical unit, phase difference in master

MasterDistance Data type: LREAL

[u] Distance master has to move

iType Data type: MC_ABB_iTypes_Enum

Interpolationtype, possible values are:

- MCA_SPLINE_COMPLETE
- MCA_SPLINE_NATURAL
- MCA_POLY5

- MCA_POLY3
- MCA_LINEAR

MasterValue-Source
Data type: MC_Source
Decide to use the actual position or reference position of master axis

BufferMode
Data type: MC_BufferMode
Not supported, default mcABORTING used

Axis
Data type: Axis_Ref
Reference to axis

Master
Data type: Axis_Ref
Reference to master axis

Output description

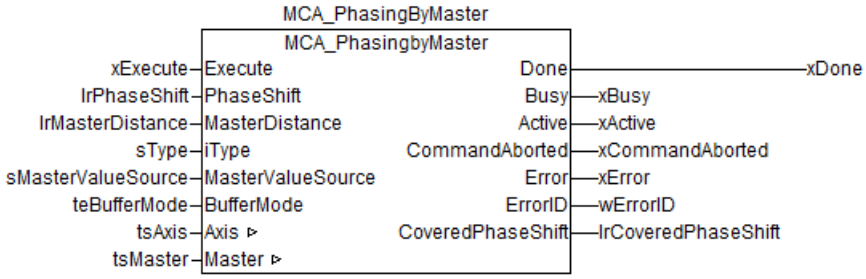


Fig. 468: Function block MCA_PhasingByMaster

Done
Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished

Busy
Data type: BOOL
The function block is not finished

Active
Data type: BOOL
Indicates that the function block has control on the axis

CommandA-borted
Data type: BOOL
Command is aborted by another command from other PLCopen function block

Error
Data type: BOOL
Signals that error has occurred within function block

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*

CoveredPhase-Shift Data type: LREAL

Actual phase shift of master axis to slave axis, valid while function block is busy

1.5.9.7 PLCopen function blocks (Coordinated motion control)

1.5.9.7.1 Standard function blocks

MC_GroupEnable

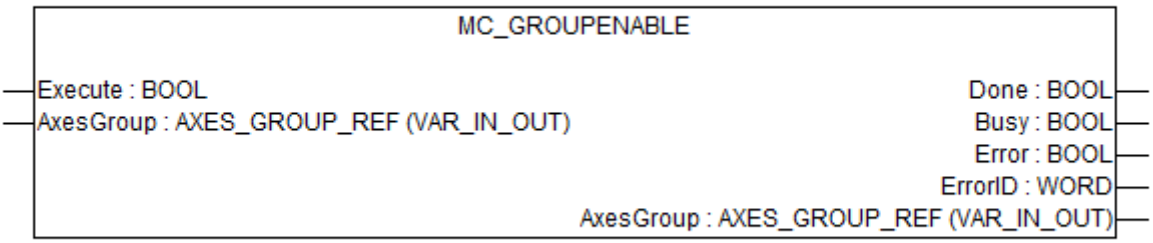


Fig. 469: MC_GroupEnable

This function block changes the state for a group from GroupDisabled to GroupStandby. This is an administrative function block, since no movement is generated.

> *The command does not influence the power state of any of the single axes in the group.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

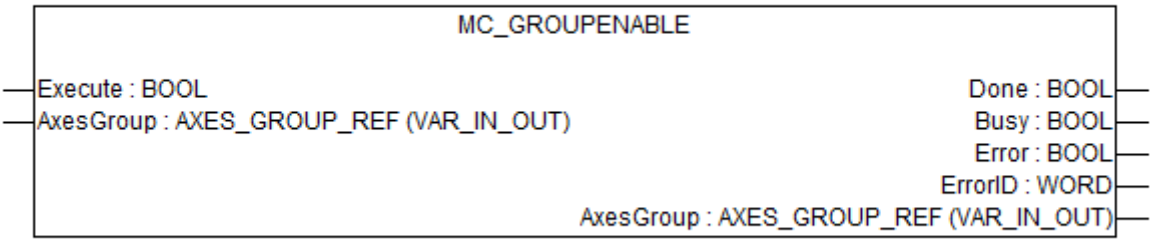


Fig. 470: MC_GroupEnable

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

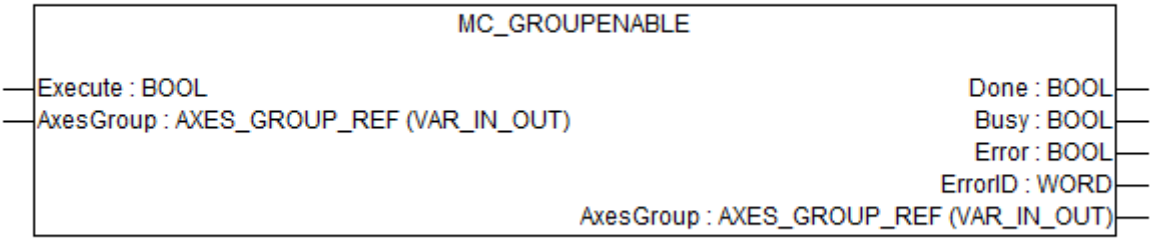


Fig. 471: MC_GroupEnable

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupDisable

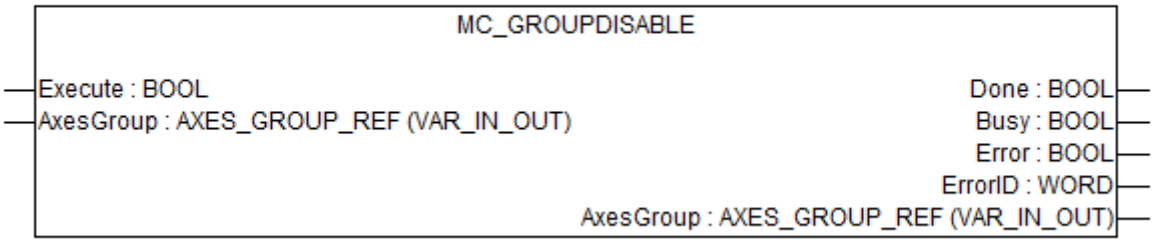


Fig. 472: MC_GroupDisable

This function block changes the state for a group to GroupDisabled, although it is an administrative FB, since no movement is generated. If the axes are not standing still while issuing this command, it is up to the application to take the necessary precautions.

> *The command does not influence the power state of any of the single axes in the group.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

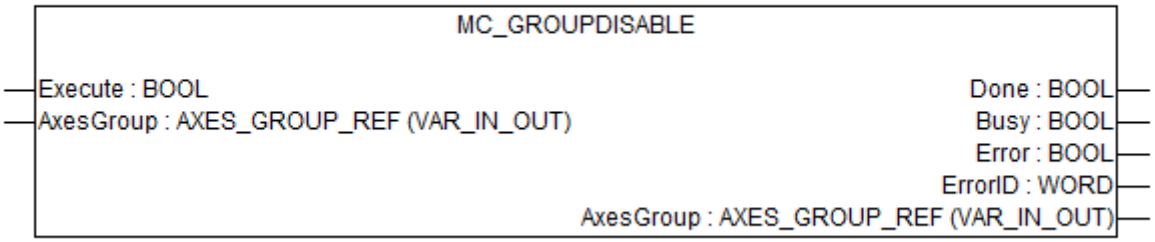


Fig. 473: MC_GroupDisable

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

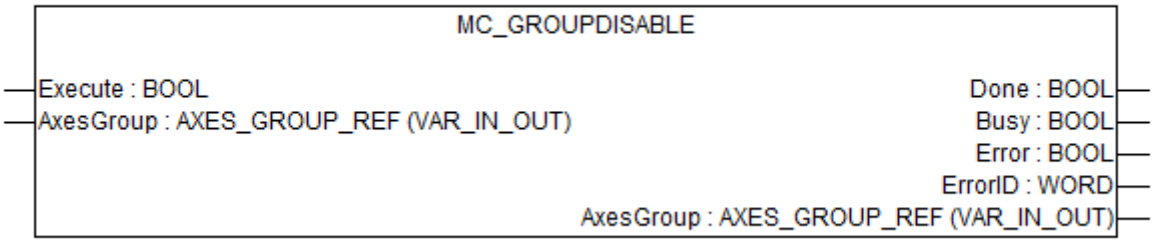


Fig. 474: MC_GroupDisable

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupReadActualPosition

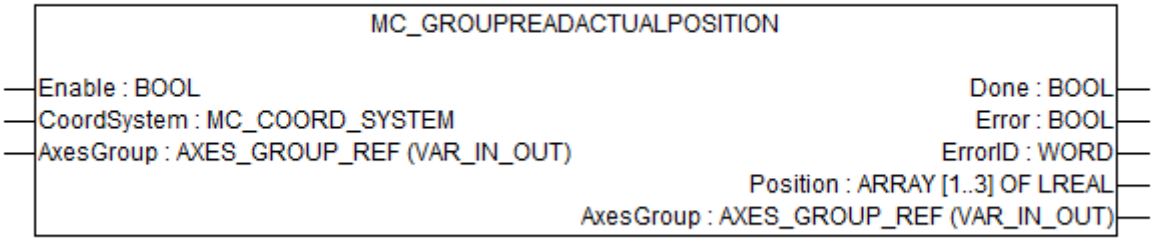


Fig. 475: Function block MC_GroupReadActualPosition

This function block returns the actual position in the selected coordinate system of an axes group. This is an administrative function block, since no movement is generated.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

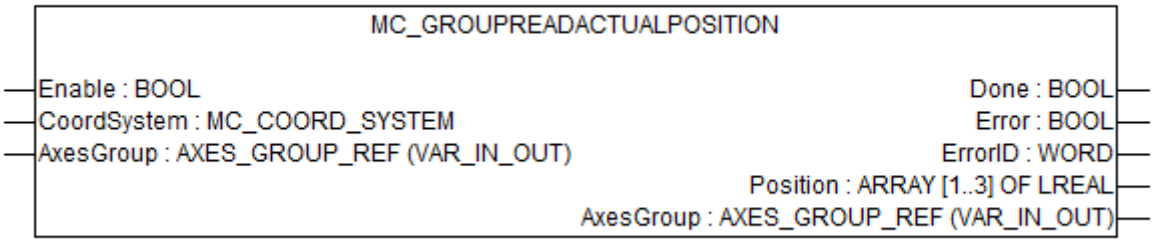


Fig. 476: Function block MC_GroupReadActualPosition

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the actual position in the selected coordinate system of the axes group continuously while enabled.

CoordSystem Data type: MC_COORD_SYSTEM; range: MC_DEFAULT_COORD, MC_MCS_COORD, MC_PCS_COORD

Reference to the coordinate system (MCS, PCS).

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

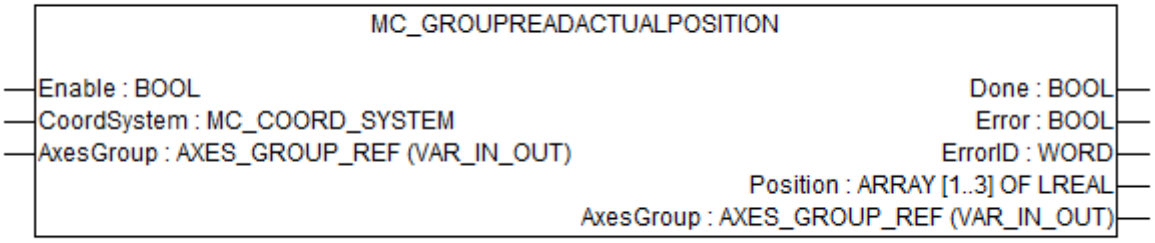


Fig. 477: Function block MC_GroupReadActualPosition

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Position Data type: ARRAY [1..3] OF REAL

Current position of the group.

MC_GroupReadActualVelocity

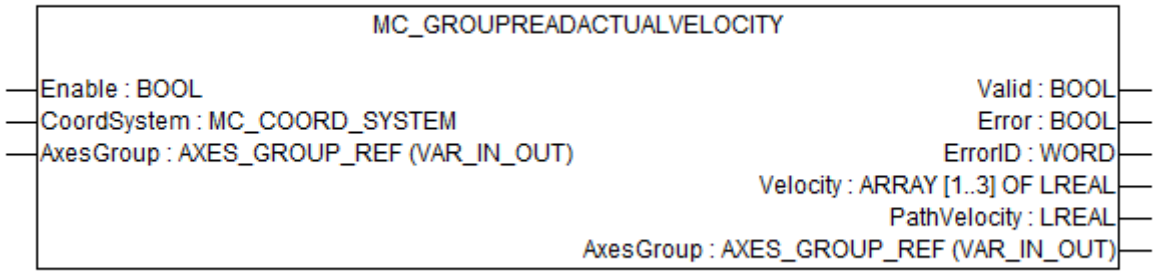


Fig. 478: MC_GroupReadActualVelocity

This function block returns the actual velocity in the selected coordinate system of an axes group. This is an ad-ministrative function block, since no movement is generated.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

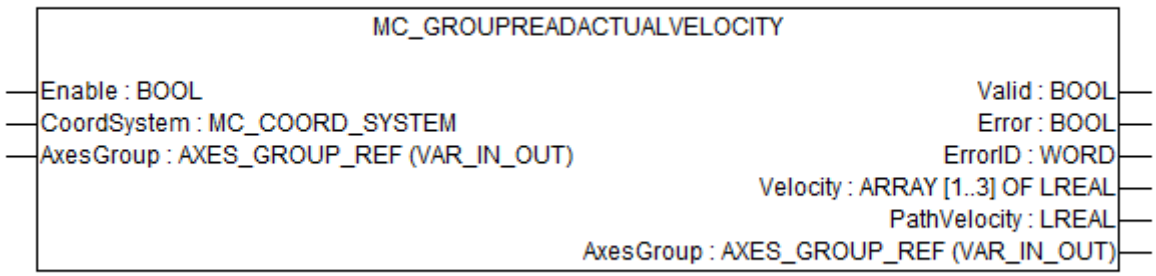


Fig. 479: MC_GroupReadActualVelocity

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the actual position in the selected coordinate system of the axes group continuously while enabled.

CoordSystem Data type: MC_COORD_SYSTEM; range: MC_DEFAULT_COORD, MC_MCS_COORD, MC_PCS_COORD

Reference to the coordinate system (MCS, PCS).

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

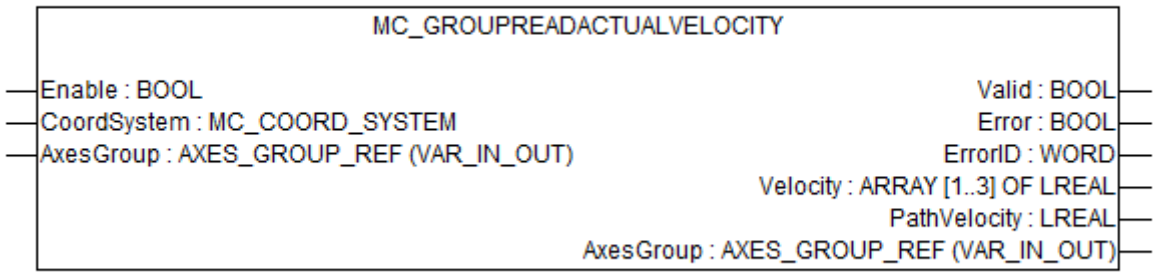


Fig. 480: MC_GroupReadActualVelocity

Valid Data type: BOOL

True if valid outputs are available.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Velocity Data type: ARRAY [1..3] OF REAL

Current velocity of the group:

- in ACS the velocities of the different axes,
- in MCS and PCS it provides the velocity of the TCP .

PathVelocity Data type: REAL

Current path velocity (speed, combined result) of the TCP.

MC_GroupStop

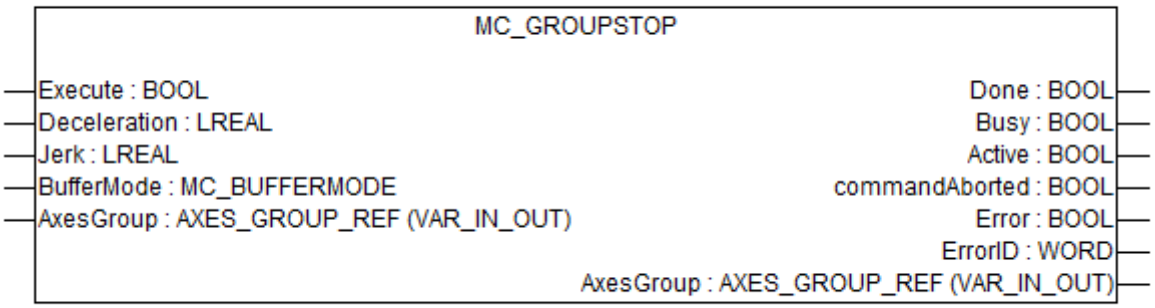


Fig. 481: MC_GroupStop

This function block commands a controlled motion stop and transfers the axes group to the state GroupStopping. It aborts any ongoing function block execution. While the axes group is in state GroupStopping, no other function block can perform any motion on the same axes group. After the axes group has reached velocity zero, the Done output is set to TRUE immediately. The axes group remains in the state GroupStopping as long as Execute is still TRUE or velocity zero is not yet reached. As soon as Done is SET and Execute is FALSE the axes group goes to state GroupStandby. The command can only be aborted by MC_GroupDisable.

- *The relevant axes stay on the path.*
- *If Deceleration is set to zero, the minimum value = 1 is used instead.*
- *If issued during a MoveDirectXxx command, the velocity/acc-/deceleration/jerk values as properties of the AxisRef of each axis are used, and not specified within this function block, and not to be exceeded during the movement.*
- *Any synchronization of the group to a master is cancelled by issuing MC_GroupStop*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Timing diagram

A typical timing diagram for MC_GroupStop is shown below, including the relevant states and state-transitions:

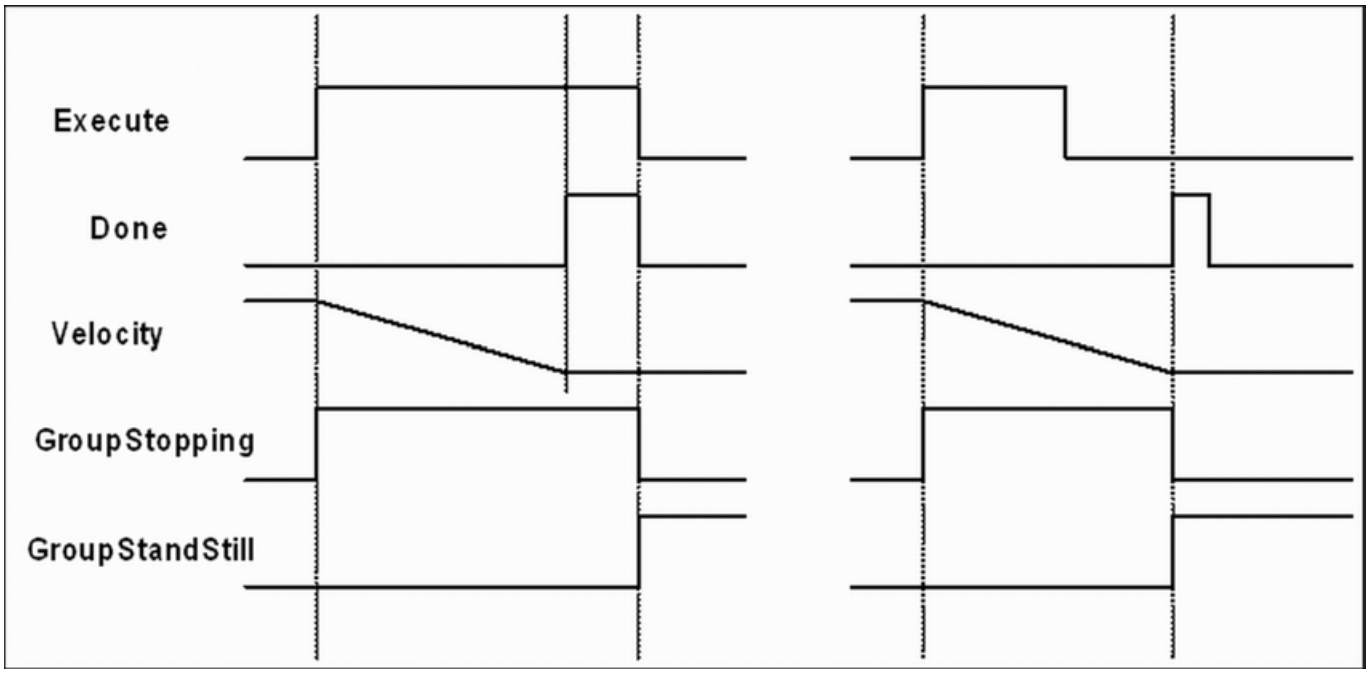


Fig. 482: MC_GroupStop timing diagram

The example below shows the behavior in combination with a MC_MoveLinearRelative:

- An axes group in linear movement is ramped down with function blocks MC_GroupStop. The group stops on the original path,
- The axes group rejects motion commands as long as MC_GroupStop parameter “Execute” = TRUE. Function block MC_MoveLinearRelative reports an error indicating the busy MC_GroupStop command. This error is an function block error, so the group is not moving to the state GroupErrorStop,
- At the 3rd “Exe1” rising edge, the group starts the next movement.

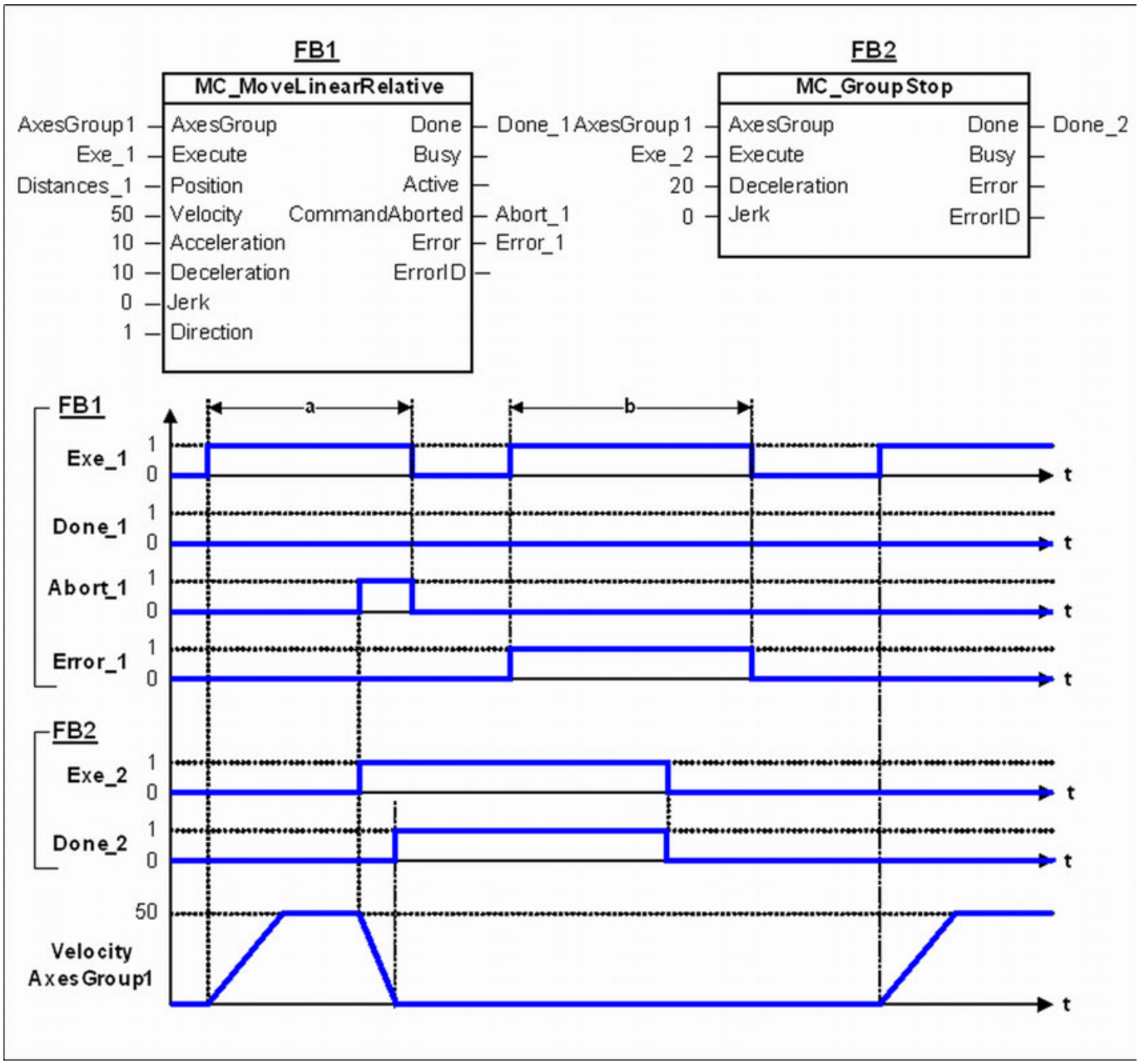


Fig. 483: Behavior of MC_GroupStop in combination with MC_MoveLinearRelative

The following example demonstrates the behavior of MC_GroupStop in combination with two MC_MoveLinearAbsolute which are blended with defined constant path velocity:

Time	Description
t_0	Two function blocks MC_MoveLinearAbsolute are commanded on axes group MyAxesGroup. The first function block becomes active immediately and MyAxesGroup starts to move from its actual position (20.0; 20.0) towards the first target position.
t_1	Shortly after the TCP has started to move on the blending contour blending Lin1 into Lin2, a FB MC_GroupStop is issued in buffermode Aborting. The state of the axes group changes from GroupMoving to GroupStopping. MyAxesGroup decelerates, following the path which would have been executed without having issued MC_GroupStop. Though the path velocity of MyAxesGroup decreases strictly monotonic while stopping, single axes of the group might accelerate in between due to the given path and kinematic transformation of MyAxesGroup.
t_2	MyAxesGroup comes to standstill. The Done output of the function block MC_GroupStop is set. Since the input Execute of the function block Stop is still set the group stays in state GroupStopping.
t_3	The input Execute of the function block Stop is reset. All outputs of the function block Stop are reset. The group state changes to GroupStandby.

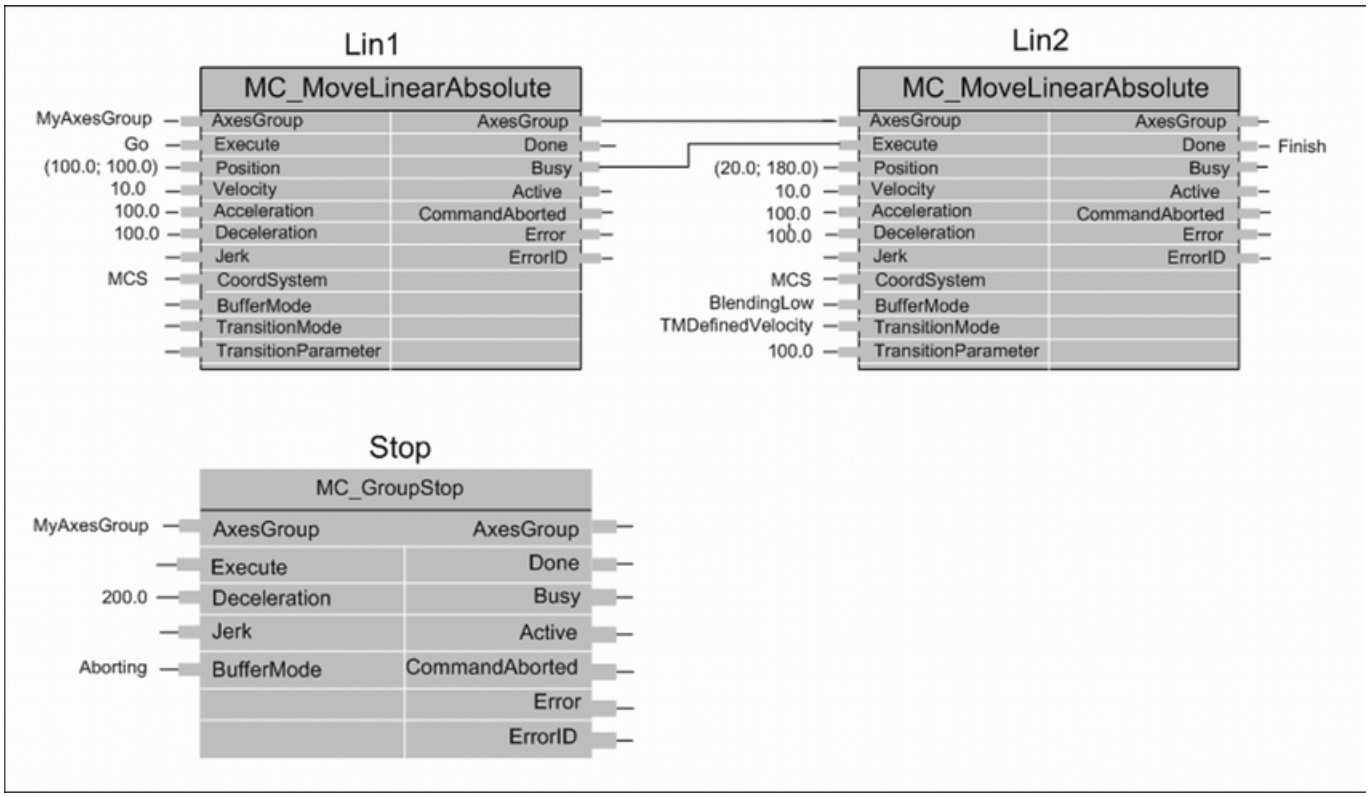


Fig. 484: Example of MC_GroupStop in combination with two MC_MoveLinearAbsolute

Input description

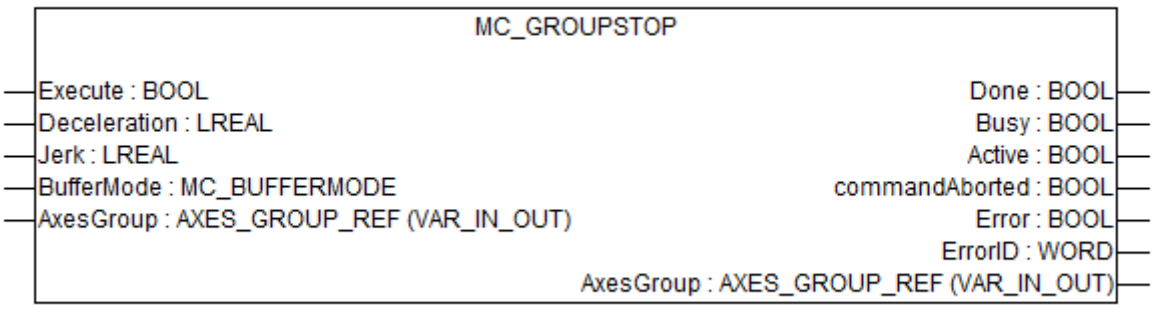


Fig. 485: MC_GroupStop

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

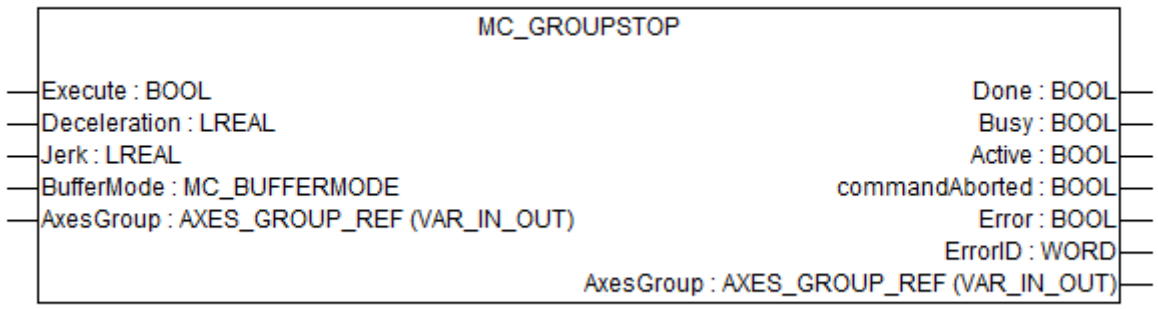


Fig. 486: MC_GroupStop

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by disabling MC_Power of one or more of the axes in the group. The state changes to GroupDisabled.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupHalt

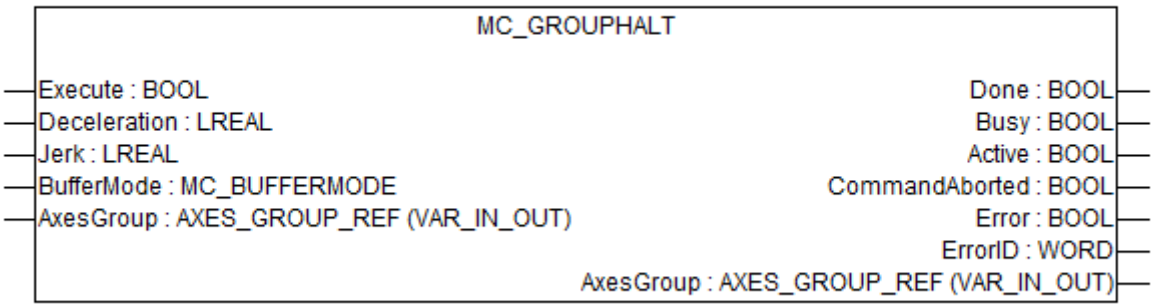


Fig. 487: MC_GroupHalt

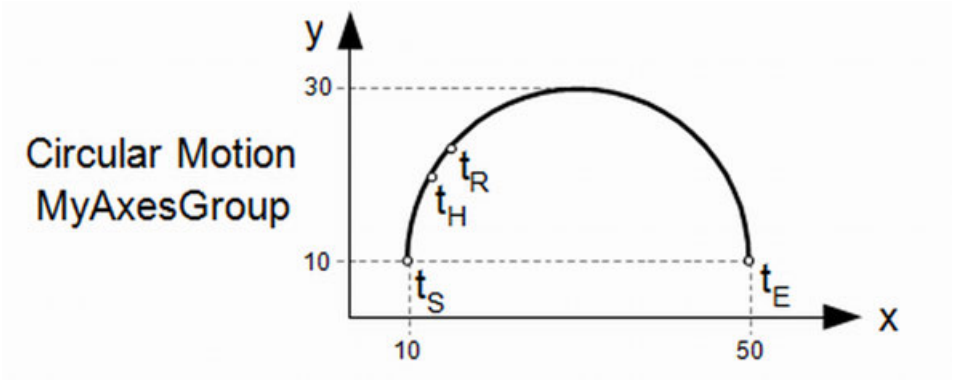
This function block commands a controlled motion stop. It aborts any ongoing function block execution. Ax-esGroup is moved to the state GroupMoving, until the velocity is zero. With the DONE output set, the state is transferred to GroupStandby.

- *MC_GroupHalt is used to stop the axes group under normal operation conditions. In non-buffered mode: during deceleration of the axes group it is possible to set another motion command, which will abort the MC_GroupHalt and will be executed immediately,*
- *If this command is active the next command can be issued. For example a driverless vehicle detects an obstacle and needs to stop. MC_GroupHalt is issued. Before the StandStill is reached the obstacle is removed and the motion can be continued by setting another motion command, so the vehicle does not stop,*
- *The relevant axes stay on the same path which would have been executed without having issued MC_GroupHalt.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

The following example shows the behavior of MC_GroupHalt in combination with a MC_MoveCircularAbsolute:	MyAxesGroup starts at Position (10.0; 10.0; 0.0). A FB MC_MoveCircularAbsolute is commanded with aux-iliary position (30.0; 30.0; 0.0) and end position (50.0; 10.0; 0.0). This results in a 180° circular motion within the xy-plane of any coordinate system.

Point of Time	Description
t_h	The circular motion is aborted by FB MC_GroupHalt. MyAxesGroup stays on the path during halt.
t_r	R) MC_MoveCircularAbsolute is executed again and aborts MC_GroupHalt. MC_GroupHalt allows this, in con-trast to MC_GroupStop. AxesGroup can accelerate again without reaching StandStill. (MC_MoveCircularAbsolute can be retriggered in order to continue the original circular motion as long as MyAxesGroup didn't pass the auxiliary position.)
t_e	E) MyAxesGroup reaches end position (50.0; 10.0; 0.0)



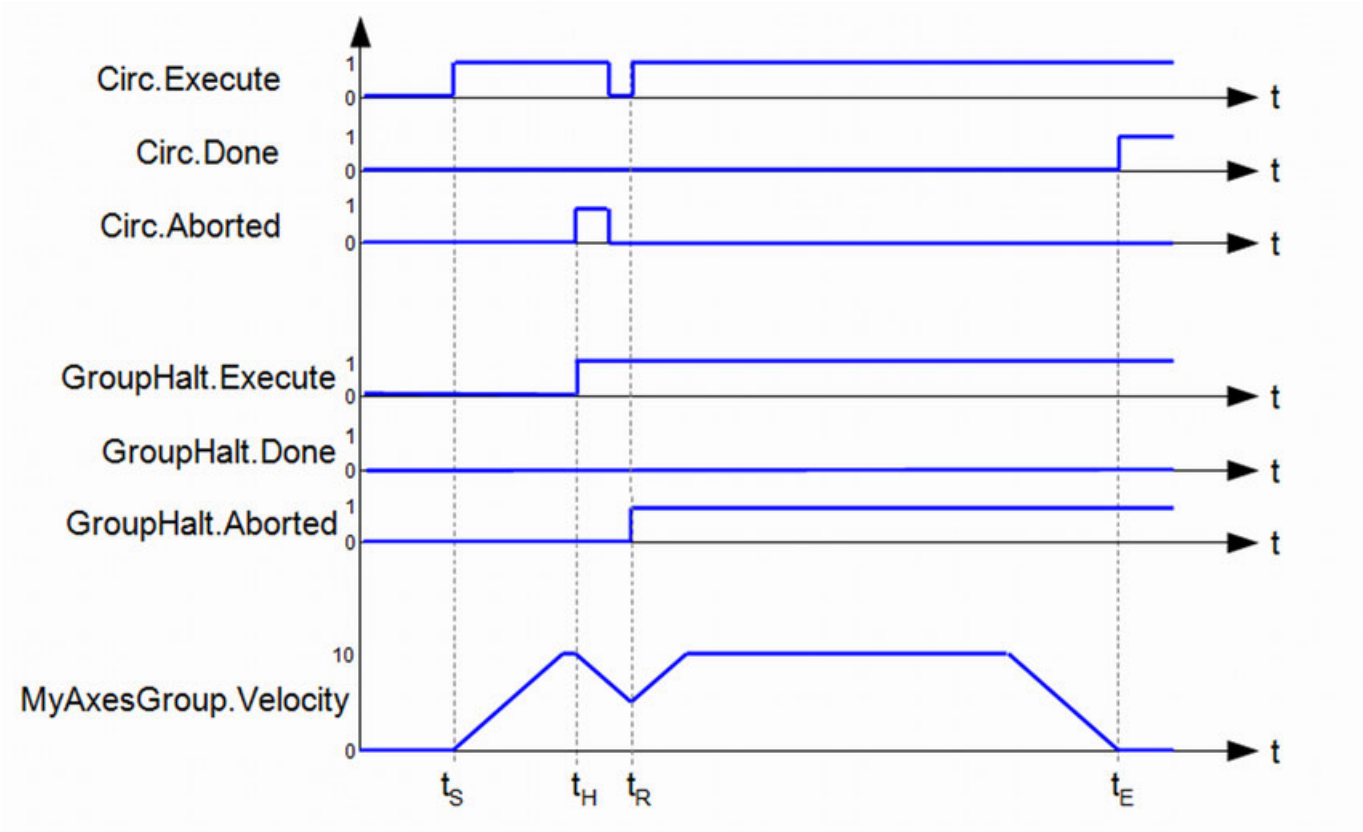
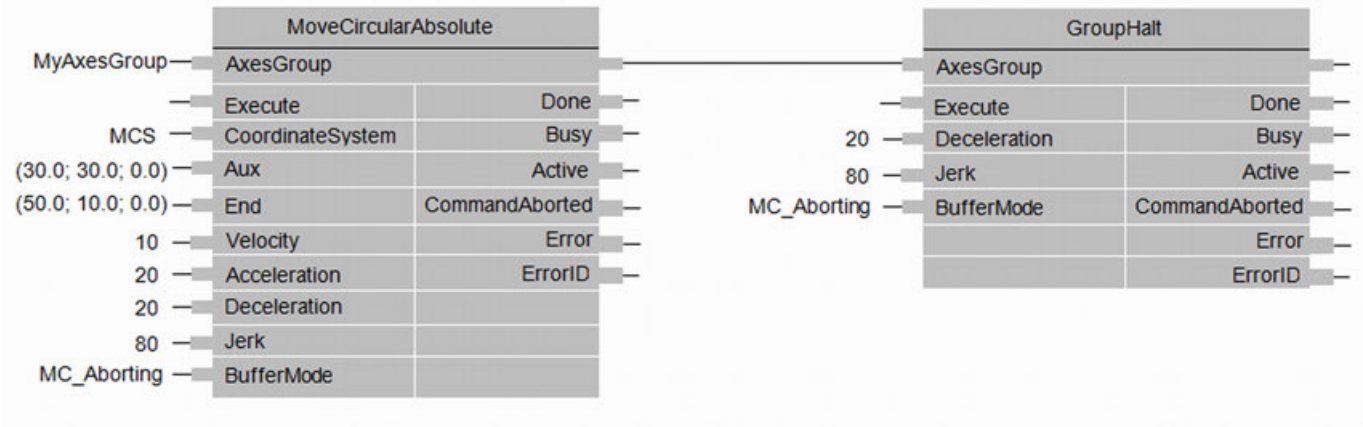


Fig. 488: Behavior of MC_GroupHalt in combination with MC_MoveCircularAbsolute



Input description

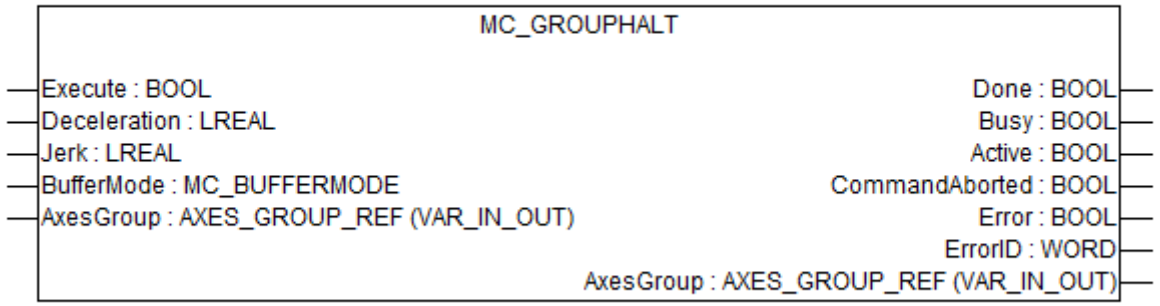


Fig. 489: MC_GroupHalt

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

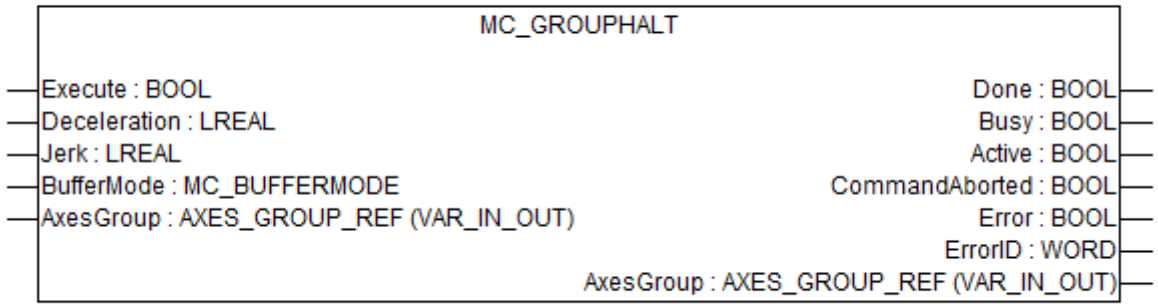


Fig. 490: MC_GroupHalt

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by disabling MC_Power of one or more of the axes in the group. The state changes to GroupDisabled.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupInterrupt

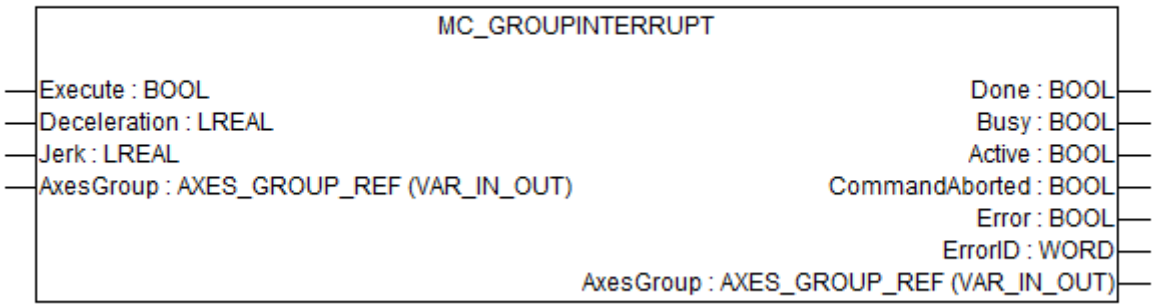


Fig. 491: MC_GroupInterrupt

This function block interrupts the on-going motion and stops the group from moving, however does not abort the interrupted motion (meaning that at the interrupted function block the output CommandAborted will not be Set, Busy is still high and Active is reset). It stores all relevant track or path information internally at the moment it becomes active. The AxesGroup stays in the original state even if the velocity zero is reached and the DONE output set.

- *This function block is coupled to MC_GroupContinue. Issuing MC_GroupContinue transfers the program back to the situation at issuing MC_GroupInterrupt.*
- *Further motion commands may be accepted by the group.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

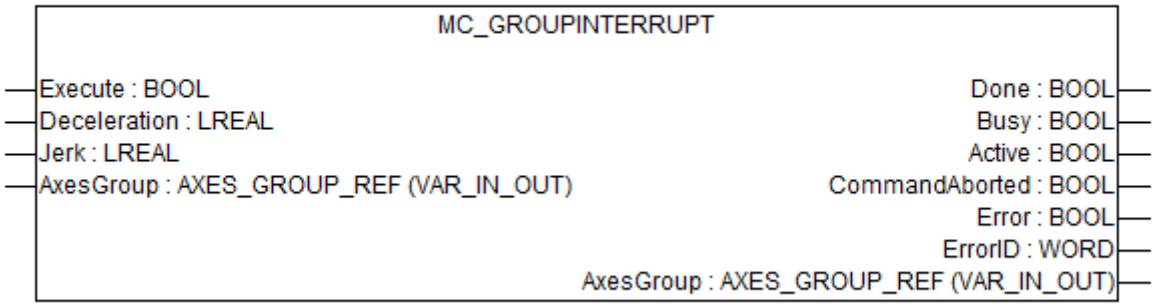


Fig. 492: MC_GroupInterrupt

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute Data type: BOOL
Starts the function block at rising edge.

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

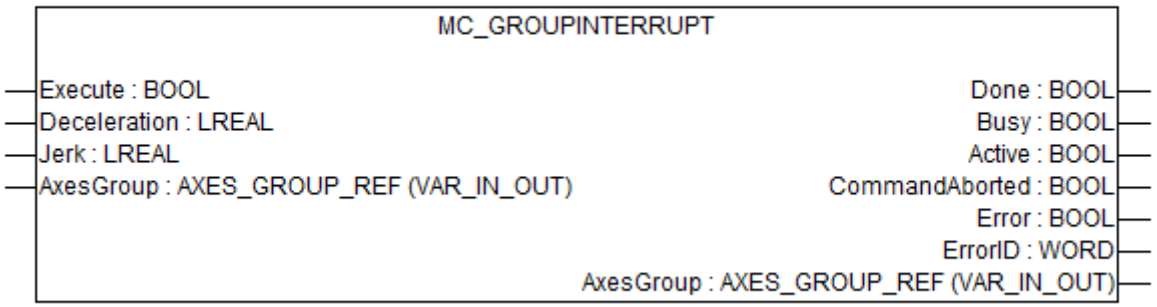


Fig. 493: MC_GroupInterrupt

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by disabling MC_Power of one or more of the axes in the group. The state changes to GroupDisabled.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupContinue

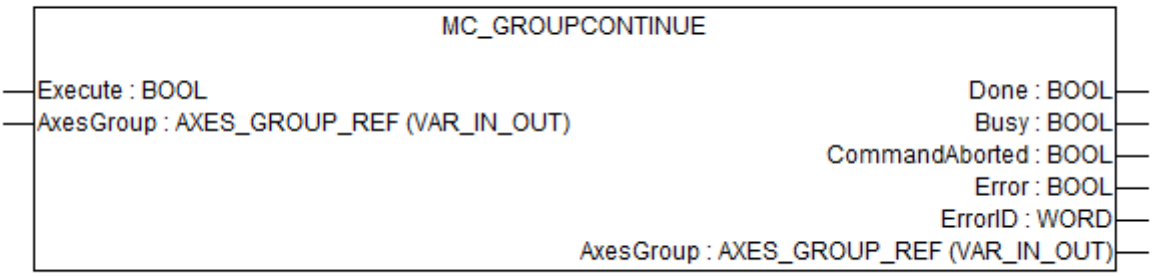


Fig. 494: MC_GroupContinue

This function block transfers the program back to the situation at issuing MC_GroupInterrupt. It uses internally the data set as stored at issuing MC_GroupInterrupt, and at the end (output DONE set) transfer the control on the group back to the original function block doing the movement on the axes group, meaning also that at the originally inter-rupted function block the output Busy is still high and Active is set again.

- *The dynamics of the function block that is continued can be used for the Velocity, Acceleration, Deceleration and Jerk,*
- *This function block can also be used to continue after an error in case the necessary set of data is stored at the occurrence of the error.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

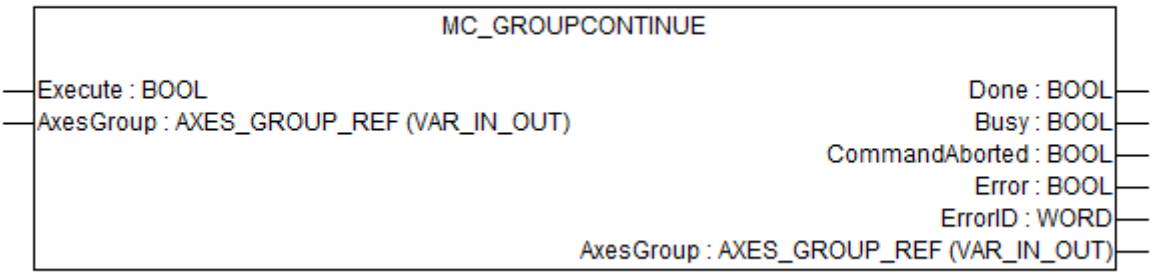


Fig. 495: MC_GroupContinue

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

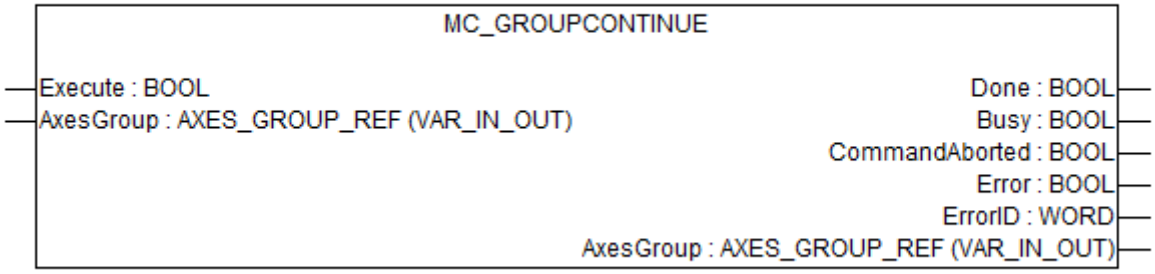


Fig. 496: MC_GroupContinue

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

CommandAborted Data type: BOOL

Command is aborted by disabling MC_Power of one or more of the axes in the group. The state changes to GroupDisabled.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_GroupReadStatus

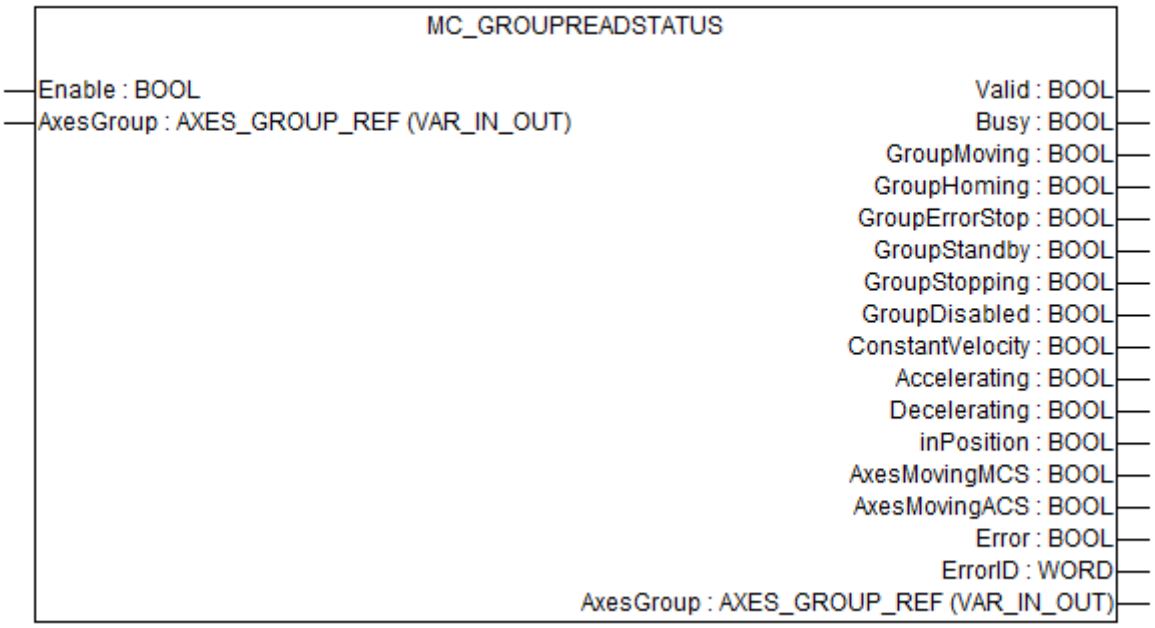


Fig. 497: MC_GroupReadStatus

This function block returns the status of an axes group according to the active Group-Functiom Block. This is an administrative Funcion Block, since no movement is generated.

> *The outputs reflect the commanded state of the group.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

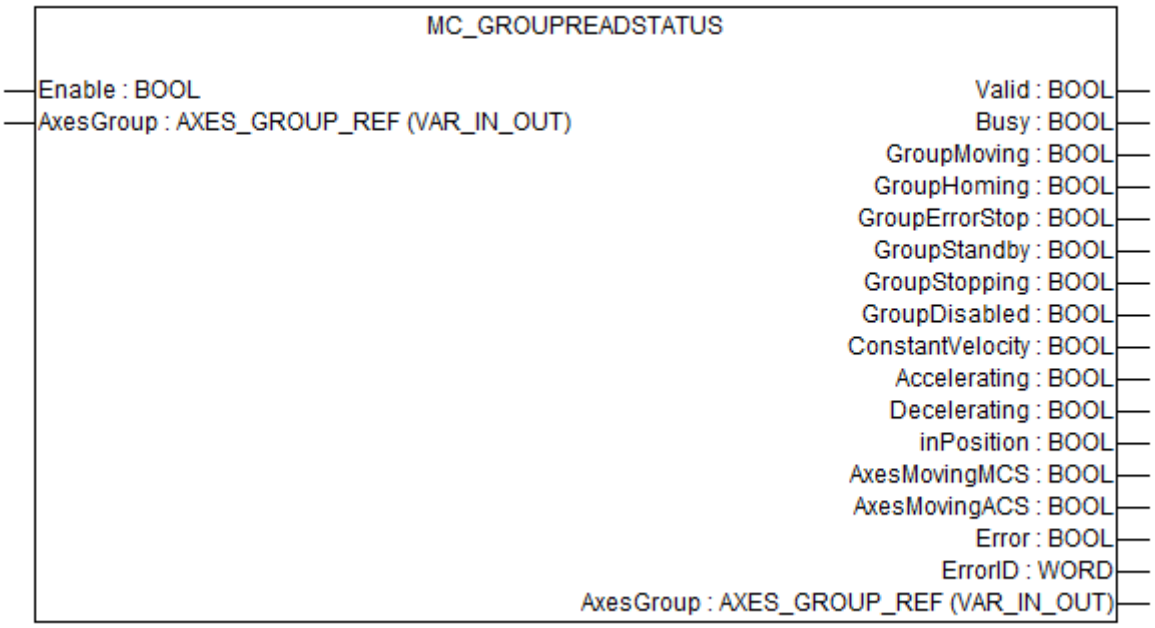


Fig. 498: MC_GroupReadStatus

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the actual position in the selected coordinate system of the axes group continuously while enabled.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

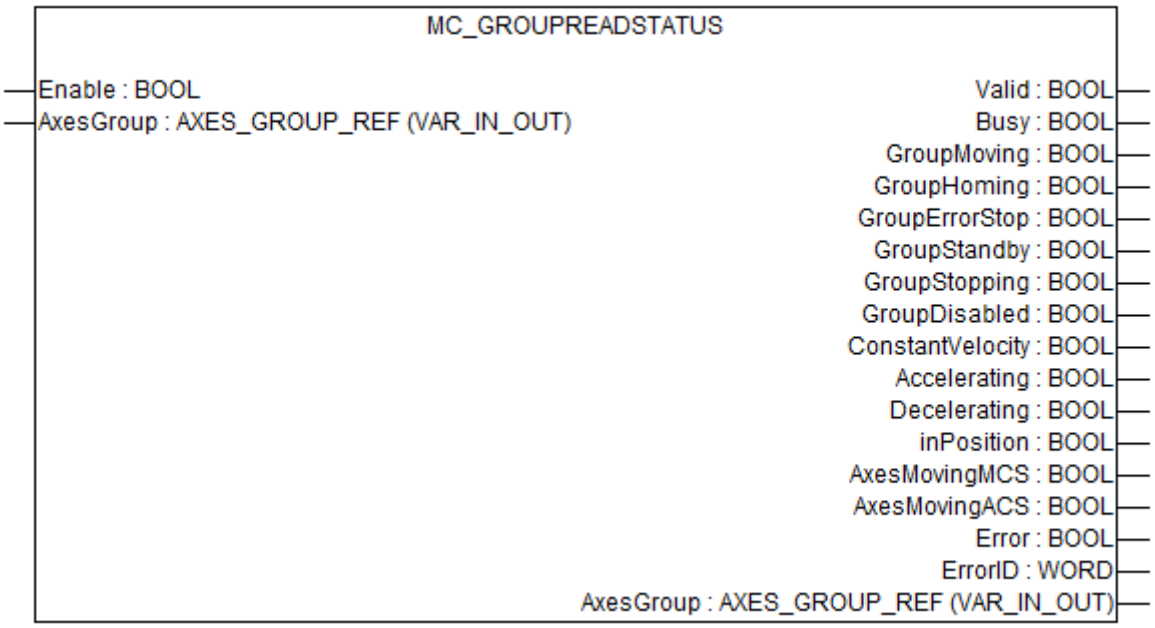


Fig. 499: MC_GroupReadStatus

Valid Data type: BOOL
True if valid outputs are available.

Busy Data type: BOOL
The function block is not finished.

GroupMoving Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

GroupHoming Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

GroupErrorStop Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

GroupStandby Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

GroupStopping Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

GroupDisabled Data type: BOOL
See group state diagram. *Chapter 1.5.9.4.9.3.1 “Group state diagram” on page 2711*

ConstantVelocity Data type: BOOL
Moving with constant velocity on commanded path.

Accelerating Data type: BOOL
Increasing Velocity on commanded path.

Decelerating Data type: BOOL
Decreasing Velocity on commanded path.

InPosition Data type: BOOL
Movement has reached target position .

MovingMCS Data type: BOOL
At least 1 MCS axis is in single axis movement.

MovingACS Data type: BOOL
At least 1 ACS axis is in single axis movement.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveLinearAbsolute

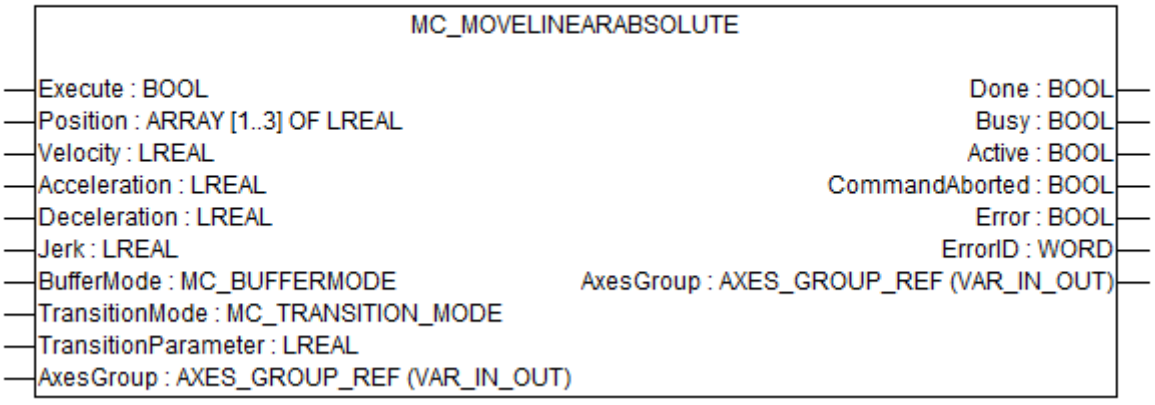


Fig. 500: MC_MoveLinearAbsolute

This function block commands an interpolated linear movement on an axes group from the actual position of the TCP to an absolute position in the specified coordinate system.

- *This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated,*
- *The behavior on interrupting an ongoing motion corresponds BufferMode=MC_Aborting, TransitionMode = TMDefinedVelocity, TransitionParameter = 100%.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

MC_MoveLinearAbsolute - Example

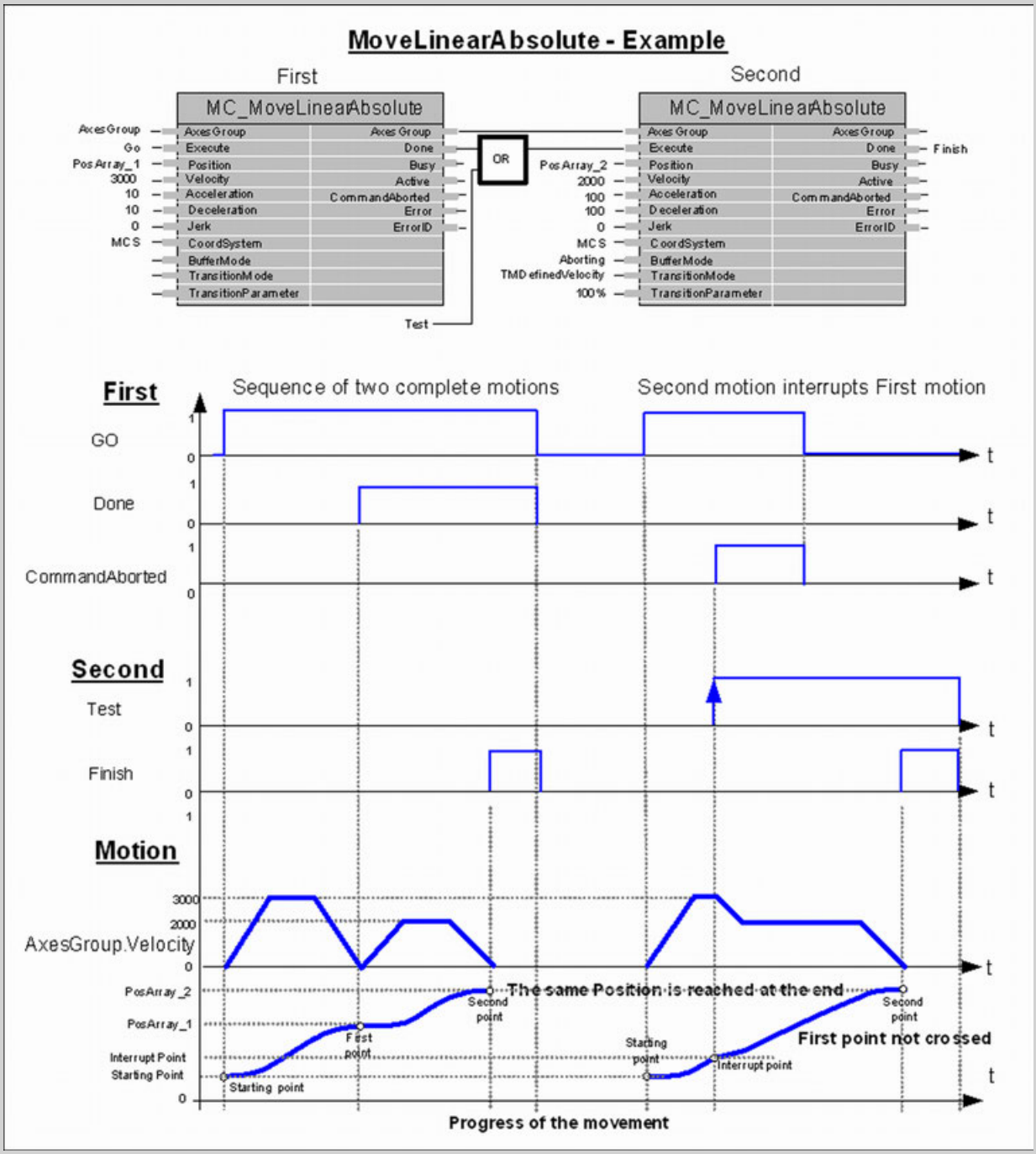
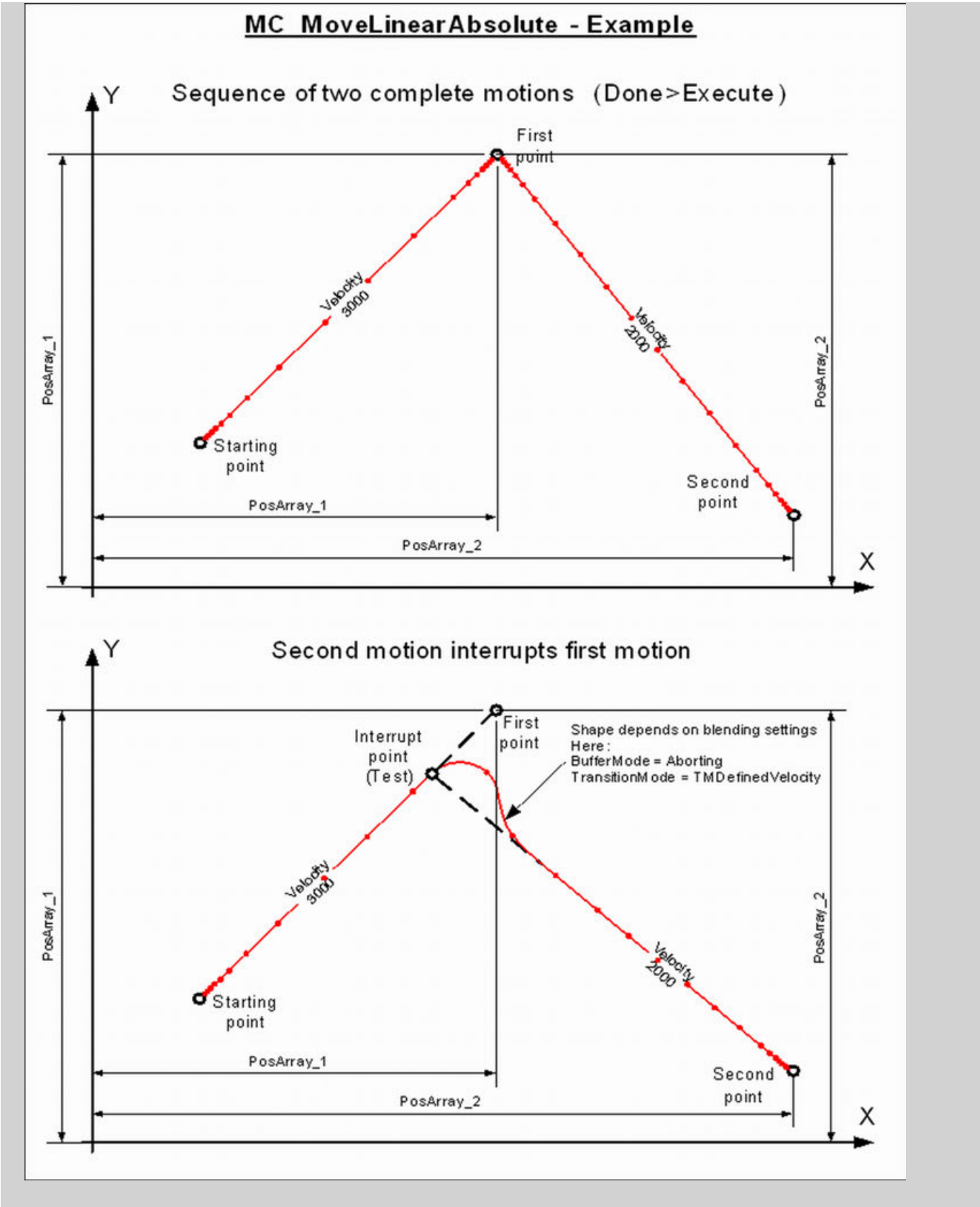


Fig. 501: MC_MoveLinearAbsolute Example

Timing diagram for example above (the dots on the red line are based on the same timing difference and representing the velocity)


MC_MoveLinearAbsolute - Example
Sequence of two complete motions (Done>Execute)
Y
First point
Velocity 3000
Velocity 2000
PosArray_1
PosArray_2
Starting point
Second point
PosArray_1
PosArray_2
X
Second motion interrupts first motion
Y
Interrupt point (Test)
First point
Shape depends on blending settings
Here:
BufferMode = Aborting
TransitionMode = TMDefinedVelocity
Velocity 3000
Velocity 2000
PosArray_1
PosArray_2
Starting point
Second point
PosArray_1
PosArray_2
X

Input description

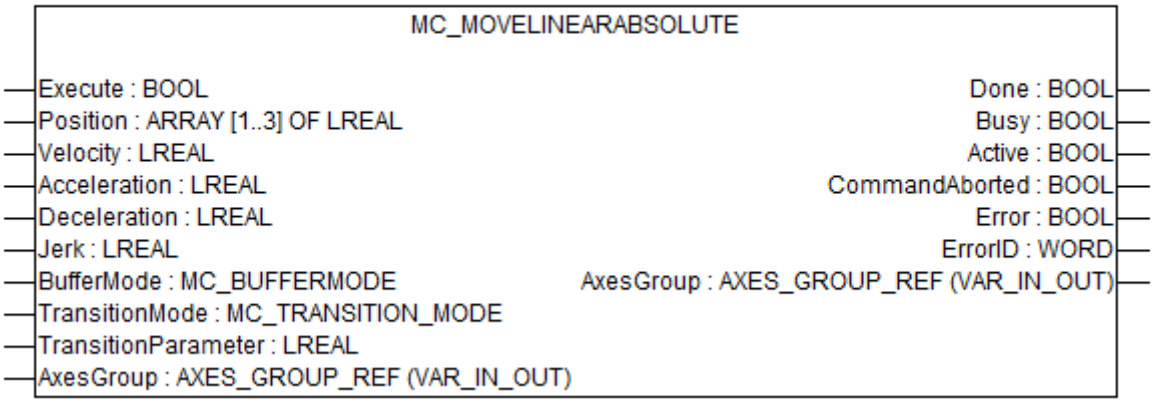


Fig. 502: MC_MoveLinearAbsolute

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute
Data type: BOOL
Starts the function block at rising edge.

Position
Data type: ARRAY [1..3] OF REAL
Array of absolute end positions for each dimension in the specified coordinate system. ↬ *"Position" on page 3038*

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE

The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL

Additional parameter for the transition mode.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

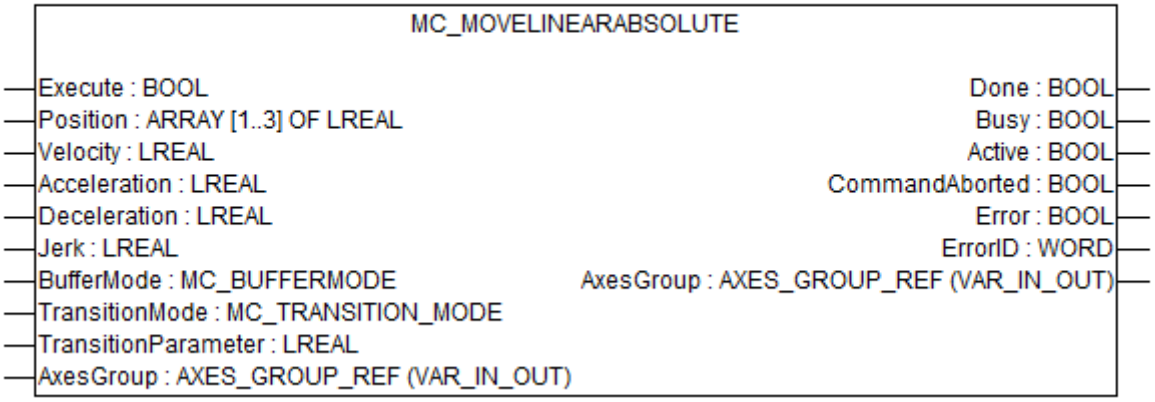


Fig. 503: MC_MoveLinearAbsolute

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by disabling MC_Power of one or more of the axes in the group. The state changes to GroupDisabled.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveLinearRelative

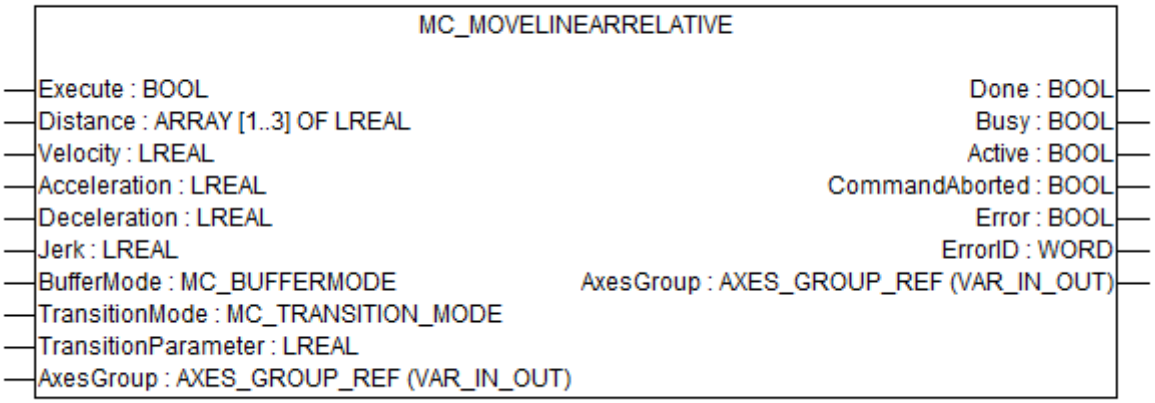


Fig. 504: MC_MoveLinearRelative

This function block commands an interpolated linear movement on an axes group from the actual position of the TCP to a relative position in the specified coordinate system.

> *This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic trans-formation when activated*

MC_MoveLinearRelative - Example 1

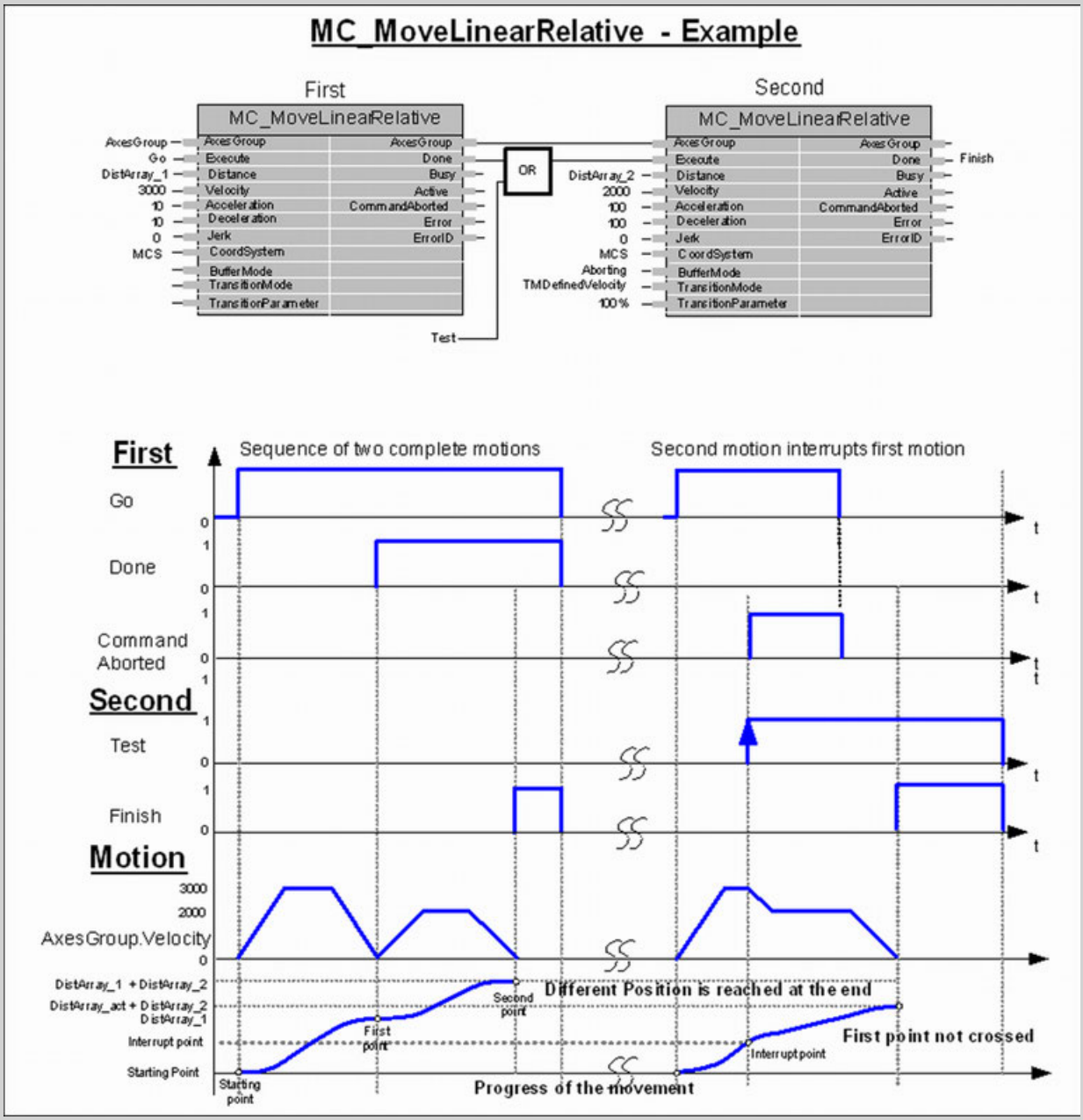


Fig. 505: Example MC_MoveLinearRelative

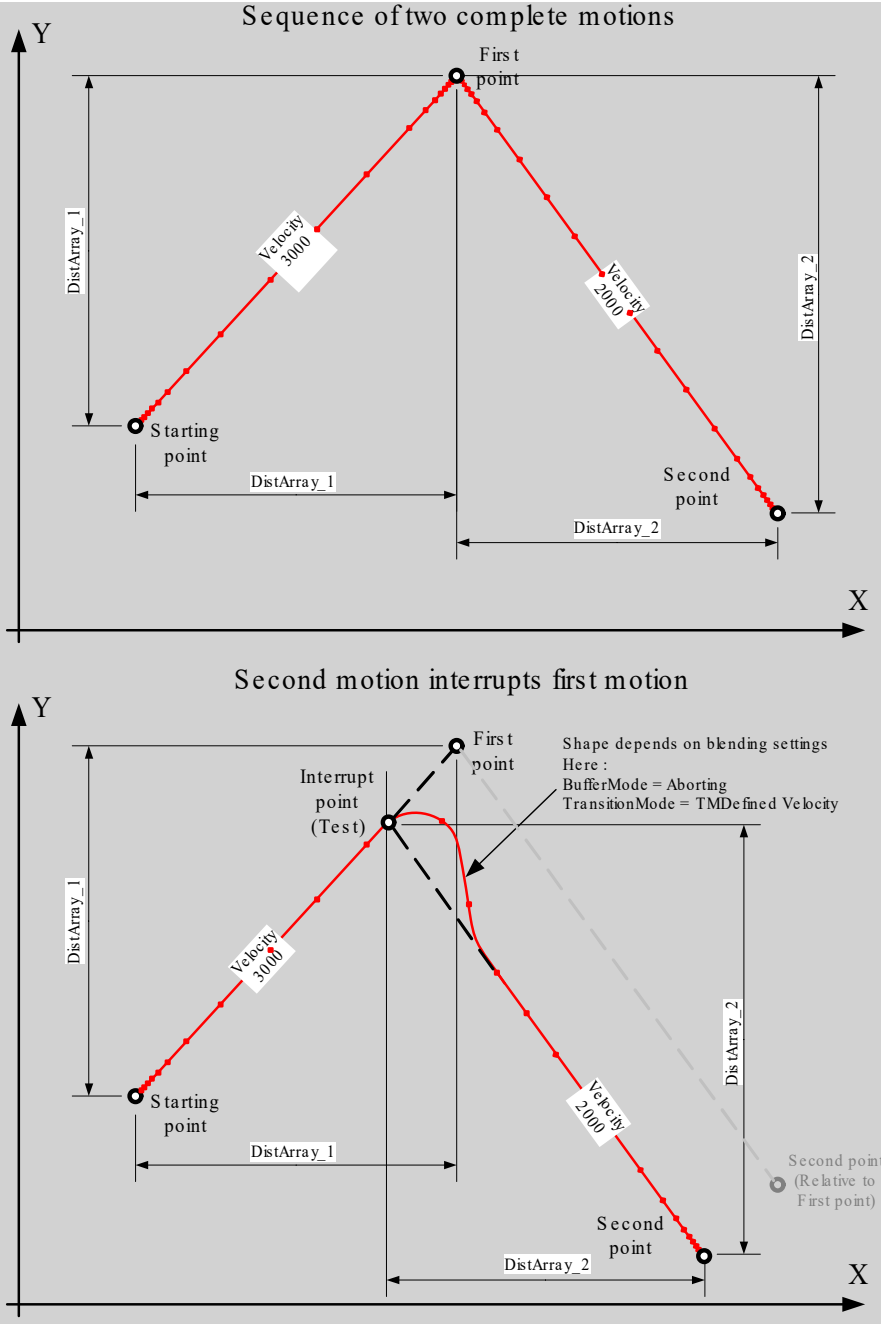


Fig. 506: Timing diagram for example above. The dots on the red line are based on the same timing difference and represent the velocity.

MC_MoveLinearRelative - Example 2

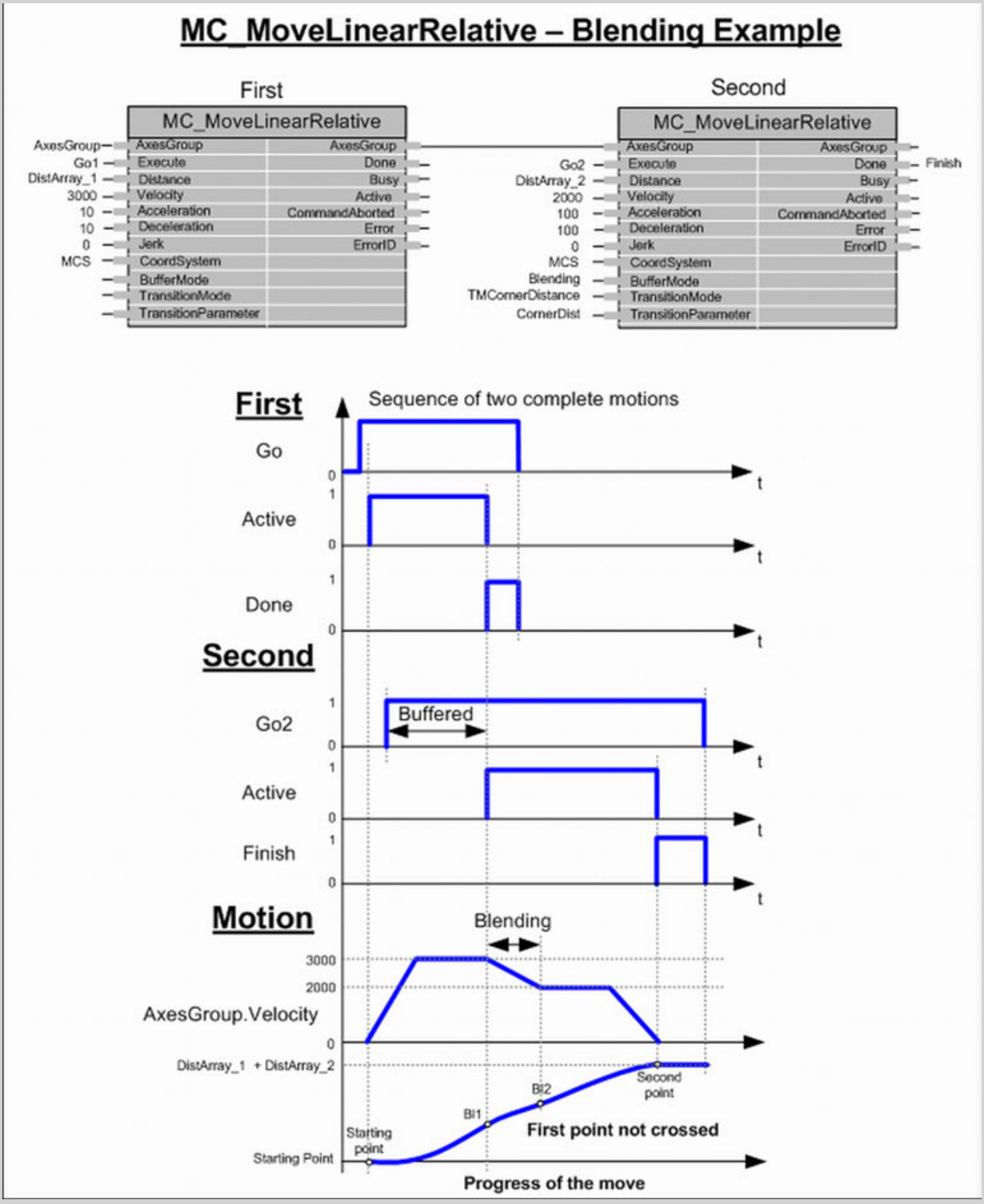
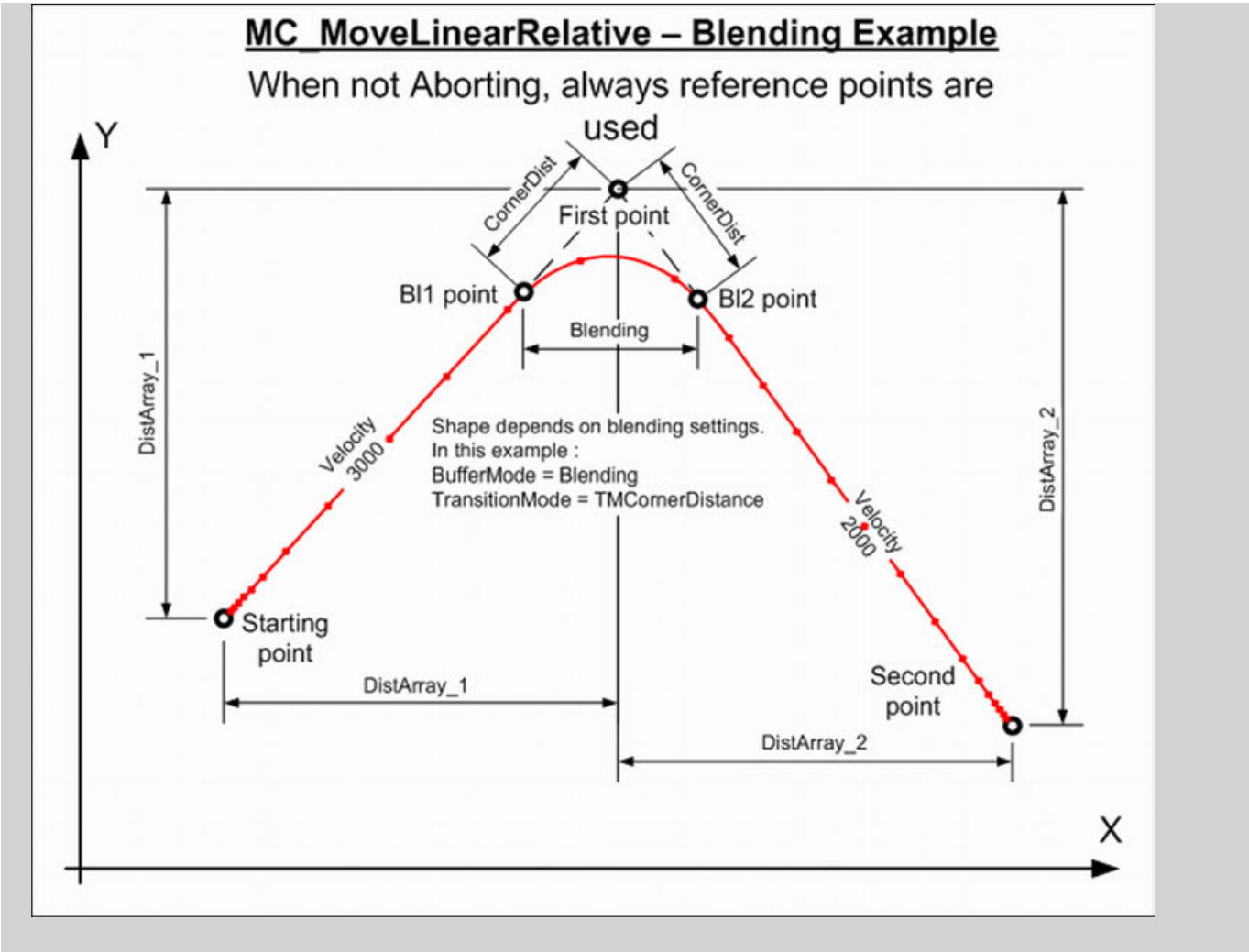


Fig. 507: Second example with MC_MoveLinearRelative and Blending

Timing diagram for example below (the dots on the red line are based on the same timing difference and representing the velocity):



Input description

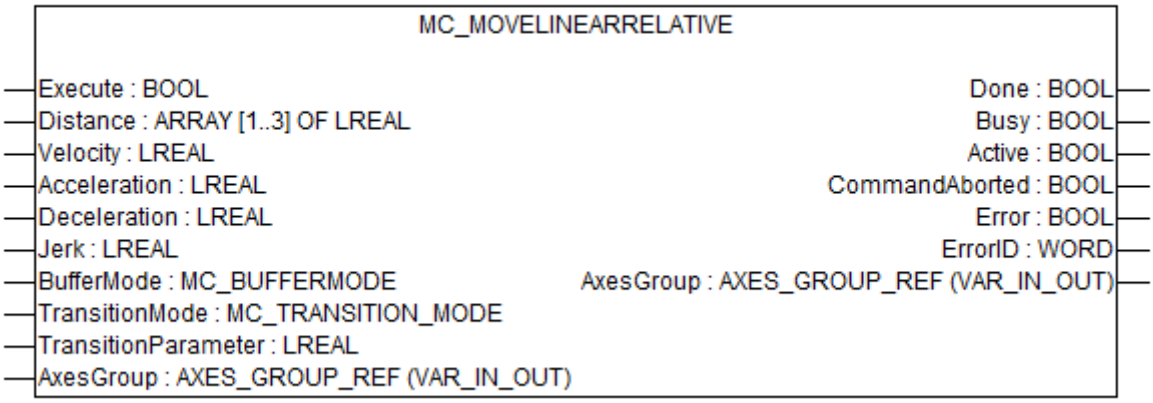


Fig. 508: MC_MoveLinearRelative

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute
Data type: BOOL
Starts the function block at rising edge.

Distance
Data type: ARRAY [1..3] OF LREAL
Array of relative distances for each dimension in the specified coordinate system.

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode
Data type: MC_TRANSITION_MODE
The realization just supports by default a transition starting with the actual velocity.

TransitionParameter
Data type: LREAL
Additional parameter for the transition mode.

AxesGroup
Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

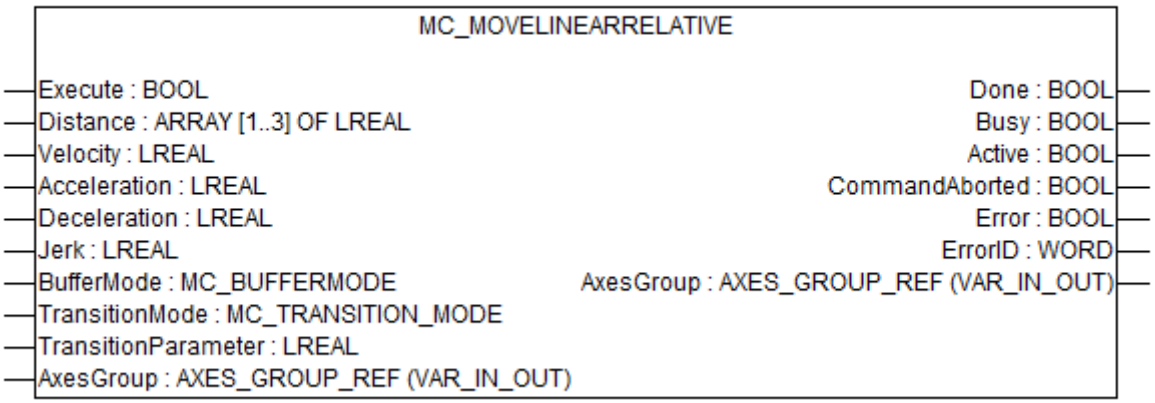


Fig. 509: MC_MoveLinearRelative

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveCircularAbsolute

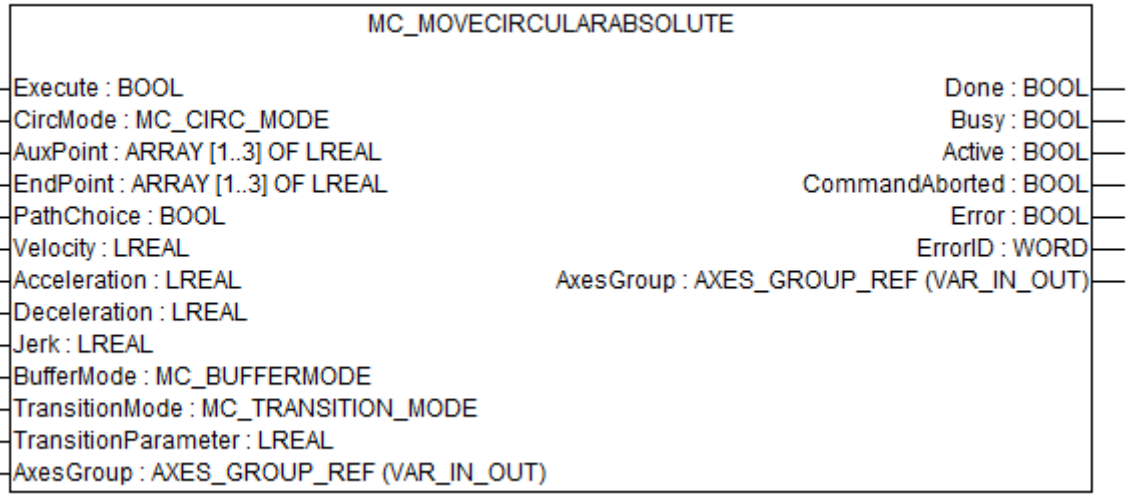


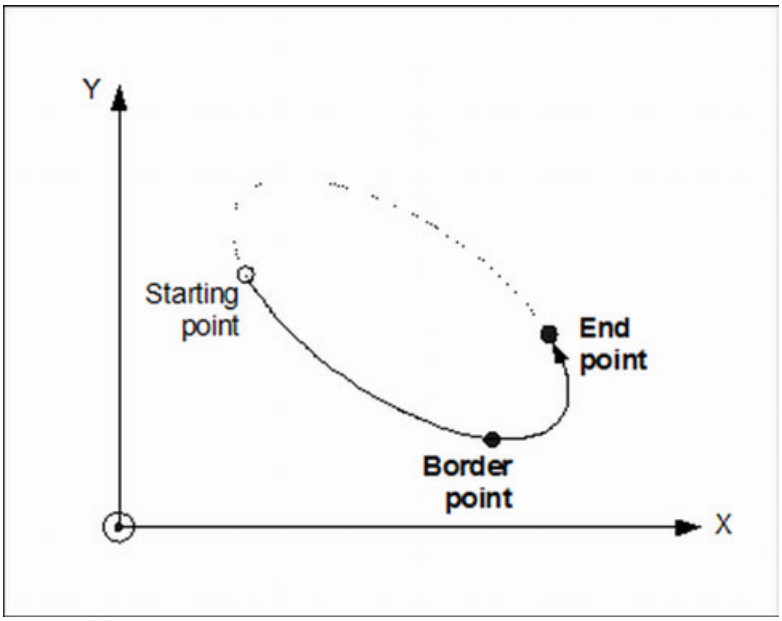
Fig. 510: MC_MoveCircularAbsolute

This function block commands an interpolated circular movement on an axes group from the actual position of the TCP. The end point as well as the auxiliary point (meaning depending on applied mode, see below) are defined absolutely in the specified coordinate system.

> *This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

CircMode = BORDER



The user defines the end point and a border point (= input 'AuxPoint') on the sector of the circle, which shall be cruised by the machine.

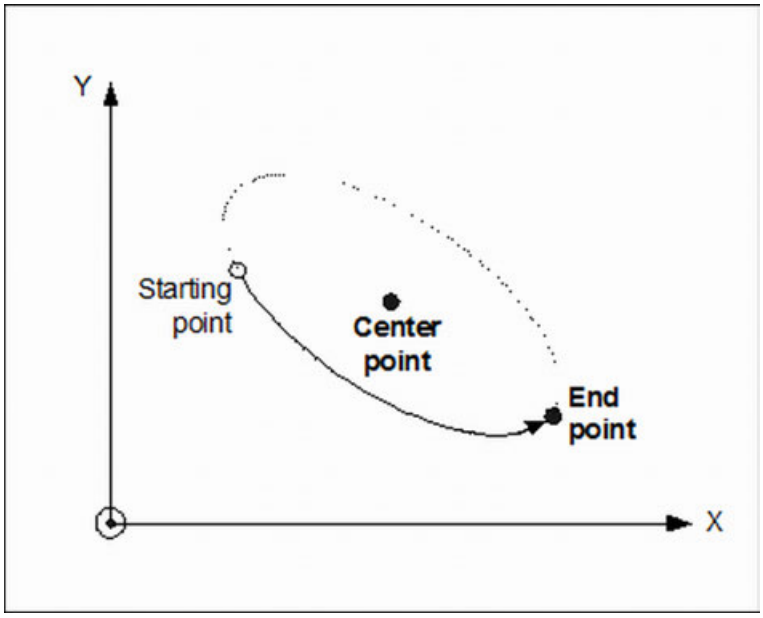
Advantages of this mode:

- The border point usually can be reached by the machine, i.e. it can be teached.

Inconvenience of this mode:

- Restriction to angles$<2\pi$ in one single command.

CircMode = CENTER

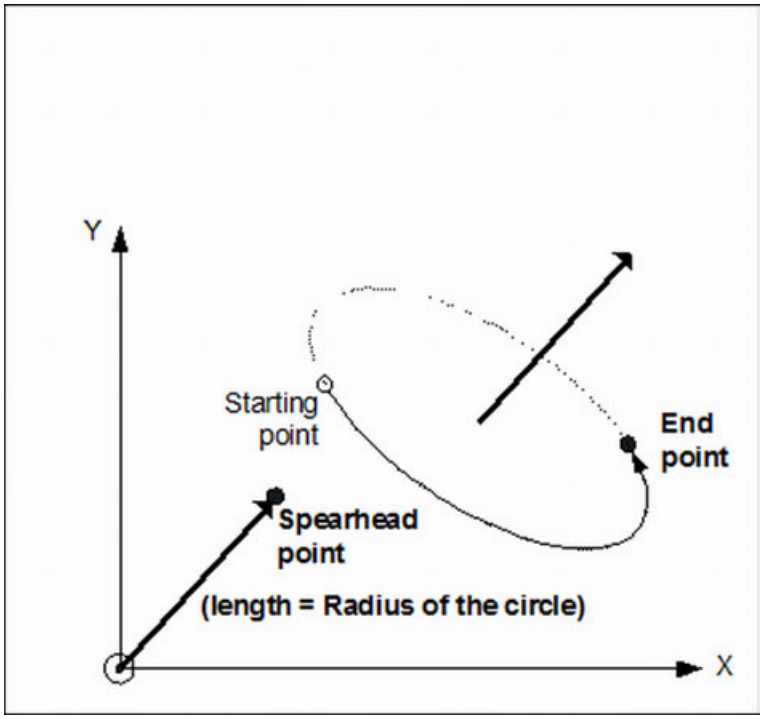


The user defines the end point and the center point (= input 'AuxPoint') of the circle. When using this mode, the input 'PathChoice' defines, if the short or the long sector has to be cruised by the machine.

Inconveniencies of this mode:

- Restriction to angles $< 2\pi$ and $\neq \pi$ in one single command,
- Overdetermination of circle equation,
- The center point usually cannot be teached in due to collisions with obstacles.

CircMode = RADIUS



The user defines the end point and the perpendicular vector of the circle plane according to the rule of right thumb (see figure below). The length of the vector corresponds to the radius of the circle. The spearhead point of the vector is the input signal 'AuxPoint' in absolute coordinates, i.e. referring to the origine of the coordinate system specified in 'CoordSystem'. If the diameter is larger than the distance between starting and end point, two different circles have to be considered. When using this mode, the input 'PathChoice' defines, if the circle with the short sector or the circle with the long sector to reach the end point has to be cruised by the machine. With positive radius value the shortest possible circle is determined, and with negative radius value the largest possible circle.

Inconvenience of this mode:

- Restriction to angles $< 2\pi$ in one single command,
- Overdetermination of circle equation.

Example:	AuxPoint = (50,0,0) →Circle in plane parallel to y-z plane with radius 50 and rotation around axis parallel to x-axis according to the rule of right thumb (CoordSystem = MCS).

MC_MoveCircularAbsolute - Sequenced example

Sequenced example of 2 MC_MoveCircularAbsolute function blocks:

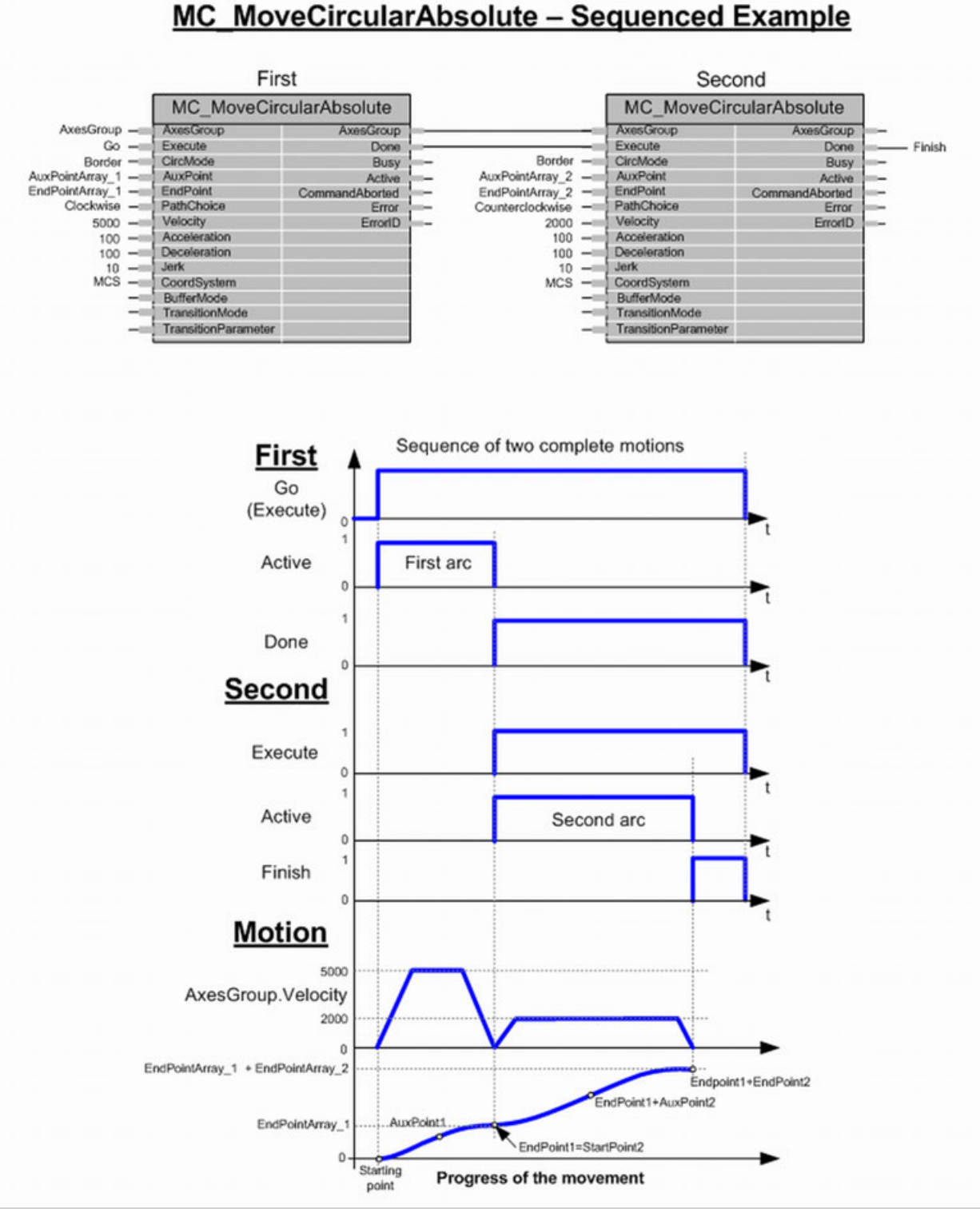


Fig. 511: Example MC_MoveCircularAbsolute

Timing diagram of example above. The dots on the red line are based on the same timing difference and representing thevelocity:

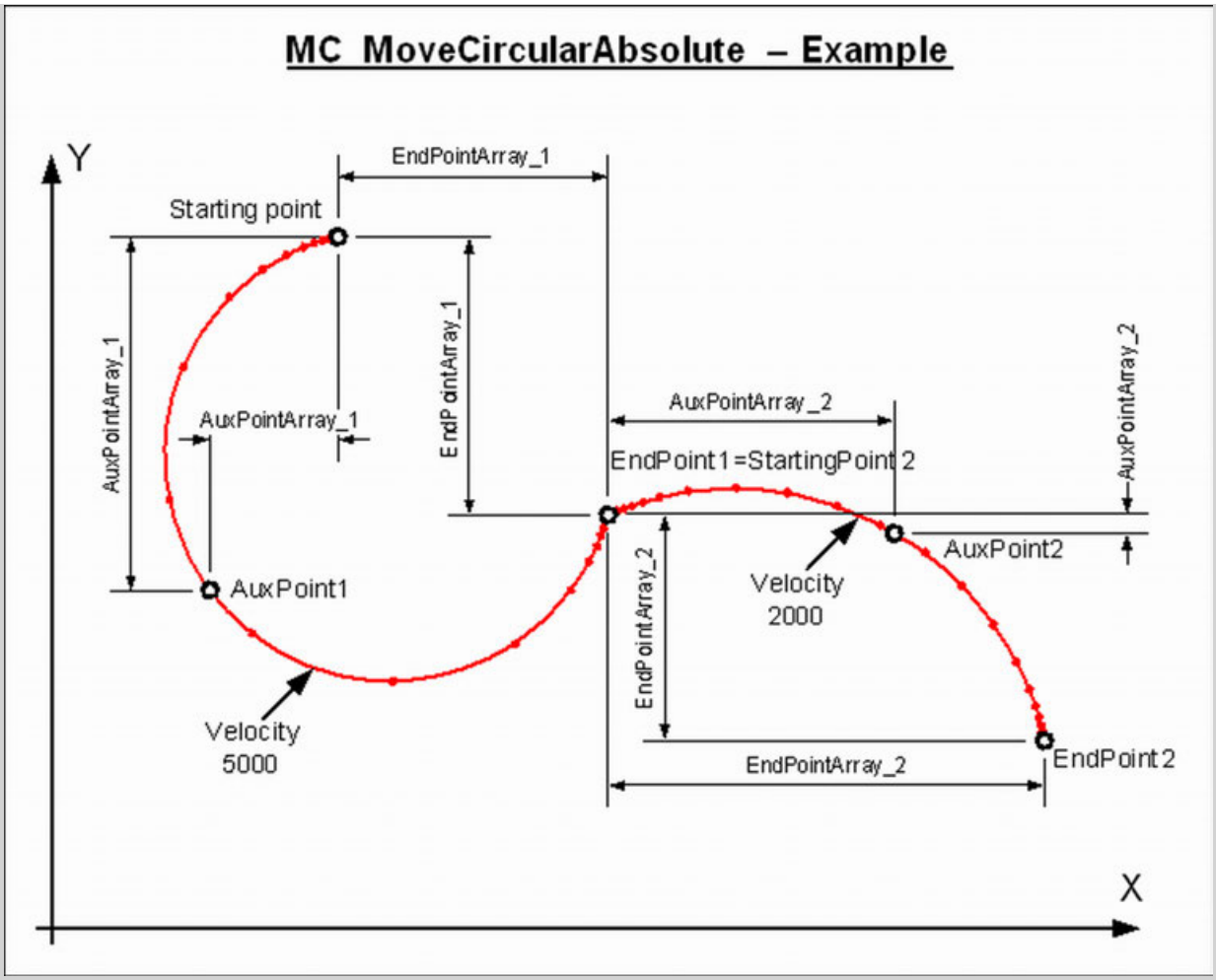


Fig. 512: Example MC_MoveCircularAbsolute

Input description

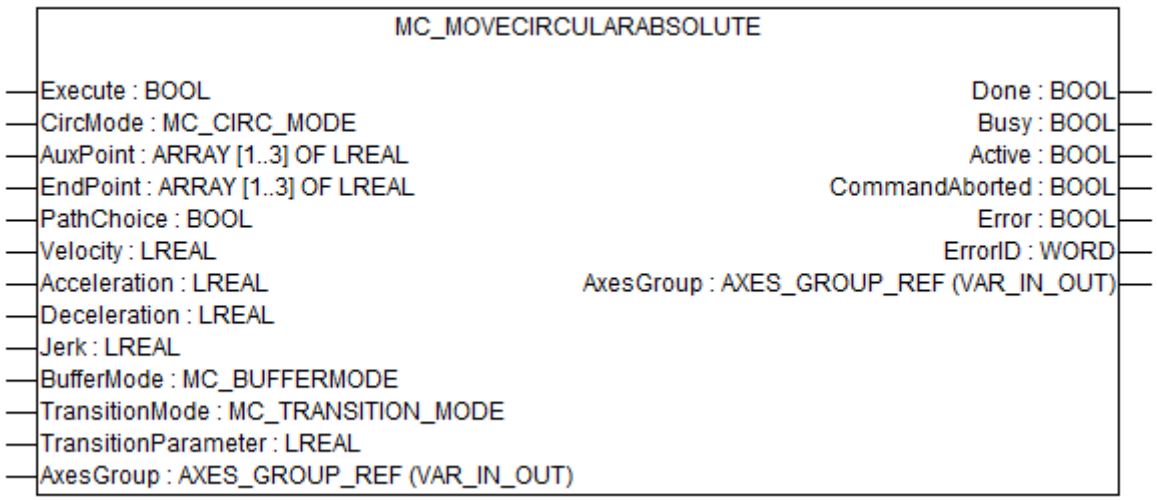


Fig. 513: MC_MoveCircularAbsolute

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute

Data type: BOOL

Starts the function block at rising edge.

CircMode

Data type: MC_CIRC_MODE

Specifies the meaning of the input signals 'AuxPoint' and 'CircDirection'.

- MC_BORDER:
  'AuxPoint' defines a point on the circle which is crossed on the path from the starting to the end point.
- MC_CENTER:
  'AuxPoint' defines the center point of the circle.
- MC_RADIUS:
  'AuxPoint' defines the spearhead point of the perpendicular of the circle plane according to the rule of right thumb. The radius of the circle is the length of the vector.

In the function block MC_MoveCircularAbsolute, the points are specified absolutely, i.e. the perpendicular vector begins in the origine and ends in the spearhead point specified at the input signal 'AuxPoint'.

AuxPoint

Data type: ARRAY [1..3] OF REAL

Array of positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

EndPoint

Data type: ARRAY [1..3] OF REAL

Array of absolute positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

PathChoice

Data type: BOOL

Choice of path: False = clockwise, true = counterclockwise.

Velocity

Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration

Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE
The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL
Additional parameter for the transition mode.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

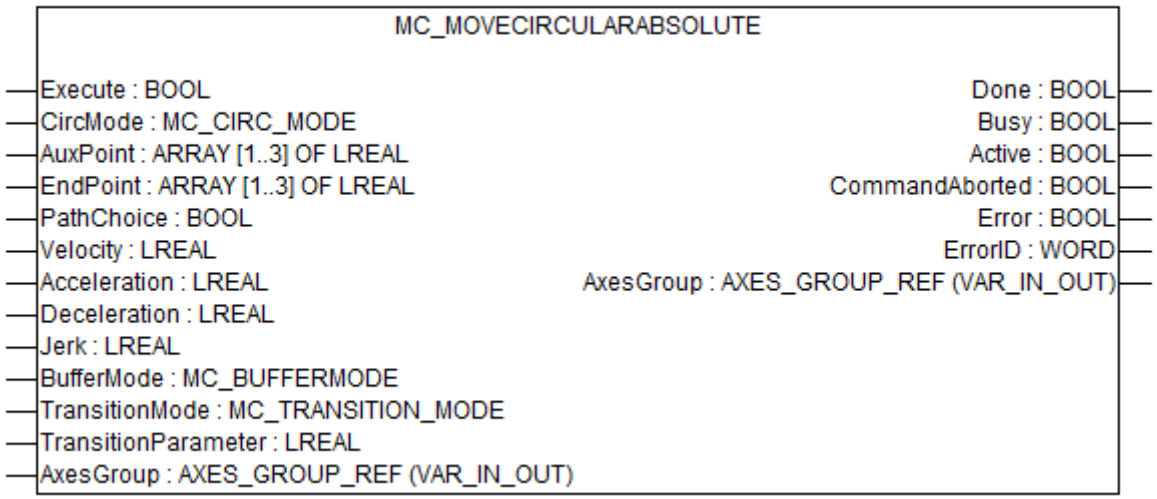


Fig. 514: MC_MoveCircularAbsolute

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveCircularRelative

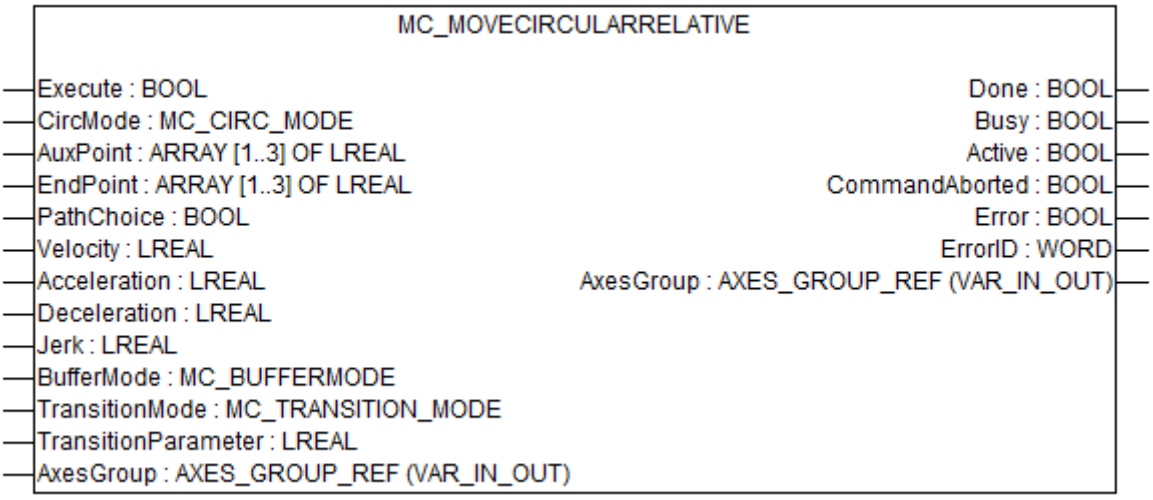


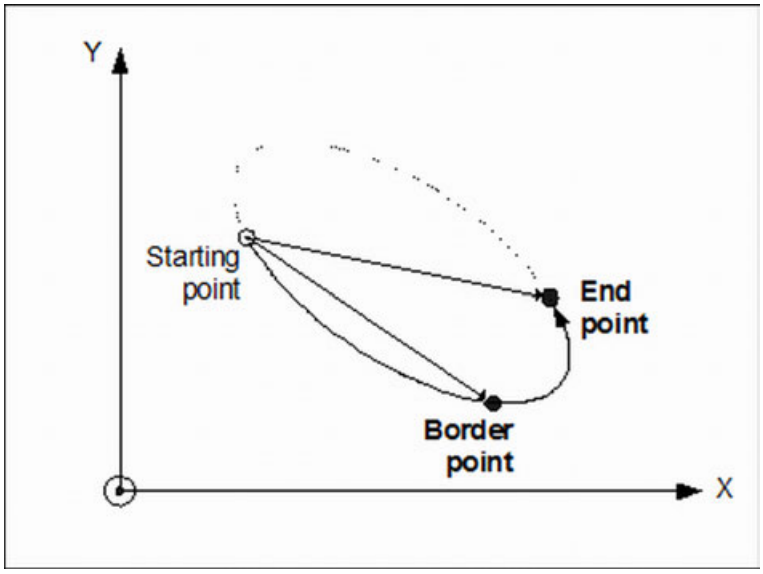
Fig. 515: MC_MoveCircularRelative

This function block commands an interpolated circular movement on an axes group from the actual position of the TCP. The end point as well as the auxiliary point (meaning depending on applied mode, see below) are defined in the specified coordinate system relatively to the starting point.

> *This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

CircMode = BORDER



The user defines the end point and a border point (= input 'AuxPoint') on the sector of the circle, which shall be cruised by the machine. Both points are defined relatively to the starting point.

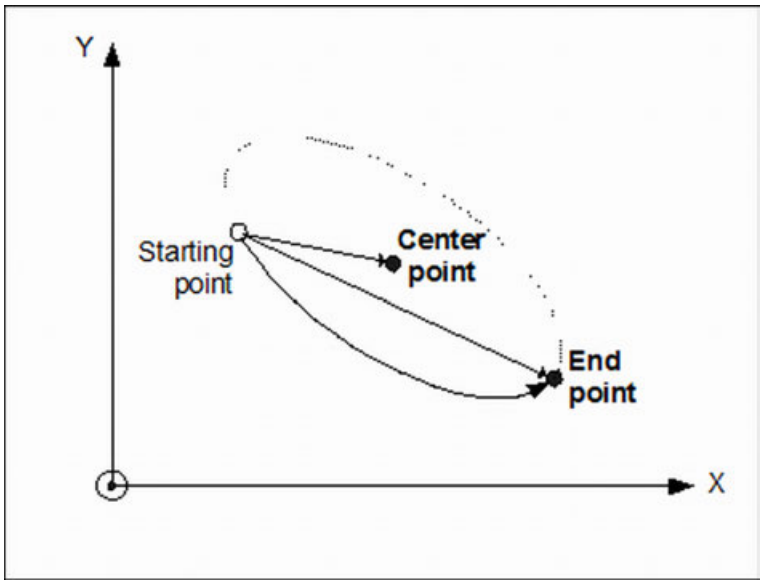
Advantages of this mode:

- The border point usually can be reached by the machine, i.e. it can be teached.

Inconvenience of this mode:

- Restriction to angles $<2\pi$ in one single command.

CircMode = CENTER

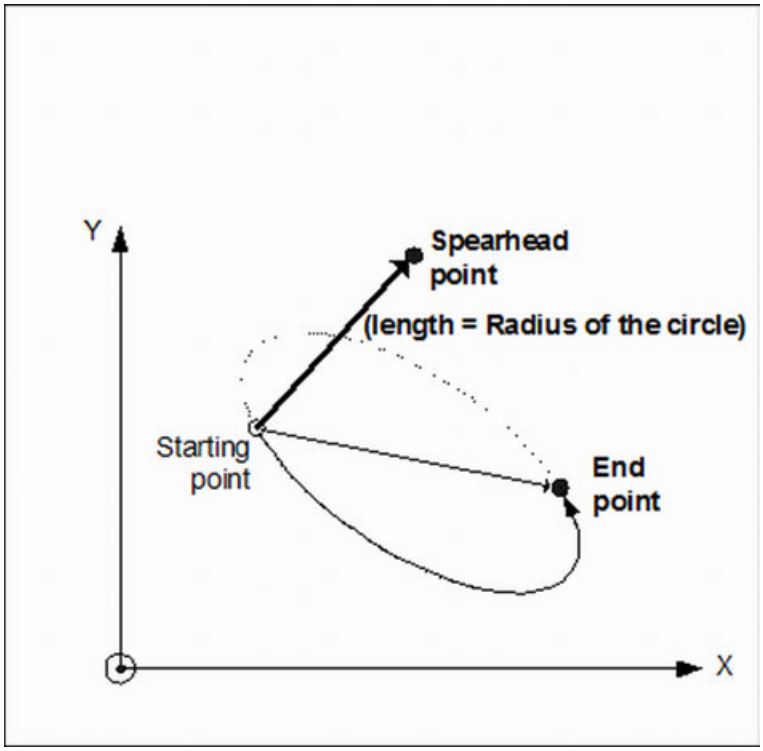


The user defines the end point and the center point (= input 'AuxPoint') of the circle. Both points are defined relatively to the starting point When using this mode, the input 'PathChoice' defines, if the short or the long sector has to be cruised by the machine.

Inconveniencies of this mode:

- Restriction to angles $< 2\pi$ and $\neq \pi$ in one single command,
- Overdetermination of circle equation,
- The center point usually cannot be teached-in due to collisions with obstacles.

CircMode = RADIUS



The user defines the end point and the perpendicular vector of the circle plane according to the rule of right thumb (see figure below). The length of the vector corresponds to the radius of the circle. The spearhead point of the vector is defined relatively to the starting point at the input signal 'AuxPoint'. If the diameter is larger than the distance between starting and end point, two different circles have to be considered. When using this mode, the input 'PathChoice' defines, if the circle with the short sector or the circle with the long sector to reach the end point has to be cruised by the machine. With positive radius value the shortest possible circle is determined, and with negative radius value the largest possible circle

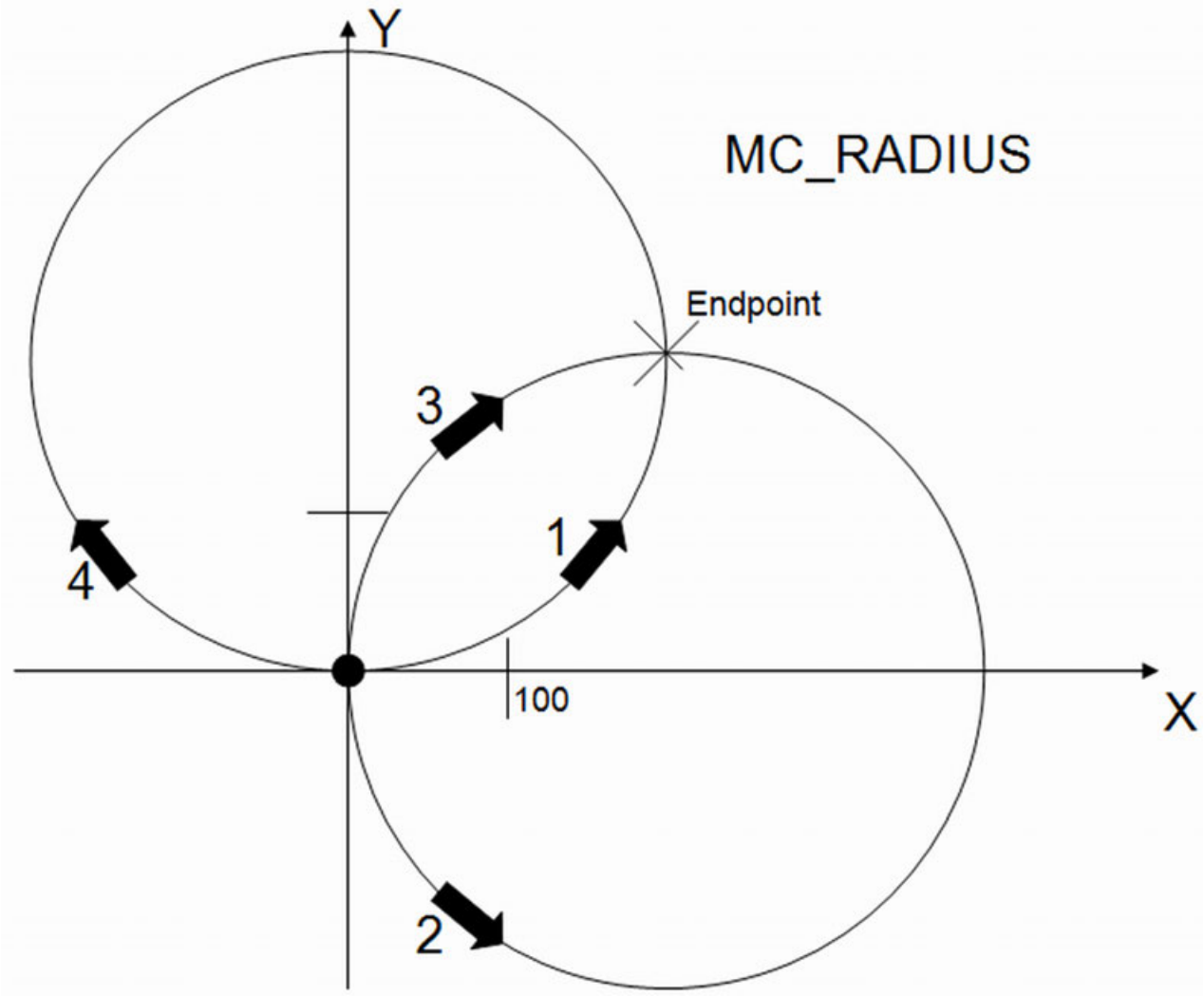
Inconvenience of this mode:

- Restriction to angles $<2\pi$ in one single command,
- The perpendicular vector has to be computed,
- Overdetermination of circle equation.

Example:	AuxPoint = (starting_point[0], starting_point[1] - 30, starting_point[2]) → Circle in plane parallel to x-z plane with radius 30 and rotation around axis parallel to y-axis contrariwise the rule of right thumb (CoordSystem = MCS).

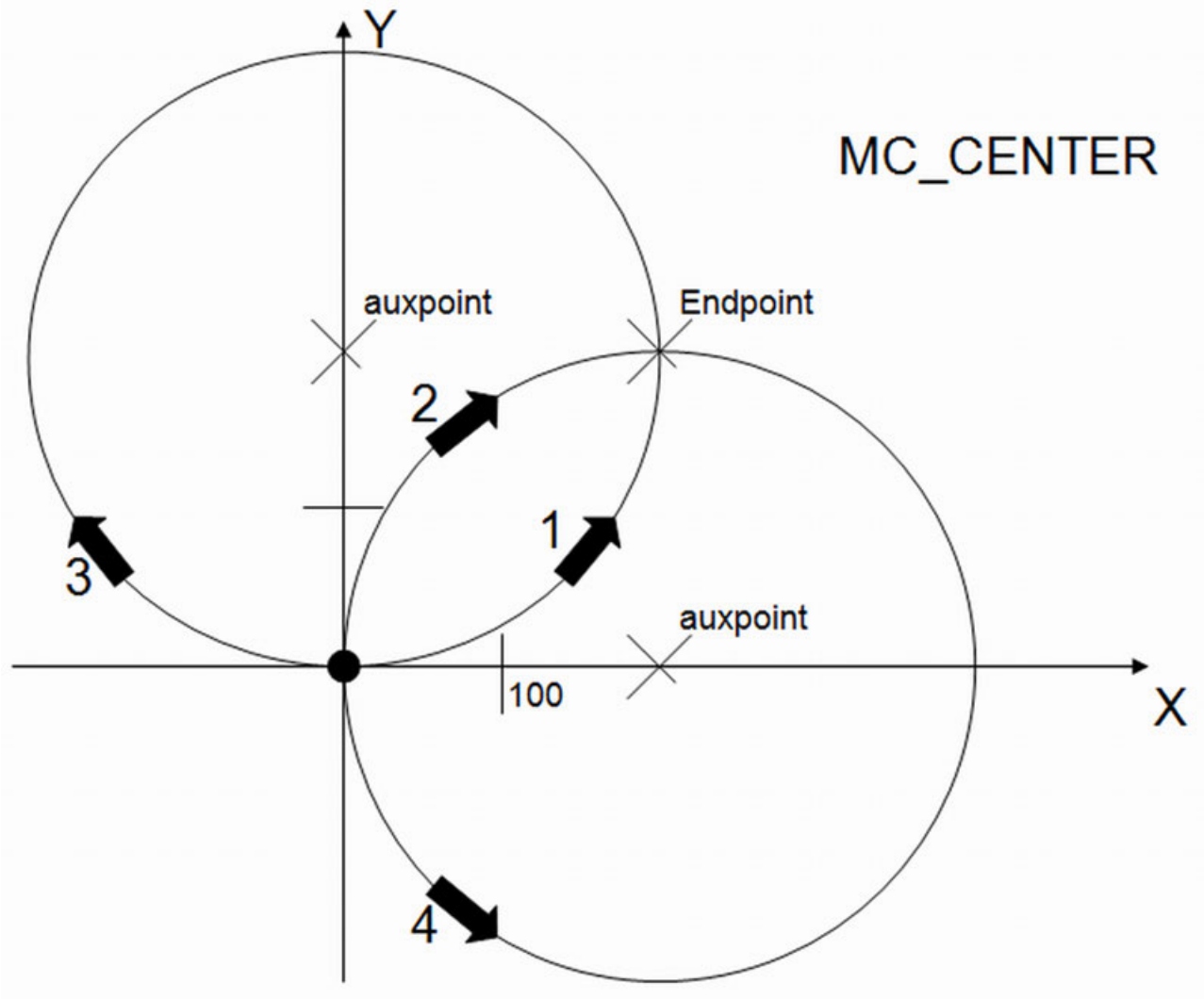
With CircMode=MC_RADIUS, Endpoint is a point at the circle and Auxpoint a vector which is perpendicular of the circle plane according to the rule of right thumb. The radius of the circle is the length of Auxpoint. The moving direction is always positive (counterclockwise), PathChoice determines if the long or short distance is moved.

MC_RADIUS



MC_RADIUS	PathChoice	Auxpoint	Endpoint
1	FALSE (short)	0,0,200	200,200,0
2	TRUE (long)	0,0,200	200,200,0
3	FALSE (short)	0,0,-200	200,200,0
4	TRUE (long)	0,0,-200	200,200,0

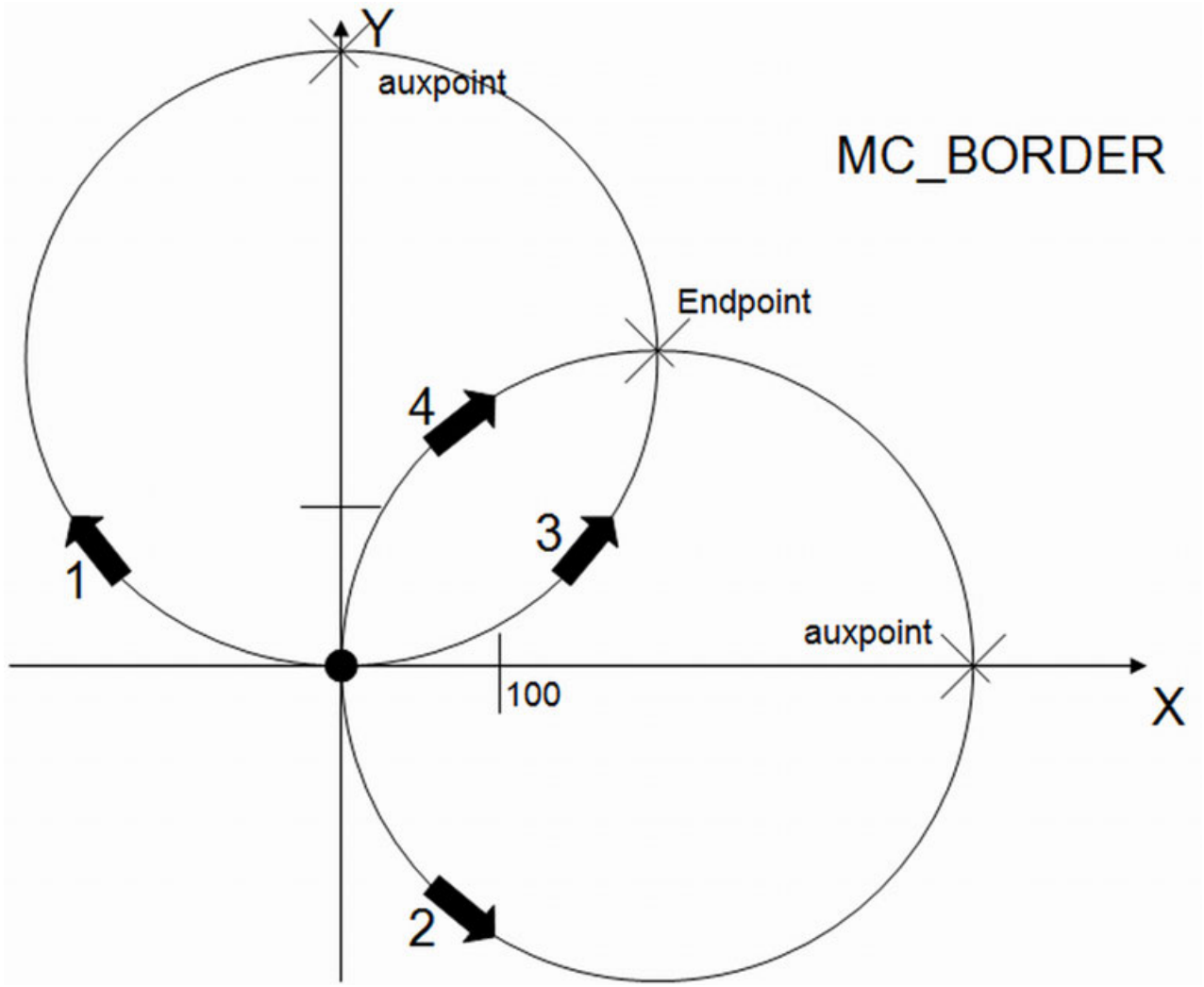
MC_CENTER



With CircMode=MC_CENTER, Endpoint is a point at the circle where the movement stops and Auxpoint is the center point. Both are relative to the start point. The circle plane is defined by Auxpoint x Endpoint. Restriction to angles $<2\pi$ and $\neq \pi$as the definition is not unique otherwise.

MC_CENTER	PathChoice	Auxpoint	Endpoint
1	FALSE (short)	0,200,0	200,200,0
2	FALSE (short)	200,0,0	200,200,0
3	TRUE (long)	0,200,0	200,200,0
4	TRUE (long)	200,0,0	200,200,0

MC_BORDER



With CircMode=MC_BORDER, Endpoint is a point at the circle where the movement stops and Auxpoint is a point on the circle. Both are relative to the start point.

MC_RADIUS	PathChoice	Auxpoint	Endpoint
1	-	0,400,0	200,200,0
2	-	400,0,0	200,200,0
3	-	141,59,0	200,200,0
4	-	59,141,0	200,200,0

Input description

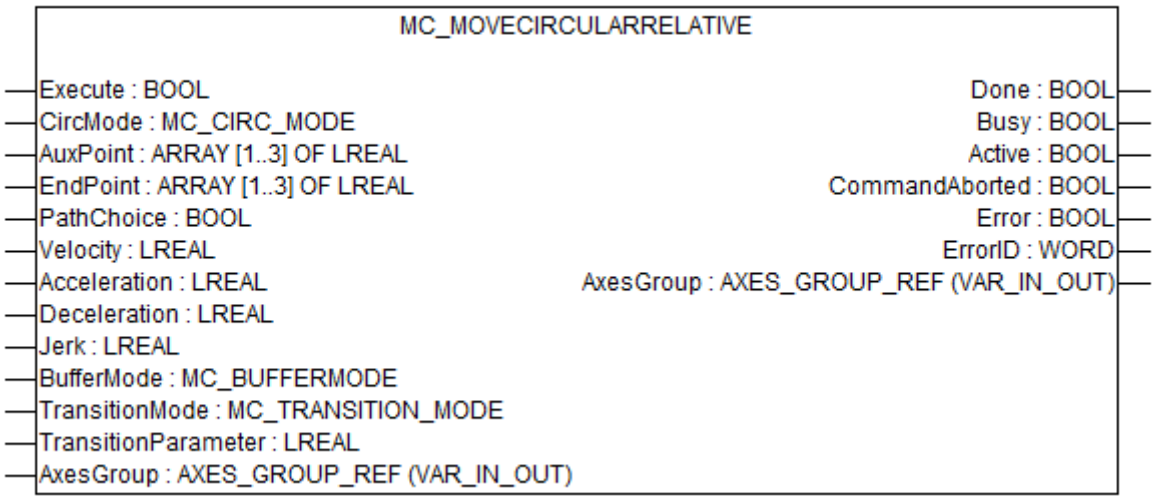


Fig. 516: MC_MoveCircularRelative

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

CircMode

Data type: MC_CIRC_MODE

Specifies the meaning of the input signals 'AuxPoint' and 'CircDirection'.

- MC_BORDER:
  'AuxPoint' defines a point on the circle which is crossed on the path from the starting to the end point.
- MC_CENTER:
  'AuxPoint' defines the center point of the circle.
- MC_RADIUS:
  'AuxPoint' defines the spearhead point of the perpendicular of the circle plane according to the rule of right thumb. The radius of the circle is the length of the vector.

In the function block MC_MoveCircularAbsolute, the points are specified absolutely, i.e. the perpendicular vector begins in the origine and ends in the spearhead point specified at the input signal 'AuxPoint'.

AuxPoint

Data type: ARRAY [1..3] OF REAL

Array of positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

EndPoint Data type: ARRAY [1..3] OF REAL

Array of absolute positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

PathChoice Data type: BOOL

Choice of path: False = clockwise, true = counterclockwise.

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE

The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL

Additional parameter for the transition mode.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

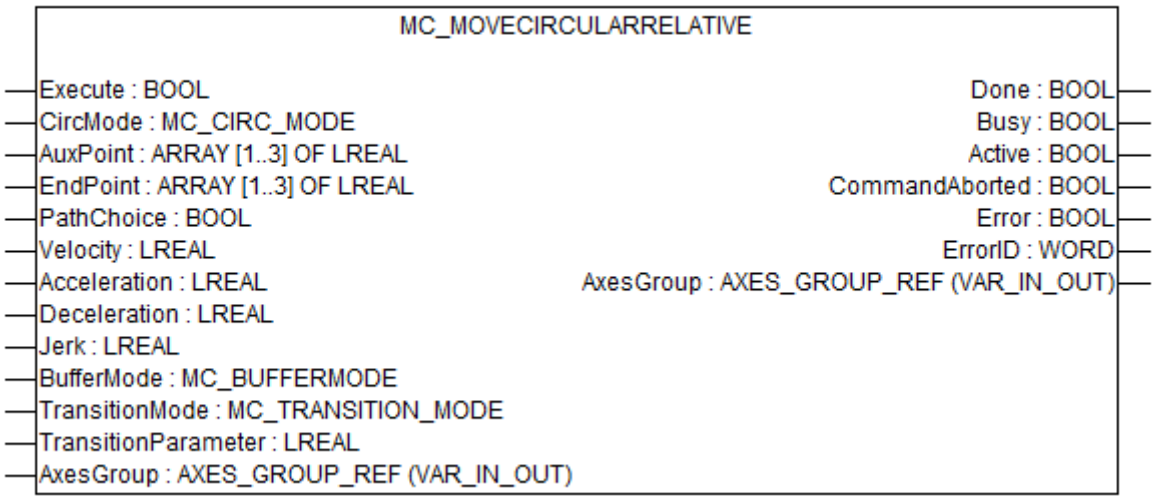


Fig. 517: MC_MoveCircularRelative

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveDirectAbsolute

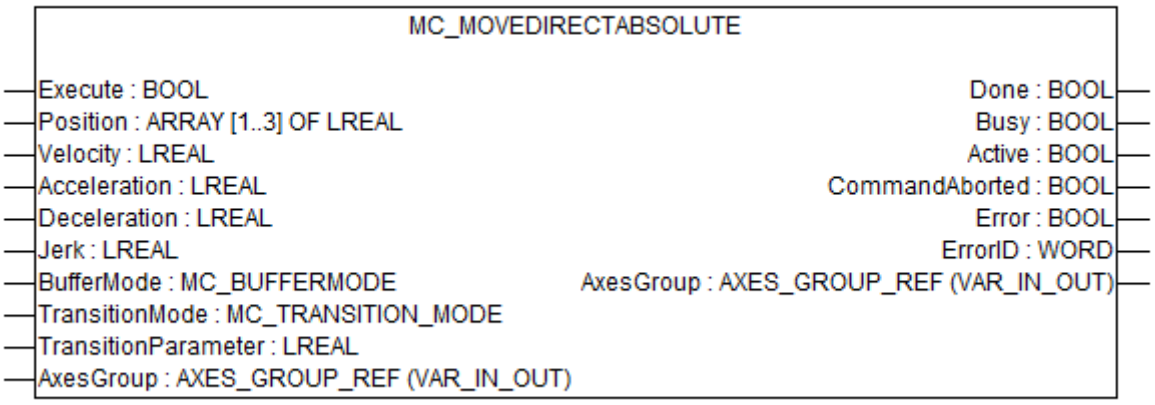


Fig. 518: MC_MoveDirectAbsolute

This function block commands a movement of an axes group to the specified absolute position in the specified coordinate system without taking care of how (on which path) the target position is reached.

- *The velocity/acceleration/deceleration/jerk of every axis are properties of each axis and not specified within this function block, but not to be exceeded during the move.*
- *This function block applies to the MCS or PCS System, depending which is activated and does not follow the dynamic transformation when activated.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

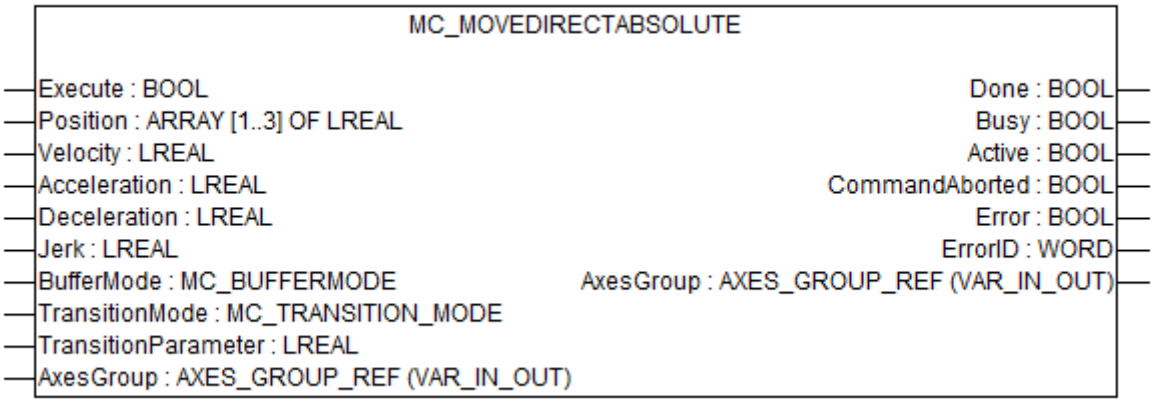


Fig. 519: MC_MoveDirectAbsolute

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute
Data type: BOOL
Starts the function block at rising edge.

Position
Data type: ARRAY [1..3] OF REAL
Array of absolute end positions for each dimension in the specified coordinate system. *"Position" on page 3038*

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode
Data type: MC_TRANSITION_MODE
The realization just supports by default a transition starting with the actual velocity.

TransitionParameter
Data type: LREAL
Additional parameter for the transition mode.

AxesGroup
Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

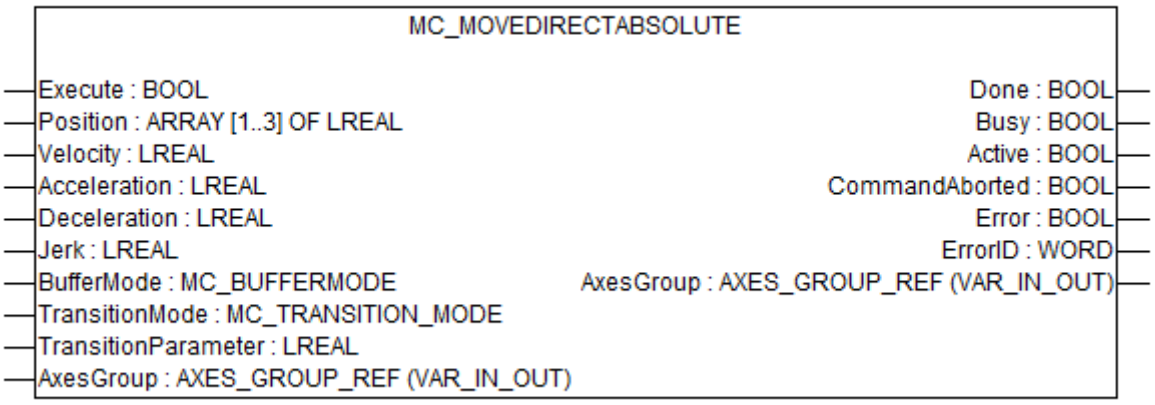


Fig. 520: MC_MoveDirectAbsolute

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MoveDirectRelative

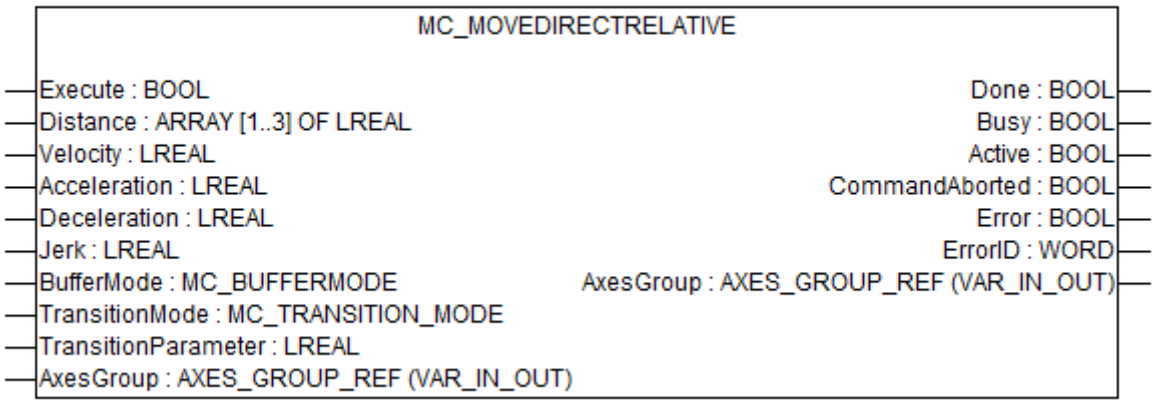


Fig. 521: MC_MoveDirectRelative

This function block commands a movement of an axes group to a relative position without taking care of how (on which path) the target position is reached. Start position is the actual position of the TCP.

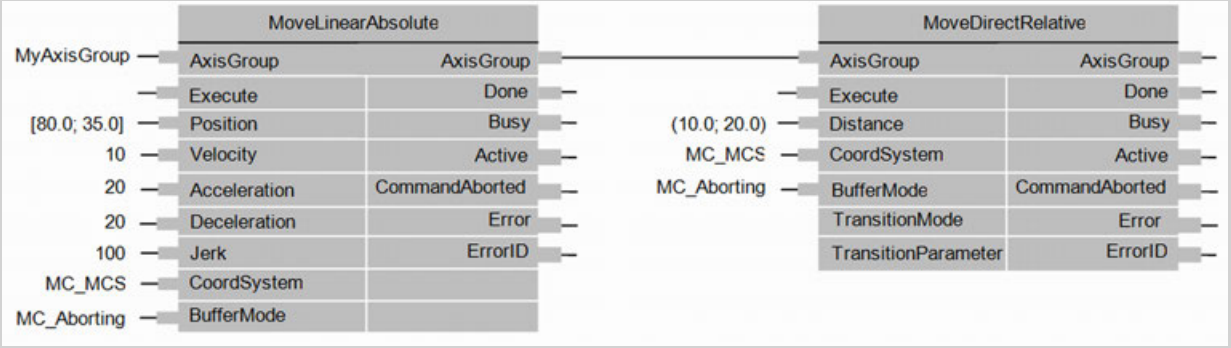
- *The velocity/acceleration/deceleration/jerk of every axis are properties of each axis and not specified within this function block, but not to be exceeded during the move.*
- *This function block applies to the MCS or PCS System, depending which is activated and does not follow the dynamic transformation when activated.*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Behavior of MC_MovePositionDirectRelative

Following example shows the behavior of MC_MovePositionDirectRelative. All positions are related to MCS:

- Starting at position p0 (10; 10) a MC_MoveLinearAbsolute to position p1 (80; 35) is commanded,
- While the TCP is moving towards p1, the MC_MoveLinearAbsolute command is aborted by a MC_MovePositionDirectRelative command. The actual position of the TCP, when MC_MovePositionDirectRelative becomes active, is (44.5; 21.63),
- The TCP leaves the line p0p1 and moves to the new target position p2 (54.5; 41.63). The resulting trajectory depends on the kinematic transformation of the axes group.



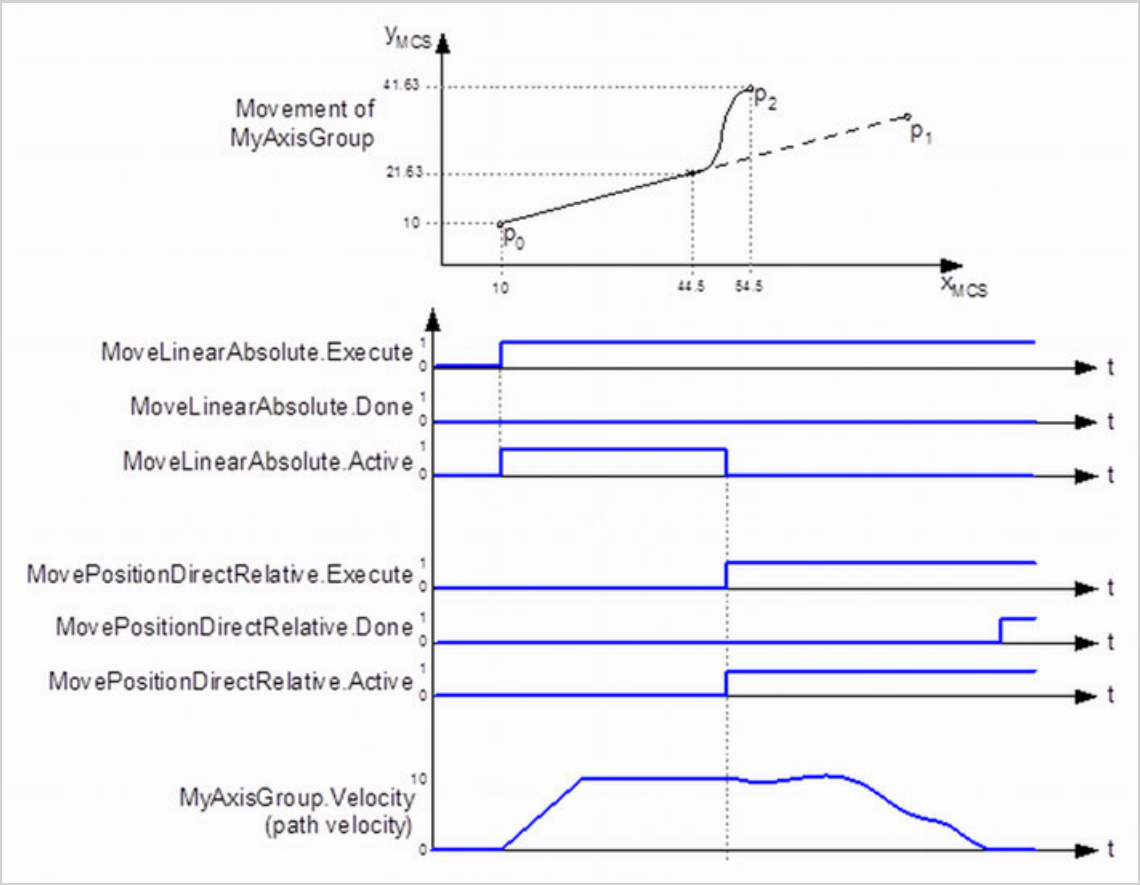


Fig. 522: Example MC_MoveDirectRelative

Input description

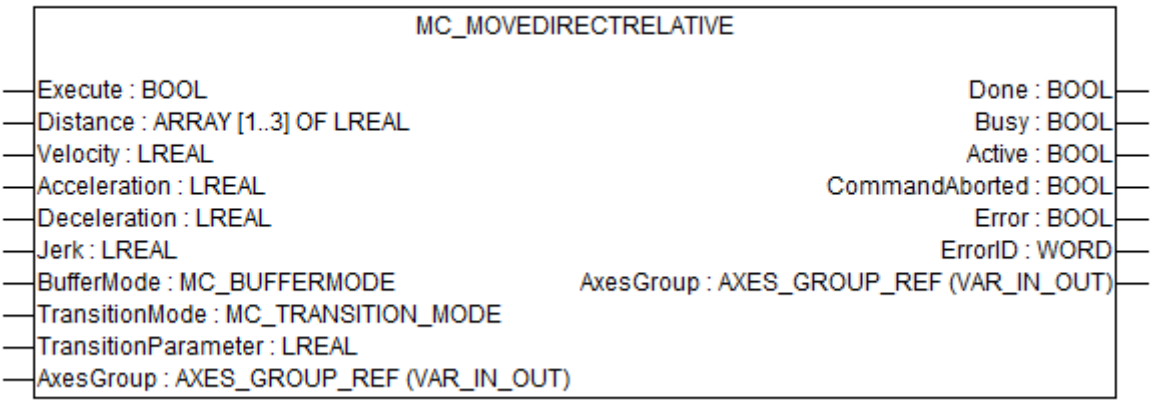


Fig. 523: MC_MoveDirectRelative

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

Distance Data type: ARRAY [1..3] OF LREAL
Array of relative distances for each dimension in the specified coordinate system.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE
The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL
Additional parameter for the transition mode.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

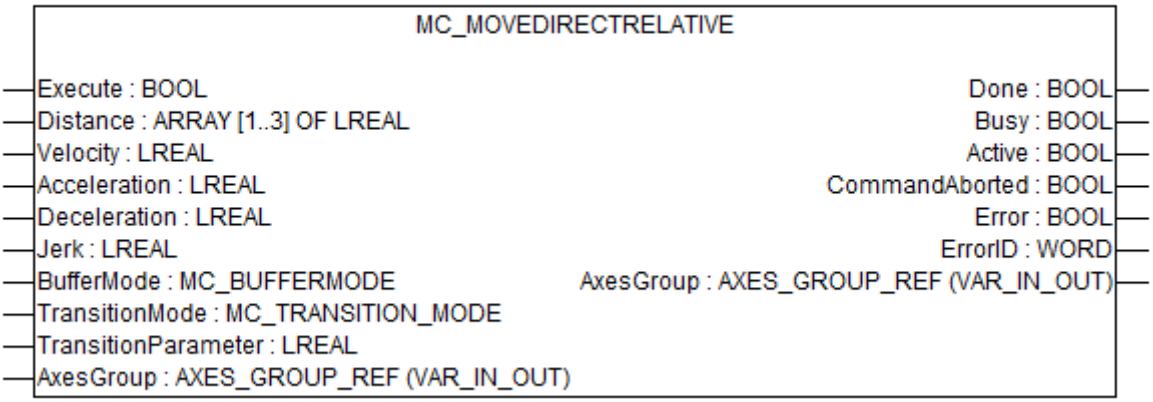


Fig. 524: MC_MoveDirectRelative

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_PathSelect

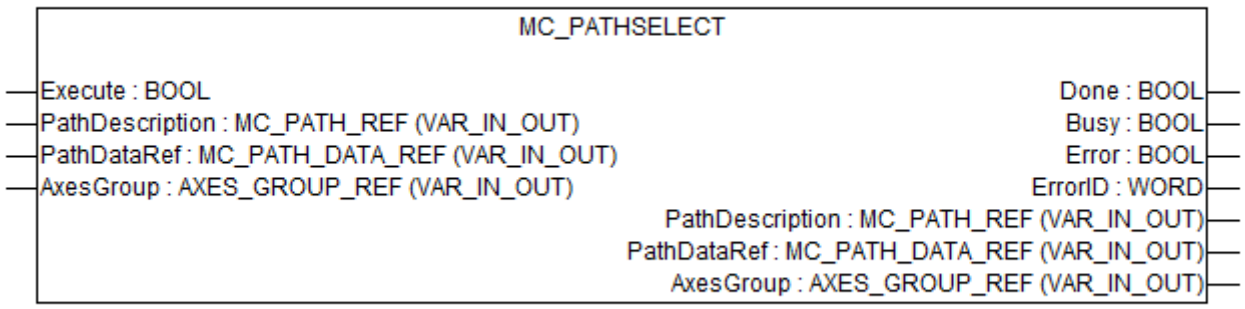


Fig. 525: MC_PathSelect

This function block prepares the relevant path data and makes these available to the system as an output (PathData). Administrative function block.

MC_PATH_DATA_REF and MC_PATH_REF are ABB specific data types.

PathSelect makes data available.This can include:

1.) Starting point of a download of a path profile, as represented in PathData and referenced by PathDescription,

2.) Start to generate a path profile.

This function block applies to the MCS or PCS System, depending which is activated.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

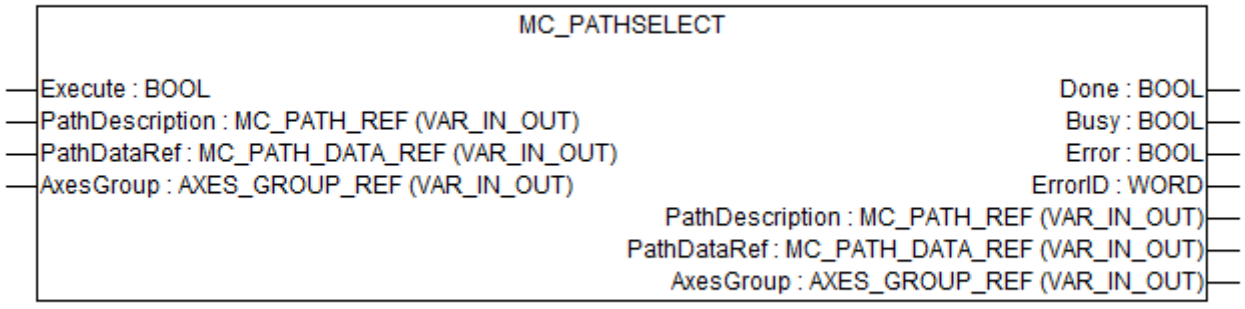


Fig. 526: MC_PathSelect

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

PathDescription Data type: MC_PATH_REF
Reference to the path description.

PathDataRef Data type: MC_PATH_DATA_REF
Reference to the path data which can be prepared by MC_PathSelect.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

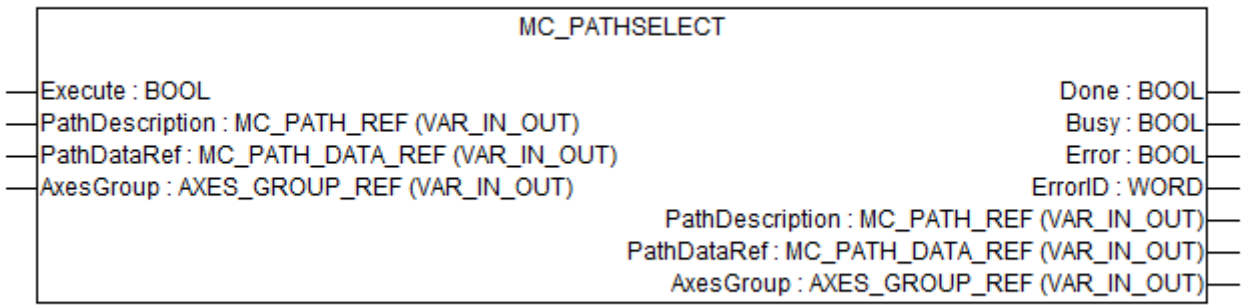


Fig. 527: MC_PathSelect

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_MovePath

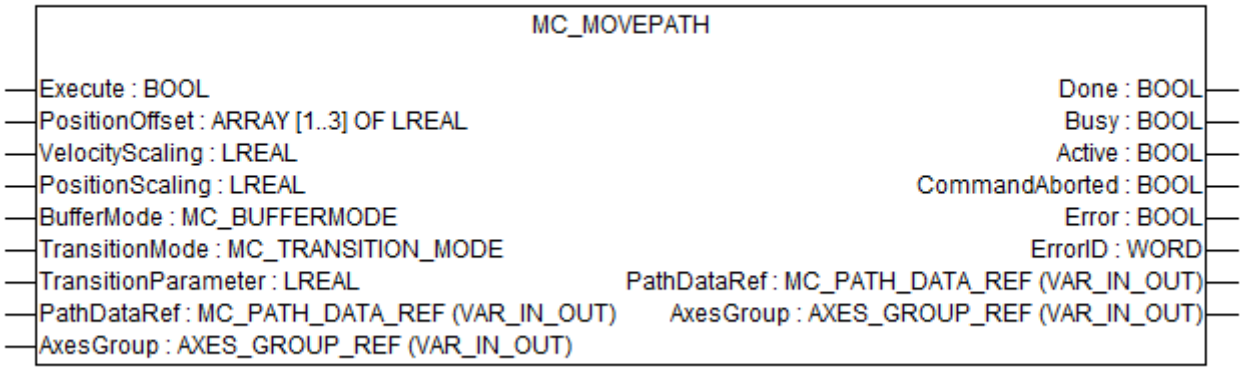


Fig. 528: MC_MovePath

This function block commands an AxesGroup to move according to the path specified in the PathData.

> *This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Explanation: The complete path could be moved with PositionOffset and could be stretched with Position-Scaling. The path velocity could be modified with the value VelocityScaling. The used velocity will be the pathvelocity (V_PATH) multiplied by VelocityScaling. When the path movement is started, the actual position for each axis should be equal to: PathPosition * PositionScaling + PositionOffset

Position-Offset	Position-Scaling	1.Path posi-tion	Actual axis position	Behavior
0	1	0	0	Ok, movement starts at position 0.
0	1	1000	1000	Ok, movement starts at position 1000.
0	2	500	1000	Ok, movement starts at position 1000.
1000	1	0	1000	Ok, movement starts at position 1000.
-1000	1	1000	0	Ok, movement starts at position 0.
-1000	2	500	0	Ok, movement starts at position 0.

Input description

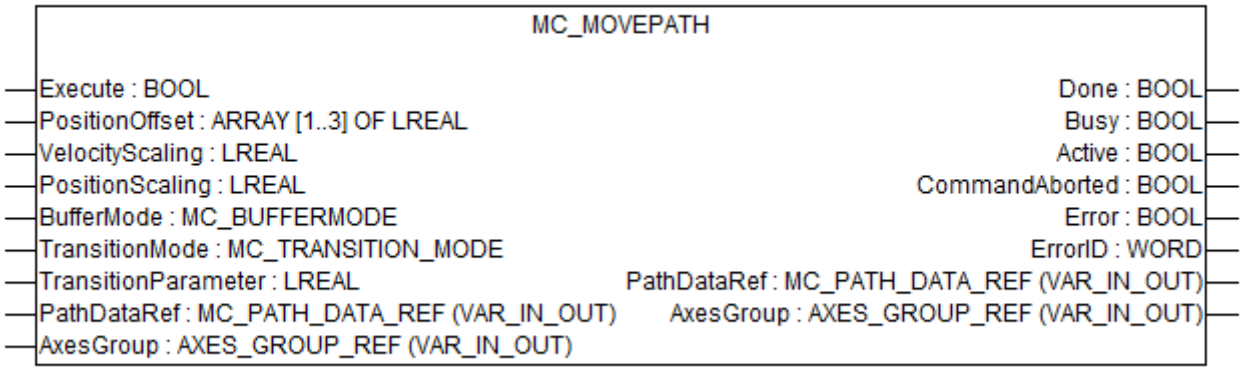


Fig. 529: MC_MovePath

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

PositionOffset Data type: ARRAY [1..3] OF LREAL

Allows to define an offset to the position in PathData. The movement starts at actual position and positionOffset. This position should be the same as the first Path Position.

VelocityScaling Data type: LREAL, default: 1

Allows to increase or decrease the path velocity.

PositionScaling Data type: LREAL, default: 1

Allows to scale the position values defined in PathData.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE

The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL

Additional parameter for the transition mode.

PathDataRef Data type: MC_PATH_DATA_REF
Reference to the path data which can be prepared by MC_PathSelect.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

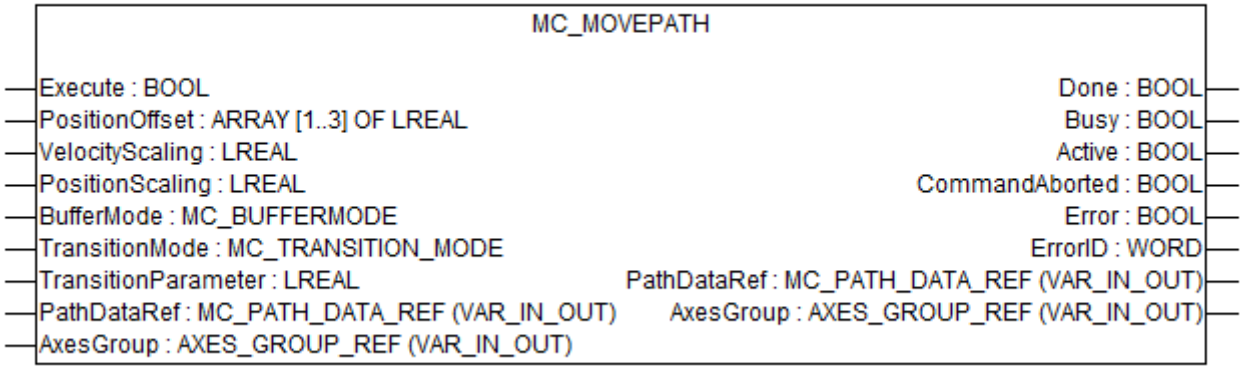


Fig. 530: MC_MovePath

Done Data type: BOOL
Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandA-borted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_SyncGroupToAxis

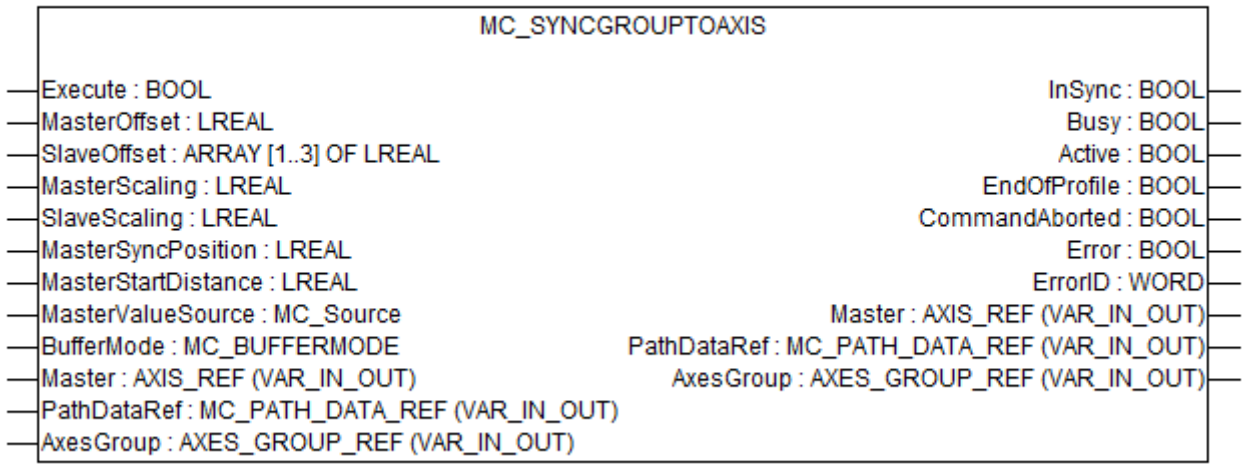


Fig. 531: Function block MC_SyncGroupToAxis

This function block commands an interpolated path movement on an axes group in the applicable coordinate system. The multi axes motion is synchronized with the Master motion like in a cam function.

- *This synchronization of the axes group can be stopped via MC_GroupStop or any other motion command,*
- *The interpolation executed corresponds to MC_MovePath, just that MC_MovePath processes the path data in relation to time while MC_SyncGroupToAxis processes the path data in relation to the master position,*
- *It is possible to start the movement in advance and ramp-in to reach the path at position MasterSyncPosition. This functionality is executed with MasterStartDistance < > 0.*

Input description

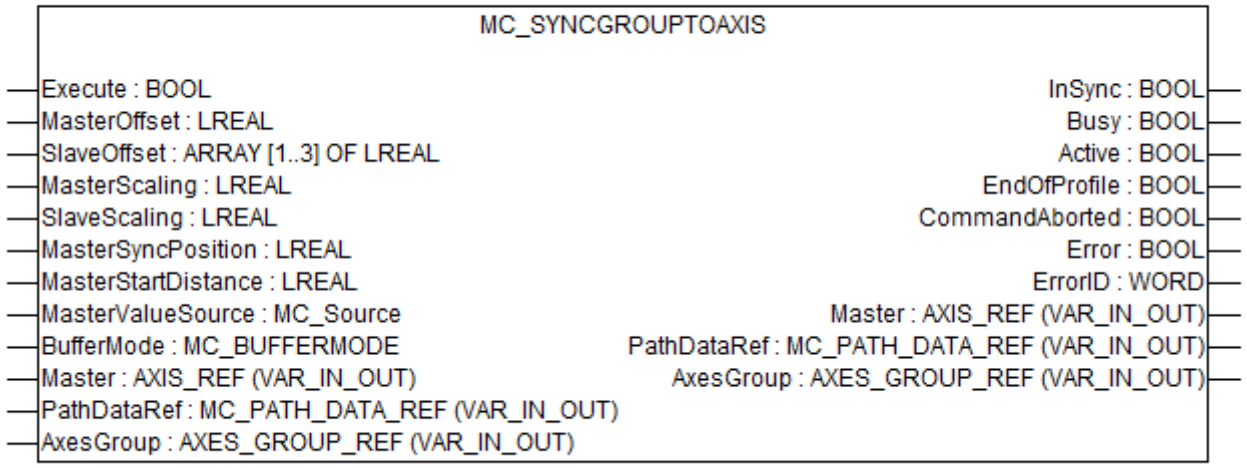


Fig. 532: Function block MC_SyncGroupToAxis

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute

Data type: BOOL

Starts the function block at rising edge.

MasterOffset

Data type: LREAL

Offset of master table. Angular offset of the master shaft to cam.

SlaveOffset

Data type: ARRAY [1..3] OF LREAL

Offset of slave table. The slave will start at the 1. path position + SlaveOffset.

MasterScaling

Data type: LREAL, default: 1.0

Factor for the master profile. From the slave point of view the master overall profile is multiplied by this factor.

SlaveScaling

Data type: LREAL, default: 1.0

Factor for the slave profile. The overall slave profile is multiplied by this factor.

MasterSyncPosition

Data type: LREAL

The position of the master in the path where the group is insync with the master. (If the ‘MasterSyncPosition’ does not exist, at the first point of the path the master and slave are synchronized).

The inputs acceleration, deceleration and jerk are not added here.

MasterStartDistance

Data type: LREAL

The master distance for the slave to start to synchronize to the master.

MasterValueSource

Data type: MC_SOURCE

Defines the source for synchronization:

mcSetValue - Synchronization on master set value.

mcActualValue - Synchronization on master actual value.

BufferMode

Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

Master

Data type: AXIS_REF

Reference to the master axis.

PathDataRef Data type: MC_PATH_DATA_REF
Reference to the path data which can be prepared by MC_PathSelect.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

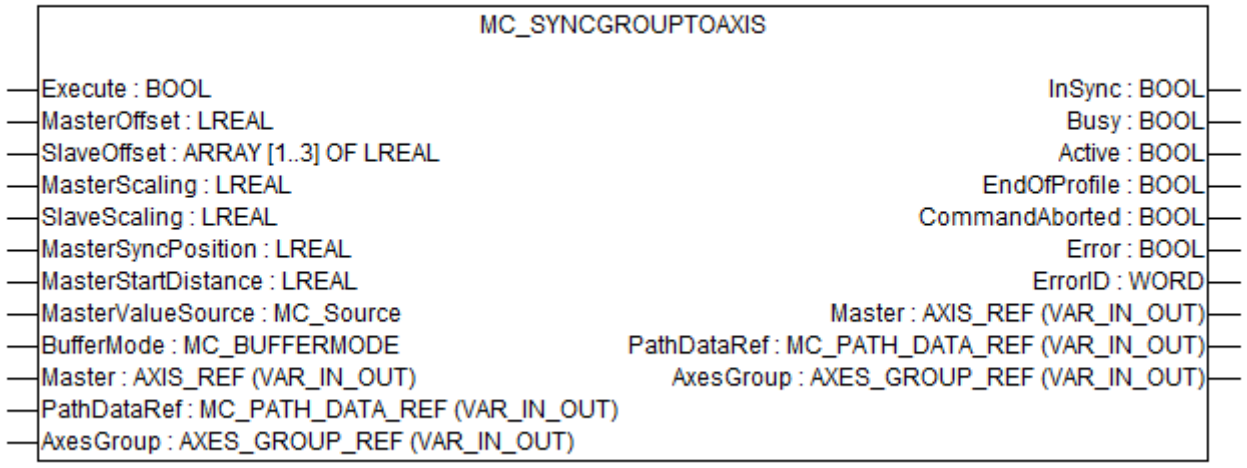


Fig. 533: Function block MC_SyncGroupToAxis

InSync Data type: BOOL
The axes group follows the master axis.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_SyncAxisToGroup

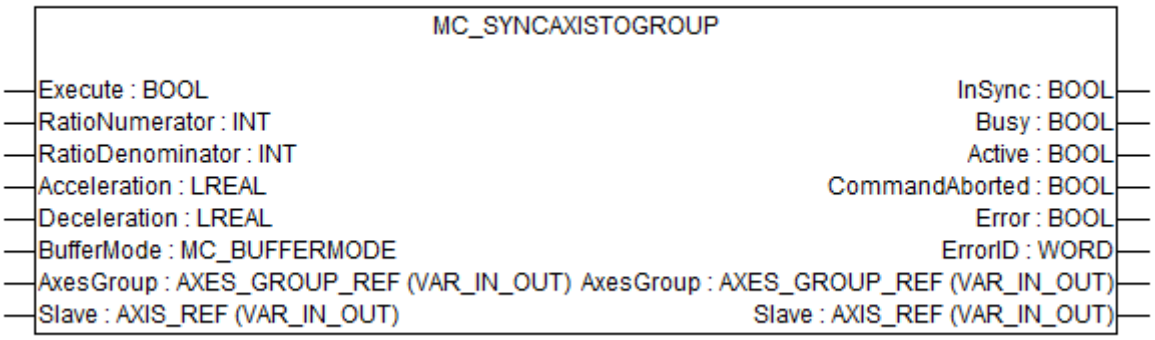


Fig. 534: MC_SyncAxisToGroup

This function block maps a single axis to a group. The single axis output represents the path length progression of the axes group. There is the ability to set a ratio between group and single axis.

- *This function block equals the mileage counter (odometer) in a car of the group and shows this via the Slave axis,*
- *The slave ramps up to the ratio of the path speed and locks in position when this is reached,*
- *The gearing ratio can be changed while function block is running, using a consecutive call of the function block,*
- *InSync is set the first time the ratio is reached,*
- *After being InSync, a position locking or just a speed locking is system specific. The function block is stopped by issuing a single axis function block.*

This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated.

This function block has to be called within the real-time task.

Input description

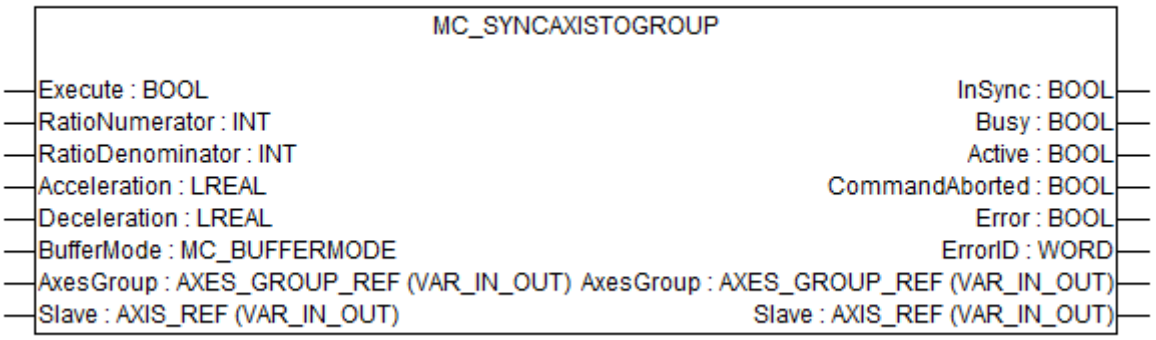


Fig. 535: MC_SyncAxisToGroup

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

RatioNumerator Data type: INT
Gear Ratio Numerator.

RatioDenominator Data type: INT
Gear Ratio Denominator.

Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Slave Data type: AXIS_REF
Reference to the axis (real or virtual).

Output description

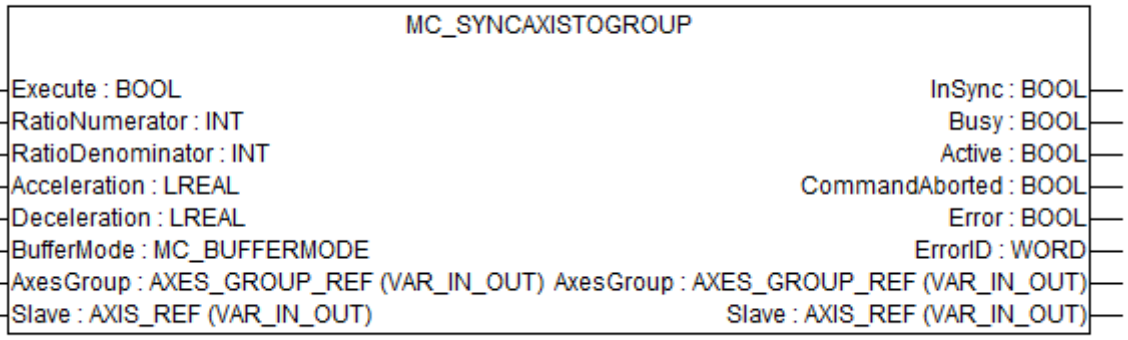


Fig. 536: MC_SyncAxisToGroup

InSync Data type: BOOL
The (virtual) slave generates valid values.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

COMC_GROUP_CARTESIAN

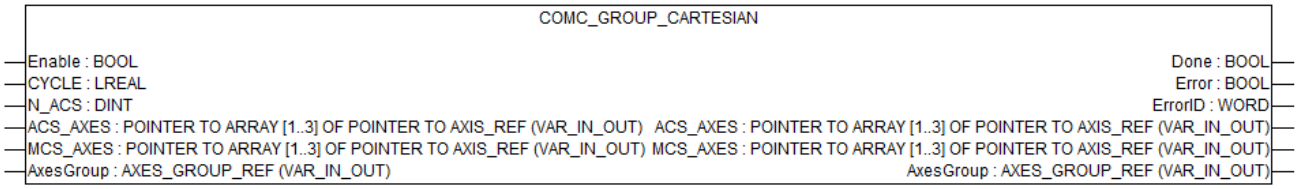


Fig. 537: Function block COMC_GROUP_CARTESIAN

Table 194: General information

Available as of runtime system	V1.2 and above
Included in library	CoordinatedMotion_AC500_V21.LIB
Type	Function block with historical values

This function block has to be called every cycle and at least once before any MC… function block is activated.

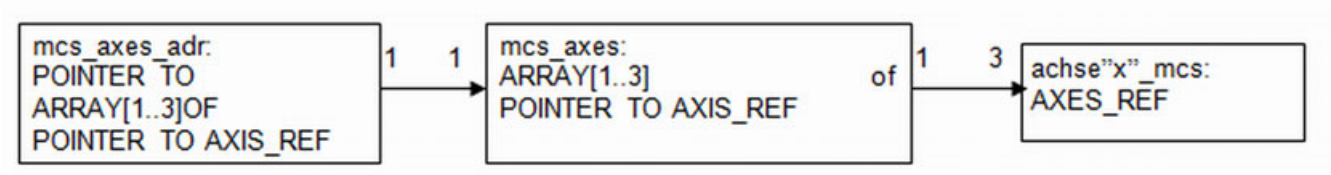
The MCS axes form a cartesian system according to the “right hand rule”. These axes are fix, so it is not possible to add or remove an axis. When just 2 dimensions are needed, a dummy-axis which runs on simulation should be created.

The addresses should be calculated as follows:

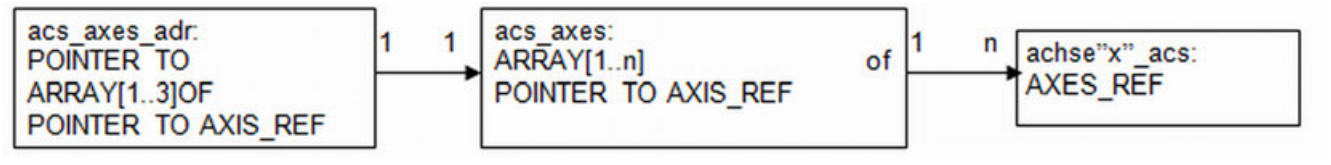
```
acs_axes[1]:=ADR(achse1_acs);
acs_axes[2]:=ADR(achse2_acs);
acs_axes[3]:=ADR(achse3_acs);
acs_axes_adr:=ADR(acs_axes);
mcs_axes[1]:=ADR(achse1_mcs);
mcs_axes[2]:=ADR(achse2_mcs);
mcs_axes[3]:=ADR(achse3_mcs);
mcs_axes_adr:=ADR(mcs_axes);
```

This should be done every cycle as on an online change, addresses of variables might be changed. The actual address has to be provided to the block in every cycle.

The different elements are linked by ADR operator.



The number of ACS axes is flexible, although an array with 3 elements is used as type. A larger array could be used instead and the number of elements be provided at N_ACS.



Input description

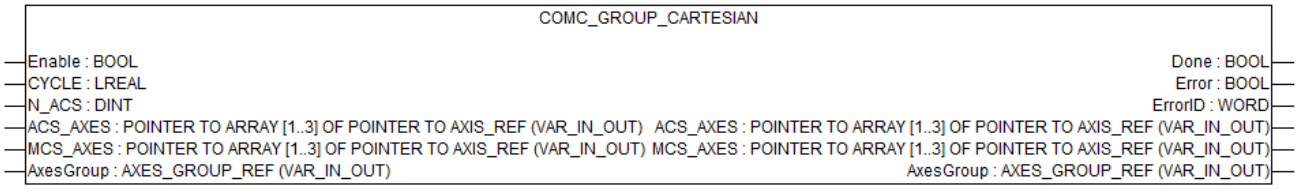


Fig. 538: Function block COMC_GROUP_CARTESIAN

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Release of function block.

CYCLE Data type: INT

Cycle time of the PLC task in ms.

N_ACS Data type: INT

Number of ACS Axes

ACS_AXES Data type: POINTER TO ARRAY[1..3] OF POINTER TO AXIS_REF

Access to the machine axes, the number is variable and the array might have a different number of elements. It depends on the used coordinate transformation

MCS_AXES Data type: POINTER TO ARRAY[1..3] OF POINTER TO AXIS_REF

Access to the Cartesian axes. These have to be 3 axes, for X, Y, Z coordinate in a right handed coordinate system

GROUP Data type: AXES_GROUP_REF

Group reference

Output description

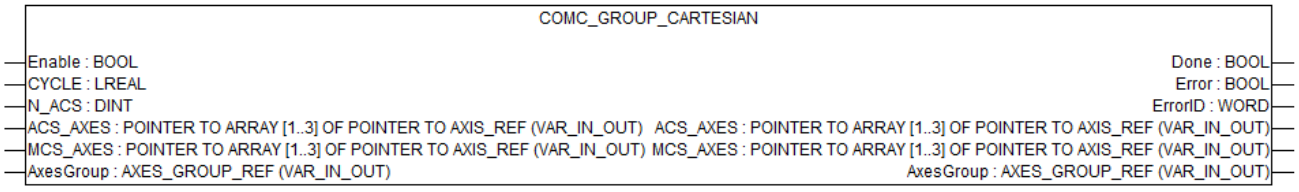


Fig. 539: Function block COMC_GROUP_CARTESIAN

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

COMC_TeachCartesianTransformation

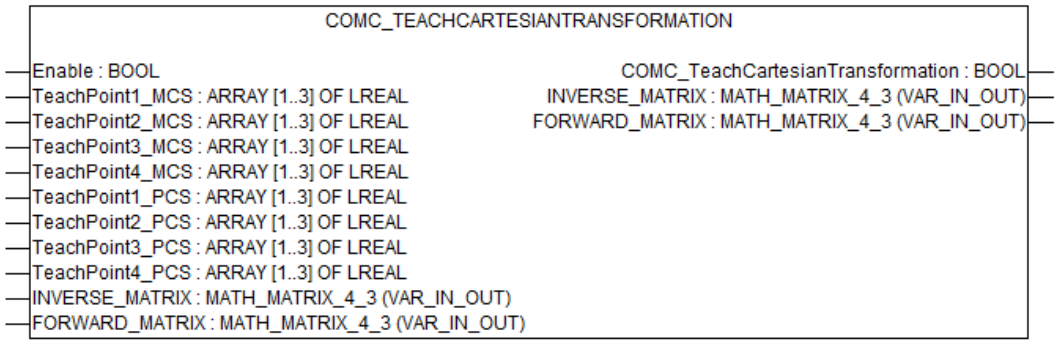


Fig. 540: Function COMC_TeachCartesianTransformation

The function calculates matrices to be used for Cartesian coordinate transformation.

> *Four teach points from both coordinate systems are necessary to do so. The points need to be linearly independent, which means it is not possible to have 3 points on a line or all 4 points on a plane. As result, the Function will calculate FORWARD_MATRIX and INVERSE_MATRIX which could be used with Funciton Block “MCA_SetCoordinateTransformation” to switch over from MCS to PCS. The Function has a BOOL value as return value which is TRUE in case of success. When FALSE is returned, the condition of linear independent points needs to be checked.*

Input description

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Activates the Funcion.

TeachPoint1_MCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in MCS. Refers to the same point as TeachPoint1_PCS in PCS coordinates.

TeachPoint2_MCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in MCS. Refers to the same point as TeachPoint2_PCS in PCS coordinates.

TeachPoint3_MCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in MCS. Refers to the same point as TeachPoint3_PCS in PCS coordinates.

TeachPoint4_MCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in MCS. Refers to the same point as TeachPoint4_PCS in PCS coordinates.

TeachPoint1_PCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in PCS.

TeachPoint2_PCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in PCS.

TeachPoint3_PCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in PCS.

TeachPoint4_PCS
Data type: ARRAY[1..3] OF LREAL
Teach point for x, y, z values in PCS.

INVERSE_MATRIX
Data type: MATRIX_4_3
Result of the Function, a matrix with 4 lines and 3 columns, to be used for transform PCS to MCS coordinates.

FORWARD_MATRIX
Data type: MATRIX_4_3
Result of the Function, a matrix with 4 lines and 3 columns, to be used for transform MCS to PCS coordinates.

1.5.9.7.2 Transformation function blocks

Although the transformation function blocks are administrative function blocks, they can be buffered. Additional transformation function blocks are available as ABB specific function blocks MCA_.... ↪ *Chapter 1.5.9.7.3 “ABB specific function blocks” on page 3008*

MC_SetCartesianTransform MCS to PCS

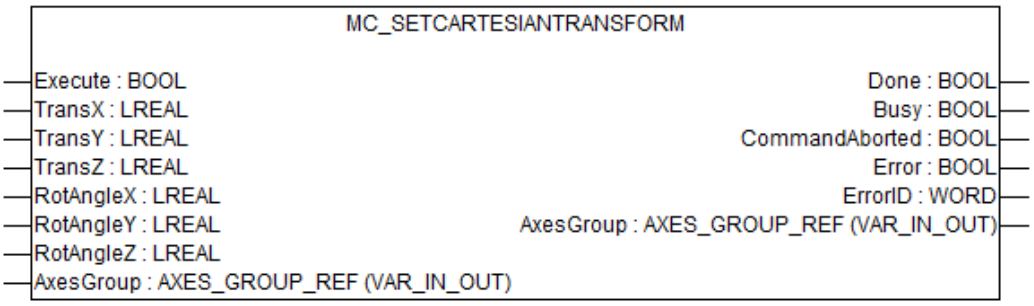


Fig. 541: Function block MC_SetCartesianTransform

This function block sets a Cartesian transformation between the MCS and PCS.

The transformation will be activated on the next group movement, e.g. MC_MoveLinearAbsolute. When the block is activated, it first will be “Busy”. The Group will activate the transformation on the next movement, the block changes to “Done”. The transformation will be used until an other transformation is activated.

- First, the rotation is applied in Z/Y/X direction and then the translation, with respect to the already rotated coordinate system. When it is more reasonable to apply first the translation, this could be done with feeding the translation values to MC_SetCoordinateTransform. This function block (or MCA_SetCoordinateTransformation) is a precondition for the use of MC_SetCartesianTransform.
- De-selection of PCS can be done by a execution of this function block with {TransX, TransY, TransZ, RotAngleX, RotAngleY, RotAngleZ }={0, 0, 0, 0, 0, 0} as translation and rotation input values.
- The values could as well be modified dynamically.
- A precondition for the use of MC_SetCartesianTransform is to activate a PCS first by MC_SetCoordianteTransform or MCA_SetCoordinateTransformation. A neutral transformation could be used for this which is available from the library as CoordTransform_neutral.
- The transformation is combined with a block using “CoordTransform” but it is not possible to combine it with a different block using translation vectors and rotation angles. In combination with CoordTransform, the CoordTransform is executed first.

The transformation will be activated on the next group movement, e.g. MC_MoveLinearAbsolute. When the function block is activated, it first will be “Busy”. The Group will activate the transformation on the next movement, the function block changes to “Done”. The transformation will be used until an other transformation is activated.

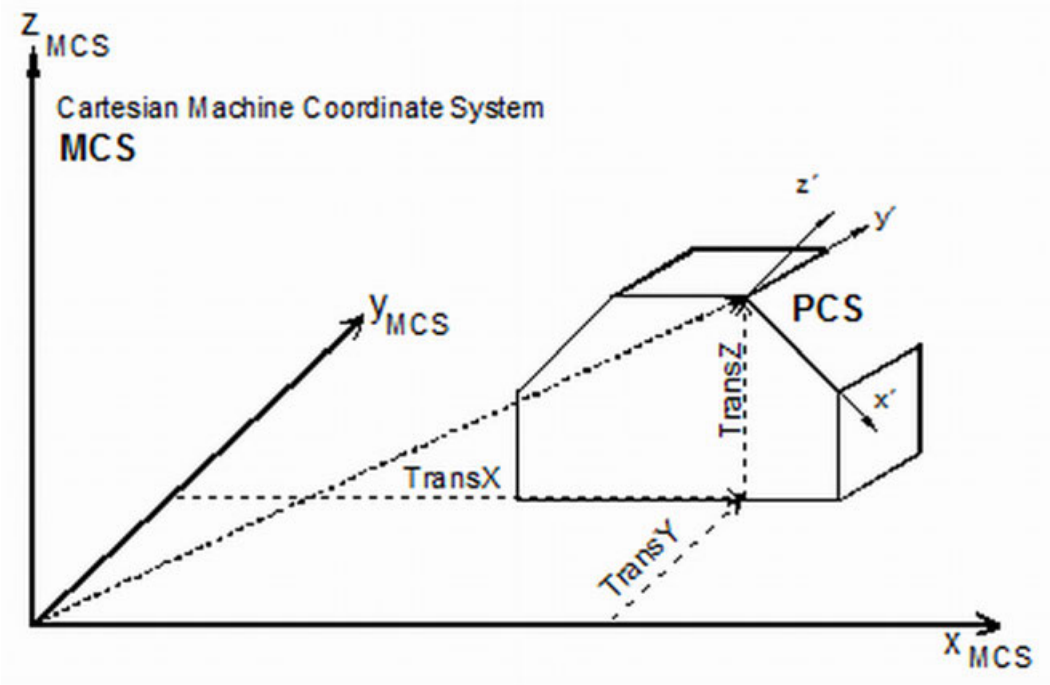


Fig. 542: Definition of the translation

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Example of the definition fo the rotation

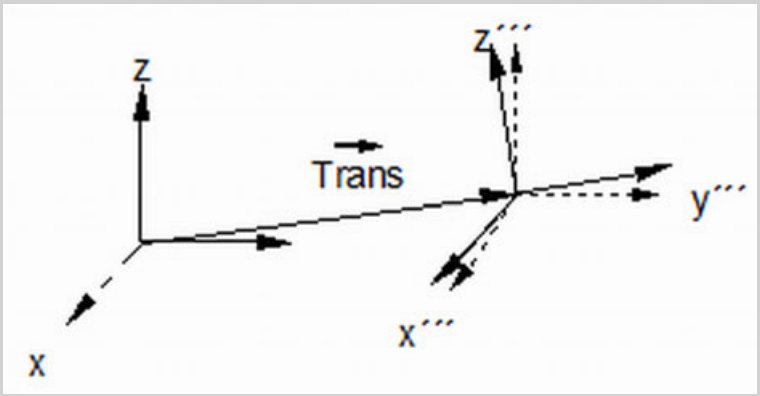


Fig. 543: The rotation is defined by a subsequent rotation around every coordinate direction beginning with the Z-direction.

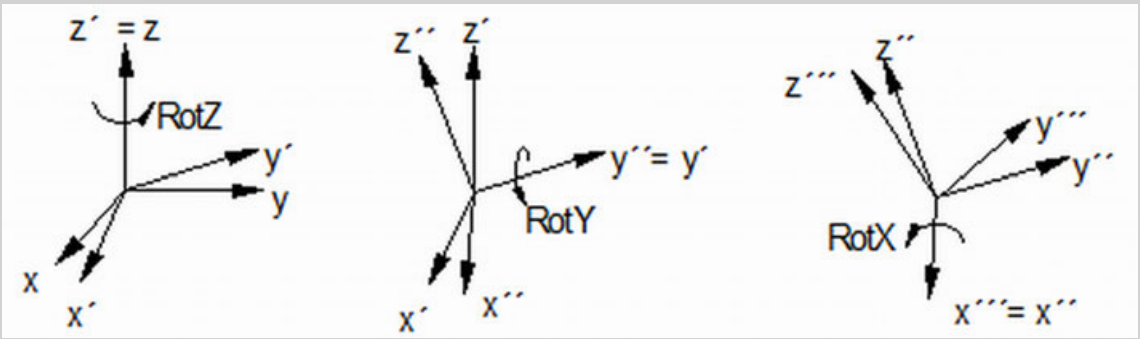


Fig. 544: Definition of the rotation

Input description

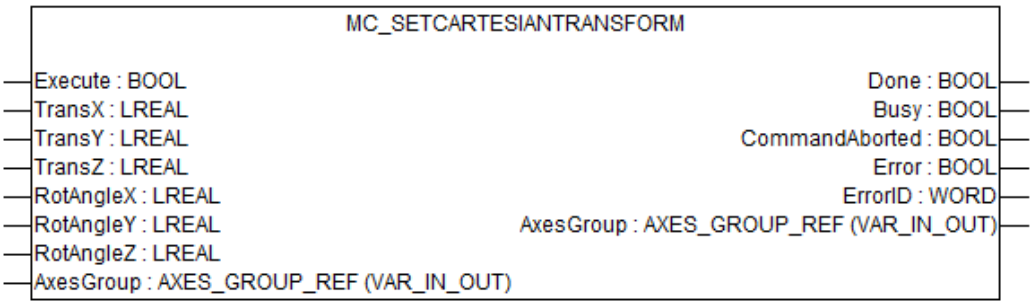


Fig. 545: Function block MC_SetCartesianTransform

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL
Starts the function block at rising edge.

TransX Data type: LREAL
X-component of Translation Vector.

TransY Data type: LREAL
Y-component of Translation Vector.

TransZ Data type: LREAL
Z-component of Translation Vector.

RotAngleX Data type: LREAL, unit: rad
Rotation angle in X-direction.

RotAngleY Data type: LREAL, unit: rad
Rotation angle in Y-direction.

RotAngleZ Data type: LREAL, unit: rad
Rotation angle in Z-direction.

AxesGroup Data type: AXES_GROUP_REF
Reference to a group of axes.

Output description

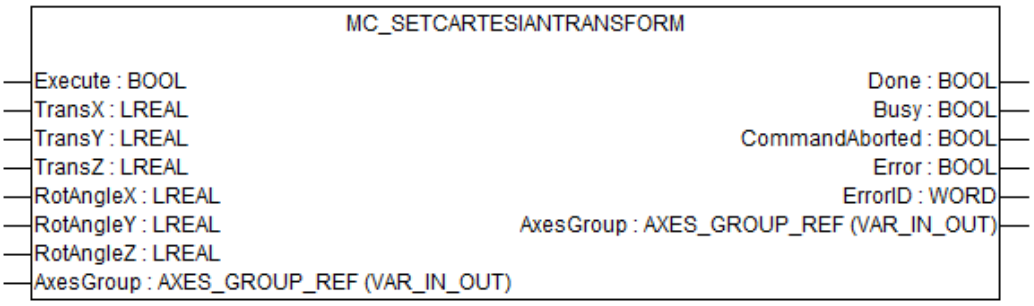


Fig. 546: Function block MC_SetCartesianTransform

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_SetCoordinateTransform MCS to PCS

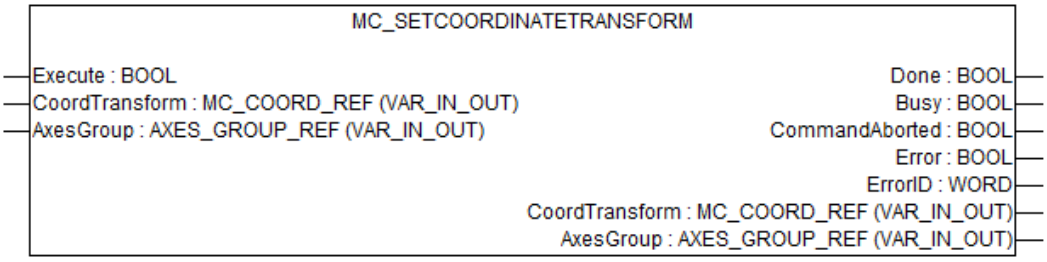


Fig. 547: MC_SetCoordinateTransform

This function block sets a coordinate transformation between the MCS and PCS.

- CoordTransform refers to a coordinate transformation including the parameters. The details of the transformation and of the parameters are outside the scope of PLCopen.
- The system may supports a neutral transformation. With activating the neutral transformation the axes are referenced in the MCS system again.
- The Transformation could as well be dynamic.
- This function block does not start a movement (administrative function block). The movement is initiated by a command in PCS.

The transformation will be activated on the next group movement, e.g. MC_MoveLinearAbsolute. When the block is activated, it first will be “Busy”. The Group will activate the transformation on the next movement, the block changes to “Done”. The transformation will be used until an other transformation is activated.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

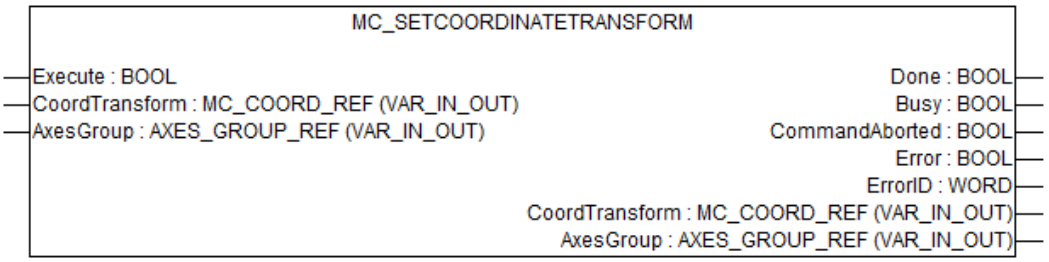


Fig. 548: MC_SetCoordinateTransform

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute Data type: BOOL

Starts the function block at rising edge.

CoordTransform Data type: MC_COORD_REF

Reference to a Coordinate Transformation. ABB specific datatype.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

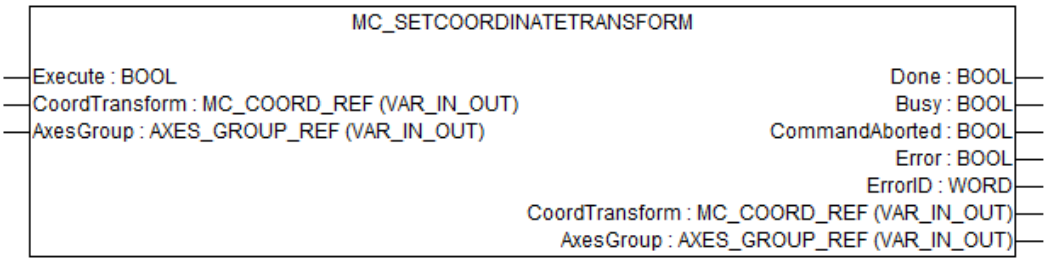


Fig. 549: MC_SetCoordinateTransform

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MC_ReadCartesianTransform MCS to PCS

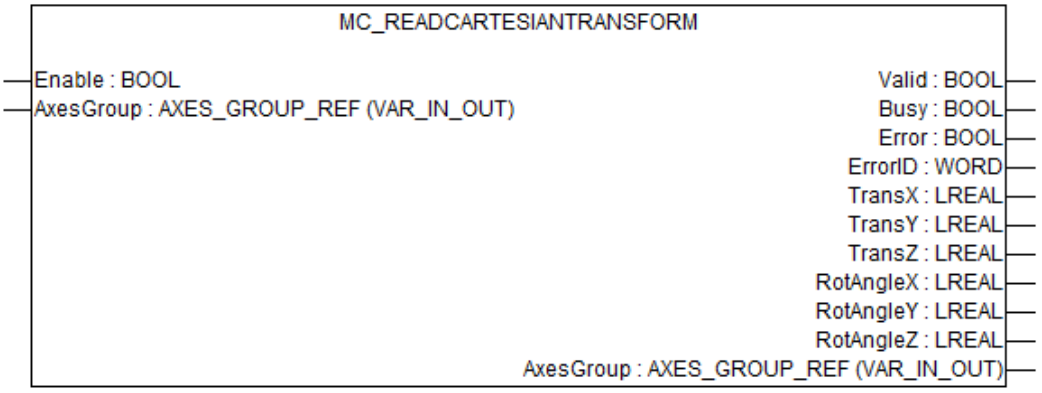


Fig. 550: Function block MC_ReadCartesianTransform

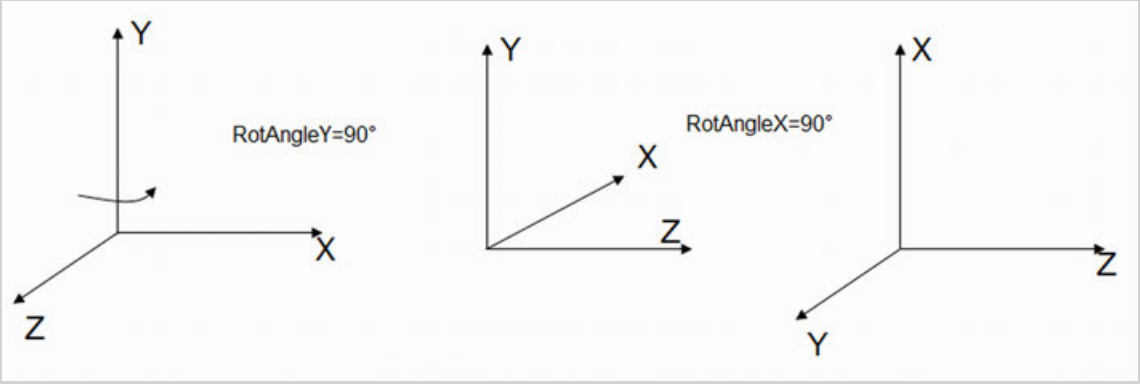
This function block reads the parameter of the cartesian transformation that is active between the MCS and PCS. .It combines the used matrix (MC_SetCoordinateTransform or MCA_SetCoordinateTransformation) and rotation angles (MCA_SetDynamicFollower or MC_SetCartesianTransform) to a rotation angles and translation vector result.

> *This result is just valid when an orthogonal matrix has been used. Any matrix with non-orthogonal vectors included could not be represented this way.*

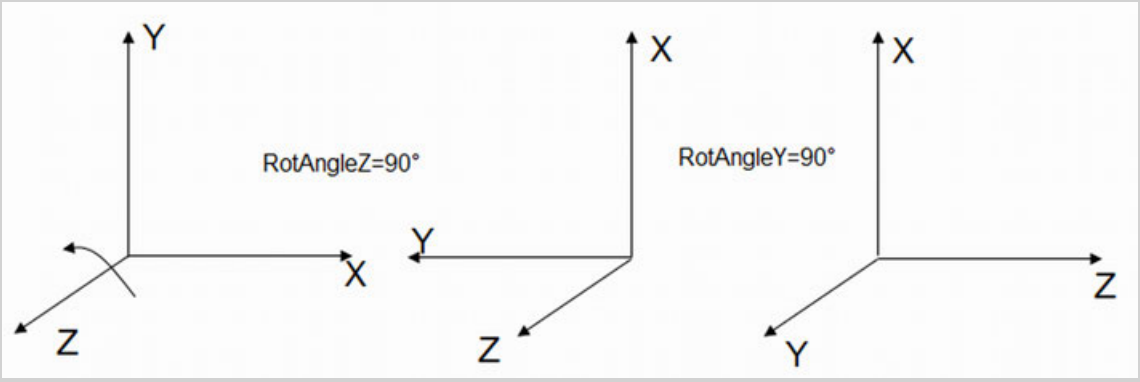
- When a dynamic transformation is activated, the function block will as well display the actual dynamic values.
- When different transformations are active, the function block will give the combined result.
- The function block will give the FORWARD (from MCS to PCS) transformation.
- Several rotations are possible to achieve the same result, so it might happen that the angles which are displayed do not match the given angel values at function block MCA_SetDynamicFollower. The possible rotation angles to achieve the same result are not unique.

Example: The following rotations achieve the same result:

	1.	2.
RotAngleX	0	90°
RotAngleY	90°	90°
RotAngleZ	90°	0



or



This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

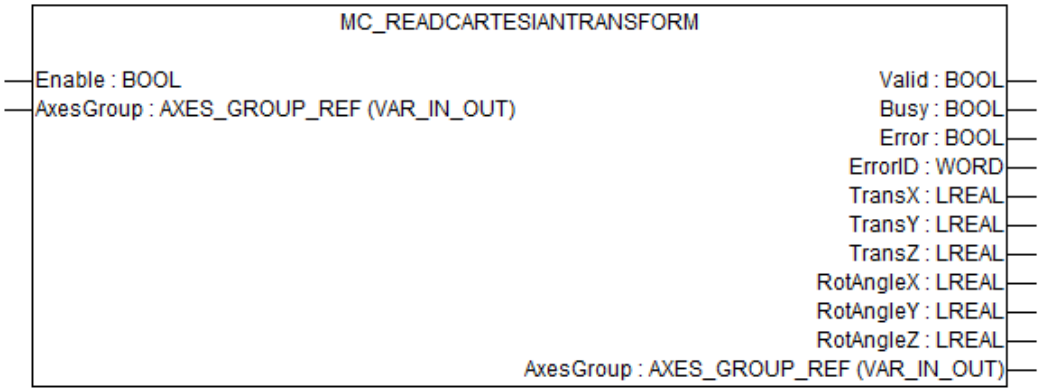


Fig. 551: Function block MC_ReadCartesianTransform

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the cartesian transformation parameter of the axes group continuously while enabled.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

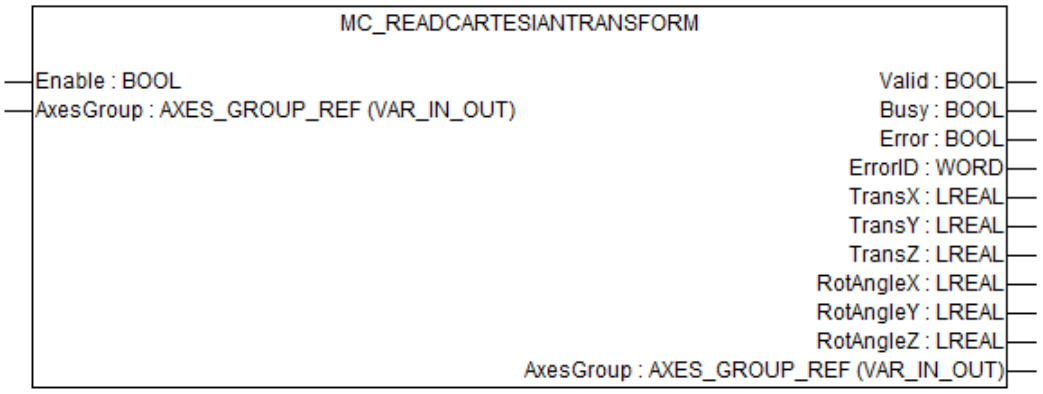


Fig. 552: Function block MC_ReadCartesianTransform

Valid Data type: BOOL

True if valid outputs are available.

Busy Data type: BOOL
The function block is not finished.

Error Data type: BOOL
Signals that an error has occurred within the function block.

ErrorID Data type: WORD
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

TransX Data type: LREAL
X-component of Translation Vector.

TransY Data type: LREAL
Y-component of Translation Vector.

TransZ Data type: LREAL
Z-component of Translation Vector.

RotAngleX Data type: LREAL, unit: rad
Rotation angle in X-direction.

RotAngleY Data type: LREAL, unit: rad
Rotation angle in Y-direction.

RotAngleZ Data type: LREAL, unit: rad
Rotation angle in Z-direction.

MC_ReadCoordinateTransform MCS to PCS

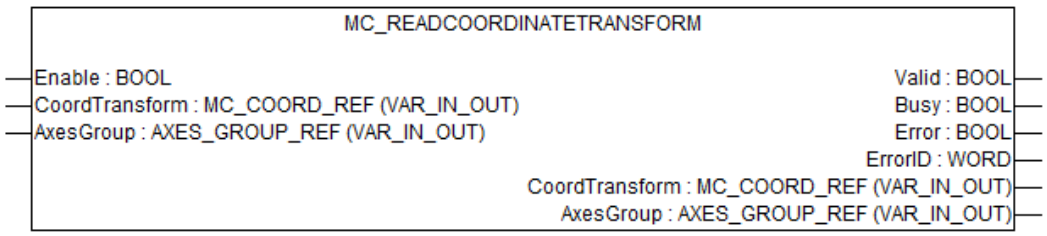


Fig. 553: Function block MC_ReadCoordinateTransform

This function block reads the coordinate transformation that is active between the MCS and PCS.

- When a dynamic transformation is activated, the function block will as well display the actual dynamic values.
- When different transformations are active, the function block will give the combined result.
- The function block will give the FORWARD (from MCS to PCS) and INVERSE (from PCS to MCS) transformation.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

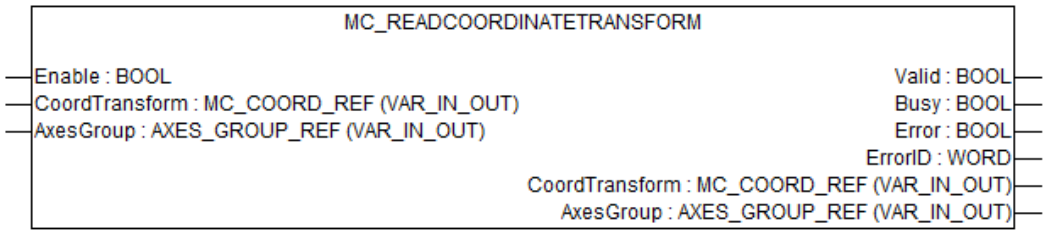


Fig. 554: Function block MC_ReadCoordinateTransform

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Get the actual coordinate transformation reference of the axes group continuously while enabled.

CoordTransform Data type: MC_COORD_REF

Reference to a Coordinate Transformation. ABB specific datatype.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

Valid Data type: BOOL

True if valid outputs are available.

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

1.5.9.7.3 ABB specific function blocks

The ABB specific function blocks follow the general rules defined by PLCopen and implement some additional features.

MCA_MoveHelixRelative

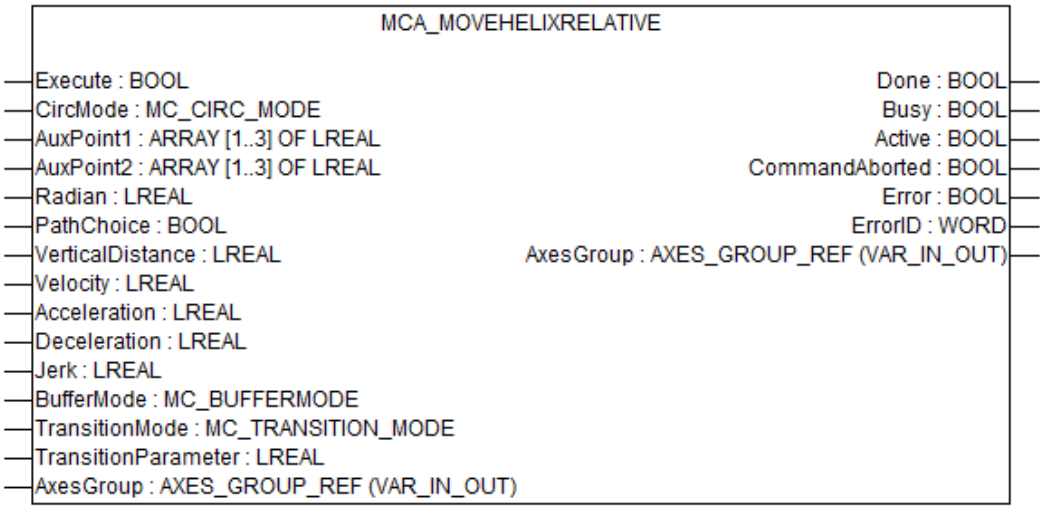


Fig. 555: Function block MCA_MoveHelixRelative

This function block commands an interpolated circular movement on an axes group from the actual position of the TCP. Two auxiliary points (meaning depending on applied mode, see below) are defined in the specified coordinate system relatively to the starting point. The 1. auxiliary point has a meaning accordingly to the CircMode input while the 2. auxiliary point just represents any point on the circle to define the plane, it does not define the end point.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

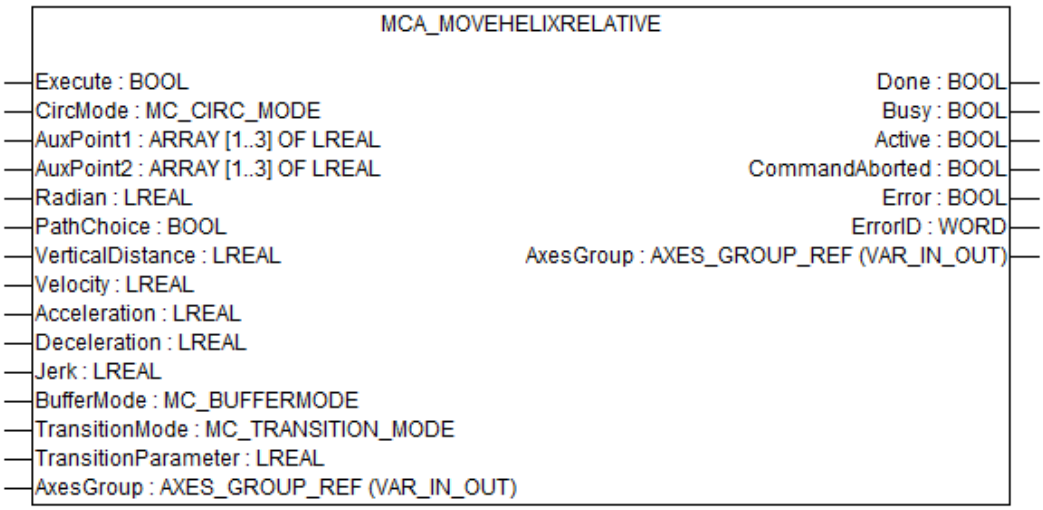


Fig. 556: Function block MCA_MoveHelixRelative

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Execute

Data type: BOOL

Starts the function block at rising edge.

CircMode

Data type: MC_CIRC_MODE

Specifies the meaning of the input signals 'AuxPoint' and 'CircDirection'.

- MC_BORDER:
  'AuxPoint' defines a point on the circle which is crossed on the path from the starting to the end point.
- MC_CENTER:
  'AuxPoint' defines the center point of the circle.
- MC_RADIUS:
  'AuxPoint' defines the spearhead point of the perpendicular of the circle plane according to the rule of right thumb. The radius of the circle is the length of the vector.

In the function block MC_MoveCircularAbsolute, the points are specified absolutely, i.e. the perpendicular vector begins in the origine and ends in the spearhead point specified at the input signal 'AuxPoint'.

AuxPoint1

Data type: ARRAY [1..3] OF LREAL

Array of positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

AuxPoint2 Data type: ARRAY [1..3] OF LREAL

Array of positions for each dimension in the coordinate system specified by the input signal CoordSystem. These positions are defined relatively to the according positions of the starting point.

Radian Data type: LREAL

The radian measure to be moved. This allows to move > 360° with a single block.

PathChoice Data type: BOOL

On mode MC_RADIUS, PathChoice=TRUE defines that Auxpoint2 is to be reached on the longer sector of the circle. In other modes, the input is ignored.

VerticalDistance Data type: LREAL

Distance to be move for the 3. direction. A basic circle is moved on a plane and with VerticalDistance <>0, an additional movement vertical to the plane is performed which combines to a helix.

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Deceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the deceleration (decreasing energy of the motor).

Jerk Data type: LREAL, range: > 0, unit: u/s^3

Value of the Jerk.

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported

Defines the behavior of the axis.

TransitionMode Data type: MC_TRANSITION_MODE

The realization just supports by default a transition starting with the actual velocity.

TransitionParameter Data type: LREAL

Additional parameter for the transition mode.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

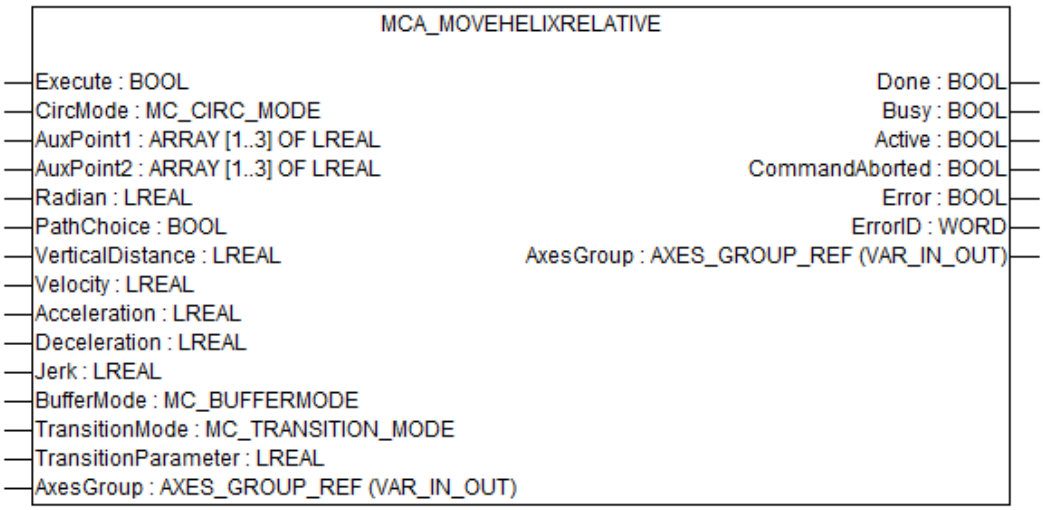


Fig. 557: Function block MCA_MoveHelixRelative

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Auxpoint1 and Auxpoint2

The Auxpoint1 and Auxpoint2 define a plane at which the circle is to be moved. The positive direction is determined by the “right hand rule”. When the thumb points in direction of the perpendicular vector, the fingers show the positive direction.

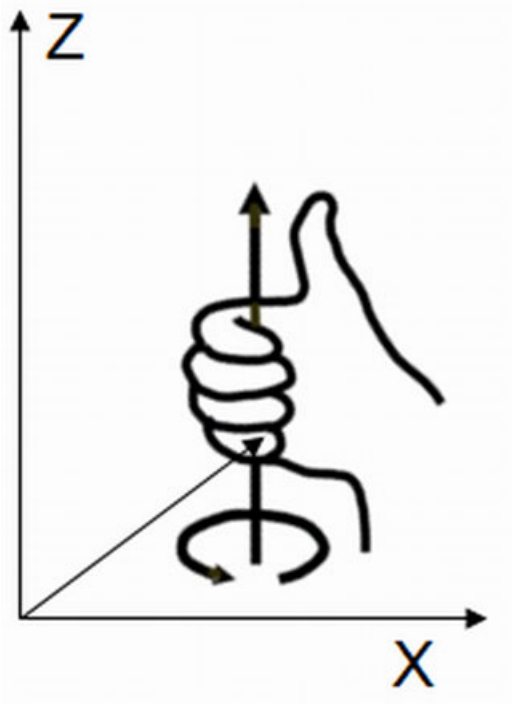
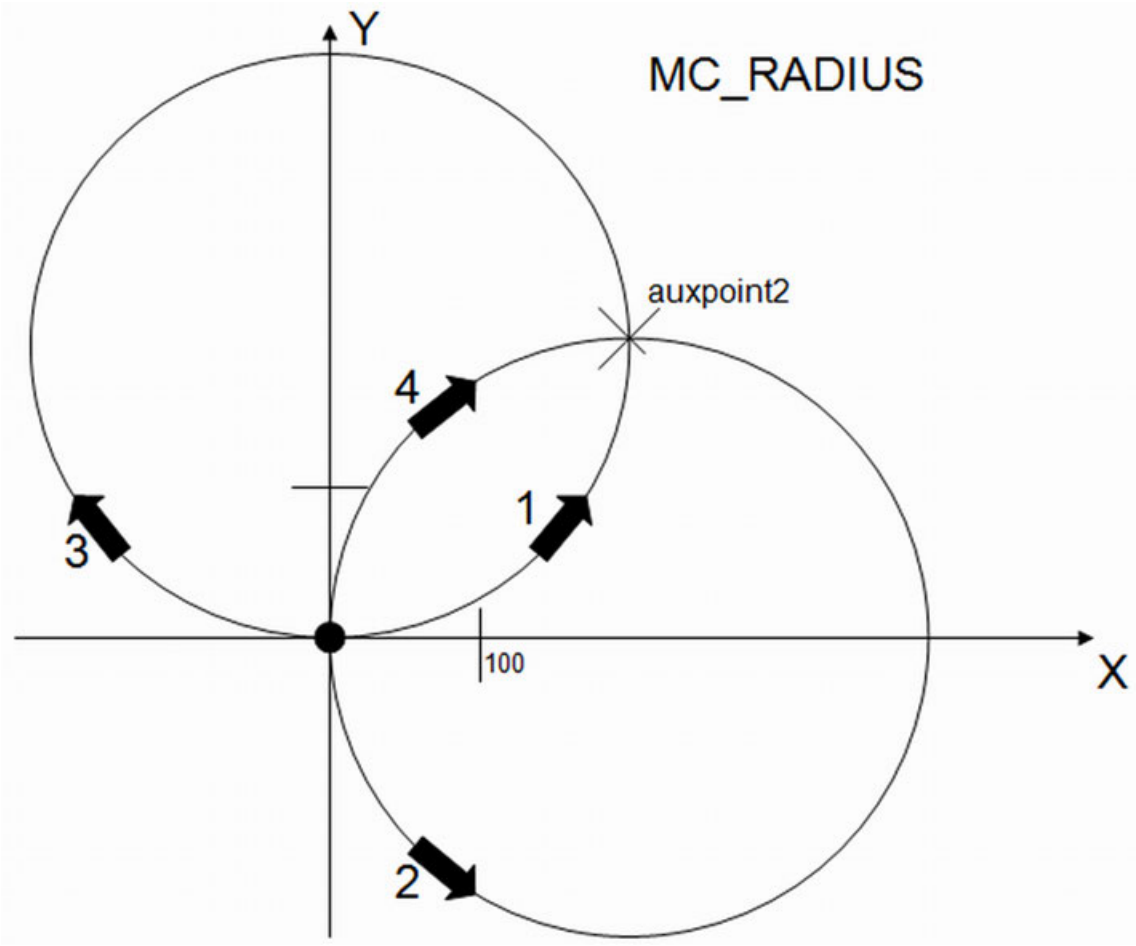


Fig. 558: Right hand rule

Examples for different input parameters with start at position 0/0/0

Circ-Mode=MC_RADIUS



With CircMode=MC_RADIUS, Auxpoint2 is a point at the circle and Auxpoint1 a vector which is perpendicular of the circle plane according to the rule of right thumb. The radius of the circle is the length of Auxpoint1. Two different cricles are possible.

- PathChoice = FALSE:: when moving in positive direction, the Auxpoint2 is reached on the short sector.
- PathChoice = TRUE: when moving in positive direction, Auxpoint2 is reached on the longer sector.

The direction is determined by the right thumb rule according to the perpendicular vector (Auxpoint1)

MC_RADIUS	PathChoice	Radian	Auxpoint1	Auxpoint2
1	FALSE	2 PI	0,0,200	200,200,0
2	TRUE	2 PI	0,0,200	200,200,0
3	FALSE	- 2 PI	0,0,200	200,200,0
4	TRUE	- 2 PI	0,0,200	200,200,0

Circ-Mode=MC_CENTER

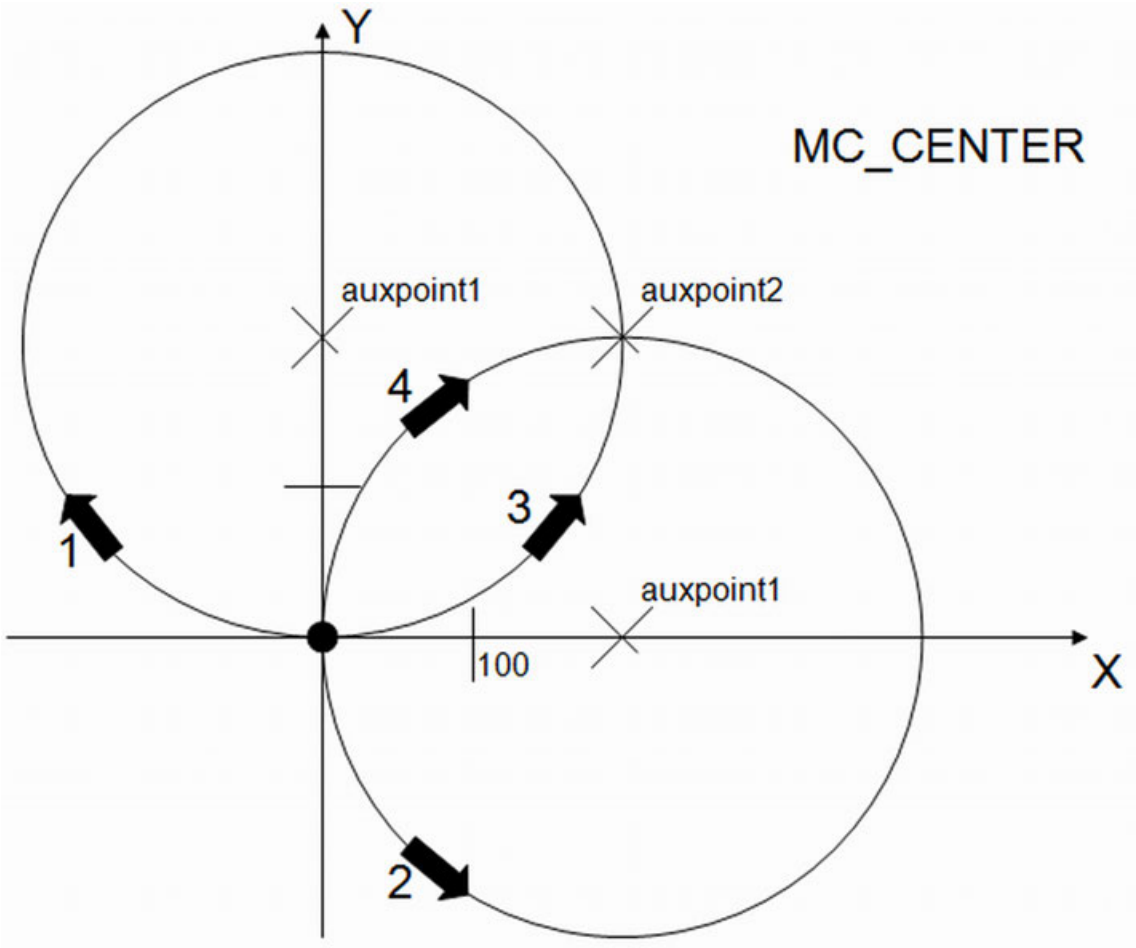


Fig. 559: CircMode=MC_CENTER

With CircMode=MC_CENTER, Auxpoint2 is a point at the circle and Auxpoint1 is the center point. Both are relative to the start point. The circle plane is defined by Auxpoint1 x Auxpoint2

MC_CENTER	PathChoice	Radian	Auxpoint1	Auxpoint2
1	-	2 PI	0,200,0	200,200,0
2	-	2 PI	200,0,0	200,200,0
3	-	- 2 PI	0,200,0	200,200,0
4	-	- 2 PI	200,0,0	200,200,0

Circ-Mode=MC_BORDER

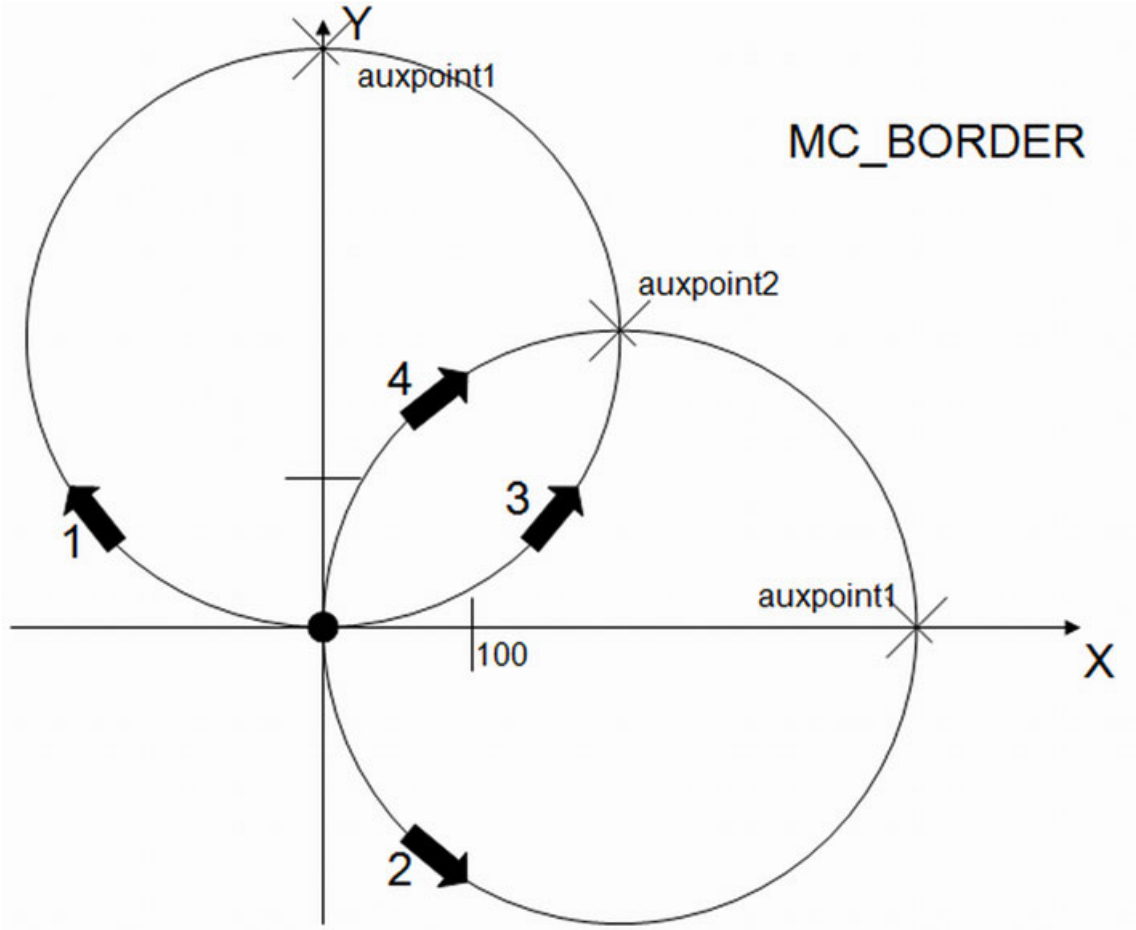


Fig. 560: CircMode=MC_BORDER

With CircMode=MC_BORDER, Auxpoint1 and Auxpoint2 are points at the circle. The circle plane is defined by Auxpoint1 x (Auxpoint1-Auxpoint2).

MC_BORDER	PathChoice	Radian	Auxpoint1	Auxpoint2
1	-	2 PI	0,400,0	200,200,0
2	-	2 PI	400,0,0	200,200,0
3	-	- 2 PI	0,400,0	200,200,0
4	-	- 2 PI	400,0,0	200,200,0

MCA_PathEvent

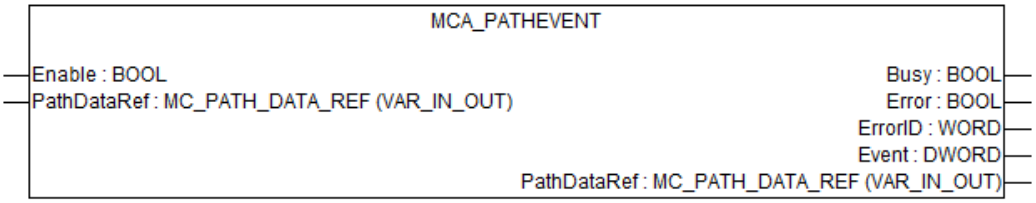


Fig. 561: Function block MCA_PathEvent

When moving the axes group on a path, it might be necessary to activate some digital event while certain positions are passed. To do so, the event is filled as a bit pattern the element EVENT in MC_PATH_POINT. *Chapter 1.5.9.4.9.2.10.2.2 “MC_PATH_POINT” on page 2704*

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

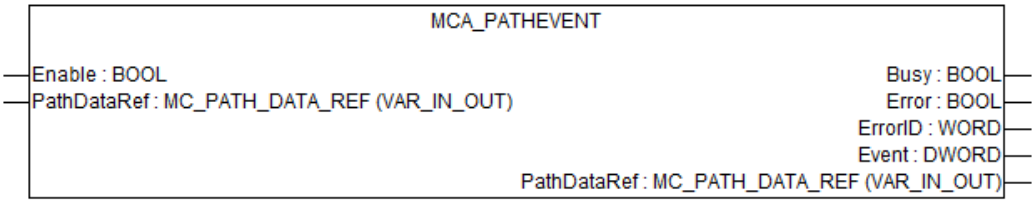


Fig. 562: Function block MCA_PathEvent

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

As long as Enable = TRUE, power is on.

PathDataRef Data type: MC_PATH_DATA_REF

Reference to the path data which can be prepared by MC_PathSelect.

Output description

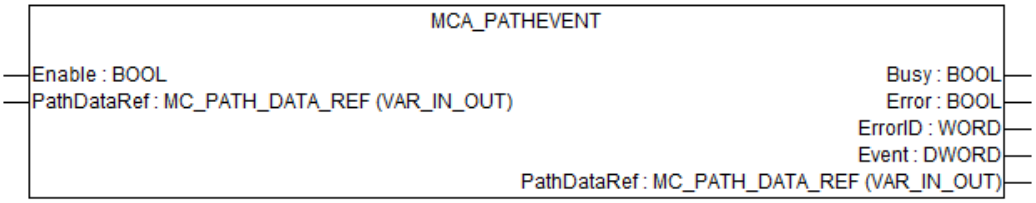


Fig. 563: Function block MCA_PathEvent

Busy Data type: BOOL

The function block is not finished.

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification ↳ *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

Event Data type: DWORD

Bit pattern which is specified at the actually crossed path section.

MCA_SetCoordinateTransformation

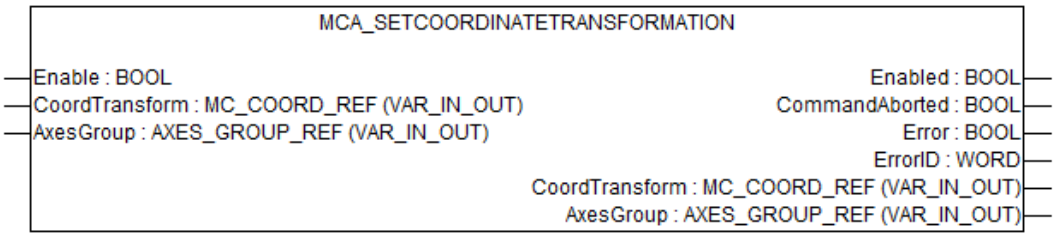


Fig. 564: Function block MCA_SetCoordinateTransformation

This function block is used to fulfill the functionality of MC_SetCoordinateTransformation by a different implementation. It will activate a PCS. The PCS system is active while Enabled=TRUE. All blocks related to the group movement or to a movement of an axis linked to the cartesian coordinate system (MCS axes of group) will then be executed using the PCS.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

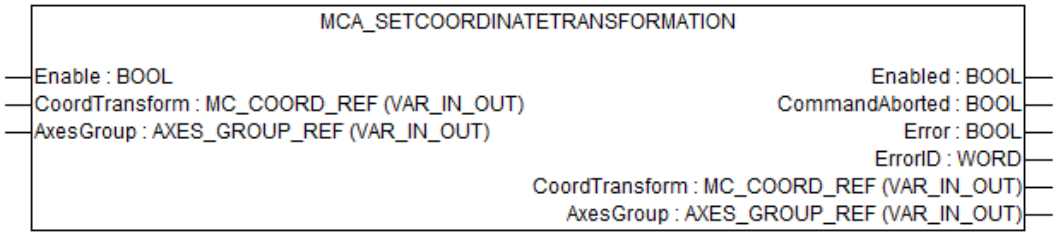


Fig. 565: Function block MCA_SetCoordinateTransformation

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

Enable Data type: BOOL

Activate the PCS.

CoordTransform Data type: MC_COORD_REF

Reference to a Coordinate Transformation. ABB specific datatype.

CoordTransform Data type: MC_COORD_REF

Holds a matrix which is used to perform a homogenous coordinate transform from PCS to MCS and a matrix to transform MCS to PCS.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

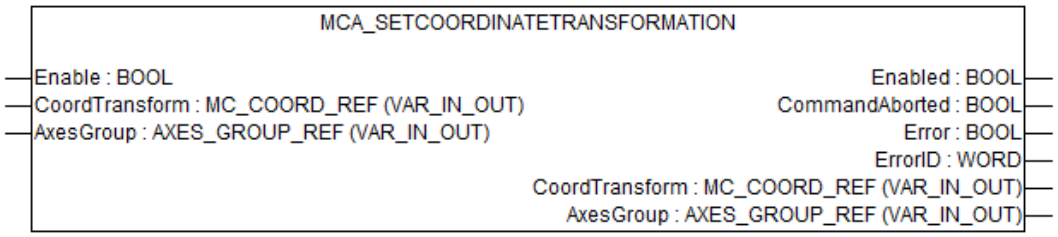


Fig. 566: Function block MCA_SetCoordinateTransformation

Enabled Data type: BOOL

The PCS is used.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_SetDynamicFollower

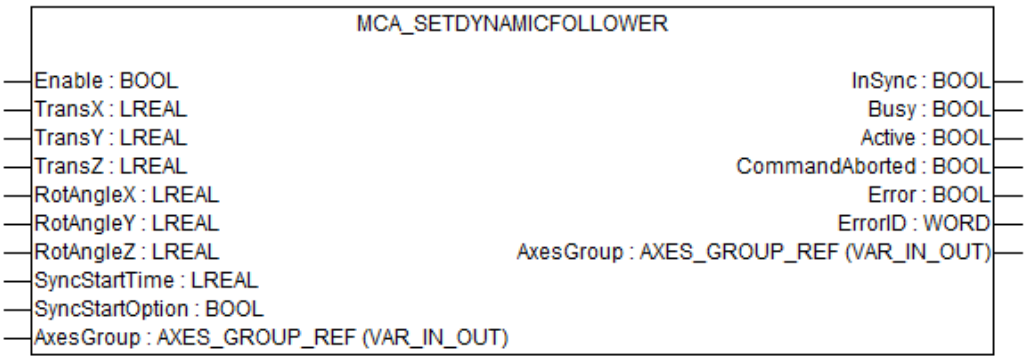


Fig. 567: Function block MCA_SetDynamicFollower

This function block activates a dynamic coordinate transformation and allows to follow a moving product. It replaces the functionality of MC_TrackConveyorBelt and MC_TrackRotaryTable.

▷ Activate a PCS first by MC_SetCoordianteTransform or MCA_SetCoordinateTransformation.

- The group is not allowed to be in any moving state when the block is activated.
- As soon as “Busy” is shown, it is allowed to start a movement.

When a synchronization to a moving coordinate system is performed, the function block allows to synchronize the group to this movement with a ramp. The two inputs SyncStartTime and SyncStartOption are available to configure the synchronization process.

A synchronization is performed with SyncStartTime > 0. The group will need SyncStartTime ms to do this. A polynomial movement is used and the result is superimposed to any other movement. The PCS position will stay constant during this process, just the ACS is moving. The process is ready when the function block shows INSYNC. A group movement can be started as soon as Busy = TRUE.

With Enable = FALSE, the function block is deactivated. This is always done without any ramp. When a ramp is needed, this has to be achieved by activating an other instance of this function block.

SyncStartTime	SyncStartOption	Behavior
0	-	The group will follow the PCS coordinate system without synchronization process.
> 0	FALSE	After the synchronization, the ACS and MCS positions will have the same values as if SyncStartTime = 0 would have been used.
> 0	TRUE	After the synchronization, the ACS and MCS positions will be shifted by the position distance which was needed to synchronize.

The function block performs first a rotation with the X/Y/Z-axis as axis of rotation and then a translation in X/Y/Z-direction. The result is a modified PCS coordinate system.

The transformation moves the coordinate system in parallel to the X/Y plane when no rotation using the X- or Y-axis is performed.

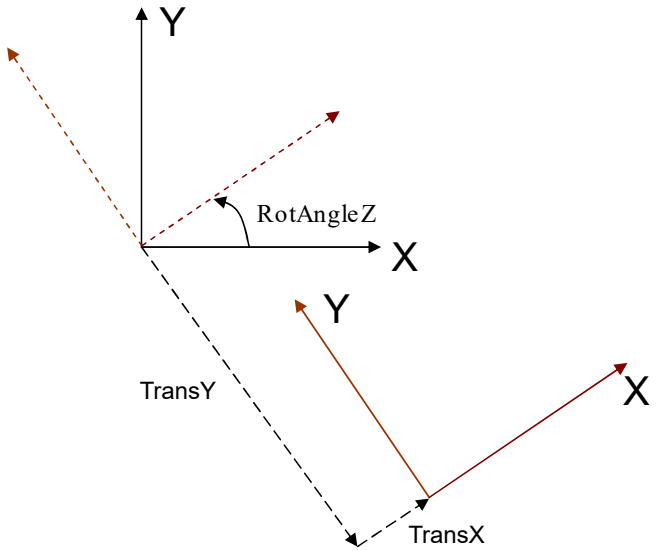


Fig. 568: MCA_SetDynamicFollower Rotation angle

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

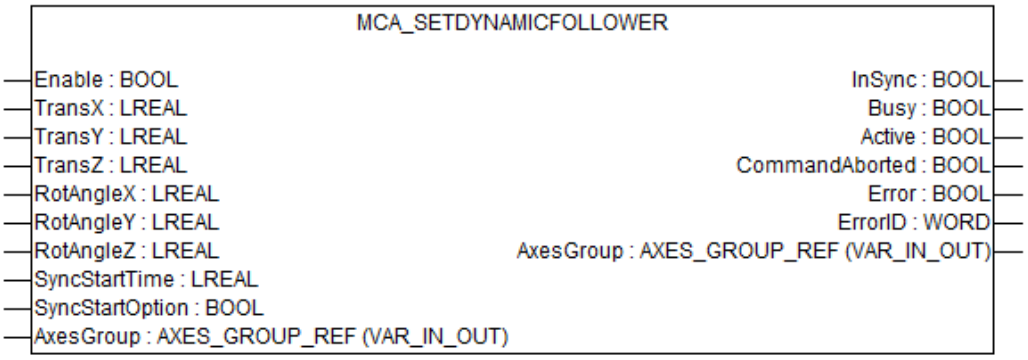


Fig. 569: Function block MCA_SetDynamicFollower

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

This block has to be called from the same task as CMC_MOTION_KERNEL_REAL.

Enable Data type: BOOL
Activate/deactivate the dynamic coordinate transformation.

TransX Data type: LREAL
Distance in X-direction.

TransY Data type: LREAL
Distance in Y-direction.

TransZ Data type: LREAL
Distance in Z-direction.

RotAngleX Data type: LREAL, unit: rad
Rotation angle in X-direction.

RotAngleY Data type: LREAL, unit: rad
Rotation angle in Y-direction.

RotAngleZ Data type: LREAL, unit: rad
Rotation angle in Z-direction.

SyncStartTime Data type: LREAL, unit: ms

A time used to synchronize to a moving transformation.

SyncStartOption Data type: BOOL

This option will result in a different synchronization profile. FALSE: Synchronize to the position where the function block was enabled, move opposite direction if necessary, this is a "CATCH UP" mode. TRUE: Allow a distance to synchronize. A later movement will nevertheless reach the same positions, just the synchronization process differs.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

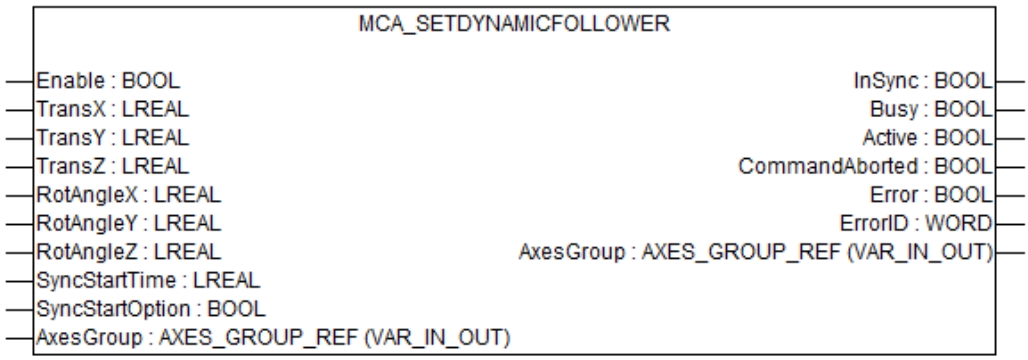


Fig. 570: Function block MCA_SetDynamicFollower

InSync Data type: BOOL

Dynamic coordinate transformation is active and synchronization ready.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandA-borted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_MovePathPos

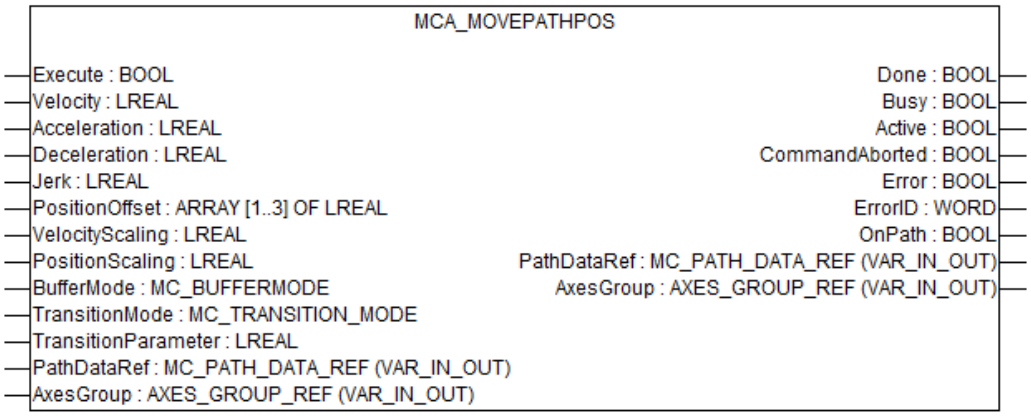


Fig. 571: Function block MCA_MovePathPos

This function block commands an AxesGroup to move according to the path specified in the PathData. The functionality is identical to MC_MovePath with the additional feature to move the group to the path start position first, using the given velocity, acceleration and deceleration.

This function block applies to the MCS or PCS System, depending which is activated and also follows the dynamic transformation when activated.

This function block is only supported for PLC-based central Motion Control with Coordinated Motion structures.

Input description

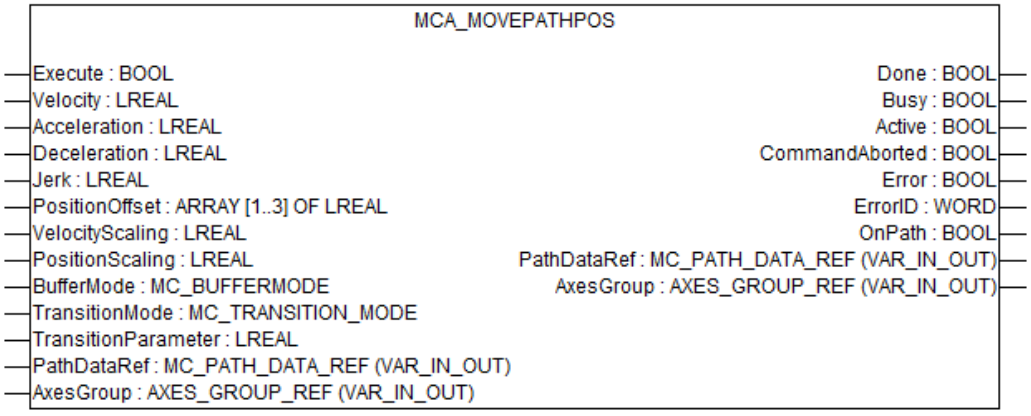


Fig. 572: Function block MCA_MovePathPos

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

Execute
Data type: BOOL
Starts the function block at rising edge.

Velocity
Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

Acceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

Deceleration
Data type: LREAL, range: > 0, unit: u/s^2
Value of the deceleration (decreasing energy of the motor).

Jerk
Data type: LREAL, range: > 0, unit: u/s^3
Value of the Jerk.

PositionOffset
Data type: ARRAY [1..3] OF LREAL
Allows to define an offset to the position in PathData. The movement starts at actual position and positionOffset. This position should be the same as the first Path Position.

VelocityScaling
Data type: LREAL, default: 1
Allows to increase or decrease the path velocity.

PositionScaling
Data type: LREAL, default: 1
Allows to scale the position values defined in PathData.

BufferMode
Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

TransitionMode
Data type: MC_TRANSITION_MODE
The realization just supports by default a transition starting with the actual velocity.

TransitionParameter
Data type: LREAL
Additional parameter for the transition mode.

PathDataRef
Data type: MC_PATH_DATA_REF
Reference to the path data which can be prepared by MC_PathSelect.

AxesGroup Data type: AXES_GROUP_REF

Reference to a group of axes.

Output description

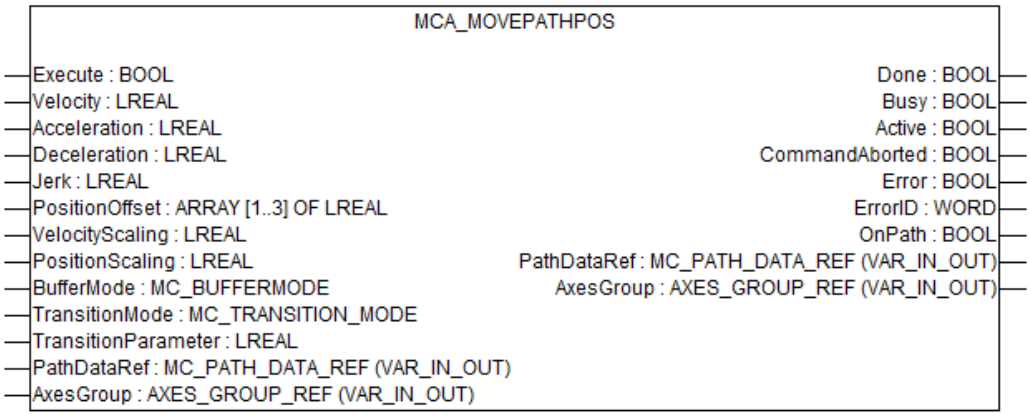


Fig. 573: Function block MCA_MovePathPos

Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

Error Data type: BOOL

Signals that an error has occurred within the function block.

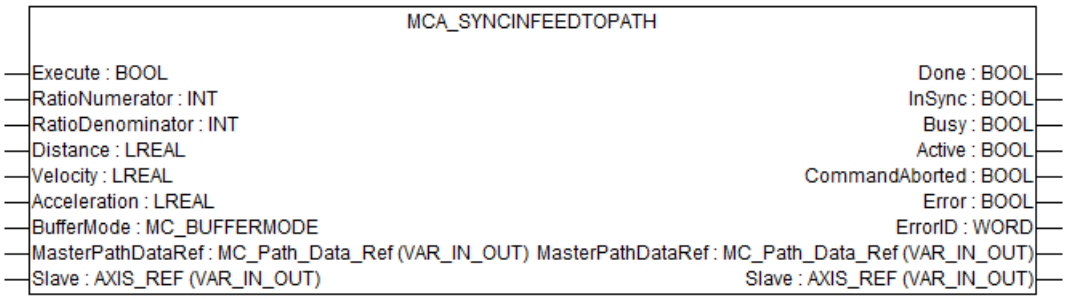
ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

OnPath Data type: BOOL

This output = TRUE when the function block starts to interpolate on the given path, so the prior positioning to the first path position is completed.

MCA_SyncInfeedToPath

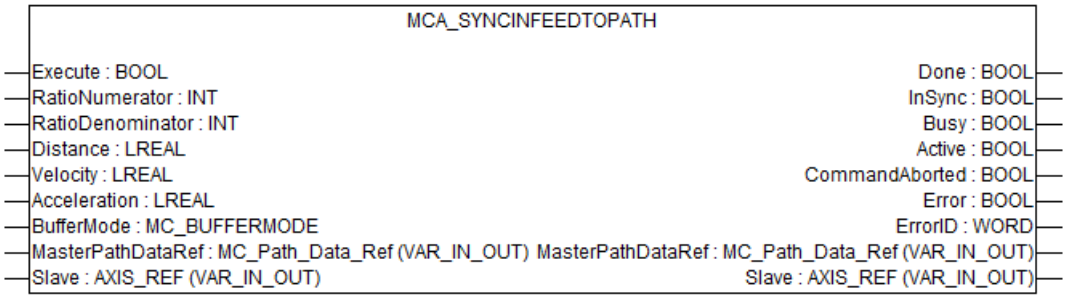


This function block maps a single axis to a group. The axis will follow a group in a path-movement.

It will either use the value given as Distance and move this distance while the groups moves the given path. When RatioNumerator and RatioDenumerator are both different from 0, the groups moving distance will be used instead and the axis be couple like GearIn to the group movement. The axis will then move the groups total distance, modified by the given ratio. The slave axis will always follow the master-group in a MCA_LINEAR mode, as a continuous master movement is expected. The slave movement is limited by the given velocity and acceleration.

> *This function block has to be called within the real-time task.*

Input description



Execute
Data type: BOOL
Starts the function block at rising edge.

RatioNumerator
Data type: INT
Gear Ratio Numerator.

RatioDenominator
Data type: INT
Gear Ratio Denominator.

Distance Data type: LREAL, unit: u
Relative distance for the motion.

Velocity Data type: LREAL, range: > 0, unit: u/s
Value of the maximum velocity (not necessarily reached).

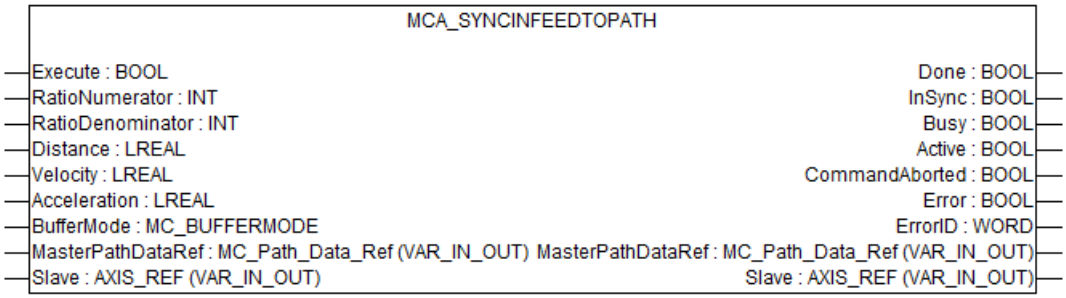
Acceleration Data type: LREAL, range: > 0, unit: u/s^2
Value of the acceleration (increasing energy of the motor).

BufferMode Data type: MC_BUFFERMODE, default: MC_Aborting, no other modes supported
Defines the behavior of the axis.

MasterPathDataRef Data type: MC_Path_Data_Ref
Reference to the path data which can be prepared by MC_PathSelect.

Slave Data type: AXIS_REF
Reference to the slave axis.

Output description



InSync Data type: BOOL
Dynamic coordinate transformation is active and synchronization ready.

Busy Data type: BOOL
The function block is not finished.

Active Data type: BOOL
Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL
Command is aborted by another command (PLCopen function block).

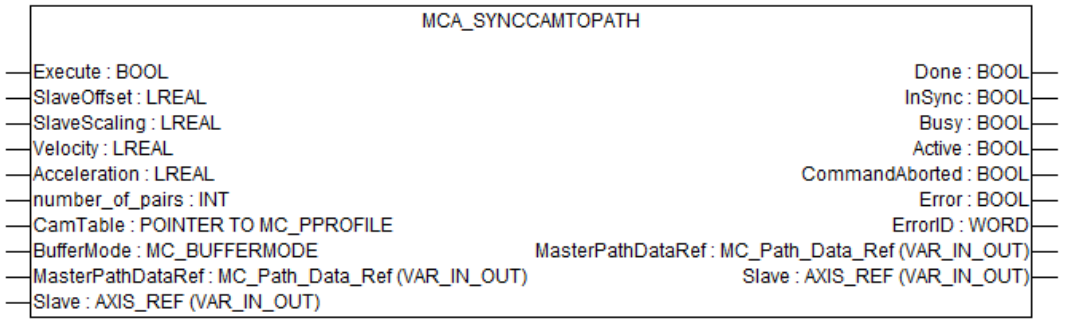
Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

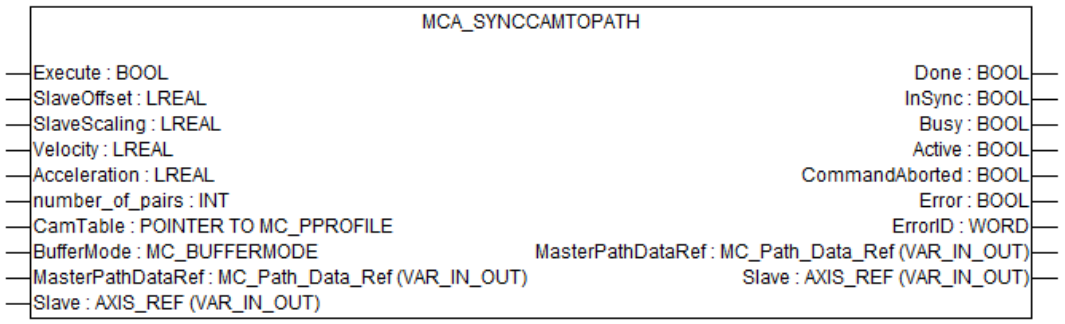
MCA_SyncCamToPath



This function block maps a single axis to a group. The axis is coupled with an own path to the group path. It will follow the group movement from point-to-point with respect to it´s own paths definition. The slave axis will always follow the master in a MCA_LINEAR mode, as a continuous movement is expected. The slave movement is limited by the given velocity and acceleration.

This function block has to be called within the real-time task.

Input description



Execute Data type: BOOL

Starts the function block at rising edge.

SlaveOffset Data type: LREAL, default: 0

Offset of slave table. Sharpened cam (i.e higher elevation and deeper depression).

SlaveScaling Data type: LREAL, default: 1.0

Factor for the slave profile. The overall slave profile is multiplied by this factor.

Velocity Data type: LREAL, range: > 0, unit: u/s

Value of the maximum velocity (not necessarily reached).

Acceleration Data type: LREAL, range: > 0, unit: u/s^2

Value of the acceleration (increasing energy of the motor).

Number_of_pairs Data type: INT

Number of points used in CamTableTimeAcceleration array.

CamTable Data type: POINTER TO MC_PPROFILE

Reference to CAM description.

BufferMode Data type: MC_BUFFERMODE, range: MC_Aborting, MC_Buffered, MC_Blending, default: MC_Aborting

Defines the behavior of the axis.

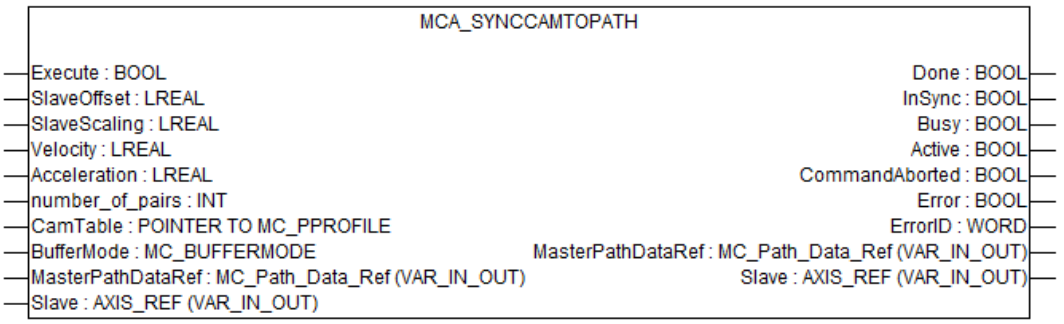
MasterPathDataRef Data type: MC_Path_Data_Ref

Reference to the path data which can be prepared by MC_PathSelect.

Slave Data type: AXIS_REF

Reference to the slave axis.

Output description



InSync Data type: BOOL

Dynamic coordinate transformation is active and synchronization ready.

Busy Data type: BOOL

The function block is not finished.

Active Data type: BOOL

Indicates that the function block has control on the axis.

CommandAborted Data type: BOOL

Command is aborted by another command (PLCopen function block).

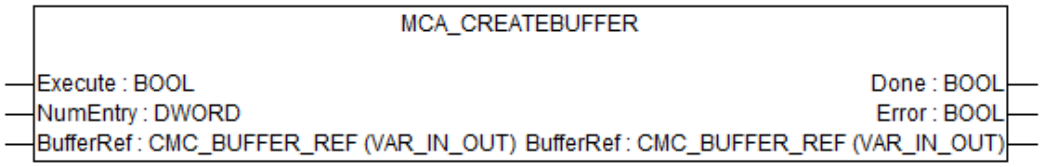
Error Data type: BOOL

Signals that an error has occurred within the function block.

ErrorID Data type: WORD

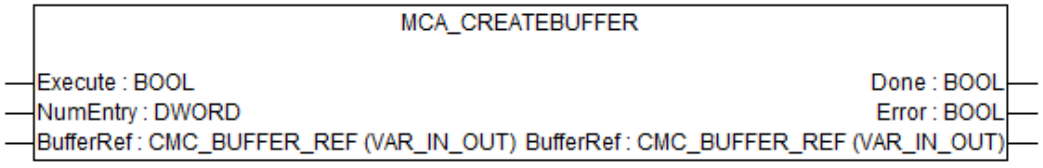
Error identification *Chapter 1.5.9.3.4 “Error codes” on page 2593*.

MCA_CreateBuffer



A movement with move buffers which can be modified on the fly, and which are executed in a FIFO mode, is possible for coordinated motion with the two function blocks: MCA_CreateBuffer and MCA_MoveBuffered *Chapter 1.5.9.7.3.9 “MCA_MoveBuffered” on page 3030*. The function block MCA_CreateBuffer is used to create a buffer with NumEntry placeholders to store the different movements. The function block will fill the variable BufferRef. This variable can be used with function block MCA_MoveBuffered. The buffers will be used as a ring, in a FIFO way, so any number of movements can be executed, just NumEntry to be stored in advance. The number of entries can be changed on a rising edge. This should not be done while the buffer is in use, as the function block will delete the contents and create a new, empty buffer.

Input description



Execute Data type: BOOL

Starts the function block at rising edge.

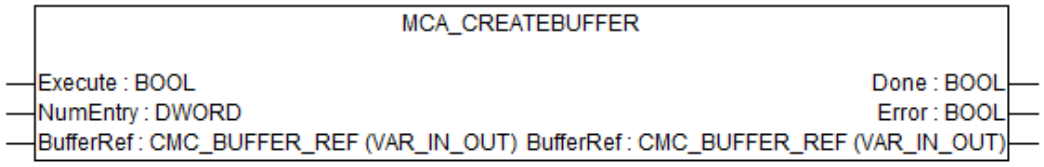
NumEntry Data type: DWORD

BufferRef

BufferRef Data type: CMC_BUFFER_REF

Container for move buffers

Output description



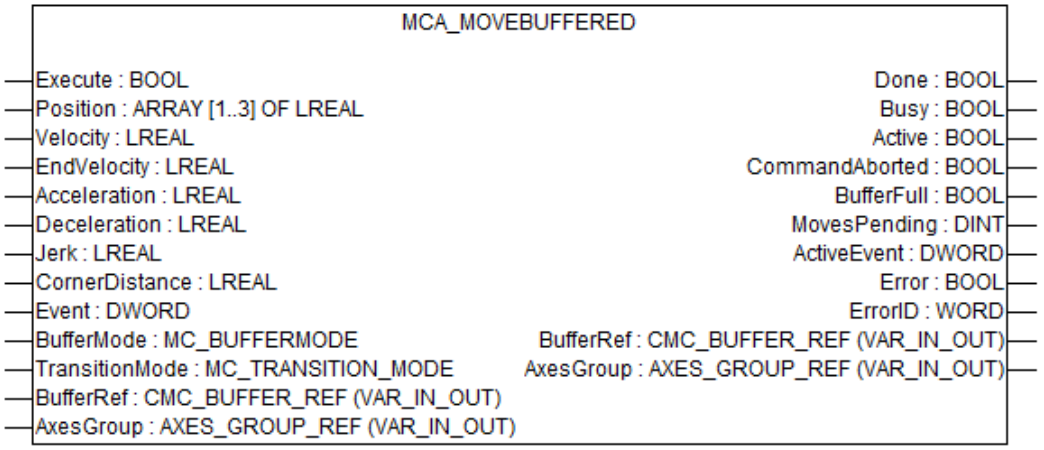
Done Data type: BOOL

Shows the status of the function block. Done = TRUE if the execution is finished.

Error Data type: BOOL

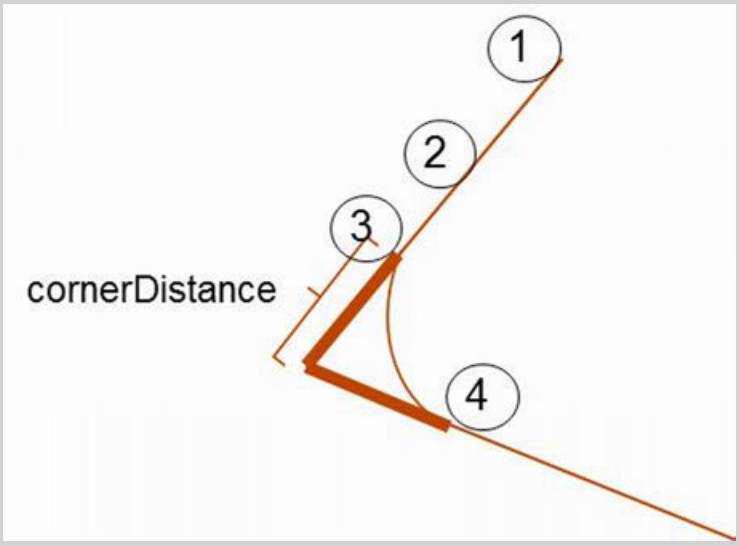
Signals that an error has occurred within the function block.

MCA_MoveBuffered



The function block triggers an absolute, linear movement at a rising edge of input Execute. The first movement will be started directly. Any following rising edge on input Execute will store the given movement to input BufferRef. The output MovesPending will indicate how many entries in BufferRef are occupied and wait for execution. They will be executed automatically and movements will be blended. The behavior of blending can be modified by the inputs EndVelocity, CornerDistance, BufferMode, TransitionMode.

Example



The movement will start at position (1) with velocity = 0. Different behavior will be reached depending on the value for EndVelocity. The velocity which is reached at position (4) depends on the chosen BufferMode. The velocity curve is also modified by the TransitionMode.

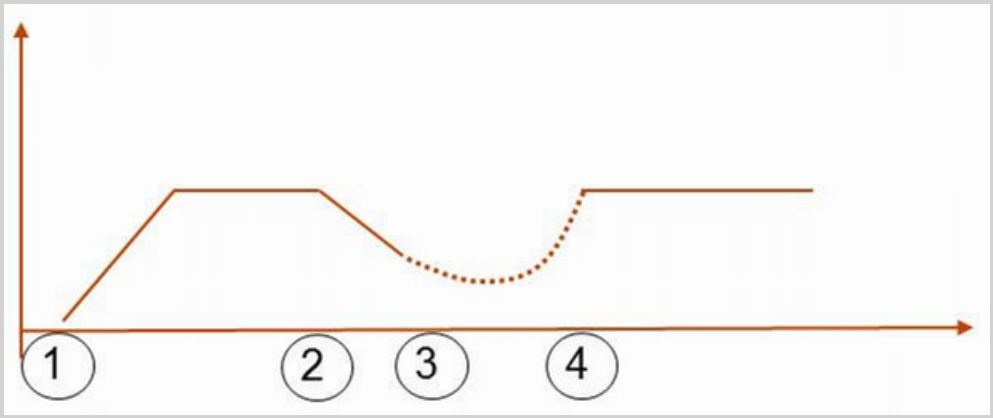
Legend for all examples: s = distance to move, v = velocity, a = deceleration

Behavior of blending modified by EndVelocity

With input EndVelocity = 0, the movement will be planned in a way that it could stop at the corner (position). So the blending might be executed in a lower velocity.

With input EndVelocity > 0, the movement will be executed in a way that at the corner EndVelocity would be reached. An EndVelocity > 0 should not be used for the last intended movement as this will mean that no deceleration ramp is planned.

Example



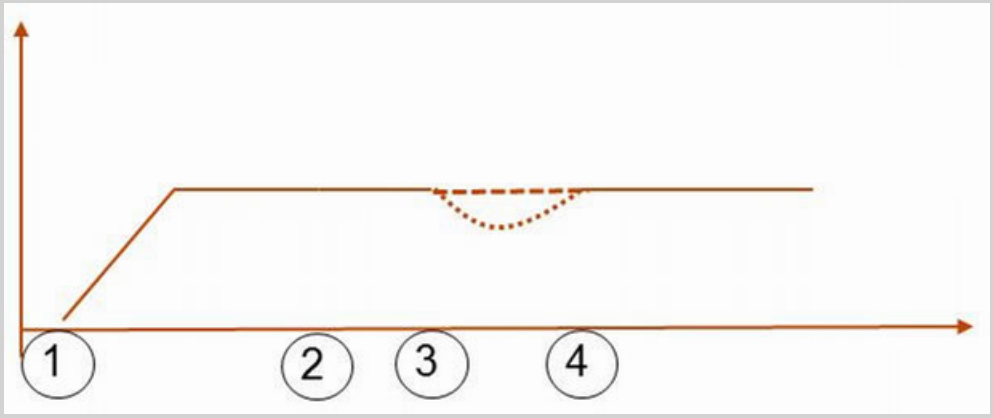
With EndVelocity = 0, depending on the value for Deceleration, it might be necessary to start a brake ramp at a certain position (2). The ramp is calculated in a way that a stop would be reached at the corner:

s = 0.5*v*v/a

s < input CornerDistance: No deceleration ramp.

s > input CornerDistance: Decelerate from a distance s from the corner, a smaller velocity is reached at position 3.

At CornerDistance, position (3), the blending to the next movement is executed.



With EndVelocity = Velocity, the movement will continue at constant speed. When CornerDistance is reached, the blending to the next movement is executed.

Behavior of blending modified by CornerDistance

With CornerDistance D = 0, the movement will stop at the given position and then continue with the next movement. This requires EndVelocity = 0.

With CornerDistance D > 0: If the distance between two positions is < 2*CornerDistance, half position distance is used instead as CornerDistance. At a distance CornerDistance from the target position of a given movement, the direction is changed towards the next position. It is assumed that the velocity for the next movement can be reached within the CornerDistance D.

Behavior of blending modified by BufferMode

The input BufferMode will determine which velocity will be reached at the end of the corner, when the next movement is running. It can be either the actual velocity at position (3) or the commanded velocity for the next movement.

Option	Description
mcBlendingLow	Default. The smaller velocity is used.
mcBlendingHigh	The higher velocity is used.

Option	Description
mcBlendingPrevious	The current velocity at positioin (3) is used.
mcBlendingNext	The commanded velocity for the next movement is used.

Behavior of blending modified by TransitionMode

At input TransitionMode you can choose TMConstantVelocity or other values.

With TransitionMode = TMConstantVelocity, the velocity will be kept constant if for position (3) and (4) an identical value is commanded. In case of two different values, a linear transition is performed. The position movement will be circular.

With TransitionMode = NOT TMConstantVelocity: Depending on the angle between the two movements, the velocity will decelerate. A smaller angle will mean a slower movement at the corner.

Angle	Velocity corner
0°	0
60°	0.5*v
90°	0.67*v
135°	0.87*v
180°	1*v = a straight line

Example 1

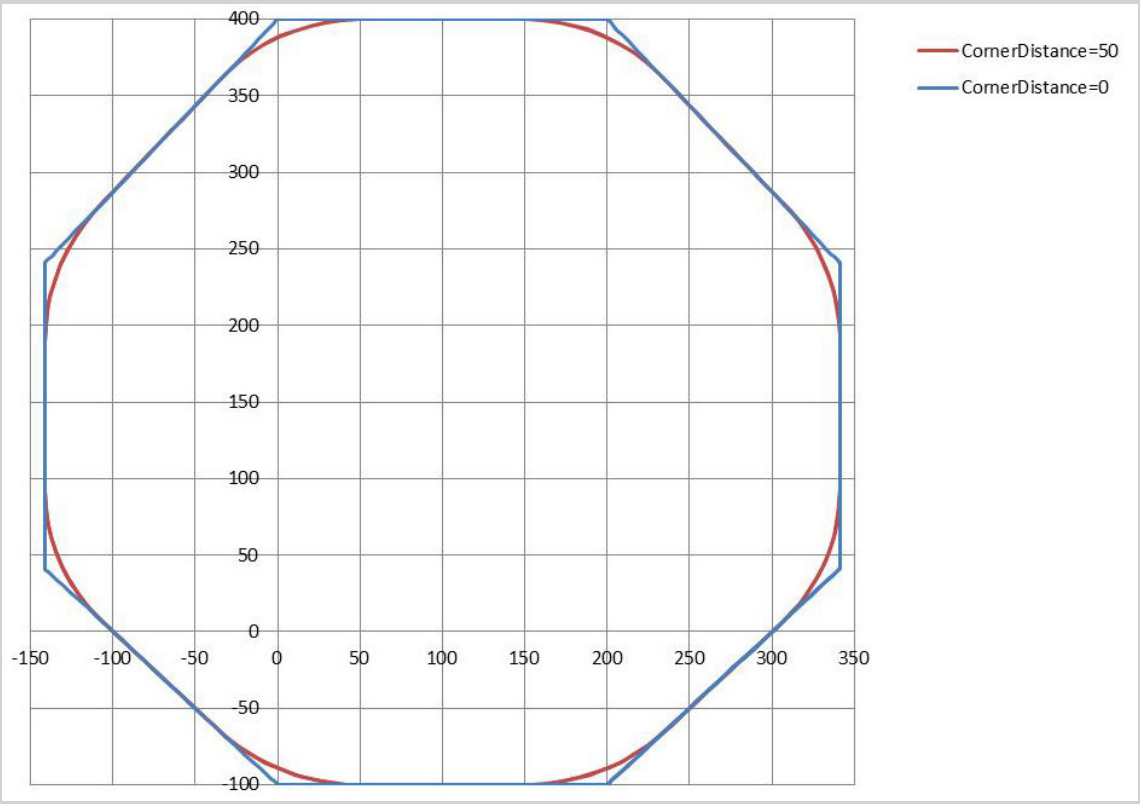


Fig. 574: Movement in X/Y coordinates modified with CornerDistance.

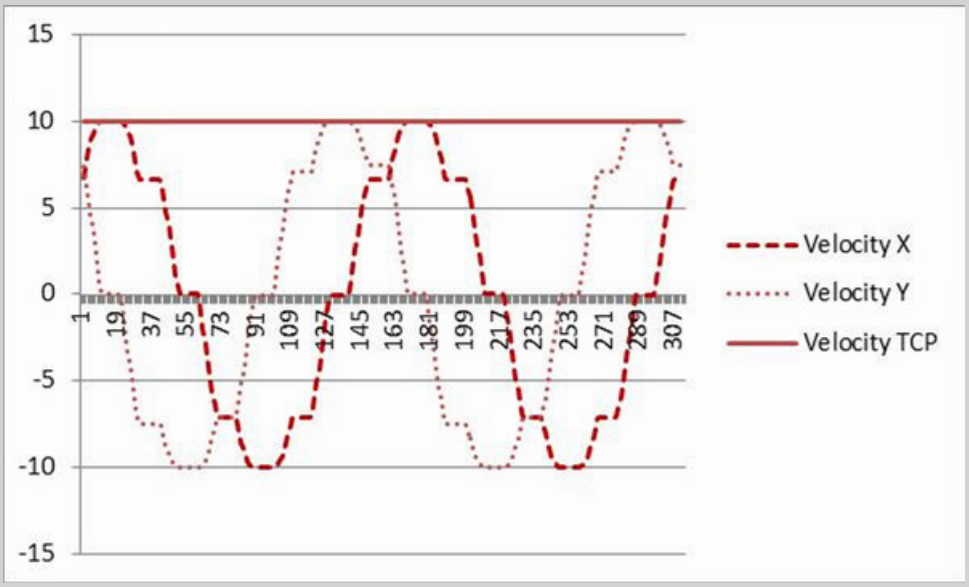


Fig. 575: Resulting velocity curves when CornerDistance = 50, TransitionMode = TMConstantVelocity and EndVelocity = Velocity. In this case, BufferMode does not make a difference as the velocity values are identical.

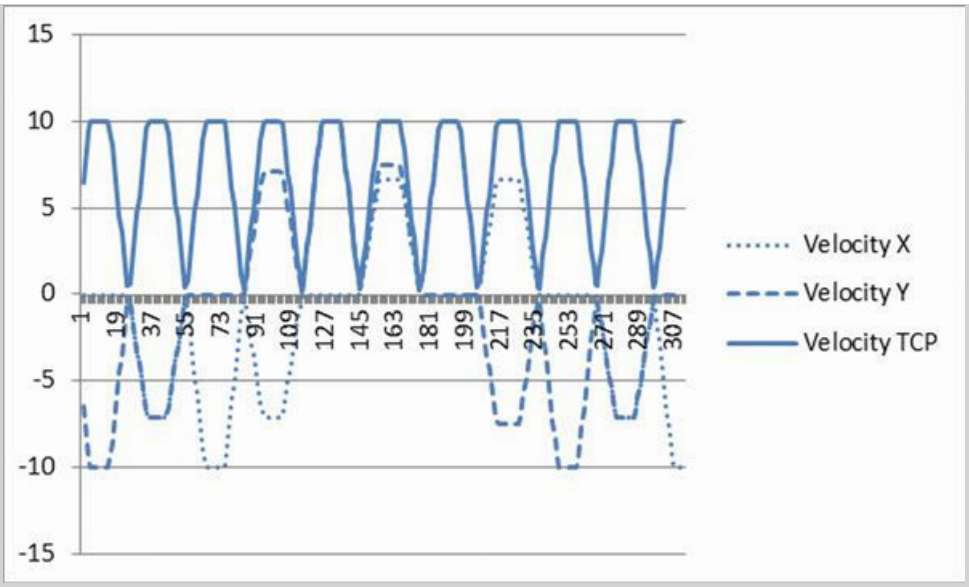


Fig. 576: Resulting velocity curves when CornerDistance = 0 and EndVelocity = 0. In this case, BufferMode and TransitionMode do not make a difference.

Example 2

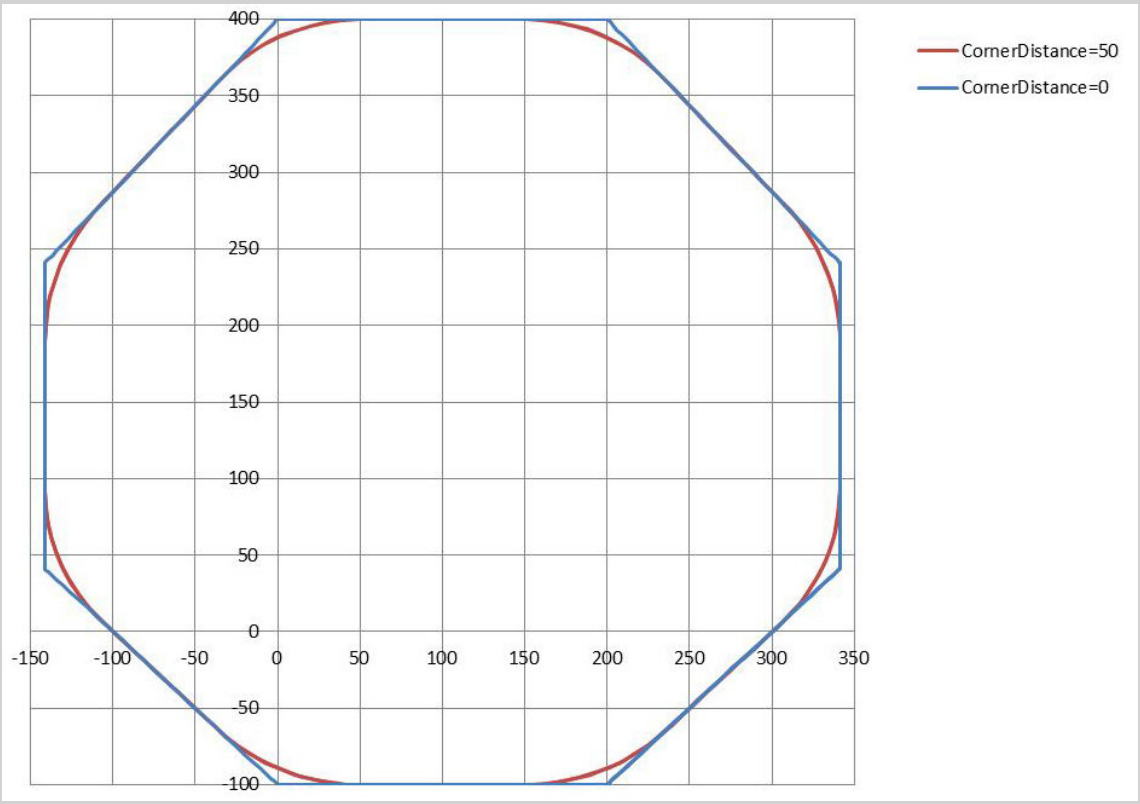


Fig. 577: Movement in X/Y coordinates modified with CornerDistance.

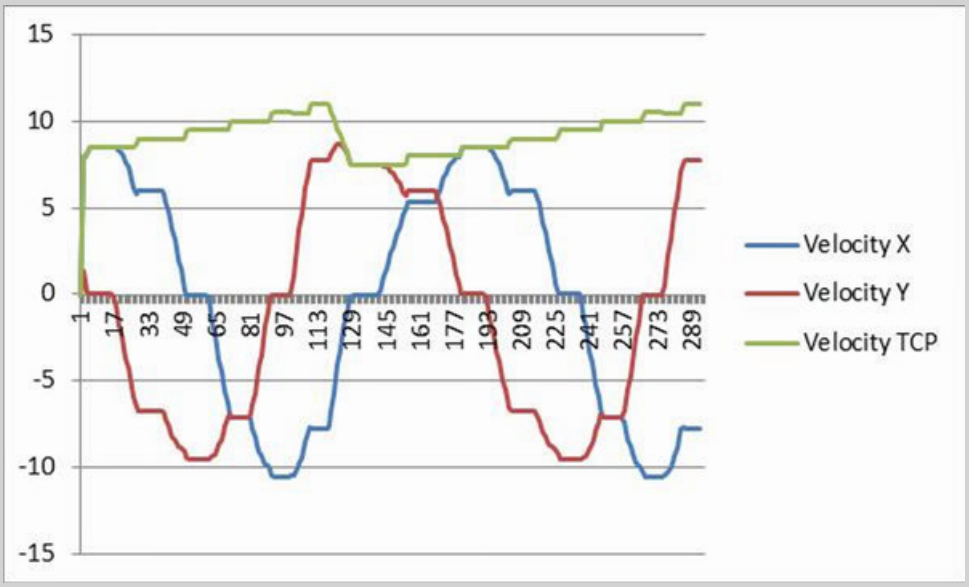


Fig. 578: Resulting velocity curves with different velocity values for each segment. BufferMode = MCBlendingLow and TransitionMode = TMConstantVelocity.

BufferMode	Transition-Mode	EndVelocity	CornerDis-tance	Behavior
x	x	0	0	Stop at the corner, use deceleration.
x	x	> 0	0	Reach EndVelocity at the corner, switch direction without ramp.

BufferMode	Transition-Mode	EndVelocity	CornerDis-tance	Behavior
!	TMConstant-Velocity	=Velocity	> 0	Move at commanded velocity until CornerDistance is reached, decide next velocity depending on BufferMode, perform any velocity transition during the corner (if different velocity are commanded).
!	NOT TMCon-stantVelocity	= Velocity	> 0	As above, but movement during the corner will be slower.
!	!	> 0	> 0	Plan the movement to reach EndVelocity at the corner, use the BufferMode and Tran-sitionMode parameter when CornerDistance is.

1.5.9.8 Glossary

ACS Axes Coordinate System: The system of coordinates related to the physical motors and the single movements caused by the single drives.

Blending A way that consecutive function blocks cooperate in the transition from the first to the next.

Contour curve Inserted curve that modifies the original path. It is the resulting curve after blending.

Coordinate system The reference system in which a coordinate or path is described.

Corner deviation The shortest distance between the programmed corner point and the contour curve.

Corner distance Distance of the start point of the contour curve to the programmed target point.

Direction The orientational components of a vector in space. (Note: this is different from the MC_Direction input as used in part 1).

Drive A unit controlling a motor via the current and timing in its coils.

Group-FB The set of function blocks that can work on a group of axes.

MCS Machine Coordinate System: The system of coordinates that is related to the machine. A Cartesian coordinate system with the origin in a fixed position relative to the machine (the origin is defined during the machine setup).

Sometimes called "World Coordinate System" or "Base Coordinate System".

(Note: with Cartesian build machines, MCS is a Cartesian Coordinate system and may be identical to ACS, or mapped via a trivial transformation). The coordinate system from the physical multiple axes ACS is linked to the MCS via a kinematic transformation (forward and backward conversion). The MCS represents an imaginable space with up to 6 dimensions.

Motor	An actuator focused to a movement, converting electrical energy in a force or torque.
Orientation	The rotational components of a vector in space.
Path	Set of continuous positions and orientation information in multi-dimensional space. Geometrical description of a space curve that the TCP of an axesgroup moves along.
PathData	Description of a path which can include additional information like velocity and acceleration.
PCS	The coordinate system of the product can be called PCS: Product Coordinate System (or "Program Coordinate System" in CNC world, or Programmers Coordinate System). The PCS is based on the MCS typically by shifting and maybe rotating the MCS. The Zero point of the PCS is related to the product and can be changed during run time by the program. The real work piece can have a rotation or shift to the MCS (machine coordinate system) or even might be moving relative to the MCS (machine coordinate system). By specifying a trajectory in PCS one is able to describe the trajectory independent from the machine situation. To map these two worlds (MCS to PCS and vice versa), a cartesian or cylindrical transformation is normally done.
Position	Apoint in space which is described by different coordinates. Depending on the used system and transformation it can consist of up to 6 dimensions (coordinates) meaning 3 Cartesian coordinates in space and 3 coordinates for the orientation. In ACS there can be even more than 6 coordinates. If the same position is described in different coordinate systems the values of the coordinates are different.
Pose (not used)	Position and orientation (DIN EN ISO 8373). Position is used instead in this document.
Scara	A special kinematic for robot or handling applications.
Speed	The absolute value of the velocity without direction.
Synchronization	Combines an axis or axes group (as slave) with an axis as master in order that the slave executes its path with synchronization to the progress of the master, meaning linked to a one-dimension source for synchronization.
TCP	Tool Centre point, the point in the machine that is commanded to move, typically the center or the head of the tool. It can be described in different coordinate systems.
Tracking	Is characterized by an axis group that follows with its movement the movement of another axis group.
Trajectory	Time dependent description of the path the TCP of an axes group moves along. Additionally to the geometrical description of the space curve, time dependent state variables like velocity, acceleration, jerk, forces etc. are specified.
Velocity	In ACS: For a group of axes this means the velocities of the different axes.

Iin MCS and PCS: It provides the velocity of the TCP.

1.5.9.9 Examples

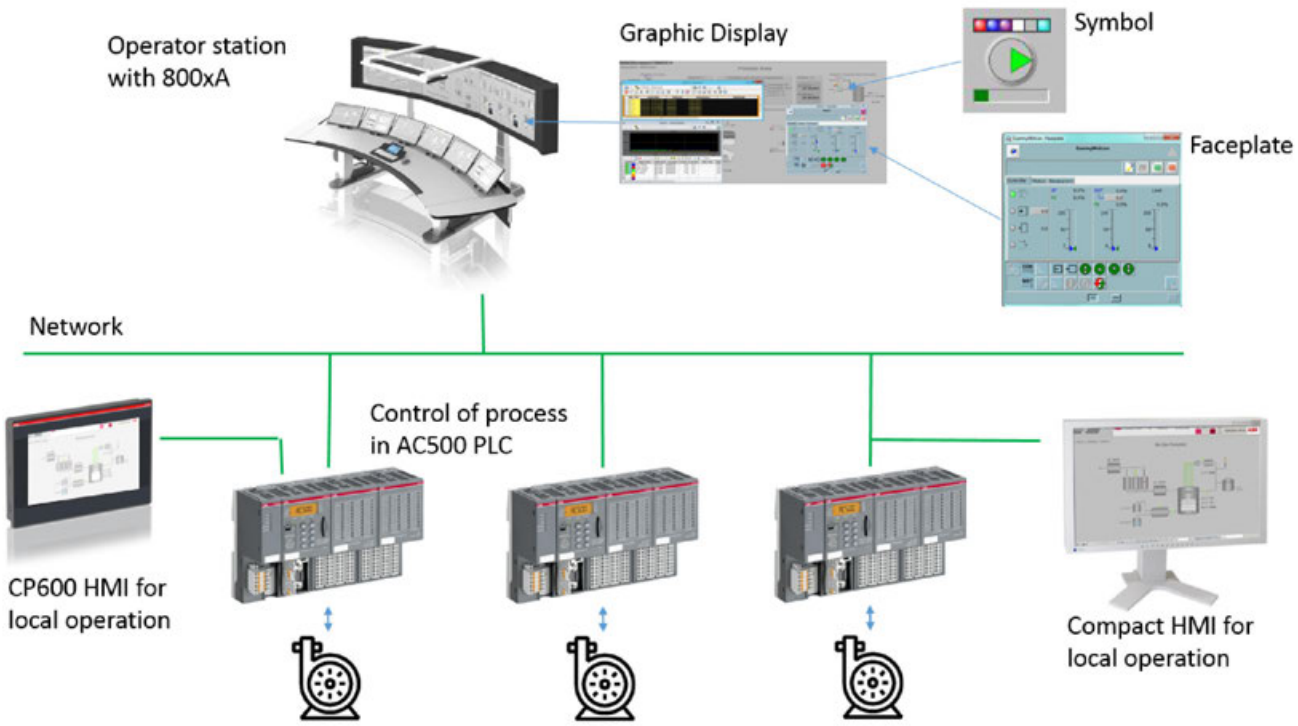
Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.10 Process control object (PCO) library

1.5.10.1 PCO library - System technology

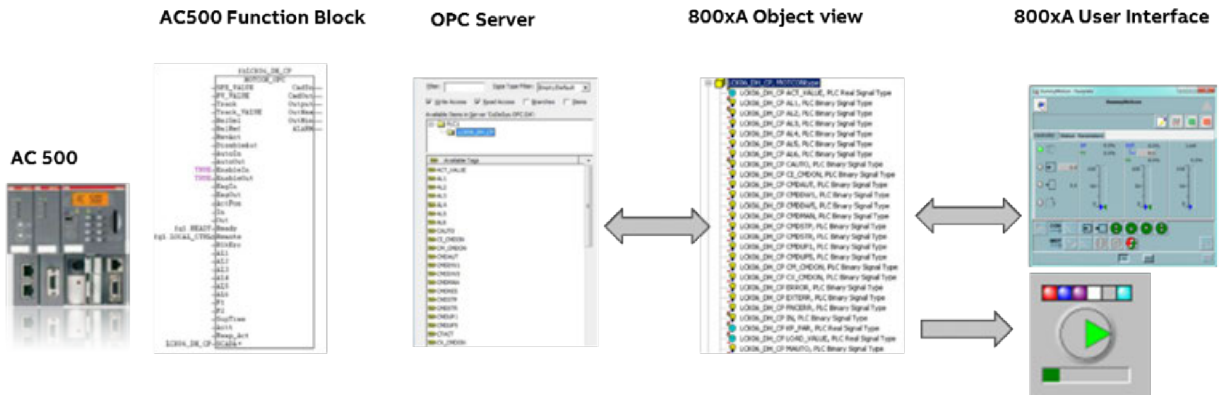
1.5.10.1.1 Introduction

800xA Connect in combination with AC500 Process Control Object (PCO) library allows to easily integrate AC500 into 800xA for process control:



The AC500 PCO Library contains function blocks for Process Control Objects like motors, valves, PID controllers, alarms, etc. Those function blocks are integrated into 800xA via OPC.

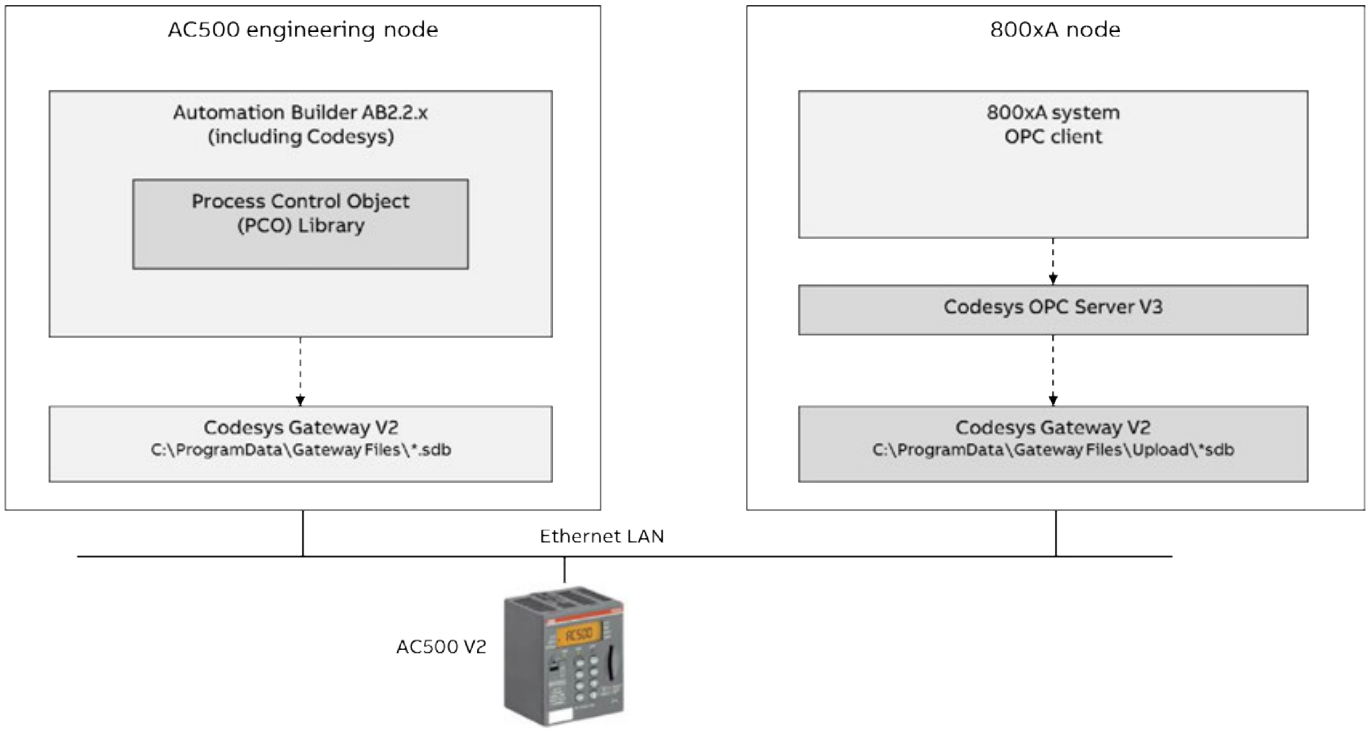
A corresponding 800xA aspect object allows control and monitoring of the Process Control Objects with 800xA User Interface like faceplates and graphic elements:


AC500 Function Block
OPC Server
800xA Object view
800xA User Interface
AC 500

1.5.10.1.2 Installation

For integration of AC500 into 800xA it is recommended to have at least two nodes:

- AC500 engineering node with Automation Builder and PCO library
- 800xA engineering node for engineering and operation of 800xA. The OPC server for AC500 (*"Codesys OPC Server V3.5"*) must be installed on this node for connectivity with AC500.



> *It is also possible to install the AC500 Automation Builder on the 800xA engineering node,*
>
> *but this document describes the general case with two nodes.*

Install Automation Builder on AC500 engineering node

Latest Automation Builder installation files and instructions can be found *here*.

During installation the PCO library can be chosen as an option.

Install AC500 PCO Library on AC500 engineering node

If the PCO library is not yet installed from the beginning it can be installed using the Automation Builder Installation Manager:

Install AC500 connect on 800xA node

> *The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.*

Install AC500 OPC server on 800xA node

The AC500 OPC Server (*"Codesys OPC Server 3.5"*) must be installed on the 800xA engineering node. The AC500 OPC server is part of the Automation Builder installation, but it is not required to install the full Automation Builder on the 800xA engineering node.

Latest Automation Builder installation files can be found *here*.

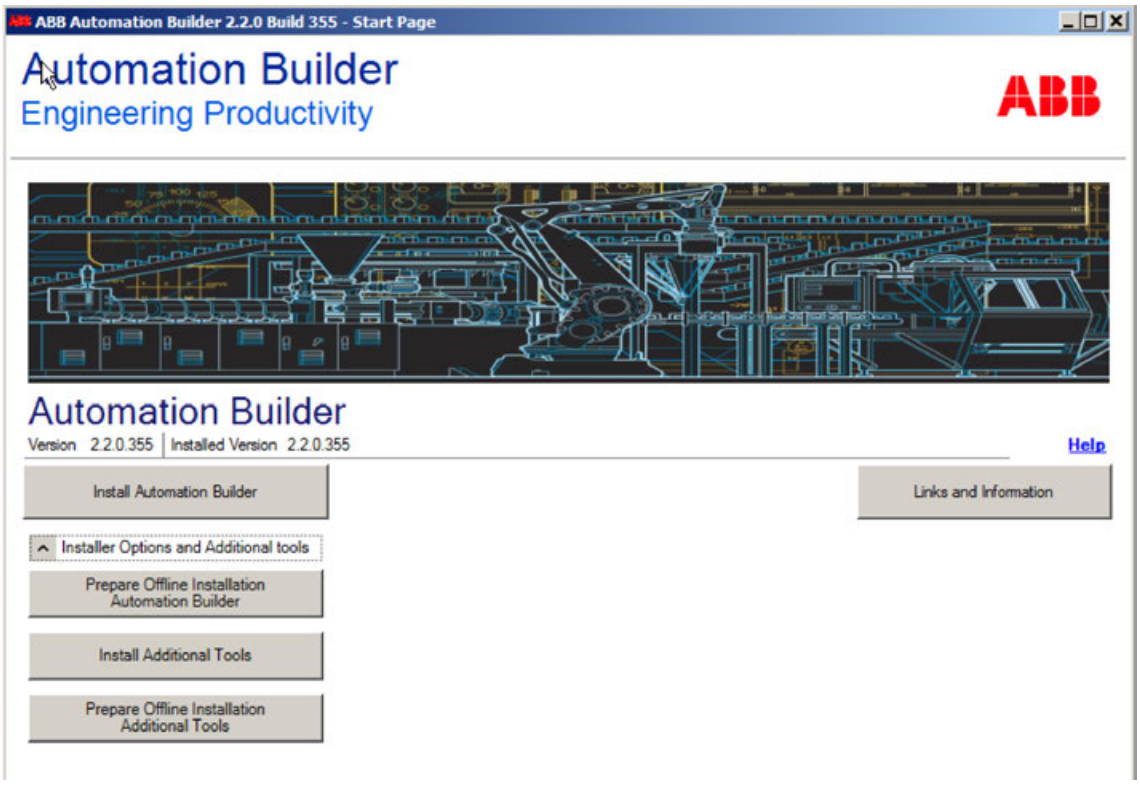
Checkpoint 1

1. Before starting the installation ensure that all OPC clients from previous versions are closed:
   - *"ABB OPC Tunnel"*
   - Gateway (CODESYS gateway server)

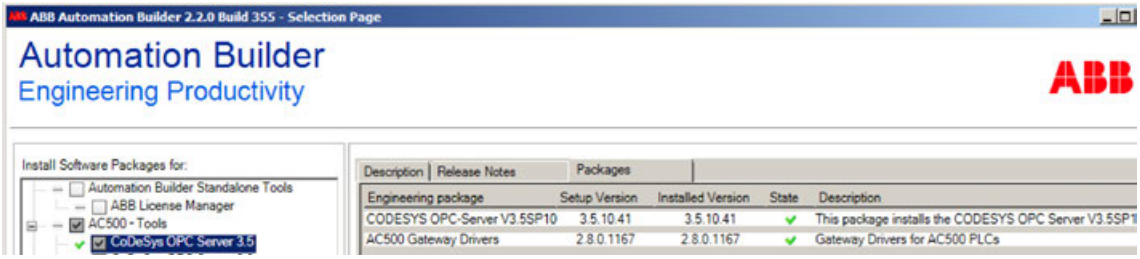
   > *This can be checked in the Windows Task Manager.*

   The following processes must have disappeared:
   - *"Gateway.exe"*,
   - *"CoDeSysOPC.exe"*,
   - *"WinCoDeSysOPC.exe"* and
   - *"OCTsvc.exe"*

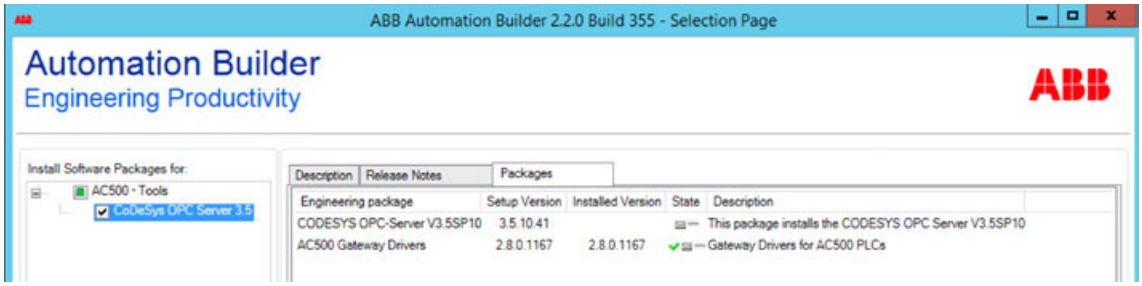
   ⇨ If not:
   - End the processes with the Windows Task Manager.
   - Stop *"ABB OPC Tunnel Windows Component Service"* in Services (local).
2. Start the Automation Builder installation.



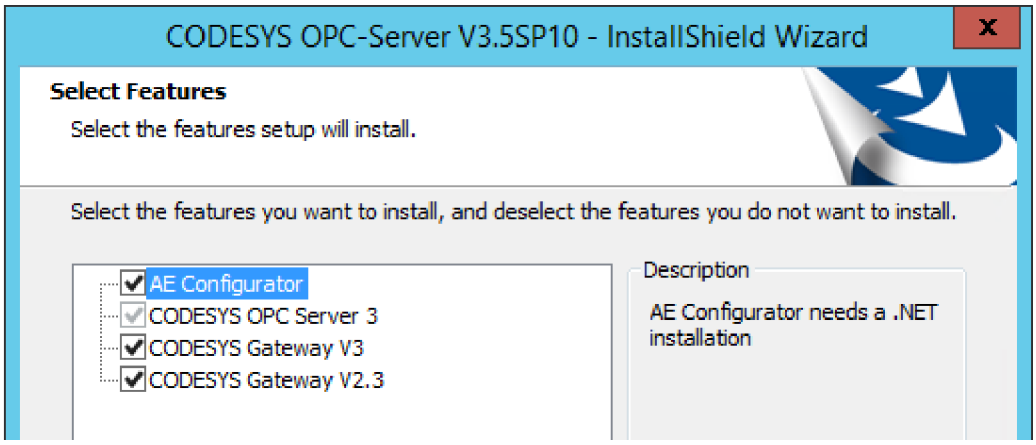
3. Select *[Install Additional Tools]* and install the *"CODESYS OPC Server 3.5"* which includes the *"AC500 Gateway Drivers"*:



4. Alternatively, you can select *[Prepare Offline Installation Additional Tools]* to create an USB-stick containing the *"CODESYS OPC Server 3.5"* and use this for installation on the 800xA engineering node.



5. During installation on the 800xA engineering node select all features.



1.5.10.1.3 Prerequesites

800xA license

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

AC500 license

A separate license is not required for the PCO library.

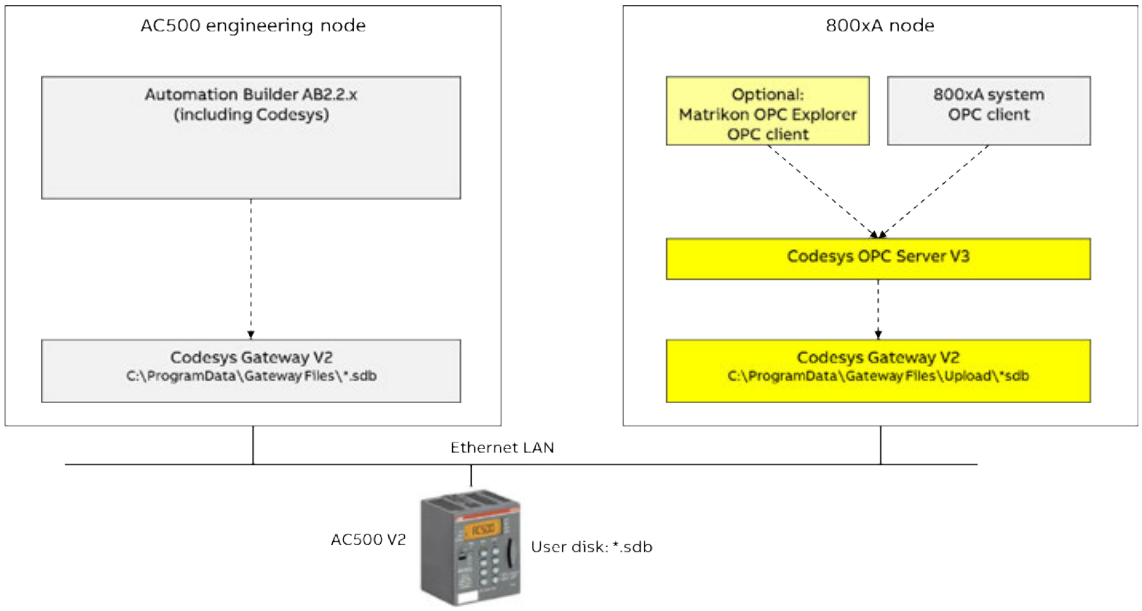
1.5.10.1.4 Configuration

This chapter descibes the configuration of AC500 engineering node and 800xA engineering node.

Configure AC500 OPC server

Communication of AC500 to 800xA is done through OPC. Therefore it is required to install the AC500 OPC server (see *Chapter 1.5.10.1.2.4 “Install AC500 OPC server on 800xA node” on page 3042*) and to configure it accordingly.

Basic configuration workflow for AC500 OPC Server: The definition of the items (symbols) to be exchanged over OPC DA are stored in the symbol file *“*.sdb”*. The symbol file is generated by the AC500 engineering tool Automation Builder which is running on the AC500 engineering node. When downloading the application to the AC500 PLC the *“*.sdb”* file is stored on the user disk of the PLC. Finally, the OPC Server running on the 800xA engineering node uploads the *“*.sdb”* file in order to configure the tags accordingly.



This chapter describes how to setup the OPC connection on the AC500 engineering node as well as on the 800xA engineering node.

AC500 engineering node

It is assumed that a simple AC500 project was configured with an AC500 engineering node running the Automation Builder. For more information please check chapter ↳ *Chapter 1.5.10.1.5.1 “Create function blocks in AC500 V2” on page 3057*.

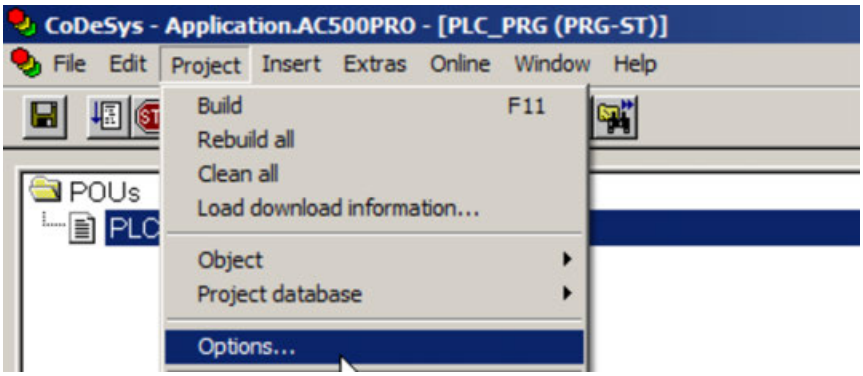
As a quick start the simple motor example “PCO_Motor_Demo_AB223.project” can be used which comes with the PCO library package, see folder

C:\Users\Public\Documents\AutomationBuilder\Examples\PS573-PCO.

The following chapter describes the configuration steps to prepare the AC500 for OPC communication to the 800xA engineering node.

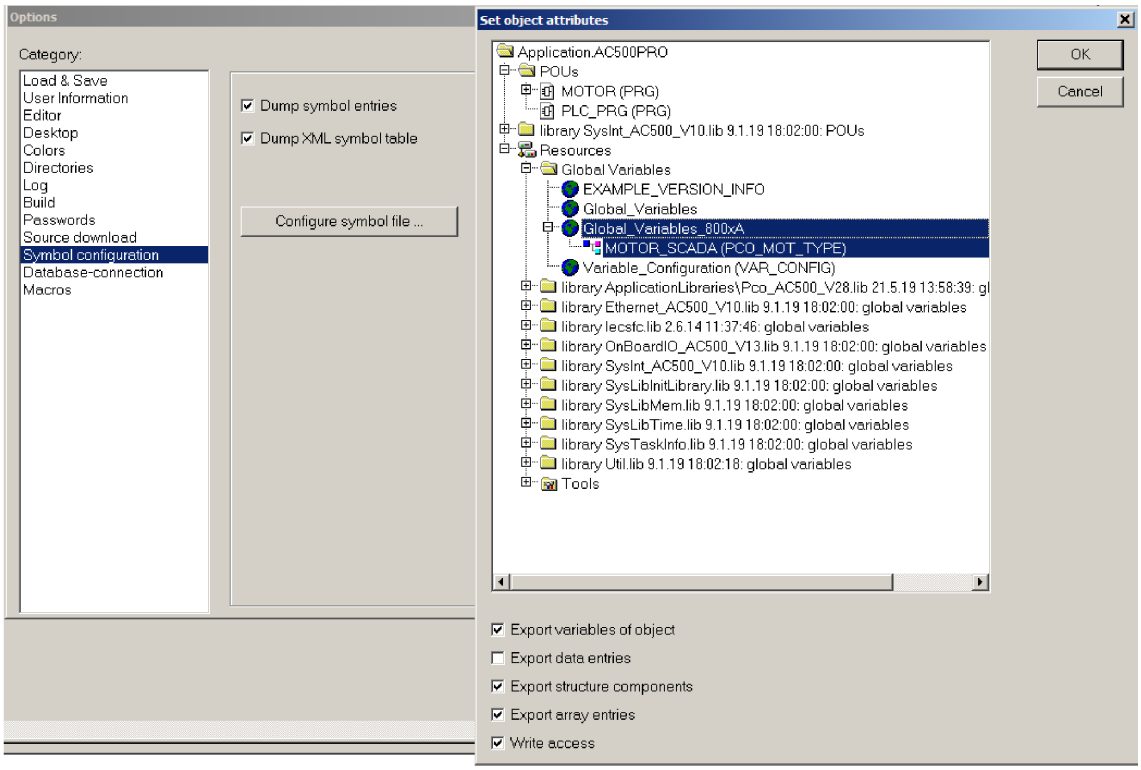
Configure symbol file in Automation Builder

1. Open Automation Builder and start the application of the project which will open CODESYS.
2. Select *“Options”* in menu *“Project”*.



3. Configure symbol file according to the instructions in the Automation Builder help: ↳ *Chapter 1.6.5.5.1.2.5.1.1 “Configure a symbol file” on page 6285*

For the example “PCO_Motor_Demo_AB223.project” it should look like this:

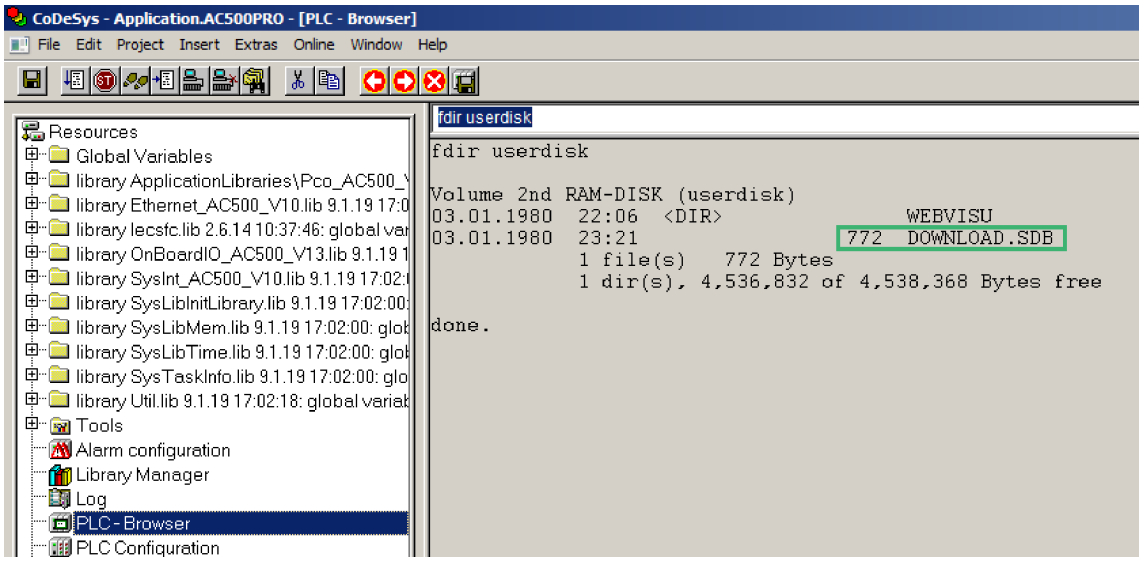


Create and download symbol file

1. Follow the instruction in the Automation Builder help: ↳ *Chapter 1.6.5.5.1.2.5.1.2 “Create and download a symbol file” on page 6287*

Checkpoint 2

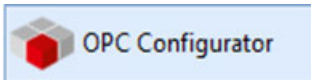
2. Verify that the symbol file is downloaded to the PLC by opening the *“PLC-Browser”* in *“CODESYS”* (online mode) and entering *“fdir userdisk”*:



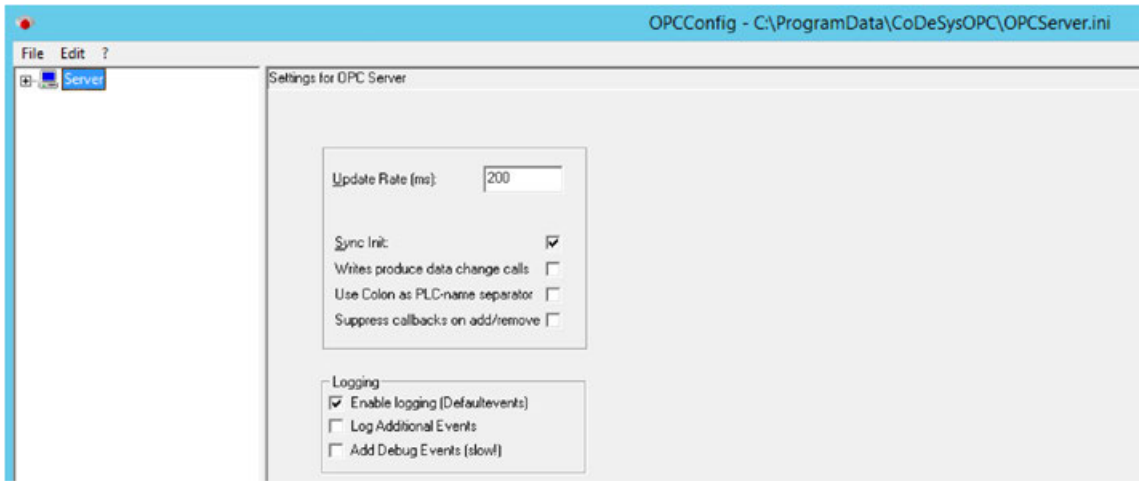
800xA engineering node

Configure OPC server

1. Start *“OPC Configurator”*.



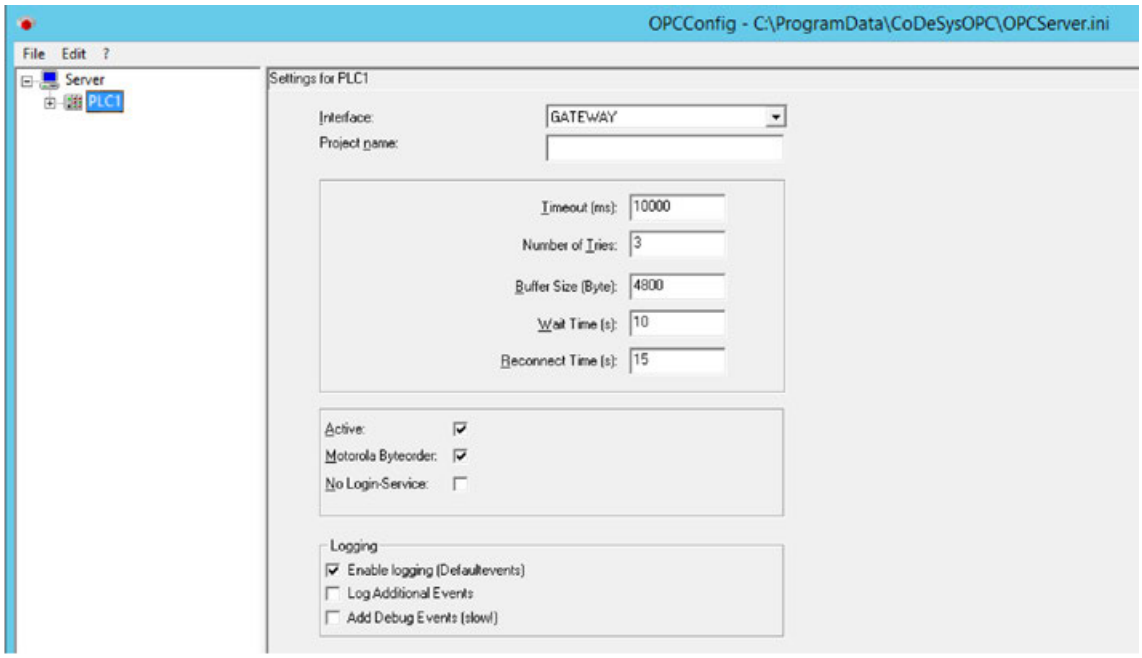
2. Default values of the server need not to be changed.



⇨

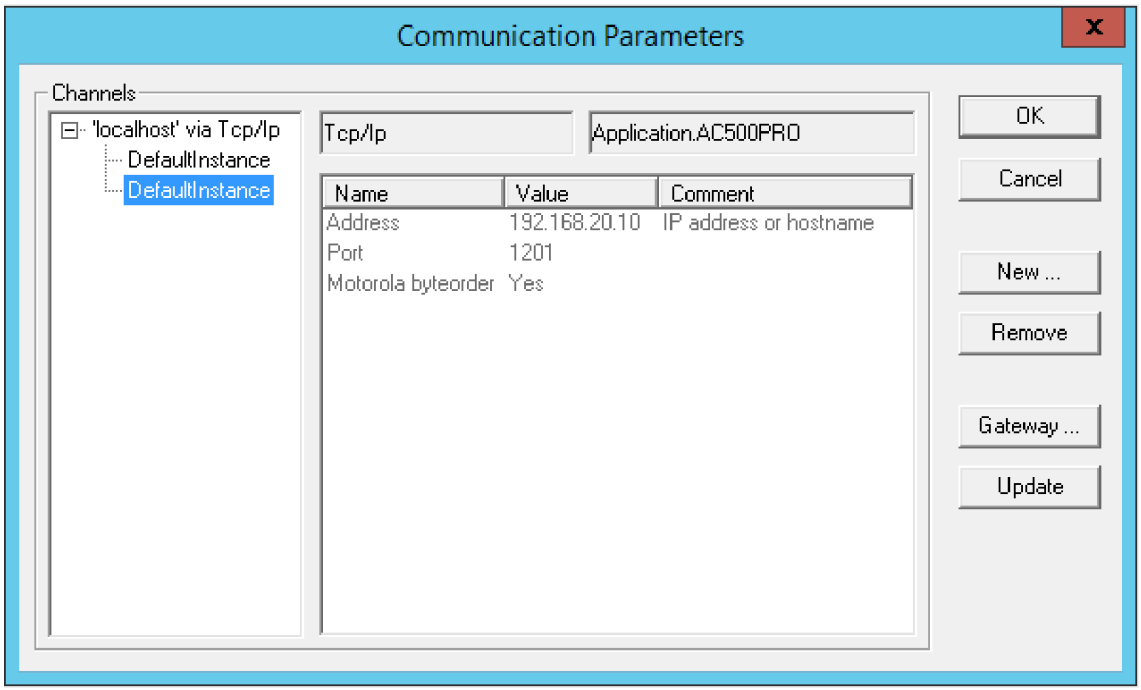
> *“Update Rate” may should not be “0 ms”! The default value of “200 ms” is suitable value of many applications. The adjustment for the update rate depends on the number of symbols (variables). For a big number of symbols it would be better to increase the update rate.*

3. Configure the PLC1.



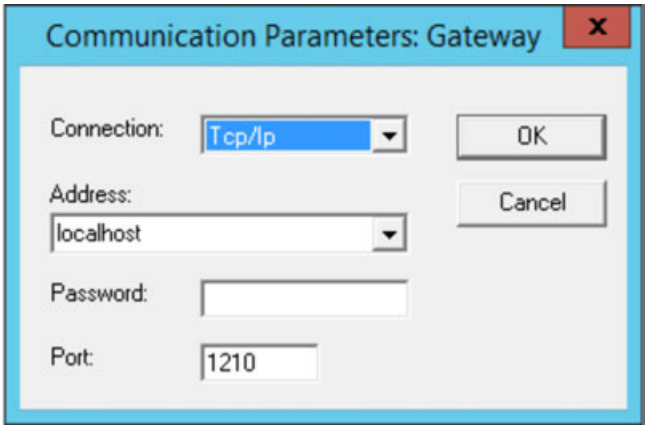
4.
   - Choose *“Interface” “GATEWAY”*.
     This is the V2 Gateway which communicates with the AC500 V2.
   - *“Project name”* can be empty.
   - Increase *“Buffer Size [Byte]”* to *“4800”*
   - The checkboxes *“Active”*, *“Motorola Byteorder”* must be checked.
   - The checkbox *“Enable logging [Defaultevents]”* allows a later diagnosis.

5. Configure connection. Click *[Edit]* and *[New]* to configure a new *"Tcp/Ip"* connection.

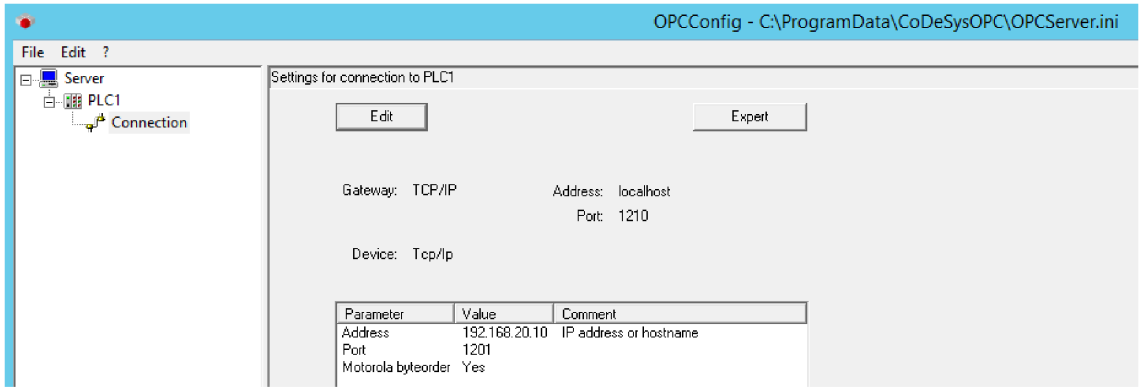


⇨ Address must be the IP address of the connected Ethernet port of AC500.

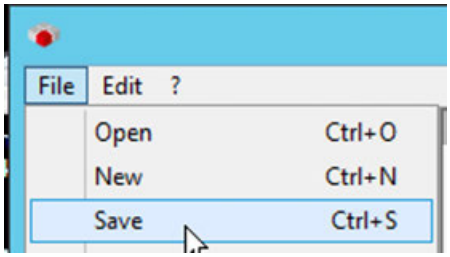
6. Click *[Gateway]* and configure *"Tcp/Ip"*.



7. Final configuration looks like this:



8. Save the configuration.



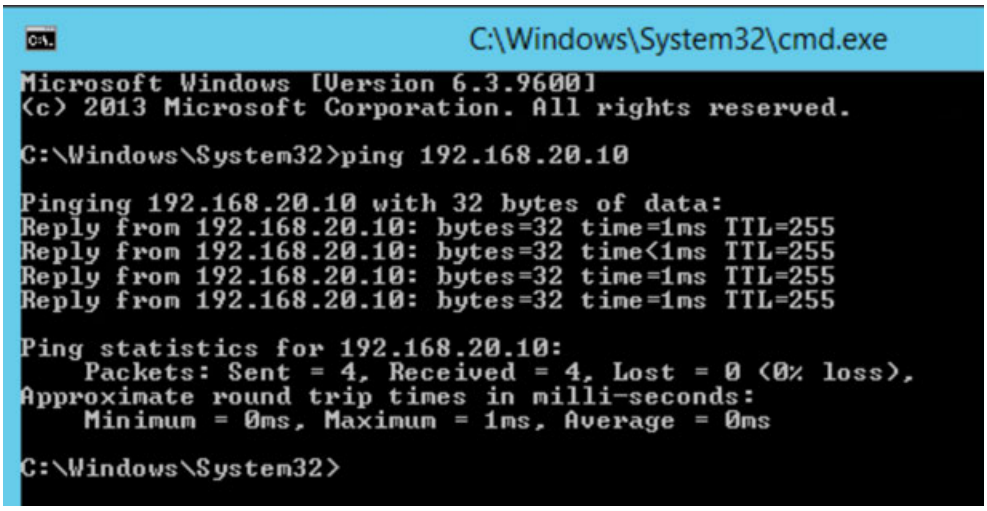
Read OPC data with matrikon test client (optional)

This chapter describes how to install and configure the Matrikon OPC test client. This is an optional step but it is recommended to test the basic OPC communication before doing any 800xA specific configuration.

The AC500 must be connected to the 800xA engineering node and its IP address must be in the same LAN.

Checkpoint 3

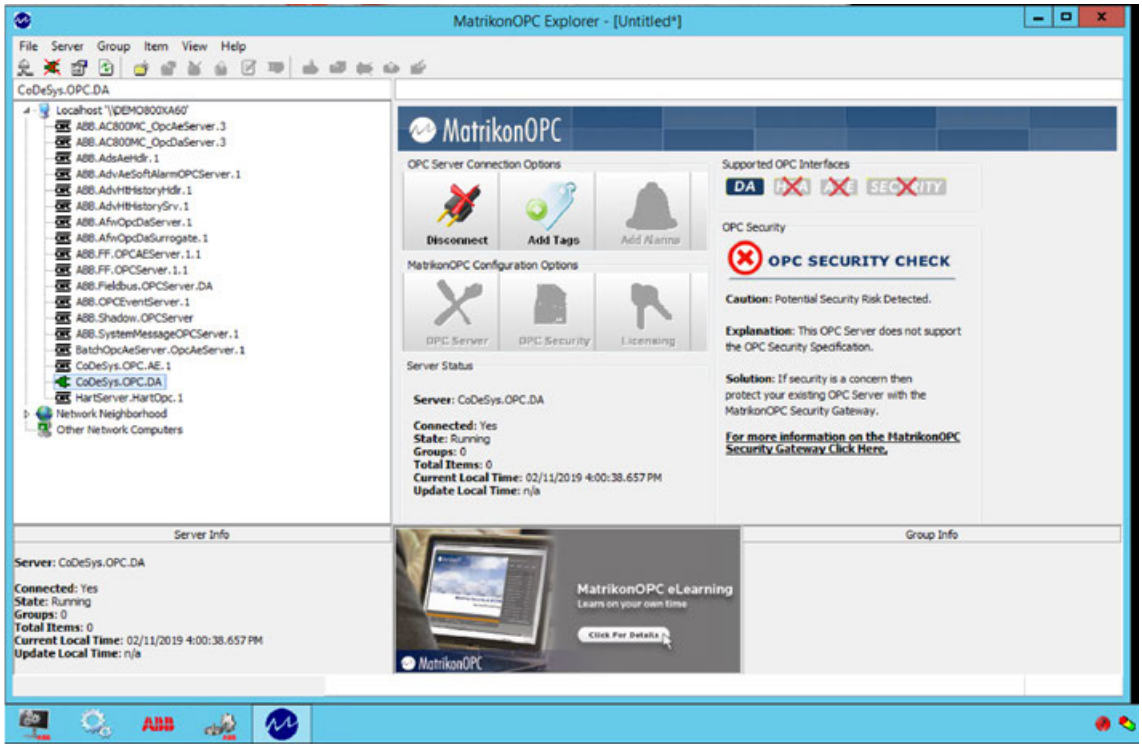
1. Test if you can ping the PLC from 800xA engineering node:



2. Install an OPC test client, for example from Matrikon.

> *More information on test clients can be found in the the Automation Builder help: ↳ Chapter 1.6.5.5.1.2.3.3 “OPC clients for tests” on page 6284*

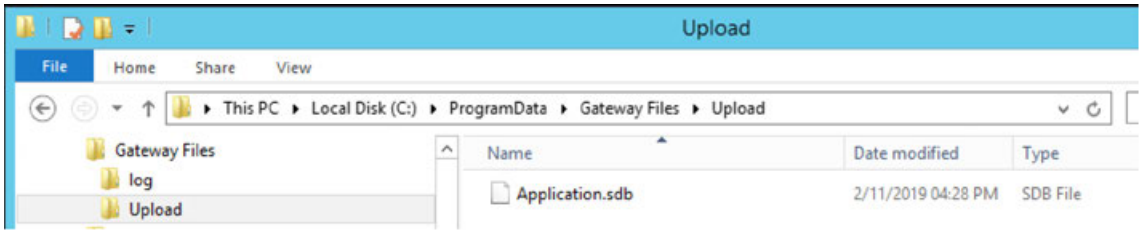
3. Start the OPC test client (here: *“Matrikon OPC Explorer”*) and connect the *“CoDeSys.OPC.DA”*:



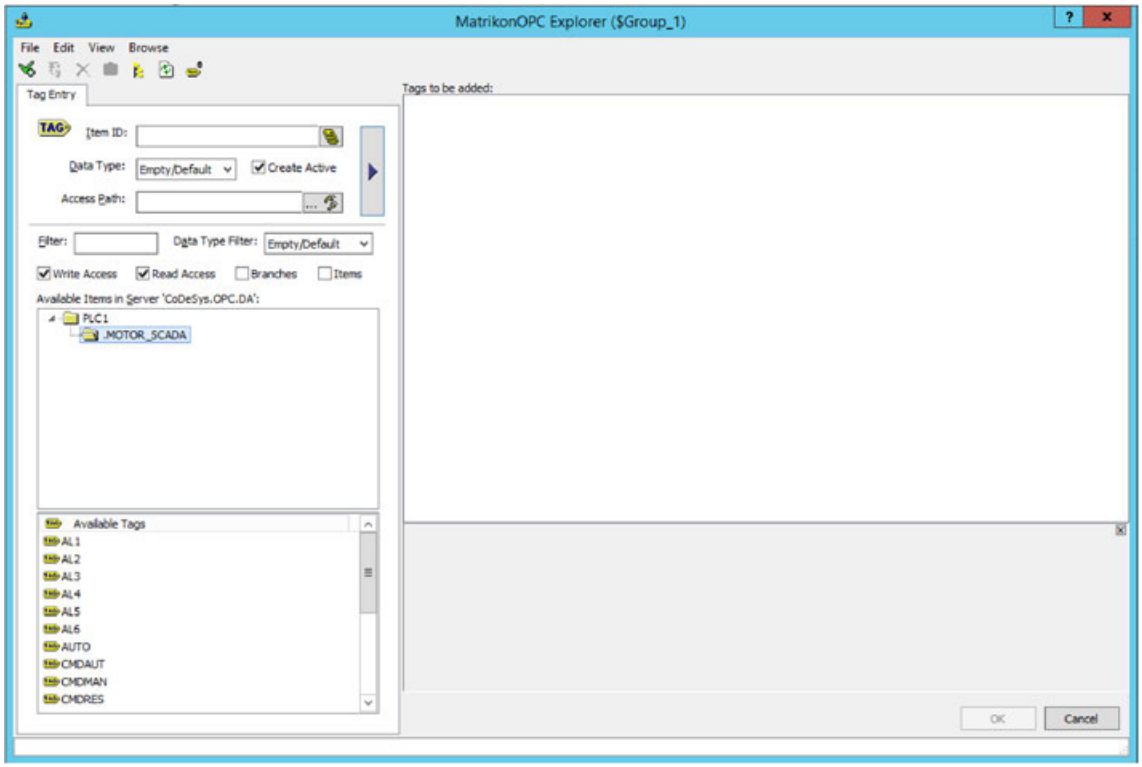
Checkpoint 4

4. After clicking *[eConnect]* the *“Tray Icon”* of the CODESYS gateway turns from idle to active .

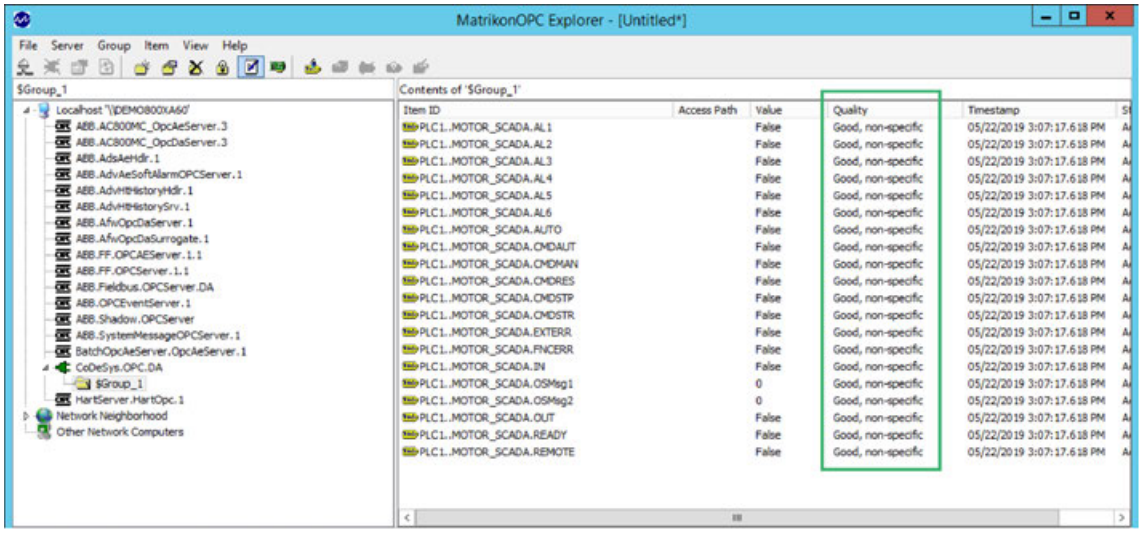
   Furthermore, the sdb file is uploaded from the PLC to the following Gateway folder: C:\ProgramData\Gateway Files\Upload



5. Add tags.



6. Watch the values of the added tags.



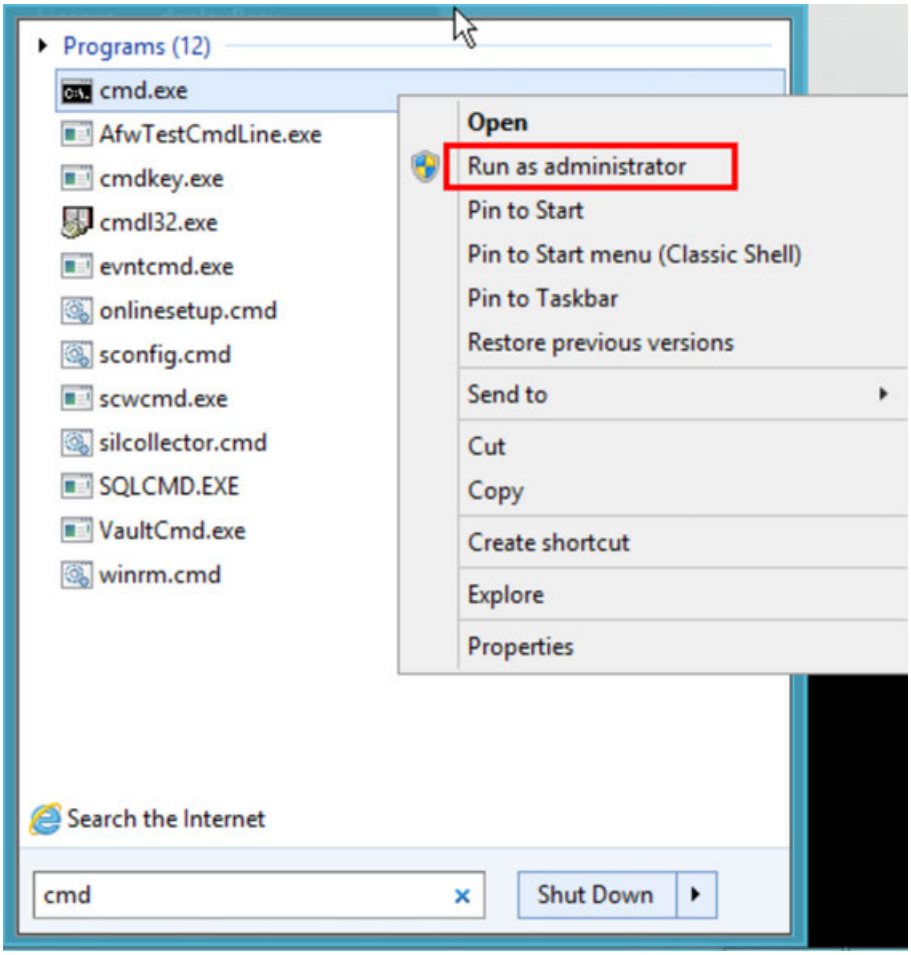
Checkpoint 5 7. Quality of the tag must be *“Good”*

Register OPC server as system service

For use with 800xA it is important that the OPC server runs with Session ID 0 (like all other 800xA services).

Therefore OPC server must be registered as service.

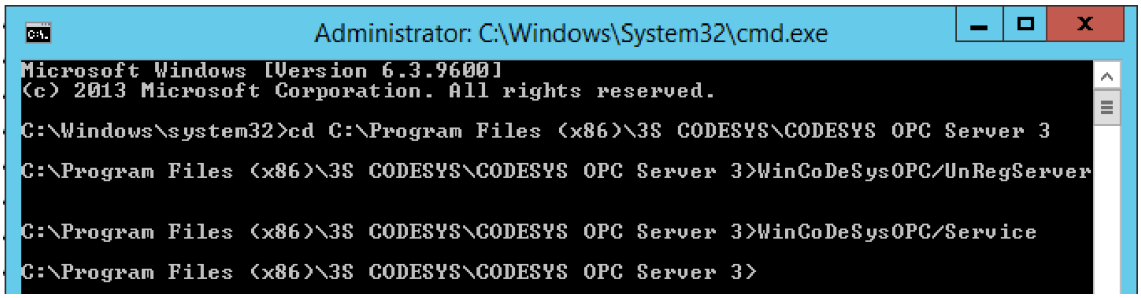
1. Start a command prompt as administrator.



2. Go to the CoDeSysOPC V3 installation folder.
3. Unregister the OPC server with *“WinCoDeSysOPC/UnRegServer”*.
4. Register the OPC server as system service with *“WinCoDeSysOPC/Service”*

Checkpoint 6

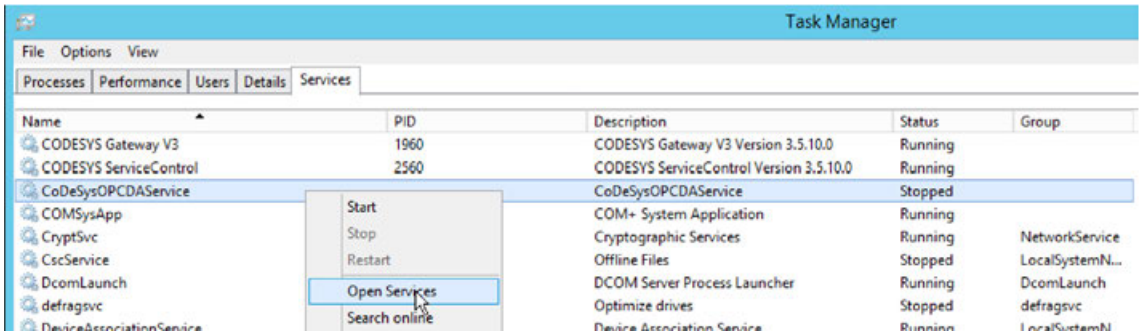
5. During this procedure there should be no errors, terminal should look like this:



⇨

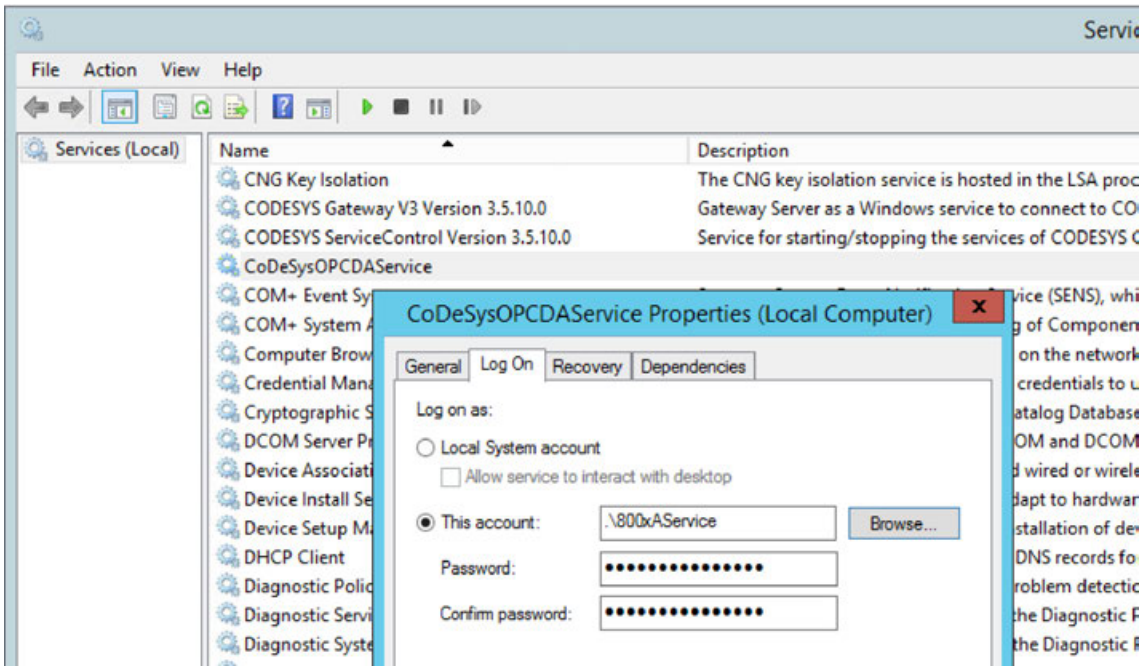
Same information in Automation Builder online help: ↳ Chapter 1.6.5.5.1.2.3.2.1 “Register OPC server V3 as a system service” on page 6284

6. Open Task Manager and select *"CoDeSysOPCDAService"* from the *"Service"* menu.
   Right-click the *"CoDeSysOPCDAService"* and select *"Open Services"*:



7. Double-click *"CoDeSysOPCDAService"* and configure properties, logon with 800xAService account.

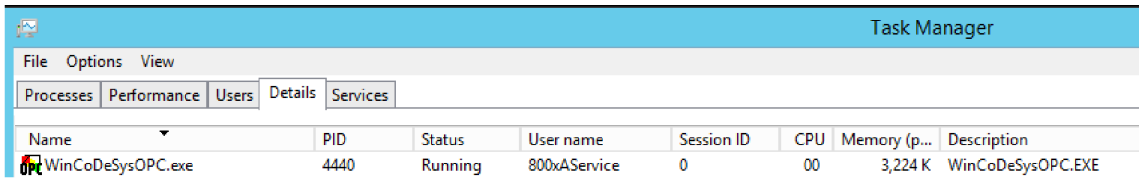
> *"800xAService" in the example, maybe different!*



8. Restart the service (right-click).

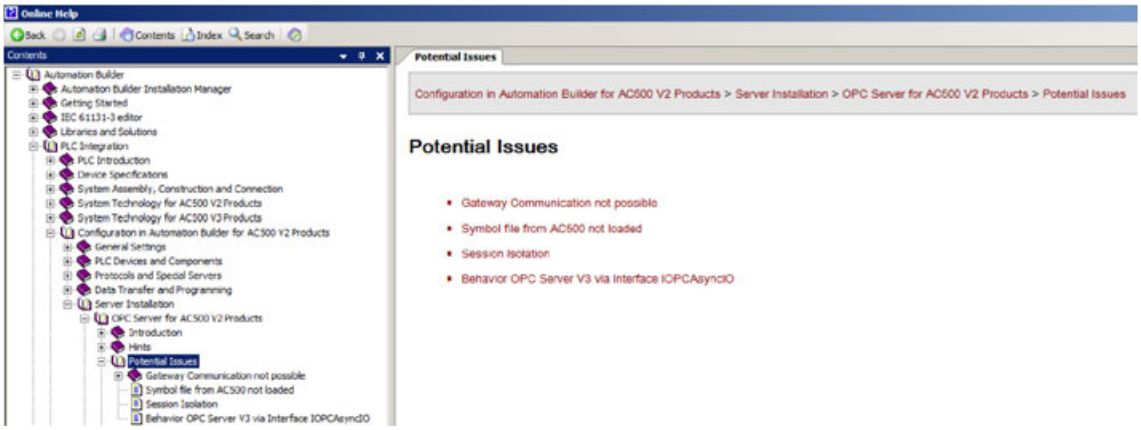
Checkpoint 7

9. Now the process runs with *User name "800xAService"* and *Session ID " 0"*.



Troubleshooting

In case of any problems please check the potential issues in the Automation Builder help:
↳ *Chapter 1.6.5.5.1.3 “Potential issues” on page 6303*



Create PLC generic control network object

This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

Configure PLC connect services

This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

Create AC500 controller and GCN configuration

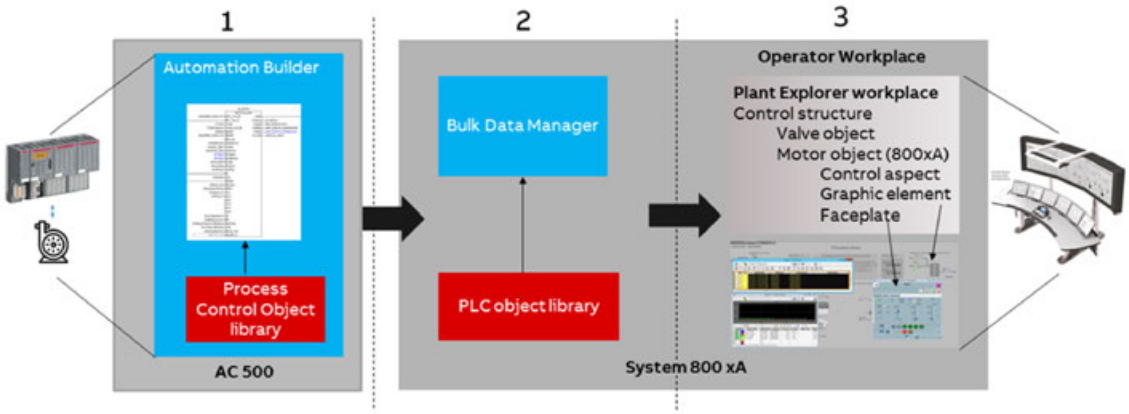
This configuration is done on the 800xA node.

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.5.10.1.5 Engineering workflow

Engineering is done in the following steps:

1. Create application in AC500 using the AC500 PCO library.
2. Create equivalent structure in *“Bulk Data Manager”*.
3. Populate PLC objects to control structure using the 800xA PLC object library.



Create function blocks in AC500 V2

1. For general introduction to AC500 configuration please refer to the chapter *Chapter 1.2 “Getting started” on page 12*.
2. Create a new project: *Chapter 1.6.5.1.1 “Project handling” on page 5757*
3. Transfer data to CODESYS: *Chapter 1.6.5.4.1 “Data transfer and CODESYS programming” on page 6196*
4. Program your application in *Chapter 1.4.1 “Development system” on page 145*

Function blocks of PCO library

The PCO library contains the AC500 function blocks (motors, valves, …) which can be integrated into 800xA.

The function blocks can be controlled and monitored by 800xA during operation.

Installation of the PCO library is described in *Chapter 1.5.10.1.2.2 “Install AC500 PCO Library on AC500 engineering node” on page 3041*.

All function blocks are described in *Chapter 1.5.10.2 “PCO library - function block description (V2)” on page 3062*.

Create instances using bulk data manager

> *The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.*

1.5.10.1.6 Capacity and Performance

AC500 function block performance

The PCO (Process Control Object) Library is usable on the whole range of AC500 platform including AC500-eCo PLC's.

The maximum number of function blocks per CPU is limited:

- By the available program memory for each CPU type.
- By the available speed for each CPU type.
- By the PLC cycle time which needs to be set to achieve desired CPU load.

The table below portrays the performance overview of various CPU's. It shows two typical configurations as examples.

These examples can be used by user to understand the performance of PLC's and choose the PLC based on application.

Small configuration

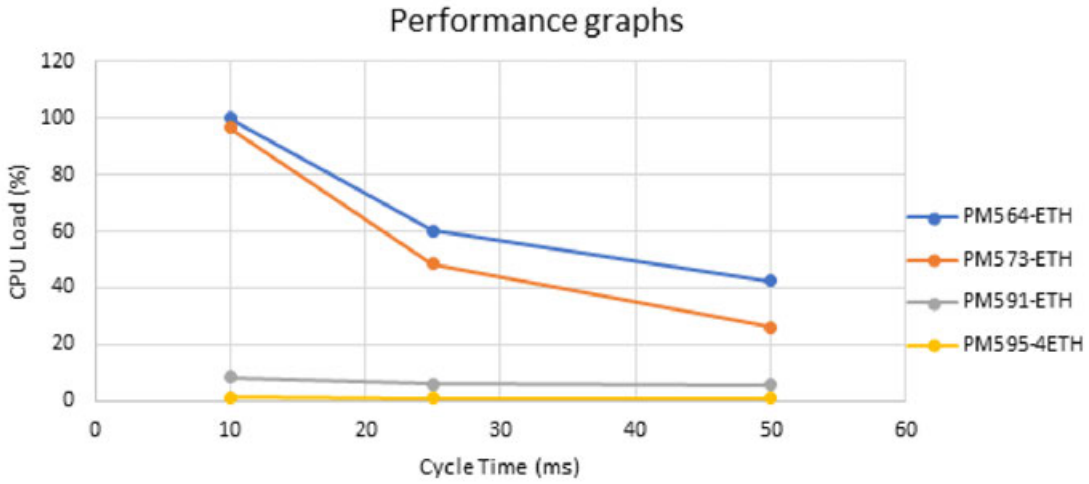
It uses 39 instances of function blocks. The various blocks used are:

Function block	Number of Instances
PCO_BINSET	1
PCO_BIN	12
PCO_ALARM	6
PCO_ANASET	1
PCO_ANA	1
PCO_ANAALM	9
PCO_ANALIM	1
PCO_MOT	3
PCO_VALV	2
PCO_MOTCON	1
PCO_PIDCON	1
PCO_VALVCON	1
Total	**39**

When these instances of function blocks were downloaded the memory coverage in various PLC's is shown in the following table:

PLC	Required User Data Memory (% used from total available)
PM564-ETH	7730 → 75 %
PM573-ETH	7730 → 3 %
PM591-ETH	7730 → 0.18 %
PM595-4ETH	7730 → 0.05 %

The cycle time versus the CPU load for various PLC's are shown in table below. The user needs to select the cycle time based on desired CPU load. Cycle time and CPU load are inversely proportional and are depicted in the following figure. It is not desirable for the CPU load to be 100 % and hence the cycle time needs to be chosen likewise.



Medium configuration

It uses 49 instances of function blocks. The various blocks used are:

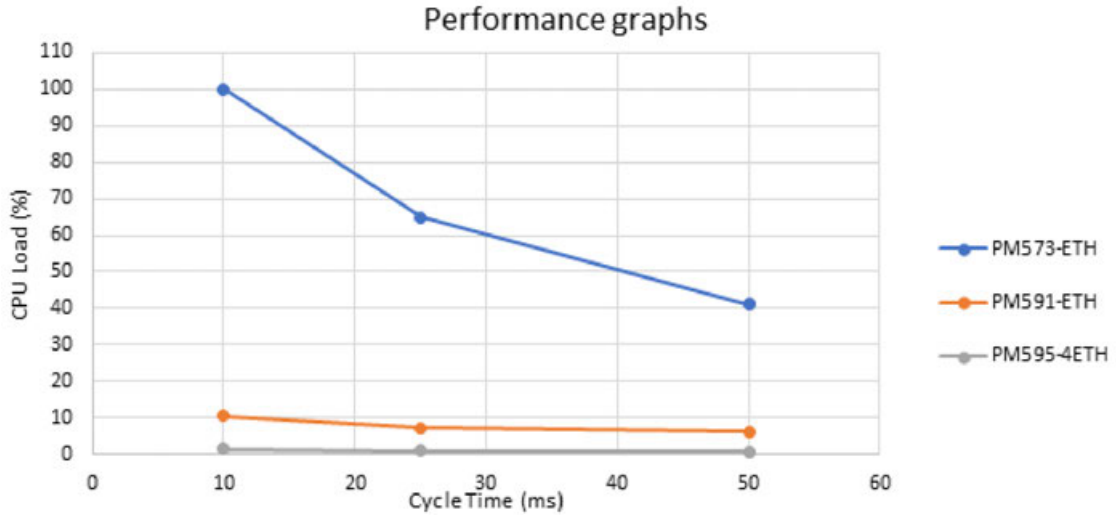
Function block	Number of Instances
PCO_BINSET	3
PCO_BIN	6
PCO_ALARM	2
PCO_ANASET	2
PCO_ANA	1
PCO_ANAALM	9
PCO_ANALIM	1
PCO_MOT	2
PCO_VALV	1
PCO_MOTCON	7
PCO_PIDCON	1
PCO_VALVCON	14
Total	**49**

When these instances of function blocks were downloaded the memory coverage in various PLC's is shown in the following table:

PLC	**Required User Data Memory (% used from total available)**
PM573-ETH	18258 → 7 %
PM591-ETH	18258 → 0.44 %
PM595-4ETH	18258 → 0.11 %

Here the program could not be downloaded to PM564-ETH because of the less memory available (Program Code Memory Max - 131072 Bytes) and here it is 18258 Bytes and hence PM573-ETH, PM591-ETH and PM595-4ETH were used for testing.

The cycle time versus the CPU load for various PLC's are shown in table below. The user needs to select the cycle time based on desired CPU load. Cycle time and CPU load are inversely proportional and are depicted in the following figure. It is not desirable for the CPU load to be 100 % and hence the cycle time needs to be chosen likewise.



Memory usage per function block

The table below shows the memory size of each function block of the PCO library.

This information can be used by the user to select the CPU type based on its application.

Component	Required User Program Memory (Bytes) (Max - 524288)	Required User Data Memory (Bytes) (Max - 65536)		Size of individual FB (Bytes)
		Number of 1 Function Block Instance	Number of 5 Function Block Instances	
Project no program	4338 (0 %)	1958 (2 %)		
Project with Library	4354 (0 %)	1962 (2 %)		
PCO_BINSET	5070 (0 %)	1976 (3 %)	2008 (3 %)	14
PCO_BIN	5174 (0 %)	1991 (3 %)	2087 (3 %)	29
PCO_ALARM	5206 (0 %)	1991 (3 %)	2087 (3 %)	29
PCO_ANASET	5018 (0 %)	1978 (3 %)	2010 (3 %)	16
PCO_ANA	5174 (0 %)	1974 (3 %)	1998 (3 %)	12
PCO_ANAALM	19522 (3 %)	2166 (3 %)	2870 (4 %)	204
PCO_ANALIM	12498 (2 %)	2078 (3 %)	2510 (3 %)	116
PCO_MOT	10922 (2 %)	2288 (3 %)	3488 (5 %)	326
PCO_VALV	11402 (2 %)	2326 (3 %)	3670 (5 %)	364
PCO_MOTCON	30814 (5 %)	2772 (4 %)	5748 (8 %)	810
PCO_PIDCON	25310 (4 %)	2468 (3 %)	4324 (6 %)	506
PCO_VALVCON	26686 (5 %)	2550 (3 %)	4678 (7 %)	588

The percentages in the bracket denote the memory consumed out of the total available. Tested using PM566-ETH.

800xA performance

The information can be found in the corresponding chapter in the ABB Ability™ System 800xA User Manual 2PAA119792.

1.5.10.2 PCO library - function block description (V2)

1.5.10.2.1 Scope and structure of this document

The purpose of this library description is to explain different components of the Process Control Object (PCO) library.

1.5.10.2.2 Process control object (PCO) library

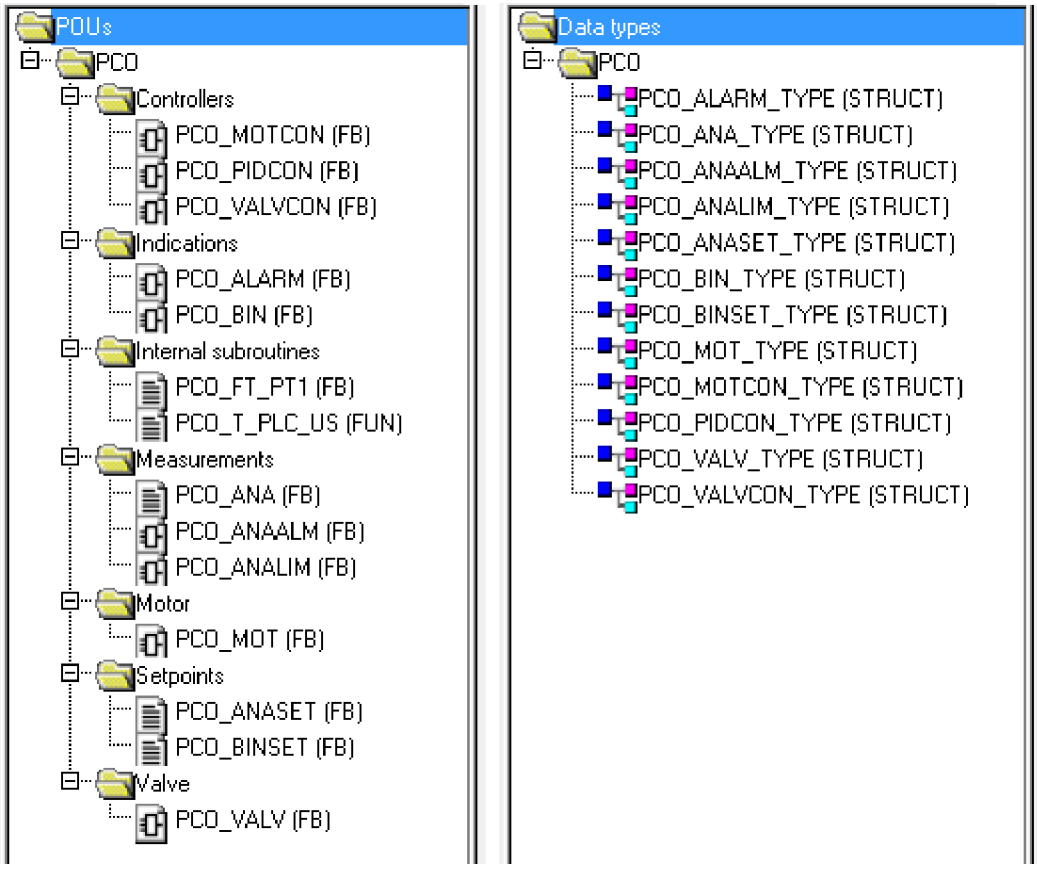
Process Control Object (PCO) Library is developed for use in any process application.

It includes function blocks for controlling motors, valves and measurements.

There are no prerequisites.

Components of the PCO library

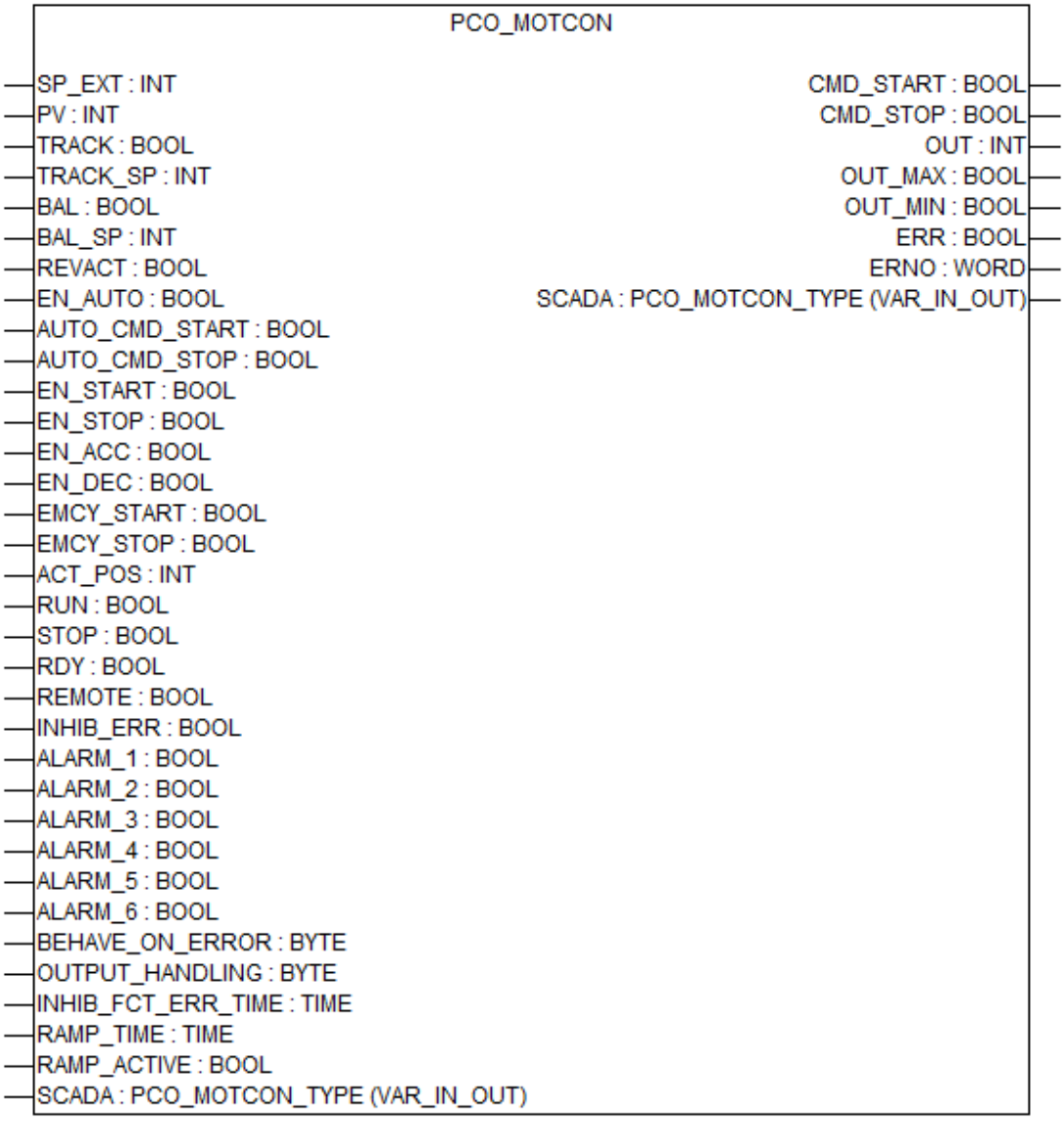
The PCO library contains the following function blocks and structures:



Function blocks

Controllers

PCO_MOTCON



This function block is designed for controlling a variable speed drive motor.

The function block is similar to PCO_MOT (Fixed speed motor controller), but in addition to PCO_MOT the function block has a built in PID controller used for speed control of the motor.

The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.
- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.
- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.

To make the controller follows various setpoints, the following variables need to be set:

The values set below are default values, the user can change based on the application.

Settings in SCADA side	Data type	When the controller must follow		
		Internal Setpoint	External Setpoint	Tracking Setpoint
SCADA.KP_PAR	INT	1	1	1
SCADA.TD_PAR	INT	0	0	0
SCADA.TI_PAR	INT	1	1	1
SCADA.OH_PAR	INT	10000	10000	10000
SCADA.OL_PAR	INT	0	0	0
SCADA.CI_CMDON	BOOL	TRUE	-	-
SCADA.SPI_PAR	INT	Needs to be set	-	-
SCADA.CX_CMDON	BOOL	-	TRUE	-

Settings in FB side	Data type	When the controller must follow		
		Internal Setpoint	External Setpoint	Tracking Setpoint
EN_AUTO	BOOL	TRUE	TRUE	TRUE
SP_EXT	INT	-	Needs to be set	-
TRACK	BOOL	-	-	TRUE
TRACK_SP	INT	-	-	Needs to be set
EN_ACC	BOOL	TRUE	TRUE	TRUE
EN_DEC	BOOL	TRUE	TRUE	TRUE

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.

The following variable needs to be configured to set the controller to manual mode.

Settings in FB side	Data type	Controller Manual Mode
No settings need to be done.		

Setting in SCADA side	Data type	Controller Manual Mode
SCADA.CM_CMDON	BOOL	Variable Rising edge needs to be given to reset auto mode.

The manual positioning can be done in two ways, both controlled form the SCADA faceplate.

1. Stepwise 1 % up / down or 5 % up / down.

   This is manually set by the following variables on the SCADA faceplate:

   - SCADA.CMDUP1
   - SCADA.CMDUP5
   - SCADA.CMDDW1
   - SCADA.CMDDW5

   The controller output varies based on the command given.

2. Download a new position. Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.

If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

When the controller is set to TRACK mode, the controller gets set to auto mode, whereas when the controller is set to BAL mode, the controller is not in auto mode. In both cases the controller output is not dependent on the actual process value.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_START and CMD_STOP can be configured. E.g. if CMD_START and CMD_STOP should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.)

Set motor to auto mode

The motor can be switched to automatic mode or manual mode, if EN_AUTO input is TRUE in the function block.

Settings in FB side	Data type	Set Motor to Auto Mode
EN_AUTO	BOOL	ACTIVE
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Motor to Auto Mode
SCADA.CMDAUT	BOOL	Rising edge needs to be given.

Start / stop motor in auto mode

Settings in FB side	Data type	Start / Stop Motor in Auto Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_START	BOOL	ACTIVE
EN_STOP	BOOL	ACTIVE
AUTO_CMD_START	BOOL	Rising edge needs to be given to AUTO_CMD_START input of FB to start the motor.
AUTO_CMD_STOP	BOOL	Rising edge needs to be given to AUTO_CMD_STOP input of FB to stop the motor.

Settings in SCADA side	Data type	Start / Stop Motor in Auto Mode
No settings need to be done.		

Set motor to manual mode

The motor is forced to manual mode. EN_AUTO = FALSE.

The motor can be operated in manual mode if EN_AUTO is either TRUE or FALSE.

The difference is that if EN_AUTO is TRUE, the switching between Auto and Manual is possible, else the motor and controller are both to Manual.

Settings in FB side	Data type	Set Motor to Manual Mode
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Motor to Manual Mode
SCADA.CMDMAN	BOOL	Rising edge of SCADA.CMDMAN input to reset auto mode if it was set before

Start / stop motor in manual mode

Settings in FB side	Data type	Start / Stop Motor in Manual Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_START	BOOL	ACTIVE
EN_STOP	BOOL	ACTIVE

Settings in SCADA side	Data type	Start / Stop Motor in Manual Mode
CMDSTR	BOOL	Rising edge of CMDSTR needs to be given.
CMDSTP	BOOL	Rising edge of CMDSTP needs to be given.

In automatic mode the motor can be started/stopped using the function block inputs (AUTO_CMD_START/AUTO_CMD_STOP).

In manual mode the motor can be started and stopped from the SCADA faceplate.

The function block includes supervision of the feedback signals RUN and STOP from the motor.

If the motor fails to run or stop within the supervision time INHIB_FCT_ERR_TIME, it will generate a functional error.

If the function block input RDY is FALSE, then an external error is generated. This function block can be used in combination with the object type MOTCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Examples

The function block is used for controlling a variable speed drive motor, which in turn can be used for controlling the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Pump
- Main power supply, circuit breaker (or similar)

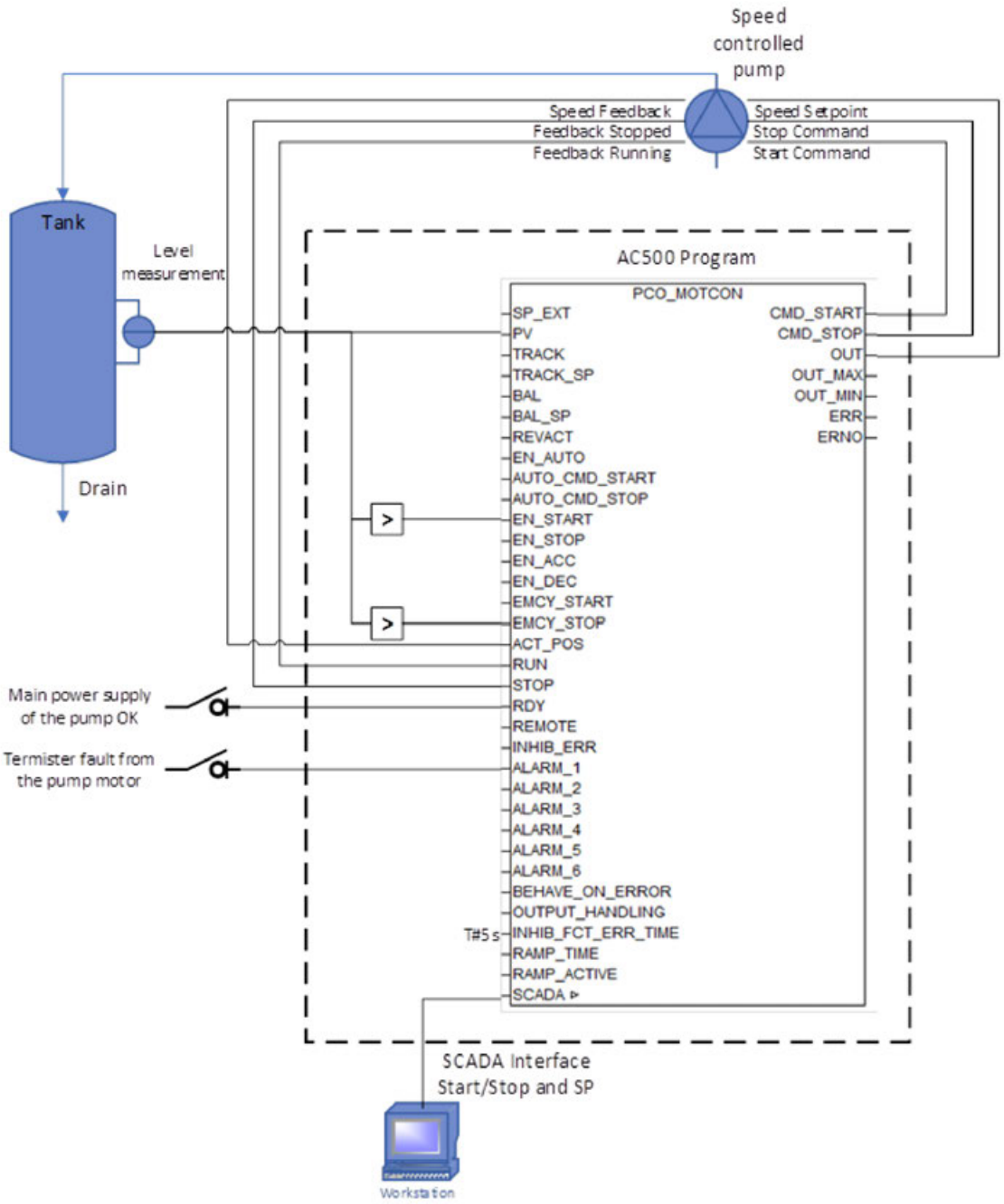
Manual mode:

- Pump controlled by the operator.

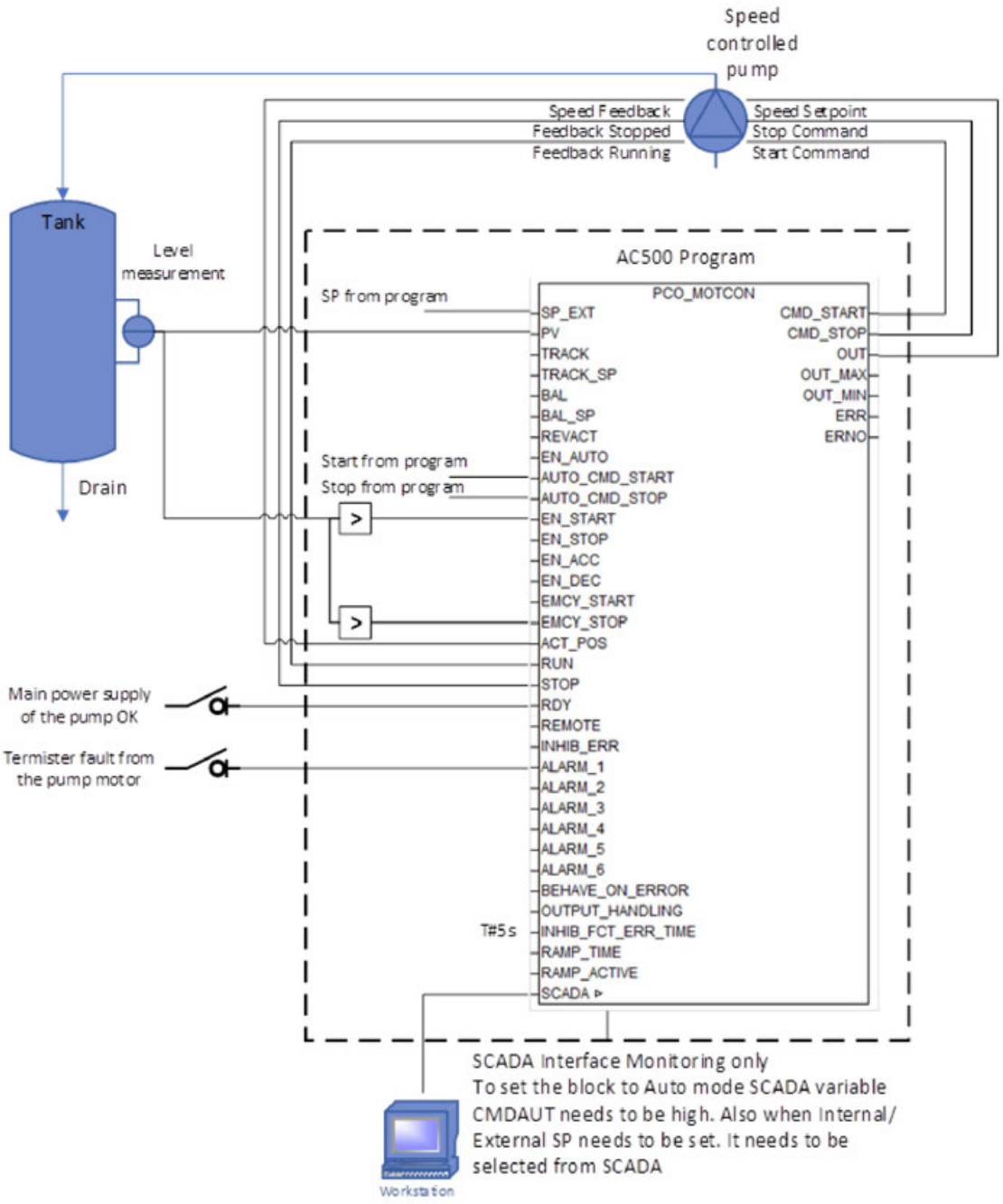
Automatic mode:

- Pump controlled by the AC500 controller.

PCO_MOTCON manual mode example



PCO_MOTCON automatic mode example



Scaling

Scaling of the SCADA Parameters

In the 800xA system the values coming from the PLC are shown by default 1:1 in the operator screens and faceplates.

The default scaling is 0 ... 100 from PLC is shown as 0 % … 100 % or 0 % ... 100.00 % for the operator (SCADA). Therefore a scaling must be done for most signals either in the *“Bulk Data Manager”* or directly in the *Engineering Workplace* > *Control Structure* > *…* > *Signal Configuration* > *Range*.

Assuming that all signals on the operator workplace should be shown in 0 % … 100 % the following table gives an overview of the scaling of the PLC values in the 800xA system.

PVmin and PVmax are here the minimum and maximum of the PV input, reflecting the process value in the PLC program. The PV typically comes from the output of the function block PCO_ANAALM. This scales its IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN … SCALE_MAX).

OPC	**FB**	**AC500**		**800xA**		**Remark**
Signal name SCADA.x	**In/Output-name**	**Range default**	**Range actual program**	**PLC Range def.: 0…100**	**SCADA Range def: 0…100% or 0…100.00%**	
PV_VALUE	PV	0…27648	PVmin…PVmax PVmin … PVmax	PVmin…PVmax	0…100	
SPX_VALUE	SP_EXT	0…27648	PVmin…PVmax	PVmin…PVmax	0…100	
	TRACK_SP	0…10000	0…10000 fix			
	BAL_SP	0…10000	0…10000 fix			
ACT_VALUE	ACT_POS	0…10000	ACTmin… ACTmax	ACTmin…ACTmax	0…100	
OUT_PAR	-	0…10000	0…10000 fix	0…10000	0…100	
OL_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
OH_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
	OUT	0…27648		-	-	
OUT_VALUE	-	0…10000		0…10000	0…100	Scaled from OUT. Seen in 800xA
SPI_PAR	-	0…27648	-	PVmin…PVmax	0…100.00	Set from 800xA
KP_PAR	-	0…100	0…(27648 / (PVmax- PVmin))	PVmin…PVmax or PVmin/100…PVmax/100	0…100.00 (%) or 0..1.00 (as factor)	
TD_PAR	-			0...1000	0...1000.00	
TI_PAR	-			0...1000	0...1000.00	

Scaling of the SCADA.PV_VALUE

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to an analog output of the AC500 I/Os.

To show the PV in % in 800xA, it must be scaled accordingly in the 800xA signal configuration:

SCADA.Signal_Configuration.Range.Low limit = 0

SCADA.Signal_Configuration.Range.High limit = 100

SCADA.Signal_Configuration.Range.Low limit in PLC = 0

SCADA.Signal_Configuration.Range.High limit in PLC = 27648

Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.

Scaling of the Parameter SCADA.KP_PAR

If the setpoint and the process value (PV) do not have equal scaling (0 ... 27648), there is a need for scaling the SCADA.KP_PAR parameter in 800xA to achieve the right proportional gain.

The scaling of the coefficient of proportionality (SCADA.KP_PAR) is dependent on the scaling of the process value (PV).

The PV typically comes from the output of the function block PCO_ANAALM. This scales the IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN ... SCALE_MAX).

The scaling of the KP [%] is then calculated by the formula:

$$KP[\%] = \frac{Out_{\max} - Out_{\min}}{PV_{\max} - PV_{\min}} \times 100 = \frac{27648 - 0}{SCALE_{\max} - SCALE_{\min}} \times 100$$

In the mentioned example, 100 % of KP.

$$100\ \%\ of\ KP = \frac{[27648 - 0]}{[16000 - 0]} \times 100 = 172.8$$

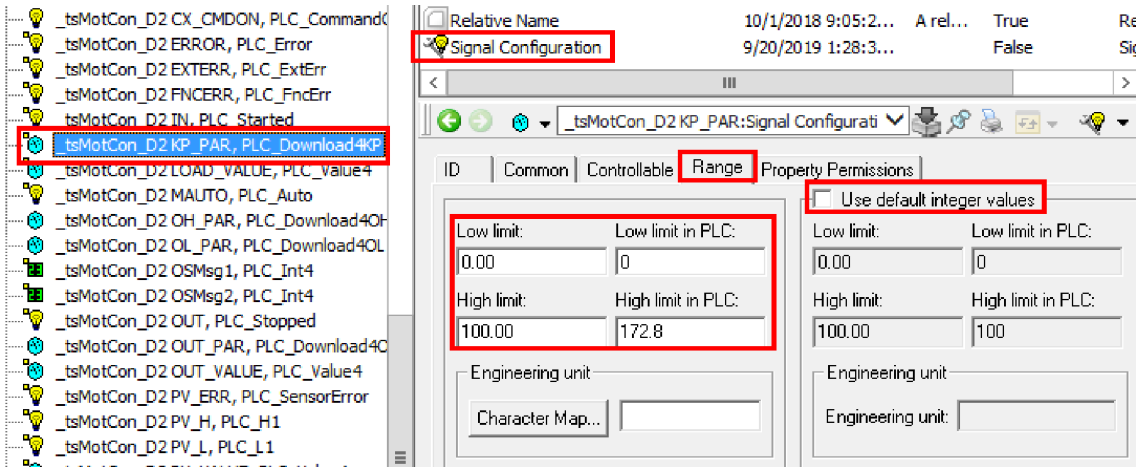
If the wanted range of SCADA.KP_PAR in SCADA is chosen to be 100 %, then the range in SCADA control structure should be adapted to

SCADA.Signal Configuration.Range.Low limit = 0

SCADA.Signal Configuration.Range.High limit =100

SCADA.Signal Configuration.Range.Low limit in PLC = 0

SCADA.Signal Configuration.Range.High limit in PLC = [27648 - 0] / [16000 - 0] × 100 = 172.8



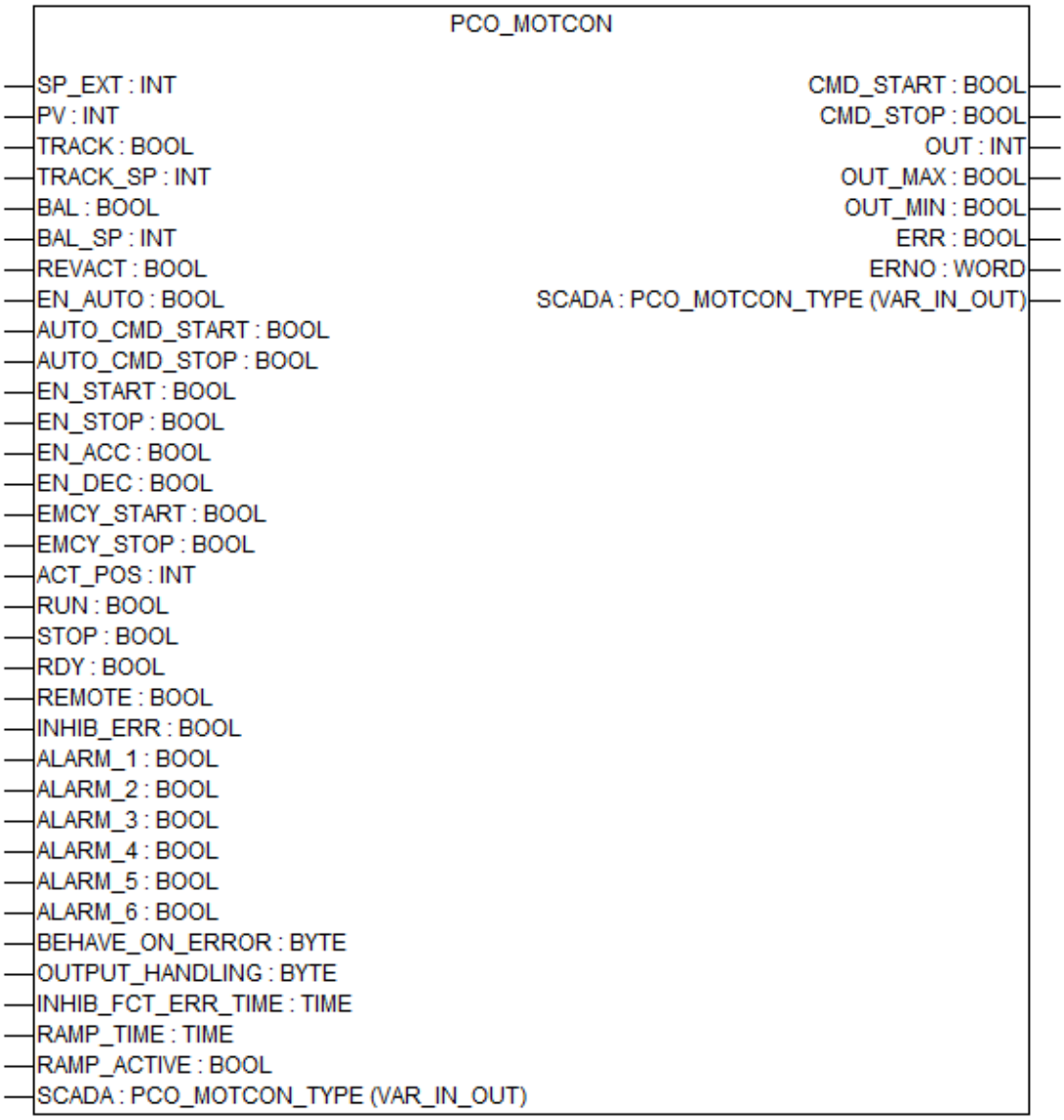
Scaling of the TRACK_SP AND BAL_SP

Input range of TRACK_SP or BAL_SP is 0 ... 10000 and the controller OUT range is 0 ... 27648.

So if the TRACK/BAL_SP is x then the controller output is defined as:

$$OUT = \frac{x}{10000} \times 27648$$

Input description



SP_EXT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

External setpoint.

Calculated setpoint or output from another PID controller (e.g. PCO_PIDCON) in a cascade control system.

PV

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Process value (PV) to be controlled.

TRACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Tracking mode.

PCO_MOTCON can be set in Tracking mode if the value of TRACK is TRUE.

Output of the controller will be set to TRACK_SP.

Tracking mode has higher priority compared to balancing mode.

TRACK is independent of operation mode of PCO_MOTCON.

TRACK_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Tracking setpoint.

Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.

BAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Balancing mode.

PCO_MOTCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.

Output of the controller will be set to BAL_SP.

BAL is independent of operation mode of PCO_MOTCON.

BAL_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Balancing setpoint.

Output of the controller will be set to BAL_SP if value of BAL is TRUE.

REVACT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Reverse Action

Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.

Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

EN_AUTO

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable automatic mode of the PCO_MOTCON.

PCO_MOTCON can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_MOTCON is forced in manual mode if value of EN_AUTO is FALSE.

AUTO_CMD_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the motor to start if it is in automatic mode.

E.g.AUTO_CMD_START could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

AUTO_CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the motor to stop if the actuator is in automatic.

AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

EN_START

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable start of the motor.

Motor can be started if value of EN_START is TRUE.

Motor cannot be started if value of EN_START is FALSE.

EN_START is independent of operation mode of PCO_MOTCON.

EN_START has no effect in case of emergency start of motor EMCY_START.

EN_START has no effect on a motor already running.

EN_STOP

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable stop of the motor.

Motor can be stopped if value of EN_STOP is TRUE.

Motor cannot be stopped if value of EN_STOP is FALSE.

EN_STOP is independent of operation mode of PCO_MOTCON.

EN_STOP has no effect in case of emergency stop of motor EMCY_STOP.

EN_STOP has no effect on an already stopped motor.

EN_ACC

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable increase of controller output.

Controller can increase output if value of EN_ACC is TRUE.

Independent of operation mode of PCO_MOTCON.

EN_DEC

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable decrease of controller output.

Controller can decrease output if value of EN_DEC is TRUE.

Independent of operation mode of PCO_MOTCON.

EMCY_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency start of motor.

Motor will start if value of EMCY_START is TRUE and value of EMCY_STOP is FALSE.

If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.

Independent of operation mode of PCO_MOTCON.

EMCY_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency stop of motor.

Motor will stop if value of EMCY_STOP is TRUE.

If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.

Independent of operation mode of PCO_MOTCON.

ACT_POS

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Feedback speed of motor.

RUN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the motor.

Motor is running if the value of RUN is TRUE.

Independent of operation mode of PCO_MOTCON.

STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the motor.

Motor is stopped if the value of STOP is TRUE.

Independent of operation mode of PCO_MOTCON.

RDY

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Motor is ready for operation.

Motor is ready for operation if value of RDY is TRUE.

Value of RDY is FALSE results in an external error.

Independent of operation mode of PCO_MOTCON.

REMOTE

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Allow motor to be controlled from SCADA.

Value of REMOTE is TRUE, control from SCADA and function block is enabled.

Value of REMOTE is FALSE, motor is controlled only from function block.

PCO_MOTCON will align the block according to feedback signals.

Independent of operation mode of PCO_MOTCON.

INHIB_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of alarms.

If value of INHIB_ERR is TRUE, all alarms from PCO_MOTCON are suppressed.

Independent of operation mode of PCO_MOTCON.

EXT_ACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

External acknowledge of alarms.

If value of EXT_ACK is TRUE, all alarms from PCO_MOTCON are acknowledged.

Independent of operation mode of PCO_MOTCON.

Alarm_1

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 1.

If value of ALARM_1 is TRUE Alarm_1 is active.

Independent of operation mode of PCO_MOTCON.

Alarm_2

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 2.

If value of ALARM_2 is TRUE Alarm_2 is active.

Independent of operation mode of PCO_MOTCON.

Alarm_3

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 3.

If value of ALARM_3 is TRUE Alarm_3 is active.

Independent of operation mode of PCO_MOTCON.

Alarm_4

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_MOTCON.

Alarm_5

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_MOTCON.

Alarm_6

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_MOTCON.

BEHAVE_ON_ERROR

Data type	Default value	Range	Unit
BYTE	0	0 ... 3	-

Actions that need to be executed by the FB at the event of error as set by the user.

The user can set a range of values from 0 ... 3.

BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).

BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.

BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.

BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto/ Manual) of PCO_MOTCON.

OUTPUT_HANDLING

Data type	Default value	Range	Unit
BYTE	0	0 ... 2	-

Behavior of command outputs, CMD_START / CMD_STOP.

The user can set a range of values from 0 ... 2

OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.

OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP)) to remain active at RUN or STOP feedback.

With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_MOTCON.

INHIB_FCT_ERR_TIME

Data type	Default value	Range	Unit
TIME	TIME#5s	-	-

Maximum delay time from command to response from process.

If response is not received within this time limit, a function error will be generated.

RAMP_TIME

Data type	Default value	Range	Unit
TIME	TIME#30s	-	-

Actuator ramp time from 0 % ... 100 %

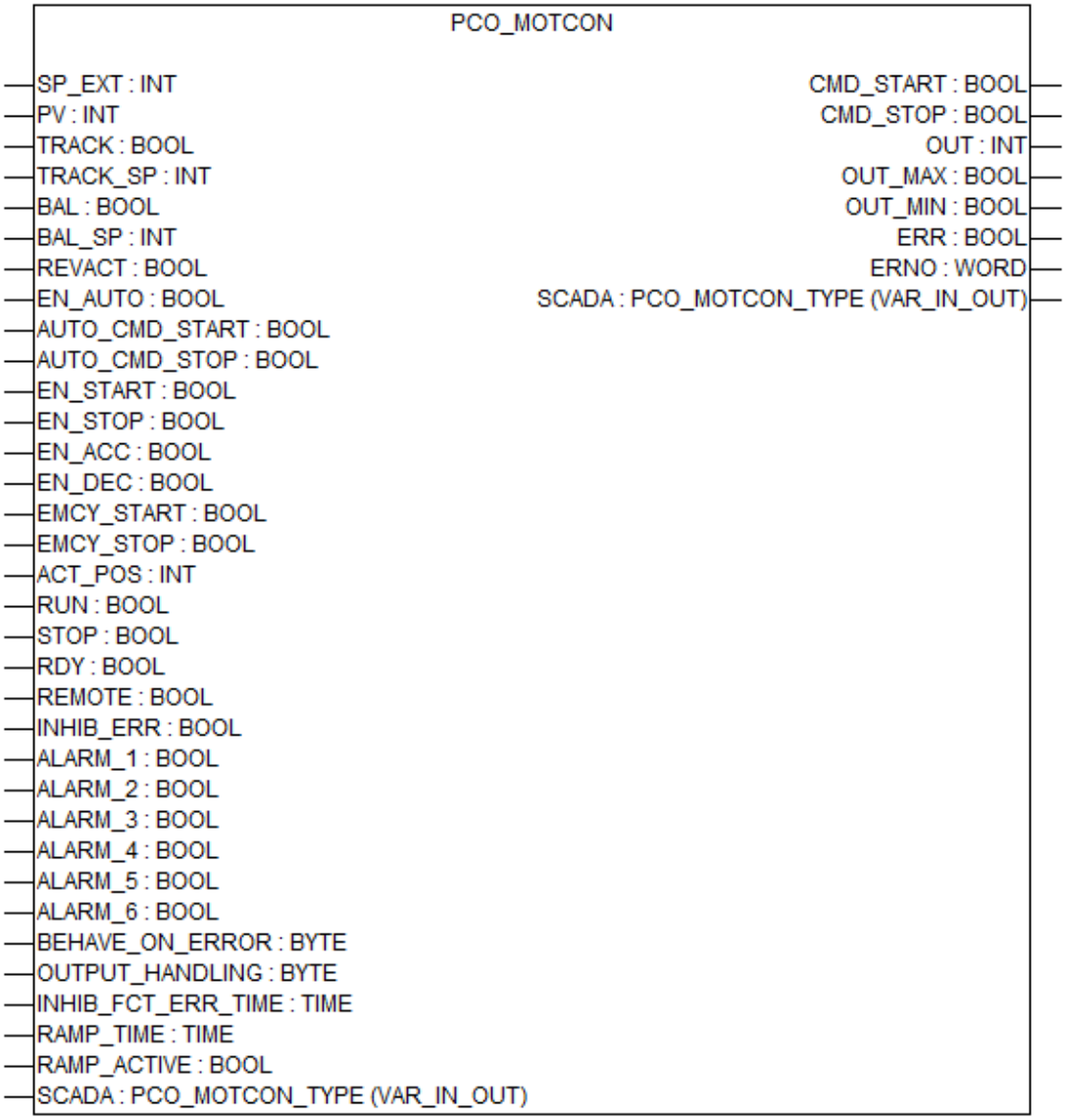
RAMP_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.

The value RAMP_ACTIVE is FALSE activates no ramp.

Output description



CMD_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output start, to be connected to hardware output.

CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output stop, to be connected to hardware output.

OUT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Controller output to the motor to attain the desired speed of the motor.

OUT_MAX

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output max limit reached, controller at max limit.

OUT_MIN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output min limit reached, controller at min limit.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, including:

- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_START
- EMCY_STOP

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

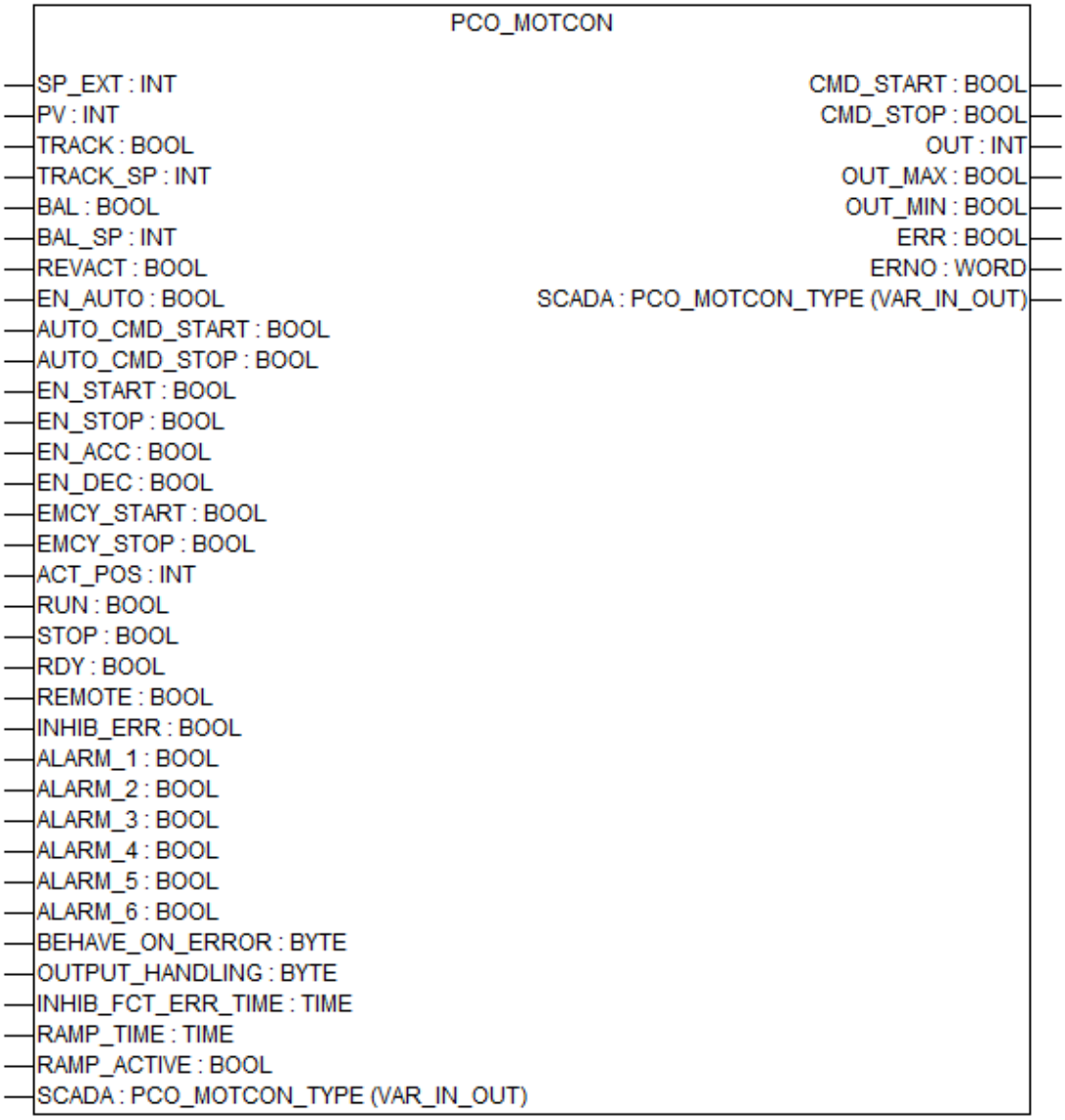
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *“Standard Function Block Libraries AC500”* in *“Error Messages of the Function Block Libraries”*.

Input/output description



SCADA

Data type	Default value	Range	Unit
PCO_MOTCON_ TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

To retain the variable value in case of power ON/OFF or download, the variable connected to the SCADA In/Output should be declared as (global) retain persistant (or use %R area) variable in the program.

Detailed information on the scaling see *“Scaling” on page 3089*.

Parameter	Data type	Description	In-/Output
SCADA.CMDSTR	BOOL	Start command in Manual mode	Input
SCADA.CMDSTP	BOOL	Stop command in Manual mode	Input

Parameter	Data type	Description	In-/Output
SCADA.CMDAUT	BOOL	Auto command	Input
SCADA.CMDMAN	BOOL	Manual command	Input
SCADA.CMDRES	BOOL	Acknowledge active alarms	Input
SCADA.MAUTO	BOOL	Motor in automatic mode	Output
SCADA.IN	BOOL	Motor is running	Output
SCADA.OUT	BOOL	Motor is stopped	Output
SCADA.REMOTE	BOOL	Motor can be controlled from OS REMOTE = FALSE → Local operation	Output
SCADA.READY	BOOL	Motor is ready for operation	Output
SCADA.FNCERR	BOOL	Functional error	Output
SCADA.EXTERR	BOOL	External error (is generated when the motor is not ready)	Output
SCADA.AL1	BOOL	Auxiliary alarm no. 1	Output
SCADA.AL2	BOOL	Auxiliary alarm no. 2	Output
SCADA.AL3	BOOL	Auxiliary alarm no. 3	Output
SCADA.AL4	BOOL	Auxiliary alarm no. 4	Output
SCADA.AL5	BOOL	Auxiliary alarm no. 5	Output
SCADA.AL6	BOOL	Auxiliary alarm no. 6	Output
SCADA.OSMsg1 *)	WORD	Word representing the status of the PCO_MOTCON	Output
SCADA.OSMsg2 *)	WORD	Word representing the status of the PCO_MOTCON	Output
SCADA.CM_CMDON	BOOL	Manual command for the controller	Input
SCADA.CI_CMDON	BOOL	Auto command to set the internal setpoint for the controller	Input
SCADA.CX_CMDON	BOOL	Auto command to set the external setpoint for the controller	Input
SCADA.CMDUP5	BOOL	Manual command to increase output by 5 %	Input
SCADA.CMDUP1	BOOL	Manual command to increase output by 1 %	Input
SCADA.CMDDW1	BOOL	Manual command to decrease output by 1 %	Input
SCADA.CMDDW5	BOOL	Manual command to decrease output by 5 %	Input
SCADA.CAUTO	BOOL	Controller is in automatic mode	Output
SCADA.SPIACT	BOOL	Internal setpoint is active	Output
SCADA.SPXACT	BOOL	External setpoint is active	Output
SCADA.CTACT	BOOL	Tracking setpoint is active	Output
SCADA.PV_H	BOOL	Process value at high limit	Output
SCADA.PV_L	BOOL	Process value at low limit	Output
SCADA.PV_ERR	BOOL	Process value error	Output
SCADA.ERROR	BOOL	For future use	Output

Parameter	Data type	Description	In-/Output
SCADA.OUT_PAR	INT	Manual position setpoint Scale 0 ... 10000	Input
SCADA.SPI_PAR	INT	Internal setpoint value	Input
SCADA.KP_PAR	INT	Coefficient of proportionality (gain) % of the controller	Input
SCADA.TD_PAR	INT	Time constant for D-part of the controller	Input
SCADA.TI_PAR	INT	Time constant for integration of the controller	Input
SCADA.OH_PAR	INT	Output high limit (0 ... 10000) of the controller	Input
SCADA.OL_PAR	INT	Output low limit (0 ... 10000) of the controller	Input
SCADA.OUT_VALUE	INT	Output from the controller Scale 0 ... 10000	Output
SCADA.PV_VALUE	INT	Process value to be controlled	Output
SCADA.SPX_VALUE	INT	External setpoint value	Output
SCADA.ACT_VALUE	INT	Actual speed of the motor, scaling as input ACT_POS (0 ... 10000)	Output
SCADA.LOAD_VALUE	INT	Load value of the motor, additional value to be shown in SCADA	Output
AspectObjectType_Motcon_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

*) structure described separately

SCADA.OSMsg1

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_MOTCON.

Bit	Description
Bit 0	Motor is running
Bit 1	Motor is stopped
Bit 2	Motor is starting
Bit 3	Motor is stopping
Bit 4	External error (Not ready)
Bit 5	Functional error
Bit 6	Motor released for start
Bit 7	Motor released for stop
Bit 8	Local operation (Not remote)
Bit 9	Common alarm (External alarm + Functional alarm + EMCY_START/EMCY_STOP + ALARM_1 ... ALARM_6)
Bit 10	Not used
Bit 11	Motor in automatic mode
Bit 12	Not used
Bit 13	Motor not released for automatic mode
Bit 14	Motor released for increase speed
Bit 15	Motor released for decrease speed

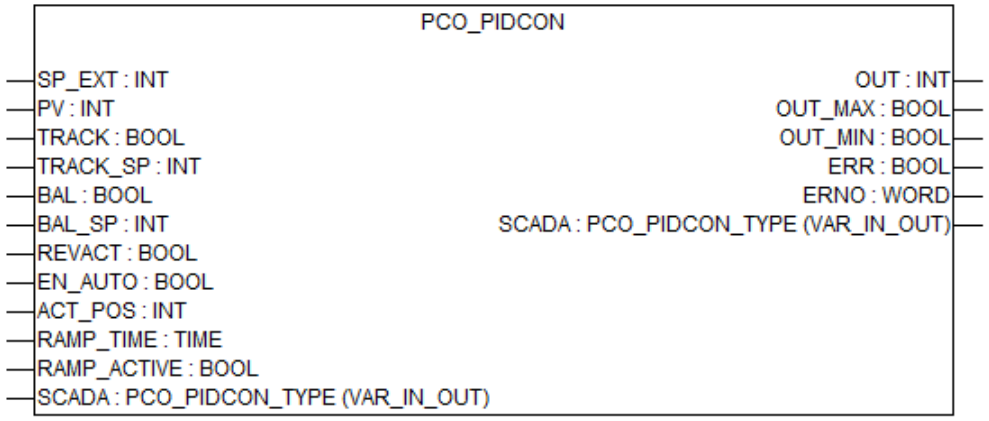
SCADA.OSMsg2

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_MOTCON.

Bit	Description
Bit 0	Emergency In
Bit 1	Emergency Out
Bit 2	Internal setpoint (PID controller)
Bit 3	External setpoint (PID controller)
Bit 4	Track setpoint (PID controller)
Bit 5	Auxiliary alarm 1
Bit 6	Auxiliary alarm 2
Bit 7	Auxiliary alarm 3
Bit 8	Auxiliary alarm 4
Bit 9	Auxiliary alarm 5
Bit 10	PID controller in automatic mode
Bit 11	PID controller in manual mode
Bit 12	Auxiliary alarm 6
Bit 13	Not used
Bit 14	Not used
Bit 15	PID Controller

PCO PIDCON



This is a standard PID controller that can be used to control any component that has an analog positioning option.

The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.
- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.
- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.

To make the controller follows various setpoints, the following variables need to be set:

The values set below are default values, the user can change based on the application.

Settings in SCADA side	Data type	When the controller must follow		
		Internal Setpoint	External Setpoint	Tracking Setpoint
SCADA.KP_PAR	INT	1	1	1
SCADA.TD_PAR	INT	0	0	0
SCADA.TI_PAR	INT	1	1	1
SCADA.OH_PAR	INT	10000	10000	10000
SCADA.OL_PAR	INT	0	0	0
SCADA.CI_CMDON	BOOL	TRUE	-	-
SCADA.SPI_PAR	INT	Needs to be set	-	-
SCADA.CX_CMDON	BOOL	-	TRUE	-

Settings in FB side	Data type	When the controller must follow		
		Internal Set-point	External Set-point	Tracking Set-point
EN_AUTO	BOOL	TRUE	TRUE	TRUE
SP_EXT	INT	-	Needs to be set	-
TRACK	BOOL	-	-	TRUE
TRACK_SP	INT	-	-	Needs to be set

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.

The manual positioning can be done in two ways, both controlled form the SCADA faceplate.

1. Stepwise 1 % up / down or 5 % up / down.

   This is manually set by the following variables on the SCADA faceplate:

   - SCADA.CMDUP1
   - SCADA.CMDUP5
   - SCADA.CMDDW1
   - SCADA.CMDDW5

   The controller output varies based on the command given.

2. Download a new position. Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.

If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

When the controller is set to TRACK mode, the controller gets set to auto mode, whereas when the controller is set to BAL mode, the controller is not in auto mode. In both cases the controller output is not dependent on the actual process value.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

6 different alarms can be connected to the function block
(ALARM_1 → ALARM_6, Torque switch etc.).

The valve can be switched to automatic or manual mode from the SCADA faceplate, if the input EN_AUTO = TRUE.

This function block can be used in combination with the object type PIDCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Examples

The function block is used for controlling a PID controller which in turn controls the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Pump

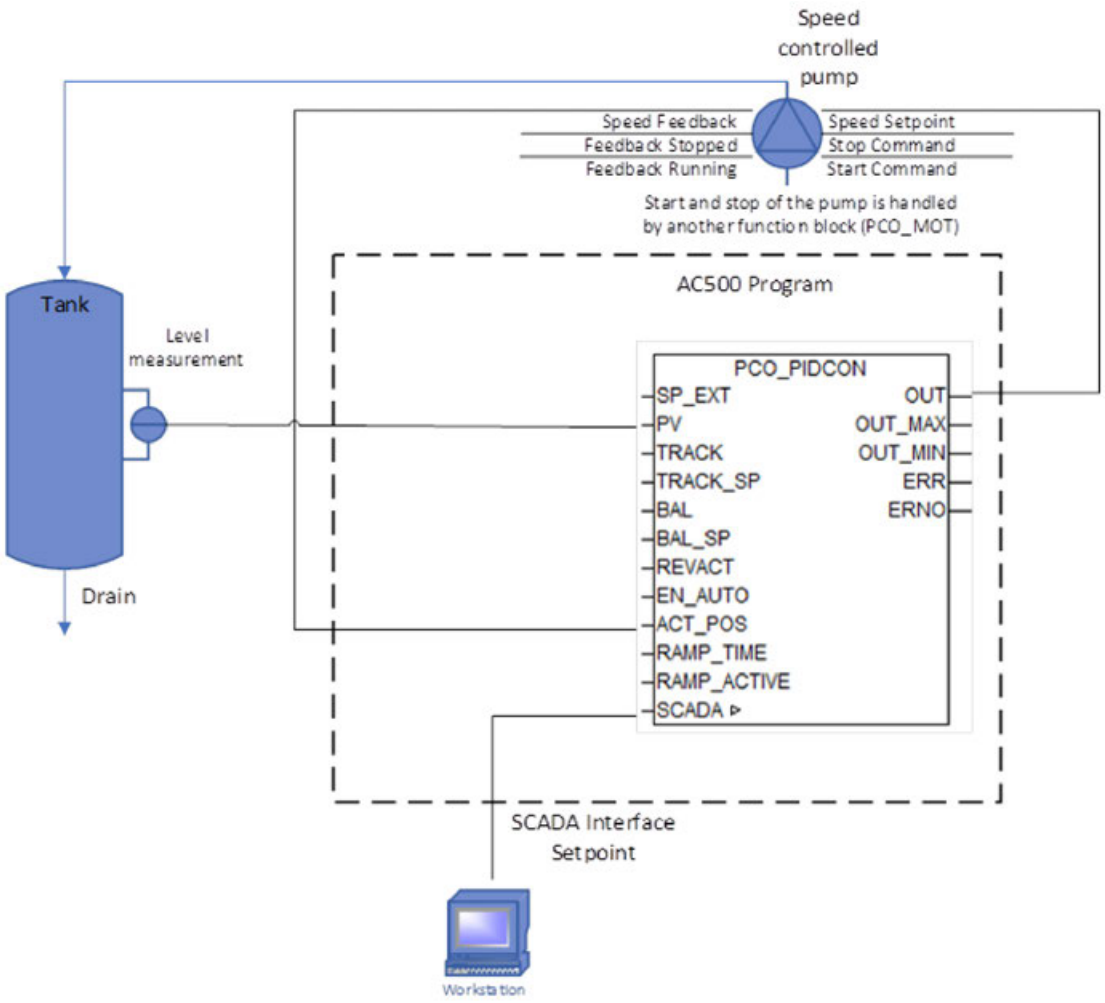
Manual mode:

- Pump controlled by the operator.

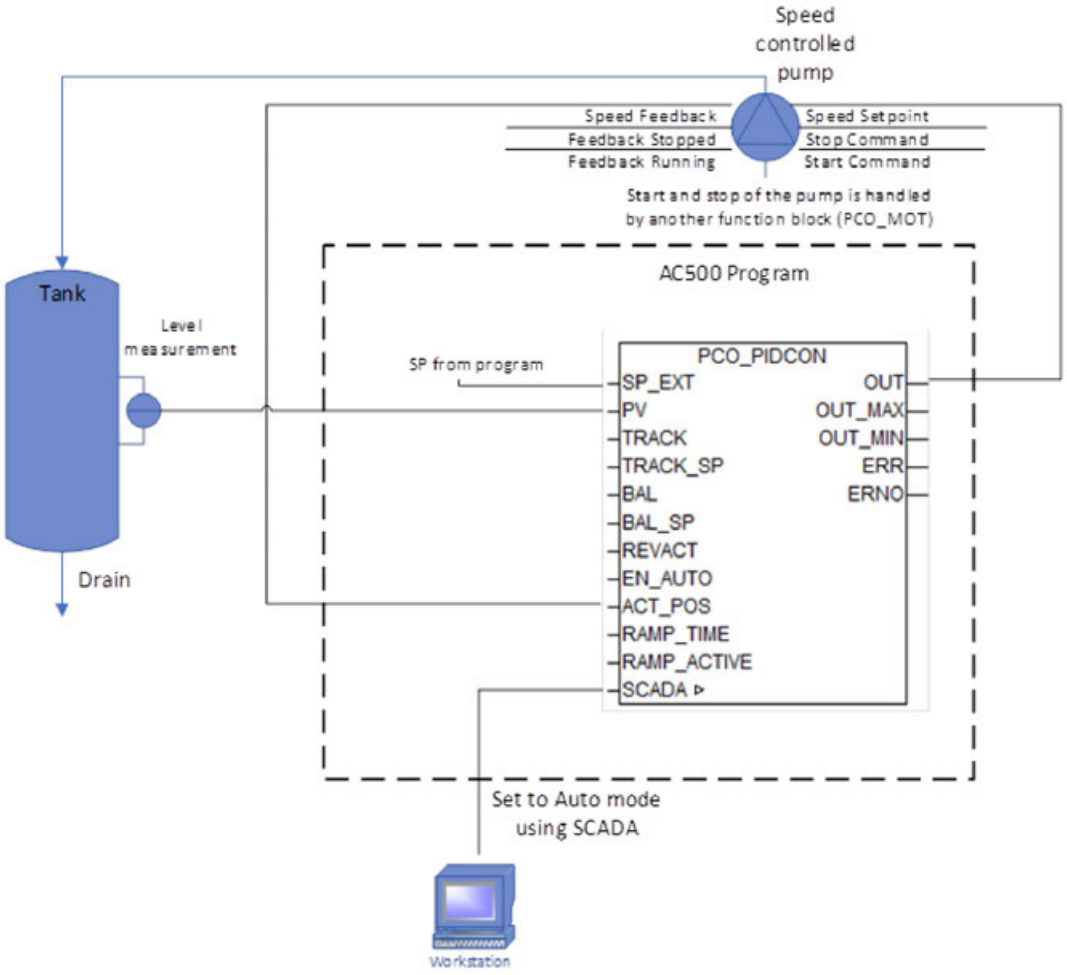
Automatic mode:

- Pump controlled by theAC500 controller.

PCO_PIDCON manual mode example



PCO_PIDCON automatic mode example



Scaling

Scaling of the SCADA Parameters

In the 800xA system the values coming from the PLC are shown by default 1:1 in the operator screens and faceplates.

The default scaling is 0 ... 100 from PLC is shown as 0 % … 100 % or 0 % ... 100.00 % for the operator (SCADA). Therefore a scaling must be done for most signals either in the *"Bulk Data Manager"* or directly in the *Engineering Workplace* > *Control Structure* > … > *Signal Configuration* > *Range*.

Assuming that all signals on the operator workplace should be shown in 0 % … 100 % the following table gives an overview of the scaling of the PLC values in the 800xA system.

PVmin and PVmax are here the minimum and maximum of the PV input, reflecting the process value in the PLC program. The PV typically comes from the output of the function block PCO_ANAALM. This scales its IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN … SCALE_MAX).

OPC	FB	AC500		800xA		Remark
Signal name SCADA.x	**In/Output-name**	**Range default**	**Range actual program**	**PLC Range def.: 0…100**	**SCADA Range def: 0…100% or 0…100.00%**	
PV_VALUE	PV	0…27648	PVmin…PVmax PVmin … PVmax	PVmin…PVmax	0…100	
SPX_VALUE	SP_EXT	0…27648	PVmin…PVmax	PVmin…PVmax	0…100	
	TRACK_SP	0…10000	0…10000 fix			
	BAL_SP	0…10000	0…10000 fix			
ACT_VALUE	ACT_POS	0…10000	ACTmin… ACTmax	ACTmin…ACTmax	0…100	
OUT_PAR	-	0…10000	0…10000 fix	0…10000	0…100	
OL_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
OH_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
	OUT	0…27648		-	-	
OUT_VALUE	-	0…10000		0…10000	0…100	Scaled from OUT. Seen in 800xA
SPI_PAR	-	0…27648	-	PVmin…PVmax	0…100.00	Set from 800xA
KP_PAR	-	0…100	0…(27648 / (PVmax- PVmin))	PVmin…PVmax or PVmin/100…PVmax/100	0…100.00 (%) or 0..1.00 (as factor)	
TD_PAR	-			0...1000	0...1000.00	
TI_PAR	-			0...1000	0...1000.00	

Scaling of the SCADA.PV_VALUE

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to an analog output of the AC500 I/Os.

To show the PV in % in 800xA, it must be scaled accordingly in the 800xA signal configuration:

SCADA.Signal_Configuration.Range.Low limit = 0

SCADA.Signal_Configuration.Range.High limit = 100

SCADA.Signal_Configuration.Range.Low limit in PLC = 0

SCADA.Signal_Configuration.Range.High limit in PLC = 27648

Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.

Scaling of the Parameter SCADA.KP_PAR

If the setpoint and the process value (PV) do not have equal scaling (0 ... 27648), there is a need for scaling the SCADA.KP_PAR parameter in 800xA to achieve the right proportional gain.

The scaling of the coefficient of proportionality (SCADA.KP_PAR) is dependent on the scaling of the process value (PV).

The PV typically comes from the output of the function block PCO_ANAALM. This scales the IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN ... SCALE_MAX).

The scaling of the KP [%] is then calculated by the formula:

$$KP[\%] = \frac{Out_{\max} - Out_{\min}}{PV_{\max} - PV_{\min}} \times 100 = \frac{27648 - 0}{SCALE_{\max} - SCALE_{\min}} \times 100$$

In the mentioned example, 100 % of KP.

$$100\ \%\ of\ KP = \frac{[27648 - 0]}{[16000 - 0]} \times 100 = 172.8$$

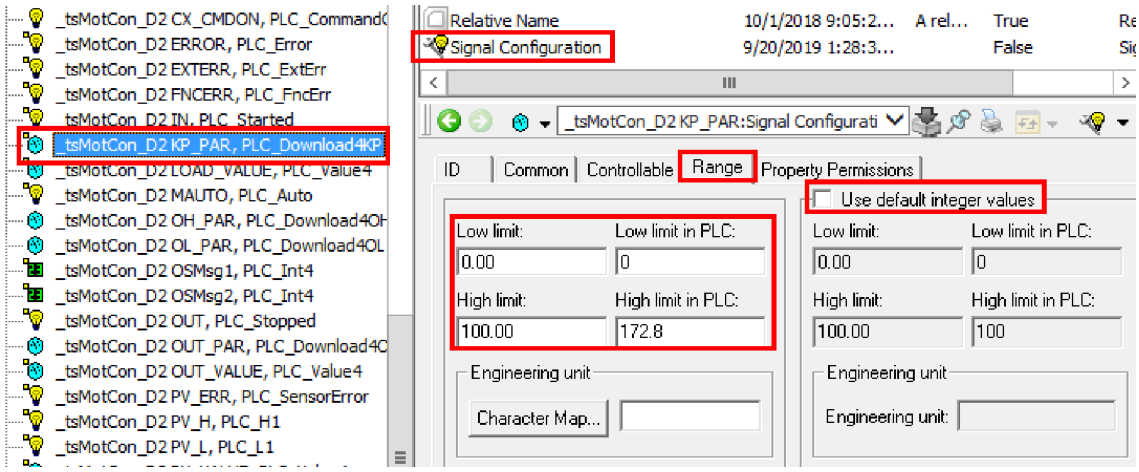
If the wanted range of SCADA.KP_PAR in SCADA is chosen to be 100 %, then the range in SCADA control structure should be adapted to

SCADA.Signal Configuration.Range.Low limit = 0

SCADA.Signal Configuration.Range.High limit =100

SCADA.Signal Configuration.Range.Low limit in PLC = 0

SCADA.Signal Configuration.Range.High limit in PLC = [27648 - 0] / [16000 - 0] × 100 = 172.8



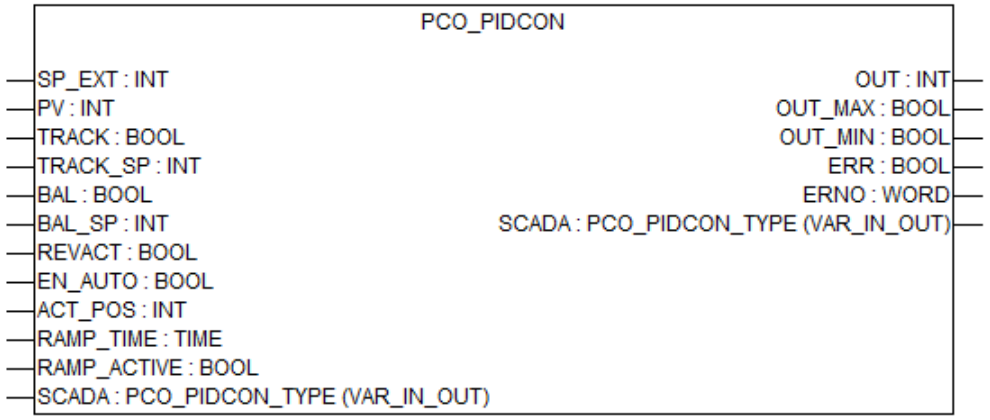
Scaling of the TRACK_SP AND BAL_SP

Input range of TRACK_SP or BAL_SP is 0 ... 10000 and the controller OUT range is 0 ... 27648.

So if the TRACK/BAL_SP is x then the controller output is defined as:

$$OUT = \frac{x}{10000} \times 27648$$

Input description



SP_EXT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

External setpoint.

Calculated setpoint or output from another PID controller (e.g. PCO_PIDCON) in a cascade control system.

PV

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Process value (PV) to be controlled.

TRACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Tracking mode.

PCO_PIDCON can be set in Tracking mode if the value of TRACK is TRUE.

Output of the controller will be set to TRACK_SP.

Tracking mode has higher priority compared to balancing mode.

TRACK is independent of operation mode of PCO_PIDCON.

TRACK_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Tracking setpoint.

Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.

BAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Balancing mode.

PCO_PIDCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.

Output of the controller will be set to BAL_SP.

BAL is independent of operation mode of PCO_PIDCON.

BAL_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Balancing setpoint.

Output of the controller will be set to BAL_SP if value of BAL is TRUE.

REVACT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Reverse Action

Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.

Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

EN_AUTO

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable automatic mode of the PCO_PIDCON.

PCO_PIDCON can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_PIDCON is forced in manual mode if value of EN_AUTO is FALSE.

ACT_POS

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Feedback speed of motor.

RAMP_TIME

Data type	Default value	Range	Unit
TIME	TIME#30s	-	-

Actuator ramp time from 0 % ... 100 %

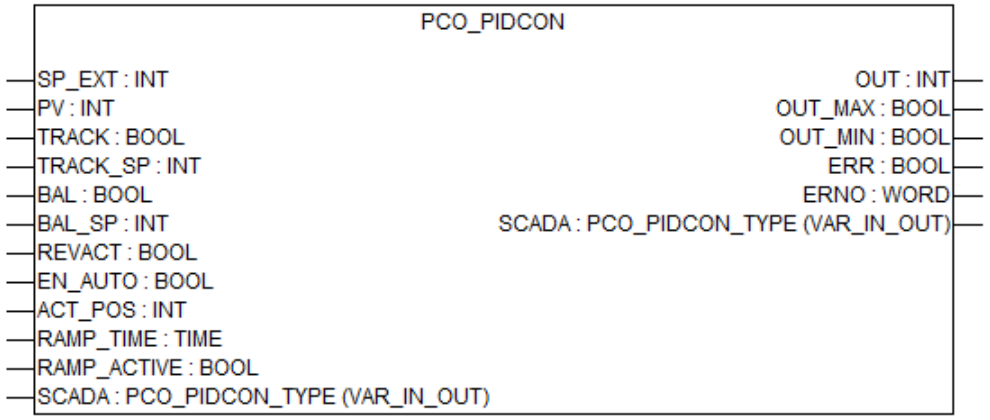
RAMP_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.

The value RAMP_ACTIVE is FALSE activates no ramp.

Output description



OUT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Controller output to the motor to attain the desired speed of the motor.

OUT_MAX

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output max limit reached, controller at max limit.

OUT_MIN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output min limit reached, controller at min limit.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, Input Parameter error.

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

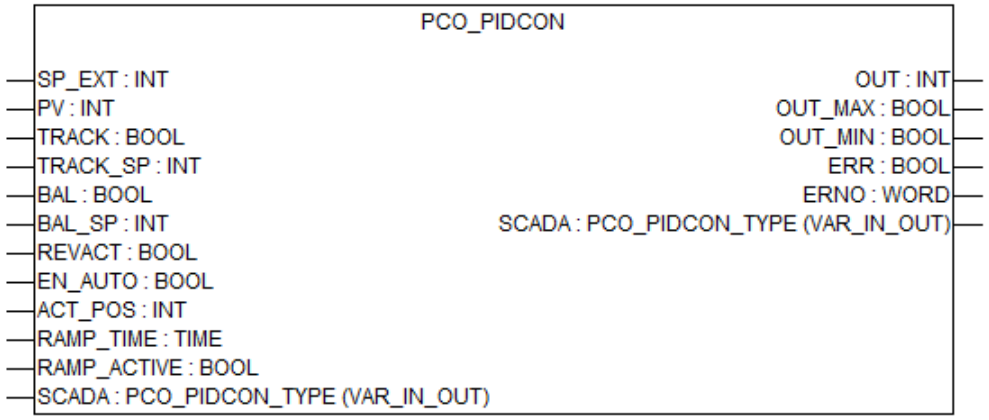
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *“Standard Function Block Libraries AC500”* in *“Error Messages of the Function Block Libraries”*.

Input/output description



SCADA

Data type	Default value	Range	Unit
STRUCTURE	-	-	-

Structure variable for communication between AC500 and SCADA system.

To retain the variable value in case of power ON/OFF or download, the variable connected to the SCADA In/Output should be declared as (global) retain persistant (or use %R area) variable in the program.

Detailed information on the scaling see *"Scaling" on page 3102*.

Parameter	Data type	Description	In-/Output
SCADA.CMDUP5	BOOL	Manual command to increase output by 5 % part of the PCO_PIDCON	Input
SCADA.CMDUP1	BOOL	Manual command to increase output by 1 % part of the PCO_PIDCON	Input
SCADA.CMDDW1	BOOL	Manual command to decrease output by 1 % part of the PCO_PIDCON	Input
SCADA.CMDDW5	BOOL	Manual command to decrease output by 5 % part of the PCO_PIDCON	Input
SCADA.CM_CMDON	BOOL	Manual command for the controller	Input
SCADA.CI_CMDON	BOOL	Auto command to set the internal setpoint for the controller	Input
SCADA.CX_CMDON	BOOL	Auto command to set the external setpoint for the controller	Input
SCADA.CAUTO	BOOL	Controller of the PCO_PIDCON is in automatic mode	Output
SCADA.SPIACT	BOOL	Internal setpoint of the PCO_PIDCON is active	Output
SCADA.SPXACT	BOOL	External setpoint of the PCO_PIDCON is active	Output
SCADA.CTACT	BOOL	Tracking setpoint of the PCO_PIDCON is active	Output
SCADA.PV_H	BOOL	Process value at high limit	Output

Parameter	Data type	Description	In-/Output
SCADA.PV_L	BOOL	Process value at low limit	Output
SCADA.PV_ERR	BOOL	Process value error	Output
SCADA.OUT_PAR	INT	Manual position setpoint Scale 0 ... 10000	Input
SCADA.SPI_PAR	INT	Internal setpoint value	Input
SCADA.KP_PAR	INT	Coefficient of proportionality (gain) % of the controller	Input
SCADA.TD_PAR	INT	Time constant for D-part of the controller	Input
SCADA.TI_PAR	INT	Time constant for integration of the controller	Input
SCADA.OH_PAR	INT	Output high limit (0 ... 10000) of the controller	Input
SCADA.OL_PAR	INT	Output low limit (0 ... 10000) of the controller	Input
SCADA.ACT_VALUE	INT	Actual speed of the motor, scaling as input ACT_POS (0 ... 10000)	Output
SCADA.OUT_VALUE	INT	Output from the controller Scale 0 ... 10000	Output
SCADA.PV_VALUE	INT	Process value to be controlled	Output
SCADA.SPX_VALUE	INT	External setpoint value	Output
SCADA.OSMsg1 *)	WORD	Word representing the status of the PCO_PIDCON	Output
AspectObjectType _Pidcon_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

*) structure described separately

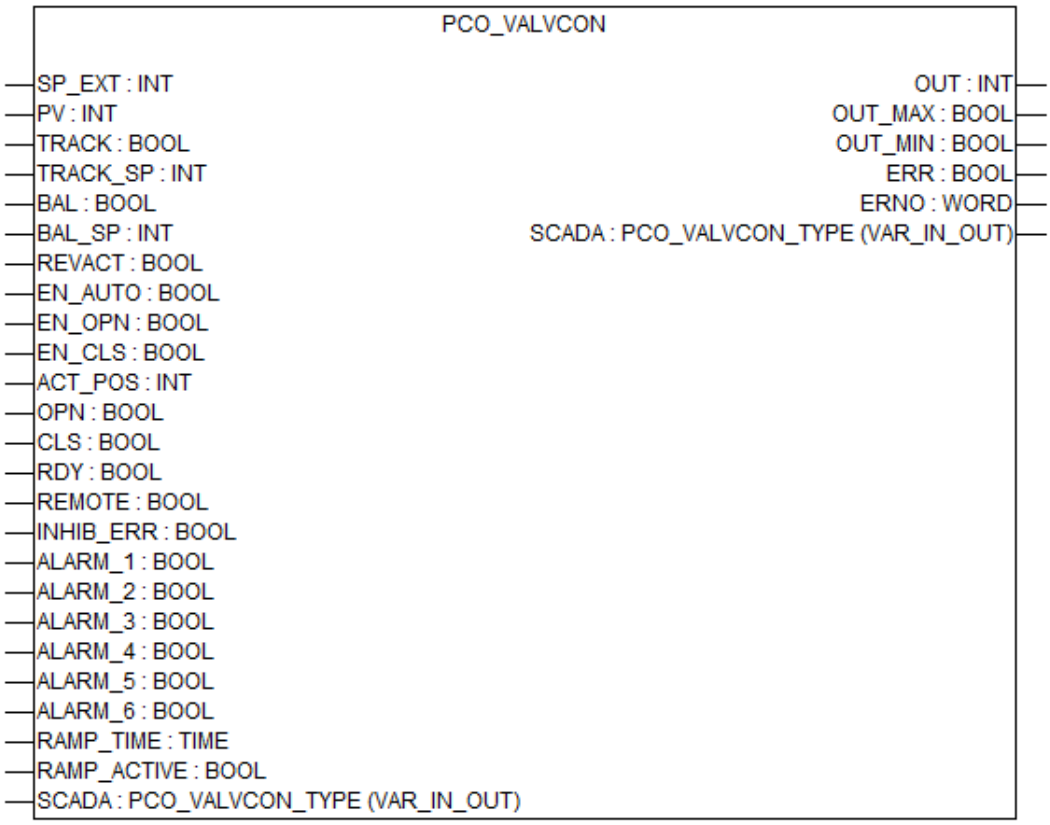
SCADA.OSMsg1

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_PIDCON.

Bit	Description
Bit 0	Not used
Bit 1	Not used
Bit 2	Internal setpoint (PID controller)
Bit 3	External setpoint (PID controller)
Bit 4	Not used
Bit 5	Not used
Bit 6	Tracking setpoint (PID controller)
Bit 7	PID controller in manual mode
Bit 8	Not used
Bit 9	Not used
Bit 10	Not used
Bit 11	PID controller in automatic mode
Bit 12	Not used
Bit 13	PID controller not released for automatic mode
Bit 14	Not used
Bit 15	Not used

PCO_VALVCON



This function block is designed for controlling a variable position valve.

The function block is similar to PCO_VALV (open/close valve), but in addition to PCO_VALV the function block has a built in PID controller, used for position control of the valve.

The PID controller can be switched between three different setpoints:

- **Internal Setpoint**
  Set from SCADA faceplate, variable in SCADA side is SPI_PAR.
- **External Setpoint**
  Function block input SP_EXT, calculated setpoint or output from another PID controller (e.g. PIDCON) in a cascade control system.
- **Tracking Setpoint**
  The output of the PID controller follows the function block input tracking setpoint, TRACK_SP and if TRACK input is active.

To make the controller follows various setpoints, the following variables need to be set:

> *The values set below are default values, the user can change based on the application.*

Settings in SCADA side	Data type	When the controller must follow		
		Internal Setpoint	External Setpoint	Tracking Setpoint
SCADA.KP_PAR	INT	1	1	1
SCADA.TD_PAR	INT	0	0	0
SCADA.TI_PAR	INT	1	1	1
SCADA.OH_PAR	INT	10000	10000	10000
SCADA.OL_PAR	INT	0	0	0
SCADA.CI_CMDON	BOOL	TRUE	-	-
SCADA.SPI_PAR	INT	Needs to be set	-	-
SCADA.CX_CMDON	BOOL	-	TRUE	-

Settings in FB side	Data type	When the controller must follow		
		Internal Set-point	External Set-point	Tracking Set-point
EN_AUTO	BOOL	TRUE	TRUE	TRUE
SP_EXT	INT	-	Needs to be set	-
TRACK	BOOL	-	-	TRUE
TRACK_SP	INT	-	-	Needs to be set

By switching to one of these setpoints, when command from SCADA, i.e. SCADA.SPIACT or SCADA.SPXACT or SCADA.CTACT is active, the PID controller will be put into automatic mode.

It is possible to set the PID controller into manual mode from the SCADA system to allow manual positioning.

The manual positioning can be done in two ways, both controlled form the SCADA faceplate.

1. Stepwise 1 % up / down or 5 % up / down.

   This is manually set by the following variables on the SCADA faceplate:

   - SCADA.CMDUP1
   - SCADA.CMDUP5
   - SCADA.CMDDW1
   - SCADA.CMDDW5

   The controller output varies based on the command given.

2. Download a new position. Manual position is set through OUT_PAR variable on the SCADA faceplate. The controller output varies accordingly.

If BAL (Input) of the function block is TRUE, then controller output varies according to BAL_SP entered at the input of the function block. This has higher priority over manual commands or external or internal setpoints.

But if TRACK input is selected then the controller follows the TRACK_SP.

When the controller is set to TRACK mode, the controller gets set to auto mode, whereas when the controller is set to BAL mode, the controller is not in auto mode. In both cases the controller output is not dependent on the actual process value.

The function block has built in a ramp to slow down the manual positioning. The ramp can be switched ON by the input RAMP_ACTIVE by setting the RAMP_ACTIVE high. The ramp time can be entered at RAMP_TIME input of the function block.

Cycle time for the program must be faster than RAMP_TIME / 100 to calculate the actuator time correctly!

6 different alarms can be connected to the function block
(ALARM_1 → ALARM_6, Torque switch etc.).

The valve can be switched to automatic or manual mode from the SCADA faceplate, if the input EN_AUTO = TRUE.

The position of the valve is controlled by the PID controller based on the process values.

If EN_Auto = FALSE, then the function block is forced to manual mode.

The position of the valve is controlled by the OUT_PAR value or by the commands CMDDW1, CMDDW5, CMDUP5, CMDUP1.

In automatic mode the valve is position controlled using the SP_EXT input as the position reference.

In manual mode the valve is position controlled using a setpoint from the SCADA interface, OUT_PAR.

Using the function block inputs EN_OPN and EN_CLS it is possible to prevent the valve from opening or closing by setting the respective input = FALSE. The inputs OPN and CLS are feedback indicators of the valve open and valve closed position. This is only used as display information for the SCADA faceplate.

When the controller is in automatic mode (following its internal, external or track setpoint) and then either the EN_AUTO input or the READY input are reset to FALSE, the controller is forced to manual mode. In this case a functional error is generated for three seconds. The three seconds are long enough to ensure that the functional error is logged in the 800xA system.

This function block can be used in combination with the object type VALVCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Examples

The function block is used for controlling variable position valve which in turn controls the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level measurement
- Valve
- Main power supply, circuit breaker (or similar)

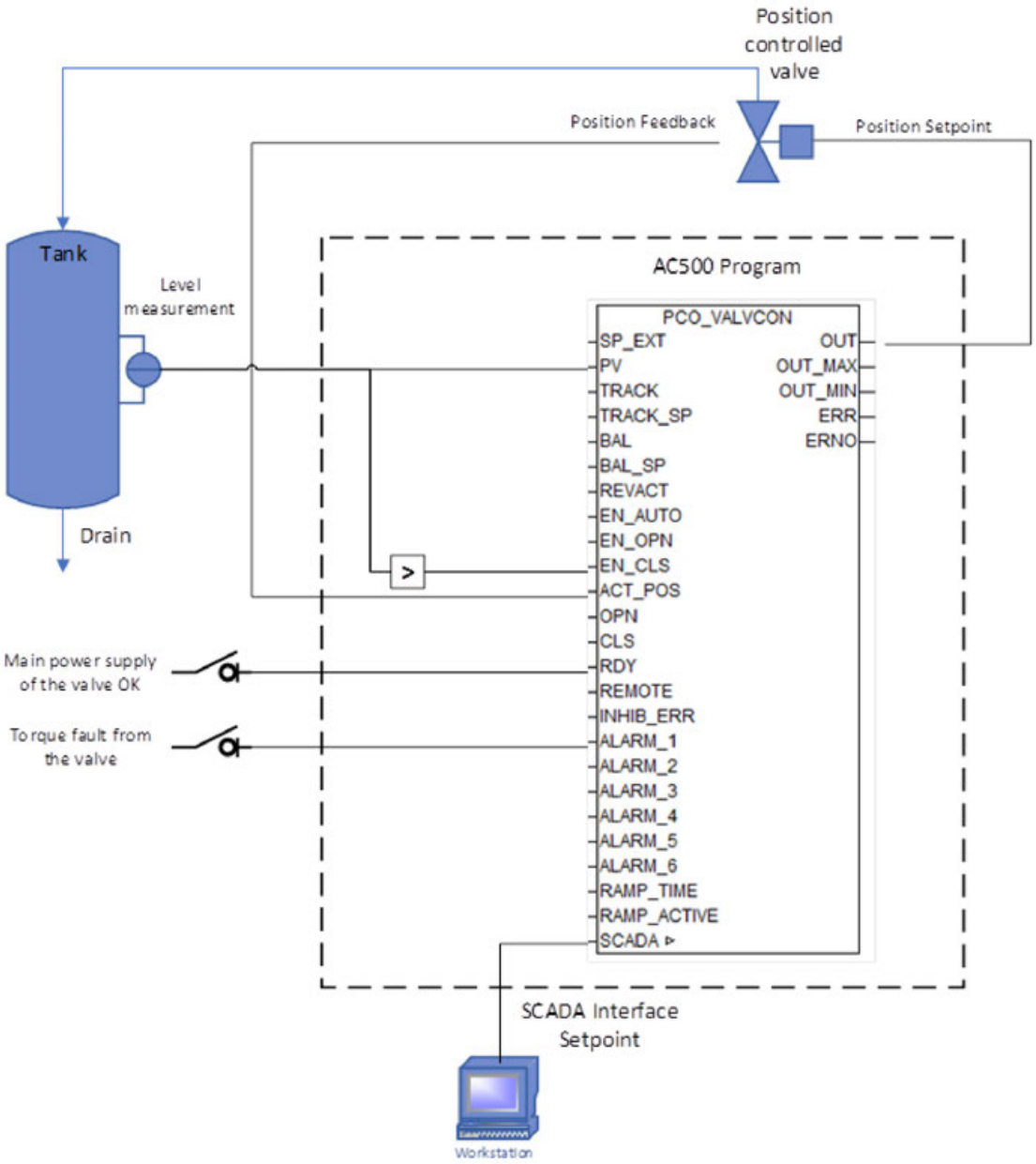
Manual mode:

- Valve controlled by the operator.

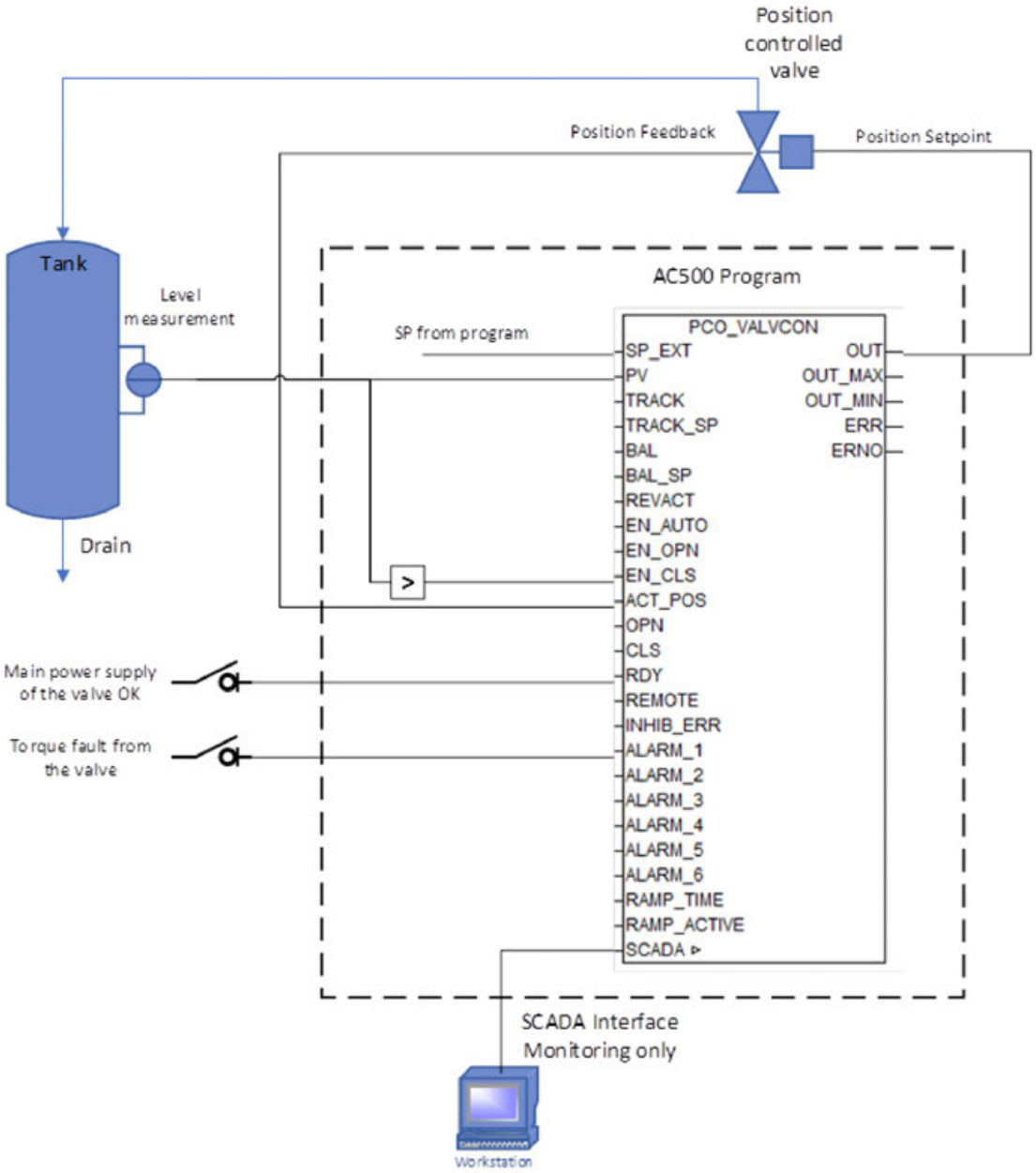
Automatic mode:

- Valve controlled by the AC500 controller.

PCO_VALVCON manual mode example



PCO_VALVCON automatic mode example



Scaling

Scaling of the SCADA Parameters

In the 800xA system the values coming from the PLC are shown by default 1:1 in the operator screens and faceplates.

The default scaling is 0 ... 100 from PLC is shown as 0 % … 100 % or 0 % ... 100.00 % for the operator (SCADA). Therefore a scaling must be done for most signals either in the *"Bulk Data Manager"* or directly in the *Engineering Workplace* > *Control Structure* > … > *Signal Configuration* > *Range*.

Assuming that all signals on the operator workplace should be shown in 0 % … 100 % the following table gives an overview of the scaling of the PLC values in the 800xA system.

PVmin and PVmax are here the minimum and maximum of the PV input, reflecting the process value in the PLC program. The PV typically comes from the output of the function block PCO_ANAALM. This scales its IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN … SCALE_MAX).

OPC	FB	AC500		800xA		Remark
Signal name SCADA.x	**In/Output-name**	**Range default**	**Range actual program**	**PLC Range def.: 0…100**	**SCADA Range def: 0…100% or 0…100.00%**	
PV_VALUE	PV	0…27648	PVmin…PVmax PVmin … PVmax	PVmin…PVmax	0…100	
SPX_VALUE	SP_EXT	0…27648	PVmin…PVmax	PVmin…PVmax	0…100	
	TRACK_SP	0…10000	0…10000 fix			
	BAL_SP	0…10000	0…10000 fix			
ACT_VALUE	ACT_POS	0…10000	ACTmin… ACTmax	ACTmin…ACTmax	0…100	
OUT_PAR	-	0…10000	0…10000 fix	0…10000	0…100	
OL_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
OH_PAR	-	0…10000	0…10000 fix	0…10000	0…100	Set from 800xA
	OUT	0…27648		-	-	
OUT_VALUE	-	0…10000		0…10000	0…100	Scaled from OUT. Seen in 800xA
SPI_PAR	-	0…27648	-	PVmin…PVmax	0…100.00	Set from 800xA
KP_PAR	-	0…100	0…(27648 / (PVmax- PVmin))	PVmin…PVmax or PVmin/100…PVmax/100	0…100.00 (%) or 0..1.00 (as factor)	
TD_PAR	-			0...1000	0...1000.00	
TI_PAR	-			0...1000	0...1000.00	

Scaling of the SCADA.PV_VALUE

The output from the controller is based on 0 % ... 100 % → 0 ... 27648 for connection direct to an analog output of the AC500 I/Os.

To show the PV in % in 800xA, it must be scaled accordingly in the 800xA signal configuration:

SCADA.Signal_Configuration.Range.Low limit = 0

SCADA.Signal_Configuration.Range.High limit = 100

SCADA.Signal_Configuration.Range.Low limit in PLC = 0

SCADA.Signal_Configuration.Range.High limit in PLC = 27648

Scaling of the Setpoints, SCADA.SPI_PAR and SCADA.SPX_VALUE

SCADA.SPI_PAR and SCADA.SPX_VALUE must have the same scaling as the process value PV.

Scaling of the Parameter SCADA.KP_PAR

If the setpoint and the process value (PV) do not have equal scaling (0 ... 27648), there is a need for scaling the SCADA.KP_PAR parameter in 800xA to achieve the right proportional gain.

The scaling of the coefficient of proportionality (SCADA.KP_PAR) is dependent on the scaling of the process value (PV).

The PV typically comes from the output of the function block PCO_ANAALM. This scales the IN, coming directly from the analog input, to the OUT according to the inputs SCALE_MIN and SCALE_MAX.

The process value, e.g. 0 bar ... 16 bar can so be rescaled to the value 0 ... 16000 (SCALE_MIN ... SCALE_MAX).

The scaling of the KP [%] is then calculated by the formula:

$$KP[\%] = \frac{Out_{\max} - Out_{\min}}{PV_{\max} - PV_{\min}} \times 100 = \frac{27648 - 0}{SCALE_{\max} - SCALE_{\min}} \times 100$$

In the mentioned example, 100 % of KP.

$$100\ \%\ of\ KP = \frac{[27648 - 0]}{[16000 - 0]} \times 100 = 172.8$$

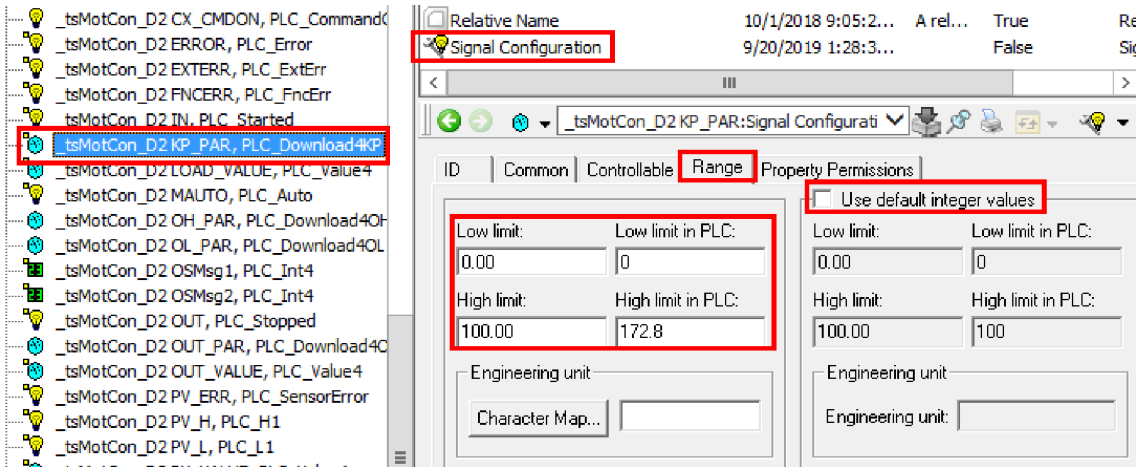
If the wanted range of SCADA.KP_PAR in SCADA is chosen to be 100 %, then the range in SCADA control structure should be adapted to

SCADA.Signal Configuration.Range.Low limit = 0

SCADA.Signal Configuration.Range.High limit =100

SCADA.Signal Configuration.Range.Low limit in PLC = 0

SCADA.Signal Configuration.Range.High limit in PLC = [27648 - 0] / [16000 - 0] × 100 = 172.8



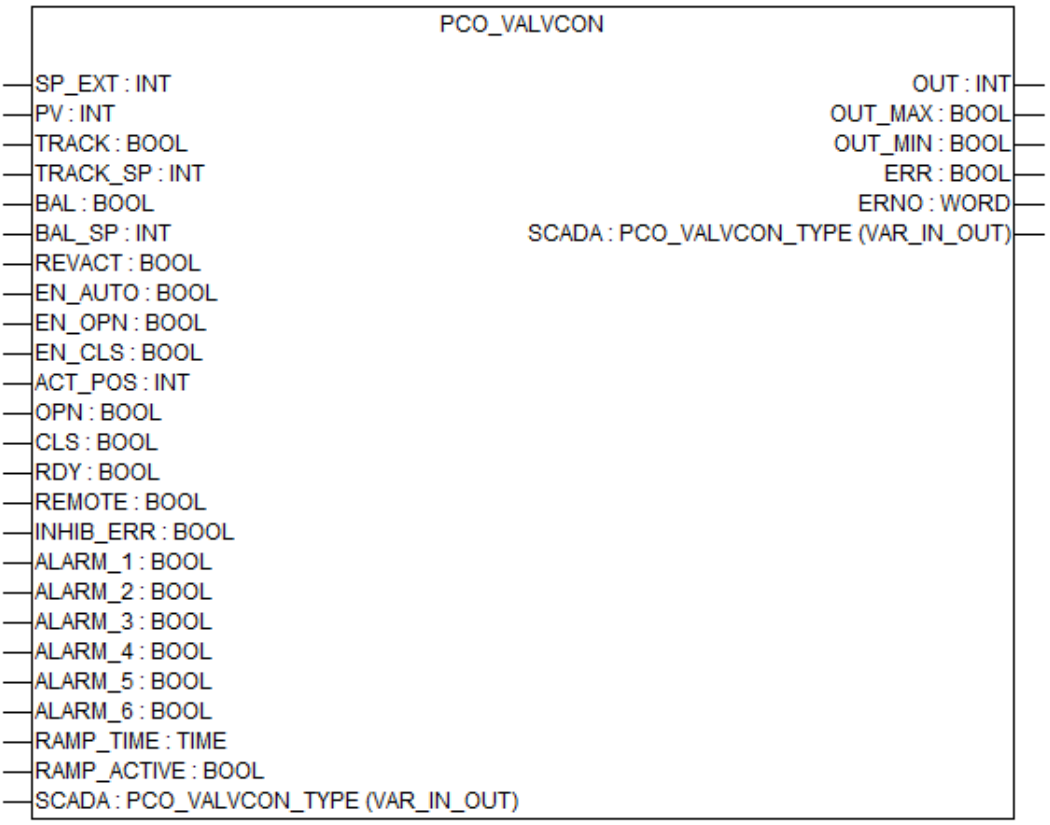
Scaling of the TRACK_SP AND BAL_SP

Input range of TRACK_SP or BAL_SP is 0 ... 10000 and the controller OUT range is 0 ... 27648.

So if the TRACK/BAL_SP is x then the controller output is defined as:

$$OUT = \frac{x}{10000} \times 27648$$

Input description



SP_EXT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

External setpoint.

Calculated setpoint or output from another PID controller (e.g. PCO_VALVCON) in a cascade control system.

PV

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Process value (PV) to be controlled.

TRACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Tracking mode.

PCO_VALVCON can be set in Tracking mode if the value of TRACK is TRUE.

Output of the controller will be set to TRACK_SP.

Tracking mode has higher priority compared to balancing mode.

TRACK is independent of operation mode of PCO_VALVCON.

TRACK_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Tracking setpoint.

Output of the controller will be set to TRACK_SP if value of TRACK is TRUE.

BAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Balancing mode.

PCO_VALVCON can be set in Balancing mode if value of BAL is TRUE and value of TRACK is FALSE.

Output of the controller will be set to BAL_SP.

BAL is independent of operation mode of PCO_VALVCON.

BAL_SP

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Balancing setpoint.

Output of the controller will be set to BAL_SP if value of BAL is TRUE.

REVACT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Reverse Action

Value of REVACT is FALSE, Controller will increase output if Setpoint is less than Process value.

Value of REVACT is TRUE, Controller will decrease output if Setpoint is greater than Process value.

EN_AUTO

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable automatic mode of the PCO_VALVCON.

PCO_VALVCON can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_VALVCON is forced in manual mode if value of EN_AUTO is FALSE.

EN_OPN

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable valve to open.

Valve can be opened if value of EN_OPN is TRUE.

Valve cannot be opened if value of EN_OPN is FALSE.

EN_CLS

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable valve to close.

Valve can be closed if value of EN_CLS is TRUE.

Valve cannot be closed if value of EN_CLS is FALSE.

ACT_POS

Data type	Default value	Range	Unit
INT	0	0 ... 10000	-

Position feedback of valve.

OPN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the valve.

If the value of OPN is TRUE, the valve is open.

CLS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the valve.

If the value of CLS is TRUE, the valve is closed.

RDY

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Valve is ready for operation.

Valve is ready for operation if value of RDY is TRUE.

Value of RDY is FALSE results in an external error.

REMOTE

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Allow valve to be controlled from SCADA.

Value of REMOTE is TRUE control from SCADA and function block is enabled.

Value of REMOTE is FALSE valve is controlled only from function block.

INHIB_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of alarms.

If value of INHIB_ERR is TRUE, all alarms from PCO_VALVCON are suppressed.

Alarm_1

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 1.

If value of ALARM_1 is TRUE Alarm_1 is active.

Independent of operation mode of PCO_VALVCON.

Alarm_2

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 2.

If value of ALARM_2 is TRUE Alarm_2 is active.

Independent of operation mode of PCO_VALVCON.

Alarm_3

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 3.

If value of ALARM_3 is TRUE Alarm_3 is active.

Independent of operation mode of PCO_VALVCON.

Alarm_4

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_VALVCON.

Alarm_5

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_VALVCON.

Alarm_6

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_VALVCON.

RAMP_TIME

Data type	Default value	Range	Unit
TIME	TIME#30s	-	-

Actuator ramp time from 0 % ... 100 %

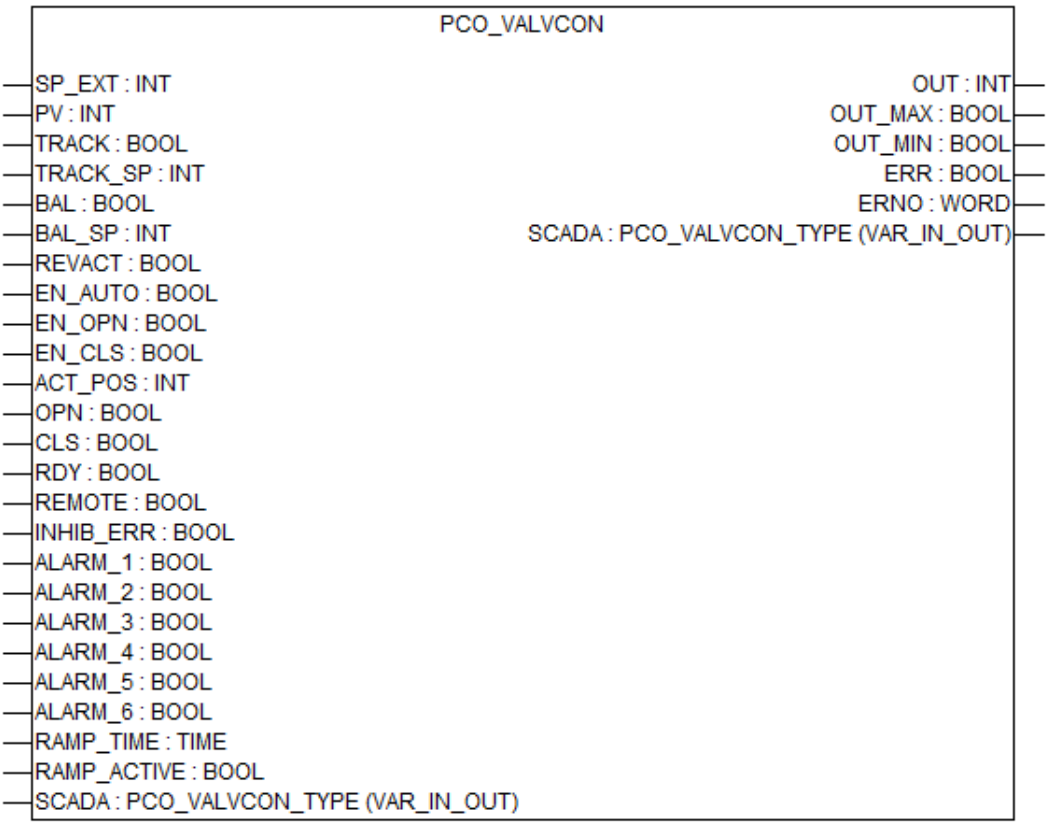
RAMP_ACTIVE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The value RAMP_ACTIVE is TRUE activates ramp under manual positioning.

The value RAMP_ACTIVE is FALSE activates no ramp.

Output description



OUT

Data type	Default value	Range	Unit
INT	0	0 ... 27648	-

Controller output to the valve to attain the desired position of the valve.

OUT_MAX

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output max limit reached, controller at max limit.

OUT_MIN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Output min limit reached, controller at min limit.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, including:

- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

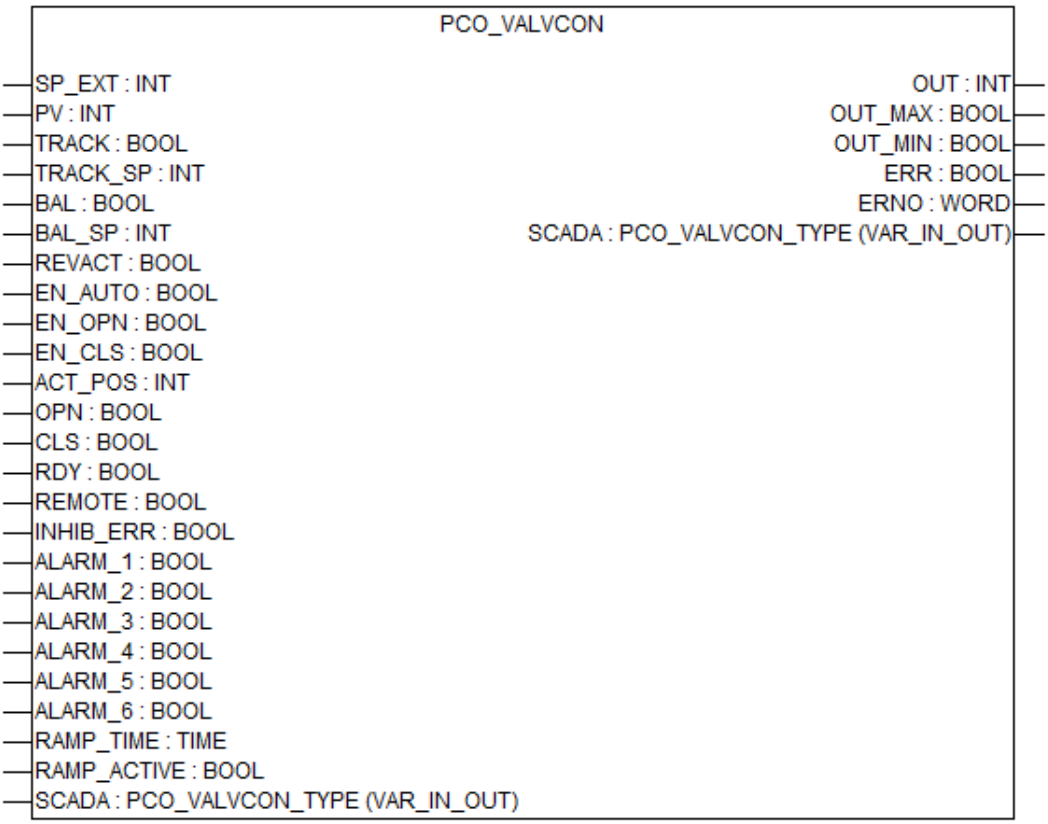
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *“Standard Function Block Libraries AC500”* in *“Error Messages of the Function Block Libraries”*.

Input/output description



SCADA

Data type	Default value	Range	Unit
PCO_VALVCON_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

To retain the variable value in case of power ON/OFF or download, the variable connected to the SCADA In/Output should be declared as (global) retain persistant (or use %R area) variable in the program.

Detailed information on the scaling see *"Scaling" on page 3102*.

Parameter	Data type	Description	In-/Output
SCADA.OPEN	BOOL	Valve is open	Output
SCADA.CLOSED	BOOL	Valve is closed	Output
SCADA.REMOTE	BOOL	Valve can be controlled from OS REMOTE = FALSE → Local operation	Output
SCADA.READY	BOOL	Valve is ready for operation	Output
SCADA.FNCERR	BOOL	Functional error	Output
SCADA.EXTERR	BOOL	External error (is generated when the valve is not ready)	Output
SCADA.AL1	BOOL	Auxiliary alarm no. 1	Output

Parameter	Data type	Description	In-/Output
SCADA.AL2	BOOL	Auxiliary alarm no. 2	Output
SCADA.AL3	BOOL	Auxiliary alarm no. 3	Output
SCADA.AL4	BOOL	Auxiliary alarm no. 4	Output
SCADA.AL5	BOOL	Auxiliary alarm no. 5	Output
SCADA.AL6	BOOL	Auxiliary alarm no. 6	Output
SCADA.ACT_VALUE	INT	Actual position of the valve, scaling as input ACT_POS (0 ... 10000)	Output
SCADA.CM_CMDON	BOOL	Manual command for the controller	Input
SCADA.CI_CMDON	BOOL	Auto command to set the internal setpoint for the controller	Input
SCADA.CX_CMDON	BOOL	Auto command to set the external setpoint for the controller	Input
SCADA.CMDUP5	BOOL	Manual command to increase output by 5 %	Input
SCADA.CMDUP1	BOOL	Manual command to increase output by 1 %	Input
SCADA.CMDDW1	BOOL	Manual command to decrease output by 1 %	Input
SCADA.CMDDW5	BOOL	Manual command to decrease output by 5 %	Input
SCADA.CAUTO	BOOL	Controller is in automatic mode	Output
SCADA.SPIACT	BOOL	Internal setpoint is active	Output
SCADA.SPXACT	BOOL	External setpoint is active	Output
SCADA.CTACT	BOOL	Tracking setpoint is active	Output
SCADA.PV_H	BOOL	Process value at high limit	Output
SCADA.PV_L	BOOL	Process value at low limit	Output
SCADA.PV_ERR	BOOL	Process value error	Output
SCADA.OUT_PAR	INT	Manual position setpoint Scale 0 ... 10000	Input
SCADA.SPI_PAR	INT	Internal setpoint value	Input
SCADA.KP_PAR	INT	Coefficient of proportionality (gain) % of the controller	Input
SCADA.TD_PAR	INT	Time constant for D-part of the controller	Input
SCADA.TI_PAR	INT	Time constant for integration of the controller	Input
SCADA.OH_PAR	INT	Output high limit (0 ... 10000) of the controller	Input
SCADA.OL_PAR	INT	Output low limit (0 ... 10000) of the controller	Input
SCADA.OUT_VALUE	INT	Output from the controller Scale 0 ... 10000	Output
SCADA.PV_VALUE	INT	Process value to be controlled	Output
SCADA.SPX_VALUE	INT	External setpoint value	Output
SCADA.OSMsg1 *)	WORD	Word representing the status of the PCO_VALVCON	Output

Parameter	Data type	Description	In-/Output
SCADA.OSMsg2 *)	WORD	Word representing the status of the PCO_VALVCON	Output
AspectObjectType _Valvcon_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

*) structure described separately

SCADA.OSMsg1

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_VALVCON.

Bit	Description
Bit 0	Valve is open
Bit 1	Valve is closed
Bit 2	Not used
Bit 3	Not used
Bit 4	External error (Not ready)
Bit 5	Functional error
Bit 6	Valve released for opening
Bit 7	Valve released for closing
Bit 8	Local operation (Not remote)
Bit 9	Common alarm (External alarm + Functional alarm + ALARM_1 ... ALARM_6)
Bit 10	Not used
Bit 11	Motor in automatic mode
Bit 12	Not used
Bit 13	Motor not released for automatic mode
Bit 14	Motor released for increase speed
Bit 15	Motor released for decrease speed

SCADA.OSMsg2

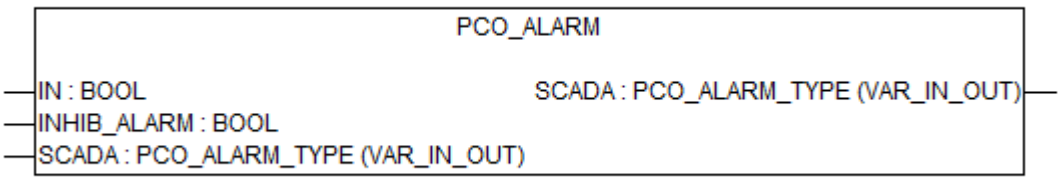
Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_VALVCON.

Bit	Description
Bit 0	Not used
Bit 1	Not used
Bit 2	Internal setpoint (PID controller)
Bit 3	External setpoint (PID controller)
Bit 4	Track setpoint (PID controller)
Bit 5	Auxiliary alarm 1
Bit 6	Auxiliary alarm 2
Bit 7	Auxiliary alarm 3
Bit 8	Auxiliary alarm 4
Bit 9	Auxiliary alarm 5
Bit 10	PID controller in automatic mode
Bit 11	PID controller in manual mode
Bit 12	Auxiliary alarm 6
Bit 13	Not used
Bit 14	Not used
Bit 15	Not used

Indications

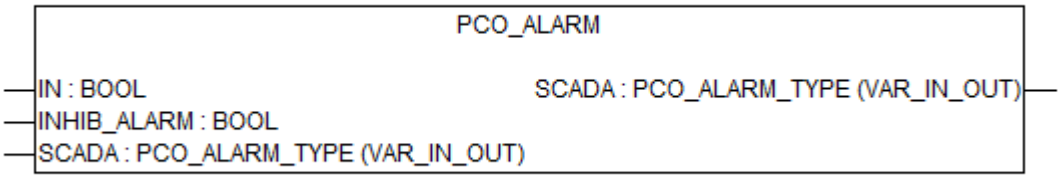
PCO_ALARM



The function block is used to send an alarm to the SCADA system.

The alarm sent to SCADA is extended minimum 3 seconds to ensure alarm detection in the SCADA system, even if the alarm at the input goes off before 3 seconds. The variable that represents the alarm in the SCADA side is "Status". This function block can be used in combination with the object type ALARM_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



IN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Alarm input.

If value of IN is TRUE, then an alarm is sent to SCADA.

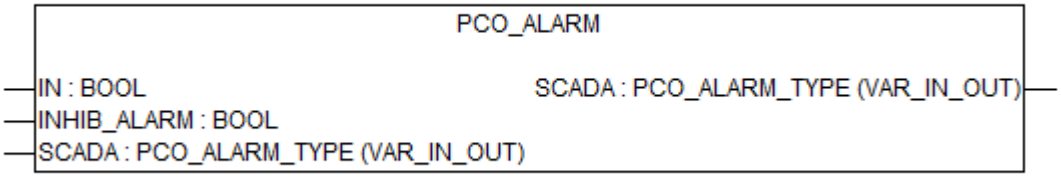
INHIB_ALARM

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Alarm inhibit.

If value of INHIB_ALARM is TRUE, then no alarm is sent to SCADA regardless of the value on IN.

Input/output description



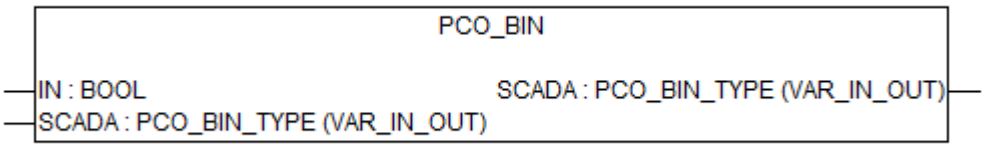
SCADA

Data type	Default value	Range	Unit
PCO_ALARM_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.STATUS	BOOL	Alarm status, TRUE = Alarm	Output
AspectObjectType _Alarm_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

PCO_BIN



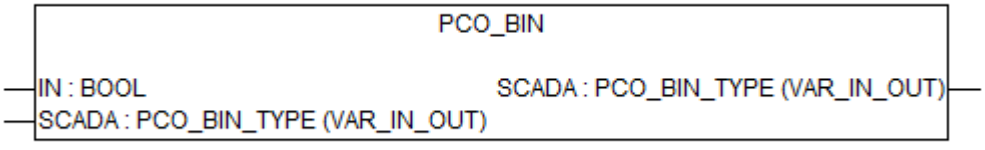
The function block is used to send an event or an indication to the SCADA system.

The event is entered at IN input of the function block and its presence is depicted as "Status" variable in SCADA.

The event sent to SCADA is extended minimum 3 seconds to ensure detection in the SCADA system.

This function block can be used in combination with the object type BIN_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



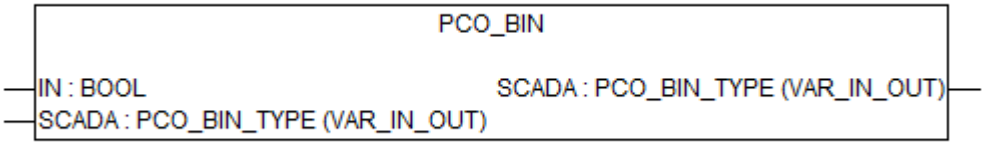
IN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Event input.

If value of IN is TRUE, then an event is sent to SCADA.

Input/output description



SCADA

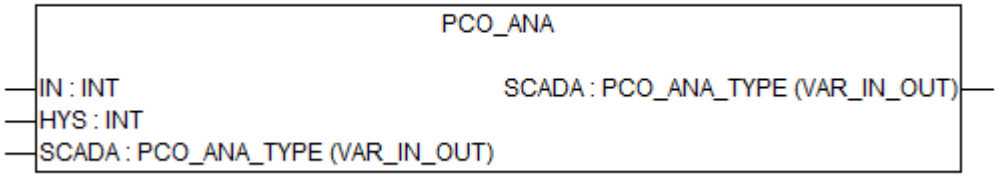
Data type	Default value	Range	Unit
PCO_BIN_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.STATUS	BOOL	Event status. If value is TRUE then event is depicted, else no event. The event is maintained for a minimum of 3 seconds, even if the event becomes FALSE before that.	Output
AspectObjectType_Bin_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

Measurements

PCO_ANA



The function block is used to send an analog value to the SCADA system with hysteresis limits.

The hysteresis limit can be entered from the FB. The value of the analog signal is depicted in SCADA through a variable named value. SCADA.VALUE is only updated, if IN input (analog input) has changed more than the HYS input limits defined.

For example:

Let the analog input be defined as 2, the SCADA.VALUE gets assigned as 2. Suppose the hysteresis is 3 then the range of up to which the input can vary is (2-3) and (2+3) which is -1 ... 5.

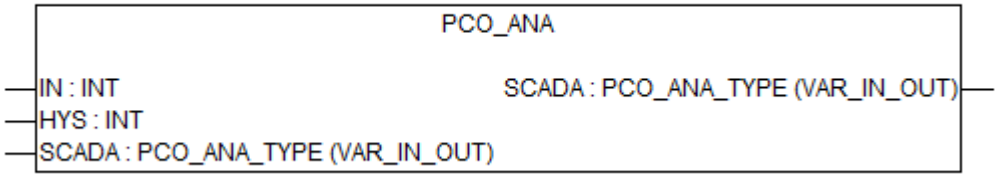
If an input -1 is given, then the SCADA.VALUE does not change (as it is in the range of -1 ... 5).

If -2 is given at the input, then the SCADA.VALUE gets set to -2 and then the input range becomes -5 ... 1.

There is no scaling in the function block.

This function block can be used in combination with the object type ANA_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



IN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

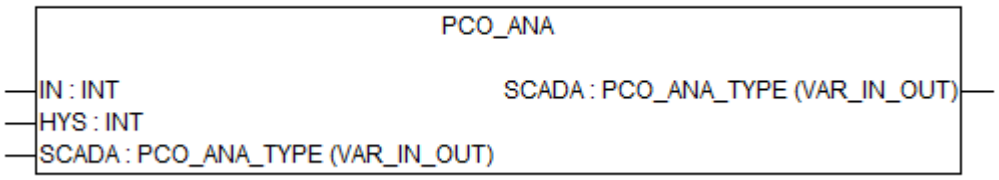
Analog input that must be sent to the SCADA system.

HYS

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Dead band limit for input.

Input/output description



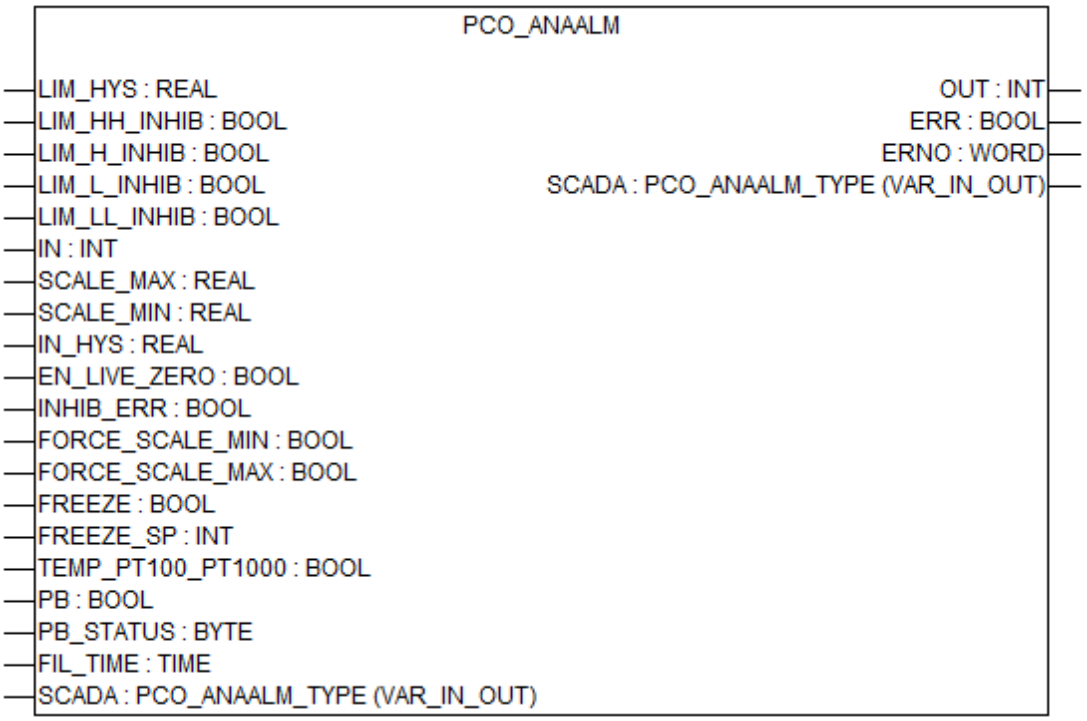
SCADA

Data type	Default value	Range	Unit
PCO_ANA_TYPE	-	-	-

Structure variable for communication to and from SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.VALUE	INT	Analog input within the hysteresis limits.	Output
AspectObjectType _Ana_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

PCO_ANAALM



The function block is designed for controlling an analog input with limit supervision.

The function block can be configured to handle the different ranges of the analog input, due to different electrical signals. E.g.

- 4 mA ... 20 mA
- 0 V ... 10 V or
- PT100 /PT1000

> *Please refer to AC500 hardware manual for detailed information regarding analog input modules and different parameters.*

This function block checks whether the analog output of the FB are within HH, H or LL , L limits.

The limits need to be entered from the SCADA side.

- SCADA.LIMH2
- SCADA.LIMH1
- SCADA.LIML1
- SCADA.LIML2

The function block rescales the analog input (IN) from the actual value at IN to SCALE_MIN → SCALE_MAX.

The IN range is from 0 ... 27648.The OUT is defined by:

$$OUT = \frac{IN}{27648} \times (SCALE_{\max} - SCALE_{\min}) + SCALE_{\min}$$

If TEMP_PT100_PT1000 or PB are to be set = TRUE, no rescaling of IN is made.

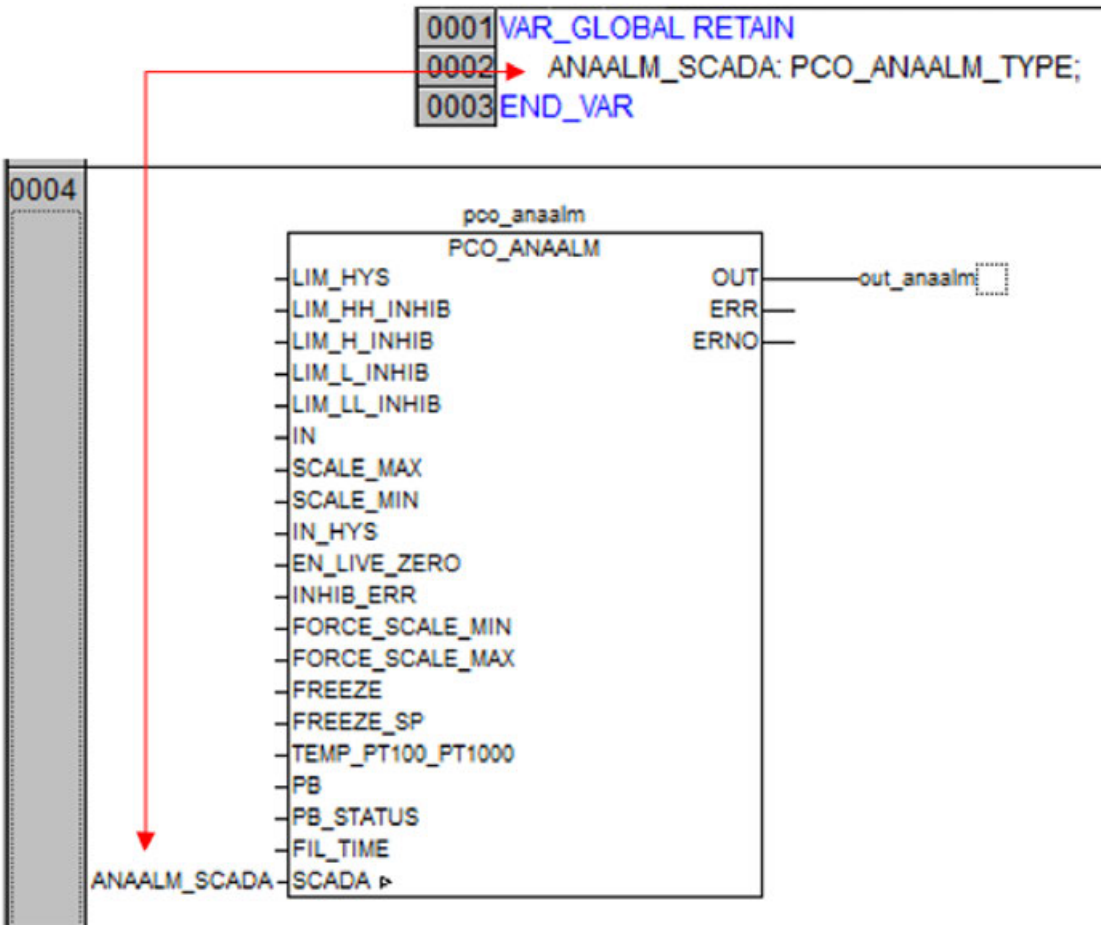
If an analog input is connected to an I/O module and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

Zero point suppression. The function block will force the input to zero when the input is between 0 and 0 - IN_HYS

The value can be preset to SCALE_MIN (FORCE_SCALE_MIN = TRUE) or SCALE_MAX (FORCE_SCALE_MAX = TRUE).

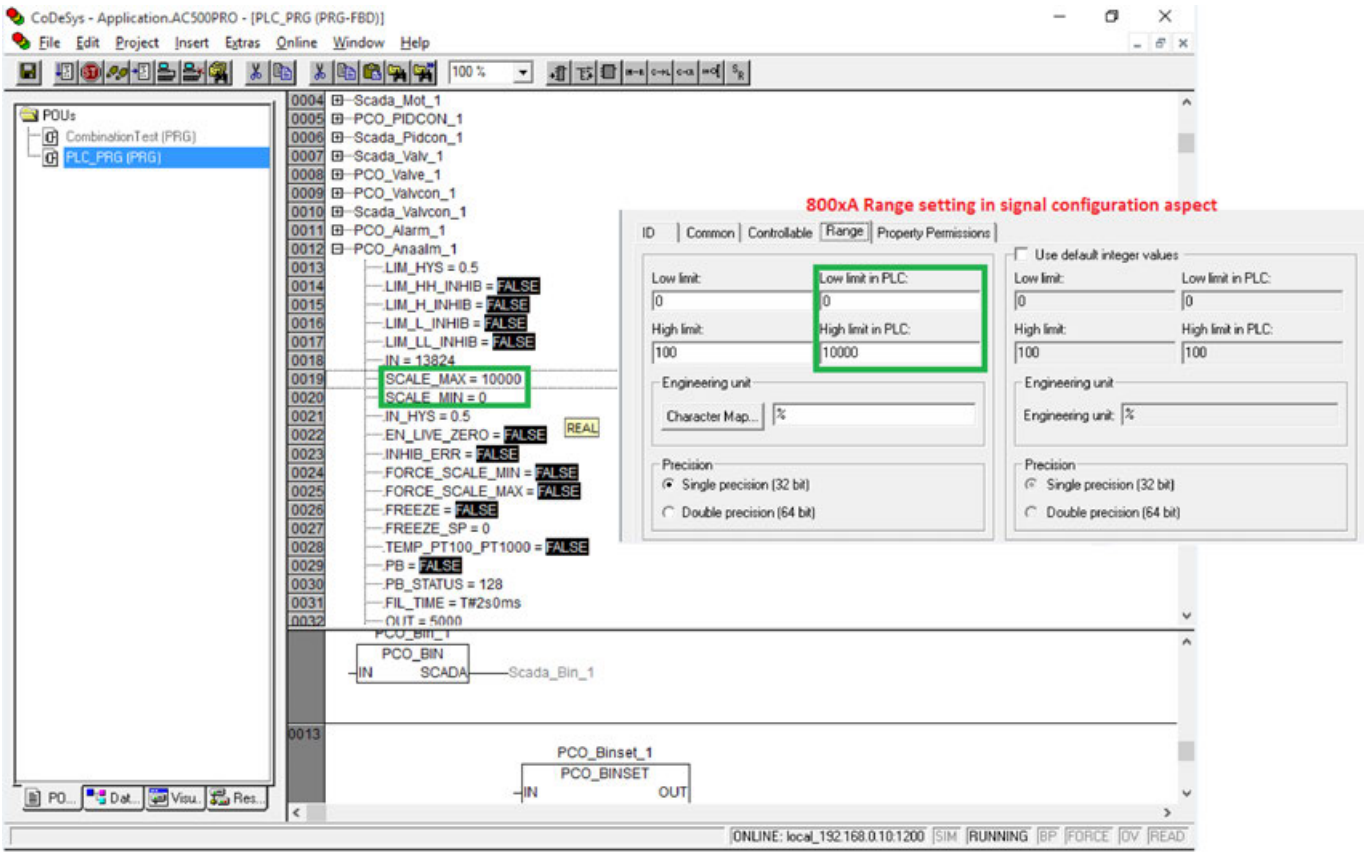
The output value from the function block can be frozen to a specific value (FREEZE = TRUE and the specific value on FREEZE_SP).

To retain the variable value in case of power ON/OFF or download, the variable connected to the SCADA In/Output should be declared as global retain persistant (or use %R area) variable in the program.

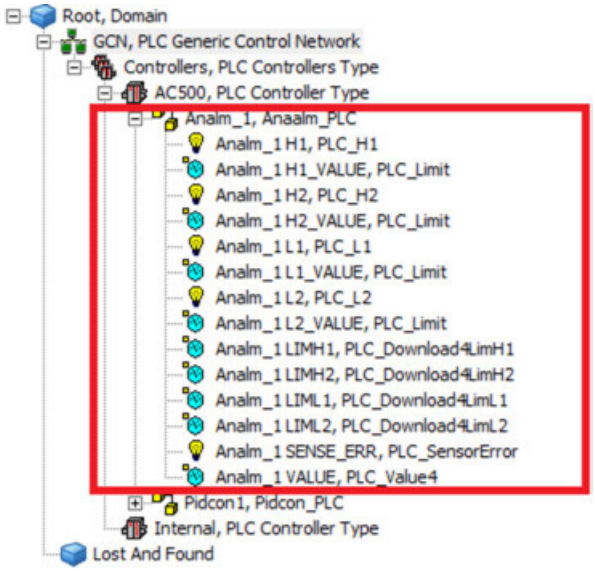


For PCO_ANAALM function block, the scaling needs to be taken care in the 800xA side. Whatever value for SCALE_MAX and SCALE_MIN is entered in the function block the similar setting needs to be done in the 800xA side scaling settings.

It is as shown in the figures below:



The same setting needs to be done for the other variables present in the ANAALM object tree.



The limit supervision consists of up to two high and two low alarms.

A SENSOR ERROR is generated in the SCADA side in the following conditions:

1. When the analog input is not within the tolerance of 1000, hence the input can vary over a range of -1000 ... 28648, after which it generates an error.

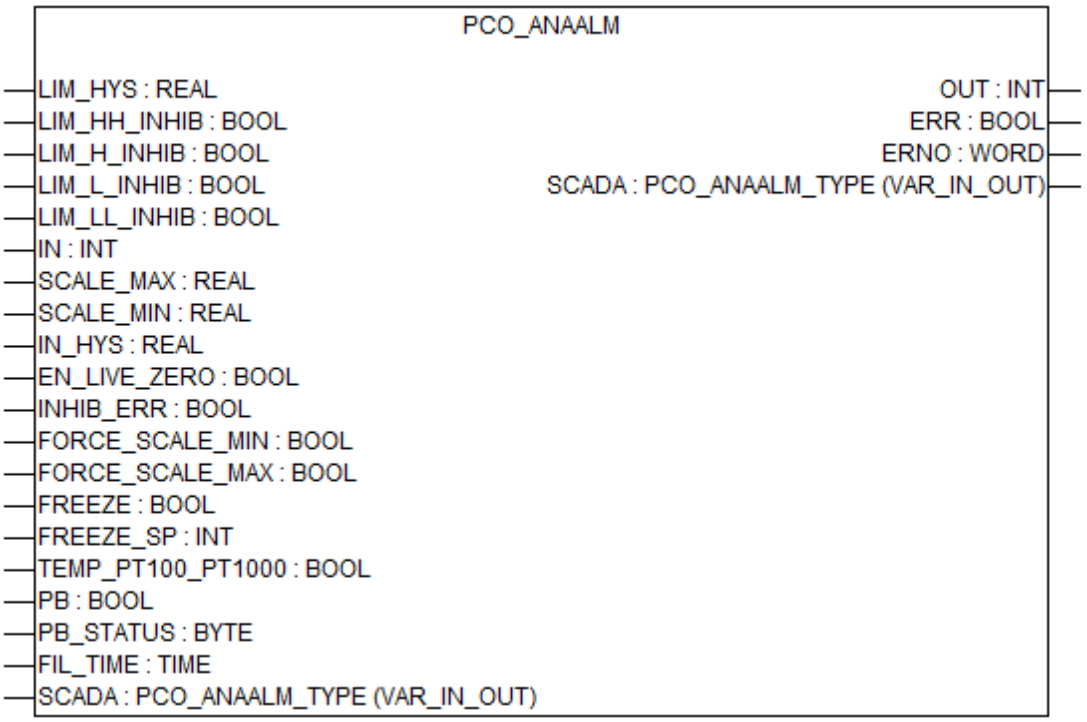
2. The output obtained at the end is not within the limits of SCALE_MIN and SCALE_MAX.

3. When the PB input is activated and the PB_STATUS is not equal to 128.

> *Inputs and outputs can stay unconnected, if their functions are not needed.*

This function block can be used in combination with the object type ANAALM_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



LIM_HYS

Data type	Default value	Range	Unit
REAL	0.5	0 ... 100	%

Dead band for Alarm Limit. % of scale (SCALE_MAX – SCALE_MIN).

LIM_HH_INHIB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of highhigh alarm.

If value of LIM_HH_INHIB is TRUE, then suppress highhigh alarm.

LIM_H_INHIB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of high alarm.

If value of LIM_H_INHIB is TRUE, then suppress high alarm.

LIM_L_INHIB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of low alarm.

If value of LIM_L_INHIB is TRUE, then suppress low alarm.

LIM_LL_INHIB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of lowlow alarm.

If value of LIM_LL_INHIB is TRUE, then suppress lowlow alarm.

IN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Analog input. The input must be in the range of 0 ... 27648.

Value to SCADA is scaled according to SCALE_MIN and SCALE_MAX.

SCALE_MAX

Data type	Default value	Range	Unit
INT	27648	-32768 ... 32767	-

Scaling parameter. Maximum value for output to SCADA and output of the function block.

IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

SCALE_MIN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Scaling parameter. Minimum value for output to SCADA and output of the function block.

IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

IN_HYS

Data type	Default value	Range	Unit
REAL	0.5	0 ... 100	%

Dead band for analog input.

If IN + IN_HYS > SCADA or IN - IN_HYS < SCADA, then SCADA value is updated.

EN_LIVE_ZERO

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Enable live zero for analog input.

An analog input is connected to an I/O module or CPU and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

INHIB_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of all alarms.

If value of INHIB_ERR is TRUE, all alarms are suppressed.

FORCE_SCALE_MIN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force output from function block.

If value of FORCE_SCALE_MIN is TRUE, output for SCADA and OUT are forced to SCALE_MIN.

When FORCE_SCALE_MAX and FORCE_SCALE_MIN inputs are made high at the same time, FORCE_SCALE_MAX has higher priority.

FORCE_SCALE_MAX

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force output from function block.

If value of FORCE_SCALE_MAX is TRUE, output for SCADA and OUT are forced to SCALE_MAX.

When FORCE_SCALE_MAX and FORCE_SCALE_MIN inputs are made high at the same time, FORCE_SCALE_MAX has higher priority.

FREEZE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force output from function block.

If value of FREEZE is TRUE, output for SCADA and OUT are forced to FREEZE_SP.

FREEZE_SP

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Freeze setpoint.

If value of FREEZE is TRUE, the SCADA and OUT will be forced to FREEZE_SP.

PB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Select PROFIBUS type of input.

If value of PB is TRUE, the analog input is not from an I/O module but from a PROFIBUS communication line.

No scaling of the SCADA and OUT value.

PB_STATUS

Data type	Default value	Range	Unit
BYTE	128	0 ... 255	-

Status of PROFIBUS communication.

If value of PB_STATUS is not equal to 128, a sensor error is generated.

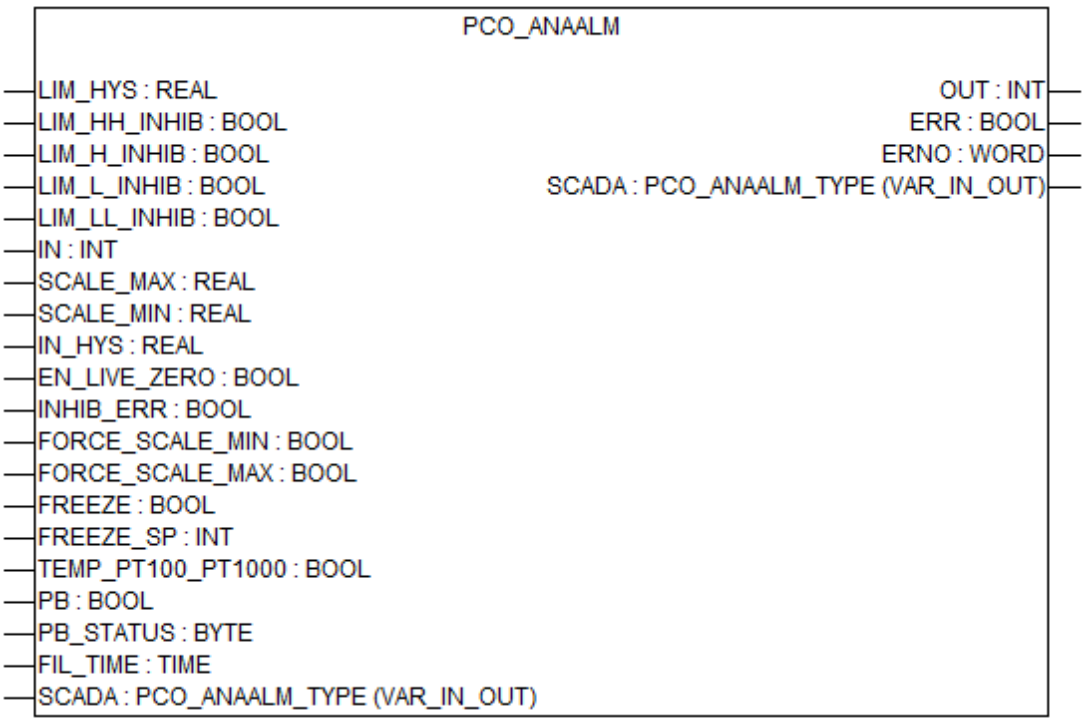
If value of PB_STATUS is 128, measurement is ok.

FIL_TIME

Data type	Default value	Range	Unit
TIME	TIME#2s	-	-

Filter time. 1st order filter for the input damps the variation of input.

Output description



OUT

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Scaled output, output scaled according to SCALE_MIN and SCALE_MAX.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, Input Parameter error.

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

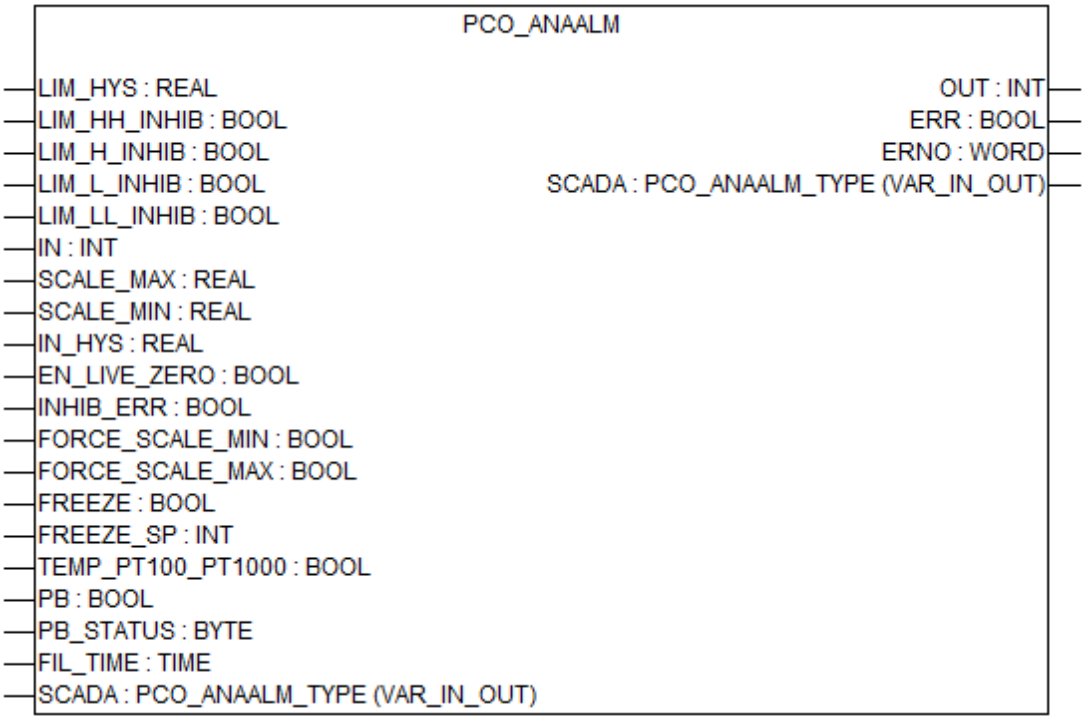
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *"Standard Function Block Libraries AC500"* in *"Error Messages of the Function Block Libraries"*.

Input/output description



SCADA

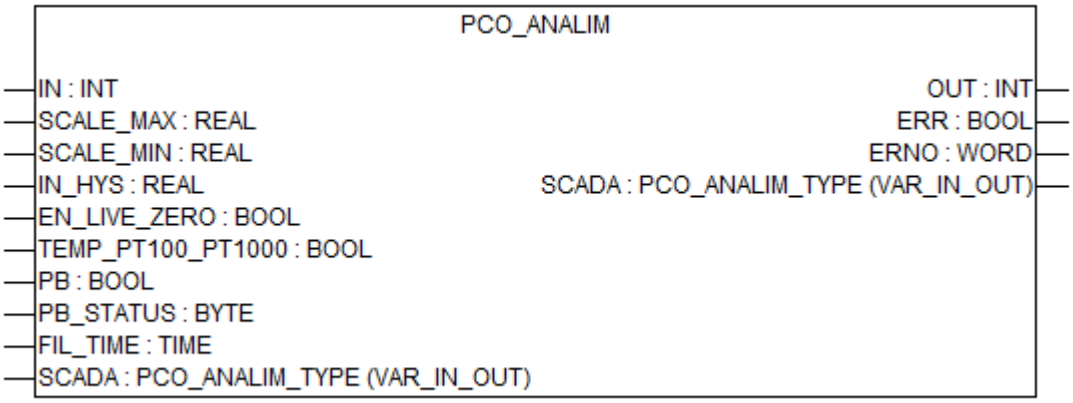
Data type	Default value	Range	Unit
PCO_ANAALM_ TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.VALUE	INT	Rescaled input value.	Output
SCADA.SENSE_ERR	BOOL	Sensor error. Open circuit or short circuit.	Output
SCADA.H2	BOOL	High-high alarm (H2).	Output
SCADA.H1	BOOL	High alarm (H1).	Output

Parameter	Data type	Description	In-/Output
SCADA.L1	BOOL	Low alarm (L1).	Output
SCADA.L2	BOOL	Low-low alarm (L2).	Output
SCADA.LIMH2	INT	Parameter from OS: Limit for H2 alarm.	Input
SCADA.LIMH1	INT	Parameter from OS: Limit for H1 alarm.	Input
SCADA.LIML1	INT	Parameter from OS: Limit for L1 alarm.	Input
SCADA.LIML2	INT	Parameter from OS: Limit for L2 alarm.	Input
SCADA.H2_VALUE	INT	Parameter to SCADA: Limit for H2 alarm	Output
SCADA.H1_VALUE	INT	Parameter to SCADA: Limit for H1 alarm	Output
SCADA.L1_VALUE	INT	Parameter to SCADA: Limit for L1 alarm	Output
SCADA.L2_VALUE	INT	Parameter to SCADA: Limit for L2 alarm	Output
AspectObjectType _Anaalm_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

PCO_ANALIM



The function block is designed for controlling an analog input without limit supervision.

The function block can be configured to handle the different ranges of the analog input, due to different electrical signals. E.g.

- 4 mA ... 20 mA
- 0 V ... 10 V or
- PT100 /PT1000

> *Please refer to AC500 hardware manual for detailed information regarding analog input modules and different parameters.*

The function block rescales the analog input (IN) from the actual value to SCALE_MIN → SCALE_MAX.

The IN range is from 0 ... 27648.The OUT is defined by:

$$OUT = \frac{IN}{27648} \times (SCALE_{\max} - SCALE_{\min}) + SCALE_{\min}$$

If TEMP_PT100_PT1000 or PB are to be set = TRUE, no rescaling of IN is made.

If an analog input is connected to an I/O module and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

Zero point suppression. The function block will force the input to zero when the input is between 0 and 0 - IN_HYS

A SENSOR ERROR is generated in the SCADA side in the following conditions:

1. When the analog input is not within the tolerance of 1000, hence the input can vary over a range of -1000 ... 28648, after which it generates an error.

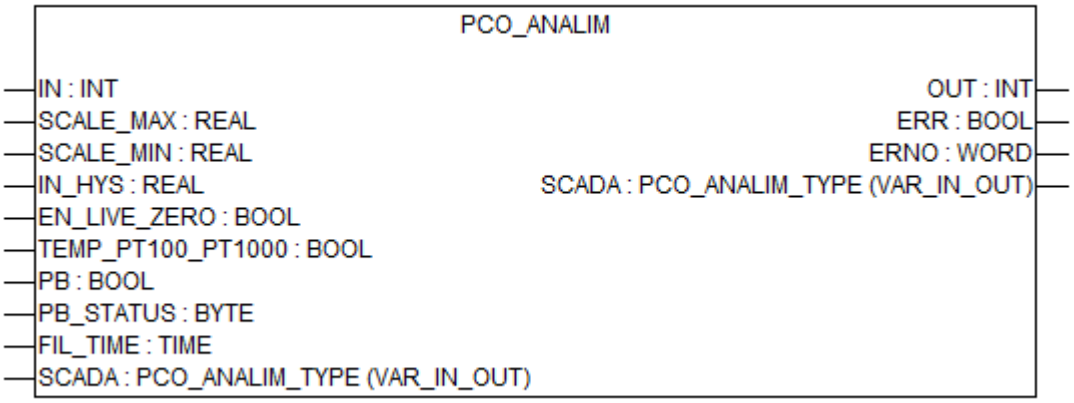
2. The output obtained at the end is not within the limits of SCALE_MIN and SCALE_MAX.

3. When the PB input is activated and the PB_STATUS is not equal to 128.

> *Inputs and outputs can stay unconnected, if their functions are not needed.*

This function block can be used in combination with the object type ANALIM_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



IN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Analog input. The input must be in the range of 0 ... 27648.

Value to SCADA is scaled according to SCALE_MIN and SCALE_MAX.

SCALE_MAX

Data type	Default value	Range	Unit
INT	27648	-32768 ... 32767	-

Scaling parameter. Maximum value for output to SCADA and output of the function block.

IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

SCALE_MIN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Scaling parameter. Minimum value for output to SCADA and output of the function block.

IN will be rescaled to the range defined between SCALE_MIN and SCALE_MAX by the user.

IN_HYS

Data type	Default value	Range	Unit
REAL	0.5	0 ... 100	%

Dead band for analog input.

If IN + IN_HYS > SCADA or IN - IN_HYS < SCADA, then SCADA value is updated.

EN_LIVE_ZERO

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Enable live zero for analog input.

An analog input is connected to an I/O module or CPU and the module cannot detect live zero (2 V ... 10 V), live zero can be set if value of EN_LIVE_ZERO is TRUE.

TEMP_PT100_PT1000

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Select temperature sensor type of input.

If value of TEMP_PT100_PT1000 is TRUE, a temperature sensor is connected direct on an I/O module.

No scaling of the SCADA and OUT value.

PB

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Select PROFIBUS type of input.

If value of PB is TRUE, the analog input is not from an I/O module but from a PROFIBUS communication line.

No scaling of the SCADA and OUT value.

PB_STATUS

Data type	Default value	Range	Unit
BYTE	128	0 ... 255	-

Status of PROFIBUS communication.

If value of PB_STATUS is not equal to 128, a sensor error is generated.

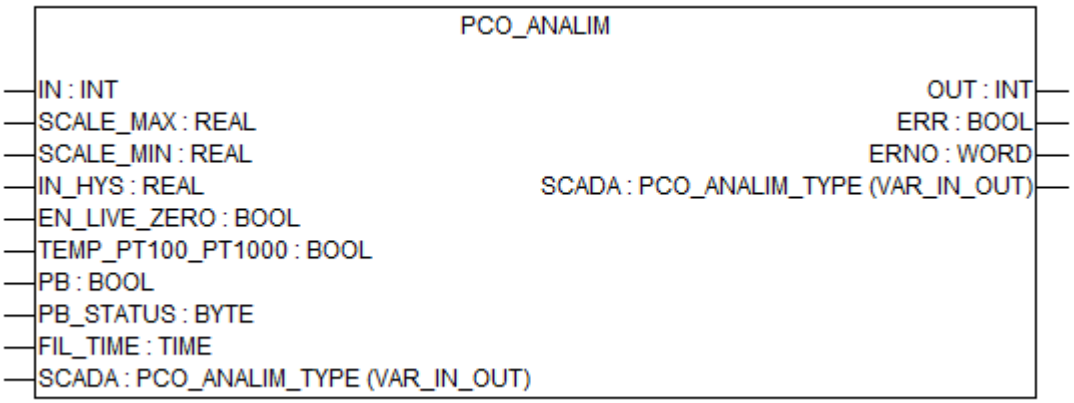
If value of PB_STATUS is 128, measurement is ok.

FIL_TIME

Data type	Default value	Range	Unit
TIME	TIME#2s	-	-

Filter time. 1st order filter for the input damps the variation of input.

Output description



OUT

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Scaled output, output scaled according to SCALE_MIN and SCALE_MAX.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, Input Parameter error.

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

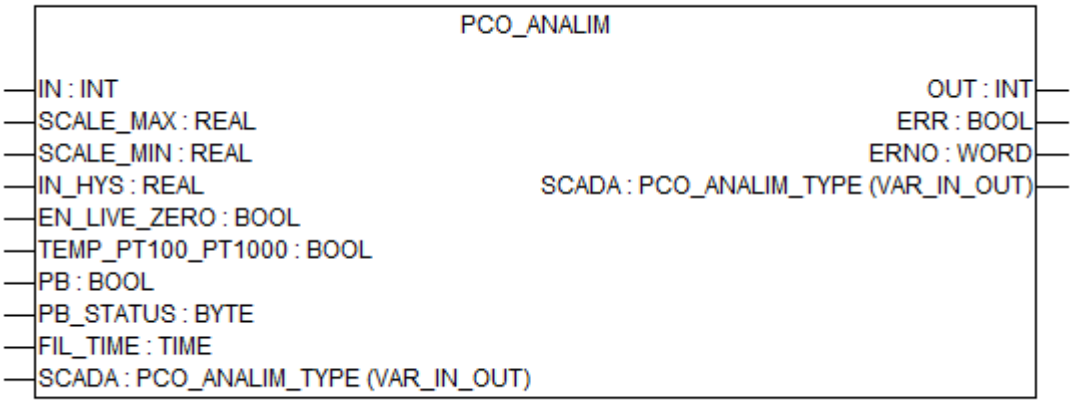
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *"Standard Function Block Libraries AC500"* in *"Error Messages of the Function Block Libraries"*.

Input/output description



SCADA

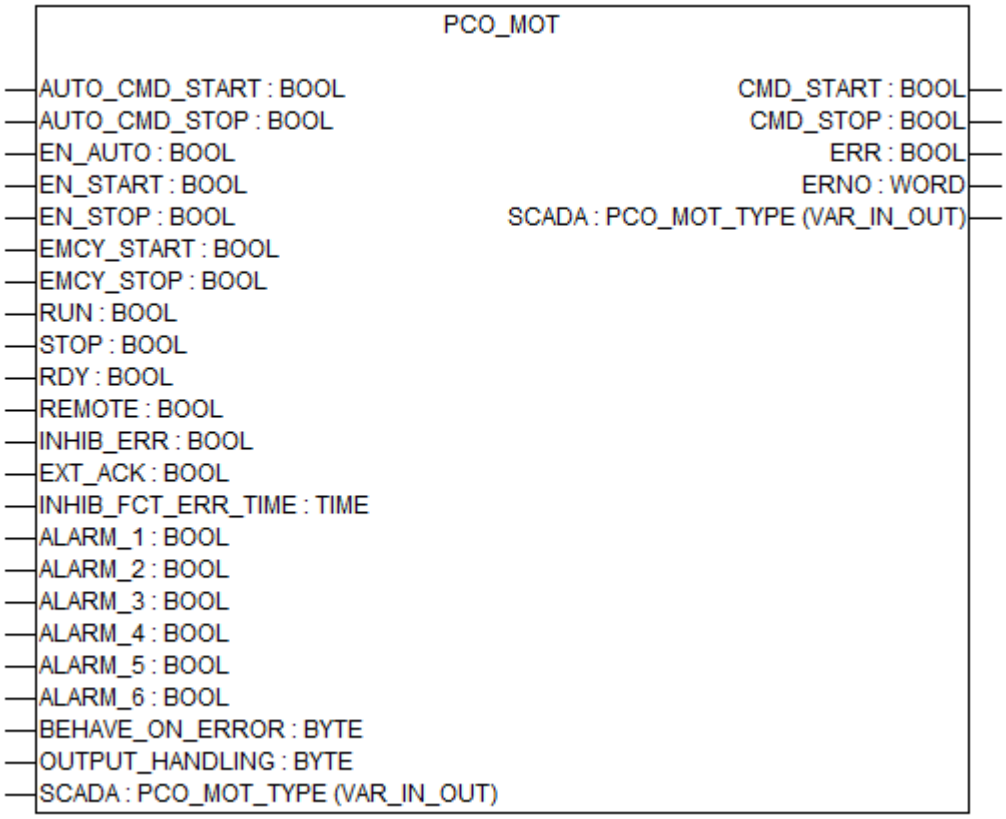
Data type	Default value	Range	Unit
PCO_ANALIM_TYPE	-	-	-

Structure variable for communication to and from SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.VALUE	INT	Rescaled input value.	Output
SCADA.SENSE_ERR	BOOL	Sensor error. Open circuit or short circuit.	Output
AspectObjectType _Analim_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

Motor

PCO_MOT



This function block is designed for controlling a fixed speed motor.

Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_START and CMD_STOP can be configured. E.g. if CMD_START and CMD_STOP should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.).

Set motor to auto mode

The motor can be switched to automatic mode from the SCADA faceplate if the input EN_AUTO is TRUE.

Settings in FB side	Data type	Set Motor to Auto Mode
EN_AUTO	BOOL	TRUE
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Motor to Auto Mode
SCADA.CMDAUT	BOOL	Rising edge needs to be given.

Start / stop motor in auto mode

Settings in FB side	Data type	Start / Stop Motor in Auto Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_START	BOOL	ACTIVE
EN_STOP	BOOL	ACTIVE
AUTO_CMD_START	BOOL	Rising edge needs to be given to AUTO_CMD_START input of FB
AUTO_CMD_STOP	BOOL	Rising edge needs to be given to AUTO_CMD_STOP input of FB

Settings in SCADA side	Data type	Start / Stop Motor in Auto Mode
No settings need to be done.		

Set motor to manual mode

The motor can be switched to manual mode.

If the input EN_AUTO is FALSE, then the function block is forced to manual mode.

The motor can be operated in manual mode if EN_AUTO is either TRUE or FALSE.

The difference is that if EN_AUTO is TRUE, the switching between Auto and Manual is possible, else the motor and controller are both to Manual.

Settings in FB side	Data type	Set Motor to Manual Mode
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Motor to Manual Mode
SCADA.CMDMAN	BOOL	Rising edge of SCADA.CMDMAN input to reset auto mode if it was set before

Start / stop motor in manual mode

Settings in FB side	Data type	Start / Stop Motor in Manual Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_START	BOOL	ACTIVE
EN_STOP	BOOL	ACTIVE

Settings in SCADA side	Data type	Start / Stop Motor in Manual Mode
CMDSTR	BOOL	Rising edge of CMDSTR needs to be given.
CMDSTP	BOOL	Rising edge of CMDSTP needs to be given.

In automatic mode the motor can be started/stopped using the function block inputs (AUTO_CMD_START/AUTO_CMD_STOP).

In manual mode the motor can be started and stopped from the SCADA faceplate.

The function block includes supervision of the feedback signals RUN and STOP from the motor.

If the motor fails to run or stop within the supervision time INHIB_FCT_ERR_TIME, it will generate a functional error.

An external error is generated when READY input is false.

This function block can be used in combination with the object type MOT_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Examples

The function block is used for controlling a fixed speed motor, which can be used for controlling the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level switch (high and low)
- Pump
- Main power supply, circuit breaker (or similar)

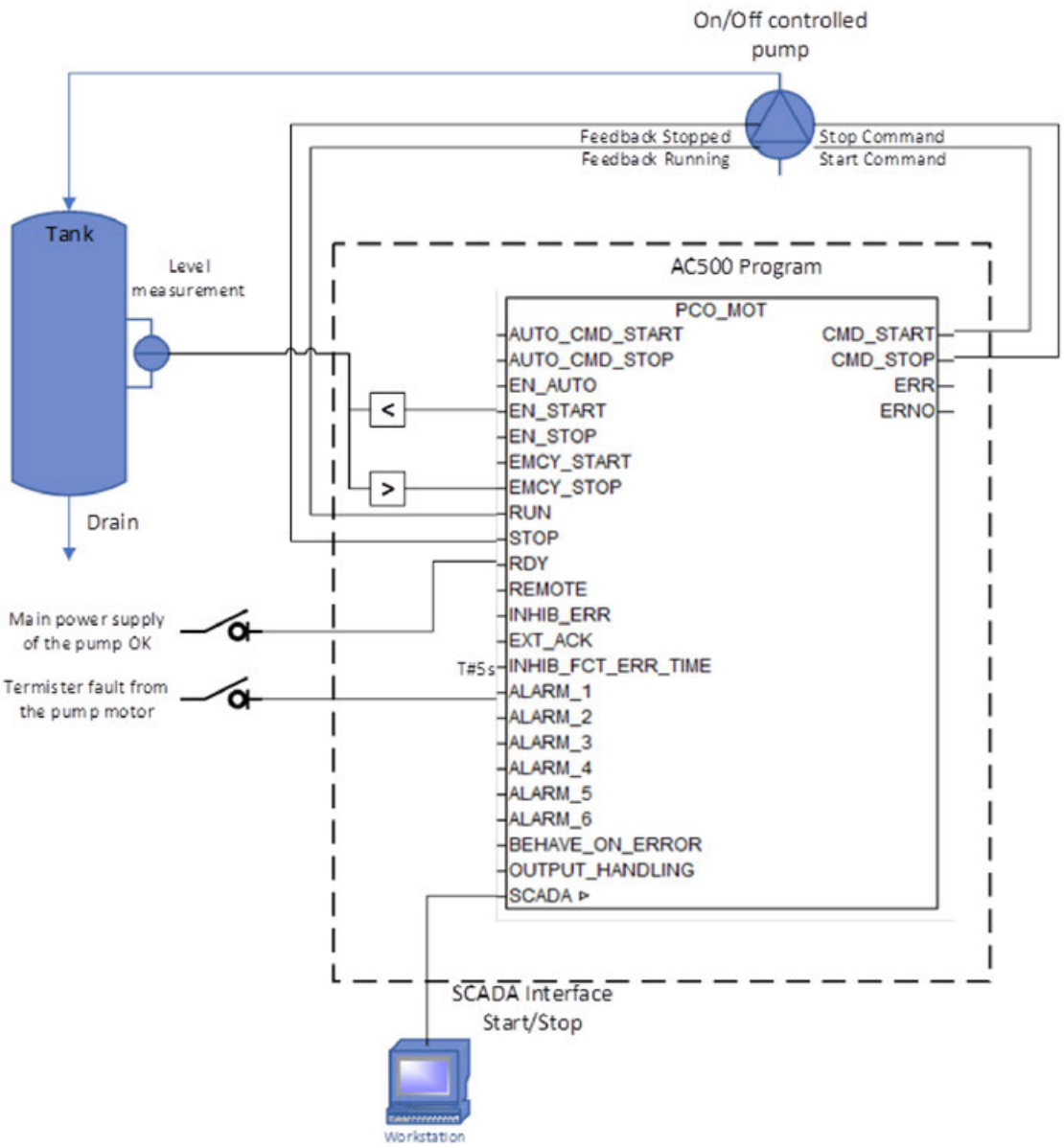
Manual mode:

- Pump controlled by the operator.

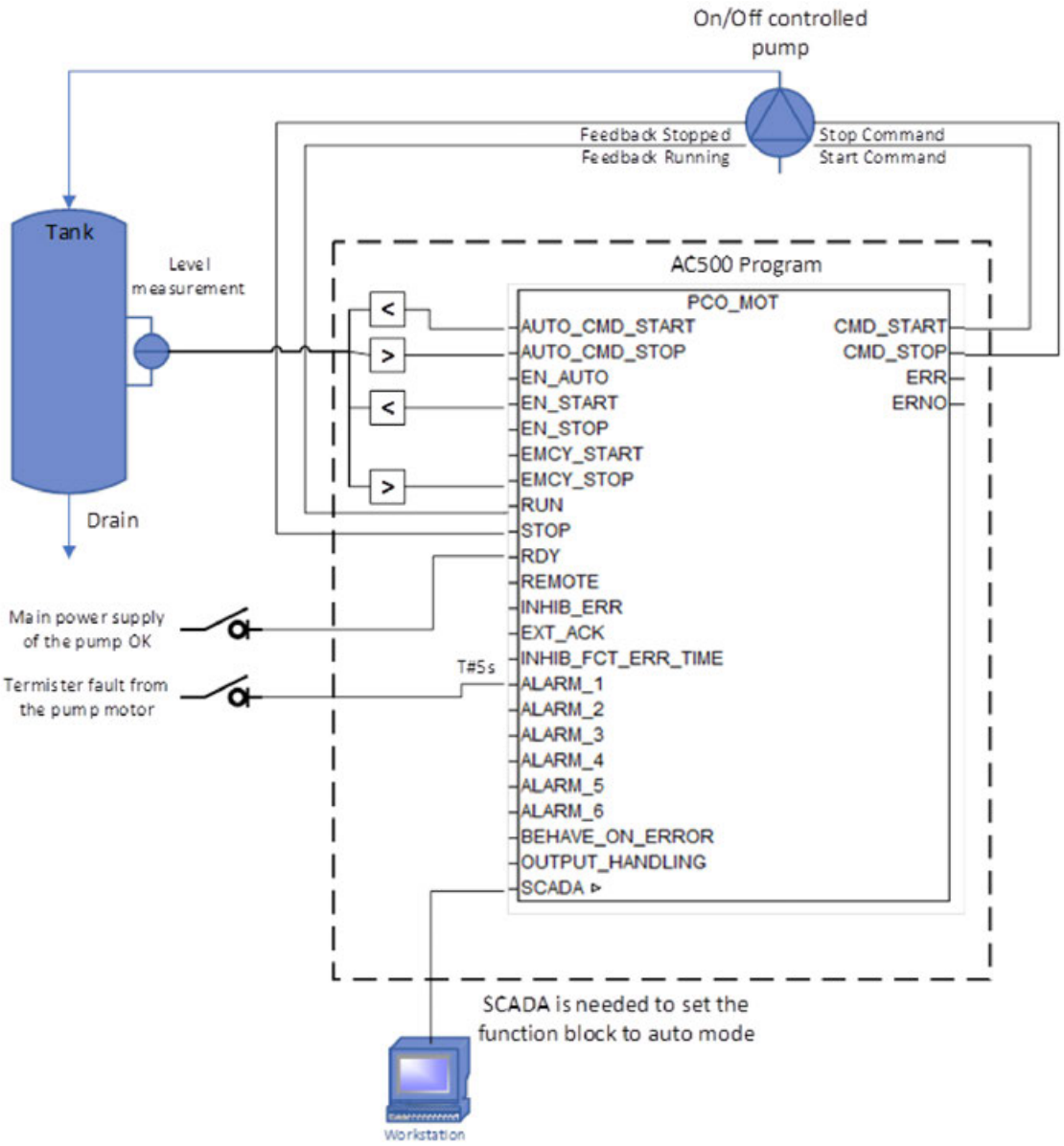
Automatic mode:

- Pump controlled by the AC500 controller.

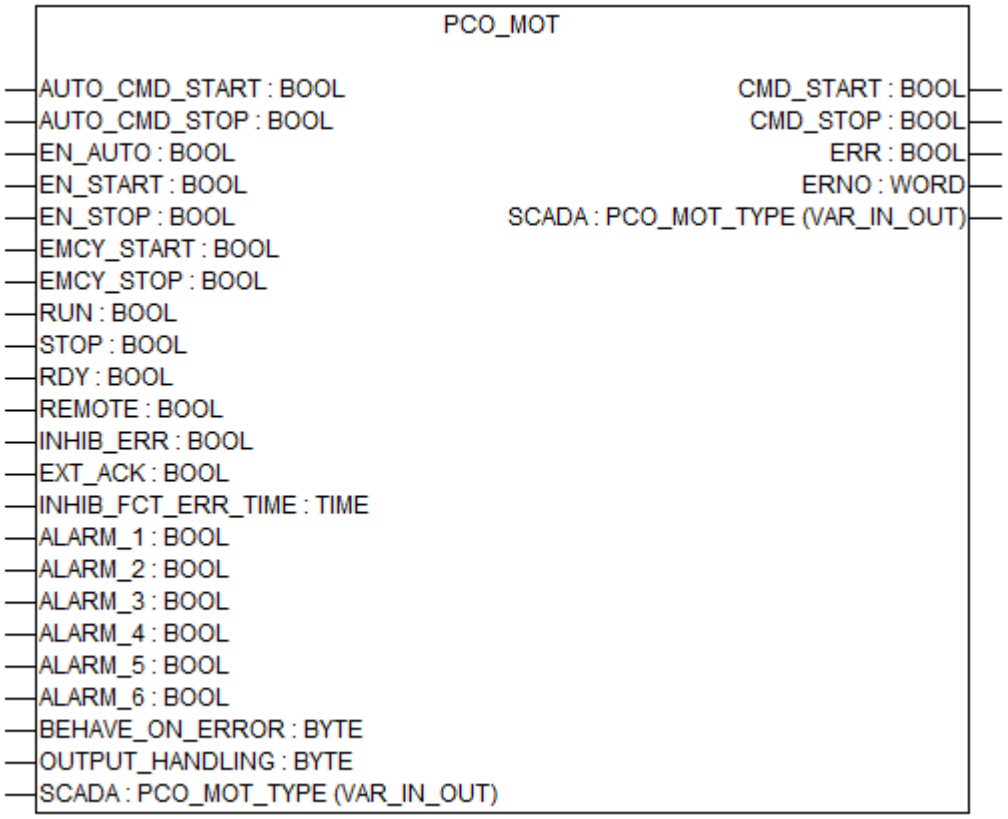
PCO_MOT manual mode example



PCO_MOT automatic mode example



Input description



AUTO_CMD_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the motor to start if it is in automatic mode.

E.g.AUTO_CMD_START could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

AUTO_CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the motor to stop if the actuator is in automatic.

AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_START and AUTO_CMD_STOP are active at the same time, then the commands CMD_START and CMD_STOP get reset.

EN_AUTO

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable automatic mode of the PCO_MOT.

PCO_MOT can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_MOT is forced in manual mode if value of EN_AUTO is FALSE.

But the motor can also be operated in manual mode if EN_AUTO is TRUE provided that manual start and stop selection is made on the SCADA side.

EN_START

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable start of the motor.

Motor can be started if value of EN_START is TRUE.

Motor cannot be started if value of EN_START is FALSE.

EN_START is independent of operation mode of PCO_MOT.

EN_START has no effect in case of emergency start of function block motor EMCY_START.

EN_START has no effect on a motor already running.

EN_STOP

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable stop of the motor.

Motor can be stopped if value of EN_STOP is TRUE.

Motor cannot be stopped if value of EN_STOP is FALSE.

EN_STOP is independent of operation mode of PCO_MOT.

EN_STOP has no effect in case of emergency stop of function block motor EMCY_STOP.

EN_STOP has no effect on an already stopped motor.

EMCY_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency start of motor.

If value of EMCY_START is TRUE and value of EMCY_STOP is FALSE, the motor will start.

If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.

Independent of operation mode of PCO_MOT.

EMCY_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency stop of the motor.

If value of EMCY_STOP is TRUE, the motor will stop.

If value of EMCY_START is TRUE and value of EMCY_STOP is TRUE, the motor will stop.

Independent of operation mode of PCO_MOT.

RUN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the motor.

Motor is running if the value of RUN is TRUE.

Independent of operation mode of PCO_MOT.

STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the motor.

Motor is stopped if the value of STOP is TRUE.

Independent of operation mode of PCO_MOT.

RDY

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Motor is ready for operation.

Motor is ready for operation if value of RDY is TRUE.

Value of RDY is FALSE results in an external error.

Independent of operation mode of PCO_MOT.

REMOTE

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Allow motor to be controlled from SCADA.

Value of REMOTE is TRUE control from SCADA and function block is enabled.

Value of REMOTE is FALSE motor is controlled only from function block.

PCO_MOT will align the block according to feedback signals.

Independent of operation mode of PCO_MOT.

INHIB_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of alarms.

If value of INHIB_ERR is TRUE, all alarms from PCO_MOT are suppressed.

Independent of operation mode of PCO_MOT.

EXT_ACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

External acknowledge of alarms.

If value of EXT_ACK is TRUE, all alarms from PCO_MOTCON are acknowledged.

Independent of operation mode of PCO_MOT.

INHIB_FCT_ERR_TIME

Data type	Default value	Range	Unit
TIME	TIME#5s	-	-

Maximum delay time from command to response from process.

If response is not received within this time limit, a function error will be generated.

Alarm_1

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 1.

If value of ALARM_1 is TRUE Alarm_1 is active.

Independent of operation mode of PCO_MOT.

Alarm_2

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 2.

If value of ALARM_2 is TRUE Alarm_2 is active.

Independent of operation mode of PCO_MOT.

Alarm_3

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 3.

If value of ALARM_3 is TRUE Alarm_3 is active.

Independent of operation mode of PCO_MOT.

Alarm_4

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_MOT.

Alarm_5

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_MOT.

Alarm_6

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_MOT.

BEHAVE_ON_ERROR

Data type	Default value	Range	Unit
BYTE	0	0 ... 3	-

Actions that need to be executed by the FB at the event of error as set by the user.

The user can set a range of values from 0 ... 3.

BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).

BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.

BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.

BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto/ Manual) of PCO_MOT.

OUTPUT_HANDLING

Data type	Default value	Range	Unit
BYTE	1	0 ... 2	-

Behavior of command outputs, CMD_START / CMD_STOP.

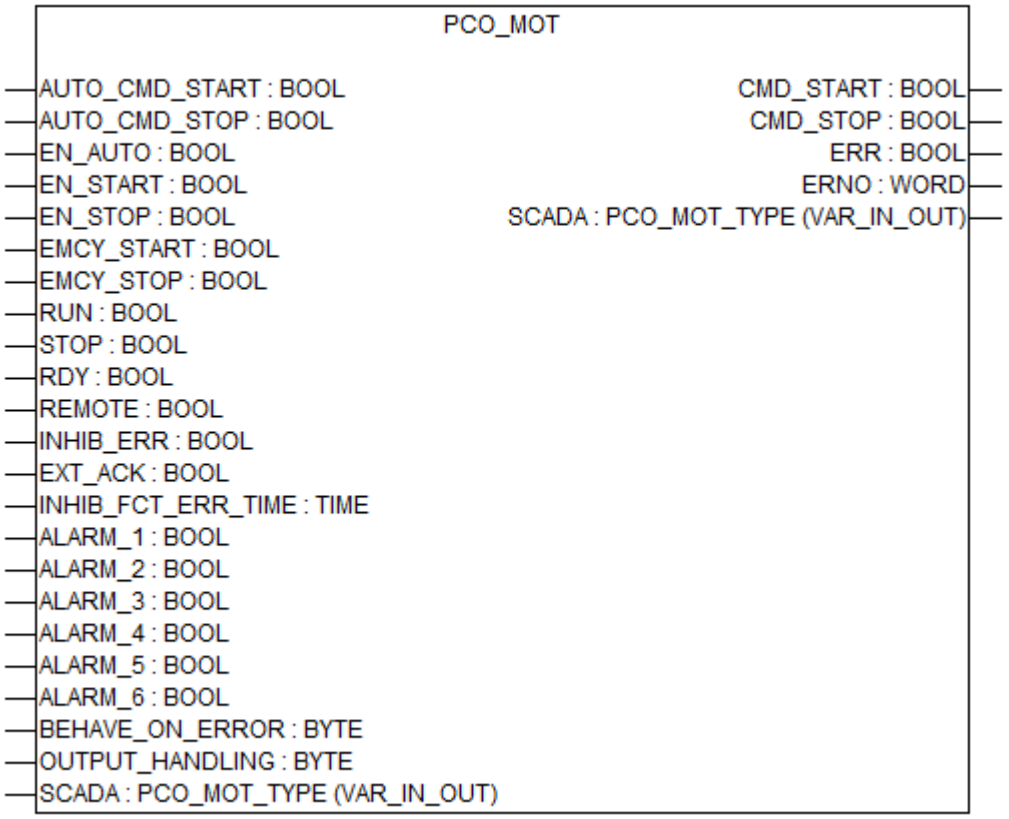
The user can set a range of values from 0 ... 2

OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.

OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP)) to remain active at RUN or STOP feedback.

With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_MOT.

Output description



CMD_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output start, to be connected to hardware output.

CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output stop, to be connected to hardware output.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, including:

- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_START
- EMCY_STOP

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

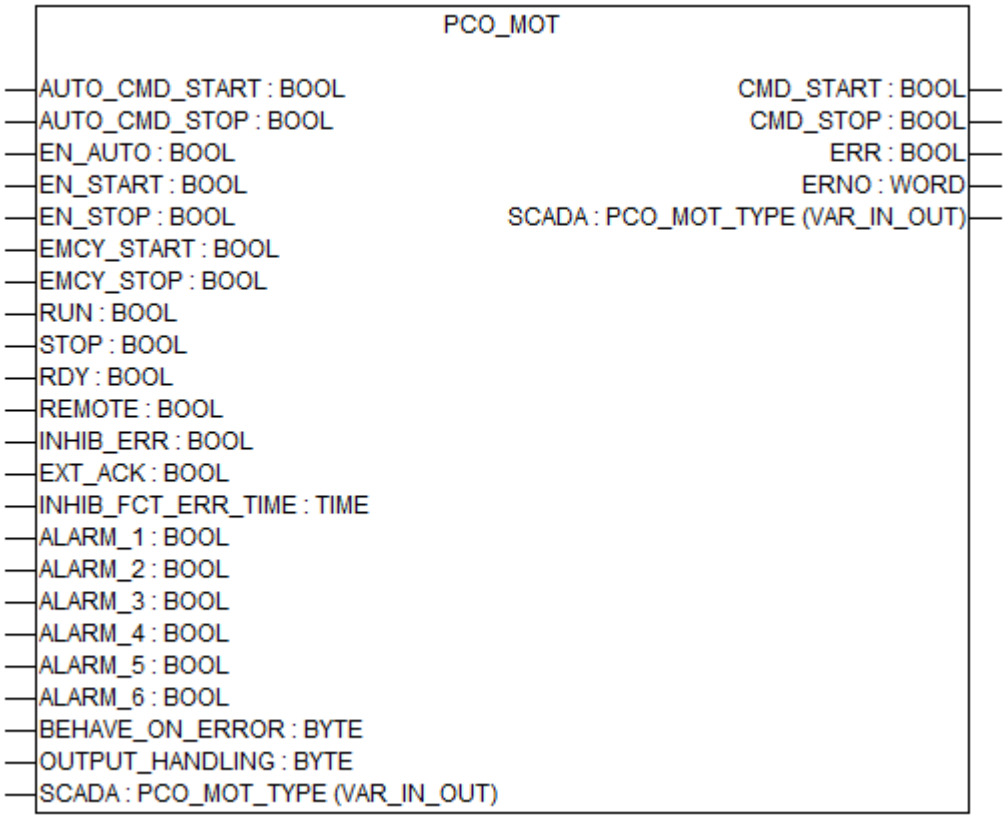
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *“Standard Function Block Libraries AC500”* in *“Error Messages of the Function Block Libraries”*.

Input/output description



SCADA

Data type	Default value	Range	Unit
PCO_MOT_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.CMDSTR	BOOL	Start command in Manual mode	Input
SCADA.CMDSTP	BOOL	Stop command in Manual mode	Input
SCADA.CMDAUT	BOOL	Auto command	Input
SCADA.CMDMAN	BOOL	Manual command	Input
SCADA.CMDRES	BOOL	Acknowledge active alarms	Input
SCADA.AUTO	BOOL	Motor in automatic mode	Output
SCADA.IN	BOOL	Motor is running	Output
SCADA.OUT	BOOL	Motor is stopped	Output
SCADA.REMOTE	BOOL	Motor can be controlled from SCADA REMOTE = FALSE → Local operation	Output
SCADA.READY	BOOL	Motor is ready for operation	Output
SCADA.FNCERR	BOOL	Functional error	Output
SCADA.EXTERR	BOOL	External error (is generated when the motor is not ready)	Output

Parameter	Data type	Description	In-/Output
SCADA.AL1	BOOL	Auxiliary alarm no. 1	Output
SCADA.AL2	BOOL	Auxiliary alarm no. 2	Output
SCADA.AL3	BOOL	Auxiliary alarm no. 3	Output
SCADA.AL4	BOOL	Auxiliary alarm no. 4	Output
SCADA.AL5	BOOL	Auxiliary alarm no. 5	Output
SCADA.AL6	BOOL	Auxiliary alarm no. 6	Output
SCADA.OSMsg1 *)	WORD	Word representing the status of the PCO_MOT	Output
SCADA.OSMsg2 *)	WORD	Word representing the status of the PCO_MOT	Output
AspectObjectType _Mot_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

*) structure described separately

SCADA.OSMsg1

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_MOT.

Bit	Description
Bit 0	Motor is running
Bit 1	Motor is stopped
Bit 2	Motor is starting
Bit 3	Motor is stopping
Bit 4	External error (Not ready)
Bit 5	Functional error
Bit 6	Motor released for start
Bit 7	Motor released for stop
Bit 8	Local operation (Not remote)
Bit 9	Common alarm (External alarm + Functional alarm + EMCY_START/EMCY_STOP + ALARM_1 ... ALARM_6)
Bit 10	Unacknowledged alarm
Bit 11	Motor in automatic mode
Bit 12	Not used
Bit 13	Motor not released for automatic mode
Bit 14	Not used
Bit 15	Not used

SCADA.OSMsg2

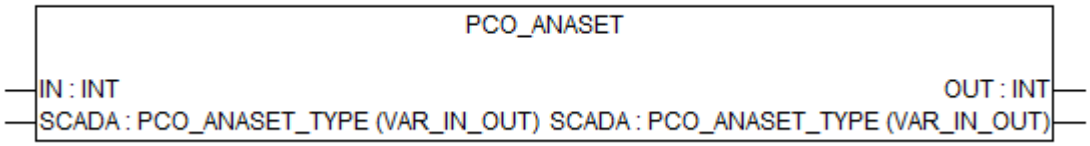
Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_MOT.

Bit	Description
Bit 0	Emergency In
Bit 1	Emergency Out
Bit 2	Auxiliary alarm 1
Bit 3	Auxiliary alarm 2
Bit 4	Auxiliary alarm 3
Bit 5	Auxiliary alarm 4
Bit 6	Auxiliary alarm 5
Bit 7	Auxiliary alarm 6
Bit 8	Not used
Bit 9	Not used
Bit 10	Not used
Bit 11	Not used
Bit 12	Not used
Bit 13	Not used
Bit 14	Not used
Bit 15	Not used

Setpoints

PCO_ANASET



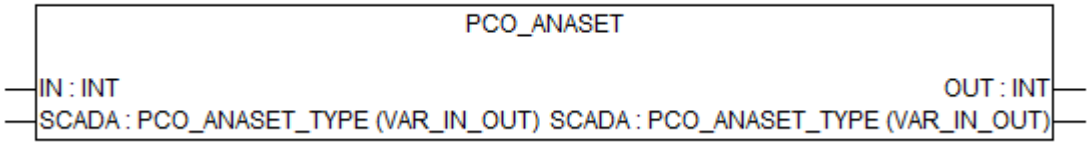
This function block is designed for receiving an analog setpoint from the SCADA system.

This value is entered at .PAR variable of the SCADA side. The actual analog setpoint is entered at IN input of the function block and it is represented by .VALUE variable in the SCADA side.

To retain the variable value in case of power ON/OFF or download, the variable connected to the SCADA In/Output should be declared as global retain persistant (or use %R area) variable in the program.

This function block can be used in combination with the object type ANASET_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



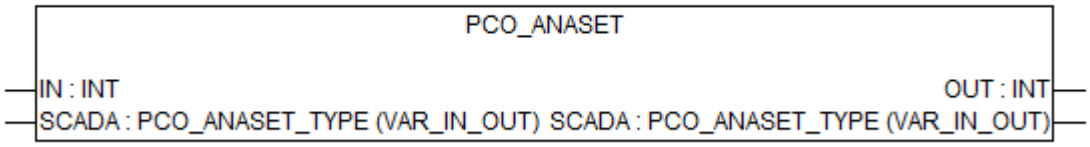
IN

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Actual setpoint.

Parameter is sent to SCADA.

Output description



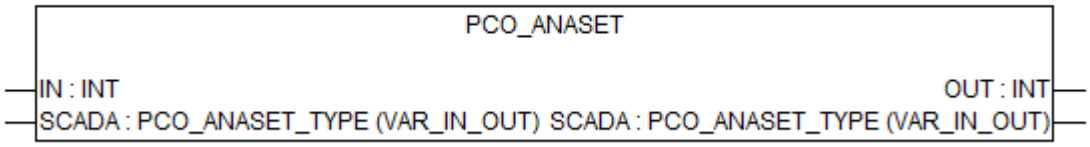
OUT

Data type	Default value	Range	Unit
INT	0	-32768 ... 32767	-

Setpoint received from SCADA.

Parameter received from SCADA.

Input/output description



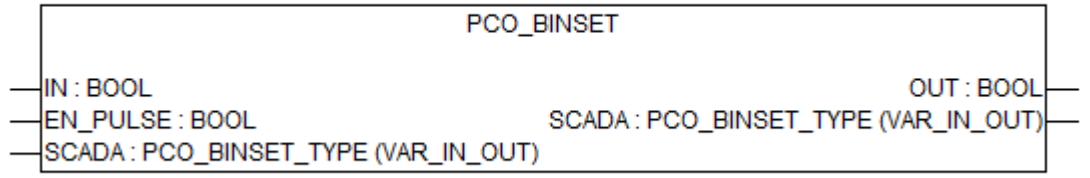
SCADA

Data type	Default value	Range	Unit
PCO_ANASET_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.VALUE	INT	Actual setpoint	Output
SCADA.PAR	INT	Setpoint from SCADA	Input
AspectObjectType _Anaset_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

PCO_BINSET



This function block is designed for receiving single command from the SCADA system.

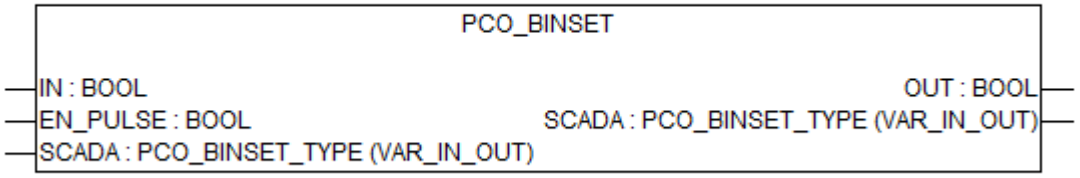
EN_PULSE input is used to reset command from SCADA. CMD variable from SCADA side is used to give command.

If value of EN_PULSE is TRUE, the command will only be active for one program cycle.

The CMD variable is connected to OUT (output) of function block.

This function block can be used in combination with the object type BINSET_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

Input description



IN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Actual status, that is the result of a command from SCADA side.

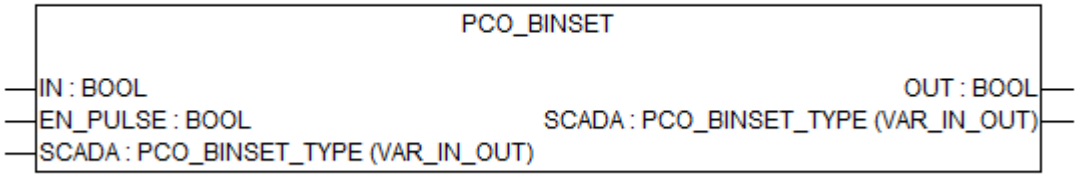
EN_PULSE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Pulse output.

If value of EN_PULSE is TRUE, CMD variable would be reset and OUT will only be active for one program cycle.

Output description



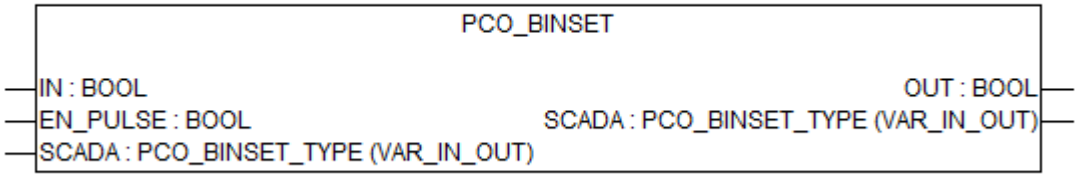
OUT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Single command from SCADA.

If value of EN_PULSE is TRUE the single command will only be active for one program cycle.

Input/output description



SCADA

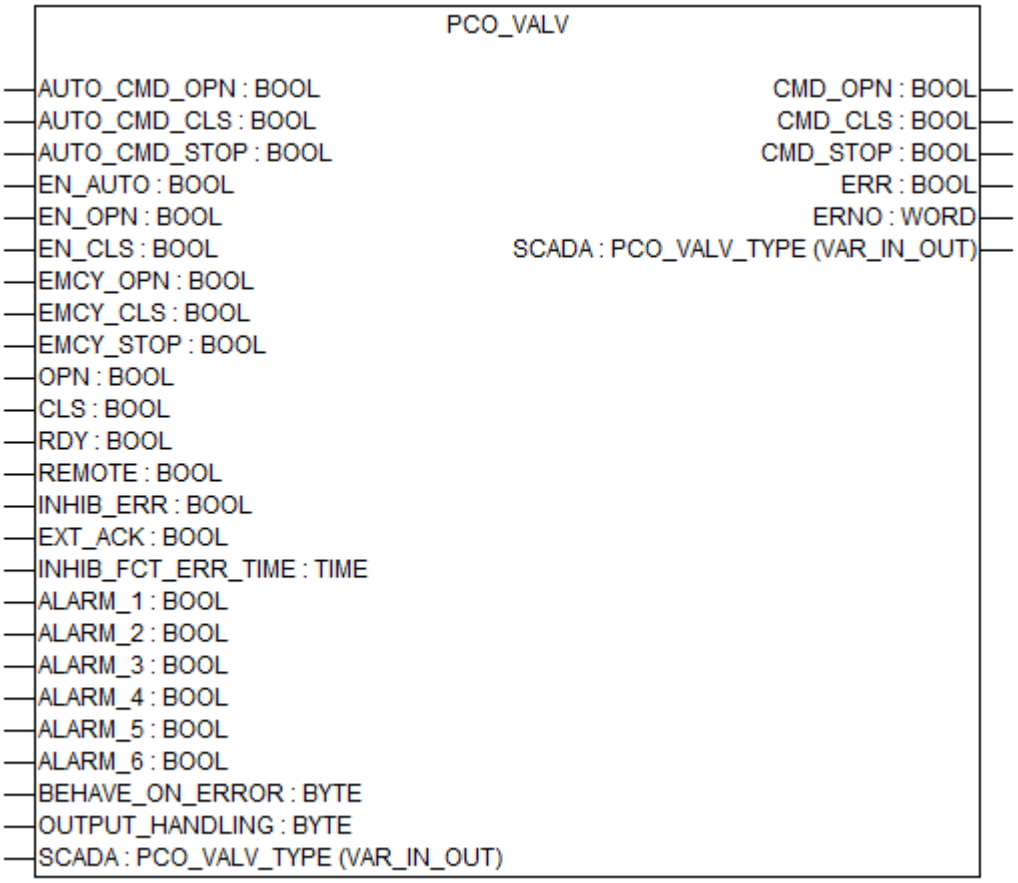
Data type	Default value	Range	Unit
PCO_BINSET_TYPE	-	-	-

Structure variable for communication between AC500 and SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.STATUS	BOOL	Actual status, feedback	Output
SCADA.CMD	BOOL	Single command from SCADA	Input
AspectObjectType _Binset_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

Valve

PCO_VALV



This function block is designed for controlling a motor controlled ON/OFF valve.

Utilizing the inputs of the function block BEHAVE_ON_ERROR and OUTPUT_HANDLING different modes of CMD_OPN, CMD_STOP and CMD_CLS can be configured. E.g. if CMD_OPN should be pulsed or persistent signals. In addition, 6 different alarms can be connected to the function block (ALARM_1 to ALARM_6, thermal switch etc.)

There are two modes in which the valve can be operated, Auto mode or Manual mode.

In automatic mode the valve can be opened/stopped/closed using the function block inputs (AUTO_CMD_OPN/AUTO_CMD_STOP/AUTO_CMD_CLS).

Set valve to auto mode

The valve is set to auto mode when the following conditions are satisfied:

Settings in FB side	Data type	Set Valve to Auto Mode
EN_AUTO	BOOL	TRUE
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Valve to Auto Mode
SCADA.CMDAUT	BOOL	Rising edge needs to be given.

Open / stop / close valve in auto mode

Settings in FB side	Data type	Open / Stop / Close Valve in Auto Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_OPN	BOOL	ACTIVE
EN_CLS	BOOL	ACTIVE
AUTO_CMD_OPN	BOOL	Rising edge needs to be given to AUTO_CMD_OPN input of FB
AUTO_CMD_STP	BOOL	Rising edge needs to be given to AUTO_CMD_STP input of FB
AUTO_CMD_CLS	BOOL	Rising edge needs to be given to AUTO_CMD_CLS input of FB

Settings in SCADA side	Data type	Open / Stop / Close Valve in Auto Mode
SCADA.CMDAUT	BOOL	Rising edge needs to be given.

Set valve to manual mode

In manual mode the valve can be opened and closed from the SCADA faceplate.

The valve is set to manual mode when the following conditions are satisfied:

Settings in FB side	Data type	Set Valve to Manual Mode
REMOTE	BOOL	ACTIVE
RDY	BOOL	ACTIVE

Settings in SCADA side	Data type	Set Valve to Manual Mode
SCADA.CMDMAN	BOOL	Rising edge of SCADA.CMDMAN input to reset auto mode if it was set before

Open / close valve in manual mode

Settings in FB side	Data type	Open / Close Valve in Manual Mode
INHIB_FCT_ERR_TIME	TIME	The value of INHIB_FCT_ERR_TIME has to be greater than 0 s to avoid functional error.
EN_OPN	BOOL	ACTIVE
EN_CLS	BOOL	ACTIVE

Settings in SCADA side	Data type	Open / Close Valve in Manual Mode
CMDOPN	BOOL	Rising edge of CMDOPN needs to be given.
CMDCLS	BOOL	Rising edge of CMDCLS needs to be given.

This function block can be used in combination with the object type VALVCON_PLC included in PLC Object Library (an object library for 800xA based PLC Connect).

The function block includes supervision of the feedback signals OPN and CLS from the valve.

If the valve fails to open or close within the supervision time INHIB_FCT_ERR_TIME, it will raise a functional error.

Actions in case of a functional error are controlled by the parameter BEHAVE_ON_ERROR.

Examples

The function block is used for controlling a valve which in turn controls the level of fluid in a tank.

These components are included in the following example:

- Tank
- Level switch (high and low)
- Valve
- Main power supply, circuit breaker (or similar)

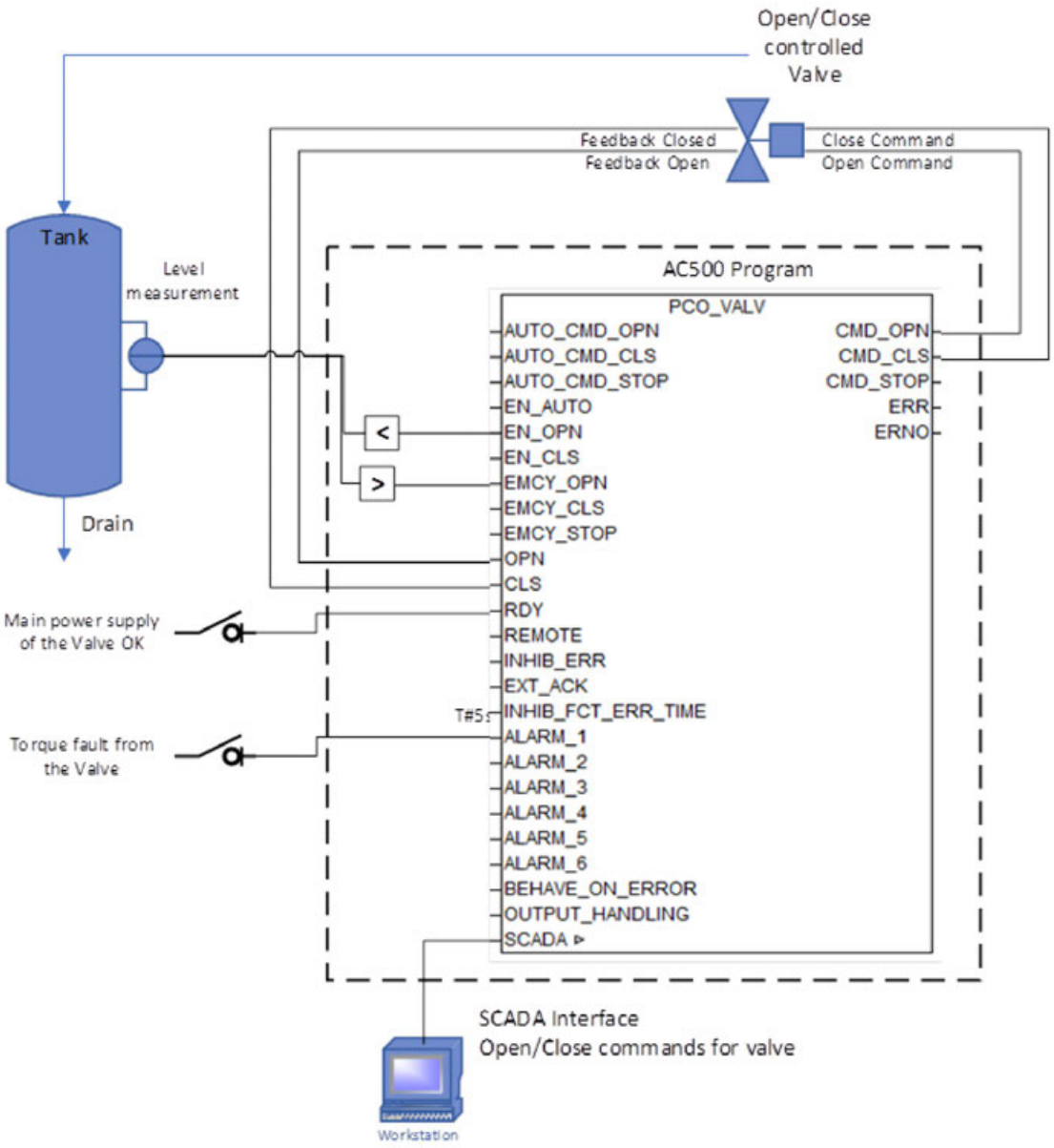
Manual mode:

- Valve controlled by the operator.

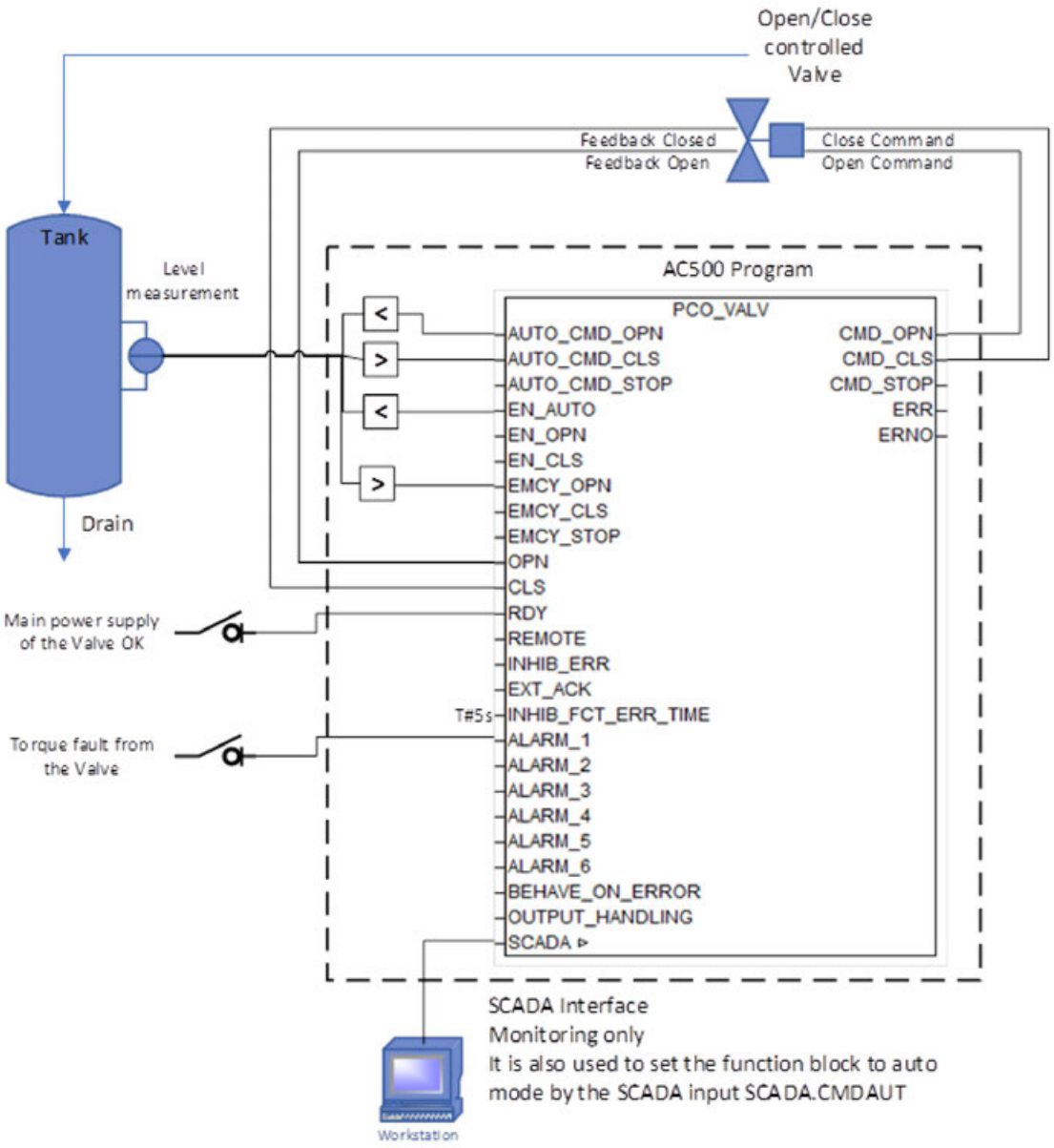
Automatic mode:

- Valve controlled by the AC500 controller.

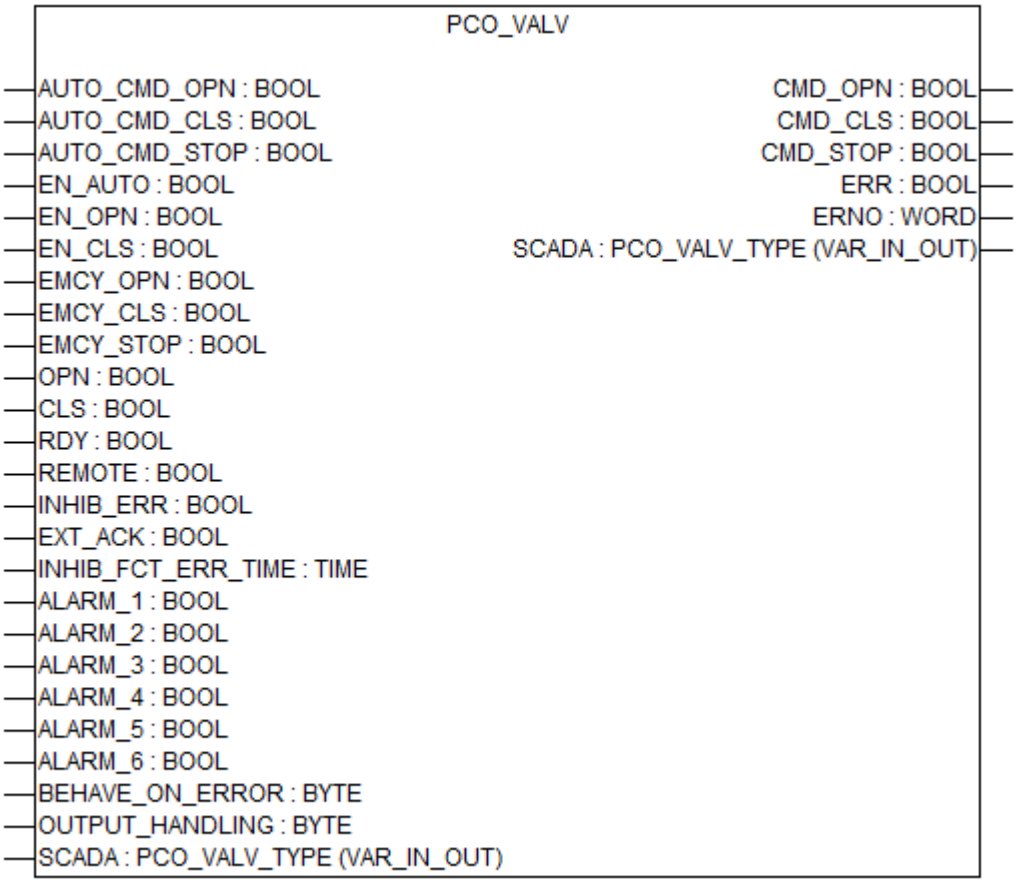
PCO_VALV manual mode example



PCO_VALV automatic mode example



Input description



AUTO_CMD_OPN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the valve to open, if the valve is in automatic mode.

AUTO_CMD_OPN could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_OPN and AUTO_CMD_CLS are active at the same time, then the commands CMD_OPN and CMD_CLS get reset.

AUTO_CMD_CLS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the valve to close, if the valve is in automatic mode.

AUTO_CMD_CLS could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

If AUTO_CMD_OPN and AUTO_CMD_CLS are active at the same time, then the commands CMD_OPN and CMD_CLS get reset.

AUTO_CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Force the valve to stop, if the valve is in automatic mode.

Valve will stop in intermediate position.

AUTO_CMD_STOP could be activated from a sequence in the PLC program. The function block reacts on "0" to "1" transition of this input.

EN_AUTO

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable automatic mode of the PCO_VALV.

PCO_VALV can be set in automatic mode if value of EN_AUTO is TRUE.

PCO_VALV is forced in manual mode if value of EN_AUTO is FALSE.

EN_OPN

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable opening of the valve.

Valve can be opened if value of EN_OPN is TRUE.

Valve cannot be opened if value of EN_OPN is FALSE.

EN_OPN has no effect in case of emergency open of the valve, e.g. EMCY_OPN.

EN_OPN has no effect on an already opened valve.

EN_CLS

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Enable closing of the valve.

Valve can be closed if value of EN_CLS is TRUE.

Valve cannot be closed if value of EN_CLS is FALSE.

EN_CLSN has no effect in case of emergency close of the valve, e.g. EMCY_CLS.

EN_CLS has no effect on an already closed valve.

EMCY_OPN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency open of the valve.

If value of EMCY_OPN is TRUE and value of EMCY_CLS is FALSE, the valve will open.

If value of EMCY_OPN is TRUE and the value of EMCY_CLS is TRUE, the valve will close.

If value of EMCY_STOP is TRUE, the valve will stop.

Independent of operation mode of PCO_VALV.

EMCY_CLS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency close of the valve.

If value of EMCY_CLS is TRUE, the valve will close.

If value of EMCY_OPN is TRUE and value of EMCY_CLS is TRUE, the valve will close.

If value of EMCY_STOP is TRUE, the valve will stop.

Independent of operation mode of PCO_VALV.

EMCY_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Emergency stop of the valve.

If value of EMCY_STOP is TRUE, the valve will stop. Highest priority.

Independent of operation mode of PCO_VALV.

OPN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the valve.

If the value of OPN is TRUE, the valve is open.

Independent of operation mode of PCO_VALV.

CLS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Feedback signal from the valve.

If the value of CLS is TRUE, the valve is closed.

Independent of operation mode of PCO_VALV.

RDY

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Valve is ready for operation.

Valve is ready for operation if value of RDY is TRUE.

Value of RDY is FALSE results in an external error.

Independent of operation mode of PCO_VALV.

REMOTE

Data type	Default value	Range	Unit
BOOL	TRUE	TRUE/FALSE	-

Allow valve to be controlled from SCADA.

Value of REMOTE is TRUE control from SCADA and function block is enabled.

Value of REMOTE is FALSE valve is controlled only from function block.

PCO_VALV will align the block according to feedback signals.

Independent of operation mode of PCO_VALV.

INHIB_ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Suppression of alarms.

If value of INHIB_ERR is TRUE, all alarms from PCO_VALV are suppressed.

Independent of operation mode of PCO_VALV.

EXT_ACK

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

External acknowledge of alarms.

If value of EXT_ACK is TRUE all alarms from PCO_VALV are acknowledged.

Independent of operation mode of PCO_VALV.

INHIB_FCT_ ERR_TIME

Data type	Default value	Range	Unit
TIME	TIME#5s	-	-

Maximum delay time from command to response from process.

If response is not received within this time limit, a function error will be generated.

Alarm_1

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 1.

If value of ALARM_1 is TRUE Alarm_1 is active.

Independent of operation mode of PCO_VALV.

Alarm_2

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 2.

If value of ALARM_2 is TRUE Alarm_2 is active.

Independent of operation mode of PCO_VALV.

Alarm_3

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 3.

If value of ALARM_3 is TRUE Alarm_3 is active.

Independent of operation mode of PCO_VALV.

Alarm_4

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 4.

If value of ALARM_4 is TRUE Alarm_4 is active.

Independent of operation mode of PCO_VALV.

Alarm_5

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 5.

If value of ALARM_5 is TRUE Alarm_5 is active.

Independent of operation mode of PCO_VALV.

Alarm_6

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Auxiliary alarm no. 6.

If value of ALARM_6 is TRUE Alarm_6 is active.

Independent of operation mode of PCO_VALV.

BEHAVE_ON_ERROR

Data type	Default value	Range	Unit
BYTE	0	0 ... 3	-

Actions that need to be executed by the FB at the event of error as set by the user.

The user can set a range of values from 0 ... 3.

BEHAVE_ON_ERROR = 0 causes the output to remain unaffected in case of a functional error or an external error (Not ready).

BEHAVE_ON_ERROR = 1 causes a stop command in case of a functional error.

BEHAVE_ON_ERROR = 2 causes a stop command in case of an external error.

BEHAVE_ON_ERROR = 3 causes a stop command in case of a functional error or an external error.

This parameter is independent of operation mode (whether Auto/ Manual) of PCO_VALV.

OUTPUT_HANDLING

Data type	Default value	Range	Unit
BYTE	1	0 ... 2	-

Behavior of command outputs, CMD_START / CMD_STOP.

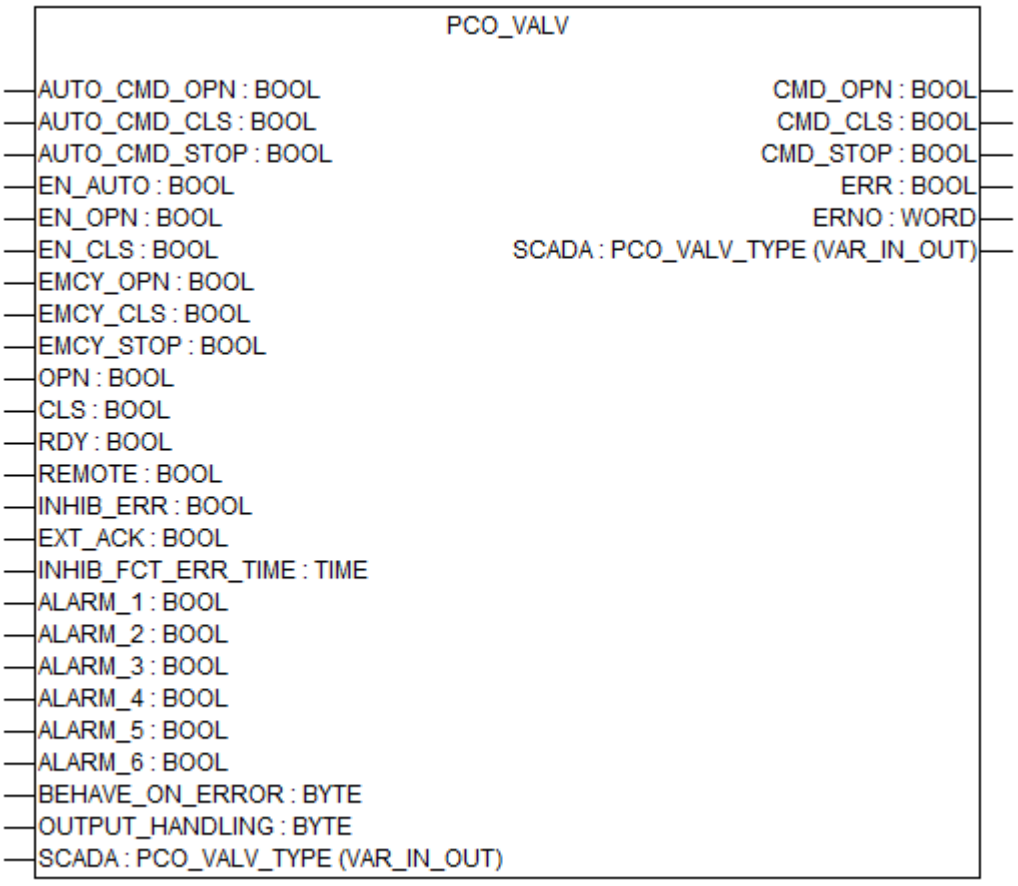
The user can set a range of values from 0 ... 2

OUTPUT_HANDLING = 0 causes the output (CMD_START / CMD_STOP) to be reset at RUN or STOP feedback.

OUTPUT_HANDLING = 1 causes the output (CMD_START / CMD_STOP)) to remain active at RUN or STOP feedback.

With OUTPUT_HANDLING = 2 the output (CMD_START / CMD_STOP) is performed as 1 s pulse. This parameter is independent of operation mode (whether Auto / Manual) of PCO_VALV.

Output description



CMD_OPN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output open to be connected to hardware output.

CMD_CLS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output close to be connected to hardware output.

CMD_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Command output stop, to be connected to hardware output.

ERR

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Common alarm, including:

- Functional error
- External error (input RDY)
- Input Parameter error
- ALARM_1
- ALARM_2
- ALARM_3
- ALARM_4
- ALARM_5
- ALARM_6
- EMCY_OPN
- EMCY_CLS
- EMCY_STOP

ERNO

Data type	Default value	Range	Unit
WORD	0	-	-

Error number

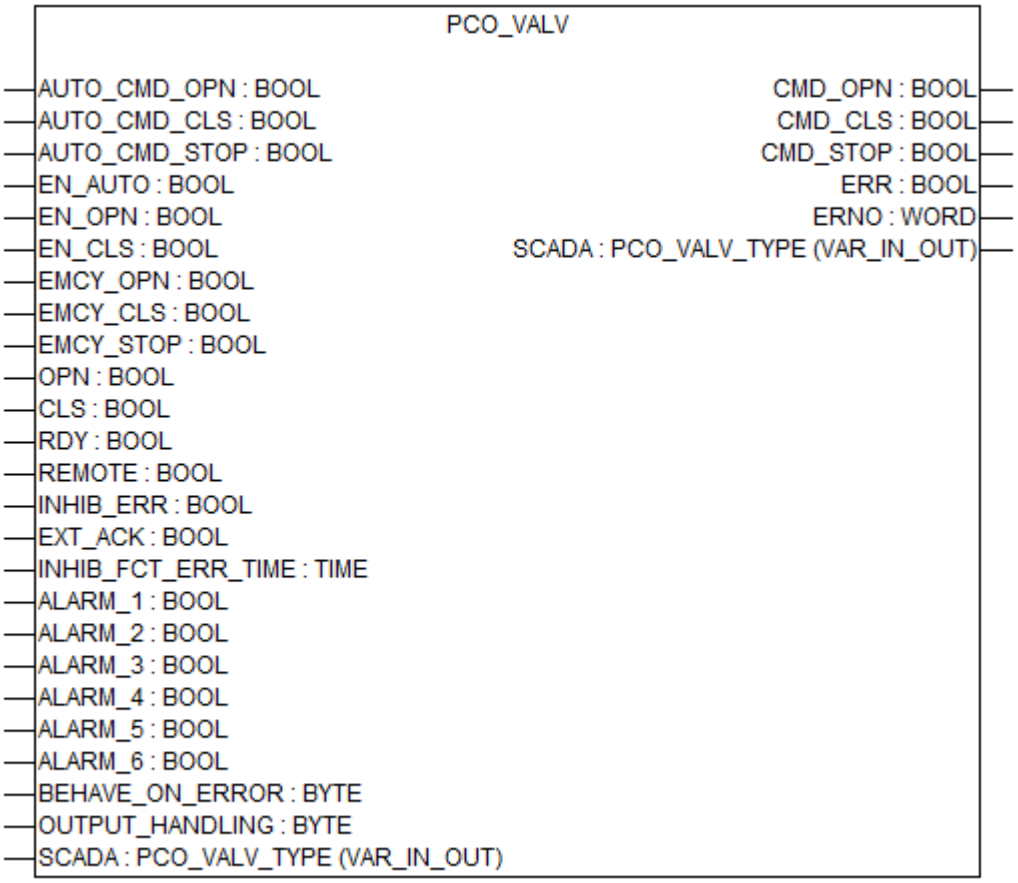
Output provides an error identifier if an invalid value was applied to an input.

ERNO always must be considered together with the output ERR.

The value output at ERNO is only valid if value of ERR is TRUE.

The error messages encoding is explained in *“Standard Function Block Libraries AC500”* in *“Error Messages of the Function Block Libraries”*.

Input/output description



SCADA

Data type	Default value	Range	Unit
PCO_VALV_TYPE	-	-	-

Structure variable for communication to and from SCADA system.

Parameter	Data type	Description	In-/Output
SCADA.CMDOPN	BOOL	Open command	Input
SCADA.CMDCLS	BOOL	Close command	Input
SCADA.CMDSTP	BOOL	Stop command	Input
SCADA.CMDAUT	BOOL	Auto command	Input
SCADA.CMDMAN	BOOL	Manual command	Input
SCADA.CMDRES	BOOL	Acknowledge active alarms	Input
SCADA.AUTO	BOOL	Valve in automatic mode	Output
SCADA.OPEN	BOOL	Valve is open	Output
SCADA.CLOSED	BOOL	Valve is closed	Output
SCADA.REMOTE	BOOL	Valve can be controlled from SCADA REMOTE = FALSE → Local operation	Output
SCADA.READY	BOOL	Valve is ready for operation	Output

Parameter	Data type	Description	In-/Output
SCADA.FNCERR	BOOL	Functional error	Output
SCADA.EXTERR	BOOL	External error (is generated when the valve is not ready)	Output
SCADA.AL1	BOOL	Auxiliary alarm no. 1	Output
SCADA.AL2	BOOL	Auxiliary alarm no. 2	Output
SCADA.AL3	BOOL	Auxiliary alarm no. 3	Output
SCADA.AL4	BOOL	Auxiliary alarm no. 4	Output
SCADA.AL5	BOOL	Auxiliary alarm no. 5	Output
SCADA.AL6	BOOL	Auxiliary alarm no. 6	Output
SCADA.OSMsg1 *)	WORD	Word representing the status of the PCO_VALV	Output
SCADA.OSMsg2 *)	WORD	Word representing the status of the PCO_VALV This includes the status of emergency status and alarm status.	Output
AspectObjectType _Valv_PLC	BOOL	(* Name of the 800xA Aspect Object type (AOT) to be used by the 800xA uploader, the live value is not used, only the name is relevant *)	Output

*) structure described separately

SCADA.OSMsg1

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_VALV.

Bit	Description
Bit 0	Valve is open
Bit 1	Valve is closed
Bit 2	Valve opening
Bit 3	Valve closing
Bit 4	External error (Not ready)
Bit 5	Functional error
Bit 6	Valve released for opening
Bit 7	Valve released for closing
Bit 8	Local operation (Not remote)
Bit 9	Common alarm (External alarm + Functional alarm + EMCY_OPN/EMCY_CLS/EMCY_STOP + ALARM_1 ... ALARM_6)
Bit 10	Unacknowledged alarm
Bit 11	Valve in automatic mode
Bit 12	Not used
Bit 13	Valve not released for automatic mode
Bit 14	Not used
Bit 15	Not used

SCADA.OSMsg2

Data type	Default value	Range	Unit
WORD	-	-	-

Word representing the status of the PCO_VALV.

This includes the status of emergency status and alarm status.

Bit	Description
Bit 0	Emergency open
Bit 1	Emergency close
Bit 2	Auxiliary alarm 1
Bit 3	Auxiliary alarm 2
Bit 4	Auxiliary alarm 3
Bit 5	Auxiliary alarm 4
Bit 6	Auxiliary alarm 5
Bit 7	Auxiliary alarm 6
Bit 8	Emergency stop
Bit 9	Not used
Bit 10	Not used
Bit 11	Not used
Bit 12	Not used
Bit 13	Not used
Bit 14	Not used
Bit 15	Not used

1.5.11 Solar library

1.5.11.1 Preconditions for the use of the Solar_AC500 library

PS562-SOLAR libraries are working on AC500 and AC500-eCo, but depending on the application, number of axes and its total program, memory restrictions on the AC500-eco can occur.

For the highly accurate NREL algorithm, meant for focusing systems with high concentration factor, PM573 or larger should be chosen.

The included example programs "Example_Solar_2Axis_ACS3XX.project" (uses NREL) and "ExamplePM564_NOAA_Solar_2Axis_ACS3XX.project" show complete examples for different accuracies and CPU families, see example program description pdf.

The function blocks of PS562-SOLAR libraries are usable and have been tested for AC500 control systems with a runtime system of version V1.3 and above.

The libraries have been tested with PM564, PM573, PM583, PM592.

The function blocks of the solar library are only working in RUN mode of the PLC. Usage of these libraries in the simulation mode will not provide always valid or usable behavior/values.

User needs to add DC541_AC500_V11 library along with Solar_AC500_V22 Library.

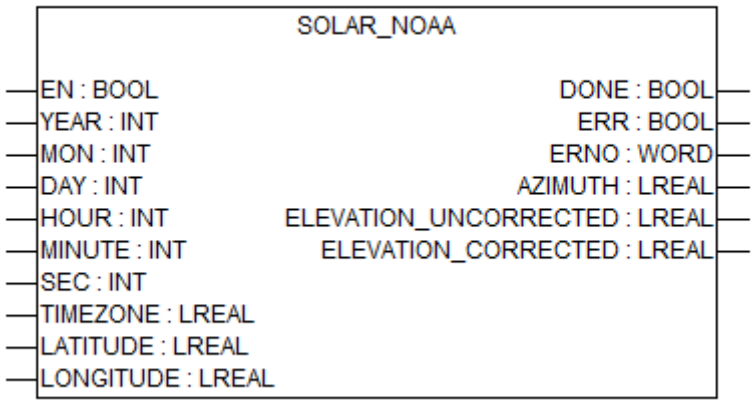
The above named examples need additionally the ACS Drives Libraries from the PS553-DRIVES package, which can be downloaded from www.abb.com/PLC:

- On right side menu under heading "Your preferences": Select "English" as language ... (country doesn´t matter).
- On right side menu under "More Info Links" : Click on "PS501 Updates".
- Select "PS501-UPDA: PS553-DRIVES…" to download *.zip file.

1.5.11.2 SOLAR_AC500 library

1.5.11.2.1 TRACK folder

SOLAR_NOAA



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function blocks without historical values.
Group	Package of functions to get the position of the sun

This function block calculates the position of the sun, elevation and azimuth, according to: date, time and location, with an azimuth error = ± 0.01° and elevation error = ±0.03°. This function embeds the SOLAR_NOAAs algorithm.

Output parameters are solar elevation and solar azimuth. These outputs are the topocentric coordinates of the sun which use the observer's location as a centre of the coordinate system.

The end-user may use the RTC of the AC500 system by using the CLOCK function block of the library named SysExt_AC500_V10.lib or may use any other source to calculate the position of the sun.

Table 195: Size of used data for SOLAR_NOAA function block and execution time for each CPU type

	PM583	PM591	PM564
Data Size	4299 bytes	4299 bytes	4299 bytes
Program Size	20622 bytes	10742 bytes	20834 bytes
Execution Time	3 ms	1 ms	5 ms

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

YEAR INT (year) Range of values: from 2000 to 6000.

MON INT (month) Range of values: from 1 to 12.

DAY INT (day) Range of values: from 1 to 31.

HOUR INT (hour) Range of values: from 0 to 23.

MINUTE INT (minute) Range of values: from 0 to 59.

SEC INT (second) Range of values: from 0 to 59.

TIMEZONE LREAL (time zone) Observer time zone (negative west of Greenwich).

Time zone = Standard Time - Universal Time

Format: hours.

Range of values: -12...12.

Resolution: 0.1 hour.

LATITUDE LREAL (latitude) Observer latitude (negative south of equator).

Format: degrees.

Range of values: from -90.00 to 90.00.

LONGITUDE LREAL (longitude) Observer longitude (negative west of Greenwich).

Format: degrees.

Range of values: from -180.00 to 180.00.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

AZIMUTH LREAL (azimuth)

Topocentric azimuth. Eastward from north (0°=360°=north, 90°=east, 180°=south, 270°=west).

Format: degrees.

Range of values: from 0 to 359.9999.

Error values:

- Northern hemisphere: area between latitudes from 70° to 24°. In this area the accuracy of function SOLAR_NOAA is ±0,02° azimuth.
- Southern hemisphere: area between latitudes from -70° to -24°. In this area the accuracy of function SOLAR_NOAA is ±0,08° azimuth.
- Equator: area between 24° and -24°. In this area the accuracy of function SOLAR_NOAA is 0,08° azimuth. Note that in this area, around 12:00 h (solar time) maximum error values may be greater because of the high speed of the azimuth axis at 12.00 h (solar time). It must move from 90° to 270° in a few seconds.

ELEVATION_UNCORRECTED LREAL (elevation_uncorrected)

Topocentric elevation angle (0°=sunrise, 90°=zenith).

Format: degrees.

Range of values: -90.00...90.00. Negative values=darkness.

Elevation error: ±0.015°.

If elevation goes through zero at special locations at sunset / sunrise, also slightly larger errors can occur (mind that energy production is close to zero anyway).

For latitudes greater than 72° north or less than 72° south, accuracy may be lower due in part to the effects of atmospheric refraction.

ELEVATION_CORRECTED LREAL (elevation_corrected)

Topocentric elevation angle (0°=sunrise, 90°=zenith). This value includes atmospheric refraction effects.

Format: degrees.

Range of values: -90.00...90.00. Negative values=darkness.

Elevation Error: ±0.015°.

If elevation goes through zero at special locations at sunset / sunrise, also slightly larger errors can occur (mind that energy production is close to zero anyway).

P = 1010 mBar.

Temp = 10°C.

Refraction = 0,5667.

Atmospheric refraction is the angular displacement of astronomical objects from their true or geometrical position, because of the bending of rays in the earth's atmosphere.

Function call in IL

CAL mySOLAR_NOAA (		
	EN	:= mySOLAR_NOAA_EN,
	YEAR	:= mySOLAR_NOAA_YEAR,
	MON	:= mySOLAR_NOAA_MON,
	DAY	:= mySOLAR_NOAA_DAY,
	HOUR	:= mySOLAR_NOAA_HOUR,
	MINUTE	:= mySOLAR_NOAA_MINUTE,
	SEC	:= mySOLAR_NOAA_SEC,
	TIMEZONE	:= mySOLAR_NOAA_TIME-ZONE,
	LATITUDE	:= mySOLAR_NOAA_LATI-TUDE,
	LONGITUDE	:= mySOLAR_NOAA_LONGI-TUDE)
LD	mySOLAR_NOAA.DONE	
ST	mySOLAR_NOAA_DONE	
LD	mySOLAR_NOAA.ERR	
ST	mySOLAR_NOAA_ERR	
LD	mySOLAR_NOAA.ERNO	
ST	mySOLAR_NOAA_ERNO	
LD	mySOLAR_NOAA.AZIMUTH	
ST	mySOLAR_NOAA_AZIMUTH	

LD	mySOLAR_NOAA.ELEVATION_UNCORRECTED
ST	mySOLAR_NOAA_ELEVATION_UNCORRECTED
LD	mySOLAR_NOAA.ELEVATION_CORRECTED
ST	mySOLAR_NOAA_ELEVATION_CORRECTED

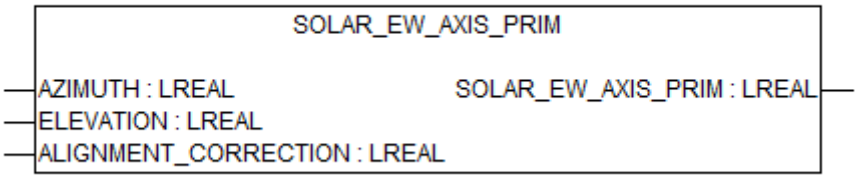
In IL, the function call has to be written in one line.

Function call in ST

mySOLAR_NOAA (		
	EN	:= mySOLAR_NOAA_EN,
	YEAR	:= mySOLAR_NOAA_YEAR,
	MON	:= mySOLAR_NOAA_MON,
	DAY	:= mySOLAR_NOAA_DAY,
	HOUR	:= mySOLAR_NOAA_HOUR,
	MINUTE	:= mySOLAR_NOAA_MINUTE,
	SEC	:= mySOLAR_NOAA_SEC,
	TIMEZONE	:= mySOLAR_NOAA_TIME-ZONE,
	LATITUDE	:= mySOLAR_NOAA_LATI-TUDE,
	LONGITUDE	:= mySOLAR_NOAA_LONGI-TUDE);
mySOLAR_NOAA_DONE		:= mySOLAR_NOAA.DONE;
mySOLAR_NOAA_ERR		:= mySOLAR_NOAA.ERR;
mySOLAR_NOAA_ERNO		:= mySOLAR_NOAA.ERNO;
mySOLAR_NOAA_AZIMUTH		:= mySOLAR_NOAA.AZI-MUTH;
mySOLAR_NOAA_ELEVATION		:= mySOLAR_NOAA.ELEVA-TION_UNCORRECTED;
mySOLAR_NOAA_ELEVATION_REFRACTION		:= mySOLAR_NOAA.ELEVA-TION_CORRECTED;

1.5.11.2.2 AXIS folder

SOLAR_EW_AXIS_PRIM



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Functions
Group	Package of functions to get the axis angle of the tracker to follow the sun

This function block calculates the angle of east to west axis according to: azimuth, elevation and correction alignment factor.

Input description

AZIMUTH LREAL (azimuth)

Topocentric azimuth angle. Eastward from north (0°=360°=north, 90°=east, 180°=south, 270°=west).

Format: degrees.

Range of values: from 0 to 359.9999.

ELEVATION LREAL (elevation)

Topocentric elevation angle (0°=sunrise, 90°=zenith).

Format: degrees.

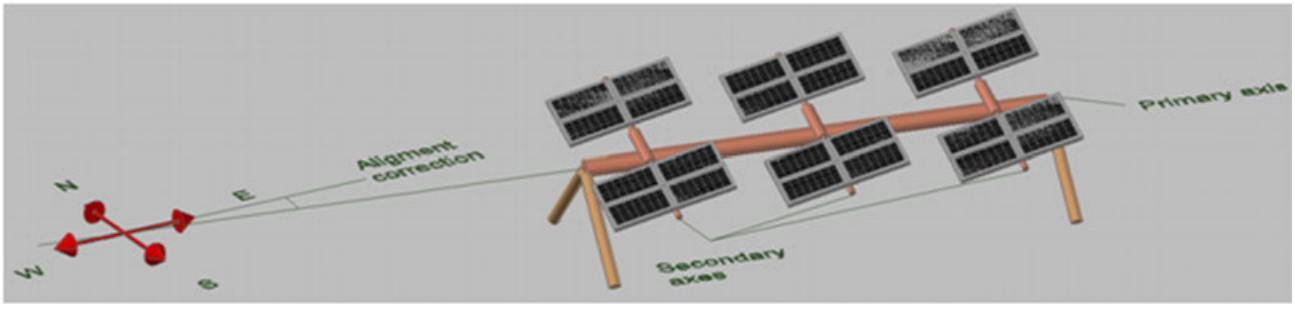
Range of values: -90.00...90.00. Negative values=darkness.

ALIGNMENT_CORRECTION LREAL (alignment correction)

Angle between system axis and east (clockwise).

Format: degrees.

Range of values: -180.00...180.00.

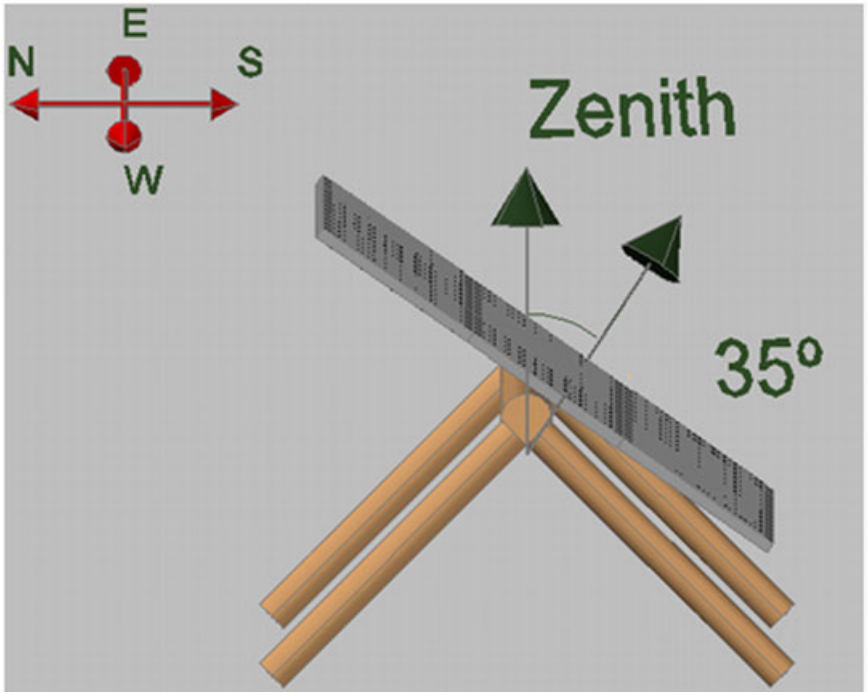


Output description

(Output) LREAL

Angle of east to west axis (0°=zenith, -90°=north, 90°=south).

Format: degrees.



Function call in IL

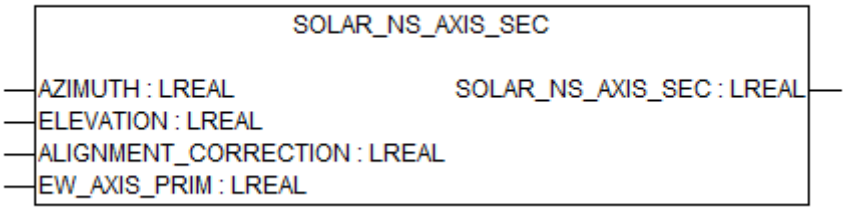
LD	EWAxis_AZIMUTH
SOLAR_EW_AXIS_PRIM	EWAxis_ELEVATION, EWAxis_ALIGN-MENT_CORRECTION
ST	EWAxis_ANGLE

In IL, the function call has to be written in one line.

Function call in ST

EWAxis_ANGLE := SOLAR_EW_AXIS_PRIM(		
	AZIMUTH	:= EWAxis_AZIMUTH,
	ELEVATION	:= EWAxis_ELEVATION,
	ALIGNMENT_CORRECTION	:= EWAxis_ALIGN-MENT_CORRECTION);

SOLAR_NS_AXIS_SEC



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the axis angle of the tracker to follow the sun

This function assumes the east to west axis as the main system axis.

This function provides the angle of secondary axis for solar tracking, in two axes, according to: elevation, azimuth, correction alignment factor and angle of main axis. This function is valid when east to west axis is acting as the main axis.

Output parameter is the angle of secondary axis.

Input description

AZIMUTH LREAL (azimuth)

Topocentric azimuth angle. Eastward from north (0°=360°=north, 90°=east, 180°=south, 270°=west).

Format: degrees.

Range of values: from 0 to 359.9999.

ELEVATION LREAL (elevation)

Topocentric elevation angle (0°=sunrise, 90°=zenith).

Format: degrees.

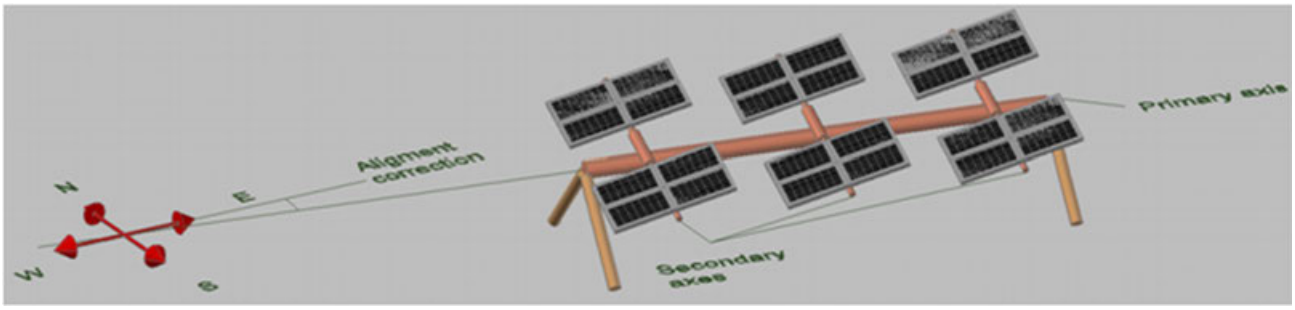
Range of values: -90.00...90.00. Negative values=darkness.

ALIGNMENT_CORRECTION LREAL (alignment correction)

Angle between system axis and east (clockwise).

Format: degrees.

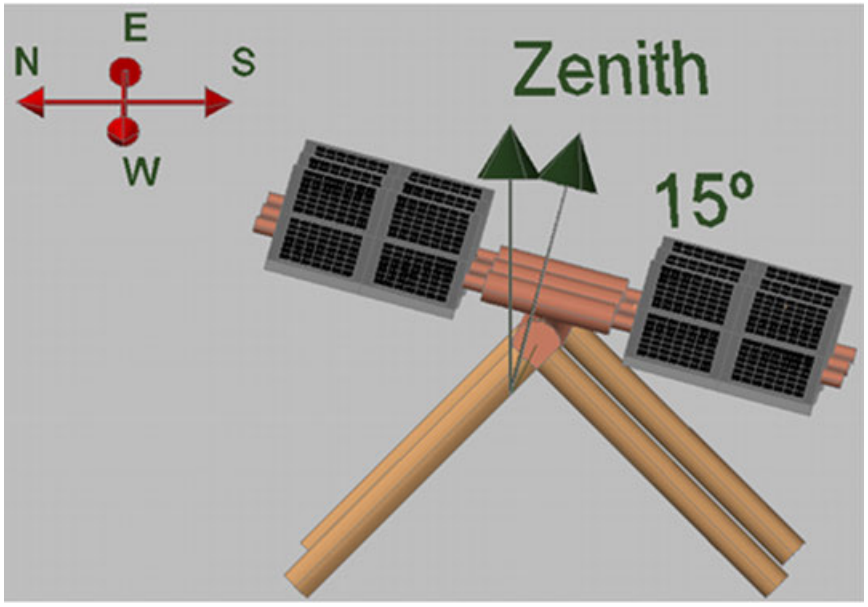
Range of values: -180.00...180.00.



SOLAR_EW_AXIS_PRIM LREAL (angle of east to west axis)

Angle of main axis (0°=zenith, -90°=north, 90°=south).

Format: degrees.



Output description

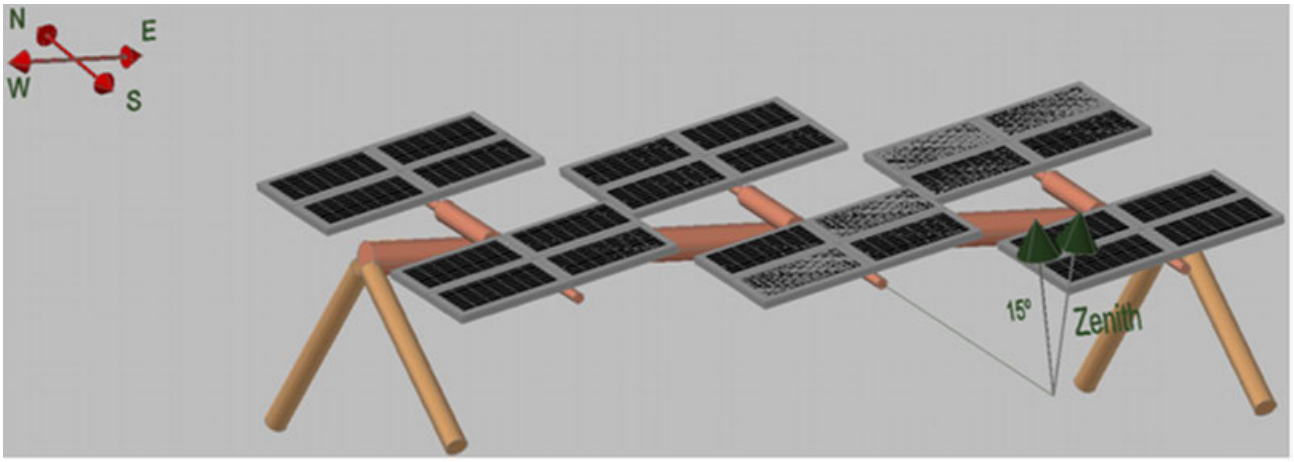
(Output) LREAL

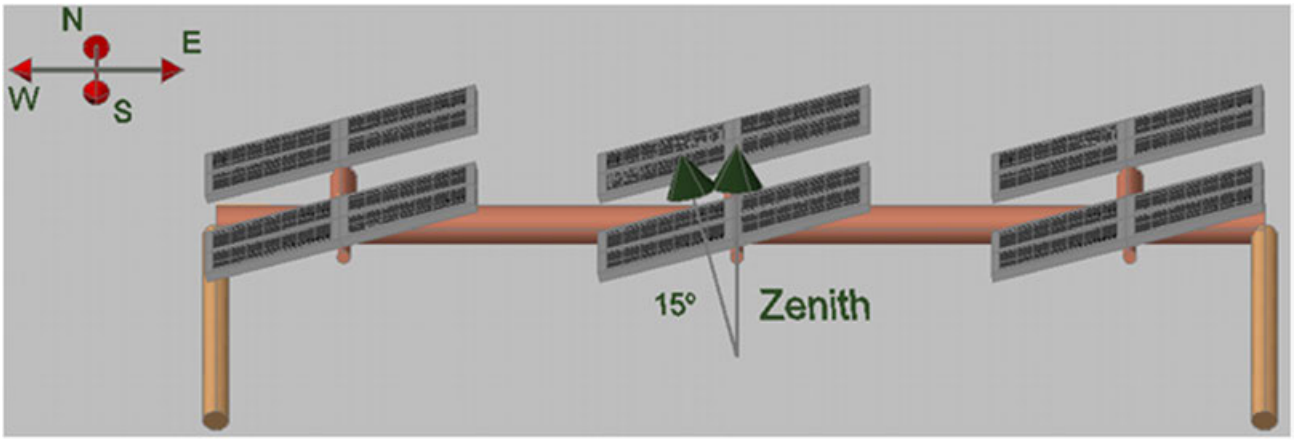
The block output is the angle of secondary axis north "“ south, where zenith indicates 0° position.

Format: degrees.

Range of values: -180.00...180.00.

Negative value indicates clockwise rotation about north "“ south axis. Otherwise, positive value indicates anticlockwise rotation about the north "“ south axis.





Function call in IL

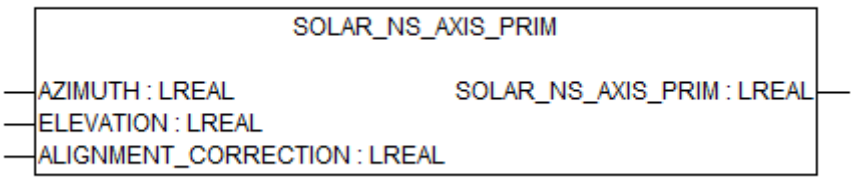
LD	NSAxis2_AZIMUTH
SOLAR_NS_AXIS_SEC	NSAxis2_ELEVATION, NSAxis2_ALIGN-MENT_CORRECTION, NSAxis2_EW_AXIS
ST	NSAxis2_ANGLE

In IL, the function call has to be written in one line.

Function call in ST

NSAxis2_ANGLE := SOLAR_NS_AXIS_SEC(	
AZIMUTH	:= NSAxis2_AZIMUTH,
ELEVATION	:= NSAxis2_ELEVATION,
ALIGNMENT_CORRECTION	:= NSAxis2_ALIGNMENT_CORRECTION,
SOLAR_EW_AXIS_PRIM	:= NSAxis2_EW_AXIS);

SOLAR_NS_AXIS_PRIM



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib

Type	Function
Group	Package of functions to get the axis angle of the tracker to follow the sun

Function block to calculate the angle of north to south axis according to: azimuth, elevation and correction alignment factor.

Output parameter is the angle of north to south axis.

Input description

AZIMUTH LREAL (azimuth)

Topocentric azimuth angle. Eastward from north (0°=360°=north, 90°=east, 180°=south, 270°=west).

Format: degrees.

Range of values: from 0 to 359.9999.

ELEVATION LREAL (elevation)

Topocentric elevation angle (0°=sunrise, 90°=zenith).

Format: degrees.

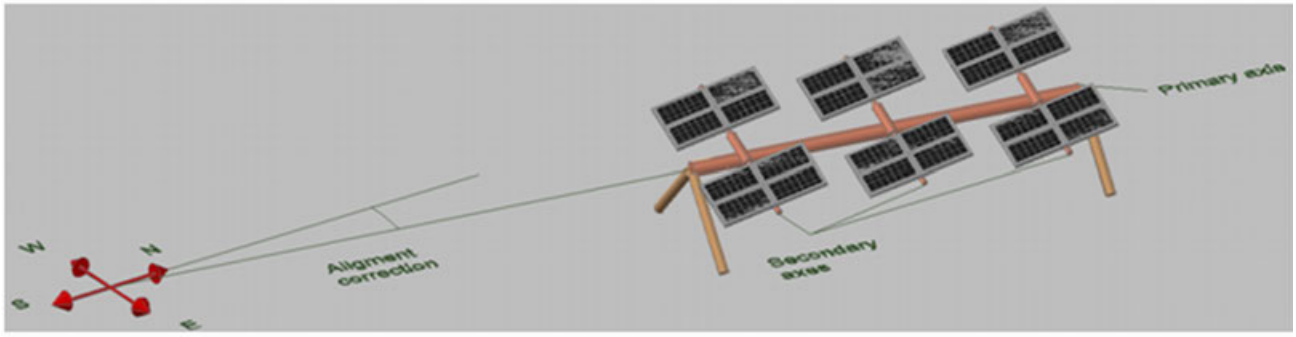
Range of values: -90.00...90.00. Negative values=darkness.

ALIGNMENT_CORRECTION LREAL (alignment correction)

Angle between system axis and north (clockwise).

Format: degrees.

Range of values: -180.00...180.00.

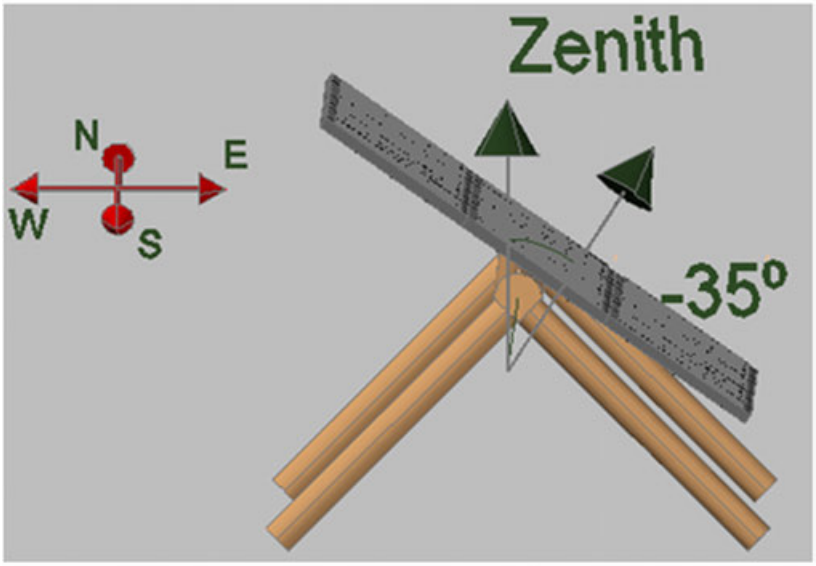


Output description

(Output) LREAL

Angle of north to south axis (0°=zenith, -90°=east, 90°=west).

Format: degrees.



Function call in IL

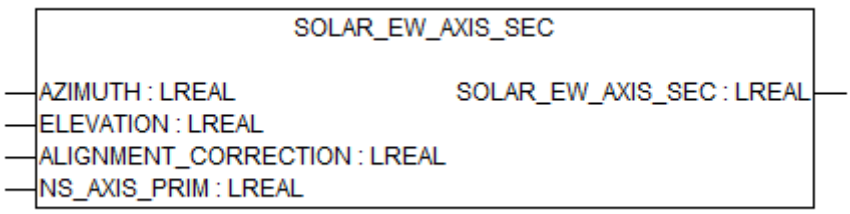
LD	NSAxis_AZIMUTH
SOLAR_NS_AXIS_PRIM	NSAxis_ELEVATION, NSAxis_ALIGN-MENT_CORRECTION
ST	NSAxis_ANGLE

In IL, the function call has to be written in one line.

Function call in ST

NSAxis_ANGLE := SOLAR_NS_AXIS_PRIM(		
	AZIMUTH	:= NSAxis_AZIMUTH,
	ELEVATION	:= NSAxis_ELEVATION,
	ALIGNMENT_CORRECTION	:= NSAxis_ALIGN-MENT_CORRECTION);

SOLAR_EW_AXIS_SEC



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the axis angle of the tracker to follow the sun

This function assumes the north to south axis as the main system axis.

This function provides the angle of secondary axis for solar tracking, in two axes, according to: azimuth, elevation, correction alignment factor and angle of main axis. This function is valid when north to south axis is acting as the main axis.

Output parameter is the angle of secondary axis.

Block type Function

Input description

AZIMUTH LREAL (azimuth) Topocentric azimuth angle. Eastward from north (0°=360°=north, 90°=east, 180°=south, 270°=west).

Format: degrees.

Range of values: from 0 to 359.9999.

ELEVATION LREAL (elevation) Topocentric elevation angle (0°=sunrise, 90°=zenith).

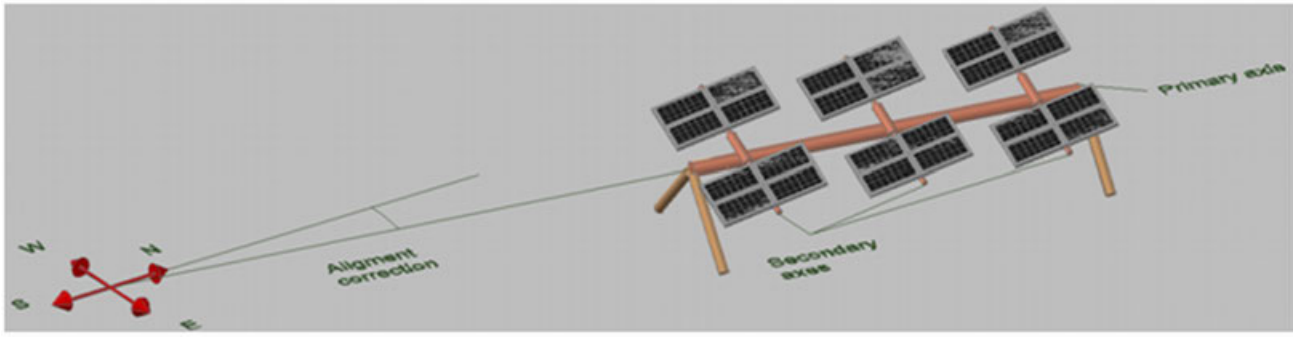
Format: degrees.

Range of values: -90.00...90.00. Negative values=darkness.

ALIGNMENT_CORRECTION LREAL (alignment correction) Angle between system axis and north (clockwise).

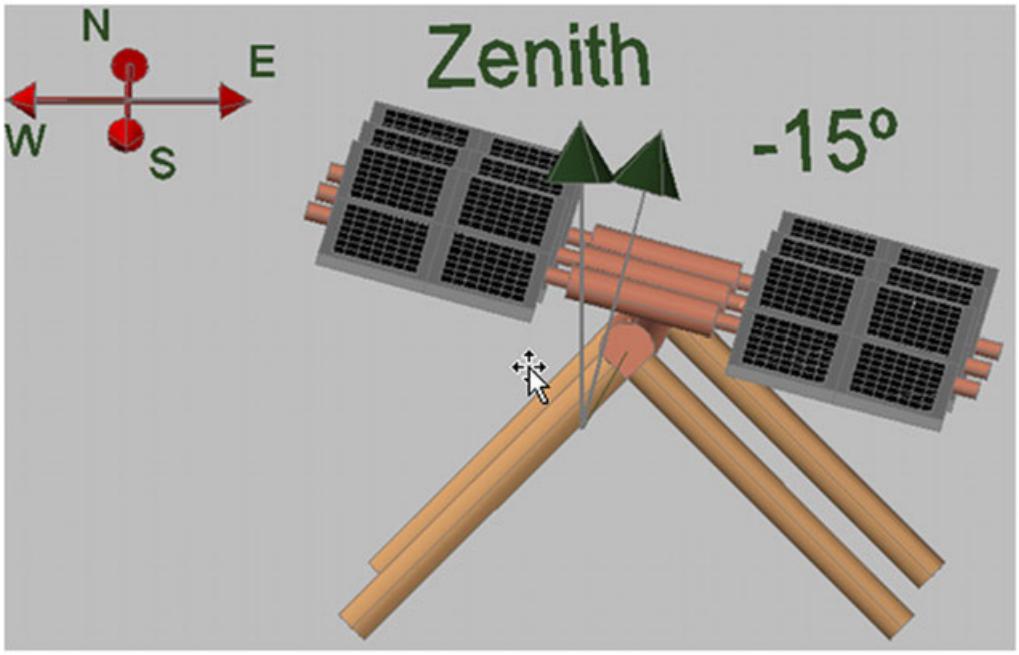
Format: degrees.

Range of values: -180.00...180.00.



SOLAR_NS_AXIS_PRIM LREAL (angle of north to south axis) Angle of main axis (0°=zenith, -90°=east, 90°=west).

Format: degrees.



Output description

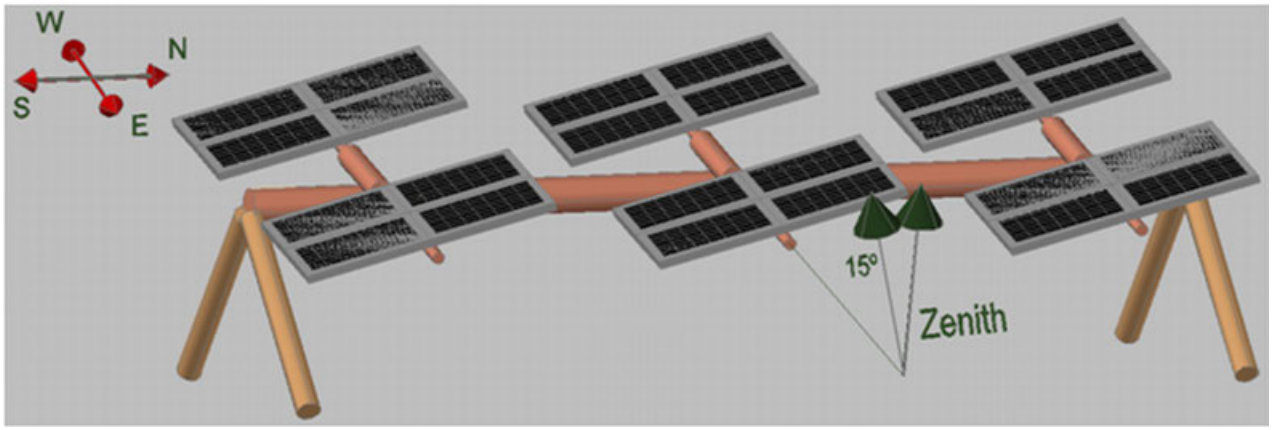
(Output)
LREAL

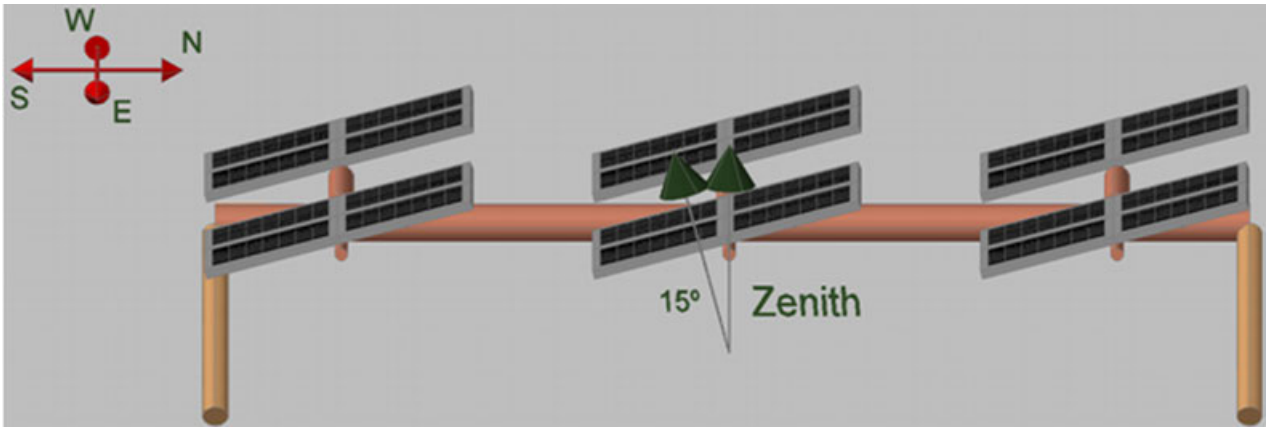
The block output is the angle of secondary axis east "“ west, where zenith indicates 0° position.

Format: degrees.

Range of values: -180.00...180.00.

Negative value indicates clockwise rotation about the east "“ west axis. Otherwise, positive value indicates anticlockwise rotation about the east "“ west axis.





Function call in IL

LD	EWAxis2_AZIMUTH
SOLAR_EW_AXIS_SEC	EWAxis2_ELEVATION, EWAxis2_ALIGN-MENT_CORRECTION, EWAxis2_NS_AXIS
ST	EWAxis2_ANGLE

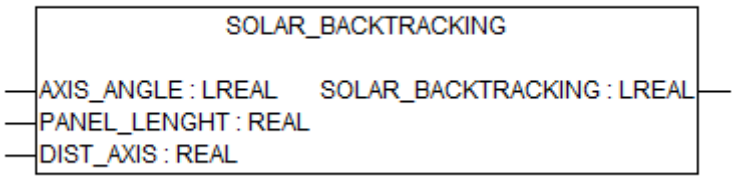
In IL, the function call has to be written in one line.

Function call in ST

EWAxis2_ANGLE := SOLAR_EW_AXIS_SEC(		
	AZIMUTH	:= EWAxis2_AZIMUTH,
	ELEVATION	:= EWAxis2_ELEVATION,
	ALIGNMENT_CORRECTION	:= EWAxis2_ALIGN-MENT_CORRECTION,
	SOLAR_NS_AXIS_PRIM	:= EWAxis2_NS_AXIS);

1.5.11.2.3 BACKTRACK folder

SOLAR_BACKTRACKING



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the optimum tracker position to avoid shadows

This function calculates the optimum tracker position to reduce shadowing by optimizing solar panel position, when position of the sun gets low (early morning + late evening). Axis angle is calculated according to: axis angle for solar tracking in one axis, calculated regardless shadows, distance between rows of Solar Trackers and height of them.

Output parameter is the optimum axis angle which avoids shadowing between rows of Solar Trackers.

Input description

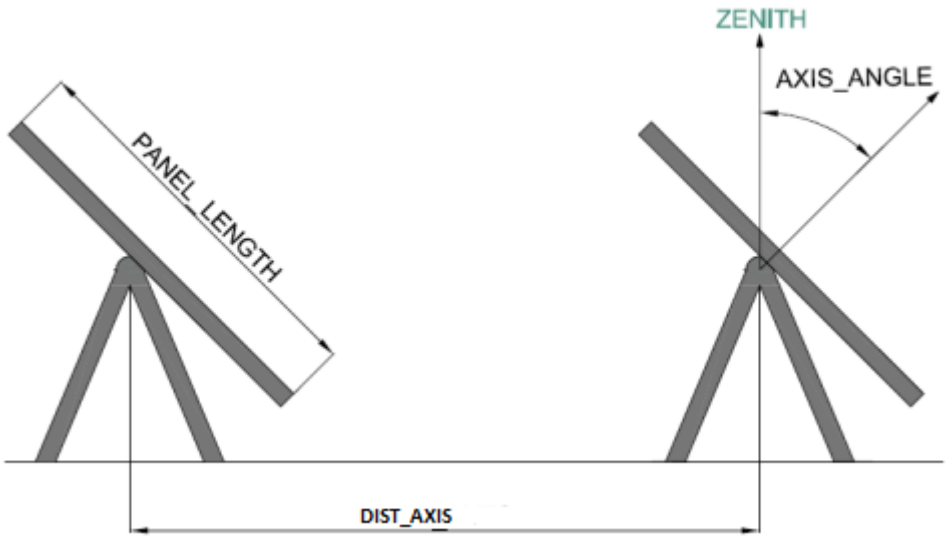
AXIS_ANGLE LREAL (axis angle)
Angle of main axis (0°=zenith, anticlockwise).
Format: degrees.

PANEL_LENGTH REAL (panel length)
Value that indicates the total length of panels.
Format: meters.

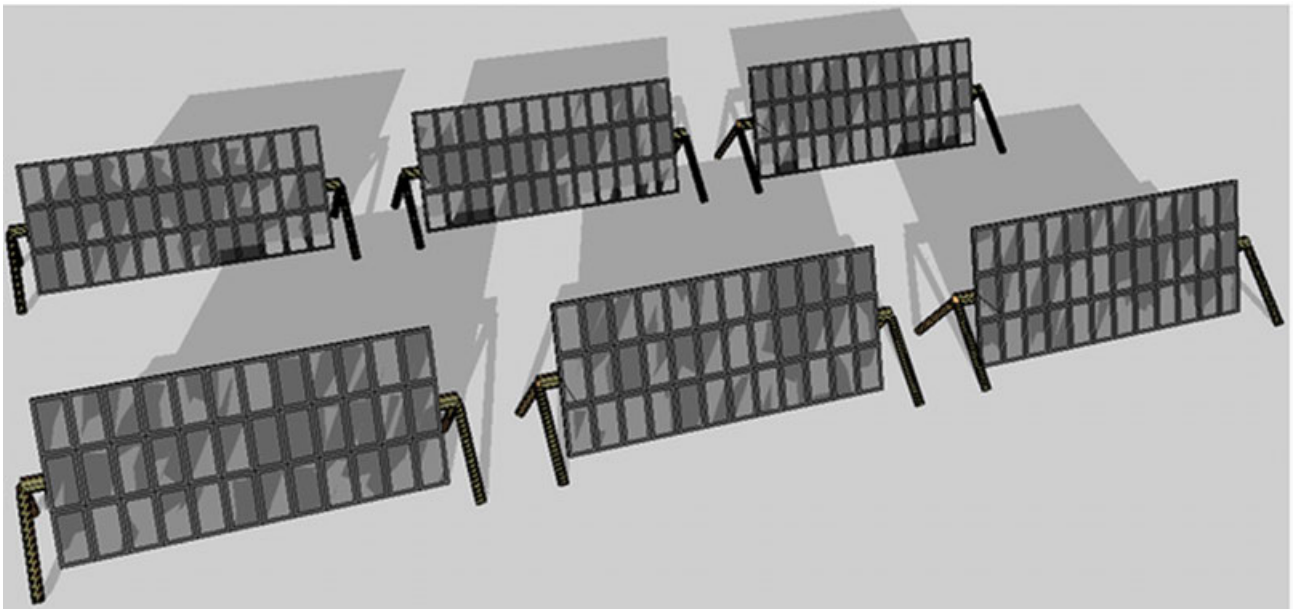
DIST_AXIS REAL (distance axis)
Distance between rows of Solar Trackers.
Format: meters.

Output description

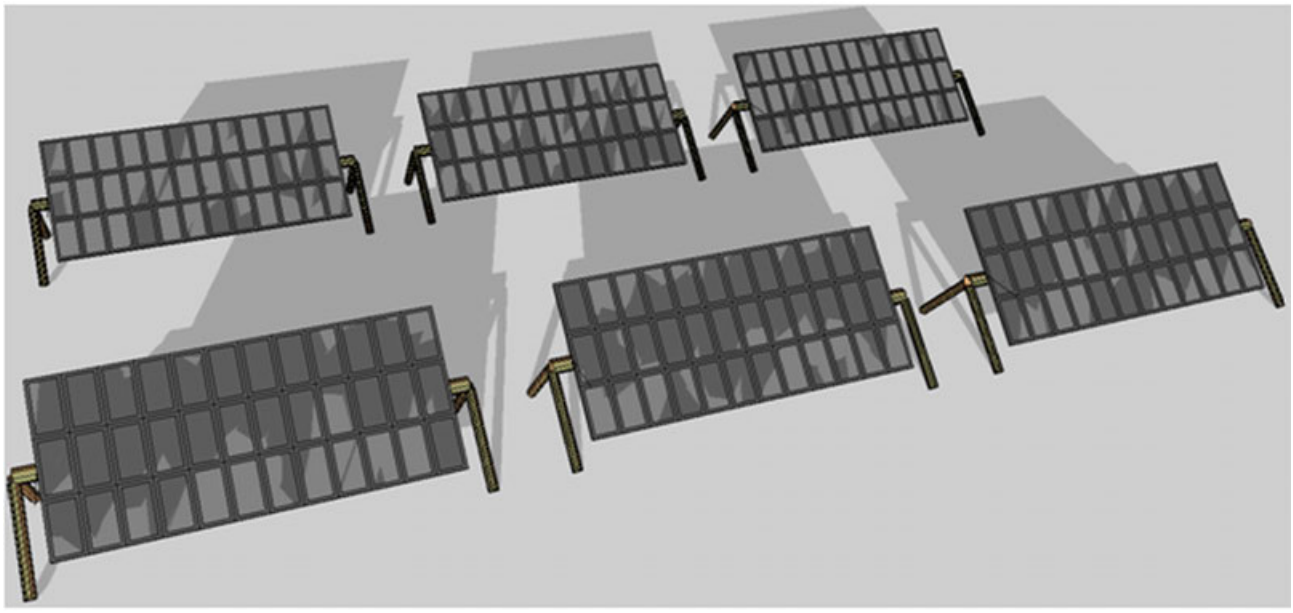
(Output) REAL
The block output is the optimum axis angle which avoids shadowing between rows of Solar Trackers.
Format: degrees.
Dimensional features:



Example:



Shadow effects, early morning, on north to south single axis trackers positioned with the angle calculated regardless shadows, using SOLAR_NS_AXIS_PRIM.



Position of the same tracker at the same date, time and location avoiding shadows between rows of trackers by correcting the axis angle with the function block SOLAR_BACKTRACKING.

Function call in IL

```
LD
Backtracking_AXIS_ANGLE
SOLAR_BACKTRACKING                Backtracking_PANEL_LENGHT,
Backtracking_DIST_AXIS
ST
            Backtracking_ANGLE
```

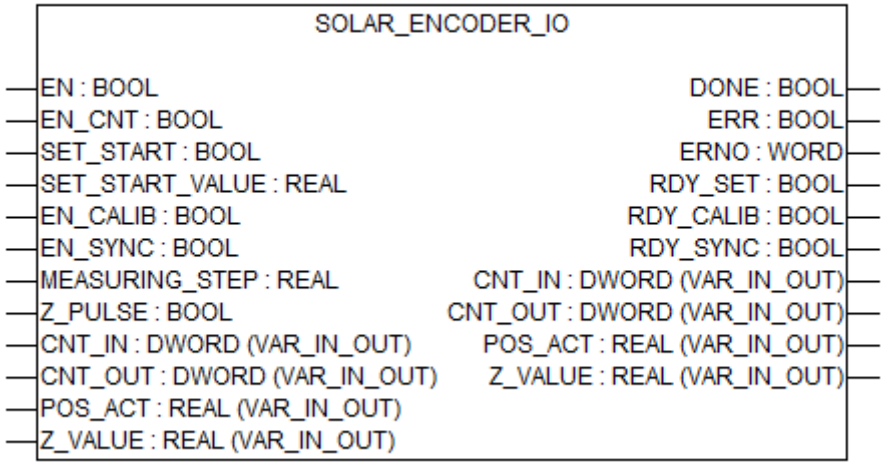
In IL, the function call has to be written in one line.

Function call in ST

```
Backtracking_ANGLE := SOLAR_BACKTRACKING (

AXIS_ANGLE                                          :=
SOLAR_BACKTRACKING_AXIS_ANGLE,
                    PANEL_HEIGHT                                    :=
SOLAR_BACKTRACKING_PANEL_LENGHT,

DIST_AXIS                                             :=
SOLAR_BACKTRACKING_DIST_AXIS);
```

1.5.11.2.4 POSITION folder

SOLAR_ENCODER_IO



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block with historical values.
Group	Package of functions to get the position of the sun

The SOLAR_ENCODER_IO is used to easily integrate, with any S500 I/O card, Incremental encoders (A, B) plus inductive proximity sensor (Z) as positioning sensors for Solar Trackers.

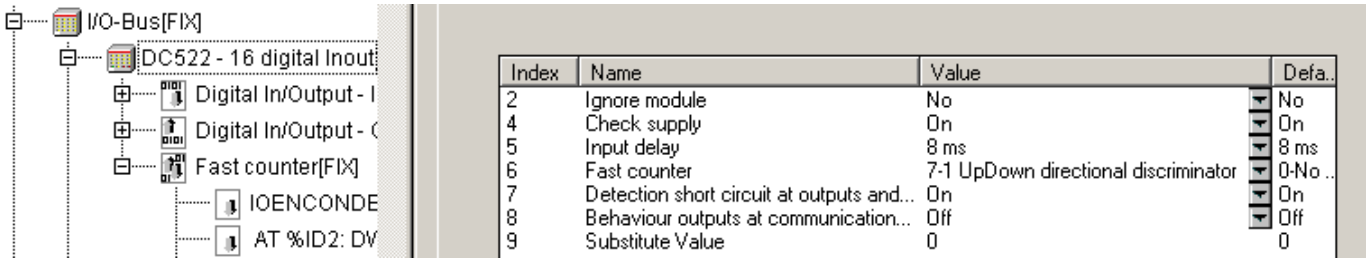
This function uses the fast counters which have several I/O modules to integrate Incremental encoders. The counting modules with fast counters, and its features, are described in ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

> *The functionality 'Fast Counter' only works with communication interface modules which are mounted at the I/O bus of an AC500 CPU. An exception is the CS31 communication interface module DC551-CS31, which contains a fast counter that is made operationally by the address setting on the module.*

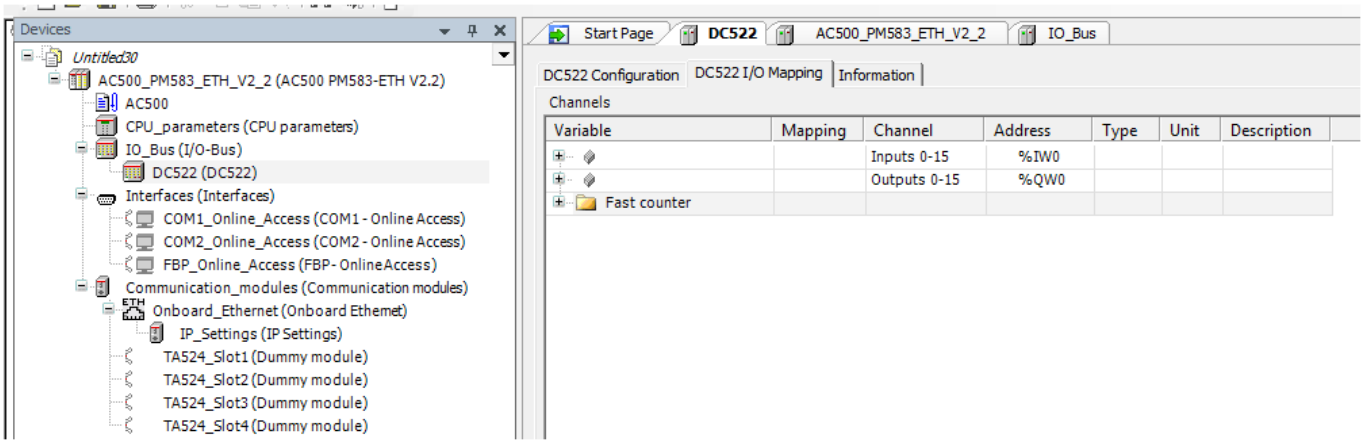
The I/O modules on the I/O bus have two inputs of fast counters per module, which are activated via software in the PLC configuration. Each module counter can be configured up to 10 possible modes. After that, it is activated during the initialization phase (power-on, cold start, warm start).

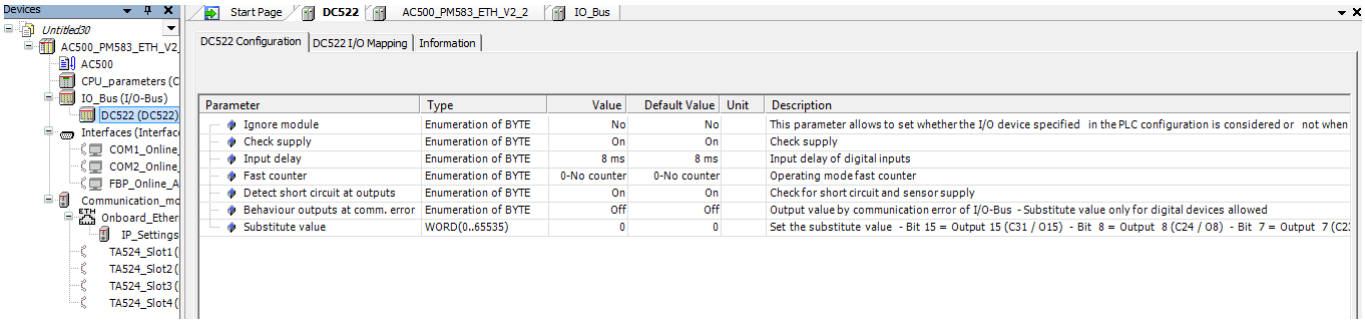
The function block SOLAR_ENCODER_IO should be used configuring the module parameter "Fast counter" with the operating modes 7, 9 or 10, to determine the counting procedure in x1, x2 or x4 count respectively. The different meanings of each fast counter mode are explained down below:

Example: Configuration of module DC522 to use the fast counter to integrate an Incremental encoder in x1 mode:



In Contol Builder Plus V2.0 and above:





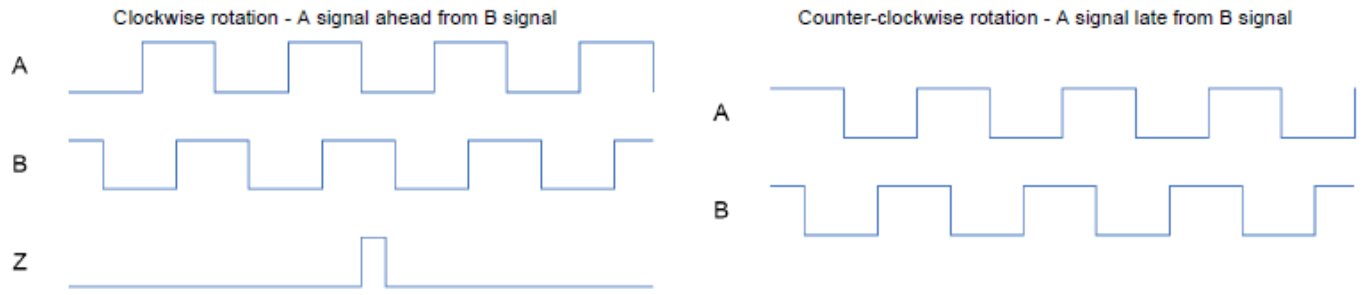
Fast counter: "7-1 UpDown directional discriminator"

The x1 counting mode is used. The encoder module discriminates the rotating way and count one pulse for each rising edge of the A signal.

In this mode the maximum counting frequency is 35 kHz.

The rotation is identified with a shift angle (90°) between A and B signal. In the I/O module, the clockwise rotation is identified with A signal in advance to B (see figure below).

Example: Direction identification using A and B signals:



Fast counter: "9-1 UpDown directional discriminator X2"

The x2 counting mode is used. The module counts both the positive edges and the negative edges of trace A.

The result is the double number of counting pulses, so the precision increases correspondingly.

In this operating mode, the maximum counting frequency is 30 kHz.

Fast counter: "10-1 UpDown directional discriminator X4"

The x4 counting mode is used. The counter counts the positive and negative edges of the traces A and B. In this operating mode, the maximum counting frequency is 15 kHz.

The I/O modules provide 2 fast counter inputs to use for relative positioning with 2 signals (A, B).

> *To acquire the Z signal, a conventional input channel should be used.*

Due to that, the Z signal will be set at least:

CPU	Minimum time to warranty the Z signal will be acquire
<583	T>10 ms
>590	T>1 ms

This table represents the time necessary to read the Z signal, using a task configured at the maximum frequency for each CPU.

The following table shows the S500 modules that contain a fast counter and which of the digital inputs are reserved for the counter.

MODULE	ASIGNED INPUTS	
	CHANNEL A	CHANNEL B
DC522	C8	C9
DC523	C16	C17
DC532	C24	C25
DC524	I24	I25
DX522	I0	I1
DC531-CS31	C16	C17

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

EN_CNT BOOL (enable counter)

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and pulses are not stored even the encoder device changes its position.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

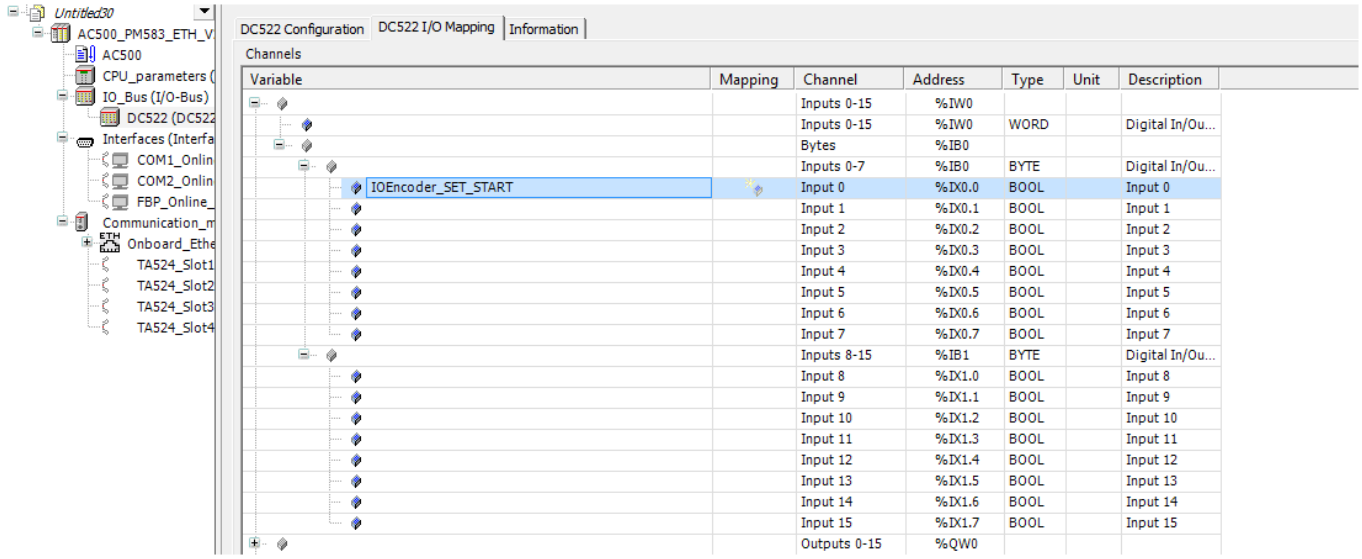
SET_START BOOL (set start)

If set input SET_START = TRUE, the counter takes the values from input SET_START_VALUE to transfer it to ACT. As long as input SET_START = TRUE, no pulses are counted because the counter is always overwritten by the input SET_START_VALUE.

To synchronize the counter value with the mechanical zero reference based on signal Z, a conventional input channel should be used to read the Z signal as follows:

Example: Use of conventional input channel to acquire the reference signal:

In Control Builder Plus V2.0 and above:



When the input channel which represents the Z channel is activated, the SET_START goes to true, and ACT goes to the value which is indicated by SET_START_VALUE.

SET_START_VALUE REAL (set start value)

The counter can be set to a Start value. This value must be applied to the input SET_START_VLAUE.

If input SET_START=TRUE, the counter takes this value.

EN_CALIB BOOL (enabling calibrate)

If EN_CALIB = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output Z_VALUE will take the stored value in POS_ACT and output RDY_CALIB is set to TRUE. If this input is continuously set to TRUE, only one calibrating process will be done even system reaches encoder´s reference point (Z) once again. In order to enable another calibrating process, input EN_CALIB has to be set to FALSE and set to TRUE again.

Calibrating process is executed out of CPU cycle, and it is able to detect the multiple rising edges that appear when a simple limit switch is used. Due to that, it is recommended to use devices assembled especially for this purpose (high accuracy limit switch).

Inputs used as reference point Z in the calibrating process are:

Counter0 -> I3

Counter1 -> I11

EN_SYNC BOOL (enabling synchronize)

If input EN_SYNC = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output POS_ACT will take the value stored at Z_VALUE, output RDY_SYNC is set to TRUE only during a program cycle. While EN_SYNC is set to TRUE, POS_ACT output will take the value stored at Z_VALUE every time that a rising edge is detected at the encoder´s reference point (Z) signal.

Inputs used as reference point Z in the synchronisation process are:

Counter0 -> Z0

Counter1 -> Z1

MEASURING_STEP REAL (measuring step)

Indicates the resolution of the encoder device (see technical data from manufacturer). For example, using an incremental encoder with a resolution of 10000 pulses each 360° and x4 counting mode (software configuration), measuring step will use the following value:

360 / (4 x 10000) = 0,009

Format: Degrees.

Z_PULSE BOOL (input signal Z)

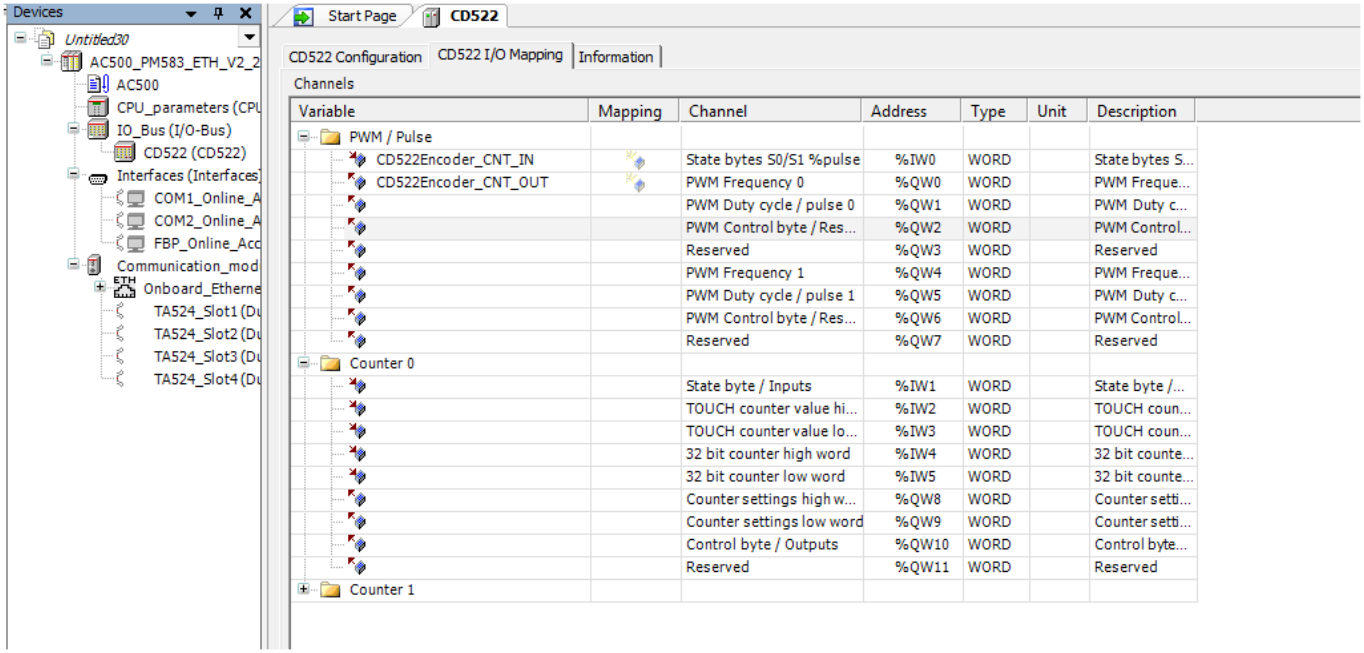
The output Z_PULSE indicates that encoder's reference point Z has been reached. This signal is only visible if programs execution time is slower than the signal of encoder's reference point (Z).

CNT_IN WORD (counter input)

First input data of counter input. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:

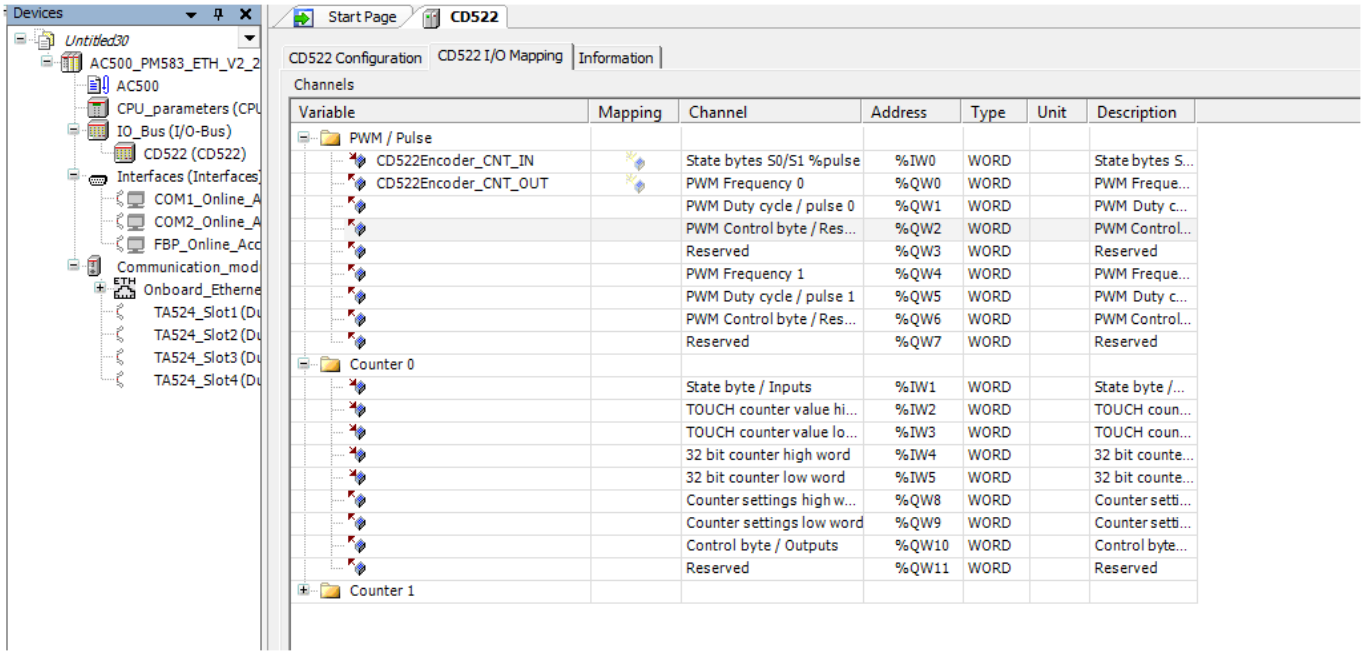


CNT_OUT WORD (counter output)

First output data of counter output. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:



POS_ACT REAL (position_actual)

The current position (actual position) can be retrieved at any time using the output POS_ACT of the function block. If a shutdown occurs, value of actual position will be lost. Due to that, that parameter must be declared as a retain variable.

If communications problem occurs between CPU and module CD522, position value stored in POS_ACT will be lost.

Format: Degrees.

Z_VALUE REAL (z value)

Indicates the position of the encoders reference point (Z). The user is able to configure where the Z position is.

Format: Degrees.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

RDY_SET BOOL (ready set start)

When the action of SET_START finishes, RDY_SET is set to TRUE for only one cycle.

RDY_CALIB BOOL (ready calibration)

The output RDY_CALIB displays the ready message of the calibrating process. While input EN_CALIB is set to TRUE, RDY_CALIB = TRUE.

If EN_CALIB = TRUE and rising edge is detected at Z_PULSE signal, RDY_CALIB will be set to TRUE until EN_CALIB = FALSE.

RDY_SYNC BOOL (ready synchronisation)

The output RDY_SYNC displays the ready message of the synchronisation process. It is set to TRUE only for one cycle.

Function call in IL

CAL IOEncoder(		
	EN	:= IOEncoder_EN,
	EN_CNT	:= IOEncoder_EN_CNT,
	SET_START	:= IOEncoder_SET_START,
	SET_START_VALUE	:= IOEncoder_SET_START_VALUE,
	EN_CALIB	:= IOEncoder_EN_CALIB
	EN_SYNC	:= IOEncoder_EN_SYNC,
	MEASURING_STEP	:= IOEncoder_MEASURING_STEP,
	Z_PULSE	:= IOEncoder_Z_PULSE
	CNT_IN	:= IOEncoder_CNT_IN
	CNT_OUT	:= IOEncoder_CNT_OUT
	POS_ACT	:= IOEncoder_POS_ACT
	Z_VALUE	:= IOEncoder_Z_VALUE);
LD		IOEncoder.DONE
ST		IOEncoder_DONE
LD		IOEncoder.ERR
ST		IOEncoder_ERR
LD		IOEncoder.ERNO
ST		IOEncoder_ERNO
LD		IOEncoder.RDY_SET
ST		IOEncoder_RDY_SET

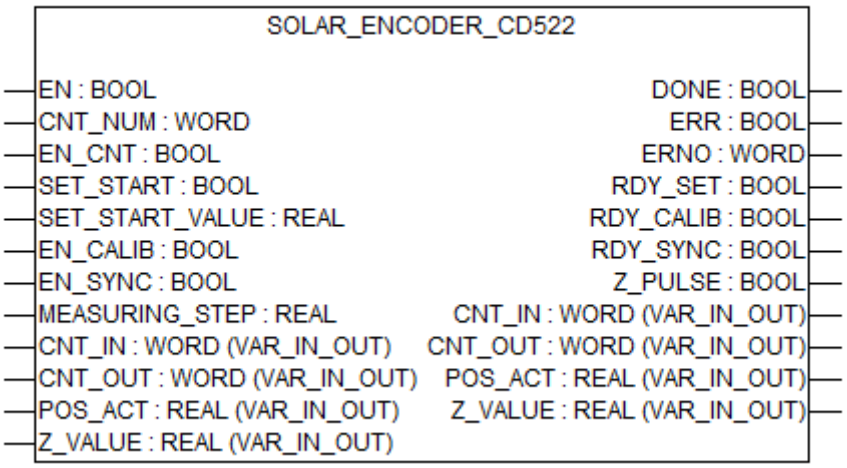
LD	IOEncoder.RDY_CALIB
ST	IOEncoder_RDY_CALIB
LD	IOEncoder.RDY_SYNC
ST	IOEncoder_RDY_SYNC

In IL, the function call has to be written in one line.

Function call in ST

IOEncoder(		
	EN	:= IOEncoder_EN,
	EN_CNT	:= IOEncoder_EN_CNT,
	SET_START	:= IOEncoder_SET_START,
	SET_START_VALUE	:= IOEncoder_SET_START_VALUE,
	EN_CALIB	:= IOEncoder_EN_CALIB,
	EN_SYNC	:= IOEncoder_EN_SYNC,
	MEASURING_STEP	:= IOEncoder_MEASURING_STEP,
	Z_PULSE	:= IOEncoder_Z_PULSE,
	CNT_IN	:= IOEncoder_CNT_IN,
	CNT_OUT	:= IOEncoder_CNT_OUT,
	POS_ACT	:= IOEncoder_POS_ACT,
	Z_VALUE	:= IOEncoder_Z_VALUE);
	IOEncoder_DONE	:= IOEncoder.DONE;
	IOEncoder_ERR	:= IOEncoder.ERR;
	IOEncoder_ERNO	:= IOEncoder.ERNO;
	IOEncoder_RDY_SET	:= IOEncoder.RDY_SET;
	IOEncoder_RDY_CALIB	:= IOEncoder.RDY_CALIB;
	IOEncoder_RDY_SYNC	:= IOEncoder.RDY_SYNC;

SOLAR_ENCODER_CD522



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block with historical values
Group	Package of functions to get the position of the sun

Function block to easily integrate, with the CD522 card, incremental encoders (A, B, Z) as positioning sensors for Solar Trackers.

The SOLAR_ENCODER_CD522 is used to easily integrate, with the CD522 card, incremental encoders (A, B, Z) as positioning sensors for Solar Trackers.

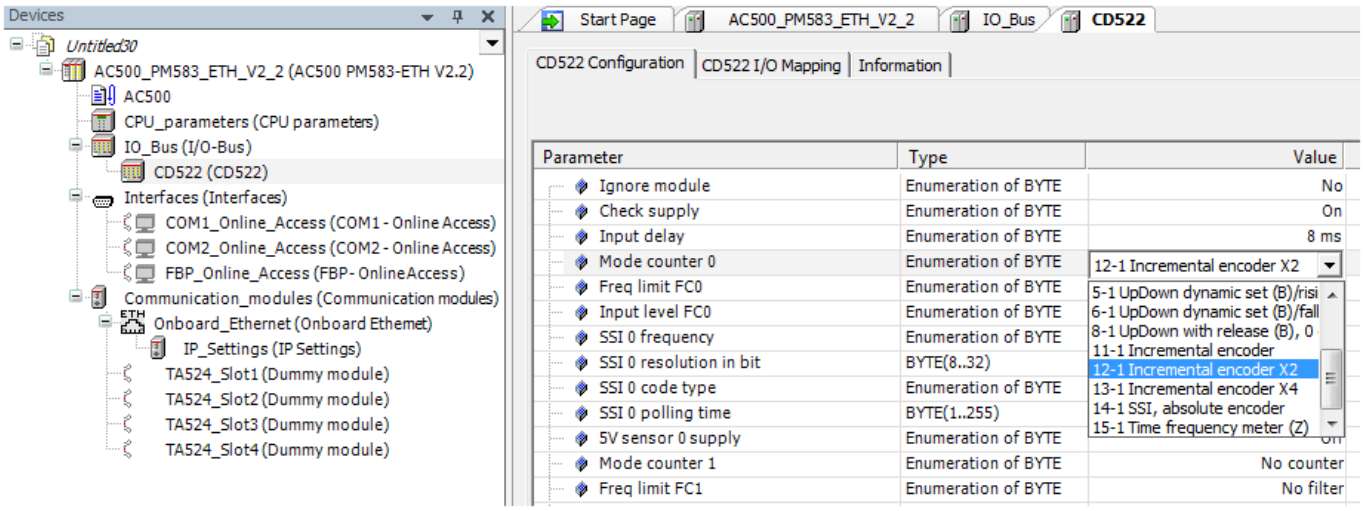
The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

In order to configure and use the function encoder of module CD522, different operating modes are available. The function block SOLAR_ENCODER_CD522 should be used by configuring the parameter 6 or 14 ("Mode counter 0" or "Mode counter 1") in modes 11, 12 or 13 to use the incremental encoder as: Incremental encoder, Incremental encoder X2 or Incremental encoder X4 respectively.

See user manual ↬ *Chapter 1.6.2.7.2.1 “CD522 - Encoder, counter and PWM module” on page 4635*

Example: Configuration of module CD522 to control an incremental encoder:

In Control Builder Plus V2.0 and above:



The meaning of the different operating modes available to control an incremental encoder is:

Operating Mode 11 "Incremental encoder"

Should be specified in PLC Configuration; parameter "Mode counter" in order to use one up/down counter for position sensor x1 count.

Operating Mode 12 "Incremental encoder X2"

Should be specified in PLC Configuration; parameter "Mode counter" in order to use one bidirectional counter for position sensor x2 count.

Operating Mode 13 "Incremental encoder X4"

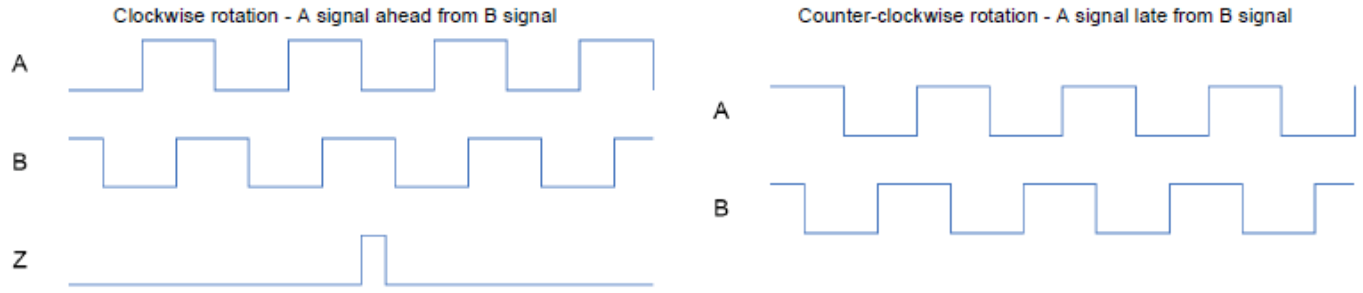
Should be specified in PLC Configuration; parameter "Mode counter" in order to use one bidirectional counter for position sensor x4 count.

The module CD522 provides 2 encoder functions for relative positioning with 3 signals. 2 signals are used for rotation discrimination and pulse count, identified by A0 and B0 for counter0 and A1 and B1 for counter1. The third one is used in multi-turn encoder to count the number of rotation (mechanical zero), identified by Z0 for counter0 and Z1 for counter1. Signals Z0 and Z1 are used in synchronization process. Inputs I3 and I11 needs to be defined as touch input.

> *If the setup needs both Calibration & Synchronization then the Z pulse from the encoder to be connected to both Z0 & I3 inputs for counter0 and Z1 & I11 inputs for counter1.*

The rotation is identified with a shift angle (90°) between A and B signal. In the module CD522, the clockwise rotation is identified with A signal in advance to B (see figure below).

Example: Direction identification using A and B signals:



Depending on which kind of operating mode is specified, the counting procedure will be x1, x2 or x4 count. Basically the x1 counting mode is used (mode 11). The encoder module discriminates the rotating way and count one pulse for each rising edge of the A signal.

In order to increase resolution, the x2 counting mode can be specified (mode 12). The encoder module counts both the positive edges and the negative edges of trace A. This results in the double number of counting pulses. The precision increases correspondingly.

The resolution could be multiplied by 4, using the x4 counting mode (mode 13). The encoder module counts a pulse on both rising and falling edge of A signal and B signal.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CNT_NUM WORD (counter number)

Indicates which counter is going to be used.

If CNT_NUM = 0, the counter 0 will be used.

If CNT_NUM = 1, the counter 1 will be used.

EN_CNT BOOL (enable counter)

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and the pulses are lost.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value POS_ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value POS_ACT will continue since previous value.

While EN_CNT is set to FALSE, it is possible to modify the stored value in POS_ACT variable. When EN_CNT is set to TRUE again, counter value POS_ACT will continue since previous value.

SET_START BOOL (set start)

If set input SET_ START = TRUE, the counter takes the values from input SET_START_VALUE to transfer it to POS_ACT. After that, RDY_SET = TRUE. As long as input SET_ START = TRUE, no pulses are counted because the counter is always overwritten by the input SET_START_VALUE.

SET_START_VALUE REAL (set start value)

The counter can be set to a Start value. This value must be applied to the input SET_START_VLAUE.

If input SET_START=TRUE, the counter takes this value.

EN_CALIB BOOL (enabling calibrate)

If EN_CALIB = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output Z_VALUE will take the stored value in POS_ACT and output RDY_CALIB is set to TRUE. If this input is continuously set to TRUE, only one calibrating process will be done even system reaches encoder´s reference point (Z) once again. In order to enable another calibrating process, input EN_CALIB has to be set to FALSE and set to TRUE again.

Calibrating process is executed out of CPU cycle, and it is able to detect the multiple rising edges that appear when a simple limit switch is used. Due to that, it is recommended to use devices assembled especially for this purpose (high accuracy limit switch).

Inputs used as reference point Z in the calibrating process are:

Counter0 -> I3

Counter1 -> I11

EN_SYNC BOOL (enabling synchronize)

If input EN_SYNC = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output POS_ACT will take the value stored at Z_VALUE, output RDY_SYNC is set to TRUE only during a program cycle. While EN_SYNC is set to TRUE, POS_ACT output will take the value stored at Z_VALUE every time that a rising edge is detected at the encoder´s reference point (Z) signal.

Inputs used as reference point Z in the synchronisation process are:

Counter0 -> Z0

Counter1 -> Z1

MEASURING_STEP REAL (measuring step)

Indicates the resolution of the encoder device (see technical data from manufacturer). For example, using an incremental encoder with a resolution of 10000 pulses each 360° and x4 counting mode (software configuration), measuring step will use the following value:

360 / (4 x 10000) = 0,009

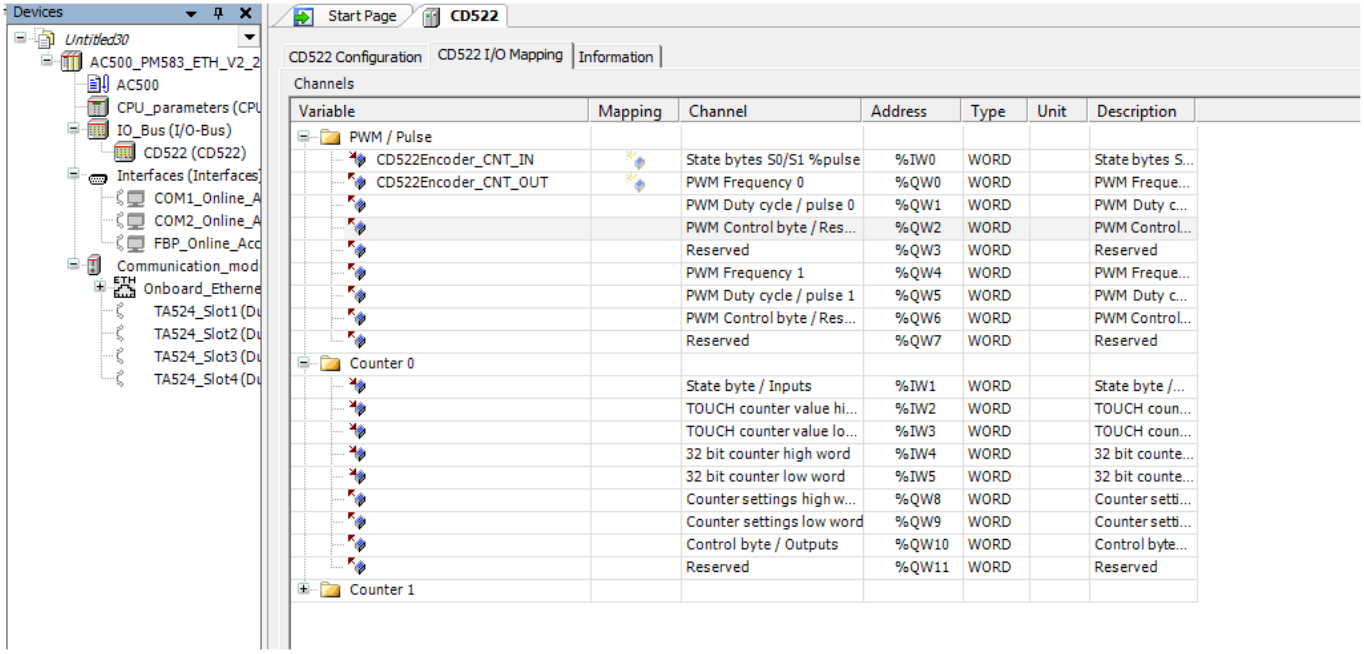
Format: Degrees.

CNT_IN WORD (counter input)

First input data of counter input. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:

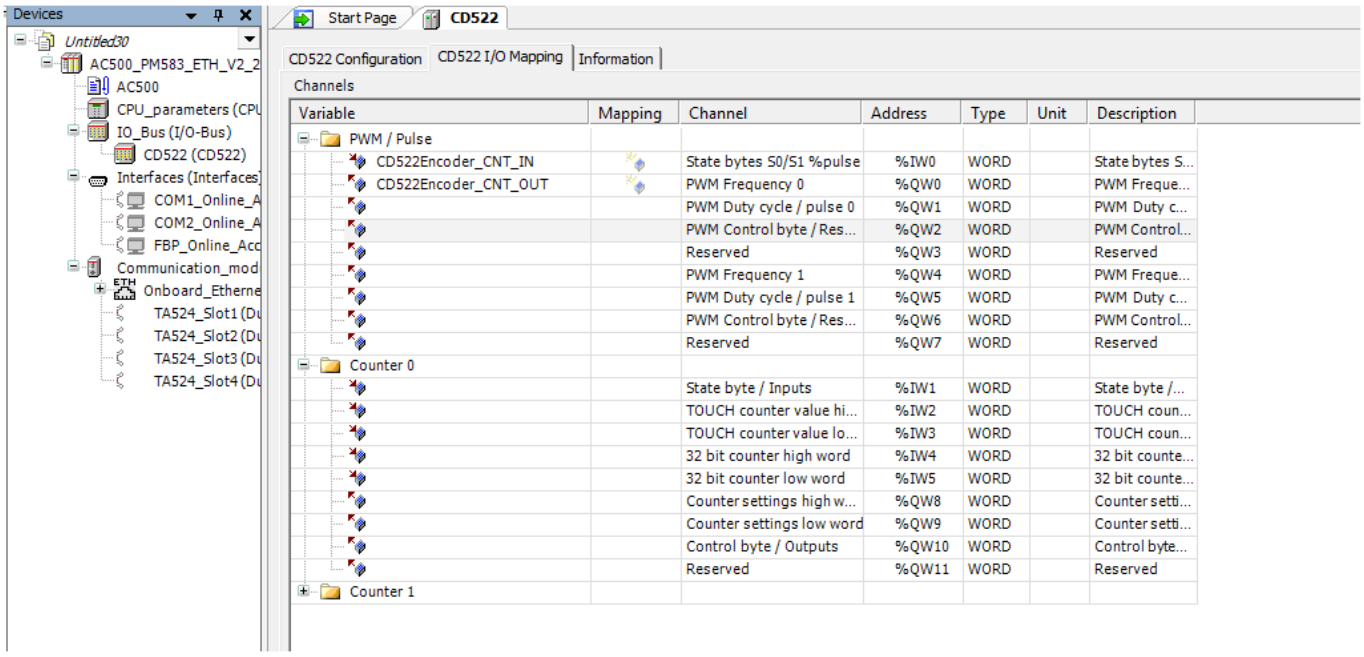


CNT_OUT WORD (counter output)

First output data of counter output. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:



POS_ACT REAL (position_actual)

The current position (actual position) can be retrieved at any time using the output POS_ACT of the function block. If a shutdown occurs, value of actual position will be lost. Due to that, that parameter must be declared as a retain variable.

If communications problem occurs between CPU and module CD522, position value stored in POS_ACT will be lost.

Format: Degrees.

Z_VALUE REAL (z value)

Indicates the position of the encoders reference point (Z). The user is able to configure where the Z position is.

Format: Degrees.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

RDY_SET BOOL (ready set start) When the action of SET_START finishes, RDY_SET is set to TRUE for only one cycle.

RDY_CALIB BOOL (ready calibration) The output RDY_CALIB displays the ready message of the calibrating process. While input EN_CALIB is set to TRUE, RDY_CALIB = TRUE.

If EN_CALIB = TRUE and rising edge is detected at Z_PULSE signal, RDY_CALIB will be set to TRUE until EN_CALIB = FALSE.

RDY_SYNC BOOL (ready synchronisation) The output RDY_SYNC displays the ready message of the synchronisation process. It is set to TRUE only for one cycle.

Z_PULSE BOOL (input signal Z) The output Z_PULSE indicates that encoder's reference point Z has been reached. This signal is only visible if programs execution time is slower than the signal of encoder's reference point (Z).

Function call in IL

CAL CD522Encoder(		
	EN	:= CD522Encoder_EN,
	CNT_NUM	:= CD522Encoder_CNT_NUM,
	SET_START	:= CD522Encoder_SET_START,
	SET_START_VALUE	:= CD522Encoder_SET_START_VALUE,
	EN_CALIB	:= CD522Encoder_EN_CALIB,
	EN_SYNC	:= CD522Encoder_EN_SYNC,
	MEASURING_STEP	:= CD522Encoder_MEASURING_STEP,
	CNT_IN	:= CD522Encoder_CNT_IN,
	CNT_OUT	:= CD522Encoder_CNT_OUT,
	POS_ACT	:= CD522Encoder_POS_ACT,
	Z_VALUE	:= CD522Encoder_Z_VALUE);
LD		CD522Encoder.DONE
ST		CD522Encoder_DONE
LD		CD522Encoder.ERR
ST		CD522Encoder_ERR
LD		CD522Encoder.ERNO
ST		CD522Encoder_ERNO

LD ST	CD522Encoder.RDY_SET CD522Encoder_RDY_SET
LD ST	CD522Encoder.RDY_CALIB CD522Encoder_RDY_CALIB
LD ST	CD522Encoder.RDY_SYNC CD522Encoder_RDY_SYNC
LD ST	CD522Encoder.RDY_Z_PULSE CD522Encoder_RDY_Z_PULSE

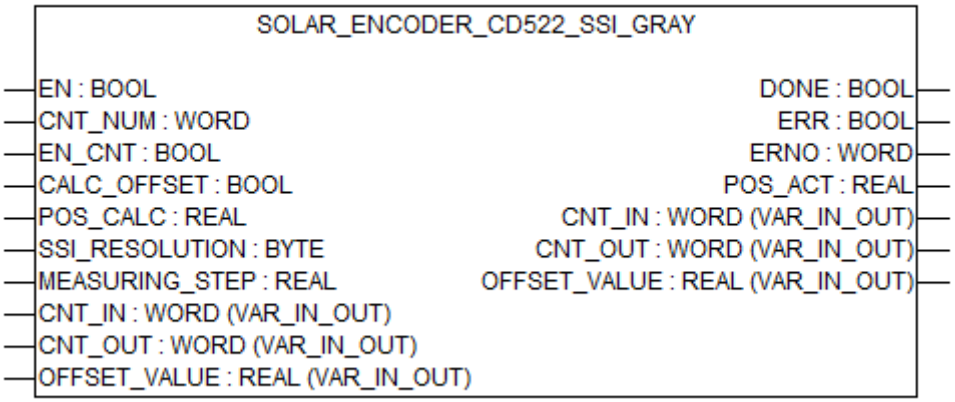
In IL, the function call has to be written in one line.

Function call in ST

CD522Encoder(		
	EN	:= CD522Encoder_EN,
	CNT_NUM	:= CD522Encoder_CNT_NUM,
	SET_START	:= CD522Encoder_SET_START,
	SET_START_VALUE	:= CD522Encoder_SET_START_VALUE,
	EN_CALIB	:= CD522Encoder_EN_CALIB,
	EN_SYNC	:= CD522Encoder_EN_SYNC,
	MEASURING_STEP	:= CD522Encoder_MEAS-URING_STEP,
	CNT_IN	:= CD522Encoder_CNT_IN,
	CNT_OUT	:= CD522Encoder_CNT_OUT,
	POS_ACT	:= CD522Encoder_POS_ACT,
	Z_VALUE	:= CD522Encoder_Z_VALUE);
	CD522Encoder_DONE	:= CD522Encode.DONE;
	CD522Encoder_ERR	:= CD522Encoder.ERR;
	CD522Encoder_ERNO	:= CD522Encoder.ERNO;
	CD522Encoder_RDY_SET	:= CD522Encoder.RDY_SET;
	CD522Encoder_RDY_CALIB	:= CD522Encoder.RDY_CALIB;

	CD522Encoder_RDY_SYNC	:= CD522Encoder.RDY_SYNC;
	CD522Encoder_Z_PULSE	:= CD522Encoder.Z_PULSE;

SOLAR_ENCODER_CD522_SSI_GRAY



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values .
Group	Package of functions to get the position of the sun

Function block to easily integrate, with the CD522 card, absolute encoders SSI GRAY as positioning sensors for Solar Trackers.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_SSI should be used configuring the parameter 6 or 14 ("Mode counter 0" or "Mode counter 1") with the operating mode:

Operating Mode 14 "SSI, absolute encoder"

Should be specified in PLC Configuration; parameter "Mode counter" in order to use absolute encoder with SSI interface.

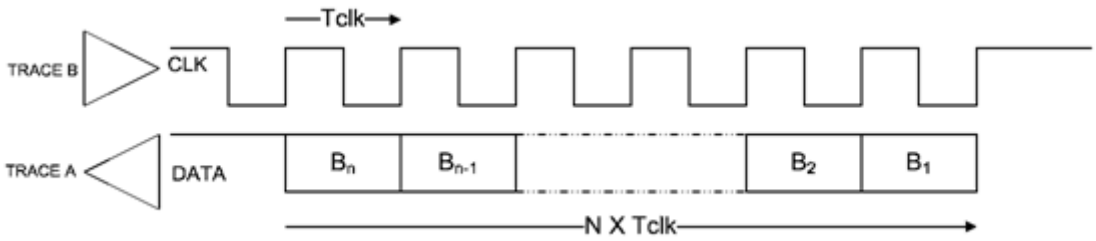
The module CD522 provides 2 SSI absolute encoder functions. There is an interface for absolute angle and linear encoders (displacement measurement systems).

It allows the transmission of absolute position information through a serial data transfer.

The transmission is based on synchronous serial communication. The device sends a clock signal to the encoder and synchronously, the encoder returns the positioning data from the most significant to the less significant bit.

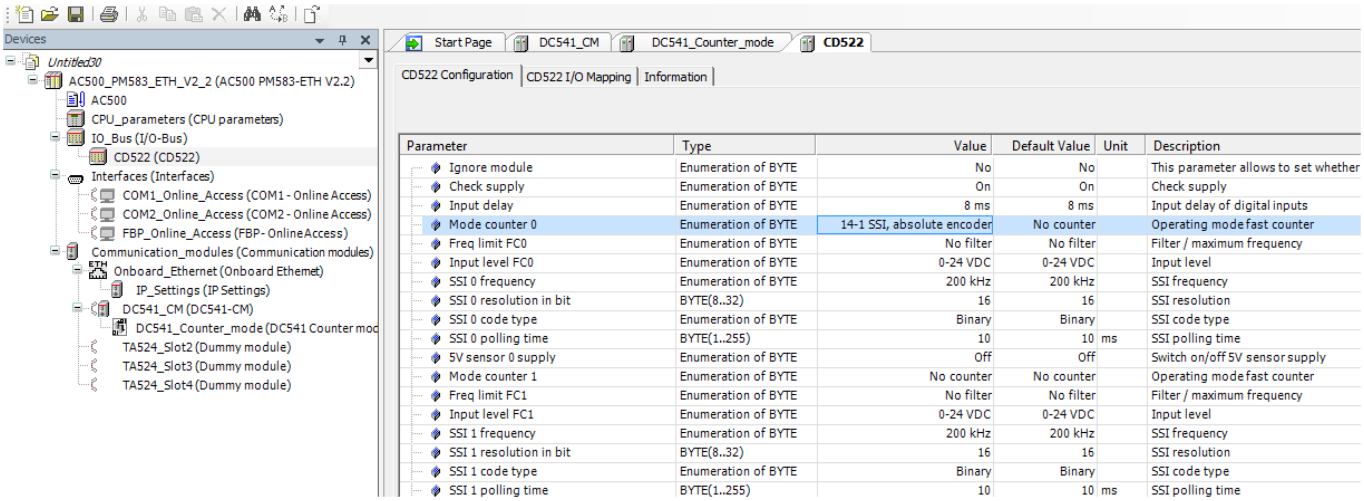
The synchronization for a new data stream is based on time without clock pulse. This quiet time depends on the encoder.

Chronogram with data organization with the clock pulse:



The resolution of the encoder device (see technical data from manufacturer) can be set (with configuration, number of bits, etc.) in PLC Configuration, parameter "SSI resolution in bit" (see figure below).

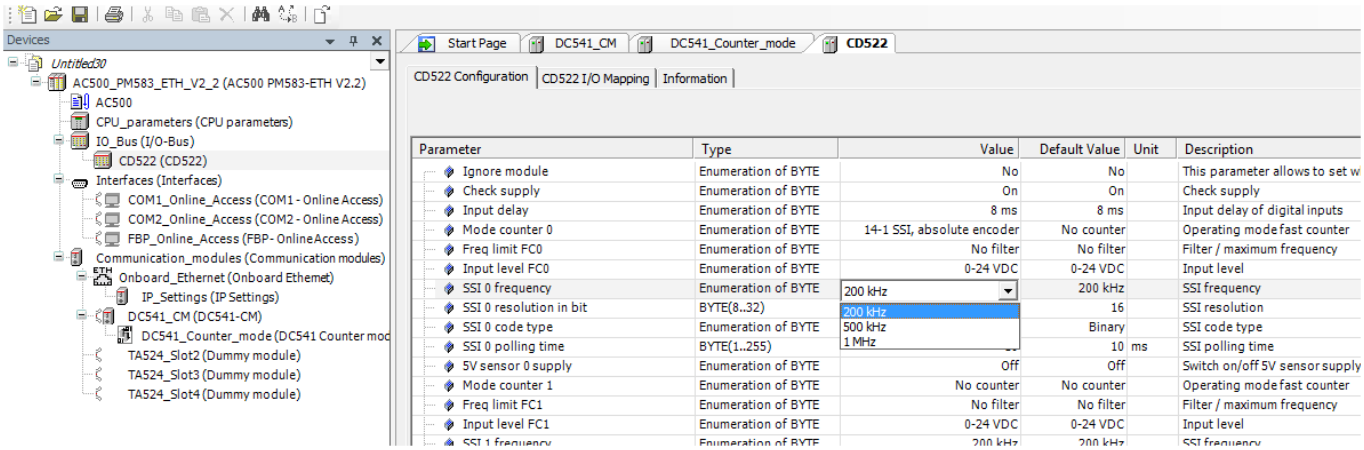
In Control Builder Plus V2.0 and above :



The trace B of module CD522 is switched as output signal (differential). On the rising edge of the signal, the sensor shifts a new value, starting from the most significant bit.

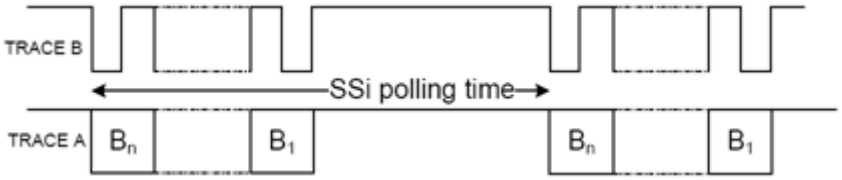
The clock frequency can be set to 200 kHz, 500 kHz, and 1 MHz and should be specified in PLC Configuration, parameter "SSI frequency" (see figure below).

In Control Builder Plus V2.0 and above:



SSI Polling time definition

The complete read sequence is launched regularly by the module CD522. The interval between each sequence can be set from 1 ms to 255 ms in PLC Configuration, parameter "SSI polling time" (see figure below).



Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CNT_NUM WORD (counter number)

Indicates which counter is going to be used:

If CNT_NUM = 0, the counter 0 will be used.

If CNT_NUM = 1, the counter 1 will be used.

EN_CNT BOOL (enable counter)

IF EN_CNT = TRUE, communications process with SSI encoder is enabled. If EN_CNT = FALSE, communications process with SSI encoder is not enabled.

CALC_OFFSET BOOL (calculated offset)

This input is used to calibrate the positioning sensor. When rising edge is detected at CALC_OFFSET, function block proceeds as follow:

OFFSET_VALUE = POS_CALC ACT_CNT,

where ACT_CNT is the read value directly from encoder (internal variable).

Once the system is calibrated, this value remains constant during the normal working of the system. This value should be check to do a correct calibration after maintenance operations.

POS_CALC REAL (position_calculted)

It is the right position of our tracker and it is used to calibrate the system. This value must be obtained by using high accuracy tools (not assembled in the tracker) or by using a solar radiation sensor to determine if tracker is right focused.

Format: Degrees.

SSI_RESOLUTION BYTE (SSI RESOLUTION)

The resolution of the encoder device setting by number of bits. Min. 8 bits, max. 32 bits.

MEASURING_STEP REAL (measuring step)

Indicates the resolution of the encoder device (see technical data from manufacturer). For example, using an incremental encoder with a resolution of 10000 pulses each 360° and x4 counting mode (software configuration), measuring step will use the following value:

360 / (4 x 10000) = 0,009

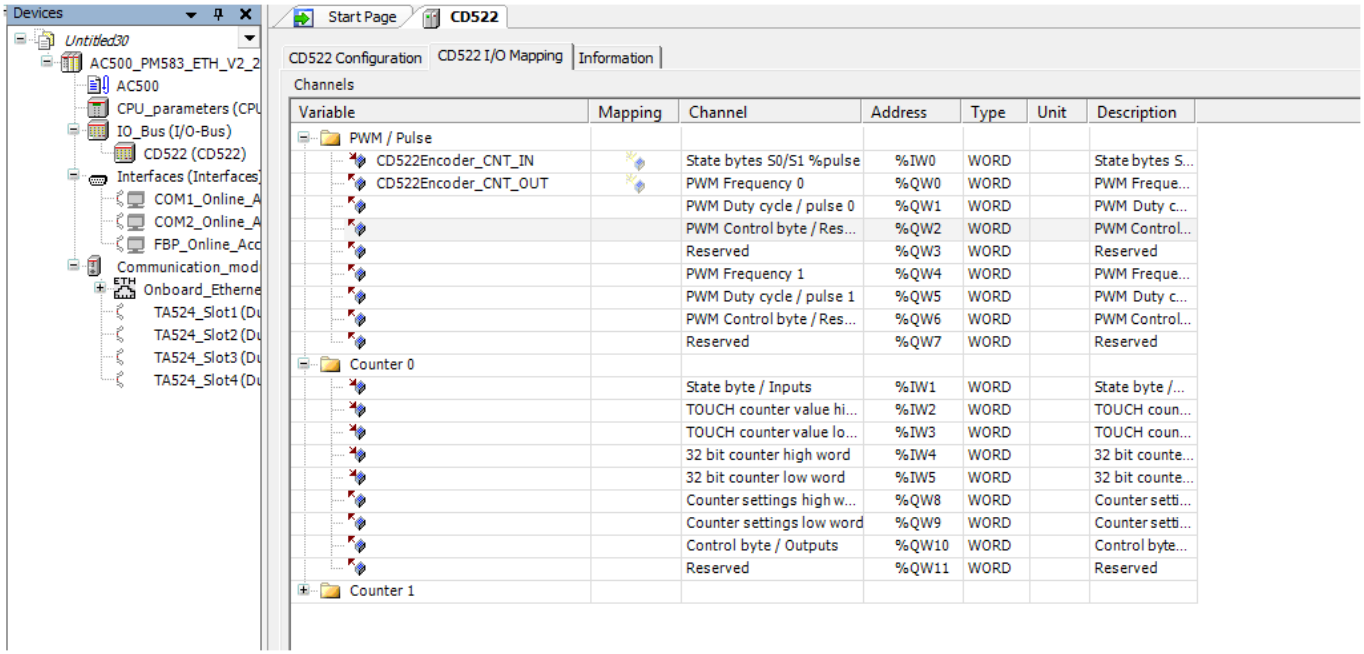
Format: Degrees.

CNT_IN WORD (counter input)

First input data of counter input. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:

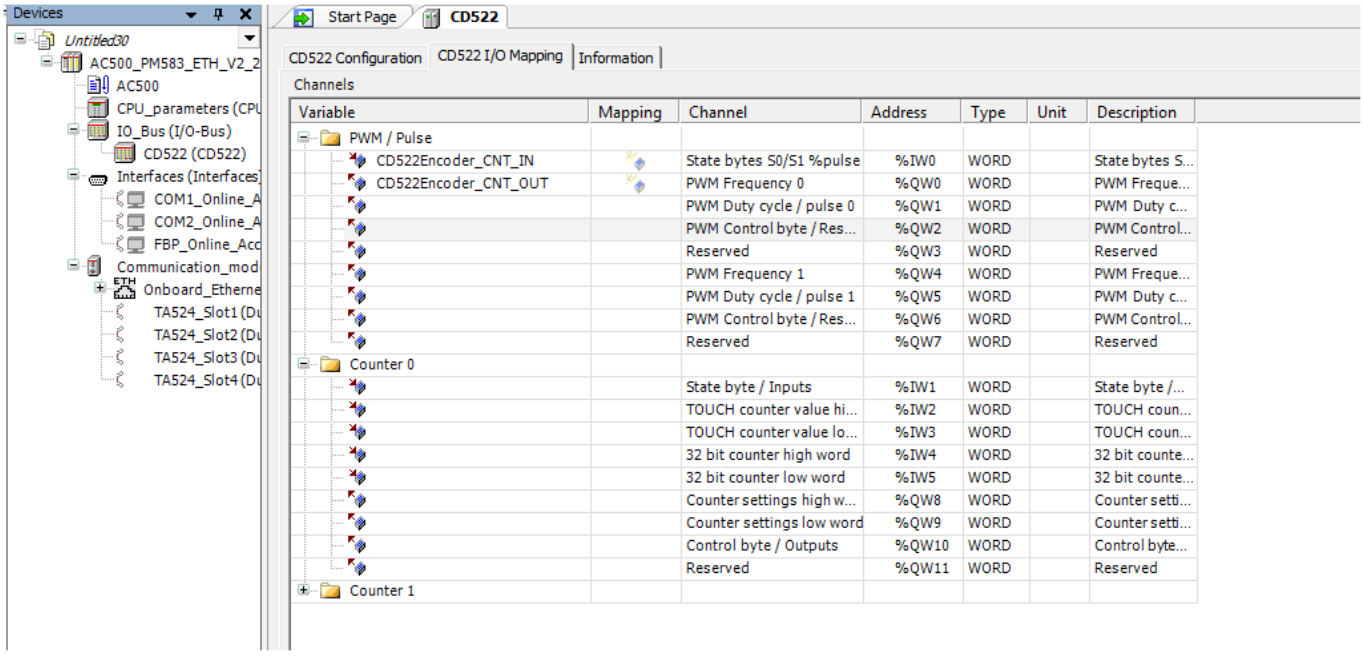


CNT_OUT WORD (counter output)

First output data of counter output. The use of an ADR operator is not needed.

Example (for counter 0):

In Control Builder Plus V2.0 and above:



OFFSET_VALUE REAL (offset value)

Value added to ACT_CNT (internal variable) to obtain a calibrating process.

POS_ACT = ACT_CNT + OFFSET_VALUE,

where ACT_CNT is the directly read value from encoder (internal variable).

Format: Degrees.

POS_ACT REAL (position_actual)

The current position (actual position) can be retrieved at any time using the output POS_ACT of the function block. If a shutdown occurs, value of actual position will be lost. Due to that, that parameter must be declared as a retain variable.

If communications problem occurs between CPU and module CD522, position value stored in POS_ACT will be lost.

Format: Degrees.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Function call in IL

CAL CD522SSI(		
	EN	:= CD522SSI_EN,
	CNT_NUM	:= CD522SSI_CNT_NUM,
	EN_CNT	:= CD522SSI_EN_CNT,
	CALC_OFFSET	:= CD522SSI_CALC_OFFSET,
	POS_CALC	:= CD522SSI_POS_CALC,
	SSI_RESOLUTION	:= CD522SSI_SSI_RESOLUTION,
	MEASURING_STEP	:= CD522SSI_MEASURING_STEP,
	CNT_IN	:= CD522SSI_CNT_IN,
	CNT_OUT	:= CD522SSI_CNT_OUT,
	OFFSET_VALUE	:= CD522SSI_OFFSET_VALUE) ;

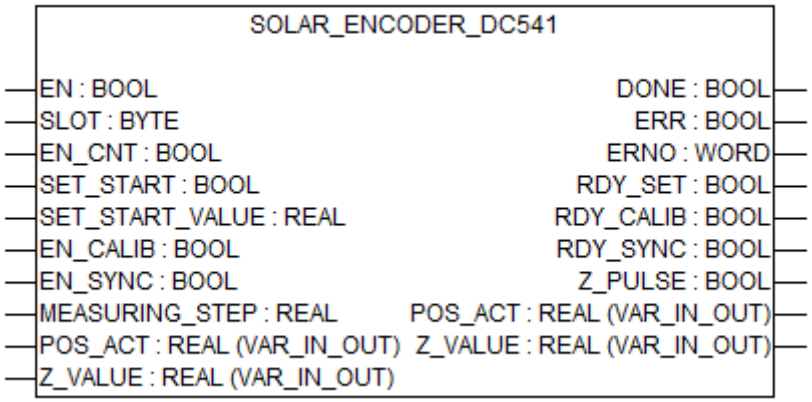
LD	CD522SSI.DONE
ST	CD522SSI_DONE
LD	CD522SSI.ERR
ST	CD522SSI_ERR
LD	CD522SSI.ERNO
ST	CD522SSI_ERNO
LD	CD522SSI.POS_ACT
ST	CD522SSI_POS_ACT

In IL, the function call has to be written in one line.

Function call in ST

CD522SSI(		
	EN	:= CD522SSI_EN,
	CNT_NUM	:= CD522SSI_CNT_NUM,
	EN_CNT	:= CD522SSI_EN_CNT,
	CALC_OFFSET	:= CD522SSI_CALC_OFFSET,
	POS_CALC	:= CD522SSI_POS_CALC,
	SSI_RESOLUTION	:= CD522SSI_SSI_RESOLUTION,
	MEASURING_STEP	:= CD522SSI_MEASURING_STEP,
	CNT_IN	:= CD522SSI_CNT_IN,
	CNT_OUT	:= CD522SSI_CNT_OUT,
	OFFSET_VALUE	:= CD522SSI_OFFSET_VALUE);
	CD522SSI_DONE	:= CD522SSI.DONE;
	CD522SSI_ERR	:= CD522SSI.ERR;
	CD522SSI_ERNO	:= CD522SSI.ERNO;
	CD522SSI_POS_ACT	:= CD522SSI.POS_ACT;

SOLAR_ENCODER_DC541



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block with historical values
Group	Package of functions to get the position of the sun

The SOLAR_ENCODER_DC541 is used to easily integrate, with the DC541 module, incremental encoders (A, B, Z) as positioning sensors for Solar Trackers.

The module DC541 can be done a count of signals up to 60 kHz. Pulse multiplication (x2 or x4) is not used.

The counter always uses the channels C0...C3 of the DC541:

C0: Track A of the incremental encoder

C1: Track B of the incremental encoder

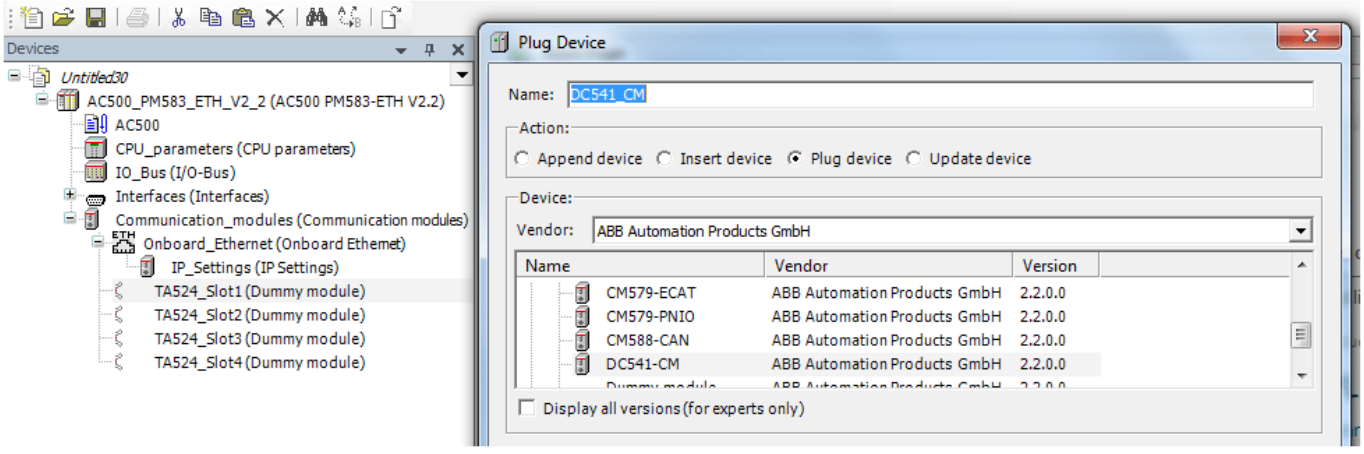
C2: Track Z of the incremental encoder

In order to configure and use the function counter of module DC541, the module should be configured as follow:

Configuration of module DC541 to control an incremental encoder.

Step 1:

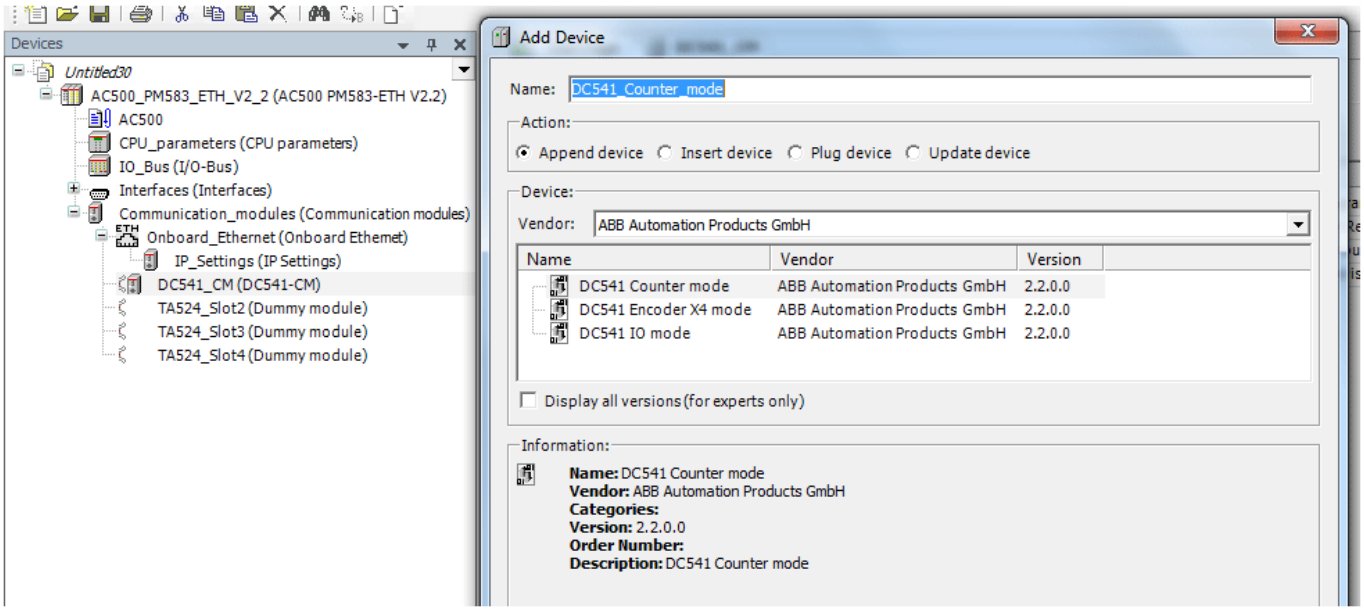
In Control Builder Plus V2.0 and above:



The function block SOLAR_ENCODER_DC541 should be used configuring the module as a "Counter mode".

In Control Builder Plus V2.0 and above:

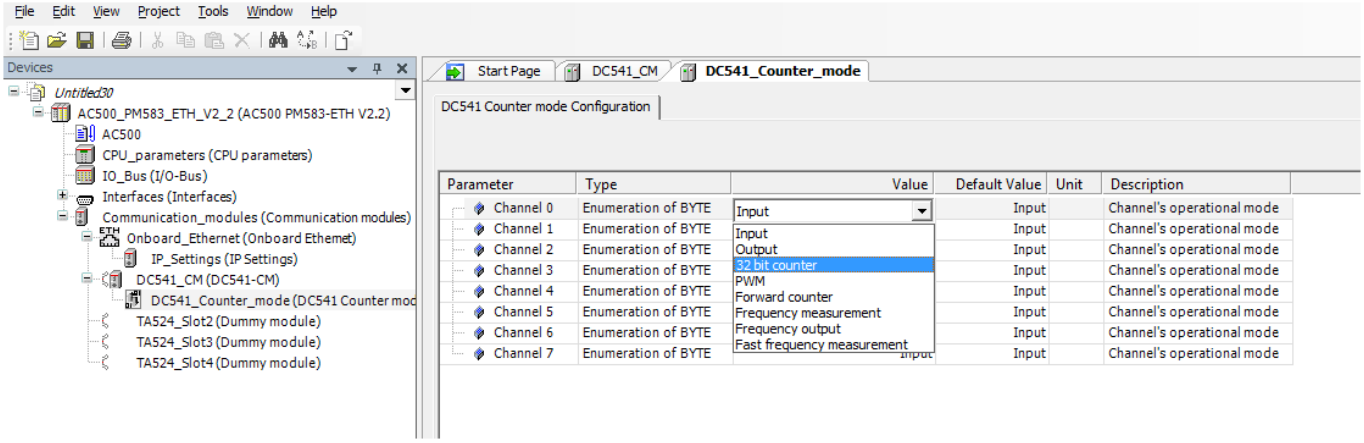
Last step is to configure the "Channel 0" as "32-bit counter".



Configuration of module DC541 to control an incremental encoder.

Step 3:

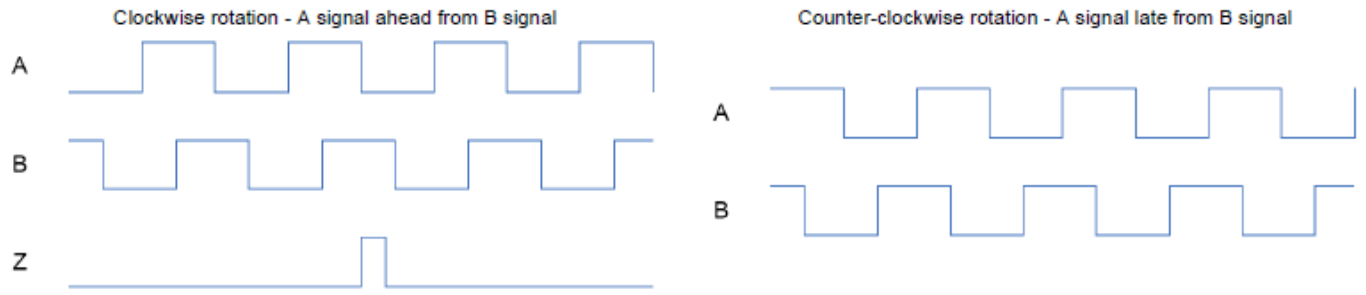
In Control Builder Plus V2.0 and above:



The module DC541 provides an encoder function for relative positioning with 3 signals. 2 signals are used for rotation discrimination and pulse. The third one is used in multi-turn encoder to count the number of rotation (mechanical zero).

The rotation is identified with a shift angle (90°) between A and B signal. In the module DC541, the clockwise rotation is identified with A signal in advance to B (see figure below).

Example: Direction identification using A and B signals:



Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLOT

Data type	Default value	Range	Unit
BYTE	0	0 ... 4 (6)	-

At input SLOT, the device slot is specified which should be used by the function block.

The internal device always has the index 0. All external devices are serially numbered from right to left, starting with 1.

EN_CNT BOOL (enable counter)

If EN_CNT = TRUE, pulse counting of counter is enabled. If EN_CNT = FALSE, no pulse counting is performed and pulses are not stored even the encoder device changes its position.

If counting has already started and if EN_CNT = FALSE, the pulse counting stops and counter value ACT is stored. If EN_CNT = TRUE again, the pulse counting will start again and counter value ACT will continue since previous value.

SET_START BOOL (set start)

If set input SET_ START = TRUE, the counter takes the values from input SET_START_VALUE to transfer it to POS_ACT. After that, RDY_SET = TRUE. As long as input SET_ START = TRUE, no pulses are counted because the counter is always overwritten by the input SET_START_VALUE.

SET_START_VALUE REAL (set start value)

The counter can be set to a Start value. This value must be applied to the input SET_START_VLAUE.

If input SET_START=TRUE, the counter takes this value.

EN_CALIB BOOL (enabling calibrate)

If EN_CALIB = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output Z_VALUE will take the stored value in POS_ACT and output RDY_CALIB is set to TRUE. If this input is continuously set to TRUE, only one calibrating process will be done even system reaches encoder´s reference point (Z) once again. In order to enable another calibrating process, input EN_CALIB has to be set to FALSE and set to TRUE again.

Calibrating process is executed out of CPU cycle, and it is able to detect the multiple rising edges that appear when a simple limit switch is used. Due to that, it is recommended to use devices assembled especially for this purpose (high accuracy limit switch).

Inputs used as reference point Z in the calibrating process are:

Counter0 -> I3

Counter1 -> I11

EN_SYNC BOOL (enabling synchronize)

If input EN_SYNC = TRUE, when a rising edge is detected at the encoder´s reference point (Z) signal, output POS_ACT will take the value stored at Z_VALUE, output RDY_SYNC is set to TRUE only during a program cycle. While EN_SYNC is set to TRUE, POS_ACT output will take the value stored at Z_VALUE every time that a rising edge is detected at the encoder´s reference point (Z) signal.

Inputs used as reference point Z in the synchronisation process are:

Counter0 -> Z0

Counter1 -> Z1

MEASURING_STEP REAL (measuring step)

Indicates the resolution of the encoder device (see technical data from manufacturer). For example, using an incremental encoder with a resolution of 10000 pulses each 360° and x4 counting mode (software configuration), measuring step will use the following value:

360 / (4 x 10000) = 0,009

Format: Degrees.

POS_ACT REAL (position_actual)

The current position (actual position) can be retrieved at any time using the output POS_ACT of the function block. If a shutdown occurs, value of actual position will be lost. Due to that, that parameter must be declared as a retain variable.

If communications problem occurs between CPU and module CD522, position value stored in POS_ACT will be lost.

Format: Degrees.

Z_VALUE REAL (z value)

Indicates the position of the encoders reference point (Z). The user is able to configure where the Z position is.

Format: Degrees.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

RDY_SET BOOL (ready set start)

When the action of SET_START finishes, RDY_SET is set to TRUE for only one cycle.

RDY_CALIB BOOL (ready calibration)

The output RDY_CALIB displays the ready message of the calibrating process. While input EN_CALIB is set to TRUE, RDY_CALIB = TRUE.

If EN_CALIB = TRUE and rising edge is detected at Z_PULSE signal, RDY_CALIB will be set to TRUE until EN_CALIB = FALSE.

RDY_SYNC BOOL (ready synchronisation)

The output RDY_SYNC displays the ready message of the synchronisation process. It is set to TRUE only for one cycle.

Z_PULSE BOOL (input signal Z)

The output Z_PULSE indicates that encoder's reference point Z has been reached. This signal is only visible if programs execution time is slower than the signal of encoder's reference point (Z).

Function call in IL

CAL DC541Encoder(		
	EN	:= CD522Encoder_EN,
	SLOT	:= DC541Encoder_SLOT,
	EN_CNT	:= DC541Encoder_EN_CNT,
	SET_START	:= DC541Encoder_SET_START,
	SET_START_VALUE	:= DC541EncoderSET_START_VALUE,
	EN_CALIB	:= DC541Encoder_EN_CALIB,
	EN_SYNC	:= DC541Encoder_EN_SYNC,
	MEASURING_STEP	:= DC541Encoder_MEASURING_STEP,
	POS_ACT	:= DC541Encoder_POS_ACT,
	Z_VALUE	:= DC541Encoder_Z_VALUE);
LD		DC541Encoder.DONE
ST		DC541Encoder_DONE
LD		DC541Encoder.ERR
ST		DC541Encoder_ERR
LD		DC541Encoder.ERNO
ST		DC541Encoder_ERNO

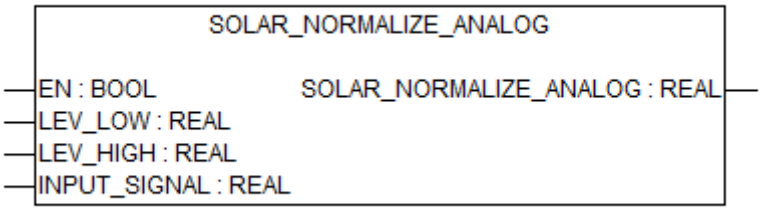
LD ST	DC541Encoder.RDY_SET DC541Encoder_RDY_SET
LD ST	DC541Encoder.RDY_CALIB DC541Encoder_RDY_CALIB
LD ST	DC541Encoder.RDY_SYNC DC541Encoder_RDY_SYNC
LD ST	DC541Encoder.RDY_Z_PULSE DC541Encoder_RDY_Z_PULSE

In IL, the function call has to be written in one line.

Function call in ST

DC541Encoder(		
	EN	:= DC541Encoder_EN,
	SLOT	:= DC541Encoder_SLOT,
	EN_CNT	:= DC541Encoder_EN_CNT,
	SET_START	:= DC541Encoder_SET_START,
	SET_START_VALUE	:= DC541Encoder_SET_START_VALUE,
	EN_CALIB	:= DC541Encoder_EN_CALIB,
	EN_SYNC	:= DC541Encoder_EN_SYNC,
	MEASURING_STEP	:= DC541Encoder_MEAS-URING_STEP,
	POS_ACT	:= DC541Encoder_POS_ACT,
	Z_VALUE	:= DC541Encoder_Z_VALUE);
	DC541Encoder_DONE	:= DC541Encode.DONE;
	DC541Encoder_ERR	:= DC541Encoder.ERR;
	DC541Encoder_ERNO	:= DC541Encoder.ERNO;
	DC541Encoder_RDY_SET	:= DC541Encoder.RDY_SET;
	DC541Encoder_RDY_CALIB	:= DC541Encoder.RDY_CALIB;
	DC541Encoder_RDY_SYNC	:= DC541Encoder.RDY_SYNC;
	DC541Encoder_Z_PULSE	:= DC541Encoder.Z_PULSE;

SOLAR_NORMALIZE_ANALOG



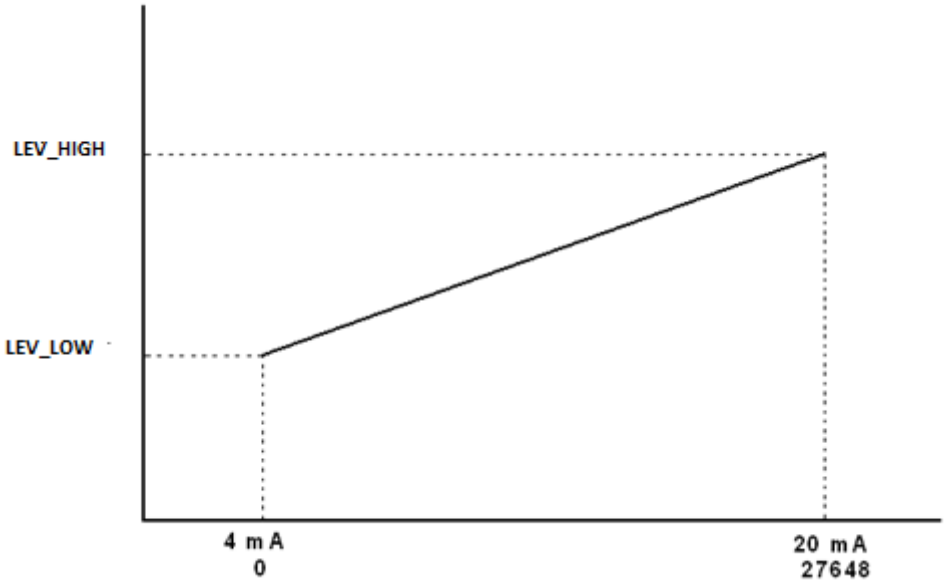
Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the position of the sun

The SOLAR_NORMALIZE_ANALOG function is used to easily integrate, with any S500 AI/AO card, standard sensor type 4 "“ 20 mA or 0 "“ 10V as positioning sensors for Solar Trackers. This function can be used to integrate analog sensors such as inclinometers.

According to the signal sensor provided and the movement range of the system, SOLAR_NORMALIZE_ANALOG gets the current system position. Internally function SOLAR_NORMALIZE_ANALOG uses the corresponding decimal values given by the AI/AO card when an analog input is detected.

Table 196: Input values (analog signals) with their corresponding decimal values for AX/521 and AX/522 modules

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital Input	Decimal Value
NORMAL RANGE	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1
	0.0000	0.0000	0	4	OFF	0



Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

LEV_LOW REAL (level low)

The lower limit of the movement range.

Format: Degrees.

LEV_HIGH REAL (level high)

The upper limit of the movement range.

Format: Degrees.

INPUT_SIGNAL REAL (input signal)

Analog sensor signal.

Format: mA.

Function call in IL

LD	AnalogSensor_LEV_LOW
SOLAR_NORMALIZE_ANALOG	AnalogSensor_LEV_HIGH, Analog-Sensor_INPUT_SIGNAL
ST	AnalogSensor_POSITION

In IL, the function call has to be written in one line.

Function call in ST

AnalogSensor_POSITION		:= SOLAR_NOR-MALIZE_ANALOG(
	LEV_LOW	:= SOLAR_NOR-MALIZE_ANALOG_LEV_LOW,
	LEV_HIGH	:= SOLAR_NOR-MALIZE_ANALOG_LEV_HIGH,
	INPUT_SIGNAL	:= SOLAR_NOR-MALIZE_ANALOG_INPUT_SIGNAL);

1.5.11.2.5 ACTUATOR folder

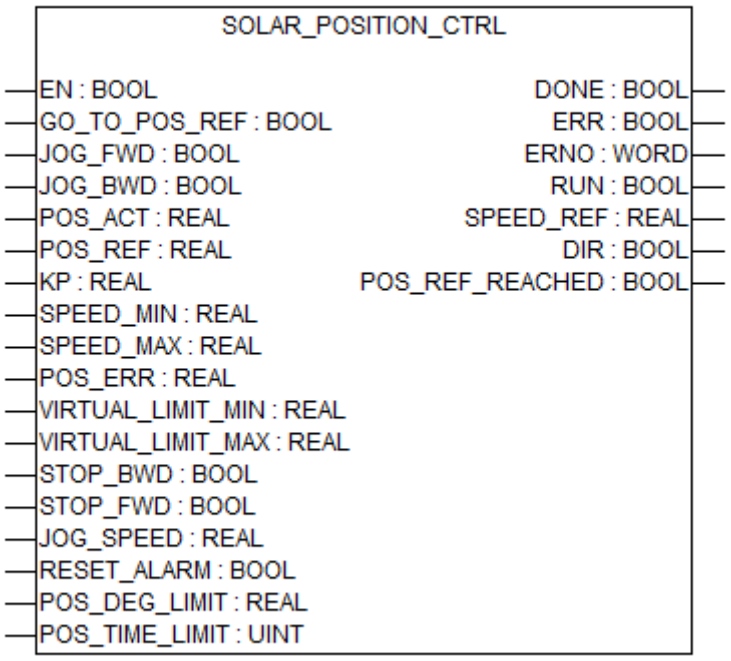
Preconditions for the use of the ACTUATOR folder

To communicate with ACS355 drive, use Chapter 1.5.6 “ACS / DCS drives libraries” on page 2192.

Refer example application for more details of configuration and programming.

Alternatively use Chapter 1.5.4.22 “Modbus library” on page 1697 also for communication with the drive.

SOLAR_POSITION_CTRL



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the position of the sun

Function block to manage the positioning control.

According to the positioning reference, it gets the speed parameters to set the system in the desired position. Output parameters are Run-Stop, Direction and Speed to reach the calculated position of the system.

Output parameters are Run, Direction and Speed to reach the desired position of the system.

Depends on the input parameters, the manipulated variable (SPEED) to control the actuators will be:

- GO_TO_POS_REF = TRUE

Select pulse start for positioning: Start by rising edge of a pulse. The signal has to stay TRUE during the positioning task.

According to the reference of positioning gets the SPEED parameter to set the system in this position, by calculating the control action by using a proportional controller.

SPEED_REF = (POS_REF-POS_ACT) * KP * (limit of [SPEED_MIN and SPEED_MAX]).

- JOG_FWD = TRUE

SPEED_REF = JOG_SPEED. To move the system forward manually.

- JOG_BWD = TRUE

SPEED_REF = (-1) * JOG_SPEED. To move the system backward manually.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

GO_TO_POS_REF BOOL (go to position reference)

In order to use the proportional control system to reach the SETPOINT, input GO_TO_POS_REF has to be set to TRUE.

JOG_FWD BOOL (jog forward)

In order to manually move the system forward, input JOG_FWD has to be set to TRUE. As long as JOG_FWD is set to TRUE, and STOP_FWD is deactivated, SPEED = JOG_SPEED.

VIRTUAL_LIMIT_MAX is not considered when JOG_FWD is activated.

JOG_BWD BOOL (jog backward)

In order to manually move the system backward, input JOG_ BWD has to be set to TRUE. As long as JOG_BWD is set to TRUE, and STOP_BWD is deactivated, SPEED = (-1) * JOG_SPEED.

VIRTUAL_LIMIT_MAX is not considered when JOG_BWD is activated.

POS_ACT REAL (position actual)

ACT indicates the current position of the system.

Format: Degrees.

POS_REF REAL (position reference)

SETPOINT indicates the desired position to locate the system.

Format: Degrees.

KP REAL (kp)

KP indicates the proportional gain of the proportional control system.

SPEED_MIN REAL (speed min)

Minimum speed of the drive, expressed as a percent of the maximum speed of the drive. This value is used during positioning control (GO_TO_POS_REF = TRUE).

This function block is having a speed range of 0 "“ 100 %. SPEED_MIN must be set between 0 "“ 100 %.

SPEED_MAX REAL (speed max)

Maximum speed of the drive, expressed as a percent of the maximum speed of the drive. This value is used during positioning control (GO_TO_POS_REF = TRUE).

This function block is having a speed range of 0 "“ 100 %. SPEED_MAX must be set between 0 "“ 100 %.

POS_ERR REAL (position error / position window)

Indicates the area where the system is considered correctly enough positioned (the control loop considers the POS_REF reached). The system is inside this window movement when the position error is smaller than POS_REF. The position error indicates the difference between the POS_REF and the POS_ACT.

POS_ERR value must be larger than resolution of the positioning devices (encoders or inclinometers).

VIRTUAL_LIMIT_MIN REAL (virtual limit minimum)

The lower limit of the movement range of the system. This value is only considered during positioning control (GO_TO_POS_REF = TRUE). JOG movements are not influenced by this limit value.

Format: Degrees.

VIRTUAL_LIMIT_MAX REAL (virtual limit maximum)

The upper limit of the movement range of the system. This value is only considered during positioning control (GO_TO_POS_REF = TRUE). JOG movements are not influenced by this limit value.

Format: Degrees.

STOP_BWD BOOL (stop backward)

This input should be used in order to integrate the lower limit switch signal which indicates the mechanical lower limit of the movement of the system.

If STOP_BWD = TRUE the mechanical lower limit of the movement range has been reached and the output RUN will be set to 0. Forward movement is allowed.

STOP_FWD BOOL (stop forward)

This input should be used in order to integrate the upper limit switch signal which indicates the mechanical upper limit of the movement of the system.

If STOP_FWD = TRUE the mechanical upper limit of the movement range has been reached and the output RUN will be set to 0. Backward movement is allowed.

JOG_SPEED REAL (jog speed)

JOG_SPEED indicates the JOG speed of the drive. This value is expressed as a percent of the maximum speed of the drive.

During a JOG movement this value will set to the SPEED output.

This function block is having a speed range of 0 "“ 100 %. JOG_SPEED must be set between 0 "“ 100 %.

RESET_ALARM BOOL (reset alarm / error)

Boolean signal that resets the ERR and ERNO outputs.

POS_DEG_LIMIT REAL (position degrees limit)

When the primary axis of the actuator (gear or hydraulic piston) changes its direction of rotation, this axis covers an angular distance, POS_DEG_LIMIT, whereas the tracker will not change its position. Only when POS_DEG_LIMIT is reached, the tracker starts its movement. This concept is similar to "backlash" in gear trains.

Format: Degrees.

This value depends on each tracker and its actuator (motors, gears or hydraulic systems).

POS_ACT (t) - POS_ACT (t + POS_TIME_LIMIT) | > POS_DEG_LIMIT,

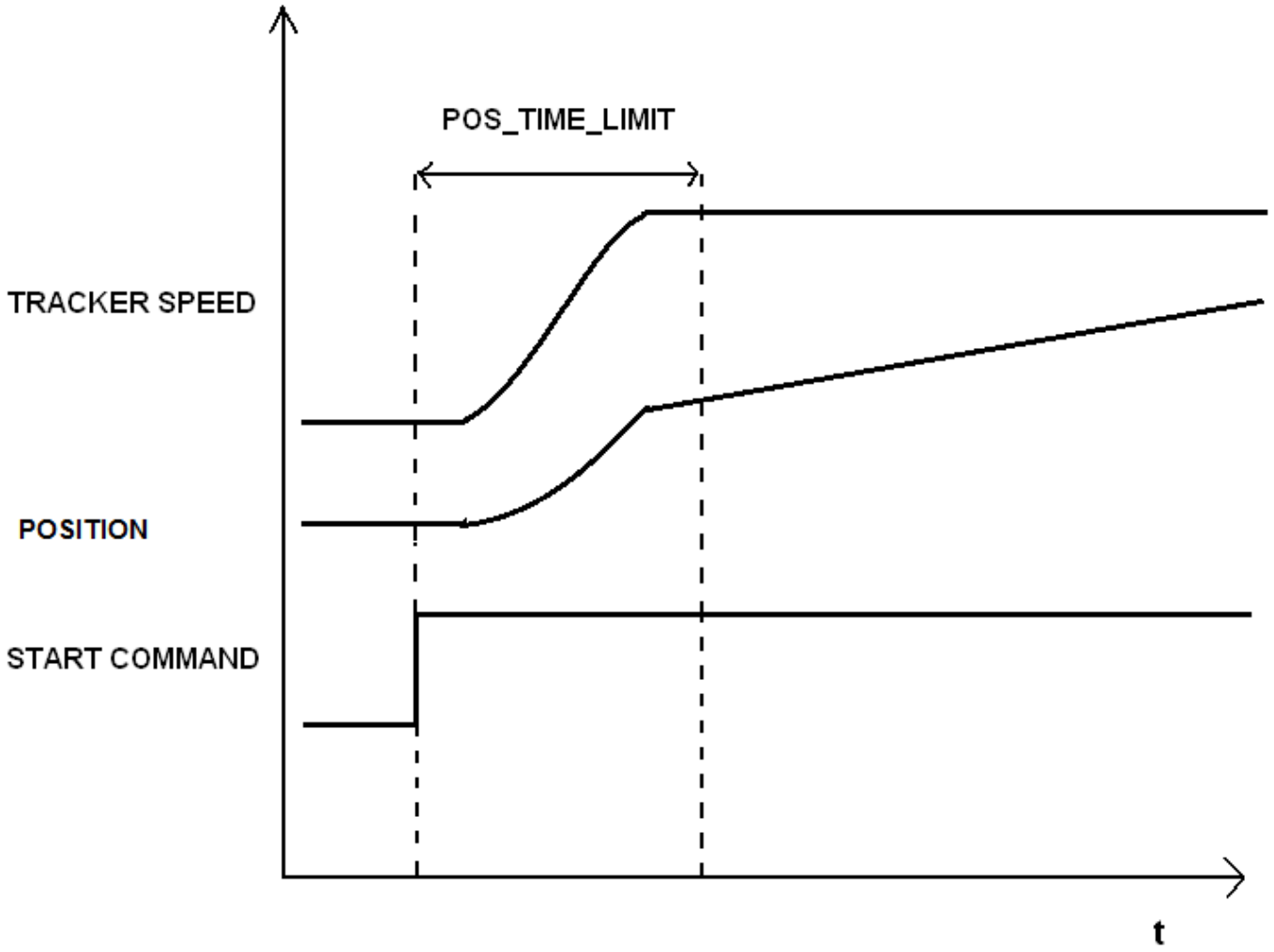
where:

t = time.

POS_TIME_LIMIT UNIT (position time limit)

Indicates the needed time to cover POS_DEG_LIMIT distance without any fault. The expected distance must be covered in a time smaller than POS_TIME_LIMIT milliseconds. Otherwise, it generates error and corresponding error is displayed.

Format: Milliseconds.



Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

RUN BOOL (run)

RUN output should be used as start command for the drive. It is set to TRUE, if no limit is reached no error occurred and either a jog command is set or normal operation sets the SPEED_REF not to zero.

SPEED_REF REAL (speed reference)

To output parameter SPEED_REF indicates the input to the drive, the manipulated variable to control the system positioning. Depending on the working mode SPEED_REF will be:

Input Parameters (Working mode)	Additional Input Parameters Considered	Output SPEED_REF
GO_TO_POS_REF = TRUE	POS_ERR < (POS_REF "“ POS_ACT) VIRTUAL_LIMIT_MIN < POS_ACT < VIR-TUAL_LIMIT_MAX STOP_BWD = FALSE STOP_FWD = FALSE	SPEED_REF = KP*(POS_REF - POS_ACT)
JOG_FWD = TRUE	STOP_FWD = FALSE	SPEED_REF = JOG_SPEED
JOG_BWD = TRUE	STOP_BWD = FALSE	SPEED = (-1) * JOG_SPEED

SPEED_REF is in terms of 0-100 %. User needs to convert this output to 0-50 Hz before giving to the drive.

DIR BOOL (direction)

If SPEED >= 0 -> DIR = TRUE

If SPEED < 0 -> DIR = FALSE

POS_REF_REACHED BOOL (position reference reached)

POS_REF_REACHED is set to TRUE as long as the system has reached the POS_REF or it is inside the area between POS_ERR > (POS_REF - POS_ACT).

Function call in IL

CAL PosControL(		
	EN	:= PosControl_EN,
	GO_TO_POS_REF	:= PosCon-trol_GO_TO_POS_REF,
	JOG_FWD	:= PosControl_JOG_FWD,
	JOG_BWD	:= PosControl_JOG_BWD,
	POS_ACT	:= PosControl_POS_ACT,
	POS_REF	:= PosControl_POS_REF,
	KP	:= PosControl_KP,
	SPEED_MIN	:= PosControl_SPEED_MIN,

	SPEED_MAX	:= PosControl_SPEED_MAX,
	POS_REF	:= PosControl_POS_REF,
	VIRTUAL_LIMIT_MIN	:= PosControl_SPEED_MIN,
	VIRTUAL_LIMIT_MAX	:= PosControl_SPEED_MAX,
	STOP_BWD	:= PosControl_STOP_BWD,
	STOP_FWD	:= PosControl_STOP_FWD,
	JOG_SPEED	:= PosControl_JOG_SPEED,
	RESET_ALARM	:= PosCon- trol_RESET_ALARM,
	POS_DEG_LIMIT	:= PosCon- trol_POS_DEG_LIMIT,
	POS_TIME_LIMIT	:= PosCon- trol_POS_TIME_LIMIT);
LD		PosControl.DONE
ST		PosControl_DONE
LD		PosControl.ERR
ST		PosControl_ERR
LD		PosControl.ERNO
ST		PosControl_ERNO
LD		PosControl.RUN
ST		PosControl_RUN
LD		PosControl.SPEED_REF
ST		PosControl_SPEED_REF
LD		PosControl.DIR
ST		PosControl_DIR
LD		PosCon- trol.POS_REF_REACHED
ST		PosCon- trol_POS_REF_REACHED

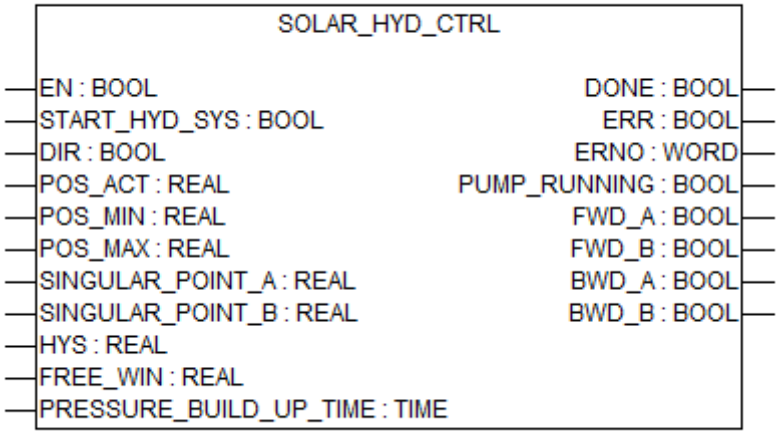
In IL, the function call has to be written in one line.

Function call in ST

PosControl(		
	EN	:= PosControl_EN,
	GO_TO_POS_REF	:= PosCon- trol_GO_TO_POS_REF,
	JOG_FWD	:= PosControl_JOG_FWD,
	JOG_BWD	:= PosControl_JOG_BWD,

	POS_ACT	:= PosControl_POS_ACT,
	POS_REF	:= PosControl_POS_REF,
	KP	:= PosControl_KP,
	SPEED_MIN	:= PosControl_SPEED_MIN,
	SPEED_MAX	:= PosControl_SPEED_MAX,
	POS_ERR	:= PosControl_POS_ERR,
	VIRTUAL_LIMIT_MIN	:= PosControl_VIR-TUAL_LIMIT_MIN,
	VIRTUAL_LIMIT_MAX	:= PosControl_VIR-TUAL_LIMIT_MAX,
	STOP_BWD	:= PosControl_STOP_BWD,
	STOP_FWD	:= PosControl_STOP_FWD,
	JOG_SPEED	:= PosControl_JOG_SPEED,
	RESET_ALARM	:= PosCon-trol_RESET_ALARM,
	POS_DEG_LIMIT	:= PosCon-trol_POS_DEG_LIMIT,
	POS_TIME_LIMIT	:= PosCon-trol_POS_TIME_LIMIT);
	PosControl_DONE	:= PosControl.DONE;
	PosControl_ERR	:= PosControl.ERR;
	PosControl_ERNO	:= PosControl.ERNO;
	PosControl_RUN	:= PosControl.RUN;
	PosControl_SPEED_REF	:= PosControl.SPEED_REF;
	PosControl_DIR	:= PosControl.DIR;
	PosCon-trol_POS_REF_REACHED	:= PosCon-trol.POS_REF_REACHED;

SOLAR_HYD_CTRL



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function
Group	Package of functions to get the position of the sun

Function block to manage two pistons as actuator device.

- Hydraulic system to move the tracker.

Hydraulic system is based in the action of two hydraulic pistons (Piston A and Piston B) that are controlled by four solenoid valves. Each hydraulic piston has two solenoid valves to control its movement. These solenoid valves are two position valves.

A1: Backward piston A (piston A reduces its length).

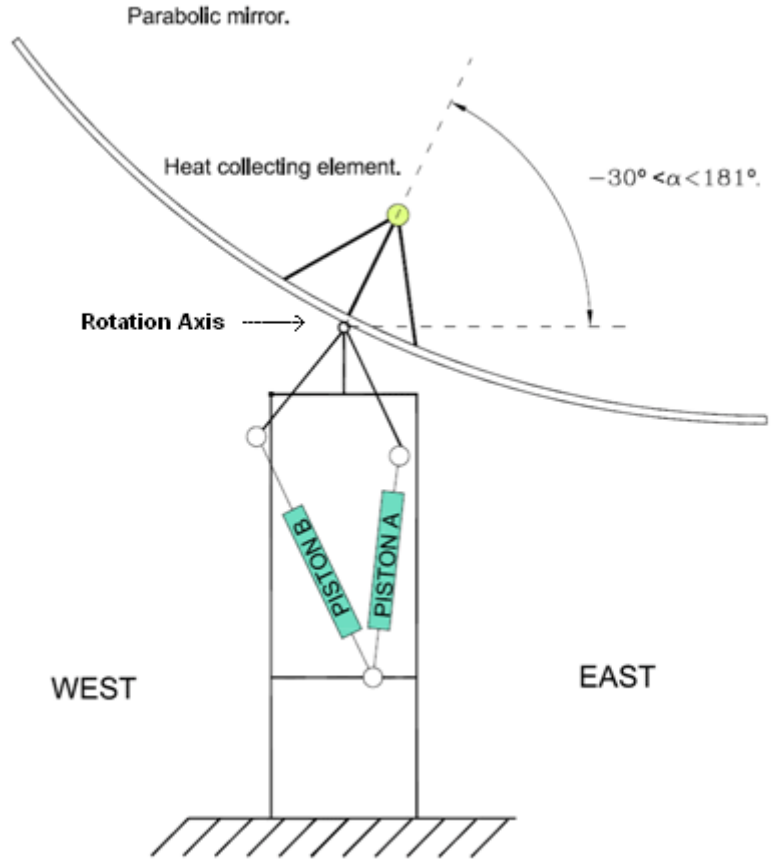
A2: Forward piston A (piston A increases its length).

B1: Backward piston B (piston B reduces its length).

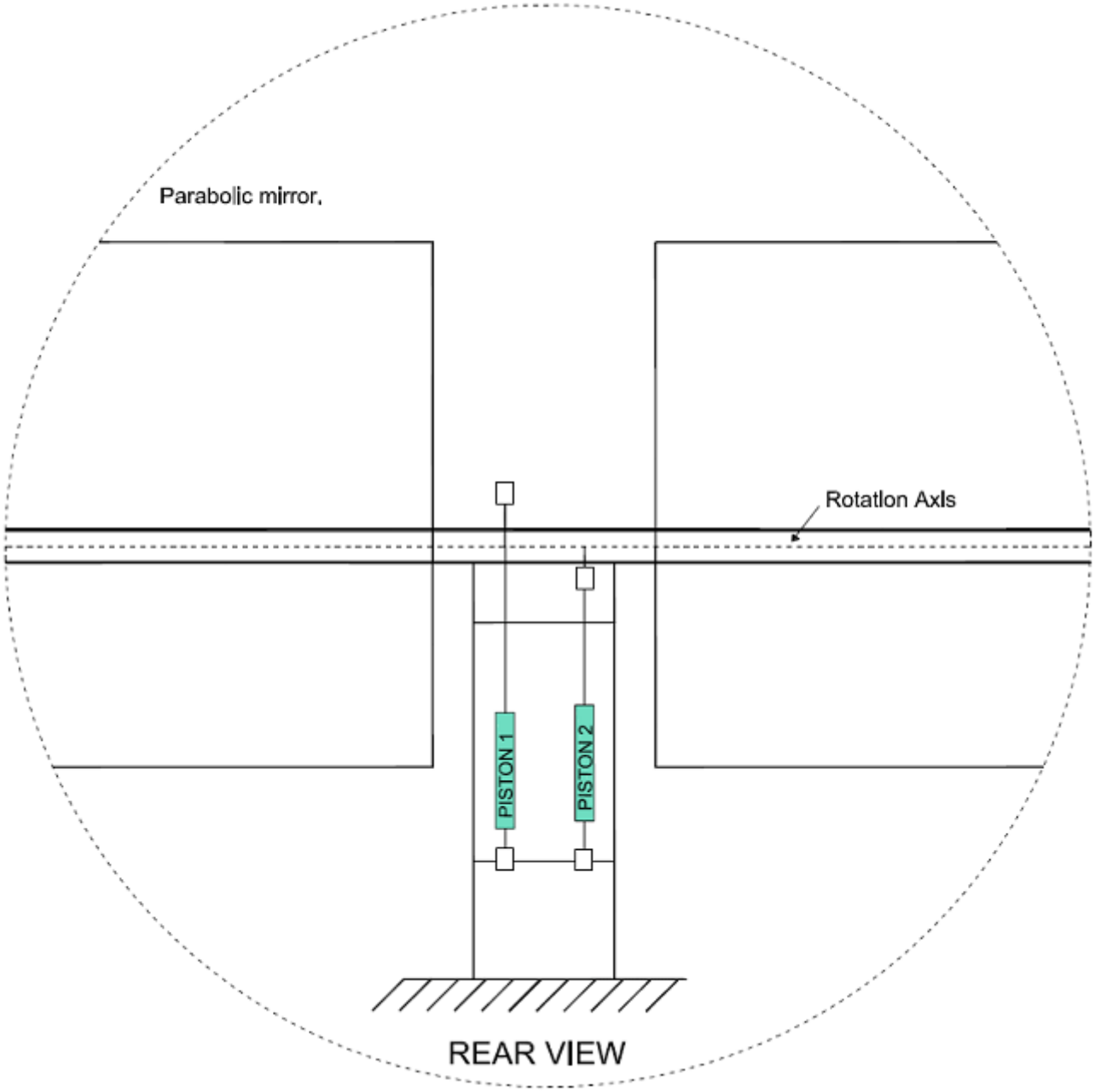
B2: Forward piston B (piston B increases its length).

Hydraulic system also has an oil pump to get pressure into the hydraulic circuit. This pumps works for a few periods of time, it doesnt work continuously.

Movement range of tracker [-30° - 181°]:



Movement cycle of the hydraulic pistons:



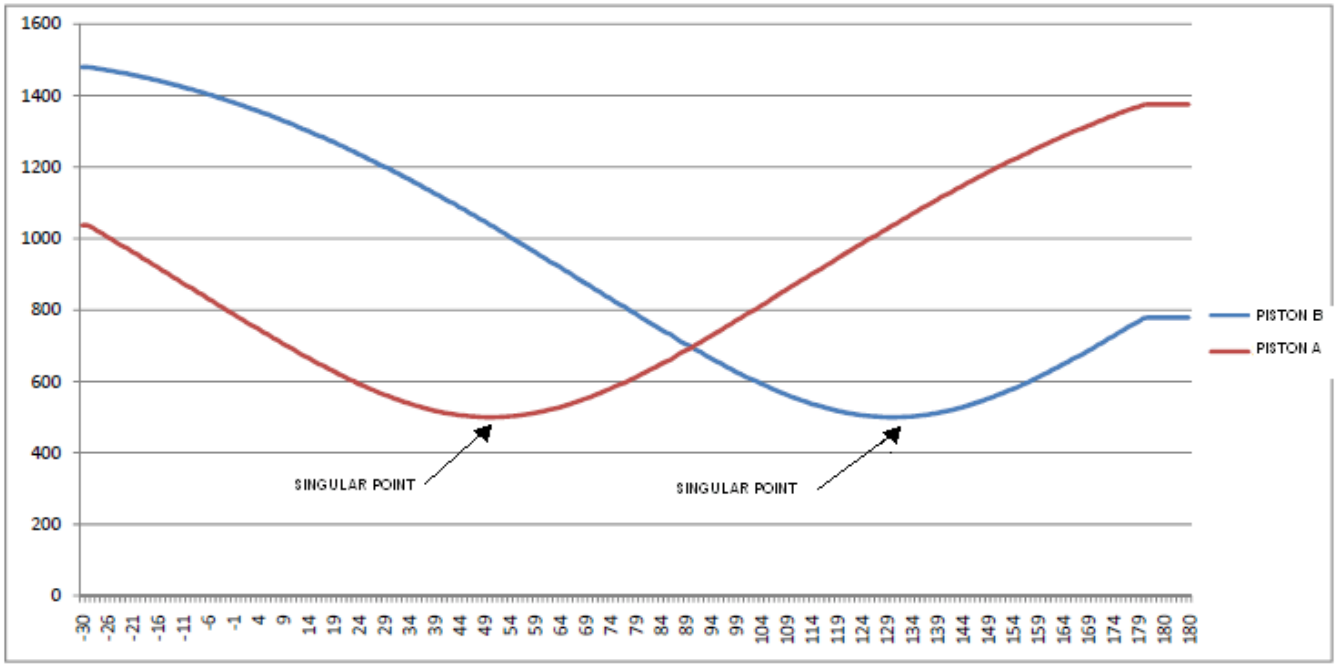
Movement cycle of the hydraulic piston is the following:

During a part of the movement cycle, both pistons are running in the same direction (both are pulling or pushing at the same time).

During a part of the movement cycle, one piston has reached its singular point. In this position, the piston does not generate any torque in the system.

During a part of the movement cycle, both pistons are running in opposite directions (one of them is pulling and the other one is pushing at the same time).

Figure below shows the movement cycle of the hydraulic pistons:



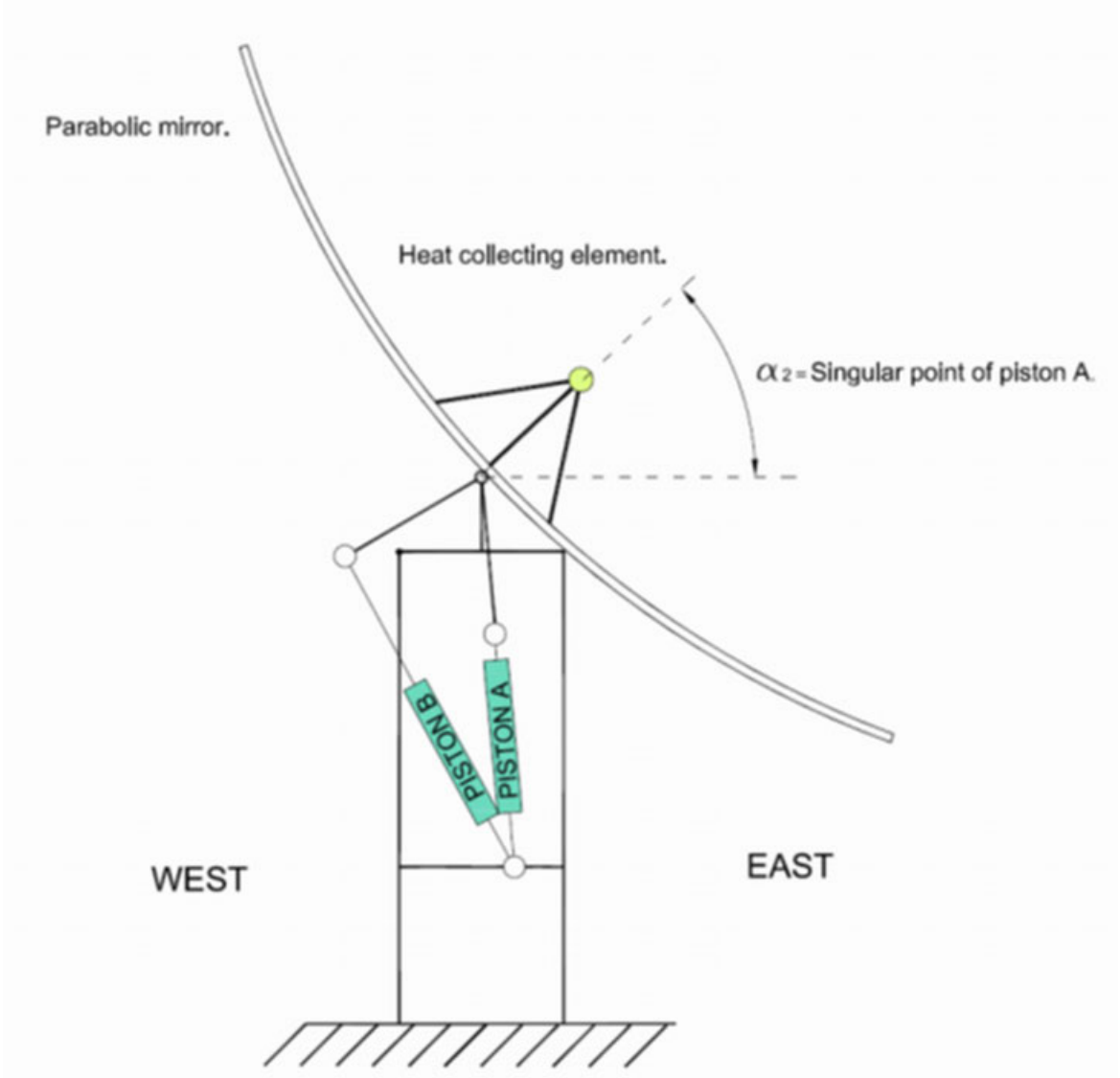
X-axis represents the allowed position range of Solar Tracker (these values are configurable by the user).

Y-axis represents the length of the hydraulic pistons.

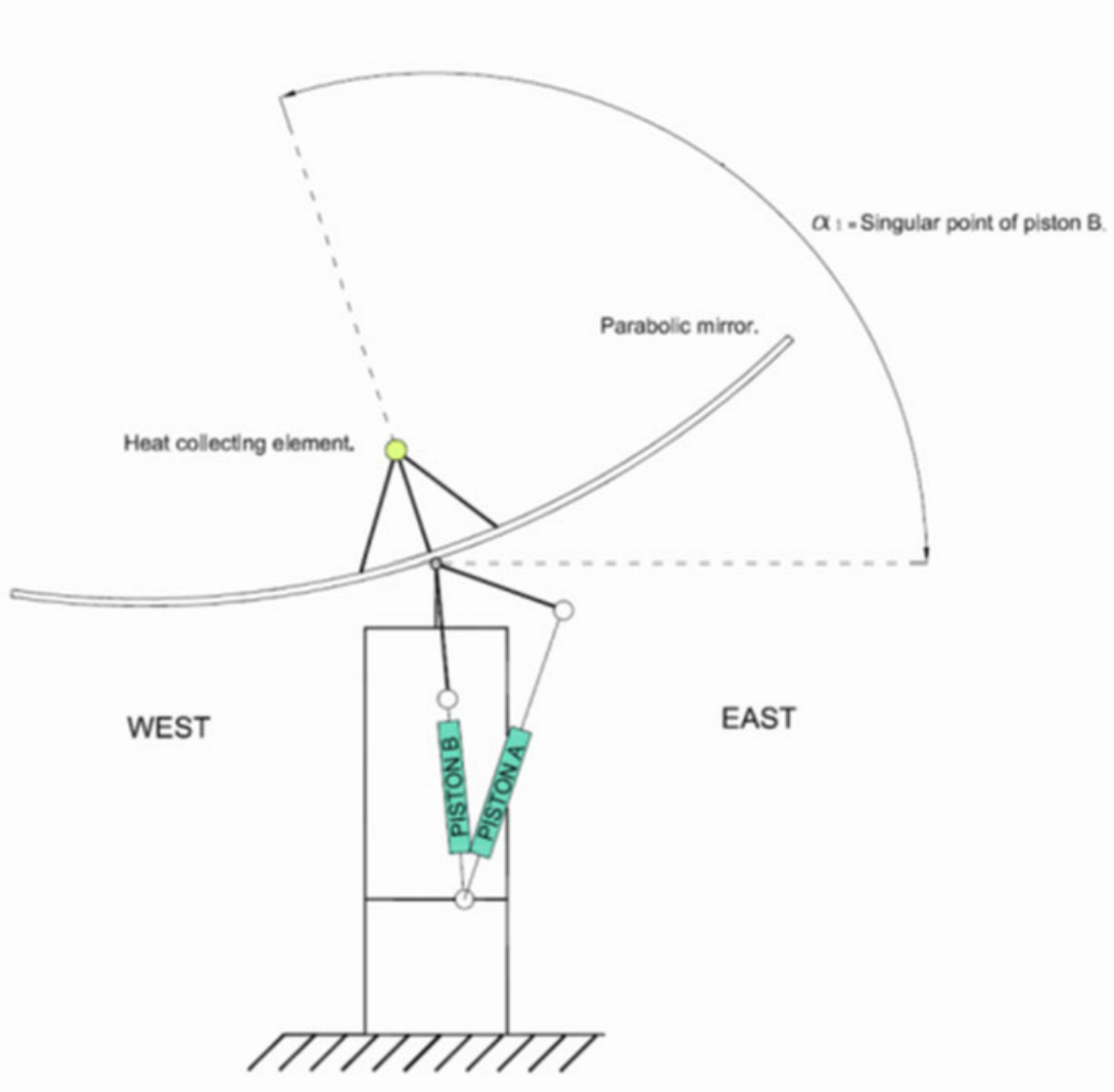
In this case, singular points are:

- Piston A: 49°.
- Piston B: 130°.

The singular point for piston A. Its value is ±2:



The singular point for piston B. Its value is ±1:



Movement control: To control the movement of the hydraulic pistons, it is necessary to operate on the solenoid valves. Depending on tracker position and direction of rotation, from east to west or from west to east, it is necessary to operate on determined solenoid valves.

Table below shows sequence of solenoid valves.

	SOLENOID VALVES			
Tracker Position Range	**A1**	**A2**	**B1**	**B2**
-30º < α < 49º-σ	1	0	1	0
49º-σ < α < 49º+σ	0 (FREE-WINDOW)	0 (FREE-WINDOW)	1	0
49º+σ < α < 130º - σ	0	1	1	0
130º- σ< α < 130º + σ	0	1	0 (FREE-WINDOW)	0 (FREE-WINDOW)
130º+σ < α < 181	0	1	0	1

Where:

A1: backward piston A A2: forward piston A.

B1: backward piston B B2: forward piston B.

1: solenoid valve is ON 0: solenoid valve is OFF.

Ïf: free-window (degrees) ±: angular position of the Solar Tracker (degrees).

> *All range position values that appear in the table above are configurable and depend on each tracker. A previous geometrical analysis is needed due to there are many kind of trackers. You can use this table as an example.*

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START_HYD_SYS BOOL (start hydraulic system) Boolean input that enables SOLAR_HYD_CTRL block.

DIR BOOL (direction) Boolean input that indicates the direction movement of Solar Tracker.

TRUE: = forward, FALSE: = backward.

POS_MIN REAL (position minimum) Lower limit of the movement range.

Format: Degrees.

POS_MAX REAL (maximum position) Upper limit of the movement range.

Format: Degrees.

SINGULAR_POINT_A REAL (singular_point_A) Indicates a position of the Solar Tracker (angular position). In that position, the mechanical structure of Solar Tracker shows a singular point concerning to piston A. In that situation, piston A does not generate any torque value. This value is given by tracker manufacturer.

Format: Degrees.

SINGULAR_POINT_B REAL (singular_point_B) Indicates a position of the Solar Tracker (angular position). In that position, the mechanical structure of Solar Tracker shows a singular point concerning to piston B. In that situation, piston B does not generate any torque value. This value is given by tracker manufacturer.

Format: Degrees.

HYS REAL (hysteresis) Hysteresis value to avoid unnecessary piston movement. This value is obtained by testing the behavior of the Solar Tracker.

Format: Degrees.

FREE_WIN REAL (free window) Value of semi angle in which hydraulic piston does not generate any torque value. Movement of hydraulic piston is free when is placed inside this window. This value is obtained by testing the behavior of the Solar Tracker.

Format: Degrees.

PRESSURE_BUILD_UP_TIME TIME (pressure build up time)

Time to achieve the correct hydraulic pressure in the circuit.

Format: Seconds.

Standard: 500ms.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

PUMP_RUNNING BOOL (pump running)

Indicates that pump is working.

FWD_A BOOL (forward a)

Indicates that hydraulic piston A is running forward.

FWD_B BOOL (forward b)

Indicates that hydraulic piston B is running forward.

BWD_A BOOL (backward a)

Indicates that hydraulic piston A is running backward.

BWD_B BOOL (backward b)

Indicates that hydraulic piston B is running backward.

Function call in IL

CAL HydControl(		
	EN	:= HydControl_EN,
	START_HYD_SYS	:= HydControl_START_HYD_SYS,
	DIR	:= HydControl_DIR,

	POS_ACT	:= HydControl_POS_ACT,
	POS_MIN	:= HydControl_POS_MIN,
	POS_MAX	:= HydControl_POS_MAX,
	SINGULAR_POINT_A	:= HydControl_SIN-GULAR_POINT_A,
	SINGULAR_POINT_B	:= HydControl_SIN-GULAR_POINT_B,
	HYS	:= HydControl_HYS,
	FREE_WIN	:= HydControl_FREE_WIN,
	PRES-SURE_BUILD_UP_TIME	:= HydControl_PRES-SURE_BUILD_UP_TIME);
LD		HydControl.DONE
ST		HydControl_DONE
LD		HydControl.ERR
ST		HydControl_ERR
LD		HydControl.ERNO
ST		HydControl_ERNO
LD		HydControl.PUMP_RUNNING
ST		HydControl_PUMP_RUN-NING
LD		HydControl.FWD_A
ST		HydControl_FWD_A
LD		HydControl.FWD_B
ST		HydControl_FWD_B
LD		HydControl.BWD_A
ST		HydControl_BWD_A
LD		HydControl.BWD_B
ST		HydControl_BWD_B

In IL, the function call has to be written in one line.

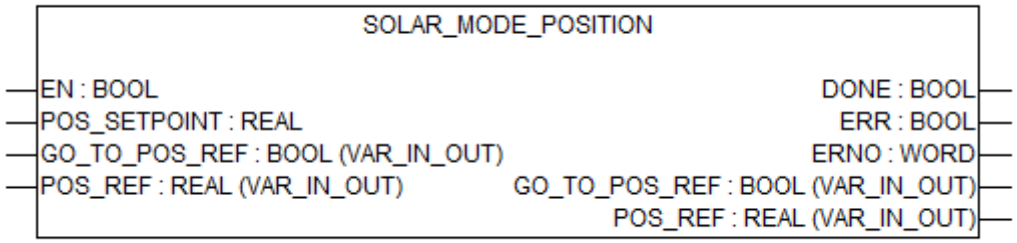
Function call in ST

HydControl(		
	EN	:= HydControl_EN,
	START_HYD_SYS	:= HydCon-trol_START_HYD_SYS,
	DIR	:= HydControl_DIR,
	POS_ACT	:= HydControl_POS_ACT,

	POS_MIN	:= HydControl_POS_MIN,
	POS_MAX	:= HydControl_POS_MAX,
	SINGULAR_POINT_A	:= HydControl_SIN-GULAR_POINT_A,
	SINGULAR_POINT_B	:= HydControl_SIN-GULAR_POINT_B,
	HYS	:= HydControl_HYS,
	FREE_WIN	:= HydControl_FREE_WIN,
	PRES-SURE_BUILD_UP_TIME	:= HydControl_PRES-SURE_BUILD_UP_TIME);
	HydControl_DONE	:= HydControl.DONE;
	HydControl_ERR	:= HydControl.ERR;
	HydControl_ERNO	:= HydControl.ERNO;
	HydControl_PUMP_RUN-NING	:= HydControl.PUMP_RUN-NING;
	HydControl_FWD_A	:= HydControl.FWD_A;
	HydControl_FWD_B	:= HydControl.FWD_B;
	HydControl_BWD_A	:= HydControl.BWD_A;
	HydControl_BWD_B	:= HydControl.BWD_B;

1.5.11.2.6 MODE folder

SOLAR_MODE_POSITION



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks for different control modes

This function block represents an operation mode that allows to move the Solar Tracker to a position defined by the user. It shall keep the Solar Tracker in this position awaiting for new commands or new position value.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

POS_SETPOINT REAL (position setpoint) Position value defined by the user.

GO_TO_POS_REF BOOL (go to position reference) Boolean expression that indicates the process is able to move Solar Tracker.

POS_REF REAL (position_reference) Desired position to locate the tracker.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Function call in IL

CAL ModePosition (		
	EN	:= ModePosition_EN,
	POS_SETPOINT	:= ModePosition_POS_SETPOINT,

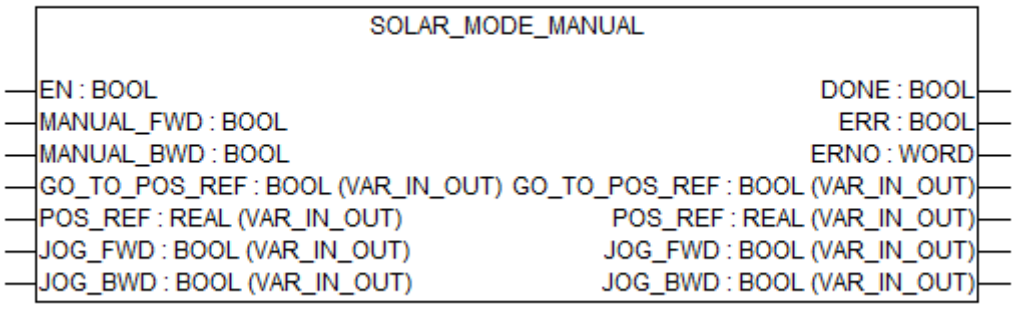
	GO_TO_POS_REF	:= ModePosi- tion_GO_TO_POS_REF,
	POS_REF	:= ModePosition_POS_REF)
LD		ModePosition.DONE
ST		ModePosition_DONE
LD		ModePosition.ERR
ST		ModePosition_ERR
LD		ModePosition.ERNO
ST		ModePosition_ERNO

In IL, the function call has to be written in one line.

Function call in ST

ModePosition (		
	EN	:= ModePosition_EN,
	POS_SETPOINT	:= ModePosition_POS_SET- POINT,
	GO_TO_POS_REF	:= ModePosi- tion_GO_TO_POS_REF,
	POS_REF	:= ModePosition_POS_REF);
	ModePosition_DONE	:= ModePosition.DONE;
	ModePosition_ERR	:= ModePosition.ERR;
	ModePosition_ERNO	:= ModePosition.ERNO;

SOLAR_MODE_MANUAL



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks for different control modes

This function block represents an operation mode that allows to set the system in manual mode. User is able to move the system by using drives directly. While this mode is active, system will not respond to any commands from the solar control field.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

MANUAL_FWD BOOL (manual forward) In order to move manually the system forward, input JOG_FWD has to be set to TRUE.

MANUAL_BWD BOOL (manual backward) In order to move manually the system backward, input JOG_BWD has to be set to TRUE.

GO_TO_POS_REF BOOL (go to position reference) Boolean expression that enables position control block.

POS_REF REAL (position_reference) Desired position to locate the tracker.

JOG_FWD BOOL (jog forward) Boolean expression that indicates that drive is running forward.

JOG_BWD BOOL (jog backward) Boolean expression that indicates that drive is running backward.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Function call in IL

CAL ModeManual (		
	EN	:= ModeManual_EN,
	MANUAL_FWD	:= ModeManual_MANUAL_FWD,
	MANUAL_BWD	:= ModeManual_MANUAL_BWD,
	GO_TO_POS_REF	:= ModeManual_GO_TO_POS_REF,
	POS_REF	:= ModeManual_POS_REF,
	JOG_FWD	:= ModeManual_JOG_FWD,
	JOG_BWD	:= ModeManual_JOG_BWD)
	LD	ModeManual.DONE
	ST	ModeManual_DONE
	LD	ModeManual.ERR
	ST	ModeManual_ERR
	LD	ModeManual.ERNO
	ST	ModeManual_ERNO

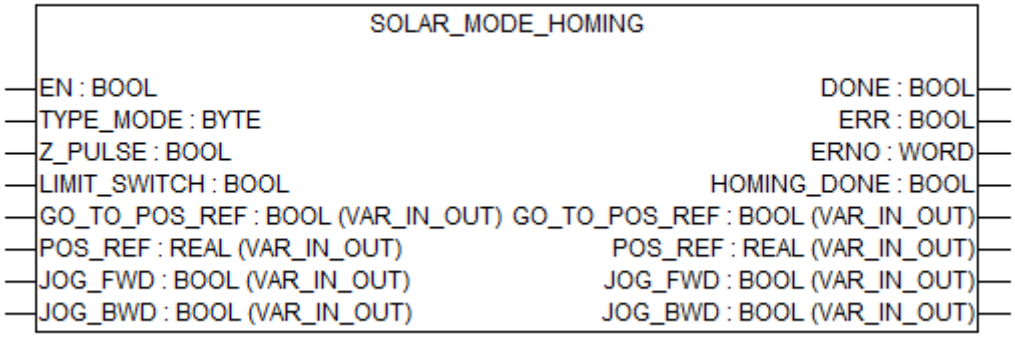
In IL, the function call has to be written in one line.

Function call in ST

ModeManual (		
	EN	:= ModeManual_EN,
	MANUAL_FWD	:= ModeManual_MANUAL_FWD,

	MANUAL_BWD	:= ModeManual_MANUAL_BWD,
	GO_TO_POS_REF	:= ModeManual_GO_TO_POS_REF,
	POS_REF	:= ModeManual_POS_REF,
	JOG_FWD	:= ModeManual_JOG_FWD,
	JOG_BWD	:= ModeManual_JOG_BWD)
	ModeManual_DONE	:= ModeManual.DONE;
	ModeManual_ERR	:= ModeManual.ERR;
	ModeManual_ERNO	:= ModeManual.ERNO;

SOLAR_MODE_HOMING



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks for different control modes

Set the system to look for the reference point initialization (Z).

This function block represents an operation mode that allows to search the reference point initialization. Systems moves until reference point is reached. This mode assumes that system has a high accuracy mechanical limit switch to indicate that home position has been reached.

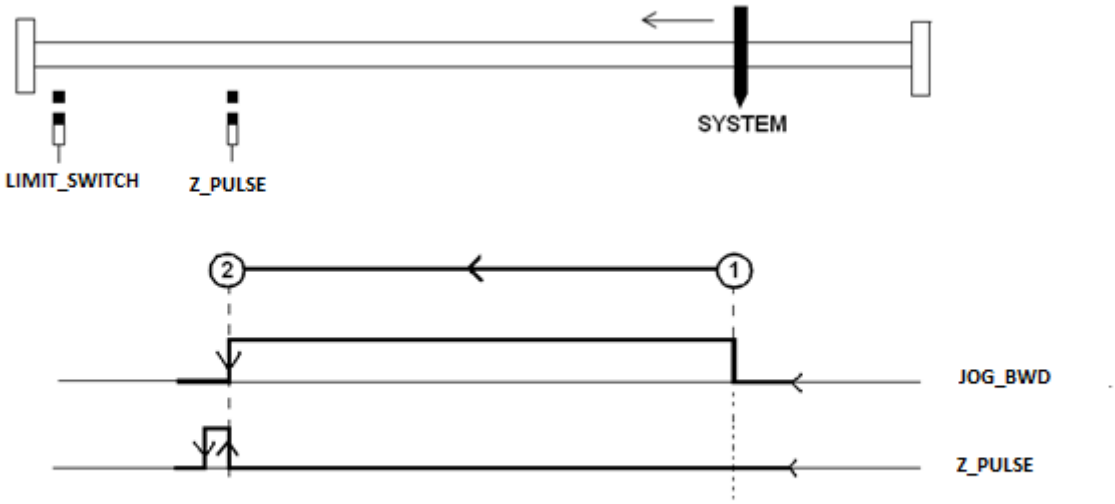
Figure below indicates how this operation mode works. In this document there are explained four basics systems according to:

- Type of device to indicate reference point initialization.
- Direction of movement. (Negative direction left and positive direction to the right).

Modules DC541 and DC522 use rising edge to synchronization and initialization process, so this four types use rising edge to calculate home position.

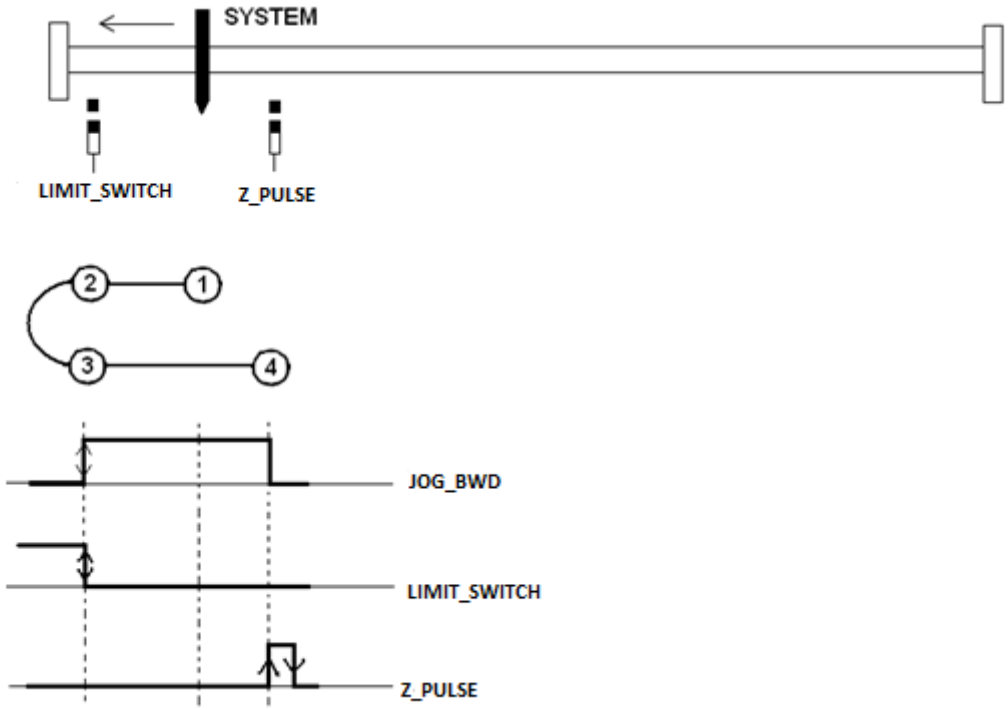
Type 1: Using encoder reference point Z_PULSE and limit switch to determinate home position.

First movement: Negative direction (left):



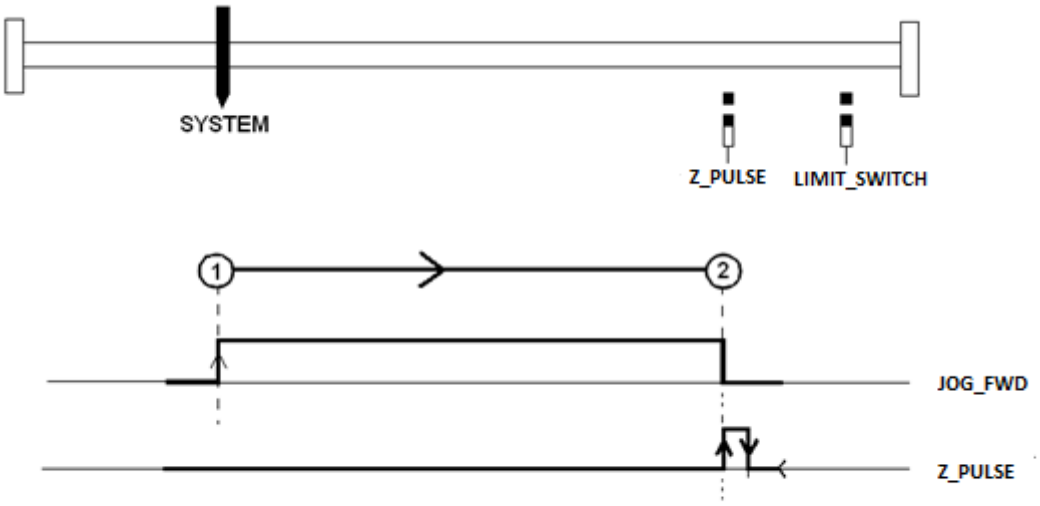
At the beginning, system is located at point 1 and limit switch is not active yet. When SOLAR_MODE_HOMING is enabled, JOG_BWD is set to TRUE. As soon as Z_PULSE shows rising edge, home position has been reached and system stops its movement at point 2.

If system is located between Z_ENCODER and LIMIT_SWITCH positions, SOLAR_MODE_HOMING operates as figure below shows:



Type 2: Using encoder reference point Z_PULSE mechanical limit switch to determinate home position.

First movement: Positive direction (right):



At the beginning, system is located at point 1 and limit switch is not active yet. When SOLAR_MODE_HOMING is enabled, JOG_FWD is set to TRUE. As soon as Z_PULSE shows rising edge, home position has been reached and system stops its movement at point 2.

If system is located between Z_ENCODER and LIMIT_SWITCH positions, SOLAR_MODE_HOMING operates as figure below shows:

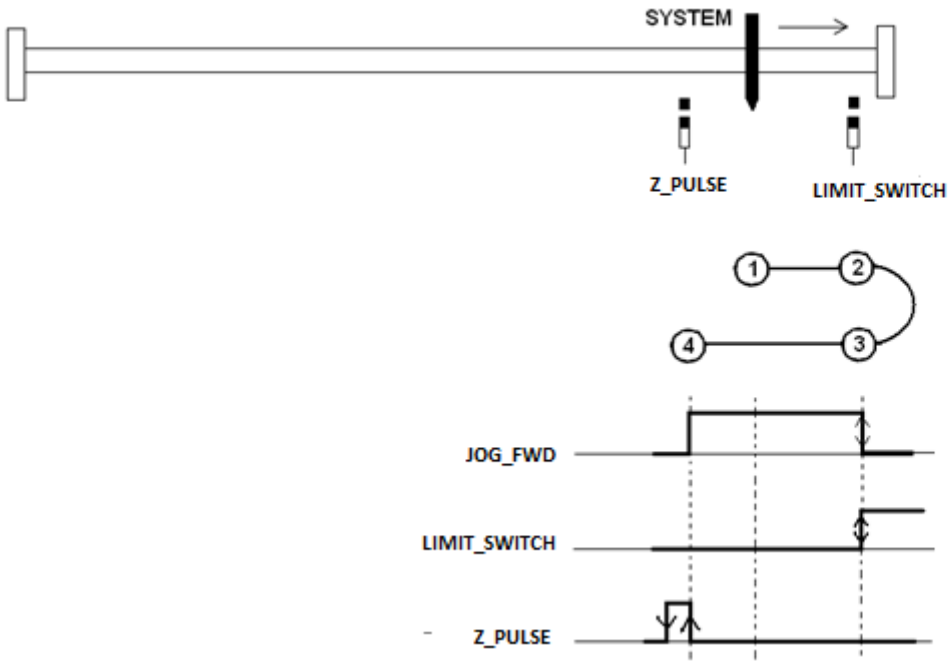
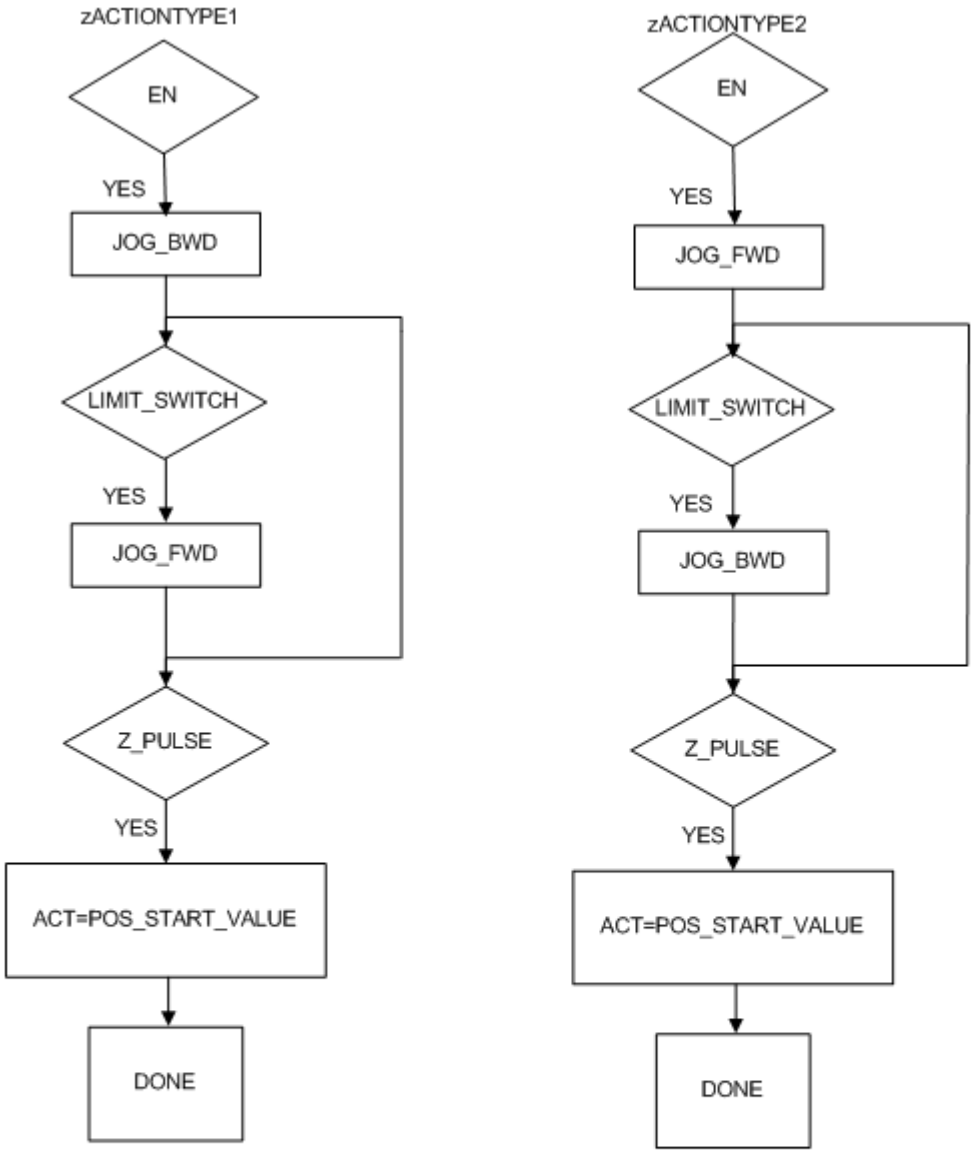


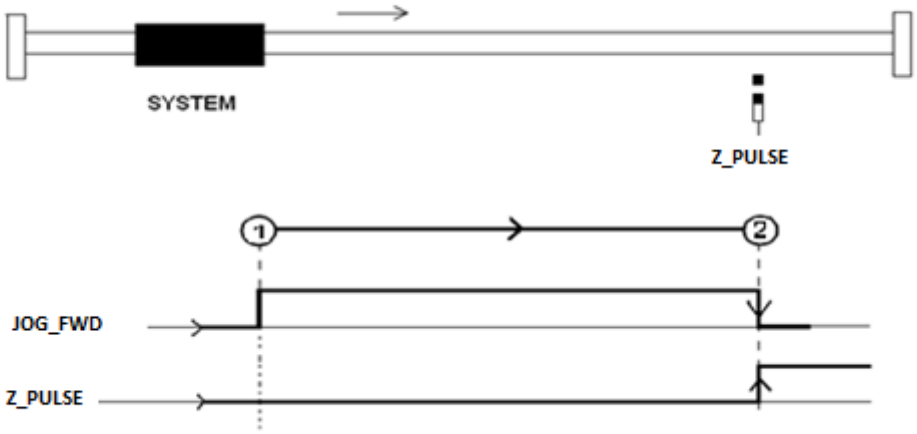
Figure below shows the logical sequence that implements SOLAR_MODE_HOMING Type 1 & Type 2:

NEGATIVE Direction (left)	POSITIVE Direction (right)

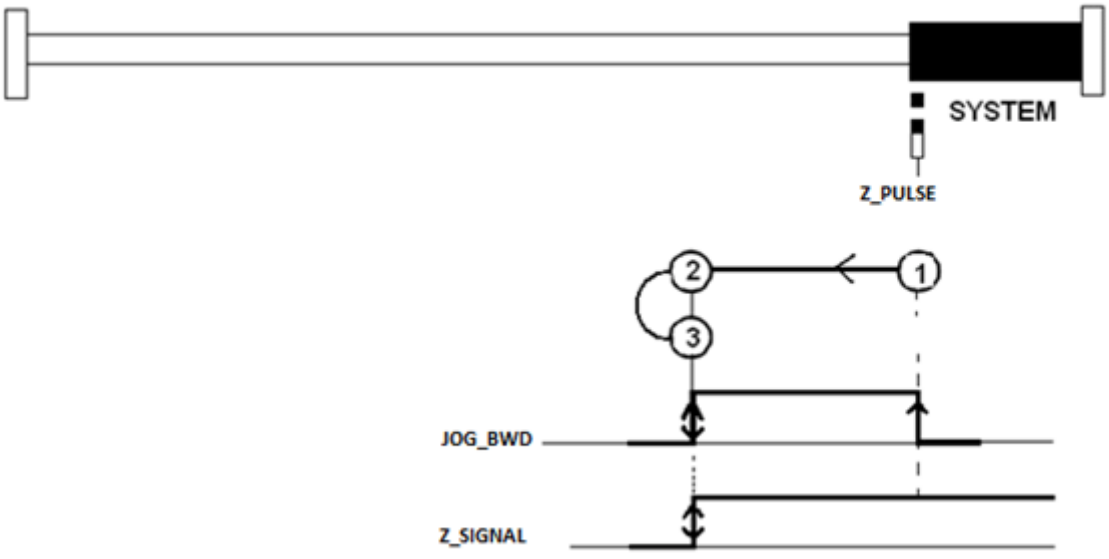


Type 3: Using Z_PULSE and cam switch to determinate home position.

First movement: Positive direction (right):

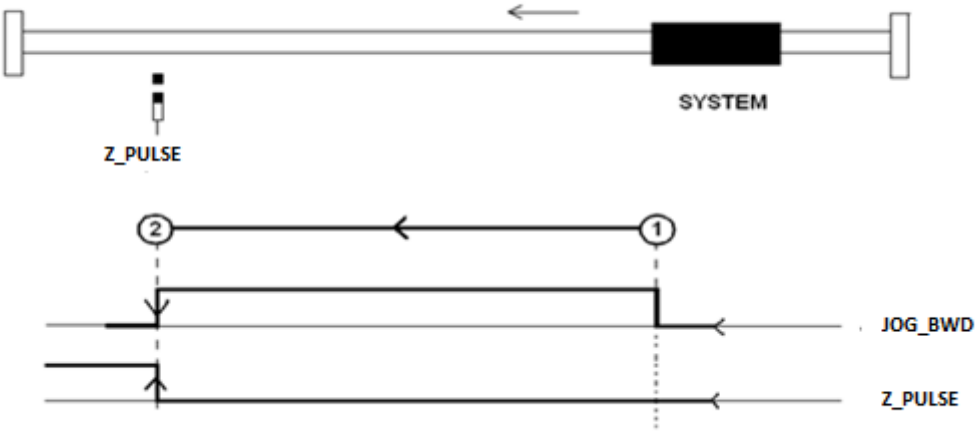


Home position has been reached when rising edge of the signal Z_PULSE is detected. Then, movement is stopped. On the other hand, if system is located at the end of its movement range and cam switch signal is set to TRUE (see figure below) home position is reached as follow:



Type 4: Using a cam switch to determinate home position.

First movement: Negative direction (left):



Home position has been reached when rising edge of the signal Z_PULSE is detected. Then, movement is stopped. On the other hand, if system is located at the end of its movement range and cam switch signal is set to TRUE (see figure below) home position is reached as follow:

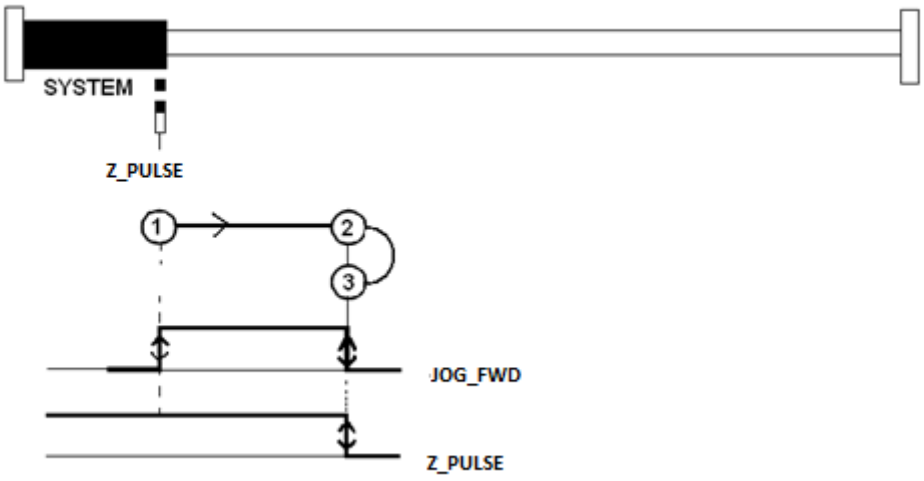
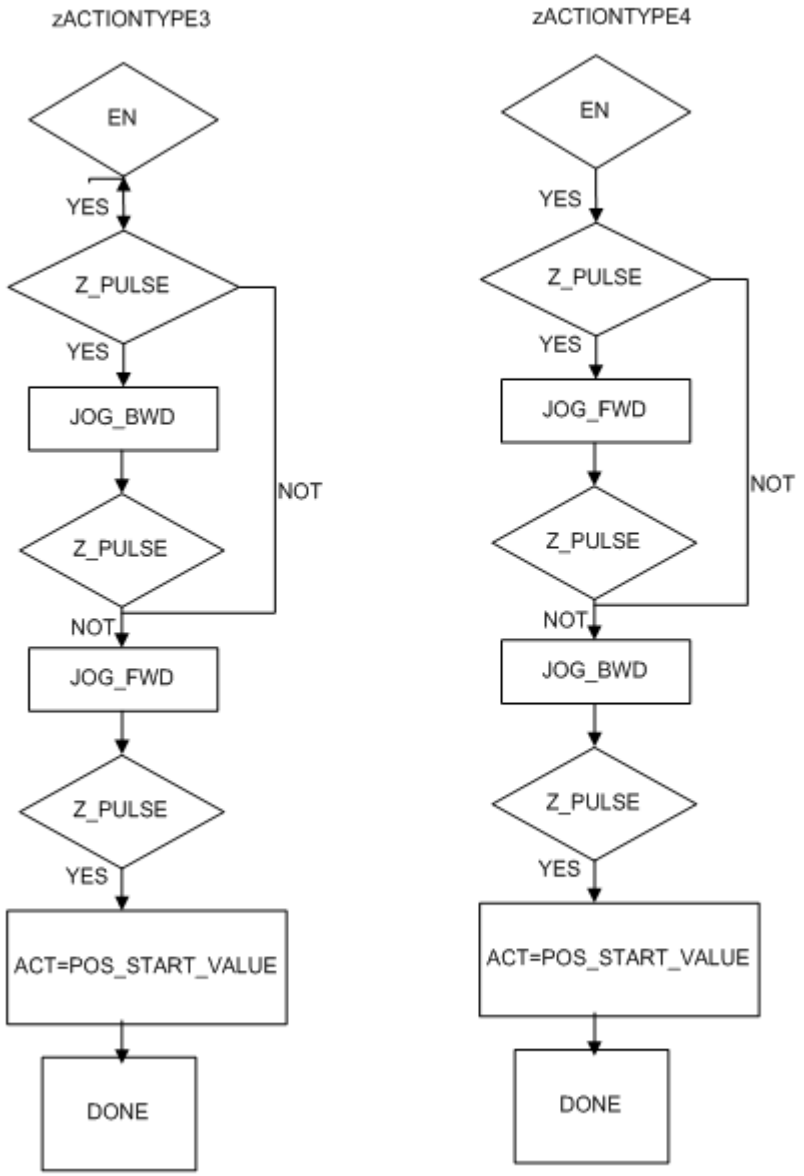


Figure below shows the logical sequence that implements SOLAR_MODE_HOMING Type 3 & Type 4:

POSITIVE Direction (right)	NEGATIVE Direction (left)



Input description

EN BOOL (enable block)

Homing process starts as soon as rising edge is detected at signal EN.

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

While input EN is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

TYPE_MODE BYTE (type mode)

This input indicates which kind of configuration sensors are going to be used to determinate home positions (see figures above).

1 -> Z_Sensor is only one pulse signal. First movement: backward.

2 -> Z_Sensor is only one pulse signal. First movement: forward.

3 -> Z_Sensor uses cam switch. First movement: forward.

4 -> Z_Sensor uses cam switch. First movement: backward.

Z_PULSE BOOL (z pulse)

Boolean expression that indicates the positioning device has reached its Z position.

TYPE 1 & 2 -> Z_PULSE = RDY_SYNC (output variable of encoders function).

TYPE 3 & 4 -> Z_PULSE = Z_PULSE (variable that indicates the state of input Z).

LIMIT_SWITCH BOOL (limit switch)

Boolean expression that indicates home position has been reached. It is used in Type 1 & Type 2.

TYPE 1 -> LIMIT_SWITCH = LIMIT_SWITCH_MIN

TYPE 2 -> LIMIT_SWITCH = LIMIT_SWITCH_MAX

GO_TO_POS_REF BOOL (go to position reference)

Boolean expression that enables position control block.

POS_REF REAL (position_reference)

Desired position to locate the tracker.

JOG_FWD BOOL (jog forward)

In order to move manually the system forward, input JOG_FWD has to be set to TRUE.

JOG_BWD BOOL (jog backward)

In order to move manually the system backward, input JOG_BWD has to be set to TRUE.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

HOMING_DONE
BOOL
(homing done)

Boolean expression that enables synchronization process and giving homing done signal to subsequent blocks.

Function call in IL

CAL MODE_HOMING (		
	EN	:= ModeHoming_EN,
	TYPE_MODE	:= Mode-Homing_TYPE_MODE,
	Z_PULSE	:= ModeHoming_Z_PULSE,
	LIMIT_SWITCH	:= Mode-Homing_LIMIT_SWITCH,
	GO_TO_POS_REF	:= Mode-Homing_GO_TO_POS_REF,
	POS_REF	:= ModeHoming_POS_REF,
	JOG_FWD	:= ModeHoming_JOG_FWD,
	JOG_BWD	:= ModeHoming_JOG_BWD)
	LD ST	ModeHoming.DONE Mode-Homing_DONE
	LD ST	ModeHoming.ERR ModeHoming_ERR
	LD ST	ModeHoming.ERNO ModeHoming_ERNO
	LD ST	ModeHoming.ERNO ModeHoming_ERNO

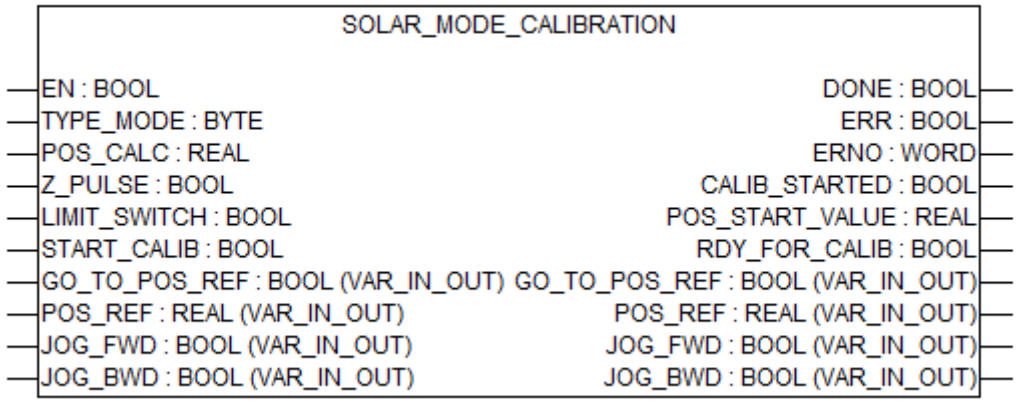
In IL, the function call has to be written in one line.

Function call in ST

MODE_HOMING (		
	EN	:= ModeHoming_EN,
	TYPE_MODE	:= Mode-Homing_TYPE_MODE,
	Z_PULSE	:= ModeHoming_Z_PULSE,
	LIMIT_SWITCH	:= Mode-Homing_LIMIT_SWITCH,
	GO_TO_POS_REF	:= Mode-Homing_GO_TO_POS_REF,
	POS_REF	:= ModeHoming_POS_REF,
	JOG_FWD	:= ModeHoming_JOG_FWD,

	JOG_BWD	:= ModeHoming_JOG_BWD)
	ModeHoming_DONE	:= ModeHoming.DONE;
	ModeHoming_ERR	:= ModeHoming.ERR;
	ModeHoming_ERNO	:= ModeHoming.ERNO;
	Mode-Homing_HOMING_DONE	:= Mode-Homing.HOMING_DONE;

SOLAR_MODE_CALIBRATION



Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks for different control modes

This function block represents an operation mode that allows to calibrate the tracker reference point.

This function is necessary to calibrate the reference point (Z) when incremental encoders are used for the first time.

Sequence of the calibration process.

Option 1:

1. Move the tracker by using SOLAR_MODE_MANUAL to a position that let you to measure that position easily.
2. Get the angular position by using a high accuracy measurement device. Store this value in POS_CALC variable.
3. Enabling function block to start calibrating process, then:
   - System will be moved automatically towards reference point.
   - System will stop as soon as reference point (Z) is reached. Encoders position function will stored the calibrating value at Z_VALUE.

   ⇨ Calibrating process finished.

Option 2:

On the other hand, if you are not able to determinate tracker position by using a high accuracy measurement device, it is possible to use a simple solar radiation sensor to determinate the correct tracker position.

Proceed as follow:

1. Move the tracker using SOLAR_MODE _MANUAL to a position ahead of the sun.
2. Using a solar radiation sensor, you have to wait until tracker will be focused. At this moment you store the calculated position of the sun (obtained by using a sun position algorithm).
3. Enabling function block to start calibrating process, then:
   - System will be moved automatically towards reference point.
   - System will stop as soon as reference point (Z) is reached. Encoders position function will stored the calibrating value at Z_VALUE.

   ⇨ Calibration process finished.

Option 2 needs a sunny day to complete the process.

Figure below shows the logical sequence that implements both options:

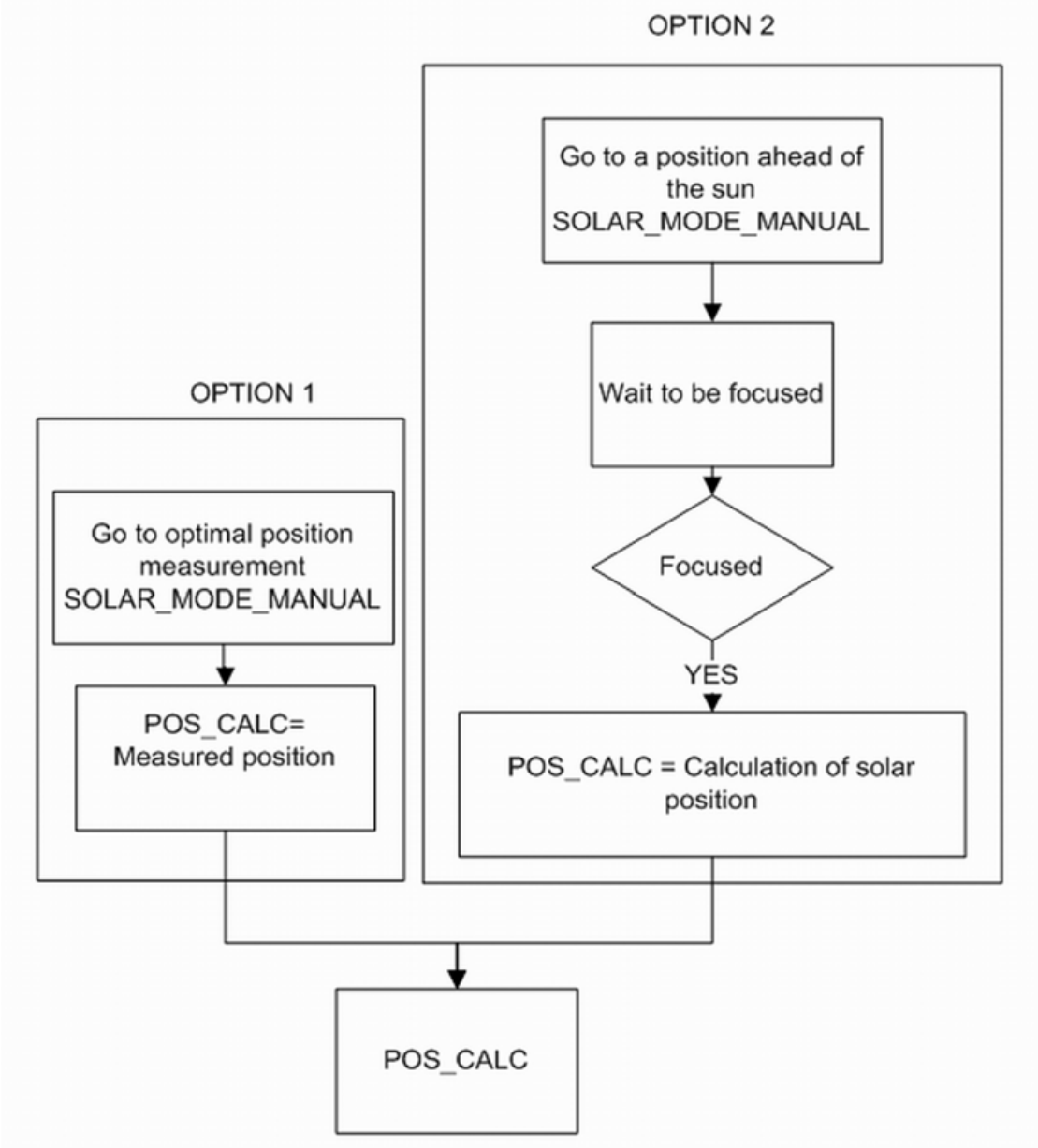
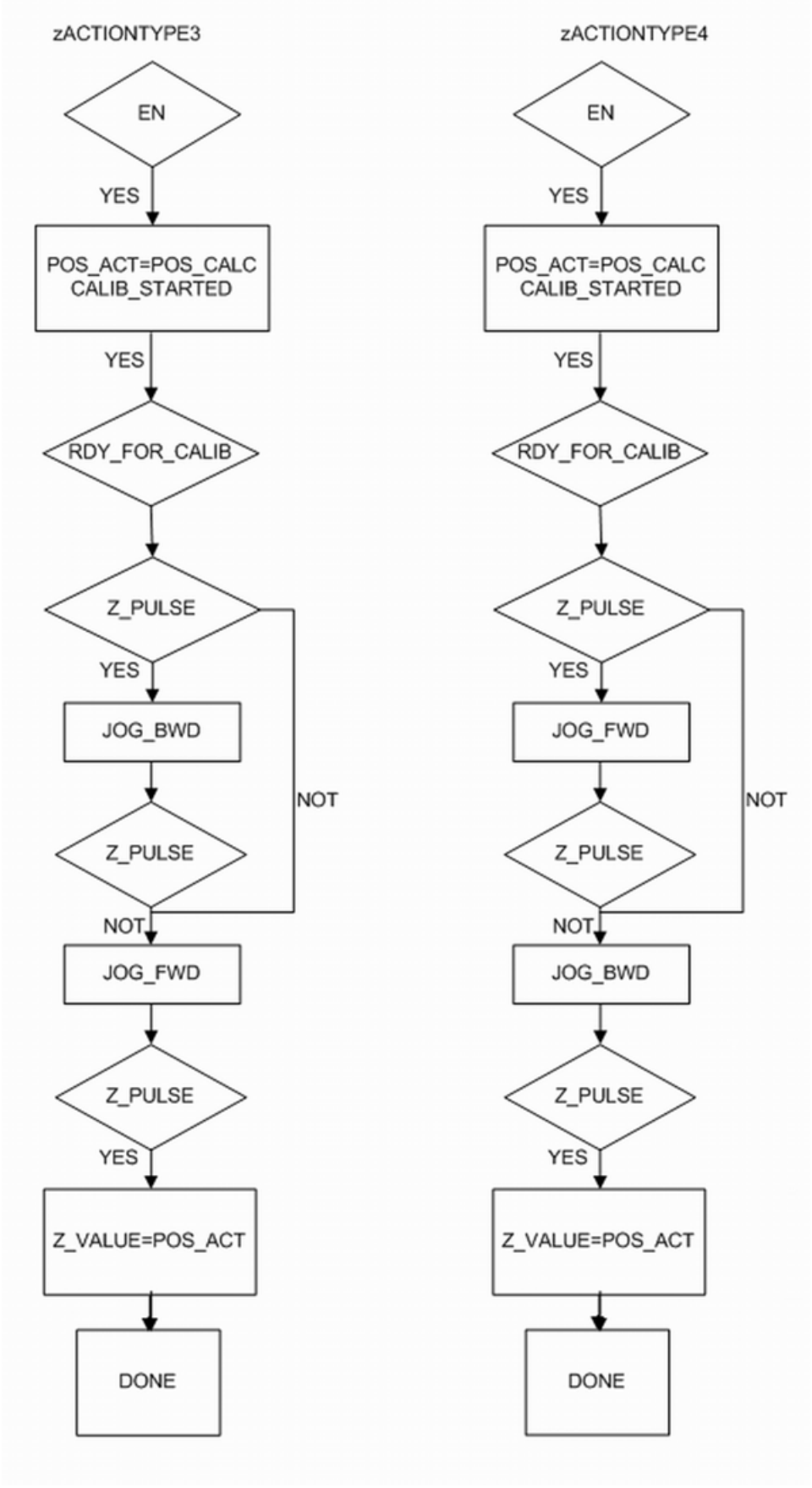


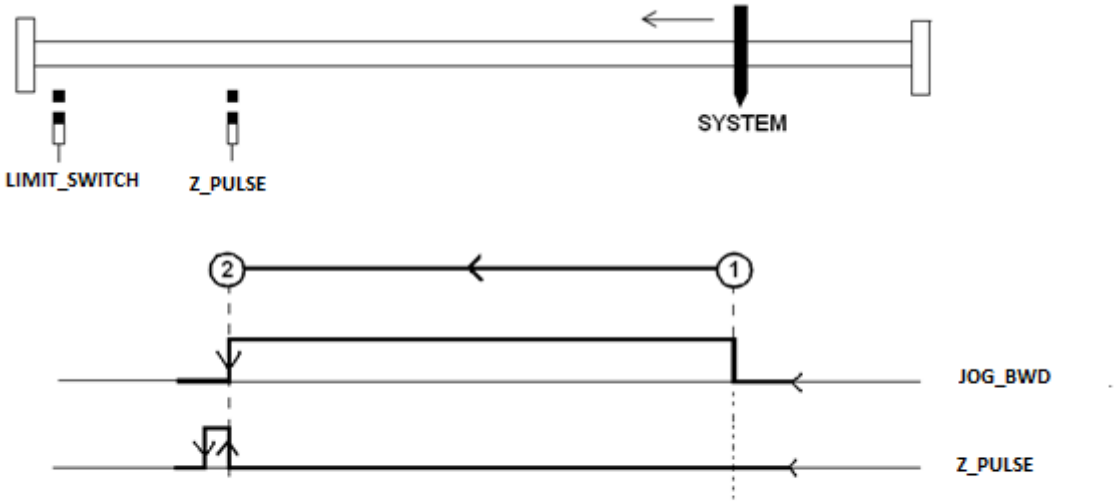
Figure below shows the logical sequence that implements SOLAR_MODE_CALIBRATION for Type 1 & Type 2:

Figure below shows the logical sequence that implements SOLAR_MODE_CALIBRATION for Type 3 & Type 4:



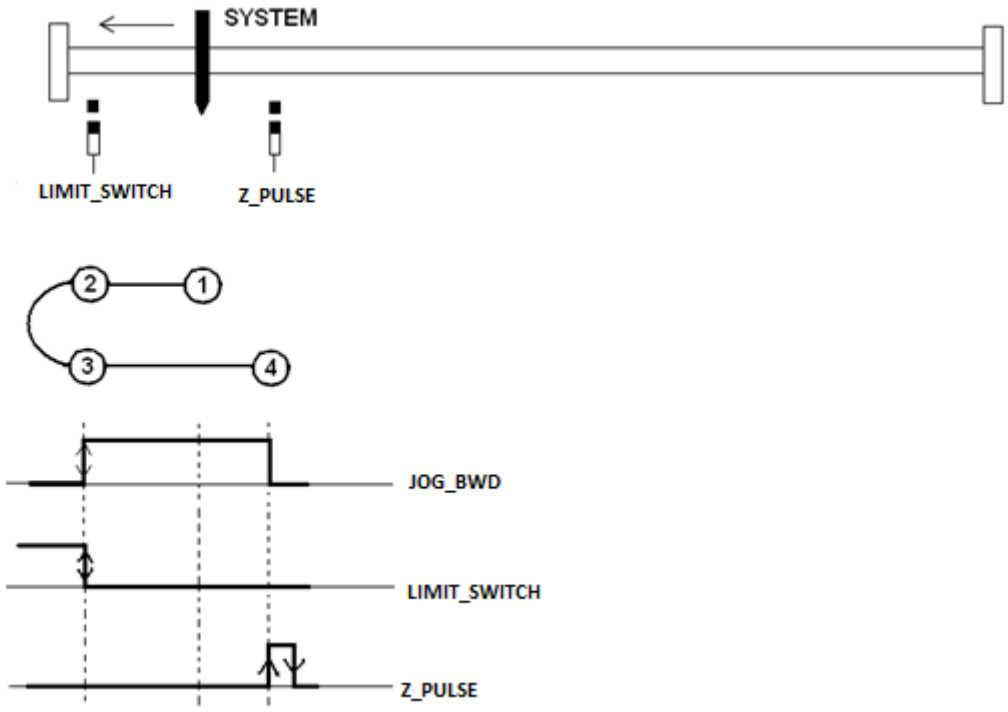
Type 1: Using encoder reference point Z_PULSE to determinate home position.

First movement: Negative direction (left):



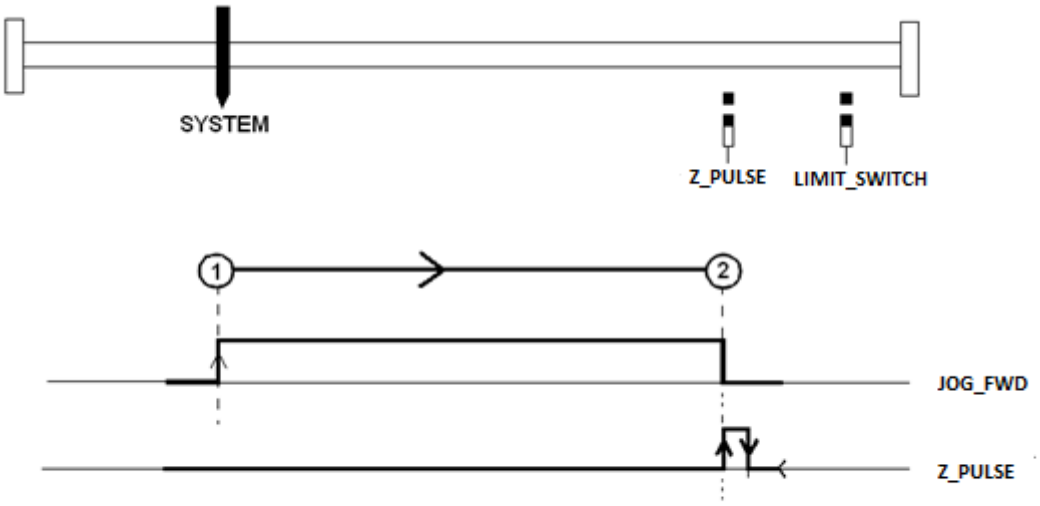
At the beginning, system is located at point 1 and limit switch is not active yet. When zACTION-TYPE1 is enabled, JOG_BWD is set to TRUE. As soon as shows rising edge, home position has been reached and system stops its movement at point 2.

If system is located between Z_PULSE and LIMIT_SWITCH positions, SOLAR_MODE_CALI-BRATION operates as figure below shows:



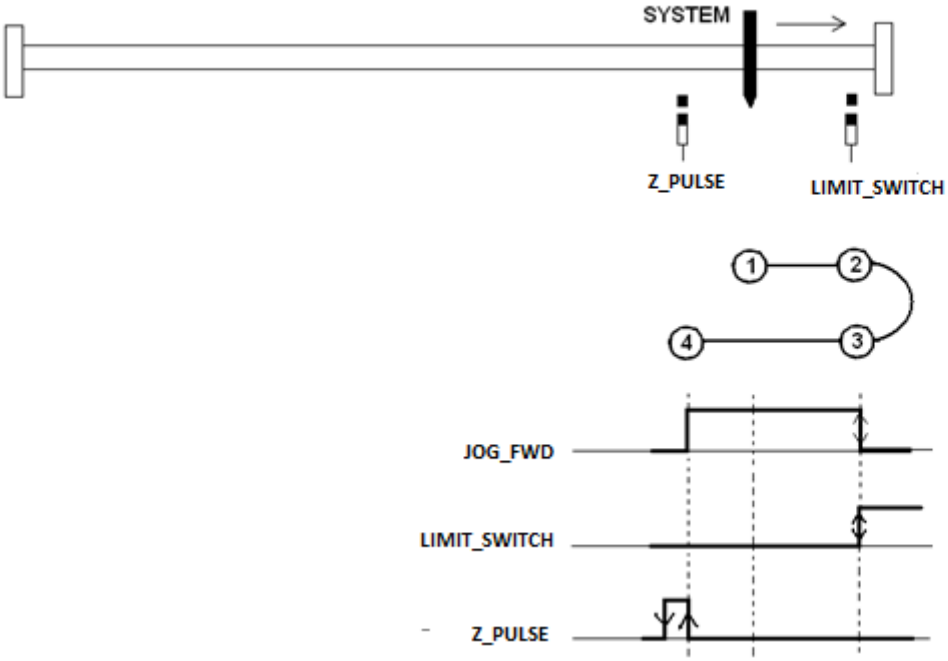
Type 2: Using encoder reference point Z_PULSE to determinate home position.

First movement: Positive direction (right):



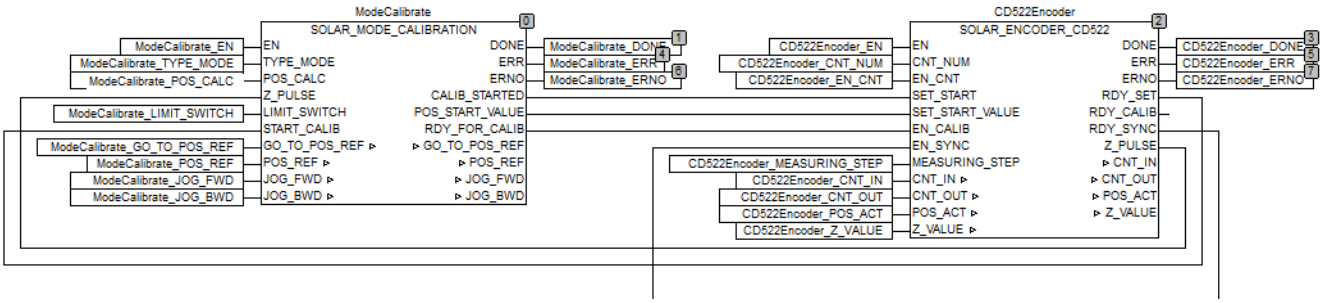
At the beginning, system is located at point 1 and Z_PULSE is not active yet. When SOLAR_MODE_HOMING is enabled, JOG_FWD is set to TRUE. As soon asZ_PULSE shows rising edge, home position has been reached and system stops its movement at point 2.

If system is located between Z_PULSE and LIMIT_SWITCH positions, SOLAR_MODE_CALIBRATION operates as figure below shows:



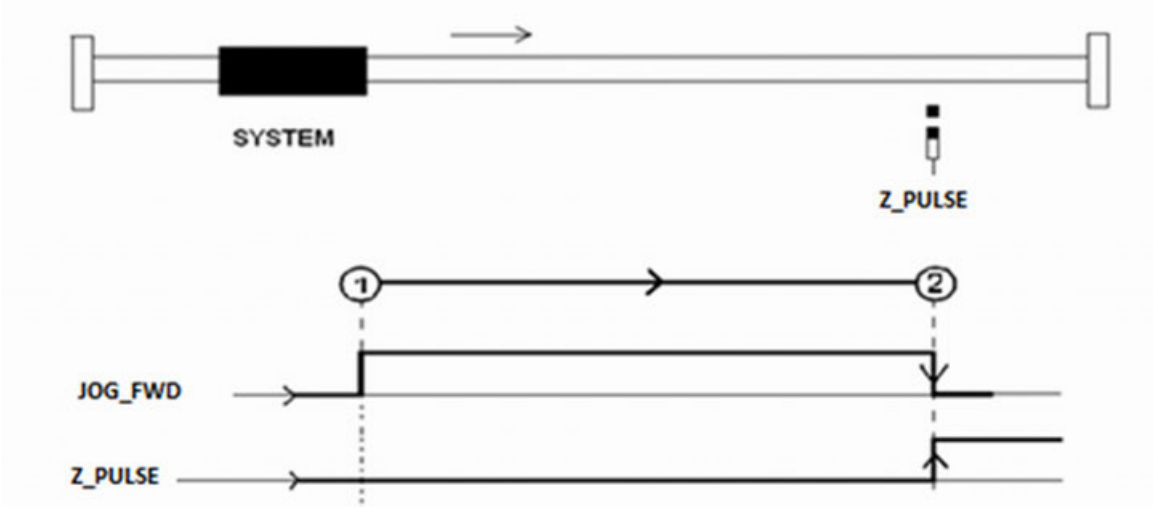
Example: SOLAR_MODE_CALIBRATION + Encoder

ModeCalibrate.Z_PULSE = CD522Encoder.RDY_TO_CALIB

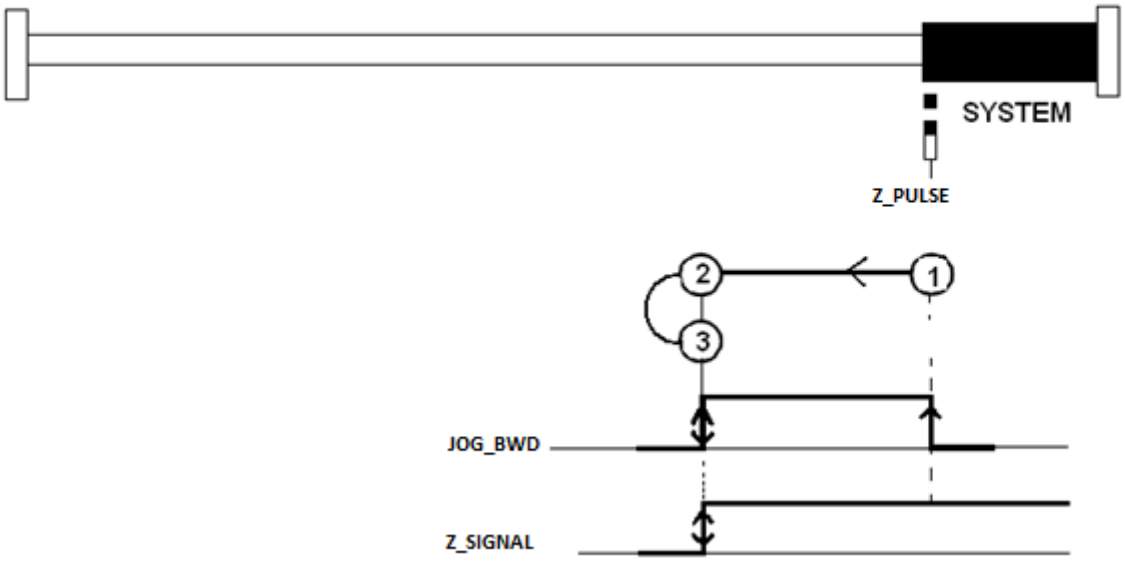


Type 3: Using Z_PULSE and cam switch to determinate home position.

First movement: Positive direction (right):

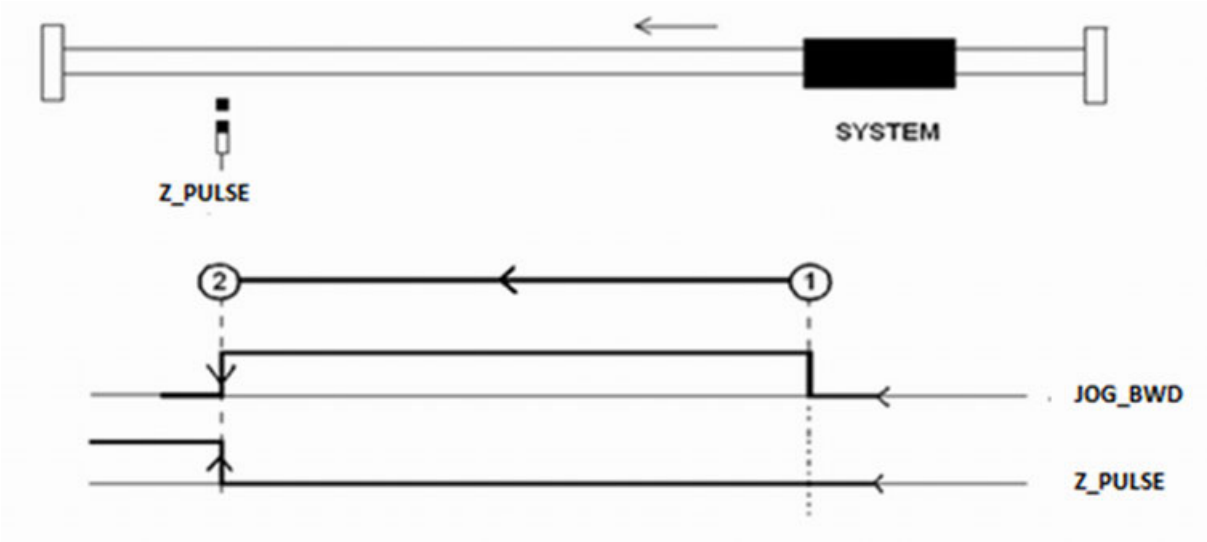


Home position has been reached when rising edge of the signal Z_PULSE is detected. Then, movement is stopped. On the other hand, if system is located at the end of its movement range and Z_PULSE signal is set to TRUE (see figure below) home position is reached as follow:

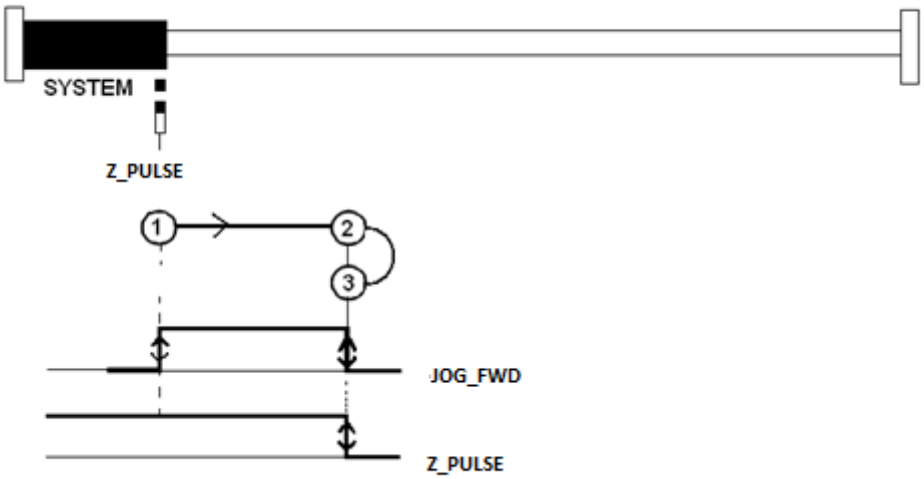


Type 4: Using Z_PULSE and cam switch to determinate home position.

First movement: Negative direction (left).

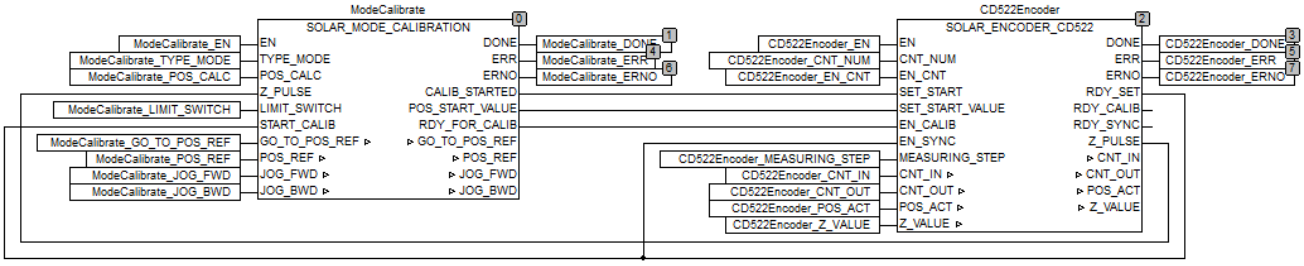


Home position has been reached when rising edge of the signal Z-PULSE is detected. Then, movement is stopped. On the other hand, if system is located at the end of its movement range and Z_PULSE signal is set to TRUE (see figure below) home position is reached as follow:



Example: SOLAR_MODE_CALIBRATION + Encoder.

> *ModeCalibrate.Z_PULSE = CD522Encoder.Z_PULSE*



Input description

EN BOOL (enable block)

Calibrating process starts as soon as rising edge is detected at signal EN.

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

While input EN is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

TYPE_MODE BYTE (type mode)

This input indicates which kind of configuration sensors are going to be used to determinate home positions (see figures above).

1 -> Z_Sensor is only one pulse signal. First movement: backward.

2 -> Z_Sensor is only one pulse signal. First movement: forward.

3 -> Z_Sensor uses cam switch. First movement: forward.

4 -> Z_Sensor uses cam switch. First movement: backward.

POS_CALC REAL (position calculated)

Calculated tracker position using a high accuracy measurement device or a sun position algorithm.

Z_PULSE BOOL (z pulse)

Boolean expression that indicates the positioning device has reached its Z position.

TYPE 1 & 2 -> Z_PULSE = RDY_CALIB (output variable of encoders function).

TYPE 3 & 4 -> Z_PULSE = (variable that indicates the state of input Z).

LIMIT_SWITCH BOOL (limit switch)

Boolean expression that indicates home position has been reached. It is used in Type 1 & Type 2.

TYPE 1 -> LIMIT_SWITCH = LIMIT_SWITCH_MIN

TYPE 2 -> LIMIT_SWITCH = LIMIT_SWITCH_MAX

START_CALIB BOOL (start calibration)

Boolean expression that indicates reading process of the encoder has finished (see SOLAR_Encoder_CD522_RDY_SET ↳ *Chapter 1.5.11.2.4.2 “SOLAR_ENCODER_CD522” on page 3194*).

POS_ACT = POS_CALC.

GO_TO_POS_REF BOOL (go to position reference)

Boolean expression that enables position control block.

POS_REF REAL (position_reference)

Desired position to locate the tracker.

JOG_FWD BOOL (jog forward)

In order to manually move the system forward, input JOG_FWD has to be set to TRUE.

JOG_BWD BOOL (jog backward)

In order to manually move the system backward, input JOG_BWD has to be set to TRUE.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

CALIB_STARTED BOOL (Calibration_Started)

Output CALIB_STARTED indicates calibrating process has started.

POS_START_VALUE REAL (position start value)

The counter can be set to a start value. This value must be applied to the input POS_START_VALUE.

If input CALIB_STARTED = TRUE, the counter takes this value.

RDY_FOR_CALIB BOOL (ready for calibration)

Boolean expression that enables calibrating process.

Function call in IL

CAL ModeCalibrate (		
	EN	:= ModeCalibrate_EN,
	TYPE_MODE	:= ModeCalibrate_TYPE_MODE,
	POS_CALC	:= ModeCalibrate_POS_CALC,
	Z_PULSE	:= ModeCalibrate_Z_PULSE,
	LIMIT_SWITCH	:= ModeCalibrate_LIMIT_SWITCH,
	RDY_TO_CALIB	:= ModeCalibrate_RDY_TO_CALIB,
	GO_TO_POS_REF	:= ModeCalibrate_GO_TO_POS_REF,
	POS_REF	:= ModeCalibrate_POS_REF,
	JOG_FWD	:= ModeCalibrate_JOG_FWD,
	JOG_BWD	:= ModeCalibrate_JOG_BWD)

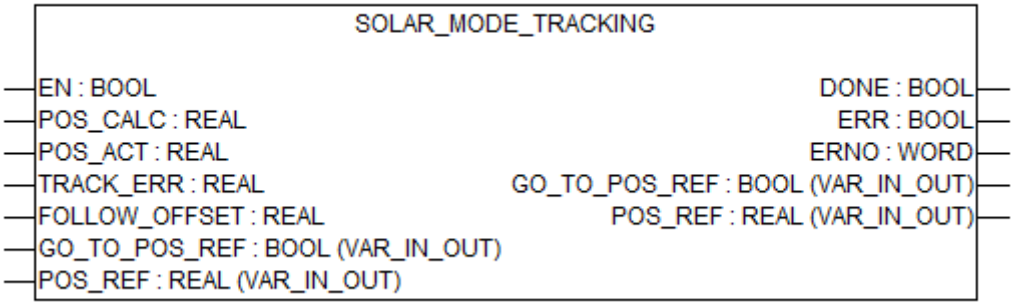
	LD	ModeCalibrate.DONE
	ST	ModeCalibrate_DONE
	LD	ModeCalibrate.ERR
	ST	ModeCalibrate_ERR
	LD	ModeCalibrate.ERNO
	ST	ModeCalibrate_ERNO
	LD	ModeCalibrate.CALIB_STARTED
	ST	ModeCalibrate_CALIB_STARTED
	LD	ModeCalibrate.POS_START_VALUE
	ST	ModeCalibrate_ERNO_VALUE
	LD	ModeCalibrate.ERNO_RDY_FOR_CALIB
	ST	ModeCalibrate_ERNO_EN_RDY_FOR_CALIB

In IL, the function call has to be written in one line.

Function call in ST

ModeCalibrate (		
	EN	:= ModeCalibrate_EN,
	TYPE_MODE	:= ModeCalibrate_TYPE_MODE,
	POS_CALC	:= ModeCalibrate_POS_CALC,
	Z_PULSE	:= ModeCalibrate_Z_PULSE,
	LIMIT_SWITCH	:= ModeCalibrate_LIMIT_SWITCH,
	RDY_TO_CALIB	:= ModeCalibrate_RDY_TO_CALIB,
	GO_TO_POS_REF	:= ModeCalibrate_GO_TO_POS_REF,
	POS_REF	:= ModeCalibrate_POS_REF,
	JOG_FWD	:= ModeCalibrate_JOG_FWD,
	JOG_BWD	:= ModeCalibrate_JOG_BWD)
	ModeCalibrate_DONE	:= ModeCalibrate.DONE;
	ModeCalibrate_ERR	:= ModeCalibrate.ERR;
	ModeCalibrate_ERNO	:= ModeCalibrate.ERNO;
	ModeCali- brate_CALIB_STARTED	:= ModeCalibrate.CALIB_STARTED;
	ModeCali- brate_POS_START_VALUE	:= ModeCalibrate.POS_START_VALUE;
	ModeCalibrateRDY_FOR_CALIB	:= ModeCalibrate.RDY_FOR_CALIB;

SOLAR_MODE_TRACKING



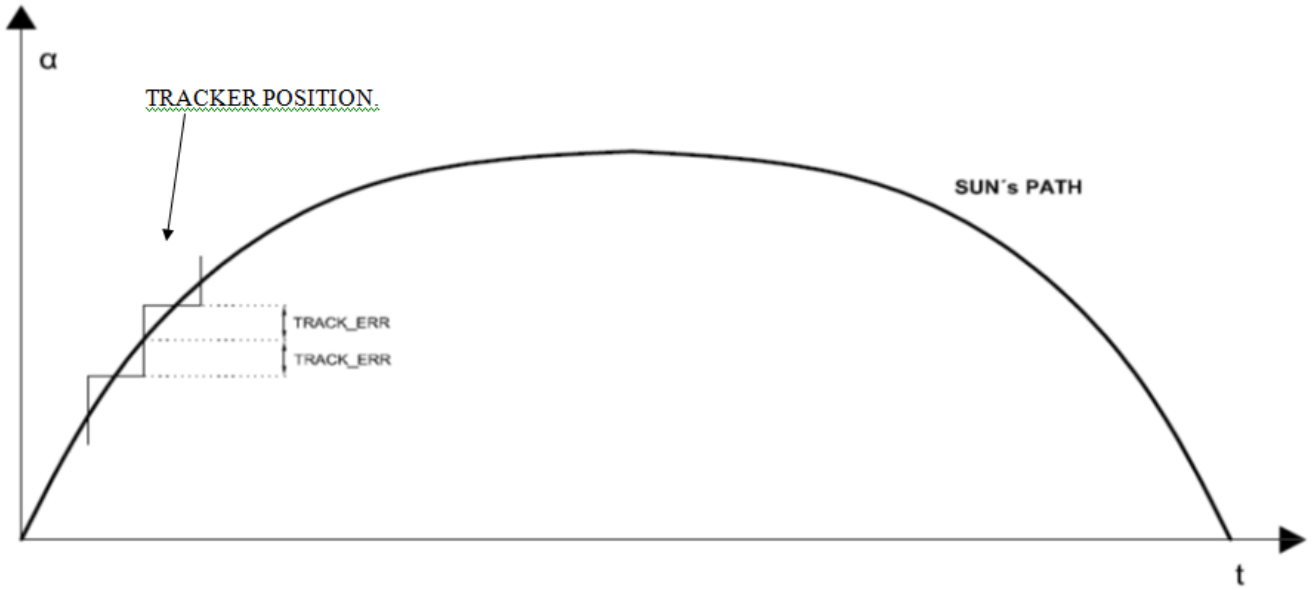
Available as of runtime system:	V1.3 and above
Included in library:	Solar_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks for different control modes

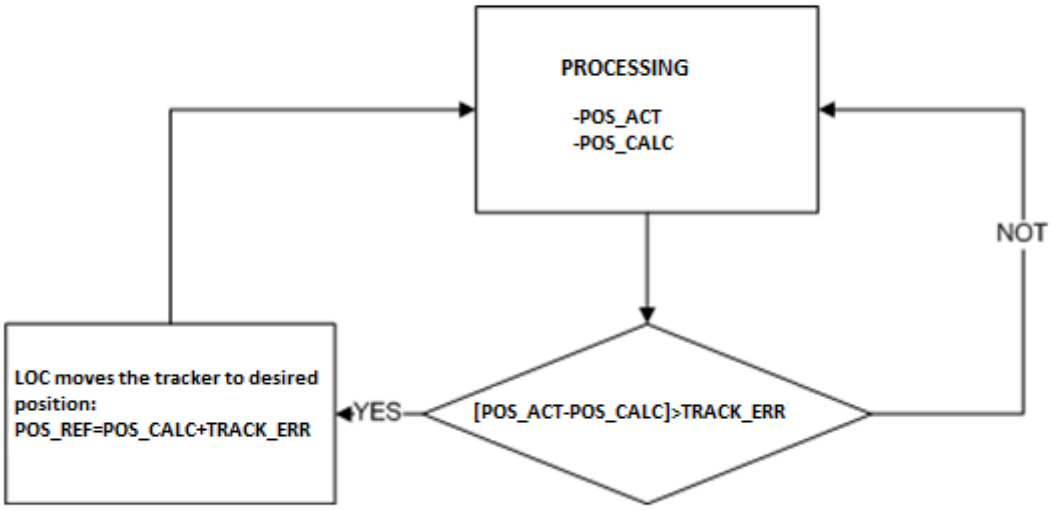
This function block represents an operation mode that allows to track the sun based on a calculated position of the sun. This position is given by an SOLAR_NOAA algorithm depending on date, latitude and longitude.

This function block calculates the absolute difference between the calculated position of the sun and the actual tracker position. If the result is greater than the value of track error (TRACK_ERR), system moves to the calculated position. Otherwise, system is stopped.

The figure below shows how this operation mode works.

Figure below shows the logical sequence that implements SOLAR_MODE_TRACKING:





It is possible to follow sun´s path with an offset. This value is indicated by the parameter FOLLOW_OFFSET.

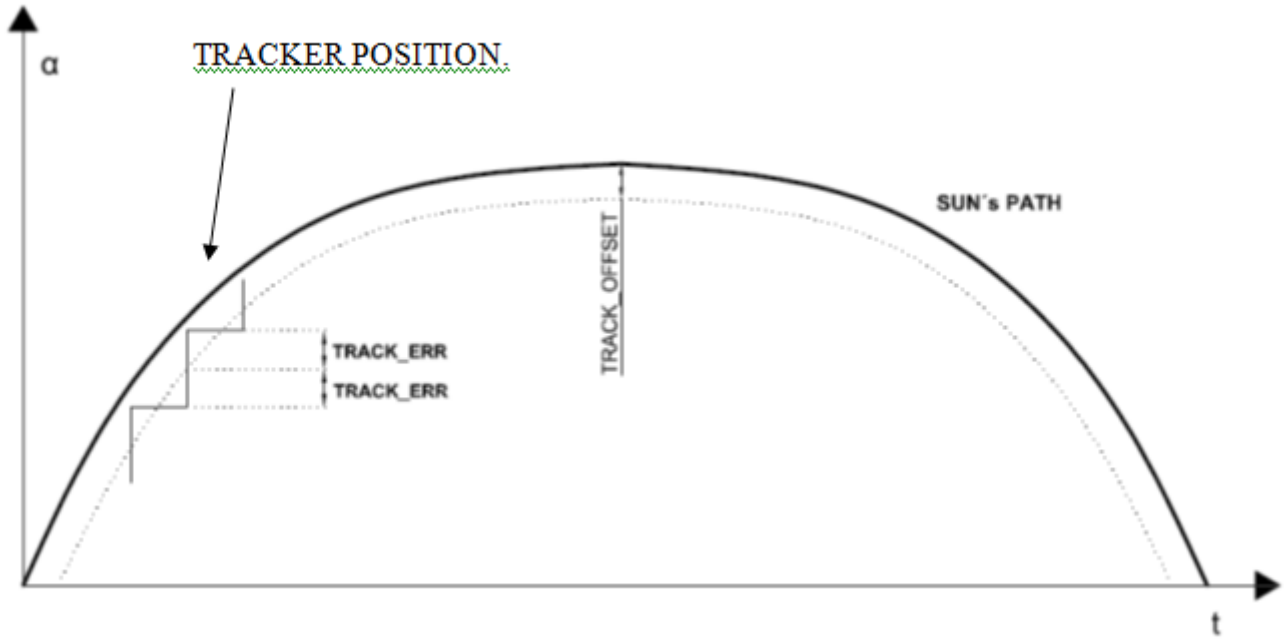
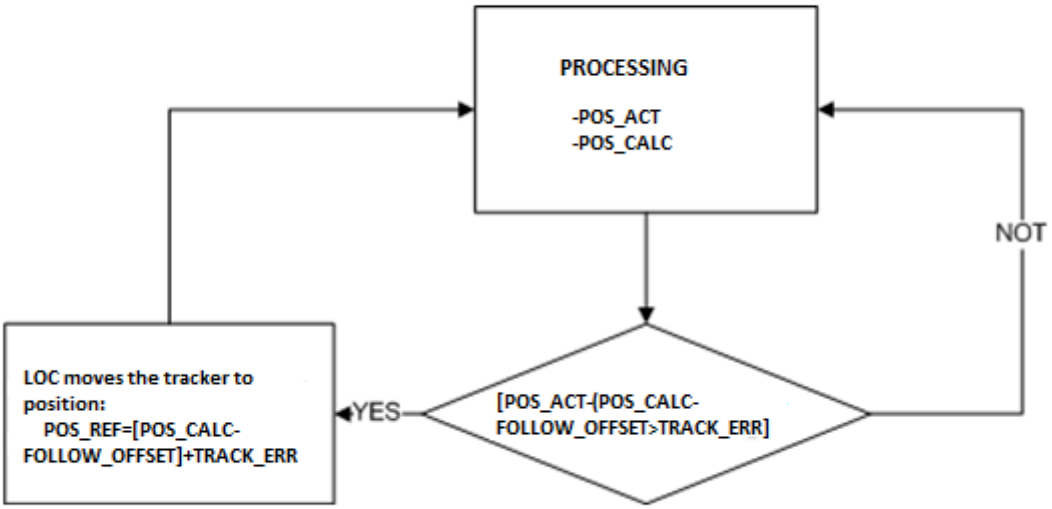
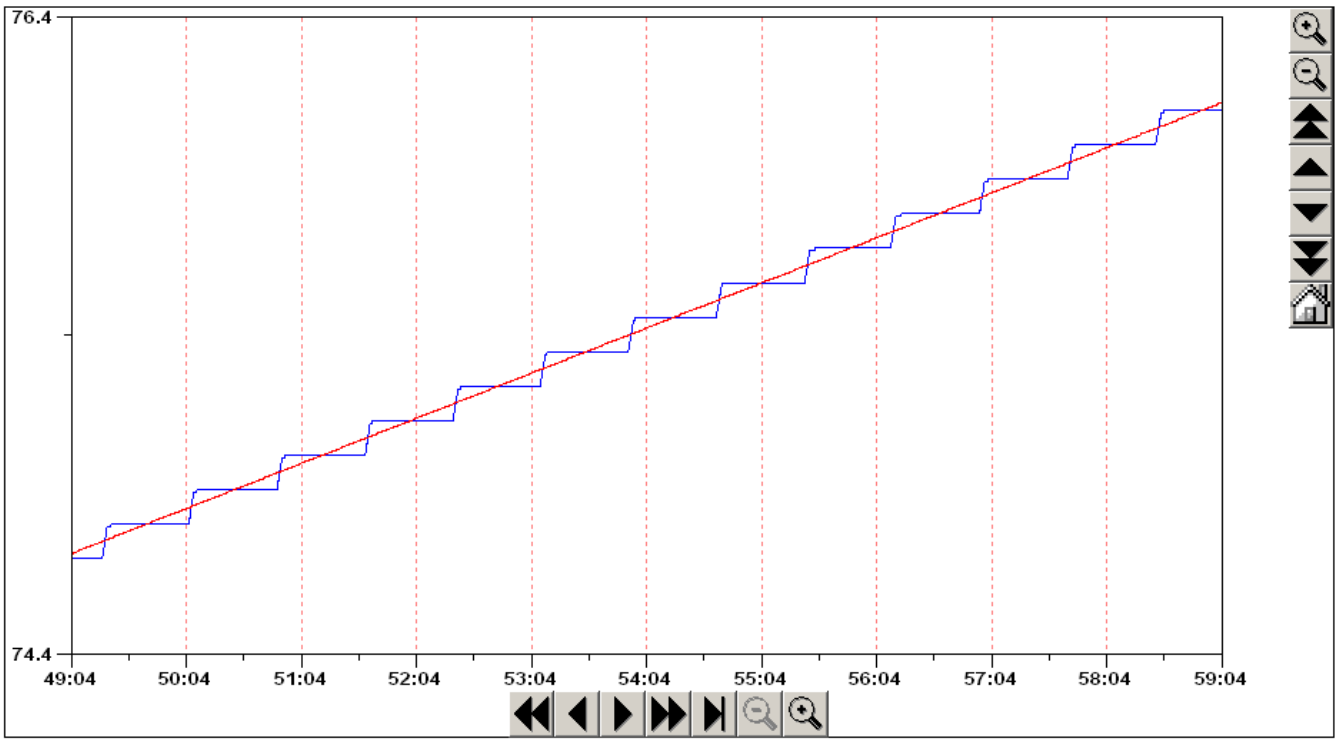


Figure below shows the logical sequence that implements SOLAR_MODE_TRACKING adding an offset value.



Tracker follows the sun by using a discontinuous movement. It is a step by step movement in order to save energy from actuators.

Chart below shows real behavior of azimuth tracker when is enabled SOLAR_MODE_TRACKING. In this case TRACK_ERROR = 0,05°.



Red line -> Calculated position using SOLAR_NOAA algorithm.

Blue line -> Actual axis position.

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

POS_CALC REAL (position_calculated)

Calculated position of the sun.

POS_ACT REAL (position actual)

Actual position of Solar Tracker.

TRACK_ERR REAL (track error)

Value of track error allowed.

FOLLOW_OFFSET REAL (follow offset)

Indicates that tracking has an offset concerning to real calculated position of the sun.

GO_TO_POS_REF BOOL (go to position reference)

Boolean expression that enables position control block.

POS_REF REAL (position_reference)

Desired position to locate the tracker.

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Function call in IL

CAL ModeTracking (		
	EN	:= ModeTracking_EN,
	POS_CALC	:= Mode-Tracking_POS_CALC,
	POS_ACT	:= ModeTracking_POS_ACT,
	TRACK_ERR	:= Mode-Tracking_TRACK_ERR,
	FOLLOW_OFFSET	:= Mode-Tracking_FOLLOW_OFFSET,
	GO_TO_POS_REF	:= Mode-Tracking_GO_TO_POS_REF,
	POS_REF	:= ModeTracking_POS_REF)
	LD	ModeTracking.DONE
	ST	ModeTracking_DONE
	LD	ModeTracking.ERR
	ST	ModeTracking_ERR
	LD	ModeTracking.ERNO
	ST	ModeTracking_ERNO

In IL, the function call has to be written in one line.

Function call in ST

ModeTracking (		
	EN	:= ModeTracking_EN,
	POS_CALC	:= Mode-Tracking_POS_CALC,
	POS_ACT	:= ModeTracking_POS_ACT,
	TRACK_ERR	:= Mode-Tracking_TRACK_ERR,
	FOLLOW_OFFSET	:= Mode-Tracking_FOLLOW_OFFSET,
	GO_TO_POS_REF	:= Mode-Tracking_GO_TO_POS_REF,
	POS_REF	:= ModeTracking_POS_REF)
	ModeTracking_DONE	:= ModeTracking.DONE;
	ModeTracking_ERR	:= ModeTracking.ERR;
	ModeTracking_ERNO	:= ModeTracking.ERNO;

1.5.11.3 Solar_NREL library

1.5.11.3.1 Preconditions for the use of the Solar_NREL library

The blocks contained in the Solar_NREL library can only be executed in RUN mode of the PLC, but not in simulation mode. NREL stands for National Renewable Energy Laboratory and SPA stands for Solar Position Algorithm.

The function blocks of the library SolarNREL_AC500_V22.lib Version 0.4 are available in AC500 control systems with a runtime system of version V1.3 or higher and S500 I/O devices (DC551) with firmware version V1.11 or higher.

Because of the complexity of the algorithm, this library is recommended to be used with a PM583 at least.

NOTICE!

Notice about the algorithm

This computer software is code in development prepared by Midwest Research Institute, (hereinafter the "Contractor"), under Contract DE-AC-99G010337 (Contract) with the Department of Energy (DOE). The United States Government has been granted for itself and others acting on its behalf a paid-up, non-exclusive, irrevocable, worldwide license in the Software to reproduce, prepare derivative works, and perform publicly and display publicly. Beginning five (5) years after the date permission to assert copyright is obtained from the DOE, and subject to any subsequent five (5) year renewals, the United States Government is granted for itself and others acting on its behalf a paid-up, non-exclusive, irrevocable, worldwide license in the Software to reproduce, prepare derivative works, distribute copies to the public, perform publicly and display publicly, and to permit others to do so. If the Contractor ceases to make this computer software available, it may be obtained from DOE's Office of Scientific and Technical Information's Energy Science and Technology Software Center (ESTSC) at P.O. Box 1020, Oak Ridge, TN 37831-1020. THIS SOFTWARE IS PROVIDED BY THE CONTRACTOR "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CONTRACTOR OR THE U.S. GOVERNMENT BE LIABLE FOR ANY SPECIAL, INDIRECT OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO CLAIMS ASSOCIATED WITH THE LOSS OF DATA OR PROFITS, WHICH MAY RESULT FROM AN ACTION IN CONTRACT, NEGLIGENCE OR OTHER TORTIOUS CLAIM THAT ARISES OUT OF OR IN CONNECTION WITH THE ACCESS, USE OR PERFORMANCE OF THIS SOFTWARE.

The Application Version of the library Solar_AC500_V22.lib is tested with the equipment and configuration used in the attached example programm "Example_Solar_2Axis_ACS3XX.pro".

This example uses a PM583 with a CD522 counter module.

ABB ACS355 drive is used with AC500 PLC via Modbus RTU communication.

This example also has option to connect DC541 or DC522 as encoder module.

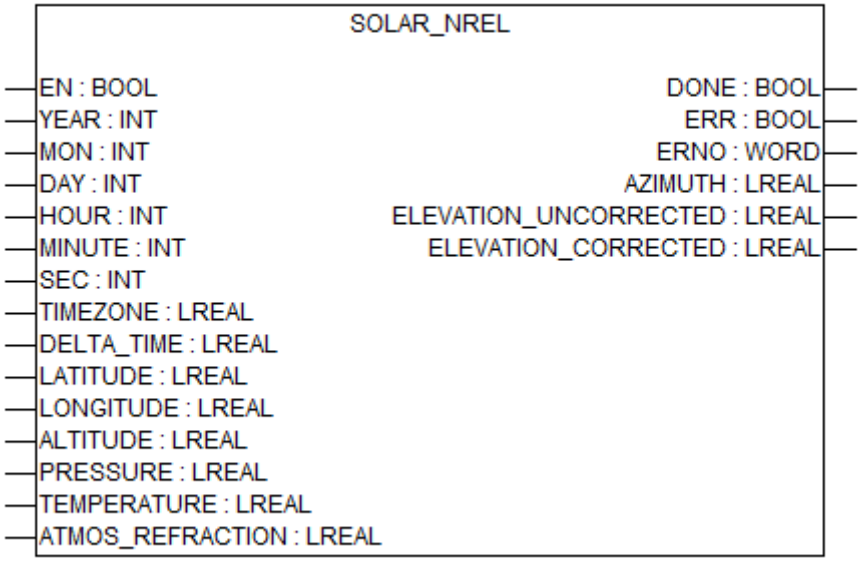
Any other combination with AC500 or drives equipment should work, but is not tested and might however not work properly.

1.5.11.3.2 Special characteristics of the Solar_NREL library

The Solar_NREL library contains all blocks necessary to calculate the position of the sun (azimuth and elevation) depending on the date, time and local environment.

A detailed description of the AC500 PLC configuration, can be found in the chapter *Chapter 1.6.5 “Configuration in Automation Builder for AC500 V2 products” on page 5757*.

1.5.11.3.3 SOLAR_NREL



Available as of runtime system:	V1.3 and above
Included in library:	SolarNREL_AC500_V22.lib
Type	Function block without historical values
Group	Package of function blocks to get the position of the sun

This function block calculates the position of the sun depending on the date, time, and localization. It also includes a function that corrects the value of the elevation thanks to the local pressure, temperature and refraction of atmosphere at sunrise and sunset. This function embeds the NREL's algorithm which can be found on the NERLs website: *http://rredc.nrel.gov/solar/codesandalgorithms/spa*.

The limitation of this function is the number of outputs which are the azimuth, elevation (uncorrected) and corrected elevation. These outputs are the topocentric coordinates of the sun which use the observer's location as the center of the coordinate system.

The end-user may use the RTC of the AC500 system by using the CLOCK function block of the library named SysExt_AC500_V10.lib or may use any other source to calculate the position of the sun.

Any additional information about the algorithm can be found on the NREL website: *http://www.nrel.gov/*

Table 197: Size of used data for NREL function block and execution time for each CPU type

	PM583	**PM591**	**PM564**
Data Size	4731 bytes	4731 bytes	4731 bytes
Program Size	79706 bytes	43270 bytes	80406 bytes
Execution Time	65 ms	1 ms	120 ms

In case of a PM58x and PM5X4 CPU´s, the program which calls this function block must be integrated in a dedicated task with a correct value of watchdog (or with watchdog disabled).

Input description

EN BOOL (enable block) In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE.

When calling the block the first time, the inputs are checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

YEAR INT (year) Range of values: from 2000 to 6000.

MON INT (month) Range of values: from 1 to 12.

DAY INT (day) Range of values: from 1 to 31.

HOUR INT (hour) Range of values: from 0 to 24.

MINUTE INT (minute) Range of values: from 0 to 59.

SEC INT (second) Range of values: from 0 to 59.

TIMEZONE LREAL (time-zone) Observer time zone (negative west of Greenwich) in hours and parts of hours.

Range of values: from -12 to 12.

DELTA_TIME LREAL (ΔT) ΔT = Terrestrial Time (TT) - Universal Time (UT) [seconds].

Difference between the earth rotation time and the terrestrial time. It is derived from observation only and it is reported in this bulletin, where _T = 32.184 + (TAI-UTC) + _UT1.

LATITUDE LREAL (lati-tude) Observer latitude (negative south of equator) in degrees.

Range of values: from -90 to 90.

LONGITUDE LREAL (longi-tude) Observer longitude (negative west of Greenwich) in degrees.

Range of values: from -180 to 180.

ALTITUDE LREAL (alti-tude) Observer elevation.

Range of values: -5000 to 5000

PRESSURE LREAL (pressure)
Local pressure in millibars.
Range of values: from 0 to 5000.

TEMPERATURE LREAL (temperature)
Local temperature in degrees Celsius.
Range of values: from -273 to 6000.

ATMOS_REFRACTION LREAL (atmosphere refraction)
Atmospheric refraction at sunrise and sunset in degrees.
Range of values: from -5 to 5. Typical: 0.5667

Output description

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

AZIMUTH LREAL (azimuth)
Topocentric azimuth angle (eastward from north) in degrees.
Range of values: from 0 to 360.
Error : ± 0,0004°.

ELEVATION_UNCORRECTED LREAL (elevation uncorrected)
Topocentric elevation angle (uncorrected) in degrees.
Range of values: from -90° to 90°.
Error : ± 0,0004°.

ELEVATION_CORRECTED LREAL (elevation corrected)
Topocentric elevation angle (corrected) in degrees. This value includes the effects of: refraction, pressure and temperature.
Range of values: from -90° to 90°.
Error : ± 0,0004°.

Function call in IL

CAL mySOLAR_NREL (		
	EN	:= SOLAR_NREL_EN,
	YEAR	:= SOLAR__NREL_OBSERVER_YEAR_CURRENT,
	MON	:= SOLAR__NREL_OBSERVER_MON_CURRENT,
	DAY	:= SOLAR__NREL_OBSERVER_DAY_CURRENT,
	HOUR	:= SOLAR__NREL_OBSERVER_HOUR_CURRENT,
	MINUTE	:= SOLAR__NREL_OBSERVER_MINUTE_CURRENT,
	SEC	:= SOLAR__NREL_OBSERVER_SEC_CURRENT,
	TIMEZONE	:= SOLAR__NREL_OBSERVER_TIMEZONE_CURRENT,
	DELTA_TIME	:= SOLAR__NREL_DELTA_TIME,
	LONGITUDE	:= SOLAR__NREL_OBSERVER_LONGITUDE,
	LATITUDE	:= SOLAR__NREL_OBSERVER_LATITUDE,
	ALTITUDE	:= SOLAR__NREL_OBSERVER_ALTITUDE,
	PRESSURE	:= SOLAR_NREL_LOCAL_PRESSURE,
	TEMPERATURE	:= SOLAR_NREL_LOCAL_TEMPERATURE,
	ATMOS_REFRACTION	:= SOLAR_NREL_LOCAL_ATMOSPHERE_REFRACTION_AT_SUNRISE_AND_SUNSET)
LD ST	mySOLAR_NREL.DONE SOLAR_NREL_DONE	
LD ST	mySOLAR_NREL.ERR SOLAR_NREL_ERR	

LD	mySOLAR_NREL.ERNO
ST	SOLAR_NREL_ERNO
LD	mySOLAR_NREL.AZIMUTH
ST	SOLAR_NREL_AZIMUTH
LD	mySOLAR_NREL.ELEVATION_UNCORRECTED
ST	SOLAR_NREL_ELEVATION_UNCORRECTED
LD	mySOLAR_NREL.ELEVATION_CORRECTED
ST	SOLAR_NREL_ELEVATION_CORRECTED

In IL, the function call has to be written in one line.

Function call in ST

mySOLAR_NREL (		
	EN	:= SOLAR_NREL_EN,
	YEAR	:= SOLAR_NREL_OBSERVER_YEAR_CURRENT,
	MON	:= SOLAR_NREL_OBSERVER_MON_CURRENT,
	DAY	:= SOLAR_NREL_OBSERVER_DAY_CURRENT,
	HOUR	:= SOLAR_NREL_OBSERVER_HOUR_CURRENT,
	MINUTE	:= SOLAR_NREL_OBSERVER_MINUTE_CURRENT,
	SEC	:= SOLAR_NREL_OBSERVER_SEC_CURRENT,
	TIMEZONE	:= SOLAR_NREL_OBSERVER_TIMEZONE_CURRENT,
	DELTA_TIME	:= SOLAR_NREL_DELTA_TIME,
	LONGITUDE	:= SOLAR_NREL_OBSERVER_LONGITUDE,
	LATITUDE	:= SOLAR_NREL_OBSERVER_LATITUDE,
	ALTITUDE	:= SOLAR_NREL_OBSERVER_ALTITUDE,

	PRESSURE	:= SOLAR_NREL_LOCAL_PRESSURE,
	TEMPERATURE	:= SOLAR_NREL_LOCAL_TEMPERATURE,
	ATMOS_REFRACTION	:= SOLAR_NREL_LOCAL_ATMOSPHERE_REFRACTION_AT_SUNRISE_AND_SUNSET)
SOLAR_NREL_DONE		:= mySOALR_NREL.DONE
SOLAR_NREL_ERR		:= mySOLAR_NREL.ERR
SOLAR_NREL_ERNO		:= mySOLAR_NREL.ERNO
SOLAR_NREL_AZIMUTH		:= mySOLAR_NREL.AZIMUTH
SOLARSOLAR_NREL_ELEVATION_UNCORRECTED		:= mySOALR_NREL.ELEVATION_UNCORRECTED
SOLAR_NREL_ELEVATION_CORRECTED		:= mySOLAR_NREL.ELEVATION_CORRECTED

1.5.11.4 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.12 Temperature control library

1.5.12.1 System technology

1.5.12.1.1 Introduction

The Temperature Control library package contains several parts and is overall designed for the demand of advanced temperature control for critical processes. It needs precise temperature control and e.g. adaptive tuning for ease of handling and changing environmental or process conditions.

The package consists of the following components:

- Libraries:
  - TECT_TEMP_CONTROL_AC500_V24 library
  - ADCTRL_AC500_V24 library (internal, automatically referenced and loaded)
  - TECT_EXT_AC500_V24 library (internal, automatically referenced and loaded)
- One example project, consisting of four examples (incl. one advanced CB610 Panel project)
- Documentation
- License code for starting the licensing procedure

Examples for use can be found in many different applications, where processing at controlled temperature levels has to be done, e.g. in extrusion and plastics but also in food & beverage or chemical applications. Especially demanding are extrusion applications, where you can find many temperature zones (up to 50) in different parts of the machinery and the overall process. For those system there will also be requirement of handling several zones in a group and several groups in a system in a structured way.

These processes require typically precise temperature control and therefore require PID instead of simple 2- or 3-point control. For example, plastics processing and especially extrusion requires the temperature to be controlled within +/- 0.5 degrees Celsius. High temp deviation can deteriorate the product quality. As PID settings need to be different, depending on the temperature zone's position along the process, e.g. the extruder screw, the adaptive option helps greatly.

Lots of machines are still using costly and difficult to setup dedicated and decentralized temperature controllers, i.e. separate controller for each zone. Due to this approach, it is not possible centrally monitoring, controlling and using data acquisition or alarm handling in a simple way. Often even the PID settings of the temperature zones need to be changed, based on the currently produced product. The decentralized system takes a much long time to make these settings and needs more attendance and skilled persons during the production.

Main purpose of TECT_TEMP_CONTROL_AC500_V24 library

TECT_TEMP_CONTROL_AC500_V24 Library gives centralized control of all the temperature zones using AC500 PLC.

TECT_TEMP_CONTROL_AC500_V24 is the main library, which provides the centralized control and auxiliary functions for all the temp zones. It also provides the AutoTune functionality for the individual zones and includes differentiation for the often different heating and cooling behavior. The blocks of this library internally call blocks of two other internal libraries (added automatically).

As this library provides centralized control, also Log and recipe functions are implemented for better user interaction and data collection. Process data can be logged on SD cardin “csv” format, which can process further for better understanding and monitoring of the system. The RECIPE function helps to store/select parameters setting for the different products.

This library also provides the option of a simplified interface to HMI´s for the CP600 Panels, which allows in a structured, modular and simple way the configuration and monitoring of all temperature zones.

1.5.12.1.2 Requirements

Hardware

The TECT_TEMP_CONTROL_AC500_V24 library is meant to run on AC500 CPUs and CPU firmware 2.4 or higher.

The number of temperature zones and auxiliary blocks, which can be used, is limited of course by memory available for program and data. Therefore on the smaller AC500 eCo typically only 3 temperature zones are possible.

Further needed are an memory card for the logging, a battery for buffering and a CP635 Panel if the panel project should be used.

Software

For TECT_TEMP_CONTROL_AC500_V24 library Automation Builder 1.2 or higher is required for programming purposes.

1.5.12.1.3 Control principle

State machine

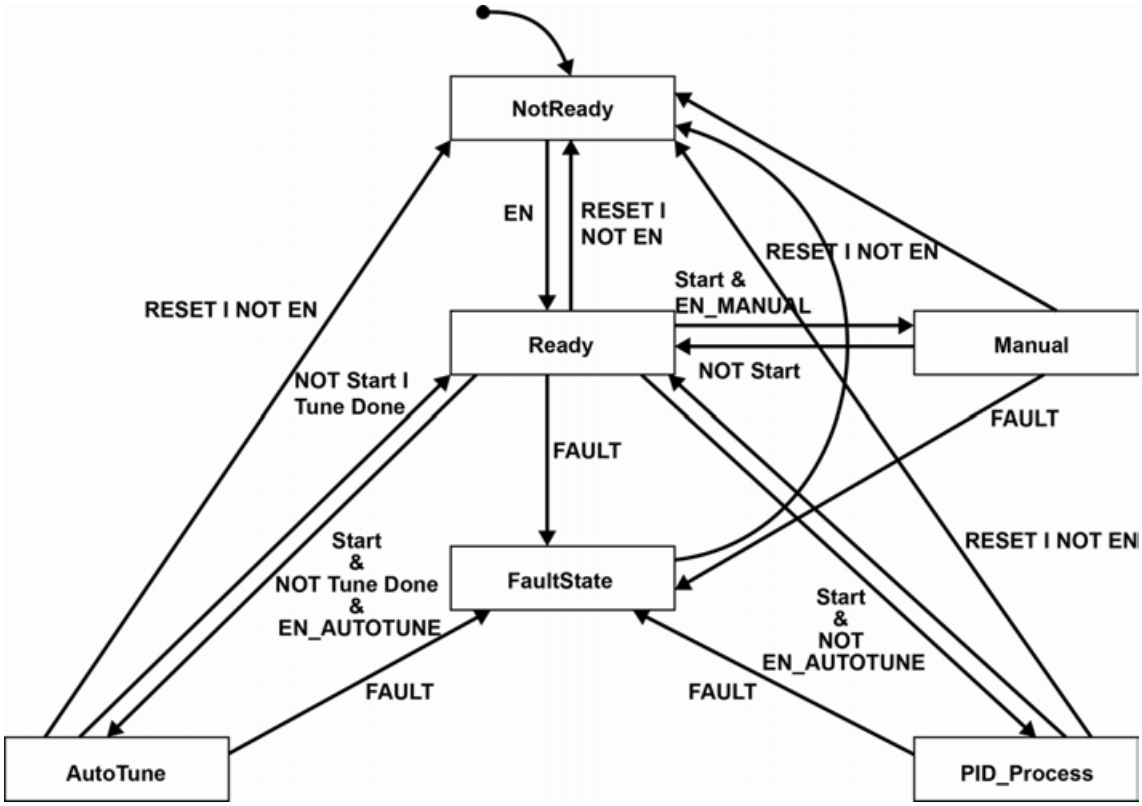


Fig. 579: Control Principle of State Machine

The PS564 Temperature Control library uses a state machine to accomplish all work, including:

- Automatic PID control (state PID_Process): Normal operation (see Automatic PID Control ↳ *Chapter 1.5.12.1.3.2 “Automatic PID control” on page 3270*)
- Auto tuning procedure (state AutoTune): Can be used during the commissioning phase to calculate the optimized parameters for PID controller. (see Auto Tuning ↳ *Chapter 1.5.12.1.3.3 “Auto tuning” on page 3271*)
- Manual Control (state Manual): Open loop control with directly change of duty cycle
- Alarm handling

Automatic PID control

PID controller principle

A proportional-integral-derivative controller (PID controller) is a closed-loop control method, widely used in industrial control systems. A PID controller manipulates a control variable to minimize the error between process actual value and a desired process set point.

The PID controller algorithm involves three parameters: The proportional, the integral and derivative values, abbreviated as P, I, and D.

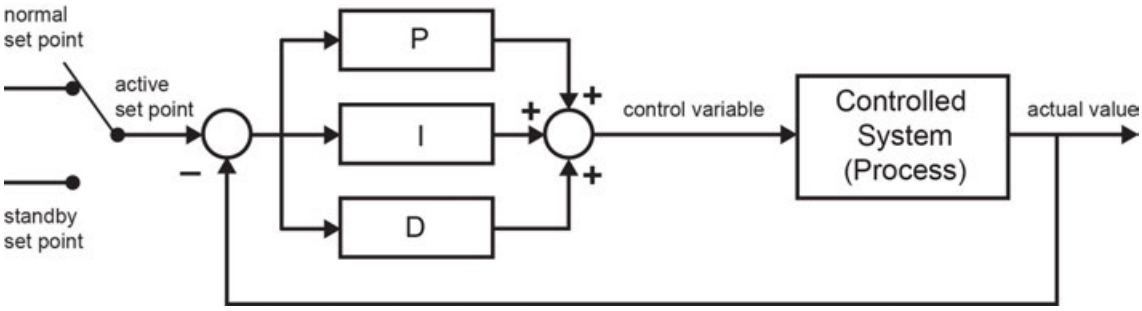


Fig. 580: PID Controller Principle

In the PS564 temperature control library, a PID controller with the first order delay T1 is used to prevent spike of derivative part D.

In the world of temperature control, set point is the desired temperature of the process, actual value is the actual temperature, and control variable is the PWM signal for controlling the heater and cooler.

In the PS564 Temperature Control library, the PWM signal is presented in means of digital value (binary outputs) as well as analogous value (duty cycle).

Set point selection

In the PS564 Temperature Control library are 3 set points available:

- Normal set point: For normal operation.
- Standby set point: For standby operation with a lower temperature to save energy, e.g. during lunch time.
- Tune set point: For auto tuning.

The first two set points use the same PID controller and the same parameters (see figure "Control Principle of State Machine"Fig. 579). The tune set point skip the PID controller and is used during an auto tuning procedure (see Auto Tuning ↳ *Chapter 1.5.12.1.3.3 “Auto tuning” on page 3271*).

Auto tuning

A controller with adaptive functionality helps the user to find out the optimized controller parameters of an unknown system using auto tuning procedure. This simplifies the machine commissioning and thus the user saves time and cost to reach an optimized result.

Function principle

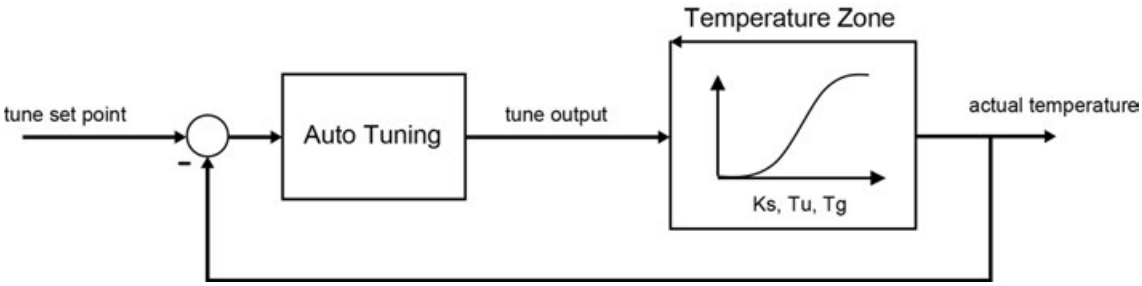


Fig. 581: Function principle of Auto Tuning

The auto tuning process serves to determine optimized controller parameters for the PID controller. The auto tuning identifies the parameters of one temperature zone (controlled system) and calculates the optimized controller parameters for that zone. The dynamic behavior of these thermal processes can be approximately acquired by means of the following characteristic quantities, which can be defined on the basis of the transfer function:

- Tu: Delay time
- Tg: Stabilization time
- Ks: Transfer factor (controlled system gain)

At the end of the auto tuning process, the actual progression of the transfer function is replaced by the linear-progressing reversing tangent placed at its turning point.

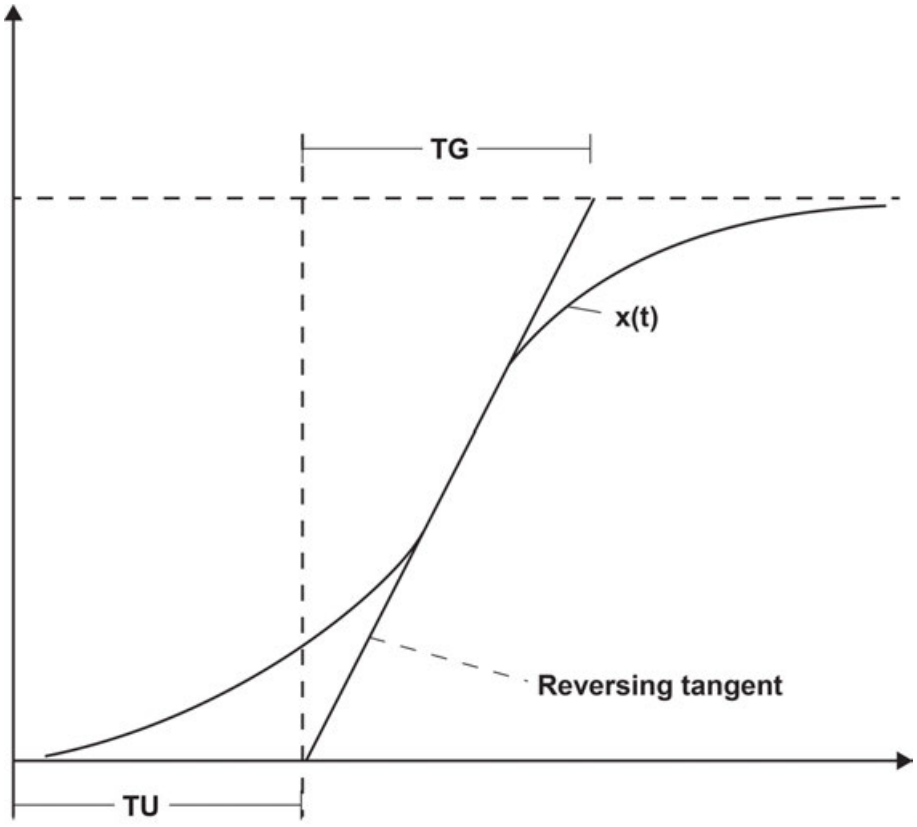


Fig. 582: Function principle of Auto Tuning

The figure above represents the transfer function, the reversing tangent and the resulting characteristic quantities for a temperature control process. As long as Tu, Tg, Ks are acquired, PID controller parameters like KP, TI, TD and T1 will be calculated.

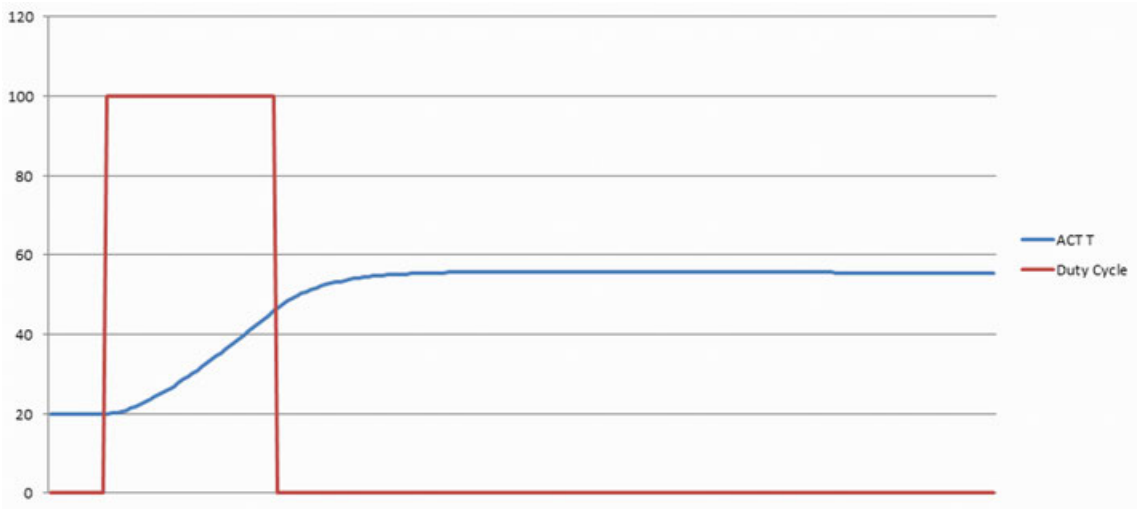
Auto tuning is only allowed when the actual temperature has reached its steady state. The auto tuning can be executed both during process start-up (initial settings) and when the value of the desired set point changes. This allows auto tuning of the control parameters in case of a process which must be operated at different set points and where the initially fixed control parameters may lead to undesired control response due to set point change. In both cases, the relative change should be as large as possible to ensure a precise measurement during the auto tuning, thus a good result of PID parameters.

Continuous auto retuning of the controller parameters is not built in.

Auto tuning for heating and cooling

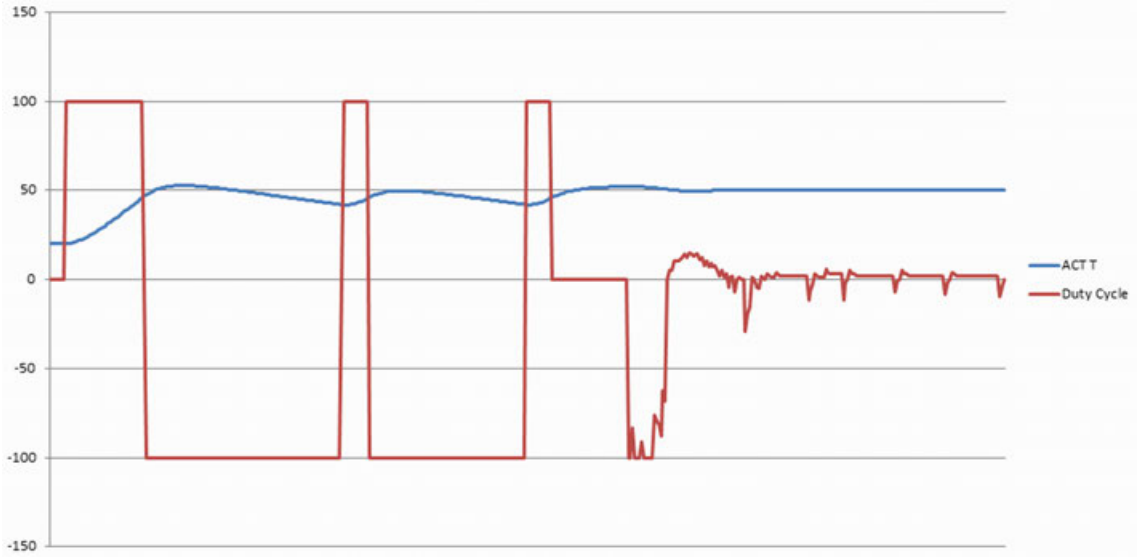
There are two possibilities for heating and cooling during auto tuning process:

Only heating



- System will be heated up with predefined Tune Output (100% duty cycle as default) until the tune set point is reached.
- Control parameters KP, TI, TD, T1 will be calculated using the temperature change (ACT T curve) during the auto tune.
- User can decide, if he would accept the auto tune result or not.
- After that he can change the KP, TI, TD, T1 parameters manually for fine tuning.

Heating and cooling



- System will be heated up with predefined Tune Output (100% duty cycle as default) until the tune set point is reached.
- Then the system will be cooled down with predefined Tune Output (-100% duty cycle as default) until the internal defined threshold is reached.
- Control parameters KP, TI, TD, T1 will be calculated for heat and a cool factor related to heat will be calculated for cool.
- The user can decide if he would accept the auto tune result or not.
- After that he can change the KP, TI, TD, T1 parameters and cool factor manually for fine tuning.

Alarm handling

In the PS564 library are two categories of alarm:

- Fault: leads to stop of the process
- Warning: process continues with warning indication

Fault is triggered if something serious happens, which must stop the process, otherwise the machine as well as the production material can be damaged.

In practice, the user can use the individual alarm bit to do some control or give some indication out of PLC, e.g. switching off a contactor or switch on an alarm light.

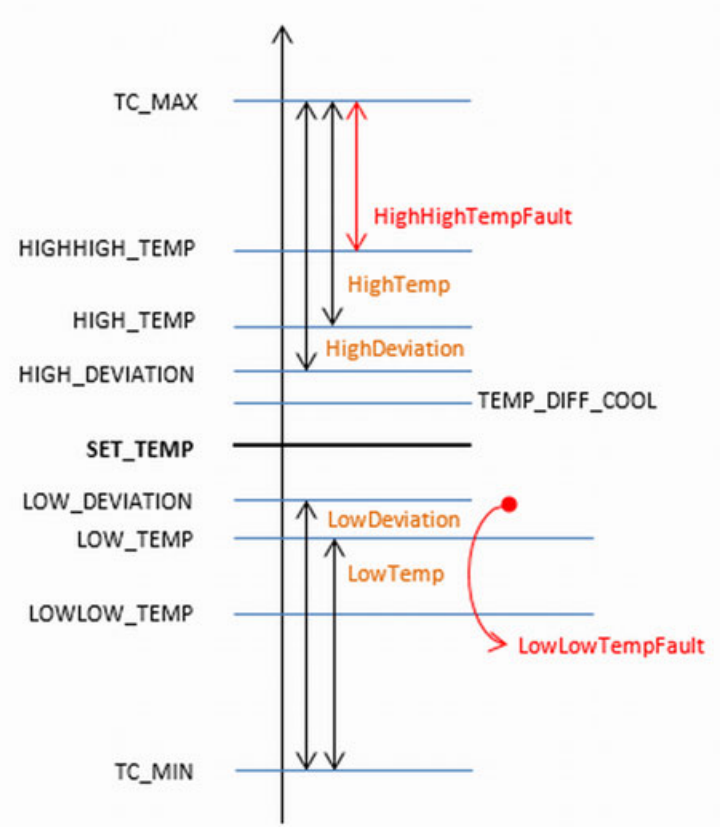
Faults

The following faults are specified:

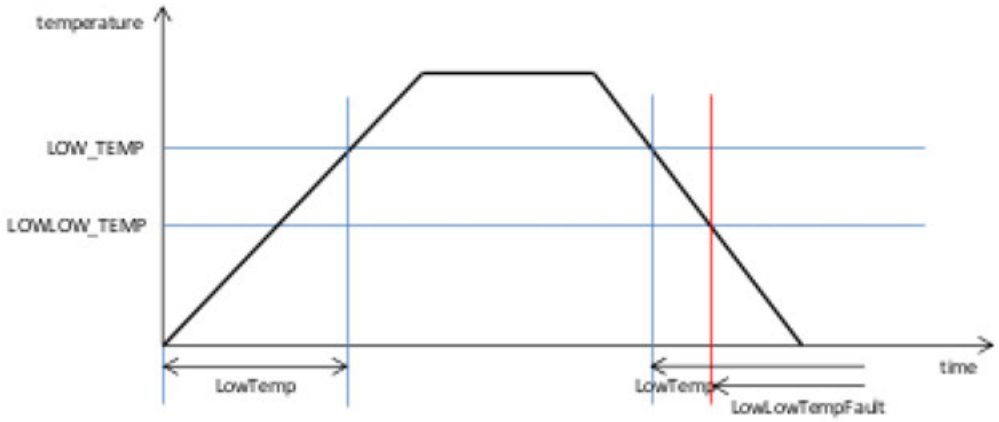
- Tune Fault
  This happens, when there is something wrong during the tune process, e.g. the temperature change is too small to calculate reasonable PID parameters.
- Thermal Coupler Fault 1
  Plausibility check of the temperature value, if it is inside the plausible range for case like wire cut.
- Thermal Coupler Fault 2
  Plausibility check of the temperature change to avoid e.g. inverted connection of two sensors.
- HighHigh Temperature Fault
  Check, if the actual temperature reaches the allowed up limit of the process.
- LowLow Temperature Fault
  Check, if the actual temperature reaches the allowed down limit of the process.

Temperature limits

There are different temperature limits for the process. They will trigger fault (red in following figures) or warning (orange in following figures) depending on the severity.

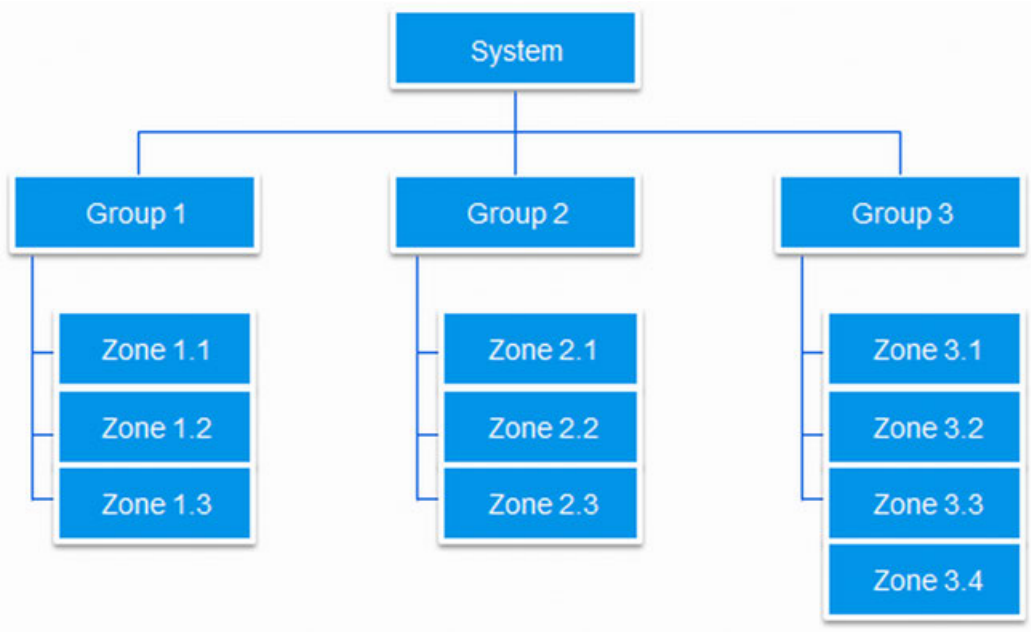


LOWLOW_TEMP check will only be active after the actual temperature has already exceeded the LOW_TEMP limit.



1.5.12.1.4 Zone management

Overview



Beside the Temperature Control process, a concept of 3-level zone management is offered by the PS564 library.

In this concept, a Temperature Control system consists of up to 255 groups, and each group consists of up to 255 zones.

The system refers to the whole machine. On the system level, it offers convenient monitoring as well as operating.

The groups are also the physical groups of the process which are normally close to each other. On the group level, several major functions are realised:

- Convenient monitoring as well as operating at group level.
- A group-wise programming of the control process (using function block TECT_GROUP) instead of programming the control process for each zone separately.
- Fault monitoring at group level.
- Coordinated output at group level.

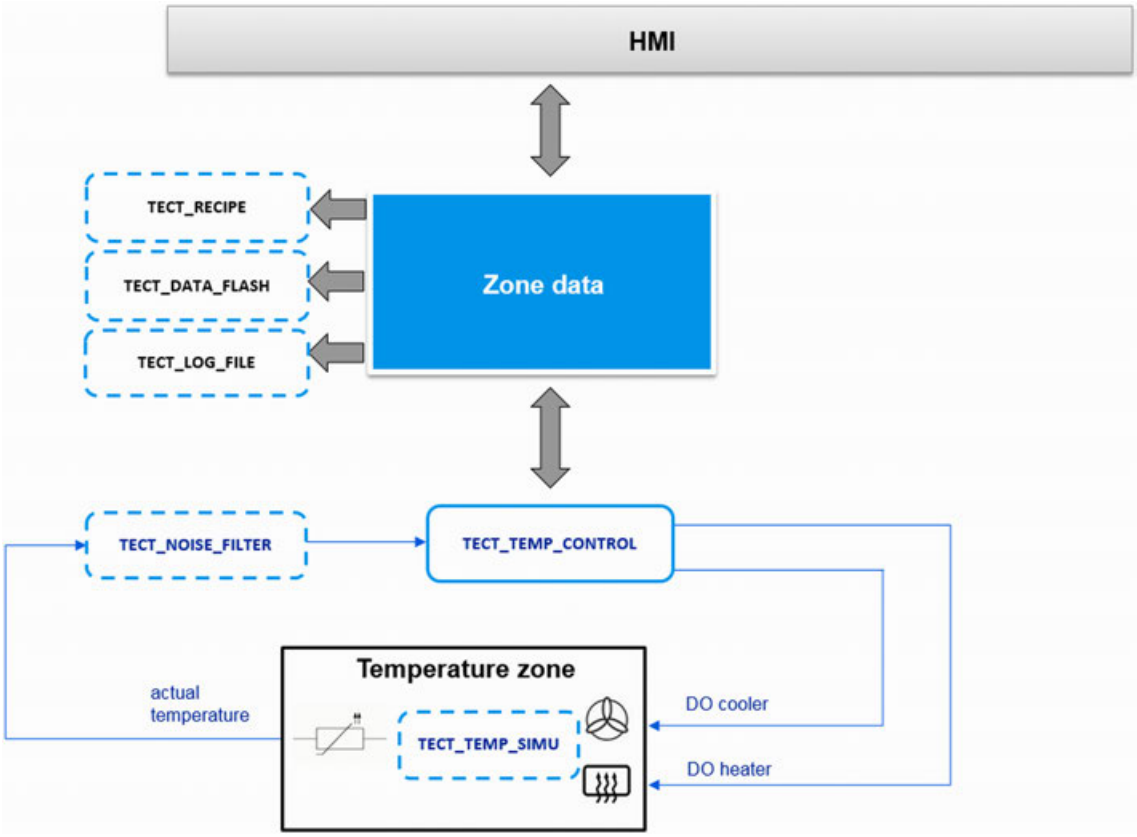
It is not mandatory to apply the zone management concept for the control process. Instead, the user can apply it with full flexiblilty: He can apply 3 levels (system-group-zone) for a large system, or 2 levels (group-zone) for a middle size system, or even 1 level (only zone) for a small machine. However the more zones a system contains, the bigger the benefit of the zone management concept will be experienced by the user.

One zone

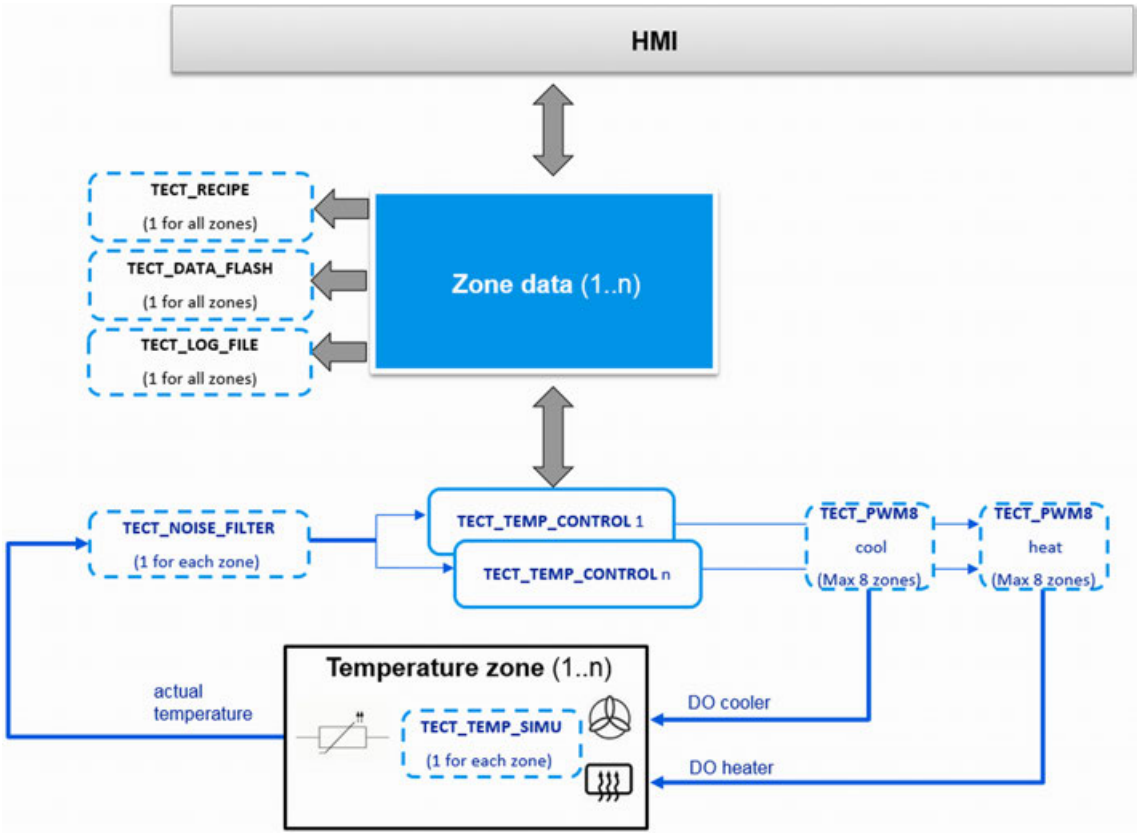
The TECT_TEMP_CONTROL is the kernel control function block, which does the control process. The input is the actual temperature feedback from the analogous sensor mounted inside the temperature zone. The outputs of the control is a PWM signal, which controls the process. It can be binary outputs, which switch the heater and cooler directly or analogous signal in duty cycle (%).

The zone data structure contains all set and status values, which are relevant for the process. The set values can be written from the user, whereas the status values can only be read, e.g. using a CP600 HMI operating panel. It is also the common interface for most of other function blocks in PS564 library.

The function block TECT_TEMP_SIMU can be used as one zone simulation instead of a real zone for test purpose.

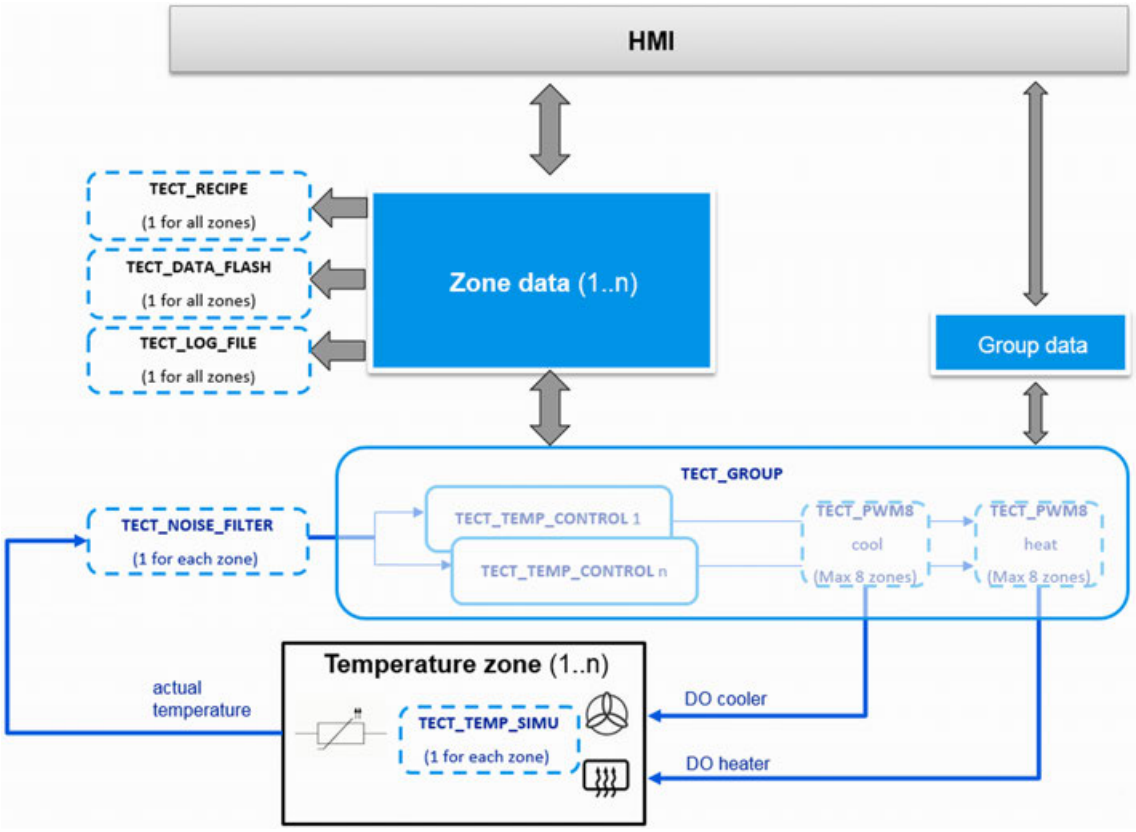


Multiple zones



For a multiple zones system, it just needs to duplicate the engineering of a one zone system. However, it is possible to use the TECT_PWM8 to coordinate the outputs.

One group



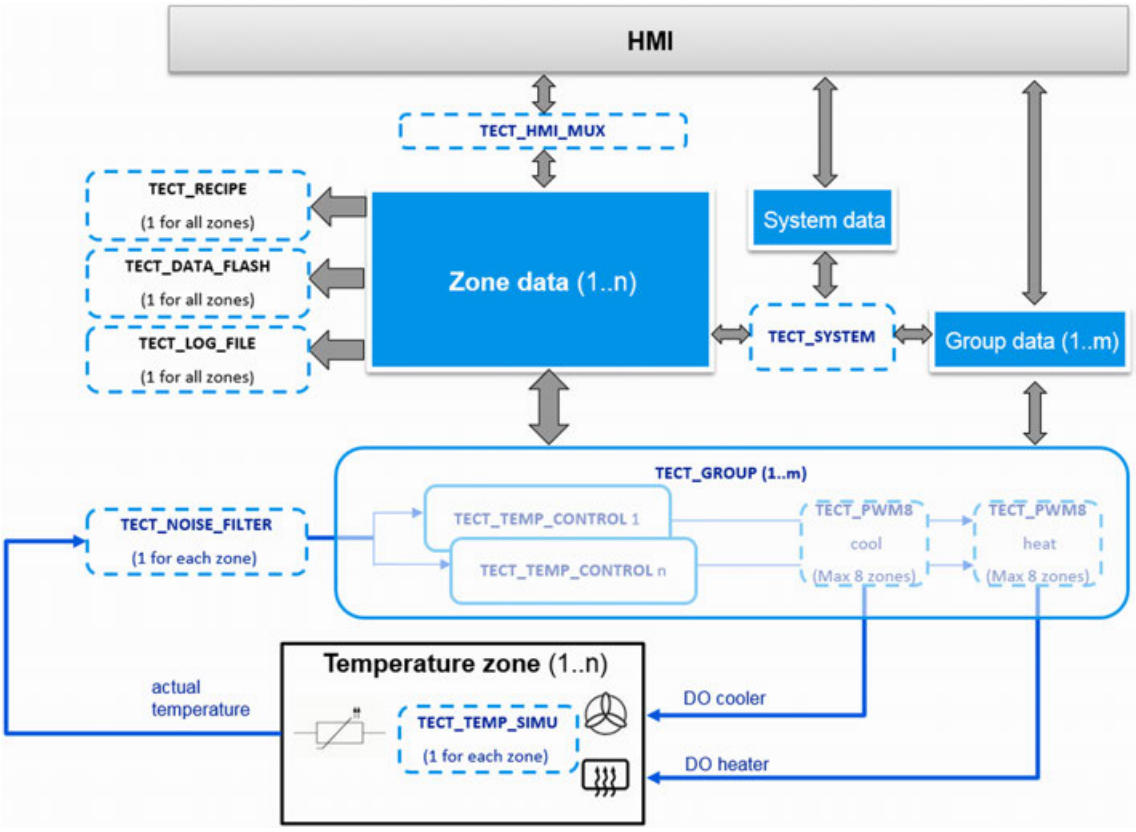
With TECT_GROUP the control function block TECT_TEMP_CONTROL is called internally multiple times for each zone.

A group data structure is needed, which contains the status of its zones as well as the control commands for the whole group, e.g. enable/disable process, enable/disable auto tune etc.

A zone data structure is always necessary to be defined for each zone.

It is possible to activate the TECT_PWM8 in TECT_GROUP internally to coordinate the outputs ↳ *Chapter 1.5.12.2.2 “TECT_GROUP” on page 3316*.

Multiple groups within a system



For a multiple groups system, it just needs to duplicate the engineering of a one group system. However, it is necessary to call TECT_SYSTEM function block and define a system data structure, which contains the status of its groups as well as the control commands for the whole system, e.g. enable/disable process, enable/disable auto tune etc.

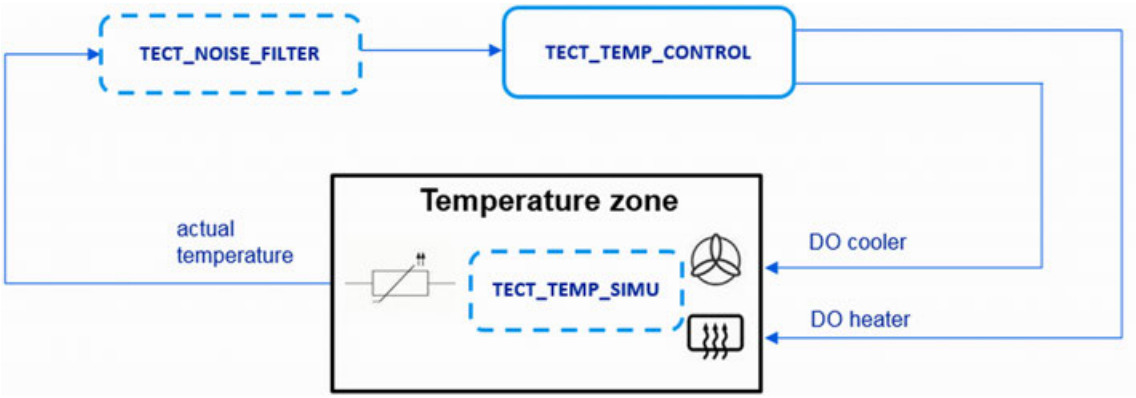
In this configuration, a TECT_HMI_MUX function block can be used as the direct communication interface to a HMI. It has the multiplexer principle and passes only one group of zone data from AC500 PLC to HMI to reduce the communication load. The usage of TECT_HMI_MUX will only make sense if more than 2 groups are defined.

1.5.12.1.5 How to realize in a program

This chapter tries to help the user to understand and combine the function blocks to realize his application. This chapter explains the principle of how to use the function blocks. The diagrams used in this chapter only demonstrate the principle instead of showing programming in the program editor. For detailed explanation of each function block please refer to another document Function Blocks Description. For how to program the application please refer to the example document.

Interface for sensors and actuators

The temperature control with PS564 is a closed loop control, as shown in the figure, the control function block TECT_TEMP_CONTROL sets the outputs as PWM signals for heater and cooler for the temperature zone. The actual temperature is connected to the input of TECT_TEMP_CONTROL as feedback.



Use a real zone

For a real zone, there are two methods for output:

- digital outputs
  Two digital outputs are available, one for heater and another for cooler. They can be connected to AC500 digital modules which has transistor outputs.
- analogous output
  o Duty Cycle as analogous output. It can be connected to AC500 analogous modules which has analogous outputs.

As feedback, a temperature sensor, e.g. thermocouple or resistance temperature detectors (RTDs), will be used. It can be connected to AC500 analog modules which supporting temperature sensor input.

Use zone simulation function block

The PS564 library has a simulation function block which can be used to simulate a temperature zone.

Consider zone management

The zone management is a feature which not only helps the user to manage his system but also give some add on functionality. But from another side, the zone management functionality as well as any further functionality will need more resource in PLC.

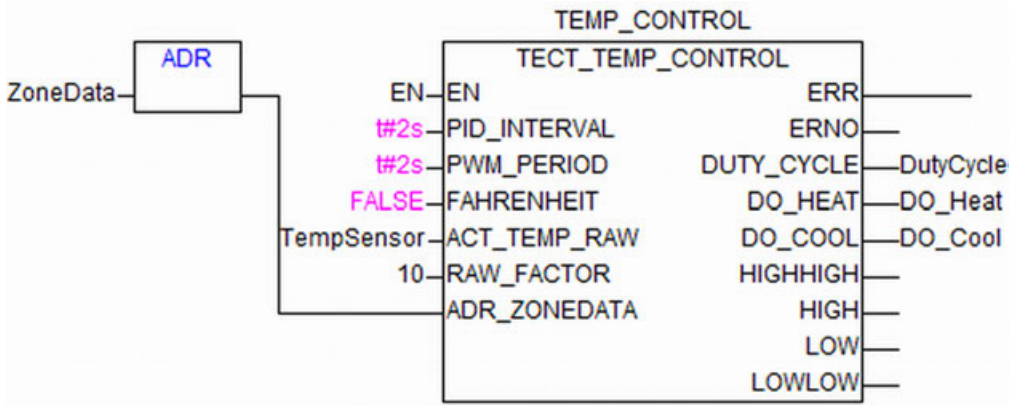
There are several criteria to decide how many levels the user should apply as zone management:

- Total number of zones
- Complexity of the machine
- The PLC type used (due to resources available in PLC)

Zone level programming

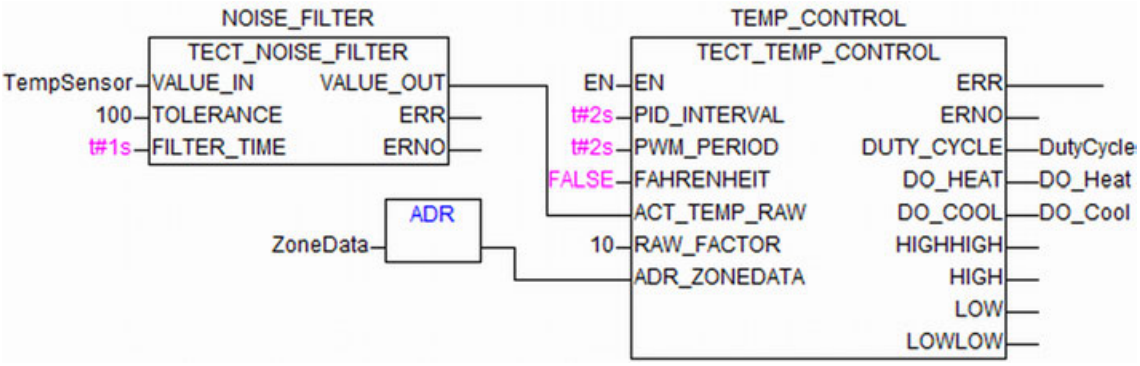
If the system has up to 5 zones, or if PLC type PM554 or PM564 is used, then only one level is recommended.

For zone level programming, the function block TECT_TEMP_CONTROL is used for each zone in the programming. The Zone Data which contains the important information of a zone needs to be connected to the function block.



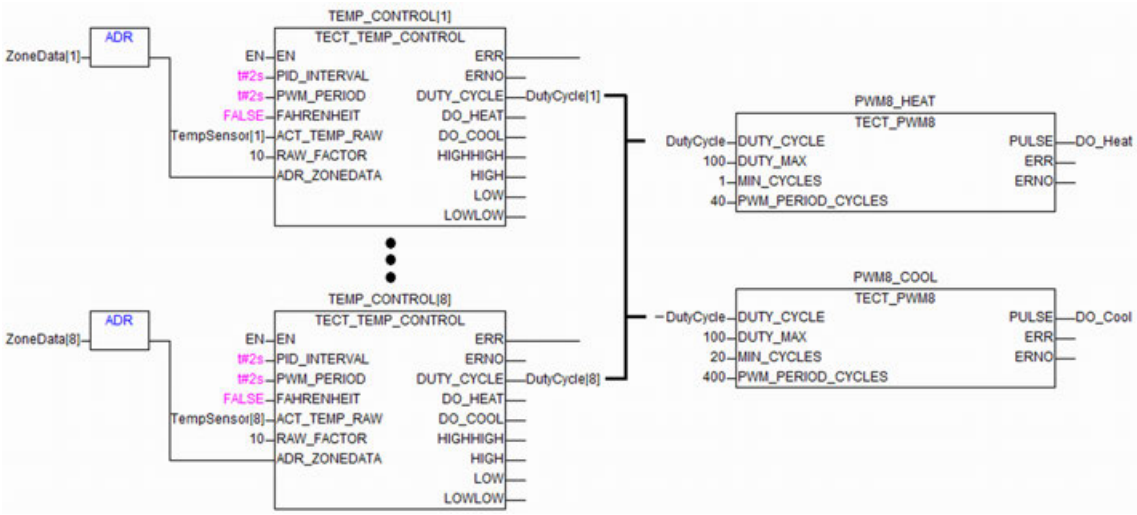
For each zone, the digital outputs DO_HEAT and DO_COOL as well as the analogous output DUTY_CYCLE can be used to connect to the output module. The temperature sensor feedback is connected to function block input ACT_TEMP_RAW.

For each zone, it is possible to add a noise filter to prevent short disturbances.



For multiple zones, it is just a multiplication of the one zone application.

Another function block TECT_PWM8 can be connected up to 8 TECT_TEMP_CONTROL function blocks to coordinate the outputs in time behavior. They have not to be switched on at the same time to avoid undesired voltage drop in the power supply system. At the same time, it compensates the error between digital outputs and duty cycle due to CPU task cycle (as resolution of digital outputs duration). For heating and cooling separate instances of this block have to be used.

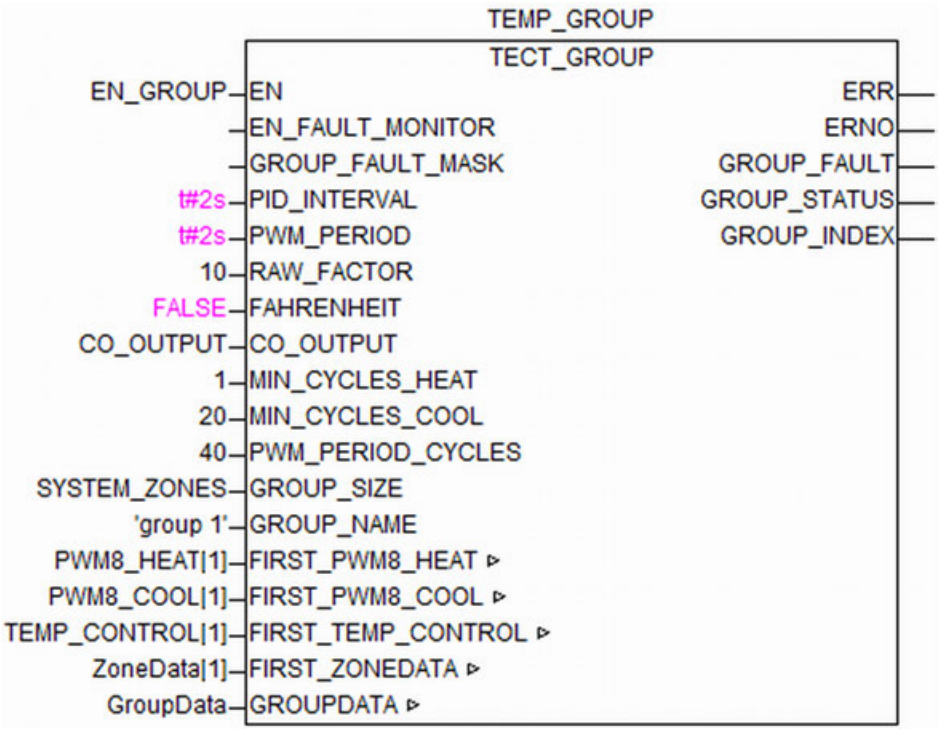


Group level programming

With group level, it will be possible to control as well as monitor the zones on group level.

Using the group level programming, one instance of function block TECT_GROUP is called in the program code instead of multiple calls of multiple zones in that group.

The first Zone Data and the Group Data structures need to be connected to the TECT_GROUP function block. Beside those, an array of TECT_TEMP_CONTROL for all zones in the group is to be declared and connected to the function block. Thus the TECT_TEMP_CONTROL function blocks will be called internally in the TECT_GROUP.



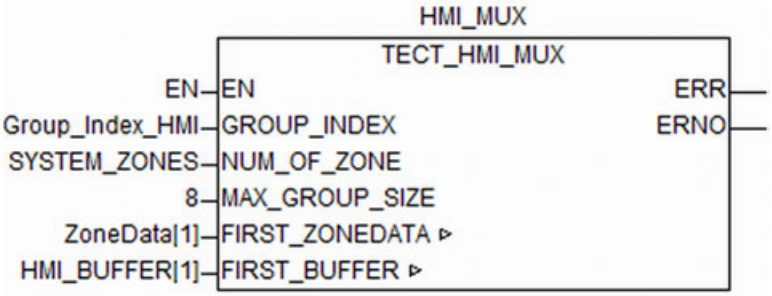
With TECT_GROUP, TECT_PWM8 function block is called internally:

- The instance for heating and cooling needs to be defined. Each instance supports 8 zones. For more than 8 zones, an array of TECT_PWM8 needs to be declared.
- This function can be enabled or disabled with input CO_OUTPUT. If it is disabled, then a dummy instance of TECT_PWM for each FIRST_PWM8_HEAT and FIRST_PWM8_COOL should be connected.
- The parameters for TECT_PWM8 are to be defined by inputs. For heating instance, MIN_CYCLES_HEAT, PWM_PERIOD_CYCLES are used. For cooling instance, MIN_CYCLES_COOL and PWM_PERIOD_CYCLES x rPeriod_Fact (in Machine Set of Zone Data structure) as PWM period.

Besides that, the user can enable the group fault monitor with input EN_FAULT_MONITOR with defined GROUP_FAULT_MASK. This function will set all zones in the group to fault state as soon as one zone in the group has fault (according to GROUP_FAULT_MASK).

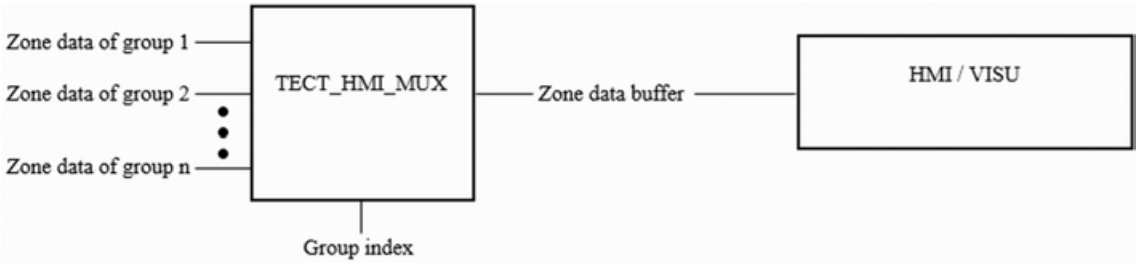
For multiple groups, it is just a multiplication of the one group application.

Another function block TECT_HMI_MUX can be used for multiple groups to reduce communication data as well as reduced engineering effort for Panel project.



It works as a multiplexer, group index decides the zone data of which group will be active for user, so that it can be read to buffer as well as be written from buffer.

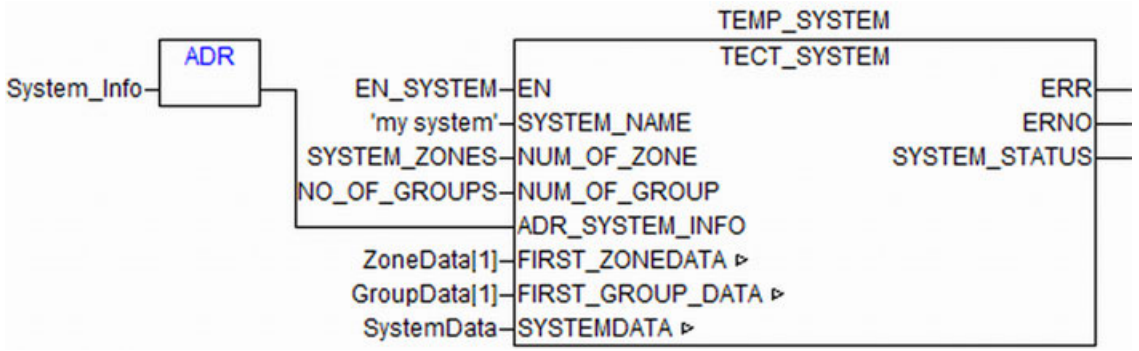
In this way, the communication data between AC500 and HMI can be reduced from n group to one group. At the same time the HMI project can also be programmed for one group instead of n groups.



System level programming

With system level, it will be possible to control as well as monitor all the zones on system level. This makes it convenient by a system with a lot of zones.

When using system level programming, the group function block TECT_GROUP will still be applied as in group level programming. Also the TECT_HMI_MUX can be used. Thus the functionalities on group level are still available.



The ADR_SYSTEM_INFO points to the ZONE_INFO which contains the actual temperature and error word. They will be used directly in the HMI for alarm handling and trends because they should be delivered to HMI without interruption by TECT_HMI_MUX.

Data and settings

Data structure

Depending on which level of zone management is applied, zone data, group data or system data will be used for saving the status and settings of control.

Zone Data structure is essential structure which contain all information for a zone. It is highly recommended to save zone data as retain variable for power safe. It has five sub structures:

- Internal status: status which is used internally in the function block, e.g. control state, group and zone index
- Machine status: machine specific status, e.g. PID parameters calculated from AutoTune.
- Process status: process specific status, e.g. actual temperature, status word, error word, duty cycle.
- Machine set values: machine specific set values, e.g. PID parameters, HighHigh and LowLow temperature limits
- Process set values: process specific values, e.g. control word, set point, High and Low limits.
- All status can only be read whereas all set values can be read and written by the user.

For group level, there is group data which contains the status of the group (e.g. control state of the whole group) and the control value of the group (e.g. enable / disable process of the zones in the group).

For system level, there is system data which contains the status of the system (e.g. control state of the whole system) and the control value of the system (e.g. enable / disable process of the zones in the system).

Initial zone data settings

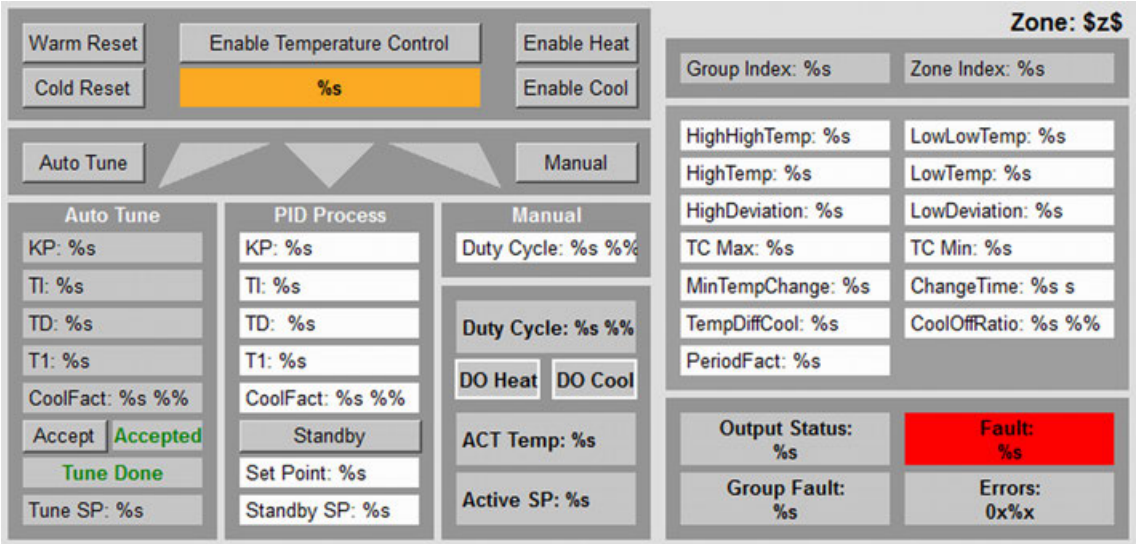
To simplify the start process of temperature control, it is highly recommended to have an initialization procedure of relevant zone data values, as those values are necessary for starting the process. Otherwise those values need to be set using HMI. For example:

- Set point:
  ZoneData.tsProcessSet.rsetpoint:=55;
- Hysteresis to switch on and off cooling: ZoneData.tsMachineSet.rTemp_diff_cool:=1; (*If value> 0: cooling will be ON with 100% when ACT_TEMP > set point + rtemp_diff_cool x iCool_Off_Ratio. If value = 0: rtemp_diff_cool is deactivated, PID controller will be active for cooling*)
- Values for Thermal Coupler Fault 2: ZoneData.tsMachineSet.uiMIN_TEMP_CHANGE:=1; ZoneData.tsMachineSet.uiCHANGE_TIME:=60;
  (*Inside uiCHANGE_TIME in seconds, a temperature change of 1 of raw value from sensor should be detected, otherwise fault will be triggered*)
- Values for Thermal Coupler Fault 1:
  ZoneData.tsMachineSet.rTC_max:=400;
  ZoneData.tsMachineSet.rTC_min:=-50;
  (*Measuring range of thermal sensor, e.g. -50°C to 400°C. If the detected sensor value is out of *)
- Limits for temperature:
  ZoneData.tsMachineSet.rhighhigh_temp:=120;
  ZoneData.tsMachineSet.rlowlow_temp:=0;
  ZoneData.tsProcessSet.rhigh_temp:=ZoneData.tsProcessSet.rsetpoint+10;
  ZoneData.tsProcessSet.rlow_temp:=ZoneData.tsProcessSet.rsetpoint-10;
  ZoneData.tsProcessSet.rhigh_deviation:=0.5;
  ZoneData.tsProcessSet.rlow_deviation:=0.5;
- Necessary temperature change to do AutoTune: ZoneData.tsMachineSet.iTune_Step:=500;
  (*This is a raw sensor value, for a RAW_FACTOR 10, it will be 50 degree.*)
- Set control bits:
  ZoneData.tsProcessSet.wcontrolWord.1:=TRUE;(*heat enabled*)
  ZoneData.tsProcessSet.wcontrolWord.2:=TRUE;(*cool enabled*)
  ZoneData.tsProcessSet.wcontrolWord.6:=TRUE;(* accept autotune result*)

Please refer to the example for the detailed realization of this data initialization part.

1.5.12.1.6 How to start control process

To have a better overview of the control process, the zone data visualization element can be used.



Do AutoTune

Prepare the settings

Before start the auto tuning process, the following points should be made clear:

- Auto tuning only for heating or for both heating and cooling.

If only heating is enabled for auto tuning, the following values should be set:

- Control bit: ZoneData.tsProcessSet.wControlWord.1:=TRUE;(*heat enabled*)
- ZoneData.tsProcessSet.wControlWord.2:=FALSE;(*cool disabled*),
  Or if cooling enabled with hysteresis:
  ZoneData.tsProcessSet.wControlWord.2:=TRUE;(*cool enabled*)
  ZoneData.tsMachineSet.rTemp_diff_cool>0;

If both heating and cooling are enabled for auto tuning, the following values should be set:

- Control bits: ZoneData.tsProcessSet.wControlWord.1:=TRUE;(*heat enabled*)
  ZoneData.tsProcessSet.wControlWord.2:=TRUE;(cool enabled*)
- Cooling with hysteresis must be disabled with:
  ZoneData.tsMachineSet.rTemp_diff_cool:=0;

Besides that, the necessary temperature change to do AutoTune needs to be checked: ZoneData.tsMachineSet.iTune_Step:=500; (*This is a raw sensor value, for a RAW_FACTOR 10, it will be 50 degree.*) The larger the value, the better the result will be. It is recommended to have 500 as raw sensor value.

Start AutoTune

Enable the control bit ZoneData.tsProcessSet.wControlWord.5 to enable AutoTune, and enable the control bit ZoneData.tsProcessSet.wControlWord.0 to start the control process.

As long the AutoTune is started, the control state machine ZoneData.tsInternalStatus.ControlState will change from Ready to AutoTune. The tune setpoint ZoneData.tsInternalStatus.rTune_SetPoint will be set internally: as long as the process set point ZoneData.tsProcessSet.rsetpoint fulfills the necessary temperature change, namely the different against actual temperature is greater than ZoneData.tsMachineSet.iTune_Step, then the AutoTune will take the process set point as tune setpoint and transfer it to the active setpoint ZoneData.tsProcessStatus.rActive_SetPoint. Otherwise it will set a higher tune setpoint, which is not lower than the HighHigh temperature limit.

AutoTune needs a long time depending on the system. If the cooling is also activated, then it will be even longer.

AutoTune done

If the AutoTune is done with success, a status bit ZoneData.tsProcessStatus.wStatusWord.6 for tune done is set and the PID parameters (KP, TI, TD, T1) will be calculated and the control state will change back to ready. If the control bit ZoneData.tsProcessSet.wControlWord.6 (accept AutoTune result) is TRUE, then the PID values will be copied from AutoTune result to PID values for automatic PID process with an indication ZoneData.tsProcessStatus.wStatusWord.7 (AutoTune accepted) and at the same time the automatic process will start with changing control state to PID_Process.

After accepting the AutoTune result, the control bit ZoneData.tsProcessSet.wControlWord.6 can be disabled for a fine tuning of the PID parameters.

Even if the zone data is saved as retain variable, it is recommended to save the AutoTune results and PID parameters into PLC internal flash or as a recipe file directly after AutoTune is done. Because the values can be reset by cold resetting the PLC or by downloading new PLC program. Please refer to the chapters for those two functionalities.

If AutoTune cannot be completed due to tune fault, the control state will change to Fault state with fault bit ZoneData.tsProcessStatus.wStatusWord.15 and error code TECT_TuneFault in ZoneData.tsProcessStatus.wErrors and ZoneData.tsProcessStatus.Latest_Error. To reset the fault, the enable AutoTune control bit needs to be disabled first and a warm reset with ZoneData.tsProcessSet.wControlWord.13 of process is necessary.

The AutoTune fault can happen if the temperature change between actual temperature and set point is not big enough. For such case, the user needs to wait until the machine temperature has been cooled down.

Restart AutoTune

To restart AutoTune, the cold reset of process is necessary. The cold reset resets not only the fault state but also the tune status including status bit tune done and tune accepted. After that, a new AutoTune process can be started again.

Do PID Process

PID process is an automatic process with a PID controller which outputs control variables according to actual temperature as feedback. To start PID process, the valid PID parameters (KP, TI, TD, T1) are necessary. Internally, it check the following conditions for valid parameters:

- KP > 0,
- TI >= 0,
- (TD>0 AND T1>0) OR (TD=0)

If the values are not valid, then the PID process cannot be started with a warning: NoPIDProcess.

The PID parameters can be accepted from AutoTune result, can be set during fine tuning or can even be set manually without AutoTune process.

If the user has already write the PID parameters into internal flash or a recipe file, then they can be read back from flash or a recipe file.

For PID process, two set points are possible. Normal set point for production and standby set point for energy saving. The switch between both is done with a control bit ZoneData.tsProcessSet.wControlWord.4 (enable standby set point). The currently used set point will be transferred to active set point.

Do manual control

The process can be controlled manually by changing duty cycle directly. The manual control can even interrupt the AutoTune and PID process.

To start manual control, the control bit ZoneData.tsProcessSet.wControlWord.3 (enable manual control) is set and the control state will change to Manual. After that, the manual duty cycle ZoneData.tsProcessSet. iManual_DutyCycle can be changed to output the control value directly.

Observe the status

There are different status which indicate what is exactly going on in the process.

Process status

Control state ZoneData.tsInternalStatus.ControlState is the main state machine for the whole process.

Status word ZoneData.tsProcessStatus.wStatusWord for general status, e.g. tune done and tune accepted.

Output status ZoneData.tsProcessStatus.OutputStatus indicates the status of PWM output, e.g. the output is in automatic heat or manual cool.

Errors (Error Word) ZoneData.tsProcessStatus.wErrors indicates all the errors (warnings and faults) which are active.

Latest error ZoneData.tsProcessStatus.Latest_Error shows only the latest activated error out of error word.

There are two levels of errors: faults (ZoneData.tsProcessStatus.wStatusWord.15=TRUE) and warnings (ZoneData.tsProcessStatus.wStatusWord.14=TRUE). Both will set the bits in error word and latest error.

The difference is: Faults will interrupt the control state and change it to Fault state. To resume the process, the control process needs to be warm reset to go to ready state first.

Warnings are only indications for the user and do not interrupt a running process.

Nevertheless, each bit in the error word can be programmed individually to control user defined functions, e.g. to control a contactor / relay or switch on a warning light.

Besides the errors for an individual zone, it is also possible to combine the zone faults to group faults.

This function will set all zones in the group to fault state and group fault bit (ZoneData.tsProcessStatus.wStatusWord.13) to TRUE as soon as one zone in the group has fault. Please refer to TECT_GROUP description for more details.

Function block status

For the function blocks in the library, there are different outputs for status of the function block, e.g. DONE, ERR, ERNO and BUSY. For ERNO (function block error numbers), please refer to that function block description. For certain special internal functionalities, e.g. CAA File Library or Flash operation, there will be a link for the original error description of those part.

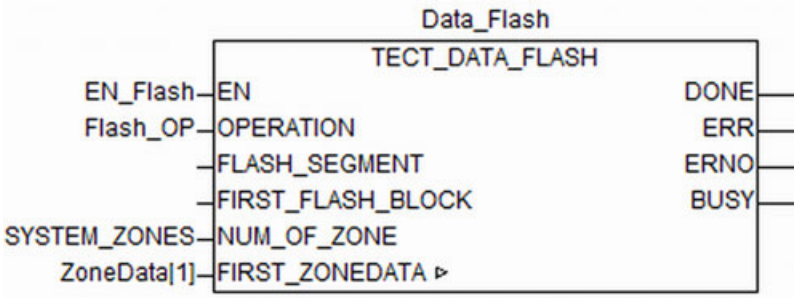
1.5.12.1.7 How to save control data

It is highly recommended to save the set values of zone data in case of power failure or reset of PLC, especially the result of AutoTune and the process PID parameters. For this purpose, there are to function blocks with different PLC resource demands are available for different function range:

TECT_DATA_FLASH: saves/loads AutoTune result and PID parameters into/from internal flash.

TECT_RECIPE: saves/loads all relevant control parameters and settings into/from recipe file in CSV format.

Save AutoTune and PID parameters into flash



Each PLC in AC500 range has a power safe internal flash data area which can be written, read and deleted from AC500 program. TECT_DATA_FLASH uses the internal flash to save and load the AutoTune results and PID parameters including:

- Internal parameters: rKS, rTU, rTG and wCoolFact_Tune
- Machine set parameters: wCoolFact, rKP, rTI, rTD, rT1

The user can perform the following operations by using this function block in the flash memory: Read, Write, Delete, Check Read, Delete Write and Check Write.

WRITE The user can save the above variables into flash by the WRITE operation by selecting the flash segment. When the WRITE command is enabled the parameters, which are listed above, will be copied and stored in the flash memory of the PLC. If the PLC is restarted or a cold reset is done these data will still be retained. Before a fresh set of data is written to the flash memory, it has to be cleared. Hence delete operation has to be executed before rewrite of the flash data.

READ The user can copy the data from the flash memory of PLC into the structure TEMPZONE_DATA by this command. When the READ operation is used, based on the flash block and segment configuration, it will read the data from a respective location. This is useful to restore/ load the settings of tune values.

DELETE This operation will clear all the data in the flash memory. It will delete data in the complete selected flash segment. Before a fresh set of data is written to the flash memory, it has to be cleared. Hence delete operation has to be executed before rewrite of the flash data.

DELETE_WRITE This operation is a combination of operation DELETE and WRITE. It will clear the complete segment first and then write the set of data into flash.

CHECK_READ This operation is used, when data read has to be restricted. It will check if the values of AutoTune results in Zone Data are valid. Only if they are invalid, it will read the values from flash memory to the Zone Data structure, otherwise the operation will do nothing.

CHECK_WRITE This operation is a combination of above two operations CHECK_READ and DELETE_WRITE. When this operation is executed, it will check if the values of AutoTune results in Zone Data are valid. Only if they are invalid, it will read the values from flash memory to the Zone Data structure. After that, it clears the flash with DELETE operation and write the set of data into flash.

In total they are 32 bytes which a data block in the flash is used for one zone. As one segment has 1927 data blocks, the maximum number of zones supported for the TECT_DATA_FLASH is 1927.

For a new data write processing to flash, the flash needs to be deleted first. This can be done with TECT_DELETE before TECT_WRITE is to be executed. As an alternative, TECT_DELETE_WRITE combines the two operations in one. *Chapter 1.5.12.2.9 “TECT_DATA_FLASH” on page 3338*

Save set values as recipe file

Function principle

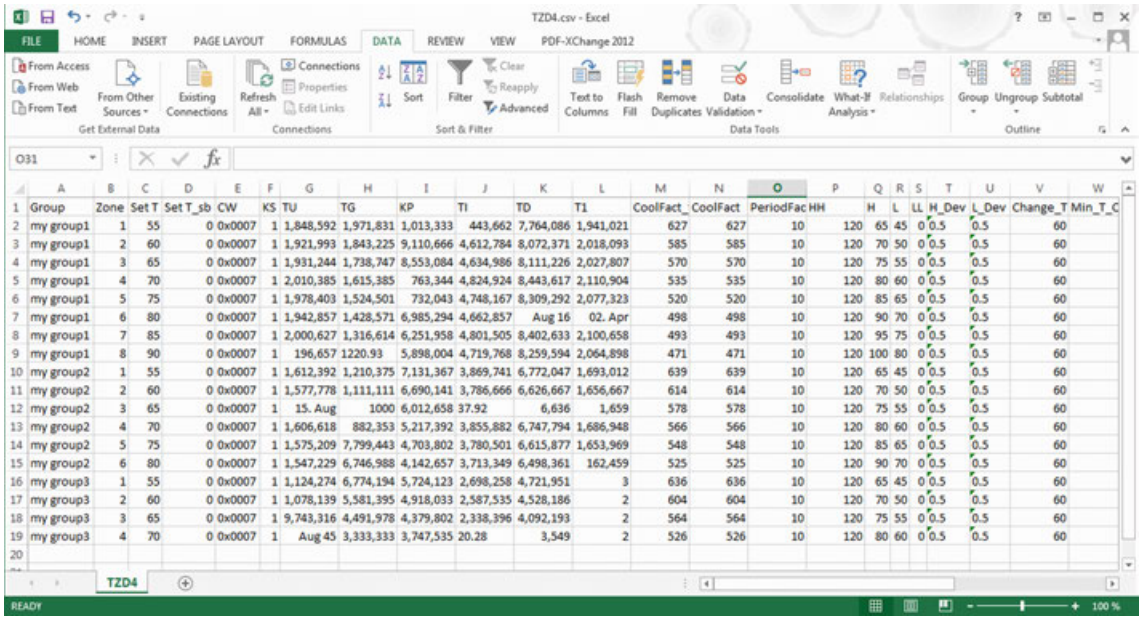
The function block TECT_RECIPE can save/write the necessary control settings to a CSV file and load/read the settings from the CSV file. The files can be accessed via an FTP client connected to AC500 FTP server. After the configuration the data can be saved into a CSV file by a WRITE operation. The same CSV file can be used in other configurations where similar initializsation is required.

Work with recipe

The Recipe function block can READ/WRITE the following parameters into the CSV file: Group_Name, Zone Index, Set temperature, Standby Set temperature, ControlWord, KS, TU, TG, KP, TI, TD, T1, CoolFact, PeriodFact, HighHigh temperature, High temperature, Low temperature, LowLow temperature, High deviation, Low deviation, Change time, Minimum temperature change, TCmax, TCmin, Temperature difference cool, Cool off ratio, Max output, Tune setpoint, Tune output, Tune min KP, Tune max KP, Tune KP Scaling and Thermocouple offset. Only one TECT_RECIPE is necessary for the whole system.

When the WRITE operation is selected, it will make a single copy of parameters listed above for the whole system into a recipe file in CSV format. The file name can be decided by the input FILE_PATH and SET_NO. If a folder is defined at FILE_PATH, e.g. ‘sdcard/Recipe’, the recipe file will be created as TZD*.csv, * stands for SET_NO. If the user wants to define his own recipe name, then a csv file name with folder needs to be defined at FILE_PATH, e.g. ‘sdcard/Recipe/MyRecipe.csv’.

The first line of the CSV file will have all the variable names and below lines list values of each variable zone by zone (see figure below).



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	Group	Zone	Set T	Set T_sb	CW	KS	TU	TG	KP	TI	TD	T1	CoolFact_	CoolFact	PeriodFac	HH	H	L	LL	H_Dev	L_Dev	Change_T	Min_T_C
2	my group1	1	55	0	0x0007	1	1,848,592	1,971,831	1,013,333	443,662	7,764,086	1,941,021	627	627	10	120	65	45	0	0.5	0.5	60	
3	my group1	2	60	0	0x0007	1	1,921,993	1,843,225	9,110,666	4,612,784	8,072,371	2,018,093	585	585	10	120	70	50	0	0.5	0.5	60	
4	my group1	3	65	0	0x0007	1	1,931,244	1,738,747	8,553,084	4,634,986	8,111,226	2,027,807	570	570	10	120	75	55	0	0.5	0.5	60	
5	my group1	4	70	0	0x0007	1	2,010,385	1,615,385	763,344	4,824,924	8,443,617	2,110,904	535	535	10	120	80	60	0	0.5	0.5	60	
6	my group1	5	75	0	0x0007	1	1,978,403	1,524,501	732,043	4,748,167	8,309,292	2,077,323	520	520	10	120	85	65	0	0.5	0.5	60	
7	my group1	6	80	0	0x0007	1	1,942,857	1,428,571	6,985,294	4,662,857	Aug 16	02. Apr	498	498	10	120	90	70	0	0.5	0.5	60	
8	my group1	7	85	0	0x0007	1	2,000,627	1,316,614	6,251,958	4,801,505	8,402,633	2,100,658	493	493	10	120	95	75	0	0.5	0.5	60	
9	my group1	8	90	0	0x0007	1	196,657	1220.93	5,898,004	4,719,768	8,259,594	2,064,898	471	471	10	120	100	80	0	0.5	0.5	60	
10	my group2	1	55	0	0x0007	1	1,612,392	1,210,375	7,131,367	3,869,741	6,772,047	1,693,012	639	639	10	120	65	45	0	0.5	0.5	60	
11	my group2	2	60	0	0x0007	1	1,577,778	1,111,111	6,690,141	3,786,666	6,626,667	1,656,667	614	614	10	120	70	50	0	0.5	0.5	60	
12	my group2	3	65	0	0x0007	1	15. Aug	1000	6,012,658	37.92	6,636	1,659	578	578	10	120	75	55	0	0.5	0.5	60	
13	my group2	4	70	0	0x0007	1	1,606,618	882,353	5,217,392	3,855,882	6,747,794	1,686,948	566	566	10	120	80	60	0	0.5	0.5	60	
14	my group2	5	75	0	0x0007	1	1,575,209	7,799,443	4,703,802	3,780,501	6,615,877	1,653,969	548	548	10	120	85	65	0	0.5	0.5	60	
15	my group2	6	80	0	0x0007	1	1,547,229	6,746,988	4,142,657	3,713,349	6,498,361	162,459	525	525	10	120	90	70	0	0.5	0.5	60	
16	my group3	1	55	0	0x0007	1	1,124,274	6,774,194	5,724,123	2,698,258	4,721,951	3	636	636	10	120	65	45	0	0.5	0.5	60	
17	my group3	2	60	0	0x0007	1	1,078,139	5,581,395	4,918,033	2,587,535	4,528,186	2	604	604	10	120	70	50	0	0.5	0.5	60	
18	my group3	3	65	0	0x0007	1	9,743,316	4,491,978	4,379,802	2,338,396	4,092,193	2	564	564	10	120	75	55	0	0.5	0.5	60	
19	my group3	4	70	0	0x0007	1	Aug 45	3,333,333	3,747,535	20.28	3,549	2	526	526	10	120	80	60	0	0.5	0.5	60	
20																							

Fig. 583: Recipe data in the CSV file after WRITE operation

This function block can be used to save the dedicated settings of a Temperature Control system. Furthermore, as its name says, it can also be used to manage multiple sets of control settings using multiple recipe sets

Configure access to memory card through FTP client in the Automation Builder.

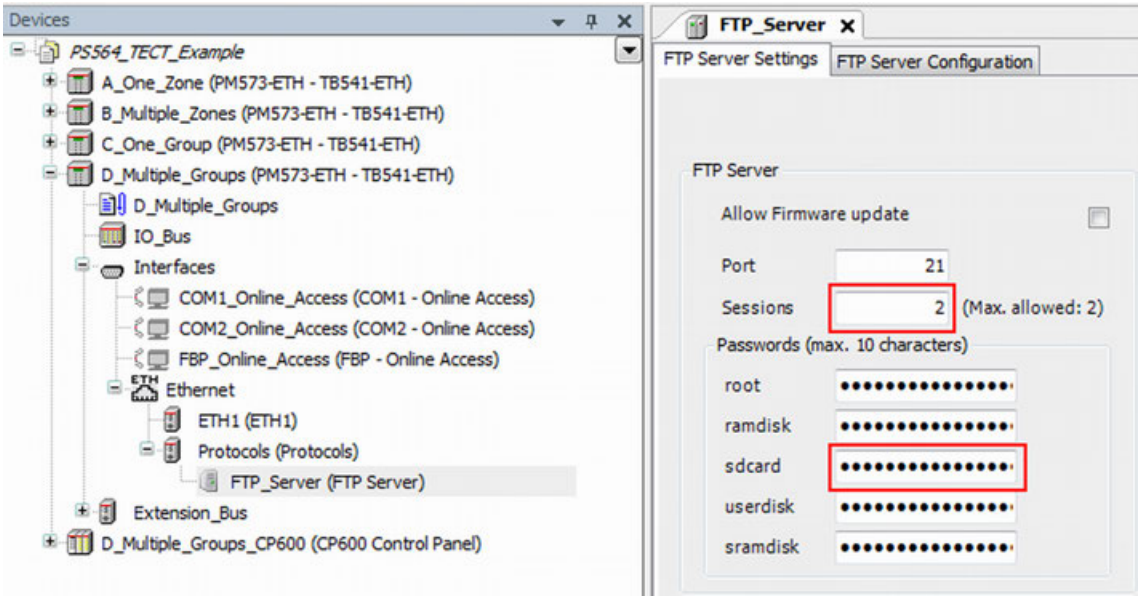


Fig. 584: Configuration of memory card access through FTP client in Automation Builder

For READ and WRITE operation on the memory card the access needs to be configured in the Automation Builder using FTP client. At this place the password for the memory card access is set.

For the following activities the tool "File Zilla" is used to access the memory card. But it is also possible to use other FTP client tools.

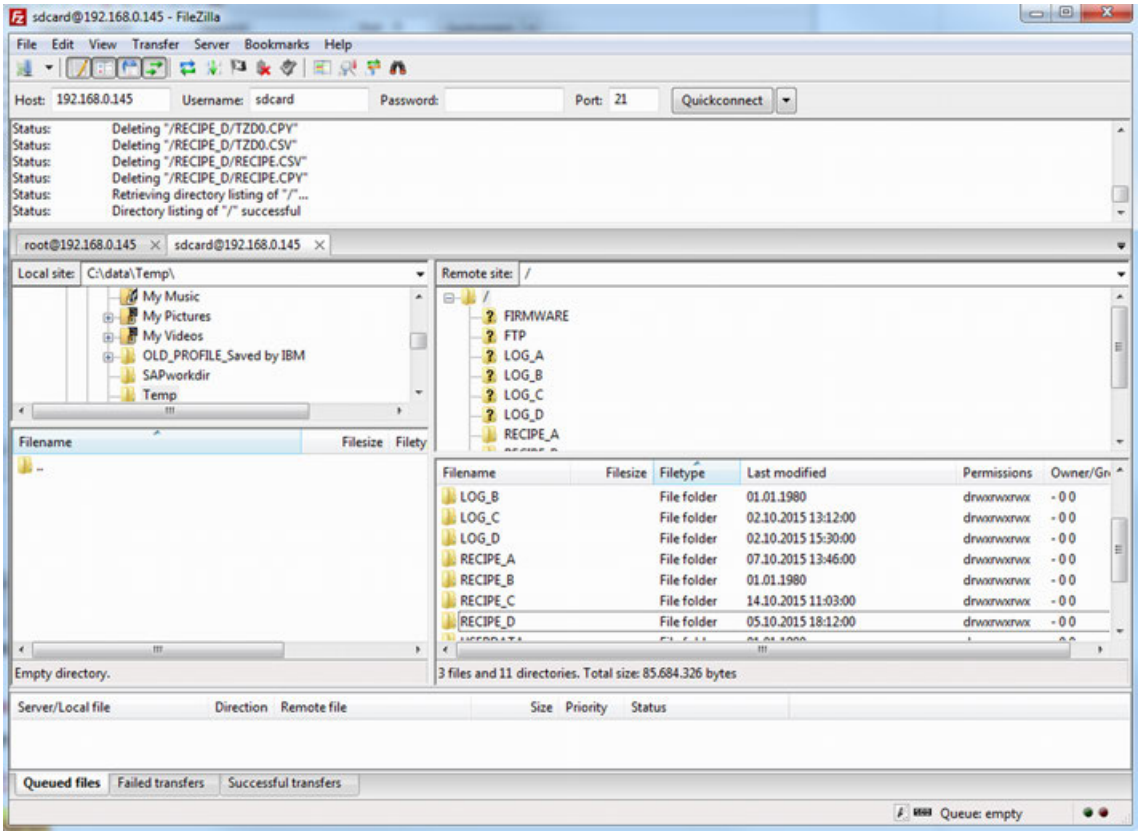


Fig. 585: Access of memory card through FTP client

Enter the PLC IP address as Host, sdcard as Username, password which is configured in the Automation Builder tool and Port number configured (default is 21).

After press “Quickconnect” button, the user can access the memory card data.

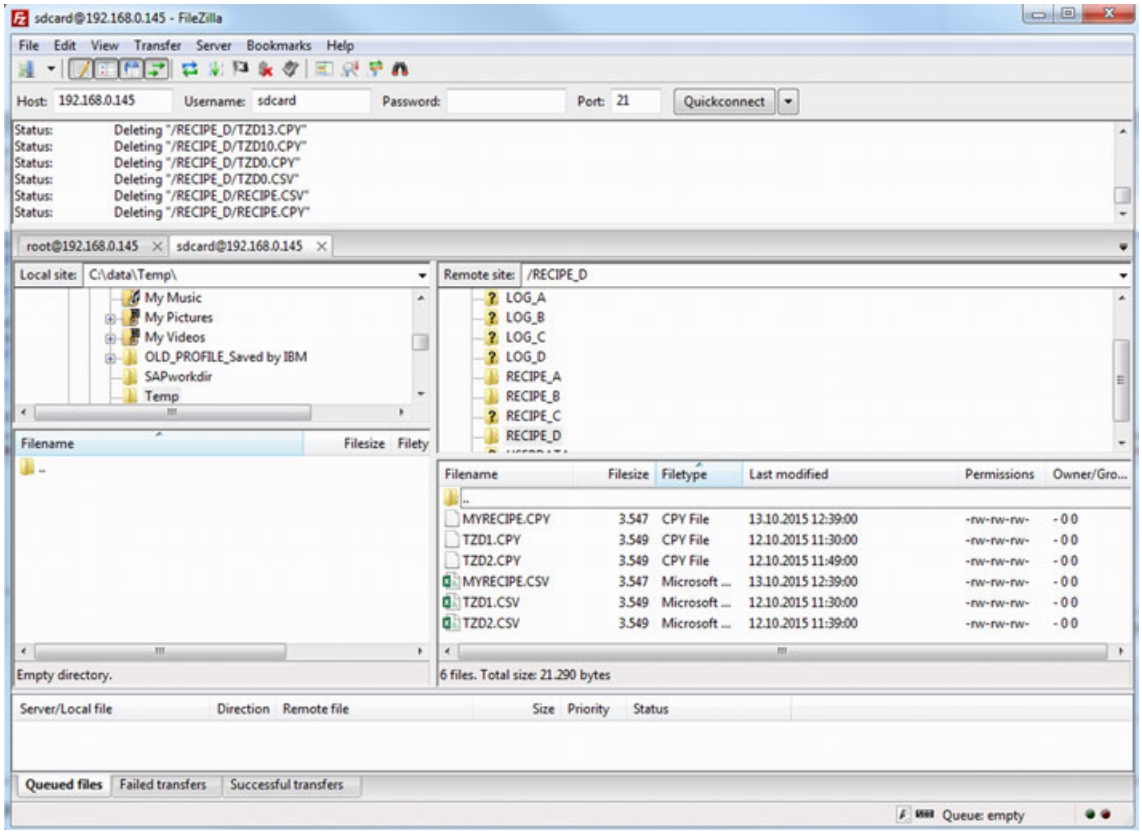


Fig. 586: Recipe files created in memory card accessed using FTP client

When the READ operation is selected, there must be a recipe file present in the Memory card in .CSV format. The name of the file must be as per the Recipe function block guideline. The Recipe block will copy the values of each variable to respective zone data structure. This will help the user to configure the system faster after a restart.

The CSV file is generated, in which all the data is added into a single line for each zone, separated by a semicolon (;) (see figure "Recipe files created in memory card accessed using FTP client"

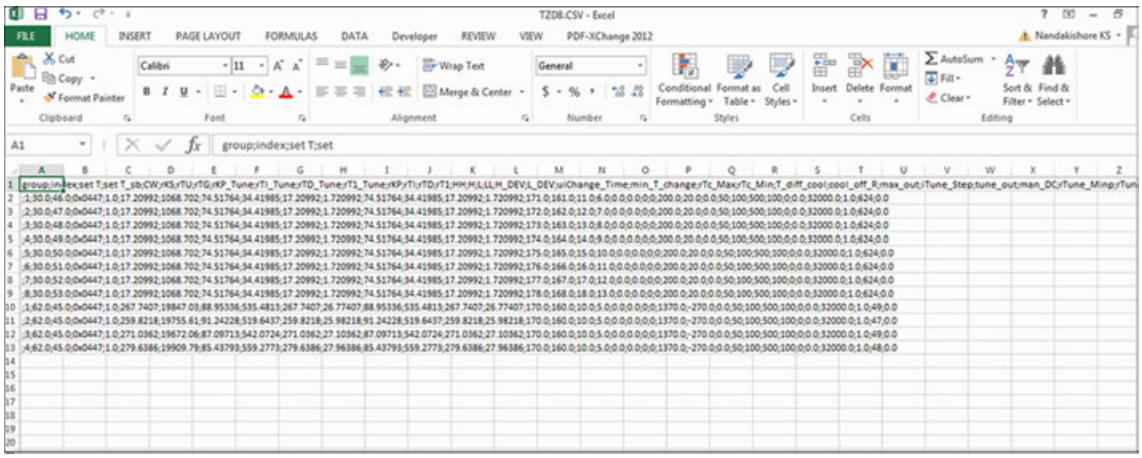
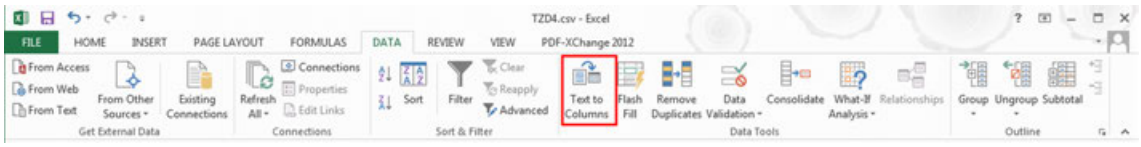
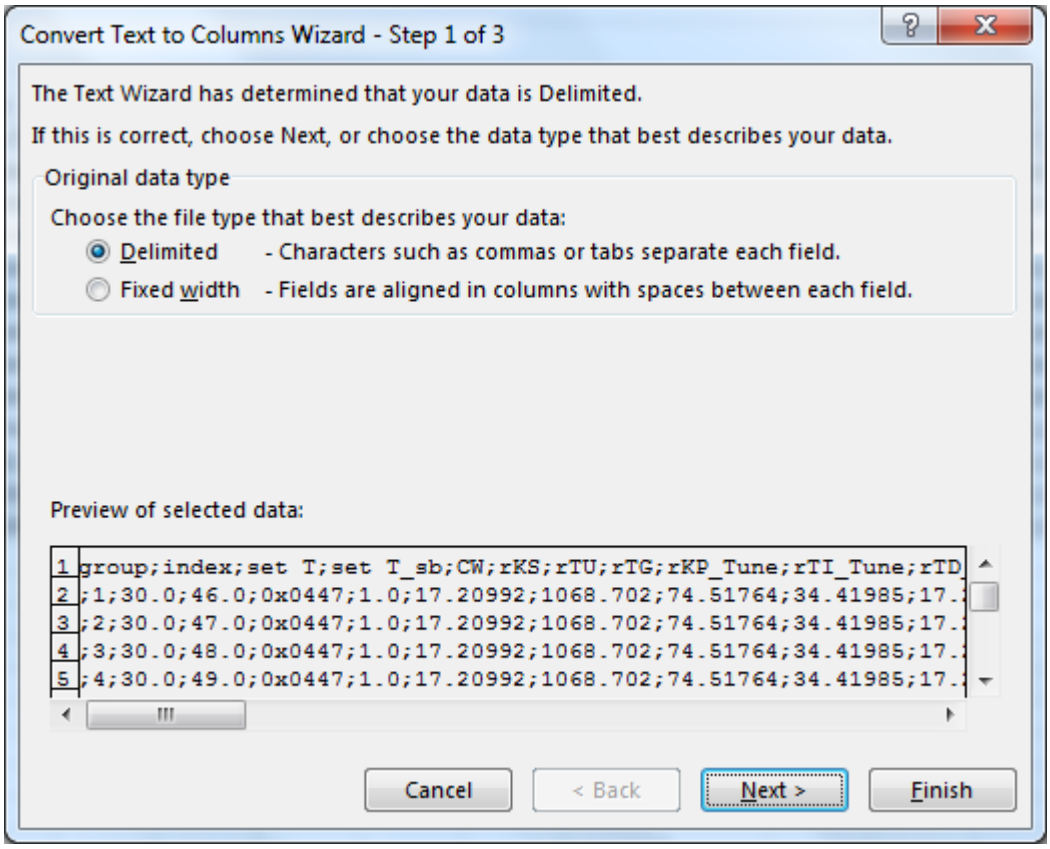


Fig. 587: Open CSV file without delimited

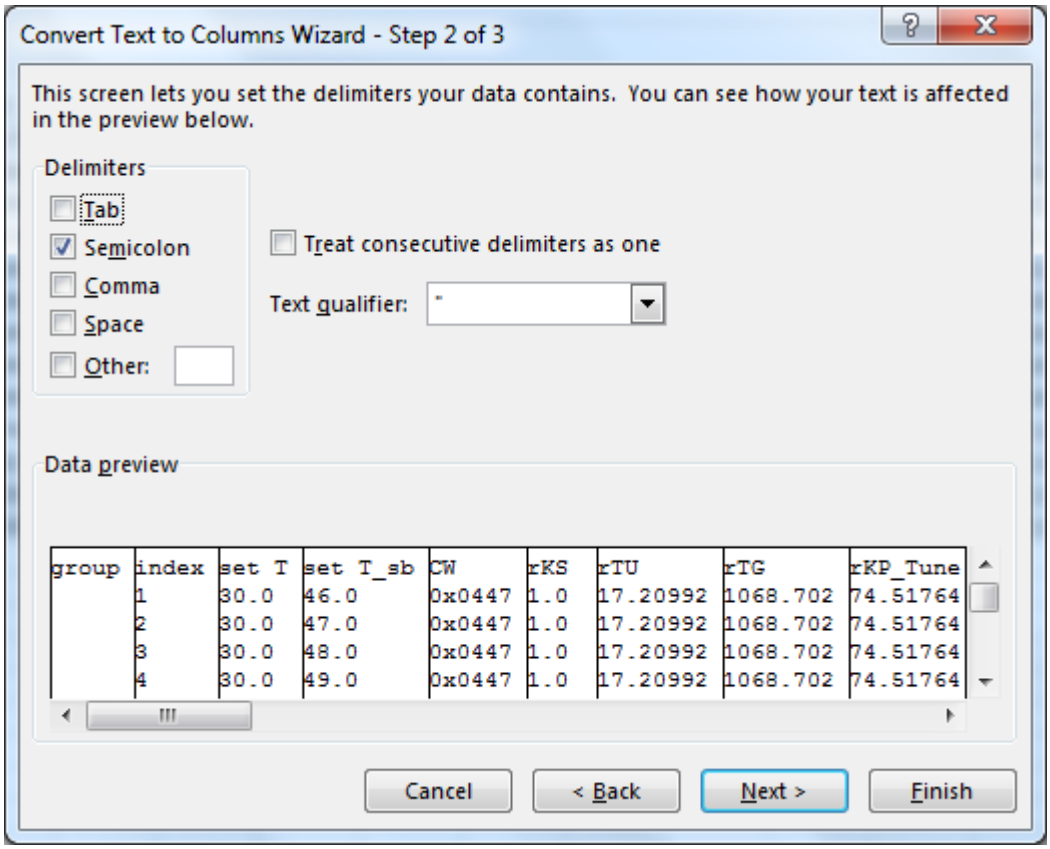
This CSV file can be displayed with separate columns using option “Text to Columns”:



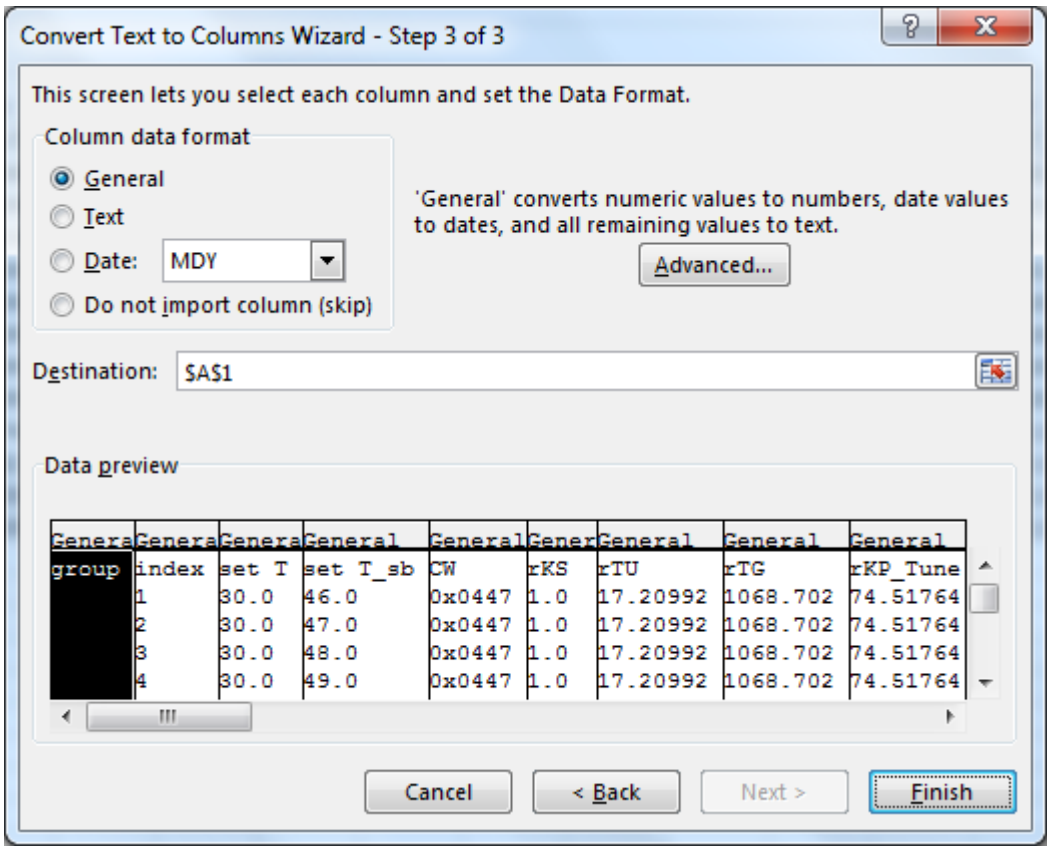
Then follow the following steps:



1. To choose the data type, choose "Delimited- Characters such as commas or tabs separate each field"

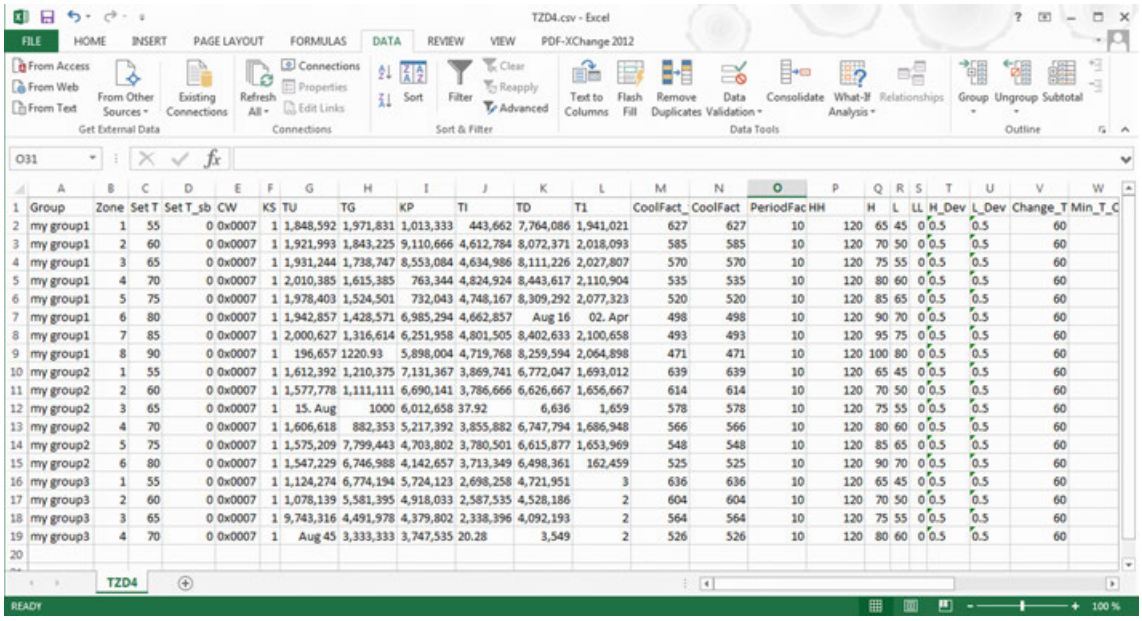


2. To set the delimiters, tick the box "Semicolon".



3. To set the data format, choose "General".

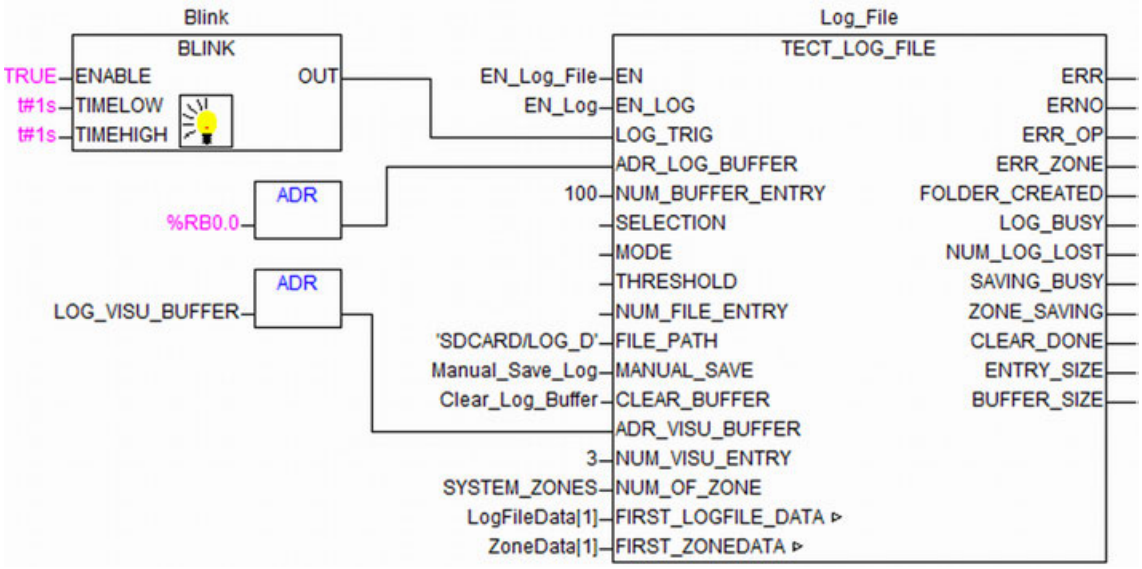
⇨ As a result:



Group	Zone	Set T	Set T_sb	CW	KS	TU	TG	KP	TI	TD	T1	CoolFact_	CoolFact	PeriodFac	HH	H	L	LL	H_Dev	L_Dev	Change_T	Min_T_C
my group1	1	55	0	0x0007	1	1,848,592	1,971,831	1,013,333	443,662	7,764,086	1,941,021	627	627	10	120	65	45	0	0.5	0.5	60	
my group1	2	60	0	0x0007	1	1,921,993	1,843,225	9,110,666	4,612,784	8,072,371	2,018,093	585	585	10	120	70	50	0	0.5	0.5	60	
my group1	3	65	0	0x0007	1	1,931,244	1,738,747	8,553,084	4,634,986	8,111,226	2,027,807	570	570	10	120	75	55	0	0.5	0.5	60	
my group1	4	70	0	0x0007	1	2,010,385	1,615,385	763,344	4,824,924	8,443,617	2,110,904	535	535	10	120	80	60	0	0.5	0.5	60	
my group1	5	75	0	0x0007	1	1,978,403	1,524,501	732,043	4,748,167	8,309,292	2,077,323	520	520	10	120	85	65	0	0.5	0.5	60	
my group1	6	80	0	0x0007	1	1,942,857	1,428,571	6,985,294	4,662,857	Aug 16	02. Apr	498	498	10	120	90	70	0	0.5	0.5	60	
my group1	7	85	0	0x0007	1	2,000,627	1,316,614	6,251,958	4,801,505	8,402,633	2,100,658	493	493	10	120	95	75	0	0.5	0.5	60	
my group1	8	90	0	0x0007	1	196,657	1220.93	5,898,004	4,719,768	8,259,594	2,064,898	471	471	10	120	100	80	0	0.5	0.5	60	
my group2	1	55	0	0x0007	1	1,612,392	1,210,375	7,131,367	3,869,741	6,772,047	1,693,012	639	639	10	120	65	45	0	0.5	0.5	60	
my group2	2	60	0	0x0007	1	1,577,778	1,111,111	6,690,141	3,786,666	6,626,667	1,656,667	614	614	10	120	70	50	0	0.5	0.5	60	
my group2	3	65	0	0x0007	1	15. Aug	1000	6,012,658	37.92	6,636	1,659	578	578	10	120	75	55	0	0.5	0.5	60	
my group2	4	70	0	0x0007	1	1,606,618	882,353	5,217,392	3,855,882	6,747,794	1,686,948	566	566	10	120	80	60	0	0.5	0.5	60	
my group2	5	75	0	0x0007	1	1,575,209	7,799,443	4,703,802	3,780,501	6,615,877	1,653,969	548	548	10	120	85	65	0	0.5	0.5	60	
my group2	6	80	0	0x0007	1	1,547,229	6,746,988	4,142,657	3,713,349	6,498,361	162,459	525	525	10	120	90	70	0	0.5	0.5	60	
my group3	1	55	0	0x0007	1	1,124,274	6,774,194	5,724,123	2,698,258	4,721,951	3	636	636	10	120	65	45	0	0.5	0.5	60	
my group3	2	60	0	0x0007	1	1,078,139	5,581,395	4,918,033	2,587,535	4,528,186	2	604	604	10	120	70	50	0	0.5	0.5	60	
my group3	3	65	0	0x0007	1	9,743,316	4,491,978	4,379,802	2,338,396	4,092,193	2	564	564	10	120	75	55	0	0.5	0.5	60	
my group3	4	70	0	0x0007	1	Aug 45	3,333,333	3,747,535	20.28	3,549	2	526	526	10	120	80	60	0	0.5	0.5	60	

1.5.12.1.8 How to do data logging

Function principle



The TECT_LOG_FILE function block can log the predefined process status and values together with time stamp in a logging buffer as long as the trigger (LOG_TRIG as well as internal trigger if enabled) has a rising edge.

The predefined process status and values logged are:

Control Word, Status Word, SET_TEMP, ACT_TEMP, Duty Cycle, Control State, Output Status, Latest Error and Errors (Error Word).

The function block contains two major functionalities: logging data into a CSV file and live data logging in the visualization.

For the live data logging visualization, all data will be displayed in the visualization as soon as a trigger signal is active.

For logging data into a CSV file, the data based on the mode and selection set by the user will be logged.

There is a logging buffer for the system with a size of NUM_BUFFER_ENTRY (number of entries) for each zone. As long as the logging entry reaches the end of logging buffer, the entries in the logging buffer will be saved into a log file in .csv format under a user defined folder. The file name is generated automatically from the zone index, its group index and the month and date value while saving. This log file can be opened in e.g. Excel with semicolon as delimiter and can be analyzed. When the TECT_LOG_FILE function block is enabled, the user has to configure SELECTION input, where he can select which data is to be logged. There are different modes of logging, which can be selected.

Logging data into a CSV file

There are some steps to start with logging data into a CSV file:

- Define what to be logged:
  Input SELECTION types are: Selection of predefined values to be logged: bit 0 to bit 7 are used for the current version. Bit x=TRUE: the corresponding value will be logged.
  - Bit 0: Control Word
  - Bit 1: Status Word
  - Bit 2: SET_TEMP
  - Bit 3: ACT_TEMP
  - Bit 4: Duty Cycle
  - Bit 5: Control State
  - Bit 6: Output State
  - Bit 7: Latest Error
  - Bit 8: Errors (Error Word)
- Define the log trigger:
  - Define an external trigger with LOG_TRIG, it is recommended to use a timer, e.g. BLINK function block. The LOG_TRIG should be defined greater than 1s, otherwise log data could get lost.
  - Additional to LOG_TRIG, an internal trigger can be activated with bit 4 of input MODE, which creates a trigger if and there is change in Control word or Status word.
- Define when to log data:
  - With bit 1 and bit 0 of input MODE, it can be defined when to log data if log trigger is active. This setting can be used to optimize the storage size for data logging by reducing the number of data sets to be logged. Bit 1, Bit 0:
    00: Log all items as long as log trigger is active.
    01: Log, if one of the log values changes, when trigger is active.
    11: Log, if CW or SW changes or ACT_TEMP changes out of threshold (+/- THRESHOLD), when trigger is active.
  - With above mode 11, the input THRESHOLD needs to be defined also.
- Define the log buffer:
  A log buffer in data area needs to be defined for all zones to collect the log data before data is written to a log file. The log buffer contains continuous area for each zone individually. The first address of the log buffer needs to be assigned to input ADR_LOG_BUFFER. It is recommended to use a data area with fixed address, e.g. %R or %M area.
  The input NUM_BUFFER_ENTRY specifies the size of a log area for each zone in number of log entries. A log entry stands for a data set containing the predefined log values with input SELECTION. The total amount of log entries of the entire log buffer will be NUM_BUFFER_ENTRY x NUM_OF_ZONE.

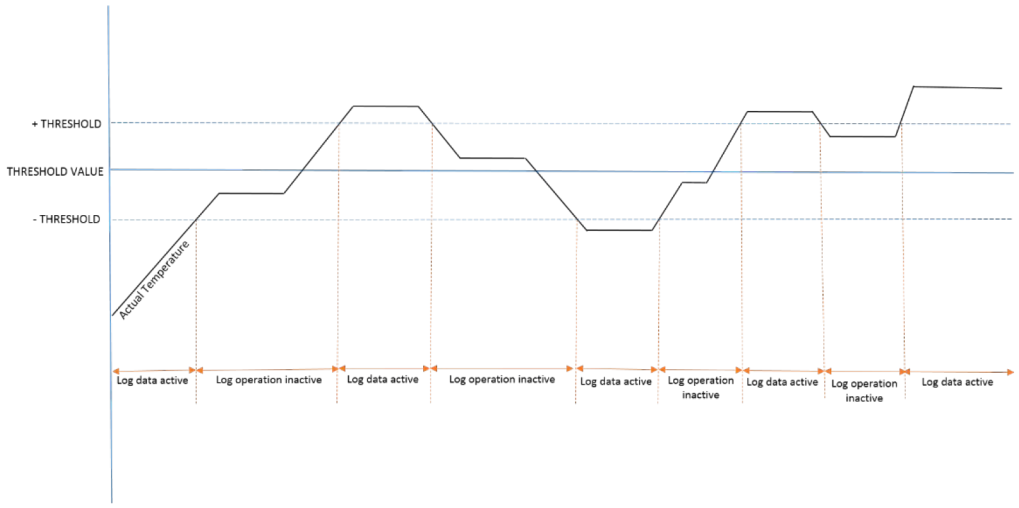
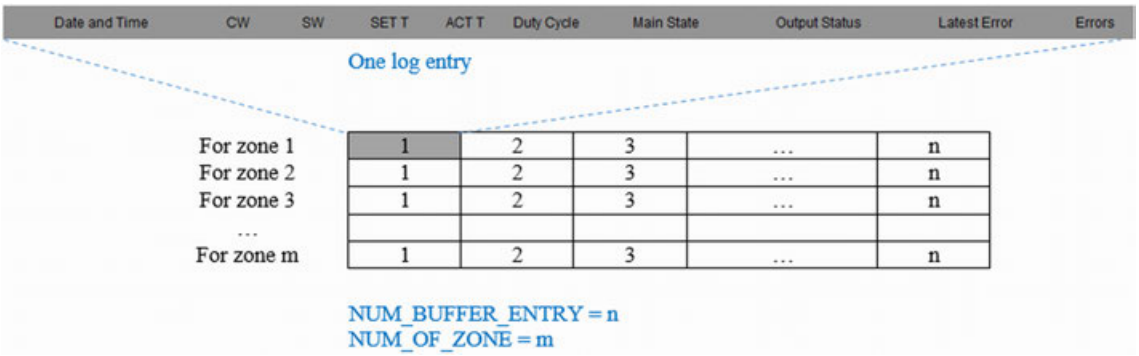


Fig. 588: Input THRESHOLD



The NUM_BUFFER_ENTRY should be defined as large as possible to reduce the frequency of saving the log buffer into log files, thus reducing CPU load. It is recommended to define NUM_BUFFER_ENTRY > NUM_OF_ZONE x 2, otherwise log data could get lost.

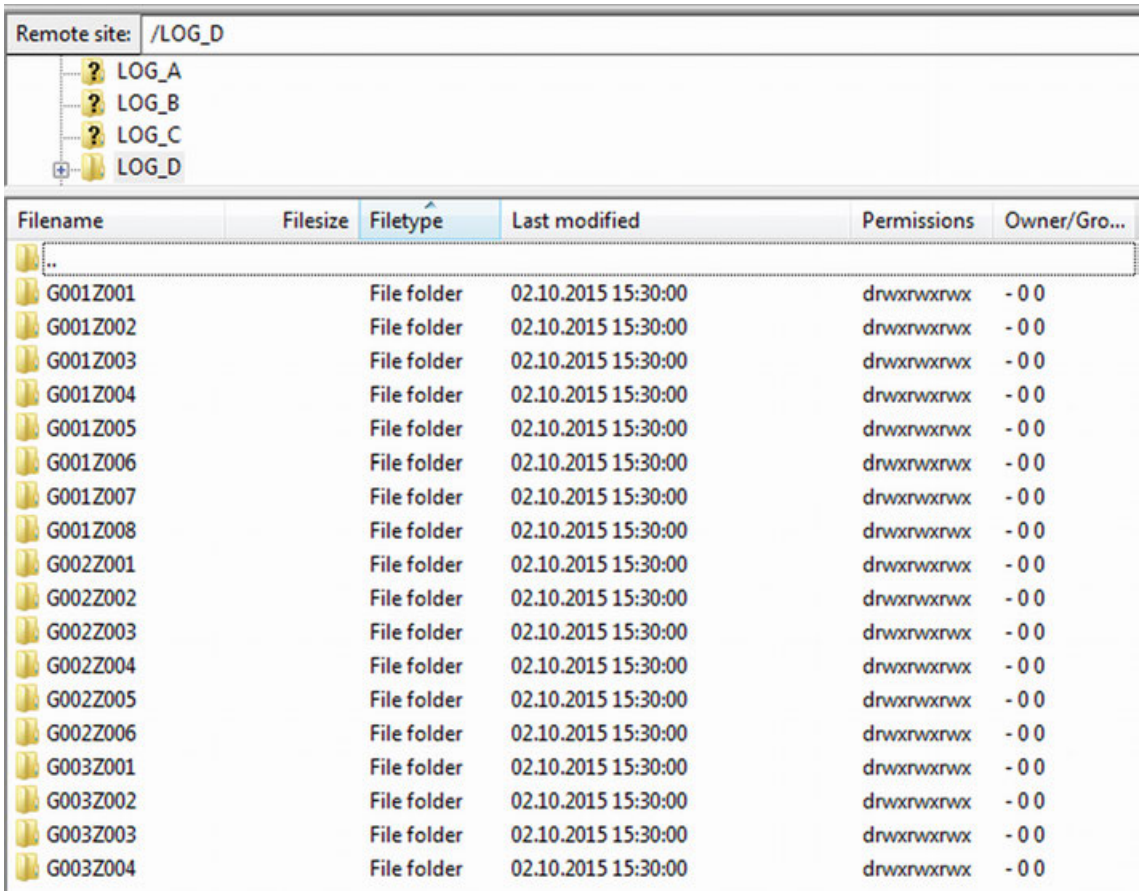
The data size of a log entry depends on the input SELECTION, the size can be about 20 bytes to 100 bytes. This information can be got at run time from the output ENTRY_SIZE and the total size of the log buffer is displayed at output BUFFER_SIZE.

It is very important to check the BUFFER_SIZE output to make sure that enough data area is available for the defined log buffer, otherwise undesired behavior can happen in PLC. The following table shows the available %M and %R area of different CPU types (status of 10.2015).

Memory Area [kB]	PM554	PM564	PM556	PM572	PM573	PM582	PM583	PM590	PM591	PM592	PM595
%M area	2	2	64	4	128	128	128	512	512	512	1024
%R area	1	1	1	4	128	128	128	512	512	512	1024

- Define where to save the log files:
  It is recommended to save log files on memory card. For CPU types PM592 and PM595 it is possible to be saved on the Flashdisk.
  The folder for saving is to be specified at input FILE_PATH, e.g. ‘SDCARD/folder’. The physical disk like SDCARD or FLASHDISK must be specified in the FILE_PATH. The folder following will be created if it does not exist yet as long as the function block is enabled with EN input. But only one folder level will be created, this means the parent folder must already exist.
  Bit 8 and bit 9 of input MODE can be used to specify the subfolder.
  – Bit 8: FALSE: Create subfolder automatically; TRUE: Do not create subfolder.
  – Bit 9: Valid, if bit 8 is FALSE.
    FALSE: subfolder name with index in DECIMAL.
    For example, for group 10, index 3, subfolder name is G010Z003.

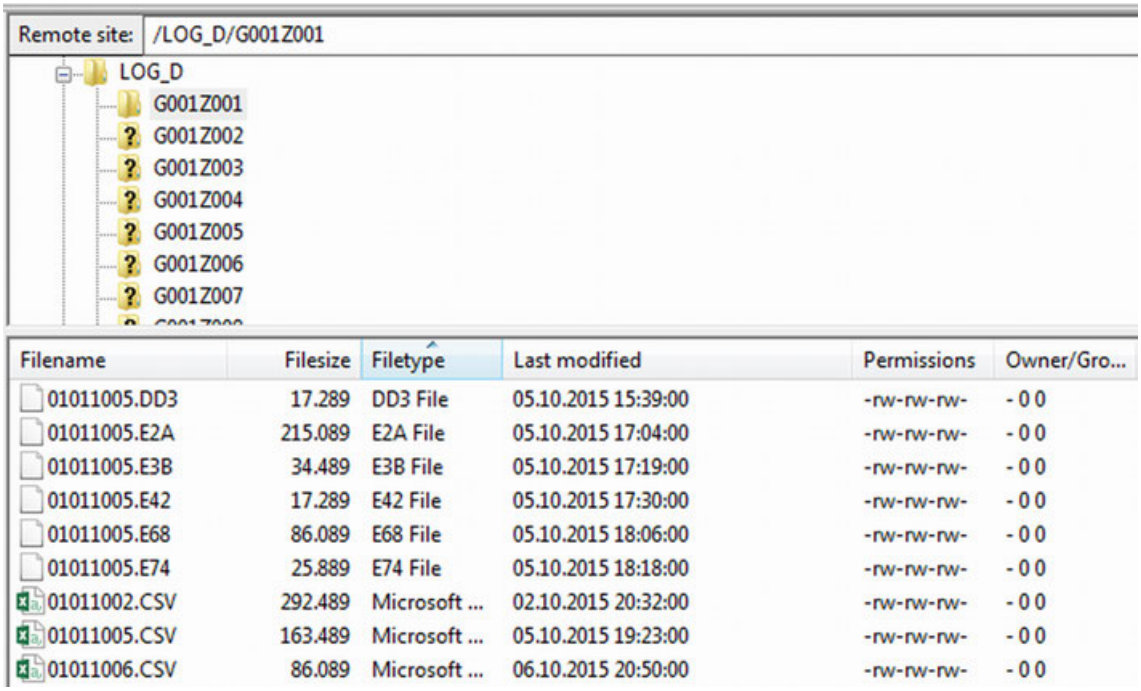
Following is an example of FILE_PATH = ‘SDCARD/LOG_D’ and Bit 8 = FALSE, Bit 9 = FALSE:



Filename	Filesize	Filetype	Last modified	Permissions	Owner/Gro...
..					
G001Z001		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z002		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z003		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z004		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z005		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z006		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z007		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G001Z008		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z001		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z002		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z003		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z004		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z005		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G002Z006		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G003Z001		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G003Z002		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G003Z003		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0
G003Z004		File folder	02.10.2015 15:30:00	drwxrwxrwx	- 0 0

- Define when to create a new log file:
  The function block creates individual log files for each zone.
  As soon as the log area for each zone is full, that area will be written into a CSV file in the predefined folder. The file name contains the group and zone index as well as date information: ggzzmmdd.csv. Where:
  – ‘gg’ for group index in hexadecimal
  – ‘zz’ for zone index in hexadecimal
  – ‘mm’ for month of the year
  – ‘dd’ for day of the month

  At input NUM_FILE_ENTRY is to be defined how many entries a log files can have before creating a new log file. Before creating a new log file, the existing one will be backed up in changing the file extension without changing the name. For example, change ggzzmmdd.csv into ggzzmmdd.E74. The extension is composed of the last three hexadecimal numbers of current time without seconds. In this way, the file overwriting is avoid and all the log history is still available.
  If NUM_FILE_ENTRY is 0, then a new log file will be created daily.
  Following is an example of saved log files:



Filename	Filesize	Filetype	Last modified	Permissions	Owner/Gro...
01011005.DD3	17.289	DD3 File	05.10.2015 15:39:00	-rw-rw-rw-	- 0 0
01011005.E2A	215.089	E2A File	05.10.2015 17:04:00	-rw-rw-rw-	- 0 0
01011005.E3B	34.489	E3B File	05.10.2015 17:19:00	-rw-rw-rw-	- 0 0
01011005.E42	17.289	E42 File	05.10.2015 17:30:00	-rw-rw-rw-	- 0 0
01011005.E68	86.089	E68 File	05.10.2015 18:06:00	-rw-rw-rw-	- 0 0
01011005.E74	25.889	E74 File	05.10.2015 18:18:00	-rw-rw-rw-	- 0 0
01011002.CSV	292.489	Microsoft ...	02.10.2015 20:32:00	-rw-rw-rw-	- 0 0
01011005.CSV	163.489	Microsoft ...	05.10.2015 19:23:00	-rw-rw-rw-	- 0 0
01011006.CSV	86.089	Microsoft ...	06.10.2015 20:50:00	-rw-rw-rw-	- 0 0

- Common inputs:
  NUM_OF_ZONE: Number of zones to be logged into the .csv file or displayed in visualization. If TECT_GROUP is used, then the range is 1…65535. If TECT_GROUP is not used, then the range is 1…255. It is recommended to use one TECT_LOG_FILE for the whole system.
  FIRST_ZONEDATA: First structure of process data and parameters for the zone. See function block description for details.
- Manual operations:
  – Save the log buffer into files manually: with a rising edge at input MANUAL_SAVE all the data in the log buffer will be saved into the log files immediately.
  – Clear log buffer: with a rising edge at input CLEAR_BUFFER all data in the log buffer will be cleared. But before clearing, the new entries in the logging buffer are written into the .csv file first. Then the .csv file is renamed into a backup file with the same file name but different extension, which is composed of the last three hexadecimal numbers of current time without seconds. In this way, even after a clear log buffer action, the history in the buffer will not get lost.
- How to reduce the value of output NUM_LOG_LOST(number of logs which are lost):
  – Don’t create subfolder (MODE.bit8=TRUE) or first enable the function block with EN=TRUE until all folders are created (FOLDER_CREATED=TRUE) then start logging with EN_LOG=TRUE.
  – Increase the interval of LOG_TRIG if possible (recommendation: not less than 2 seconds).
  – Reduce the SELECTION to the necessity and increase the NUM_BUFFER_ENTRY if possible.
  – Increase the size of the second buffer (i2ndNEntry of TECT_LOGFILE_DATA_TYPE). Recommendation: 0…4.

The second buffer will allocate dynamic memory on PLC. Depending on the application, it could cause system error if the second buffer is so large that the system cannot allocate memory any more.

Live data logging in the visualization

All available status in the logger will be displayed with log visualization, thus input SELECTION has no influence on log visualization.

The buffer for visualization is to be defined, which is independent of the log buffer for logging data into CSV files. The first data address of visualization buffer needs to be assigned to input ADR_VISU_BUFFER.

The size of each visualization buffer is specified at input NUM_VISU_ENTRY as number of visualization entries.

The total size of the visualization buffer is depending on how many entries should be displayed in the visualization and how many zones are defined.

For example, if there are 4 zones to be logged and each zone has 10 log visualization entries, then a one dimensional or two dimensional array can be defined as the type “TECT_LOG-INFO_TYPE”:

visu_array : ARRAY[1..40] OF TECT_LOGINFO_TYPE; (in one dimensional array).

visu_array : ARRAY[1..4, 1..10] OF TECT_LOGINFO_TYPE; (in two dimensional array).

In the library, there are two predefined visualization elements:

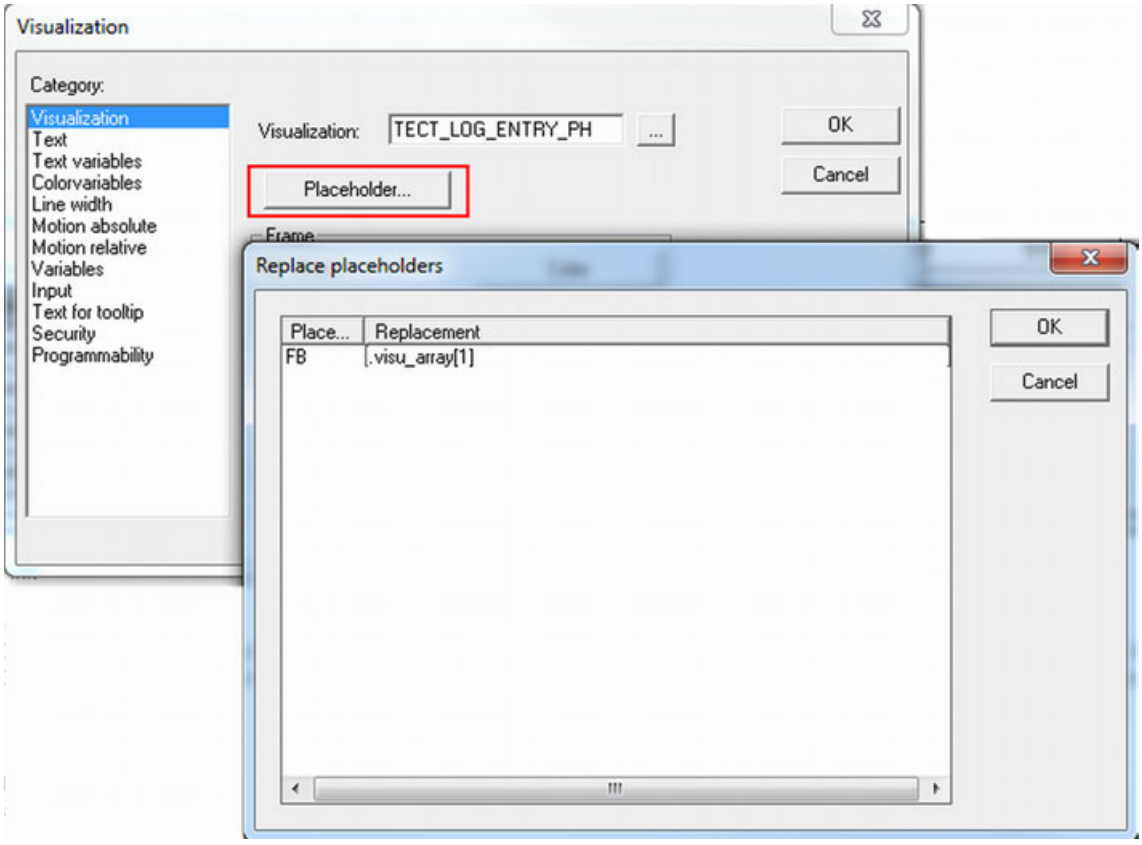
Log header:

Date and Time	CW	SW	SET T	ACT T	Duty Cycle	Main State	Output Status	Latest Error	Errors

Log entry:

%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s	0x%x

Each entry needs to be attach to one element of the defined visu_array[i] as place holder.



Then the user can build his own log visualization according to his demand. For example one zone with 5 log visualization entries:

Date and Time	CW	SW	SET T	ACT T	Duty Cycle	Main State	Output Status	Latest Error	Errors
DT#2015-10-22-15:46:23	0x6	0x4001	0	20	0	TECT_Ready	TECT_Disabled	TECT_LowTemp	0x100
DT#2015-10-22-15:46:21	0x6	0x4001	0	20	0	TECT_Ready	TECT_Disabled	TECT_LowTemp	0x100
DT#2015-10-22-15:46:19	0x6	0x4001	0	20	0	TECT_Ready	TECT_Disabled	TECT_LowTemp	0x100
DT#2015-10-22-15:46:17	0x6	0x4001	0	20	0	TECT_Ready	TECT_Disabled	TECT_LowTemp	0x100
DT#2015-10-22-15:46:15	0x6	0x4001	0	20	0	TECT_Ready	TECT_Disabled	TECT_LowTemp	0x100

Another variable FIRST_LOGFILE_DATA needs to be defined as an array of type “TECT_LOG-FILE_DATA_TYPE” in size NUM_OF_ZONE. This variable contains internal log settings and values for each zone individually.

Work with log file

The log files can be downloaded using a FTP client, e.g. FileZilla. For detailed operation of FTP please refer to the chapter for recipe functionality.

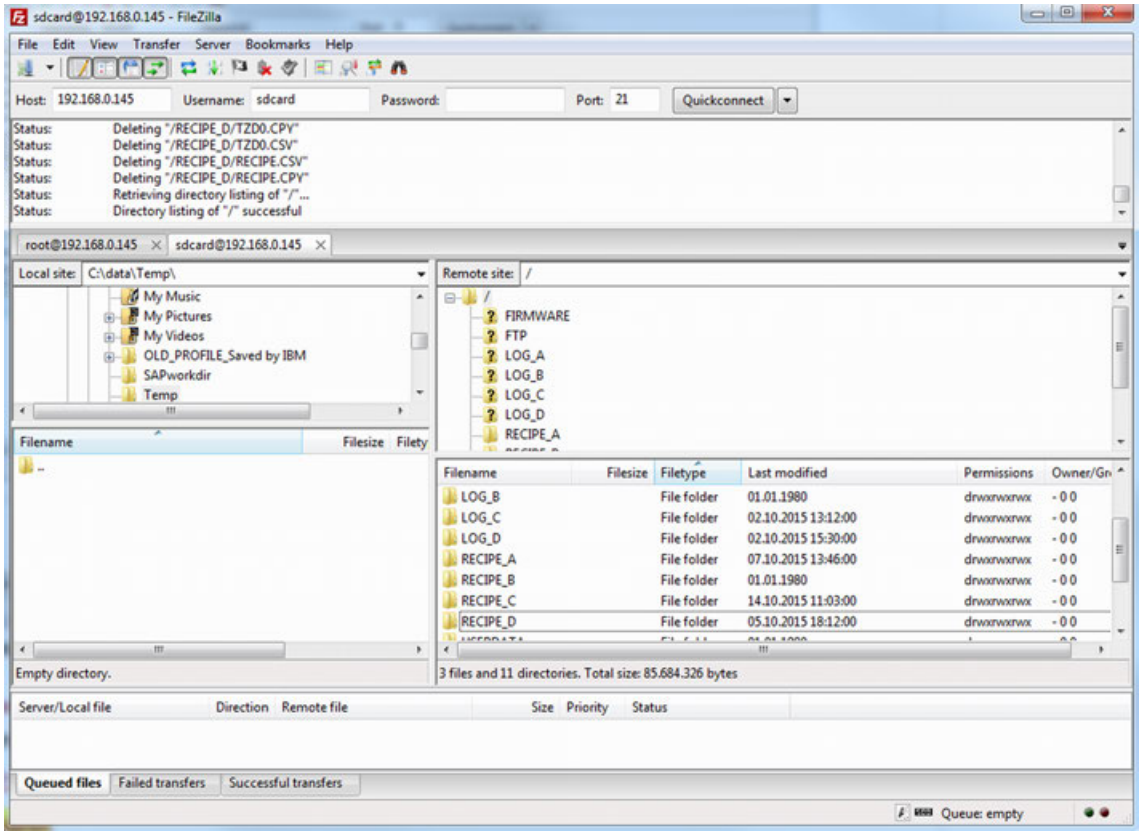


Fig. 589: Log files created in memory card accessed using FTP client

After a log file downloaded to PC, it can be opened with Excel in separate columns (delimited by semicolon) ↬ *Chapter 1.5.12.1.7.2 “Save set values as recipe file” on page 3289*.

For the backup files (files without .CSV extension), they can be renamed into CSV file and be opened with Excel. For example, can be 01011005.E42 renamed into 01011005.E42.CSV (keeping the original extension to avoid duplicated file name).

	A	B	C	D	E	F	G	H	I	J
1	Date and Time	CW	SW	SET T	ACT T	Duty Cycle	Main State	Output Status	Latest Error	Errors
2	2015-10-23-09:22:37	0x0006	0x4001	0	20	0	Ready	Disabled	LTemp	0x0100
3	2015-10-23-09:22:39	0x0006	0x4001	0	20	0	Ready	Disabled	LTemp	0x0100
4	2015-10-23-09:22:41	0x0006	0x4001	0	20	0	Ready	Disabled	LTemp	0x0100
5	2015-10-23-09:22:43	0x0006	0x4001	0	20	0	Ready	Disabled	LTemp	0x0100
6	2015-10-23-09:22:45	0x0066	0x4001	0	20	0	Ready	Disabled	LTemp	0x0100
7	2015-10-23-09:22:47	0x0067	0x4003	50	20	100	Tune	Tuning	LTemp	0x0100
8	2015-10-23-09:22:49	0x0067	0x4003	50	20	100	Tune	Tuning	LTemp	0x0100
9	2015-10-23-09:22:51	0x0067	0x4003	50	20.1	100	Tune	Tuning	LTemp	0x0100
10	2015-10-23-09:22:53	0x0067	0x4003	50	20.1	100	Tune	Tuning	LTemp	0x0100
11	2015-10-23-09:22:55	0x0067	0x4003	50	20.2	100	Tune	Tuning	LTemp	0x0100
12	2015-10-23-09:22:57	0x0067	0x4003	50	20.3	100	Tune	Tuning	LTemp	0x0100
13	2015-10-23-09:22:59	0x0067	0x4003	50	20.5	100	Tune	Tuning	LTemp	0x0100
14	2015-10-23-09:23:01	0x0067	0x4003	50	20.6	100	Tune	Tuning	LTemp	0x0100
15	2015-10-23-09:23:03	0x0067	0x4003	50	20.8	100	Tune	Tuning	LTemp	0x0100
16	2015-10-23-09:23:05	0x0067	0x4003	50	20.9	100	Tune	Tuning	LTemp	0x0100
17	2015-10-23-09:23:07	0x0067	0x4003	50	21.1	100	Tune	Tuning	LTemp	0x0100

Fig. 590: Log file example opened in Excel

The log data can be analyzed more easily with Excel, e.g. using line chart.

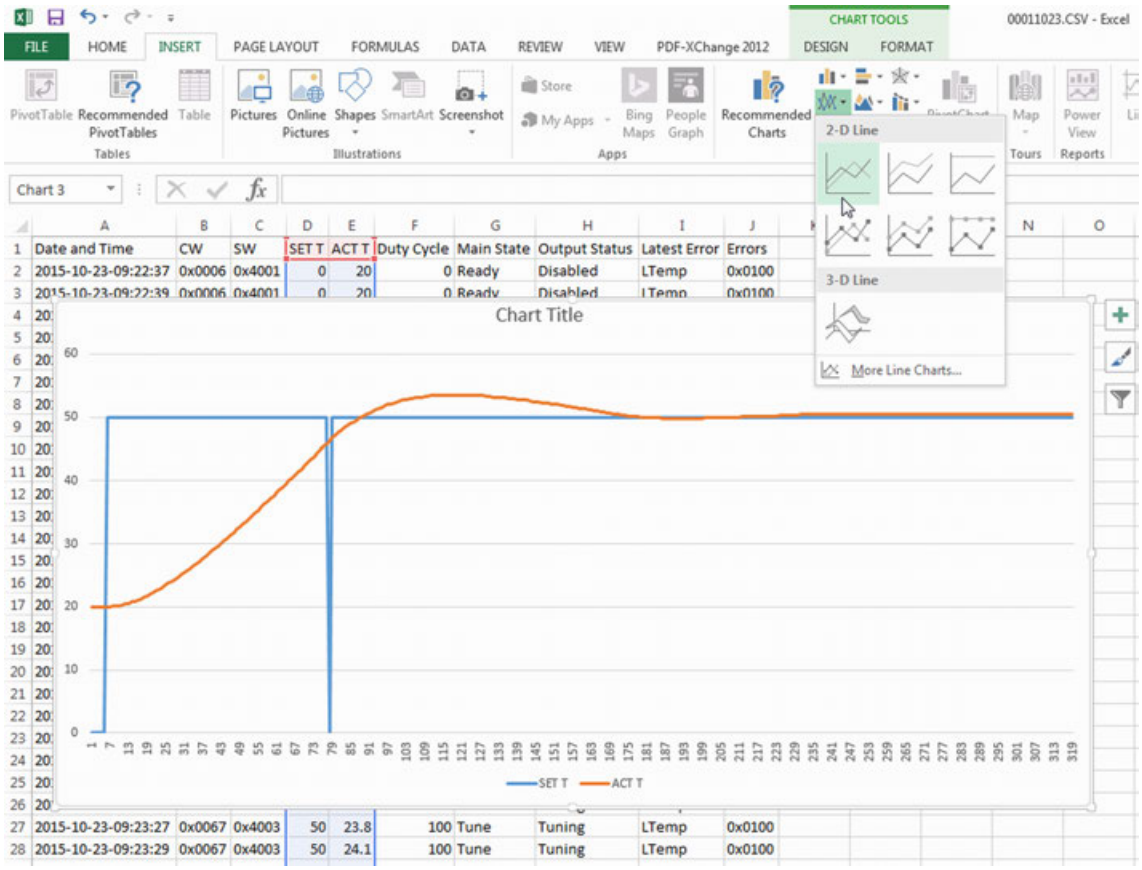


Fig. 591: Log file example opened in Excel with line chart

After editing the log file, it can be saved in .xlsx format to keep the line chart.

1.5.12.1.9 Current monitoring

Current monitoring feature is available with TECT_GROUP function block. Current can be measured using common current sensor for all zones in the group or individual current sensor for each zone. Single or three phase heaters system can be selected by user.

Industry uses resistive heaters for acquiring desired process temperature. Resistive heaters either become open or short circuited due to overheating and age over prolonged use. These conditions can be detected and respective error message generated using current sensors. Current sensors measures circuit current and sends it to AC500, which in turn compares it with rated current of heaters. On faulty condition one of the following error generated for the zone.

Open circuit

Most of the time heater coil breaks due to continuous / overheating and causes open circuit. In open circuit condition current will not flow. Hence we can safely assume a condition of open circuit, if current is measurement is zero.

Short circuit

If heater coil comes in contact with its metal container, then it causes short circuit. In short circuit condition, heater will try to draw max current capacity of the input supply. E.g. it will draw a max current of the SSR / MCB.

Control circuit failure

If there is no supply at the input side of SSR and still we are able to measure current at the output side, then it means there some problem in the control circuit.

Single or common current sensors

Advantage of common current sensor: is less costs, as single current sensor is used for all zones in the group. It also leads to less maintenance cost.

Disadvantage of common current sensor is slower recognition of faults, as single current sensor is used for all zones in the group. Single current sensor will measure current for one zone at a time. Once measurement is done sensor measure current for next zone.

Example: Circuit diagrams for individual and common current sensors

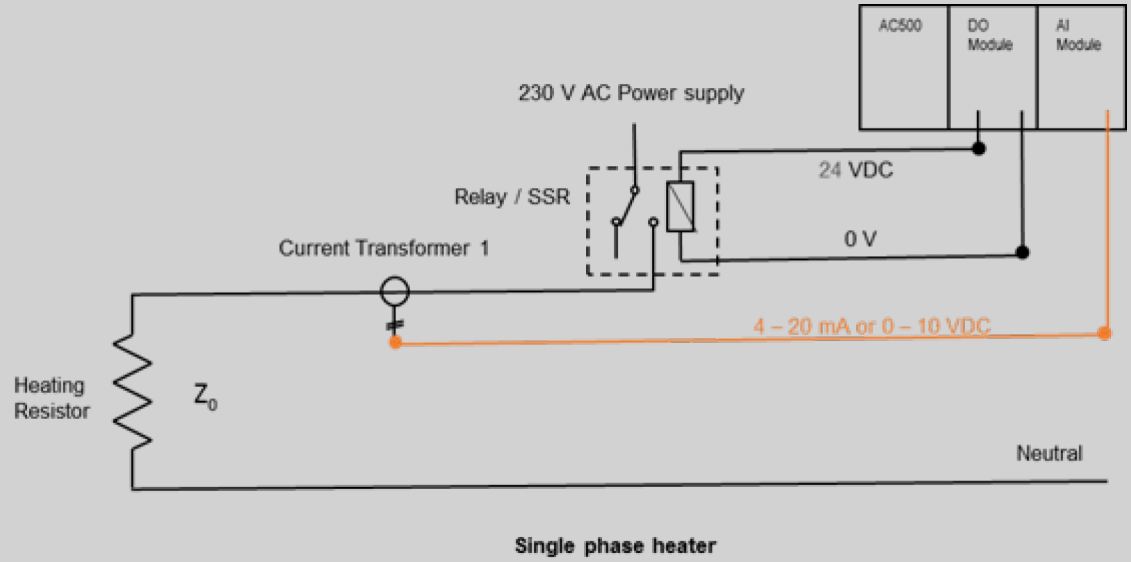


Fig. 592: Individual current sensor for single phase heater

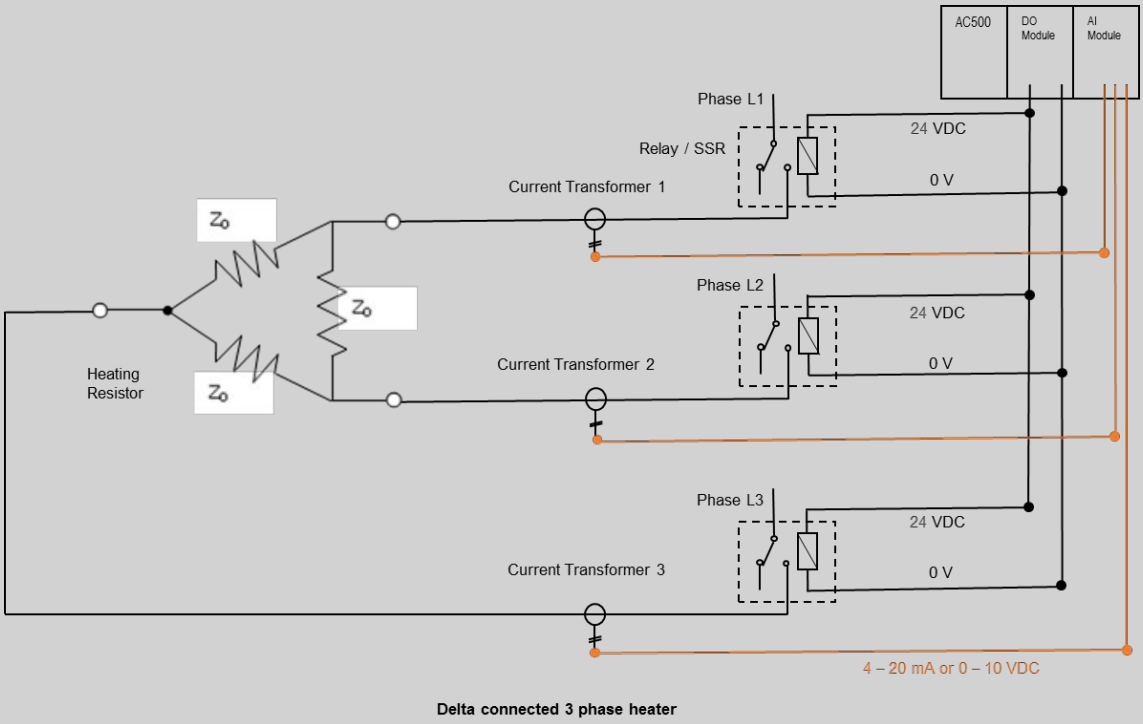


Fig. 593: Individual current sensor for Delta connected three phase heater

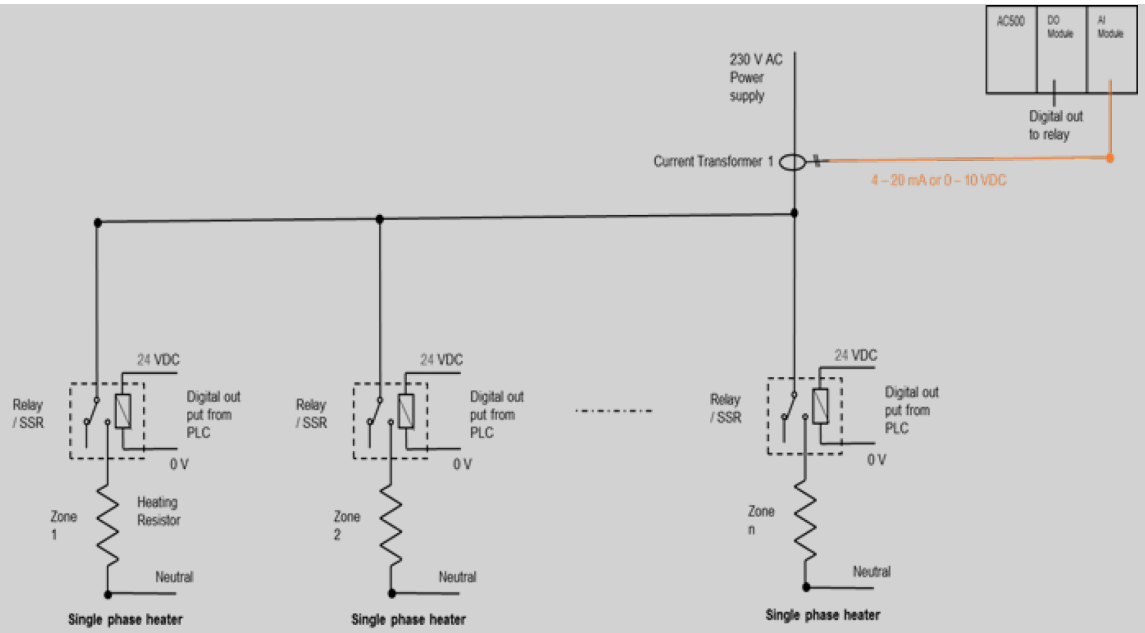


Fig. 594: Common current sensor for single phase heaters

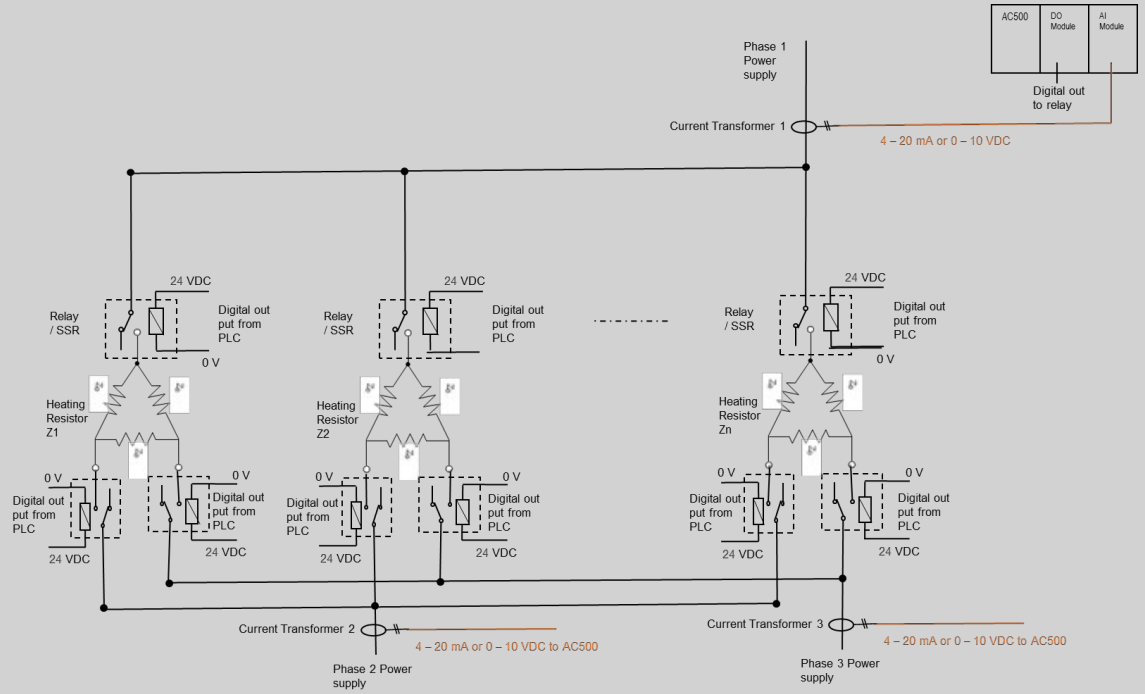


Fig. 595: Common current sensor for Delta connected three phase heater

Individual current sensor monitoring

Individual current sensor is connected to each zone. If three phase zone is used then three current sensors are required for zone, one for each phase. User needs to define following variables for individual current monitoring.

Selecting single or three phase current monitoring:

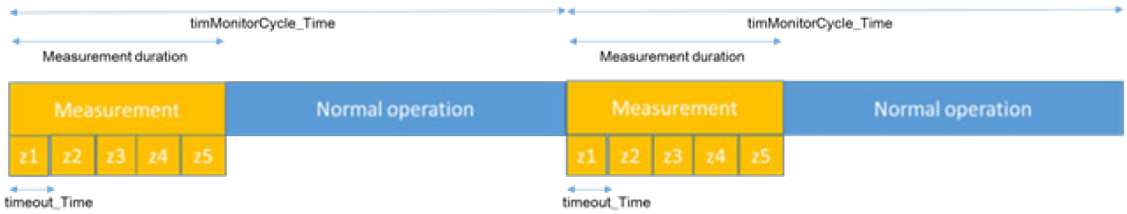
System will understand whether single or three phase heaters are used using “TECT_PROCESS_SET_TYPE“ structure “wControlWord“ variable bit 14. If this bit is turned ON then it means three phase current monitoring is selected else single phase current monitoring is selected.

xEn_Indivi_Monitor: User must set “xEn_Indivi_Monitor“ variable TRUE in the group data to turn on current monitoring feature for individual current sensor.

rRated_Current: User should use this variable from “TECT_MACHINE_SET_TYPE “ structure for connecting rated current of the heater. For 3 phase heaters same rating is used for balancing. Hence only one varaible for rated current is sufficient for all three phases.

arActual_Current: User should use this variable from “TECT_MACHINE_STATUS_TYPE“ structure for connecting actual current measured from the sensor. If single phase heater is used then only first array element should be connected. For 3 phase heaters sensor input needs to be connected to 3 array elements respectively.

timMonitorCycle_Time: Monitor cycle can be defined in seconds by user in “ TECT_GROUP_DATA_TYPE“ structure variable. At the start of this duration Zones enter in current monitoring function. One after another zones are checked until all zones are checked. After this monitoring function is stopped, and control of the zones goes back to process till “timMonitorCycle_Time“ elapsed. On next cycle function will start monitoring from zone 1 again.



It is recommended that user should set a value around 150-300 seconds.

Each zone takes approx. 5 seconds time for current monitoring. Each monitoring cycle has 2 parts: current monitoring and process. It is recommended that process time should be at least twice the current monitoring. As a thumb rule we can calculate minimum monitoring time as timMonitorCycle_Time := Number of zones * 5 * 3

wErrors2: This variable in TECT_PROCESS_STATUS_TYPE structure shows current monitoring status of the zone. If error is condition is removed, then error will be removed and it does not require RESET.

timLastMeasurement: This variable in “TECT_GROUP_DATA_TYPE“ structure shows time taken by all the zones to complete monitoring function in the last measurement.

Common current sensor monitoring

Common current sensor is used for all the zones in the group. If three phase zones are used then three current sensors are required for each phase. User needs to define following variables for common sensor current monitoring.

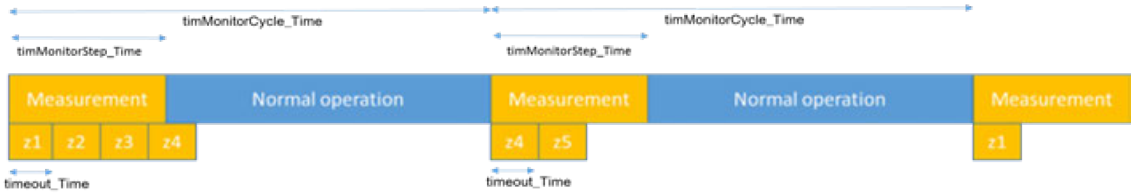
Selecting single or three phase current monitoring: System will understand whether single or three phase heaters are used using “TECT_PROCESS_SET_TYPE“ Structure “wControlWord“ variable bit 14. If this bit is turned on then it means three phase current monitoring is selected else single phase current monitoring is selected.

xEn_Co_Monitor: User can must set “xEn_Co_Monitor“ variable TRUE in the group data to turn on current monitoring feature for common current sensor.

rRated_Current: User should use this variable from “TECT_MACHINE_SET_TYPE “ structure for connecting rated current of the heater. For 3 phase heaters same rating is used for balancing. Hence only one variable for rated current is sufficient for all three phases.

arActual_Current: User should use this variable from “ TECT_GROUP_DATA_TYPE“ structure for connecting actual current measured from the sensor. If single phase heater is used then only first array element should be connected. For 3 phase heaters sensor input needs to be connected to 3 array elements respectively.

timMonitorCycle_Time: Monitor cycle can be defined in seconds by user in “ TECT_GROUP_DATA_TYPE“ structure varaiblevariable. At the start of this duration Zones enter in current monitoring function. One after another zones are checked untilluntil “timMonitorStep_Time“ is elapsed. After this monitoring function is stopped, zone number is stored internally and control of the zones goes back to process till “timMonitorCycle_Time“ elapsed. On next cycle monitoring function starts from internally stored zone number. Once all zones are checked, function will start monitoring from zone 1 again.



It is recommended that user should set a value around 150-300 seconds.

timMonitorStep_Time: Monitor cycle can be defined in seconds by user in “ TECT_GROUP_DATA_TYPE“ structure variable. In the monitoring cycle when this time is elapsed current monitoring function stops and control of the zones goes back to process.

wErrors2: This variable in TECT_PROCESS_STATUS_TYPE structure shows current monitoring status of the zone. If error is condition is removed, then error will be removed and it does not require RESET.

timLastMeasurement This variable in “TECT_GROUP_DATA_TYPE“ structure shows time taken by all the zones to complete monitoring function in the last measurement.

1.5.12.1.10 Explanation of zone data structure

Overview

Zone data values have been grouped into machine and process set, machine and process status, and internal status. This chapter contains details for the initial configurationset values of the Temperature Control system.

Set values / parameters

Process set values / parameters

TECT_PROCESS_SET_TYPE structure contains all the process related parameters.

This structure contains process set parameters of the zone data. User can define all parameters which are related to the process, i.e. parameters which are set as per process properties. These parameters will be changed continuously as per process requirement, e.g. in structure Control Word.

PID Set Point, Control Word, High Temp, Low Temp settings comes under this structure.

Control Word (wControlWord)

The Control Word setting will define the working of the machine / process. The user needs to set the Control Word bits per requirements .These settings will decide the working philosophy of the machine. The user also needs to decide which parameters he want to set permanently and which parameter he wants to change frequently.

E.g. Heat_en and Cool_en can be set permanently and Enable can be used to turn on and off the temperature zone.

- Enable (Bit 0):
  When this is TRUE, it will enable the process of temperature control. If this bit is FALSE, the complete process will stop and wait for next on command in Ready State.
- Heat_en (Bit 1):
  It will enable the zone as heat zone and the heating process starts when the duty cycle is greater than 0.
- Cool_en (Bit 2):
  It will enable the zone as a cooling zone and the cooling process starts when duty cycle is greater than 0.
- Manual_en (Bit 3):
  This is the highest priority mode. When this is TRUE, all the other modes are disabled and the manual control becomes active.
- Standby_en (Bit 4):
  When this bit is TRUE, the Standby set point becomes an active set point for the process.
- Tune_en (Bit 5):
  It will enable the auto tune process for the system using a tune step. It will start the tuning of the zone and calculate the optimal KP, TI, TD and T1 values.
- Accept_auto_tune (Bit 6):
  If the TRUE system will accept the calculated auto tune parameters.
- Co_Output (Bit 7):
  When this bit is TRUE, it will consider coordinated output for the particular group. It avoids switching on all zones simultaneously, thus avoiding a voltage drop in the power supply system. At the same time, it compensates the error between digital outputs and duty cycle due to CPU task cycle.
- Control Optimization (Bit 10, Bit 9):
  A combination of these two bits decide the working principle for the PID control.
  – 00 – APERIODIC:
  – 01 – OVERSHOOT:
  – 10 – DEADTIME COMPENSATION:
  – 11 – Default: APERIODIC
- Warm_Reset (Bit 13):
  When this bit is TRUE, it will reset the all alarms and process in the zone, but it will retain the auto tune status.
- En_3Phase_Monitor (Bit 14):
  When this bit is true, 3 phase current monitoring is enabled. If false, then single phase current monitoring is enabled.
- Cold_Reset (Bit 15):
  When this bit is TRUE, it will reset everything in the control process and the Status Word of the process will become zero. Auto tune has be carried out once again for the zone, as auto tune values have been reseted by cold reset.

PWM output in Manual mode. The Manual duty cycle needs to be set by the user when the manual mode (i.e. control word bit 3) is enabled for the process.

In the Manual mode the Manual duty cycle is passed on as a process duty cycle. That means the process duty cycle is not dependent on the set point and actual temperature. If not monitored properly, this can lead to accident.

Set Point (rSet-Point):

This is a secondary process set point which has to be set by the user. This value is considered for the process when Standby_en (control word bit 4) is TRUE.

Standby set point: (rStandBy_Set-Point): This is a secondary process set point which has to be set by the user, this value is considered for the process when Standby_en (control word bit 4) is TRUE

High Temperature (rHigh_Temp): The High Temperature value has to be set by the user as per process requirement. If the actual temperature becomes more than a High temperature, then the heat output is disabled and an alarm is generated.

Low Temperature (rLow_Temp): The Low Temperature value has to be set by the user as per process requirement. If the actual temperature becomes less than Low temperature, cool output is disabled and an alarm is generated.

High Deviation (rHigh_Deviation): If a difference between the actual value and the set value is more than the High Deviation, then an alarm is generated.

Low Deviation (rLow_Deviation): If a difference between the actual value and the set value is less than Low Deviation, then an alarm is generated.

Machine set values / parameters

TECT_MACHINE_SET_TYPE structure contains all the process related parameters.

This structure contains machine set parameters of the zone data. The user can define all parameters which are related to the machine, i.e. parameters which are set as per machine properties and will be defined only once at the time of commissioning. These parameters need to be changed, only when some machine property has changed, e.g. the temperature sensor has changed.

PID parameter, HighHigh, LowLow temp, and temperature sensors settings come under this structure.

Cool Factor (wCoolFact): Cool factor in percentage.

KP (rKP): PID parameter for the machine, P in Real value.

TI (rTI): PID parameter for control: I in seconds.

TD (rTD): PID parameter for control: Filter for D in seconds.

HighHigh Temperature (rHigh-High_Temp): This is the highest value of temperature the machine can reach and beyond that it could damage the system / machine. When this value is reached HighHigh Temperature fault will be generated, the control state will be in Fault state and all the outputs are disabled.

LowLow Temperature (rLowLow_Temp): This is the lowest value of temperature the machine can reach and below that value it can damage the system / the machine. When this value is reached, LowLow Temperature error is generated, the control state will be in Fault state and all the outputs are disabled.

Change time (uiChange_Time): For the TC_FAULT_2 the time (in seconds) is taken to register the Minimum temperature change when the duty cycle is greater than or equal to tune output value.

Minimum Temperature Change (uiMin_Temp_Change):	For the TC_FAULT_2 the Minimum Temperature Change, that can happen in the change time, is assigned when the duty cycle is greater than or equal to tune output value.
TC_Max (rTc_Max):	It corresponds to the maximum measurement range of the temperature sensor. When the temperature of the zone reaches this value the fault TC_FAULT_1 is generated.
TC_Min (rTc_Min):	It corresponds to the minimum measurement range of the temperature sensor. When the temperature of the zone reaches this value the fault TC_FAULT_1 is generated.
Temperature Difference Cool (rTemp_Diff_Cool):	If value > 0: cooling will be ON with 100% when ACT_TEMP > set point + rtemp_diff_cool, and will be OFF when ACT_TEMP < set point + rtemp_diff_cool x iCool_Off_Ratio. If value = 0: rtemp_diff_cool is deactivated, PID controller will be active for cooling.
Cool Off Ratio (iCool_Off_Ratio):	To be used with rTemp_Diff_Cool. ↳ *"Temperature Difference Cool (rTemp_Diff_Cool):" on page 3309*
Maximum Output (iMax_Output):	This input defines the maximum value of the duty cycle for the respective zone.
Tune Setpoint (iTune_Step):	Necessary temperature change in raw sensor value for auto tune process.
Tune Output (iTune_Output):	PWM duty cycle which is used during tuning.
Tune Minimum P (rTune_MinP):	Minimum allowable KP value for the auto tune to accept.
Tune Maximum P (rTune_MaxP)	Maximum allowable KP value for the auto tune to accept.
Tune P Multiplier (rTune_PMult):	Scaling for calculating KP value after tuning.
TC Offset (rTc_Offset):	Compensation for the thermocouple in degree.
Period Factor (rPeriod_Fact):	Period factor for cooling against heating.
Minimum Cool on Time (timMin_Cool_On):	Minimum time for which xDO_Cool output is ON.
Feed Forward Time Ratio (iFF_Time_Ratio):	Feed forward time ratio in percentage for internal set point ramp generator. The smaller the value, the steeper is the ramp.
Rated current (rRated_Current):	Rated current of phase 1, 2 and 3 of zone. Heaters have same current rating for all 3 phases.

1.5.12.2 Function blocks

Overview of the PS564-TEMPCTRL package

The temperature control library package (PS564-TEMP_CTRL) designed for the demand of advanced temperature control for critical processes, which need precise temperature control and e.g. adaptive tuning for ease of handling and changing environmental or process conditions. The package consists of libraries and examples. This document describes the function blocks of the main library TECT_TEMP_CONTROL_V24.lib. All other library files: (ADCTRL_AC500_V24.lib, TECT_EXT_AC500_V24.lib and TECT_EXT_AC500_V24…obj) are internal and need not be called by the end user.

Table 198: Overview of function blocks

Function Block	Description	Use Range
TECT_TEMP_CONTROL	Temperature control function block.	For one zone
TECT_LOG_FILE	Logging of process value change.	For entire system
TECT_TEMP_SIMU	Temperature zone simulation.	For one zone
TECT_NOISE_FILTER	Filter for short time noise.	For one zone
TECT_GROUP	Temperature control function block for a group	For one group
TECT_PWM8	Coordination of PWM outputs up to 8 zones.	For up to 8 zones
TECT_RECIPE	Read/Write recipe data in a file.	For entire system
TECT_DATA_FLASH	Read/Write/Delete AutoTune data on flash.	For entire system
TECT_HMI_MUX	Manage data transfer of active group to be monitored on operating HMI.	For entire system
TECT_SYSTEM	Function block for system fault monitoring, system control and zone information.	For entire system

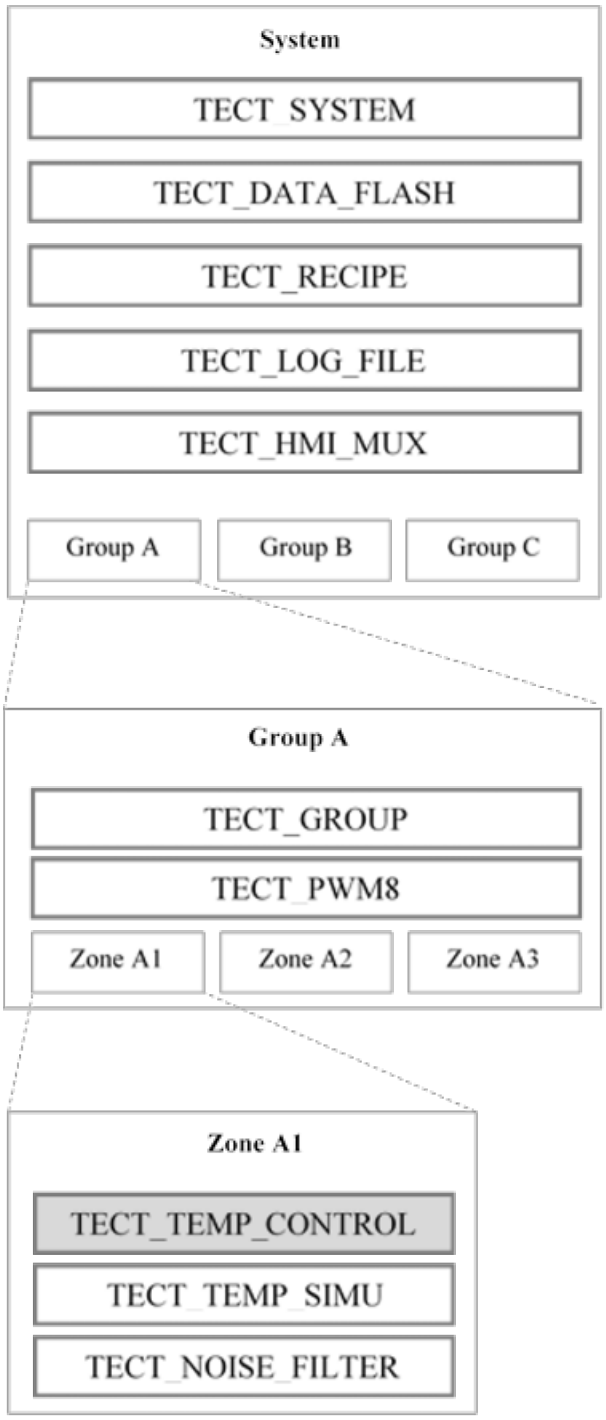
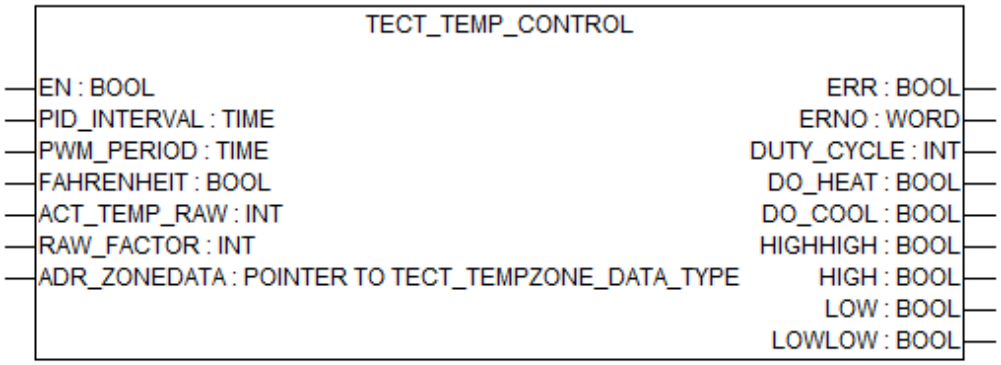


Fig. 596: Overview: Function blocks

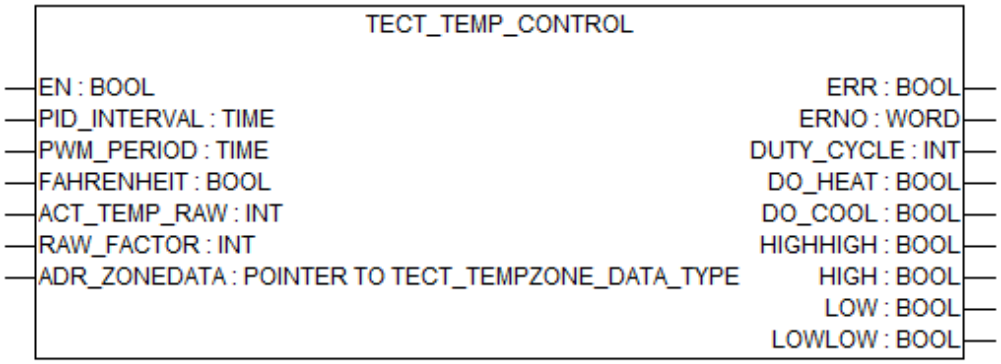
- Function blocks must be used.
- Optional function blocks.

1.5.12.2.1 TECT_TEMP_CONTROL



TECT_TEMP_CONTROL is the main control function block of temperature control process. An input variable ADR_ZONEDATA saves all relevant parameters, settings and values for one temperature zone. All function block for the zone interact with each other through ADR_ZONEDATA variable.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PID_INTERVAL Data type: TIME, Range: > 1 ms, Default value: 2 s.

Interval of PID and AutoTune process.

For PID process: The interval time, at which the PID updates the duty cycle of output based PWM_PERIOD.

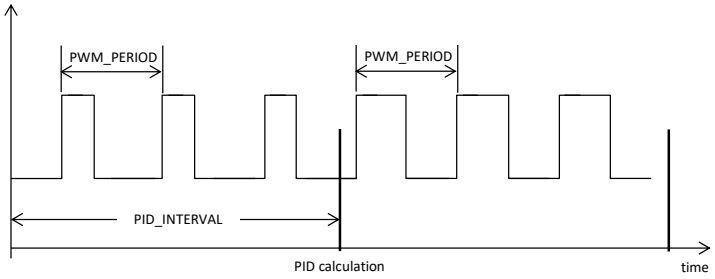


Fig. 597: PID Interval time

PWM_PERIOD Data type: TIME, Range: > 1 ms, Default value: 2 s.

Pulse Width Modulation (PWM) output period.

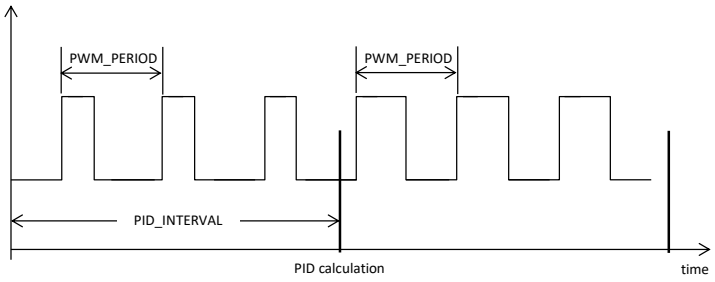


Fig. 598: PWM output period

> *PWM_PERIOD should be always less than or equal to PID_INTERVAL.*

FAHRENHEIT Data type: BOOL

A TRUE at the input FAHRENHEIT of the Funktion Block displays the temperature in Fahrenheit, a FALSE displays the temperature in Celsius.

ACT_TEMP_RAW Data type: INT

Actual raw temperature value from analogous input sensor.

RAW_FACTOR Data type: INT, Default value: 10.

Sensor provides the the ACT_TEMP_RAW, this value doesn't match to the temperature in Celcius. Hence RAW_FACTOR is necessary to scale the temperature into Celsius scale.

Example for RAW_FACTOR If ACT_TEMP_RAW is 200 which indicates 20 degree, then the RAW_FACTOR will be 200 / 20 = 10.

> *The RAW_FACTOR should not be zero.*

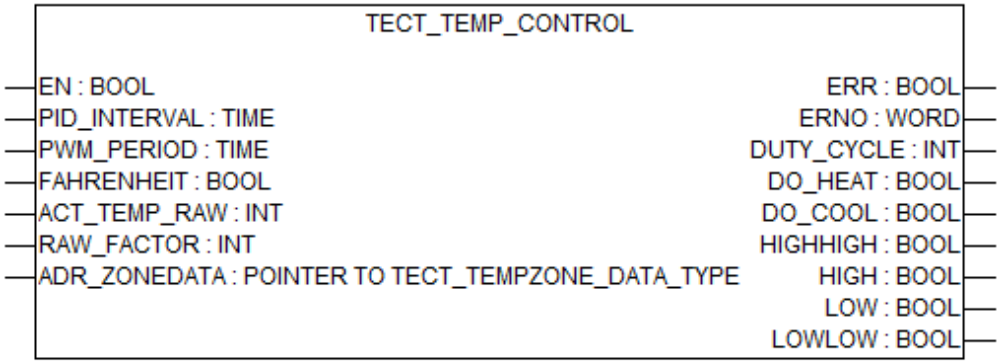
ADR_ZONE-DATA

Data type: POINTER TO TECT_TEMPZONE_DATA_TYPE

POINTER TO ZONEDATA defined as TECT_TEMPZONE_DATA_TYPE.

Input as POINTER TO ZoneData. This variable can be connected to the output signal of the ADR-Block. An input variable ZoneData saves all relevant parameters, settings and values for one temperature zone. All function blocks for the zone interact with each other through ZoneData variable *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

DUTY_CYCLE

Data type: INT

PWM (Pulse Width Modulation) output DUTY_CYCLE is displayed in percentage.

Positive value indicates heating, negative value indicates cooling.

DO_HEAT

Data type: BOOL

Pulse output for heating process. This variable can be connected to the digital output signal.

DO_COOL

Data type: BOOL

Pulse output for cooling process. This variable can be connected to the digital output signal.

HIGHHIGH Data type: BOOL

This variable is TRUE, when output HIGHHIGH temperature error is active. The limit value is declared in the data type TECT_TEMPZONE_DATA_TYPE. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

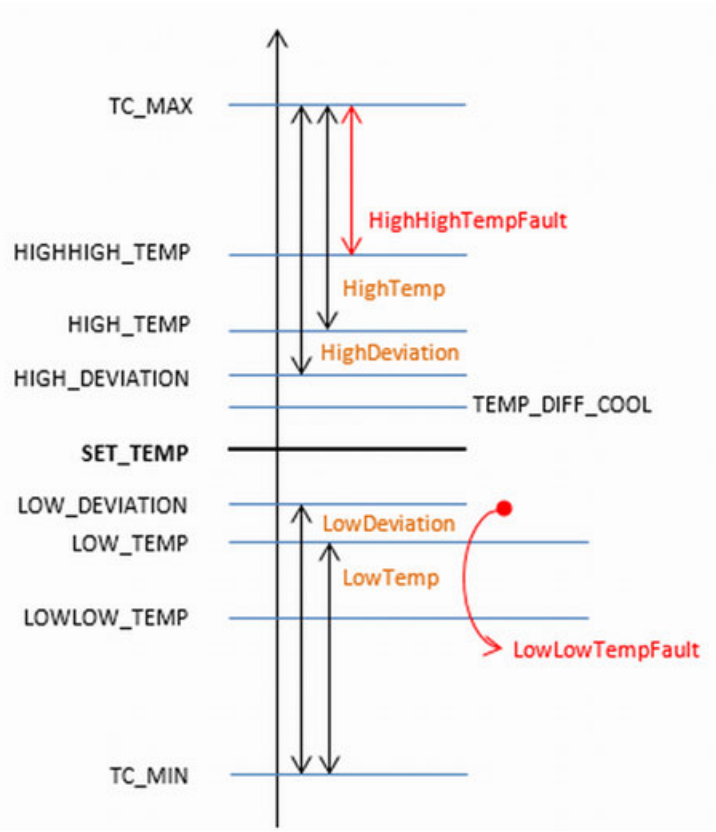


Fig. 599: High temperature alarm and HighHigh temperature fault.

HIGH Data type: BOOL

This output variable is TRUE, when HIGH temperature alarm is active. The limit value is declared in the data type TECT_TEMPZONE_DATA_TYPE. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

LOW Data type: BOOL

This output variable is TRUE, when low temperature alarm (LOW_TEMP) is active. The limit value is declared in the data type TECT_TEMPZONE_DATA_TYPE. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

Example for LowTemp alarm and LowLowTempFault alarm.

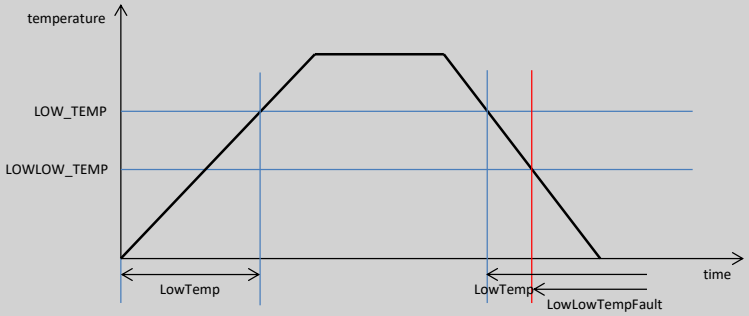
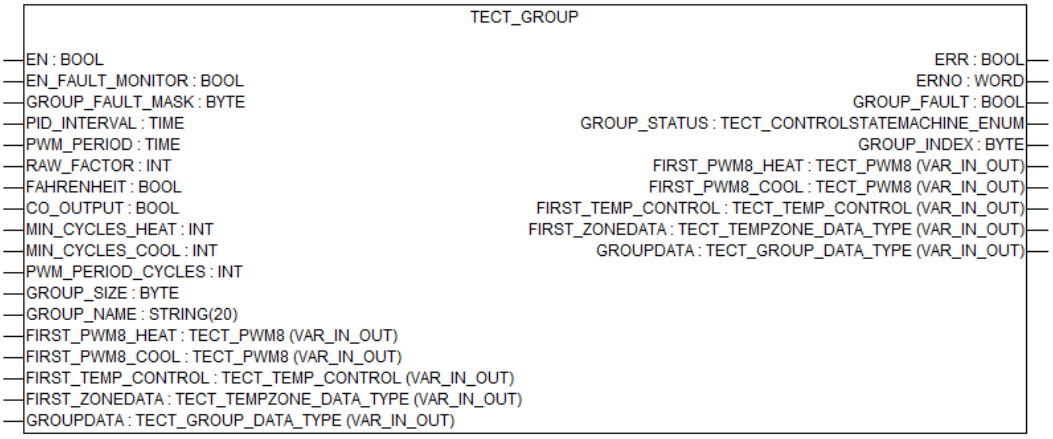


Fig. 600: LowTemp alarm, LowLowTempFault alarm

LOWLOW Data type: BOOL

This output variable is TRUE, when LOWLOW temperature error is active. Limit value is declared in the data type TECT_TEMPZONE_DATA_TYPE. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

1.5.12.2.2 TECT_GROUP

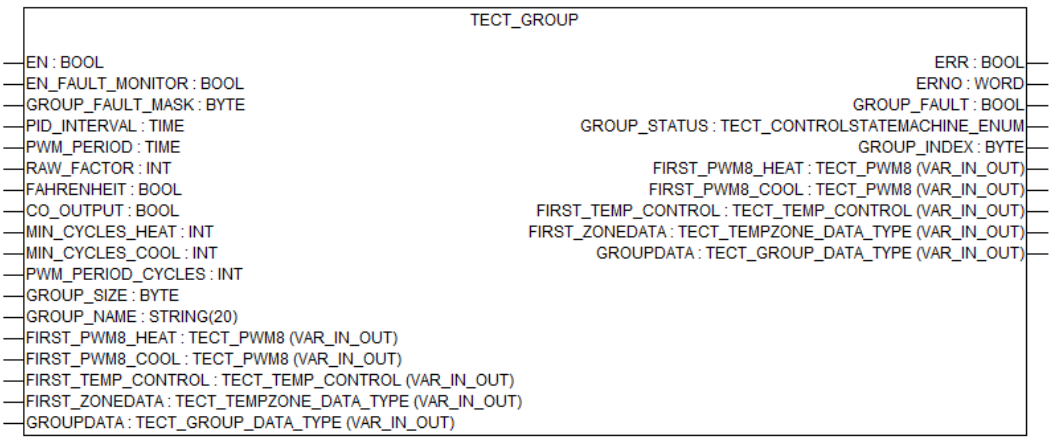


Function block TECT_GROUP realizes the following functions:

- Group fault monitoring.
- Group control: enable process, reset process, enable AutoTune.
- Coordinated output of the zones in the group.

Function block TECT_GROUP calls the function block TECT_TEMP_CONTROL for each zone internally. Thus a separate call of function block TECT_TEMP_CONTROL for each zone is not necessary. Function block TECT_GROUP also calls separately Funktion Block TECT_PWM8 for heating and cooling.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

EN_FAULT_MONITOR

Data type: BOOL

A TRUE at the input enables group fault monitoring.

A FALSE at the input disables group fault monitoring.

Group monitoring of faults is predefined through input GROUP_FAULT_MASK.

GROUP_FAULT_MASK

Data type: BYTE, Default value: 2#00001111

The first four Bits stand for four different error sources:

Bit 0: TC_Fault 1

Bit 1: TC_Fault2

Bit 2: HighHighTempFault

Bit 3: LowLowTempFault

This means, all of the four faults are monitored as group fault. Before enabling the group of zones of the function block TECT_GROUP, define the faults to be masked.

PID_INTERVAL

Data type: TIME, Range: > 1 ms, Default value: 2 s.

Interval of PID and AutoTune process.

For PID process: The interval time, at which the PID updates the duty cycle of output based PWM_PERIOD.

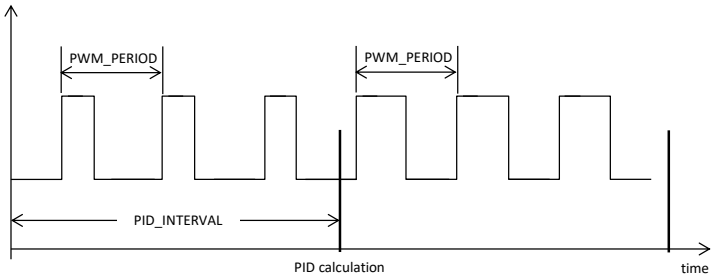


Fig. 601: PID Interval time

PWM_PERIOD

Data type: TIME, Range: >T#1 ms, Default value: T#2 s.

Pulse Width Modulation (PWM) output period.

PWM_PERIOD should be always less than or equal to PID_INTERVAL.

RAW_FACTOR Data type: INT, Default value: 10.

Sensor provides the the ACT_TEMP_RAW, this value doesn't match to the temperature in Celcius. Hence RAW_FACTOR is necessary to scale the temperature into Celsius scale.

Example for RAW_FACTOR If ACT_TEMP_RAW is 200 which indicates 20 degree, then the RAW_FACTOR will be 200 / 20 = 10.

The RAW_FACTOR should not be zero.

FAHRENHEIT Data type: BOOL

A TRUE at the input FAHRENHEIT of the Funktion Block displays the temperature in Fahrenheit, a FALSE displays the temperature in Celsius.

CO_OUTPUT Data type: BOOL

The Input CO_OUTPUT enables coordinated output for PWM8 output of a group. It avoids switching on all zones simultaneously, thus avoiding a voltage drop in the power supply system. At the same time, it compensates the error between digital outputs and duty cycle due to CPU task cycle (as resolution of digital outputs duration). The coordination is realized with function block TECT_PWM8 , called internally.

Input FIRST_PWM8_HEAT and FIRST_PWM8_COOL define the data area of the FIRST_TECT_PWM8 input for heating and cooling. Each function block TECT_PWM8 supports maximum 8 zones. For more than 8 zones, two arrays of function block TECT_PMW8 must be defined for heating and cooling accordingly. If CO_OUTPUT is not used (FALSE), then one dummy variable still needs to be defined for both Inputs FIRST_PWM8_HEAT and FIRST_PWM8_COOL.

MIN_CYCLES_HEAT Data type: INT, Default value = 1, Range: > 0.

Minimum length of ON and OFF time of PWM signal in number of CPU task cycles for heating.

Example Minimum length = 100 ms; CPU task cycle = 50 ms, then the number of cycles for heating is T_MIN_TZ_HEAT = 100 ms / 50 ms = 2.

MIN_CYCLES_COOL Data type: INT, Default value = 2, Range: > 0.

Minimum length of ON and OFF time of PWM signal in number of CPU task cycles for cooling.

Example Minimum length = 100 ms; CPU task cycle = 50 ms, then the number of cycles for heating is T_MIN_TZ_HEAT = 100 ms / 50 ms = 2.

PWM_PERIOD_CYCLES Data type: INT, Default value = 20, Range: > MIN_CYCLES HEAT and MIN_CYCLES_COOL.

Duration of PWM signal period in number of CPU task cycles.

Example	PWM_PERIOD = 1 s, CPU task cycle = 50 ms, then the number of task cycles is PERIOD_TZ = 1 s / 50 ms = 20.

GROUP_SIZE Data type: BYTE, Default value: 1, Range: > 0.

Size of the group or total number of zones in the particular group.

GROUP_NAME Data type: STRING[20]

The name of the group. It will be saved in TECT_TEMPZONE_DATA_TYPE structure.
↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

FIRST_PWM8_HEAT Data type: TECT_PWM8

First instance of function block TECT_PWM8 for heating in the declaration. Each instance supports maximum 8 zones. The function block TECT_PWM8 is called internally. If more instances are needed for one group, they must be defined in an array. If the coordinated output is not used, a dummy instance is still needed. Please see also description of input CO_OUTPUT
↳ *Chapter 1.5.12.2.6 “TECT_PWM8” on page 3327*.

All instances must be defined without memory break. It is highly recommended to define them in an array.

FIRST_PWM8_COOL Data type: TECT_PWM8

First instance of function block TECT_PWM8 for cooling in the declaration. Each instance supports maximum 8 zones. The function block TECT_PWM8 is called internally. If more instances are needed for one group, they must be defined in an array. If the coordinated output is not used, a dummy instance is still needed. Please see also description of CO_OUTPUT ↳ *Chapter 1.5.12.2.6 “TECT_PWM8” on page 3327*.

All instances must be defined without memory break. It is highly recommended to define them in an array.

FIRST_TEMP_CONTROL data type: TECT_TEMP_CONTROL

First instance of function block TECT_TEMP_CONTROL in the declaration. Each zone in the group needs one instance. The function block TECT_TEMP_CONTROL is called internally. Thus a separate call of function block TECT_TEMP_CONTROL for each zone is not necessary
↳ *Chapter 1.5.12.2.1 “TECT_TEMP_CONTROL” on page 3312*.

All instances must be defined without memory break. It is highly recommended to define them in an array.

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE

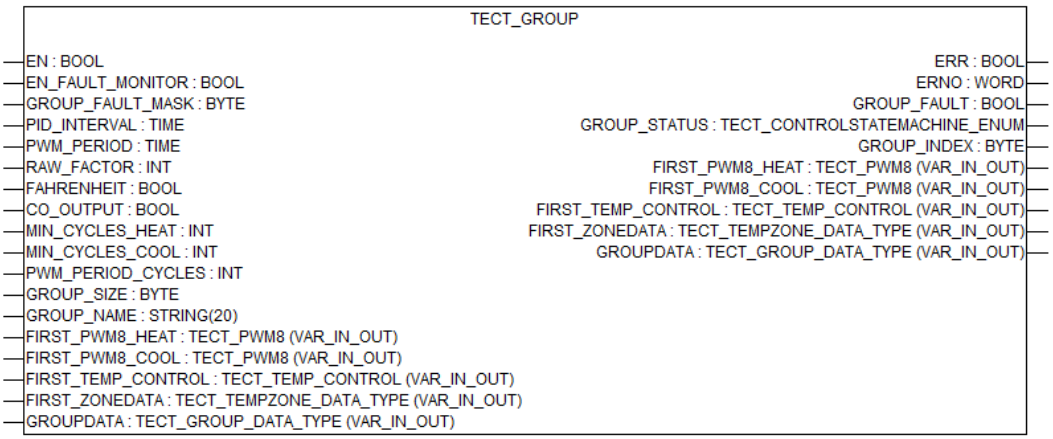
Monitoring of the first zone data structure of the zone group. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

> *All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.*

GROUPDATA Data type: TECT_GROUP_DATA_TYPE

This inout structure of the group will have variables related to the complete group *Chapter 1.5.12.3.4 “TECT_GROUP_DATA_TYPE” on page 3352*.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

GROUP_FAULT Data type: BOOL

A TRUE at the output activates GROUP_FAULT. This will be TRUE only when input EN_FAULT_MONITOR is enabled. It leads to disable the outputs of the whole group.

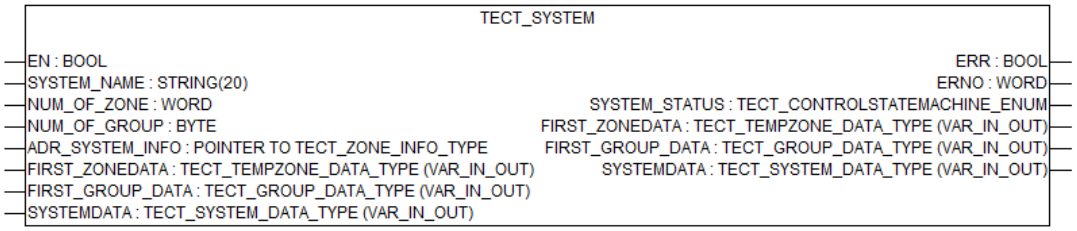
GROUP_STATUS Data type: TECT_CONTROLSTATEMACHINE_ENUM

All status related to the group will be updated to each zone by the enumeration TECT_CONTROLSTATEMACHINE_ENUM *Chapter 1.5.12.3.7 “TECT_CONTROLSTATEMACHINE_ENUM” on page 3357*.

GROUP_INDEX Data type: BYTE

This is the variable which differentiate each group by unique number.

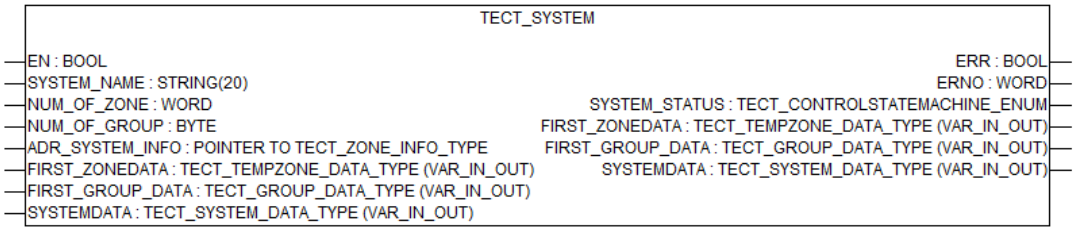
1.5.12.2.3 TECT_SYSTEM



The inputs marked with a triangle are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

The function block TECT_SYSTEM can be used to configure and monitor all groups and zones in the temperature control system. Status of the groups and zones in the system can be monitored. Using the structure TECT_SYSTEM_DATA *Chapter 1.5.12.3.5 “TECT_SYSTEM_DATA_TYPE” on page 3355* different operations for all groups/zones can also be configured.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SYSTEM_NAME Data type: STRING[20]

Configuration of system name. The name does not exceed 20 characters.

NUM_OF_ZONE Data type: WORD, Default value: 1, Range: > 0.

Total number of zones in the system.

NUM_OF_GROUP Data type: BYTE, Default value: 1, Range: > 0.

Number of groups to be operated in the system.

ADR_SYSTEM_INFO Data type: POINTER TO TECT_ZONE_INFO_TYPE.

It will point to the structure TECT_ZONE_INFO_TYPE ↳ *Chapter 1.5.12.3.6 “TECT_ZONE_INFO_TYPE” on page 3356*.

All zones must be declared one after another without break. It is highly recommended to define an array of TECT_ZONE_INFO_TYPE for one groupthe whole system, Maximum size of the array should be total number of zones in the system.

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE

Monitoring of the first zone data structure of the zone group. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.

FIRST_GROUP-DATA Data type: TECT_GROUP_DATA_TYPE

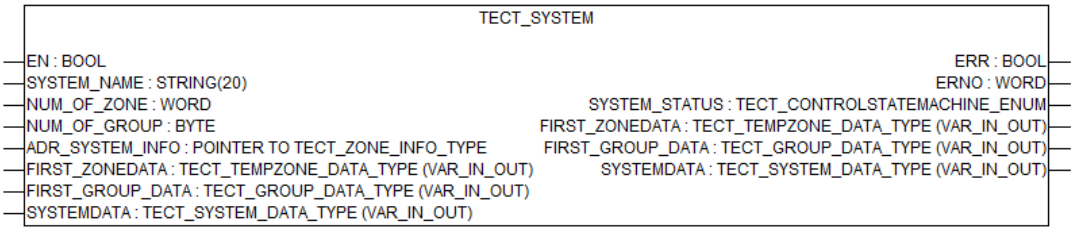
First GROUPDATA of the system in the declaration. All groups must be declared one after another without break ↳ *Chapter 1.5.12.3.4 “TECT_GROUP_DATA_TYPE” on page 3352*.

All groups must be declared one after another without break. It is highly recommended to define an array of TECT_GROUP_DATA_TYPE for one system.

SYSTEMDATA Data type: TECT_SYSTEM_DATA_TYPE

Declaration of data which is common to the complete system ↳ *Chapter 1.5.12.3.4 “TECT_GROUP_DATA_TYPE” on page 3352*.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

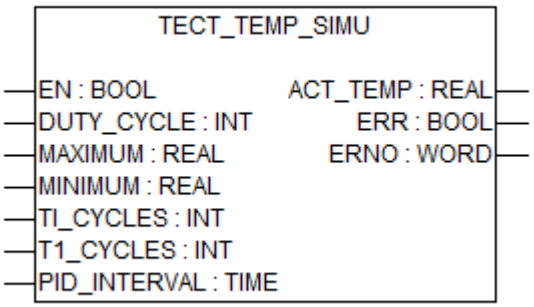
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

SYSTEM_STATUS

Data type: TECT_CONTROLSTATEMACHINE_ENUM

The status of the system including its zones and groups is updated to the enumeration TECT_CONTROLSTATEMACHINE_ENUM *Chapter 1.5.12.3.7 “TECT_CONTROLSTATE-MACHINE_ENUM” on page 3357*.

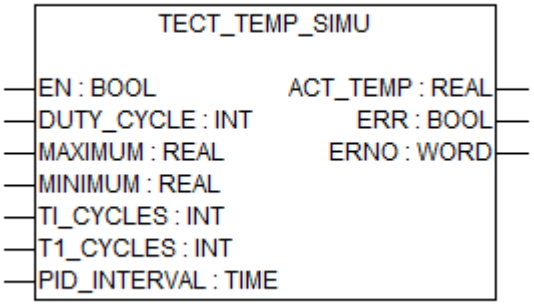
1.5.12.2.4 TECT_TEMP_SIMU



The function block TECT_TEMP_SIMU simulates a temperature zone, e.g. for simulation of the behavior during commissioning.

Connect DUTY_CYCLE input to DUTY_CYCLE output of TECT_TEMP_CONTROL function block. The ACT_TEMP outputs data of the simulated zone temperature.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

DUTY_CYCLE Data type: INT, Range: -100 ... +100, Default value: 0

The PWM duty cycle is for one temperature zone. The structure TECT_TEMP-ZONE_DATA_TYPE defines the duty cycle parameters. ↳ *Chapter 1.5.12.3.1 “TECT_TEMP-ZONE_DATA_TYPE” on page 3343*

MAXIMUM Data type: REAL, Default value: 0

Preset maximum temperature allowed for the simulation. The raw temperature value should be used as in function block TECT_TEMP_CONTROL.

> *Input MAXIMUM should be greater than input MINIMUM.*

MINIMUM Data type: REAL, Default value: 0

Preset minimum temperature allowed for the simulation. The raw temperature value should be used as in function block TECT_TEMP_CONTROL.

> *Input MINIMUM should be less than input MAXIMUM.*

TI_CYCLES Data type: INT

Internal control parameter for simulation. It has similar value as TU from structure TECT_INTERNAL_STATUS_TYPE. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

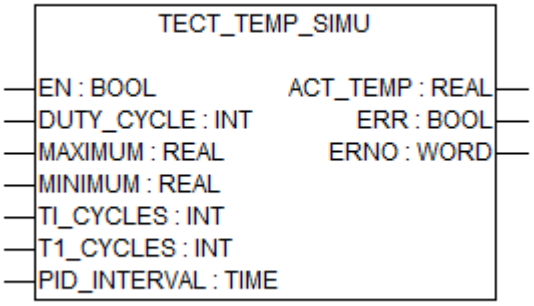
T1_CYCLES Data type: INT, Range: > 0, Default value: 20

Internal control parameter for simulation. It has a similar value as TG from structure TECT_INTERNAL_STATUS_TYPE. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

PID_INTERVAL Data type: TIME, Range: >0 ms, Default value: 2 s

Interval of internal simulation control. Do the setting in the same way as for input PID_INTERVAL from function block TECT_TEMP_CONTROL *Chapter 1.5.12.2.1 “TECT_TEMP_CONTROL” on page 3312*.

Output description



ACT_TEMP (actual temperature) Data type: REAL

The output simulates the actual temperature.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

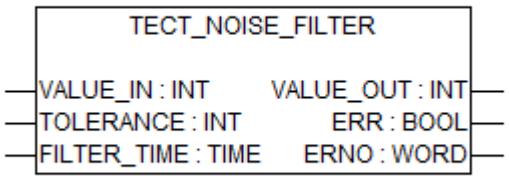
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

1.5.12.2.5 TECT_NOISE_FILTER



The function block TECT_NOISE_FILTER can be used to filter the short disturbance on the sensor input. The VALUE_IN will be ignored, if the change of the value is greater than TOLERANCE for less than FILTER_TIME.

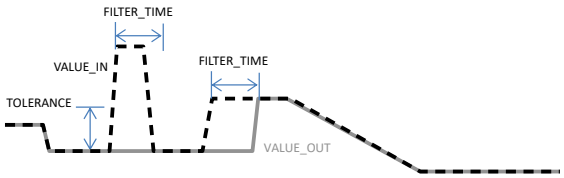
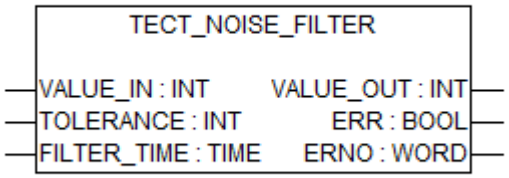


Fig. 602: Noise Filter

Input description



VALUE_IN Data type: INT

Input value from sensor to be filtered.

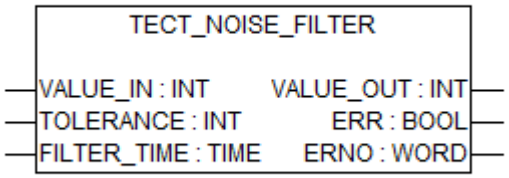
TOLERANCE Data type: INT, Default value: 0.

Tolerance for input value.

FILTER_TIME Data type: TIME, Default value: 0 s, Range: ≥ 0 ms.

Filter time.

Output description



VALUE_OUT Data type: INT

Output value after filtering. It is the input value for ACT_TEMP_RAW in function block TECT_TEMP_CONTROL .

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

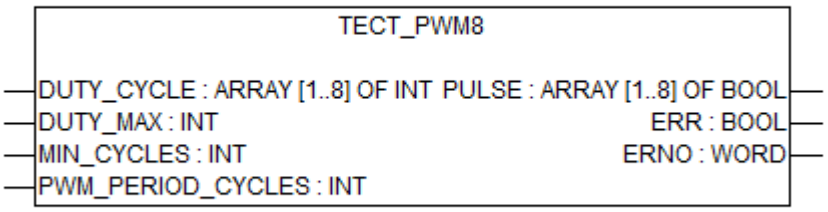
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

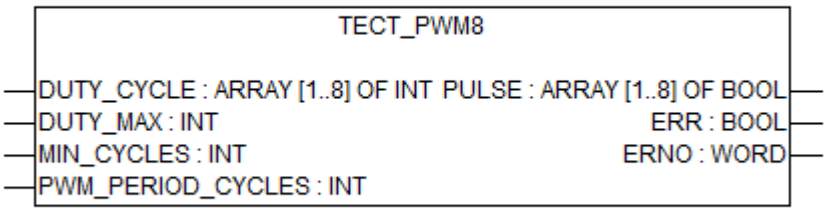
At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

1.5.12.2.6 TECT_PWM8



The function block TECT_PWM8 coordinates up to 8 PWM signals in time behavior. They have not to be switched on at the same time to avoid undesired voltage drop in the power supply system. At the same time, it compensates the error between digital outputs and duty cycle due to CPU task cycle (as resolution of digital outputs duration). For heating and cooling separate instances of this block have to be used.

Input description



DUTY_CYCLE Data type: ARRAY[1..8] OF INT, Range: 0…100

PWM DUTY_CYCLE coordinates maximum 8 zones in a group. If there are more than 8 zones in one group, more function blocks can be used.

DUTY_MAX Data type: INT, Default value: 100, Range: +10 ... + 100.

Maximum value for a duty cycle.

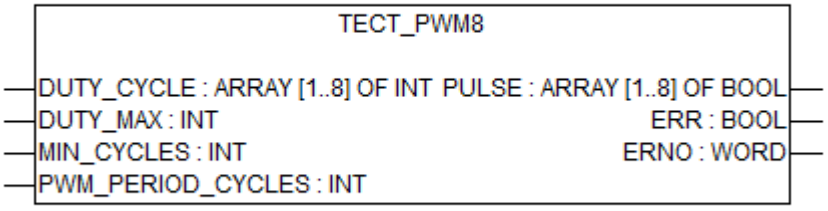
MIN_CYCLES Data type: INT, Default value: 1, Range: > 1.

Minimum length of ON and OFF time of PWM signal in number of CPU task cycles.

PWM_PERIOD_CYCLES Data type: INT, Default value: 20, Range: > MIN_CYCLES

Duration of PWM signal period in number of CPU task cycles.

Output description



PULSE Data type: ARRAY[1..8] OF BOOL

PWM signals for 8 zones.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

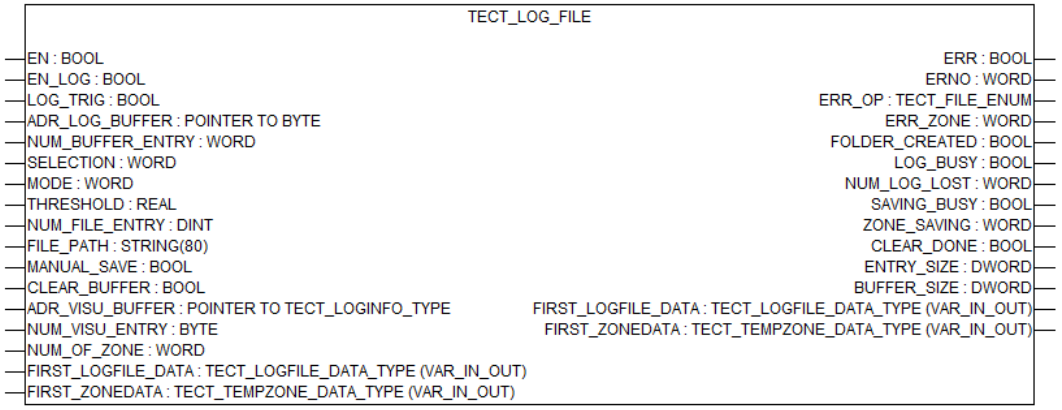
It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

1.5.12.2.7 TECT_LOG_FILE



Function block TECT_LOG_FILE logs the predefined process status and values together with time stamp in a logging buffer when trigger LOG_TRIG has a rising edge. To execute this block memory card is recommended for saving log files. The predefined process status and values are:

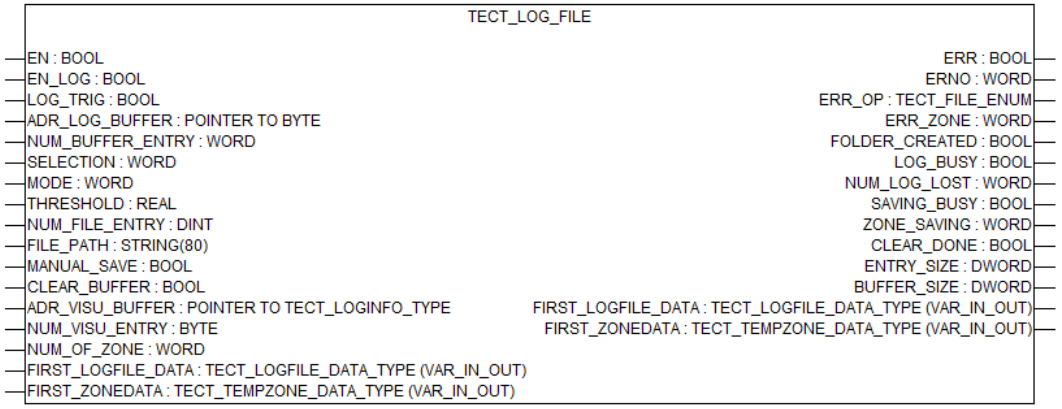
- Control Word
- Status Word
- SET_TEMP
- ACT_TEMP
- Duty Cycle
- Control State
- Output Status
- Latest Error
- Errors (Error Word).

Please see “Description of Structure TECT_TEMPZONE_DATA_TYPE” for details of the process status. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

The logging buffer has a size of NUM_BUFFER_ENTRY (number of entries) for each zone. As long as the logging entry reaches the end of logging buffer, the entries in logging buffer will be saved as a log file in .csv format under a user defined folder. The file name is generated automatically from the zone index, its group index and the month and date value while saving. This log file can be opened in e.g. Excel with comma as delimiter and can be analyzed.

The second function of this function block is to display entries in logging buffer using CODESYS visualization as live log. This function block can be used not only to monitor the process but also for system diagnosis.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN (enable)

Data type: BOOL

The function block is activated by a TRUE at the input EN. It enables live logging visualization. With a rising edge, the folder to save log file is created. Depending on the MODE, a subfolder will be created for each zone. With a falling edge, clear logging buffer action is executed and backup file is created. After that, FOLDER_CREATED is reset to FALSE. See CLEAR_BUFFER and FOLDER_CREATED for details.

EN_LOG

Data type: BOOL

A TRUE at the input will enable logging to the buffer and copying of files from buffer to .csv file when buffer is full.

LOG_TRIG

Data Type: BOOL

A TRUE at the input executes logging a data entry into logging buffer with a rising edge.

ADR_LOG_BUFFER

Data type: POINTER TO BYTE

Start address of the logging buffer since this is POINTER TO BYTE. It is recommended to connect the input through ADR and point it to %RB area.

NUM_BUFFER_ENTRY

Data type: WORD, Range: 1 ... 65535, Default value: 100.

The size of logging buffer for each zone is the number of entries.

Following conditions to be checked to avoid data loss while logging in the buffe:

- *NUM_BUFFER_ENTRY should always be greater than NUM_OF_ZONE. Recommendation: NUM_BUFFER_ENTRY > NUM_OF_ZONE x 2.*
- *Interval of trigger for logging LOG_TRIG must be greater than one second.*
- *Monitor the output NUM_LOG_LOST for the logs/data lost in the buffer as indication of possible adaptation of log configuration.*

SELECTION Data type: WORD, Default value: 2#11111111.

Selection of predefined values to be logged: bit 0 to bit 8 indicates the current version.

Bit x = TRUE: the corresponding value will be logged.

- Bit 0: Control Word,
- Bit 1: Status Word,
- Bit 2: SET_TEMP,
- Bit 3: ACT_TEMP,
- Bit 4: Duty Cycle,
- Bit 5: Control State,
- Bit 6: Output State,
- Bit 7: Latest Error.
- Bit 8: Errors

MODE Data type: WORD, Default value: 2#00

Define different mode of logging.

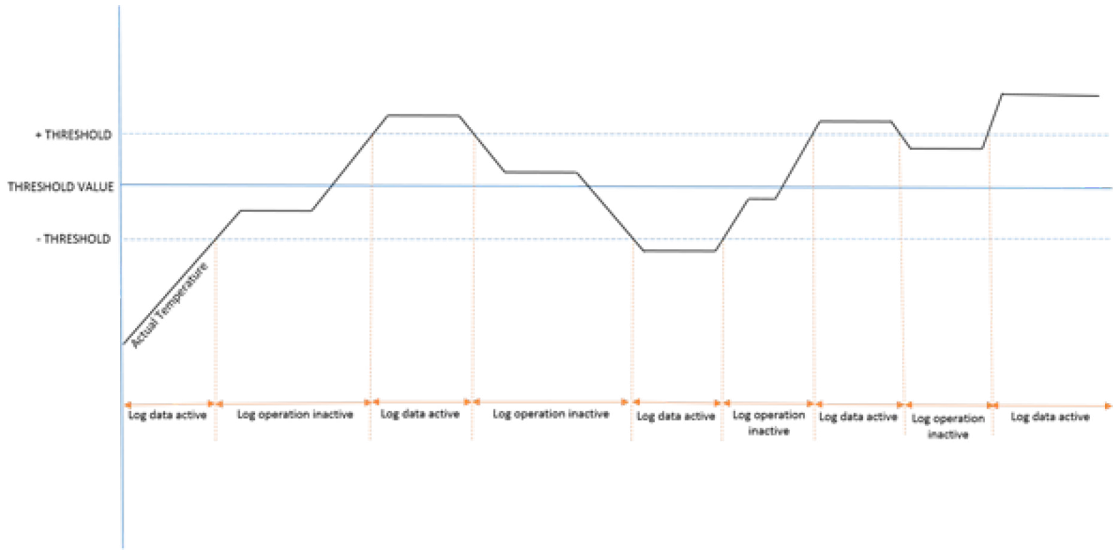
This table defines the different modes of logging:

- Bit 1, Bit 0: Log mode
  - 00: Log all items as long as log trigger is active.
  - 01: Log, if one of the log values changes, when trigger is active.
  - 11: Log, if CW or SW changes or ACT_TEMP changes out of threshold (+/- THRESHOLD), when trigger is active.
- Bit 4: Internal log trigger, TRUE: It will provide internal trigger to log the files. When Bit 4 is TRUE and there is change in Control word or Status word, a trigger signal will be created. It is a trigger additional to LOG_TRIG.
-
- Bit 8: FALSE: Create subfolder automatically; TRUE: Do not create subfolder.
- Bit 9: Valid, if bit 8 is FALSE.
  - FALSE: subfolder name with index in DECIMAL.
    For example, for group 10, index 3, subfolder name is G010Z003.
  - TRUE: subfolder name with index in HEXADECIMAL.
    For example, for group 10, zone 12, subfolder name is G0AZ0C.

THRESHOLD Data type: REAL

Threshold of actual temperature. Set the value in combination with MODE (log mode). Log operation will be active when acutal temperature is out of +/- of the Threshold value. See the below figure which explains when is active or inactive.

Bit 1-0: 11: Threshold of actual temperature.



NUM_FILE_ENTRY Data type: DINT, Range: >= 0, Default value: 0

The logging of data depends on the value of NUM_FILE_ENTRY input is assigned.

- If NUM_FILE_ENTRY = 0, then a new log file is created daily, previous day file is backed up with the same file name but a different suffix. The suffix is the last three HEX numbers of current time (without second).
- If NUM_FILE_ENTRY >0, a new file is created when number of log entries is equal to NUM_FILE_ENTRY defined. Old file will be backed up with the same file name but a different suffix. The suffix is the last three HEX numbers of current time (without second).
- If NUM_FILE_ENTRY <0, it is a invalid condition and function block will generate error 16#4090.

FILE_PATH Data type: STRING[80]

File Pathfor saving log file (followed with or without '/'), e.g. 'SDCARD/folder'. The physical disk like SDCARD or FLASHDISK must be specified in the FILE_PATH. The folder following will be created if it does not exist yet. But only one folder level will be created, this means the parent folder must already exist.

For Bit 8 of input MODE: FALSE: a subfolder will be created for each zone. The log files will be saved under FILE_PATH/subfolder. See MODE, bit 8 and bit 9 for details. It is recommended to save the log file on memory card.

MANUAL_SAVE Data type: BOOL

A rising edge at the input will save logging buffer to .csv log file manually.

CLEAR_BUFFER Data type: BOOL

A rising edge at the input clears the logging buffer: New entries in the logging buffer are written into the .csv file. The .csv file is renamed into a backup file with same file name but different extension. The extension is composed of the last three HEX Numbers of current time without seconds.

ADR_VISU_BUFFER Data type: POINTER TO TECT_LOGINFO_TYPE

Start address of the log visualization data. It is recommended to define the log visualization data in array. This can be used only with CODESYS visualization. For example, if there are 4 zones to be logged and each zone has 10 log visualization entries, then a one dimensional or two dimensional array can be defined ↬ *Chapter 1.5.12.3.3 "TECT_LOGINFO_TYPE" on page 3351*:

visu_array : ARRAY[1..40] OF TECT_LOGINFO_TYPE; (in one dimensional array).

visu_array : ARRAY[1..4, 1..10] OF TECT_LOGINFO_TYPE; (in two dimensional array).

NUM_VISU_ENTRY Data type: BYTE, Default value: 10.

Number of log entries displayed in log visualization for one zone.

NUM_OF_ZONE Data type: WORD, Default value: 1.

If TECT_GROUP is used, then Range: 1…65535. If TECT_GROUP is not used, then Range: 1…255.

Number of zones to be logged into the .csv file or displayed in visualization.

FIRST_LOG-FILE_DATA Data type: TECT_LOGFILE_DATA_FILE

First structure of data logging internal settings and values *Chapter 1.5.12.3.2 “TECT_LOG-FILE_DATA_TYPE” on page 3350*.

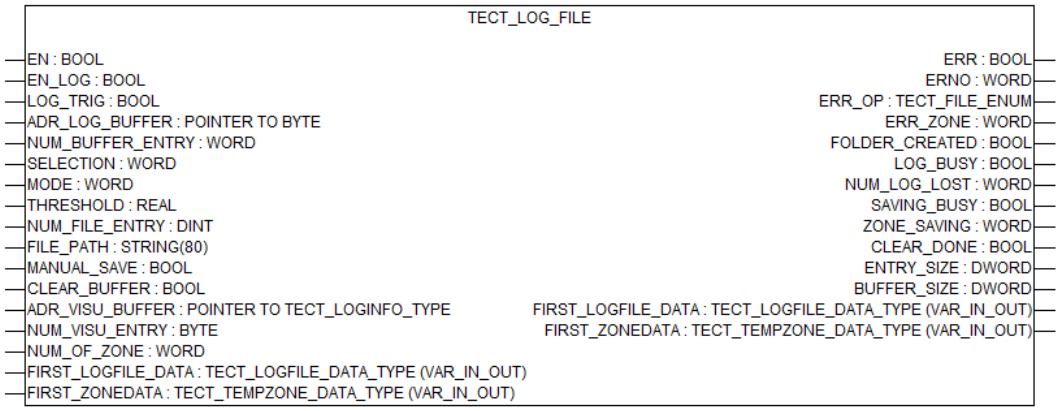
> *It is highly recommended to define an array of TECT_LOGFILE_DATA_TYPE for this input. The size of the array must be the same as the input MUM_OF_ZONE.*

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE

First structure of process data and parameters for the zone. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

> *All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.*

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ERR_OP

Data type: TECT_FILE_ENUM

Display the logging operation, during which the error occurs *Chapter 1.5.12.3.9 “TECT_FILE_ENUM” on page 3358*.

ERR_ZONE

Data type: WORD

Indicate the zone in which error is generated.

FOLDER_CREATED

Data type: BOOL

This output variable is TRUE, if folder and subfolder have been created. Each time input EN has a rising edge, creating folder action will be executed. Depending on the number of zones and MODE 8, it could take some seconds to some minutes to create the folder and subfolders. If the folder already exists, the files inside will be kept untouched. It is recommended to enable logging (EN_LOG = TRUE) after FOLDER_CREATED is TRUE. Otherwise it could cause a loss of log data.

LOG_BUSY

Data type: BOOL

This output variable is TRUE, as long as input EN is TRUE and input EN_LOG is TRUE.

This output variable is FALSE, if input EN is TRUE and input EN_LOG is FALSE (pause), or if input EN has a falling edge. Then LOG_BUSY will be FALSE as soon as the backup of all log files have been created.

NUM_LOG_LOST

Data type: WORD

Number of log entries lost in the logging buffer due to busy of file operation, e.g. log saving operation of other zones.

SAVING_BUSY

Data type: BOOL

This output variable is TRUE, as long as log files are getting saved to the log folder.

This output variable is FALSE, if no saving of files to log folder is ongoing.

ZONE_SAVING

Data type: WORD

The particular zone, for which the file operation is executing (require: SAVING_BUSY is TRUE)

CLEAR_DONE

Data type: BOOL

Clear logging buffer is done.

ENTRY_SIZE Data type: DWORD

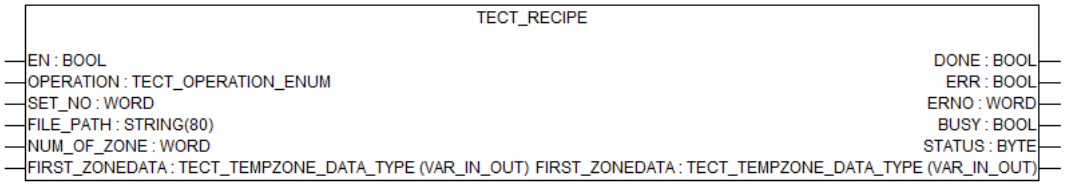
Size of one log entry in byte. The size depends on input data SELECTION.

BUFFER_SIZE Data type: DWORD

Size of whole buffer for all zones in byte. The size depends on output data ENTRY_SIZE and input data NUM_BUFFER_ENTRY and NUM_OF_ZONE.

> *Check this value to make sure the BUFFER_SIZE doesn't exceed the available size of specified data area.*

1.5.12.2.8 TECT_RECIPE



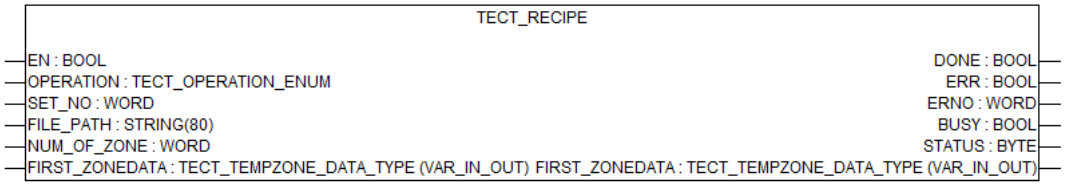
> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

The function block can save/write the necessary control settings to a CSV file. It also can load/read the settings from the CSV file.

This block will read/write from a file system, e.g. memory card. The files can be accessed via an FTP client connected to AC500 FTP server. A backup copy of the CSV file will be created with the same file name but with suffix .cpy after a WRITE operation and before a READ operation. In case of file corruption, the CPY file can be renamed into CSV file manually.

This function block can be used to save the dedicated control settings and settings for a temperature control system. Furthermore, it can also be used to manage recipes (multiple sets of control settings).

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

OPERATION Data type: TECT_OPERATION_ENUM

1: READ operation.

2: WRITE operation ↳ *Chapter 1.5.12.3.10 “TECT_OPERATION_ENUM” on page 3359*.

SET_NO Data type: WORD, Default value: 0, Range: 0 ... 65535

Recipe set number. This number will be used in the recipe CSV file name. See description of FILE_PATH.

FILE_PATH Data type: STRING[80]

There are two possibilities to define FILE_PATH, thus two methods to define the recipe file name:

- Default recipe file name: define a folder for saving recipe file (followed with or without ‘\’). The recipe file will be saved under FILE_PATH with a default name TZD*.csv (* stands for SET_NO).
  For example, if SET_NO = 3, FILE_PATH = SDCARD\RECIPE, then a recipe file TZD3.csv will be created (OPERATION = WRITE) or read (OPERATION = READ) under SDCARD\RECIPE.
- User defined recipe file name: define folder with CSV file name. In this case SET_NO has no meaning.
  For example, if FILE_PATH = SDCARD\RECIPE\myRecipe.csv, then a recipe file MYRECIPE.csv will be created (OPERATION = WRITE) or read (OPERATION = READ) under SDCARD\RECIPE.

The recipe file and the last level of the folder can be created by the function block, but the parent folder must already exist.

For example, if FILE_PATH = SDCARD\Level1\Level2\Level3 or FILE_PATH = SDCARD\Level1\Level2\Level3\myRecipe.csv, then SDCARD\Level1\Level2 must already exist but \Level3 can be created by the function block.

NUM_OF_ZONE Data type: WORD, Default value: 1, Range: > 0.

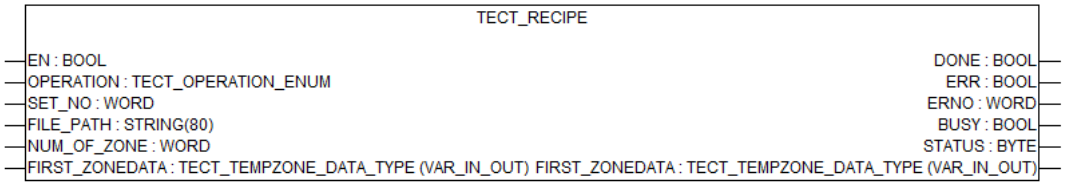
Total number of zones in the system.

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE

Monitoring of the first zone data structure of the zone group. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

BUSY

Data type: BOOL

TRUE: Will indicate that recipe operation READ/WRITE is ongoing and system is busy.

STATUS

Data type: BYTE

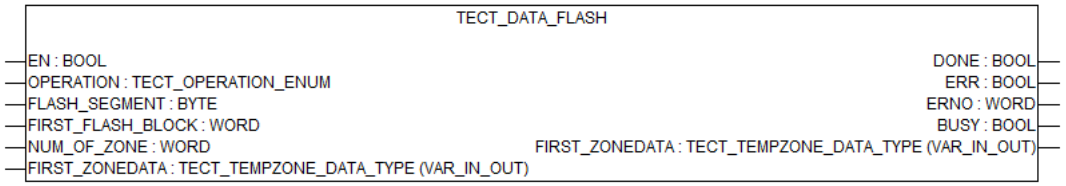
Output of this block indicates status of the process in the recipe function block. This input is helpful in displaying process status in the HMI.

Table 199: Function block TECT_RECIPE - STATUS description

STATUS	Description
0	Function block is disabled or no process running.
1	Recipe function block is BUSY, either READ or WRITE process going on
2	Recipe process is complete
3	Wrong parameter at input OPERATION
4	Wrong parameter at input NUM_OF_ZONE
5	Timeout of the recipe operation state
6	Read data failed: no correct format of data items.

STATUS	Description
7	Mismatch in number of zones declared and zone data in recipe.
8	CAA file error: Time limit exceeded
9	CAA file error: Order has been aborted by activating input xAbort
10	CAA file error: Invalid handle
11	CAA file error: Directory or file does not exist
12	CAA file error: Directory or file already exists
13	CAA file error: No further entries are available
14	CAA file error: File or directory is not empty
15	CAA file error: Drive, file or directory is write-protected
16	CAA file error: Wrong parameter(s) at function block
17	CAA file error: Unknown Error
18	CAA file error: Not all data has been written
19	CAA file error: Function not supported
20	CAA file error: No file handles or user tasks available (too many files open or tasks accessing file system)
21	CAA file error: Volume is full
22	CAA file error: Cannot open disk or no valid disk present
23	CAA file error: File/directory could not be accessed to execute operation
24	CAA file error: Internal or not further specified error
25	CAA file error: Source file could not be opened to execute required operation
26	CAA file error: Destination file could not be opened to execute required operation
27	CAA file error: Data could not be written to destination file
28	CAA file error: Data could not be read from source file
29	CAA file error: Formatting operation has not been confirmed as xEnable before activation the Function block with xExecute.

1.5.12.2.9 TECT_DATA_FLASH



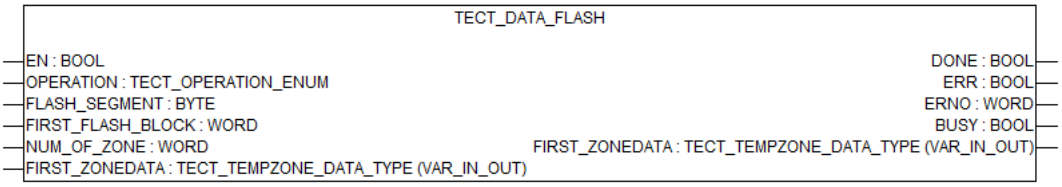
The function block TECT_DATA_FLASH can be used to save following data on flash:

- Internal parameters: rKS, rTU, rTG and wCoolFact_Tune *Chapter 1.5.12.3.1.1 “TECT_INTERNAL_STATUS_TYPE” on page 3343*).
- Machine set parameters: wCoolFact, rKP, rTI, rTD, rT1 *Chapter 1.5.12.3.1.2 “TECT_MACHINE_SET_TYPE” on page 3344*.

The main purpose of the flash operation is to save the parameters calculated from AutoTune, as they need a long time to be calculated. This function block is used as the second safety copy of AutoTune parameters besides saving as retain variables.

To rewrite the content on the flash, current data needs to be deleted before write operation. One instance of this function block has to be called for the complete system.

Input description



> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in edge triggered mode.

The execution of the function block is started with a positive edge on the input EN.

In the first call it evaluates its inputs and creates a local copy of them. Afterwards it starts its internal algorithm. It may take several PLC task cycles until it is executed.

Once the execution is completed DONE and its other outputs are set. To start the function block again a new positive edge on the input EN has to be applied.

OPERATION

Data type: TECT_OPERATION_ENUM

1: Enable READ operation: READ data from flash to TECT_TEMPZONE_DATA_TYPE.

2: Enable WRITE operation: WRITE data from TECT_TEMPZONE_DATA_TYPE to flash.

> *Each function block can only be written once. For a second WRITE action, the whole flash segment needs to be deleted first.*

3: Enable DELETE operation: Delete the whole flash segment.

4: Enable CHECK_READ operation: Check, if there are already valid parameters (KS>0) in TECT.

5: Enable DELETE_WRITE operation: Before write data from TECT_TEMPZONE_DATA_TYPE to flash, delete the whole flash segment first. It is used to avoid error 4101 by WRITE: Block was already written.

6: Enable CHECK_WRITE operation: Combination of CHECK_READ and DELETE_WRITE.

FLASH_SEGMENT

Data type: BYTE, Default value: 1, Range: 1 or 2

Number of the data segment. Each segment has 1927 blocks (0 ... 1926).

FIRST_FLASH_BLOCK Data type: WORD, Default value: 0, Range: 0 ... 1926.

Each segment has 1927 blocks (0…1926). Each block has 32 bytes. Each temperature zone needs one block (ie. 32 bytes).

The first block is assigned for FIRST_ZONEDATA.

NUM_OF_ZONE Data type: WORD, Default value: 1, Range: > 0.

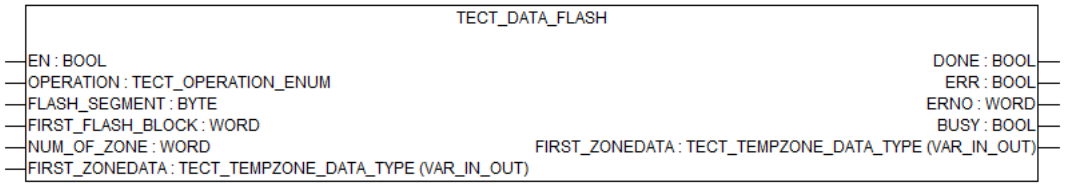
Number of zones in the entire system.

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE.

Monitoring of the first zone data structure of the zone group.

> *All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.*

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

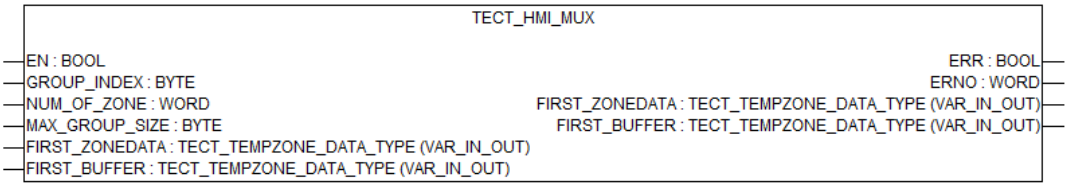
Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

BUSY Data type: BOOL

TRUE: Will indicate that flash operation READ/WRITE/DELETE is ongoing and system is busy.

1.5.12.2.10 TECT_HMI_MUX

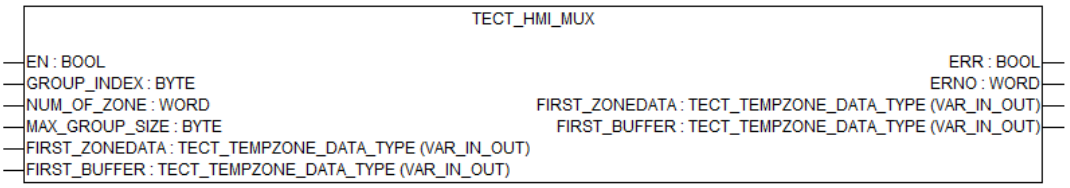


The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

The function block TECT_HMI_MUX

- helps to reduce the communication effort by reducing the data exchanged between PLC and HMI. It will also reduce the engineering efforts in configuring the panel projects. This function block can be used to commission CP600 projects or CODESYS visualization tool. When TECT_HMI_MUX function block is used in the program, BUFFER inout variable must be used for interface between PLC and HMI or visulaization.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

GROUP_INDEX Data type: BYTE, Default value: 1, Range: > 0.

Index of active Group on HMI, set from HMI. Group Index will be changed while navigating through HMI pages.

NUM_OF_ZONE Data type: WORD, Default value: 1, Range: > 0.

Total number of zones.

MAX_GROUP_SIZE Data type: Byte, Default value: 1, Range: > 0.

Size of the largest group in the system.

FIRST_ZONE-DATA Data type: TECT_TEMPZONE_DATA_TYPE

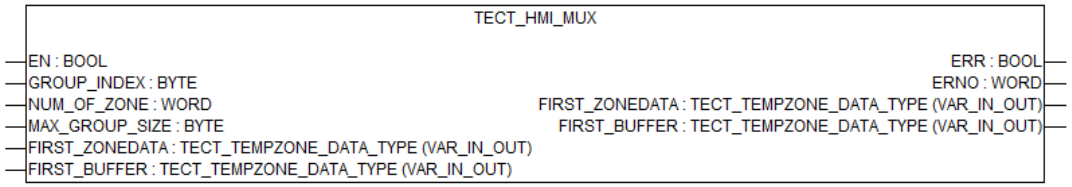
Monitoring of the first zone data structure of the zone group. ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

> *All zones must be declared one after another without break. It is highly recommended to define an array of TECT_TEMPZONE_DATA_TYPE for the whole system.*

FIRST_BUFFER Data type: TECT_TEMPZONE_DATA_TYPE

The first ZONEDATA of the buffer in the declaration. Buffer exchanges the data between page/screens of HMI with the respective zones.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see ↳ *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Table 200: Function block TECT_HMI_MUX - Error code description

DEC	HEX	Error Description
16416	16#4020	GROUP_INDEX is less than or equal to zero
16432	16#4030	NUM_OF_ZONE is less than or equal to zero
16448	16#4040	MAX_GROUP_SIZE is less than or equal to zero
29330	16#7292	Zone_size and zone_index are not assigned

1.5.12.3 Structures and enumerator

The PS564-TEMP_CTRL package (the package containing TECT_TEMP_CONTROL_AC500_V24 library) contains the following structures:

- TECT_TEMPZONE_DATA_TYPE ,
- TECT_LOGFILE_DATA_TYPE,
- TECT_LOGINFO_TYPE,
- TECT_GROUP_DATA_TYPE,
- TECT_SYSTEM_DATA_TYPE,
- TECT_ZONE_INFO_TYPE.

Most variables in the structure cannot be accessed directly from the function blocks. All of them can be read and some can be written directly.

The PS564-TEMP_CTRL package (the package containing TECT_TEMP_CONTROL_AC500_V23 library) contains the following enumerator:

- TECT_CONTROLSTATEMACHINE_ENUM,
- TECT_ERRORCODE_ENUM,
- TECT_FILE_ENUM,
- TECT_OPERATION_ENUM,
- TECT_OUTPUTSTATUS_ENUM.

1.5.12.3.1 TECT_TEMPZONE_DATA_TYPE

TECT_TEMPZONE_DATA_TYPE (Temperature Zone Data) is a structure, which defines the control words, status words, limits and parameters for temperature zone control process.

It is recommmeded to declare TECT_TEMPZONE_DATA_TYPE as retain variables for power safe. All zones must be declared one after another without break. A zone group must be declared together without break. It is recommended to define an array of TECT_TEMPZONE_DATA_TYPE for one group.

Example: If there are three groups with group 1, 2 and 3 having 8, 6 and 4 zones, then TECT_TEMPZONE_DATA_TYPE has to be declared or attached into the function block TECT_GROUP as follows: In global retain variable as tsZonedata := ARRAY[1..18] OF TECT_TEMPZONE_DATA_TYPE. For group 1 as tsZonedata[1], for group 2 as tsZonedata[9] and for group 3 as tsZonedata[15] as the first zone for respective group.

The structure TECT_TEMPZONE_DATA_TYPE consist of 5 internal structures. These substructures are defined as per function of the parameters.

- TECT_INTERNAL_STATUS_TYPE,
- TECT_MACHINE_SET_TYPE,
- TECT_MACHINE_STATUS_TYPE,
- TECT_PROCESS_SET_TYPE,
- TECT_PROCESS_STATUS_TYPE.

TECT_INTERNAL_STATUS_TYPE

This structure contains internal status of the zone data. User can define group size and group name. User can also read control status and auto tune parameters.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	ControlState	TECT_CONTROLSTATEMACHINE *Chapter 1.5.12.3.7 “TECT_CONTROLSTATEMACHINE_ENUM” on page 3357*		Not-Ready	Control state of the machine	FB
2	iRaw_Factor	INT	Raw Value	10	Factor of ACT_TEMP_RAW to actual temperature in °Celsius	FB
3	pGroupData	POINTER TO TECT_GROUP_DATA_TYPE *Chapter 1.5.12.3.4 “TECT_GROUP_DATA_TYPE” on page 3352*			GROUPDATA values	RO
4	byGroup_Size	BYTE		0	Group Size	RW
5	byGroup_Index	BYTE		0	Group Index	RO
6	byZone_Index	BYTE		0	Zone Index	RO
7	rTune_SetPoint	REAL		0	Auto tune setpoint to be calculated	RW
8	rKS	REAL		0	Internal PID parameter KS calculated by tuning	FB
9	rTU	REAL		0	Internal PID parameter TU calculated by tuning	FB
10	rTG	REAL		0	Internal PID parameter TG calculated by tuning	FB
11	wCoolFact_Tune	WORD		1000	Cool factor tune in percentage	FB

TECT_MACHINE_SET_TYPE

This structure contains machine set parameters of the zone data. User can define all parameters which are related to the machine, i.e. parameters which are set as per machine properties and will be defined only once at the time of commissioning. These parameters need to be changed only when some machine properties have changed, e.g. when a temperature sensor changed.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	wCoolFact	WORD		1000	Cool factor in percentage	RW
2	rKP	REAL		0	PID parameter for control P in REAL value	RW

No	Name	Data Type	Unit	Initial Value	Description	Access
3	rTI	REAL		0	PID parameter for control I in seconds	RW
4	rTD	REAL		0	PID parameter for control D in seconds	RW
5	rT1	REAL		0	PID parameter for control filter for D in seconds	RW
6	rHigh-High_Temp	REAL		32767	HighHigh temperature alarm - disables all outputs	RW
7	rLowLow_Temp	REAL		- 32767	LowLow temperature alarm - disables all outputs	RW
8	uiChange_Time	UINT		0	For TC_Fault_2: time in second used to register min_temp_change when duty_cycle ≥ tune_output	RW
9	uiMin_Temp_Change	UINT		0	For TC_Fault_2: minimum temperature change in change_time when duty_cycle ≥ tune_output	RW
10	rTc_Max	REAL		32767	For TC_Fault_1: maximum plausible rActual_Temp	RW
11	rTc_Min	REAL		0	For TC_Fault_1: minimum plausible rActual_Temp	RW
12	rTemp_Diff_Cool	REAL		0	> 0: if temperature is temp_diff_cool higher than setpoint, then cooling is 100% ON until the temp_diff_cool*cool_off_ratio/100 over setpoint is reached. = 0: cooling is controlled as PID	RW
13	iCool_OFF_Ratio	INT		50	In percentage, if temp_diff_cool is active, see temp_diff_cool > 0	RW
14	iMax_Output	INT		100	Maximum allowable duty cycle	RW
15	iTune_Step	INT		500	Set point change in RAW value to be reached during auto tune process	RW
16	iTune_Output	INT		100	PWM duty cycle used during tuning	RW
17	rTune_MinP	REAL		0	Minimum allowable KP value for auto accept	RW
18	rTune_MaxP	REAL		32000	Maximum allowable KP value for auto accept	RW
19	rTune_PMult	REAL		1	Scaling for KP value	RW

No	Name	Data Type	Unit	Initial Value	Description	Access
20	rTc_Offset	REAL		0	Thermocouple compensation in degree (Celsius or Fahrenheit) - if required	RW
21	rPeriod_Fact	REAL		10	Period factor between cooling and heating	RW
22	timMin_Cool_ON	TIME		t# 3 s	Minimum time for which xDO_Cool output to be ON	RW
23	iFF_Time_Ratio	INT		125	>=100, Feedforward time ratio in percentage: for internal set point ramp generator. The smaller the value, the steeper the ramp	R/W
24	rRated_Current	REAL		0	Rated heater current of the Zone. It is always same for all 3 phases.	R/W

TECT_MACHINE_STATUS_TYPE

This structure contains machine status parameters of the zone data. It contains all the AutoTune parameters.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	rKP_TUNE	REAL		0	PID parameter for control P in REAL value calculated by tuning	RO
2	rTI_Tune	REAL		0	PID parameter for control I in seconds calculated by tuning	RO
3	rTD_Tune	REAL		0	PID parameter for control D in seconds calculated by tuning	RO
4	rT1_Tune	REAL		0	PID parameter for control filter for D in seconds calculated by tuning	RO
5	arActual_Current	ARRAY [1..3] OF REAL			Actual heater current of phase 1,2 and 3 of the Zone	R/W

TECT_PROCESS_SET_TYPE

This structure contains process set parameters of the zone data. User can define all parameters which are related to the process, i.e. parameters which are set as per process properties. These parameters will be changing continuously as per process requirement, e.g. Control word.

PID setpoint, control word, high and low temp settings come under this structure.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	wControlWord	WORD		0	Control word	RW
	enable:	BOOL			Bit 0: enable Process.	
	heat_en:	BOOL			Bit 1: enable zone as heat zone. If both heat_en and cool_en are FALSE then zone is monitor only	
	cool_en:	BOOL			Bit 2: enable zone as cooling zone. If both heat_en and cool_en are FALSE then zone is monitored only	
	manual_en:	BOOL			Bit 3: manual mode - highest priority mode, disable all other modes.	
	standby_en:	BOOL			Bit 4: standby setpoint is used.	
	tune_en:	BOOL			Bit 5: enable auto tune with large setpoint step.	
	accept_auto_tune:	BOOL			Bit 6: accept or reject tuning parameters.	
	co_output:	BOOL			Bit 7: coordinated output of one group.	
	Reserved	BOOL			Bit 8: Reserved	
	Control_Optimization:	BOOL			Bit 10, Bit 9: - 00: APERIODIC, - 01: OVERSHOOT, - 10: DEADTIME COMPENSATION, - 11: NO OPTIMIZATION .	
	warm_reset:	BOOL			Bit 13: reset the alarm and process but not AutoTune status.	
	en_3phase_Monitor	BOOL			Bit 14 - True :3phase current monitoring False: single phase monitoring	
	cold_reset:	BOOL			Bit 15: reset all in control process.	
2	iManual_Duty_Cycle	INT		0	PWM output in manual mode.	RW
3	rSetPoint	REAL		0	Process setpoint.	RW
4	rStandBy_SetPoint	REAL		0	Secondary process setpoint for standby.	RW
5	rHigh_Temp	REAL		32767	High temperature alarm, disables heat output.	RW
6	rLow_Temp	REAL		- 32767	Low temperature alarm, disables cool output.	RW

No	Name	Data Type	Unit	Initial Value	Description	Access
7	rHigh_Deviation	REAL		0	High deviation alarm, relative to setpoint.	RW
8	rLow_Deviation	REAL		0	Low deviation alarm, relative to setpoint.	RW

TECT_PROCESS_STATUS_TYPE

This structure contains process status parameters of the zone data. User can read all parameters which are related to the process. These parameters will be changing continuously as per process status, e.g. Status word.

Act temp, status word, error come under this structure.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	rActive_SetPoint	REAL		0	Active setpoint currently used by process.	RO
2	rActual_Temp	REAL		0	Actual value from the Thermocouple.	RO
3	wStatusWord	WORD			Status WORD	RO
					Bit 0: reserved	
					Bit 1: reserved	
					Bit 2: reserved	
					Bit3: reserved	
	standby_on:	BOOL			Bit 4: StandBy mode active.	
	fahrenheit_on:	BOOL			Bit 5: TRUE: Fahrenheit	
	tune_done:	BOOL			Bit 6: AutoTune complete.	
	tune_ accepted:	BOOL			Bit 7: AutoTune values accepted.	
	tune_ state	BOOL			Bit 8: FALSE: start AutoTune with heat. TRUE: start AutoTune with cool.	
	hmi_visible:	BOOL			Bit 12: TRUE: Zone is visible in the HMI. FALSE: zone is invisible.	

No	Name	Data Type	Unit	Initial Value	Description	Access
	group_fault:	BOOL			Bit 13: Group fault active. All zones in that group: Outputs disabled. Process stopped.	
	warning:	BOOL			Bit 14: Warning active, only displayed. Process going on.	
	fault:	BOOL			Bit 15: Fault active, outputs disabled. Process stopped.	
4	OutputStatus	TECT_OUTPUTSTATUS_ENUM *Chapter 1.5.12.3.11 "TECT_OUTPUTSTATUS_ENUM" on page 3359*			Status Code of PWM output.	RO
5	iTune_Status	INT		0	Present tuned value output.	RO
6	Latest Error	TECT_ERRORCODE_ENUM *Chapter 1.5.12.3.8 "TECT_ERRORCODE_ENUM" on page 3357*			Error code of latest error.	RO
7	wErrors	WORD		0	All active errors.	RO
8	wErrors2	WORD		0	Second error word	RO
	ShortCircuit	BOOL		FALSE	Bit 0- Short Circuit Detected in the power circuit	
	OpenCircuit	BOOL		FALSE	Bit 1- Open Circuit Detected in the heater power circuit	
	ControlCircuitFailure	BOOL		FALSE	Bit 2- Control circuit failure detected	
	ZeroRatedCurrentAssigned	BOOL		FALSE	Zero rated current assigned for zone	
9	xDO_Heat	BOOL		FALSE	PWM output for heat cycle.	RO
10	xDO_Cool	BOOL		FALSE	PWM output for cool cycle.	RO
11	iDuty_Cycle	INT		0	Output duty cycle in per cent (%)	RO

1.5.12.3.2 TECT_LOGFILE_DATA_TYPE

TECT_ LOGFILE_DATA_TYPE is a structure which is used for the data of TECT_LOG_FILE function block. It contains all information of zone log data including status of the log process. E.g., user can check the status of folder: / csv file name, save status, file token information.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	diFileEntries	DINT		- 1	1: File not yet created.	RO
2	iNewEntries	INT		0	No of new entries.	RO
3	stFileName	String[12]			\' + 8.3 file name	RO
4	stSubFolder	String[9]			Subfolder name	RO
5	xErr	BOOL		FALSE	TRUE: Error is generated. FALSE: No error.	RO
6	wErno	WORD		0	Error number	RO
7	Err_Op	TECT_FILE_ENUM ↳ *Chapter 1.5.12.3.9 “TECT_FILE_ENUM” on page 3358*			Error operation to the internal structure TECT_FILE_ENUM.	RO
8	bySaveState	TECT_FILE_ENUM ↳ *Chapter 1.5.12.3.9 “TECT_FILE_ENUM” on page 3358*		TECT_NO_OPERATION	Save state to TECT_FILE_ENUM structure.	RO
9	wMyFileToken	WORD		0	Current file token.	RO
10	xSave	BOOL		FALSE	Start save into file.	RO
11	xDirCreated	BOOL		FALSE	TRUE: Directory created.	RO
12	xLogBusy	BOOL		FALSE	TRUE: Log process ongoing.	RO
13	xInit	BOOL		FALSE	One execution after restart/ reset program: to backup (rename) the existing file.	RO
14	xNewFile	Bool		FALSE	To create a new file.	RO
15	xInThreshold	Bool		FALSE	TRUE: Act_temp is in threshold range.	RO
16	wRetryCycle	WORD		0	Number of CPU cycles have been passed after saving failed.	RO
17	xSaveFailed	BOOL		FALSE	Save (write) to file failed.	RO
18	x2ndBuffer	BOOL		FALSE	Second buffer allocated.	RO
19	i2ndNEntry	INT		0	Size of 2nd buffer in number of entries.	RO

No	Name	Data Type	Unit	Initial Value	Description	Access
20	i2ndEntryPosLog	INT		0	Position of current entry in 2nd buffer: 0.. i2ndNEntry-1.	RO
21	ptr2ndBuffer	POINTER TO BYTE			Pointer to begin of the 2nd buffer.	RO
22	wNumLogLost	WORD		0	Number of log lost for the zone	RO
23	ptrLogBuffer	POINTER TO BYTE			Pointer to begin of the zone log buffer.	RO
24	ptrEntryLog	POINTER TO BYTE			Pointer to current entry to be logged in buffer.	RO
25	iEntryPosLog	INT		0	Position of current buffer entry: 0.. NUM_BUFFER_ENTRY-1.	RO
26	iEntryPosWrite	INT		0	Position of buffer entry before which the data has been written into file: 0.. NUM_BUFFER_ENTRY-1.	RO
27	wOldStatus	Array[1..2] OF Word		2 (0)	Old status of the log.	RO

1.5.12.3.3 TECT_LOGINFO_TYPE

TECT_LOGINFO_TYPE is a structure which defines the log info for the log visualization. It contains all information of Log visualization including control word, status word, and control status. User can use visualization to observe this data.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	timtimestamp	DT			Date and time stamp.	RO
2	wControlWord	WORD		0	Control word	RO
3	wStatusWord	WORD		0	Status word	RO
4	rActive_SetPoint	REAL		0	Active setpoint currently used by process.	RO
5	rActual_Temp	REAL		0	Actual value from the Thermocouple.	RO
6	iDuty_Cycle	INT		0	Output Duty Cycle in per cent (%).	RO

No	Name	Data Type	Unit	Initial Value	Description	Access
7	ControlState	TECT_CONTROLSTATEMACHINE_ ENUM *Chapter 1.5.12.3.7 “TECT_CONTROLSTATEMACHINE_ENUM” on page 3357*			Control State of the machine.	RO
8	OutputStatus	TECT_OUTPUTSTATUS_ENUM *Chapter 1.5.12.3.11 “TECT_OUTPUTSTATUS_ENUM” on page 3359*			Status code of PWM output.	RO
9	Latest_Error	TECT_ERRORCODE_ENUM *Chapter 1.5.12.3.8 “TECT_ERRORCODE_ENUM” on page 3357*			Error code of latest error.	RO
10	wErrors	WORD		0	All active errors.	RO
11	wErrors2	WORD		0	Second error word.	RO

1.5.12.3.4 TECT_GROUP_DATA_TYPE

TECT_GROUP_DATA_TYPE structure has the data related to each group in the system. It has all the status of TECT_GROUP function block.

No	Name	Data Type	Unit	Initial Value	Description	Access
1	stGroup_Name	STRING[20]			Name of the group.	RW
2	byGroup_Index	BYTE		0	Group index	RO
3	byGroup_Size	BYTE		0	Size of the group.	RW
4	ControlState	TECT_CONTROLSTATEMACHINE_ ENUM *Chapter 1.5.12.3.7 “TECT_CONTROLSTATEMACHINE_ENUM” on page 3357*		TECT_NotReady	Control state of the machine.	RO
5	byZones_Not_Ready	BYTE		0	Number of zones at Not-Ready status.	RO
6	byZones_Ready	BYTE		0	Number of zones at Ready status.	RO

No	Name	Data Type	Unit	Initial Value	Description	Access
7	byZones_Manual	BYTE		0	Number of zones at Manual status.	RO
8	byZones_AutoTune	BYTE		0	Number of zones at AutoTune status.	RO
9	byZones_PID_Process	BYTE		0	Number of zones at PID process status.	RO
10	byZones_Fault	BYTE		0	Number of zones at Fault status.	RO
11	xEn_Process	BOOL		FALSE	Enable process for all zones.	RW
12	xEn_Heat	BOOL		FALSE	Enable Heat process for all zones.	RW
13	xEn_Cool	BOOL		FALSE	Enable Cool process for all zones.	RW
14	xEn_Standby	BOOL		FALSE	Enable Standby for all zones.	RW
15	xEn_AutoTune	BOOL		FALSE	Enable Auto-Tune process for all zones.	RW
16	xEn_Accept_Auto-Tune	BOOL		FALSE	Accept Auto-Tune values for all zones.	RW
17	xReset_Warm	BOOL		FALSE	Reset Warm for all zones.	RW
18	xReset_Cold	BOOL		FALSE	Reset Cold for all zones.	RW
19	xDisable_Process	BOOL		FALSE	Disable process for all zones.	RW
20	xDisable_Heat	BOOL		FALSE	Disable Heat process for all zones.	RW
21	xDisable _Cool	BOOL		FALSE	Disable Cool process for all zones.	RW
22	xDisable _Standby	BOOL		FALSE	Disable Standby for all zones.	RW

No	Name	Data Type	Unit	Initial Value	Description	Access
23	xDisable _Autotune	BOOL		FALSE	Disable Auto-Tune process for all zones.	RW
24	xDisable _Accept_Autotune	BOOL		FALSE	Disable Auto-Tune values for all zones.	RW
25	byControlByte	BYTE		0	Group Control Byte	RW
	Transfer	BOOL			Bit 0 - Enables command transfer from group to indvidual zones	RW
26	arActual_Current	ARRAY [1..3] OF REAL			Actual current of phase L1,2 and 3 of the group	RW
27	timMonitor-Cycle_Time	TIME		T#600s	Current monitoring cycle time	RW
28	timMonitor-Step_Time	TIME		T#10s	Current monitoring step time in the cycle	RW
29	timSettling_Time	TIME		T#800ms	Settling time allocated for current sensor	RW
30	timLastMeasure-ment	TIME		T#0s	Time taken to complete current measurement of all the zones for last measurement	RW
31	xEn_Co_Monitor	BOOL		FALSE	Enable common sensor current monitoring for all zones	RW
32	xEn_Indivi_Monitor	BOOL		FALSE	Enable individual sensor current monitoring for all zones	RW
33	xControl_Circuit_Failure	BOOL		FALSE	Control circuit failure of one of the zone e.g. SSR failure	RW

No	Name	Data Type	Unit	Initial Value	Description	Access
34	byMaxPercentage	BYTE	%	150	Maximum percentage current allowed for current monitoring	RW
35	byMinPercentage	BYTE	%	50	Minimum percentage current allowed for current monitoring	RW
36	byZeroDetectPercentage	BYTE	%	10	Percentage current to detect control circuit failure	RW

1.5.12.3.5 TECT_SYSTEM_DATA_TYPE

No	Name	Data Type	Unit	Initial value	Description	Access
1	stSystem_Name	STRING[20]			Name of the system.	RW
2	bySystem_Size	BYTE		0	Size of the system.	RW
3	ControlState	TECT_CONTROLSTATEMACHINE_ENUM *Chapter 1.5.12.3.7 “TECT_CONTROLSTATEMACHINE_ENUM” on page 3357*(		TECT_NotReady	Control state of the machine.	RO
4	byGroups_Not_Ready	Byte		0	Number of groups NotReady status.	RO
5	byGroups_Ready	Byte		0	Number of groups Ready status.	RO
6	byGroups_Manual	Byte		0	Number of groups Manual status.	RO
7	byGroups_Autotune	Byte		0	Number of groups Autotune status.	RO
8	byGroups_PID_Process	Byte		0	Number of Groups PID Process status.	RO
9	byGroups_Fault	Byte		0	Number of groups Fault status.	RO
10	byGroups_Mixed_State	Byte		0	Number of groups Mixed status.	RO
11	xEn_Process	BOOL		FALSE	Enable process for all groups.	RW

No	Name	Data Type	Unit	Initial value	Description	Access
12	xEn_Heat	BOOL		FALSE	Enable Heat process for all groups.	RW
13	xEn_Cool	BOOL		FALSE	Enable Cool process for all groups.	RW
14	xEn_Standby	BOOL		FALSE	Enable Standby process for all groups.	RW
15	xEn_Autotune	BOOL		FALSE	Enable Autotune process for all groups.	RW
16	xEn_Accept_Autotune	BOOL		FALSE	Accept Autotune values for all zones.	RW
17	xReset_Warm	BOOL		FALSE	Reset Warm process for all groups.	RW
18	xReset_Cold	BOOL		FALSE	Reset Cold process for all groups.	RW
19	xDisable_Process	BOOL		FALSE	Disable process for all groups.	RW
20	xDisable_Heat	BOOL		FALSE	Disable Heat process for all groups.	RW
21	xDisable_Cool	BOOL		FALSE	Disable Cool process for all groups.	RW
22	xDisable_Standby	BOOL		FALSE	Disable Standby process for all groups.	RW
23	xDisable_Autotune	BOOL		FALSE	Disable Autotune process for all groups.	RW
24	xDisable_Accept_Autotune	BOOL		FALSE	Disable Autotune values for all zones.	RW

1.5.12.3.6 TECT_ZONE_INFO_TYPE

No	Name	Data Type	Unit	Initial Value	Description	Access
1	rActual_Temp	REAL		0	Actual value from the Thermocouple.	RO
2	wErrors	WORD		0	First error word	RO

No	Name	Data Type	Unit	Initial Value	Description	Access
3	wErrors2	WORD		0	Second error word	
4	byGroup_Index	BYTE		0	Group index	RO
5	byZone_Index	BYTE		0	Zone index	RO

1.5.12.3.7 TECT_CONTROLSTATEMACHINE_ENUM

The structure TECT_CONTROLSTATEMACHINE_ENUM enumeration gives the Control status of function block TECT_TEMP_CONTROL State machine action status.

No	Name	Description
1	TECT_NotReady	State machine is not ready.
2	TECT_Ready	State machine is in Ready state.
3	TECT_Manual	State machine is in Manual mode.
4	TECT_AutoTune	Automatic mode in PID process with standby setpoint - Heating.
5	TECT_PID_Process	State machine PID process is running.
6	TECT_FaultState	State machine is in Fault state.
7	TECT_MixedState	State machine is in Mixed state.
8	DummyState	DummyState = 256, to ensure the TECT_CONTROLSTATEMACHINE_ENUM has two bytes.

1.5.12.3.8 TECT_ERRORCODE_ENUM

The structure TECT_ERRORCODE_ENUM enumeration is to save and display all the special errors and warnings.

No	Name	Description
1	TECT_NoError	No Error
2	TECT_TuneFault	Bit 0: TuneFault: Tuning failed - Outputs disabled, check TECT_TEMPZONE_DATA_TYPE ↳ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*
3	TECT_TC_Fault_1	Bit 1: TC Fault_1: Bad thermocouple reading - outputs disabled.
4	TECT_TC_Fault_2	Bit 2: TC Fault_2: Plausability check not passed - outputs disabled.
5	TECT_HighHighTempFault	Bit 3: HighHighTempFault: HighHigh Temperature alarm - outputs disabled.
6	TECT_LowLowTempFault	Bit 4: LowLowTempFault: LowLow Temperature alarm - outputs disabled.
7	TECT_HighTemp	Bit 7: HighTemp: High Temperature alarm.
8	TECT_LowTemp	Bit 8: LowTemp: Low Temperature alarm.

No	Name	Description
9	TECT_HighDeviation	Bit 9: HighDeviation: High Deviation alarm
10	TECT_LowDeviation	Bit 10: LowDeviation: Low Deviation alarm.
11	TECT_NoHighHighLowLow	Bit 11: NoHighHighLowLow: No HighHighTemp/ LowLowTemp limit is defined.
12	TECT_WrongLimits	Bit 12: WrongLimits is configured, HighTemp/Low temp and HighHighTemp/LowLowTemp values are not plausible.
13	Reserved	Bit 13: Reserved
14	TECT_NoAutoTune	Bit 14: NoAutoTune: Auto tune cannot be started.
15	TECT_NoPIDProcess	Bit 15: NoPIDProcess: PID process cannot be started or parameters KP, TN, TV, TD are not valid.
16	TECT_ShortCircuit	Bit 16: Short Circuit detected in the heater power circuit.
17	TECT_OpenCircuit	Bit 17: Open Circuit detected in the heater power circuit.
18	TECT_ControlCircuitFailure	Bit 18: Heater power Control Circuit failure detected.
19	TECT_NoRatedCurrent	Bit 19: No valid rated current defined for current monitoring

1.5.12.3.9 TECT_FILE_ENUM

The structure TECT_FILE_ENUM enumeration will give the current status of TECT_LOG_FILE function block operation.

Value	Name	Description
0	TECT_NO_OPERATION	No operation is active in TECT_LOG_FILE function block.
5	TECT_CREATE_SUBFOLDER	Subfolder creation is active in TECT_LOG_FILE function block.
20	TECT_WAIT FOR TOKEN	Waiting for token in TECT_LOG_FILE function block.
40	TECT_OPEN_FILE	Opening file is active in TECT_LOG_FILE function block.
45	TECT_GET_POSITION	Get position is active in TECT_LOG_FILE function block.
50	TECT_WRITE_HEADER	Writing header is active in TECT_LOG_FILE function block.
60	TECT_WRITE_ENTRY	Writing entry is active in TECT_LOG_FILE function block.
80	TECT_CLOSE_FILE	Close file is active in TECT_LOG_FILE function block.
90	TECT_CREATE_BACKUP	Creating backup is active in TECT_LOG_FILE function block.
100	TECT_RENAME_FILE	Renaming is active in TECT_LOG_FILE function block.

Value	Name	Description
105	TECT_RETRY_RENAME	Retrying rename is active in TECT_LOG_FILE function block.
200	TECT_RELEASE_TOKEN	Releasing token is active in TECT_LOG_FILE function block.
- 5	TECT_CREATE_FOLDER	Folder creation is active in TECT_LOG_FILE function block. (*<0: creating FOLDER, not for saving log data state.)

1.5.12.3.10 TECT_OPERATION_ENUM

The structure TECT_OPERATION_ENUM enumeration is to configure operations in function block TECT_DATA_FLASH and function block TECT_RECIPE.

Value	Name	Description
1	TECT_READ	Read data from RECIPE/DATA_FLASH to TECT_TEMPZONE_DATA_TYPE. ↯ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*
2	TECT_WRITE	Write data to RECIPE/DATA_FLASH from TECT_TEMPZONE_DATA_TYPE. ↯ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*
3	TECT_DELETE	Delete data written into DATA_FLASH.
4	TECT_CHECK_READ	Check, if there are already valid parameters (KS > 0) in TECT_TEMPZONE_DATA_TYPE. ↯ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343* Read data only from DATA_FLASH, if KS ≤ 0.
5	TECT_DELETE_WRITE	First delete the whole flash segment. Then write data from TECT_TEMPZONE_DATA_TYPE to flash. ↯ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343* It is used to avoid error 4101 by WRITE: Block was already written.
6	TECT_CHECK_WRITE	First read data from flash to TECT_TEMPZONE_DATA_TYPE, if KS ≤ 0. ↯ *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343* Then delete the flash. It is a combination of CHECK_READ and DELETE_WRITE.

1.5.12.3.11 TECT_OUTPUTSTATUS_ENUM

The structure TECT_OUTPUTSTATUS_ENUM enumeration will give the status of the PWM output.

Value	Name	Description
0	TECT_Disabled	Process disabled.
1	TECT_AutoHeat	Automatic mode in PID process with setpoint: Heating.
2	TECT_AutoCool	Automatic mode in PID process with setpoint: Cooling.
3	TECT_AutoHeat Standby	Automatic mode in PID process with standby setpoint: Heating.

Value	Name	Description
4	TECT_AutoCool Standby	Automatic mode in PID process with standby setpoint: Cooling.
5	TECT_ManualHeat	Manual mode: Heating
6	TECT_ManualCool	Manual mode: Cooling
7	TECT_TuneActive	AutoTune is active. Heating with tune_output.
8	TECT_TuneComplete	AutoTune is complete. Waiting for accept of AutoTune results. No PWM output.
9	TECT_Inactive	Duty cycle = 0. No PWM output.
10	TECT_NoOutput	NoOutput: Monitor Only
11	TECT_CoMonitorActive	Common sensor current monitoring is active
12	TECT_CoMonitorWait	Common sensor current monitoring is in waiting mode
13	TECT_IndiviMonitor	Individual sensor current monitoring is active
256	TECT_DummyStatus	DummyStatus := 256 to ensure the TECT_OUTPUT-STATUS_ENUM has two bytes

1.5.12.4 Visualization

1.5.12.4.1 Visualization

Visualization gives an opportunity to the user to monitor and control function blocks. Some template are also created for ease of operation.

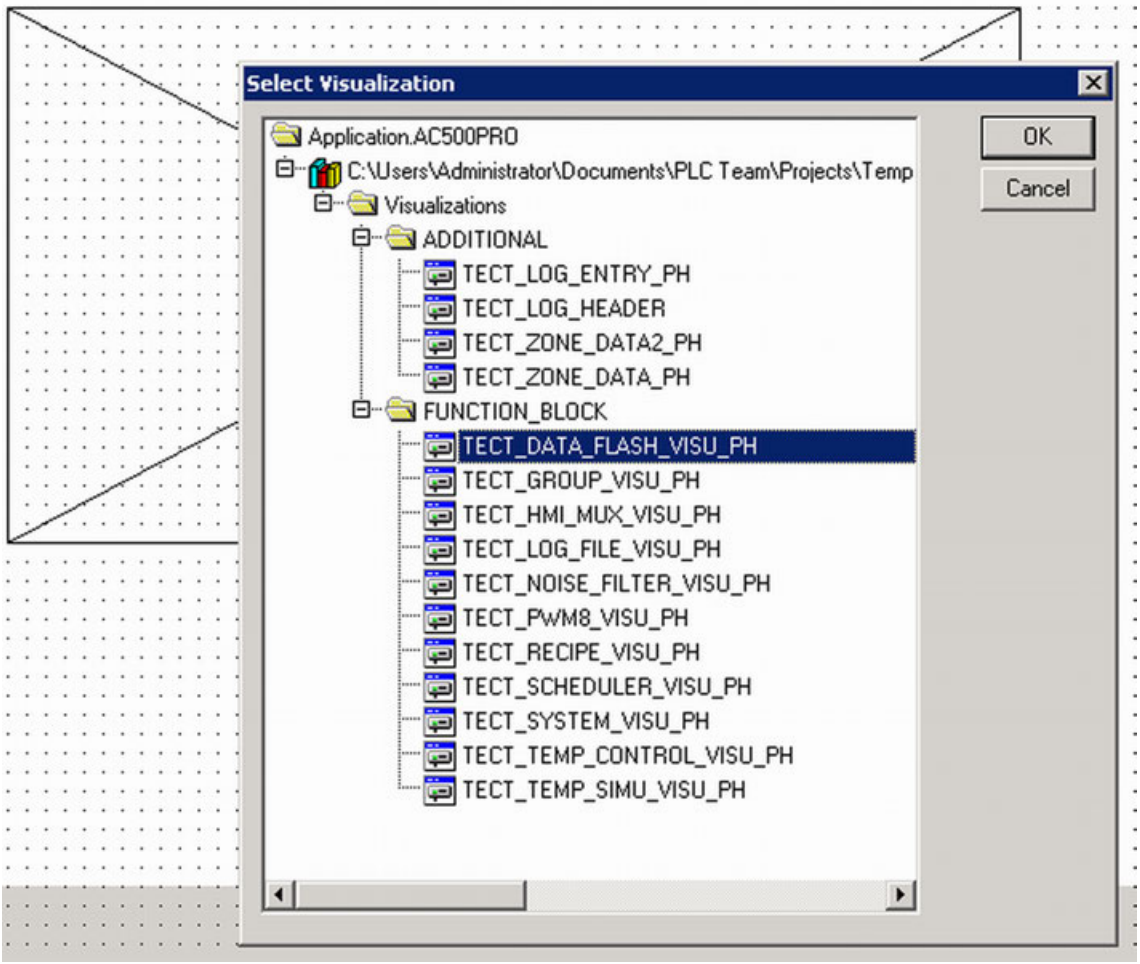


Fig. 603: TECT visualization list

TECT_ZONE_DATA_PH

TECT_ZONE_DATA_PH can be used as template for zone data. The most important values of a zone data during the control process can be monitored out of here. Each instance of this visualization has to be attached with respective zones to read and write the parameters to the particular zone. In the placeholder attach corresponding array element of the variable declared for structure TECT_TEMPZONE_DATA_TYPE. *Chapter 1.5.12.3.1 “TECT_TEMPZONE_DATA_TYPE” on page 3343*

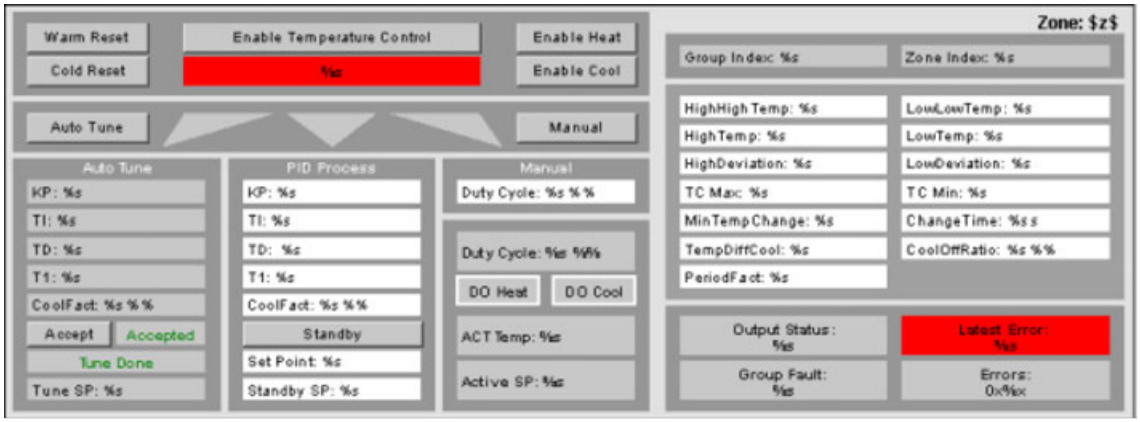


Fig. 604: TECT_ZONE_DATA_PH template in offline mode

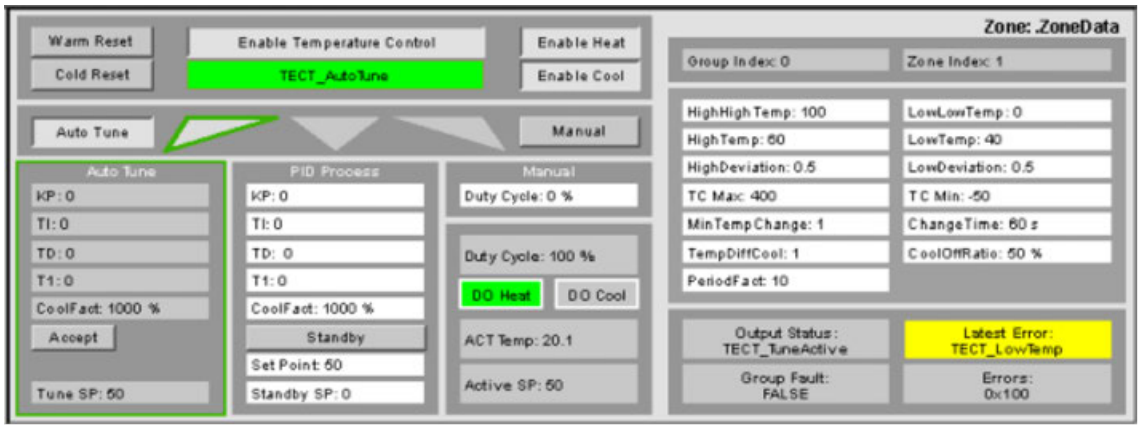


Fig. 605: TECT_ZONE_DATA_PH template in online mode

TECT_LOG_ENTRY_PH

The items of data TECT_LOG_ENTRY_PH array in logging buffer can be displayed in CODESYS Visualization, using dedicated visualization template. The following example shows how to realize it.

Visualization example of logging buffer

Definition of TECT_LOG_ENTRY_PH array

1. Define an array of TECT_LOG_ENTRY_PH with the size in number of entries (NUM_VISU_ENTRY = 10) to be displayed.

   ptsLog_Visu: ARRAY[1...10] of TECT_LOGINFO_TYPE ↳ *Chapter 1.5.12.3.3 “TECT_LOGINFO_TYPE” on page 3351*

   Visu_array: ARRAY[1...10] of Visu_LogEntry

VISU_BUFFER

2. Data type: POINTER TO TECT_LOGINFO_TYPE

   Start address of the log visualization data. It is recommended to define the log visualization data in array. This can be used only with CODESYS visualization. For example, if there are 4 zones to be logged and each zone has 10 log visualization entries, then a one dimensional or two dimensional array can be defined (see TECT_LOGINFO_TYPE: ↳ *Chapter 1.5.12.3.3 “TECT_LOGINFO_TYPE” on page 3351*

   visu_buffer: ARRAY[1..40] OF TECT_LOGINFO_TYPE; (in one dimensional array).

   visu_buffer: ARRAY[1..4, 1..10] OF TECT_LOGINFO_TYPE; (in two dimensional array).

NUM_VISU_ENTRY

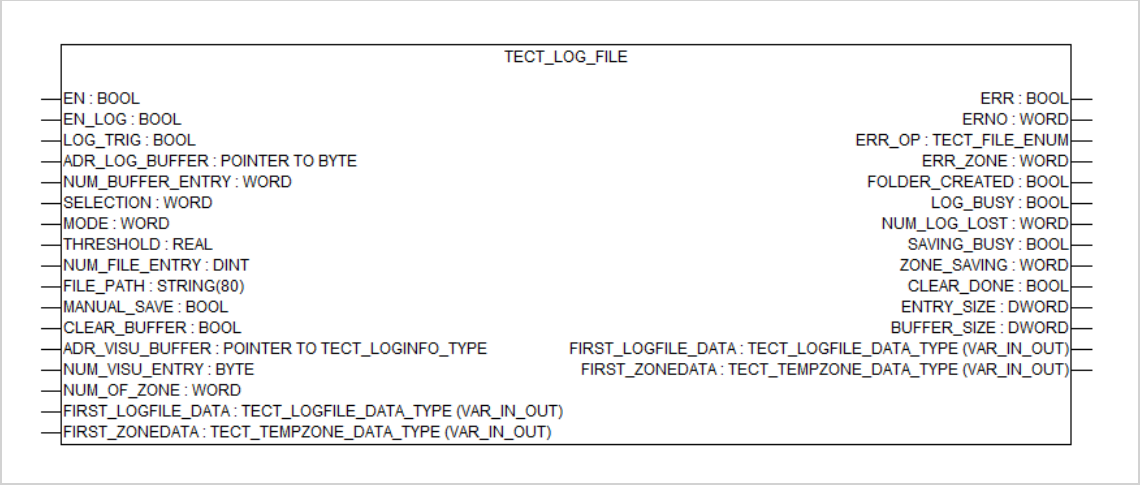
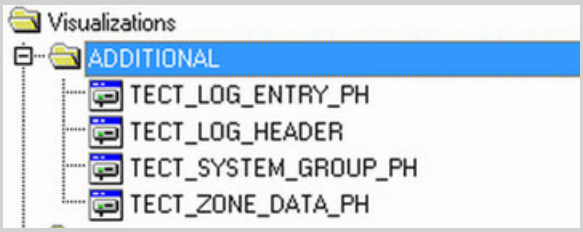


Fig. 606: FB_TECT_LOG_FILE 2

3. Data type: BYTE, Default value: 10

   Number of log entries displayed in log visualization for one zone.

Creation of VISU element of type TECT_LOG_HEADER



4. Create a VISU element of type TECT_LOG_HEADER.

Creation of VISU element of type TECT_LOG_ENTRY_PH

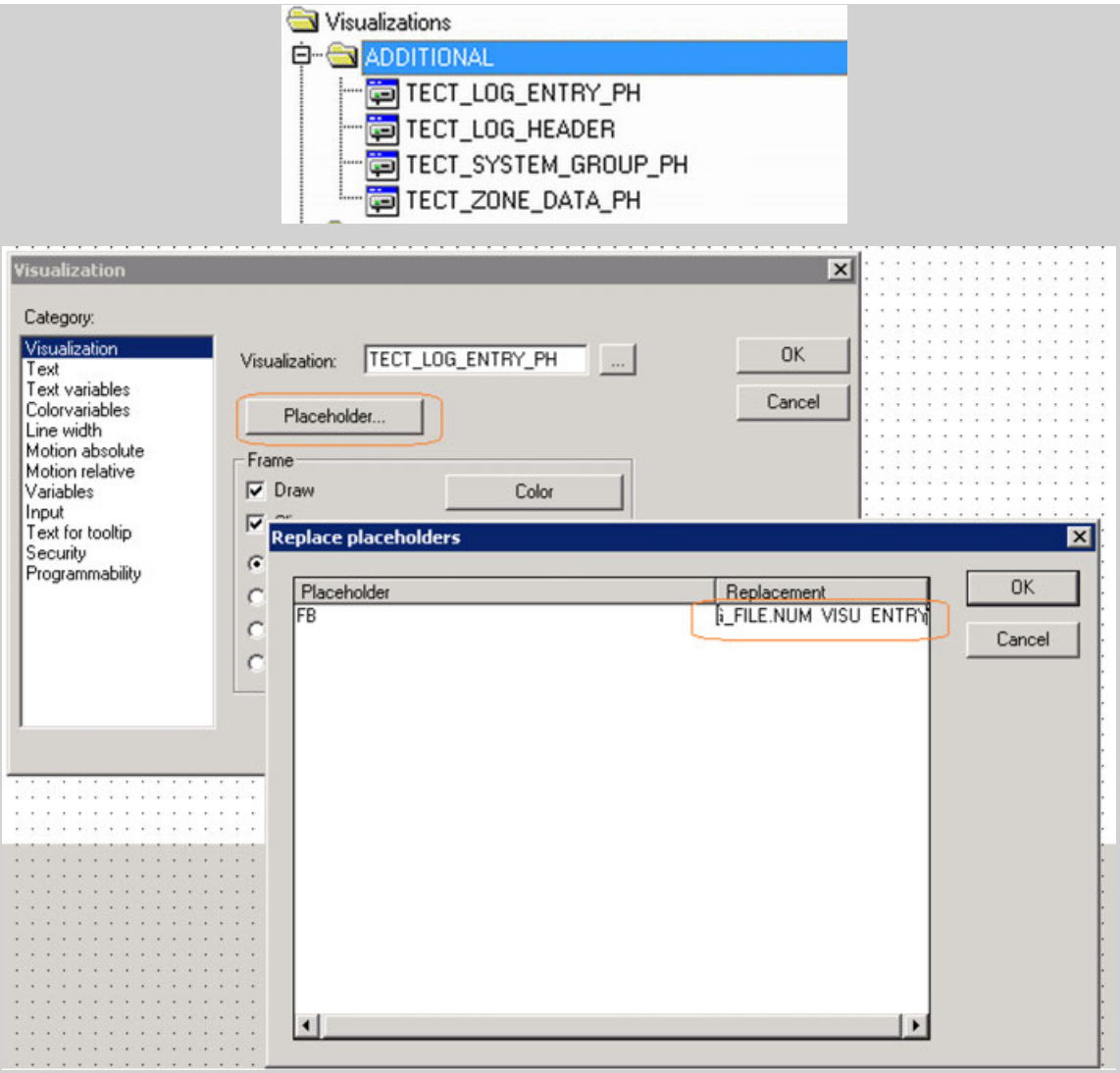


Fig. 607: TECT_LOG_ENTRY_PH placeholder

Date and Time	CW	SW	SET T	ACT T	Duty Cycle	Control State	Output Status	Error Status
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s

5. Create a VISU element of type TECT_LOG_ENTRY_PH. Set the Visu_buffer [1] as replacement for the VISU placeholder.

   Resize the VISU element as the same size of TECT_LOG_HEADER.

Creation of more VISU elements of type TECT_LOG_ENTRY_PH

6. Create more 9 VISU elements of type TECT_LOG_ENTRY_PH (or create 9 copies of the first TECT_LOG_ENTRY_PH item).

   Set the visu_buffer [2] to visu_buffer [10] as replacement for the placeholders to the 9 VISU items.

Organization of VISU items

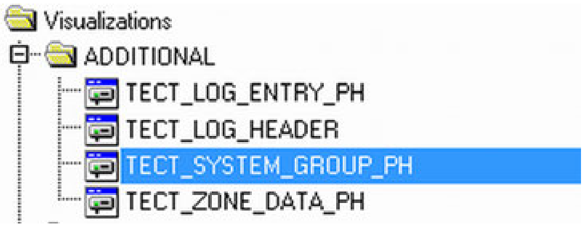
Date and Time	CW	SW	SET T	ACT T	Duty Cycle	Control State	Output Status	Error Status
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s
%s	0x%x	0x%x	%s	%s	%s	%s	%s	%s

Fig. 608: TECT_LOG_ENTRY_PH order place

7. Place the 10 VISU items in an ascending order (the oldest entry Visu_array[1] on the top) or a descending order (the newest entry Visu_array[10] on the top).

TECT_SYSTEM_GROUP_PH

TECT_SYSTEM_GROUP_PH can be used as template to control and monitor complete system.



Using this visualization user can enable or disable the process, start or stop autotune process, aceept the autotune parameters for the complete system. It also has option to Cold reset and Warm reset the process for complete system.

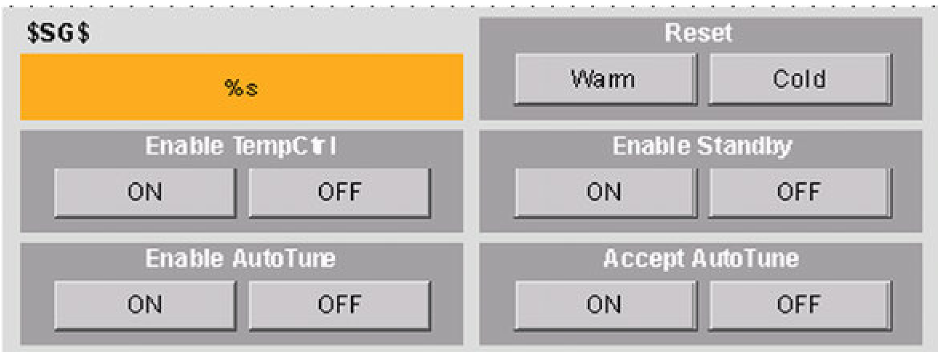


Fig. 609: TECT_SYSTEM_GROUP_PH in offline mode

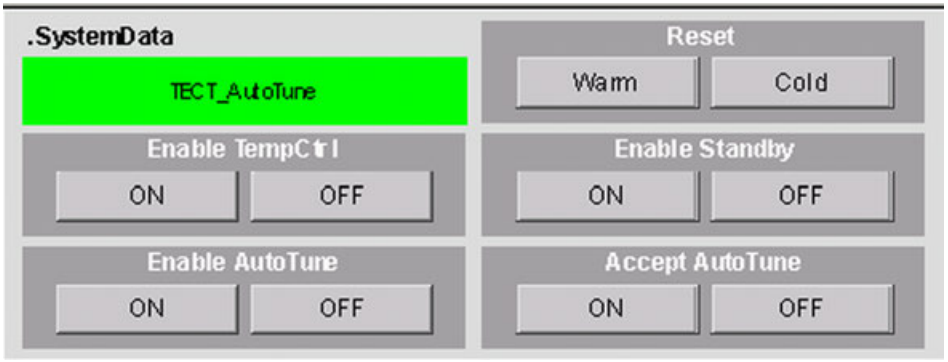


Fig. 610: TECT_SYSTEM_GROUP_PH in online mode

TECT_TEMP_CONTROL_VISU_PH

Description

Visualization element TECT_TEMP_CONTROL_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_TEMP_CONTROL_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program. The figures show the visualization in offline and online mode.

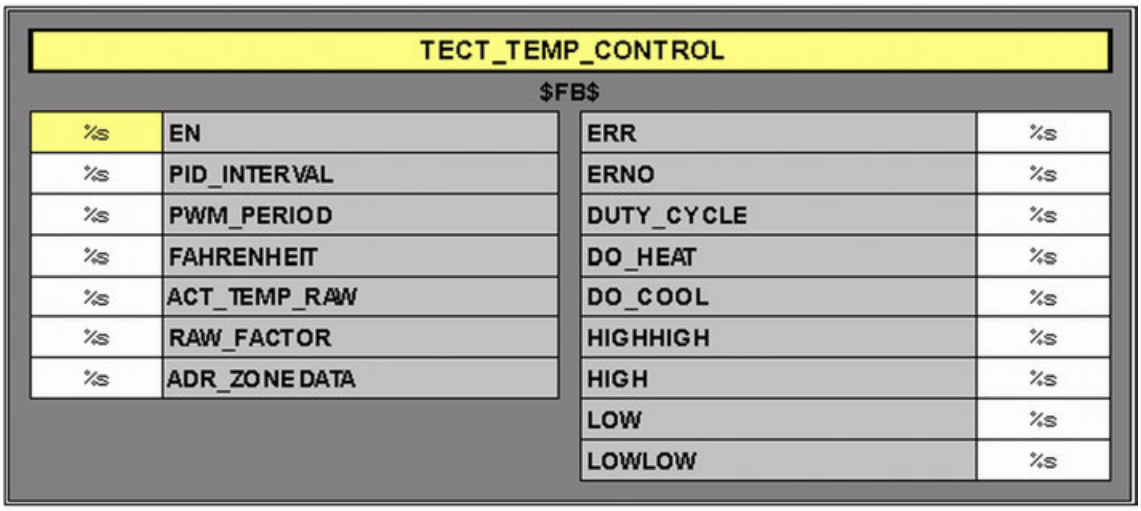


Fig. 611: TECT_TEMP_CONTROL_VISU_PH template in offline mode

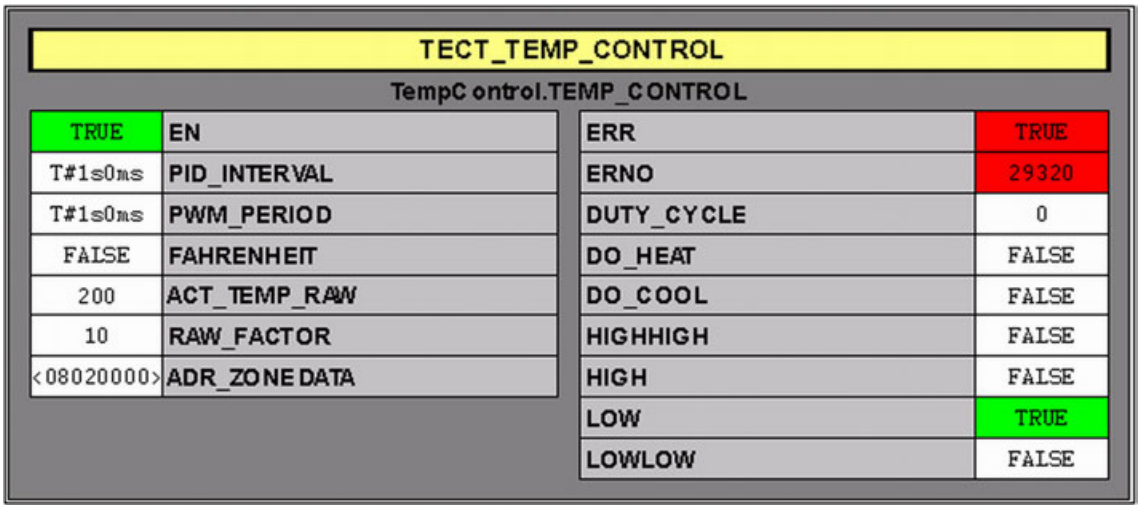


Fig. 612: TECT_TEMP_CONTROL_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
PID_INTERVAL	R/W	Keypad	Interval time for PID processing
PWM_PERIOD	R/W	Keypad	PWM Period. It should not be greater than PID_INTERVAL
FAHREN-HEIT	R/W	Toggle	Converts process temperature into Fahrenheit when TRUE
ACT_TEMP_RAW	R/W	Numpad	TC sensor input as actual temperature in raw value: Celsius with RAW_FACTOR
RAW_FACTOR	R/W	Numpad	Factor of ACT_TEMP_RAW to actual temperature in Celsius
ADR_ZONE-DATA	R		Pointer to Zone Data defined as TECT_TEMP-ZONE_DATA_TYPE
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
DUTY_CYCLE	R		Output duty cycle
DO_HEAT	R		Pulse for heating
DO_COOL	R		Pulse for cooling
HIGHHIGH	R		HighHigh alarm active
HIGH	R		High alarm active
LOW	R		Low alarm active
LOWLOW	R		LowLow alarm active

All inputs of TECT_TEMP_CONTROL function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTect-ControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_GROUP_VISU_PH

Description

Visualization element TECT_GROUP_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_GROUP_VISU_PH. The visualization is also used to control the function block by those inputs which are not connected inside the program.

The figures show the visualization in offline and online mode.

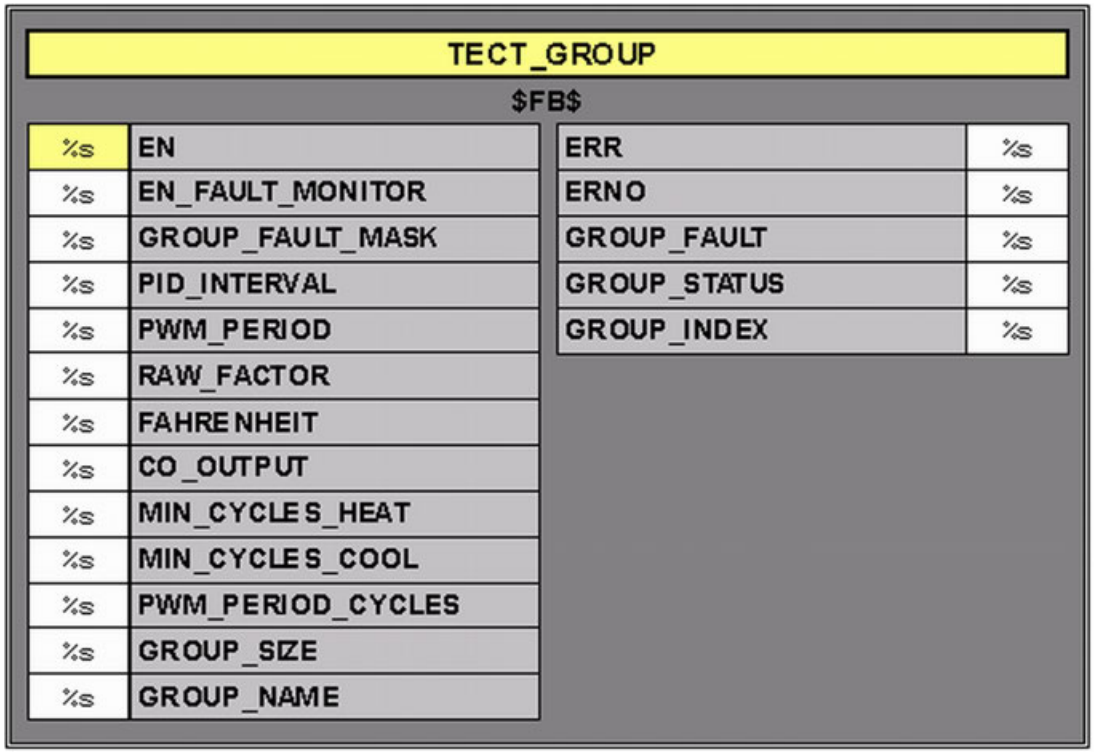


Fig. 613: TECT_GROUP_VISU_PH template in offline mode

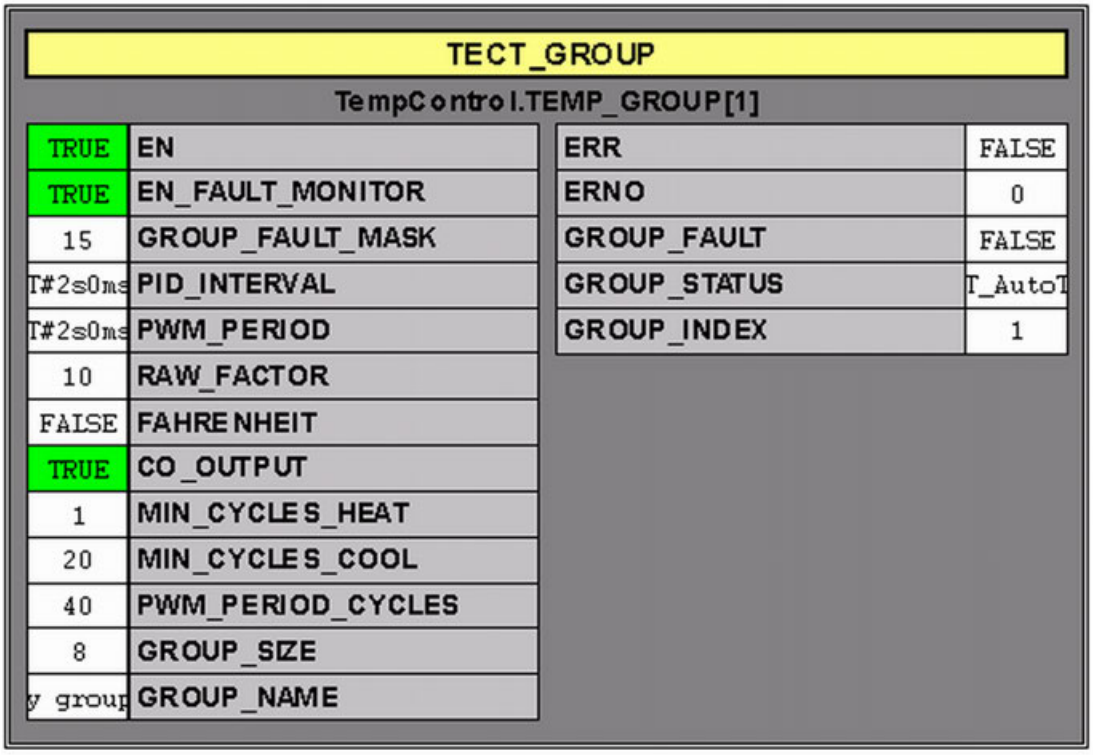


Fig. 614: TECT_GROUP_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
EN_FAULT_MONITOR	R/W	Toggle	It enables the Fault monitoring function when TRUE
GROUP_FAULT_MASK	R/W	Numpad	It enables masking of the Group Fault for the bits which are with value 1
PID_INTERVAL	R/W	Keypad	Interval time for PID processing
PWM_PERIOD	R/W	Keypad	PWM Period. It should not be greater than PID_INTERVAL
RAW_FACTOR	R/W	Numpad	Factor of ACT_TEMP_RAW to actual temperature in Celsius
FAHRENHEIT	R/W	Toggle	Converts process temperature into Fahrenheit when TRUE
ACT_TEMP_RAW	R/W	Numpad	TC sensor input as actual temperature in raw value: Celsius with RAW_FACTOR
CO_OUTPUT	R/W	Toggle	Enables coordinated output
MIN_CYCLES_HEAT	R/W	Numpad	ON or OFF Period should have a minimum length in number of CPU cycles
MIN_CYCLES_COOL	R/W	Numpad	ON or OFF Period should have a minimum length in number of CPU cycles
PWM_PERIOD_CYCLES	R/W	Numpad	Period duration referred to number of CPU cycles
GROUP_SIZE	R/W	Numpad	Size of the group
GROUP_NAME	R/W	Keypad	Name of the Group upto 20 characters
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
GROUP_FAULT	R		Group fault active
GROUP_STATUS	R		Status of the group
GROUP_INDEX	R		Group Index

All inputs of TECT_GROUP function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_SYSTEM_VISU_PH

Description

Visualization element TECT_SYSTEM_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_SYSTEM_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

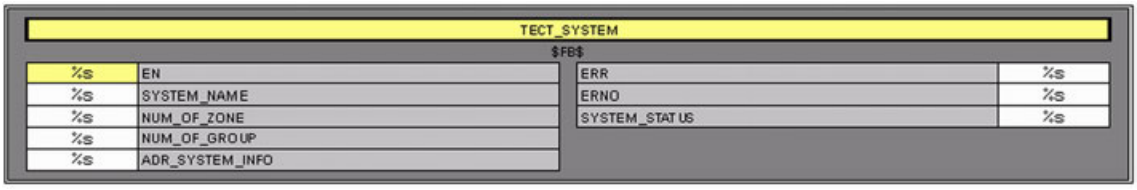


Fig. 615: TECT_SYSTEM_VISU_PH template in offline mode

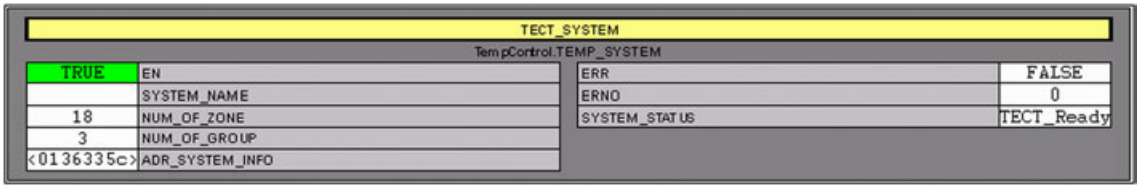


Fig. 616: TECT_SYSTEM_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
SYSTEM_NAME	R/W	Text	Name of the System upto 20 characters
NUM_OF_ZONE	R/W	Numpad	Number of zones in the system to be operated
NUM_OF_GROUP	R/W	Numpad	Interval time for PID processing
ADR_SYSTEM_INFO	R		Pointer to Zone info defined as TECT_ZONE_INFO_TYPE
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
SYSTEM_STATUS	R		System state machine status

All inputs of TECT_SYSTEM function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTect-ControlVisuBackgroundColor.

TECT_TEMP_SIMU_VISU_PH

Description

Visualization element TECT_TEMP_SIMU_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_TEMP_SIMU_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

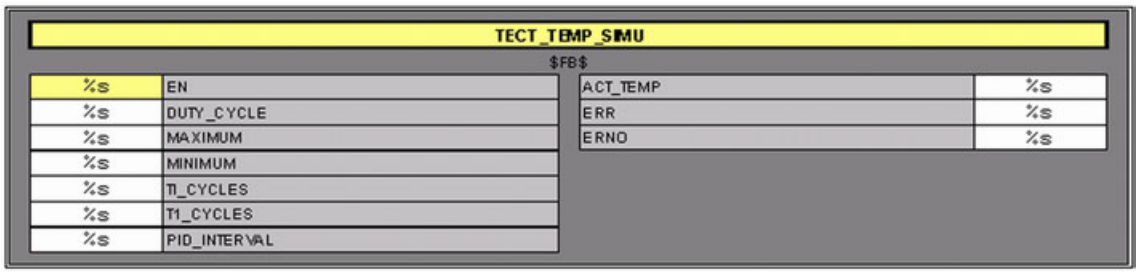


Fig. 617: TECT_TEMP_SIMU_VISU_PH template in offline mode

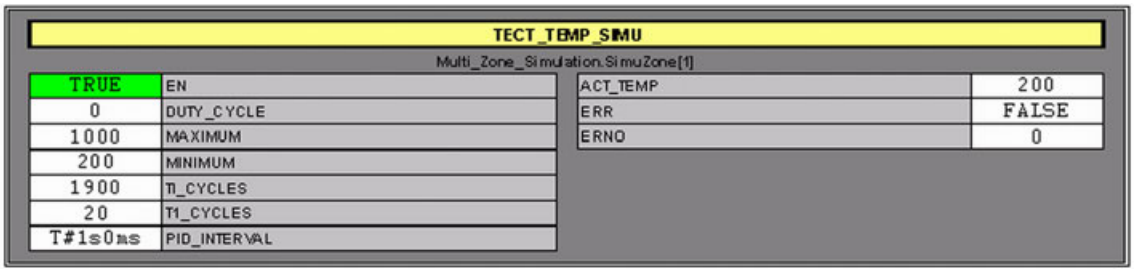


Fig. 618: TECT_TEMP_SIMU_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
DUTY_CYCLE	R/W	Numpad	PWM duty cycle for a temperature zone
MAXIMUM	R/W	Numpad	Maximum temperature allowed for the simulation
MINIMUM	R/W	Numpad	Minimum temperature allowed for the simulation
TI_CYCLES	R/W	Numpad	Internal control parameter for simulation
T1_CYCLES	R/W	Numpad	Internal control parameter for simulation
PID_INTERVAL	R/W	Keypad	Interval time for PID processing
ACT_TEMP	R		Simulated actual temperature
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes

All inputs of TECT_TEMP_SIMU function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_NOISE_FILTER_VISU_PH

Description

Visualization element TECT_NOISE_FILTER_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_NOISE_FILTER_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

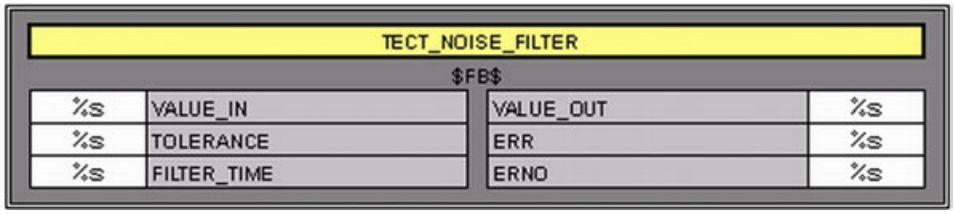


Fig. 619: TECT_NOISE_FILTER_VISU_PH template in offline mode

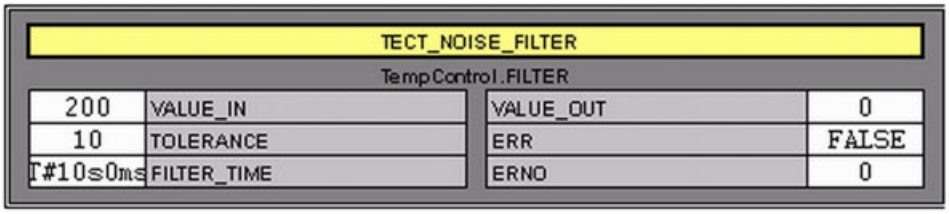


Fig. 620: TECT_NOISE_FILTER_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
VALUE_IN	R/W	Numpad	Input value to be filtered
TOLERANCE	R/W	Numpad	Tolerance for the input value
FILTER_TIME	R/W	Keypad	Filter time
VALUE_OUT	R		Output value after filtering
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes

All inputs of TECT_NOISE_FILTER function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_PWM8_VISU_PH

Description

Visualization element TECT_PWM8_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_PWM8_VISU_PH. The visualization is also used to control the function block by those inputs which are not connected inside the program.

The figures show the visualization in offline and online mode.

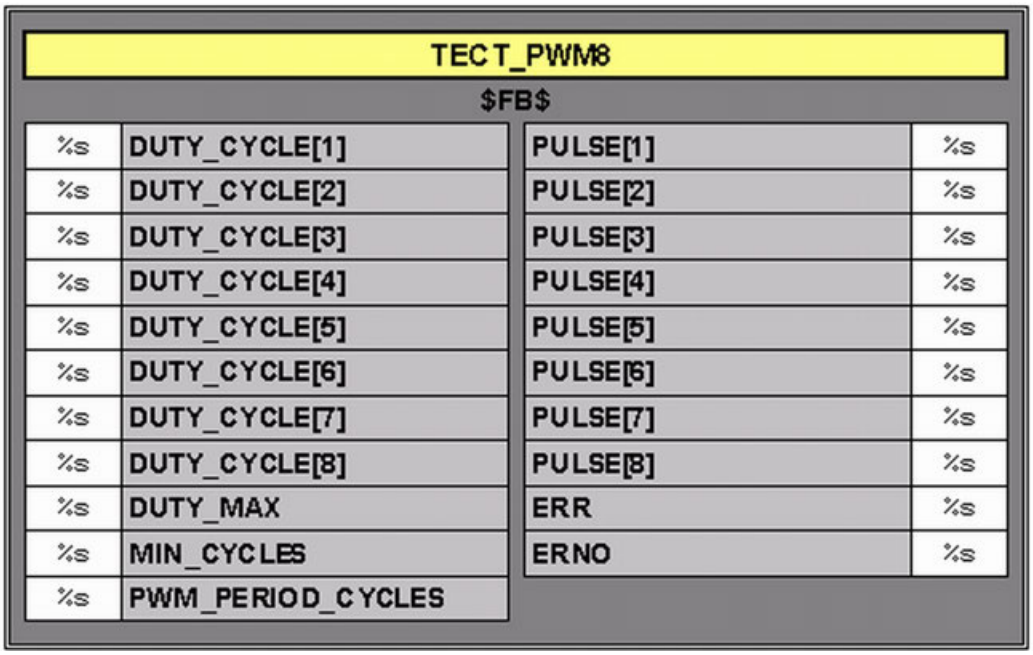


Fig. 621: TECT_PWM8_VISU_PH template in offline mode

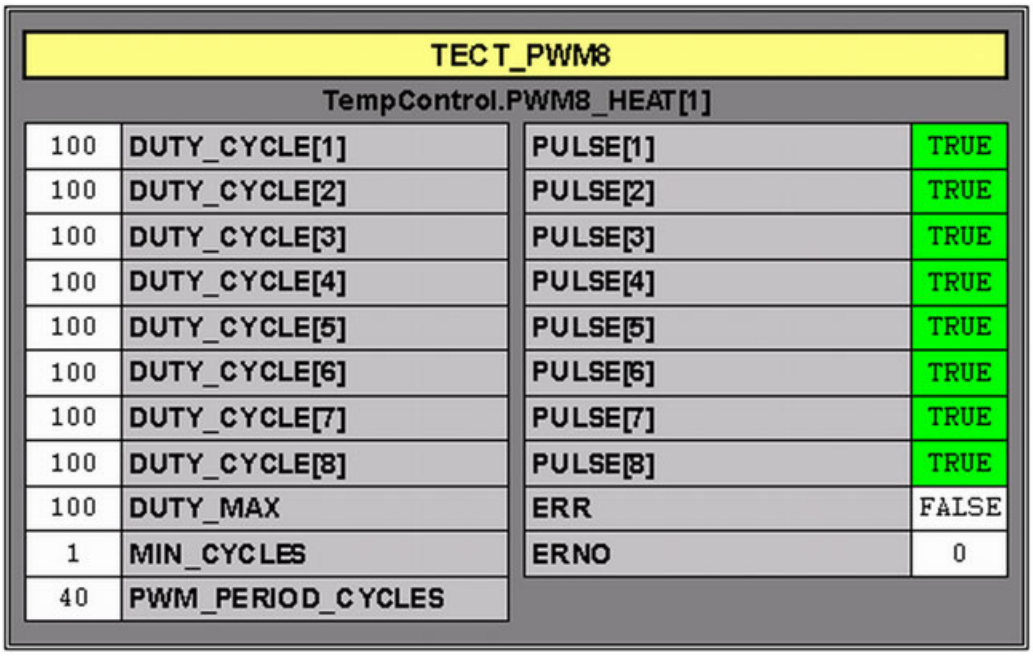


Fig. 622: TECT_PWM8_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
DUTY_CYCLE[1]	R/W	Numpad	PWM duty cycle for a first zone
DUTY_CYCLE[2]	R/W	Numpad	PWM duty cycle for a second zone
DUTY_CYCLE[3]	R/W	Numpad	PWM duty cycle for a third zone
DUTY_CYCLE[4]	R/W	Numpad	PWM duty cycle for a fourth zone
DUTY_CYCLE[5]	R/W	Numpad	PWM duty cycle for a fifth zone
DUTY_CYCLE[6]	R/W	Numpad	PWM duty cycle for a sixth zone
DUTY_CYCLE[7]	R/W	Numpad	PWM duty cycle for a seventh zone
DUTY_CYCLE[8]	R/W	Numpad	PWM duty cycle for a eighth zone
DUTY_MAX	R/W	Numpad	Max value for DUTY_CYCLE
MIN_CYCLES	R/W	Numpad	ON or OFF Period should have a minimum length in number of CPU cycles
PWM_PERIOD_CYCLES	R/W	Numpad	Internal control parameter for simulation
PULSE[1]	R		Pulse duration modulated signal of first zone
PULSE[2]	R		Pulse duration modulated signal of second zone
PULSE[3]	R		Pulse duration modulated signal of third zone
PULSE[4]	R		Pulse duration modulated signal of fourth zone
PULSE[5]	R		Pulse duration modulated signal of fifth zone
PULSE[6]	R		Pulse duration modulated signal of sixth zone
PULSE[7]	R		Pulse duration modulated signal of seventh zone
PULSE[8]	R		Pulse duration modulated signal of eighth zone
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes

All inputs of TECT_PWM8 function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_LOG_FILE_VISU_PH

Description

Visualization element TECT_LOG_FILE_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_LOG_FILE_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

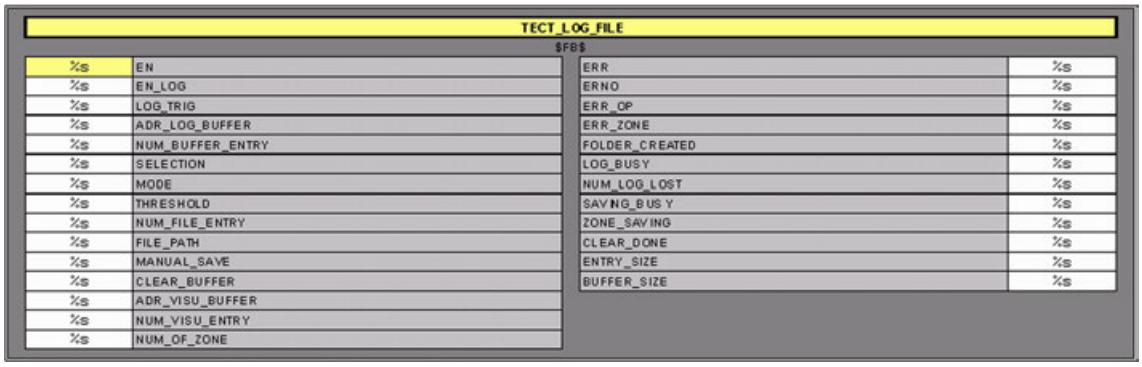


Fig. 623: TECT_LOG_FILE_VISU_PH template in offline mode

Fig. 624: TECT_LOG_FILE_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
EN_LOG	R/W	Toggle	Enable logger and save log buffer to file automatically
LOG_TRIG	R/W	Toggle	Execute log with rising edge
ADR_LOG_BUFFER	R		Start address of the logging buffer
NUM_BUFFER_ENTRY	R/W	Numpad	Size of logging buffer: number of entries
SELECTION	R/W	Numpad	Selection of predefined values to be logged
MODE	R/W	Numpad	To select the different modes for logging
THRESHOLD	R/W	Numpad	Value needs to be set in combination of MODE: MODE.Bit2-0 = 101
NUM_FILE_ENTRY	R/W	Numpad	Value needs to be set in combination of MODE: MODE.Bit7-6 = 01
FILE_PATH	R/W	Text	Folder for saving log file, should not be empty, one level will be created

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
MANUAL_SAVE	R/W	Toggle	Save log buffer to file manually
CLEAR_BUFFER	R/W	Toggle	Clear log buffer
ADR_VISU_BUFFER	R		First address of data type TECT_TYPLOGINFO_TYPE
NUM_VISU_ENTRY	R/W	Numpad	Number of log entries displayed in VISU
NUM_OF_ZONE	R/W	Numpad	Number of zones in the system to be operated
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
ERR_OP	R		Error operation, combined with ERR and ERNO
ERR_ZONE	R		Error zone, zone which causes the error
FOLDER_CREATED	R		Folder and subfolder created
LOG_BUSY	R		Logging entry in buffer is busy
NUM_LOG_LOST	R		Number of logs lost
SAVING_BUSY	R		Save logging buffer to file is busy
ZONE_SAVING	R		The zone of which the file operation is executing
CLEAR_DONE	R		Clear logging buffer is done
ENTRY_SIZE	R		Size of one log entry in byte, depending on SELECTION
BUFFER_SIZE	R		Size of whole buffer for all zones in byte, depending on ENTRY_SIZE, NUM_BUFFER_ENTRY and NUM_OF_ZONE

All inputs of TECT_LOG_FILE function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_RECIPE_VISU_PH

Description

Visualization element TECT_RECIPE_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_RECIPE_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

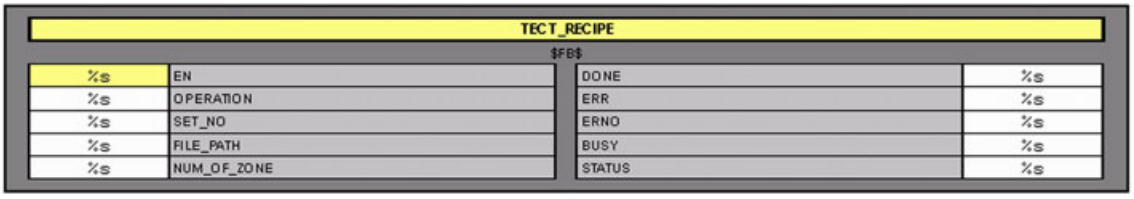


Fig. 625: TECT_RECIPE_VISU_PH template in offline mode

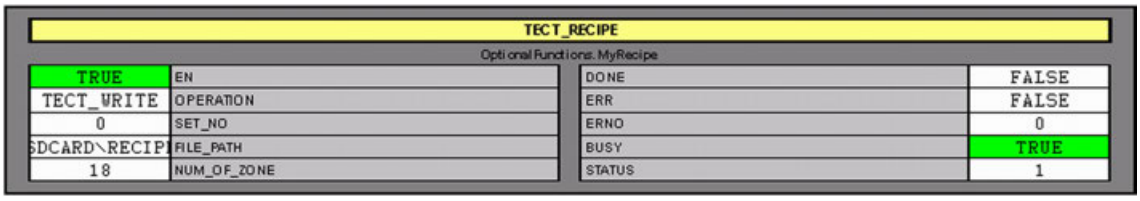


Fig. 626: TECT_RECIPE_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
OPERATION	R/W	Numpad	Operation type := 1: READ, 2: WRITE.
SET_NO	R/W	Numpad	Recipe set number: 0..65535
FILE_PATH	R/W	Text	1. Folder name: default file will be created. 2. folder+file name: user defined file will be created.
NUM_OF_ZONE	R/W	Numpad	Number of zones in the system to be operated
DONE	R		Execution finished, when output DONE = TRUE
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
BUSY	R		Recipe operation busy
STATUS	R		Status of the recipe block

All inputs of TECT_RECIPE function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTect-ControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_DATA_FLASH_VISU_PH

Description

Visualization element TECT_DATA_FLASH_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_DATA_FLASH_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program. The figures show the visualization in offline and online mode.

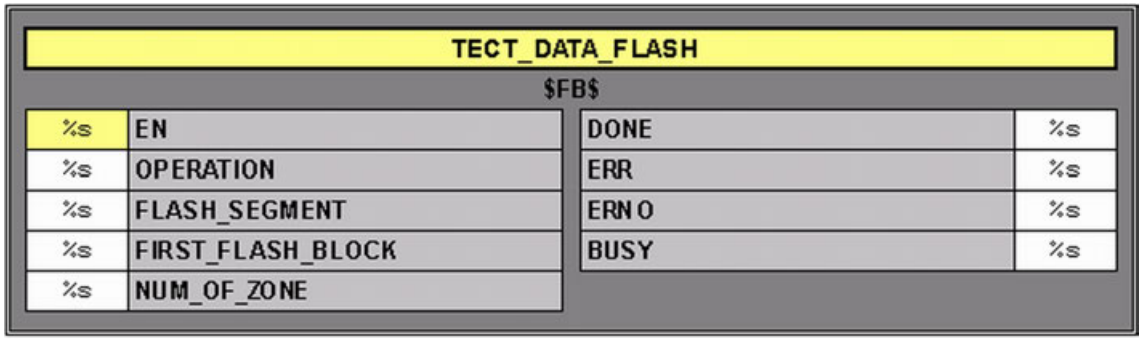


Fig. 627: TECT_DATA_FLASH_VISU_PH template in offline mode

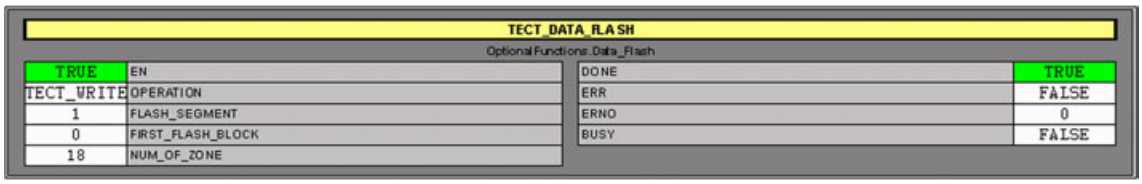


Fig. 628: TECT_DATA_FLASH_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
OPERATION	R/W	Numpad	Operation type := 1: READ, 2: WRITE.
FLASH_SEGMENT	R/W	Numpad	Number of the data segment; 1 or 2
FIRST_FLASH_BLOCK	R/W	Numpad	Number of the flash block within the data segment: 0...1926
NUM_OF_ZONE	R/W	Numpad	Number of zones in the system to be operated
DONE	R		Execution finished, when output DONE = TRUE
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes
BUSY	R		Recipe operation busy

All inputs of TECT_DATA_FLASH function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTectControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

TECT_HMI_MUX_VISU_PH

Description

Visualization element TECT_HMI_MUX_VISU_PH is used to show the actual values of all inputs and outputs of the instance of TECT_HMI_MUX_VISU_PH. The visualization is also used to control the function block by those inputs, which are not connected inside the program.

The figures show the visualization in offline and online mode.

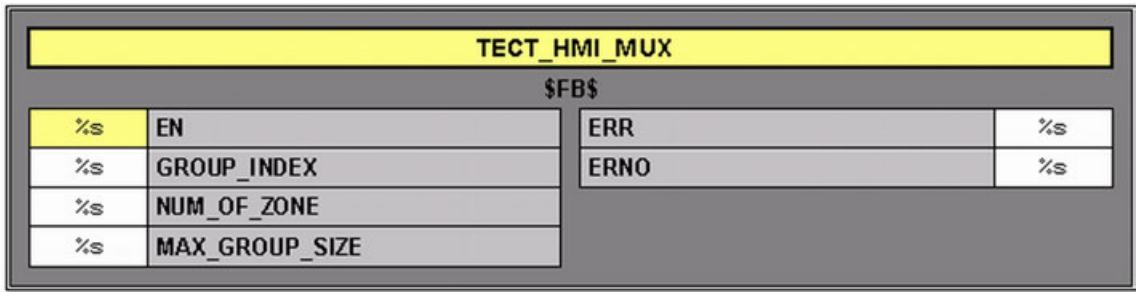


Fig. 629: TECT_HMI_MUX_VISU_PH template in offline mode

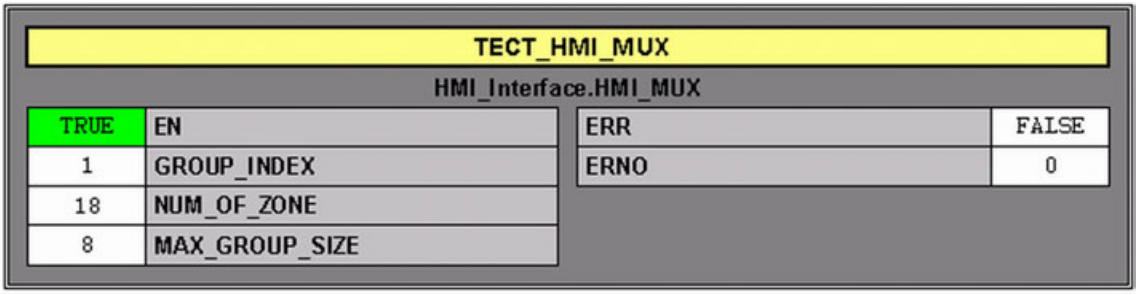


Fig. 630: TECT_HMI_MUX_VISU_PH template in online mode

Colors

The color of the variables has the following meaning:

- WHITE: Actual FALSE and should be FALSE in normal operation
- GREEN: Actual TRUE and should be TRUE in normal operation
- YELLOW: Actual FALSE, but should be TRUE in normal operation
- RED: Actual TRUE, but should be FALSE in normal operation

Visualization parameters

Variable Element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	Enable function block by TRUE level at input EN
GROUP_INDEX	R/W	Numpad	Index of active Group on HMI, it will be set from HMI and should be >0
NUM_OF_ZONE	R/W	Numpad	Number of zones in the system to be operated and should be >0
MAX_GROUP_SIZE	R/W	Numpad	Size of the largest group in the system and should be >0
ERR	R		Error occured during execution when output ERR = TRUE
ERNO	R		Error codes

All inputs of TECT_HMI_MUX function block, which are not connected to a variable (left open), can be written from this faceplate. The function block can be controlled from the visualization as long as the inputs are left open.

The color of the background can be changed by writing a value to the global variable dwTect-ControlVisuBackgroundColor. The color of the title can be changed by writing a value to the global variable dwTectControlVisuTitleColor.

1.5.12.5 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

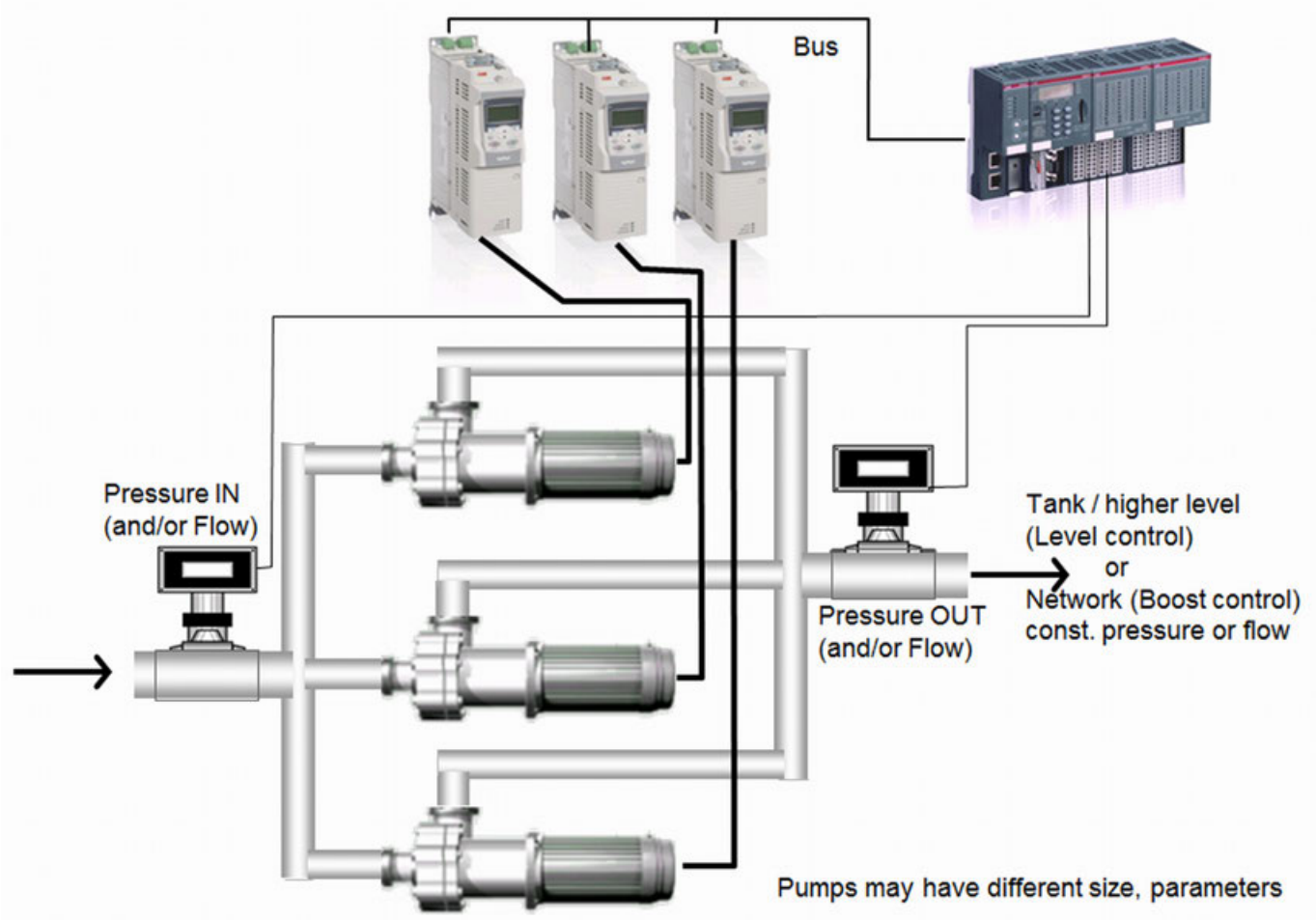
1.5.13 Water library

1.5.13.1 Pumping library

1.5.13.1.1 Overview

The Pumping Library is intended for smaller pump stations (or pump skids) and medium sized stations. It contains the block for the basic functionalities as well as advanced functions for very different applications.

This version named PUMP_AC500_V23.lib for Control Builder Plus V2.3 is usable for 1-3 pumps and is intended for configurations as it can be seen in the below figure It is best suited for the use of frequency converters/drives for each pump but also for the traditional setup with one drive and 1-2 switched outputs for the pumps.



The minimal sensor equipment for boost control applications is a pressure or flow sensor, for level control a level sensor. The pressure sensor can be at input and or/output, the flow sensor should be at the output. Often additionally also at the input a sensor, typically pressure, is placed for advanced protection and diagnosis.

The two main pumping applications in water and waste water are:

- Boost, typically pressure boost control: Used in network feeding, but also flow boost control for irrigation, water transport, cooling and washing applications.
- Level control: Used for tank or reservoir filling or emptying applications e.g. also lift applications e.g. in waste water networks.

For both applications a special application block is available, which has all the needed basic functionality, so that with minimal programming and mainly configuration an application can be setup.

Further auxiliary and optimization Function Blocks are part of the library, which help in putting additional advanced functionality with again minimal programming effort.

This function block type concept helps in always having the necessary functionality without using too much PLC resources (memory) and keeping the applications as simple as possible. Therefore the Pumping Library can be used throughout the AC500 Platform, starting from the small AC500-eCo.

Example

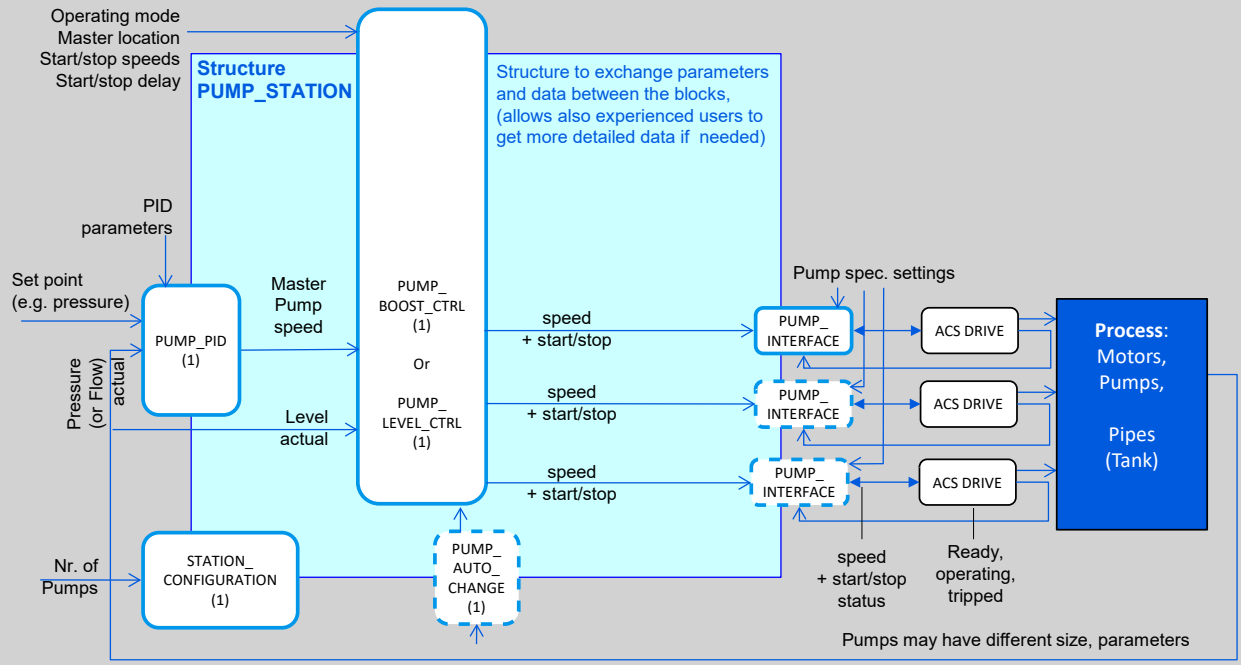


Fig. 631: A simple but typical application for small pump station or pump-skid (suitable e.g. also for AC500-eCo with 3 pumps)

→ Other specific parameters
(x) x shows how often a function block is used in a setup with 3 pumps
- - - Optional function blocks depending on the application

The station itself is configured by the function block STATION_CONFIGURATION with an ID, name and number of used pumps.

The main application Function Block is PUMP_BOOST_CTRL or PUMP_LEVEL_CTRL.

When using the function block PUMP_BOOST_CTRL: The actual pressure is read from the sensor and sent to the function block PUMP_PID. The PID corrections are given to the function block PUMP_BOOST_CTRL.

When using the PUMP_LEVEL_CTRL: The actual level is read from the sensor. It is not sent to PUMP_PID, it is sent directly to the function block PUMP_LEVEL_CTRL which does a fill or an empty operation.

The function block PUMP_INTERFACE is used for detailed configuration of each pump and for interfacing to the drives/actuators. The function blocks PUMP_INTERFACE allow a variety of actuators: One drive with direct-on-line "DOL" pumps, hard- or soft-switched; all drive controlled pumps; discrete or bus connection to the drives.

The number of function blocks PUMP_INTERFACE is depending on the number of pumps. 1 pump = 1 function block PUMP_INTERFACE, 2 pumps = 2 function blocks PUMP_INTERFACE etc.

The parameter and status exchange between the different blocks is done by a structured variable (symbolized by the light blue layer in the above figure which is connected to all blocks with control functionality. It contains substructures depending on the function and use.

This concept minimizes memory usage, the connection work and still allows advanced diagnosis if necessary.

Function blocks

Application function blocks

These are main application and control function blocks.

- The boost-control function block works with a closed loop control signal, typically coming from a PID controller with a pressure (or flow) sensor, see ↳ *Chapter 1.5.13.1.6 “PUMP_BOOST_CTRL Boost Control” on page 3404*. The PID gives a speed set-point for the boost pumping station. The boost control block then distributes start/stop and speed commands, depending on its chosen operating mode and parameters, to e.g. a multi-pump setup.

  The boost application is typically used with drives on all pumps and for:
  - Network feeding to control a pressure in the network over a wide range of flow, which is given by the varying demand in the network.
  - Irrigation, where either pressure or the flow is controlled to achieve a uniform and controlled irrigation.
  - Transport, cooling and washing applications (more flow control).
- The level-control function block works with discrete definable levels to switch the pumps and set appropriate fixed speeds, see ↳ *Chapter 1.5.13.1.8 “PUMP_LEVEL_CTRL Level Control” on page 3414*. Due to this fixed speed operation level control may also be used with the secondary pumps being switched.

Auxiliary function blocks

The auxiliary Function Blocks provide necessary additional functionality in a modular way to further minimize programming needs and mainly do configuration instead.

- PID control with extended functions for pumping applications with this library.
  ↳ *Chapter 1.5.13.1.7 “PUMP_PID” on page 3410*.
- Autochange functionality (ensures distribution of operation hours on the pumps).
  ↳ *Chapter 1.5.13.1.9 “PUMP_AUTOCHANGE” on page 3422*.
- Station configuration and pump interface function blocks to simplify configuration and interfacing to the drives.
  ↳ *Chapter 1.5.13.1.2 “PUMP_STATION_CFG” on page 3384*.
  ↳ *Chapter 1.5.13.1.3 “PUMP_INTERFACE” on page 3386*.
- The retain Function block stores selected values which have to sustain through power failures.
  ↳ *Chapter 1.5.13.1.4 “PUMP_RETAIN_DATA” on page 3391*.

Optimization function blocks

The optimization Function Blocks help in further optimizing the pumping station operation by providing additional information or functionality to save energy and protect the equipment (pumps, pipes, tanks).

- The Flow Calculation block estimates with the help of the drives measured power output and few pump curve data points of the PQ curve the actual flow, which can save a separate instrument and its integration work.
  Chapter 1.5.13.1.11 “PUMP_FLOW_CALC” on page 3430
- The Energy Calculation block totalizes the flows and energy consumptions and provides Efficiency values. If no flow meter is there, the Flow calculation block can be used.
  Chapter 1.5.13.1.14 “PUMP_ENERGY_CALC” on page 3450
- The Sleep function helps to save energy in PID operation if demand is low (e.g. at night in a network) and works closely together with PID and boost block.
  Chapter 1.5.13.1.12 “PUMP_SLEEP” on page 3435
- The Protection block provides advanced protections features for pump, pipe and tank based on the available information in the station.
  Chapter 1.5.13.1.13 “PUMP_PROTECTION” on page 3442
- The Anti-Jam block provides a configurable cleaning function per used pump with a drive.
  Chapter 1.5.13.1.10 “PUMP_ANTIJAM” on page 3426

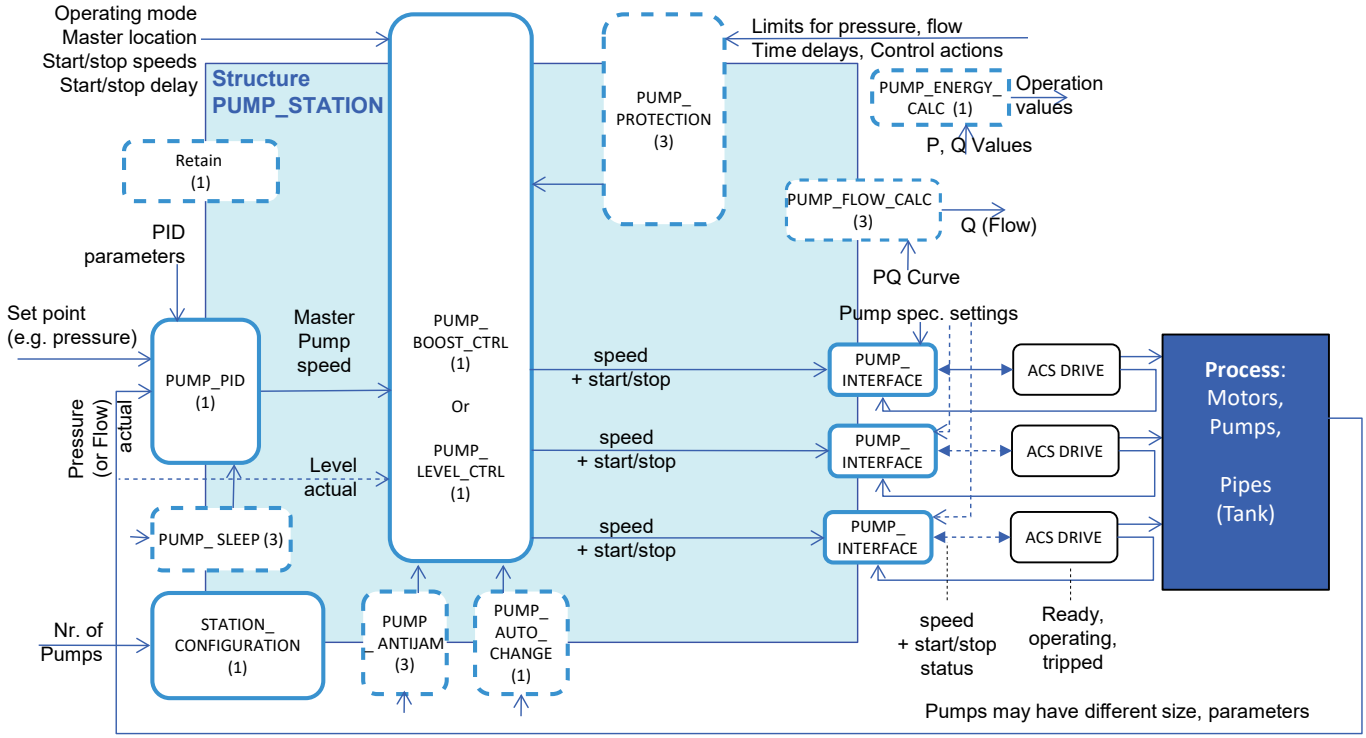


Fig. 632: Overview of the concept and its options with the Pumping Library, including auxiliary and optimization blocks.

There are further blocks provided enabling a simple simulation of the process (DOL, Drive, TANK) to try out an application program without hardware.

For further details please check the application examples and their documentation which are provided with the PS563-WATER library product package. Please check also if a more up to date version or comments exist as download on *www.abb.com/PLC*.

Preconditions for the use of the pumping library

Examples for this library can be found in C:\Users\Public\Documents\AutomationBuilder\Examples\PS563-WATER. Alternative is to go to *“Automation Builder ➔ Help ➔ Project Examples ➔ Examples”*.

The function blocks of the Pumping Library are only working in the RUN mode of the PLC. Usage of these libraries in the simulation mode may not provide any valid or usable diagnosis information.

For compatibility of the library PUMP_AC500_V23.lib please check the latest release notes.

1.5.13.1.2 PUMP_STATION_CFG

This function block must be called before all other function blocks of the library in the task configuration.

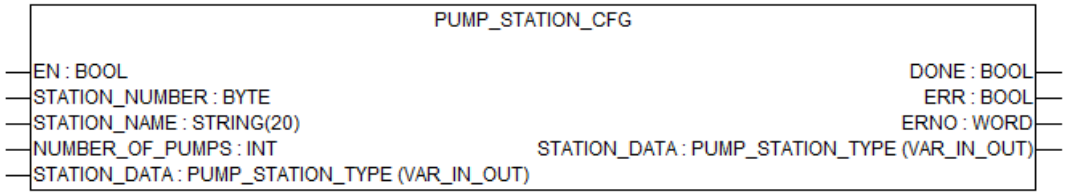


Fig. 633: Function block PUMP_STATION_CFG

Table 201: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block is used to configure a pump station. Number of pumps, station name and number is configured.

Input description

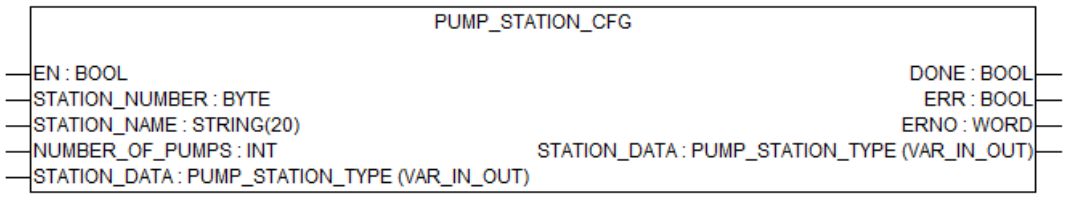


Fig. 634: Function block PUMP_STATION_CFG

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

STATION_NUMBER (station number)

Data type: BYTE, default value: 1, range: > 0

Number assigned to the station.

STATION_NAME (station name)

Data type: STRING

This input stores the station number, which is used as a unique identification number when there are more than one stations configured in a single PLC .

NUMBER_OF_PUMPS (number of pumps)

Data type: INT, default value: 1, range: 1-3

Total number of pumps used in the application.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

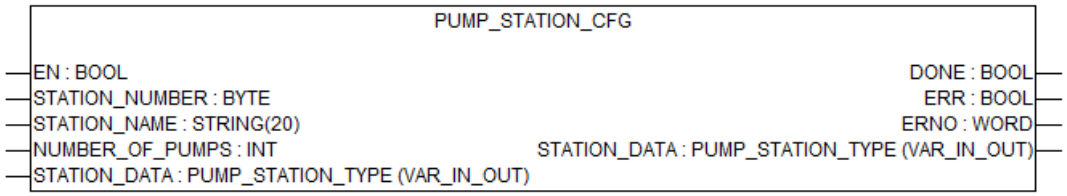


Fig. 635: Function block PUMP_STATION_CFG

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error
16416	4020	Value of STATION_NUMBER is erroneous
16448	4040	Value of NUMBER_OF_PUMPS is erroneous

1.5.13.1.3 PUMP_INTERFACE

All the instances of the function block PUMP_INTERFACE must be called after the function block PUMP_STATION_CFG and before all other function blocks of the pumping library.

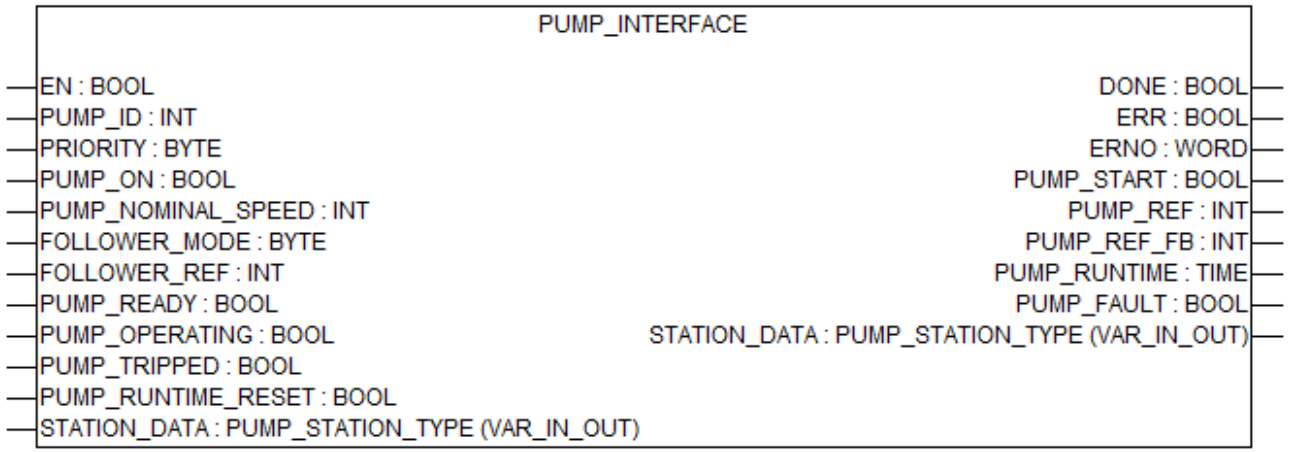


Fig. 636: Function block PUMP_INTERFACE

Table 202: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

Pump_Interface function block receives Pump ID, priority, follower mode and status of the drive and process this data. It gives out pump start,pump reference and pump actual runtime data. Follower drive behavior is defined using follower mode and follower reference and should be set to the same mode for all pumps of one pump station

1.The follower modes in each of the three function blocks PUMP_INTERFACE should be set to the same mode.

2. Each pump must have a unique PUMP_ID.

ACS Drives behavior:

- *When the ON command is withdrawn from the drive, AND it receives an ON command again (rising edge) before it has halted, the drive will not start. To restart the drive, the ON command needs to be made FALSE, let the drive come to halt and then made again TRUE.*

This case can occur by following ways:

- *Momentary power withdrawal from the drive, communication cable disturbance, or Pump_ON is made - FALSE then immediately TRUE again from the PUMP_INTERFACE e.g. during commissioning by manual operation in the engineering or by addition manual buttons around the application.*
- *This is not occurring in the normal operation. Nnevertheless it could be changed by selecting the drive on-the fly restart functionality, if deemed necessary and appropriate.*

Input description

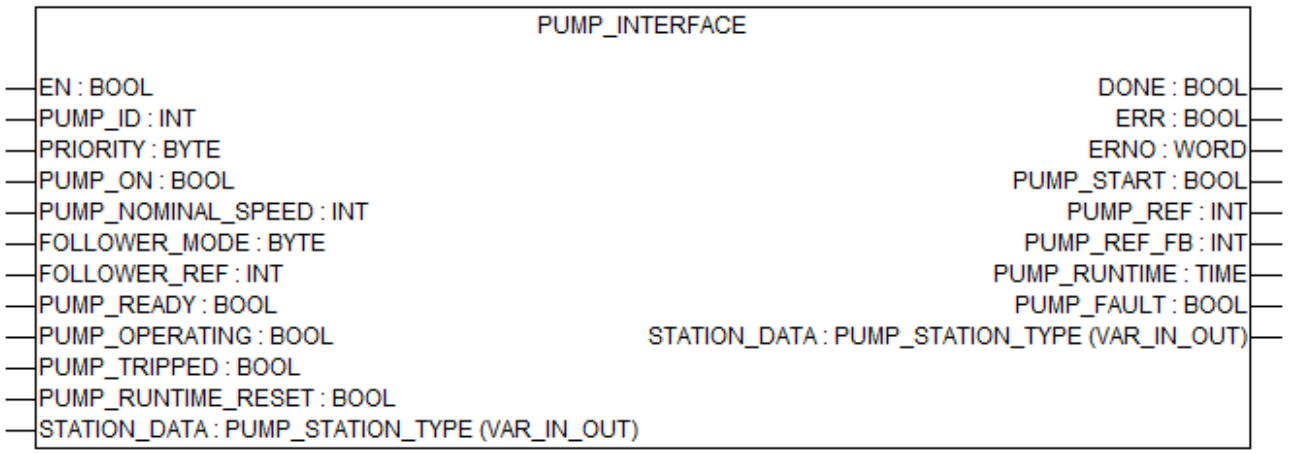


Fig. 637: Function block PUMP_INTERFACE

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID (pump ID)

Data type: INT

Pump identification number for which this function block to be called. Range of values: from 1 to 3 .

PRIORITY (priority)

Data type: BYTE

Pump Priority is used to decide, which drives need to be turn on demand increases.

PUMP_ON (pump on)

Data type: BOOL

This input turns the pump on or off running in boost control or level control. In case of a FALSE value, the user can remove the pump from the auto mode operations such as boost control or level control. Then the user can perform a manual task, such as manual anti-jam, or any other stand alone manual maintenance related job which actually are not part of the library. This input can also be configured using an external digital input signal.

PUMP_NOMINAL_SPEED (pump nominal speed in rpm)

Data type: INT

It stores the nominal value of the speed of the pump. This nominal value is put in the internal structure.

Example

STATION_DATA.atsPump[1].tsConfiguration.iPumpMaxSpeed

This is used to scale the field bus speed setpoint in the scale of -20000 to +20000. This data is also used in the flow calculations.

FOLLOWER_MODE (follower mode)

Data type: BYTE

Follower drives are started and stopped by the control logic. When flow demand increases , new pumps are started. Behavior of the follower pumps are defined by follower mode.

Follower mode	Speed	Explanation
0	Fixed constant speed	Follower starts at pre-defined fixed speed of master and runs at follower reference.
1	Copy master	Follower drives follows the same start/stop commands and speed reference like masterFig. 647.
2	Master speed	Follower starts at pre-defined fixed speed levels of master speed and then runs at the same master reference speed.

FOLLOWER_REF (follower reference)

Data type: INT

The speed reference in RPM for the pumps when the follower mode = 0 is selected.

PUMP_READY (pump ready)

Data type: BOOL

Input to attach the ready status of the drive of the pump.

PUMP_OPERATING (pump operating)

Data type: BOOL

Input to attach the operating status of the drive of the pump.

PUMP_TRIPPED (pump tripped)

Data type: BOOL

Input to attach the fault status of the drive of the pump.

PUMP_RUNTIME_RESET (pump runtime reset)

Data type: BOOL

TRUE value is used to reset the run time of the pump.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

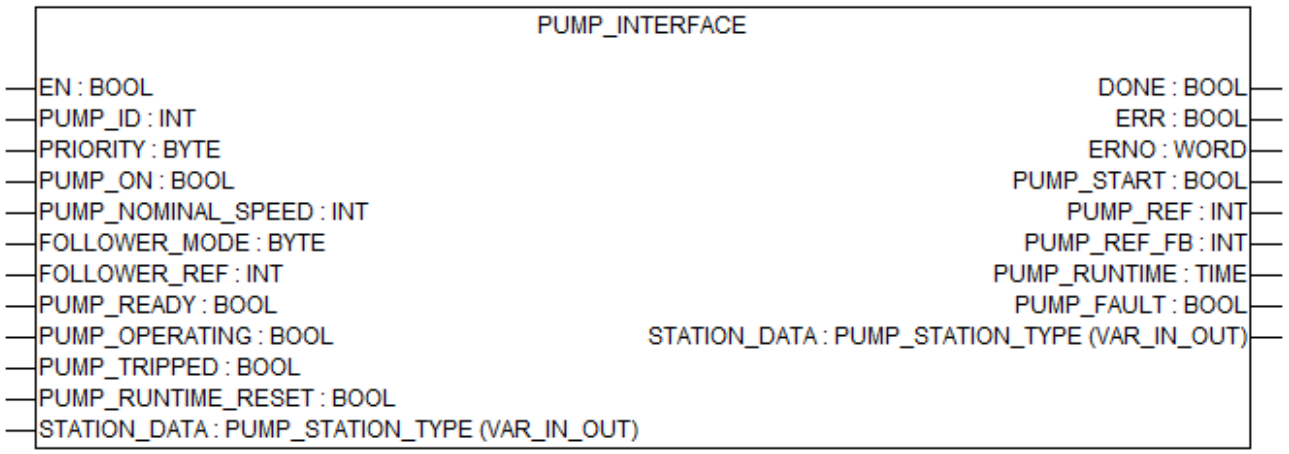


Fig. 638: Function block PUMP_INTERFACE

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

PUMP_START (pump start)

Data type: BOOL

On command to the pump.

PUMP_REF (pump reference)

Data type: INT

Speed reference to the pump in terms of RPM.

PUMP_REF_FB (pump reference field bus)

Data type: INT

Field bus speed reference to the pump in the range of -20000 to 20000.

PUMP_RUN-TIME (pump runtime)

Data type: TIME

Runtime value for the pump.

PUMP_FAULT (pump fault)

Data type: BOOL

TRUE when the pump encountered a fault. This fault can be because of drive tripped, pump protection fault, or due to PUMP_ON=FALSE when the boost control or the level control is running.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of PUMP_ID is erroneous.
16432	4030	Value of PRIORITY is erroneous.
16464	4050	Value of PUMP_NOMINAL_SPEED is erroneous.
16480	4060	Value of FOLLOWER_MODE is erroneous.
16483	4063	All pumps do not have same FOLLOWER_MODE.
16496	4070	Value of FOLLOWER_REF is erroneous.

1.5.13.1.4 PUMP_RETAIN_DATA

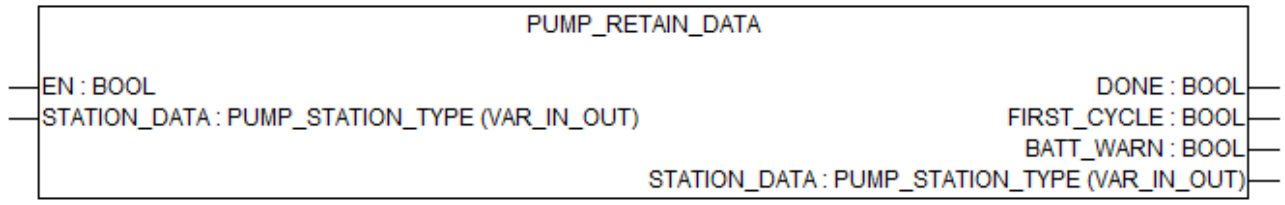


Fig. 639: Function block PUMP_RETAIN_DATA

General Information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block with historical values.

This function block is used to retain important data in the event of power failure or the PLC reboot. It stores the actual run time of the pump, last operating sequence of the pumps, total flow of the pump, total energy consumed.

Input description

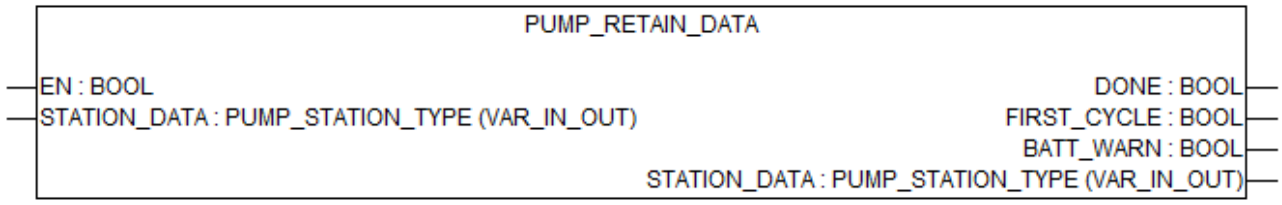


Fig. 640: Function block PUMP_RETAIN_DATA

The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

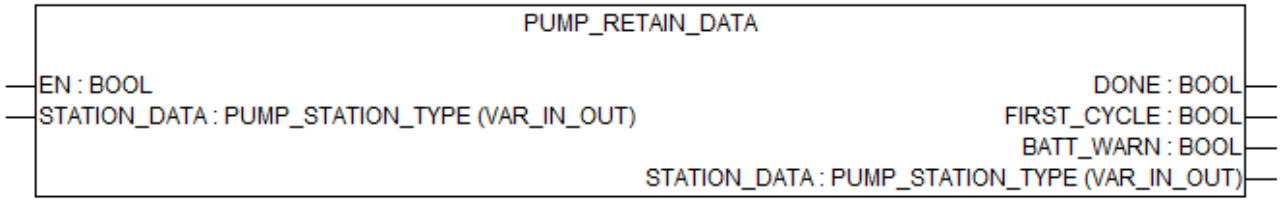


Fig. 641: Function block PUMP_RETAIN_DATA

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

FIRST_CYCLE (error)

Data type: BOOL

Indicates FIRST_CYCLE after download, reset, repower or stop running - set for only the FIRST_CYCLE.

BAT-TERY_WARN (battery warning)

Data type: BOOL

TRUE when battery is not inserted or less than minimum load reached.

1.5.13.1.5 STRUCTURE (PUMP_STATION_TYPE)

This structure is used to transfer and receive data in all function blocks. This is used as an in-out parameter to all function blocks. Every function block receives data, process it and modifies data of this structure. This structure is divided in substructure as per functions of the data. This structure basically stores values and parameters which are coming from the different function blocks of the library. Therefore these might be explained in more detail at the blocks where this parameters are entered or where the values are created.

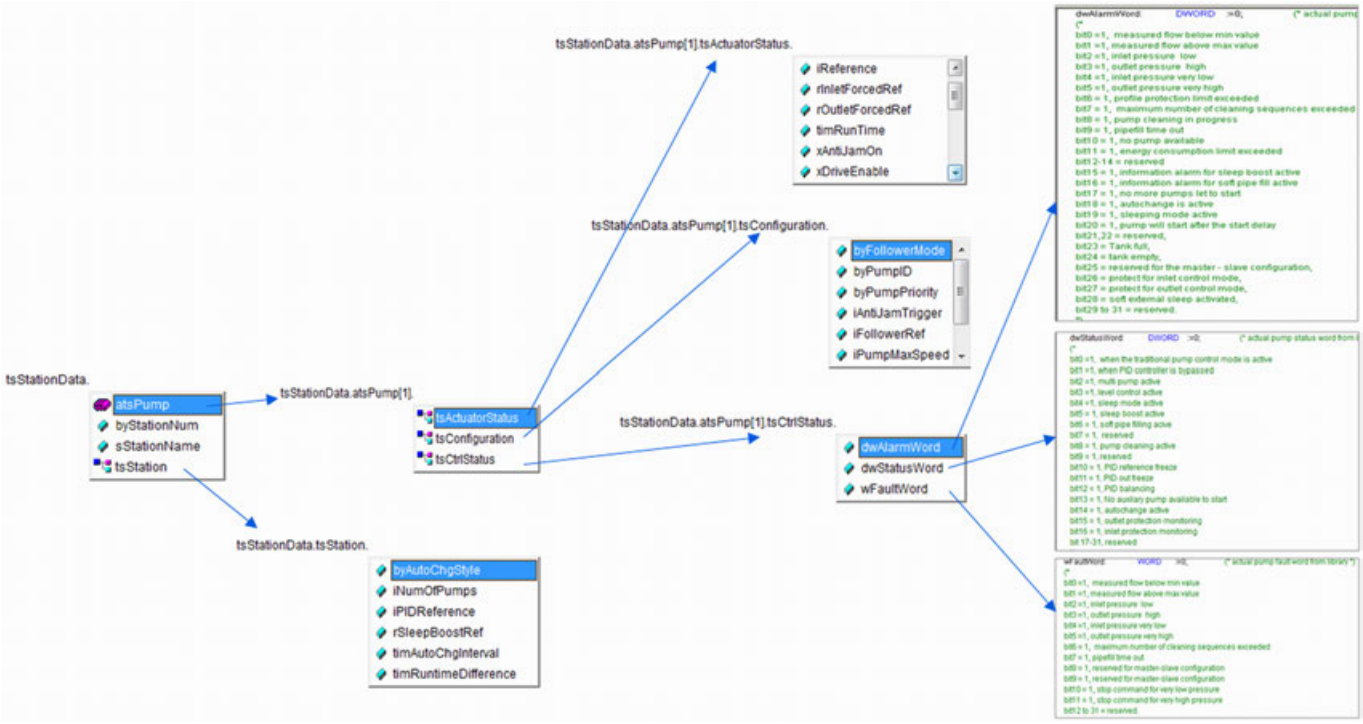


Fig. 642: Quick overview of the concept

PUMP_STATION_TYPE

byStationNum (station number)

Data type: BYTE, default value: 1, range: > 0

If more than one pump station are programmed, then byStationNum can be used to differentiate the stations.

sStationName (station name)

Data type: STRING(20)

If more than one pump station are programmed, then sStationName can be used to differentiate the stations in a "speaking" way and e.g. use for HMI purposes.

tsStation (station data)

Data type: zPUMP_STATION_DATA_TYPE

This structure stores station data.

atsPump (pumps data)

Data type: ARRAY[1..3] OF zPUMP_DATA_TYPE

An array is created for each pump. All pump information is stored in this structure.

byStationOpMode (station operating mode)

Data type: BYTE, default value: 2, range: 0-2

This variable stores the operating mode of the pumping station.

byMasterLoc (master location)

Data type: BYTE, default value: 1, range: 0-1

This variable stores the master location of the pumping station.

zPUMP_STATION_DATA_TYPE Station Configuration Data

iNumOfPumps (number of pumps)

Data type: INT

This variable defines the number of pumps in the station.

byAutoChgStyle (autochange style)

Data type: BYTE

Style	Short description	Description
0	None	Autochange function disabled.
1	Fixed	The starting sequence is shifted periodically at pre-defined intervals Autochg interval. In traditional pump control.
2	Run-time diff	The starting sequence is rearranged when the difference between the run times of two pumps exceed a limit, run-time difference. In the new sequence, the pump with the lowest run time will be started first, the pump with the highest run time will be started last.
3	All stop	The starting sequence is shifted every time the pump stops.

timAutoChgInterval (auto change interval time)

Data type: TIME

Time interval to elapse for the fixed autochange to activate.

timRuntimeDifference (run-time difference time)

Data type: TIME

Maximum permited runtime diff between the two pumps in the network.

rSleepBoostRef (sleep boost reference)

Data type: REAL

Speed step from the sleep function block. This step is used as additional setpoint to the pump before going to sleep mode.

byOperationMode (operation mode)

Data type: BYTE

Stores the operating mode of the pump boost control.

zPUMP_STATION_DATA_TYPE Station Actual Status

iPIDReference (pid reference)

Data type: INT

Value of speed correction coming from the PID function block.

xBoostEn (boost enable)

Data type: BOOL, default value: FALSE

Gives the true value when the pump boost enabled.

xBoostStart (boost start)

Data type: BOOL

Gives the true value when the pump boost starts.

xLevelCtrlEn (level control enable)
Data type: BOOL, default value: FALSE
Gives the TRUE value when the pump level control is enabled

xLevelCtrlStart (level control start)
Data type: BOOL
Gives the true value when the level control starts.

iStationMode (station mode)
Data type: BYTE
0 = none, 1 = pump bosst mode, 2 = level control mode.

aiSequenceID (sequence ID)
Data type: ARRAY[1..3] OF INT
Stores the sequence of the pumps operating in the station.

aiOriginalID (original ID)
Data type: ARRAY[1..3] OF INT
Stores the sequence of the pumps operating in the station without any sorting.

xSleepModeActive (sleep mode active)
Data type: BOOL
True value indicates that the sleep mode has been activated.

xSoftFillActive (soft fill active)
Data type: BOOL
True value indicates that the soft fill has been activated.

xAutoChgLevelCtrl (Autochange in the level control)
Data type: BOOL, default value: FALSE
When the auto change occurs in the level control, xAutoChgLevelCtrl becomes true for one scan cycle to ensure the changeover between the pumps.

xAssign3 (xAssign3)
Data type: BOOL, default value: FALSE
True value indicates the interface function block to assign the data in the structure

xAssign2 (xAssign2)
Data type: BOOL, default value: FALSE
True value indicates the interface function block to assign the data in the structure

xAssign1 (xAssign1)
Data type: BOOL, default value: FALSE
True value indicates the interface function block to assign the data in the structure

xAutoChgRunning (auto change running)
Data type: BOOL
True values indicates that auto change is running state.

zPUMP_DATA_TYPE

tsConfiguration (configuration structure)
Data type: zPUMP_CONFIGURATION_TYPE
This structure stores configuration data of pump.

tsActuator-Status (actuator status structure)
Data type: zPUMP_ACTUATOR_STATUS_TYPE
This structure stores actual values of pump.

tsCtrlStatus (actuator status structure)
Data type: zPUMP_CTRL_STATUS_TYPE
This structure stores control values of pump.

zPUMP_CONFIGURATION_TYPE

byPumpID (pump ID)
Data type: INT
This variable denotes Pump identification number.

byPumpPriority (pump priority)
Data type: BYTE
This variable denotes pump priority.

xPumpOn (pump on)
Data type: BOOL
By using FALSE value for this variable the user can remove this pump from the network in case of any problem/fault.

iPumpMax-Speed (pump max speed)
Data type: INT
This variable denotes maximum speed of the pump.

byFollowerMode (follower mode)
Data type: BYTE
0 = fixed, 1 = copy master, 2 = master speed.

iFollowerRef (follower reference)
Data type: INT
Follower speed ref in RPM for a pump, when follower mode is fixed.

xResetRuntime (reset run time)
Data type: BOOL
To reset the runtime value of the pump.

iAntiJamTrigger (antijam trigger)
Data type: INT
Trigger to clean the pump. 0 = not enabled, 1 = master enabled, 2 = follower enabled, 3 = at start, 4 = manual.

zPUMP_ACTUATOR_STATUS_TYPE

xDriveEnable (drive enable)
Data type: BOOL
This variable gives start command to the drive.

iReference (reference)
Data type: INT
Speed reference of the pump.

xMasterStatus (master status)
Data type: BOOL
TRUE value indicates that the pump is a MASTER and the FALSE value indicates that the pump is FOLLOWER.

xStopMode (stop mode)
Data type: BOOL
This identifies that the drive has to stop in the protection mode.

xInletRedSpeed-Mode (inlet reduced speed mode)
Data type: BOOL
This identifies that the pump has to run at the inlet reduced speed in the protection mode.

xOutletRed-SpeedMode (outlet reduced speed mode)
Data type: BOOL
This identifies that the drive has to run at the outlet reduced speed in the protection mode.

xReady (ready)
Data type: BOOL
True value tells that the pump is in ready state.

xOperating (operating)
Data type: BOOL
True value tells that the pump is in running state.

xFault (fault)
Data type: BOOL
True value indicates the fault in a pump.

xPumpProtec-tionFault (pump protection fault)
Data type: BOOL, default value: FALSE
True value indicates the fault generated by the protection bit.

xDriveFault (drive fault)
Data type: BOOL
True value indicates the drive fault.

xStandBy (standby)
Data type: BOOL
True value indicates that the pump is at standby state.

xRFGDisable (RFG disable)
Data type: BOOL
To disable the RFG of the drive of the pump.

xAntiJamOn (antijam on)
Data type: BOOL
On command to the pump in the manual mode of antijam function.

iAntiJamSpeed (antijamspeed)
Data type: INT
Speed reference to the pump in the manual mode of antijam function.

timRunTime(run time)
Data type: TIME
This variable stores the current run time of the pump.

rInletForcedRef (inlet forced reference) Data type: REAL

Forced speed ref when inlet mode=3, from pump_protection function block.

rOutletForcedRef (outlet forced reference) Data type: REAL

Forced speed ref when outlet mode=3, from pump_protection function block.

zPUMP_CTRL_STATUS_TYPE

dwStatusWord (status word) Data type: DWORD

Pump status word.

bit	Description
bit0 =1	When the traditional pump control mode is active.
bit1 =1	When PID controller is bypassed.
bit2 =1	Multi pump active.
bit3 =1	Level control active.
bit4 =1	Sleep mode active.
bit5 =1	Sleep boost active.
bit6 =1	Soft pipe filling active.
bit7 =1	Reserved
bit8 =1	Pump cleaning active.
bit9 =1	Reserved
bit10 =1	PID reference freeze.
bit11 =1	PID out freeze.
bit12 =1	PID balancing.
bit13 =1	No auxilary pump available to start.
bit14 =1	Autochange active.
bit15 =1	Outlet protection monitoring.
bit16 =1	Inlet protection monitoring.
bit17-31	Reserved

dwAlarmWord (alarm word)

Data type: DWORD

Pump alarm word.

bit	Description
bit0 =1	Measured flow below min value.
bit1 =1	Measured flow above max value.
bit2 =1	Inlet pressure low.
bit3 =1	Outlet pressure high.
bit4 =1	Inlet pressure low.
bit5 =1	Outlet pressure high.
bit6 =1	Profile protection limit exceeded.
bit7 =1	Maximum number of cleaning sequences exceeded.
bit8 =1	Pump cleaning in progress.
bit9 =1	Pipefill time out.
bit10 =1	No pump available.
bit11 =1	Energy consumption limit exceeded.
bit12-14 =1	Reserved.
bit15 =1	Information alarm for sleep boost active.
bit16 =1	Information alarm for soft pipe fill active.
bit17 =1	No more pumps let to start.
bit18 =1	Autochange is active.
bit19 =1	Sleeping mode active.
bit20 =1	Pump will start after the start delay.
bit21,22 =1	Reserved
bit23 =1	Tank full.
bit24 =1	Tank empty.
bit25 =1	Reserved for the master-slave configuration.
bit26 =1	Protect for inlet control mode.
bit27 =1	Protect for outlet control mode.
bit28-31	Reserved

wFaultWord (fault word)

Data type: WORD

Pump fault word.

bit	Description
bit0 =1	Measured flow below min value.
bit1 =1	Measured flow above max value.
bit2 =1	Inlet pressure low.
bit3 =1	Outlet pressure high.
bit4 =1	Inlet pressure very low.
bit5 =1	Outlet pressure very high.
bit6 =1	Maximum number of cleaning sequences exceeded.
bit7 =1	Pipefill time out.
bit8,9 =1	Reserved for master-slave configuration.
bit10 =1	Stop command for very low pressure.
bit11 =1	Stop command for very high pressure.
bit12-31	Reserved

Structure Mapping

Structure	Variable Name	Type	Block which writes and/or reads: LEVEL_CTRL	BOOST_CTRL	PID	FLOW_CALC	STATION_CFG	RETAIN_DATA	INTERFACE	SLEEP	ANTIJAM	AUTOCHANGE	PROTECTION	ENERGY_CALC
PUMP_STATION_TYPE														
	byStationNum	BYTE					w/r							
	sStationName	STRING(20)					w/r							
	tsStation	zPUMP_STATION_DATA_TYPE;	r	r	r	r	w/r	r	r	r	r	r	r	
	atsPump	ARRAY[1..3] OF zPUMP_DATA_TYPE;	w/r	w/r	w/r	r	r	r	w/r	r	w/r	w/r	w/r	
	byStationOpMode	BYTE	w/r	w/r								r		
	byMasterLoc	BYTE		w/r								r		
PUMP_STATION_DATA_TYPE														
Station Configuration Data														
	iNumOfPumps	INT	r	r		r	w/r	r	r	r	r	r	r	
	byAutoChgStyle	BYTE							r			w/r		
	timAutoChgInterval	TIME												
	timRuntimeDifference	TIME												
	rSleepBoostRef	REAL			r					w/r				
	byOperationMode	BYTE		w/r								r		
	xAutoChgLevelCtrl	BOOL	r									w/r		
Station Actual Status														
	iPIDReference	INT		w/r	w/r							r		
	xBoostStart	BOOL		w/r							r	r		
	xLevelCtrlStart	BOOL	w									r		
	byOperationMode	BYTE		r/w			r					r		
	iStationMode	BYTE	w/r	w/r					r			r		
	aiSequenceID	ARRAY[1..3] OF INT (sequence ID)						r	r			w/r		
	aiOriginalID	ARRAY[1..3] OF INT (original ID)	w/r	w/r								r		
	xSleepModeActive	BOOL		r						w/r				
	xBoostEn	BOOL	r	r/w										
	xLevelCtrlEn	BOOL	r/w	r										
	xAssign1	BOOL	r/w	r/w					r/w					
	xAssign2	BOOL	r/w	r/w					r/w					
	xAssign3	BOOL	r/w	r/w					r/w					
	xAutoChgRunning	BOOL	r	r				r	r			w/r		
PUMP_DATA_TYPE														
	tsConfiguration	PUMP_CONFIGURATION_TYPE	r	r	r	r		r	w/r	r	w/r	r	r	
	tsActuatorStatus	PUMP_ACTUATOR_STATUS_TYPE	w/r	w/r	w/r		r	r	w/r	r	w/r	w/r	w/r	
	tsCtrlStatus	PUMP_CTRL_STATUS_TYPE	w/r	w/r	r				w/r	w/r	w/r	w/r	w/r	
PUMP_CONFIGURATION_TYPE														
	byPumpID	INT	r	r		r		r	w/r	r	r	r	r	
	byPumpPriority	BYTE	r	r				r	w/r					
	xPumpOn	BOOL	r	r					w/r		r	r		
	iPumpMaxSpeed	INT	r	r	r	r			w/r		r			
	byFollowerMode	BYTE		r					w/r			r		
	iFollowerRef	INT		r					w/r			r		
	xResetRuntime	BOOL							w/r					
	iAntiJamTrigger	INT		r					r		w/r	r		
PUMP_ACTUATOR_STATUS_TYPE														
	xDriveEnable	BOOL	w/r	w/r					r			w/r		
	iReference	INT	w/r	w/r					r		w/r	w/r		
	xMasterStatus	BOOL		w/r	r					r		w/r		
	xStopMode	BOOL	r	r	r								w/r	
	xInletRedSpeedMode	BOOL	r	r	r								w/r	
	xOutletRedSpeedMode	BOOL	r	r	r								w/r	
	xReady	BOOL	r	r					w/r		r	r		
	xOperating	BOOL	r	r					w/r		r	r		
	xFault	BOOL	w/r	w/r					r		r	r		
	xDriveFault	BOOL	r	r					w/r		r			
	xPumpProtectionFault	BOOL	r	r							r		r/w	
	xStandBy	BOOL		w/r										
	xRFGDisable	BOOL												
	xAntiJamOn	BOOL							r		w/r			
	iAntiJamSpeed	INT							r		w/r			
	timRunTime	TIME						r	w/r			r		
	rInletForcedRef	REAL	r	r	r								w/r	
	rOutletForcedRef	REAL	r	r	r								w/r	

Fig. 643: Structure Mapping 1

			Block which writes and/or reads										
Structure	Variable Name	Type	LEVEL_CTRL	BOOST_CTRL	PID	FLOW_CALC	STATION_CFG	RETAIN_DATA	INTERFACE	SLEEP	ANTUAM	AUTOCHANGE	PROTECTION	ENERGY_CALC
PUMP_CTRL_STATUS_TYPE														
	dwStatusWord	DWORD												
	bit0 =1, when the traditional pump control mode is active			w/r								r		
	bit1 =1, when PID controller is bypassed													
	bit2 =1, multi pump active			w/r										
	bit3 =1, level control active		w/r											
	bit4 =1, sleep mode active													
	bit5 = 1, sleep boost active													
	bit6 = 1, soft pipe filling acive			r	r									
	bit7 = 1, reserved													
	bit8 = 1, pump cleaning active			w/r							w/r	r		
	bit9 = 1, reserved													
	bit10 = 1, PID reference freeze													
	bit11 = 1, PID out freeze													
	bit12 = 1, PID balancing													
	bit13 = 1, No auxilary pump available to start													
	bit14 = 1, autochange active											w		
	bit15 = 1, outlet protection monitoring												w	
	bit16 = 1, inlet protection monitoring												w	
	bit 17-31, reserved													
	dwAlarmWord	DWORD												
	bit0 =1, measured flow below min value												w	
	bit1 =1, measured flow above max value												w	
	bit2 =1, inlet pressure low												w	
	bit3 =1, outlet pressure high												w	
	bit4 =1, inlet pressure low												w	
	bit5 =1, outlet pressure high												w	
	bit6 = 1, profile protection limit exceeded												w	
	bit7 = 1, maximum number of cleaning sequences exceeded										w/r			
	bit8 = 1, pump cleaning in progress										w/r			
	bit9 = 1, pipefill time out													
	bit10 = 1, no pump available													
	bit11 = 1, energy consumption limit exceeded													
	bit12-14 = reserved													
	bit15 = 1, information alarm for sleep boost active													
	bit16 = 1, information alarm for soft pipe fill active													
	bit17 = 1, no more pumps let to start													
	bit18 = 1, autochange is active											w		
	bit19 = 1, sleeping mode active				r					w/r				
	bit20 = 1, pump will start after the start delay													
	bit21,22 = reserved,													
	bit23 = Tank full,		w											
	bit24 = tank empty,		w											
	bit25 = reserved for the master - slave configuration,													
	bit26 = protect for inlet control mode,				r								w/r	
	bit27 = protect for outlet control mode,				r								w/r	
	bit28 to 31 = reserved.													
	wFaultWord	WORD												
	bit0 =1, measured flow below min value												w	
	bit1 =1, measured flow above max value												w	
	bit2 =1, inlet pressure low												w	
	bit3 =1, outlet pressure high												w	
	bit4 =1, inlet pressure very low												w	
	bit5 =1, outlet pressure very high												w	
	bit6 = 1, maximum number of cleaning sequences exceeded													
	bit7 = 1, pipefill time out													
	bit8 = 1, reserved for master-slave configuration													
	bit9 = 1, reserved for master-slave configuration													
	bit10 = 1, stop command for very low pressure												w	
	bit11 = 1, stop command for very high pressure												w	
	bit12 to 31 = reserved.													

Fig. 644: Structure Mapping 2

1.5.13.1.6 PUMP_BOOST_CTRL Boost Control

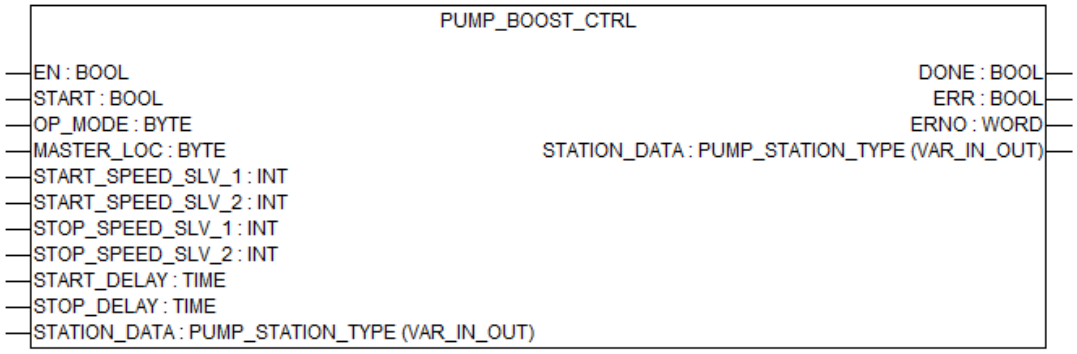


Fig. 645: Function block PUMP_BOOST_CTRL

Table 203: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This application Function Block provides a typical logic for pump boost operations with several pumps - where the user can select the operation mode i.e. single pump, multi pump or traditional pumping for up to three pumps. The switching on and off of further pumps is triggered via start and stop speeds.

With this function block the

- traditional / conventional (only one pump with drives, other DOL* motors) and
- advanced (all pumps operated by drive-motors) pumping stations can be configured and operated. (* DOL = direct online: Motor without a drive but a switch or soft-starter connected to e.g. 50Hz AC line voltage).

PUMP_PID uses the speed reference based on the process demands to PUMP_BOOST_CTRL. PUMP_BOOST_CTRL controls the start/stop sequence for the pumps based on the start and stop speeds to fullfill process requirements.

Follower modes in boost operation

The following figures show the different control modes, resulting in different criteria how to run the followers. The follower mode itself is set in each PUMP_INTERFACE Block.

The follower modes in each of the three PUMP_INTERFACE Blocks have to be set to the same mode.

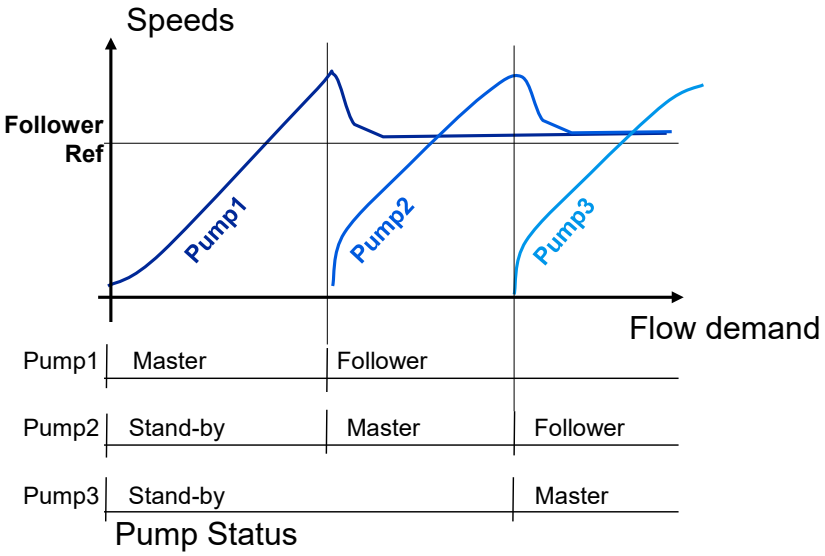


Fig. 646: Constant speed

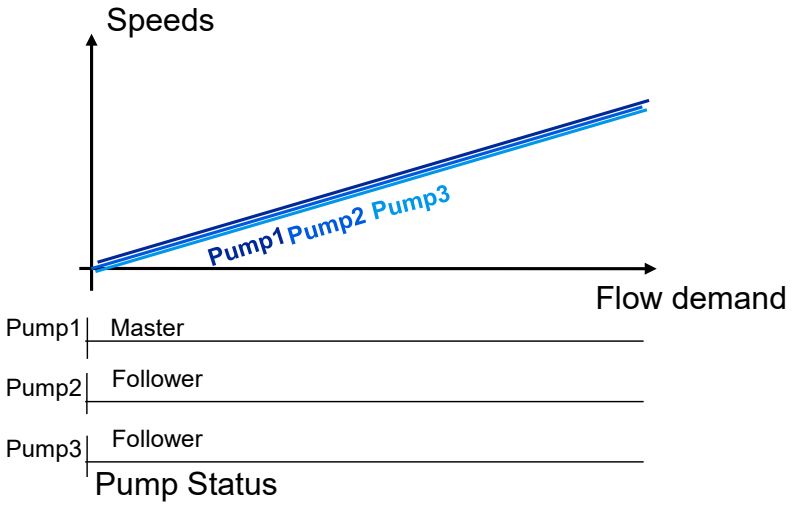


Fig. 647: Copy of “Master”/direct follower

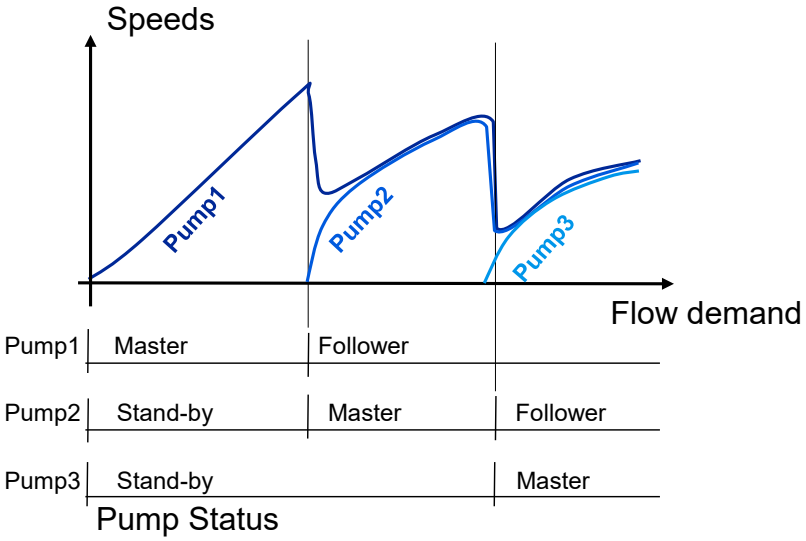


Fig. 648: "Master" Speed

Input description

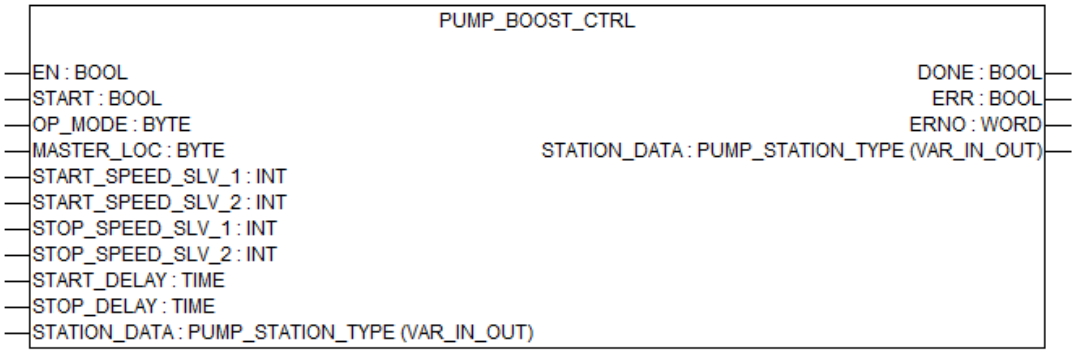


Fig. 649: Function block PUMP_BOOST_CTRL

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START (start command)

Data type: BOOL

To start the boost control operation with the TRUE value of the START variable. When START = FALSE, all the pumps stop.

OP_MODE (operation mode)

Data type: BYTE, default value: 0, range: 0-2

This input sets the operating mode of the pumping station.

Operation mode	Short description	Description
0	single pump	Operated by the drive.
1	traditional pump	One pump is driven by the drive and others are auxiliaries driven by contactors.
2	multi pump	The pumps are controlled by the drives.

Range of values: from 0 to 2

MASTER_LOC (master location of the pumping station)

Data type: BYTE, default value: 0, range: 0-1

This input defines the master location.

MASTER_LOC value	MASTER_LOC type	Description
0	INSTART	The master in the network is not fixed and "in the start" phase the last started pump in the network will be master. Master means basically it is the only pump which always takes the speed given by the PID. The followers speed depends on the chosen follower mode. Example in the input OP_MODE = 2 (Multi-pump): When the pumping station is started, Pump1 (ID=1) will start as a master. Demand increase will prompt the Pump2 also to start and it becomes the master and the Pump1 is a follower (whose operation is decided by the follower mode set on function block PUMP_INTERFACE). If the demand increases further, also Pump3 is started and becomes the master. If the demand then decreases again, Pump3 is stopped first (as per the sequence 1-2-3). Now Pump1 (follower) and Pump2 is again the master. Demand further decreases, Pump2 stops next and Pump1 now runs as the master. Remember if the start sequence is 1-2-3 the stop sequence is always 3-2-1.
1	FIXED	Master is always fixed, follower changes. In this mode the first pump in the sequence is and stays always the master. Example: When the pumping station is started, Pump1 (ID=1) will start as a master. Demand increase will prompt the Pump2 also to start as a follower. If the demand further increases, Pump3 also starts and take the follower speeds based on the chosen FOLLOWER _Mode. Here the Pump1 always remains the master. When the demand was high (3 pumps running) and the demand decreases, Pump3 will stop first (as per the sequence 1-2-3), then Pump2 will also stop if the demand decreases even further. But Pump1 stays always master. Principle: If the start sequence is 1-2-3, the stop sequence is always 3-2-1.

Range of values: from 0 to 1.

START_SPEED_SLV_1 (start speed for slave 1)

Data type: INT, default value: 0, range: ≥ 0, unit: rpm

The speed of the master at which the first follower pump must start in the follower mode 0 and 2.

START_SPEED_SLV_2 (start speed for slave 2)

Data type: INT, default value: 0, range: ≥ 0, unit: rpm

The speed of the master which the second follower pump must start in the follower mode 0 and 2.

STOP_SPEED_SLV_1 (stop speed for slave 1)

Data type: INT, default value: 0, range: ≥ 0, unit: rpm

The speed of master at which the first follower pump must stop in the follower mode 0 and 2.

STOP_SPEED_SLV2 (stop speed for slave 2)

Data type: INT, default value: 0, range: ≥ 0, unit: rpm

The speed of master at which the second follower pump must stop in the follower mode 0 and 2.

START_DELAY (start delay)

Data type: TIME, default value: 10, range: > 0, unit: s

This input is a time delay in seconds. The next pump starts only when the condition to start this pump has stayed for the duration mentioned in this time delay. This input is to be given when the operating mode is 1 or 2.

STOP_DELAY (stop delay)

Data type: TIME, default value: 10, range: > 0, unit: s

This input is atime delay in seconds. When the condition to stop the pump has stayed for the duration mentioned in this input, the pump stops. This input is to be given when the operating mode is 1 or 2.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

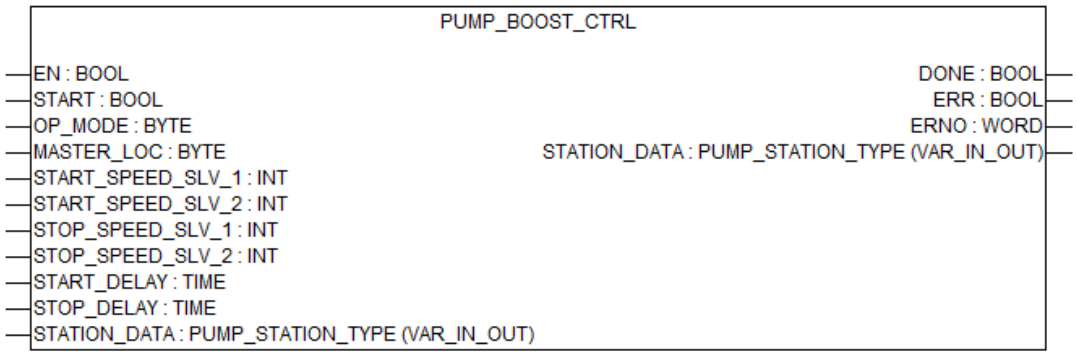


Fig. 650: Function block PUMP_BOOST_CTRL

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16403	4013	EN signal for boost control given, although the pump level control is enabled.
16432	4030	Value of OP_MODE erroneous.
16448	4040	Value of MASTER_LOC erroneous.
16464	4050	Value of START_SPEED_SLV_1 erroneous.
16480	4060	Value of START_SPEED_SLV_2 erroneous.
16496	4070	Value of STOP_SPEED_SLV_1 is erroneous.
16512	4080	Value of STOP_SPEED_SLV_2 is erroneous.

1.5.13.1.7 PUMP_PID

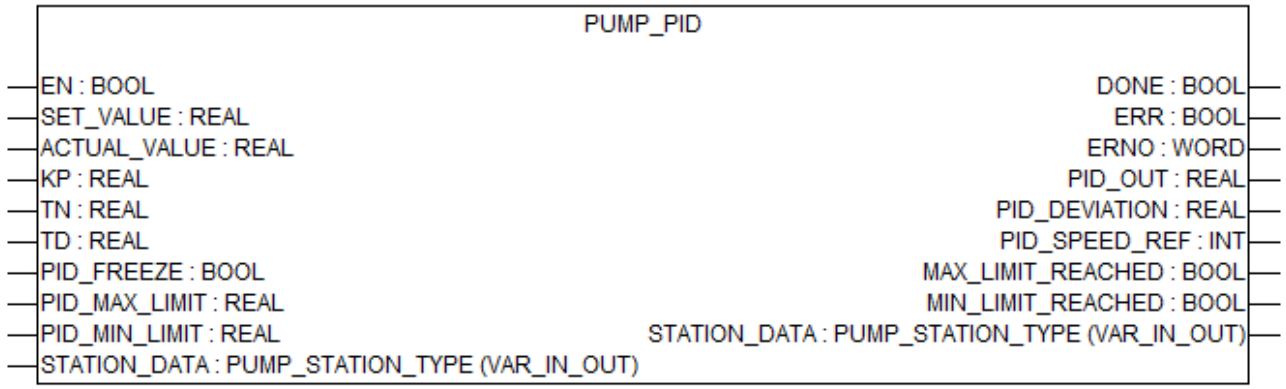


Fig. 651: Function block PUMP_PID

Table 204: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block is used for closed loop control of the process parameter such as pressure.

The setpoint and the actual value are compared and error value is given as input to the PID. PID output is calculated based on the Proportional Gain KP, Integral Time TN, and Derivative Time TD.

Y = KP *(e +1/TN ∫e dt + TD de/dt) with

Y =PID_OUTPUT

e =Error between SET_VALUE – ACTUAL_VALUE

The PID output is then scaled and converted into required speed reference to the drive of the pump. The output PID_SPEED_REF is internally attached via structures to the PUMP_BOOST_CTRL.

Input description

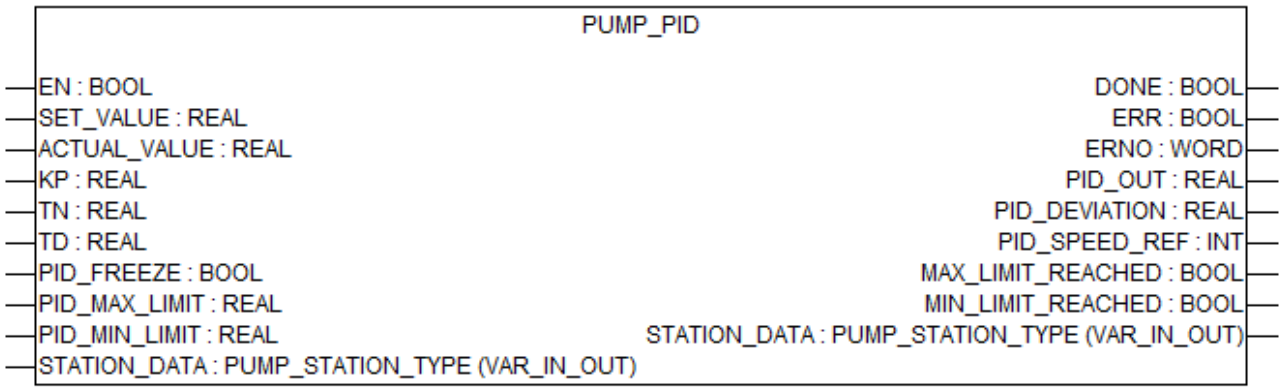


Fig. 652: Function block PUMP_PID

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SET_VALUE (set point of the PID)

Data type: REAL, default value: 0, range: ≥ 0, unit: %

Set value of the process variable.

Range of values: from 0 to 100 .

ACT_VALUE (actual point of the PID)

Data type: REAL, default value: 0, range: ≥ 0, unit: %

Actual value of the process variable. User need to scale actual value as percentage and attached to this variable.

Range of values: from 0 to 100.

KP (proportional gain)

Data type: REAL, default value: 0, range: ≥ 0

Proportional gain of the PID.This value has to be manually tuned as per system requirements.

TN (integration time)

Data type: REAL, default value: 0, range: ≥ 0, unit: ms

Integral time of the PID in terms of milliseconds. If the value entered is 500.0, it is taken as 500 ms.This value has be manually tuned as per system requirements.

TD (derivative time)

Data type: REAL, default value: 0, range: ≥ 0, unit: ms

Derivative time of the PID in terms of milliseconds. If the value entered is 500.0, it is taken as 500 ms.This value has be manually tuned as per system. Erroneous setting of derivative time can results in overshooting of the PID-OUT value.

PID_FREEZE (PID freeze)

Data type: BOOL

For the TRUE value the setpoint to the PID block is frozen. Actual value is overwritten to the setpoint.

PID_MAX-IMUM_LIMIT (pid maximum limit value)

Data type: REAL, default value: 100, range: 0-100, unit: %

Maximum permitted value of the PID output. Using the minimum and maximum limits, it is possible to restrict the operation range of the PID.

PID_MIN-IMUM_LIMIT (pid minimum limit value)

Data type: REAL, default value: 0, range: -100.0 to +100.0, unit: %

Minimum permitted value of the PID output. Using the minimum and maximum limits, it is possible to restrict the operation range of the PID.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

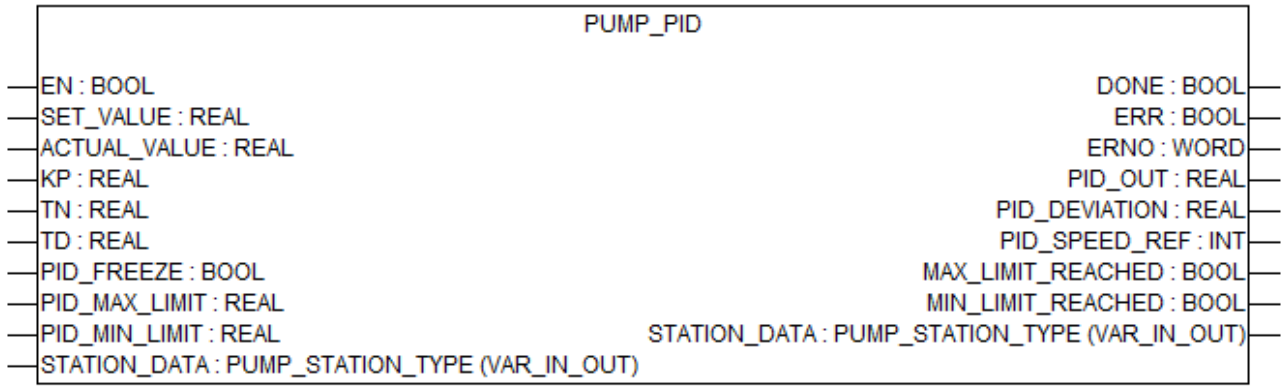


Fig. 653: Function block PUMP_PID

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

PID_OUT (pid output value)

Data type: REAL, default value: 0, range: -100 to +100, unit: %

Output PID_OUT after the manipulation.

PID_DEVIATION (deviation of set value from actual value)

Data type: REAL, default value: 0, range: -100 to +100, unit: %

Deviation of set value from actual value. It is also known as PID error.

PID_SPEED_REF (pid speed reference)

Data type: INT, default value: 0, range: -20000 to +20000

Speed reference as a correction to the main speed reference, calculated by the PID. This variable is meant for the user. The PID_SPEED_REF is also internally attached with the PUMP_BOOST_CTRL function block to process the start/stop of the pumps.

MAXIMUM_LIMIT_REACHED (maximum limit reached)

Data type: BOOL

Indicates that the PID output reached the MAXIMUM limit.

MINIMUM_LIMIT_REACHED (minimum limit reached)

Data type: BOOL

Indicates that the PID output reached the MINIMUM limit.

1.5.13.1.8 PUMP_LEVEL_CTRL Level Control

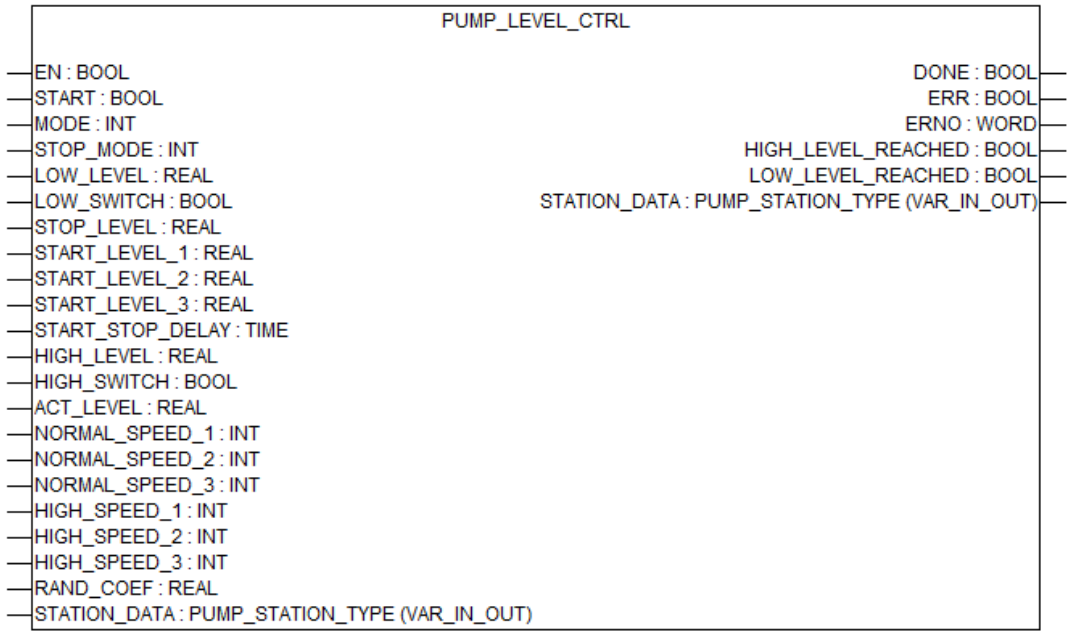


Fig. 654: Function block PUMP_LEVEL_CTRL

Table 205: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

The level control macro is designed for controlling a station of 1 to 3 pumps that is used for either emptying or filling a container. The level control functionality is activated by setting parameter level mode to emptying or filling. The start levels for the pumps (as well as the alarm levels) are set by parameters.

Filling mode

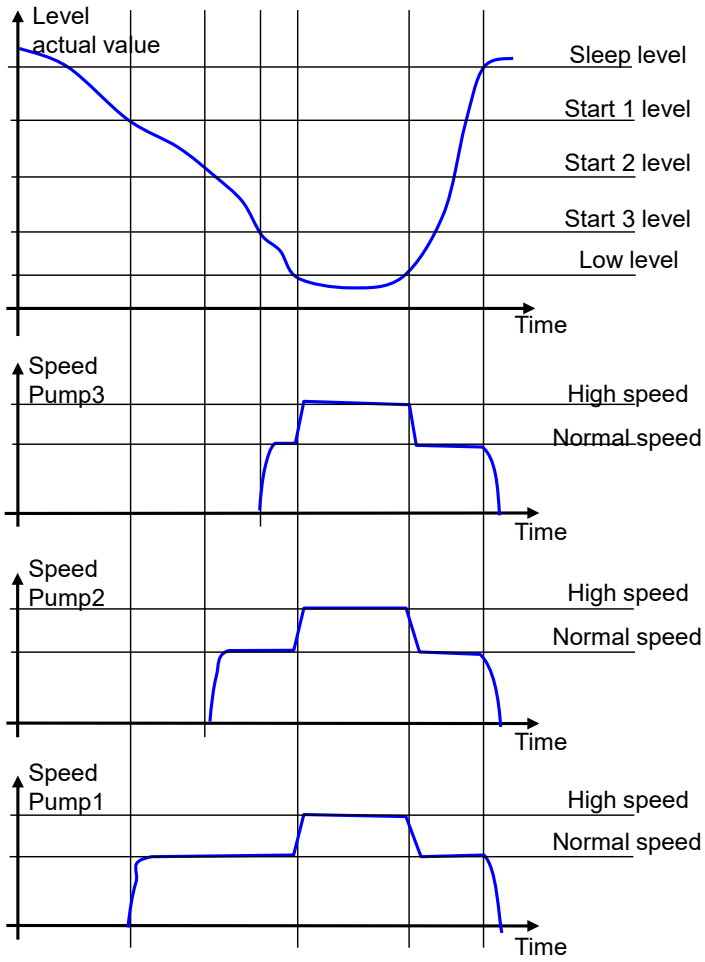


Fig. 655: Behavior and meaning of the parameters if the “Filling Mode” is set

The pump station above is used for filling a tank, decreasing its level due to a larger outlet flow. The diagram shows the start, stop and supervision levels for filling. Parameter “Stopping mode” is assumed to be set to “Common stop”; “Start stop delay” is assumed to be set to 0.

A larger outlet flow e.g. to a fresh water network via a supply tank / reservoir:	The pumps need to pump fresh water to keep the level possibly constant to feed a network with constant pressure.

Emptying mode

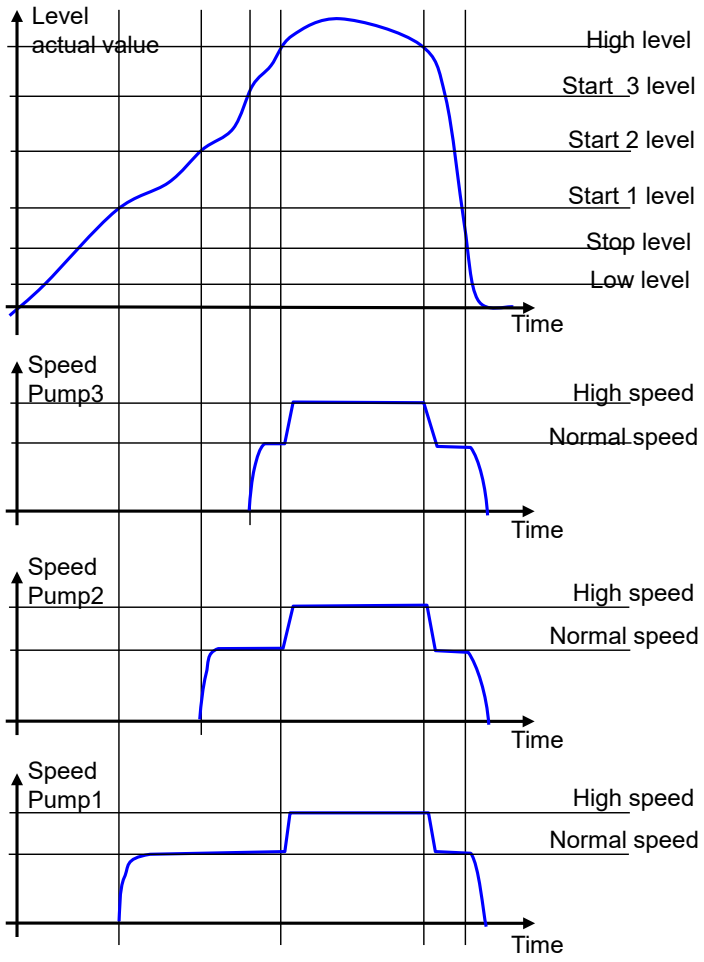


Fig. 656: Pump station using for emptying a container, increasing its level due to inflow

The diagram above shows the start, stop and supervision levels for emptying mode. Parameter “stopping mode” is assumed to be set to “common stop”; “start stop delay” is assumed to be set to 0.

Example is a reservoir or tank with a larger inlet flow (e.g. in a waste water lift stations):	The pumps need to work in emptying mode, meaning to pump waste water further to avoid spilling of the reservoir.

Input description

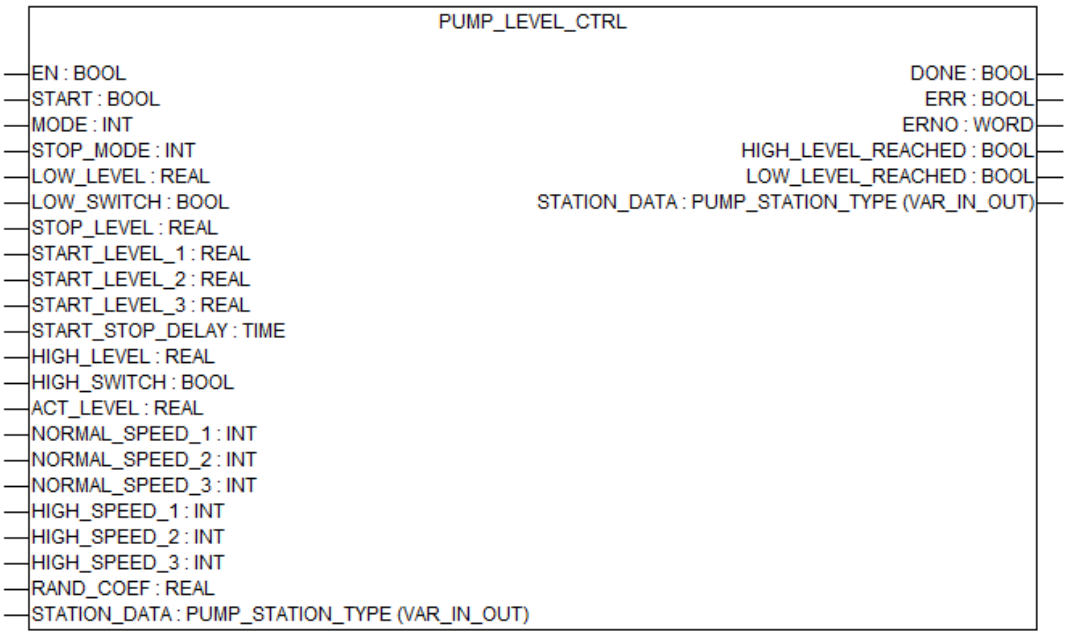


Fig. 657: Function block PUMP_LEVEL_CTRL

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START (start block)

Data type: BOOL

To start the level controller.

MODE (mode) Data type: INT, default value: 1, range: 0-2

To select the mode of operation.

Mode	Short description	Description
0	Off	Level control disabled.
1	Emptying	Pump station is used for emptying a container.
2	Filling	Pump station is used for filling a container.

STOP_MODE (stop mode) Data type: INT, default value: 0, range: 0-1

Mode	Description
0	Stable stop. Stops the pump at individual defined stop levels. For example in the fill mode if the START_LEVEL_1 is defined 60%. Now if the actual level reaches on or above 60%, the pump 1 would stop.
1	Common stop. Stops all the pumps together at defined level STOP_LEVEL. Which means, both in the emptying and filling mode, the pumps start at their respective start levels, but they stop when they reach the STOP_LEVEL.

LOW_SWITCH (low switch) Data type: BOOL, default value: FALSE

Digital input with TRUE indicates that the low level has reached.

STOP_LEVEL (stop level) Data type: REAL, default value: 10.0, range: >0, unit: %

Stop level defines the level at which all the pumps would stop, when the STOP_MODE = 1. This can occur while filling when the actual level is more than the stop level. In emptying mode all pumps stop when the actual level reaches below the stop level.

START_LEVEL_1 (start level 1) Data type: REAL, default value: 70, range: > 0, unit: %

Defines the start level for pump 1 in terms of full tank capacity. The same level is used to stop the pump when actual level comes back at this level. Refer the figure in the above section.

START_LEVEL_2 (start level 2) Data type: REAL, default value: 50, range: > 0, unit: %

Defines the start level for pump 2 in terms of full tank capacity. The same level is used to stop the pump when actual level comes back at this level. Refer the figure in the above section.

START_LEVEL_3 (start level 3) Data type: REAL, default value: 30, range: > 0, unit: %

Defines the start level for pump 3 in terms of full tank capacity. The same level is used to stop the pump when actual level comes back at this level. Refer the figure in the above section.

START_STOP_DELAY (start stop delay) Data type: TIME, default value: 10, range: ≥ 0, unit: s

Defines the time delay for pump to start and stop when respective level is reached.

HIGH_LEVEL (high level)
Data type: REAL, default value: 95, range: > 0, unit: %
Defines the high level in terms of full tank capacity.

HIGH_SWITCH (high switch)
Data type: BOOL
Digital input which with TRUE indicates that the high level has reached.

ACT_LEVEL (actual level)
Data type: REAL, default value: 0, range: ≥ 0, unit: %
Actual level read from the analog input of level sensor.

NORMAL_SPEED_1 (normal speed for pump 1)
Data type: INT, default value: 1100, range: ≥ 0, unit: rpm
Normal operating speed of the pump 1 while Filling/Emptying.

NORMAL_SPEED_2 (normal speed for pump 2)
Data type: INT, default value: 1100, range: ≥ 0, unit: rpm
Normal operating speed of the pump 2 while Filling/Emptying.

NORMAL_SPEED_3 (normal speed for pump 3)
Data type: INT, default value: 1100, range: ≥ 0, unit: rpm
Normal operating speed of the pump 3 while Filling/Emptying.

HIGH_SPEED_1 (high speed for pump 1)
Data type: REAL, default value: 1500, range: ≥ 0, unit: rpm
Defines the speed of the pump 1 for:
- Filling pump level falls below LOW_LEVEL,
- Emptying pump level rises above the HIGH_LEVEL.

HIGH_SPEED_2 (high speed for pump 2)
Data type: REAL, default value: 1500, range: ≥ 0, unit: rpm
Defines the speed of the pump 2 for:
- Filling pump level falls below LOW_LEVEL,
- Emptying pump level rises above the HIGH_LEVEL.

HIGH_SPEED_3 (high speed for pump 3)
Data type: REAL, default value: 1500, range: ≥ 0, unit: rpm
Defines the speed of the pump 3 for:
- Filling pump level falls below LOW_LEVEL,
- Emptying pump level rises above the HIGH_LEVEL.

RAND_COEF (random coefficient)
Data type: REAL, default value: 0, range: ≥ -10 to +10, unit: %
Random coefficient randomizes the start levels to avoid cake formation.

STATION_DATA (station data structure)
Data type: PUMP_STATION_TYPE
This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

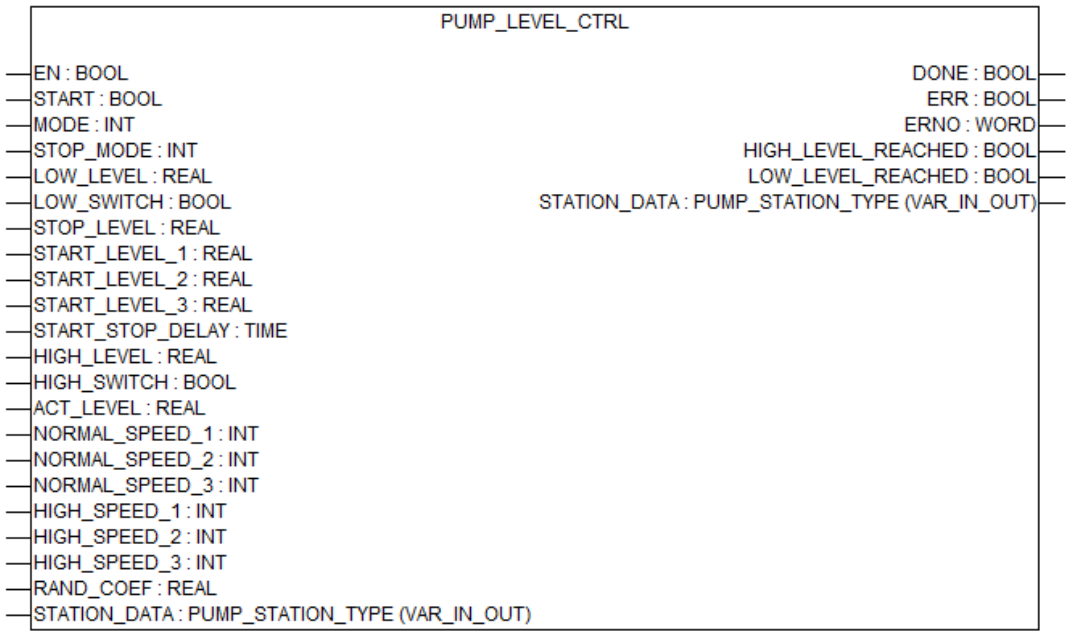


Fig. 658: Function block PUMP_LEVEL_CTRL

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

HIGH_LEVEL_REACHED (high level reached)

Data type: BOOL

TRUE value indicates that the actual level is more than the high level.

LOW_LEVEL_REACHED (low level reached)

Data type: BOOL

TRUE value indicates that the actual level is below the low level.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16403	4013	EN signal for level control given, although the pump boost control is enabled.
16432	4030	Value of MODE is erroneous.
16435	4033	Value of MODE level settings is erroneous.
16448	4040	Value of STOP_MODE is erroneous.
16464	4050	Value of LOW_LEVEL is erroneous.
16480	4060	Value of STOP_LEVEL is erroneous.
16512	4080	Value of START_LEVEL_1 is erroneous.
16528	4090	Value of START_LEVEL_2 is erroneous.
16544	40A0	Value of START_LEVEL_3 is erroneous.
16576	40C0	Value of HIGH_LEVEL is erroneous.
16608	40E0	Value of ACT_LEVEL is erroneous.
16624	40F0	Value of NORMAL_SPEED_1 is erroneous.
16640	4100	Value of NORMAL_SPEED_2 is erroneous.
16656	4110	Value of NORMAL_SPEED_3 is erroneous.
16672	4120	Value of HIGH_SPEED_1 is erroneous.
16688	4130	Value of HIGH_SPEED_2 is erroneous.
16704	4140	Value of HIGH_SPEED_3 is erroneous.
16720	4150	Value of RAND_COEF is erroneous.

1.5.13.1.9 PUMP_AUTOCHANGE

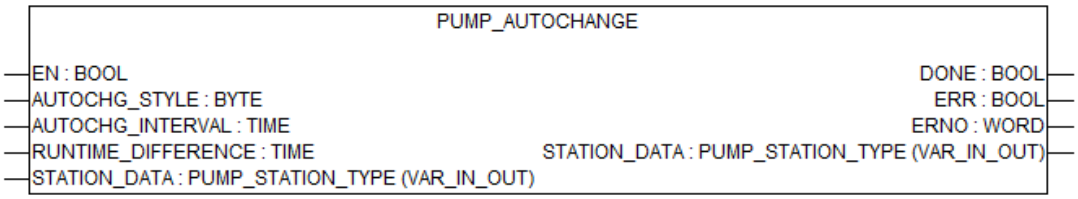


Fig. 659: Function block PUMP_AUTOCHANGE

Table 206: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

The Autochange function can be used to equalize duty time between multiple pumps, by varying the sequence in which pumps are started.

There are three autochange modes selectable by parameter autochange style:

- Fixed intervals: The starting sequence is shifted periodically at pre-defined intervals Autochange interval,
- Runtime equalization (hours count): The starting sequence is rearranged when the difference between the run times of two pumps exceed a limit, run-time difference. In the new sequence, the pump with the lowest run time will be started first. The pump with the highest run time will be started last,
- Autochange when stopped (all stop): The starting sequence for the next start is shifted every time the pump stops.

Example of a change in the sequence by AUTOCHANGE

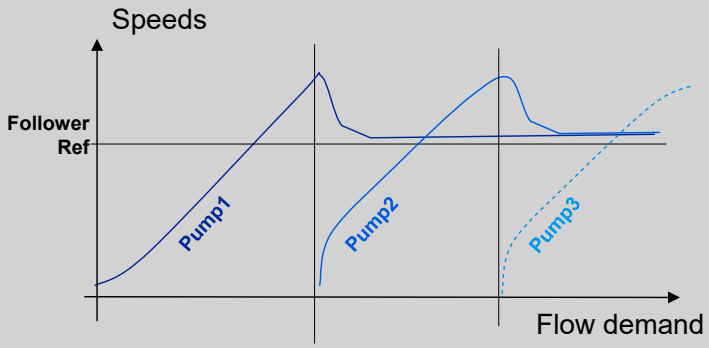


Fig. 660: Before sequence change: 2 pumps run

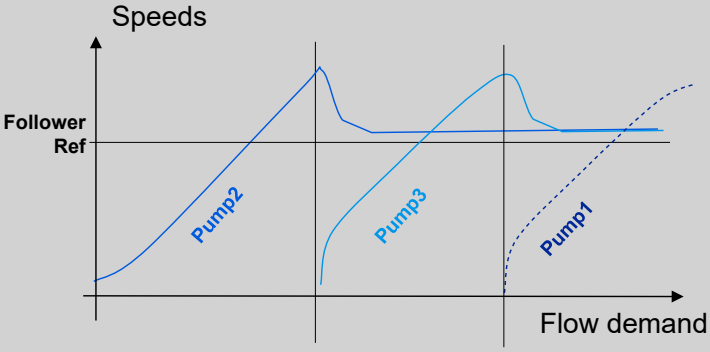


Fig. 661: After sequence change: Pump 1 was "oldest" →standby, 2 pumps run

Input description

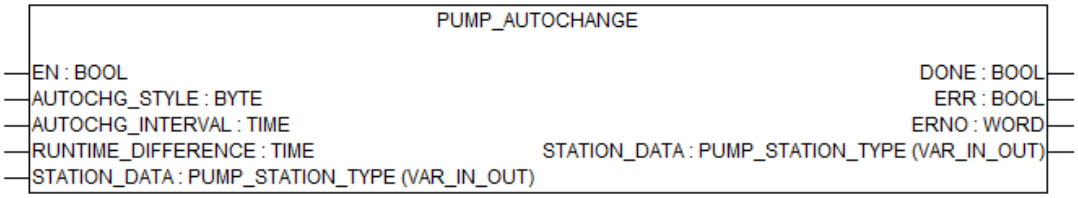


Fig. 662: Function block PUMP_AUTOCHANGE

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

AUTOCHG_STYLE (auto change style)

Data type: BYTE, default value: 0, range: 0-3

Mode	Short description	Description
0	None	Autochange function disabled.
1	Fixed	The starting sequence is shifted periodically at pre-defined intervals Autochange interval. For the Autochange to occur in this mode, it is necessary that atleast one pump is in standby condition. With this, when the Autochange occurs the pump with the maximum run time would stop and the new pump would start.
2	Runtime diff	The starting sequence is rearranged when the difference between the runtimes of two pumps exceed a limit, run-time difference. In the new sequence, the pump with the lowest runtime will be started first, the pump with the highest run time will be started last. For the autoch-ange to occur in this mode, it is necessary that atleast one pump is in standby condition. With this, when the autochange occurs the pump with the maximum run time would stop and the new pump would start.
3	All stop	The starting sequence is shifted every time the pump stops. This mode is also available to operate in the traditional pumping station.

AUTOCHG_INTERVAL (auto change interval)

Data type: TIME, default value: 60, range: > 0, unit: s

Time delay to elapse for the fixed Autochange to activate when the Autochange style selected is = 1.

RUNTIME_DIFFERENCE (run-time difference)

Data type: TIME, default value: 60, range: > 0, unit: s

Maximum permitted run-time difference between the two pumps in the network when the Autochange style selected is = 2.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

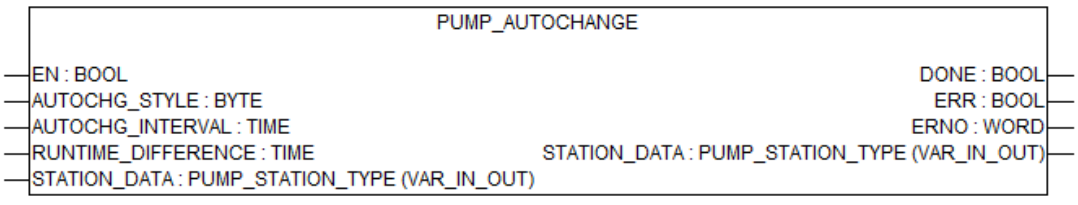


Fig. 663: Function block PUMP_AUTOCHANGE

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of AUTOCHG_STYLE is erroneous.
16419	4023	Value of AUTOCHG_STYLE and related setting is erroneous.

1.5.13.1.10 PUMP_ANTIJAM

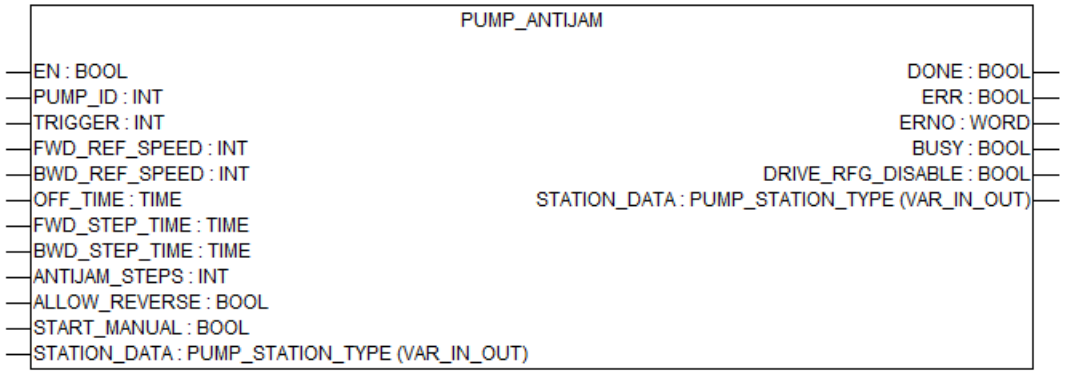


Fig. 664: Function block PUMP_ANTIJAM

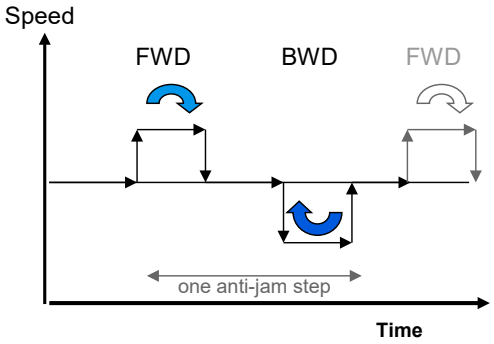
Table 207: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

Performs the pump antijam/cleaning functionality by running the pump at high speeds without any ramp up/down time. For the cleaning function to be performed properly it is important to disable the drive ramp function generator. This can be done if the programmer latches the DRIVE_RFG_DISABLE output with the corresponding bit in the drive control word.

Pump cleaning function can be used to prevent solids from building up on pump impellers or piping. The function consists of a programmable sequence of forward and reverse runs of the pump to shake off any residue on the impeller or piping. This is especially useful with booster and wastewater pumps.

The cleaning sequence can be programmed to occur at suitable intervals, or whenever certain triggering conditions are met.



Not all pumps can be rotated in the reverse direction.

> *For the cleaning function to be performed properly it is important to disable the drive ramp function generator. This can be done if the programmer latches the DRIVE_RFG_DISABLE output with the corresponding bit in the drive control word.*

Input description

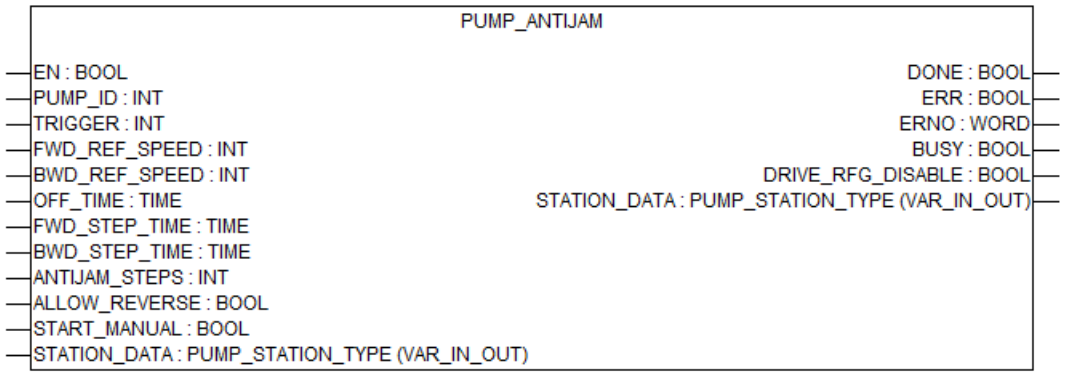


Fig. 665: Function block PUMP_ANTIJAM

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID (id of pump to be cleaned)

Data type: INT, default value: 1, range: 1-3

Enter ID of the pump which is to be cleaned.

TRIGGER (trigger)

Data type: INT, default value: 0, range: 0-4

Trigger to clean the pump.

Mode	Short description	Description
0	Not enbaled	Cleaning sequence is disabled.
1	Master enabled	Cleaning sequence is allowed when the drive is master.
2	Follower enabled	Cleaning sequence is allowed when the drive is follower.
3	At start	Cleaning sequence is performed at every start.
4	Manual mode	Cleaning sequence is performed manually. To operate the ANTIJAM in the manual mode, START_MANUAL has to be made FALSE →TRUE.

FWD_REF_SPEED (forward reference speed)

Data type: INT, default value: 1, range: ≥ 0, unit: rpm

Speed reference to clean the pump in the forward direction.

BWD_REF_SPEED (backward reference speed)

Data type: INT, default value: 1, range: ≥ 0, unit: rpm

Speed reference to clean the pump in the backward direction.

OFF_TIME (off time)

Data type: TIME, default value: 10, range: > 0, unit: s

Time delay between the forward and the reverse movement in antijam.

FWD_STEP_TIME (forward step time)

Data type: TIME, default value: 10, range: > 0, unit: s

Time duration of one clean-step for which the pump moves in the forward direction.

BWD_STEP_TIME (backward step time)

Data type: TIME, default value: 10, range: > 0, unit: s

Time duration of one clean-step for which the pump moves in the backward direction.

ANTIJAM_STEPS (antijam steps)

Data type: INT, default value: 1, range: 1-5

Number of steps to be performed in an antijam process. One step of antijam is shown in the figure above.

ALLOW_REVERSE (allow reverse direction)

Data type: BOOL

True value means the pump can also move in the reverse direction to perform the antijam.

START_MANUAL (manual mode)

Data type: BOOL

For the TRUE value it runs the antijam in the manual mode.

STATION_DATA (station data structure)

Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

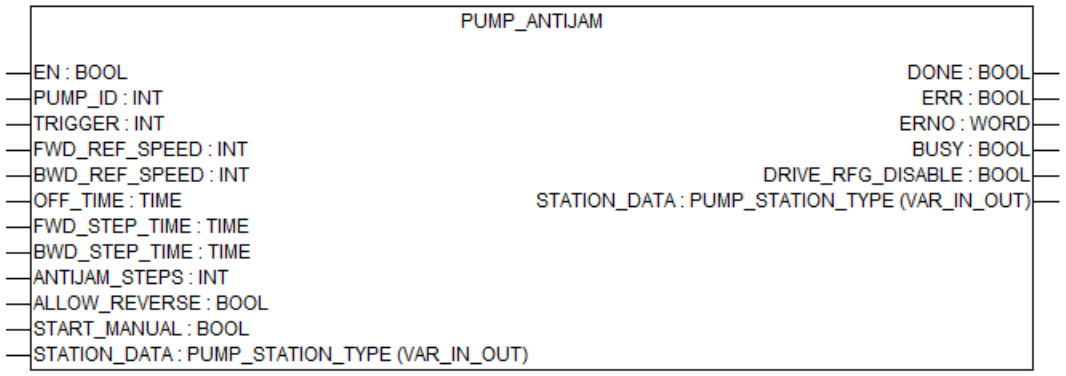


Fig. 666: Function block PUMP_ANTIJAM

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

BUSY (busy)

Data type: BOOL

TRUE value indicates that the function is running.

DRIVE_RFG_DISABLE (drive ramp function generator disable)

Data type: BOOL

Option given to the user bypass the ramp function generator.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of PUMP_ID is erroneous.
16419	4023	Value of PUMP_ID and related setting is erroneous.
16432	4030	Value of TRIGGER is erroneous.
16448	4040	Value of FWD_REF_SPEED is erroneous.
16464	4050	Value of BWD_REF_SPEED is erroneous.
16528	4090	Value of ANTIJAM_STEPS is erroneous.

1.5.13.1.11 PUMP_FLOW_CALC

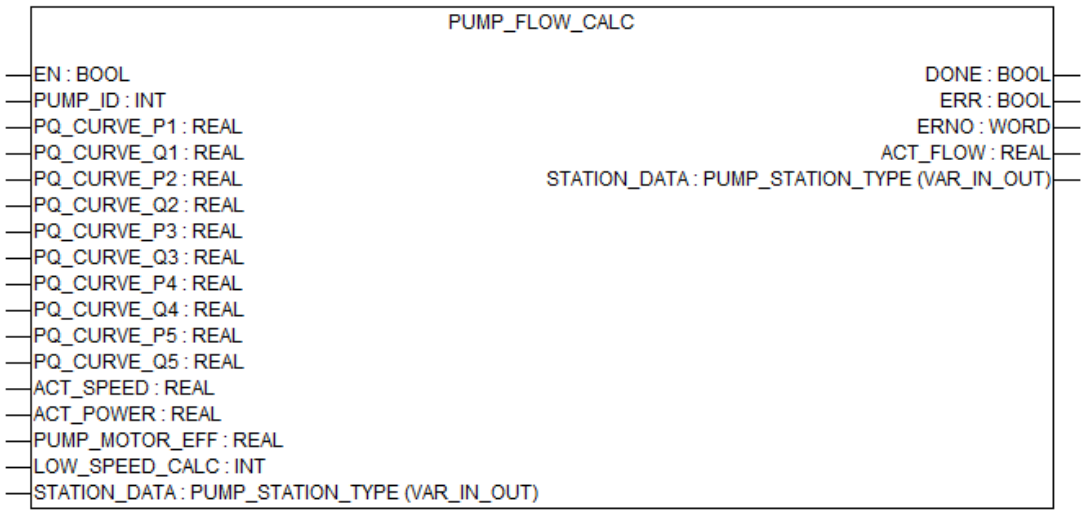


Fig. 667: Function block PUMP_FLOW_CALC

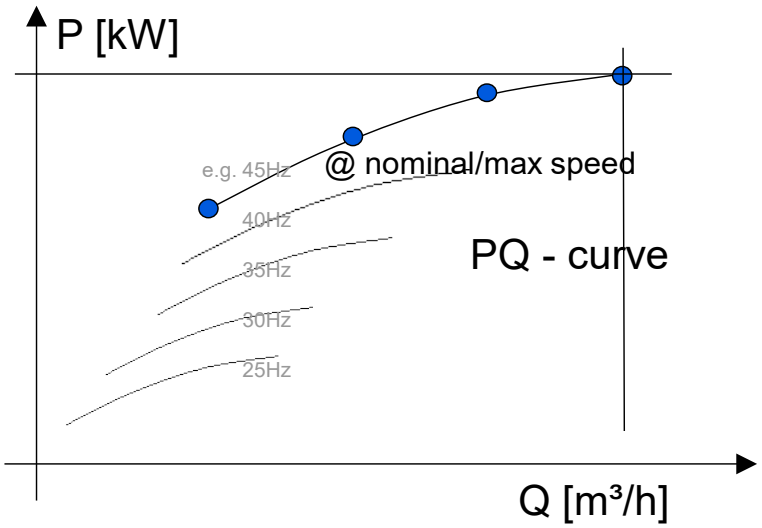
Table 208: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

The flow Calculation Function provides a reasonably accurate (typically ±3…6%) calculation of the flow without the installation of a separate flow meter. The flow is calculated on the basis of pump characteristics.

This function block must be called cyclically with the cycle time of 50ms in the task configuration.

The user can define a PQ (power/flow) performance curve of the pump for the Flow Calculation function. The P (power input) and Q (flow rate) coordinates of five points on the curve are entered. The values are provided by the pump manufacturer. All points defined should lie within the practical operating range of the pump. The points are taken at max operation speed (as entered in the PUMP_INTERFACE block for this pump → the value comes via the connected STATION structure), and scaled down by the block according to the actual speed. By this a good accuracy is achieved with limited parameterization effort.



- *The Flow Calculation function is not to be used for invoicing purposes,*
- *The Flow Calculation function cannot be used outside the normal operating range of the pump given by the pump curves.*

Input description

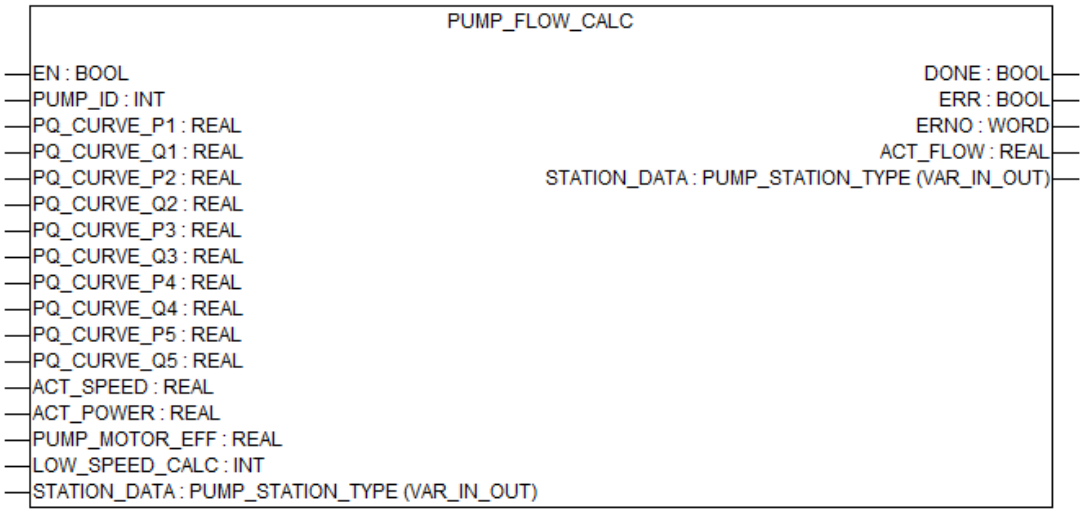


Fig. 668: Function block PUMP_FLOW_CALC

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID (pump number)

Data type: INT, default value: 1, range: 1-3

Pump identification number for which this function block to be called.

PQ_CURVE_P1 (PQ curve P1)

Data type: REAL, default value: 1, range: > 0, unit: kW

Power input of pump at point 1 on the PQ performance curve.

PQ_CURVE_Q1 (PQ curve Q1)

Data type: REAL, default value: 1, range: > 0, unit: m^3/h

Flow rate at point 1 on the PQ performance curve.

PQ_CURVE_P2 (PQ curve P2)

Data type: REAL, default value: 1, range: > 0, unit: kW

Power input of pump at point 2 on the PQ performance curve.

PQ_CURVE_Q2 (PQ curve Q2) Data type: REAL, default value: 1, range: > 0, unit: m^3/h
Flow rate at point 2 on the PQ performance curve.

PQ_CURVE_P3 (PQ curve P3) Data type: REAL, default value: 1, range: > 0, unit: kW
Power input of pump at point 3 on the PQ performance curve.

PQ_CURVE_Q3 (PQ curve Q3) Data type: REAL, default value: 1, range: > 0, unit: m^3/h
Flow rate at point 3 on the PQ performance curve.

PQ_CURVE_P4 (PQ curve P4) Data type: REAL, default value: 1, range: > 0, unit: kW
Power input of pump at point 4 on the PQ performance curve.

PQ_CURVE_Q4 (PQ curve Q4) Data type: REAL, default value: 1, range: > 0, unit: m^3/h
Flow rate at point 4 on the PQ performance curve.

PQ_CURVE_P5 (PQ curve P5) Data type: REAL, default value: 1, range: > 0, unit: kW
Power input of pump at point 5 on the PQ performance curve.

PQ_CURVE_Q5 (PQ curve Q5) Data type: REAL, default value: 1, range: > 0, unit: m^3/h
Flow rate at point 4 on the PQ performance curve.

ACT_SPEED (actual speed) Data type: REAL, default value: 1, range: ≥ 0, unit: rpm
Actual speed of the motor.

ACT_POWER (actual power) Data type: REAL, default value: 1, range: ≥ 0, unit: kW
Actual power of the motor.

PUMP_MOTOR_EFF (pump motor efficiency) Data type: REAL, default value: 1, range: ≥ 0
Pump and the motor combined efficiency.

LOW_SPEED_CALC (low speed calculation) Data type: INT, default value: 1, range: ≥ 0, unit: rpm
Speed below which the calculation will not take place.

STATION_DATA (station data structure) Data type: PUMP_STATION_TYPE
This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

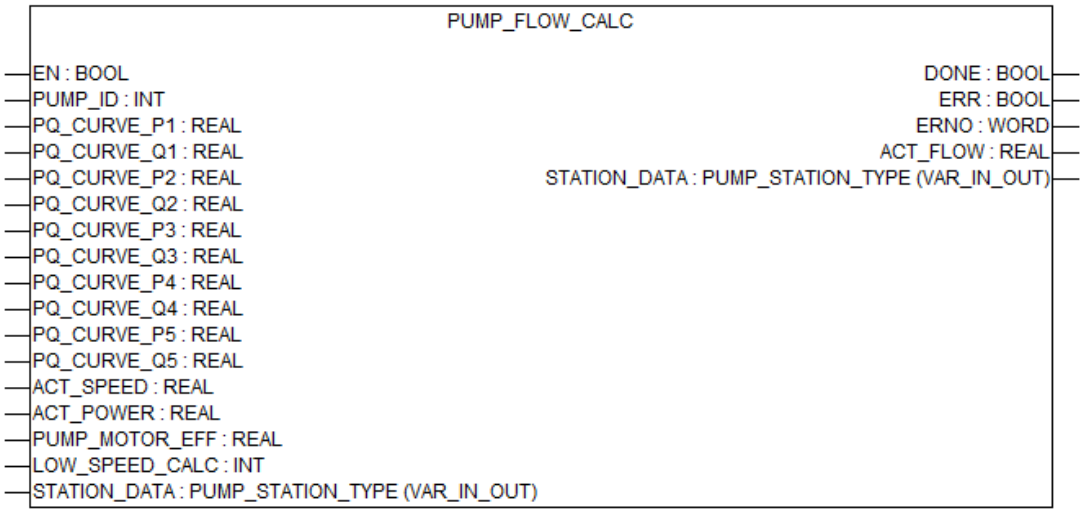


Fig. 669: Function block PUMP_FLOW_CALC

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ACT_FLOW (actual flow)

Data type: REAL, default value: 0, range: ≥ 0, unit: m^3/h

Calculated actual flow of the pump. It is calculated using PQ (power/flow) performance curve of the pump.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of PUMP_ID is erroneous.
16435	4033	Value of PQ_CURVE_P1 to PQ_CURVE_P5 is erroneous.
16541	4043	Value of PQ_CURVE_Q1 to PQ_CURVE_Q5 is erroneous.
16640	4100	Value of PUMP_MOTOR_EFF is erroneous.
16656	4110	Value of LOW_SPEED_CALC is erroneous.

1.5.13.1.12 PUMP_SLEEP

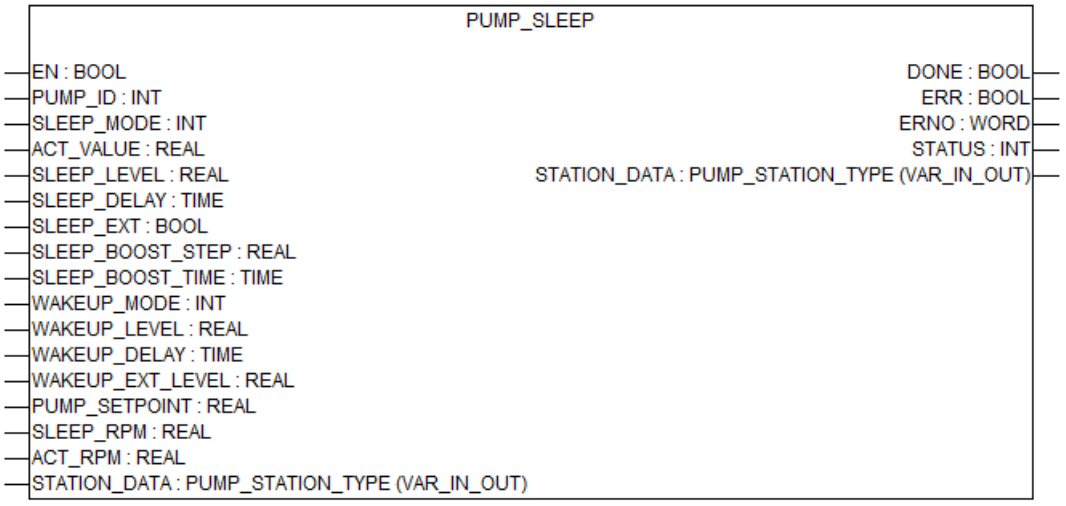


Fig. 670: Function Block PUMP_SLEEP

Table 209: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

The sleep function is suitable for PID control applications where the consumption varies, such as clean water pumping systems. When used, it stops the pump completely during low demand, instead of running the pump slowly below its efficient operating range.

The water consumption falls at night. As a consequence, the process PID controller decreases the motor speed. However, due to natural losses in the pipes and the low efficiency of the centrifugal pump at low speeds, the motor would never stop rotating. The sleep function detects the slow rotation and stops the unnecessary pumping after the sleep delay has passed. The drive shifts into sleep mode, still monitoring the pressure. The pumping resumes when the pressure falls under the predefined minimum level and the wake-up delay has passed.

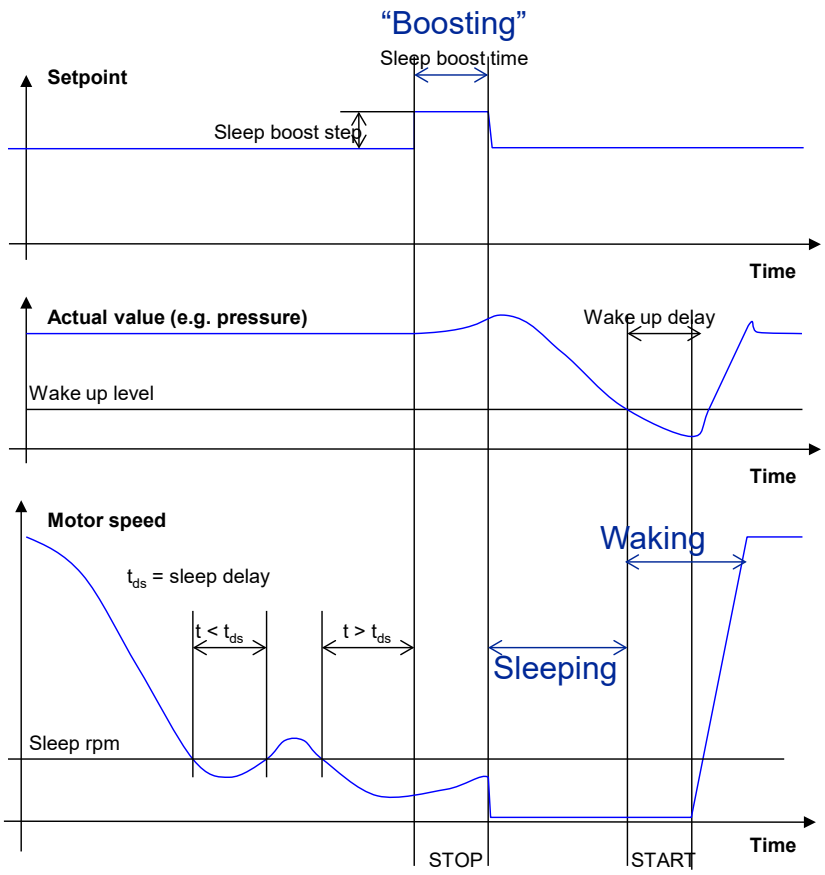


Fig. 671: Visualization of operation and parameters

Input description

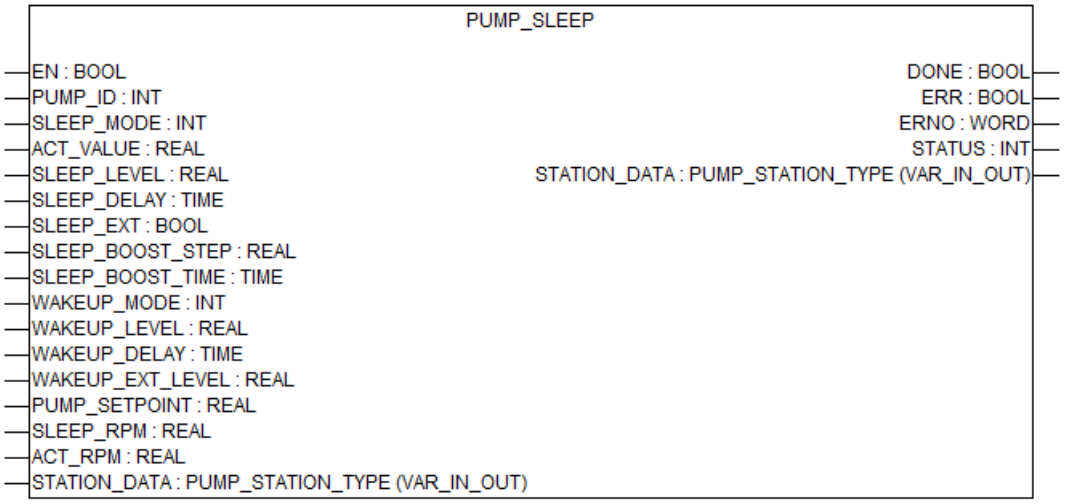


Fig. 672: Function block PUMP_SLEEP

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID (pump number)

Data type: INT, default value: 1, range: 1-3

Pump identification number for which this function block to be called.

SLEEP_MODE Data type: INT, default value: 1, range: 0-4

Table 210: Sleep mode

Mode	Short description	Description
0	Not Used	
1	Internal	Actual value compared with sleep level and if sleep delay is elapsed then pump goes in sleep mode.
2	External	Sleep function is activated using sleep external parameter value.
3	Internal and External	If Sleep_External is true, the Sleep Mode will follow the Internal Mode.
4	Soft External	If Sleep_External is true, the input of the PID controller will be set to 0. After the pump enters sleep mode, it will not wake up until the Sleep_External becomes false.

ACT_VALUE (actual value input of PID block)

Data type: REAL, default value: 1, range: 0-100, unit: %

Actual value of the process variable. This value is scaled as percentage values and attached to this variable. This is the same input which goes as the actual value into the PID.

SLEEP_LEVEL (sleep level)

Data type: REAL, default value: 1, range: 0-100, unit: %

This input defines the level to activate the sleep function. When actual value is equal to sleep level and pump rpm is below sleep rpm, then system waits for sleep delay. Once sleep delay is elapsed pump goes in sleep.

SLEEP_DELAY (sleep delay)

Data type: TIME, default value: 5, range: ≥ 0, unit: s

Time delay for the sleep function to activate. If the actual RPM value is below the parameter input SLEEP_RPM, the system will check if this condition stays for the duration of SLEEP_DELAY and then goes in to sleep-mode (shuts off).

SLEEP_EXTERNAL (sleep external)

Data type: BOOL

This input is used to enable the sleep function externally via a digital input signal

SLEEP_BOOST_STEP (sleep boost step)

Data type: REAL, default value: 1, range: 0-100, unit: %

If the drive enters sleep mode, the setpoint will be increased by this percentage for the time defined by sleep boost time. This allows the optimization of the resulting duty cycle by the sleep function.

SLEEP_BOOST_TIME (sleep boost time)

Data type: TIME, default value: 5, range: ≥ 0, unit: s

Time for which the sleep boost step will be used as an additional setpoint to the pump.

WAKEUP_MODE (wakeup mode) Data type: INT, default value: 1, range: 0-3

Table 211: Wakeup mode

Mode	Short description	Description
0	Wake >Ref	If the actual value remains below setpoint multiplied by the wakeup level for longer than the wakeup delay, the pump will wake up.
1	Wake <Ref	If the actual value remains above setpoint multiplied by the wakeup level for longer than the wakeup delay, the pump will wake up.
2	Wake >Ext	If the wakeup external level remains below actual value for longer than the wakeup delay, the pump will wake up.
3	Wake <	If the wakeup external level remains above actual value for longer than the wakeup delay, the pump will wake up.

WAKEUP_LEVEL (wakeup level) Data type: REAL, default value: 1, range: 0-100, unit: %

Defines the level to activate the wakeup function. It is a percentage of setpoint.

WAKEUP_DELAY (wakeup delay time) Data type: TIME, default value: 5, range: ≥ 0, unit: s

Time delay for the wakeup function to activate.

WAKEUP_EXTERNAL_LEVEL (wakeup external level) Data type: REAL, default value: 1, range: 0-100, unit: %

Wakeup external level. For wake up mode 2 and 3. This value is compared by actual value.

PUMP_SET-POINT (pump set point) Data type: REAL, default value: 1, range: 0-100, unit: %

Process setpoint - the same setpoint connected in the PUMP_PID function block.

SLEEP_RPM (sleep rpm) Data type: REAL, default value: 100, range: ≥ 0, unit: rpm

Pump RPM below which pump should go in to the sleep mode. At very low speed pump efficiency is very low and piping losses are large. By setting this parameter, we can initiate sleep function at lower pump RPM.

ACT_RPM (actual rpm) Data type: REAL, default value: 1, range: ≥ 0, unit: rpm

Actual RPM of the pump. This is compared by sleep RPM to initiate sleep function.

STATION_DATA (station data structure) Data type: PUMP_STATION_TYPE

This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

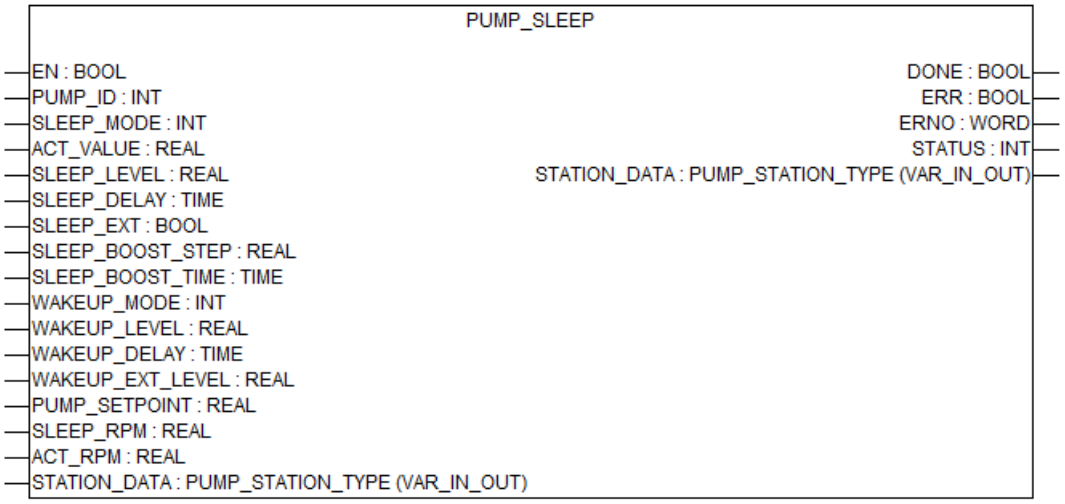


Fig. 673: Function block PUMP_SLEEP

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during data reception. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

STATUS (status) Data type: INT, default value: 0, range: 0-3

Table 212: Status of Sleep Function

Mode	Description
0	Function is inactive.
1	Sleep Function is active.
2	Boost Function is active.
3	Wakeup Function is active.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of PUMP_ID is erroneous
16432	4030	Value of SLEEP_MODE is erroneous.
16448	4040	Value of ACT_VALUE is erroneous.
16464	4050	Value of SLEEP_LEVEL is erroneous.
16480	4060	Value of SLEEP_DELAY is erroneous.
16512	4080	Value of SLEEP_BOOST_STEP is erroneous.
16528	4090	Value of SLEEP_BOOST_TIME is erroneous.
16544	40A0	Value of WAKEUP_MODE is erroneous.
16560	40B0	Value of WAKEUP_LEVEL is erroneous.
16576	40C0	Value of WAKEUP_DELAY is erroneous.
16592	40D0	Value of WAKEUP_EXTERNAL_LEVEL is erroneous.
16608	40E0	Value of PUMP_SETPOINT is erroneous.
16624	40E0	Value of SLEEP_RPM is erroneous.

1.5.13.1.13 PUMP_PROTECTION

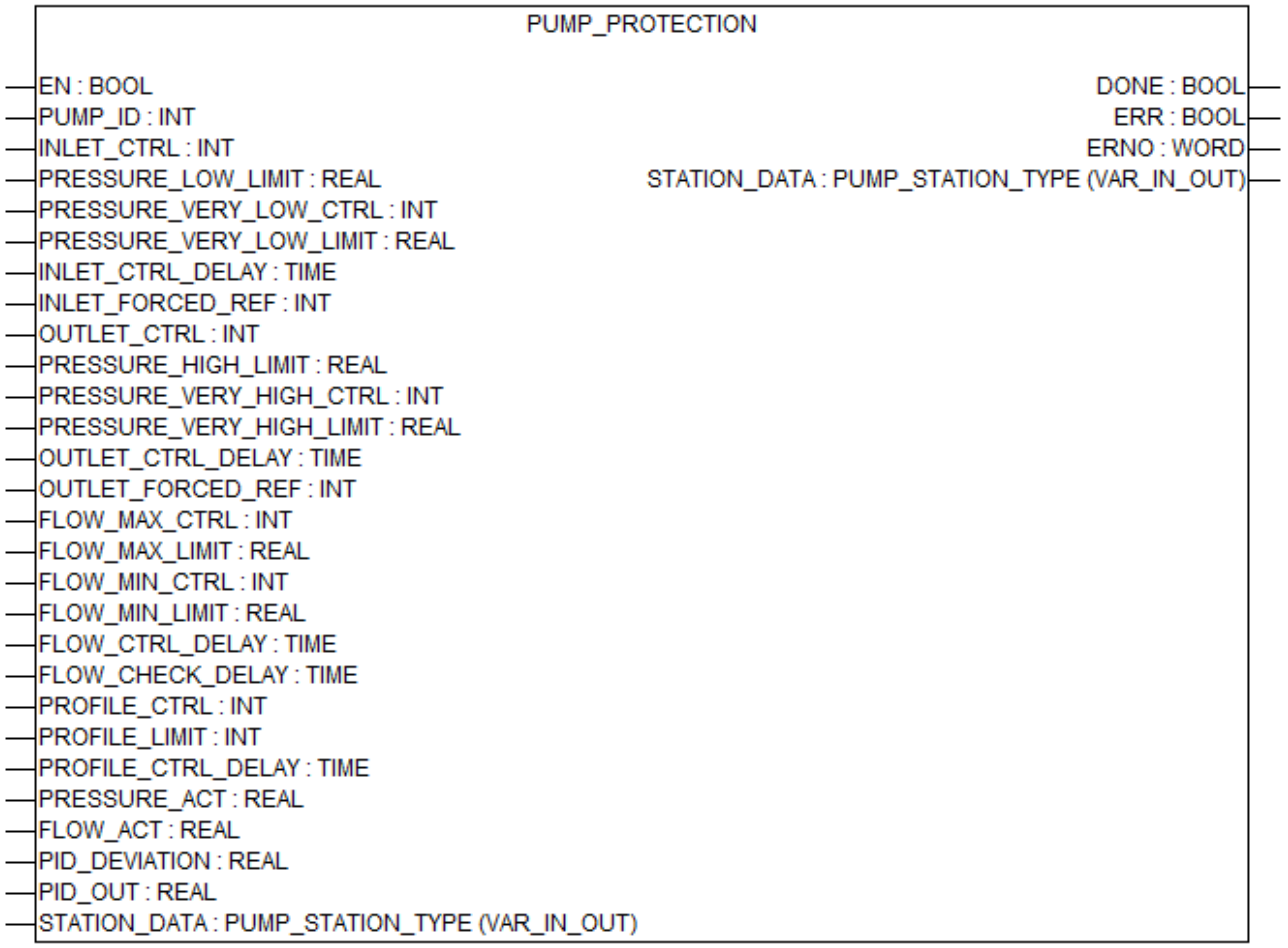


Fig. 674: Function block PUMP_PROTECTION

Table 213: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block contains two level protection for the inlet and outlet pressure monitoring. The actual pressure for the comparison is an analog input. In the first level primary protection, the drive indicates depending on the chosen mode: A warning, trips on a fault, or starts to follow a predefined reference. In the second level secondary protection, the drive either will stop or trip on a fault. For flow and the profile monitoring there is only one level of protection, in which the drive responds as per the control selected.

Example

If for the inlet the primary (first limit) protection for the low pressure is set, the INLET_CTRL mode is set to 3. Lower pressures than given in the limit are measured: The station will run in the reduced speed mode. In that case the PID output is frozen (in the PID). The current master gets the reduced speed INLET_FORCED_REF. This new ref speed of the master will decide whether to run followers or not.

Flow Monitoring

The control program has a monitoring function for flow that can be configured to generate an alarm or a fault whenever the flow falls below or rises above predefined limits. The flow can either be calculated or measured using a flow meter connected to, for example, an analog input.

Application Profile Monitoring

The application profile monitoring function can be used for long-term supervision of an actual signal. If the selected signal remains above the supervision limit for a specified time, an alarm will be generated.

Input description

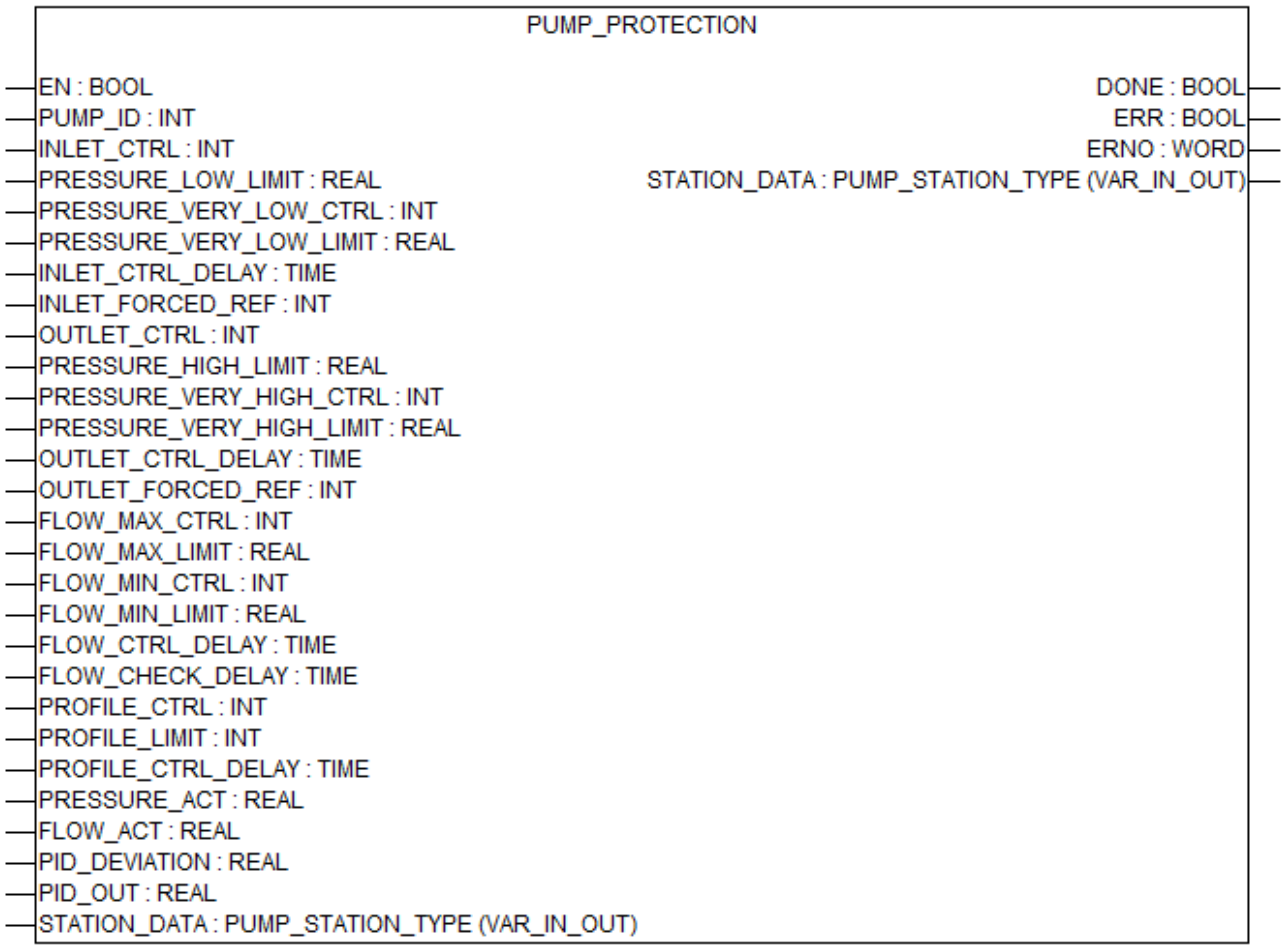


Fig. 675: Function block PUMP_PROTECTION

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID (pump identification number)

Data type: INT, default value: 1, range: 1-3

Identifies which function block instance is used for which physical pump.

INLET_CTRL (inlet control)

Data type: INT, default value: 0, range: 0-3

Table 214: To enable primary protection

Mode	Short description	Description
0	Disabled	Primary inlet pressure supervision is not used.
1	Alarm	Detection of low inlet pressure produces an alarm after the inlet control delay.
2	Fault	Detection of low inlet pressure trips drive after the inlet control delay.
3	Reduced speed	Detection of low inlet pressure produces an alarm after the inlet control delay. The Pump speed is reduced to the speed specified in inlet forced reference.

PRESSURE_LOW_LIMIT (pressure low limit)

Data type: REAL, default value: 1, range: 0-100, unit: %

Low pressure level limit for primary protection.

PRESSURE_VERY_LOW_CTRL (pressure very low control)

Data type: INT, default value: 1, range: 0-2

Table 215: To enable pressure very low

Mode	Short description	Description
0	Disabled	Secondary inlet pressure supervision is not used.
1	Fault	Detection of very low inlet pressure trips drive.
2	Stop	Detection of very low inlet pressure stops drive. The drive will restart, if pressure rises above PRESSURE_VERY_LOW_LIMIT value.

PRESSURE_VERY_LOW_LIMIT (pressure very low limit)

Data type: REAL, default value: 1, range: 0-100, unit: %

Very low pressure level limit at secondary protection.

INLET_CTRL_DELAY (inlet control delay)

Data type: TIME, default value: 1, range: 0-600, unit: s

Delay for primary (actual pressure less than the low pressure limit) and secondary supervision (actual pressure less than the very low pressure limit) of pump inlet pressure.

Range: 0 to 600 seconds .

INLET_FORCED_REF (inlet forced reference)

Data type: INT, default value: 1, range: ≥ 0, unit: rpm

Inlet forced reference for very low pressure. Detection of low inlet pressure produces an alarm after the inlet control delay. The pump speed is reduced to the speed specified in inlet forced reference.

OUTLET_CTRL (outlet control)

Data type: INT, default value: 1, range: 0-3

Table 216: To enable outlet protection

Mode	Short description	Description
0	Disabled	Primary outlet pressure supervision is not used.
1	Alarm	Detection of high outlet pressure produces an alarm after the outlet control delay.
2	Fault	Detection of high outlet pressure trips drive after the outlet control delay.
3	Reduced speed	Detection of high outlet pressure produces an alarm after the outlet control delay. Pump speed: Reduced to the speed specified in outlet forced reference.

PRESSURE_HIGH_LIMIT (pressure high limit)

Data type: REAL, default value: 1, range: 0-100, unit: %

High pressure level limit for primary protection.

PRESSURE_VERY_HIGH_CTRL (pressure very high control)

Data type: INT, default value: 1, range: 0-2

Table 217: To enable pressure very high control

Mode	Short description	Description
0	Disabled	Secondary outlet pressure supervision is not used.
1	Fault	Detection of very high outlet pressure trips drive.
2	Stop	Detection of very high outlet pressure stops drive. The drive will restart, if pressure rises above limit.

PRESSURE_VERY_HIGH_LIMIT (pressure very high limit)

Data type: REAL, default value: 1, range: 0-100, unit: %

Very high pressure level limit at secondary protection.

OUTLET_CTRL_DELAY (outlet control delay)

Data type: TIME, default value: 1, range: 0-600, unit: s

Delay for primary and secondary supervision of pump outlet pressure.

OUTLET_FORCED_REF (outlet forced reference)

Data type: INT, default value: 1, range: ≥ 0, unit: rpm

Outlet Forced Reference for very high pressure. Detection of high outlet pressure produces an alarm after the outlet control delay. The Pump speed is reduced to the speed specified in Outlet Forced Reference.

FLOW_MAX_CTRL (flow maximum control) Data type: INT, default value: 1, range: 0-2

Mode	Description
0	Protection from high flow rates disabled.
1	Alarms generated – minimum flow.
2	Fault generated - minimum flow, drive trips.

FLOW_MAX_LIMIT (flow maximum limit) Data type: REAL, default value: 0, range: 0-32767, unit: m^3/h

Value of max flow rate limit in m3/h.

FLOW_MIN_CTRL (flow minimum control) Data type: REAL, default value: 1, range: 0-2

Mode	Description
0	Protection from low flow rates disabled.
1	Alarms generated – minimum flow.
2	Fault generated - minimum flow, drive trips.

FLOW_MIN_LIMIT (flow minimum limit) Data type: INT, default value: 0, range: 0-32767, unit: m^3/h

Value of minimum flow rate limit.

FLOW_CTRL_DELAY (flow control delay) Data type: TIME, default value: 1, range: 0-12600, unit: s

Delay before executing the control due to low or high flow rates.

FLOW_CHECK_DELAY (flow check delay) Data type: TIME, default value: 1, range: 0-12600, unit: s

Delay after the start of the pump before monitoring the flow rates.

PROFILE_CTRL (profile control) Data type: INT, default value: 1, range: 0-2

Mode	Description
0	Protection from high flow rates disabled.
1	Alarms generated – maximum flow.
2	Fault generated - maximum flow, drive trips.

PROFILE_LIMIT (profile limit) Data type: INT, default value: 1, range: 0-32767

Value of profile protection limit.

PROFILE_CTRL_DELAY (profile control delay) Data type: TIME, default value: 1, range: ≥ 0, unit: s

Time delay before the Profile Control activates.

PRESSURE_ACT (actual pressure)	Data type: INT, default value: 1, range: 0-100, unit: % Actual pressure of the system.
FLOW_ACT (actual flow)	Data type: INT, default value: 1, range: ≥ 0, unit: m^3/h Actual flow of the pump.
PID_DEVIATION (pid deviation)	Data type: REAL, default value: 0.0, range 0-100.0, unit % Deviation of Set value from actual value. It is also known as PID error.
PID_OUT (pid out)	Data type: REAL, default value: 1, range: -100 to +100, unit: % PID Out is generated by PUMP_PID function block. PID control process error value gives output.
STATION_DATA (station data structure)	Data type: PUMP_STATION_TYPE This structure contains pumping station data. All the function blocks receive some data, process it and write it back to the structure.

Output description

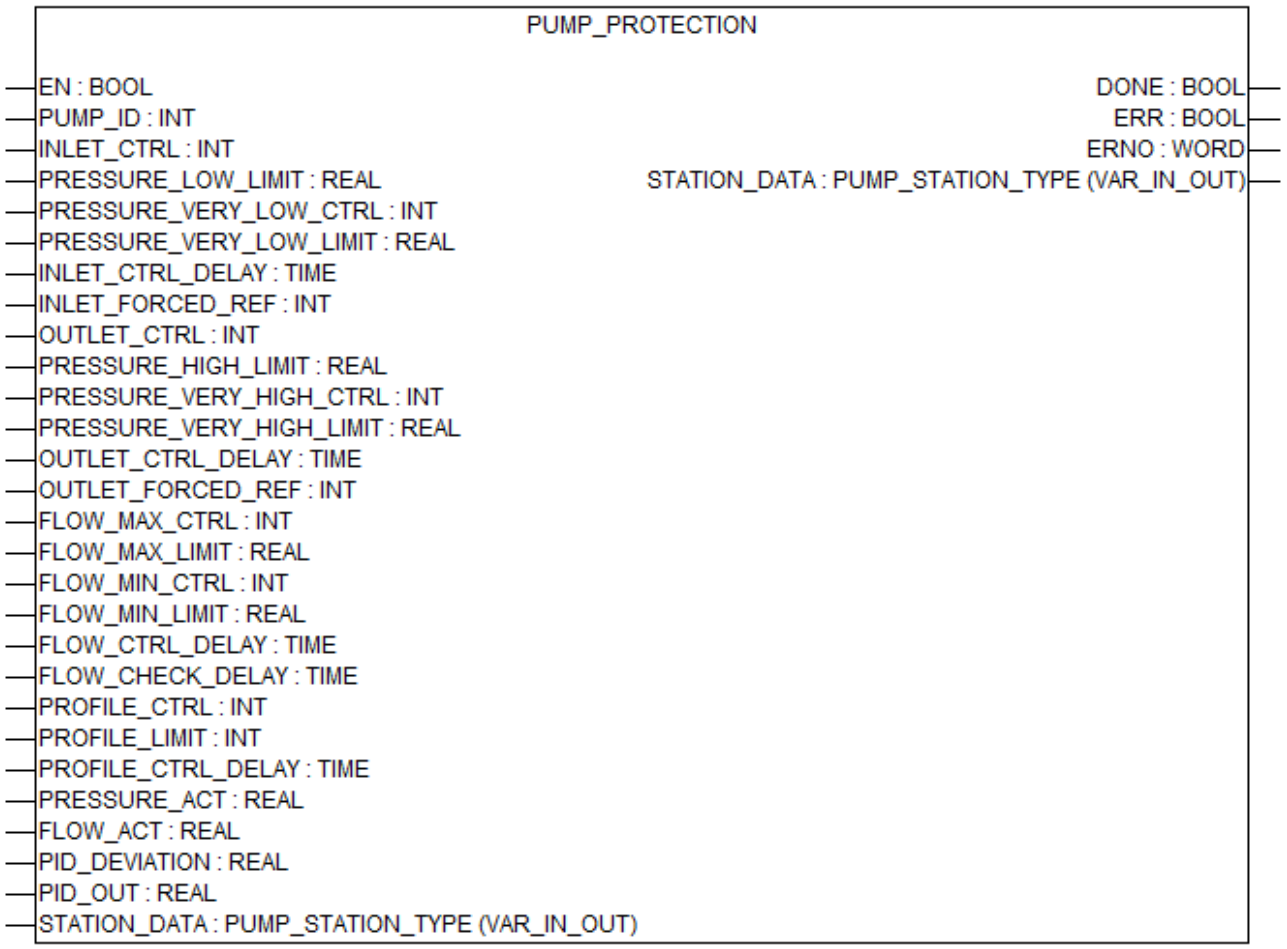


Fig. 676: Function block PUMP_PROTECTION

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of PUMP_ID is erroneous.
16432	4030	Value of INLET_CTRL is erroneous.
16448	4040	Value of PRESSURE_LOW_LIMIT is erroneous.
16451	4043	Value of either PRESSURE_LOW_LIMIT or PRESSURE_VERY_LOW_LIMIT is erroneous.
16464	4050	Value of PRESSURE_VERY_LOW_CTRL is erroneous.
16480	4060	Value of PRESSURE_VERY_LOW_LIMIT is erroneous.
16496	4070	Value of INLET_CTRL_DELAY is erroneous.
16512	4080	Value of INLET_FORCED_REF is erroneous.
16528	4090	Value of OUTLET_CTRL is erroneous.
16544	40A0	Value of PRESSURE_HIGH_LIMIT is erroneous.
16547	40A3	Value of either PRESSURE_HIGH_LIMIT or PRESSURE_VERY_HIGH_LIMIT is erroneous.
16560	40B0	Value of PRESSURE_VERY_HIGH_CTRL is erroneous.
16576	40C0	Value of PRESSURE_VERY_HIGH_LIMIT is erroneous.
16592	40D0	Value of OUTLET_CTRL_DELAY is erroneous.
16608	40E0	Value of OUTLET_FORCED_REF is erroneous.

Dec	Hex	Error Description
16624	40F0	Value of FLOW_MAX_CTRL is erroneous.
16640	4100	Value of FLOW_MAX_LIMIT is erroneous .
16643	4103	Value of either FLOW_MAX_LIMIT or FLOW_MIN_LIMIT is erroneous.
16656	4110	Value of FLOW_MIN_CTRL is erroneous.
16672	4120	Value of FLOW_MIN_LIMIT is erroneous.
16688	4130	Value of FLOW_CTRL_DELAY is erroneous.
16704	4140	Value of FLOW_CHECK_DELAY is erroneous.
16720	4150	Value of PROFILE_CTRL is erroneous.
16736	4160	Value of PROFILE_LIMIT is erroneous.
16752	4170	Value of PROFILE_CTRL_DELAY is erroneous.

1.5.13.1.14 PUMP_ENERGY_CALC

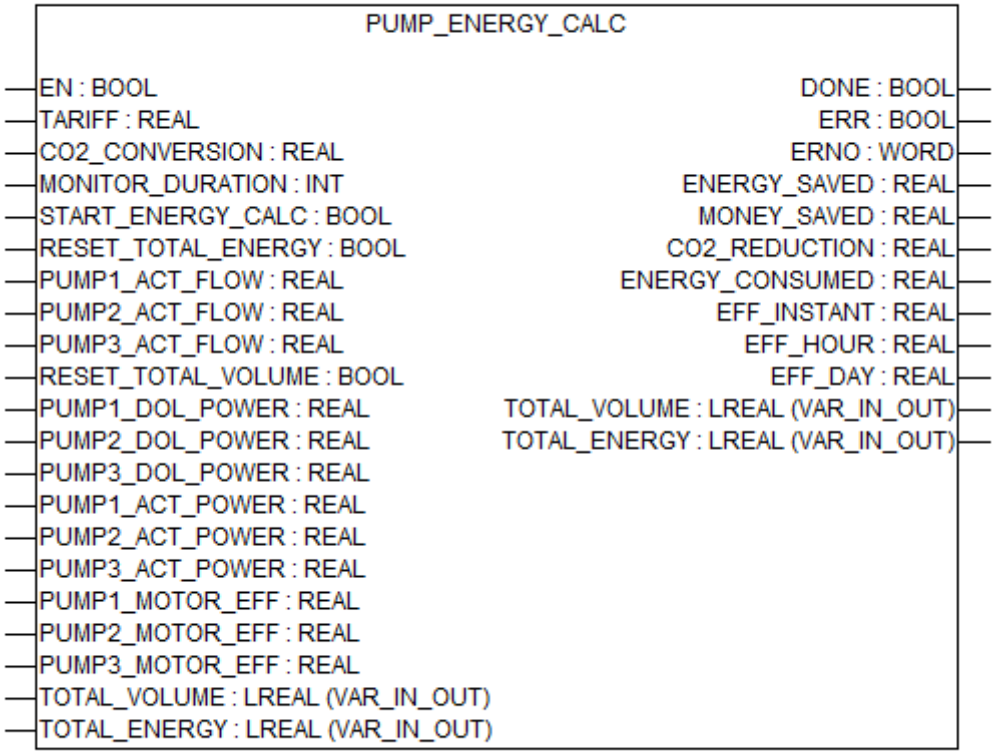


Fig. 677: Function block PUMP_ENERGY_CALC

Table 218: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block calculates energy consumed by direct on line pumps and drive connected pumps. This is then used to calculate saved energy by using drives. Based on energy saved the CO_2 emission reductions and money saved is also computed.

It also calculates efficiency of total flow (= volume) per total power (energy) in m^3/kWh. Instant, hourly and 24 hourly (daily) efficiency are calculated continuously.

Total volume and total energy is calculated and connected as in or out parameter to the function block.

This function block must be called cyclically with the cycle time of 50 ms in the task configuration.

To retain this data, global retain variables need to be connected in the user program.

Input description

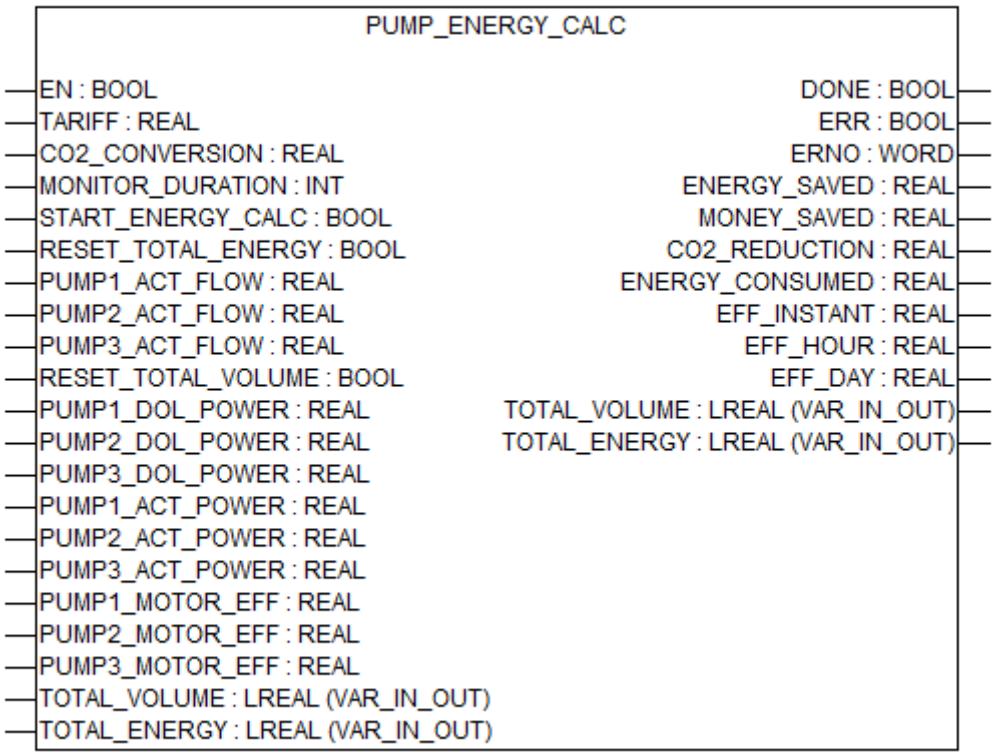


Fig. 678: Function block PUMP_ENERGY_CALC

> *The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.*

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

TARIFF (tariff) Data type: REAL, default value: 0, range: ≥ 0, unit: currency (e.g. Dollar, Euro) /kWh

This is used with energy saved to calculate money saved by using drives (instead of DOL).

CO2_CONVERSION (CO_2 conversion) Data type: REAL, default value: 0, range: ≥ 0

Factor which helps in converting the energy consumed in terms of CO_2 emission tn/MWh.

MONITOR_DURATION (monitor duration)
Data type: INT, default value: 0, range: > 0, unit: min
Time duration to calculate the energy consumed.

START_ENERGY_CALC (start energy calculation)
Data type: BOOL
The energy calculation will start if the input is TRUE. If the input becomes FALSE the energy calculation will stop.

RESET_TOTAL_ENERGY (reset total energy)
Data type: BOOL
If this input is TRUE it will reset all the stored values zero. If this input is FALSE the stored values will get updated continuously.

PUMP1_ACT_FLOW (pump 1 actual flow)
Data type: REAL, default value: 0, range: ≥ 0, unit: m^3/h
Actual flow of PUMP 1 is connected to this parameter. Actual flow can be measured using standard flow meters or theoretically calculated using PQ curves of function block PUMP_FLOW_CALC.

PUMP2_ACT_FLOW (pump 2 actual flow)
Data type: REAL, default value: 0, range: ≥ 0, unit: m^3/h
Actual flow of PUMP 2 is connected to this parameter. Actual flow can be measured using standard flow meters or theoretically calculated using PQ curves of function block PUMP_FLOW_CALC.

PUMP3_ACT_FLOW (pump 3 actual flow)
Data type: REAL, default value: 0, range: ≥ 0, unit: m^3/h
Actual flow of PUMP 3 is connected to this parameter. Actual flow can be measured using standard flow meters or theoretically calculated using PQ curves of function block PUMP_FLOW_CALC.

RESET_TOTAL_VOLUME (reset total volume)
Data type: BOOL
Total volume is calculated using the flow of the available pumps. If this input is TRUE it will reset stored total volume value to zero. If this input is FALSE the stored values will get updated continuously.

PUMP1_DOL_POWER (pump 1 dol power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump1 Direct On Line power, if directly connected to the motor.

PUMP2_DOL_POWER (pump 2 dol power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump2 Direct On Line power, if directly connected to the motor.

PUMP3_DOL_POWER (pump 3 dol power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump3 Direct On Line power, if directly connected to the motor.

PUMP1_ACT_POWER (pump 1 actual power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump1 actual power read from the drive.

PUMP2_ACT_POWER (pump 2 actual power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump2 actual power read from the drive.

PUMP3_ACT_POWER (pump 3 actual power)
Data type: REAL, default value: 0, range: ≥ 0, unit: kW
Pump3 actual power read from the drive.

PUMP1_MOTOR_EFF (pump 1 motor efficiency)
Data type: REAL, default value: 0, range: 0-100.0, unit: %
Motor efficiency in % of Pump1.

PUMP2_MOTOR_EFF (pump 2 motor efficiency)
Data type: REAL, default value: 0, range: 0-100.0, unit: %
Motor efficiency in % of Pump2.

PUMP3_MOTOR_EFF (pump 3 motor efficiency)
Data type: REAL, default value: 0, range: 0-100.0, unit %
Motor efficiency in % of Pump3.

TOTAL_VOLUME (total volume)
Data type: LREAL, default value: -, range: ≥ 0, unit: m^3
Total volume is consumed, which is calculated using flow of all the pumps. It is cumulative and the user can connect global retain variable in program can retain this value.

TOTAL_ENERGY (total energy)
Data type: LREAL, default value: -, range: ≥ 0, unit: kWh
Total energy which was consumed, calculated using actual power of all pumps. It is cumulative and the user should connect global retain variable in program to retain this value.

Output description

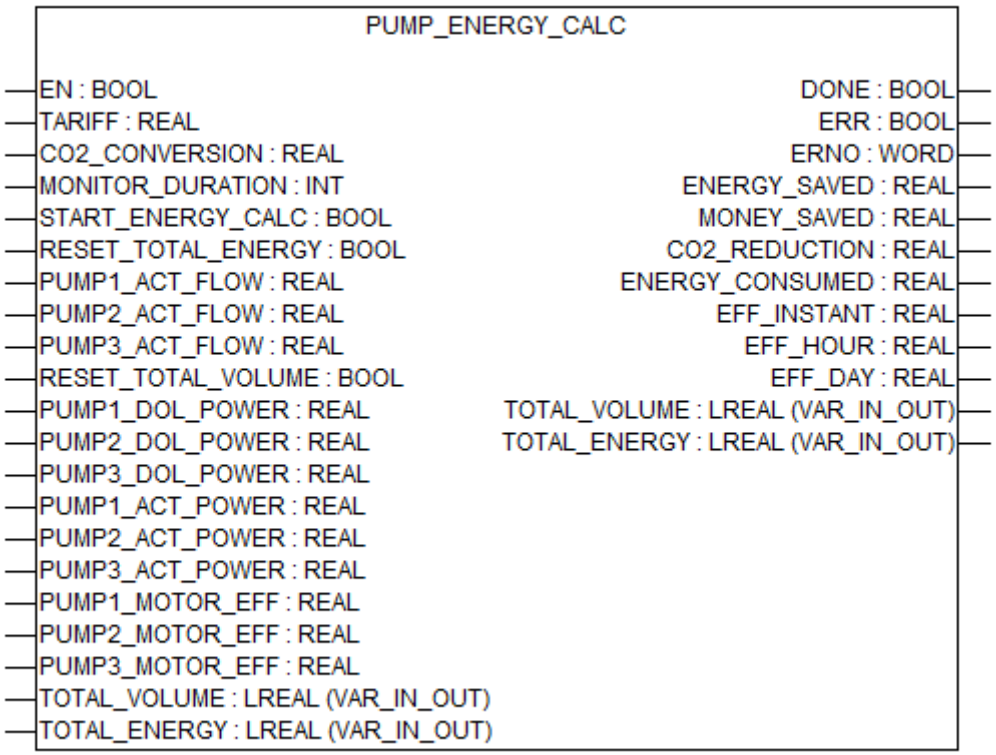


Fig. 679: Function block PUMP_ENERGY_CALC

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ENERGY_SAVED (energy saved)

Data type: REAL, default value: 0, range: ≥ 0, unit: kWh

Stores the energy saved.

Difference of the energy consumed, considering the pumps motor connected directly to the grid (DOL), compared to the using a drive to control the motor.

MONEY_SAVED (money saved)

Data type: REAL, default value: 0, range: ≥ 0, unit: currency (e.g. Dollar, Euro)

Money saved is calculated by multiplying ENERGY_SAVED and TARIFF. Its unit is the same unit as TARIFF.

CO2_REDUCTION (CO_2 reduction)

Data type: REAL, default value: 0, range: ≥ 0, unit: t/kWh

Indicates the reduction in carbon dioxide emissions.

ENERGY_CONSUMED (energy consumed)

Data type: REAL, default value: 0, range: ≥ 0, unit: kWh

Indicates the value of energy consumed in the monitoring duration. While input TOTAL_ENERGY indicates cumulative value of energy consumed.

EFF_INSTANT (instant efficiency)

Data type: REAL, default value: 0, range: ≥ 0

Instant flow power efficiency of the system in m^3/kWh.

EFF_HOUR (hourly efficiency)

Data type: REAL, default value: 0, range: ≥ 0

Hourly flow power efficiency of the system in m^3/kWh.

EFF_DAY (daily efficiency)

Data type: REAL, default value: 0, range: ≥ 0

Daily flow power efficiency of the system in m^3/kWh.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error.
16416	4020	Value of TARIFF is erroneous.
16432	4030	Value of CO2_CONVERSION is erroneous.

Dec	Hex	Error Description
16448	4040	Value of MONITOR_DURATION is erroneous.
16560	40B0	Value of PUMP1_DOL_POWER is erroneous.
16576	40C0	Value of PUMP2_DOL_POWER is erroneous.
16592	40D0	Value of PUMP3_DOL_POWER is erroneous.
16656	4110	Value of PUMP1_MOTOR_EFF is erroneous.
16672	4120	Value of PUMP2_MOTOR_EFF is erroneous.
16688	4130	Value of PUMP3_MOTOR_EFF is erroneous.

1.5.13.1.15 Pump_DOL_SIMU

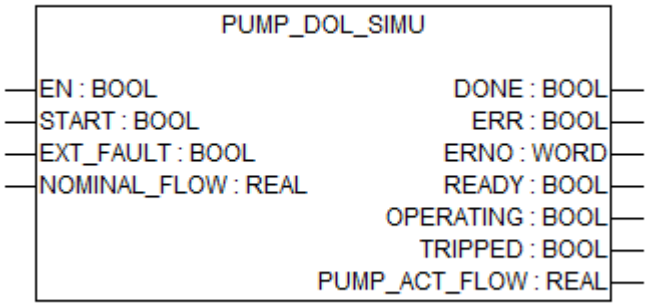


Fig. 680: Function block PUMP_DOL_SIMU

Table 219: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block simulates a Direct Online Pump (DOL) without a drive . The function block generates the feedback signals READY, OPERATING and TRIPPED which have to be connected to the PUMP_INTERFACE function block . The trip signal is simulated by forcing the input EXT_FAULT. The input value “NOMINAL_FLOW" is the nominal (rated) flow of the pump which is forwarded to the output “PUMP_ACT_FLOW” if the pump is in operating condition. The output “PUMP_ACT_FLOW” is zero if the pump is off or tripped.

The function block simulates a Direct Online (DOL) pump in boost control applications traditional pumping

The function block simulates the pump only in general behavior. The behavior in the real process may be slightly different Outputs of this function block may or may not be as accurate as real process.

Input description

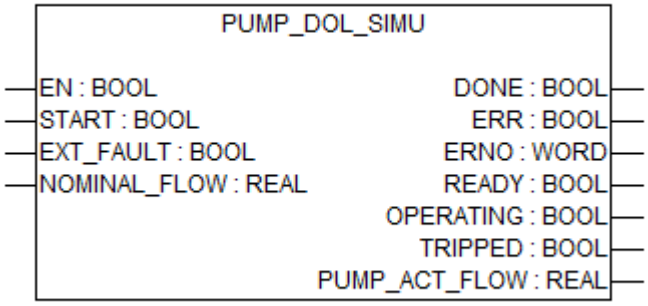


Fig. 681: Function block PUMP_DOL_SIMU

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START (start)

Data type: BOOL, default value: FALSE

The input is connected to the pump start command. The start command is activated by the PUMP_START output of the PUMP_INTERFACE function block.

TRUE starts the DOL pump and the output “OPERATING” is TRUE. FALSE stops the pump and the output “OPERATING” is FALSE The output “READY” keeps TRUE independent of the input state.

EXT_FAULT (external fault)

Data type: BOOL, default value: FALSE

This input simulates the trip condition. The user can trigger the input signal “EXT_FAULT".

TRUE activate a pump trip; the output ”TRIPPED” is TRUE and stops the pump; the output “OPERATING” is FALSE; the output “READY” is FALSE

FALSE deactivate a trip the output “TRIPPED” is FALSE.

NOM-INAL_FLOW (nominal flow)

Data type: REAL, default value: 0, range >0, unit: m^3/ h

This input parameter has to be designed according the nominal flow of the pump.

Output description

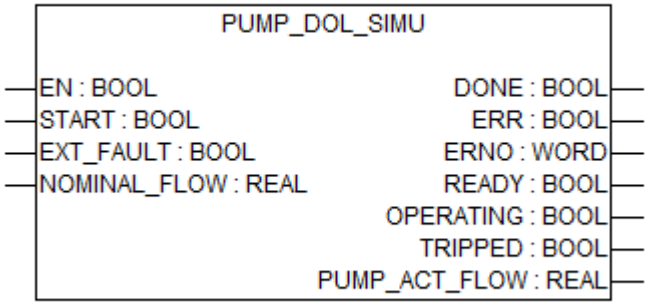


Fig. 682: Function block PUMP_DOL_SIMU

DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

READY (ready)

Data type: BOOL

TRUE indicates that the pump is ready to start and there is no fault (input “EX_FAULT” is FALSE).

OPERATING (operating)

Data type: BOOL

TRUE indicates that the pump is operating.

TRIPPED (tripped)

Data type: BOOL

TRUE indicates the pump is tripped (input “EX_FAULT” is TRUE)

PUMP_ACT_FLOW (pump actual flow)

Data type: REAL, unit: m^3/h

Actual flow of the Direct Online (DOL) pump.

The output value corresponds to the input parameter “NOMINAL FLOW” if the pump is running.

The output value is zero if the pump is stopped.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error
16448	4040	NOMINAL_FLOW is less than zero

1.5.13.1.16 PUMP_DRIVE_SIMU

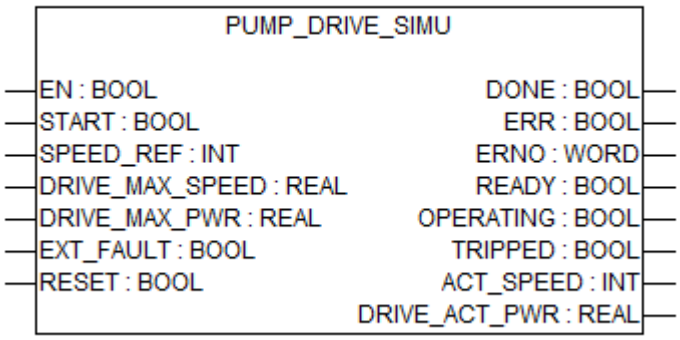


Fig. 683: Function block PUMP_DRIVE_SIMU

Table 220: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block simulates a drive. In combination with the function block PUMP_FLOW_CALC a drive controlled pump can be simulated.

The function block generates the feedback signals READY, OPERATING and TRIPPED which have to be connected to the PUMP_INTERFACE function block. The output ACT_SPEED and DRIVE_ACT_PWR can be connected to the PUMP_FLOW_CALC function block. The trip signal is simulated by forcing the input EXT_FAULT

> *The function block simulates a Drive controlled pump in level control or boost control applications multimode or traditional pumping*
>
> *The function block simulates the Drive controlled pump only in general behavior. The behavior in the real process may be slightly different.*

Input description

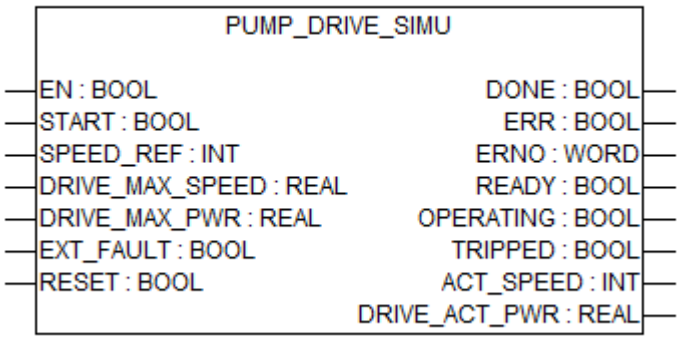


Fig. 684: Function block PUMP_DRIVE_SIMU

EN (enable)

Data type: BOOL, default value: FALSE

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

START (start)

Data type: BOOL, default value: FALSE

The input is connected to the pump start command. The start command is activated by the PUMP_INTERFACE function block.

TRUE starts the pump and the output “OPERATING” is TRUE.

FALSE stops the pump the output “OPERATING” is FALSE.

The output “READY” keeps TRUE independent of the input state.

SPEED_REF (speed reference)

Data type: INT, default value: 0, range: > 0, unit: RPM

The input is connected to the pump reference speed.

The pump reference speed is generated by the application and connected from the PUMP_INTERFACE function block.

DRIVE_MAX_SPEED (drive maximum speed) Data type: REAL, default value: 1500, range: > 0, unit: RPM

This input parameter has to be designed according the nominal speed of the pump.

DRIVE_MAX_PWR (drive maximum power) Data type: REAL, default value: 5, range: > 0, unit: kW

This input parameter has to be designed according the nominal power of the pump.

Maximum rated power of the drive.

EXT_FAULT (external fault) Data type: , BOOL, default value: FALSE

This input simulates the trip condition. The user can trigger the input signal “EXT_FAULT”

TRUE activate a pump trip; the output ”TRIPPED” is TRUE and stops the pump; the output “OPERATING” is FALSE;

When the input signal “EXT_FAULT” changes back to FALSE it does not deactivate the TRIPPED status. This can only be done using the RESET input explained below.

RESET (reset) Data type: , BOOL, default value: FALSE

This input can be used to reset the TRIPPED status on the drive.

When the RESET changes from FALSE to TRUE, the output TRIPPED becomes FALSE in the absence of EXT_FAULT. However RESET does not deactivate the TRIPPED status when the EXT_FAULT is already present

Output description

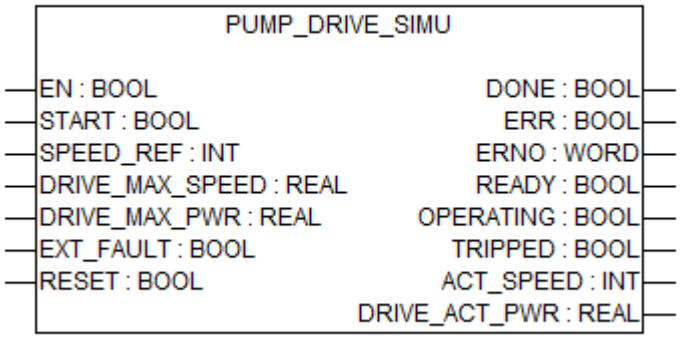


Fig. 685: Function block PUMP_DRIVE_SIMU

DONE (done) Data type: BOOL

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL

Output ERR indicates whether an error occurred during data reception. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number)

Data type: WORD, default value: 0, range: ≥ 0

Output ERNO provides an error identifier if an invalid value was applied to an input. ERNO always has to be considered together with the output ERR. The value output at ERNO is only valid if ERR is TRUE. The error messages encoding at output ERNO are explained at the table below.

READY (ready)

Data type: BOOL

TRUE indicates that the drive is ready to start and there is no fault.

(input “EX_FAULT” is FALSE)

OPERATING (operating)

Data type: BOOL

TRUE indicates that the drive is operating.

TRIPPED (tripped)

Data type: BOOL

TRUE indicates the pump is tripped (input “EX_FAULT” is TRUE).

ACT_SPEED (actual speed)

Data type: REAL , unit: RPM

Actual speed of the drive.

DRIVE_ACT_PWR (drive actual power)

Data type: REAL, unit: kW

Drive calculated power. It is based on the DRIVE_MAX_SPEED and DRIVE_MAX_PWR inputs.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error
16432	4030	SPEED_REF is less than zero.
16448	4040	DRIVE_MAX_SPEED is less or equal than zero.
16464	4050	DRIVE_MAX_PWR is less or equal than zero.

1.5.13.1.17 PUMP_TANK_SIMU

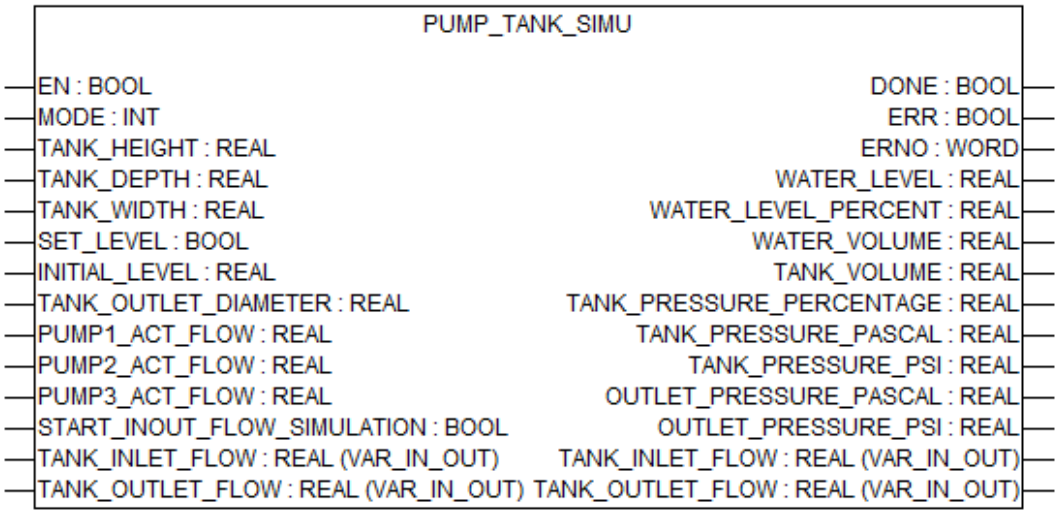


Fig. 686: Function block PUMP_TANK_SIMU

Table 221: General information

Available as of runtime system	V1.3 and above
Included in library	PUMP_AC500_V23.lib
Type	Function block without historical values.

This function block simulates a simple tank in a water pumping process. User can configure the tank size and attach the flow calculated from the PUMP_FLOW_CALC function block. Based on the user configuration, either emptying or filling mode of tank can be implemented. This function block outputs Water level or Pressure can be used as inputs to the level control or boost control application respectively.

The function block will be used to simulate tank behavior in level control and boost control applications.

The function block simulates the tank behavior only in general. The behavior in the rael process may be slightly different.

Input description

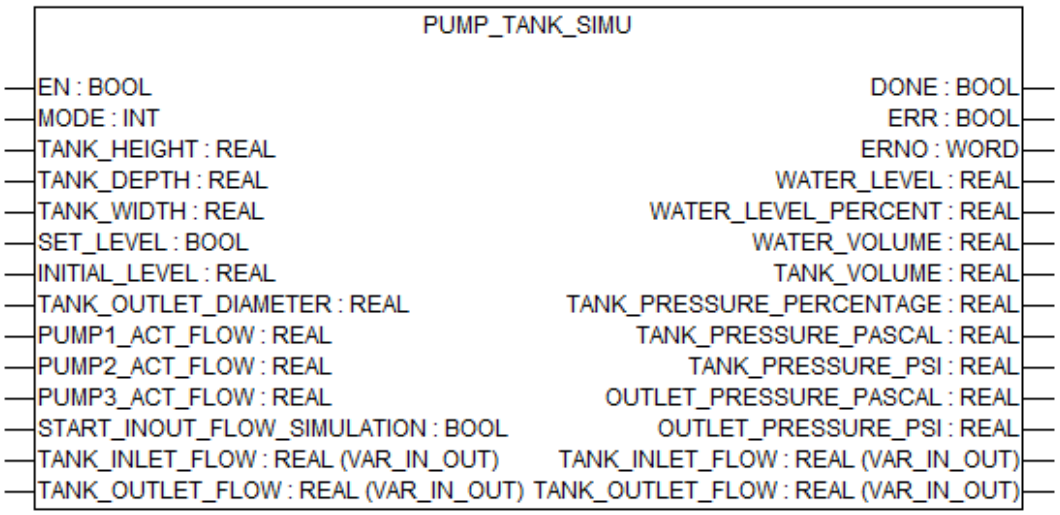


Fig. 687: Function block TANK_SIMU

EN (enable) Data type: BOOL, default value: FALSE

In order to enable the Function Block processing, input EN has to be continuously set to TRUE. The block is not processed if input EN = FALSE. While input is set to TRUE, the inputs are continuously checked for validity and plausibility. If this is not the case, processing is aborted and corresponding error is displayed at output ERR/ERNO.

MODE (mode) Data type: INT, default value: 1, range: 1 or 2.

The input parameter selects the tank operation mode

1 – Emptying of the tank

2 – Filling of the tank

TANK_HEIGHT (tank height) Data type: REAL, default value: 0, range: ≥ 0, unit: m

The input parameter specifies the tank height.

TANK_DEPTH (tank depth) Data type: REAL, default value: 0, range: ≥ 0, unit: m

The input parameter specifies the tank depth.

TANK_WIDTH (tank width) Data type: REAL, default value: 0, range: ≥ 0, unit: m

The input parameter specifies the tank width.

SET_LEVEL (set level) Data type: , BOOL, default value: FALSE

The user can trigger the input and set the tank initial value for simulation TRUE set the initial tank level as declared in the input INITIAL_LEVEL.

FALSE starts the simulation considering zero level in the tank.

INITIAL_LEVEL (initial level)

Data type: REAL, default value: 1, range: > 0, unit: m

The input parameter specifies the initial tank level to start the simulation. This input value will be considered for the tank simulation only if input SET_LEVEL is made TRUE.

TANK_OUTLET_DIAMETER (tank outlet diameter)

Data type: REAL, default value: 1, range: > 0, unit: m

The input parameter specifies the diameter of the outlet pipe.

PUMP1_ACT_FLOW (pump 1 actual flow)

Data type: REAL, default value: 0, range: >0, unit: m^3/h

Actual flow from the pump 1.

For simulation process, the calculated ACT_FLOW output from the PUMP_FLOW_CALC function block is used.

PUMP2_ACT_FLOW (pump 2 actual flow)

Data type: REAL, default value: 0, range: >0, unit: m^3/h

Actual flow from the pump 2.

For simulation process,

- the calculated ACT_FLOW output from the PUMP_FLOW_CALC function block is used if the pump is simulated by the PUMP_DRIVE_SIMU .
- the ACT_FLOW output from the PUMP_DOL_SIMU function block is used if the pump is simulated by the PUMP_DOL_SIMU.

PUMP3_ACT_FLOW (pump 3 actual flow)

Data type: REAL, default value: 0, range: >0, unit: m^3/h

Actual flow from the pump 3.

For simulation process,

- the calculated ACT_FLOW output from the PUMP_FLOW_CALC function block is used. if the pump is simulated by the PUMP_DRIVE_SIMU.
- the ACT_FLOW output from the PUMP_DOL_SIMU function block is used if the pump is controlled by the PUMP_DOL_SIMU.

START_INOUT_FLOW_SIMULATION (start input flow simulation)

Data type: BOOL, default value: FALSE

TRUE value starts the simulation of filling or emptying of a tank.

FALSE stopps the simulation of filling or emptying of a tank.

TANK_INLET_FLOW

Data type: REAL, default value: 0, range: >0, unit: m^3/h

Simulates the filling flow of the tank in emptying mode.

TANK_OUTLET_FLOW

Data type: REAL, default value: 0, range: >0, unit: m^3/h

Simulates the discharge flow of the tank in filling mode.

Output description

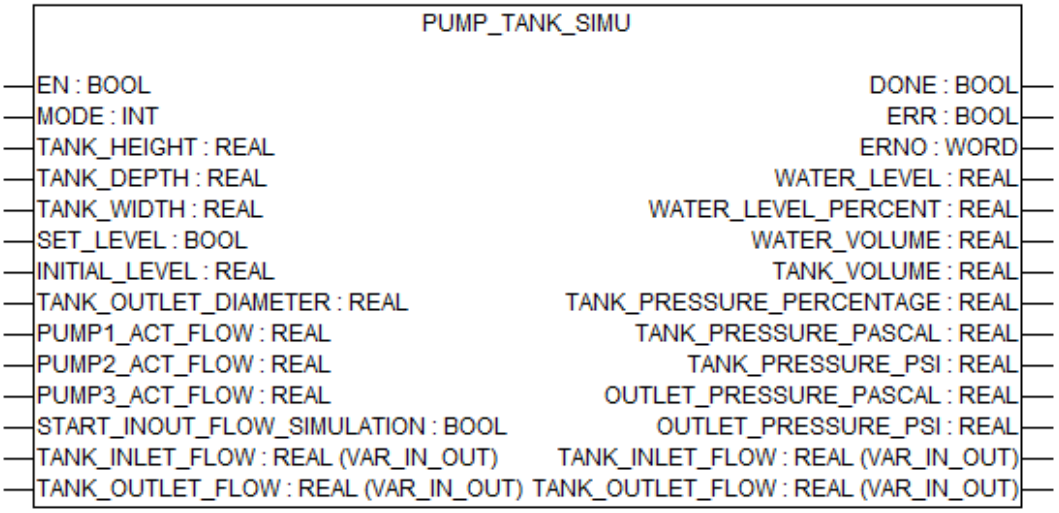


Fig. 688: Function block TANK_SIMU

DONE (done) Data type: BOOL

Output DONE indicates the processing state of the block. After completion or abortion of processing (due to an error), DONE is set to TRUE for one cycle. This output always has to be considered together with output ERR. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERR (error) Data type: BOOL Output ERR indicates whether an error occurred during data reception. If ERR is TRUE, an error occurred. In this case, the error number can be read at output ERNO.

ERNO (error number) Data type: WORD, default value: 0, range: ≥ 0

Output ERNO provides an error identifier if an invalid value was applied to an input. ERNO always has to be considered together with the output ERR. The value output at ERNO is only valid if ERR is TRUE. The error messages encoding at output ERNO are explained at the table below.

WATER_LEVEL (water level) Data type: REAL, unit: m

Level of the water tank.

WATER_LEVEL_PERCENT (water level percent) Data type: REAL, unit: %, range: 0-100

Level of the water tank in percentage.

WATER_VOLUME (water volume) Data type: REAL, unit: m^3

Actual volume of water in the tank .

TANK_VOLUME (tank volume) Data type: REAL, unit: m^3

Total volume of the tank.

TANK_PRESSURE_PERCENTAGE (tank pressure percentage)
Data type: REAL, unit: %, range: 0-100
Total pressure in the tank in percentage.

TANK_PRESSURE_PASCAL (tank pressure Pascal)
Data type: REAL, unit: Pa
Pressure at the bottom of tank.

TANK_PRESSURE_PSI (tank pressure psi)
Data type: REAL, unit: PSI
Pressure at the bottom of the tank.

OUTLET_PRESSURE_PASCAL (outlet pressure pascal)
Data type: REAL, unit: PSI
Pressure at the outlet pipe.

OUTLET_PRESSURE_PSI (outlet pressure psi)
Data type: REAL, unit: PSI
Pressure at the outlet pipe.

4000hex...4FFFhex - Block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

Dec	Hex	Error Description
0	0000	No error
16416	4020	MODE input value is less than 1 or greater than 2
16432	4030	TANK_HEIGHT is less than or equal to zero.
16448	4040	TANK_LEN is less than or equal to zero.
16464	4050	TANK_WIDTH is less than or equal to zero.
16496	4070	INITIAL_LEVEL is less than zero.
16512	4080	TANK_OUTLET_DIAMETER is less than or equal to zero.
16528	4090	PUMP1_ACT_FLOW is less than zero.
16544	40A0	PUMP2_ACT_FLOW is less than zero.
16560	40B0	PUMP3_ACT_FLOW is less than zero.

1.5.13.1.18 Visualizations

Every visualization element can be used to show the actual values of all inputs and outputs of the instance of the corresponding function block. In the application program, the programmer can add the visualization object in his project. In this object the programmer needs to add the faceplate of the required function block.

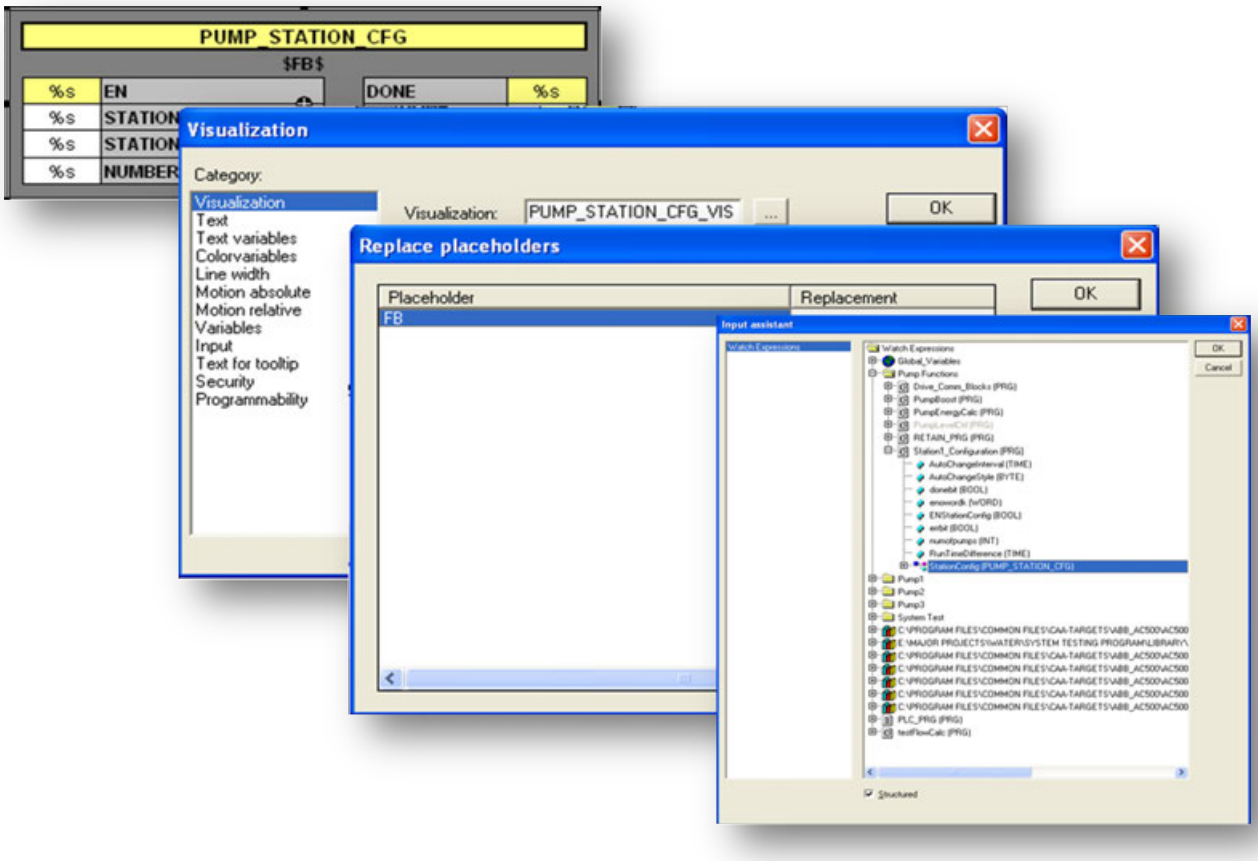


Fig. 689: Adding the faceplate of the required function block in the visualization object.

The visualization could also be used to control the function block by those inputs which are not connected inside the program.

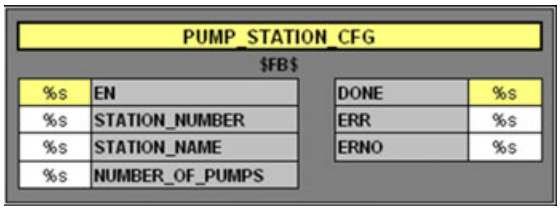
All inputs of the corresponding function block which are not connected to a variable (left open) can be written from this faceplate. So the function block can be controlled from the visualization as long as the inputs are left open.

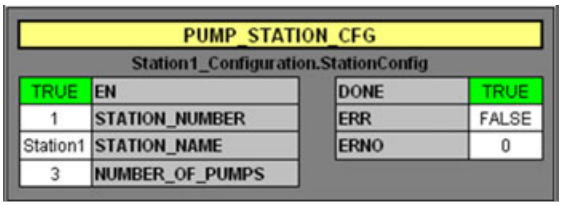
The color of the background can be changed by writing a value to the global variable `dwAcsVisuBackgroundColor`.

The color of the title can be changed by writing a value to the global variable `dwAcsVisuTitleColor`.

PUMP_STATION_CFG_VISU_PH

Faceplate for the function block PUMP_STATION_CFG.





Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 222: Colors of the variables

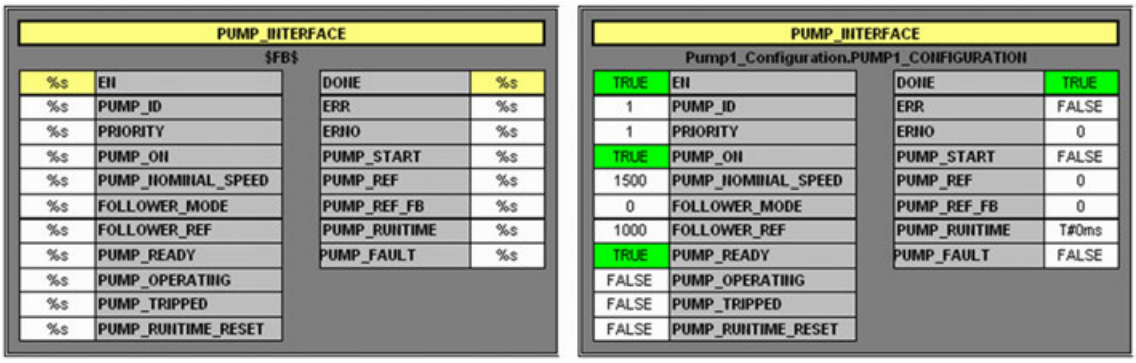
Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 223: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
STATION_NUMBER	R/W	Numpad – value 1 onwards	Station number as a unique identification for the station.
STATION_NAME	R/W	Text	Name of the pumping station.
NUMBER_OF_PUMPS	R/W	Numpad – value 1 to 3	Number of pumps in the pumping station.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.

PUMP_INTERFACE_VISU_PH

Faceplate for the function block PUMP_INTERFACE.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 224: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

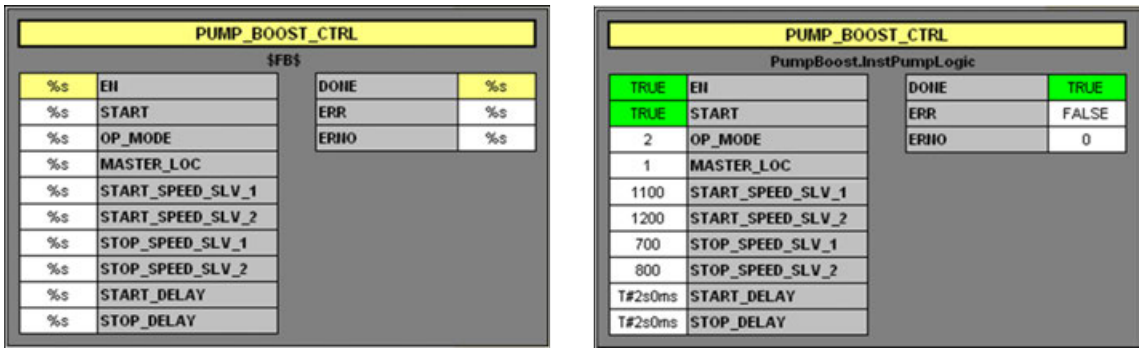
Table 225: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
PUMP_ID	R/W	Numpad – value 1 to 3	To enter the ID of the pump.
PRIORITY	R/W	Numpad – value 1 to 3	To enter the priority of the pump.
PUMP_ON	R/W	Toggle	To switch off/On the pump individually. For the FALSE value the user can remove this pump from the network in case of any problem/fault.
PUMP_NOMINAL_SPEED	R/W	Numpad – value 0 onwards	To enter the nominal speed of the pump motor.
FOLLOWER_MODE	R/W	Numpad – value 0 to 2	Follower mode to decide at what speed the follower pumps must move. 0= Fixed, 1=copy master, 2=master speed.
FOLLOWER_REF	R/W	Numpad – value 0 onwards	follower speed reference in RPM, when FOLLOWER_MODE =0, i.e. fixed.
PUMP_READY	R/W	Toggle	Input to attach the ready status of the drive of the pump.
PUMP_OPERATING	R/W	Toggle	Input to attach the operating status of the drive of the pump.
PUMP_TRIPPRED	R/W	Toggle	Input to attach the fault status of the drive of the pump.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
PUMP_RUNTIME_RESET	R/W	Toggle	TRUE value of this variable resets the actual run time to Zero value.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.
PUMP_START	R		On command to the pump.
PUMP_REF	R		Speed reference to the pump in RPM.
PUMP_REF_FB	R		Field bus speed reference to the pump in the range of -20000 to 20000.
PUMP_RUNTIME	R		Actual run time of the pump.

PUMP_BOOST_CTRL_VISU_PH

Faceplate for the function block PUMP_BOOST_CTRL.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 226: Colors of the variables

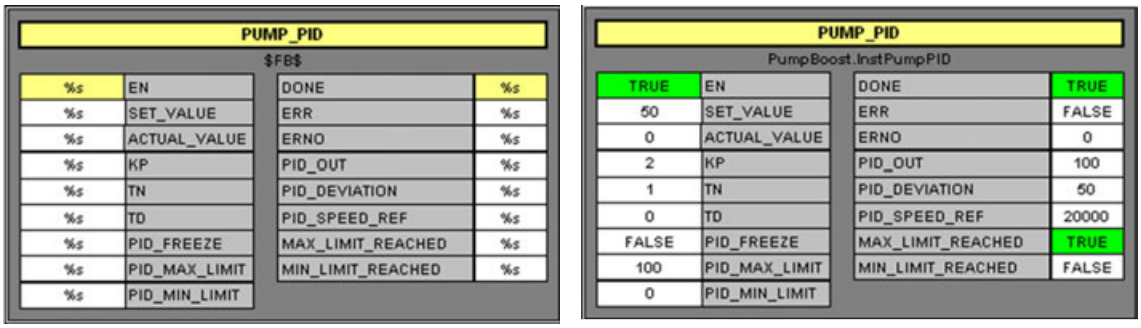
Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 227: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
START	R/W	Toggle	To start the Boost control functionality.
OP_MODE	R/W	Numpad – value 0 to 2	Operation mode of the station, 0= single pump, 1=traditional pump, 2=multi pump.
MASTER_LOC	R/W	Numpad – value 0 to 1	Defines the master location. INSTART: youngest pump in the network is the master. 1- STABLE: master is always fixed.
START_SPEED_SLV_1	R/W	Numpad- value 0 onwards	Speed of master in RPM at which the follower pump 1 starts.
START_SPEED_SLV_2	R/W	Numpad- value 0 onwards	Speed of master in RPM at which the follower pump 2 starts.
STOP_SPEED_SLV_1	R/W	Numpad- value 0 onwards	Speed of master in RPM at which the follower pump 1 stops.
STOP_SPEED_SLV_2	R/W	Numpad- value 0 onwards	Speed of master in RPM at which the follower pump 2 stops.
START_DELAY	R/W	Numpad- value 0 onwards	Time delay in seconds for starting the pump.
STOP_DELAY	R/W	Numpad- value 0 onwards	Time delay in seconds for stopping the pump.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.

PUMP_PID_VISU_PH

Faceplate for the function block PUMP_PID.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 228: Colors of the variables

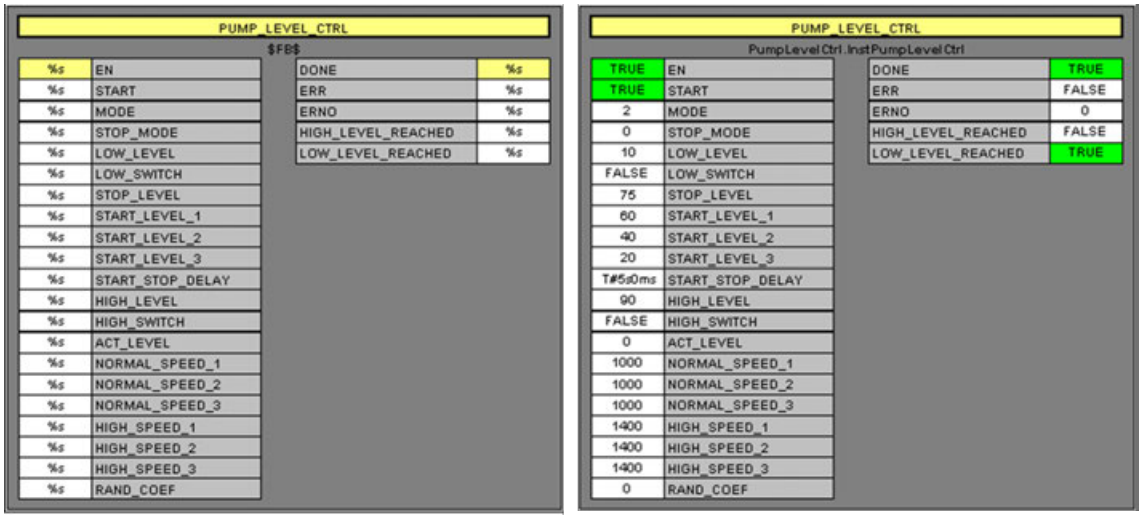
Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 229: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
SET_VALUE	R/W	Numpad – value 0.0 to 100.0	Set value of the process variable in terms of percentage.
ACTUAL_VALUE	R/W	Numpad – value 0.0 to 100.0	Actual value of the process variable in terms of percentage.
KP	R/W	Numpad – value 0.0 to 100.0	Proportional gain of the PID.
TN	R/W	Numpad – value 0.0 to 100.0	Integral time of the PID in terms of seconds.
TD	R/W	Numpad – value 0.0 to 100.0	Derivative time of the PID in terms of seconds.
PID_FREEZE	R/W	Toggle	For the TRUE value the setpoint to the PID block is frozen. Actual value is then attached to the setpoint.
PID_MAX_LIMIT	R/W	Numpad – value 0.0 to 100.0	Limit for the maximum PID Output.
PID_MIN_LIMIT	R/W	Numpad – value 0.0 to 100.0	Limit for the minimum PID Output.
PUMP_RUN-TIME_RESET	R/W	Toggle	TRUE value of this variable resets the actual run time to Zero value.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.
PID_OUT	R		PID output after the manipulation in terms of percentage.
PID_DEVIATION	R		Deviation of Set value from actual value. It is also known as PID error.
PID_SPEED_REF	R		Speed reference as a correction to the main speed reference, calculated by the PID - in the range of 0 to 20000.
MAX_LIMIT_REACHED	R		Indicates if the PID output touches the MAXIMUM limit.
MIN_LIMIT_REACHED	R		Indicates if the PID output touches the MINIMUM limit.

PUMP_LEVEL_CTRL_VISU_PH

Faceplate for the function block PUMP_LEVEL_CTRL.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 230: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

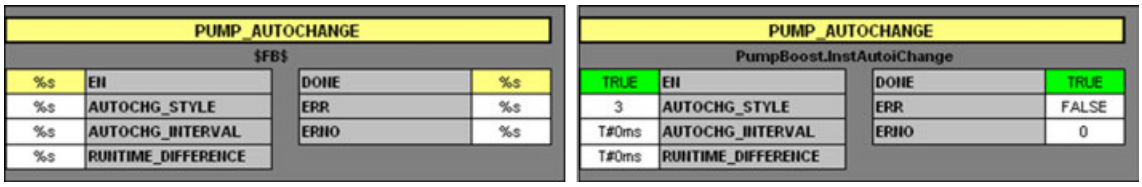
Table 231: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
START	R/W	Toggle	To start the level control functionality.
MODE	R/W	Numpad – value 1 to 2	To select the mode of operation. 1 = emptying, 2= filling.
STOP_MODE	R/W	Numpad – value 0 to 1	Defines the stop mode stage stop. stops the pump at individual defined start stop levels. 1, common stop. stops all the pump as common stop at defined STOP_LEVEL.
LOW_LEVEL	R/W	Numpad- value 0.0 to 100.0	Defines the low level in terms of % of full tank capacity.
LOW_SWITCH	R/W	Toggle	Digital input with TRUE indicates that the low level has reached.
STOP_LEVEL	R/W	Numpad- value 0.0 to 100.0	Defines the stop level in terms of % of full tank capacity.
START_LEVEL_1	R/W	Numpad- value 0.0 to 100.0	Defines the start level for pump 1 in terms of % of full tank capacity.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
START_LEVEL_2	R/W	Numpad- value 0.0 to 100.0	Defines the start level for pump 2 in terms of % of full tank capacity.
START_LEVEL_3	R/W	Numpad- value 0.0 to 100.0	Defines the start level for pump 3 in terms of % of full tank capacity.
START_STOP_DELAY	R/W	Numpad- value 0 onwards	Defines the time delay for pump to start and stop when respective level is reached.
HIGH_LEVEL	R/W	Numpad- value 0.0 to 100.0	Defines the high level in terms of % of full tank capacity.
HIGH_SWITCH	R/W	Toggle	Digital input with TRUE indicates that the high level has reached.
ACT_LEVEL	R/W	Numpad- value 0.0 to 100.0	Actual level in % - read from the analog input of level sensor.
NORMAL_SPEED_1	R/W	Numpad- value 0 onwards	Normal operating speed of the pump1 while Filling/Emptying.
NORMAL_SPEED_2	R/W	Numpad- value 0 onwards	Normal operating speed of the pump2 while Filling/Emptying.
NORMAL_SPEED_3	R/W	Numpad- value 0 onwards	Normal operating speed of the pump3 while Filling/Emptying.
HIGH_SPEED_1	R/W	Numpad- value 0 onwards	Defines the speed of the pump1 for : - Filling- pump level falls below LOW_LEVEL/ Emptying - pump level rises above the HIGH_LEVEL.
HIGH_SPEED_2	R/W	Numpad- value 0 onwards	Defines the speed of the pump2 for : - Filling- pump level falls below LOW_LEVEL/ Emptying – pump3 level rises above the HIGH_LEVEL.
HIGH_SPEED_3	R/W	Numpad- value 0 onwards	Defines the speed of the pump for : - Filling- pump level falls below LOW_LEVEL/ Emptying - pump level rises above the HIGH_LEVEL.
RAND_COEF	R/W	Numpad – value -10.0 to + 10.0	Random coefficient randomizes the start levels to avoid cake formation. Value between -10.0 to 10.0. Value 1.0 means 1.0%.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.
HIGH_LEVEL_REACHED	R		TRUE value indicates that the actual level is more than the high level.
LOW_LEVEL_REACHED	R		TRUE value indicates that the actual level is below the low level.

PUMP_ AUTOCHANGE _VISU_PH

Faceplate for the function block PUMP_ AUTOCHANGE.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 232: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 233: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
AUTOCHG_STYLE	R/W	Numpad- value 0 to 3	Mode of auto change. 0=None, 1=fixed, 2=runtime difference, 3=all stop.
AUTOCHG_INTERVAL	R/W	Numpad- value 0 onwards	Time delay after which the autochange starts when AUTOCHG_STYLE =1.
RUNTIME_DIFFERENCE	R/W	Numpad- value 0 onwards	Maximum permited run-time difference between the two pumps after which the autochange starts when AUTOCHG_STYLE =2.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.

PUMP_ ANTIJAM _VISU_PH

Faceplate for the function block PUMP_ ANTIJAM.

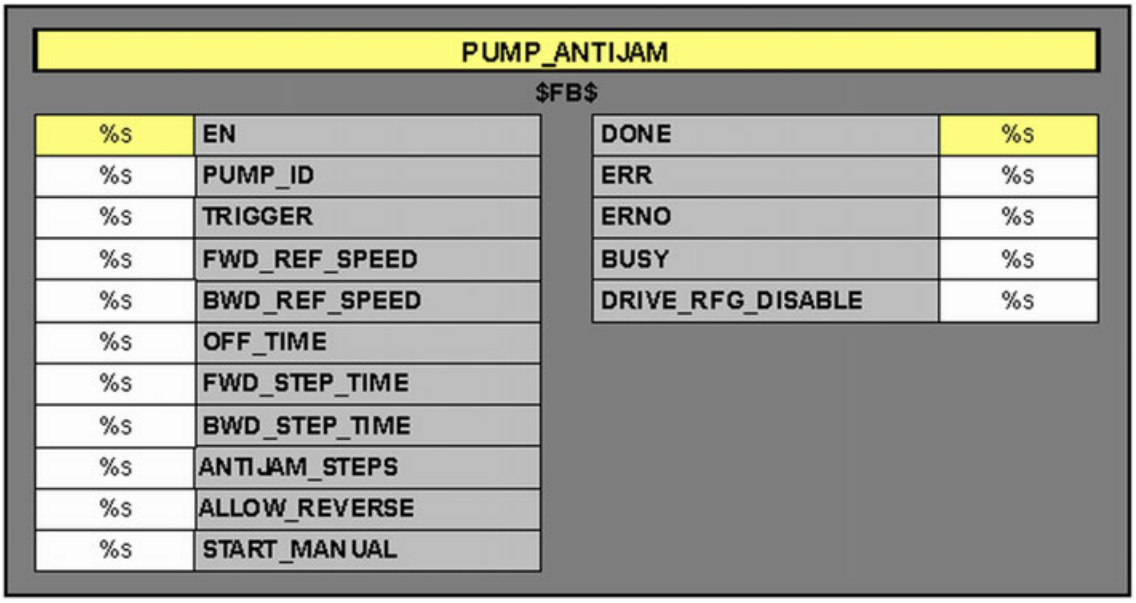


Fig. 690: Visualization in offline mode.

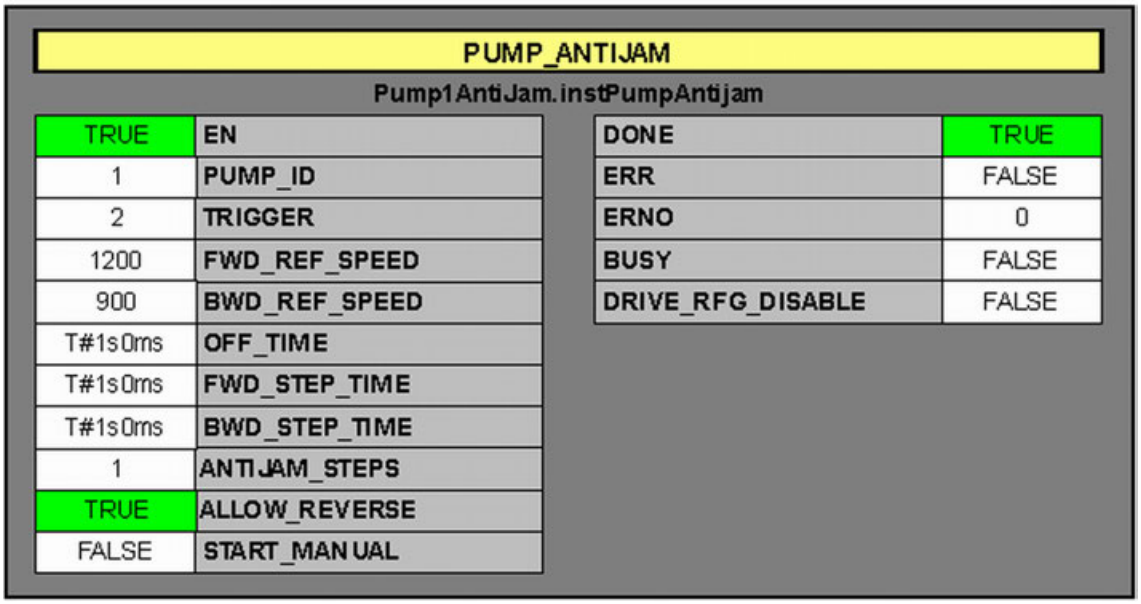


Fig. 691: Visualization in online mode.

Table 234: Colors of the variables

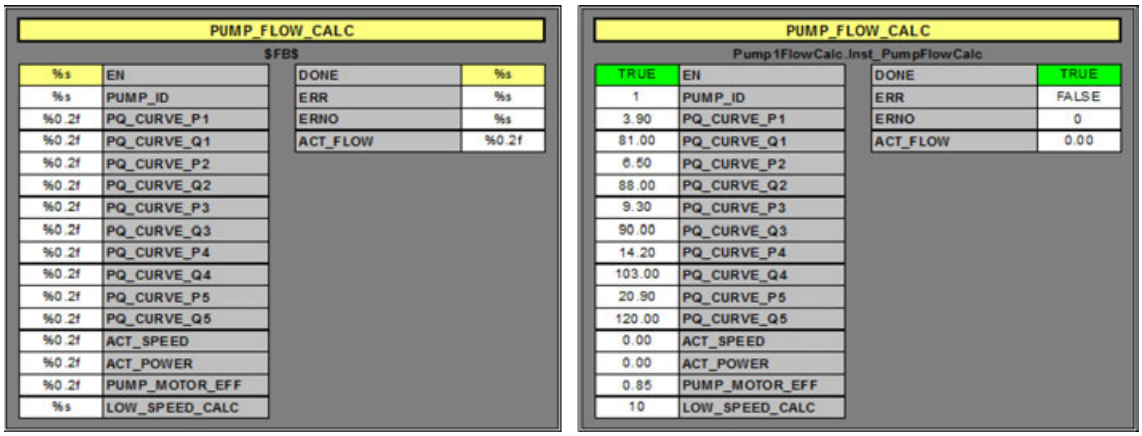
Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 235: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
PUMP_ID	R/W	Numpad- value 1 to 3	To enter the ID of the pump which is to be cleaned
TRIGGER	R/W	Numpad- value 0 to 4	Trigger to clean the pump. 0=not enabled, 1=master enabled, 2=follower enabled, 3=at start, 4=manual mode
FWD_REF_SPEED	R/W	Numpad- value 0 onwards	Speed reference to clean the pump in the forward direction
BWD_REF_SPEED	R/W	Numpad- value 0 onwards	Speed reference to clean the pump in the backward direction
OFF_TIME	R/W	Numpad- value 0 onwards	Time delay in seconds between the forward and the backward movement in anti-jam
FWD_STEP_TIME	R/W	Numpad- value 0 onwards	Time duration in seconds for which the pump moves in the forward direction in anti-jam
BWD_STEP_TIME	R/W	Numpad- value 0 onwards	Time duration in seconds for which the pump moves in the reverse direction in anti-jam
ANTIJAM_STEPS	R/W	Numpad- value 1 onwards	Number of steps to be performed in an anti-jam process
ALLOW_REVERSE	R/W	Toggle	True value means the pump can also move in the reverse direction to perform the anti jam
START_MANUAL	R/W	Toggle	For the TRUE anti-jam is performed in the manual mode i.e. TRIGGER =4
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.

PUMP_ FLOW_CALC_VISU_PH

Faceplate for the function block PUMP_ FLOW_CALC.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 236: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

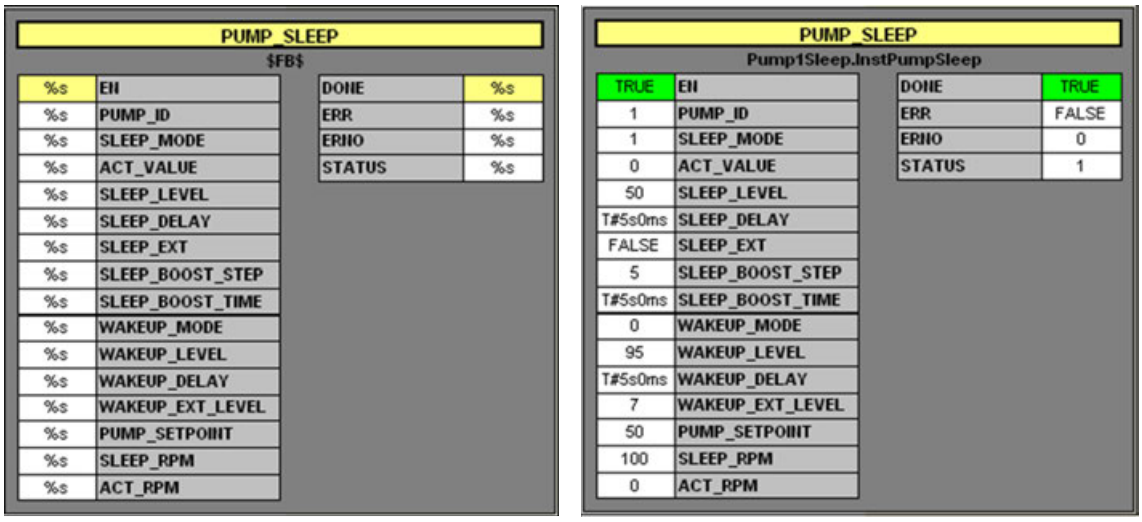
Table 237: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
PUMP_ID	R/W	Numpad- value 1 to 3	To enter the ID of the pump which is to be cleaned.
PQ_CURVE_P1	R/W	Numpad- value 0 onwards	Pump PQ curve P1.
PQ_CURVE_Q1	R/W	Numpad- value 0 onwards	Pump PQ curve Q1.
PQ_CURVE_P2	R/W	Numpad- value 0 onwards	Pump PQ curve P2.
PQ_CURVE_Q2	R/W	Numpad- value 0 onwards	Pump PQ curve Q2.
PQ_CURVE_P3	R/W	Numpad- value 0 onwards	Pump PQ curve P3.
PQ_CURVE_Q3	R/W	Numpad- value 0 onwards	Pump PQ curve Q3.
PQ_CURVE_P4	R/W	Numpad- value 0 onwards	Pump PQ curve P4.
PQ_CURVE_Q4	R/W	Numpad- value 0 onwards	Pump PQ curve Q4.
PQ_CURVE_P5	R/W	Numpad- value 0 onwards	Pump PQ curve P5.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
PQ_CURVE_Q5	R/W	Numpad- value 0 onwards	Pump PQ curve Q5.
NOMINAL_POWER		Numpad- value 0 onwards	Nominal power of the motor in kW.
ACT_SPEED	R		Actual speed of the motor in RPM is connected to the input.
ACT_POWER	R		Actual power of the motor in kW.
PUMP_MOTOR_EFF			Combined efficiency of the motor and the pump set.
LOW_SPEED_CALC			Speed below which the calculation will not take place.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.
ACT_FLOW	R		Calculated flow of the pump.

PUMP_ SLEEP _VISU_PH

Faceplate for the function block PUMP_ SLEEP.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 238: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

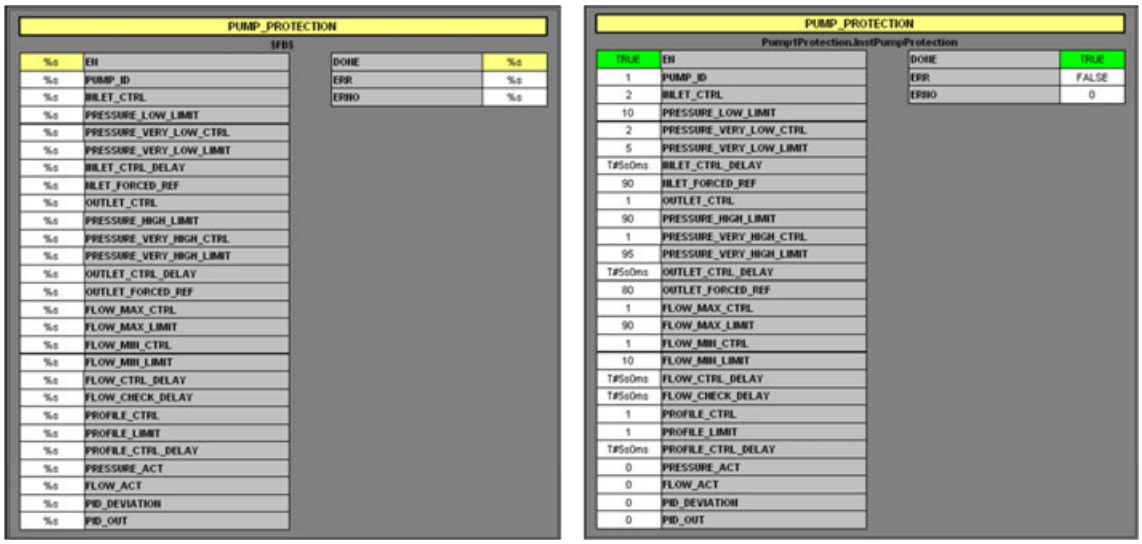
Table 239: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
PUMP_ID	R/W	Numpad- value 1 to 3	To enter the ID of the pump which is to be cleaned.
SLEEP_MODE	R/W	Numpad- value 0 to 4	SLEEP MODE: 0= Not Used, 1= Internal, 2 = External, 3 = Int+Ext, 4 = Soft Ext.
ACT_VALUE	R/W	Numpad- value 0 to 100.0	Actual process value which is compared with the sleep level.
SLEEP_LEVEL	R/W	Numpad- value 0 to 100.0	Defines the level to active the sleep function.
SLEEP_DELAY	R/W	Numpad- value 0 onwards	Time delay for the sleep function to get activated.
SLEEP_EXT	R/W	Toggle	To enable the sleep function externally.
SLEEP_BOOST_STEP	R/W	Numpad- value 0 to 100.0	The setpoint is increased by boost step percentage for the time defined by sleep boost time.
SLEEP_BOOST_TIME	R/W	Numpad- value 0 onwards	Time in seconds for which sleep boost is to be operational.
WAKEUP_MODE	R/W	Numpad- value 0 to 3	Wake up mode: 0= wake > ref, 1= wake < ref , 2 = wake > Ext, 3 = wake < Ext.
WAKEUP_LEVEL	R/W	Numpad- value 0 to 100.0	Defines the level to active the wake up function. It is a percentage of setpoint.
WAKEUP_DELAY	R/W	Numpad- value 0 onwards	Time delay for the wake up function to activate.
WAKEUP_EXT_LEVEL	R/W	Numpad- value 0 to 100.0	Wake up external level. For wake up mode 2 and 3, this value is compaired with actual value.
PUMP_SETPOINT	R/W	Numpad- value 0 to 100.0	Process setpoint - the same setpoint connected in the PUMP_PID function block.
SLEEP_RPM	R/W	Numpad- value 0 onwards	Pump RPM below which pump should go in to the sleep mode.
ACT_RPM	R/W	Numpad- value 0 onwards	Actual RPM of the pump.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
ENO	R		Error code.
STATUS	R		Status of sleep function. 0 = Function is inactive 1 = Sleep Function is active 2 = Boost Function is active 3 = Wake up Function is active

PUMP_PROTECTION_VISU_PH

Faceplate for the function block PUMP_PROTECTION.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 240: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

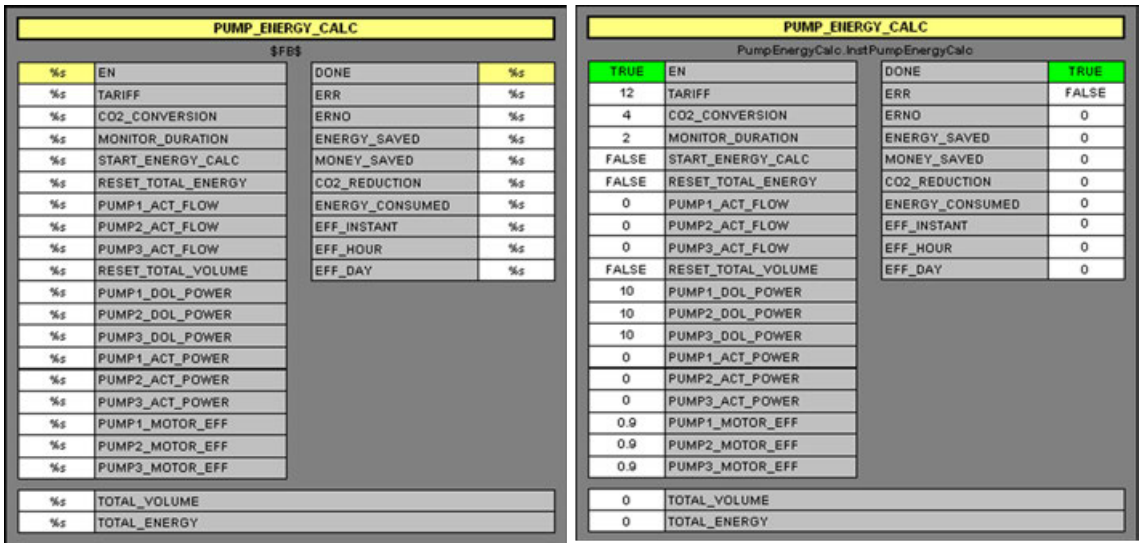
Table 241: Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
PUMP_ID	R/W	Numpad- value 1 to 3	ID of the pump for which the function block is to be called.
INLET_CTRL	R/W	Numpad – value 0 to 3	To enable control for primary protection against low inlet pressure: 0 = disabled, 1 = alarm, 2 = Fault, 3 = reduced speed.
PRESSURE_LOW_LIMIT	R/W	Numpad – value 0.0 to 100.0	Low pressure level limit for primary protection.
PRESSURE_VERY_LOW_CTRL	R/W	Numpad – value 0 to 2	To enable control for secondary protection against very low inlet pressure: 0 = disabled, 1 = fault, 2 = stop.
PRESSURE_VERY_LOW_LIMIT	R/W	Numpad – value 0.0 to 100.0	Very low pressure level limit for secondary protection.
INLET_CTRL_DELAY	R/W	Numpad – value 0 onwards	Delay in seconds for primary and secondary supervision to start. 0-600 s.
INLET_FORCED_REF	R/W	Numpad – value 0 onwards	Safe speed reference for the drive when the INLET_CTRL = 3 i.e. reduced speed mode.
OUTLET_CTRL	R/W	Numpad – value 0 to 3	To enable control for primary protection against high outlet pressure: 0 = disabled, 1 = alarm, 2 = fault, 3 = reduced speed.
PRESSURE_HIGH_LIMIT	R/W	Numpad – value 0.0 to 100.0	High pressure level limit for primary protection.
PRESSURE_VERY_HIGH_CTRL	R/W	Numpad – value 0 to 2	To enable control for secondary protection against very high outlet pressure: 0 = disabled, 1 = fault, 2 = stop.
PRESSURE_VERY_HIGH_LIMIT	R/W	Numpad – value 0.0 to 100.0	Very high pressure level limit at for secondary protection.
OUTLET_CTRL_DELAY	R/W	Numpad – value 0 onwards	Delay in seconds for primary and secondary supervision to start.
OUTLET_FORCED_REF	R/W	Numpad – value 0 onwards	Safe speed reference for the drive when the OUTLET_CTRL = 3 i.e. reduced speed mode.
FLOW_MAX_CTRL	R/W	Numpad – value 0 to 2	To enable control for protection against maximum flow condition: 0 – Disabled, 1 – Alarms, 2 – fault.
FLOW_MAX_LIMIT	R/W	Numpad – value 0 to 32767	Value of max flow rate limit in m^3/h.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
FLOW_MIN_CTRL	R/W	Numpad – value 0 to 2	To enable control for protection against minimum flow condition: 0 – Disabled, 1 – Alarms, 2 – fault.
FLOW_MIN_LIMIT	R/W	Numpad – value 0 to 32767	Value of min flow rate limit in m^3/h.
FLOW_CTRL_DELAY	R/W	Numpad – value 0 onwards	Delay in seconds before executing the control due to low or high flow rates.
FLOW_CHECK_DELAY	R/W	Numpad – value 0 onwards	Delay in seconds after the start of the pump before monitoring the flow rates.
PROFILE_CTRL	R/W	Numpad – value 0 to 2	0 – profile protection disabled. 1 – PID deviation, generates alarm , PROFILE HIGH, if PID deviation exceeds PROFILE_LIMIT. 2 – PID output, generates alarm , PROFILE HIGH, if PID output exceeds PROFILE_LIMIT.
PROFILE_LIMIT	R/W	Numpad – value 0 to 32767	Value of profile protection limit.
PROFILE_CTRL_DELAY	R/W	Numpad – value 0 onwards	Time delay in seconds before the Profile Ctrl activates.
PRESSURE_ACT	R/W	Numpad – value 0.0 to 100.0	Actual pressure of the system.
FLOW_ACT	R/W	Numpad – value 0.0 to 100.0	Actual flow of the pump.
PID_DEVIATION	R/W	Numpad – value 0.0 to 100.0	PID deviation generated by PUMP_PID function block.
PID_OUT	R/W	Numpad – value 0.0 to 100.0	PID ouput generated by PUMP_PID function block.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.

PUMP_ ENERGY_CALC_VISU_PH

Faceplate for the function block PUMP_ ENERGY_CALC.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 242: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Table 243: Visualization parameters

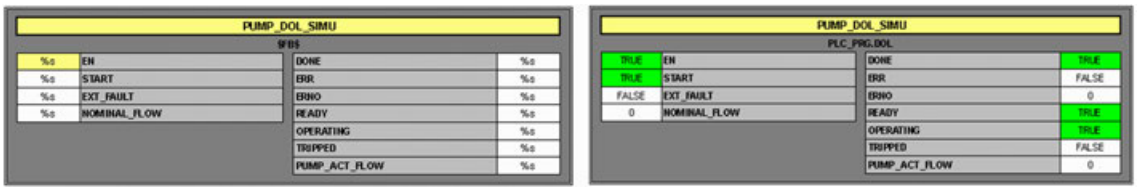
Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
TARIFF	R/W	Numpad- value 0 onwards	Per kwhr tariff rate.
CO2_CONVERSION	R/W	Numpad- value 0 onwards	Factor which helps in converting the energy consumed in terms of CO2 emission tn/Mwh.
MONITOR_DURATION	R/W	Numpad- value 0 onwards	Time duration in minutes to monitor the energy consumption.
START_ENERGY_CALC	R/W	Toggle	To enable energy calculation.
RESET_TOTAL_ENERGY	R/W	Toggle	Resets total energy value.
PUMP1_ACT_FLOW	R	Numpad- value 0 onwards	Calculated Actual flow of PUMP1 in m^3/hr.
PUMP2_ACT_FLOW	R	Numpad- value 0 onwards	Calculated Actual flow of PUMP2 in m^3/hr.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
PUMP3_ACT_FLOW	R	Numpad- value 0 onwards	Calculated Actual flow of PUMP3 in m^3/hr.
RESET_TOTAL_VOLUME	R/W	Toggle	Reset Total volume.
PUMP1_DOL_POWER	R/W	Numpad- value 0 onwards	Pump1 Direct On Line power, if directly connected to the motor in kW.
PUMP2_DOL_POWER	R/W	Numpad- value 0 onwards	Pump2 Direct On Line power, if directly connected to the motor in kW.
PUMP3_DOL_POWER	R/W	Numpad- value 0 onwards	Pump3 Direct On Line power, if directly connected to the motor in kW.
PUMP1_ACT_POWER	R/W	Numpad- value 0 onwards	Pump1 Actual power in kW.
PUMP2_ACT_POWER	R/W	Numpad- value 0 onwards	Pump2 Actual power in kW.
PUMP3_ACT_POWER	R	Numpad- value 0 onwards	Pump3 Actual power in kW.
PUMP1_MOTOR_EFF	R/W	Numpad- value 0 to 1.0	Pump1 motor combined efficiency.
PUMP1_MOTOR_EFF	R/W	Numpad- value 0 to 1.0	Pump2 motor combined efficiency.
PUMP1_MOTOR_EFF	R/W	Numpad- value 0 to 1.0	Pump3 motor combined efficiency.
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code.
ENERGY_SAVED	R		Stores the energy saved. Difference of the energy consumed by pump with drive and the energy consumed if the pump was connected to the DOL motor.
MONEY_SAVED	R		Saving in terms of money.
CO2_REDUCTION	R		Reduction in terms of carbon dioxide emissions.
ENERGY_CONSUMED	R		Stores the energy consumed in the monitoring duration.
EFF_INSTANT	R		Instant flow vs power efficiency of the station.
EFF_HOUR	R		Hourly flow vs power efficiency of the station.
EFF_DAY	R		Daily flow vs power efficiency of the station.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
TOTAL_VOLUME	R		Total volume which is calculated using flow of all the pumps.
TOTAL_ENERGY			Total actual energy which is calculated using actual power of all the pumps.

PUMP_DOL_SIMU _VISU_PH

Faceplate for the function block PUMP_DOL_SIMU.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 244: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

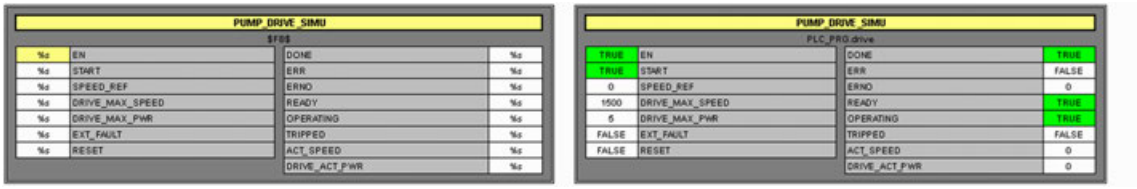
Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
START	R/W	Toggle	Start the simulation DOL pump is operating.
EXT_FAULT	R/W	Toggle	To force the fault externally to check the fault condition.
NOMINAL_FLOW	R/W	Numpad- value 0 onwards	Nominal flow of the pump in m^3/h.

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
DONE	R		Execution finished when output DONE = TRUE.
ERR	R		Error occurred during execution when output ERR = TRUE.
ENO	R		Error code
READY OPERATING TRIPPED PUMP_ACT_FLOW	R R R R		Pump is in ready state for operation Pump is Operating Pump is tripped Actual flow of the DOL pump in m^3/h

PUMP_DRIVE_SIMU _VISU_PH

Faceplate for the function block PUMP_DRIVE_SIMU.



Left figure Visualization in offline mode.
Right figure Visualization in online mode.

Table 245: Colors of the variables

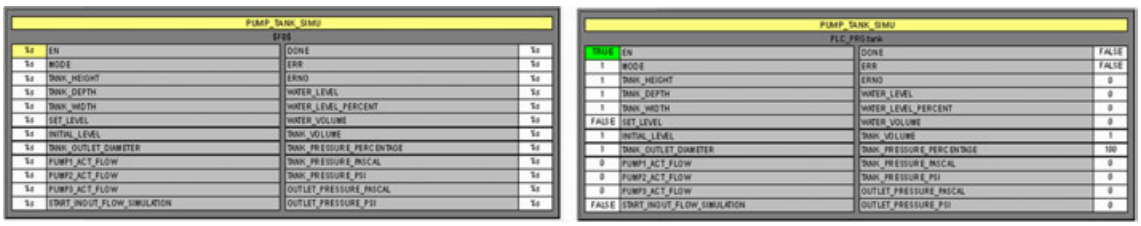
Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value.
START	R/W	Toggle	Start the simulation drive is operating.
SPEED_REF	R/W	Numpad – value 0 onwards	Speed reference from the PUMP_INTERFACE FB in RPM
DRIVE_MAX_SPEED	R/W	Numpad – value 0 onwards	Maximum speed of the drive in RPM
DRIVE_MAX_PWR	R/W	Numpad – value 0 onwards	Maximum power of the drive in kW
EXT_FAULT	R/W	Toggle	To force the fault externally to check the fault condition.
RESET	R/W	Toggle	Reset drive fault and reset ERNO output
DONE	R	-	Execution finished when output DONE = TRUE.
ERR	R	-	Error occurred during execution when output ERR = TRUE.
ENO	R	-	Error code.
READY OPERATING TRIPPED ACT_SPEED DRIVE_ACT_PWR	R R R R R	- - - - -	Pump is in ready state for operation Pump is Operating Pump is tripped Actual speed in RPM Actual power of drive in kW

PUMP_TANK_SIMU _VISU_PH

Faceplate for the function block PUMP_TANK_SIMU.



Right figure Visualization in online mode.

Table 246: Colors of the variables

Color	Description
white	Actual FALSE and should be FALSE in normal operation
green	Actual TRUE and should be TRUE in normal operation
yellow	Actual FALSE but should be TRUE in normal operation
red	Actual TRUE but should be FALSE in normal operation

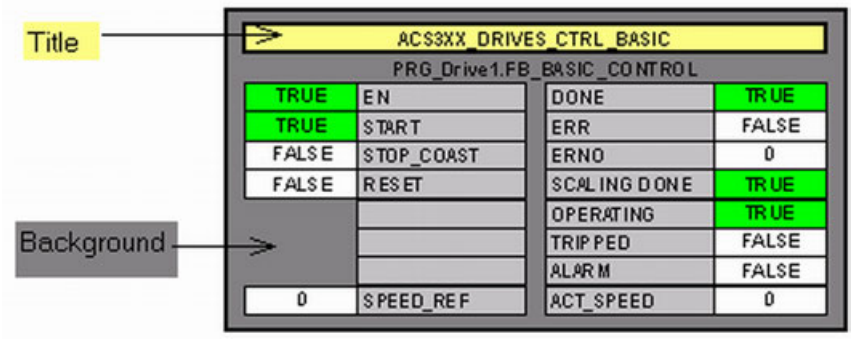
Visualization parameters

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
EN	R/W	Toggle	To enable the function block with the true value
MODE	R/W	Numpad – 1 or 2	To select the mode of operation. 1 = emptying, 2= filling
TANK_HEIGHT	R/W	Numpad – value 0 onwards	Height of Tank in m
TANK_DEPTH	R/W	Numpad – value 0 onwards	Depth of tank in m
TANK_WIDTH	R/W	Numpad – value 0 onwards	Width of tank in m
SET_LEVEL	R/W	Toggle	TRUE: Sets INITIAL_LEVEL as initial value of tank volume in m^3
INITIAL_LEVEL	R/W	Numpad – value 0 onwards	Initial value of tank level in m^3
TANK_OUTLET_DIAMETER	R/W	Numpad – 0 value onwards	Outlet diameter of pipe in m
PUMP1_ACT_FLOW	R/W	Numpad – 0 value onwards	Calculated Actual flow of PUMP1 in m^3/h
PUMP2_ACT_FLOW	R/W	Numpad – 0 value onwards	Calculated Actual flow of PUMP2 in m^3/h
PUMP3_ACT_FLOW	R/W	Numpad – 0 value onwards	Calculated Actual flow of PUMP3 in m^3/h
START_INOUT_FLOW_SIMUALTION	R/W	Toggle	Start filling or emptying flow flow simulation
DONE	R	-	Execution finished when output DONE = TRUE.
ERR	R	-	Error occurred during execution when output ERR = TRUE.
ENO	R	-	Error code.
WATER_LEVEL	R	-	Level of the tank in m

Variable element	Access	Access via	Description (all elements refer to the function block instance, replaced for the placeholder FB)
WATER_LEVEL_PERCENT	R	-	Level of the tank in percentage
WATER_VOLUME	R	-	Actual water volume in m^3
TANK_VOLUME	R	-	Total tank volume in m^3
TANK_PRESSURE_PERCENTAGE	R	-	Pressure in percentage at the bottom of tank
TANK_PRESSURE_PASCAL	R	-	Pressure at the tank bottom in PASCAL
TANK_PRESSURE_PSI	R	-	Pressure at the tank bottom in PSI
OUTLET_PRESSURE_PASCAL	R	-	Pressure at the outlet pipe in PASCAL
OUTLET_PRESSURE_PS	R	-	Pressure at the outlet pipe in PSI

1.5.13.1.19 Global variables

The background color and the color of the title in the visualization elements of the library can be changed with the two global variables dwAcsVisuBackgroundColor and dwAcsVisuTitleColor.



1.5.13.1.20 Glossary

BOOL Variables of the type BOOL can have the values TRUE and FALSE. For this, 8 bit of memory space are reserved.

Integer data types

BYTE, DINT, DWORD, INT, WORD

The different numerical types are responsible for a different numerical range. Due to this, it is possible that information are lost when converting greater data types to smaller data types.

Table 247: For integer data types the following range limits are valid:

Type	BYTE	INT	DINT	WORD	DWORD
Lower limit	0	-32768	-2147483648	0	0
Upper limit	255	32767	2147483647	65535	4294967295
Memory space	8 bits	16 bits	32 bits	16 bits	32 bits

Functions

Functions are subroutines which have multiple input parameters and return exactly one result element. The returned result can be of an elementary or a derived data type. Due to this, a function may also return an array, a structure, an array of structures and so on. For the same input parameters, functions always return the same result (they do not have an internal memory).

Therefore, the following rules can be derived:

- Within functions, global variables can neither be read nor written.
- Within functions, absolute operands can neither be read nor written.
- Within functions, function 'Function Blocks' must not be called.

Function blocks

Function blocks are subroutines which can have as many inputs, outputs and internal variables as required. They are called from a program or from another function block. As they can be used several times (with different data records), function blocks (code and interface) can be considered as type. When assigning an individual data record (declaration) to the function block, a function block instance is generated. In contrast to functions, function blocks can contain statically local data which are saved from one call to the next. Therefore e.g. counters can be realized which may not forget their counter value. I.e. function blockscan have an internal memory.

Functions and function blocks differ in two essential points:

- A function block has multiple output parameters, a function only one. The output parameters of functions and function blocks differ syntactically.
- In contrast to a function, a function block can have an internal memory.

Function blocks with historical values (memory)

For function blocks with historical values it has to be observed that instance names may not be defined several times if different data sets should be called.

Function blocks without historical values (memory)

For function blocks without historical values only one instance has to be defined for the function block type. This instance can be used for several calls of the function block (also with different I/O values).

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

1.5.13.1.21 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.13.2 Datalogging library

1.5.13.2.1 Overview

The Datalogging function block Library contains five function blocks for the purpose of advanced time-stamped data logging for different use cases. In typical use cases the AC500 application program generates data which are normally transmitted to a telecontrol system for storage and further processing or displaying to the end user. Typically, these may be remote applications like water pumping stations or solar power plants where the connection between the remote station (AC500) and a central SCADA/telecontrol station is not always stable or only sporadically connected. Sporadically connected can be by intention e.g. to save communication costs or open ports/connections to be handled by a control station. Then the Datalogging function blocks can store data in case of a broken or intentionally interrupted connection between AC500 and the telecontrol system.

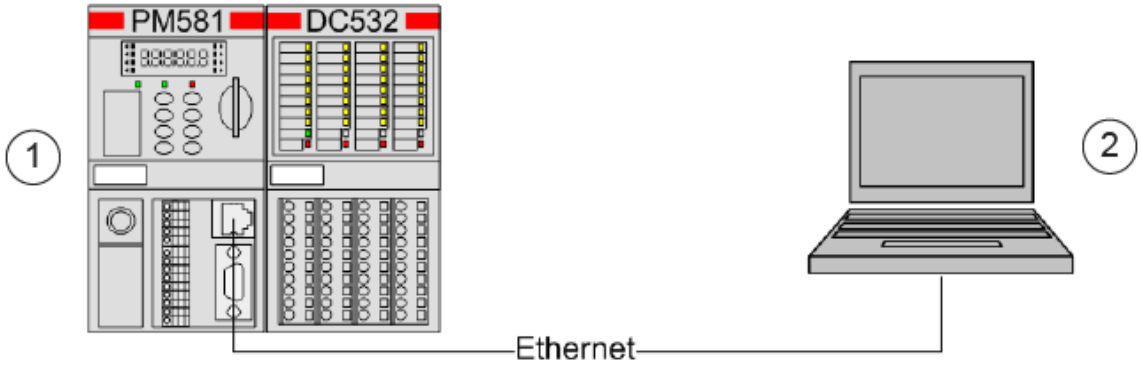


Fig. 692: Overview

1 AC500 application
2 telecontrol

- The Datalogging Library can be used as an event recorder. In this special mode data is continuously recorded in a ring buffer which can be read out after a certain event x (e.g. outage) in order to analyze the values especially before but also after the event x.
  OR
- Data can be logged only and on command transferred to the ftp area to be analyzed offline or taken out via the memory card.

The following figure gives an overview of the described interaction of the Datalogging function blocks. There is always an input function block needed which transfers the input data into data sets with timestamp for use by the Datalogger. An output function block receives the current or retrieved data from the Datalogger in case of communication or further processing. The input function blocks "LOG_xxx_INPUT", the function block "LOG_HANDLING" and the output function blocks "LOG_xxx_OUTPUT" communicate via SRAM FIFOin and FIFOout areas (defined in %M/%R area) in the memory. This SRAM FIFOs are intermediate buffers and help in decoupling time wise and speeding up the necessary write/read operations on the logging file structures significantly. These read/write operations on the files are done in blocks of data sets, enabling a comparably fast interaction with the otherwise slow file system.

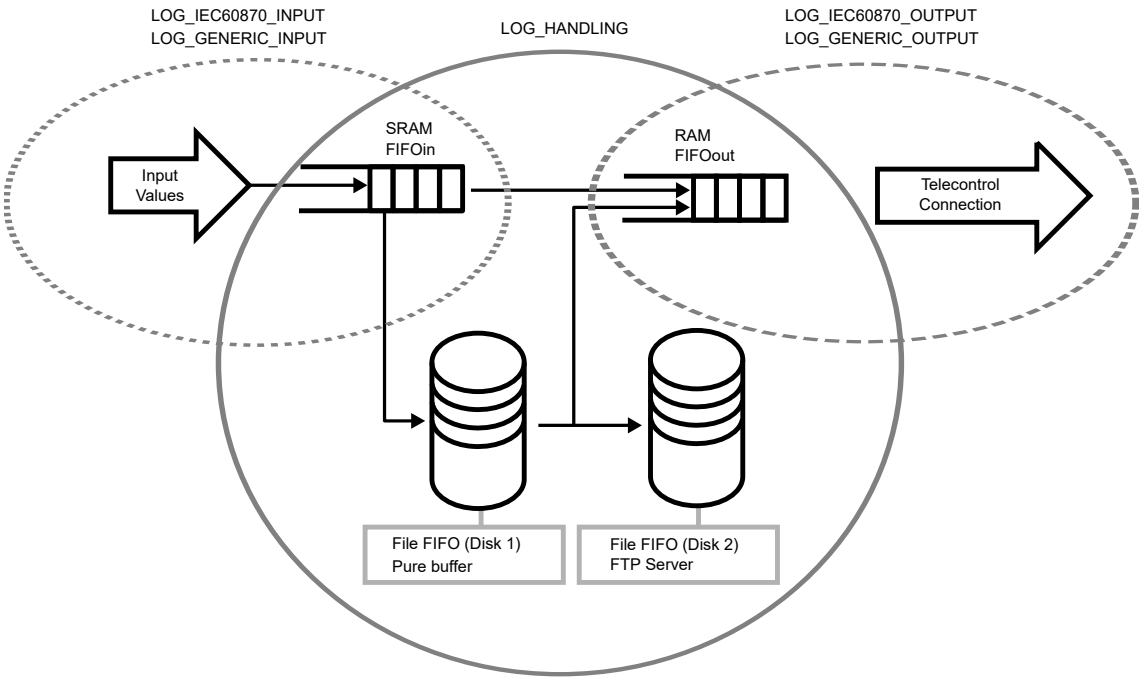


Fig. 693: Overview function blocks

Each Datalogging application requires the main function block LOG_HANDLING, one of the input function blocks to provide data to be logged and one output function block to retrieve the data.

As input and output function blocks two different types exist:

- For logging data of an interrupted IEC60870 communication, the function blocks LOG_IEC60870_INPUT and LOG_IEC60870_OUTPUT are provided. The IEC60870 Datalogging function blocks support the IEC data types and work internally with the standard AC500 IEC60870 Library. The IEC Datalogger output function block does not need special handling or control/inputs.
- For other types of general data LOG_GENERIC_INPUT and LOG_GENERIC_OUTPUT are provided. The generic Datalogging function blocks support an even larger variety of data types. The generic output function block needs to be hand-shaked with for each data set, in order to retrieve the data from the Datalogging files. Therefore the generic function blocks can also be used to integrate the data logging into any other protocol, e.g. Modbus.

The function block LOG_HANDLING ensures that also several consecutive and fast interruptions can be handled without losing data. While the log file is replayed, arriving new data is stored in the SRAM FIFOin and added to the Datalogging files (File FIFO) if the SRAM FIFOin becomes full (during that short time the log file replay is paused). Nevertheless any data send to a control station via a communication is always with the oldest data first (FIFIO = "First In First Out").

As it takes up to 30 seconds before a communication break is detected (e.g. with TCP/IP protocols by the AC500 hardware/firmware), the data rate at which data should be logged in case of a communication break has to be calculated and limited.

As an improvement a ping mechanism can be implemented in the substation. This was done in the example program for the IEC logger. With this ping the interruption is already detected after 1-2 seconds (can be configured in the example program).

As the SRAM FIFO has to store data during this time its size limits the data rate. The SRAM FIFO size is 160 data sets. This means the data rate should be lower than approximately max. datarate = 160 datasets/2 seconds => <5 data sets/second

The data rates for storing only without this detection can be much higher and depends on the CPU and memory type chosen. The data is always logged in directly readable csv format ↬ *Chapter 1.5.13.2.1.5 “CSV file formats” on page 3500*. Depending on the input function block and data type, the log file contains only one or up to 32 data variables per timestamped data set. The Datalogging files can be configured (up to 65k data sets, up to 999 concsecutive log files, name format).

Operating modes

This chapter describes the different operating modes of the Datalogging and their behavior.

- Mode 0/1: Buffer and disposal in chronologic order
  - Mode 0: Limited storage (keeps oldest, but stops if full)
  - Mode 1: Endless (ring buffer) operation modes (deletes oldest)
- Mode 2: Buffer and disposal via FTP, Log file(s) copied to ftp server area for further use
- Mode 3: Events Recorder, logs data before and after an event.

Mode 0/1: Buffer and disposal in chronologic order

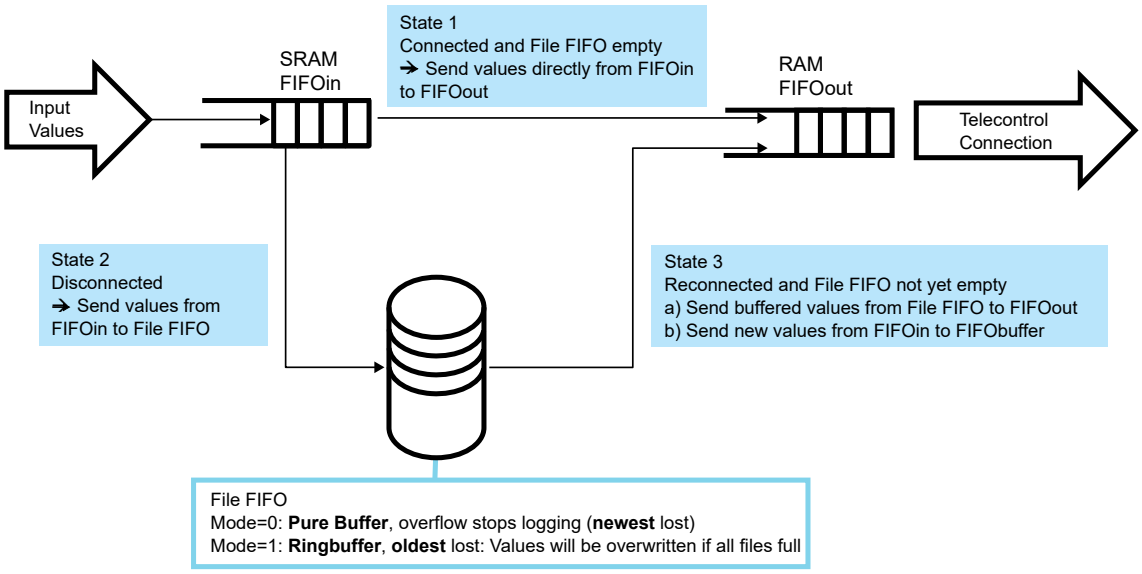


Fig. 694: Overview Mode 0/1

Mode 0/1 is for buffering the values from the AC500 application in case of a broken or intentionally interrupted connection between AC500 and telecontrol. In the normal state 1 the values are directly sent from the FIFOin (input values from application) to FIFOout (telecontrol connection). As soon as the connection is interrupted, the Datalogger changes to working state 2. The values are sent to the File FIFO instead. When the File FIFO is full, the Datalogging is stopped (Mode 0) or the oldest data will be overwritten (Mode 1 = ringbuffer). When the connection is established again and the RELEASE_HISTORY pin is triggered, the Datalogger changes to working state 3. It cares for disposal of the values in chronological order. The buffered values are written to FIFOout (working state 3a). This may take some time during which new values are coming from the application and stored into FIFOin. Before the FIFOin overflows the Datalogger switches to working state 3b and buffers the new values. After that it can continue with working state 3a. Only if the File FIFO is empty (all files deleted) the Datalogger changes back to normal state 1.

The advantage of Mode 0/1 is that all values (directly and buffered) are sent to telecontrol in strictly chronological order which is expected by most control stations (SCADA systems/historians).

As it takes up to 30 seconds before a communication break is detected (e.g. with TCP/IP protocols by the AC500 hardware/firmware), the data rate at which data should be logged in case of a communication break has to be calculated and limited.

Mode 2: Buffer and disposal via FTP

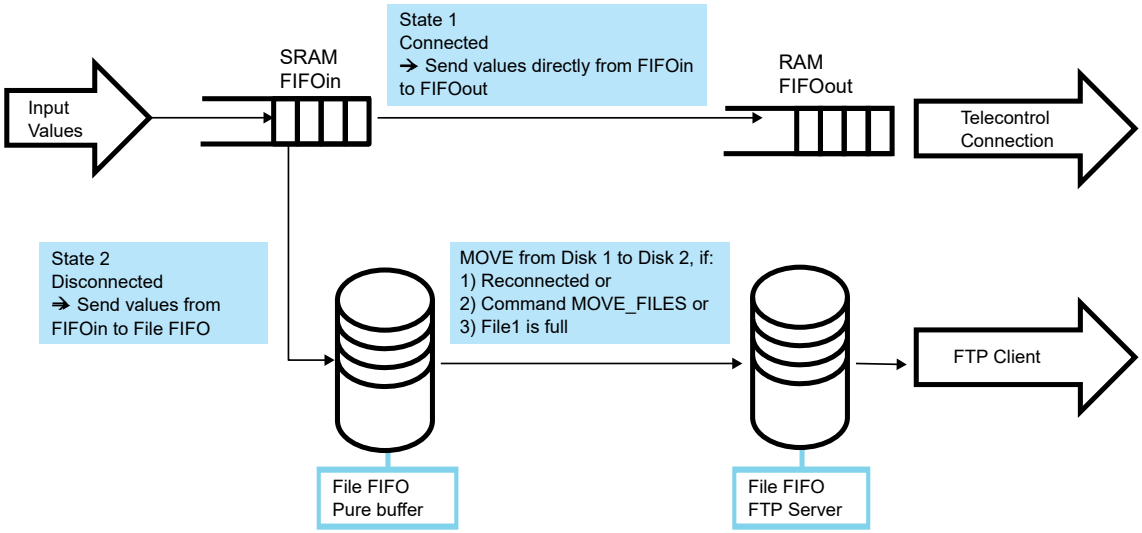


Fig. 695: Overview Mode 2

Mode 2 is also used for buffering the values from AC500 application in case of a broken connection between AC500 and telecontrol. State 1 and state 2 are similar to Mode 0/1. The difference is the disposal. When the connection is established again the Datalogger changes directly back to state 1 and the input values in FIFOin are directly sent to FIFOout (telecontrol connection). The buffered values in File FIFO are internally moved from disk 1 to disk 2 which can then be accessed or used by FTP (client or server). This move action can also be triggered by the command MOVE_FILES, or when file 1 is full. The advantage of Mode 2 is the immediate availability of the latest and all current values after an outage.

Mode 3: Events recorder

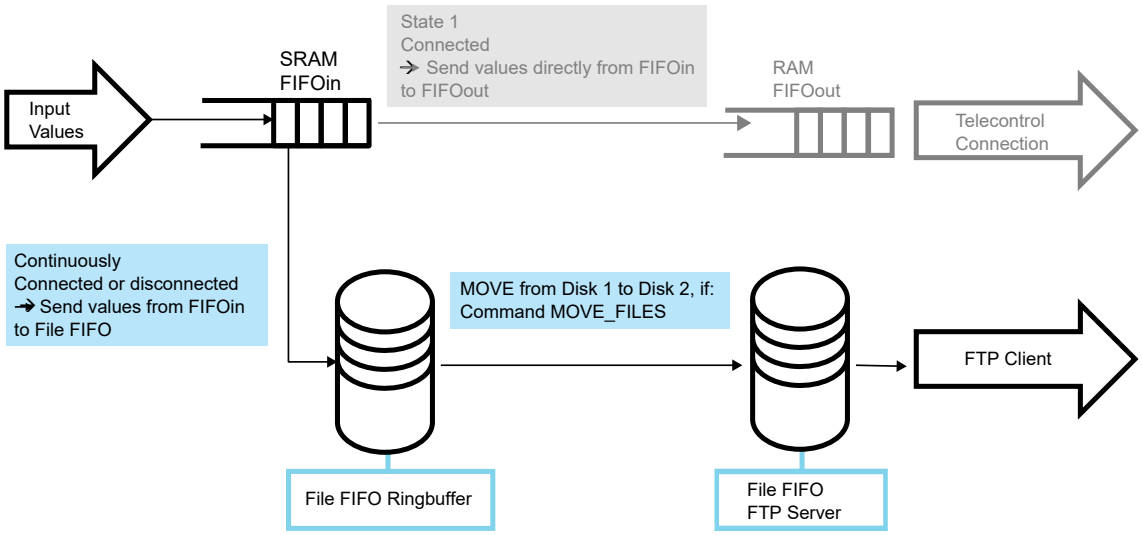


Fig. 696: Overview Mode 3

Mode 3 is used to record data values around an event, before and after the event x, e.g. outage of a part of the plant. The values are continuously recorded into the File FIFO file system independent of the connection status to telecontrol. If the File FIFO is full the oldest values are overwritten (ring buffer). Thus the File FIFO always contains the values from the past period n, which is depending on the amount of values per second and on the size of the File FIFO. When a certain event x occurs, the command MOVE_FILES can be given directly or after the period m. With the command MOVE_FILES the values in File FIFO are internally moved from disk1 to disk 2 and can be read out by an FTP action (client or server) when required.

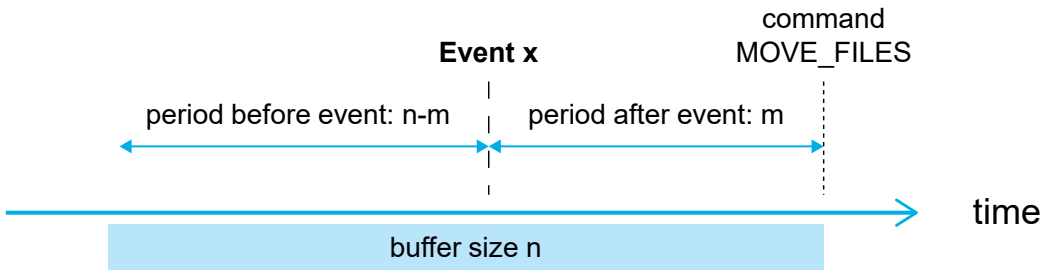


Fig. 697: The buffered values represent the time before the event (n-m) and after the event (m).

The advantage of mode 3 is that the values from the time period before the event (n-m) and after the event (m) are recorded and can help to reconstruct the cause and effect of the event.

Technical details

Parameter	Value
IEC 60870-5-104 protocol integrated in Datalogging, IEC data types	SinglePoint SP1/16, DoublePoint DP, IntegratedTotal IT1/16, MeasurementValue ME1/16
Generic logging to file(s); AC500 data types	BIN, BYTE, INT, DINT, WORD, DWORD, REAL
Trigger	Cyclic, event, tolerance
File format	CSV, including local timestamp. Generic Logging with separate ID (max. 8 characters), IEC Logging with IEC addresses. Timestamped data sets contain 1-16 values (IEC) depending on type logged. Generic contain different number of values, depending on type logged. BINARY: max. 58 BYTE: max. 88 INT: max. 50 WORD: max. 58 DINT: max. 29 DWORD: max. 31 REAL: max. 27
Datalogging target	Internal flashdisk or memory card, power fail input for memory card (from USV)
Datalogging file sizes and storage depth	FIFO storage in file system, Datalogging depth only limited by memory size
Configurable File FIFO	Number of files (max. 999); number of data sets per file (max. 65565)
Internal SRAM FIFO size	160 data sets
Block write mode into files	Up to 50 data sets/second per max. 88 values

Parameter	Value
Operation modes	Mode 0: Buffer and disposal in chronologic order Limited storage (keeps oldest, but stops if full) Mode 1: Buffer and disposal in chronologic order Endless (ring buffer) operation modes (deletes oldest) Mode 2: Buffer and disposal via FTP, Log file(s) copied to ftp server area for further use Mode 3: Events Recorder, logs data before and after an event.
Suppported software/firmware	V2.3 or higher
Current restrictions	One logger per PLC One IEC 60870 connection only: While log file is replayed, no other current information via IEC available Usable solutions: ● Delay replay of log file after connection returned to allow a “general inquiry” ● Use of Mode 2

File names

File names are renamed according to storing time with an accuracy of 100ms. The files are renamed from "filename.csv" to a file name with timestamp and with or without file extension, according to input DISK2_EXTENSION *“DISK2_EXTENTION (DISK2 extention)” on page 3507*.

File name with timestamp	02281448.593 = February 28th, 2:48pm (14:48), 59s, 300ms
File name with timestamp and file extension	02281448.csv = February 28th, 2:48pm (14:48)

Preconditions

The Datalogging Library supports CPU PM573 or higher.

The Datalogging Library does not support CPUs of the AC500-eCo-series.

CAUTION!
Failure in Processing of the function blocks.
– The function blocks LOGxxxxxx_INPUT, LOG_HANDLING and LOGxxxxxx_OUTPUT must be put in the same task.

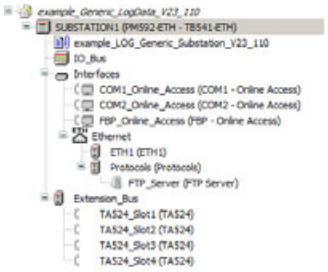
CPU firmware must be V 2.3.3 or higher.

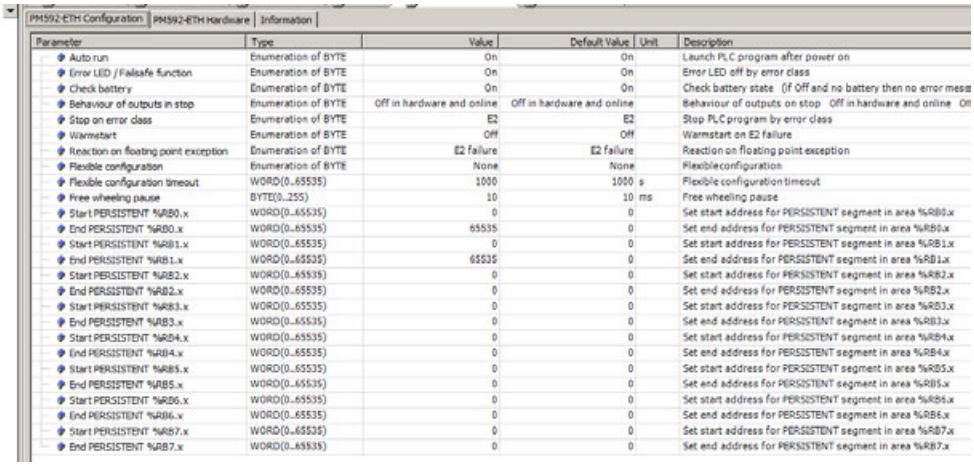
Use memory card from ABB with sufficient free space, at least 1,5 x file size as configured (file size is depending on input MAX_DS_IN_FILE (Log_Handling)).

Maximum number of files (input of FBLOG_HANDLING) is limited to 500 (ABB memory card is formatted with FAT).

The %R area must be set to PERSISTANT.

▷ Do the following settings in the AC500 configuration:





Automatically the following areas in the global %R/%M area are occupied:

- In the segment 0 of the %R/%M-area only the upper part is occupied in each case, %R starting at %RD13000, %M starting at %MD16370.
- The segment 1 is occupied completely in each case.

```
VAR_GLOBAL
(*
*************************************************************Segment
0 %R ************************************************************ *)
    RD0.0 ... RD0.12999 free for user - e.g.  MODBUS
(*
*************************************************************Segment
0 %M ************************************************************ *)
    MD0.0 ... MD0.16369 free for user - e.g.  MODBUS
```

CSV file formats

Generic data logger

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ID	TIMESTAMP	MSEC	TYPE(NUM)	TYP(TXT)	LENGTH	DATA 1...max																
2	BOOL	DT#2015-05-18-14:54:33	491	1	BOOL	58	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
3	BYTE	DT#2015-05-18-14:54:33	491	2	BYTE	88	2	18	5	223	14	54	33	7	8	9	10	11	12	13	14	15	16
4	INT	DT#2015-05-18-14:54:33	491	3	INT	50	3	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
5	WORD	DT#2015-05-18-14:54:33	491	4	WORD	58	4	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
6	DINT	DT#2015-05-18-14:54:33	492	5	DINT	29	5	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
7	DWORD	DT#2015-05-18-14:54:33	492	6	DWORD	31	6	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16
8	REAL	DT#2015-05-18-14:54:33	492	7	REAL	27	7.0	18.0	5.0	2015.0	14.0	54.0	33.0	7.777	8.888	9.999	10.0	11.0	12.0	13.0	14.0	15.0	16.0

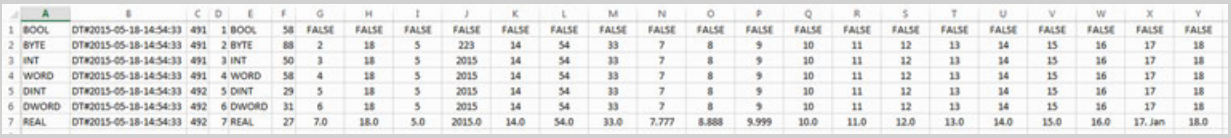
Fig. 698: Explanation of the file formats

1 data set consists of:

ID (8 any char) + TimeStamp + msec + Datatype(num) + Datatype(txt) + Length(following data) + max 32 data

Parameter	Value
ID =	ID of LOG_GENERIC_INPUT (max 8 any characters)
Datatype =	DATATYPE of LOG_GENERIC_INPUT (1…7)
Length =	LENGTH of LOG_GENERIC_INPUT (max 88) BINARY (58); BYTE (88); INT (50); WORD (58); DINT (29); DWORD (31); REAL (27)

Example



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	BOOL	DT#2015-05-18-14:54:33	491	1	BOOL	58	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE	FALSE
2	BYTE	DT#2015-05-18-14:54:33	491	2	BYTE	88	2	18	5	223	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
3	INT	DT#2015-05-18-14:54:33	491	3	INT	50	3	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
4	WORD	DT#2015-05-18-14:54:33	491	4	WORD	58	4	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
5	DINT	DT#2015-05-18-14:54:33	492	5	DINT	29	5	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
6	DWORD	DT#2015-05-18-14:54:33	492	6	DWORD	31	6	18	5	2015	14	54	33	7	8	9	10	11	12	13	14	15	16	17	18
7	REAL	DT#2015-05-18-14:54:33	492	7	REAL	27	7.0	18.0	5.0	2015.0	14.0	54.0	33.0	7.777	8.888	9.999	10.0	11.0	12.0	13.0	14.0	15.0	16.0	17. Jan	18.0

Fig. 699: File opened directly with Excel

IEC60870 data-logger

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	TimeStamp	msec	IEC_TYPE	TypText	Slot	Con	Idx	NoDP	Quality_Bits(Byte)	Quality (SQ)	GADU	IAD3/2/1(1)	Var1	IAD3/2/1(2)	Var2	IAD3/2/1(3)	Var3
2	DT#2012-02-08-10:52:04	566	1	SP1	0	0	198	1	0	0	65535	66816	TRUE				
3	DT#2012-02-08-10:52:04	566	2	SP16	0	0	0	16	0	0	65535	1114369	TRUE	1179905	FALSE	1245441	FALSE
4	DT#2012-02-08-10:52:04	566	4	IT1	0	0	80	1	0	0	65535	768	4991880				
5	DT#2012-02-08-10:52:04	566	5	IT16	0	0	85	16	0	0	65535	1049344	8	1114880	2	1180416	220
6	DT#2012-02-08-10:52:04	567	6	ME1	0	0	203	1	0	0	65535	512	374391.0				
7	DT#2012-02-08-10:52:04	567	7	ME16	0	0	117	16	0	0	65535	1049088	8.0	1114624	2.0	1180160	220.0

Fig. 700: Explanation of the file formats

Table 248: 1 data set consists of the following parameters

Parameter	Explanation
IEC_TYPE =	IEC_TYPE of LOG_IEC60870_INPUT (1…7)
Slot/Con/Idx/NoDP =	PINGROUP of LOG_IEC60870_INPUT (1…7)
Quality_Bits(Byte) =	IV/NT/SB/BL/CA/CY/QOV (packed in 1 byte) of LOG_IEC60870_INPUT (1…7)
Quality (SQ) =	SQ of LOG_IEC60870_INPUT (1…7)
GADU =	calculated internally, from CBP-Configurator (Gadu1+Gadu2)
IAD3/2/1(n) =	calculated internally, for every datapoint separately, from CBP-Configurator (IAD1+IAD2+IAD3)
n =	1 or 16, in case of DP is n=2
VAR(n) =	variable

IEC type	Values	Meaning
SP1	-	SinglePoint 1
SP16	-	SinglePoint 16
DP	-	DoublePoint
IT1	-	IntegratedTotal 1
IT1616	-	IntegratedTotal 16
ME1	-	MeasurementValue 1

IEC type	Values	Meaning
ME16	-	MeasurementValue 16
Quality_Bits(Byte):	Quality.0 := IV;	Quality with quality invalid
	Quality.1 := NT;	Quality not topical
	Quality.2 := SB;	Quality substituted
	Quality.3 := BL;	Quality blocked
	Quality.4 := CA;	Quality with quality carry
	Quality.5 := CY;	Quality with quality counter adjusted
	Quality.6 := QOV;	Quality Overflow Quality
	Quality.7 := Reserve;	(*Reserve - Quality*)
Quality SQ(Byte)	SQ	Quality sequence number

Example

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y
1	DT#2012-02-08-10:52:04	566	1	SP1	0	0	198	1	0	0	65535	66816	TRUE												
2	DT#2012-02-08-10:52:04	566	2	SP16	0	0	0	16	0	0	65535	1114369	TRUE	1179905	FALSE	1245441	FALSE	1310977	FALSE	1376513	TRUE	1442049	FALSE	1507585	FALSE
3	DT#2012-02-08-10:52:04	566	4	IT1	0	0	80	1	0	0	65535	768	4991880												
4	DT#2012-02-08-10:52:04	566	5	IT16	0	0	85	16	0	0	65535	1049344	8	1114880	2	1180416	220	1245952	10	1311488	52	1377024	4	1442560	249594
5	DT#2012-02-08-10:52:04	567	6	ME1	0	0	203	1	0	0	65535	512	374391.0												
6	DT#2012-02-08-10:52:04	567	7	ME16	0	0	117	16	0	0	65535	1049088	8.0	1114624	2.0	1180160	220.0	1245696	10.0	1311232	52.0	1376768	4.0	1442304	249594.0

Fig. 701: File opened directly with Excel

1.5.13.2.2 LOG_HANDLING

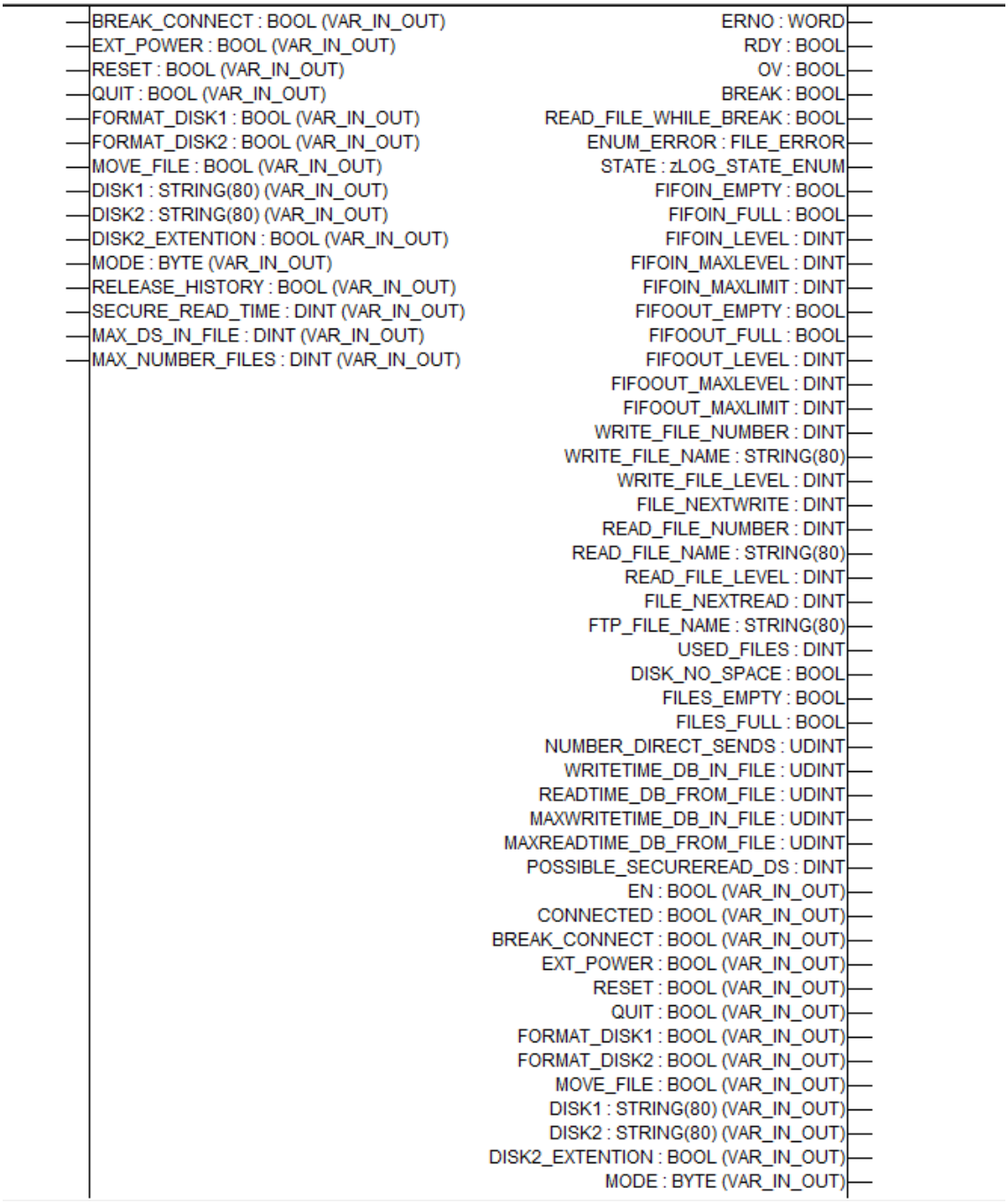
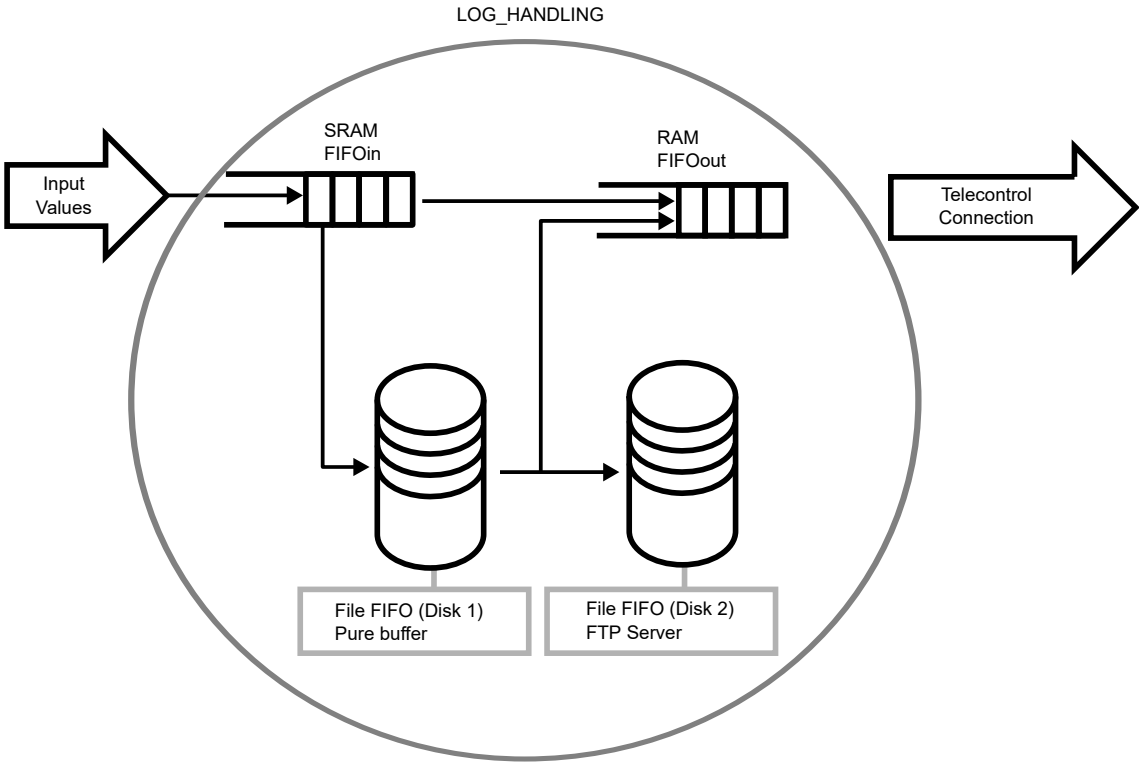


Table 249: General information

Available as of runtime system	V2.3 and above
Included in library	LogData_AC500_V23.lib
Type	Function block with historical values

Depending on the selected mode this function block is used to

- directly transfer data messages in IEC60870 format or in generic format from SRAM_FIFOin to the RAM_FIFOout in case of an existing communication.
- store data sets in IEC60870 format or in generic format in files on flash disk or memory card for other use or because a communication line is broken.
- provide these logged data sets at the RAM_FIFOout output in case of recurring communication with higher priority than the current data traffic or via FTP server of the PLC.

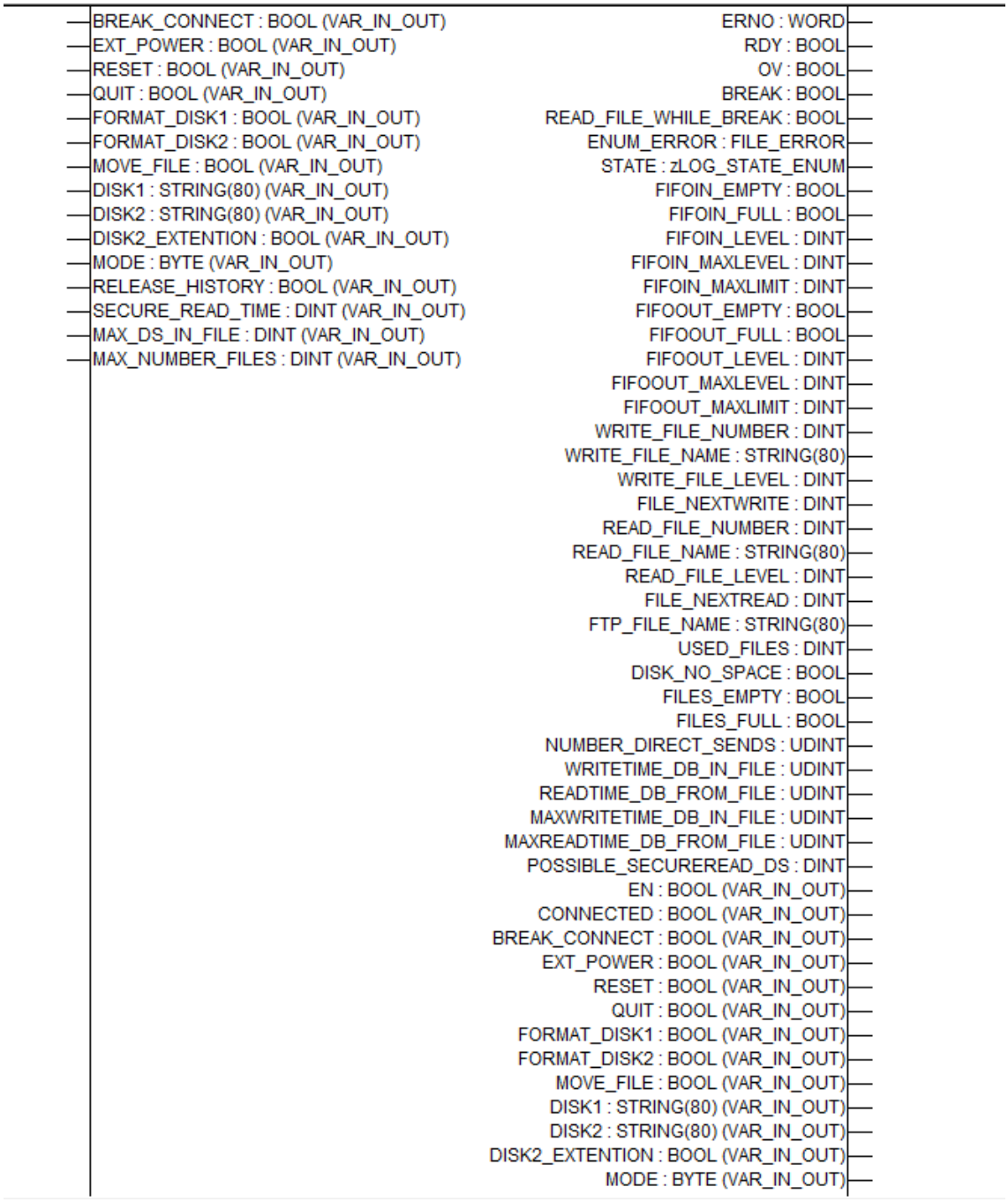


Depending on the mode the function block can have three main working states:

- State 1: Existing communication (CONNECT) und empty FILE-FIFO: Data sets are written into SRAM_FIFOin (First In First Out), which are either from LOG_IEC60870_INPUT or LOG_GENERIC_INPUT (cannot be mixed), and are transported to RAM_FIFOout (First In First Out). There they are either
  - decoded by LOG_IEC60870_OUTPUT and transmitted or
  - decoded by LOG_GENERIC_OUTPUT and provided on its outputs as real data for further transmission (dependant on the chosen input variant: IEC or generic).
- State 2: Communication failure (DISCONNECT): Data sets, which were written either by LOG_IEC60870_INPUT or LOG_GENERIC_INPUT (cannot be mixed) into the SRAM_FIFOin, will be transmitted into the FILE-FIFO (First In First Out) and stored.
- State 3: Write/Read Modes and FTP Modes *Chapter 1.5.13.2.1.1 “Operating modes” on page 3496*

This function block realises writing and reading of max. 999 files with max. 65535 data sets in a FIFO principle. Data storage can be either the FLASHDISK of a PM59x (with SafeFAT), or an SDCARD (without SafeFAT). In case of SDCARD it must be ensured that after main power failure, for approximately 3...5 sec the 24 V supply is maintained (input extPower = FALSE), in order to close open files properly.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

CONNECTED (connected)

Data type: BOOL

Input signal, if the connection with the receiving/control station is established.

It is irrelevant in FTP Mode (Mode 3) because in FTP Mode no data is send by the function block to the control station.

BREAK_CONNECT (break connect)

Data type: BOOL

For testing purposes. TRUE indicates to the function block that the connection to the other station is interrupted.

It is irrelevant in FTP Mode (Mode 3) because in FTP Mode no data is send by the function block to the control station.

EXT_POWER (external power)

Data type: BOOL

External power/opener. FALSE: the function block completes the last write and read actions, closes all open files and goes to an idle-state. This is signaled at the DONE output. On the output STATE the text „zLOG_NO_EXTERNAL_POWER“ is given out.

RESET (reset)

Data type: BOOL

Complete reset of all data fields, historical data and actual status of the function block. During this action the output STATE shows the text „zLOG_RESET_ACTIVE“. The end of this action is signaled on the output DONE.

QUIT (quit)

Data type: BOOL

The input will acknowledge at an 0/1 edge the error message of the function block, if the message is not active anymore.

FORMAT_DISK1 (format DISK1)

Data type: BOOL

At an 0/1 edge the input formats the drive given on DISK1. During this action the output STATE shows the text „zLOG_DISK1_FORMAT_ACTIVE“. The end of this action is signaled on the output DONE. On the output STATE the text „zLOG_DISK_HAS_BEEN_FORMATTED“ is shown.

FORMAT_DISK2 (format DISK2)

Data type: BOOL

At an 0/1 edge the input formats the drive given on DISK2. During this action the output STATE shows the text „zLOG_DISK2_FORMAT_ACTIVE“. The end of this action is signaled on the output DONE. On the output STATE the text „zLOG_DISK_HAS_BEEN_FORMATTED“ is shown.

MOVE_FILE (move file)

Data type: BOOL

Only active in Mode 2 and Mode 3. At an 0/1 edge the input closes and moves the file (Mode 2) or multiple files (Mode 3) currently written on primary DISK1 into the folder FTP of the secondary DISK2 (both DISKs can be identical). The folder FTP is created automatically. The files are renamed.

After read-out via FTP client the moved and renamed files can be deleted on the FTP server by the user. During the action the output STATE shows the text „zLOG_MOVE_ACTIVE“. The end of this action is signaled on the output DONE.

DISK1 (DISK1)

Data type: STRING

Name of the primary drive ("flashdisk" or "sdcard") which is used for File FIFO buffer. DISK1 and DISK2 can be identical. Internally the primary drive is used with "file1...file999..." (Mode 0/1/3) or with "file1" (Mode 2) which the function block automatically creates and deletes.

DISK2 (DISK2)

Data type: STRING

Name of the secondary drive ("flashdisk" or "sdcard") which is used for File FIFO FTP Server. DISK1 and DISK2 can be identical.

DISK2_EXTENTION (DISK2 extention)

Data type: BOOL

Extention-Switch for the naming of the files to be moved to the FTP folder. The files stored in the FTP folder are renamed with a name representing a timestamp, in order to clearly be able to see the timely order when reading or processing the files.

Possible are:

- DISK2_EXTENTION = FALSE
- DISK2_EXTENTION = TRUE

CAUTION!

Accidentially overwritten files

When DISK2_EXTENSION = TRUE and the creation time between two files is too short to be visible in the name (< 1 minute), the first file is overwritten.

MODE (mode)

Data type: BYTE

- Mode 0 *Chapter 1.5.13.2.1.1 “Operating modes” on page 3496*
- Mode 1 *Chapter 1.5.13.2.1.1 “Operating modes” on page 3496*
- Mode 2 *Chapter 1.5.13.2.1.1 “Operating modes” on page 3496*
- Mode 3 *Chapter 1.5.13.2.1.1 “Operating modes” on page 3496*

RELEASE_HISTORY (release historical data transfer)

Data type: BOOL

External release for reading out the saved data from storage. This input is necessary when using IEC60870 logging. After communication is established again, the input keeps historical data back until control station sends a general interogation to get the current status. If this function is not needed, input=TRUE.

SECURE_READ_TIME (secure read time in ms)

Data type: DINT

Estimation of time (in ms) which passes after the failure of a connection until this failure is detected and the output CONNECT changes to FALSE. Under circumstances a communication failure can only be detected after a longer time without data transfer. Therefore in normal operation a measurement should be done for getting the average number of data sets arriving to the SRAM-FIFOin by SECURE_READ_TIME. In case of Connection fault this amount of data sets in the SRAM-FIFOin is taken from past and additionally stored into the FILE-FIFO, in order to avoid data losses by the delayed Connection fault recognition. At "0" no SecureRead is done.

MAX_DS_IN_FILE (max number of data sets in one file)

Data type: DINT

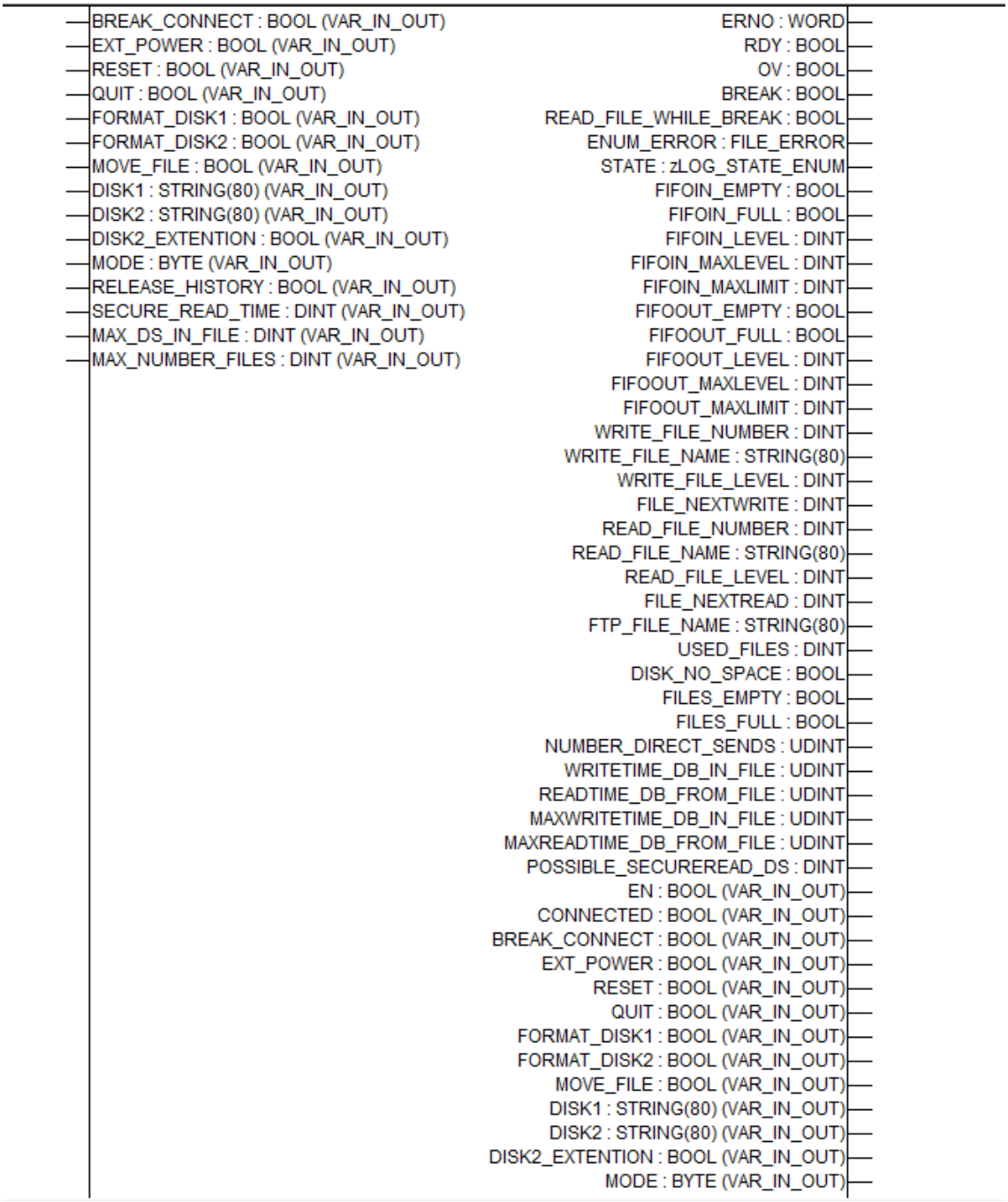
After reaching this limit the file is closed and a new file is opened. The configurable upper limit is 65535. An error is created if the input value is higher. A change of this value after start of the Datalogger is not allowed and will lead to faults in Mode 0 and Mode 1 when reading back values.

MAX_NUMBER_FILES (max number of files)

Data type: DINT

After reaching this limit no further data sets can be stored. If further data sets are arriving OV is signaled. The adjustable limit is 999. An error will be created if the input value is higher. A change of this value after start of the Datalogger is not allowed and will lead to faults in Mode 0 and Mode 1 when reading back values. An exception is Mode 3, no values are read back and therefore a change in operation is allowed, e.g. to obtain flexible timing.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

In addition, if errors occur while executing the IEC standard function blocks which are used internally, on the output ERNO a related error code *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529* will be given (ERR = TRUE).

RDY (ready)

Data type: BOOL

Output will be TRUE, if all external triggered activities of the function block are processed (FORMAT, MOVE, RESET, QUIT). Blinks if input EXT_POWER is missing.

OV (overrun)

Data type: BOOL

Output will be TRUE, if an overflow occurs. On this output a rising edge (FALSE -> TRUE) shows that communication/logging request occurred too fast and therefore the SRAM-FIFOin is full.

BREAK (break connect)

Data type: BOOL

Active, while available connection and while reading of historical values the SRAM-FIFOin is nearly full with new data sets. If BREAK is active, the SRAM-FIFOin-data will be written to FILE-FIFO despite CONNECT, in order to widely avoid OV of the SRAM-FIFOin. However, if incoming data sets arrive faster as they can be written to files, the system will reach a limit. Then an OV stop cannot be avoided.

READ_FILE_WHILE_BREAK (read file while break)

Data type: BOOL

Safety function: Output will be active, if all available files are full during CONNECT and BREAK. In this case primarily the oldest file of the FILE-FIFO is read out, regardless of OV.

ENUM_ERROR (eError from CAA Library)

Data type: FILE_ERROR

Detailed error messages of the used CAA_File function block Library. Output will be flagged, if an error occurs while execution of the function block. For error messages refer to chapter 'Error Messages' of the CAA_File function block Library.

STATE (enumeration of type zLOG_STATE_ENUM)	Data type: zLOG_STATE_ENUM Clear text messages of errors and states. The output uses the enumeration of the data type zLOG_STATE_ENUM declared in "data types".
FIFOIN_EMPTY (FIFOin empty)	Data type: BOOL SRAM-FIFOin is empty.
FIFOIN_FULL (FIFOin full)	Data type: BOOL SRAM-FIFOin is full (maximum 161 data sets).
FIFOIN_LEVEL (FIFOin level)	Data type: DINT Output represents the current level of SRAM-FIFOin %R area (zLOG_FIFOIN_LEVEL).
FIFOIN_MAX-LEVEL (FIFOin max level)	Data type: DINT Maximum SRAM-FIFOin filling level since EN=TRUE.
FIFOIN_MAX-LIMIT	Data type: DINT Standard value = 161.
FIFOOUT_EMPTY (FIFOout empty)	Data type: BOOL RAM-FIFOout is empty.
FIFOOUT _FULL (FIFOout full)	Data type: BOOL RAM-FIFOout is full (max 161 data sets).
FIFOOUT _LEVEL (FIFOout level)	Data type: DINT RAM-FIFOout filling level. Output represents current level of SRAM-FIFOin %R area (stored value).
FIFOOUT _MAX-LEVEL (FIFOout max level)	Data type: DINT Maximum RAM-FIFOout filling level since EN=TRUE.
FIFOOUT_MAX-LIMIT	Data type: DINT Standard value = 161.
WRITE_FILE_NUMBER (number of current file written to)	Data type: DINT Number of the current file written to (1…999).
WRITE_FILE_NAME (name of current file written to)	Data type: STRING Name of the current file written to (e.g. file1…file999).

WRITE_FILE_LEVEL (level of current file written to)
Data type: DINT
Filling level of the file currently written to.

FILE_NEXT-WRITE (next write position in current file written to)
Data type: DINT
Next write position in current file written to.

READ_FILE_NUMBER (number of current file)
Data type: DINT
Number of current file read from (1…999).

READ_FILE_NAME (name of current file)
Data type: STRING
Name of current file read from (for example file1…file999).

READ_FILE_LEVEL (level of current file)
Data type: DINT
Filling level of current file read from.

FILE_NEXT-READ (next read position in current file)
Data type: DINT
Next read position in current file read from.

FTP_FILE_NAME (name of current file written to)
Data type: STRING
Complete current path and name of the file which will be transferred to the folder FTP (e.g. "flashdisk\FTP\02141149.545") in Mode 2 in the following cases:

- file 1 is full
- communication returns
- per external trigger by setting the input MOVE_FILE

In Mode 3 the move is triggered only by setting input MOVE_FILE.

USED_FILES (currently used files)
Data type: DINT
Number of files written to.

DISK_NO_SPACE (disk no space)
Data type: BOOL
Data storage is full.

FILES_EMPTY (files empty)
Data type: BOOL
All files are empty.

FILES_FULL (files full)
Data type: BOOL
All files are full. No further data sets can be stored.

NUMBER_DIRECT_SENDS (number of direct sent data sets
Data type: UDINT
Number of the current data sets sent to the control station side since EN=TRUE and coming from SRAM-FIFO (without storing in FILE-FIFO).

WRITE-TIME_DB_IN_FILE (current time to write one data set in file)

Data type: UDINT

Current time for writing one data set into the file. When this time is growing over time, this is a sign of a growing fragmentation of the disk. In Mode 0 and Mode 1 and "flashdisk", the flash disk is defragmented automatically (FORMAT) as soon as all the historical data has been sent to the control station. In Mode 2 this may not happen as under circumstances files are stored in the FTP folder and have not been picked up by FTP yet.

In case of a memory card as storage medium, a FORMAT is basically locked because otherwise it may cause incompatibility between PLC and PC. That means a memory card defragmented via FORMAT in the PLC is possibly no longer readable on a PC.

READ-TIME_DB_FROM_FILE (current time to read one data set from file)

Data type: UDINT

Current time for reading a data set from the file. When this time is growing over time, this is a sign of a growing fragmentation of the disk. In Mode 0 and Mode 1 and "flashdisk," the flash disk is defragmented automatically (FORMAT) as soon as all the historical data has been sent to the control station. In Mode 2 this may not happen as under circumstances files are stored in the FTP folder and have not been picked up by FTP yet.

In case of a memory card as storage medium, a FORMAT is basically locked because otherwise it may cause incompatibility between PLC and PC. That means a memory card defragmented via FORMAT in the PLC is possibly no longer readable on a PC.

MAXWRITE-TIME_DB_IN_FILE (max time to write one data block in file since EN)

Data type: UDINT

Maximum time needed until now to write one data block in the file. When this time is growing over time, this is a sign of a growing fragmentation of the disk. In Mode 0 and Mode 1 and "flashdisk," the flash disk is defragmented automatically (FORMAT) as soon as all the historical data has been sent to the control station. In Mode 2 this may not happen as under circumstances files are stored in the FTP folder and have not been picked up by FTP yet.

In case of a memory card as storage medium, a FORMAT is basically locked because otherwise it may cause incompatibility between PLC and PC. That means a memory card defragmented via FORMAT in the PLC is possibly no longer readable on a PC.

MAXREAD-TIME_DB_FROM_FILE (max time to read one data block from file since EN)

Data type: UDINT

Maximum time needed until now to read one data block from the file. When this time is growing over time, this is a sign of a growing fragmentation of the disk. In Mode 0 and Mode 1 and "flashdisk," the flash disk is defragmented automatically (FORMAT) as soon as all the historical data has been sent to the control station. In Mode 2 this may not happen as under circumstances files are stored in the FTP folder and have not been picked up by FTP yet.

In case of a memory card as storage medium, a FORMAT is basically locked because otherwise it may cause incompatibility between PLC and PC. That means a memory card defragmented via FORMAT in the PLC is possibly no longer readable on a PC.

POSSIBLE_SECURE READ_DS (possible secure read data sets)

Data type: UDINT

Number of data sets delivered to SRAM-FIFOin (during the time indicated on input "SECURE_READ_TIME" in ms). In case of connection fault this number of past data sets in SRAM-FIFOin are taken additionally into account for writing to FILE-FIFO in order to avoid data losses caused by the delayed connection fault – recognition.

Integrated visualization

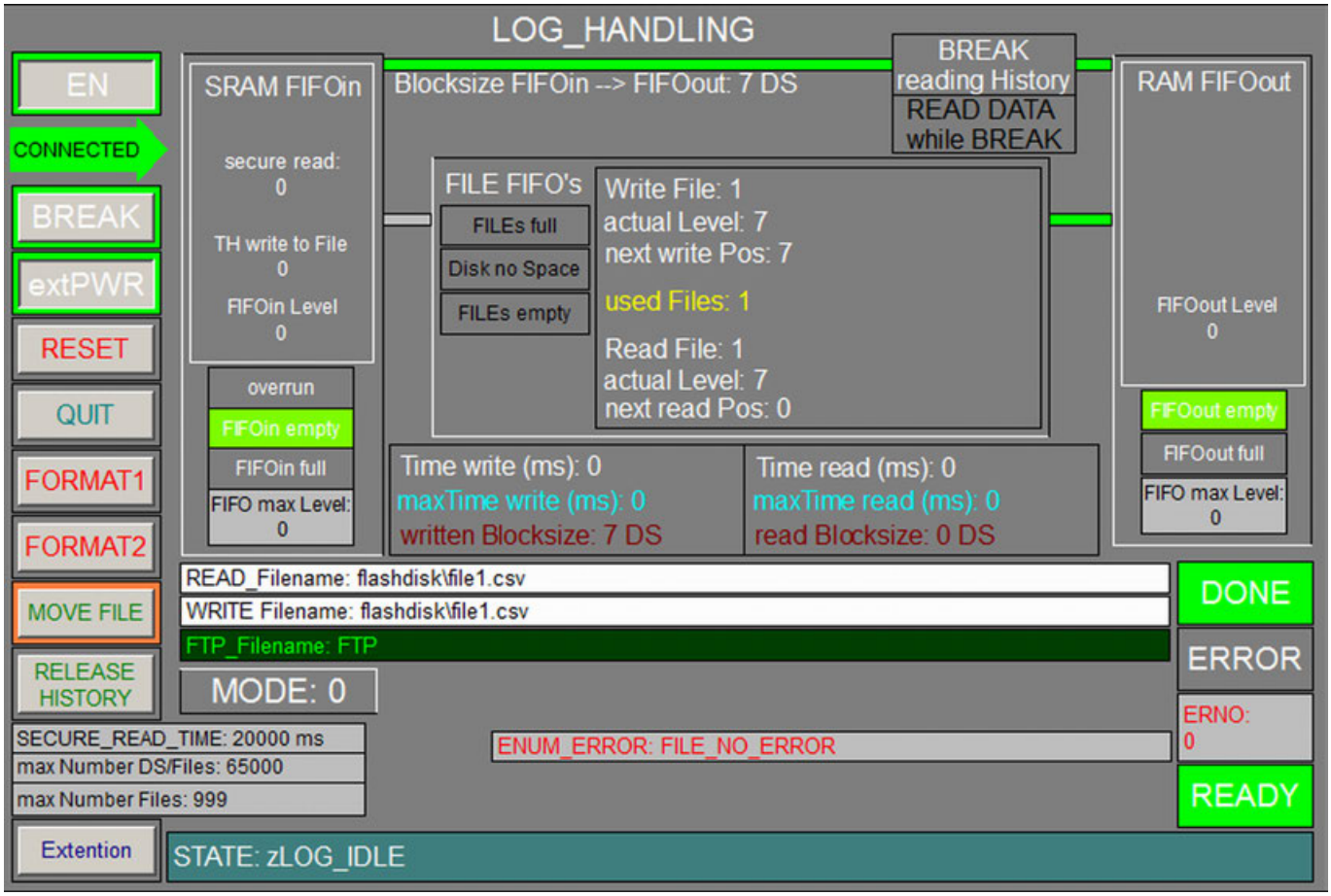


Fig. 702: Visualization

1.5.13.2.3 LOG_IEC60870_INPUT

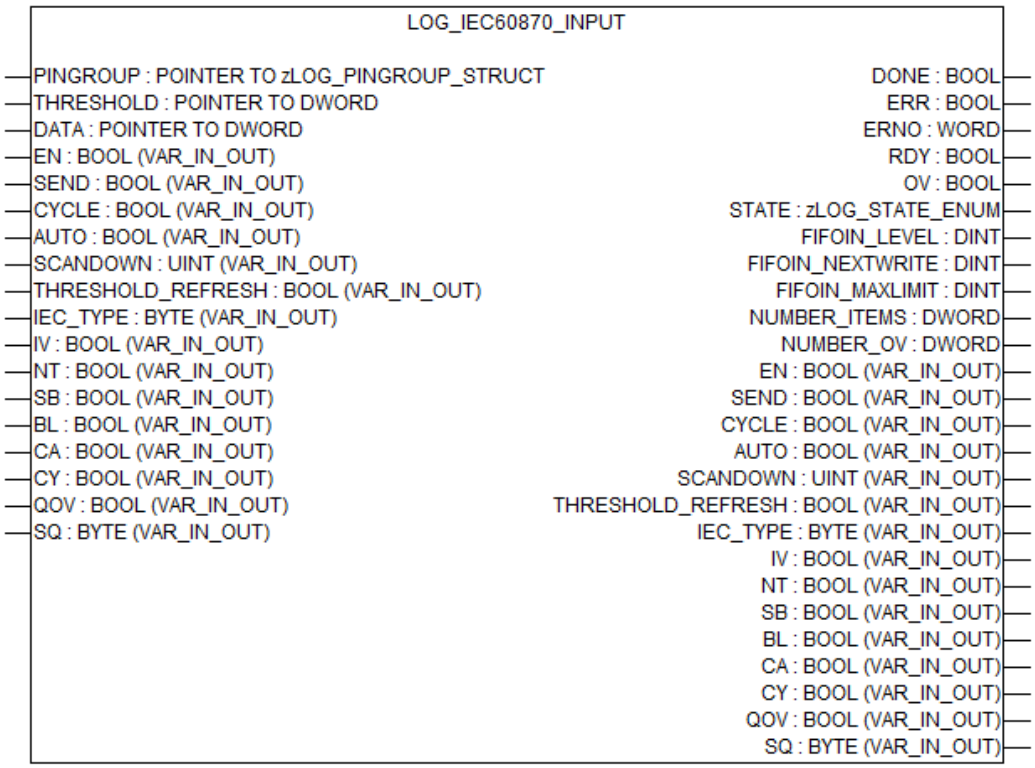


Table 250: General information

Available as of runtime system	V2.3 and above
Included in library	LogData_AC500_V23.lib
Type	Function block with historical values

The function block is used for writing data sets into the SRAM-FIFOin of the Datalogger according to IEC 60870-5-104 protocol. According to the corresponding data type up to 16 data points of this data type can be written at the same time.

The data sets are written in format *.csv as two-dimensional ARRAY. Every part of each data set is separated by SEMICOLON. Each data set is ended with a CR/LF. Every segment of the %R area of the SRAM-FIFOin contains maximum one ARRAY[0..160] OF ARRAY[0..399] OF BYTE. The ARRAY reserves the complete segment 1 of the %R area of the controler.

A complete segment (segment 1 of the %R-area) is used for the SRAM-FIFOin (zLOG_CONST_FIFOLIMIT =160).

Here, the function block is used to write the data sets into the SRAM-FIFOin of the PLC CPU and not to be sent directly to the control station. This is done with function block LOG_IEC60870_OUTPUT which uses the above mentioned function blocks.

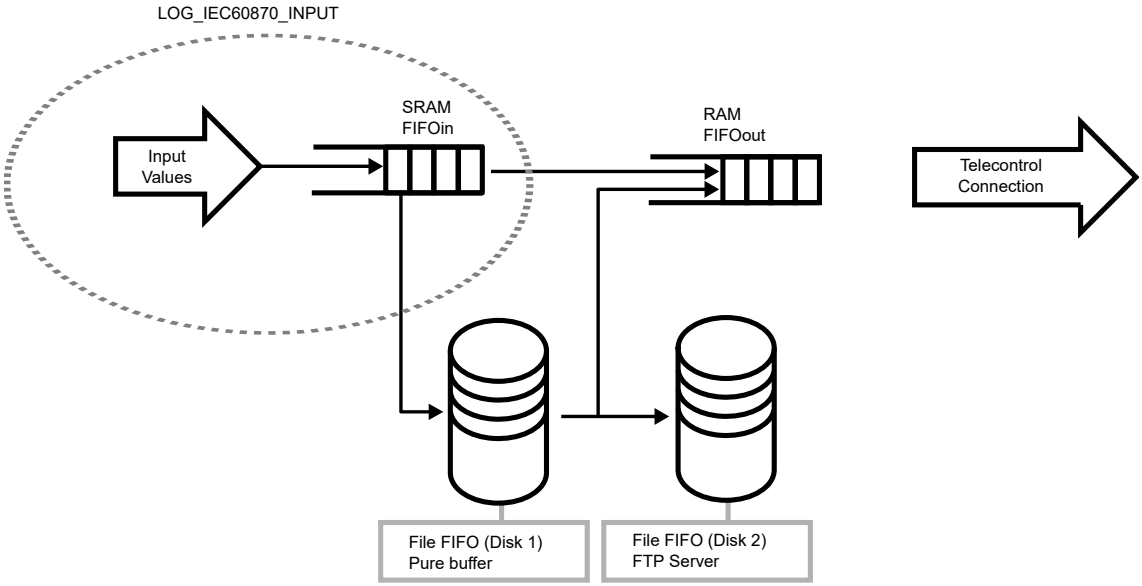
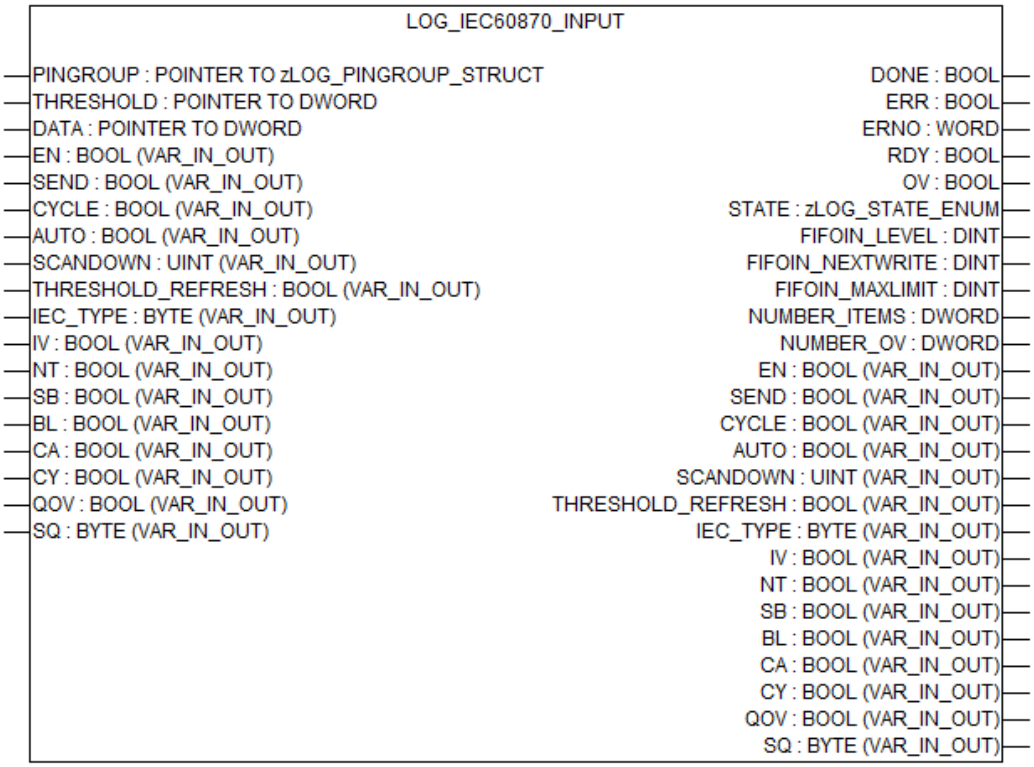


Fig. 703: Functionality

CAUTION!
Each used instance of the function block needs only one cycle for writing the data into FIFOin. For "n" succeeding instances called in the same cycle, "n" entrances/data sets are entered into FIFOin. Consider that FIFOin is limited by the limit of the segments of the %R area. That means, the upper limit of entries to FIFOin at the same time (in the same cycle) is 160.

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

PINGROUP (pin group via ADR-Operator)

Data type: POINTER TO zLOG_IEC60870_FIFO_ENTRY

On this input the data point is set which should be received by the function block. The Pin-Group of e.g. the data type IEC60870_MeasuredValue relates to the one defined by the Control-/substation in the global address list of Automation Builder defined data type (see *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*). After the declarations in Automation Builder are done, the data type is available in the global variable list in CODESYS (command list, Command(CONSTANT)). The IEC 60870 data are send or received to the data point defined by the address in Automation Builder. The sent address data point must be the same as the received data point (see *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*). As the input is a POINTER TO it has to be connected to an preceding ADR-Operator, to which any PINGROUP can be put.

THRESHOLD (threshold via ADR-Operator)

Data type: POINTER TO DWORD

When exceeding or falling below a threshold, a writing operation of a data set into the FIFOin is started. After this operation, either only the single threshold difference (between input and threshold) is deleted (TH_REFRESH=FALSE) or all differences (TH_REFRESH=TRUE). As the input is a POINTER TO, it has to be connected to an preceding ADR-Operator, to which any available input-type can be provided: BOOL, BYTE, INT, WORD, DINT, DWORD, REAL or an ARRAY of these data types. Make sure that THRESHOLD is always of the same type as in IN. E.g. if IN is an ARRAY[0..n] OF BYTE, THRESHOLD has to be an ARRAY[0..n] OF BYTE, too.

This input is only relevant for IEC_TYPE's 4(IT1), 5(IT16), 6(ME1) und 7(ME16), not for bool/digital. For the BOOL type inputs, THRESHOLD is not needed.

DATA (input via ADR-Operator)

Data type: POINTER TO DWORD

Input value as single value or as ARRAY for the possible 0 to 16 values. As the input is a POINTER TO, it has to be connected to a preceding ADR-Operator, to which any available input-type can be provided: BOOL, BYTE, INT, WORD, DINT, DWORD, REAL or an ARRAY of these data types.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SEND (send)

Data type: BOOL

At a rising edge, this input writes a data set into the FIFOin

CYCLE (cycle)

Data type: BOOL

This input writes all SCANDOWN cycles of a data set into FIFOin. At SCANDOWN=0 a data set is written in each cycle.

AUTO (auto)

Data type: BOOL

With each change of an input value or exceeding/falling down of THRESHOLD, this input writes a data set into FIFOin.

SCANDOWN (scandown)

Data type: UINT

This input will be valid if input CYCLE = TRUE. Then cyclical writing is only executed within each scandown cycle.

Example

SCANDOWN=100: In one out of 100 cycles a data set is written to FIFOin.

SCANDOWN=0: In every cycle a data set is written.

THRESHOLD_REFRESH (threshold refresh)

Data type: BOOL

FALSE: Deletion of the threshold value difference of each input value after its overwriting of the threshold level. All other differences will be ignored if they have not yet been rising above their threshold. A higher trigger rate can be expected.

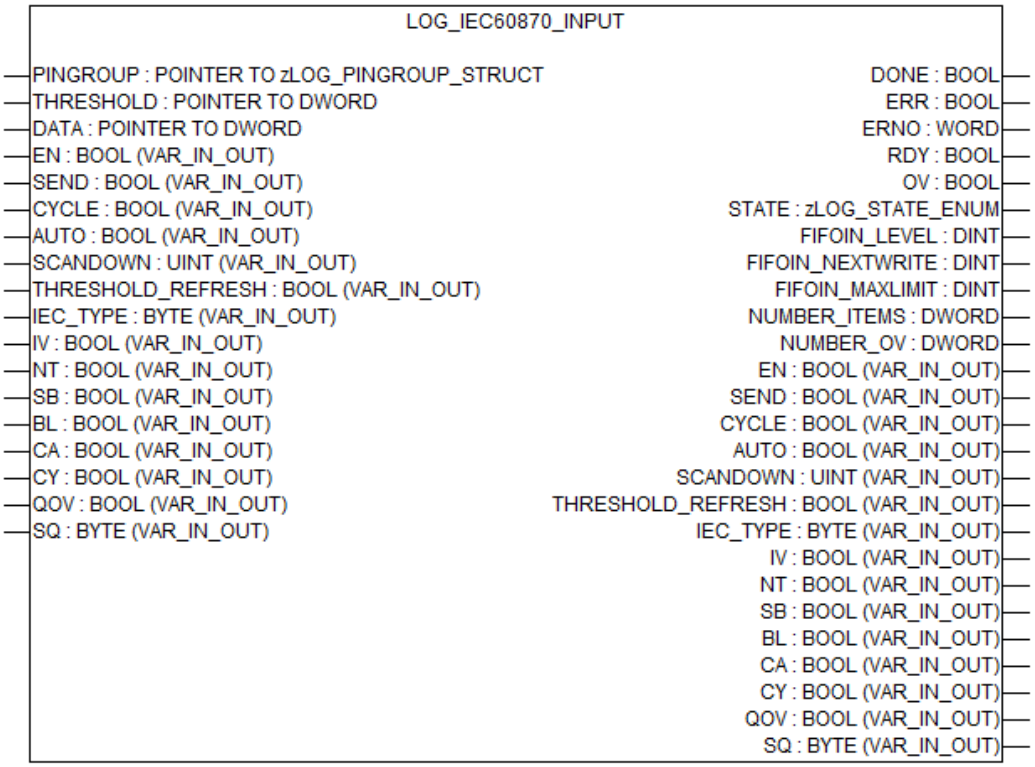
TRUE: Deletion of all threshold value differences after exceeding one of the thresholds. All other differences will also be cleared even if they have not yet been exceeded the threshold. A lower trigger rate can be expected. As always all data of the Funciton Block are sent, the data values are anyway always up to date.

This input is only relevant for IEC_TYPE's 2(SP16), 3(DP), 5(IT16) and 7(ME16), more than 1 input.

For the IEC_TYPE's 2(SP16) und 3(DP) BOOL the input TRESHOLD has no effect, as the inputs are of the type BOOL and therefore cannot have a threshold.

IEC_TYPE (IEC60870 Type)	Data type: BYTE 1=SP1, 2=SP16, 3=DP, 4=IT1, 5=IT16, 6=ME1, 7=ME16
IV (invalid)	Data type: BOOL Quality: invalid (SP1, SP16, DP, IT1, IT16, ME1, ME16)
NT (not topical)	Data type: BOOL Quality: not topical (SP1, SP16, DP, ME1, ME16)
SB (substituted)	Data type: BOOL Quality: substituted (SP1, SP16, DP, ME1, ME16)
BL (blocked)	Data type: BOOL Quality: blocked (SP1, SP16, DP, ME1, ME16)
CA (carry)	Data type: BOOL Quality: carry (IT1, IT16)
CY (counter adjusted)	Data type: BOOL Quality: counter adjusted (IT1, IT16)
QOV (overflow quality)	Data type: BOOL Quality: overflow (ME1,ME16)
SQ (sequence number)	Data type: BYTE Quality: sequence number (IT1,IT16)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

RDY (ready)

Data type: BOOL

Output will be TRUE, if a command is been received. As this is only true for one cycle, also RDY = TRUE for one cycle only.

OV (overrun)

Data type: BOOL

Output will be TRUE, if an overflow occurs. On this output a rising edge (FALSE -> TRUE) shows that the transmit requests are coming too fast and thereby the SRAM-FIFOin is fully filled.

STATE (enumeration of type zLOG_STATE_ENUM)

Data type: zLOG_STATE_ENUM

Clear text messages of errors and states. The output uses the enumeration of the data type zLOG_STATE_ENUM declared in "data types".

FIFOIN_LEVEL (FIFOin level)

Data type: DINT

Output represents the current level of SRAM-FIFOin %R area (zLOG_FIFOIN_LEVEL).

FIFOIN_NEXTWRITE (next write position in FIFOin)

Data type: DINT

Shown is the next write position in SRAM-FIFOin %R area (zLOG_FIFOIN_NEXTWRITE).

FIFOIN_MAXLIMIT (max number of data sets in FIFOin)

Data type: DINT

Shows the value of the constant FIFOLIMIT of the data sets in SRAM-FIFOin %R area (zLOG_CONST_FIFOLIMIT).

NUMBER_ITEMS (max number of items)

Data type: DINT

Shows the executed write operation into the SRAM-FIFO since EN=TRUE.

NUMBER_OV (max number of overflow-items)

Data type: DINT

Shows the not executed write operations into the SRAM-FIFO since EN=TRUE.

Wiring examples

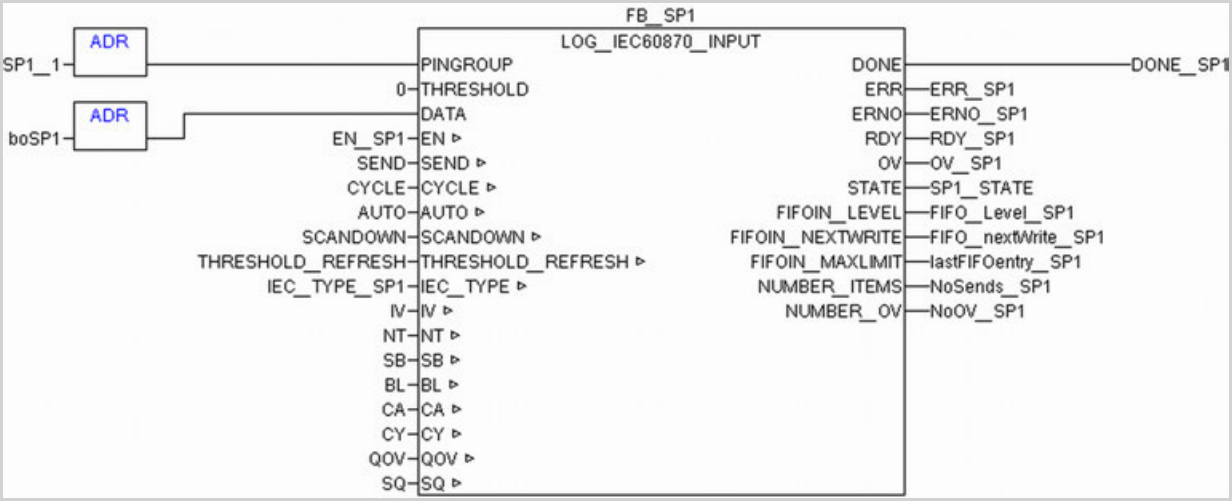


Fig. 704: Wiring example for SP1

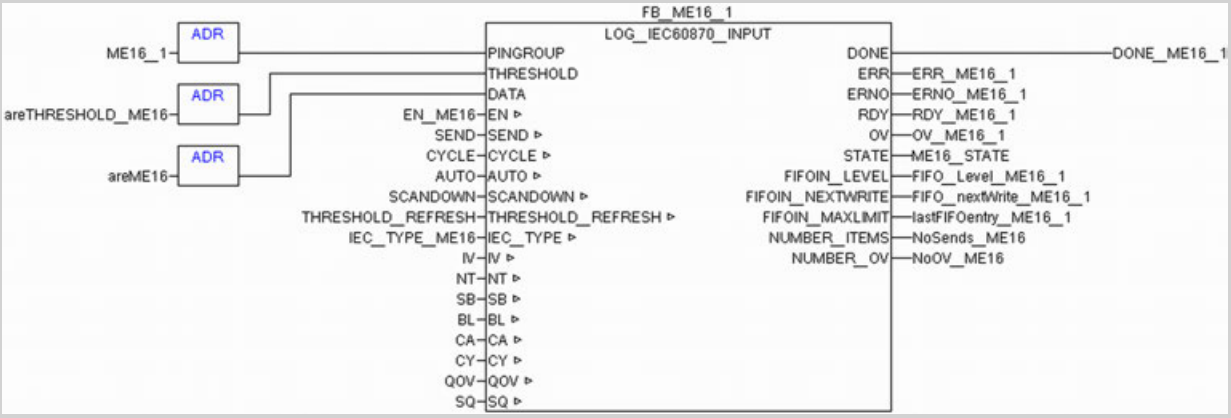


Fig. 705: Wiring example for ME16

Integrated visualization

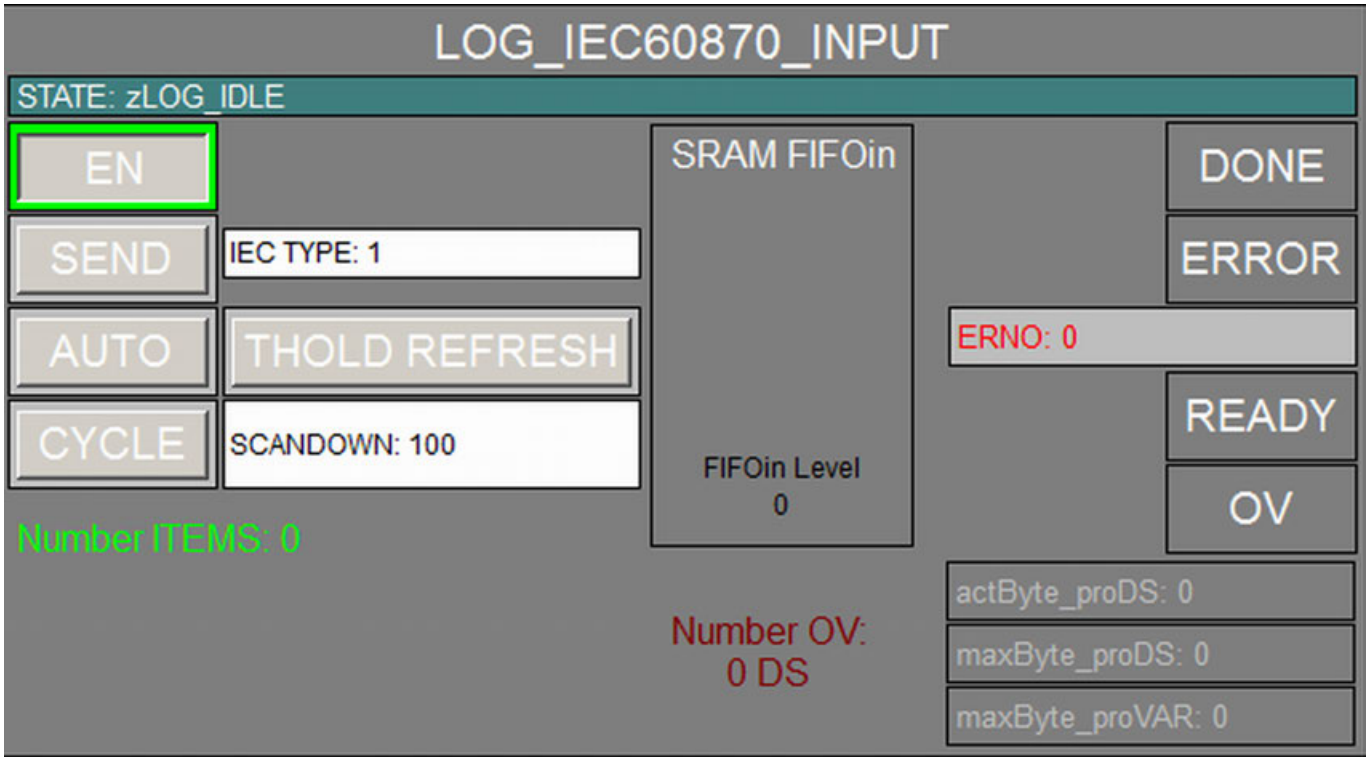


Fig. 706: Visualization

1.5.13.2.4 LOG_IEC60870_OUTPUT

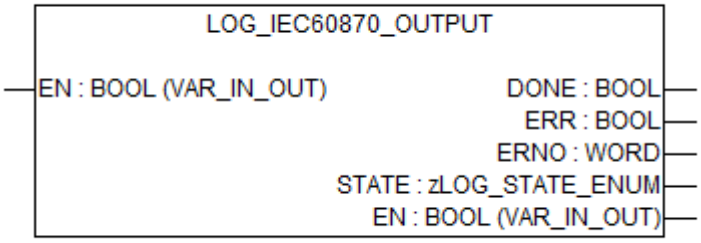


Table 251: General information

Available as of runtime system	V2.3 and above
Included in library	LogData_AC500_V23.lib
Type	Function block with historical values

This function block decodes a data set formated in *.csv and sends the current data type from the RAM-FIFOout of the Datalogger to the control station.

If enabled the function block automatically reads the oldest data set at a filling level of FIFOout > 0 and existing communication connection. Then the function block decodes the data set from *.csv format, sends the data via internal call of IEC function blocks and shows status at output STATE = zLOG_OUTPUT_DATA_HAS_BEEN_SENT. This happens until FIFOOUT_LEVEL=0 and the function block signals "EMPTY". At the same time at output STATE the message "zLOG_FIFOOUT_EMPTY" is given. While existing communication and exhausted FILE-FIFO,

the data sets, newly written by LOG_IEC60870_INPUT to the (in the %R area residing) FIFOin, are continuosly copied by LOG_HANDLING directly to the (in the %M-area residing) FIFOout, as soon as there is space in the FIFOout for new data sets. For this the FIFOout should not be empty (EMPTY). It is different when FILE-FIFO is not empty. In this case at first FIFOout is fully decremented until FIFOOUT_LEVEL=0 and the function block signals "EMPTY". After that a new data set block is copied from FILE-FIFO into FIFOout. The capacity of the RAM-FIFOout per segment of the %M-area is at max ARRAY[0..160] OF ARRAY[0..399] OF BYTE. The ARRAY ocupies the complete segment 1 in the %M-area of the CPU.

A complete segment (segment 1 of the %R-area) is used for the SRAM-FIFOin (zLOG_CONST_FIFOLIMIT =160).

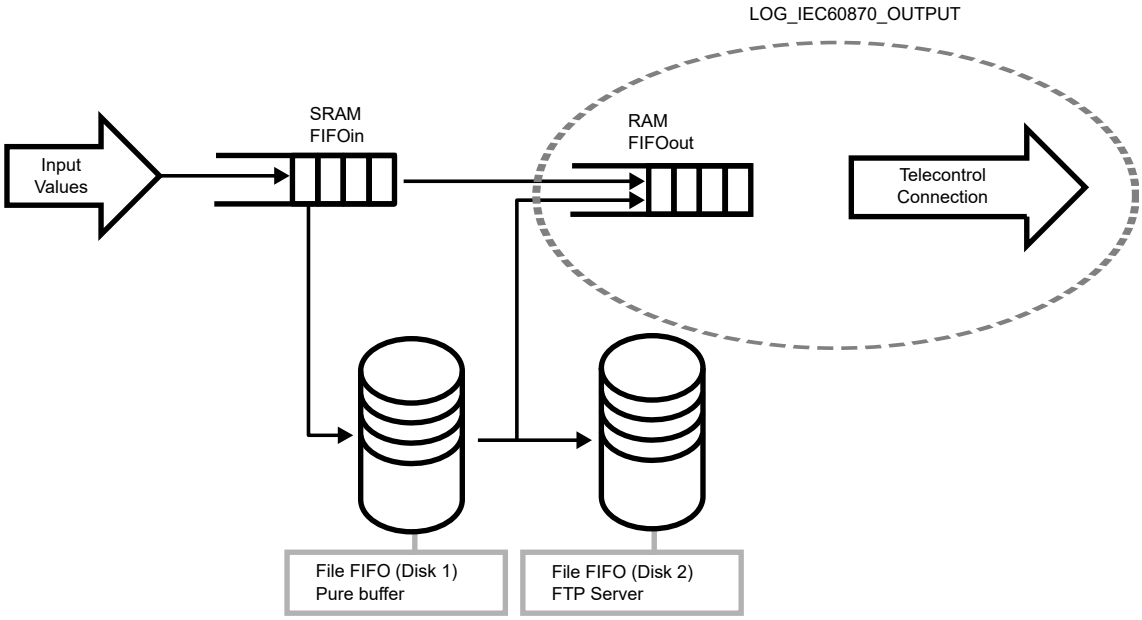
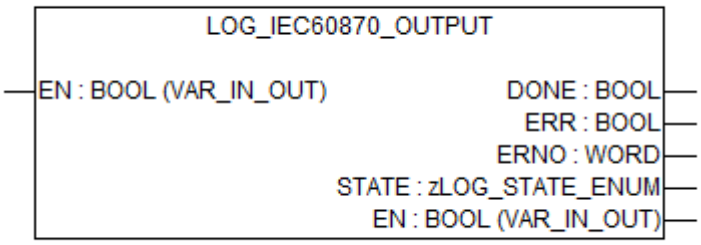


Fig. 707: Functionality

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

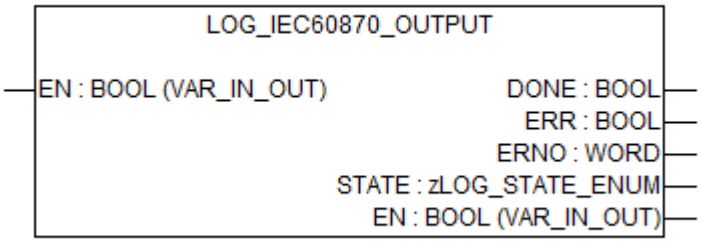
Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

Output description



DONE (done)

Data type: BOOL

On this output the status of the function block execution is shown. DONE = TRUE if a new data set is given to the remote control list of the CPU. By this the sending of the data set is considered.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

STATE (enumeration of type zLOG_STATE_ENUM)

Data type: zLOG_STATE_ENUM

Clear text messages of errors and states. The output uses the enumeration of the data type zLOG_STATE_ENUM declared in "data types".

Integrated visualization

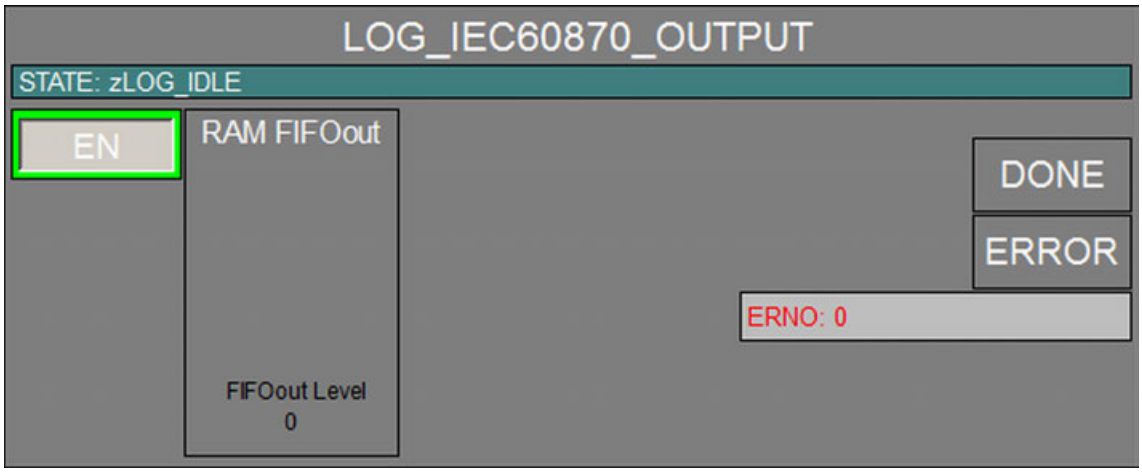


Fig. 708: Visualization

1.5.13.2.5 LOG_GENERIC_INPUT

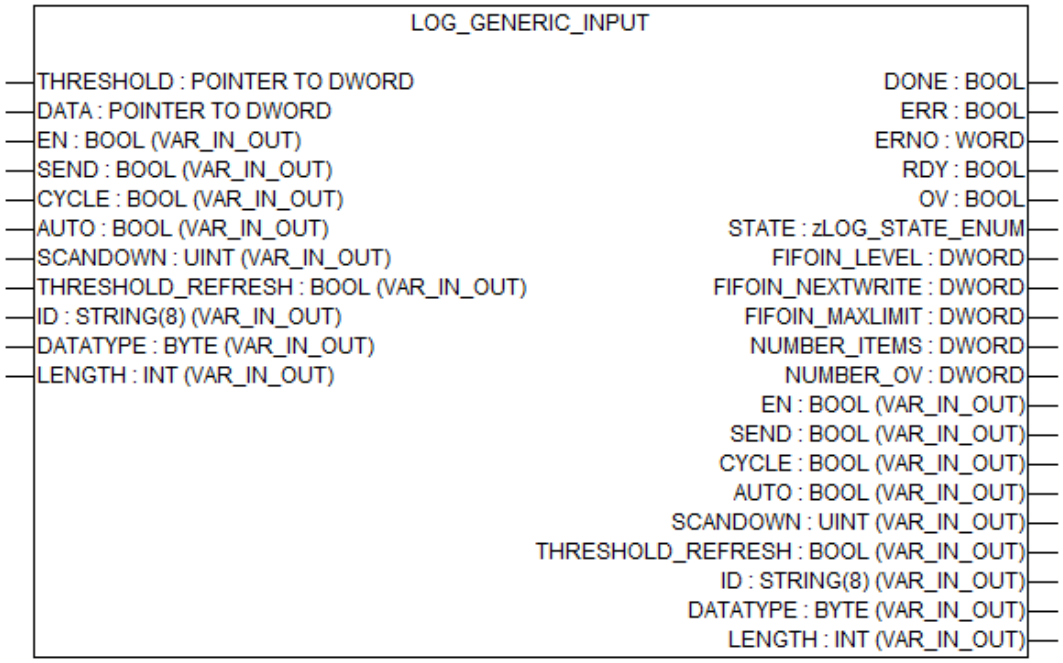


Table 252: General information

Available as of runtime system	V2.3 and above
Included in library	LogData_AC500_V23.lib
Type	Function block with historical values

At the same time, this function block writes data sets of the data types 1=BOOL, 2=BYTE, 3=INT, 4=WORD, 5=DINT, 6=DWORD, 7=REAL with a max length of 32 values into the SRAM-FIFOin of the Datalogger.

The data sets are written in format *.csv as two-dimensional ARRAY. Every part of each data set is separated by SEMICOLON. Each data set is ended with a CR/LF. Every segment of the %R area of the SRAM-FIFOin contains maximum one ARRAY[0..160] OF ARRAY[0..399] OF BYTE. The ARRAY reserves the complete segment 1 of the %R area of the controler.

A complete segment (segment 1 of the %R-area) is used for the SRAM-FIFOin (zLOG_CONST_FIFOLIMIT =160).

The function block writes data sets in the SRAM-FIFOin of the controller and doesn't send them directly to a central control station. This is done by the user. With the function block LOG_GENERIC_OUTPUT the data to be sent are provided as an interface.

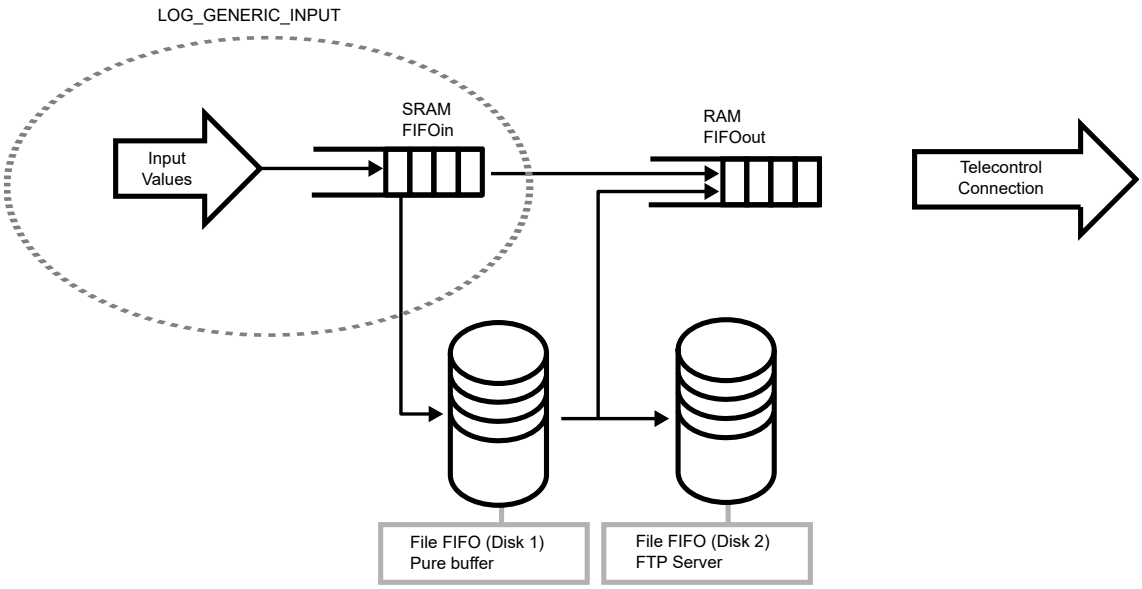
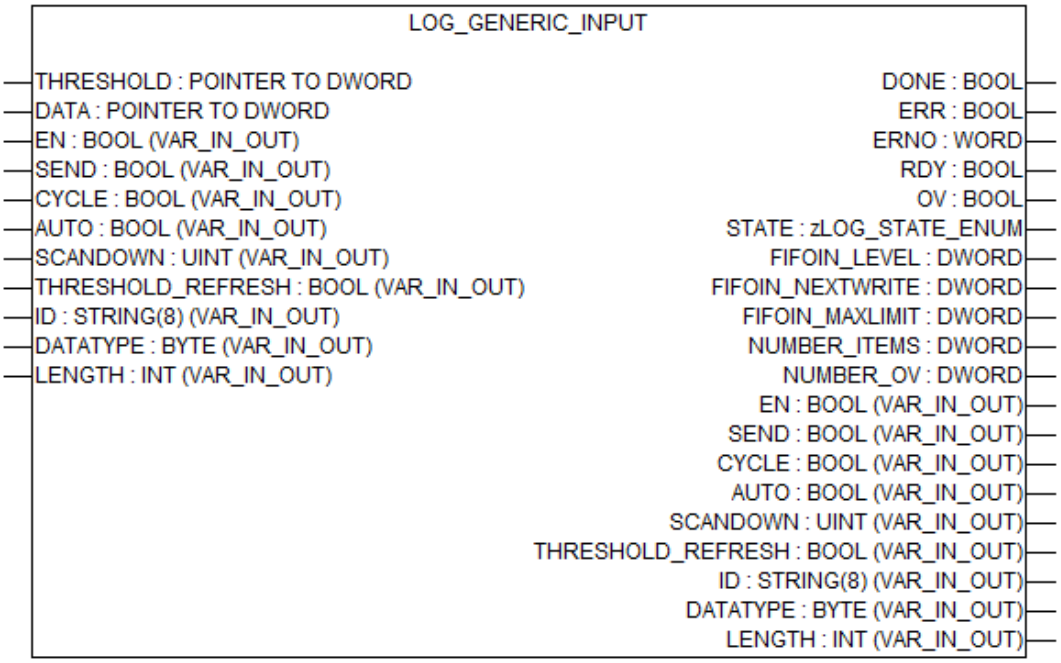


Fig. 709: Functionality

CAUTION!
Each used instance of the function block needs only one cycle for writing the data into FIFOin. For "n" succeeding instances called in the same cycle, "n" entrances/data sets are entered into FIFOin. Consider that FIFOin is limited by the limit of the segments of the %R area. That means, the upper limit of entries to FIFOin at the same time (in the same cycle) is 160.

Input Description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

THRESHOLD (threshold via ADR-Operator)

Data type: POINTER TO DWORD

When exceeding or falling below a threshold, a writing operation of a data set into the FIFOin is started. After this operation, either only the single threshold difference (between input and threshold) is deleted (TH_REFRESH=FALSE) or all differences (TH_REFRESH=TRUE). As the input is a POINTER TO, it has to be connected to an preceding ADR-Operator, to which any available input-type can be provided: BOOL, BYTE, INT, WORD, DINT, DWORD, REAL or an ARRAY of these data types. Make sure that THRESHOLD is always of the same type as in IN. E.g. if IN is an ARRAY[0..n] OF BYTE, THRESHOLD has to be an ARRAY[0..n] OF BYTE, too.

The input is only relevant for IN_TYPE's 2=BYTE, 3=INT, 4=WORD, 5=DINT, 6=DWORD, 7=REAL, not for bool/digital. For the IN_TYPE 1=BOOL, TRESHOLD has no effect, as the inputs are of the type BOOL and therefore cannot have a threshold.

DATA (input via ADR-Operator)

Data type: POINTER TO DWORD

Input value as single value or as ARRAY for the possible 0 to 31 values. As the input is a POINTER TO, it has to be connected to a preceding ADR-Operator, to which any available input-type can be provided: BOOL, BYTE, INT, WORD, DINT, DWORD, REAL or an ARRAY of these data types.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SEND (send)

Data type: BOOL

At a rising edge, this input writes a data set into the FIFOin

CYCLE (cycle)

Data type: BOOL

This input writes all SCANDOWN cycles of a data set into FIFOin. At SCANDOWN=0 a data set is written in each cycle.

AUTO (auto)

Data type: BOOL

With each change of an input value or exceeding/falling down of THRESHOLD, this input writes a data set into FIFOin.

SCANDOWN (scandown)

Data type: UINT

This input will be valid if input CYCLE = TRUE. Then cyclical writing is only executed within each scandown cycle.

Example	SCANDOWN=100: In one out of 100 cycles a data set is written to FIFOin. SCANDOWN=0: In every cycle a data set is written.

THRESHOLD_REFRESH (threshold refresh)

Data type: BOOL

FALSE: Deletion of the threshold value difference of each input value after its overwriting of the threshold level. All other differences will be ignored if they have not yet been rising above their threshold. A higher trigger rate can be expected.

TRUE: Deletion of all threshold value differences after exceeding one of the thresholds. All other differences will also be cleared even if they have not yet been exceeded the threshold. A lower trigger rate can be expected. As always all data of the Funciton Block are sent, the data values are anyway always up to date.

The input is only relevant for IN_TYPE’s 2=BYTE, 3=INT, 4=WORD, 5=DINT, 6=DWORD, 7=REAL, not for bool/digital.

For the IN_TYPE 1=BOOL the input TRESHOLD has no effect, as the inputs are of the type BOOL and therefore cannot have a threshold.

ID (identity)

Data type: STRING

Free selectable name/text for identification of this data set. Maximum 8 characters.

DATATYPE (Datatype)

Data type: BYTE

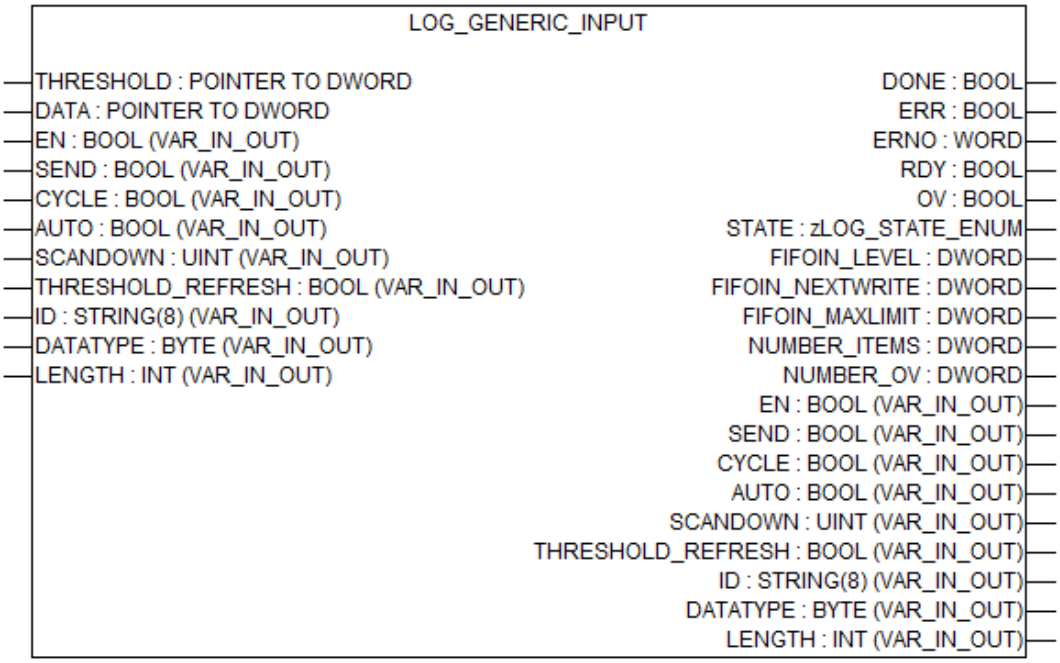
1=BOOL, 2=BYTE, 3=INT, 4=WORD, 5=DINT, 6=DWORD, 7=REAL

LENGTH (length of valid values)

Data type: INT

Number of valid values (maximum 88 values, depending of type).

Output Description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

Monitoring of the global constant diconFIFOmaxLimit defined in the the segment 0 of the %R area. Error in case of diconFIFOmaxLimit < 1 or diconFIFOmaxLimit > 1120.

RDY (ready)

Data type: BOOL

Output will be TRUE, if a command is been received. As this is only true for one cycle, also RDY = TRUE for one cycle only.

OV (overrun)

Data type: BOOL

Output will be TRUE, if an overflow occurs. On this output a rising edge (FALSE -> TRUE) shows that the transmit requests are coming too fast and thereby the SRAM-FIFOin is fully filled.

STATE (enumeration of type zLOG_STATE_ENUM)

Data type: zLOG_STATE_ENUM

Clear text messages of errors and states. The output uses the enumeration of the data type zLOG_STATE_ENUM declared in "data types".

FIFOIN_LEVEL (FIFOin level)

Data type: DINT

Output represents the current level of SRAM-FIFOin %R area (zLOG_FIFOIN_LEVEL).

FIFOIN_NEXTWRITE (next write position in FIFOin)

Data type: DINT

Shown is the next write position in SRAM-FIFOin %R area (zLOG_FIFOIN_NEXTWRITE).

FIFOIN_MAXLIMIT (max number of data sets in FIFOin)

Data type: DINT

Shows the value of the constant FIFOLIMIT of the data sets in SRAM-FIFOin %R area (zLOG_CONST_FIFOLIMIT).

NUMBER_ITEMS (max number of items)

Data type: DINT

Shows the executed write operation into the SRAM-FIFO since EN=TRUE.

NUMBER_OV (max number of overflow-items)

Data type: DINT

Shows the not executed write operations into the SRAM-FIFO since EN=TRUE.

Wiring examples

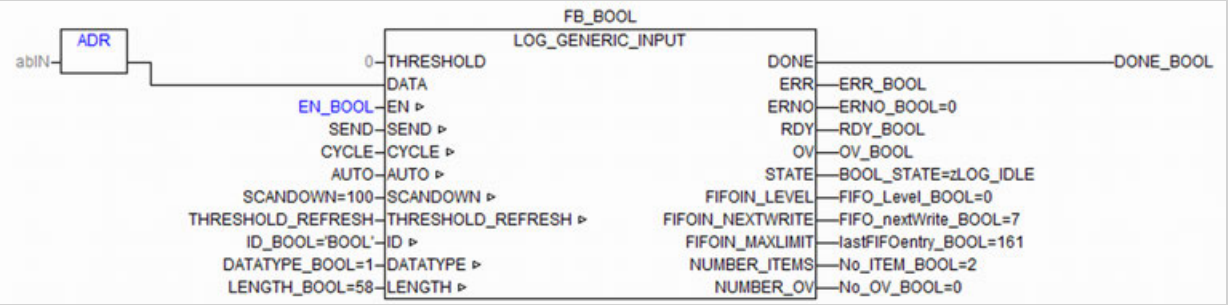


Fig. 710: Wiring example for BOOL

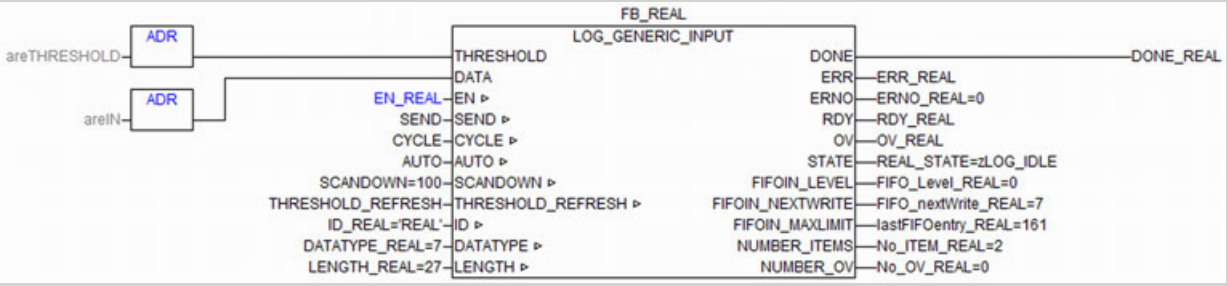


Fig. 711: Wiring example for REAL

Integrated Visualization

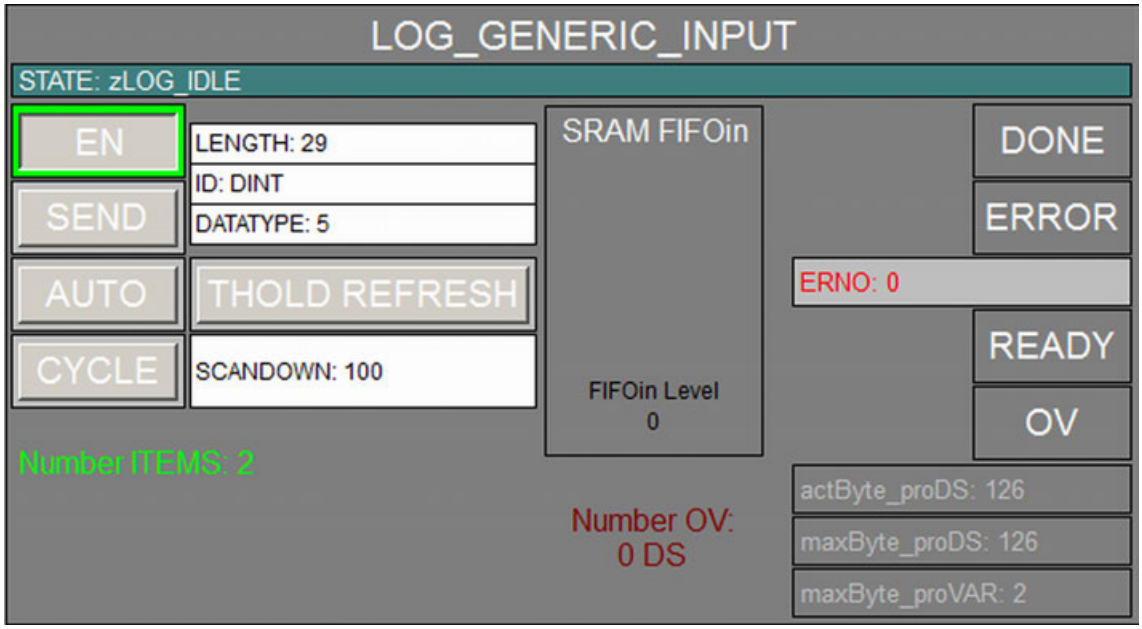


Fig. 712: Visualization

1.5.13.2.6 LOG_GENERIC_OUTPUT

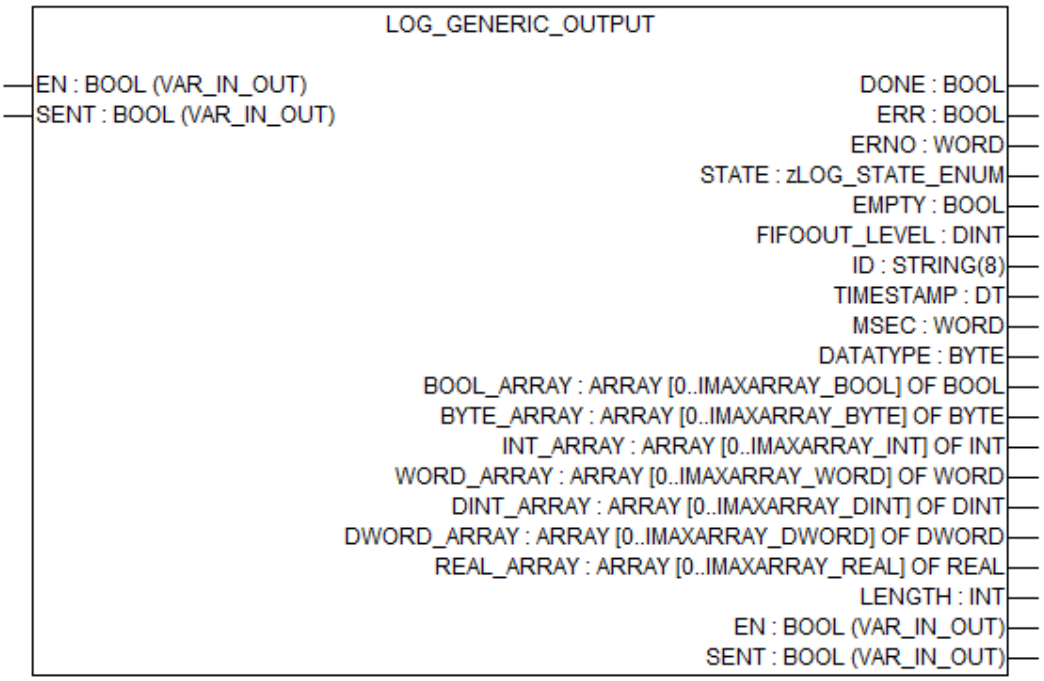


Table 253: General information

Available as of runtime system	V2.3 and above
Included in library	LogData_AC500_V23.lib
Type	Function block with historical values

This function block provides a decoded data set of the corresponding data types from the SRAM-FIFOout of the Datalogger. The function block does not cover the transmission of the data set. But it provides an interface for the user on which the data can be read and send according to application specific requirements and protocols.

The function block automatically reads the oldest data set at a filling level of FIFOout > 0. Then the function block decodes the data set from *.csv format and provids it to the outputs. The function block signals its availability through DONE = TRUE and STATE = zLOG_OUTPUT_DATA_AVAILABLE. If this data set is processed by the user, the user will have to signal this by SENT = TRUE to the function block. By the edge 0/1 the RAM-FIFOout is decremented by the value 1, DONE changes to "FALSE" and on the output STATE the text "zLOG_OUTPUT_DATA_HAS_BEEN_SENT" appears. By a 1/0 edge on SENT the function block reads the next data set (which is the same as the self triggered read without command, as long as SENT = false). This happens until FIFOOUT_LEVEL=0" and the function block signals "EMPTY". At the same time at output STATE the message "zLOG_FIFOOUT_EMPTY" is given and all output values are deleted. While existing communication and exhausted FILE-FIFO the data sets, newly written by LOG_GENERIC_INPUT to the (in the %R area residing) FIFOin are continuously copied by the LOG_HANDLING directly to the (in the %M-area residing) FIFOout, as soon as there is space in the FIFOout for new data sets. For this the FIFOout should not be empty (EMPTY). It is different when FILE-FIFO is not empty. In this case at first FIFOout is fully decremented until FIFOOUT_LEVEL=0 and the function block signals "EMPTY". After that a new data set block is copied from FILE-FIFO into FIFOout. The capacity of the RAM-FIFOout per segment of the %M-area is at max ARRAY[0..160] OF ARRAY[0..399] OF BYTE. The ARRAY ocupies the complete segment 1 in the %M area of the CPU.

A complete segment (segment 1 of the %R-area) is used for the SRAM-FIFOin (zLOG_CONST_FIFOLIMIT =160).

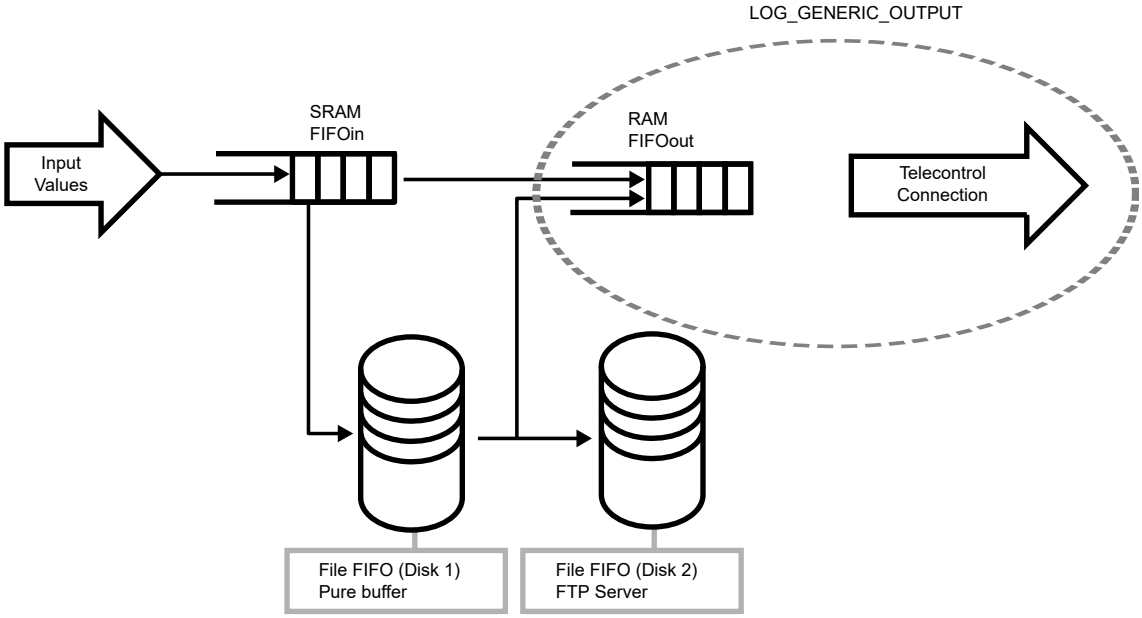
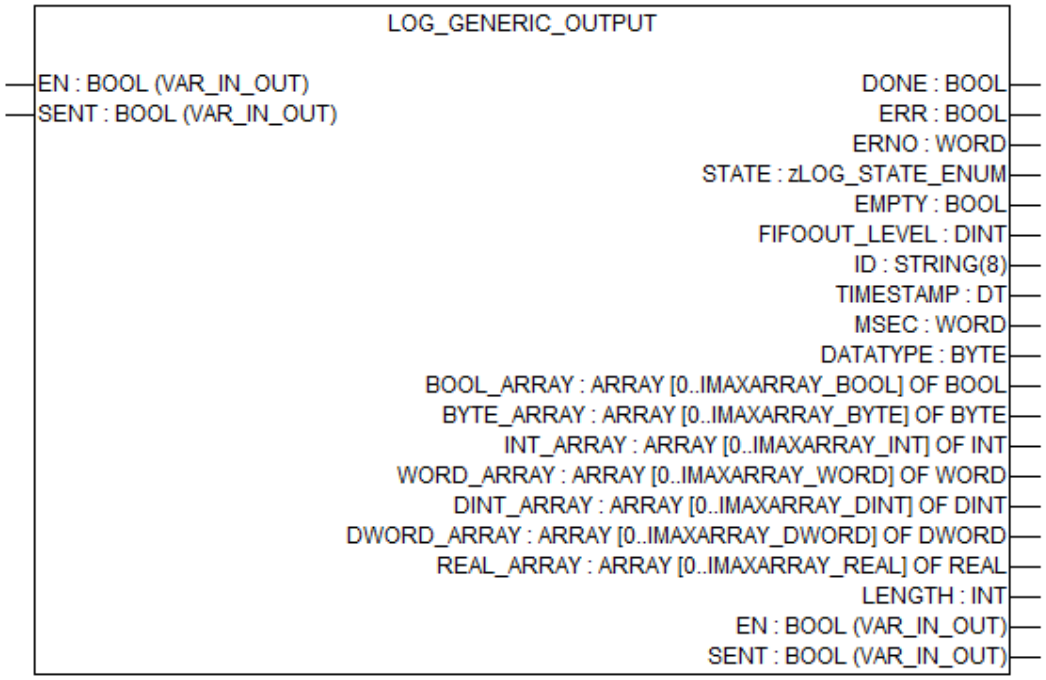


Fig. 713: Functionality

Input description



The inputs marked with a triangle ▷ are of the class VAR_IN_OUT (input and output variable). These inputs must be connected to a variable.

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

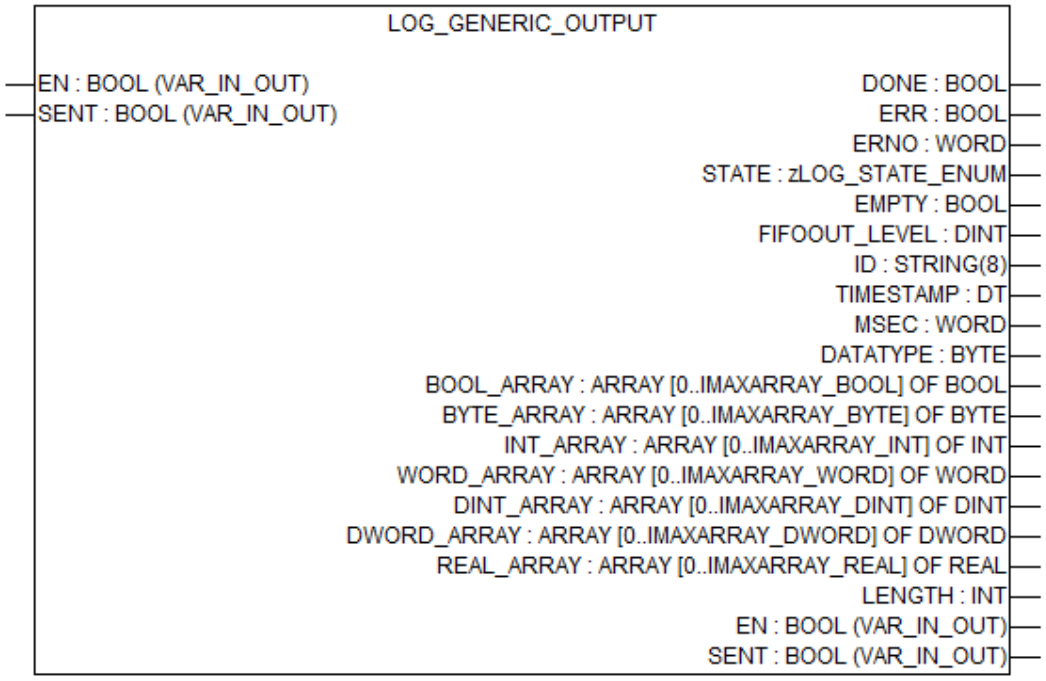
While it is executed its inputs are continuously evaluated.

SENT (output data were sent)

Data type: BOOL

Feedback from the user program that the currently provided data set was sent. By the 0/1 edge the RAM-FIFOout is decremented by a value of 1 (FIFOOUT_LEVEL -1). DONE changes to "FALSE" and on the output STATE the text "zLOG_OUTPUT_DATA_HAS_BEEN_SENT" appears. By the 1/0 edge on SENT the function block reads the next data set (it is the same as the automatic reading without a command, as long as SENT = FALSE).

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

STATE (enumeration of type zLOG_STATE_ENUM)

Data type: zLOG_STATE_ENUM

Clear text messages of errors and states. The output uses the enumeration of the data type zLOG_STATE_ENUM declared in "data types".

EMPTY (empty)

Data type: BOOL

FIFOout "EMPTY" is shown.

FIFOOUT_LEVEL (level FIFOout)

Data type: DINT

The current level of RAM-FIFOout %M area (zLOG_FIFOOUT_LEVEL) is displayed.

ID (identity)

Data type: STRING(8)

Output of the ID or name of the data set as string chain (no special characters), maximum 8 characters.

TIMESTAMP (timestamp)

Data type: DT

Output of the time stamp of the data set in the format year-month-day-hour-minute-second.

MSEC (milliseconds of timestamp)

Data type: WORD

Output of the milliseconds of the time stamp of the data set.

DATATYPE (data type of data set)

Data type: BYTE

Output of the data set data type: 1=BOOL, 2=BYTE, 3=INT, 4=WORD, 5=DINT, 6=DWORD, 7=REAL.

BOOL_ARRAY (array 0..57 of bool)

Data type: ARRAY[0..57] OF BOOL

Output of the data set as ARRAY OF BOOL (max 58 values).

BYTE_ARRAY (array 0..87 of byte)
Data type: ARRAY[0..87] OF BYTE
Output of the data set as ARRAY OF BYTE (max 88 values).

INT_ARRAY (array 0..49 of int)
Data type: ARRAY[0..49] OF INT
Output of the data set as ARRAY OF INT (max 48 values).

WORD_ARRAY (array 0..57 of word)
Data type: ARRAY[0..57] OF WORD
Output of the data set as ARRAY OF WORD (max 58 values).

DINT_ARRAY (array 0..28 of dint)
Data type: ARRAY[0..28] OF DINT
Output of the data set as ARRAY OF DINT (max 29 values).

DWORD_ARRAY (array 0..30 of dword)
Data type: ARRAY[0..30] OF DWORD
Output of the data set as ARRAY OF DWORD (max 31 values).

REAL_ARRAY (array 0..26 of real)
Data type: ARRAY[0..26] OF REAL
Output of the data set as ARRAY OF REAL (maximum 27 values).

LENGTH (length of valid values)
Data type: INT
Number of valid values (maximum 88 values, depending of type).

Integrated visualization

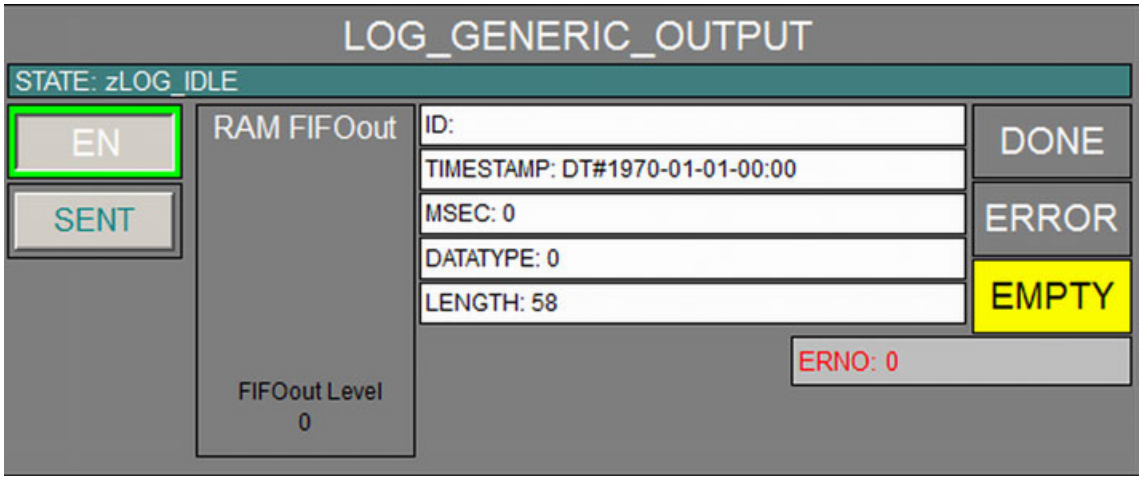


Fig. 714: Visualization

1.5.13.2.7 Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples

1.5.14 Pumping library 2

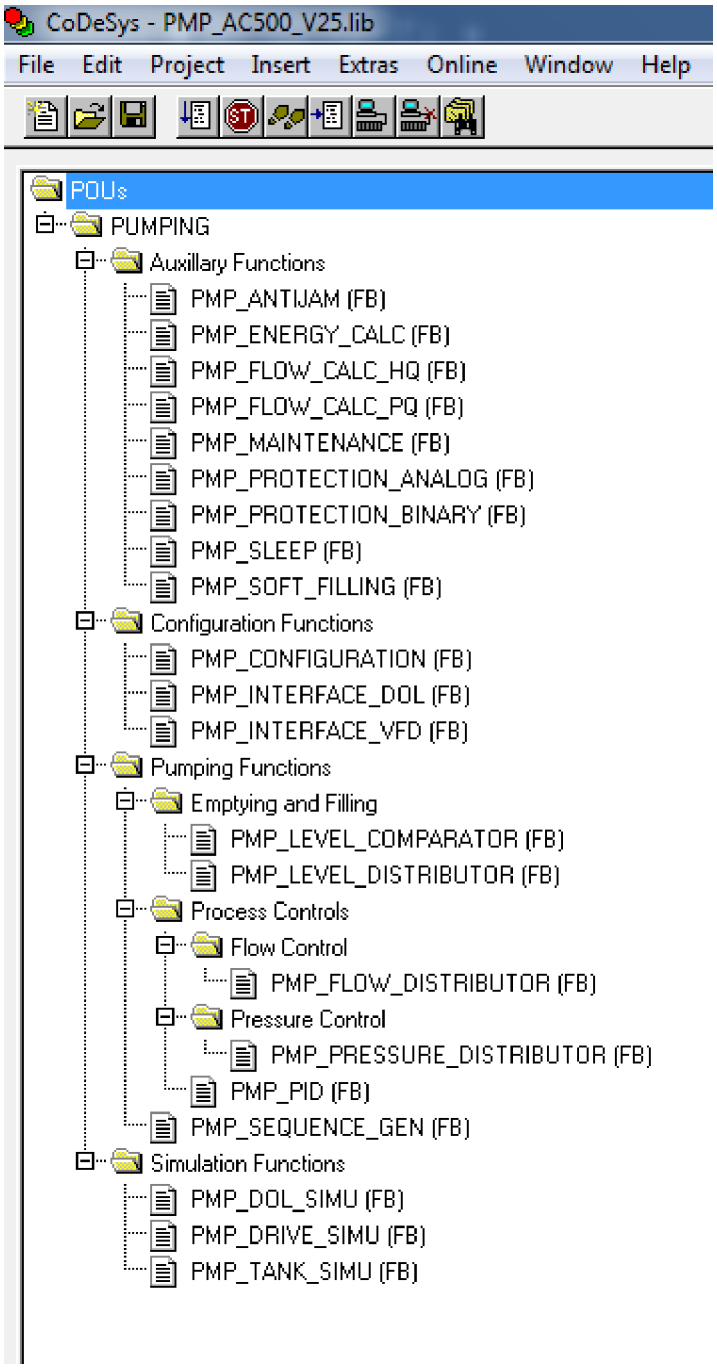
1.5.14.1 System technology

1.5.14.1.1 Components of pumping library

Overview

> *The pumping library, AC500_PMP_V25.lib is the extension and improvement in the library AC500_PUMP_V23.lib.*

The pumping library functions are categorized as follows. Refer to the figure below from the library itself.



The library contains function blocks needed for functions of pumping application (pressure control, flow control or level control).

It also contains function blocks for some advanced functions needed for different applications.

Compatibility

The library is compatible with following versions.

Type	Version
Pumping Library	**PMP_AC500_V25.lib**
Automation Builder	V2.0 and later

Required sensors

A minimal sensor equipment is required for the following measurements:

- Actual pressure measurement for pressure control.
- Actual flow measurement for flow control.
- Actual level measurement for level control.

The pressure sensor can be at the input and/or at the output. The flow sensor should be only at the output.

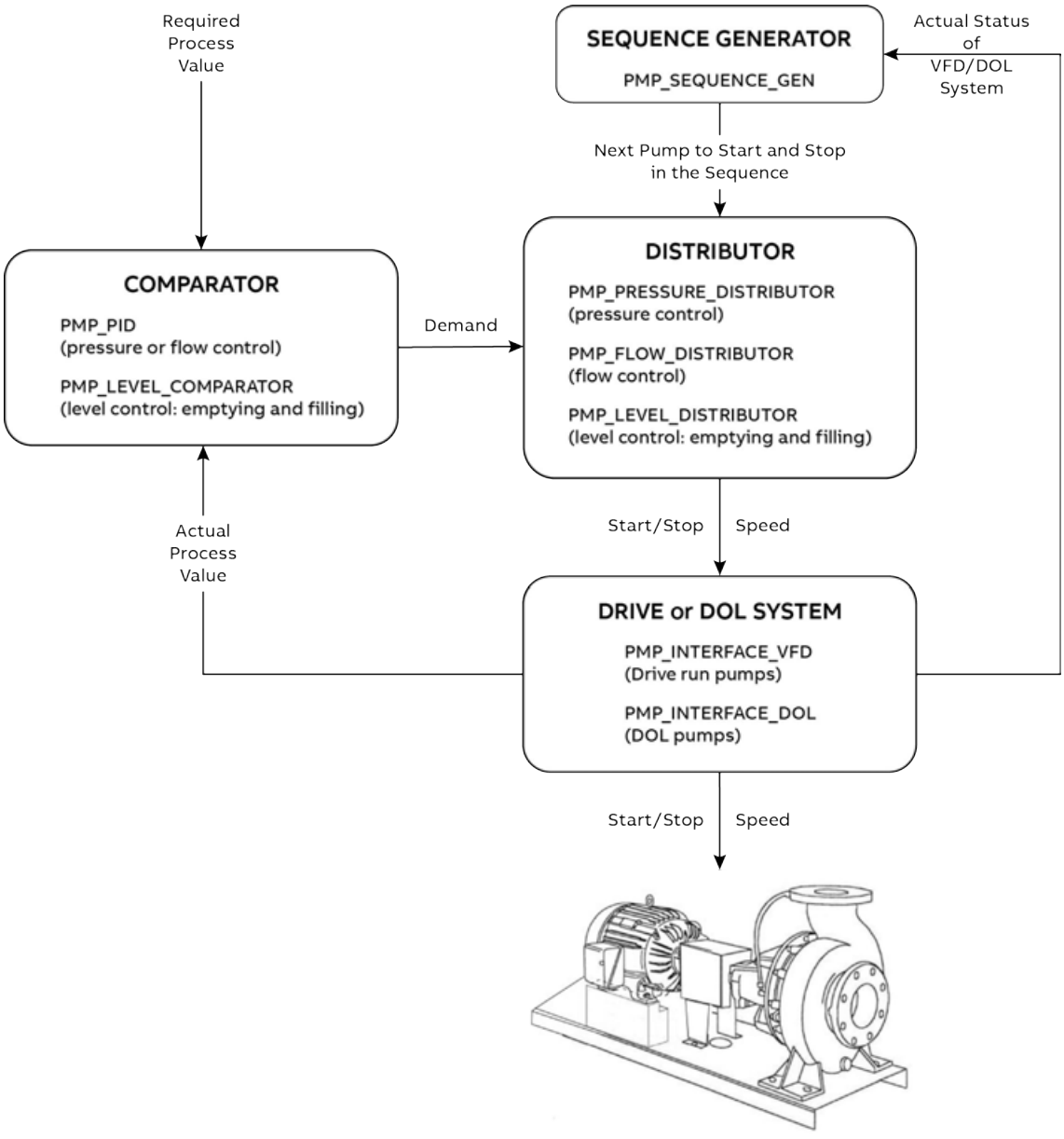
Often, at the input a sensor, typically pressure, is placed for advanced protection and diagnosis.

1.5.14.1.2 Control philosophy of pumping library

The Pumping library is used for controlling the following three water pumping processes:

- Pressure control
- Flow control
- Level control – Emptying or filling

The figure below gives an overview of the pumping library and explains how the library processes the data from the user and then controls the processes.



To run any process, the pumping library follows the four stages described below:

Stage 1: Comparator

The set process value (e.g., pressure in psi, flow rate in m^3/h) is compared with the measured actual value. The distributor uses the output of the comparator to decide the number of pumps and their operating speeds to meet the demand.

- **Pressure control and Flow control:**
  In pressure and flow controls, the function block PMP_PID acts as a comparator which gives the PID_OUT in [%] to the function block PMP_PRESSURE_DISTRIBUTOR or PMP_FLOW_DISTRIBUTOR. In the distributor the PID output is then converted as reference speed set point to the pumps.
- **Emptying or Filling mode:**
  In emptying or filling modes, the function block PMP_LEVEL_COMPARATOR is used to compare the actual level of the tank with the set levels to start or stop the pumps. This information is sent to the distributor by the output N_DEMAND. For example, in filling mode, at lower actual level more pumps are needed to run to fill the tank.

Stage 2: Sequence generator

The PMP_SEQUENCE_GEN function block is used. The following functions are included:

- Decides which pump is ready to run in the auto mode.
- Decides which pump to start in the sequence, based on the least actual runtime hours.
  Exception – when the pump station is traditional type (with master pump on VFD, rest on DOL), then the pump ID = 1 which is attached to the VFD will always run as master and is first to start and last to stop.
- Decides which pump to stop in the sequence, based on the highest actual runtime hours.
  Exception – when the pump station is traditional type (with master pump on VFD, rest on DOL), then the pump ID = 1 which is attached to the VFD will always run as master and is first to start and last to stop.
- Assigns a master status to the pump in the sequence.
- Indicates how many pumps are ready for automation and how many are running.

The figure below explains a use case where the demand increases and then decreases. The sequence is generated based on this criteria by the function block PMP_SEQUENCE_GEN.

DEMAND	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1
Master	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Next Pump to Start	2	3	4	5	6	7	8		2	Whichever pump has minimum runtime will start next					
Next Pump to Stop	1	2	2	2	2	2	2	2	3	4	5	6	7	8	1
Pumps running	1	1, 2	1, 2, 3	1, 2, 3, 4	1, 2, 3, 4, 5	1, 2, 3, 4, 5, 6	1, 2, 3, 4, 5, 6, 7	1, 2, 3, 4, 5, 6, 7, 8	1, 3, 4, 5, 6, 7, 8	1, 4, 5, 6, 7, 8	1, 5, 6, 7, 8	1, 6, 7, 8	1, 7, 8	1, 8	1

Stage 3: Distributor

The distributor function block distributes the start/ stop command and speed reference to all VFD operated pumps.

- • Pressure control: For pressure control, the function block PMP_PRESSURE_DISTRIBUTOR is used. The PID output coming from the function block PMP_PID (comparator) is converted as reference speed setpoint to the master pump. The function works based on the reference speed of the master pump. If the reference speed is higher, more number of pumps are needed to meet the demand. If the reference speed is lower, the already running pumps in the network need to be stopped. The function block:
  - Calculates the speed reference of the master pump and then starts/ stops the follower pumps based on the settings in the inputs START_SPEED_FWR or STOP_SPEED_FWR.
  - Distributes the speed references to VFD follower pumps based on the settings in the input FOLLOWER_MODE.
- Flow control: For flow control, the function block PMP_FLOW_DISTRIBUTOR is used.
  - In the flow distributor the number of pumps to run is decided by the following ratio: FLOW_SETPOINT/ (NUMBER OF PUMPS IN STATION * NOMINAL FLOW OF PUMP).
  - The function block distributes the speed references to VFD follower pumps based on the settings in the input FOLLOWER_MODE.
- Emptying/Filling mode: For emptying or filling operation, the function block PMP_LEVEL_DISTRIBUTOR is used. The function block starts/ stops the pumps based on the input N_DEMAND (demand) which it receives from function block PMP_LEVEL_COMPARATOR.
  - If the actual level is in the normal range, then the speed reference of the pumps are based on the input NORMAL_SPEED.
  - If for example while filling the actual level reaches below the LOW_LEVEL, then all pumps run at HIGH_SPEED, to quickly fill the pump. Similar analogy is followed when the pump is in emptying mode.

Stage 4: DRIVE or DOL system

The output of the distributors (AUTO_START_CMD and AUTO_SPEED_REF) are connected to function block PMP_INTERFACE_VFD and PMP_INTERFACE_DOL. The function blocks establish the link between library and field devices i.e. pumps.

- For pumps driven by VFD motors, the function block PMP_INTERFACE_VFD can be connected with the drives communication function blocks to establish communication and control with the VFD.
- For pumps driven by DOL motor, the function block PMP_INTERFACE_DOL can be used for start/stop controls.

For more information, see the detailed controlled philosophy in the following sections.

1.5.14.1.3 Application functions

There are three main application and control function blocks:

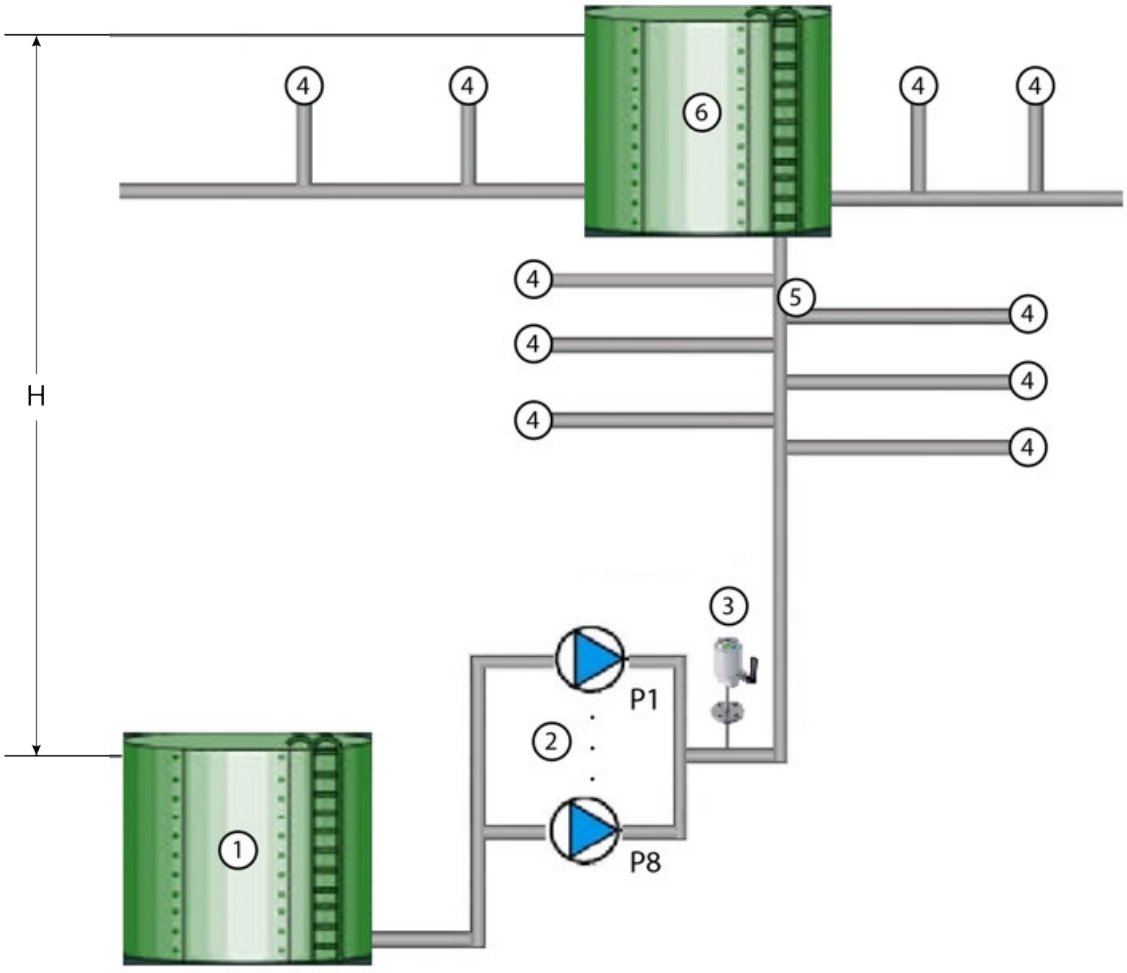
- Pressure control function blocks
- Flow control function blocks
- Level-control function block

Pressure control function block

Pressure control process flow diagram

Pressure control is used in applications with individual or multiple water consumers.

A typical pressure control application diagram is shown below.



1 Suction tank
2 Parallel operating pumps P1 ... P8
3 Pressure setpoint, discharge pressure
4 Consumer
5 Water distribution network
6 Water tower buffer tank
H Water head between the levels in the suction tank and water tower buffer tank

Using pressure control

Pressure control in the pumping library can help in the following operations:

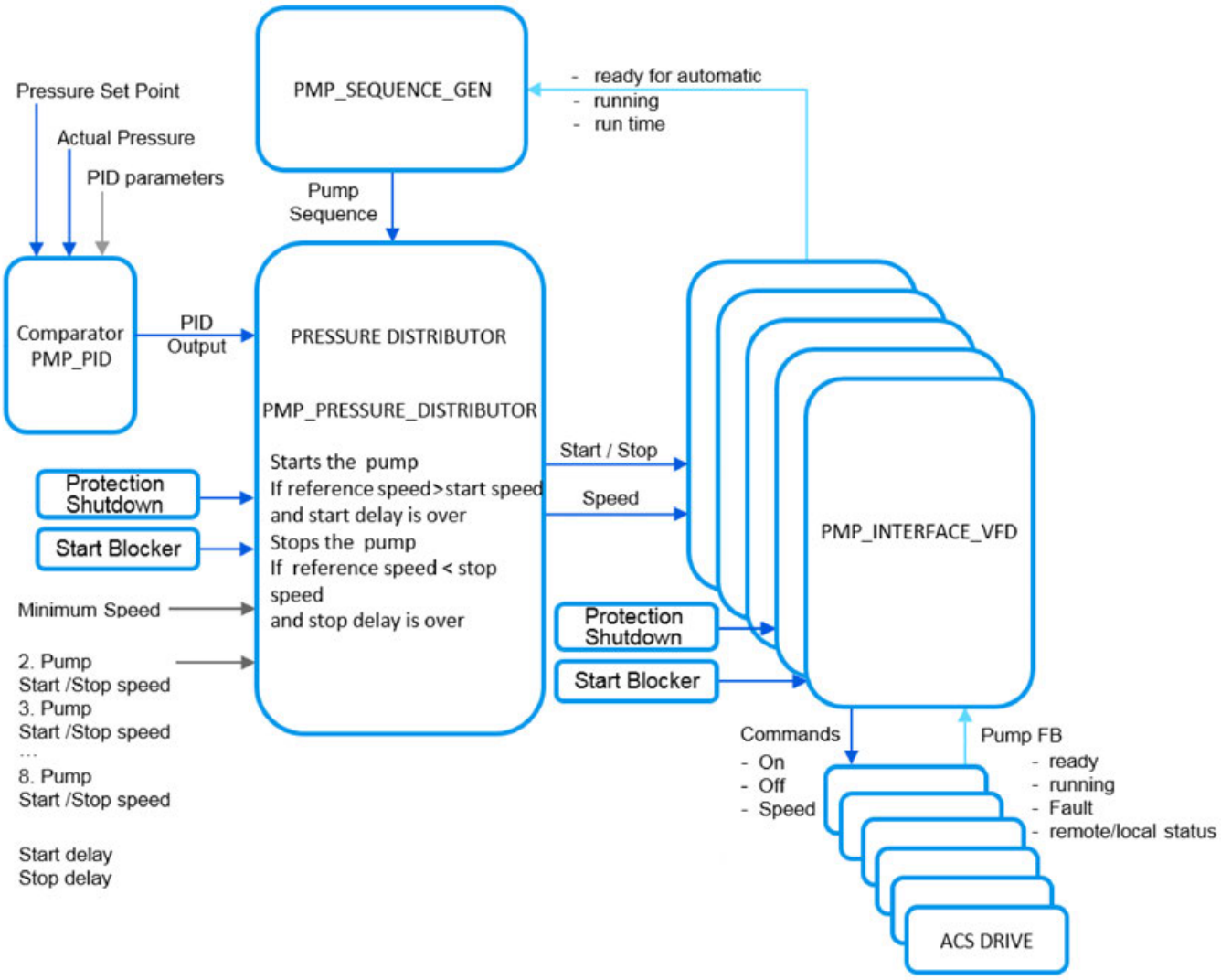
- Water consumption fluctuates always and may be discontinuous over a time period, because of which it is difficult to have a continuous flow rate like in the flow control process.
- The number of pumps and the pump speed is decided by the water consumption in the water distribution system.
- Water is supplied to the pump station through suction pipeline from a suction tank.
- The pumps may be operating in parallel.
- The discharge pressure is measured in the discharge pipeline. The measured discharge pressure is controlled according to the pressure set point.
- Each pump must be started with a minimum speed to build up the pressure which is required to produce a minimum flow.
- The pressure to deliver a minimum flow depends of the water head (H) between the water levels in the suction tank and target tank.
- If there is pump station shutdown due to protection shutdown input, the pumps must be stopped sequentially to prevent water hammering.
- Variable speed drives should be stopped through a speed ramp. Otherwise the water hammering will quickly wear out the pipeline connections and the pump station equipment.

Pump combinations for pressure control

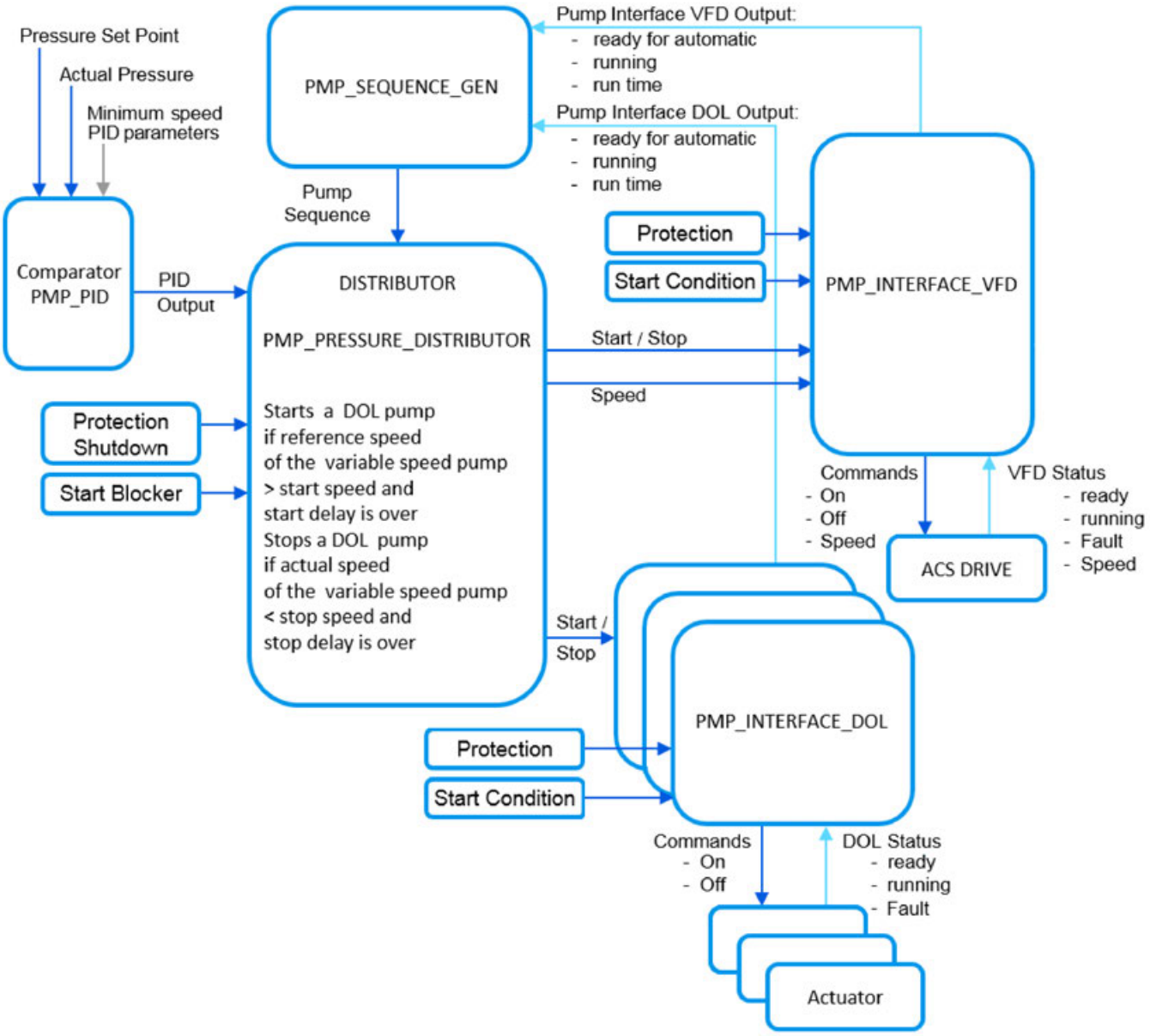
Pressure control mode works for two types of pump combinations: Multi pump and traditional pump.

The combination of function blocks to execute a pressure control is shown in the figures below.

- Multi pump – In this mode all pumps are run using VFDs.



- Traditional pump – In this mode only one pump is run using the VFD and the rest of the pumps are run using the Direct On Line (DOL) motors.



Pressure control mode is not supported by DOL pump.

The combination with any VFD pump is necessary.

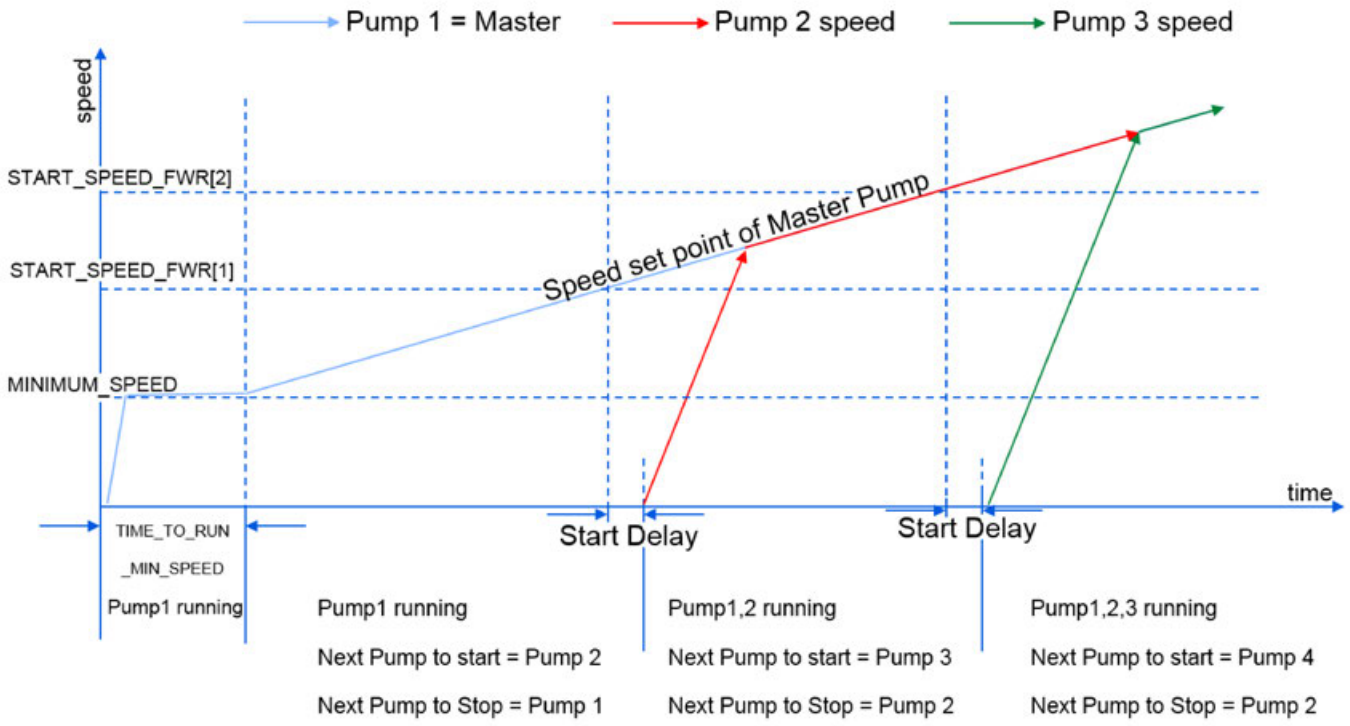
Control philosophy of pressure control mode

The pressure control mode follows this sequence of operation:

1. When the process is started, the first pump runs at MINIMUM_SPEED for the time defined in the TIME_TO_RUN_MIN_SPEED duration. With this process the pipe starts filling gradually and then the normal operation of PID control takes over.
2. The PID compares the required pressured and the actual pressure to generate the output in terms of percentage. This output must be connected to the function block PMP_PRESSURE_DISTRIBUTOR.

3. The function block PMP_PRESSURE_DISTRIBUTOR receives the PID_OUT. The distributor converts the PID output in terms of speed for the master pump.
4. The function block PMP_SEQUENCE_GEN has the MASTER_PUMP information.
5. The PID_OUT is scaled in terms of speed in this method:

   Speed reference of MASTER_PUMP = (NOMINAL_SPEED of MASTER_PUMP) * (PID_OUT/100)
6. As the PID output increases, the speed of the master pump increases. A higher PID output indicates a high demand. In case one pump is not able to cater the requirement, then more pump (followers) are needed to start and supply water to maintain the pressure.
7. If speed reference of the MASTER_PUMP increases such that it is more than the START_SPEED_FWR[1], then the first follower in the network will start. The information about which pump to start comes from function block PMP_SEQUENCE_GEN at the output PNSTART.
8. If the speed reference of the MASTER_PUMP increases further such that it is more than START_SPEED_FWR[2], then second follower pump starts. This sequence is followed till there is an increase in demand. See the timing diagram below.



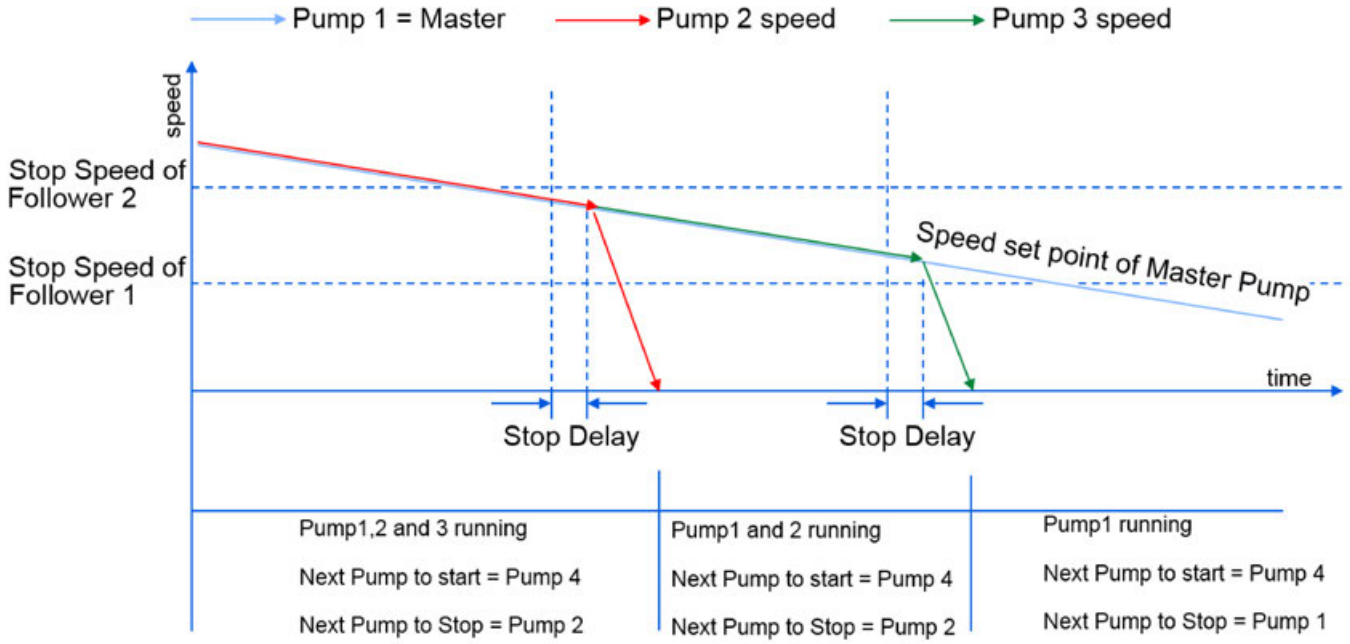
Speed setpoint of Master Pump = PID_OUT % * Nominal speed of master Pump

Start Delay: Delay time [s] to start the next pump

9. Similar when the demand decreases, the speed reference of master decreases. See the timing diagram below. As the speed reference of the master pump goes below STOP_SPEED_FWR[1], the first follower stops. The ID of next pump to stop comes from PMP_SEQUENCE_GEN as PNSTOP.

   If the speed reference of the MASTER_PUMP decreases further such that it is less than STOP_SPEED_FWR[2], then second follower pump stops. This sequence is followed till there is a decrease in demand.

> *The follower pumps in the above figure is considered to be in the FOLLOWER_MODE = 1, copy master speed. They can also run at their individual speed if FOLLOWER_MODE = 2, fixed speed.*



Speed setpoint of Master Pump = PID_OUT % * Nominal speed of master Pump

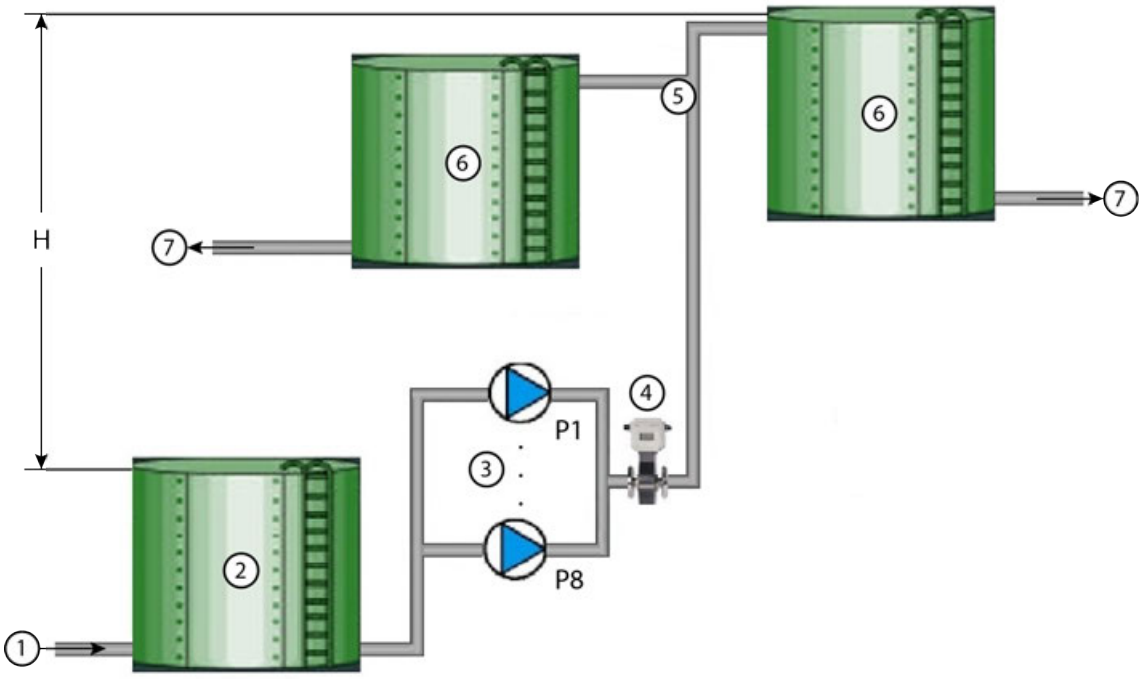
Stop Delay: Delay time [s] to stop the next pump

Flow control function block

Flow control process flow diagram

Flow control is used in applications where continuous flow distribution to one or more target tanks or reservoirs is needed.

A typical flow control application diagram is shown below.



1 From supply station
2 Suction tank
3 Parallel operating pumps P1 ... P8
4 Flow setpoint, discharge pressure
5 Water distribution network
6 Target tank/Reservoir
7 To next station or distribution network
H Water head between the levels in the suction tank and target tank

Using flow control

Flow control in the pumping library can help in followings pumping operations:

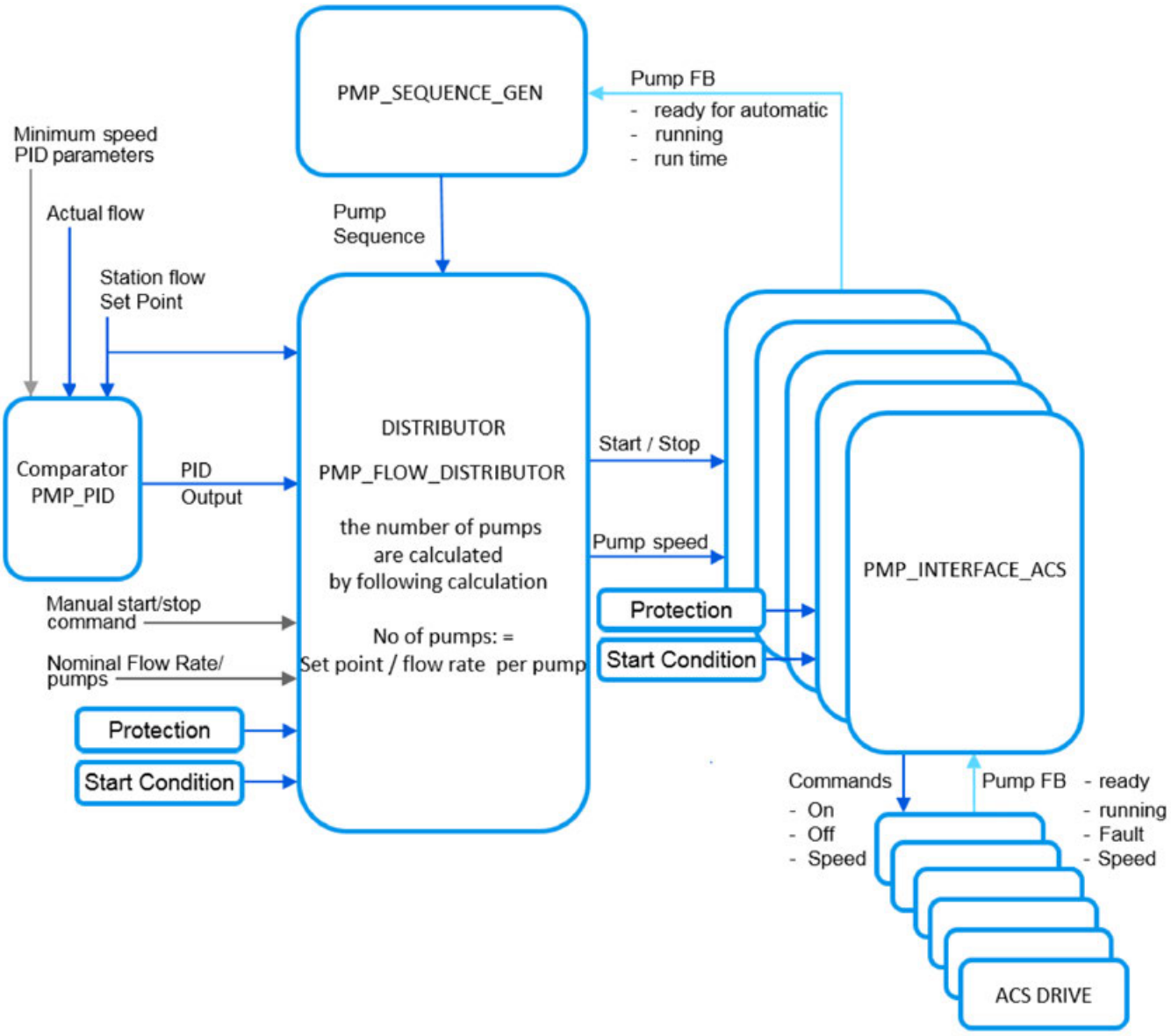
- Supplying water to the pump station through the suction pipeline from a suction tank.
- Operating pumps in parallel.
- Controlling continuous water flow over a time period.
- Measuring the discharge flow in the discharge pipeline.
- Defining the number of pumps and pump speed based on water flow demand (flow set point).
- Starting each pump with minimum speed to build up the pressure required to produce a minimum flow.

  The pressure depends on the water head (H) between the water levels in the suction tank and the target tank.
- In case of station shutdown stops the pumps sequentially to prevent water hammering.
- Stops variable speed drives through a speed ramp.

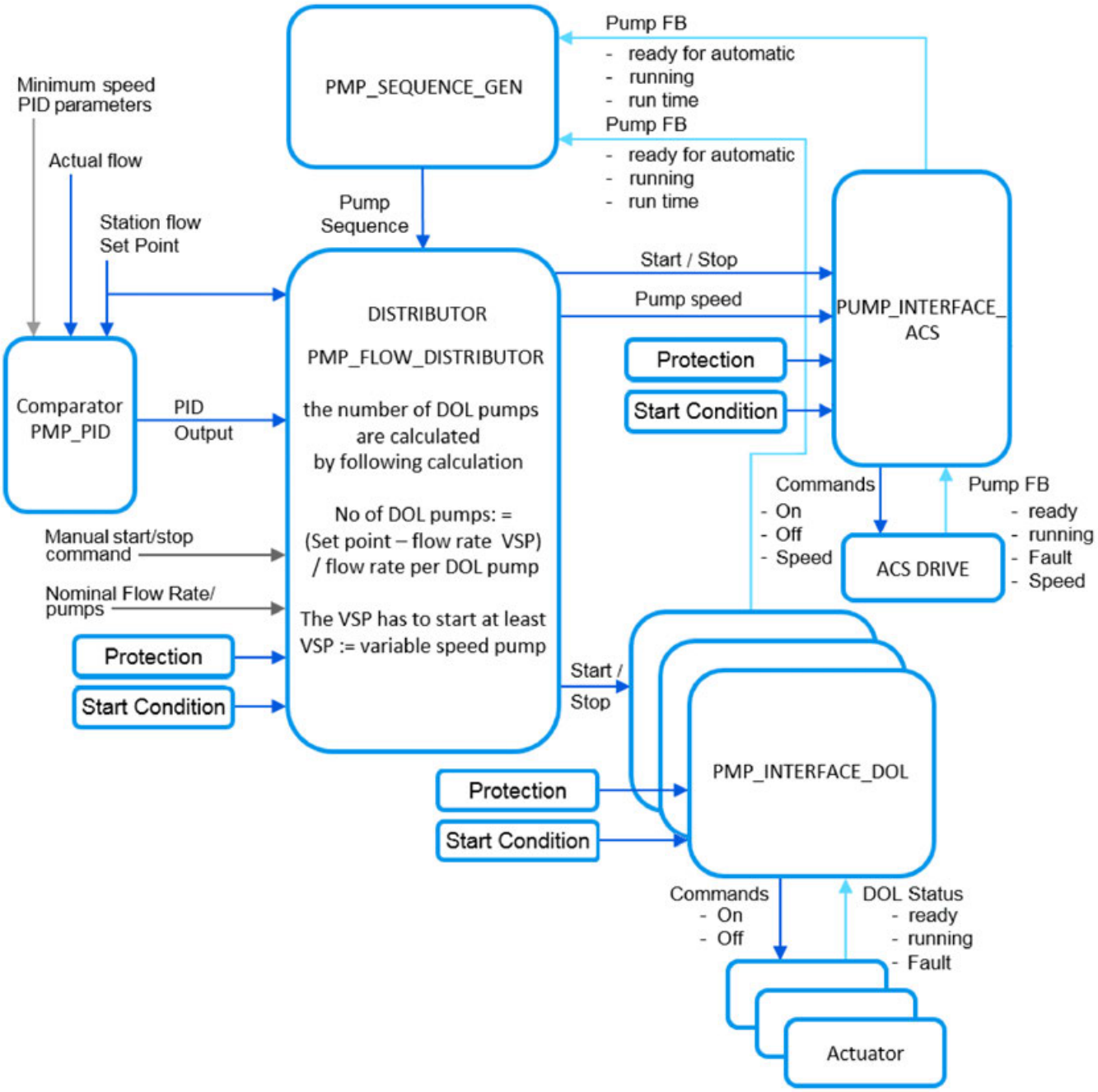
Pump combinations for flow control

Flow control works for all the three types of pump combinations: Multi pump, traditional pump, and DOL pump.

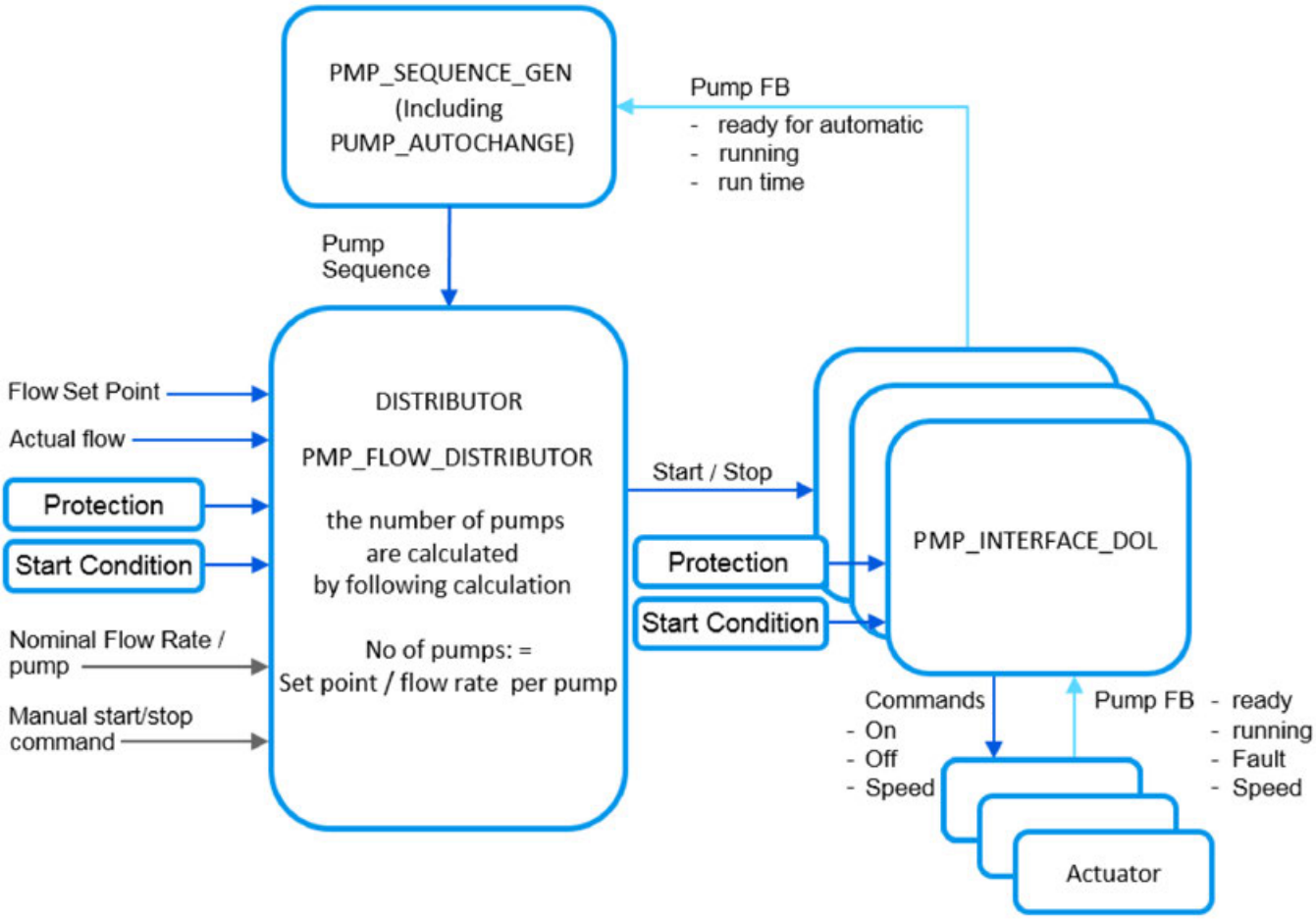
- Multi pump – In this mode all pumps are run using VFDs. See the example diagram below.



- Traditional pump – In this mode only one pump is run using the VFD and rest are run using the Direct On Line (DOL) motors. See the example diagram below.



- DOL Pump – In this mode all the pumps are operated by DOL motors. The function block PMP_PID is not configured as there is no possibility to change the speed of the pump. See the example diagram below.



Control philosophy of flow control mode

The above figures show the combination of function blocks to execute the flow control. As it can be seen,

- When the process is started, the first pump runs at the MINIMUM_SPEED for the time defined in the TIME_TO_RUN_MIN_SPEED duration. This is to gradually start filling the pipe. After this the normal operation of PID control takes over.
- The PID acts as a comparator: the PID compares the required flow and the actual flow to generate output in terms of percentage. This needs to be connected to the PMP_FLOW_DISTRIBUTOR.
- PMP_ FLOW _DISTRIBUTOR takes the PID_OUT. Then converts this output in terms of speed for the master pump. The information of master pump comes from the MASTER_PUMP of the PMP_SEQUENCE_GEN. The scaling of the PID_OUT in terms of speed is done in following ways.

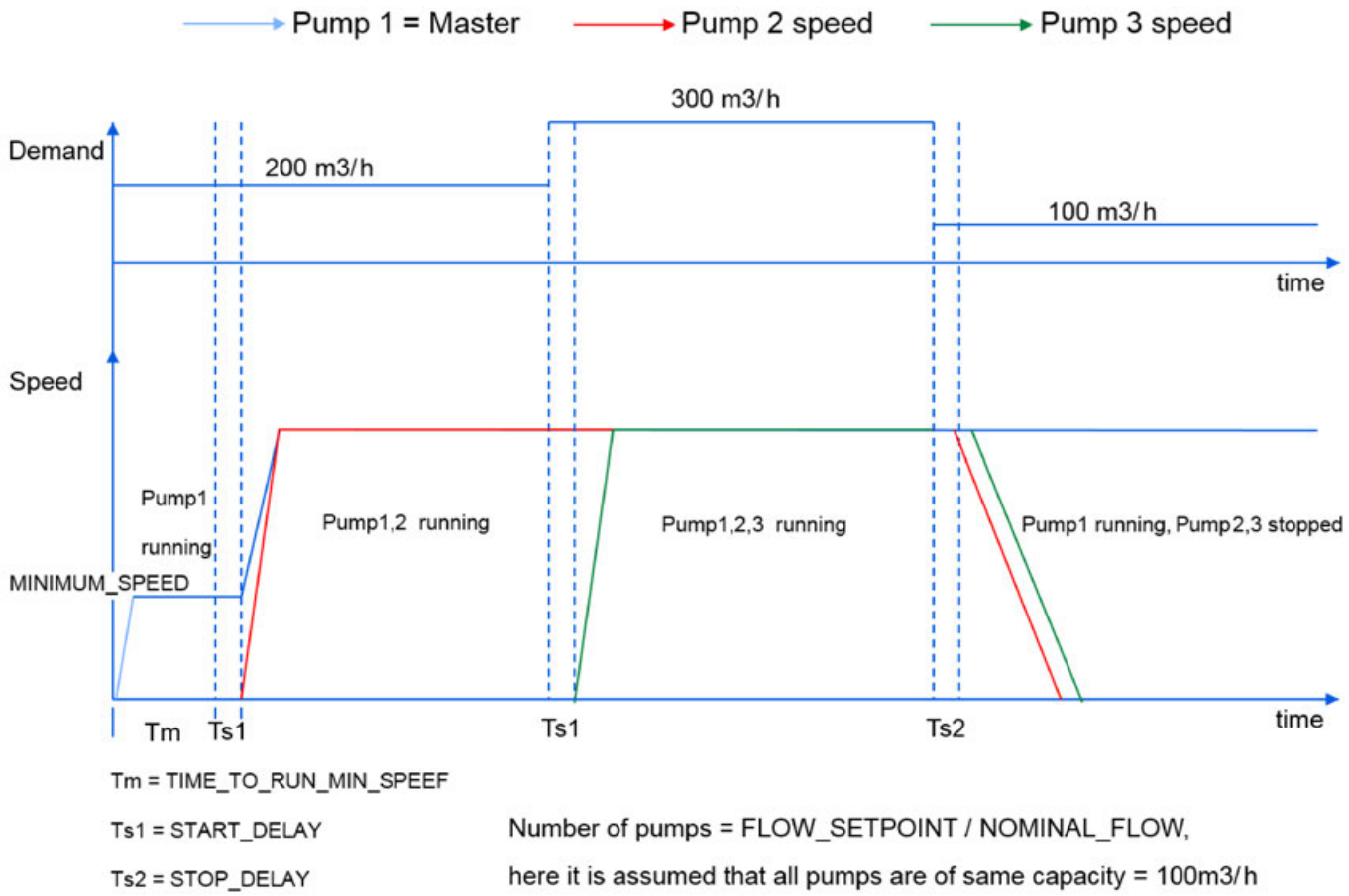
  Speed reference MASTER_PUMP = (NOMINAL_SPEED of MASTER_PUMP) * (PID_OUT/100)
- This speed reference is given to the master and the follower (if FOLLOWER_MODE = 1). If FOLLOWER_MODE = 2, then follower pumps will run at their fixed speed given in the input FOLLOWER_SPEED[1..8].
- The distributor then decides about the number of pumps to operate based on the following formula,

  Number of pumps = FLOW_SETPOINT / NOMINAL_FLOW, here it is assumed that all pumps are of same rating.

It is assumed that all the pumps are of same capacity.

- As the flow set point increases, the demand of pumps increases.
- Example: if each pump is of capacity = 100 cubic meters per hour and the flow set point is 400 cubic meters per hour, then four pumps will start together. This is unlike the pressure control where the pumps do not start together but based on speed reference of the master pump.

Flow control with distributor



- Similarly when the demand decreases, the speed reference of the master decreases. As the speed reference of the master pump goes below than STOP_SPEED_FWR[1], the first follower stops. The ID of next pump to stop comes from PMP_SEQUENCE_GEN as PNSTOP.
- If the speed reference of the MASTER_PUMP decreases further such that it is less than STOP_SPEED_FWR[2], then second follower pump stops. This sequence is follows till there is a decrease in demand.

The follower pumps in the above figures are considered to be in the FOLLOWER_MODE = 1, copy master speed.

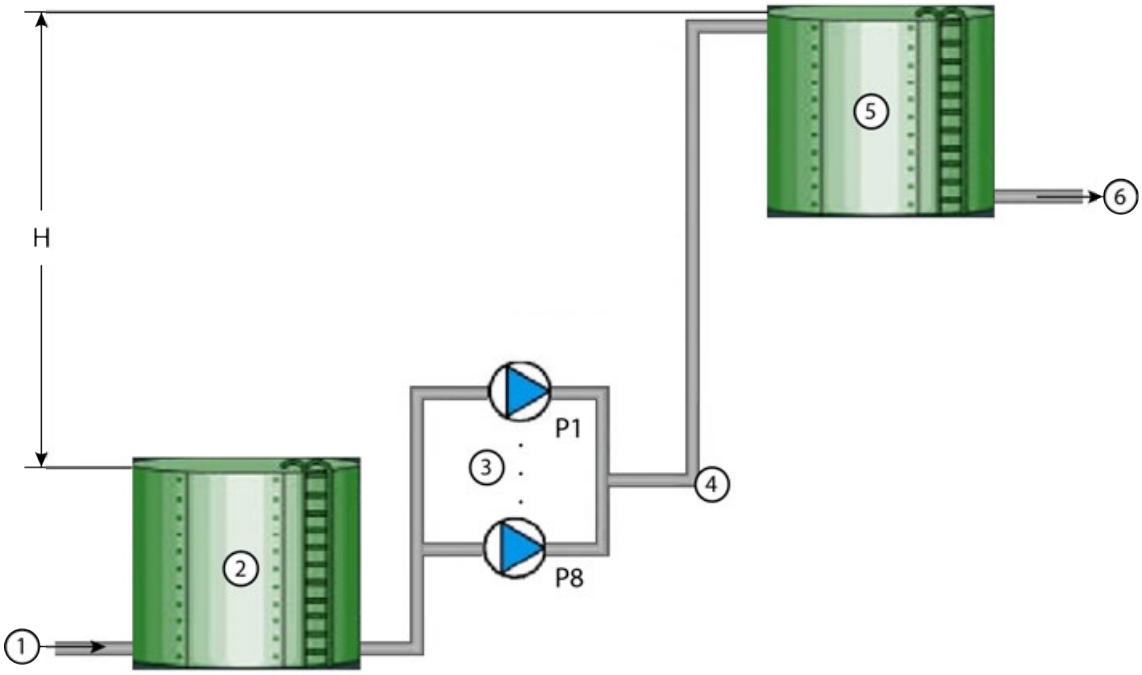
They can also run at their individual speed if FOLLOWER_MODE = 2, fixed speed.

Level control – Emptying or filling

Level control process flow diagram

Level control is used in applications where continuous flow distribution into one target tank or reservoir is needed.

A typical flow control application diagram is shown below.



1 From supply station
2 Suction tank
3 Parallel operating pumps P1 ... P8
4 Water distribution network
5 Target tank/Reservoir
6 To next station or distribution network
H Water head between the levels in the suction tank and target tank

Using level control

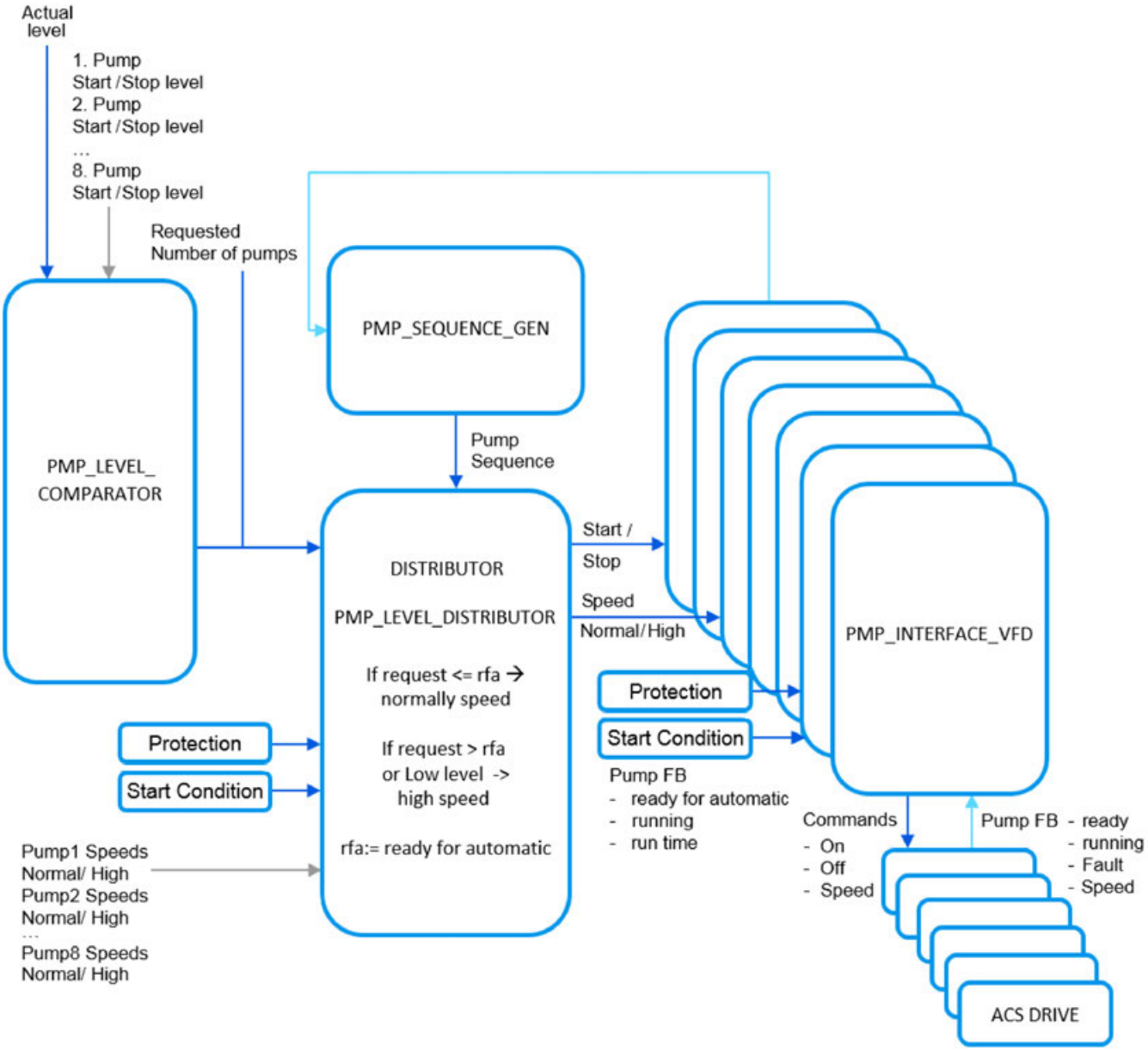
Level control in the pumping library can help in followings pumping operations:

- Emptying the target tank/reservoir by the next pump station or water distribution network.
- Filling the suction tank from a supply station.
- Defining the number of running pumps based on water level in the water tank.
- Start/stop the pumps with predefined start/stop levels.
- In case of a station shutdown, stop pumps sequentially to prevent water hammering.

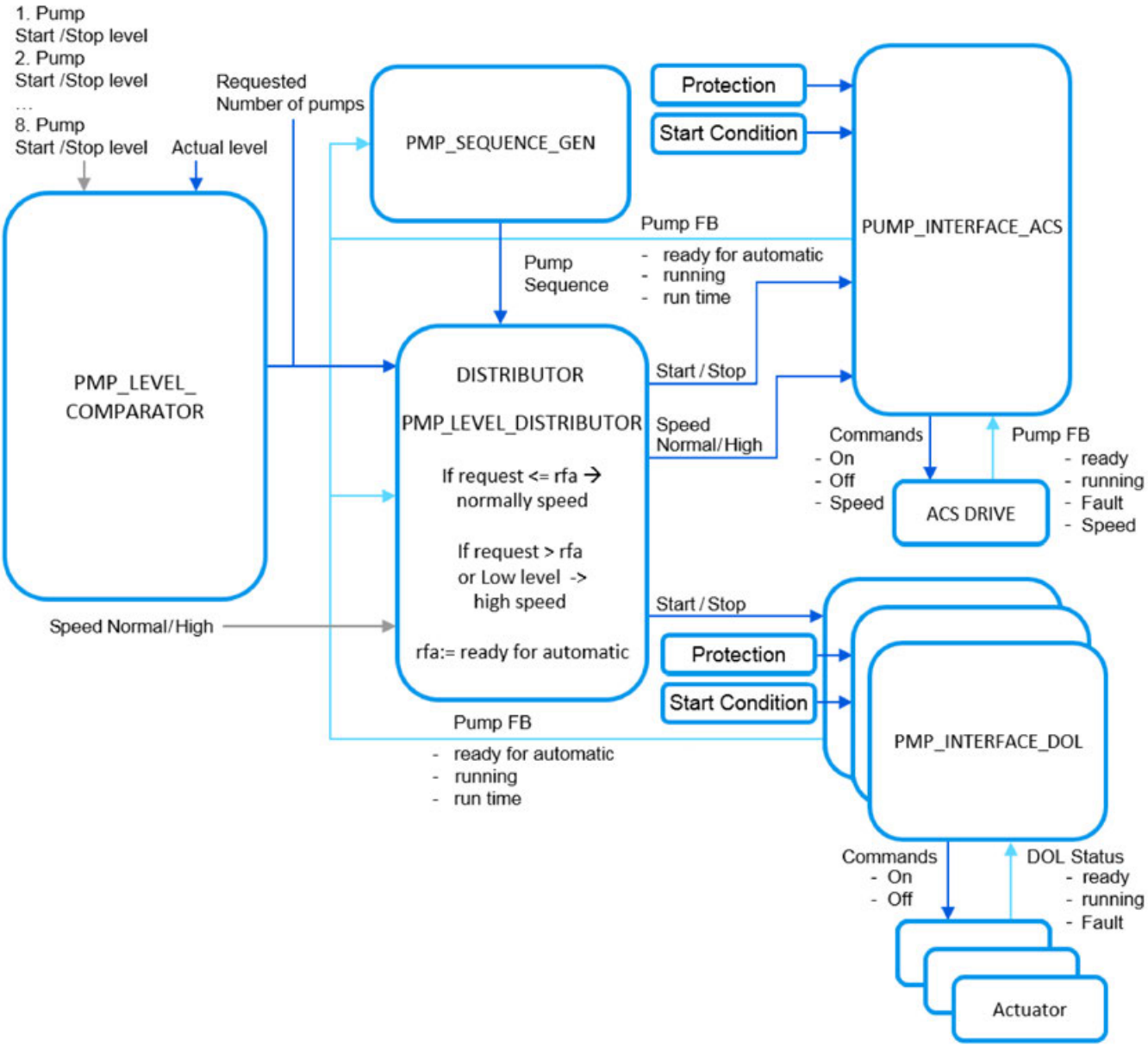
Pump combinations for level control

Level control works for all the three types of pump combinations: Multi pump, traditional pump, and DOL pump.

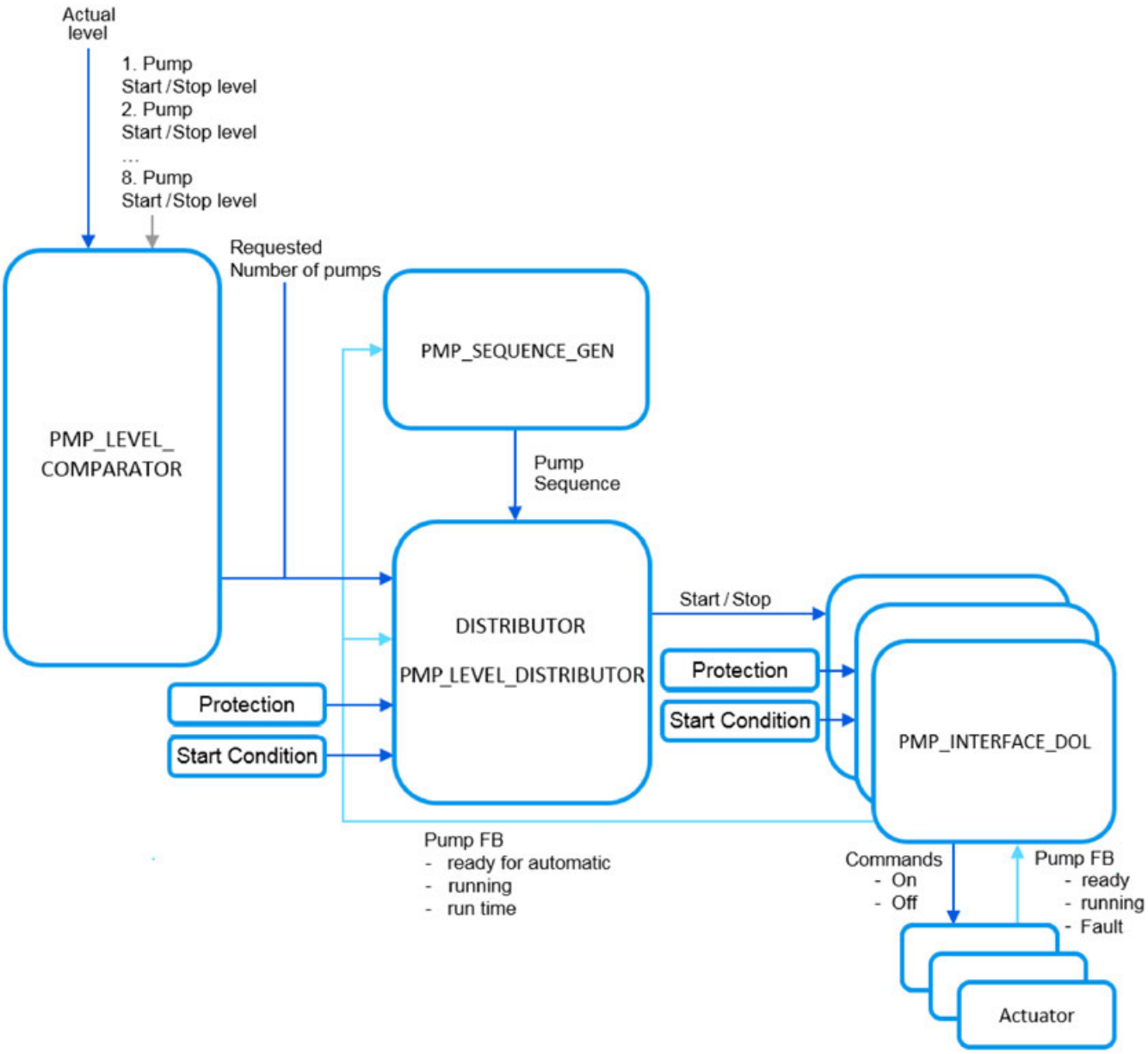
- Multi pump – In this mode all pumps are run using VFDs. See the example diagram below.



- Traditional pump – In this mode only one pump is run using the VFD and rest using the Direct On Line (DOL) motors. See the example diagram below.



- DOL Pump – In this mode all the pumps are operated by DOL motors. In this case the PMP_PID is not configured as there is no possibility to change the speed of the pump. See the example diagram below.



Control philosophy of level control mode

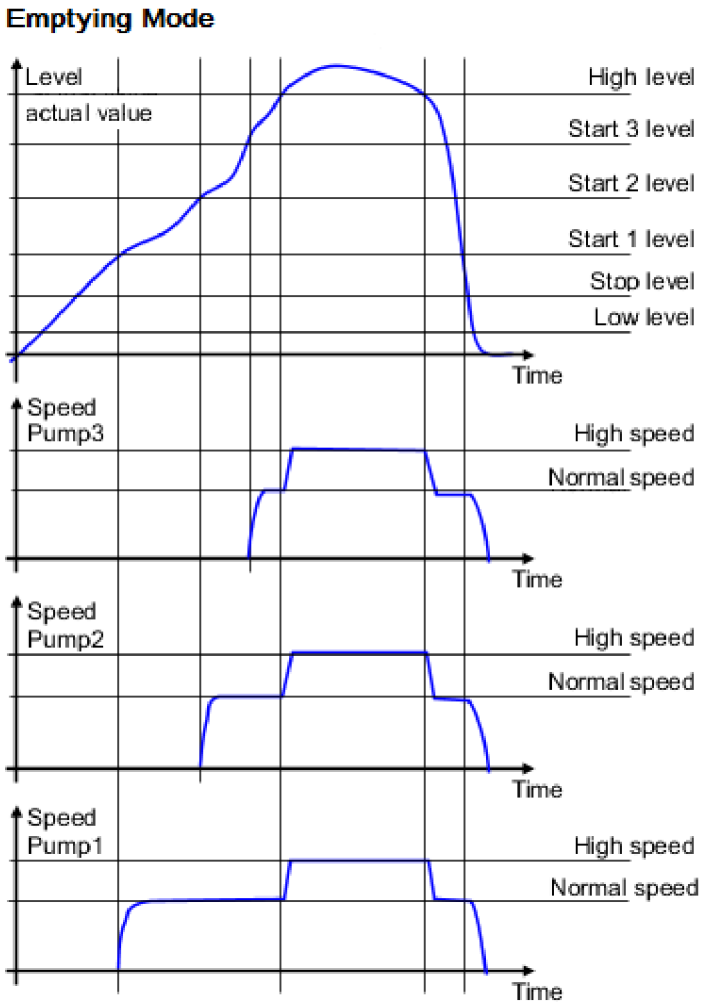
The level control is based on two modes.

- Emptying mode
- Filling mode

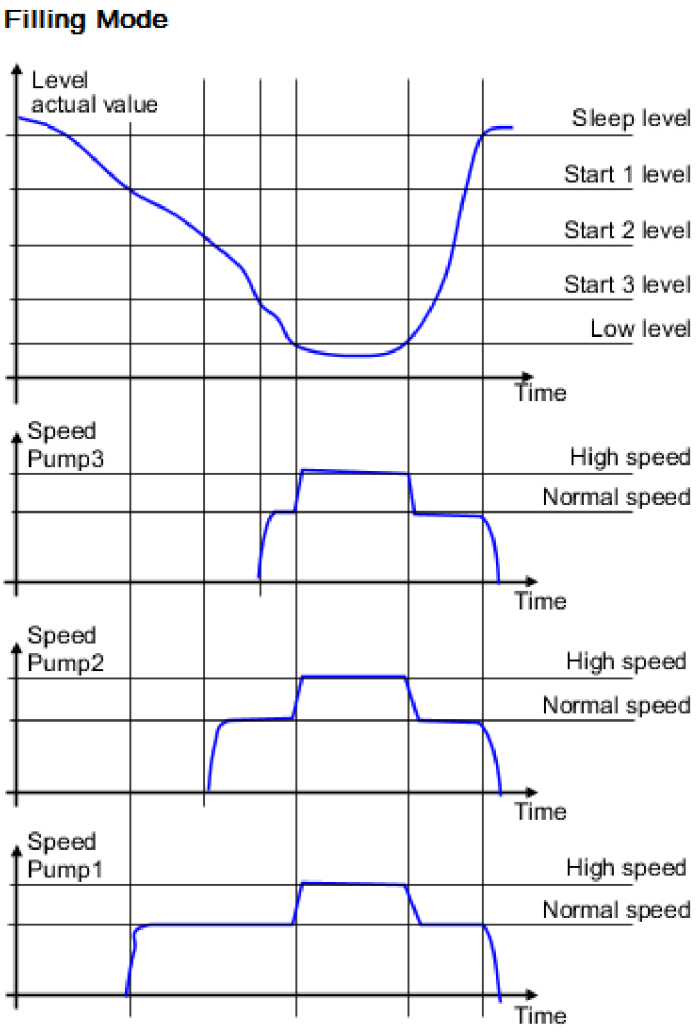
The level control mode follows this sequence of operation:

- The PMP_LEVEL_COMPARATOR compares the actual level (ACT_LEVEL) with the set start levels (START_LEVEL [1...8]). Based on which the comparator decides the demand (N_DEMAND). This demand goes as an input information to the PMP_ LEVEL _DISTRIBUTOR.
- The PMP_LEVEL_DISTRIBUTOR then starts the required pumps. In case the pumps are driven by VFD, the distributor also gives the speed reference to each pump. The information of which pump to start in the sequence is given to the distributor by PMP_SEQUENCE_GEN.

Time diagram: Emptying mode

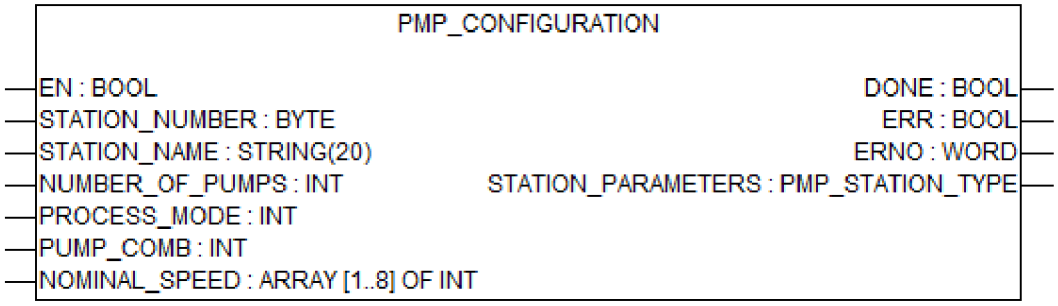


Time diagram:
Filling mode



1.5.14.2 Function block description

1.5.14.2.1 PMP_CONFIGURATION

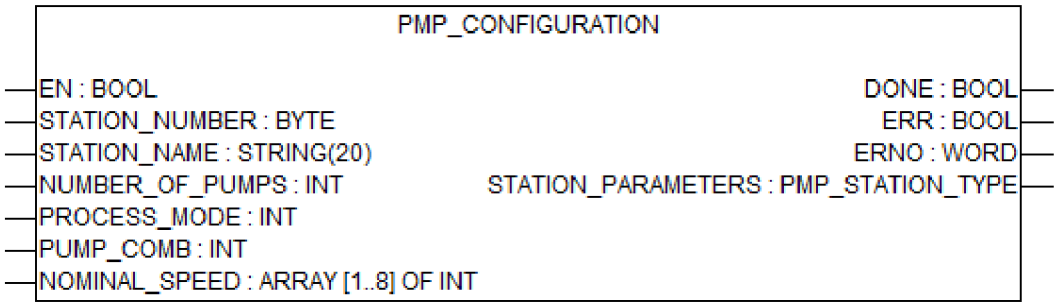


The function block PMP_CONFIGURATION configures the pump station. The function block receives the pumping station details (e.g., station name, station number) and stores in the structure PMP_STATION_TYPE.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

STATION_NUMBER

Data type	Default value	Range	Unit
BYTE	1	> 0	-

The input STATION_NUMBER defines the assigned station number, which is used as a unique identification number when more than one station is configured in a single PLC.

STATION_NAME

Data type	Default value	Range	Unit
STRING(20)	'Station1'	20 characters	-

The input STATION_NAME input defines the configured station name.

NUMBER_OF_PUMPS

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input NUMBER_OF_PUMPS defines the number of pumps used in the pumping station.

PROCESS_MODE

Data type	Default value	Range	Unit
INT	1	1 ... 3	-

The input PROCESS_MODE selects the process control mode for pump operation. The function block contains following control modes:

1 = Pressure control: for pressure stabilization

2 = Flow control: for maintaining the required flow in the process

3 = Level control: for emptying or filling the tank

PUMP_COMB

Data type	Default value	Range	Unit
INT	1	1 ... 3	-

The input PUMP_COMB selects the process combination mode for pump operation. The function block contains following modes:

1 = Multimode: this mode drives all pumps fed by VFDs.

2 = Traditional mode: in this mode only one pump is fed by VFD and rest of the pumps are fed by DOL motors.

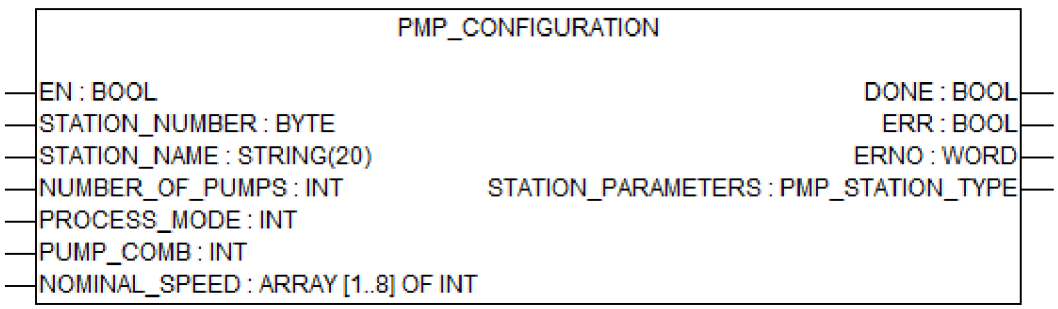
3 = DOL mode: in this mode all pumps are fed by DOL motors.

NOMINAL_SPEED

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	8(1500)	-	rpm

The input NOMINAL_SPEED defines the rated pump speed in rpm. This input is an array where the first index is meant for PUMP_ID = 1 in the interface function blocks PMP_INTERFACE_DOL and PMP_INTERFACE_VFD.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control

Parameter	Data type	Default value	Description
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Error codes

The error codes of function block PMP_CONFIGURATION are listed below:

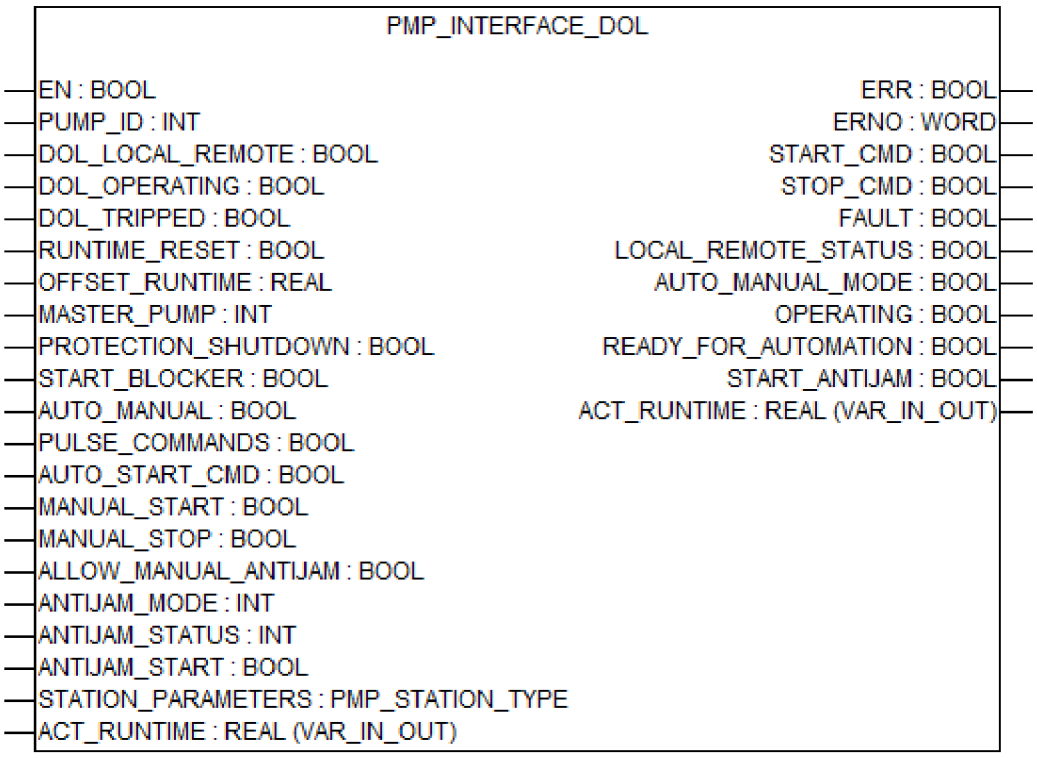
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value of STATION_NUMBER is less than 1
16449	4041	Value of NUMBER_OF_PUMPS is less than 1
16450	4042	Value of NUMBER_OF_PUMPS is beyond 8
16465	4051	Value of PROCESS_MODE is less than 1
16466	4052	Value of PROCESS_MODE is beyond 3
16481	4061	Value of PUMP_COMB is less than 1
16482	4062	Value of PUMP_COMB is beyond 3
16483	4063	PUMP_COMB cannot be = 3 for the PROCESS_MODE = 1
16497	4071	Value of NOMINAL_SPEED is less than or equal to 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.2 PMP_INTERFACE_DOL

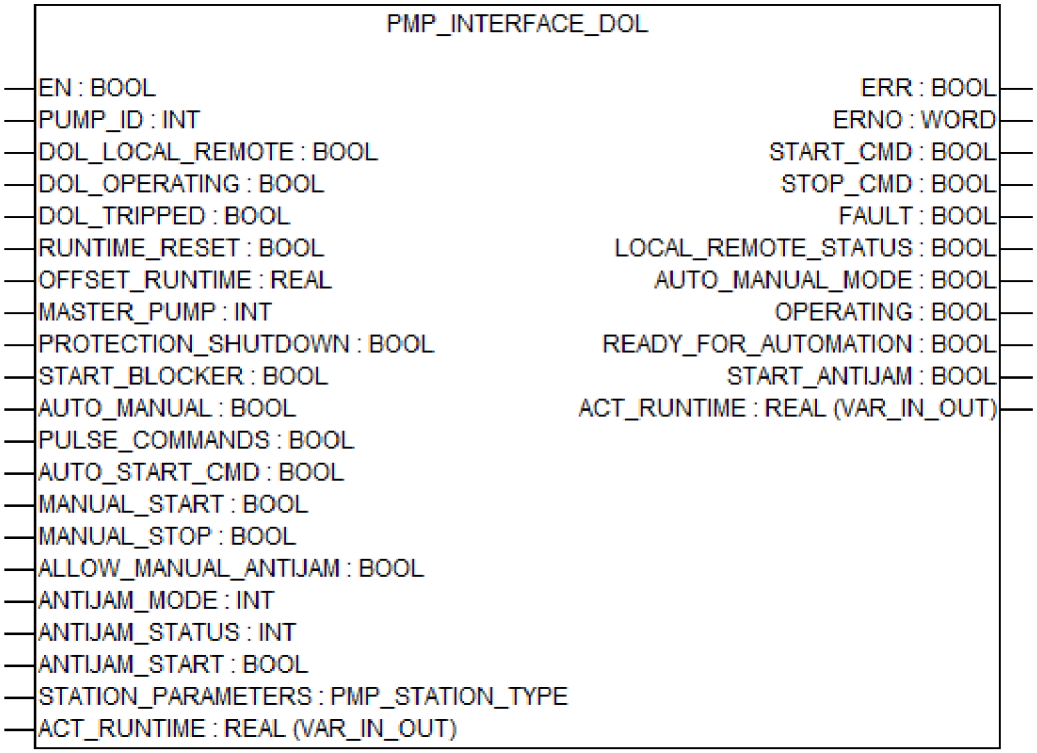


The function block PMP_INTERFACE_DOL is used to communicate with all pumps in the field that are run by direct online (DOL) motor. The function block configures pump settings and gives the status of pump operation. This function block is effective only for pumps fed by DOL motors.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PUMP_ID defines the pump identification number.

DOL_LOCAL_REMOTE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DOL_LOCAL_REMOTE connects the local or remote mode status of the DOL system. The function block uses this information to decide whether the pump is ready for automation.

- When input is TRUE, pump operates in LOCAL mode.
- When input is FALSE, pump operates in REMOTE mode.

DOL_ OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DOL_OPERATING connects the operating status of the DOL system. The function block uses this information to decide whether the pump is ready for automation.

- When input is TRUE, pump is operating.
- When input is FALSE, pump is not ready for operation.

DOL_TRIPPED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DOL_TRIPPED connects the faulty/healthy status of the DOL system. The function block uses this information to decide whether the pump is ready for automation.

- When input is TRUE, pump has tripped due to fault.
- When input is FALSE, pump operation is healthy.

RUNTIME_ RESET

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RUNTIME_RESET resets the actual run time in the output ACTUAL_RUNTIME with value in the input OFFSET_RUNTIME for the pump.

- When input is TRUE, output ACTUAL_RUNTIME is reset to value in input OFFSET_RUNTIME.
- When input is FALSE, pump continues operation in current run time.

OFFSET_ RUNTIME

Data type	Default value	Range	Unit
REAL	1.0	-	-

The input OFFSET_RUNTIME defines offset run time of the pump that overwrites actual run time in the output ACTUAL_RUNTIME when input RUNTIME_RESET = TRUE.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

PROTECTION_ SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_BLOCKER receives a digital signal to prevent the starting of pump station for protection. The input comes from the output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

The input START_BLOCKER will not stop the already running pump.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

AUTO_MANUAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input AUTO_MANUAL selects the operating mode of the pump.

- When input is TRUE, pump operates in automatic mode.
- When input is FALSE, pump operates in manual mode.

PULSE_ COMMANDS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PULSE_COMMANDS selects the Start/Stop command type of the pump run by DOL motor.

- When input is TRUE, pump starts with the pulse input of start command.
- When input is FALSE, pump starts with the continuous start command.

AUTO_START_ CMD

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input AUTO_START_CMD receives the start command of the pump to start in automatic mode. The input comes from the output AUTO_START_CMD of function block PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR. Automatic start works only when input AUTO_MANUAL = TRUE.

- When input is TRUE, pump starts in automatic mode.
- When input is FALSE, pump stops if running in automatic mode.

MANUAL_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input MANUAL_START receives the start command to start the pump in manual mode.

This manual start function applies only for a pump fed by DOL motor. The function works only when input AUTO_MANUAL = FALSE.

The function also depends on the input PULSE_COMMANDS:

If input PULSE_COMMANDS = FALSE i.e. continuous mode is selected.

- When input MANUAL_START = TRUE, pump starts in manual mode.
- When input MANUAL_START = FALSE, pump stops if running in manual mode.

If input PULSE_COMMANDS = TRUE i.e. pulse mode is selected.

- Pump starts in manual mode at the rising edge i.e. FALSE ➨ TRUE.

MANUAL_STOP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

This input MANUAL_STOP stops the pump under following conditions:

- Input PULSE_COMMANDS = TRUE i.e. pulse mode is selected.
- Inputs AUTO_MANUAL = FALSE and MANUAL_STOP = TRUE.

This manual stop function applies only for a pump fed by DOL motor.

ALLOW_ MANUAL_ ANTIJAM

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input ALLOW_MANUAL_ANTIJAM allows antijam operation of the pump in manual mode. The input is effective only when the input AUTO_MANUAL = FALSE and ANTIJAM_MODE = Manual mode.

- When input is TRUE, antijam operation is allowed.
- When input is FALSE, antijam operation is not allowed.

ANTIJAM_ MODE

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE receives the antijam mode. The input comes from the output ANTIJAM_MODE of the function block PMP_ANTIJAM.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the antijam operation status of the pump. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output ANTIJAM_START of the function block PMP_ANTIJAM starts the antijam operation. This output is connected with the input ANTIJAM_START of the function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL.

- When output is TRUE, antijam operation starts.
- When output is FALSE, antijam operation stops if running.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

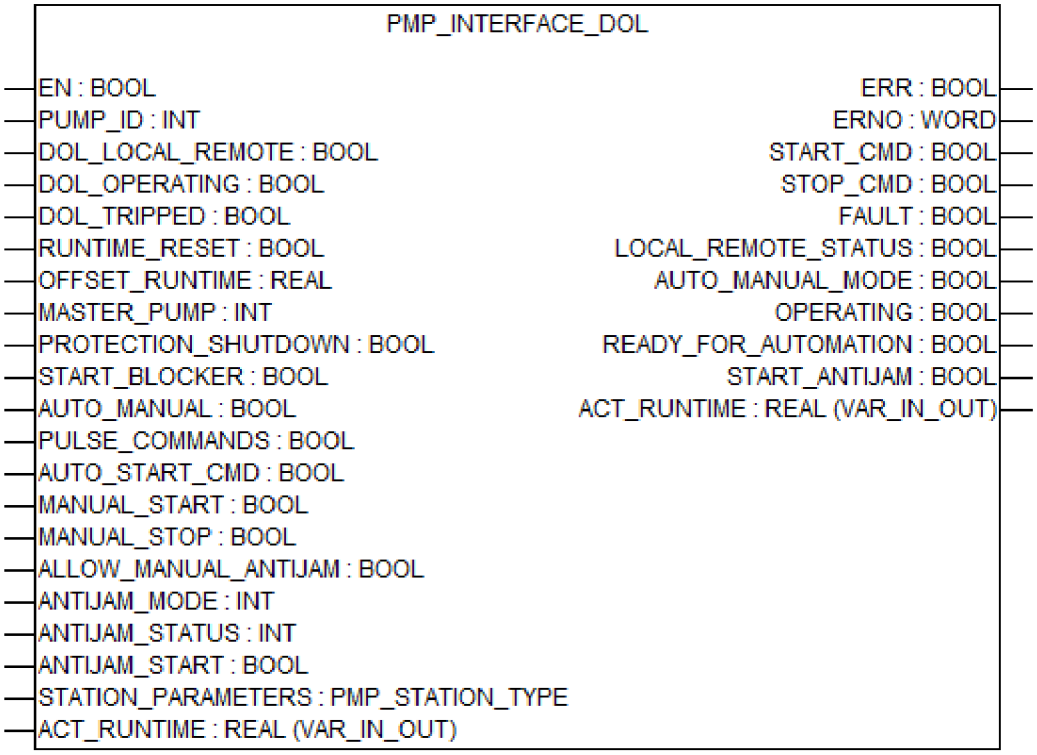
Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

ACT_RUNTIME

Data type	Default value	Range	Unit
REAL	-	-	h

The input ACT_RUNTIME defines the actual run time of pump in hours.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

START_CMD

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output START_CMD is the command to start the pump fed by DOL motor. The command works only when input PULSE_COMMANDS = FALSE, i.e. continuous mode is selected.

- When output is TRUE, pump starts running.
- When output is FALSE, pump stops if already running.

When input PULSE_COMMANDS = TRUE, the command generates a single pulse to start the motor.

STOP_CMD

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output STOP_CMD is the command to stop the pump fed by DOL motor. The output works based on the value in input PULSE_COMMANDS.

- If PULSE_COMMANDS = FALSE, the output will stay FALSE and unused.
- If PULSE_COMMAND = TRUE, the output generates a single pulse to stop the motor.

FAULT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output FAULT indicates a fault in the pump. The fault may be in the pump protection blocks.

- When output is TRUE, pump fault occurred.
- When output is FALSE, pump operation is healthy.

LOCAL_ REMOTE_ STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output LOCAL_REMOTE_STATUS indicates the operating mode of the pump system (DOL/VFD).

- When output is TRUE, pump system is in local mode.
- When output is FALSE, pump system is in remote mode.

AUTO_MANUAL _MODE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output AUTO_MANUAL_MODE indicates the operation mode of the pump.

- When output is TRUE, pump is in automatic mode.
- When output is FALSE, pump is in manual mode.

OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output OPERATING indicates that the device is operating.

- When output is TRUE, device is operating.
- When output is FALSE, device is not operating.

READY_FOR_AUTOMATION

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output READY_FOR_AUTOMATION of the pump interface function block indicates that the pump is ready for operation in automatic mode. This function is effective only when the function block is connected to the input READY_FOR_AUTOMATION of the following function blocks: PMP_SEQUENCE_GEN, PMP_FLOW_DISTRIBUTOR, PMP_LEVEL_DISTRIBUTOR and PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, pump is ready for operation in automatic mode.
- When output is FALSE, pump is not ready for operation in automatic mode.

START_ANTIJAM

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output START_ANTIJAM starts antijam operation in manual mode. This function is effective only when the function block is connected to input START_MANUAL of function block PMP_ANTIJAM.

- When output is TRUE, antijam operation starts in manual mode.
- When output is FALSE, antijam operation stops if running.

Error codes

The error codes of function block PMP_INTERFACE_DOL are listed below:

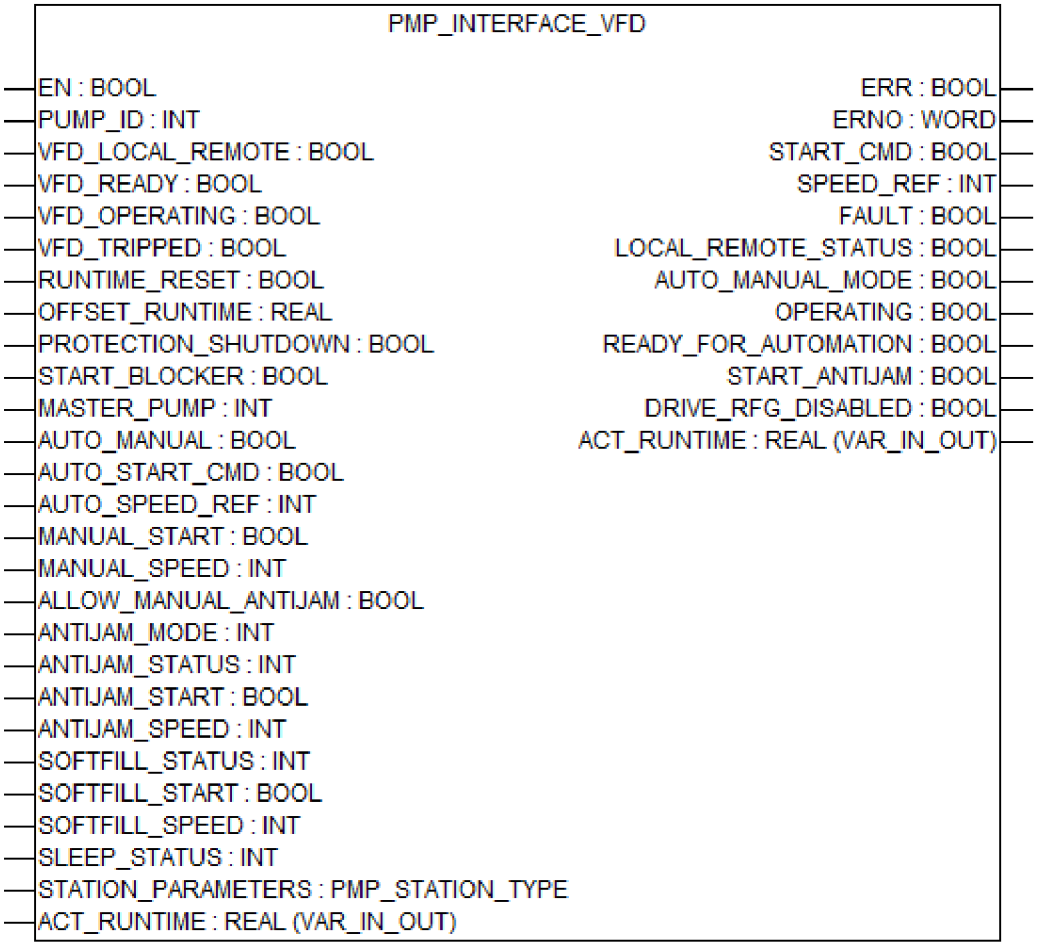
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value of PUMP_ID is less than 1
16418	4022	Value of PUMP_ID is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16641	4101	Value of ANTIJAM_MODE is less than 0
16642	4102	Value of ANTIJAM_MODE is greater than 3
16657	4111	Value of ANTIJAM_STATUS is less than 0
16658	4112	Value of ANTIJAM_STATUS is greater than 2
16611	40E3	Value of MANUAL_START and ALLOW_MANUAL_ANTIJAM is TRUE simultaneously. Either one operation is allowed at a time in manual mode, i.e. manual pump start or manual antijam.

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.3 PMP_INTERFACE_VFD



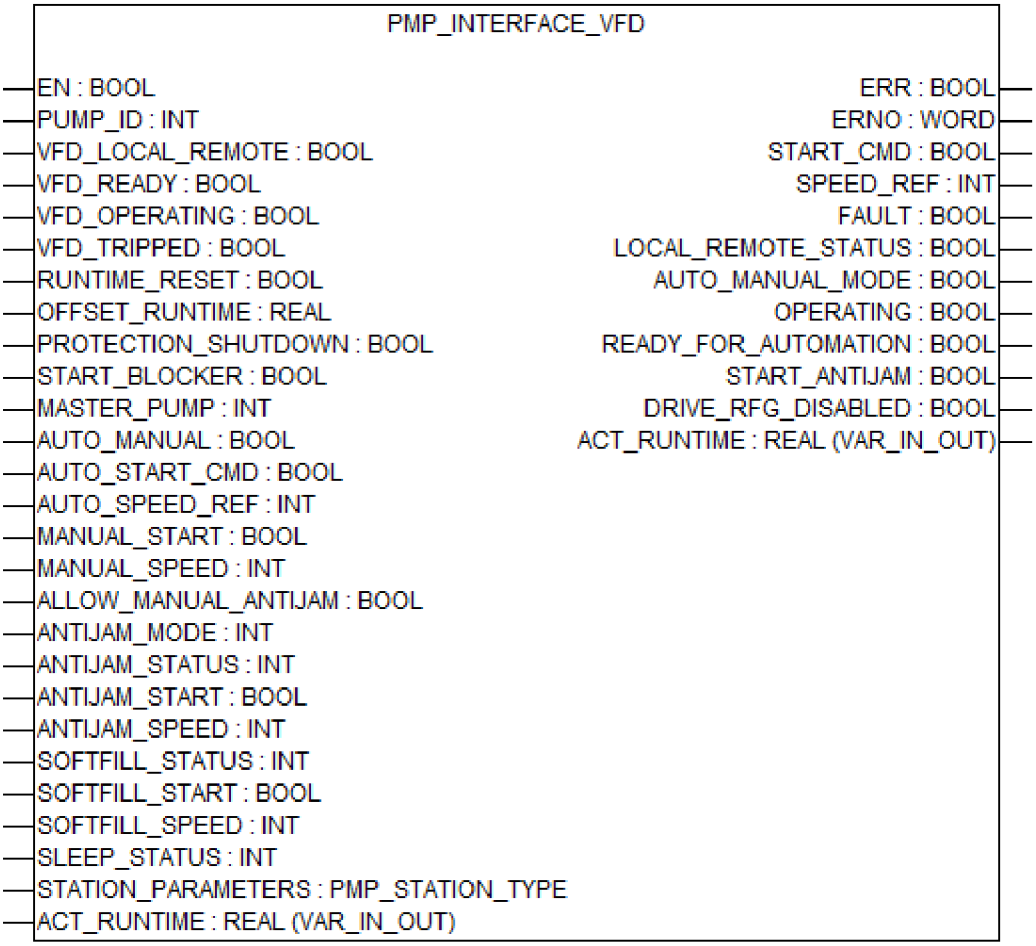
The function block PMP_INTERFACE_VFD enables communication within all connected pumps in the field run by VFD motors. The function block configures pump settings and gives the status of pump operation. This function block is effective only for pumps fed by VFD motors.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

The function block PMP_INTERFACE_VFD checks the status of variable frequency drives (VFDs) and sends the ON/OFF command and speed signals back to the drives.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PUMP_ID defines the pump identification number.

VFD_LOCAL_REMOTE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input VFD_LOCAL_REMOTE receives the operating mode of the variable frequency drive.

- When input is TRUE, drive is operating in LOCAL mode.
- When input is FALSE, drive is operating in REMOTE mode.

VFD_READY

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input VFD_READY receives the ready for operation status of the variable frequency drive.

- When input is TRUE, drive is ready for operation.
- When input is FALSE, drive is not ready for operation.

VFD_OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input VFD_OPERATING receives the operating status of the variable frequency drive.

- When input is TRUE, drive is operating.
- When input is FALSE, drive is not operating.

VFD_TRIPPED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input VFD_TRIPPED receives the tripped status of the variable frequency drive.

- When input is TRUE, drive has tripped due to a fault.
- When input is FALSE, drive is healthy.

RUNTIME_RESET

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RUNTIME_RESET resets the actual run time in the output ACTUAL_RUNTIME with value in the input OFFSET_RUNTIME for the pump.

- When input is TRUE, output ACTUAL_RUNTIME is reset to value in input OFFSET_RUNTIME.
- When input is FALSE, pump continues operation in current run time.

OFFSET_RUNTIME

Data type	Default value	Range	Unit
REAL	1.0	-	-

The input OFFSET_RUNTIME defines offset run time of the pump that overwrites actual run time in the output ACTUAL_RUNTIME when input RUNTIME_RESET = TRUE.

PROTECTION_SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

START_BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_BLOCKER receives a digital signal to prevent the starting of pump station for protection. The input comes from the output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

The input START_BLOCKER will not stop the already running pump.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

AUTO_MANUAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input AUTO_MANUAL selects the operating mode of the pump.

- When input is TRUE, pump operates in automatic mode.
- When input is FALSE, pump operates in manual mode.

AUTO_START_CMD

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input AUTO_START_CMD receives the start command of the pump to start in automatic mode. The input comes from the output AUTO_START_CMD of function block PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR. Automatic start works only when input AUTO_MANUAL = TRUE.

- When input is TRUE, pump starts in automatic mode.
- When input is FALSE, pump stops if running in automatic mode.

AUTO_SPEED_REF

Data type	Default value	Range	Unit
INT	0	-	rpm

The input AUTO_SPEED_REF defines the speed reference of the pump run by variable frequency drive in automatic mode. The input comes from the output AUTO_SPEED_REF of function blocks PMP_FLOW_DISTRIBUTOR/ PMP_PRESSURE_DISTRIBUTOR/ PMP_LEVEL_DISTRIBUTOR, depending on the process selected in the function bock PMP_CONFIGURATION.

MANUAL_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input MANUAL_START gives the start command to start the pump run by variable frequency drive in manual mode. This input is effective only when input AUTO_MANUAL = FALSE.

- When input is TRUE, pump starts in manual mode.
- When input is FALSE, pump stops if running in manual mode.

MANUAL_SPEED

Data type	Default value	Range	Unit
INT	0	> 0	rpm

The input MANUAL_SPEED defines the speed reference of the pump run by variable frequency drive in manual mode. Manual speed works only when input AUTO_MANUAL = FALSE.

ALLOW_MANUAL_ANTIJAM

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input ALLOW_MANUAL_ANTIJAM allows antijam operation of the pump in manual mode. The input is effective only when the input AUTO_MANUAL = FALSE and ANTIJAM_MODE = Manual mode.

- When input is TRUE, antijam operation is allowed.
- When input is FALSE, antijam operation is not allowed.

ANTIJAM_MODE

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE receives the antijam mode. The input comes from the output ANTIJAM_MODE of the function block PMP_ANTIJAM.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the antijam operation status of the pump. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output ANTIJAM_START of the function block PMP_ANTIJAM starts the antijam operation. This output is connected with the input ANTIJAM_START of the function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL.

- When output is TRUE, antijam operation starts.
- When output is FALSE, antijam operation stops if running.

ANTIJAM_ SPEED

Data type	Default value	Range	Unit
INT	0	-	rpm

The output ANTIJAM_SPEED of the function block PMP_ANTIJAM indicates the speed for antijam operation. This output is connected to the input ANTIJAM_SPEED of the function block PMP_INTERFACE_VFD.

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

SOFTFILL_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_START of the used pump distributor starts the soft filling operation. This function is effective only when this output is connected to the input START of the function block PMP_SOFT_FILLING.

- When input is TRUE, softfill function starts.
- When input is FALSE, softfill function stops if running.

SOFTFILL_SPEED

Data type	Default value	Range	Unit
INT	0	> 0	rpm

The output SOFTFILL_SPEED of the function block PMP_SOFT_FILLING indicates the speed reference for soft filling function. The output must be connected to the input SOFTFILL_SPEED of the interface function block PMP_INTERFACE_VFD.

SLEEP_STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The output SLEEP_STATUS of the function block PMP_SLEEP indicates the sleep status of the pump.

0 = Inactive/Wakeup inactive

1 = Boost activated

2 = Sleep mode active

3 = Wakeup function active

STATION_PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

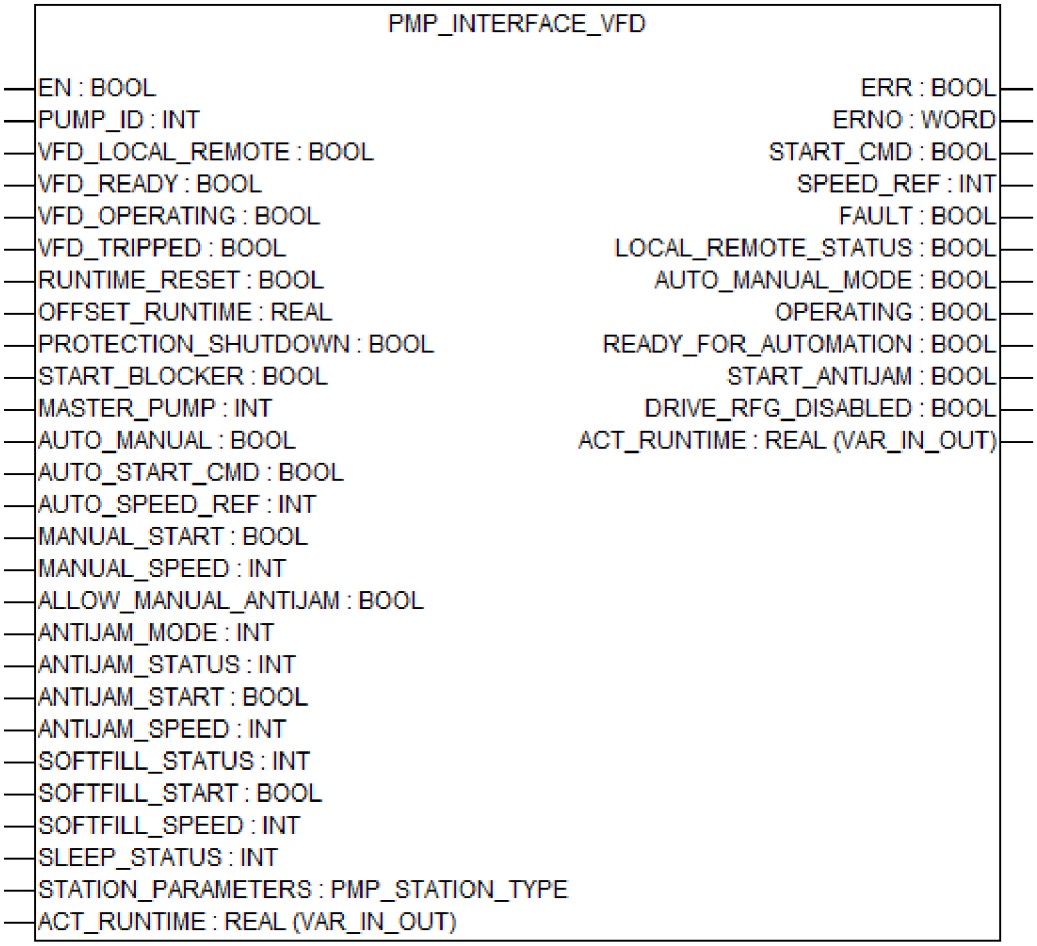
Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

ACT_RUNTIME

Data type	Default value	Range	Unit
REAL	-	-	h

The input ACT_RUNTIME defines the actual run time of pump in hours.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

START_CMD

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output START_CMD is the start command to start the pump fed by variable frequency drive (VFD).

The VFD can run only if is connected to the on command of the VFD.

This can be done using the PLC - VFD communication library, which is not in the scope of the pump library.

- When output is TRUE, pump starts running.
- When output is FALSE, pump stops if already running.

SPEED_REF

Data type	Default value	Range	Unit
INT	0	> 0	rpm

The output SPEED_REF of the interface function block PMP_INTERFACE_VFD shows the speed reference of the variable frequency drive (VFD) in rpm.

The speed reference can be given to the drive only if it is connected to the correct parameter.

This can be done using the PLC - VFD communication library, which is not in the scope of the pump library.

The input SPEED_REF of the function block PMP_DRIVE_SIMU comes from the output SPEED_REF of the interface function block PMP_INTERFACE_VFD.

FAULT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output FAULT indicates a fault in the pump. The fault may be in the pump protection blocks.

- When output is TRUE, pump fault occurred.
- When output is FALSE, pump operation is healthy.

LOCAL_ REMOTE_ STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output LOCAL_REMOTE_STATUS indicates the operating mode of the pump system (DOL/VFD).

- When output is TRUE, pump system is in local mode.
- When output is FALSE, pump system is in remote mode.

AUTO_MANUAL _MODE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output AUTO_MANUAL_MODE indicates the operation mode of the pump.

- When output is TRUE, pump is in automatic mode.
- When output is FALSE, pump is in manual mode.

OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output OPERATING indicates that the device is operating.

- When output is TRUE, device is operating.
- When output is FALSE, device is not operating.

READY_FOR_ AUTOMATION

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output READY_FOR_AUTOMATION of the pump interface function block indicates that the pump is ready for operation in automatic mode. This function is effective only when the function block is connected to the input READY_FOR_AUTOMATION of the following function blocks: PMP_SEQUENCE_GEN, PMP_FLOW_DISTRIBUTOR, PMP_LEVEL_DISTRIBUTOR and PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, pump is ready for operation in automatic mode.
- When output is FALSE, pump is not ready for operation in automatic mode.

START_ ANTIJAM

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output START_ANTIJAM starts antijam operation in manual mode. This function is effective only when the function block is connected to input START_MANUAL of function block PMP_ANTIJAM.

- When output is TRUE, antijam operation starts in manual mode.
- When output is FALSE, antijam operation stops if running.

DRIVE_RFG_ DISABLED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output DRIVE_RFG_DISABLED bypasses the ramp function generator (RFG) during Antijam operation. The output is effective only when connected to the control word (CW) of the drive.

- When output is TRUE, drive bypasses RFG during antijam operation.
- When output is FALSE, antijam operation runs without bypassing RFG.

Error codes

The error codes of function block PMP_INTERFACE_VFD are listed below:

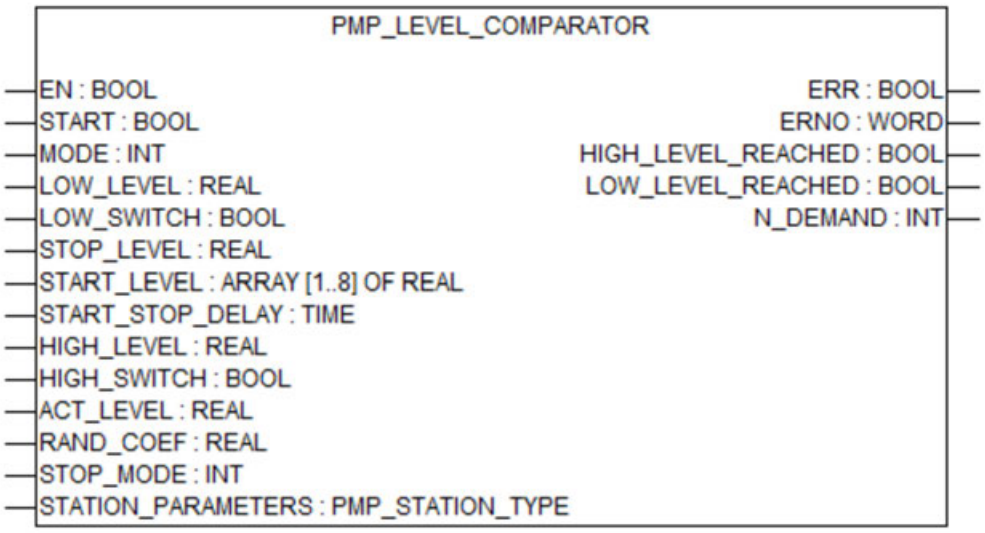
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value of PUMP_ID is less than 1
16418	4022	Value of PUMP_ID is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16561	40B1	Value of MASTER_PUMP is less than 1
16562	40B2	Value of MASTER_PUMP is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16609	40E1	Value of AUTO_SPEED_REF is less than 0
16641	4101	Value of MANUAL_SPEED is less than 0
16673	4121	Value of ANTIJAM_MODE is less than 0
16674	4122	Value of ANTIJAM_MODE is greater than 3
16689	4131	Value of ANTIJAM_STATUS is less than 0
16690	4132	Value of ANTIJAM_STATUS is greater than 2
16737	4161	Value of SOFTFILL_STATUS is less than 0
16738	4162	Value of SOFTFILL_STATUS is greater than 4
16769	4181	Value of SOFTFILL_SPEED is less than 0
16785	4191	Value of SLEEP_STATUS is less than 0
16786	4192	Value of SLEEP_STATUS is greater than 3
16627	40F3	Value of MANUAL_START and ALLOW_MANUAL_ANTIJAM is TRUE simultaneously. Either one operation is allowed at a time in manual mode, i.e., manual pump start or manual antijam.

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.4 PMP_LEVEL_COMPARATOR



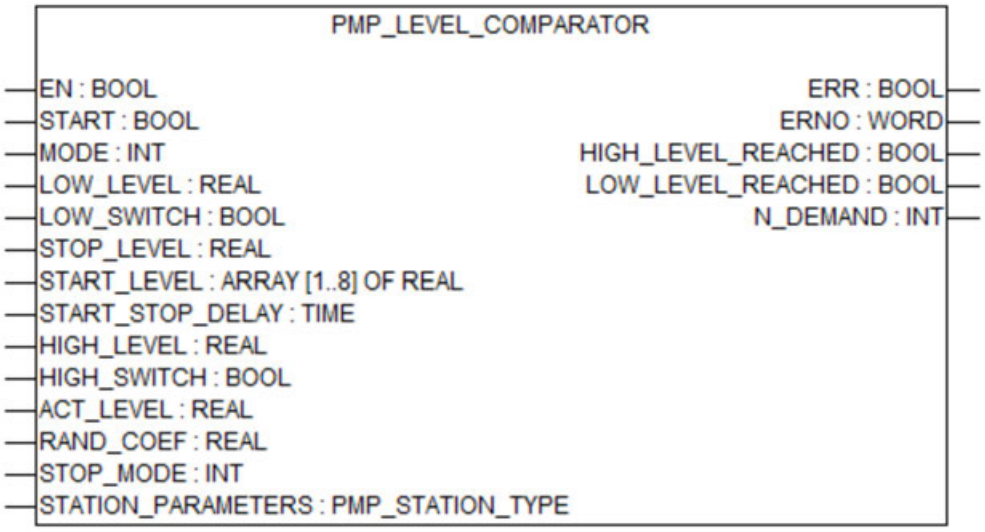
The function block PMP_LEVEL_COMPARATOR compares the actual level with the set level and generates a demand for the function block PMP_LEVEL_DISTRIBUTOR. The function block is used for tank filling or emptying operations.

> *The function block is used with function block PMP_LEVEL_DISTRIBUTOR, when the function block PMP_CONFIGURATION has the input PROCESS_MODE = 3 (Level control).*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input MODE selects the pumping operation mode.

1 = Emptying.

2 = Filling.

LOW_LEVEL

Data type	Default value	Range	Unit
REAL	10.0	0.0 ... 100.0	%

The input LOW_LEVEL defines the low level limit in percentage of full tank capacity.

LOW_SWITCH

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input LOW_SWITCH receives a digital signal from the pumping process when a low level has reached. The input is effective only when the sensor is present in the system, otherwise the default value is FALSE.

- When input is TRUE, pumping process has reached low level.
- When input is FALSE, pumping process is above low level.

STOP_LEVEL

Data type	Default value	Range	Unit
REAL	90.0	0.0 ... 100.0	%

The input STOP_LEVEL defines the stop level limit in percentage of full tank capacity.

START_LEVEL

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	0.0 ... 100.0	%

The input START_LEVEL defines the array of start level limits in percentage of full tank capacity.

START_STOP_DELAY

Data type	Default value	Range	Unit
TIME	10	-	s

The input START_STOP_DELAY defines the delay time to start or stop the pump when the respective start/stop levels have reached.

HIGH_LEVEL

Data type	Default value	Range	Unit
REAL	95.0	0.0 ... 100.0	%

The input HIGH_LEVEL defines the high level limit in percentage of full tank capacity.

HIGH_SWITCH

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input HIGH_SWITCH receives a digital signal from the pumping process when a high level has reached. The input is effective only when the sensor is present in the system, otherwise the default value is FALSE.

- When input is TRUE, pumping process has reached high level.
- When input is FALSE, pumping process is below high level.

ACT_LEVEL

Data type	Default value	Range	Unit
REAL	0.0	0.0 ... 100.0	%

The input ACT_LEVEL shows the actual level in percentage of full tank capacity. The value is read from the analog input level sensor.

Scaling of analog input is not in the scope of this library, but is done in the application program.

RAND_COEF

Data type	Default value	Range	Unit
REAL	0.0	-10.0 ... 10.0	%

The input RAND_COEF defines the random coefficient in percentage for randomizing the start level to avoid cake formation.

STOP_MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input STOP_MODE selects the appropriate stop mode.

Stop mode 1 = used in filling and emptying operation to stop each running pump.

- In filling operation, pump is stopped when input ACT_LEVEL > START_LEVEL.
- In emptying operation, pump is stopped when input ACT_LEVEL < START_LEVEL.

Stop mode 2 = common stop mode that stops all pumps with the level defined in input STOP_LEVEL.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

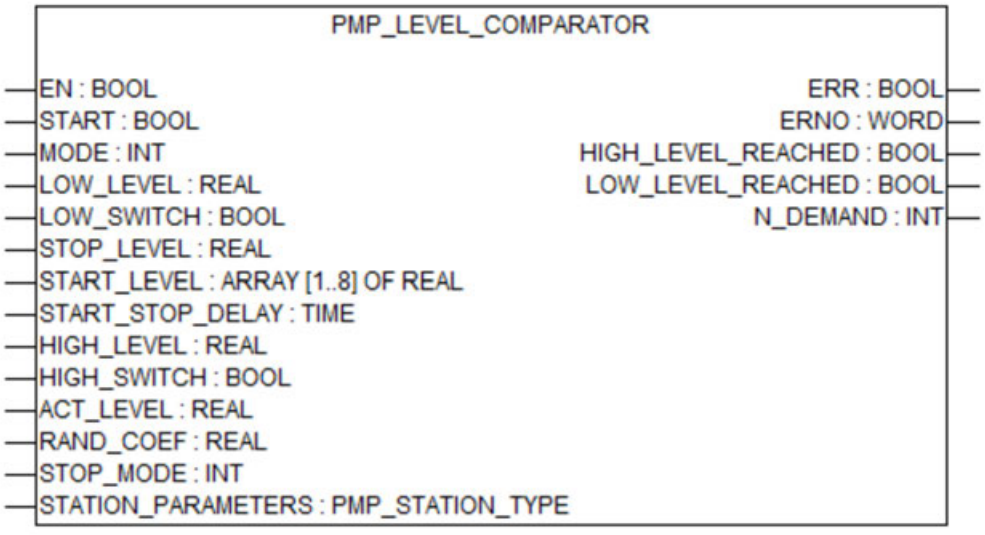
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

> *All function blocks will receive some data, process it and write it back to the structure.*

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combina-tion	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

HIGH_LEVEL_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output HIGH_LEVEL_REACHED of the function block PMP_LEVEL_COMPARATOR indicates that actual level is more than high level. This function is effective only when this output is connected to the input HIGH_LEVEL_REACHED of the function block PMP_LEVEL_DISTRIBUTOR.

- When output is TRUE, actual level is more than high level.
- When output is FALSE, actual level is less than high level.

LOW_LEVEL_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output LOW_LEVEL_REACHED of the function block PMP_LEVEL_ COMPARATOR indicates that actual level is less than low level. This function is effective only when this output is connected to the input LOW_LEVEL_REACHED of the function block PMP_LEVEL_ DISTRIBUTOR.

- When output is TRUE, actual level is less than low level.
- When output is FALSE, actual level is more than low level.

N_DEMAND

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input N_DEMAND of the function block PMP_LEVEL_DISTRIBUTOR receives the number of pumps demanded to run. This input comes from the output N_DEMAND of the function block PMP_LEVEL_COMPARATOR.

Error codes

The error codes of function block PMP_LEVEL_COMPARATOR are listed below:

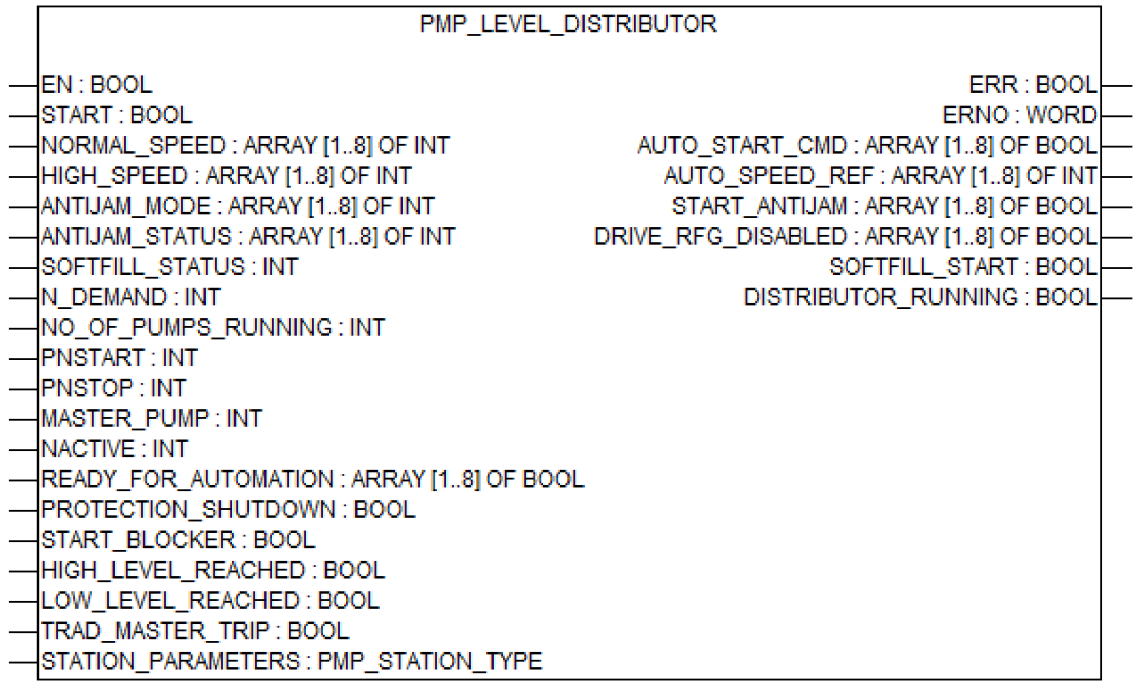
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16433	4031	Value of MODE is less than 1
16434	4032	Value of MODE is greater than 2
16449	4041	Value of LOW_LEVEL is less than 0.0
16450	4042	Value of LOW_LEVEL is greater than 100.0
16481	4061	Value of STOP_LEVEL is less than 0.0
16482	4062	Value of STOP_LEVEL is greater than 100.0
16496	4070	If MODE = 1 (Emptying), the array of START_LEVEL values must be in ascending order If MODE = 2 (Filling), the array of START_LEVEL values must be in descending order
16499	4073	If MODE = 1 (Emptying), the order of values must be LOW_LEVEL < STOP_LEVEL < START_LEVEL [1] ... [8] < HIGH_LEVEL If MODE = 2 (Filling), the order of values must be LOW_LEVEL < START_LEVEL [8] ... [1] < STOP_LEVEL < HIGH_LEVEL
16529	4091	Value of HIGH_LEVEL is less than 0.0
16530	4092	Value of HIGH_LEVEL is greater than 100.0
16577	40C1	Value of RAND_COEF is less than -10.0
16578	40C2	Value of RAND_COEF is greater than 10.0
16593	40D1	Value of STOP_MODE is less than 1
16594	40D2	Value of STOP_MODE is greater than 2

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.5 PMP_LEVEL_DISTRIBUTOR



The function block PMP_LEVEL_DISTRIBUTOR generates start commands and pump speed references based on the demand raised by PMP_LEVEL_COMPARATOR.

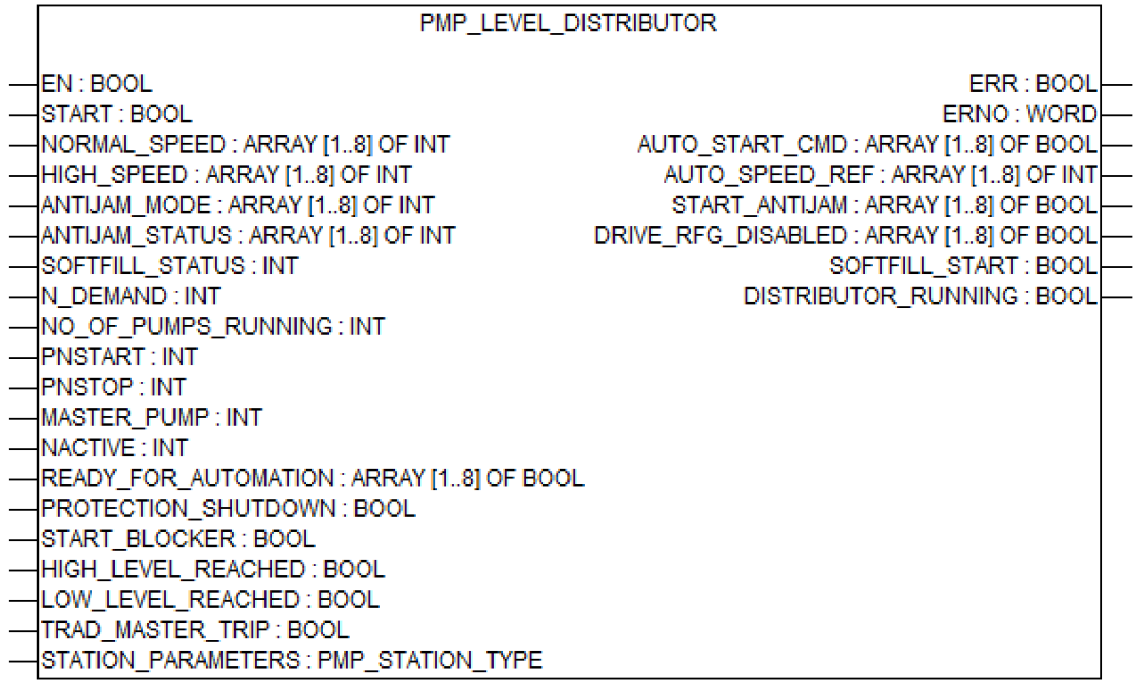
The function block receives the number of pumps in demand from the input N_DEMAND of function block PMP_LEVEL_COMPARATOR based on the process demand. The function block then sends pump ON/OFF command to function blocks PMP_INTERFACE_VFD or PMP_INTERFACE_DOL in the automatic mode of the pumping station. For the pumps run by variable frequency drive, the function block also sends speed references to function block PMP_INTERFACE_VFD.

> *This function block is used with function block PMP_LEVEL_COMPARATOR, when the function block PMP_CONFIGURATION has the input PROCESS_MODE = 3 (Level control).*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

NORMAL_SPEED

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	8 (1000)	> 0	rpm

The input NORMAL_SPEED defines the array of normal operating speed of all pumps for filling or emptying operations.

For e.g., the speed value in array index 1 points to PUMP_ID 1, and so on.

HIGH_SPEED

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	8 (1400)	> 0 ... NORMAL SPEED	rpm

The input HIGH_SPEED defines the array of speed values of all pumps for filling or emptying operations in the conditions below:

Filling – if pump level falls below the LOW_LEVEL

Emptying – if pump level rises above the HIGH_LEVEL.

For e.g., the speed value in array index 1 points to PUMP_ID 1, and so on.

ANTIJAM_ MODE

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE of the used pump distributor function block receives the antijam mode. This input comes from the output ANTIJAM_MODE of the function blocks PMP_ANTIJAM.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

N_DEMAND

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input N_DEMAND of the function block PMP_LEVEL_DISTRIBUTOR receives the number of pumps demanded to run. This input comes from the output N_DEMAND of the function block PMP_LEVEL_COMPARATOR.

NO_OF_PUMPS_RUNNING

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NO_OF_PUMPS_RUNNING receives the number of pumps running. The input comes from the output NO_OF_PUMPS_RUNNING of the function block PMP_SEQUENCE_GEN.

PNSTART

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTART receives the identification number of the next pump to start in the sequence. The input comes from the output PNSTART of the function block PMP_SEQUENCE_GEN.

PNSTOP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTOP receives the identification number of the next pump to stop in the sequence. The input comes from the output PNSTOP of the function block PMP_SEQUENCE_GEN.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

NACTIVE

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NACTIVE receives the number of pumps active for operation in automatic mode. The input comes from the output NACTIVE of the function block PMP_SEQUENCE_GEN.

READY_FOR_AUTOMATION

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	FALSE	TRUE/FALSE	-

The input READY_FOR_AUTOMATION receives an array of pumps ready for automatic operation. The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

The READY_FOR_AUTOMATION[1] connects to the PUMP ID = 1. This pattern must be followed for all pump IDs.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

PROTECTION_ SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_BLOCKER receives a digital signal to prevent the starting of pump station for protection. The input comes from the output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

The input START_BLOCKER will not stop the already running pump.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

HIGH_LEVEL_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output HIGH_LEVEL_REACHED of the function block PMP_LEVEL_COMPARATOR indicates that actual level is more than high level. This function is effective only when this output is connected to the input HIGH_LEVEL_REACHED of the function block PMP_LEVEL_DISTRIBUTOR.

- When output is TRUE, actual level is more than high level.
- When output is FALSE, actual level is less than high level.

LOW_LEVEL_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output LOW_LEVEL_REACHED of the function block PMP_LEVEL_ COMPARATOR indicates that actual level is less than low level. This function is effective only when this output is connected to the input LOW_LEVEL_REACHED of the function block PMP_LEVEL_ DISTRIBUTOR.

- When output is TRUE, actual level is less than low level.
- When output is FALSE, actual level is more than low level.

TRAD_MASTER _TRIP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Indicates that the master pump in the traditional pump combination has tripped. This input comes from the output TRAD_MASTER_TRIP of the function block PMP_SEQUENCE_GEN.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

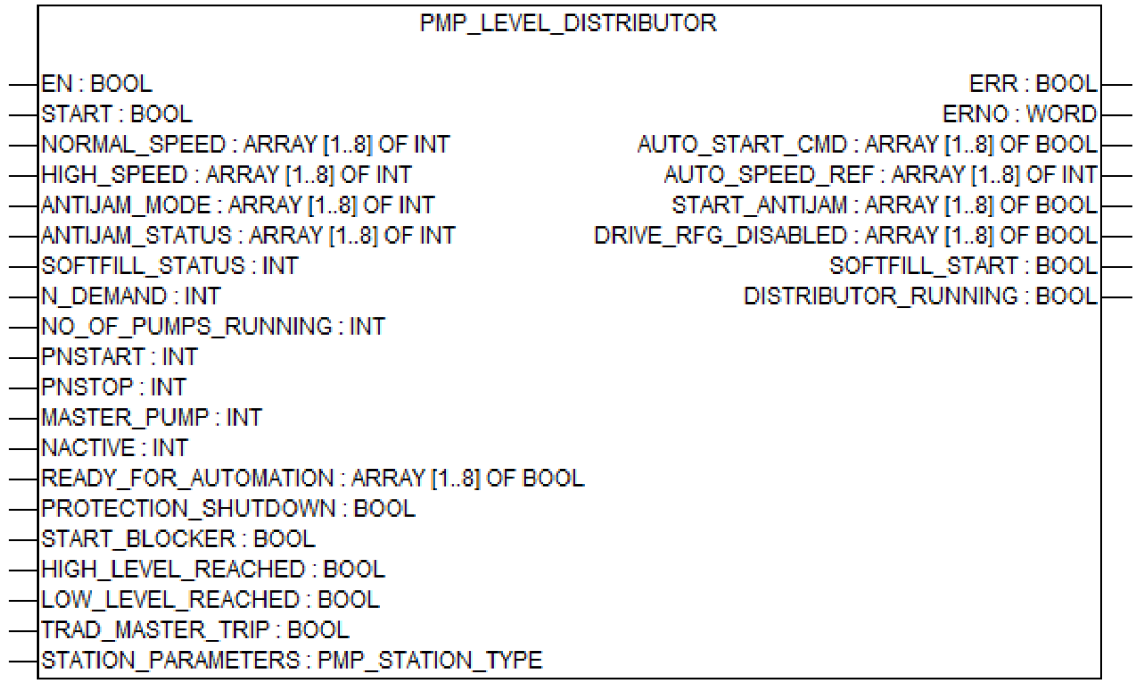
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

AUTO_START_CMD

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output AUTO_START_CMD sends the automatic start command to the input AUTO_START_CMD of interface function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL for the required pump ID.

The output AUTO_START_CMD[1] must be connected to the input AUTO_START_CMD of function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL with PUMP ID = 1. This pattern must be followed for all pump IDs.

AUTO_SPEED_REF

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	-	-

The output AUTO_SPEED_REF sends the automatic speed reference value in rpm to the input PUMP_AUTO_SPEED_REF of interface function block PMP_INTERFACE_VFD of the required pump ID.

The output AUTO_SPEED_REF[1] must be connected to the input AUTO_SPEED_REF of function block PMP_INTERFACE_VFD with PUMP ID = 1. This pattern must be followed for all pump IDs.

START_ANTIJAM

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output START_ANTIJAM starts the antijam operation in automatic mode. The operation is effective only when the output is connected to the input START_AUTO of the function block PMP_ANTIJAM of the required pump ID.

The output START_ANTIJAM[1] must be connected to the input START_AUTO of function block PMP_ANTIJAM with PUMP ID = 1. This pattern must be followed for all pump IDs.

DRIVE_RFG_DISABLED

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output DRIVE_RFG_DISABLED bypasses the ramp function generator (RFG) during antijam operation. The output is effective only if the function block is connected to the control word (CW) of the drive.

SOFTFILL_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_START of the used pump distributor starts the soft filling operation. This function is effective only when this output is connected to the input START of the function block PMP_SOFT_FILLING.

- When input is TRUE, softfill function starts.
- When input is FALSE, softfill function stops if running.

DISTRIBUTOR_RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DISTRIBUTOR_RUNNING indicates the running status of the distributor.
This input is connected to the output DISTRIBUTOR_RUNNING of any of the the distributor function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, distributor is running.
- When output is FALSE, distributor is not running.

Error codes

The error codes of function block PMP_LEVEL_DISTRIBUTOR are listed below:

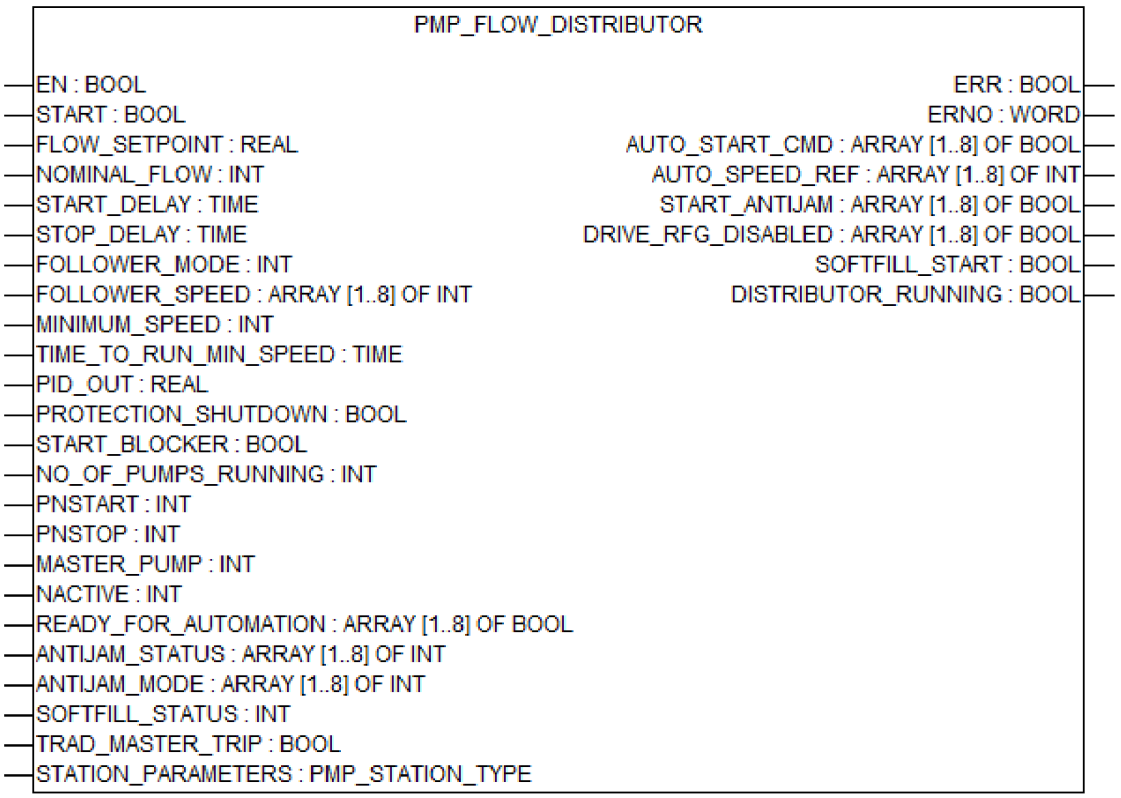
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16400	4010	Flow control can run only when the PROCESS_MODE in PMP_CONFIGURATION = 2
16433	4031	Value of FLOW_SET_POINT is less than 0
16449	4041	Value in NOMINAL_FLOW is less than or equal to 0
16450	4042	Value of HIGH_SPEED is more than NORMAL_SPEED
16465	4051	Value of ANTIJAM_MODE is less than 0
16466	4052	Value of ANTIJAM_MODE is greater than 3
16481	4061	Value of ANTIJAM_STATUS is less than 0
16482	4062	Value of ANTIJAM_STATUS is greater than 2
16497	4071	Value of SOFTFILL_STATUS is less than 0
16498	4072	Value of SOFTFILL_STATUS is greater than 3
16513	4081	Value of N_DEMAND is less than 1
16514	4082	Value of N_DEMAND is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16529	4091	Value of NO_OF_PUMPS_RUNNING is less than 0
16530	4092	Value of NO_OF_PUMPS_RUNNING is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16545	40A1	Value of PNSTART is less than 1
16546	40A2	Value of PNSTART is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16561	40B1	Value of PNSTOP is less than 1
16562	40B2	Value of PNSTOP is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16577	40C1	Value of MASTER_PUMP is less than 1
16578	40C2	Value of MASTER_PUMP is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16593	40D1	Value of NACTIVE is less than 1
16594	40D2	Value of NACTIVE is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.6 PMP_FLOW_DISTRIBUTOR



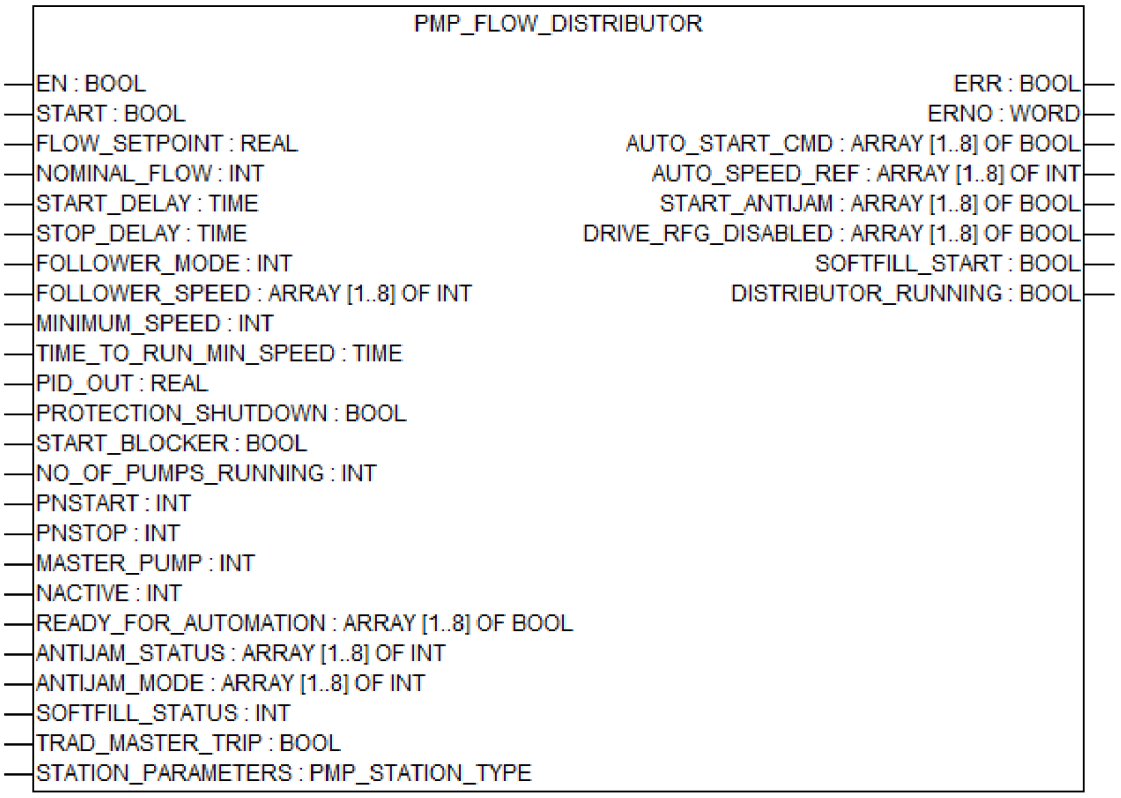
The function block PMP_FLOW_DISTRIBUTOR is used to maintain the flow in the network. The function block sends the pump ON/OFF command to the function blocks PMP_INTERFACE_VFD or PMP_INTERFACE_DOL in the automatic mode of the pumping station. For the pumps run by variable frequency drive, the function block also sends speed references to function block PMP_INTERFACE_VFD.

This function block is effective only when the input PROCESS_MODE = 2 (Flow control) in the function block PMP_CONFIGURATION.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

FLOW_SETPOINT

Data type	Default value	Range	Unit
REAL	0.0	-	m^3/h

The input FLOW_SETPOINT defines the flow setpoint in cubic meters per hour.

NOMINAL_FLOW

Data type	Default value	Range	Unit
INT	100	-	m^3/h

The input NOMINAL_FLOW defines the nominal flow in cubic meters per hour for each pump. The input is effective in pumps of the same capacity.

START_DELAY

Data type	Default value	Range	Unit
TIME	0	-	s

The input START_DELAY defines the delay time to start the follower pumps.

STOP_DELAY

Data type	Default value	Range	Unit
TIME	0	-	s

The input STOP_DELAY defines the delay time to stop the follower pumps.

FOLLOWER_MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input FOLLOWER_MODE selects the follower mode of the follower pumps.

1 = Copy master mode to run at the master pump speed

2 = Fixed speed mode to run at its own speed

FOLLOWER_SPEED

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	> 0	rpm

The input FOLLOWER_SPEED defines the follower speed in rpm for each pump when the input FOLLOWER_MODE = 2 (Fixed speed mode to run at its own speed).

MINIMUM_SPEED

Data type	Default value	Range	Unit
INT	100	> 0	rpm

The input MINIMUM_SPEED defines the minimum speed in rpm at which the pump should run to maintain the minimum flow.

TIME_TO_RUN_MIN_SPEED

Data type	Default value	Range	Unit
TIME	5	-	s

The input TIME_TO_RUN_MIN_SPEED defines the time limit at which the pump should run at the value in the input MINIMUM_SPEED. After this time duration has elapsed the PID control take over the control of flow or pressure and the pumps start running based on the PID output and the follower mode speed.

PID_OUT

Data type	Default value	Range	Unit
REAL	0.0	-	%

The input PID_OUT of the function block PMP_SLEEP indicates the PID output in percentage. The value is the same as the output PID_OUT of the function blocks PMP_PID.

PROTECTION_ SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_BLOCKER receives a digital signal to prevent the starting of pump station for protection. The input comes from the output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

The input START_BLOCKER will not stop the already running pump.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

NO_OF_PUMPS _RUNNING

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NO_OF_PUMPS_RUNNING receives the number of pumps running. The input comes from the output NO_OF_PUMPS_RUNNING of the function block PMP_SEQUENCE_GEN.

PNSTART

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTART receives the identification number of the next pump to start in the sequence. The input comes from the output PNSTART of the function block PMP_SEQUENCE_GEN.

PNSTOP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTOP receives the identification number of the next pump to stop in the sequence. The input comes from the output PNSTOP of the function block PMP_SEQUENCE_GEN.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

NACTIVE

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NACTIVE receives the number of pumps active for operation in automatic mode. The input comes from the output NACTIVE of the function block PMP_SEQUENCE_GEN.

READY_FOR_ AUTOMATION

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	FALSE	TRUE/FALSE	-

The input READY_FOR_AUTOMATION receives an array of pumps ready for automatic operation. The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

The READY_FOR_AUTOMATION[1] connects to the PUMP ID = 1. This pattern must be followed for all pump IDs.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ MODE

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE of the used pump distributor function block receives the antijam mode. This input comes from the output ANTIJAM_MODE of the function blocks PMP_ANTIJAM.

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

TRAD_MASTER _TRIP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Indicates that the master pump in the traditional pump combination has tripped. This input comes from the output TRAD_MASTER_TRIP of the function block PMP_SEQUENCE_GEN.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

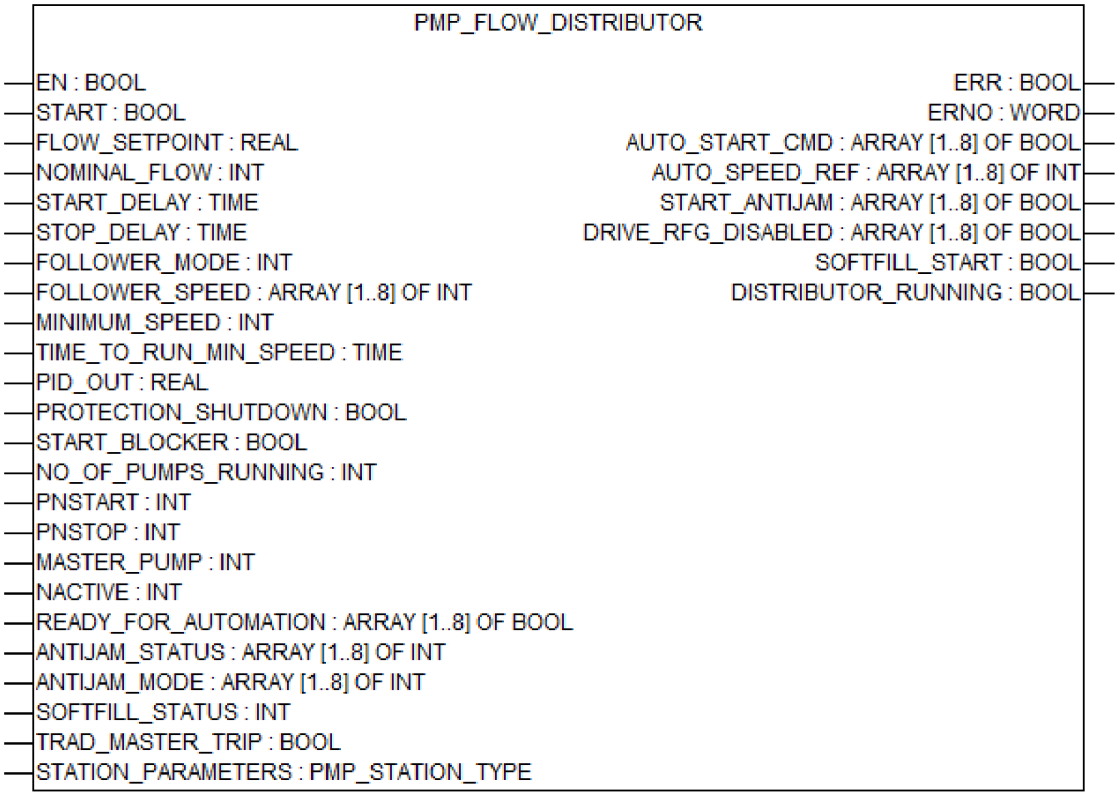
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combina-tion	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

AUTO_START_CMD

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output AUTO_START_CMD sends the automatic start command to the input AUTO_START_CMD of interface function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL for the required pump ID.

> *The output AUTO_START_CMD[1] must be connected to the input AUTO_START_CMD of function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL with PUMP ID = 1. This pattern must be followed for all pump IDs.*

AUTO_SPEED_REF

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	-	-

The output AUTO_SPEED_REF sends the automatic speed reference value in rpm to the input PUMP_AUTO_SPEED_REF of interface function block PMP_INTERFACE_VFD of the required pump ID.

The output AUTO_SPEED_REF[1] must be connected to the input AUTO_SPEED_REF of function block PMP_INTERFACE_VFD with PUMP ID = 1. This pattern must be followed for all pump IDs.

START_ANTIJAM

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output START_ANTIJAM starts the antijam operation in automatic mode. The operation is effective only when the output is connected to the input START_AUTO of the function block PMP_ANTIJAM of the required pump ID.

The output START_ANTIJAM[1] must be connected to the input START_AUTO of function block PMP_ANTIJAM with PUMP ID = 1. This pattern must be followed for all pump IDs.

DRIVE_RFG_DISABLED

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output DRIVE_RFG_DISABLED bypasses the ramp function generator (RFG) during antijam operation. The output is effective only if the function block is connected to the control word (CW) of the drive.

SOFTFILL_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_START of the used pump distributor starts the soft filling operation. This function is effective only when this output is connected to the input START of the function block PMP_SOFT_FILLING.

- When input is TRUE, softfill function starts.
- When input is FALSE, softfill function stops if running.

DISTRIBUTOR_RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DISTRIBUTOR_RUNNING indicates the running status of the distributor.
This input is connected to the output DISTRIBUTOR_RUNNING of any of the the distributor function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, distributor is running.
- When output is FALSE, distributor is not running.

Error codes

The error codes of function block PMP_FLOW_DISTRIBUTOR are listed below:

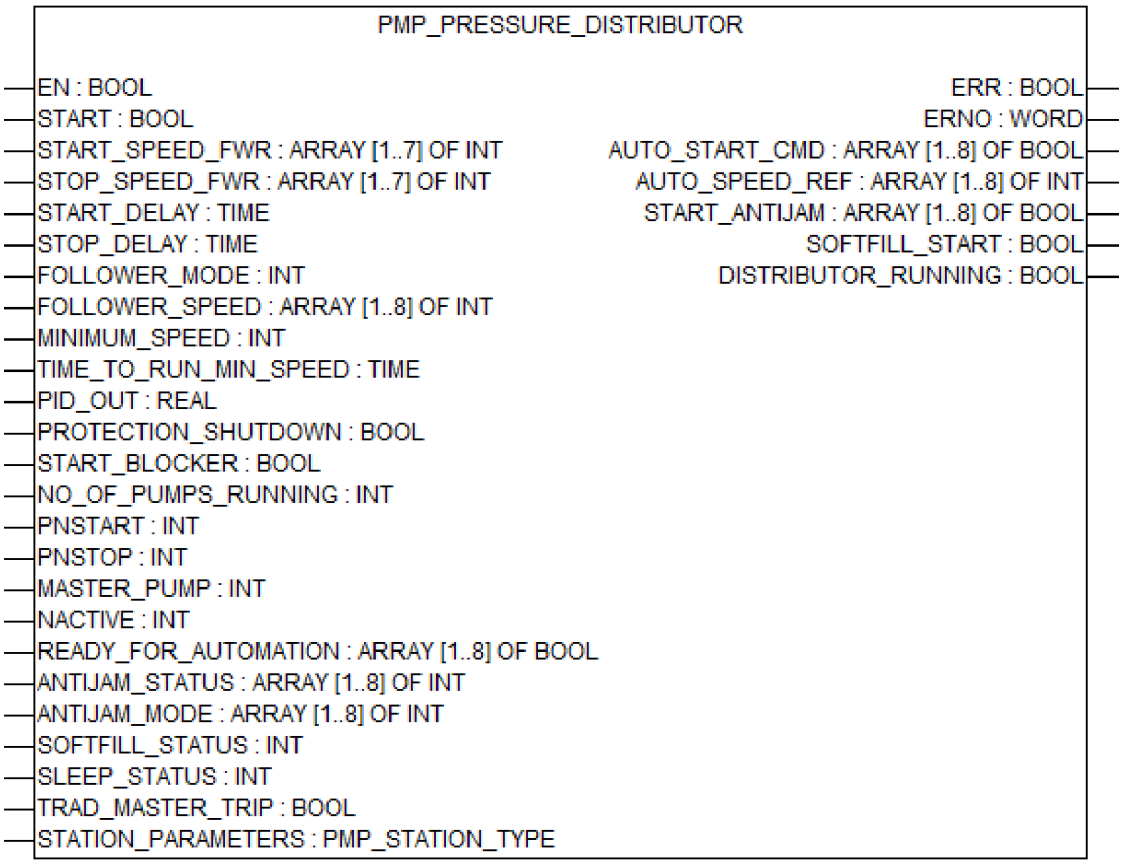
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16400	4010	Flow control can run only when the PROCESS_MODE in PMP_CONFIGURATION = 2
16433	4031	Value of FLOW_SET_POINT is less than 0
16449	4041	Value in NOMINAL_FLOW is less than or equal to 0
16497	4071	Value in FOLLOWER_MODE is less than 1
16498	4072	Value in FOLLOWER_MODE is greater than 2
16512	4080	Array in FOLLOWER_SPEED is less than or equal to 0
16529	4091	Value in MINIMUM_SPEED is less than 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.7 PMP_PRESSURE_DISTRIBUTOR



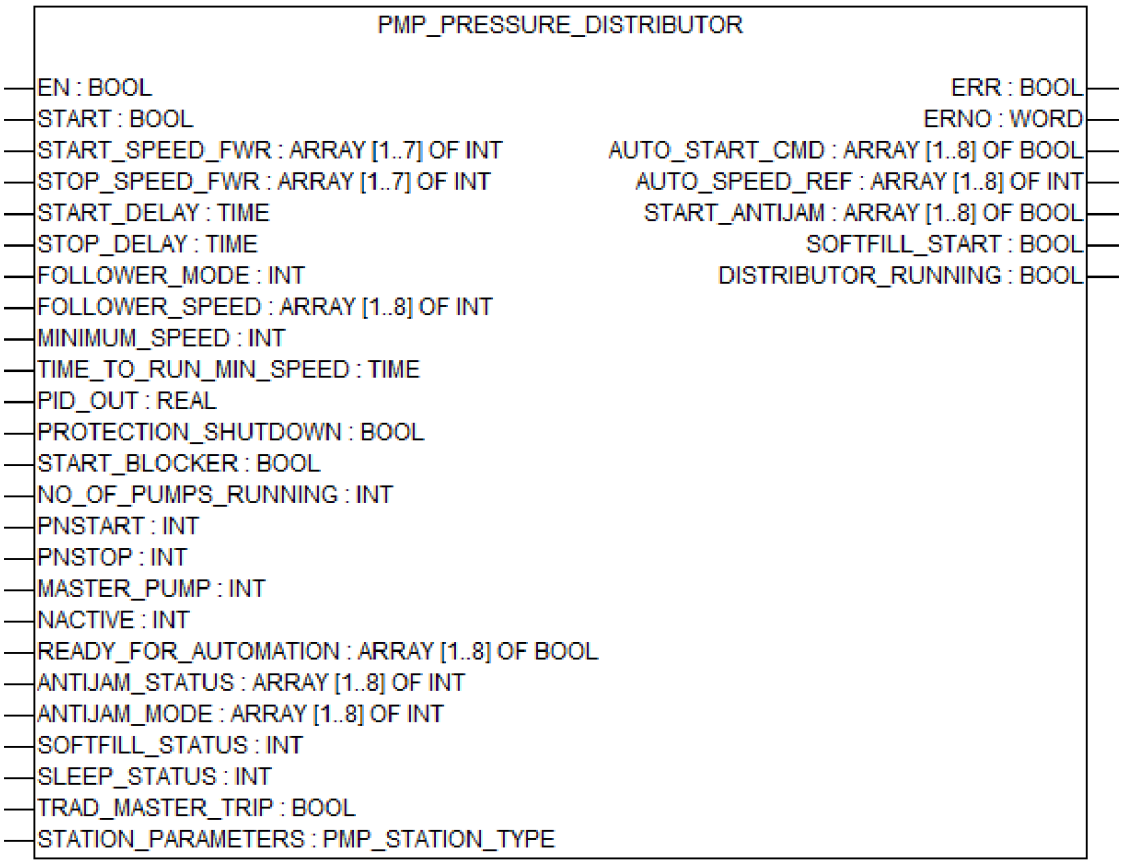
The function block PMP_PRESSURE_DISTRIBUTOR is used to stabilize pressure in the network. The function block is used together with function block PMP_PID to stabilize pressure in the network. The function block then sends pump ON/OFF command to the function blocks PMP_INTERFACE_VFD or PMP_INTERFACE_DOL in the automatic mode of the pumping station. For pumps run by variable frequency drive, the function block also sends speed references to function block PMP_INTERFACE_VFD.

> *This function block is effective only when the input PROCESS_MODE = 1 (Pressure control) in function block PMP_CONFIGURATION.*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

START_SPEED_FWR

Data type	Default value	Range	Unit
ARRAY[1..7] of INT	-	-	rpm

The input START_SPEED_FWR defines the start speed in rpm for the seven follower pumps.

STOP_SPEED_FWR

Data type	Default value	Range	Unit
ARRAY[1..7] of INT	-	-	rpm

The input STOP_SPEED_FWR defines the stop speed in rpm for the seven follower pumps.

START_DELAY

Data type	Default value	Range	Unit
TIME	0	-	s

The input START_DELAY defines the delay time to start the follower pumps.

STOP_DELAY

Data type	Default value	Range	Unit
TIME	0	-	s

The input STOP_DELAY defines the delay time to stop the follower pumps.

FOLLOWER_MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input FOLLOWER_MODE selects the follower mode of the follower pumps.

1 = Copy master mode to run at the master pump speed

2 = Fixed speed mode to run at its own speed

FOLLOWER_SPEED

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	> 0	rpm

The input FOLLOWER_SPEED defines the follower speed in rpm for each pump when the input FOLLOWER_MODE = 2 (Fixed speed mode to run at its own speed).

MINIMUM_SPEED

Data type	Default value	Range	Unit
INT	100	> 0	rpm

The input MINIMUM_SPEED defines the minimum speed in rpm at which the pump should run to maintain the minimum flow.

TIME_TO_RUN_MIN_SPEED

Data type	Default value	Range	Unit
TIME	5	-	s

The input TIME_TO_RUN_MIN_SPEED defines the time limit at which the pump should run at the value in the input MINIMUM_SPEED. After this time duration has elapsed the PID control take over the control of flow or pressure and the pumps start running based on the PID output and the follower mode speed.

PID_OUT

Data type	Default value	Range	Unit
REAL	0.0	-	%

The input PID_OUT of the function block PMP_SLEEP indicates the PID output in percentage. The value is the same as the output PID_OUT of the function blocks PMP_PID.

PROTECTION_ SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_BLOCKER receives a digital signal to prevent the starting of pump station for protection. The input comes from the output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

The input START_BLOCKER will not stop the already running pump.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

NO_OF_PUMPS _RUNNING

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NO_OF_PUMPS_RUNNING receives the number of pumps running. The input comes from the output NO_OF_PUMPS_RUNNING of the function block PMP_SEQUENCE_GEN.

PNSTART

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTART receives the identification number of the next pump to start in the sequence. The input comes from the output PNSTART of the function block PMP_SEQUENCE_GEN.

PNSTOP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTOP receives the identification number of the next pump to stop in the sequence. The input comes from the output PNSTOP of the function block PMP_SEQUENCE_GEN.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

NACTIVE

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NACTIVE receives the number of pumps active for operation in automatic mode. The input comes from the output NACTIVE of the function block PMP_SEQUENCE_GEN.

READY_FOR_ AUTOMATION

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	FALSE	TRUE/FALSE	-

The input READY_FOR_AUTOMATION receives an array of pumps ready for automatic operation. The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

The READY_FOR_AUTOMATION[1] connects to the PUMP ID = 1. This pattern must be followed for all pump IDs.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ MODE

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE of the used pump distributor function block receives the antijam mode. This input comes from the output ANTIJAM_MODE of the function blocks PMP_ANTIJAM.

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

SLEEP_STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The output SLEEP_STATUS of the function block PMP_SLEEP indicates the sleep status of the pump.

0 = Inactive/Wakeup inactive

1 = Boost activated

2 = Sleep mode active

3 = Wakeup function active

TRAD_MASTER _TRIP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Indicates that the master pump in the traditional pump combination has tripped. This input comes from the output TRAD_MASTER_TRIP of the function block PMP_SEQUENCE_GEN.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

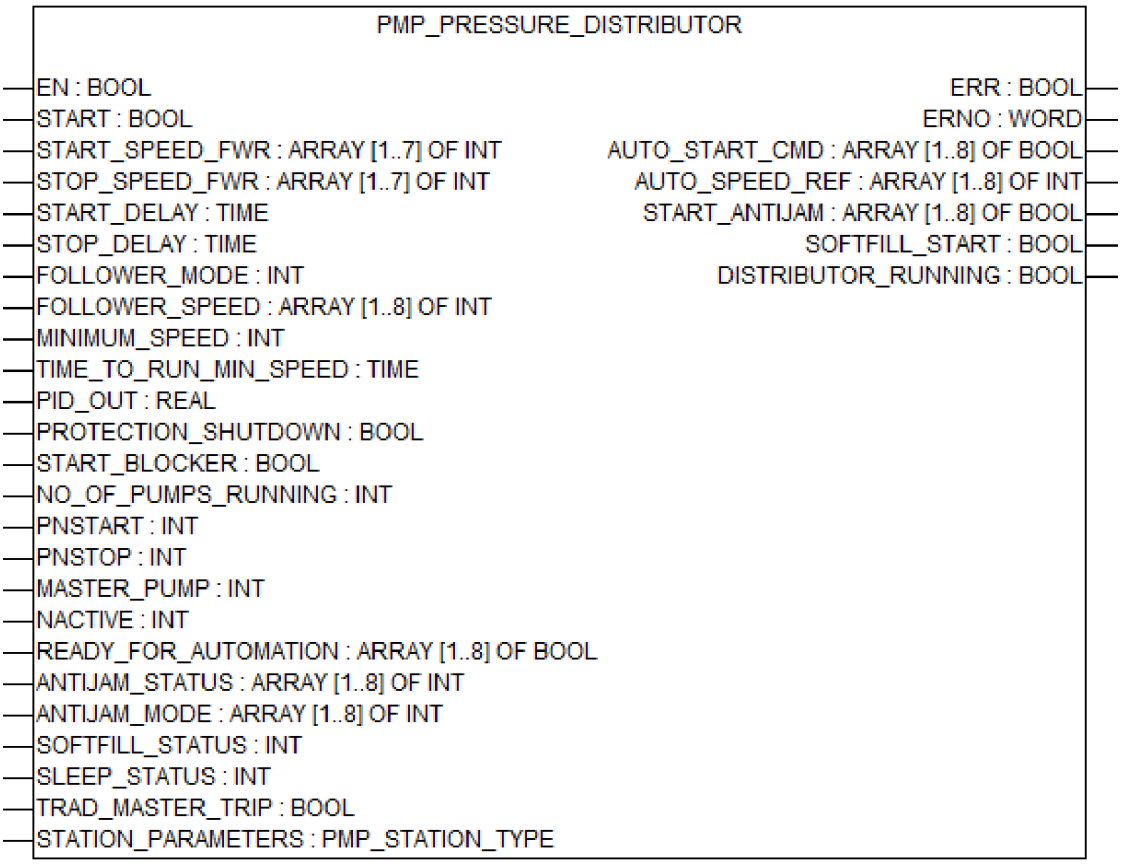
All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station

Parameter	Data type	Default value	Description
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combina-tion	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

AUTO_START_CMD

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output AUTO_START_CMD sends the automatic start command to the input AUTO_START_CMD of interface function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL for the required pump ID.

The output AUTO_START_CMD[1] must be connected to the input AUTO_START_CMD of function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL with PUMP ID = 1. This pattern must be followed for all pump IDs.

AUTO_SPEED_REF

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	-	-

The output AUTO_SPEED_REF sends the automatic speed reference value in rpm to the input PUMP_AUTO_SPEED_REF of interface function block PMP_INTERFACE_VFD of the required pump ID.

The output AUTO_SPEED_REF[1] must be connected to the input AUTO_SPEED_REF of function block PMP_INTERFACE_VFD with PUMP ID = 1. This pattern must be followed for all pump IDs.

START_ANTIJAM

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The output START_ANTIJAM starts the antijam operation in automatic mode. The operation is effective only when the output is connected to the input START_AUTO of the function block PMP_ANTIJAM of the required pump ID.

The output START_ANTIJAM[1] must be connected to the input START_AUTO of function block PMP_ANTIJAM with PUMP ID = 1. This pattern must be followed for all pump IDs.

SOFTFILL_START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_START of the used pump distributor starts the soft filling operation. This function is effective only when this output is connected to the input START of the function block PMP_SOFT_FILLING.

- When input is TRUE, softfill function starts.
- When input is FALSE, softfill function stops if running.

DISTRIBUTOR_RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DISTRIBUTOR_RUNNING indicates the running status of the distributor. This input is connected to the output DISTRIBUTOR_RUNNING of any of the the distributor function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, distributor is running.
- When output is FALSE, distributor is not running.

Error codes

The error codes of function block PMP_FLOW_DISTRIBUTOR are listed below:

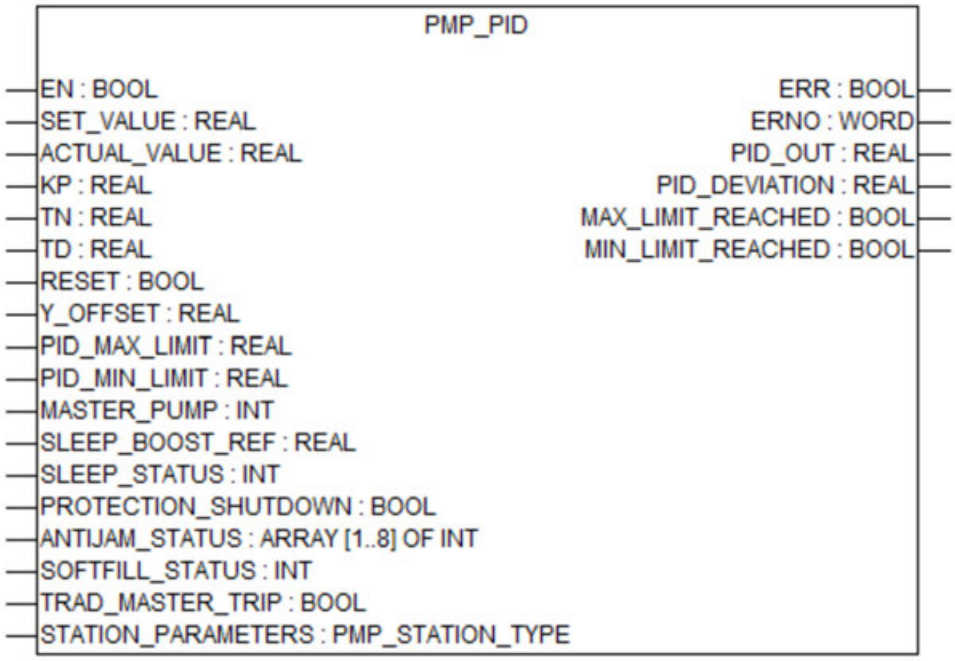
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16400	4010	Flow control can run only when the PROCESS_MODE in PMP_CONFIGURATION = 2
16432	4030	Value in START_SPEED_SLV is not in ascending order
16448	4040	Value in STOP_SPEED_SLV is not in ascending order
16451	4043	Value in START_SPEED_SLV[1] is not greater than STOP_SPEED_SLV[1] and so on for the rest of the pumps
16497	4071	Value in FOLLOWER_MODE is less than 1
16498	4072	Value in FOLLOWER_MODE is greater than 2
16512	4080	Array in FOLLOWER_SPEED is not in ascending order
16529	4091	Value in MINIMUM_SPEED is less than the minimum limit

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.8 PMP_PID

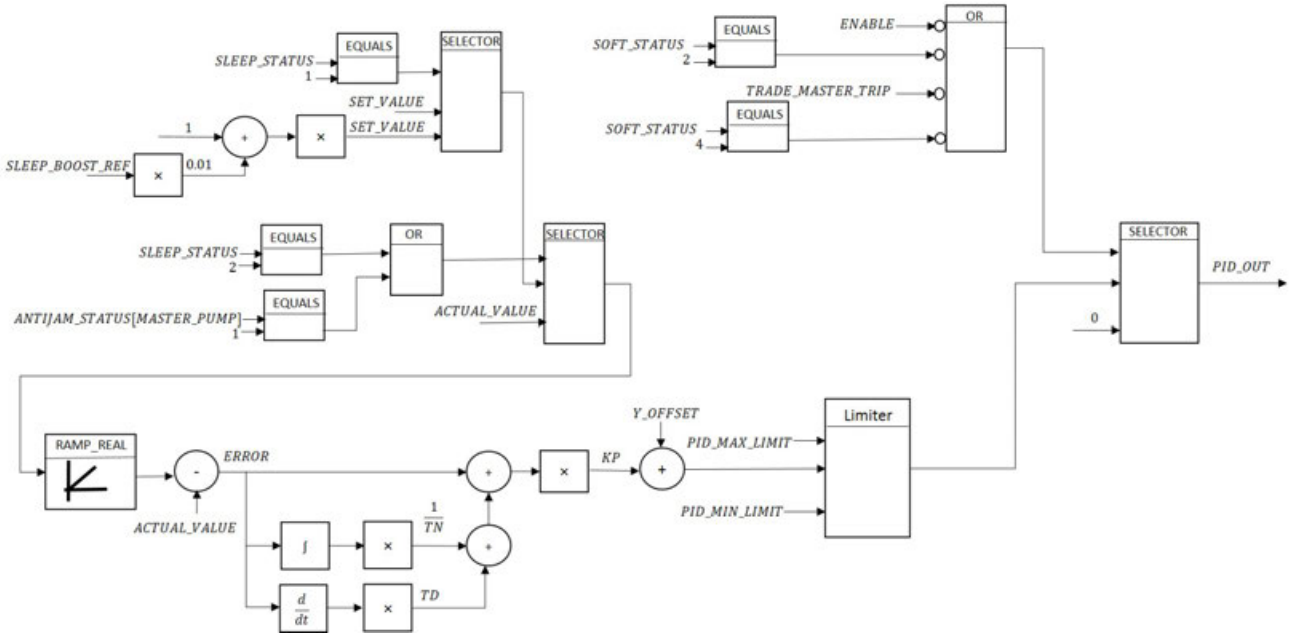


The function block PMP_PID is used for closed loop control of pressure or flow in the network.

The function block receives the set value and actual value from the process and does the closed loop control. The function block is used to control pressure or flow when the input PROCESS_MODE of function block PMP_CONFIGURATION is selected as either pressure control (1) or flow control (2) in the multi pumping and traditional pumping station.

> *This function is not used on DOL pumping stations.*

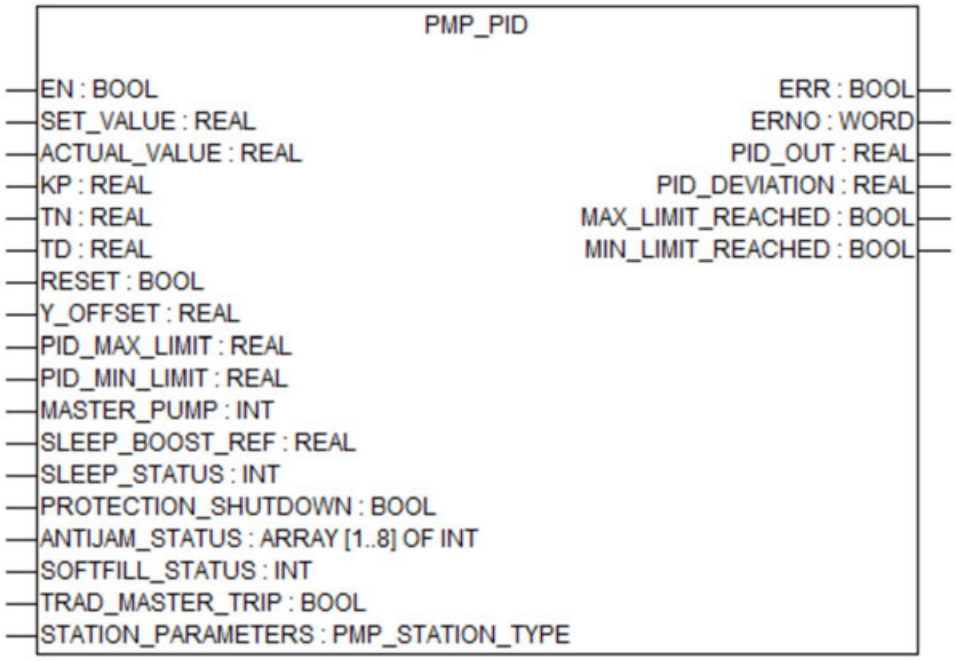
Functionality The functionality of the PMP_PID is explained in the figure below.



General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SET_VALUE

Data type	Default value	Range	Unit
REAL	1.0	> 0.0	Pressure: bar Flow: m^3/h

The input SET_VALUE defines the set value of process variable in real values, e.g.,

bar for pressure,

cubic meters per hour for flow.

ACTUAL_ VALUE

Data type	Default value	Range	Unit
REAL	0.0	> 0.0	Pressure: bar Flow: m^3/h

The input ACTUAL_VALUE defines the actual value of process variable in real values. The unit is same as the unit in the input SET_VALUE.

KP

Data type	Default value	Range	Unit
REAL	0.1	> 0.0	

The input KP defines the proportional gain of the PID.

TN

Data type	Default value	Range	Unit
REAL	1.0	> 0.0	s

The input TN defines the integral time of the PID in seconds.

TD

Data type	Default value	Range	Unit
REAL	0.0	> 0.0	s

The input TD defines the derivative time of the PID in seconds.

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET resets the output of the PID and works as AntiWind Up.

- When input is TRUE, PID output is reset.
- When input is FALSE, PID output is not reset.

Y_OFFSET

Data type	Default value	Range	Unit
REAL	0.0	0.0 ... 200.0	-

The input Y_OFFSET adds the offset to the PID output.

PID_MAX_LIMIT

Data type	Default value	Range	Unit
REAL	100.0	0.0 ... 200.0	-

The input PID_MAX_LIMIT defines the maximum limit of the PID output.

PID_MIN_LIMIT

Data type	Default value	Range	Unit
REAL	0.0	0.0 ... 200.0	-

The input PID_MIN_LIMIT defines the minimum limit of the PID output.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

SLEEP_ BOOST_REF

Data type	Default value	Range	Unit
REAL	0.0	> 0.0	-

The output SLEEP_BOOST_REF of the function block PMP_SLEEP indicates the sleep boost reference. This output is connected as input into PMP_PID function block.

SLEEP_STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The output SLEEP_STATUS of the function block PMP_SLEEP indicates the sleep status of the pump.

0 = Inactive/Wakeup inactive

1 = Boost activated

2 = Sleep mode active

3 = Wakeup function active

PROTECTION_ SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PROTECTION_SHUTDOWN receives a digital signal to shut down the pump station for protection. The input comes from the output PROTECTION_SHUTDOWN of the protection function blocks PMP_PROTECTION_ANALOG and PMP_PROTECTION_BINARY.

- When input is TRUE, pump station is shutdown.
- When input is FALSE, pump station operation continues.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

TRAD_MASTER _TRIP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Indicates that the master pump in the traditional pump combination has tripped. This input comes from the output TRAD_MASTER_TRIP of the function block PMP_SEQUENCE_GEN.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

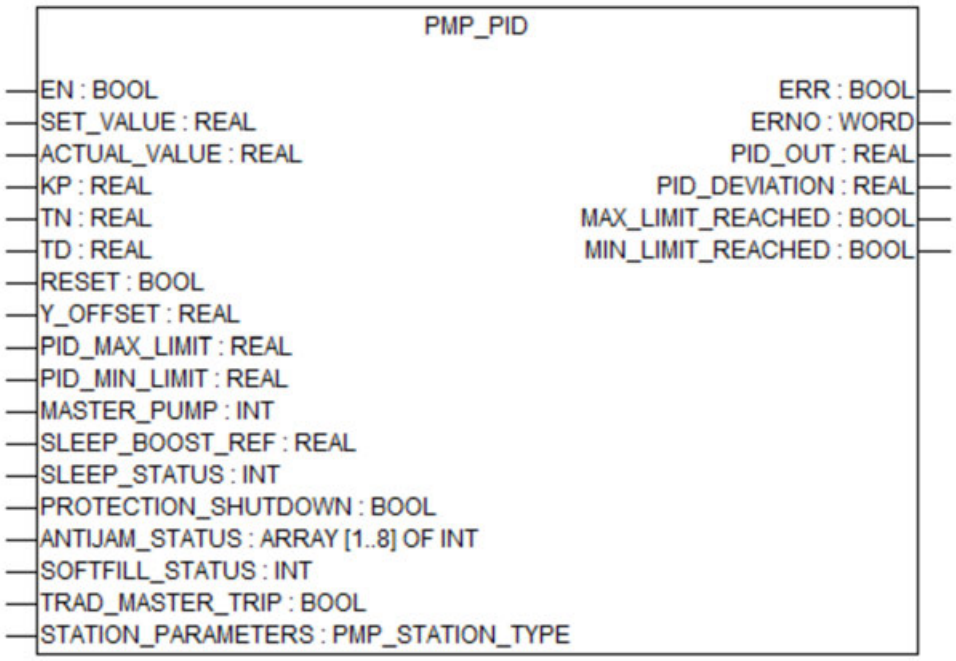
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combina-tion	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

PID_OUT

Data type	Default value	Range	Unit
REAL	0.0	-	%

The input PID_OUT of the function block PMP_SLEEP indicates the PID output in percentage. The value is the same as the output PID_OUT of the function blocks PMP_PID.

PID_DEVIATION

Data type	Default value	Range	Unit
REAL	0.0	-	%

The output PID_DEVIATION indicates PID error which is deviation of set value from actual value.

MAX_LIMIT_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output MAX_LIMIT_REACHED indicates that the PID output has reached the maximum limit.

- When output is TRUE, PID output has reached maximum limit.
- When output is FALSE, PID output is below the maximum limit.

MIN_LIMIT_ REACHED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output MIN_LIMIT_REACHED indicates that the PID output has reached the minimum limit.

- When output is TRUE, PID output has reached minimum limit.
- When output is FALSE, PID output is above the minimum limit.

Error codes

The error codes of function block PMP_PID are listed below:

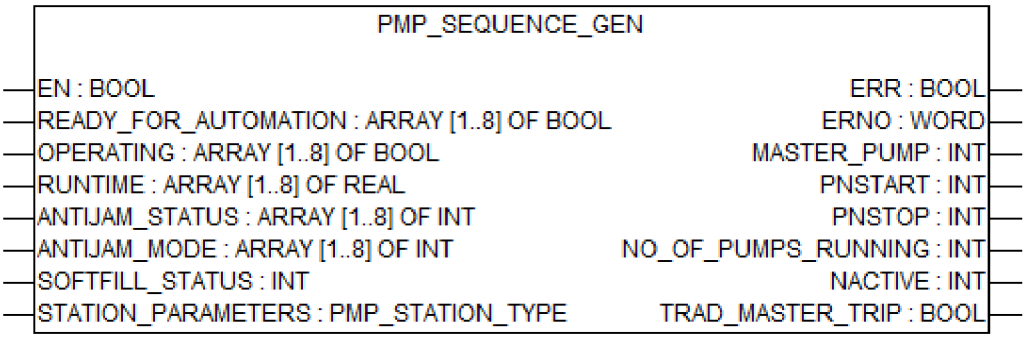
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value in input SET_VALUE is less than 0.0
16433	4031	Value in input ACTUAL_VALUE is less than 0.0
16449	4041	Value in input KP is less than 0.0
16465	4051	Value in input TN is less than 0.0
16481	4061	Value in input TD is less than 0.0
16513	4081	Value in input Y_OFFSET is less than 0.0
16514	4082	Value in input Y_OFFSET is more than PID_MAX_LIMIT.
16529	4091	Value in input PID_MAX_LIMIT is less than PID_MIN_LIMIT
16530	4092	Value in input PID_MAX_LIMIT is greater than 200.0
16545	40A1	Value in input PID_MIN_LIMIT is less than 0.0
16577	40C1	Value in input SLEEP_BOOST_REF is less than 0.0
16593	40D1	Value in input SLEEP_STATUS is less than 0
16594	40D2	Value in input SLEEP_STATUS is greater than 3
16625	40F1	Value in input ANTIJAM_STATUS is less than 0
16626	40F2	Value in input ANTIJAM_STATUS is greater than 2
16641	4101	Value in input SOFTFILL_STATUS is less than 0
16642	4102	Value in input SOFTFILL_STATUS is greater than 4

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.9 PMP_SEQUENCE_GEN



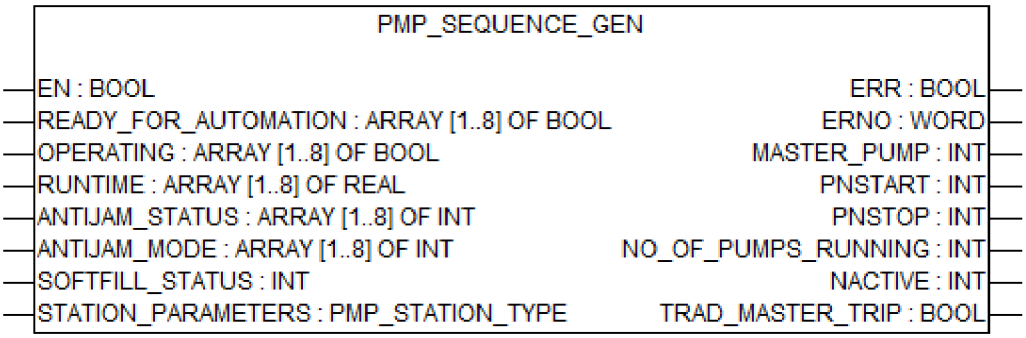
The function block PMP_SEQUENCE_GEN is used to generate the pump start and stop sequence. The function block executes the following start and stop sequence functions:

- Decides the sequence of the pumps based on the process demand.
- Decides the sequence in which a pump should start and stop next.
- Indicates the pump that is least run in the network to start next.
- Indicates the pump that has run maximum in the network to stop next.
- Sends the start and stop sequence information to the distributor function blocks.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

READY_FOR_AUTOMATION

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	FALSE	TRUE/FALSE	-

The input READY_FOR_AUTOMATION receives an array of pumps ready for automatic operation. The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

The READY_FOR_AUTOMATION[1] connects to the PUMP ID = 1. This pattern must be followed for all pump IDs.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

OPERATING

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	-	TRUE/FALSE	-

The input OPERATING receives the operating status of all pumps (both variable frequency drives and direct online motors). The array is created from the output PUMP_OPERATING of the interface function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL.

RUNTIME

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	-	-

The input RUNTIME receives the actual runtime of all pumps (both variable frequency drives and direct online motors) in hours. The array is created from the output ACT_RUNTIME of the interface function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL.

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ MODE

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	-	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE of the used pump distributor function block receives the antijam mode. This input comes from the output ANTIJAM_MODE of the function blocks PMP_ANTIJAM.

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

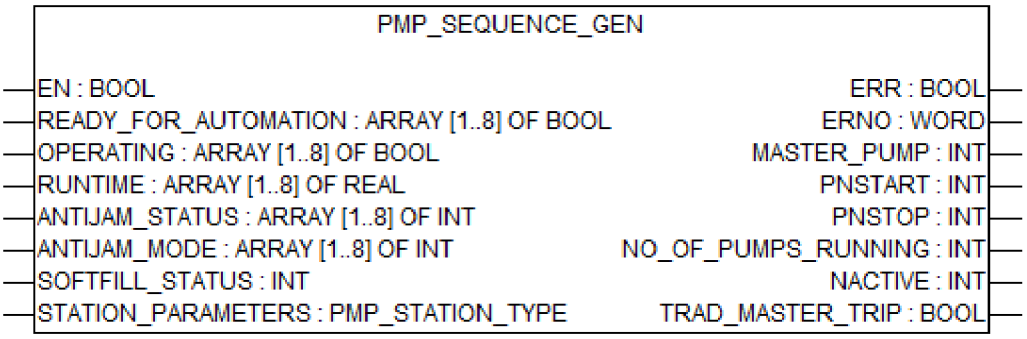
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

PNSTART

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTART receives the identification number of the next pump to start in the sequence. The input comes from the output PNSTART of the function block PMP_SEQUENCE_GEN.

PNSTOP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PNSTOP receives the identification number of the next pump to stop in the sequence. The input comes from the output PNSTOP of the function block PMP_SEQUENCE_GEN.

NO_OF_PUMPS_RUNNING

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NO_OF_PUMPS_RUNNING receives the number of pumps running. The input comes from the output NO_OF_PUMPS_RUNNING of the function block PMP_SEQUENCE_GEN.

NACTIVE

Data type	Default value	Range	Unit
INT	0	1 ... 8	-

The input NACTIVE receives the number of pumps active for operation in automatic mode. The input comes from the output NACTIVE of the function block PMP_SEQUENCE_GEN.

TRAD_MASTER_TRIP

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

Indicates that the master pump in the traditional pump combination has tripped. This input comes from the output TRAD_MASTER_TRIP of the function block PMP_SEQUENCE_GEN.

Error codes

The error codes of function block PMP_SEQUENCE_GEN are listed below:

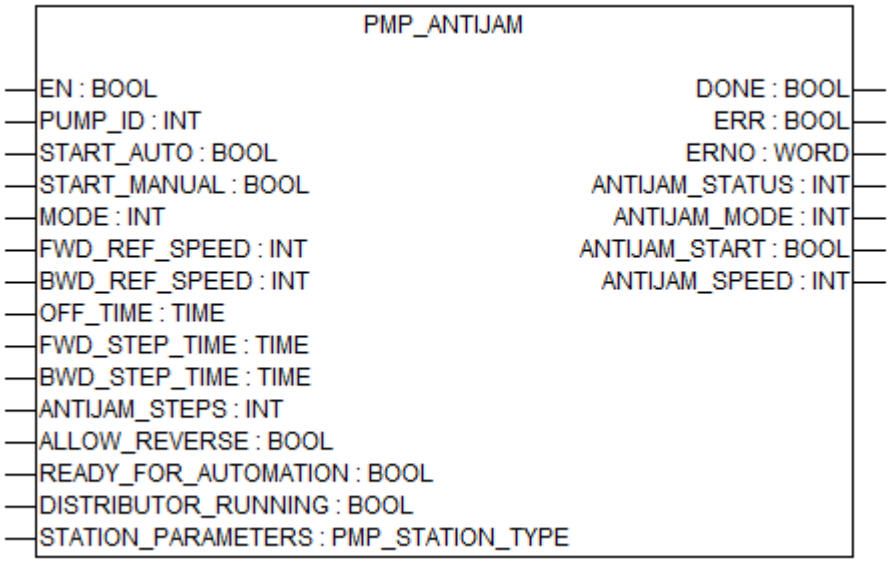
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16449	4041	Value of RUNTIME is less than 0
16465	4051	Value in input ANTIJAM_STATUS is less than 0
16466	4052	Value in input ANTIJAM_STATUS is greater than 2
16481	4061	Value in input ANTIJAM_MODE is less than 0
16482	4062	Value in input ANTIJAM_MODE is greater than 3
16497	4071	Value in input SOFTFILL_STATUS is less than 0
16498	4072	Value in input SOFTFILL_STATUS is greater than 4

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

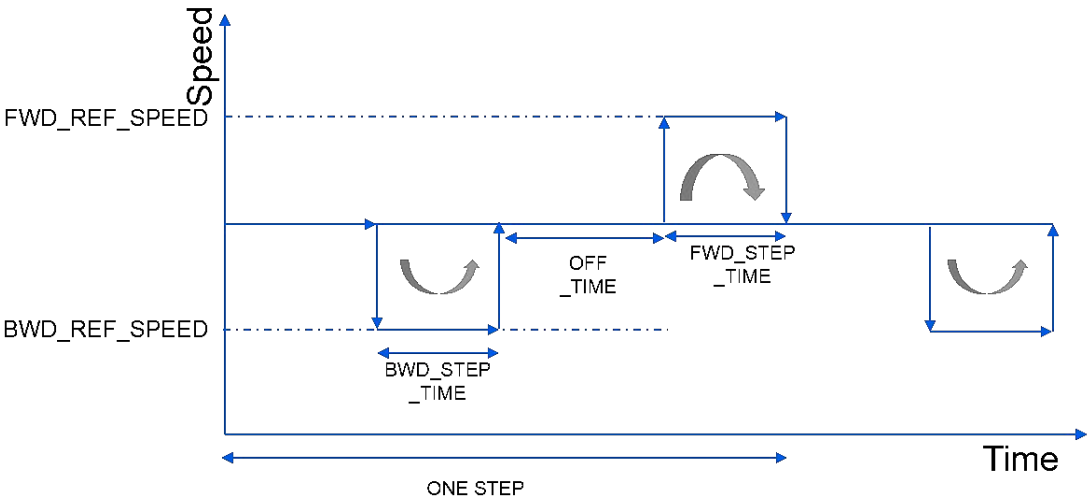
X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.10 PMP_ANTIJAM



Antijam overview

The pump antijam function is used to prevent building up of solids on the pump impellers or piping. The function consists of a programmable sequence of forward and reverse runs of the pump to shake off any residue on the impeller or piping. See timing diagram below. This function is mostly used with booster and wastewater pumps.



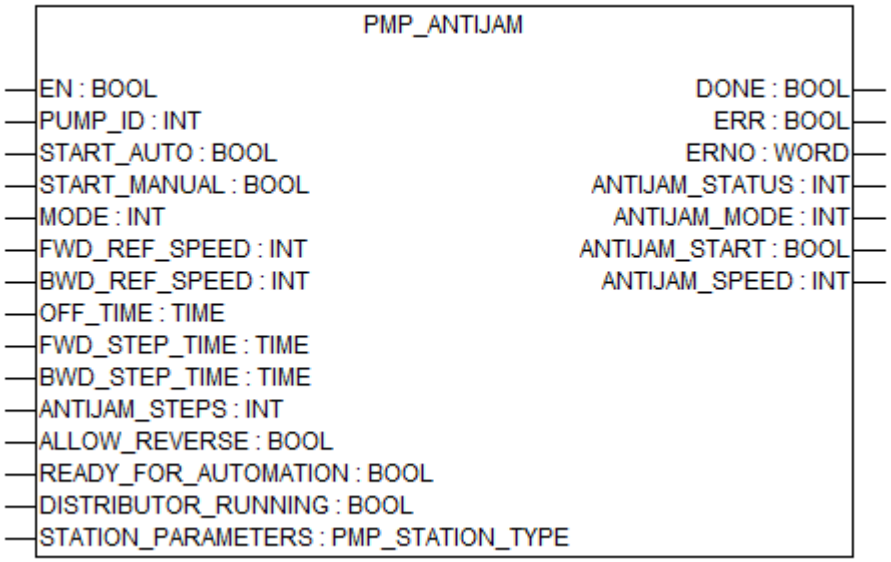
Function block description

The function block PMP_ANTIJAM performs the antijam function by running the pump at high speeds without any ramp up/down time.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ID

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input PUMP_ID defines the pump identification number.

START_AUTO

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_AUTO receives a digital signal to start antijam operation in automatic mode. The input comes from the output START_ANTIJAM of function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR and PMP_PRESSURE_DISTRIBUTOR. The input is not effective if the value in input MODE = 3 (Manual mode).

The input START_AUTO of PUMP_ID = 1 must be connected to the output START_ANTIJAM[1] of distributors. This pattern must be followed for all pump IDs.

- When input is TRUE, antijam operation starts in automatic mode.
- When input is FALSE, antijam operation stops if running.

START_ MANUAL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START_MANUAL receives a digital signal to start antijam operation in manual mode. The input comes from the output START_ANTIJAM of function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL. The input is effective only when value in input MODE = 3 (Manual mode).

- When input is TRUE, antijam operation starts in manual mode.
- When input is FALSE, antijam operation stops if running.

MODE

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The input MODE selects the antijam operation mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

FWD_REF_ SPEED

Data type	Default value	Range	Unit
INT	100	> 0	rpm

The input FWD_REF_SPEED defines the forward direction speed reference in rpm to clean the pump.

BWD_REF_ SPEED

Data type	Default value	Range	Unit
INT	100	> 0	rpm

The input BWD_REF_SPEED defines the backward direction speed reference in rpm to clean the pump.

This input is effective only if the input ALLOW_REVERSE = TRUE.

OFF_TIME

Data type	Default value	Range	Unit
TIME	10	-	s

The input OFF_TIME defines the time period for which antijam operation will pause after either a forward or backward movement is completed.

FWD_STEP_ TIME

Data type	Default value	Range	Unit
TIME	10	-	s

The input FWD_STEP_TIME defines the time duration for which the pump moves in forward direction for antijam operation.

BWD_STEP_ TIME

Data type	Default value	Range	Unit
TIME	10	-	s

The input BWD_STEP_TIME defines the time duration for which the pump moves in backward direction for antijam operation.

ANTIJAM_ STEPS

Data type	Default value	Range	Unit
INT	1	> 0	-

The input ANTIJAM_STEPS defines the number of steps in forward or backward movement to be performed in antijam operation.

ALLOW_ REVERSE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input ALLOW_REVERSE allows the pump run by variable frequency drive to move in reverse direction to perform antijam operation.

The input ALLOW_REVERSE is not effective for DOL pumps.

- When input is TRUE, pump with VFD performs antijam operation in the reverse direction.
- When input is FALSE, pump with VFD stops antijam operation in the reverse direction.

READY_FOR_ AUTOMATION

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

DISTRIBUTOR_ RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DISTRIBUTOR_RUNNING indicates the running status of the distributor.
This input is connected to the output DISTRIBUTOR_RUNNING of any of the the distributor function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, distributor is running.
- When output is FALSE, distributor is not running.

STATION_ PARAMETERS

Data type	Default value	Range	Unit
PMP_STA-TION_TYPE	-	-	-

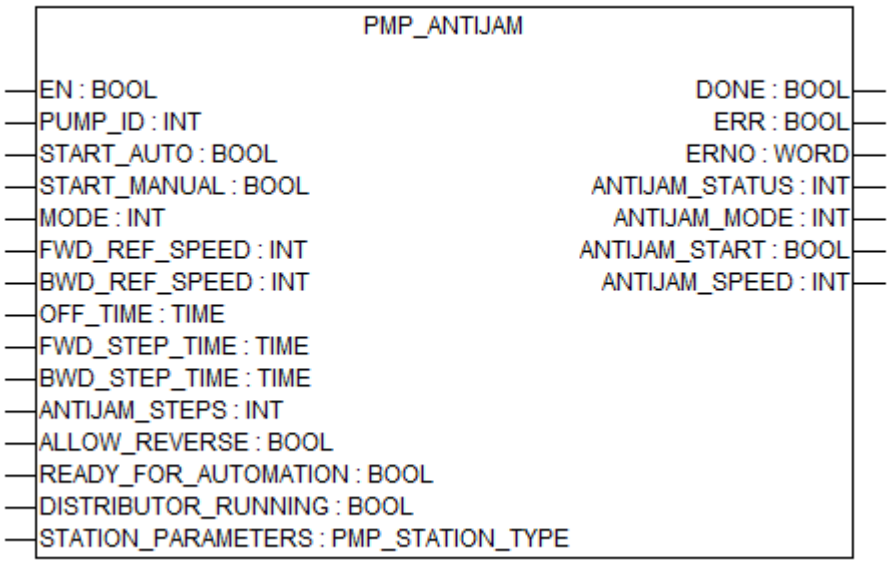
The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ANTIJAM_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 2	-

The input ANTIJAM_STATUS receives the antijam operation status of the pump. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

ANTIJAM_ MODE

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The input ANTIJAM_MODE indicates the selected antijam mode.

0 = Disable antijam

1 = Master enabled

2 = At start

3 = Manual mode

The input ANTIJAM_MODE receives the antijam mode. The input comes from the output ANTIJAM_MODE of the function block PMP_ANTIJAM.

ANTIJAM_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output ANTIJAM_START of the function block PMP_ANTIJAM starts the antijam operation. This output is connected with the input ANTIJAM_START of the function blocks PMP_INTERFACE_VFD and PMP_INTERFACE_DOL.

- When output is TRUE, antijam operation starts.
- When output is FALSE, antijam operation stops if running.

ANTIJAM_SPEED

Data type	Default value	Range	Unit
INT	0	-	rpm

The output ANTIJAM_SPEED of the function block PMP_ANTIJAM indicates the speed for antijam operation. This output is connected to the input ANTIJAM_SPEED of the function block PMP_INTERFACE_VFD.

Error codes

The error codes of function block PMP_ANTIJAM are listed below:

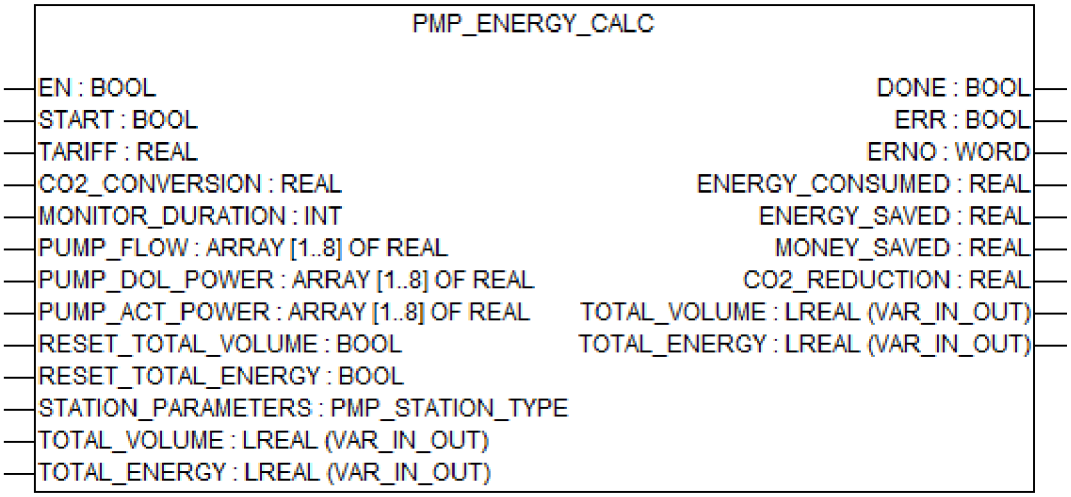
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value of PUMP_ID is less than 0
16418	4022	Value of PUMP_ID is greater than NUMBER_OF_PUMPS in PMP_CONFIGURATION
16435	4033	Value of START_AUTO = TRUE when MODE = 3
16451	4043	Value of START_MANUAL = TRUE when MODE is not 3
16465	4051	Value of MODE is less than 0
16466	4052	Value of MODE is more than 3
16481	4061	Value of FWD_REF_SPEED is less than 0
16497	4071	Value of BWD_REF_SPEED is less than 0
16561	40B1	Value of ANTIJAM_STEPS is less than 0
16579	40C3	Value of ALLOW_REVERSE = TRUE for the pump fed by DOL motor

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.11 PMP_ENERGY_CALC



The function block PMP_ENERGY_CALC calculates the energy consumption of the pumping station run by variable frequency drive and gives the information on cumulative flow in cubic meters. The function includes following calculations:

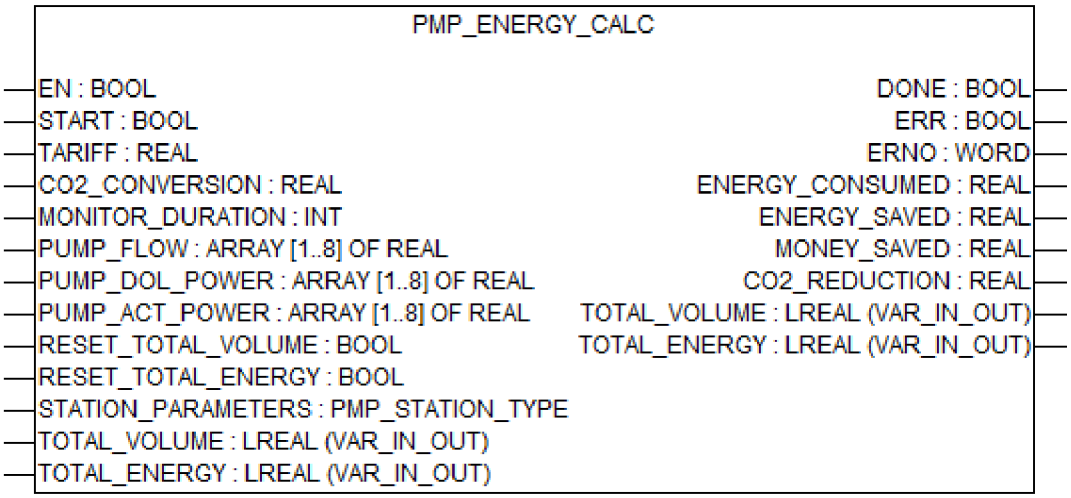
- Total energy consumed
- Total energy saved
- Total cost saved
- Reduced CO_2 emission

> *For correct calculations this function block must be called in the task manager with 50 ms cyclic time.*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

TARIFF

Data type	Default value	Range	Unit
REAL	1.0	> 0.0	-

The input TARIFF defines the tariff rate per kWh in different currencies, e.g., Dollar, Euro, etc.

CO2_ CONVERSION

Data type	Default value	Range	Unit
REAL	1.0	> 0.0	t/MWh

The input CO2_CONVERSION defines the factor to convert the energy consumed in CO_2 emission in t/MWh.

MONITOR_ DURATION

Data type	Default value	Range	Unit
INT	10	> 0	h

The input MONITOR_DURATION defines the time duration in hours to monitor the energy consumption.

PUMP_FLOW

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	-	m^3/h

The input PUMP_FLOW receives the actual flow of each pump in cubic meters per hour (m3/h).

The index 1 array is connected to the PUMP_ID = 1. This pattern follows for all pump IDs.

PUMP_DOL_ POWER

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	-	kW

The input PUMP_DOL_POWER defines the direct online power of each pump in kW, if the pumps are directly connected to the motor. The nominal power of the motor is connected to the pump.

PUMP_ACT_POWER

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	-	kW

The input PUMP_ACT_POWER defines the actual power of each pump in kW. The power is read from the variable frequency drive.

RESET_TOTAL_VOLUME

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET_TOTAL_VOLUME resets the total volume.

- When input is TRUE, total volume is reset.
- When input is FALSE, pump continues with total volume.

RESET_TOTAL_ENERGY

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET_TOTAL_ENERGY resets the total energy value.

- When input is TRUE, total energy value is reset.
- When input is FALSE, pump continues with total energy value.

STATION_PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control

Parameter	Data type	Default value	Description
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

TOTAL_VOLUME

Data type	Default value	Range	Unit
LREAL	-	-	m^3

The variable input/output TOTAL_VOLUME indicates the cumulative flow volume in cubic meters.

The total volume is calculated by combining the flow of all pumps in cubic meters.

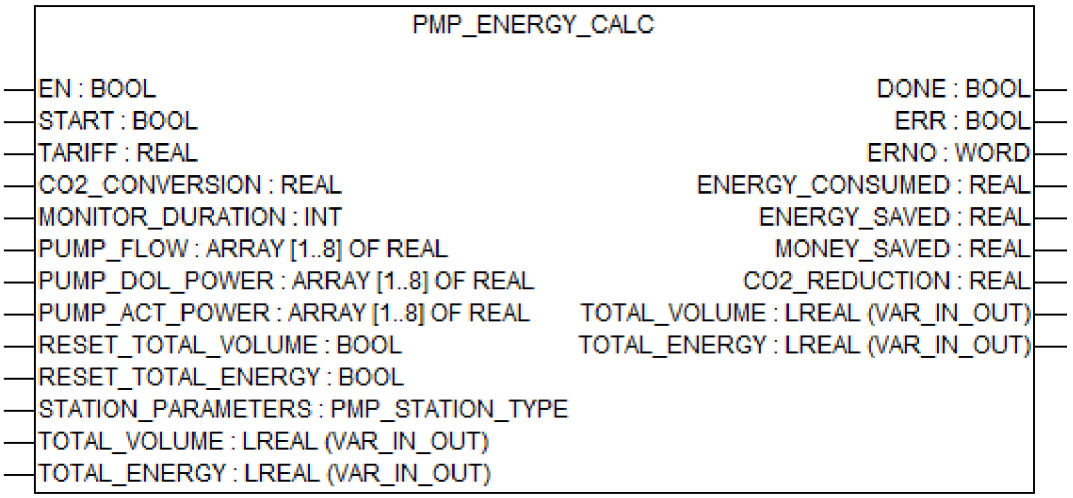
TOTAL_ENERGY

Data type	Default value	Range	Unit
LREAL	-	-	kWh

The variable input/output TOTAL_ENERGY indicates the total actual energy in kWh.

The total energy is calculated by combining the actual power of all pumps in kW in a hour.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ENERGY_ CONSUMED

Data type	Default value	Range	Unit
REAL	0.0	-	kWh

The output ENERGY_CONSUMED indicates the energy consumed (kWh) in the monitoring duration.

ENERGY_ SAVED

Data type	Default value	Range	Unit
REAL	0.0	-	kWh

The output ENERGY_SAVED indicates the saved energy in kWh. Saved energy is the difference of energy consumed with VFD pump and energy consumed if it was a DOL pump.

MONEY_SAVED

Data type	Default value	Range	Unit
REAL	0.0	-	-

The output MONEY_SAVED indicates the money saved in tariff units.

CO2_ REDUCTION

Data type	Default value	Range	Unit
REAL	0.0	-	t/MWh

The output CO2_REDUCTION indicates the reduction of CO2 emission in t/MWh.

Error codes

The error codes of function block PMP_ENERGY_CALC are listed below:

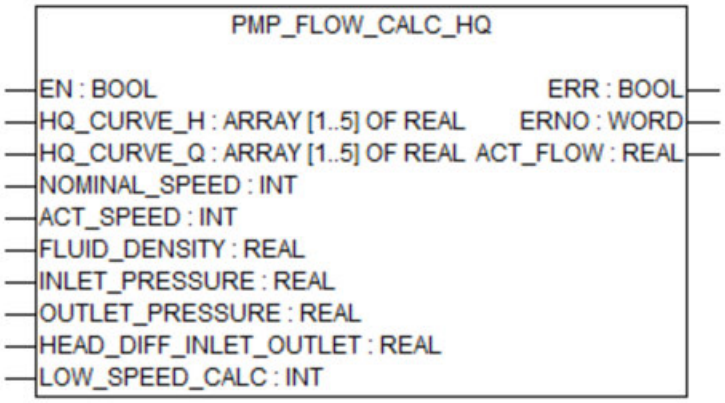
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16433	4031	Value of TARIFF is less than the minimum limit
16449	4041	Value of CO2_CONVERSION is less than the minimum limit
16465	4051	Value of MONITOR_DURATION is less than or equal to 0
16497	4071	Value of PUMP_DOL_POWER is less than or equal to 0.0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.12 PMP_FLOW_CALC_HQ

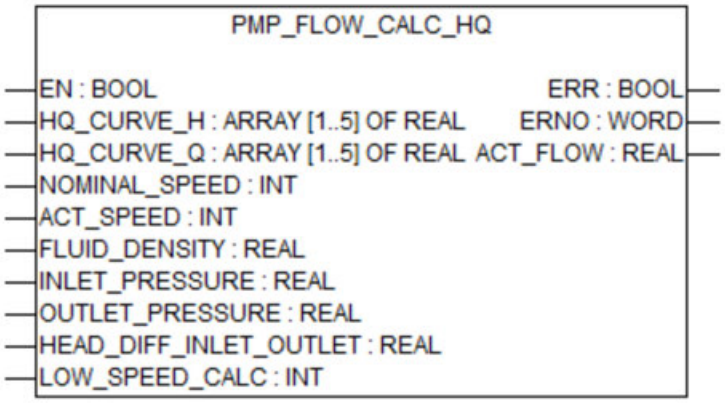


The function block PMP_FLOW_CALC_HQ calculates the flow rate of pumps using the pump head flow (HQ) curve characteristics. This function is applicable only for pumps run by variable frequency drives.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

HQ_CURVE_H

Data type	Default value	Range	Unit
ARRAY[1..5] of REAL	-	-	m

The input HQ_CURVE_H defines an array of head values for the HQ curve of pumps run by variable frequency drives. The values are in meters. HQ parameters are available in the pump characteristics of manufacturer’s specification.

HQ_CURVE_Q

Data type	Default value	Range	Unit
ARRAY[1..5] of REAL	-	-	m^3/h

The input HQ_CURVE_Q defines an array of flow rate values for HQ curve of pumps run by variable frequency drives. The values are in cubic meters per hour. HQ parameters are available in the pump characteristics of manufacturer’s specification.

NOMINAL_ SPEED

Data type	Default value	Range	Unit
INT	1400	> 0	rpm

The input NOMINAL_SPEED defines the rated speed in rpm of the pump run by variable frequency drive.

ACT_SPEED

Data type	Default value	Range	Unit
INT	10	> 1	rpm

The input ACT_SPEED indicates the actual speed of the variable frequency drive (VFD) in rpm.

FLUID_DENSITY

Data type	Default value	Range	Unit
REAL	1000.0	> 0.0	kg/m^3

The input FLUID_DENSITY defines the fluid density in kg/m3.

INLET_ PRESSURE

Data type	Default value	Range	Unit
REAL	0.0	-	-

The input INLET_PRESSURE receives the inlet pressure from the sensor that measures suction pressure.

OUTLET_ PRESSURE

Data type	Default value	Range	Unit
REAL	0.0	-	-

The input OUTLET_PRESSURE receives the outlet pressure from the sensor that measures discharge pressure.

HEAD_DIFF_ INLET_OUTLET

Data type	Default value	Range	Unit
REAL	0.0	-	m

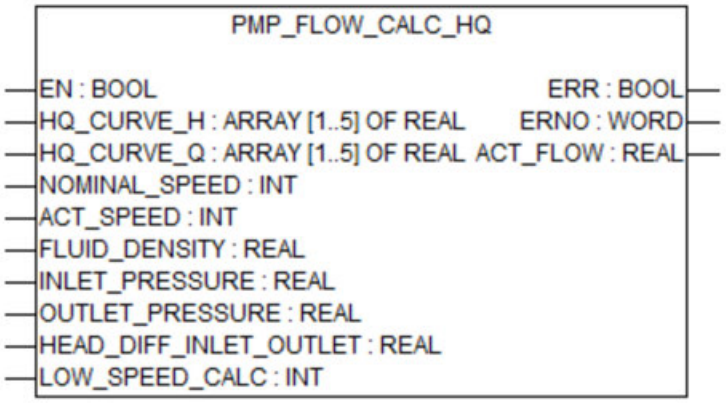
The input HEAD_DIFF_INLET_OUTLET defines the differential pressure of inlet and outlet head values in meters. The input is used only when direct measurement of differential pressure is available. If direct measurement of differential pressure is not available, then this input must be left as unassigned.

LOW_SPEED_ CALC

Data type	Default value	Range	Unit
INT	50	-	rpm

The input LOW_SPEED_CALC defines the low speed limit in rpm. This function block calculates the actual flow only when input ACTUAL_SPEED is more than this value.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ACT_FLOW

Data type	Default value	Range	Unit
INT	0	-	m^3/h

The output ACT_FLOW indicates the calculated actual flow in cubic meters per hour.

Error codes

The error codes of function block PMP_FLOW_CALC_HQ are listed below:

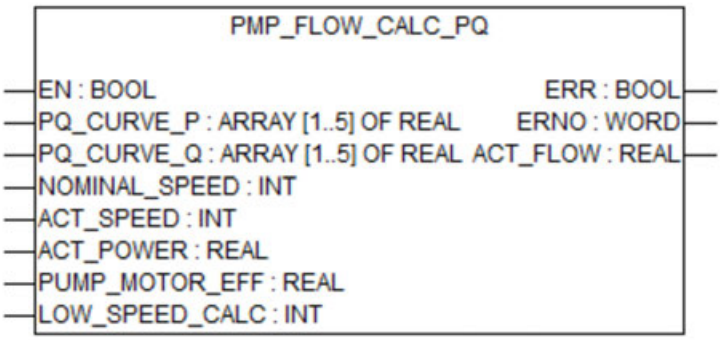
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16416	4020	Array of values in HQ_CURVE_H are not in ascending order
16432	4030	Array of values in HQ_CURVE_Q are not in ascending order
16449	4041	Value of NOMINAL_SPEED is less than 0
16481	4061	Value of INLET_PRESSURE is less than 0
16497	4071	Value of OUTLET_PRESSURE is less than 0
16499	4073	Value of INLET_PRESSURE is greater than OUTLET_PRESSURE
16513	4081	Value of FLUID_DENSITY is less than or equal to 0
16529	4091	Value of HEAD_DIFF_INLET_OUTLET is less than 0
16545	40A1	Value of LOW_SPEED_CALC is less than 0
16547	40A3	Value of LOW_SPEED_CALC is greater than or equal to NOMINAL_SPEED

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.13 PMP_FLOW_CALC_PQ

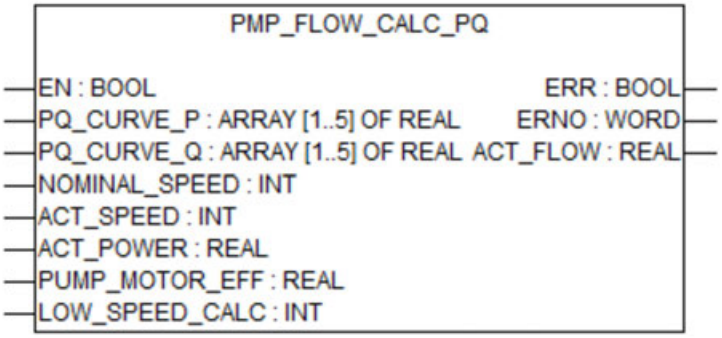


The function block PMP_FLOW_CALC_PQ calculates the flow rate of pumps using the pump pressure flow rate (PQ) curve characteristics. This function is applicable only for pumps run by variable frequency drives.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PQ_CURVE_P

Data type	Default value	Range	Unit
ARRAY[1..5] of REAL	-	-	m

The input PQ_CURVE_P defines an array of input power for PQ curve of pumps run by variable frequency drives. The values are in meters. PQ parameters are available in the pump characteristics of manufacturer’s specification.

HQ_CURVE_Q

Data type	Default value	Range	Unit
ARRAY[1..5] of REAL	-	-	m^3/h

The input HQ_CURVE_Q defines an array of flow rate values for HQ curve of pumps run by variable frequency drives. The values are in cubic meters per hour. HQ parameters are available in the pump characteristics of manufacturer’s specification.

PQ_CURVE_Q

Data type	Default value	Range	Unit
ARRAY[1..5] of REAL	-	-	m^3/h

The input PQ_CURVE_Q defines an array of flow rate values for PQ curve of pumps run by variable frequency drives. The values are in cubic meters per hour. PQ parameters are available in the pump characteristics of manufacturer’s specification.

NOMINAL_ SPEED

Data type	Default value	Range	Unit
INT	1400	> 0	rpm

The input NOMINAL_SPEED defines the rated speed in rpm of the pump run by variable frequency drive.

ACT_SPEED

Data type	Default value	Range	Unit
INT	10	> 1	rpm

The input ACT_SPEED indicates the actual speed of the variable frequency drive (VFD) in rpm.

ACT_POWER

Data type	Default value	Range	Unit
REAL	1.0	-	kW

The input ACT_POWER defines the actual power of the motor in kW. The value is read from the drive parameters.

PUMP_MOTOR_ EFF

Data type	Default value	Range	Unit
REAL	0.85	0.00 ... 1.00	-

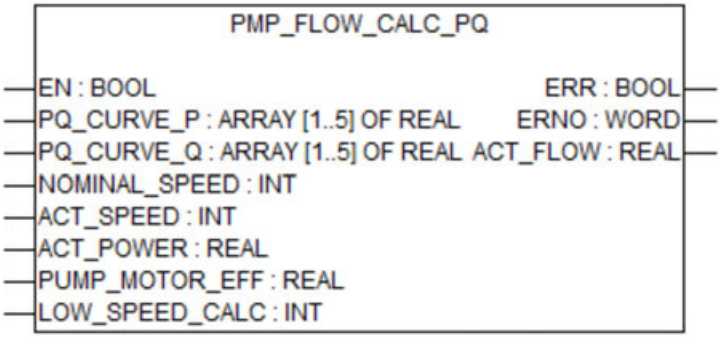
The input PUMP_MOTOR_EFF defines the combined efficiency of motor and pump set.

LOW_SPEED_ CALC

Data type	Default value	Range	Unit
INT	50	-	rpm

The input LOW_SPEED_CALC defines the low speed limit in rpm. This function block calculates the actual flow only when input ACTUAL_SPEED is more than this value.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ACT_FLOW

Data type	Default value	Range	Unit
INT	0	-	m^3/h

The output ACT_FLOW indicates the calculated actual flow in cubic meters per hour.

Error codes

The error codes of function block PMP_FLOW_CALC_PQ are listed below:

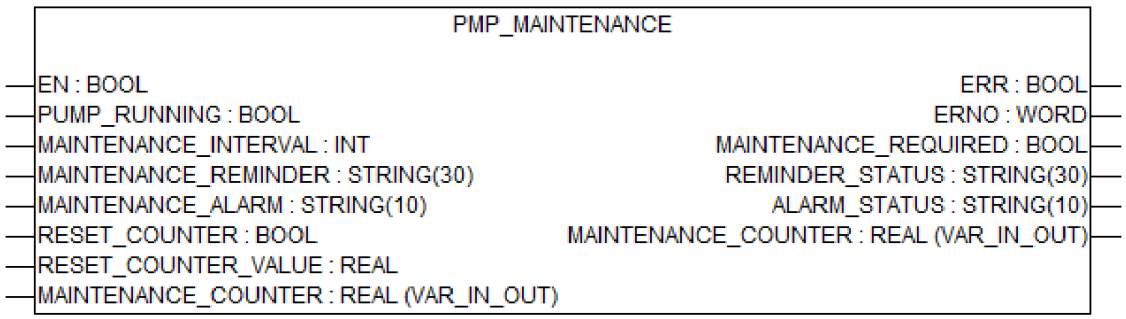
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16416	4020	Array of values in PQ_CURVE_P are not in ascending order
16432	4030	Array of values in PQ_CURVE_Q are not in ascending order
16449	4041	Value of NOMINAL_SPEED is less than or equal to 0
16497	4071	Value of MOTOR_EFF is less than or equal to 0.00
16498	4072	Value of MOTOR_EFF is greater than 1.00
16513	4081	Value of LOW_SPEED_CALC is less than 0
16515	4083	Value of LOW_SPEED_CALC is greater than the NOMINAL_SPEED

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.14 PMP_MAINTENANCE

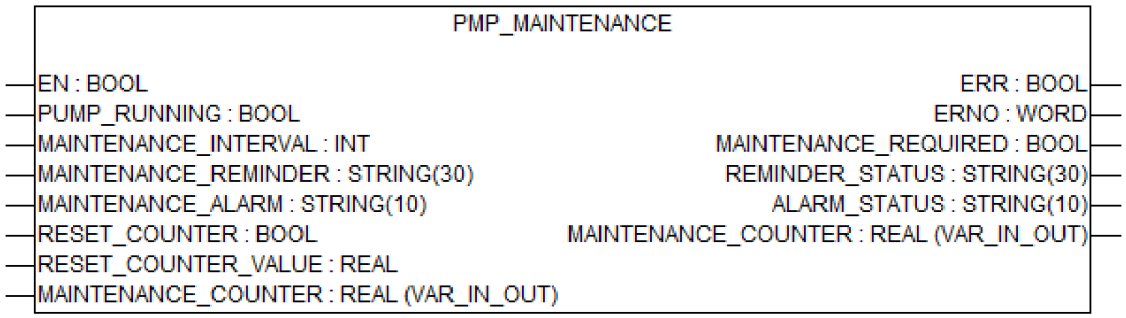


The function block PMP_MAINTENANCE is used to record the maintenance schedule of the pump. The function block maintains an hourly counter to track the pump maintenance interval and generates timely reminders and alarms.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

PUMP_ RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input PUMP_RUNNING receives the pump running status. The input comes from the output PUMP_OPERATING of the function blocks PMP_INTERFACE_VFD/ PMP_INTERFACE_DOL.

- When input is TRUE, pump is operating.
- When input is FALSE, pump is not operating.

MAINTENANCE_ INTERVAL

Data type	Default value	Range	Unit
INT	1	0 ... 32767	-

The input MAINTENANCE_INTERVAL defines the pump maintenance interval in hours. When specific maintenance intervals are reached, function block generates reminders and alarms.

MAINTENANCE_ REMINDER

Data type	Default value	Range	Unit
STRING (30)	-	-	-

The input MAINTENANCE_REMINDER defines the user-defined reminder message for service maintenance.

MAINTENANCE_ALARM

Data type	Default value	Range	Unit
STRING (10)	-	-	-

The input MAINTENANCE_ALARM defines the user-defined alarm message for any maintenance activity.

RESET_COUNTER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET_COUNTER resets the output MAINTENANCE_COUNTER to the value in the input RESET_COUNTER_VALUE.

- When input is TRUE, value in input RESET_COUNTER_VALUE overwrites value in output MAINTENANCE_COUNTER.
- When input is FALSE, no reset is initiated.

RESET_COUNTER_VALUE

Data type	Default value	Range	Unit
INT	0	0 ... 32767	-

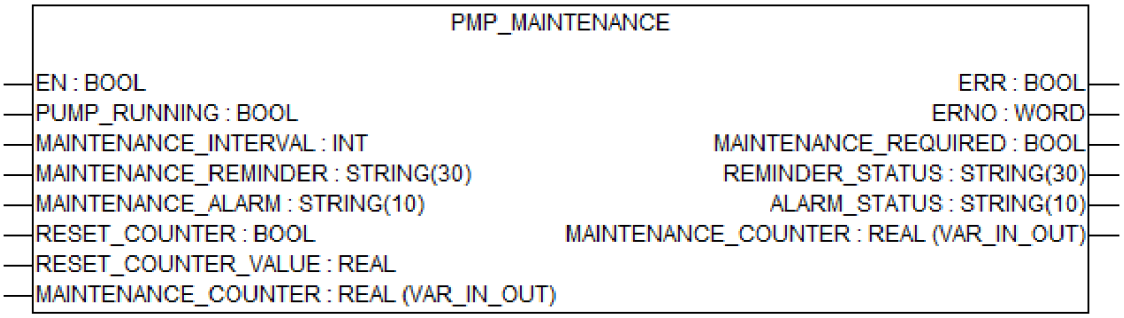
The input RESET_COUNTER_VALUE defines the counter value that is used to overwrite the output MAINTENANCE_COUNTER. The input is effective only when input RESET_COUNTER = TRUE.

MAINTENANCE_COUNTER

Data type	Default value	Range	Unit
REAL	-	-	-

The variable input/output MAINTENANCE_COUNTER provides the count of hourly activity. The count is overwritten by value in the input RESET_COUNTER_VALUE, when input RESET_COUNTER = TRUE

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

MAINTENANCE_REQUIRED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output MAINTENANCE_REQUIRED indicates that maintenance is required.

- When output is TRUE, maintenance is required.
- When output is FALSE, no maintenance required.

REMINDER_STATUS

Data type	Default value	Range	Unit
STRING (30)	-	-	-

The output REMINDER_STATUS indicates the status of service reminder message defined in the input MAINTENANCE_REMINDER.

ALARM_STATUS

Data type	Default value	Range	Unit
STRING (10)	-	-	-

The output ALARM_STATUS indicates the status of maintenance alarm message defined in the input MAINTENANCE_ALARM.

Error codes

The error codes of function block PMP_MAINTENANCE are listed below:

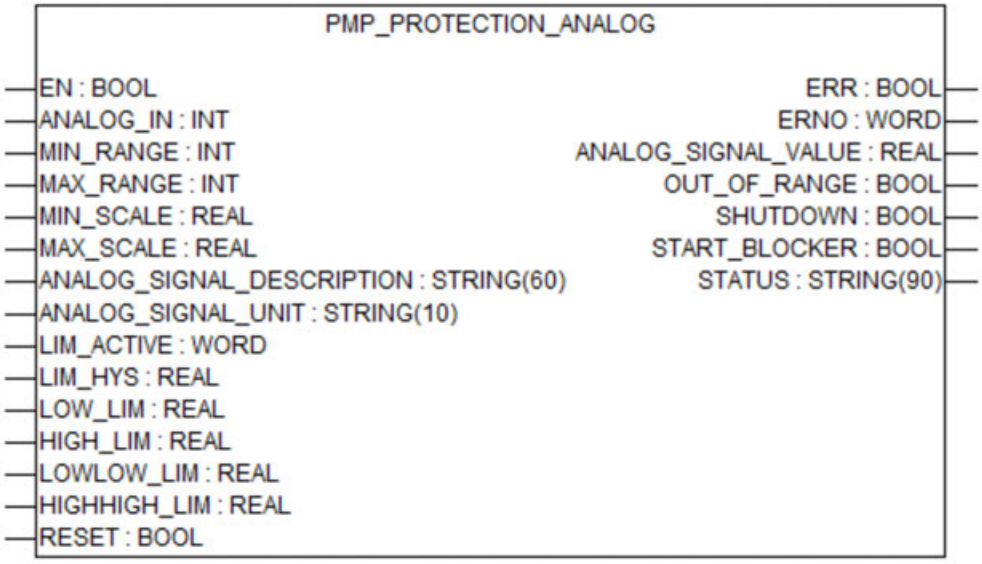
Error code		Error Description
Decimal	Hexadecimal	
0	0000	No error occurred
16432	4030	Value in MAINTENANCE_INTERVAL is less than or equal to 0
16496	4070	Value in RESET_COUNTER_VALUE is less than 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.15 PMP_PROTECTION_ANALOG



The function block PMP_PROTECTION_ANALOG monitors analog inputs and generates signals to shut down or to block the starting of pump or the process. The function block sends the signals to shut down or to block the starting of pump through inputs PROTECTION_SHUTDOWN or START_BLOCKER of these distributor and pump interface function blocks: PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR, PMP_PRESSURE_DISTRIBUTOR, PMP_INTERFACE_VFD, and PMP_INTERFACE_DOL.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Pump protection configuration example

The function block PMP_PROTECTION_ANALOG can be used for:

- Process specific protection or
- Pump specific protection

Process specific protection with many such sensors

The function block can be used to configure the alarm (start blocker) or fault (protection shutdown) for the process value coming from an analog input (sensor). The actual process value from the analog input is compared with the limits. See the example diagram below.

- If high/low limits are breached, the alarm start blocker is generated.
- If highhigh/lowlow limits are breached the protection shutdown is generated.

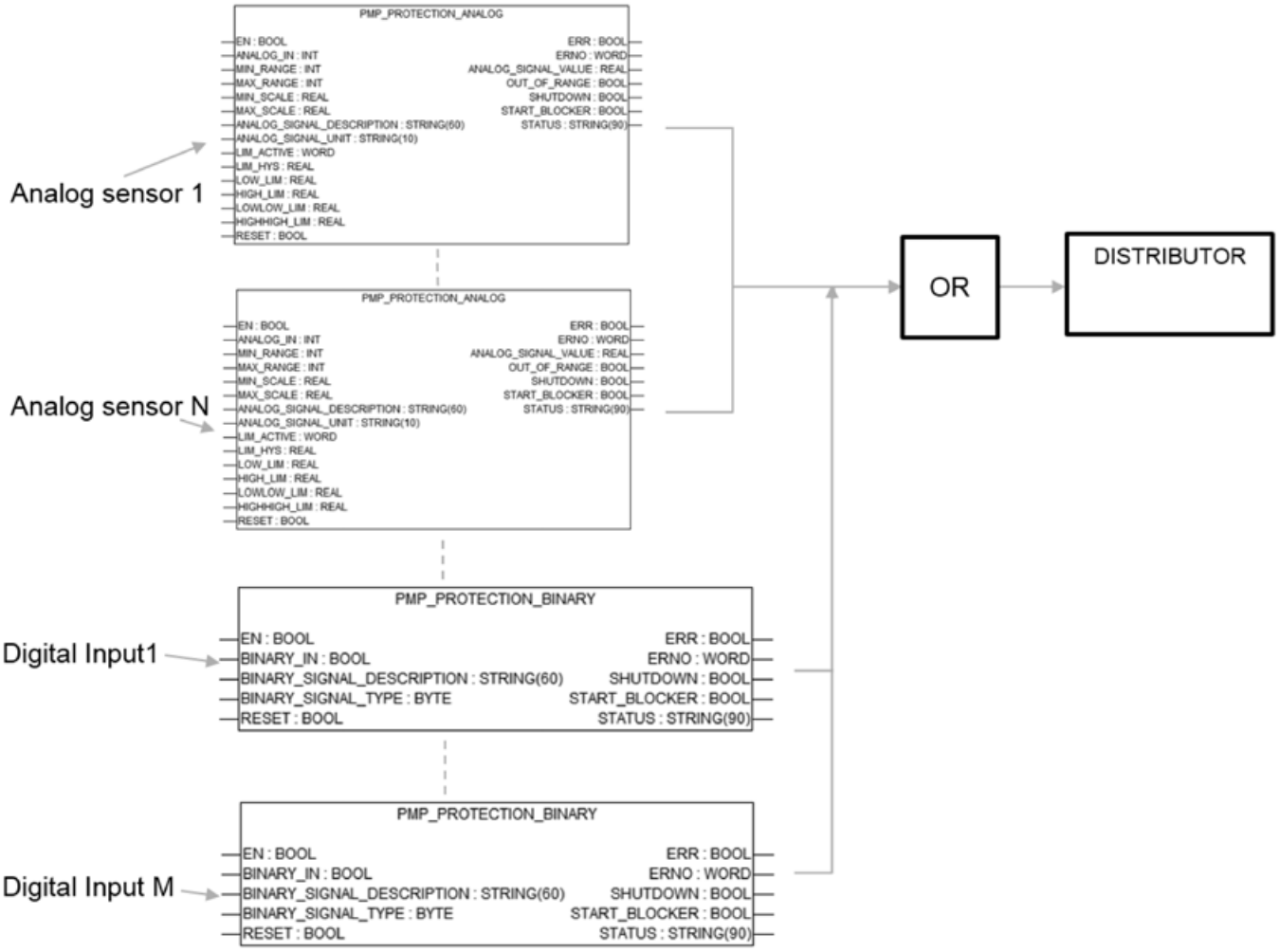


Fig. 715: Analog/Digital FB - process specific protection

Pump specific protection with many such sensors

The function block is flexible and can be used based on the pump specific requirements. If more than one sensor is used, then configure the function block for each sensor. The outputs of each function block can be logically OR'ed in the application program, as shown below.

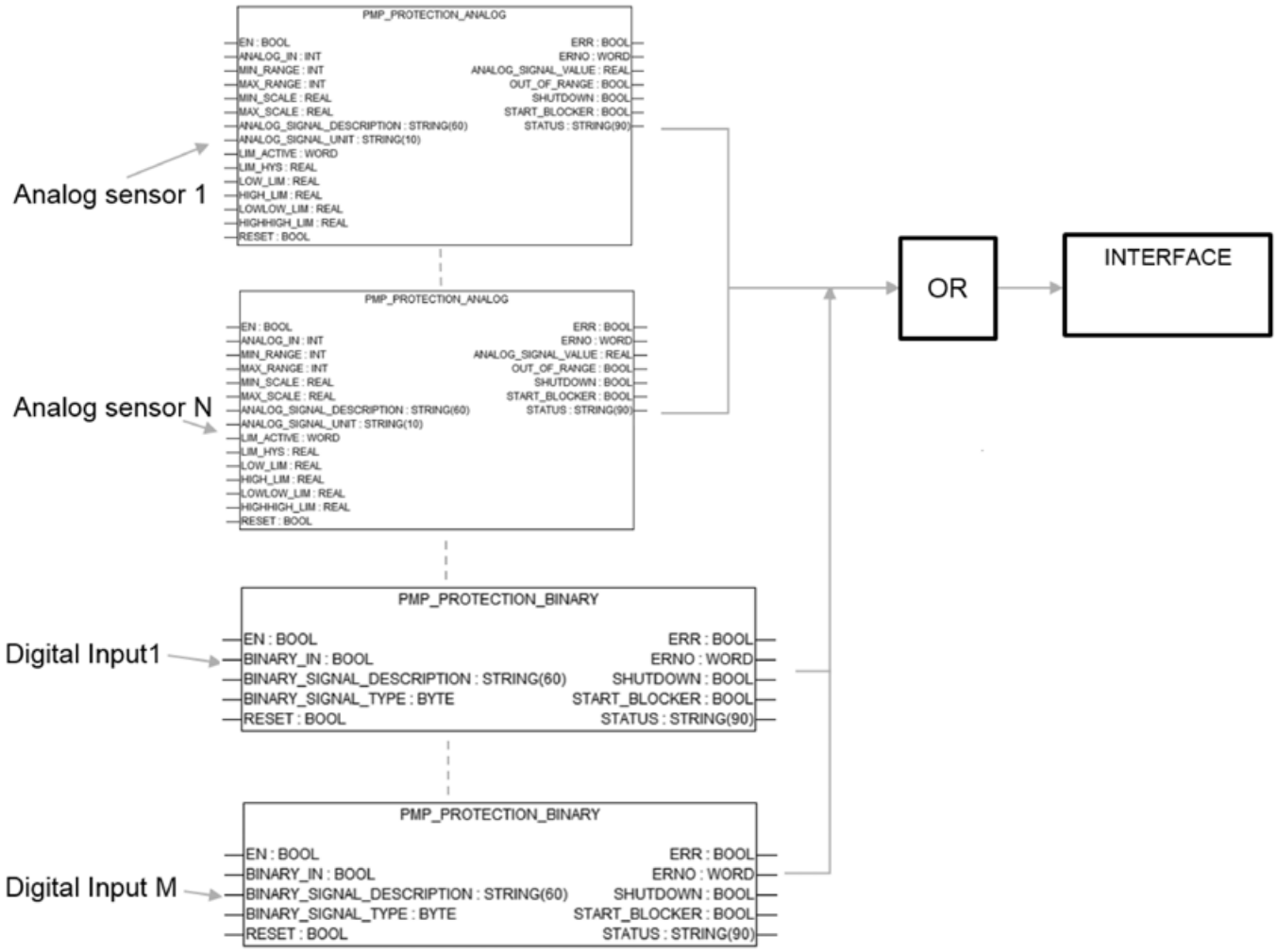
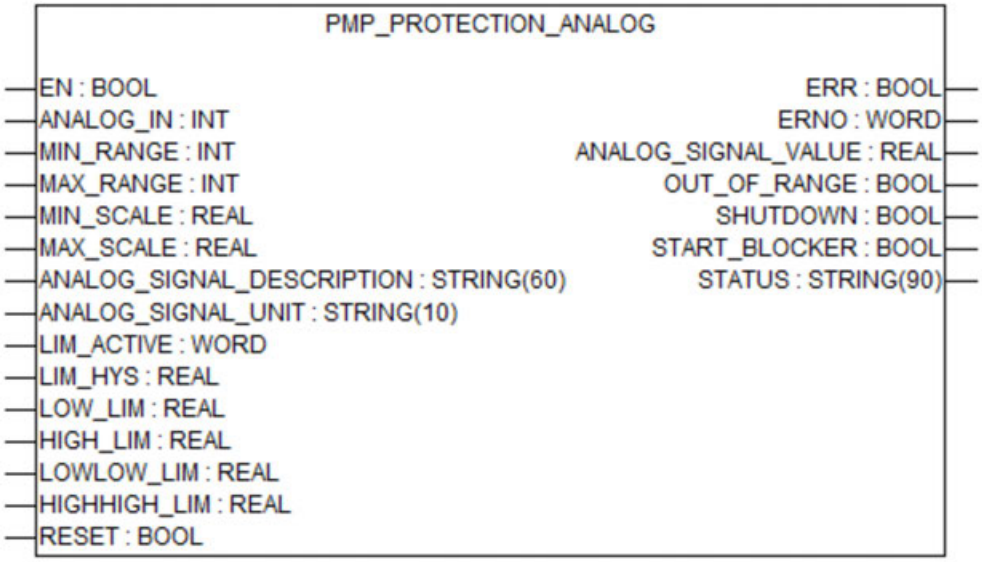


Fig. 716: Analog/Digital FB - pump specific protection

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

ANALOG_IN

Data type	Default value	Range	Unit
INT	10	-	-

The input ANALOG_IN receives analog input from the process. The process input is compared with the limits to perform the protection function.

MIN_RANGE

Data type	Default value	Range	Unit
INT	0	-	-

The input MIN_RANGE defines the minimum range of the analog input signal.

MAX_RANGE

Data type	Default value	Range	Unit
INT	27648	-	-

The input MAX_RANGE defines the maximum range of the analog input signal.

MIN_SCALE

Data type	Default value	Range	Unit
REAL	0.0	-	-

The input MIN_SCALE defines the minimum scale of the analog input signal in the physical unit, e.g., 1 bar.

MAX_SCALE

Data type	Default value	Range	Unit
REAL	100.0	-	-

The input MAX_SCALE defines the maximum scale of the analog input signal in physical unit, e.g., 5 bar.

ANALOG_ SIGNAL_ DESCRIPTION

Data type	Default value	Range	Unit
STRING (60)	-	-	-

The input ANALOG_SIGNAL_DESCRIPTION allows the user to add the description of analog input signal. e.g., actual pressure.

ANALOG_ SIGNAL_UNIT

Data type	Default value	Range	Unit
STRING (10)	-	-	-

The input ANALOG_SIGNAL_UNIT allows the user to declare the unit of analog input signal. e.g., PSI

LIM_ACTIVE

Data type	Default value	Range	Unit
WORD	2#1111	-	-

The input LIM_ACTIVE selects the active limit of the analog input signal.

bit 0: Low limit active

bit 1: High limit active

bit 2: Low low limit active

bit 3: High high limit active

The selected limit only will be compared with the actual process values to generate the PROTECTION_SHUTDOWN or START_BLOCKER signal.

LIM_HYS

Data type	Default value	Range	Unit
REAL	0.0	-	-

The input LIM_HYS allows the user to declare the hysteresis limit of analog input signal. This value is the percent of the limits mentioned in LOW_LIM/ LOWLOW_LIM/ HIGH_LIM/ HIGHHIGH_LIM.

The limit ± hysteresis is compared with the actual process value.

LOW_LIM

Data type	Default value	Range	Unit
REAL	10.0	-	-

The input LOW_LIM declares the low limit of the analog input.

HIGH_LIM

Data type	Default value	Range	Unit
REAL	80.0	-	-

The input HIGH_LIM declares the high limit of the analog input.

LOWLOW_LIM

Data type	Default value	Range	Unit
REAL	5.0	-	-

The input LOWLOW_LIM declares the very low limit of the analog input.

HIGHHIGH_LIM

Data type	Default value	Range	Unit
REAL	90.0	-	-

The input HIGHHIGH_LIM declares the very high limit of the analog input.

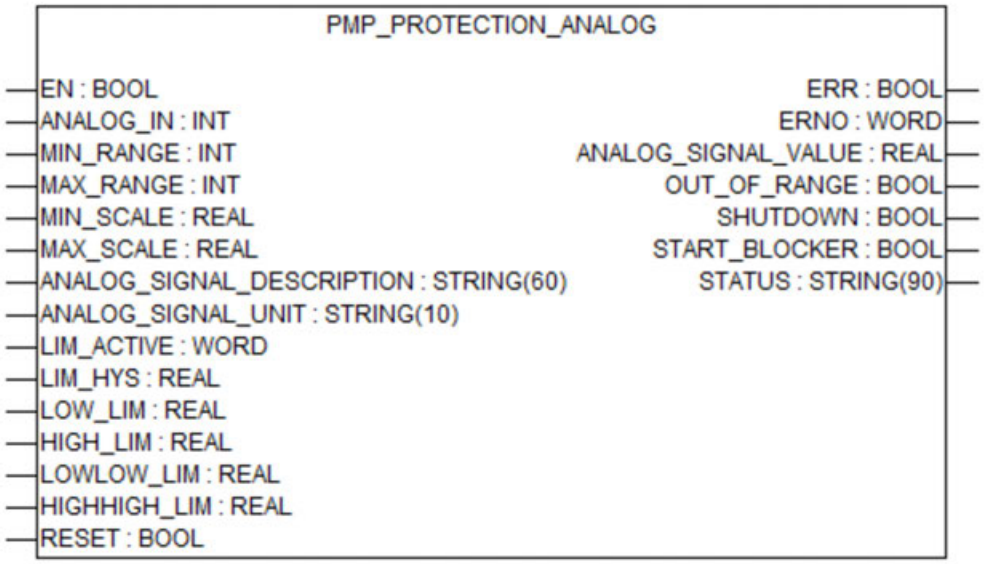
RESET

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET resets any active alarm/fault and alarm messages generated by analog or binary input signal.

- When output is TRUE, alarm/fault and alarm message are reset.
- When output is FALSE, no reset.

Output description



ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

ANALOG_ SIGNAL_VALUE

Data type	Default value	Range	Unit
REAL	0.0	-	-

The output ANALOG_SIGNAL_VALUE indicates the actual value of the analog input signal in physical units, e.g., bar.

OUT_OF_ RANGE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output OUT_OF_RANGE indicates that the analog input is out of range.

- When output is TRUE, analog input is out of range.
- When output is FALSE, analog input is within the range.

SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output SHUTDOWN indicates pump shutdown depending on the analog or binary input signal.

- If the requirement is to stop the complete process, then the output must be connected to the input PROTECTION_SHUTDOWN of the distributor function blocks.
- If the requirement is stop only a particular pump, then the output must be connected to the input PROTECTION_SHUTDOWN of the interface function blocks.

The output is generated when actual process value is compared with the limits in the input HIGHHIGH_LIM and/or LOWLOW_LIM.

- When output is TRUE, pump station or pump is shutdown.
- When output is FALSE, pump station or pump’s operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and/or PMP_PROTECTION_BINARY stops the pump from starting depending on the analog or binary input signal.

- If the requirement is to block starting of the complete process, then the output must be connected to the input START_BLOCKER of the distributor function blocks.
- If the requirement is block the starting of only a particular pump, then the output must be connected to the input START_BLOCKER of the interface function blocks.

The output is generated when actual process value is compared with the limits in the input HIGH_LIM and/or LOW_LIM.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

The input START_BLOCKER will not stop the already running pump.

STATUS

Data type	Default value	Range	Unit
STRING (90)	-	-	-

The output STATUS indicates the status of fault/alarm messages generated by this function block.

Error codes

The error codes of function block PMP_PROTECTION_ANALOG are listed below:

Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16433	4031	Value in MIN_RANGE is less than 0
16449	4041	Value in MAX_RANGE is less than or equal to 0
16451	4043	Value in MAX_RANGE is less than or equal to MIN_RANGE
16481	4051	Value in MIN_SCALE is less than 0
16481	4061	Value in MAX_ SCALE is less than or equal to 0
16483	4063	Value in MAX_ SCALE is less than or equal to MIN_ SCALE
16528	4090	LIM_ACTIVE is changed when EN is TRUE
16545	40A1	Value in LIM_HYS is less than 0
16546	40A2	Value in LIM_HYS is greater than 100
16563	40B3	HIGH_LIM ≤ LOW_LIM or HIGHHIGH_LIM ≤ LOW_LIM
16595	40D3	HIGHHIGH_LIM ≤ LOWLOW_LIM or HIGH_LIM ≤ LOWLOW_LIM
16611	40E3	HIGHHIGH_LIM ≤ HIGH_LIM or HIGHHIGH_LIM ≤ 0 or HIGH_LIM < 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.16 PMP_PROTECTION_BINARY

The function block PMP_PROTECTION_BINARY monitors binary inputs and generates signals to shut down or to stop the starting of pump. The function block sends the signals to shut down or to stop the pump start through inputs PROTECTION_SHUTDOWN or START_BLOCKER of these distributor and pump interface function blocks: PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR, PMP_PRESSURE_DISTRIBUTOR, PMP_INTERFACE_VFD, and PMP_INTERFACE_DOL.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Pump protection configuration example

The function block PMP_PROTECTION_BINARY can be used for:

- Process specific protection or
- Pump specific protection

Process specific protection with many such sensors

The function block can be used to configure the alarm (start blocker) or fault (protection shutdown) for the digital input signal. The input coming from the digital sensor can be used to generate the alarm (start blocker) or fault (protection shutdown) using the settings in the function block. See example diagram below:

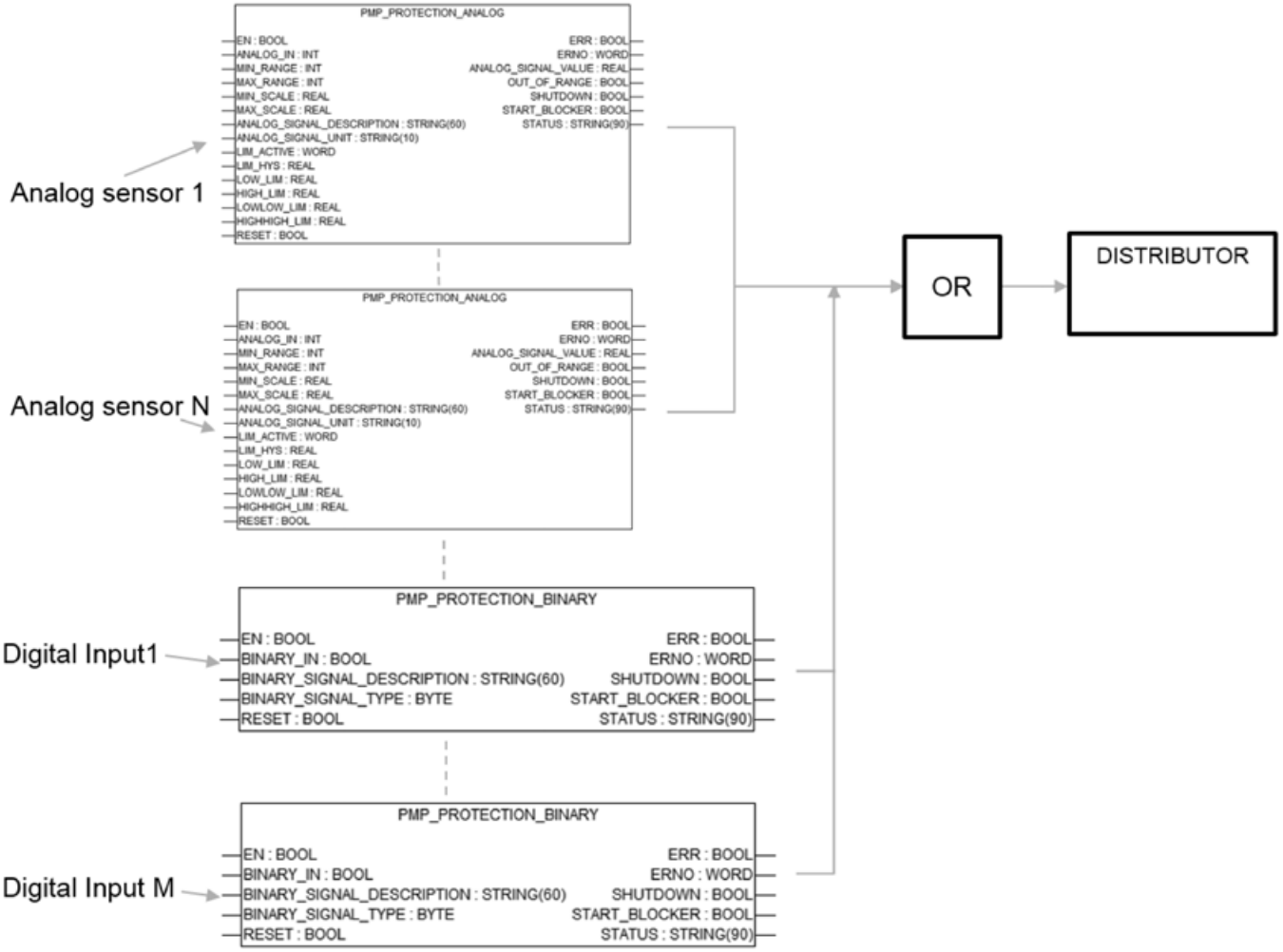


Fig. 717: Analog/Digital FB - process specific protection

Pump specific protection with many such sensors

The function block is flexible and can be used based on the requirements. If more than one sensor is used, then configure the function block for each digital sensor. The outputs of each function block can be logically OR'ed in the application program, as shown below.

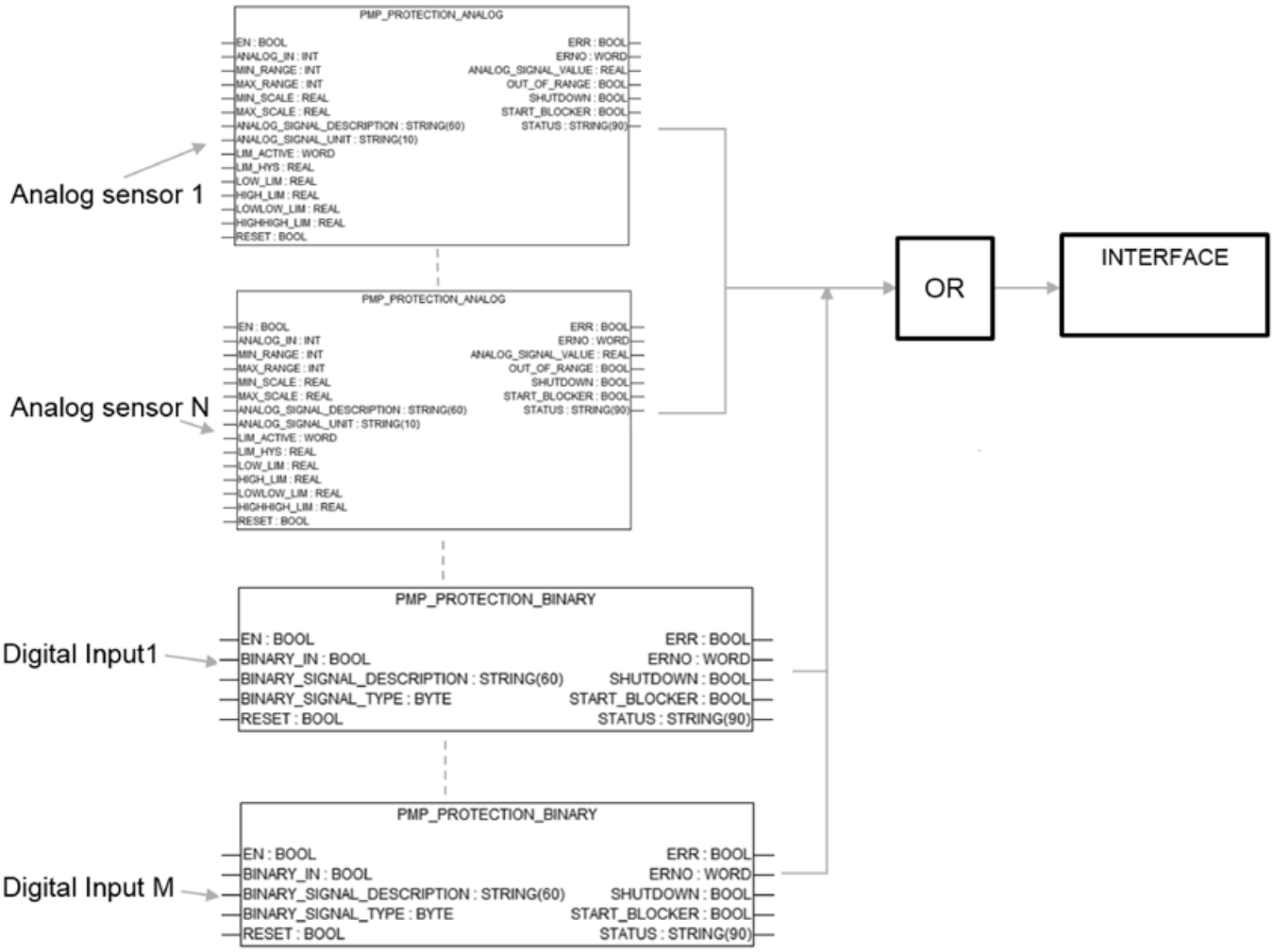


Fig. 718: Analog/Digital FB - pump specific protection

Input description

EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

BINARY_IN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input BINARY_IN receives the binary input from the process.

- When input is TRUE, binary input received.
- When input is FALSE, no input received.

BINARY_ SIGNAL_ DESCRIPTION

Data type	Default value	Range	Unit
STRING (60)	-	-	-

The input BINARY_SIGNAL_DESCRIPTION allows the user to add the description of the binary input signal.

BINARY_ SIGNAL_TYPE

Data type	Default value	Range	Unit
BYTE	0	0 ... 2	-

The input BINARY_SIGNAL_TYPE selects the type of the binary input signal.

0 = Not used (no input signal)

1 = To generate alarm (binary input signal to generate an alarm)

2 = To generate fault (binary input signal to indicate a fault)

RESET

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input RESET resets any active alarm/fault and alarm messages generated by analog or binary input signal.

- When output is TRUE, alarm/fault and alarm message are reset.
- When output is FALSE, no reset.

Output description

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

SHUTDOWN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output SHUTDOWN indicates pump shutdown depending on the analog or binary input signal.

- If the requirement is to stop the complete process, then the output must be connected to the input PROTECTION_SHUTDOWN of the distributor function blocks.
- If the requirement is stop only a particular pump, then the output must be connected to the input PROTECTION_SHUTDOWN of the interface function blocks.

The output is generated when actual process value is compared with the limits in the input HIGHHIGH_LIM and/or LOWLOW_LIM.

- When output is TRUE, pump station or pump is shutdown.
- When output is FALSE, pump station or pump’s operation continues.

START_ BLOCKER

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE / FALSE	-

The output START_BLOCKER of the protection function blocks PMP_PROTECTION_ANALOG and/or PMP_PROTECTION_BINARY stops the pump from starting depending on the analog or binary input signal.

- If the requirement is to block starting of the complete process, then the output must be connected to the input START_BLOCKER of the distributor function blocks.
- If the requirement is block the starting of only a particular pump, then the output must be connected to the input START_BLOCKER of the interface function blocks.

The output is generated when actual process value is compared with the limits in the input HIGH_LIM and/or LOW_LIM.

- When input is TRUE, pump start is not allowed.
- When input is FALSE, pump start is allowed.

The input START_BLOCKER will not stop the already running pump.

STATUS

Data type	Default value	Range	Unit
STRING (90)	-	-	-

The output STATUS indicates the status of fault/alarm messages generated by this function block.

Error codes

The error codes of function block PMP_PROTECTION_BINARY are listed below:

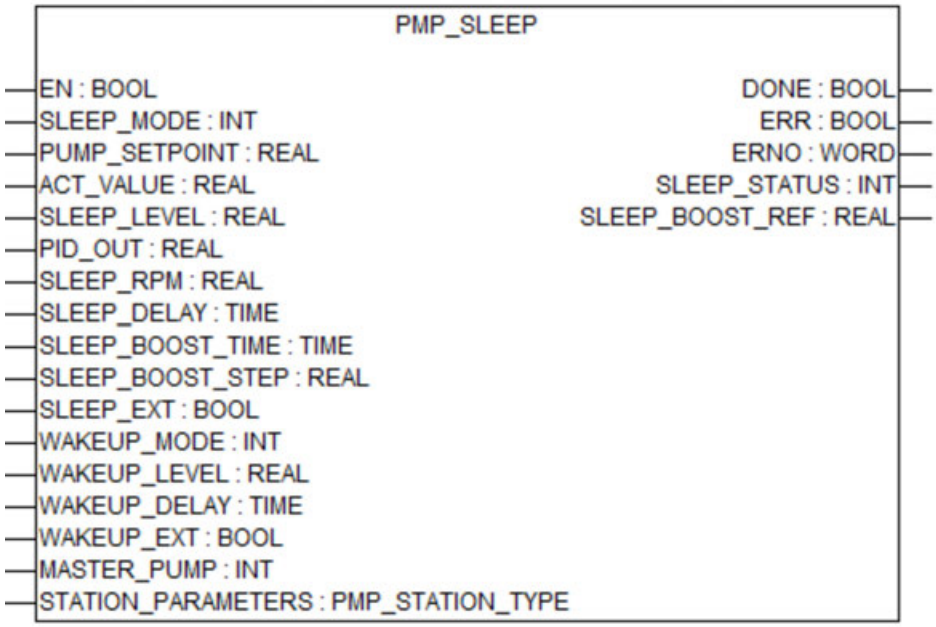
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16449	4041	Value in BINARY_SIGNAL_TYPE is less than 0
16450	4042	Value in BINARY_SIGNAL_TYPE is greater than 2

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.17 PMP_SLEEP



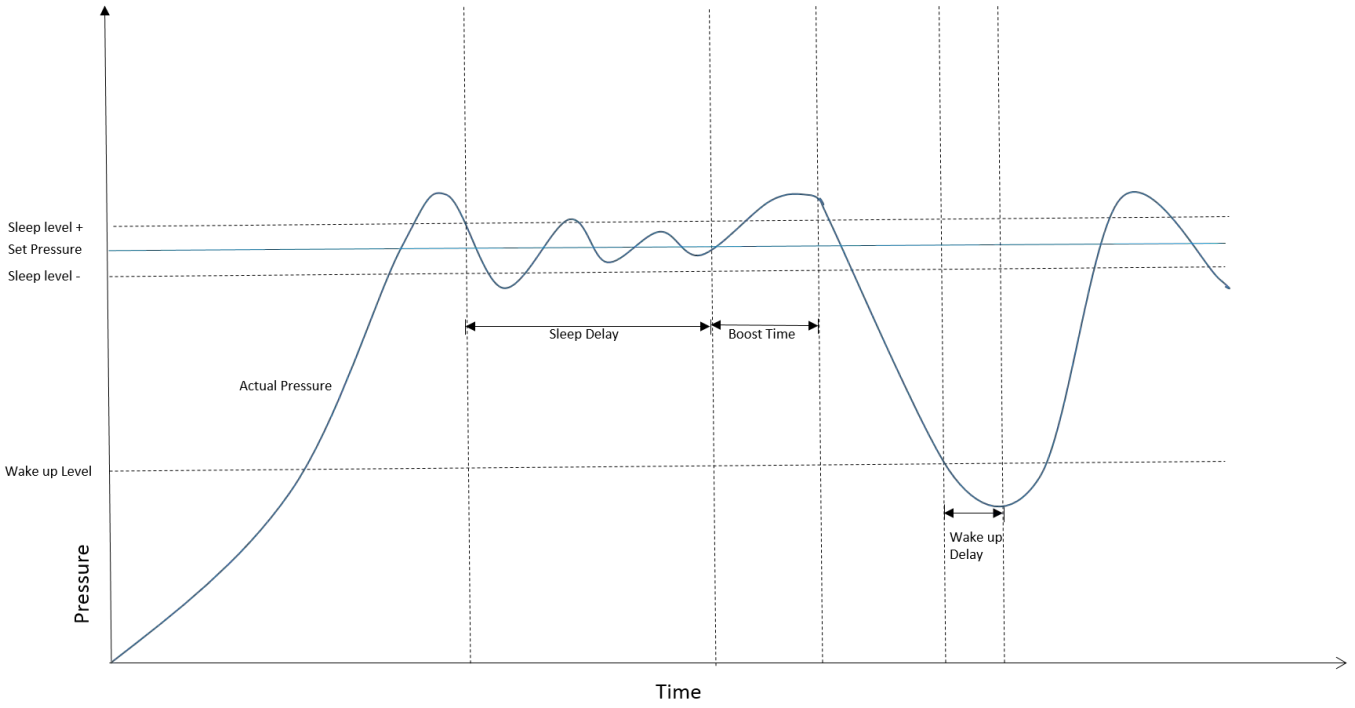
The function block PMP_SLEEP stops all pumps in the pressure control process during low demand. The function works only for pressure control (when function block PMP_CONFIGURATION has the input PROCESS_MODE = 1 (Pressure control) with multi pump station or traditional pump station.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Key features

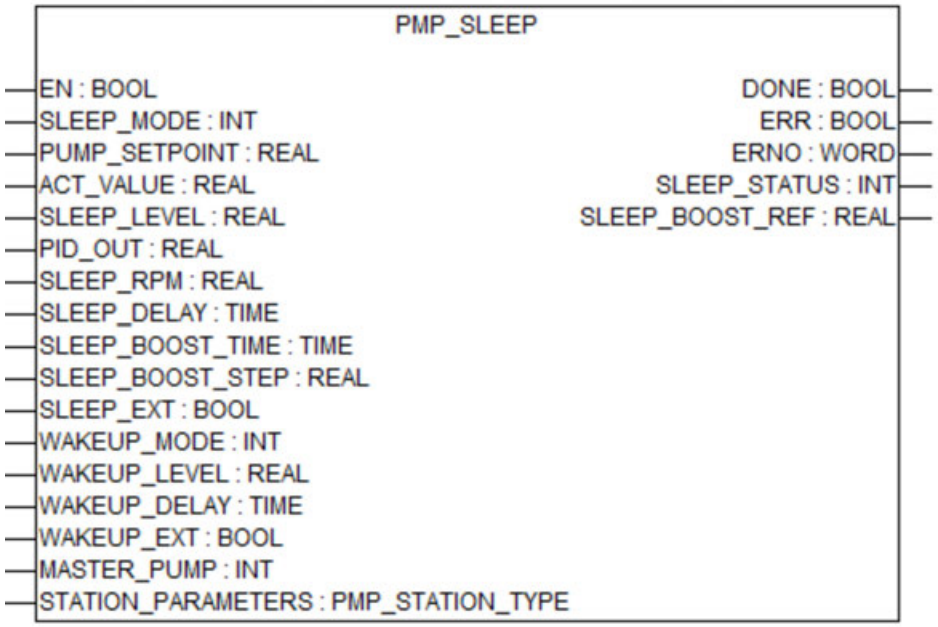
- The function block is used to conserve energy when the demand is very low and the pump is running at very less RPM
- For Sleep_Mode = 1
  - Absolute difference between set and actual pressure must be less than sleep level. Sleep level is defined as percentage of set point e.g. sleep level 10 means 10 % of set value.
  - Master pump RPM must be less than SLEEP_RPM.
  - If above both conditions are true for more than SLEEP_DELAY time then only pump gets ready to go to sleep.
  - Process increases set point by SLEEP_BOOST_STEP in terms of percentage for the time duration SLEEP_BOOST_TIME. This allows actual pressure to increase and in turn increases sleep duration.
  - Once SLEEP_BOOST_TIME is over, master pump goes in sleep and its ON command is removed.
  - When the actual pressure goes below WAKEUP_LEVEL then the wake up function starts. It waits for WAKEUP_DELAY and then the ON command is issued to the master drive and the pressure distributor process takes over.



This figure shows the sleep and wake up sequence based on the change in the actual pressure with time.

Sleep_Mode = 2 has not been implemented yet in this version.

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

SLEEP_MODE

Data type	Default value	Range	Unit
INT	0	1 ... 2	-

The input SLEEP_MODE selects the sleep mode.

0 = Disabled

1 = Internal

2 = External

PUMP_SETPOINT

Data type	Default value	Range	Unit
REAL	1.0	> 0	-

The input PUMP_SETPOINT defines the pressure setpoint. The value is same as the input SET_VALUE of the function block PMP_PID.

ACT_VALUE

Data type	Default value	Range	Unit
REAL	1.0	-	

The input ACT_VALUE defines the actual pressure value.

SLEEP_LEVEL

Data type	Default value	Range	Unit
REAL	5.0	1.0 ... 100.0	%

The input SLEEP_LEVEL defines the sleep activation level, which is a percentage of the input PUMP_SETPOINT.

PID_OUT

Data type	Default value	Range	Unit
REAL	0.0	-	%

The input PID_OUT of the function block PMP_SLEEP indicates the PID output in percentage. The value is the same as the output PID_OUT of the function blocks PMP_PID.

SLEEP_RPM

Data type	Default value	Range	Unit
REAL	100.0	> 0.0	rpm

The input SLEEP_RPM defines the speed in rpm below which the pump should go to sleep mode.

SLEEP_DELAY

Data type	Default value	Range	Unit
TIME	5	-	s

The input SLEEP_DELAY defines the delay time in seconds at which the sleep function must be activated.

SLEEP_ BOOST_TIME

Data type	Default value	Range	Unit
TIME	5	-	s

The input SLEEP_BOOST_TIME defines the time period in seconds for which the sleep boost must be operated.

SLEEP_ BOOST_STEP

Data type	Default value	Range	Unit
REAL	5.0		%

The input SLEEP_BOOST_STEP defines the step limit in percentage at which the input PUMP_SETPOINT must increase for the time duration defined in the input SLEEP_BOOST_TIME.

SLEEP_EXT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input SLEEP_EXT enables sleep function externally, when input SLEEP_MODE = 2 (External).

- When input is TRUE, sleep function is enabled externally.
- When input is FALSE, sleep function is disabled or continues in the current mode.

WAKEUP_MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input WAKEUP_MODE selects the wake up mode.

1 = Internal

2 = External

WAKEUP_LEVEL

Data type	Default value	Range	Unit
REAL	7.0	1.0 ... 100.0	%

The input WAKEUP_LEVEL defines the wake up activation level in percentage of the setpoint defined in the input PUMP_SETPOINT.

WAKEUP_DELAY

Data type	Default value	Range	Unit
TIME	5	-	s

The input WAKEUP_DELAY defines the delay time in seconds to activate the wake up function.

WAKEUP_EXT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input WAKEUP_EXT receives a digital signal to wake up the system when the input WAKEUP_MODE = 2 (External).

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

STATION_PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

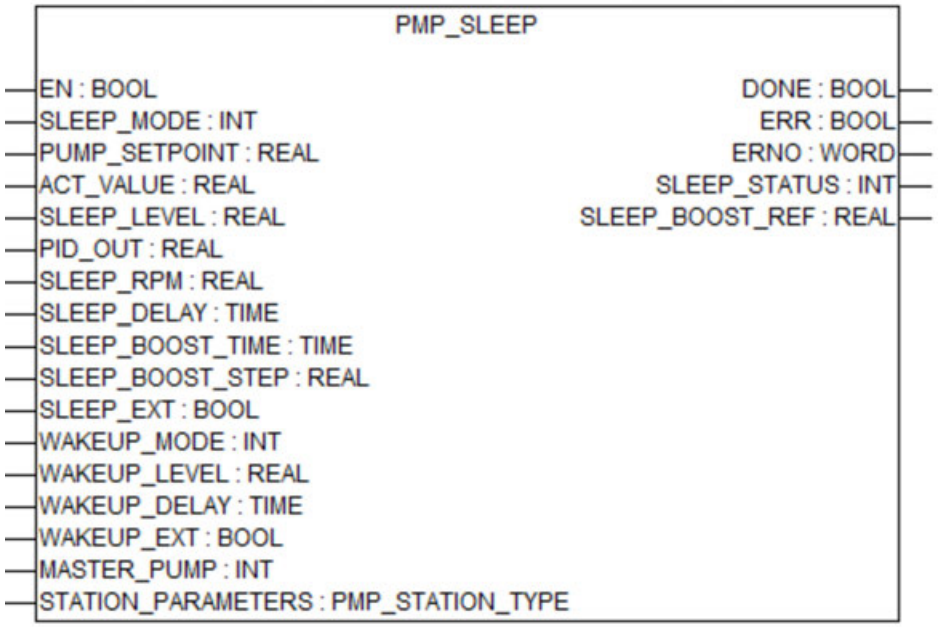
All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations

Parameter	Data type	Default value	Description
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combina-tion	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

SLEEP_STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 3	-

The output SLEEP_STATUS of the function block PMP_SLEEP indicates the sleep status of the pump.

0 = Inactive/Wakeup inactive

1 = Boost activated

2 = Sleep mode active

3 = Wakeup function active

SLEEP_ BOOST_REF

Data type	Default value	Range	Unit
REAL	0.0	> 0.0	-

The output SLEEP_BOOST_REF of the function block PMP_SLEEP indicates the sleep boost reference. This output is connected as input into PMP_PID function block.

Error codes

The error codes of function block PMP_SLEEP are listed below:

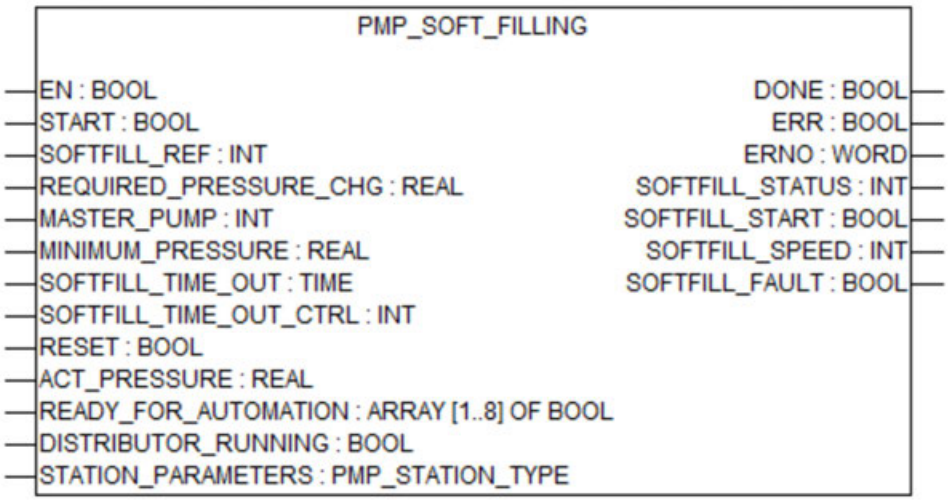
Error code		Error Description
Decimal	Hexadecimal	
0	0000	No error occurred
16403	4013	Sleep function is active only when input PROCESS_MODE = 1 in function block PMP_CONFIGURATION
16417	4021	Value of SLEEP_MODE is less than 0
16418	4022	Value of SLEEP_MODE is more than 2
16433	4031	Value of PUMP_SETPOINT is less than 0
16465	4051	Value of SLEEP_LEVEL is less than or equal to 0
16497	4071	Value of SLEEP_RPM is less than 0
16577	40C1	Value of WAKEUP_MODE is less than 0
16578	40C2	Value of WAKEUP_MODE is greater than 2
16593	40D1	Value of WAKEUP_LEVEL is less than 0.0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.18 PMP_SOFT_FILLING



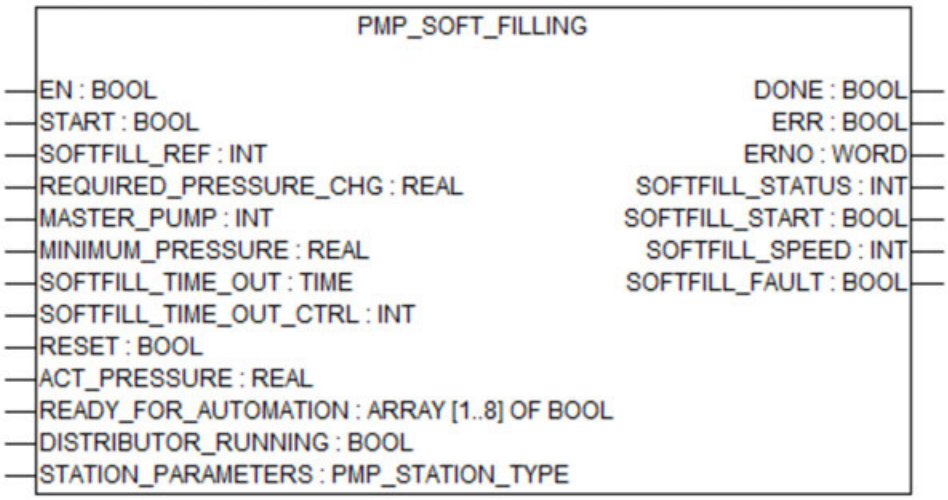
The function block PMP_SOFT_FILLING enables slow filling operation in pipes in the beginning of the process to avoid damages. If this operation is selected, it is executed before the actual process control takes over.

This function block cannot run when the PUMP_COMB = 3, DOL mode in PMP_CONFIGURATION. Soft filling needs the VFD drive to run and execute gradual filling of water.

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START receives the soft filling start command from the output SOFTFILL_START of the function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When input is TRUE, the soft filling starts.
- When input is FALSE, the soft filling stops if running.

SOFTFILL_REF

Data type	Default value	Range	Unit
INT	100	0 ... 20 % of master pump nominal speed	rpm

The input SOFTFILL_REF defines the softfill speed reference in rpm.

REQUIRED_ PRESSURE_ CHG

Data type	Default value	Range	Unit
REAL	0.0	0.0 ... 100.0	%

The input REQUIRED_PRESSURE_CHG defines the percentage change in the actual pressure during soft filling operation.

MASTER_PUMP

Data type	Default value	Range	Unit
INT	1	1 ... 8	-

The input MASTER_PUMP receives the identification number of the master pump in the sequence. The input comes from the output MASTER_PUMP of the function block PMP_SEQUENCE_GEN.

MINIMUM_ PRESSURE

Data type	Default value	Range	Unit
REAL	0.0	≥ 0	-

The input MINIMUM_PRESSURE defines the minimum pressure above which the soft filling operation is disabled.

SOFTFILL_ TIME_OUT

Data type	Default value	Range	Unit
TIME	10	-	s

The input SOFTFILL_TIME_OUT defines the time limit in seconds after which the soft filling operation is stopped, regardless of the minimum pressure limit in input MINIMUM_PRESSURE.

SOFTFILL_ TIME_OUT_ CTRL

Data type	Default value	Range	Unit
INT	1	1 ... 2	s

The input SOFTFILL_TIME_OUT_CTRL selects the pump behavior when softfill operation stopped because of input SOFTFILL_TIME_OUT. The function will stop the process if already running. This indicates a leakage in the pipe and due to which the minimum pressure is not established.

1 = Keep running normal PID operation

2 = Fault trips the station

RESET

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The input RESET resets the output SOFTFILL_FAULT.

- When output is TRUE, softfill fault is reset.
- When output is FALSE, no reset.

ACT_ PRESSURE

Data type	Default value	Range	Unit
REAL	0.0	-	-

The input ACT_PRESSURE receives the actual pressure in physical units, e.g. bar, as measured by the pressure sensor.

READY_FOR_AUTOMATION

Data type	Default value	Range	Unit
ARRAY[1..8] of BOOL	FALSE	TRUE/FALSE	-

The input READY_FOR_AUTOMATION receives an array of pumps ready for automatic operation. The input comes from the output READY_FOR_AUTOMATION of the function block PMP_INTERFACE_VFD or PMP_INTERFACE_DOL.

The READY_FOR_AUTOMATION[1] connects to the PUMP ID = 1. This pattern must be followed for all pump IDs.

- When input is TRUE, pump is ready for operation in automatic mode.
- When input is FALSE, pump is not ready for operation in automatic mode.

DISTRIBUTOR_RUNNING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input DISTRIBUTOR_RUNNING indicates the running status of the distributor.
This input is connected to the output DISTRIBUTOR_RUNNING of any of the the distributor function blocks PMP_LEVEL_DISTRIBUTOR, PMP_FLOW_DISTRIBUTOR or PMP_PRESSURE_DISTRIBUTOR.

- When output is TRUE, distributor is running.
- When output is FALSE, distributor is not running.

STATION_PARAMETERS

Data type	Default value	Range	Unit
PMP_STATION_TYPE	-	-	-

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

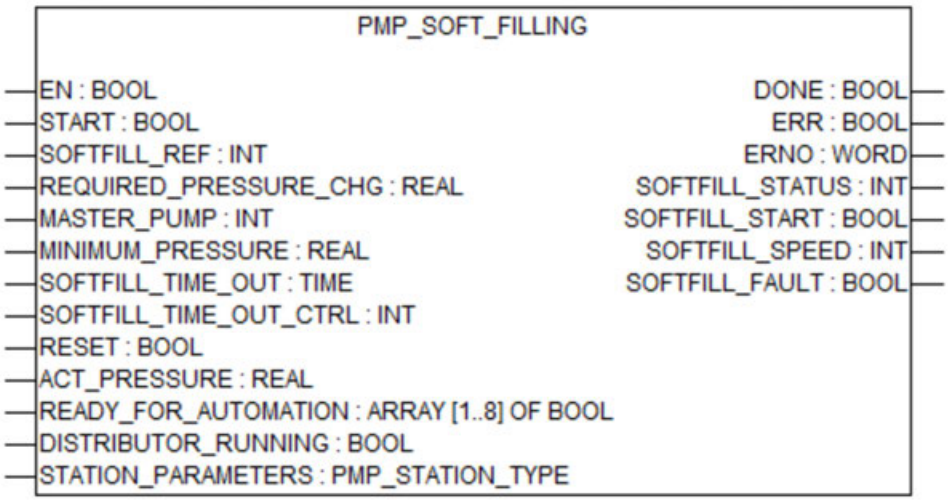
All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control

Parameter	Data type	Default value	Description
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

SOFTFILL_ STATUS

Data type	Default value	Range	Unit
INT	0	0 ... 4	-

The input SOFTFILL_STATUS indicates the status of softfilling function.

0 = Disable the softfill (Softfill operation is disabled)

1 = Ready to start (Softfill operation is ready to start)

2 = Softfill in progress (Softfill operation has started)

3 = Softfill completed (Softfill operation is completed)

4 = Softfill with fault (Softfill operation stopped due to a fault)

SOFTFILL_ START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_START of the used pump distributor starts the soft filling operation. This function is effective only when this output is connected to the input START of the function block PMP_SOFT_FILLING.

- When input is TRUE, softfill function starts.
- When input is FALSE, softfill function stops if running.

SOFTFILL_ SPEED

Data type	Default value	Range	Unit
INT	0	> 0	rpm

The output SOFTFILL_SPEED of the function block PMP_SOFT_FILLING indicates the speed reference for soft filling function. The output must be connected to the input SOFTFILL_SPEED of the interface function block PMP_INTERFACE_VFD.

SOFTFILL_ FAULT

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output SOFTFILL_FAULT indicates a fault in the soft filling function. The fault occurs when actual pressure in input ACT_PRESSURE has not reached PID_ENABLE_PRESSURE in the time limit defined in the input SOFTFILL_TIME_OUT.

- When output is TRUE, softfill fault occurred.
- When input is FALSE, no fault occurred.

Error codes

The error codes of function block PMP_SOFT_FILLING are listed below:

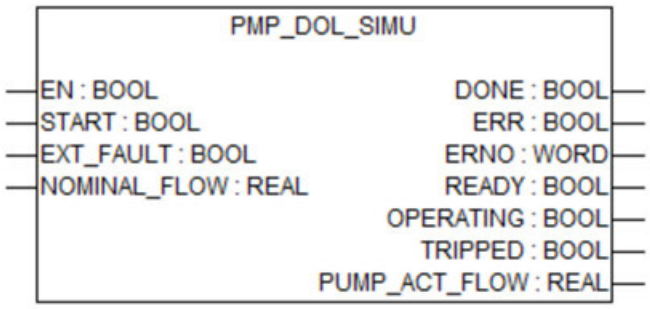
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16403	4013	Softfill function is incorrectly activated when the PUMP_COMB = 3, DOL type station in PMP_CONFIGURATION
16433	4031	Value of SOFTFILL_REF is less than 0
16435	4033	SOFTFILL_REF is greater than 20 % of master pumps nominal speed
16449	4041	Value of REQUIRED_PRESSURE_CHG is less than 0.0
16465	4051	Value of MASTER_PUMP is less than 1
16466	4052	Value of MASTER_PUMP is more than NUMBER_OF_PUMPS in the PMP_CONFIGFURATION function block
16481	4061	Value of MINIMUM_PRESSURE is less than 0.0
16513	4081	Value of SOFTFILL_TIME_OUT_CTRL is less than 1
16514	4082	Value of SOFTFILL_TIME_OUT_CTRL is greater than 2

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.19 PMP_DOL_SIMU



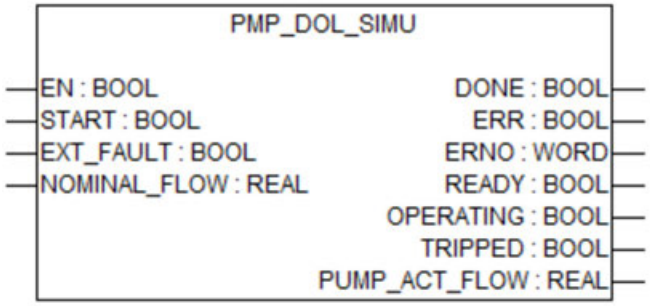
The function block PMP_DOL_SIMU is a simple simulation of the direct online pumps.

> *The outputs of the function block may or may not be as accurate as real process. Use this function block only for simulation testing and not on the actual system.*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

EXT_FAULT

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The input EXT_FAULT simulates the external fault condition.

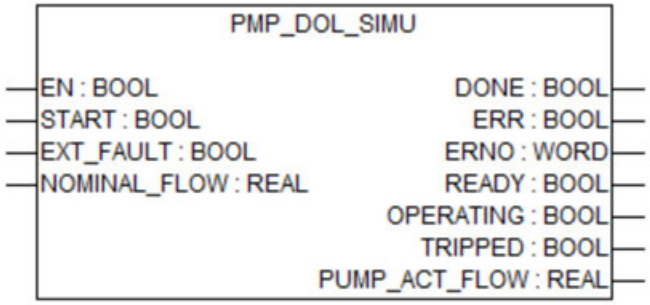
- When input is TRUE, external fault condition is simulated.
- When input is FALSE, no simulation.

NOMINAL_ FLOW

Data type	Default value	Range	Unit
REAL	100.0	> 0.0	m^3/h

The input NOMINAL_FLOW defines the nominal flow of the direct online (DOL) pump in cubic meters per hour.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

READY

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output READY indicates that the used system (DOL/VFD) is ready for operation.

- When output is TRUE, system is ready for operation.
- When output is FALSE, system is not ready for operation.

OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output OPERATING indicates that the device is operating.

- When output is TRUE, device is operating.
- When output is FALSE, device is not operating.

TRIPPED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output TRIPPED indicates that the used system (DOL/VFD) has tripped.

- When output is TRUE, system tripped.
- When output is FALSE, system is healthy.

PUMP_ACT_FLOW

Data type	Default value	Range	Unit
REAL	0.0	-	m^3/h

The output PUMP_ACT_FLOW indicates the actual flow of direct online pumps in cubic meters per hour.

Error codes

The error codes of function block PMP_DOL_SIMU are listed below:

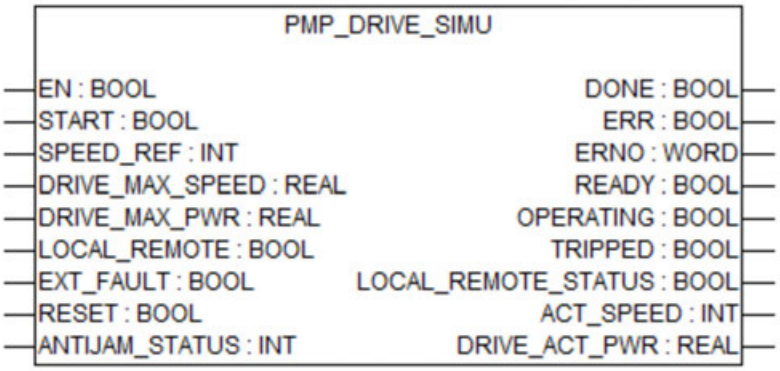
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16449	4041	Value in input NOMINAL_FLOW is less than or equal to 0.0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.20 PMP_DRIVE_SIMU



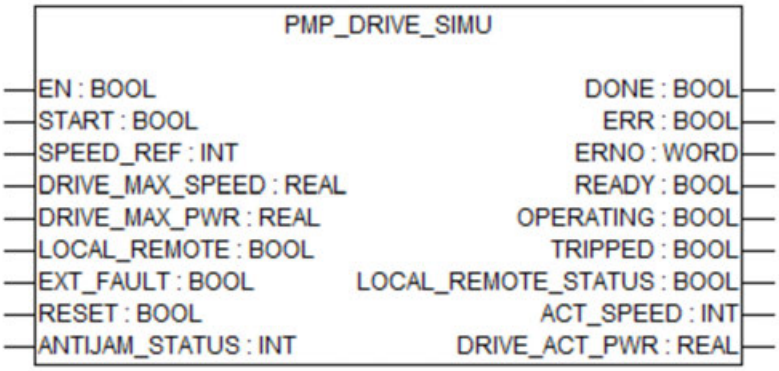
The function block PMP_DRIVE_SIMU is used to simulate the variable frequency drive (VFD) pumps and for PLC communication.

- *The function block is not similar to the PS553 drives library function block.*
- *The outputs of the function block may or may not be as accurate as real drive communication blocks.*
- *Use this function block only for simulation testing and not on the actual system.*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

START

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input START starts the execution of the function block.

- When input is TRUE, the execution starts.
- When input is FALSE, the execution stops if running.

SPEED_REF

Data type	Default value	Range	Unit
INT	0	> 0	rpm

The output SPEED_REF of the interface function block PMP_INTERFACE_VFD shows the speed reference of the variable frequency drive (VFD) in rpm.

> *The speed reference can be given to the drive only if it is connected to the correct parameter.*
>
> *This can be done using the PLC - VFD communication library, which is not in the scope of the pump library.*

The input SPEED_REF of the function block PMP_DRIVE_SIMU comes from the output SPEED_REF of the interface function block PMP_INTERFACE_VFD.

DRIVE_MAX_SPEED

Data type	Default value	Range	Unit
REAL	1500.0	1.0 ... 1500.0	rpm

The input DRIVE_MAX_SPEED defines the maximum speed of the variable frequency drive (VFD) in rpm.

DRIVE_MAX_POWER

Data type	Default value	Range	Unit
REAL	5	0.1 ... 5.0	kW

The input DRIVE_MAX_POWER defines the maximum power of the variable frequency drive (VFD) in kW.

LOCAL_REMOTE

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input LOCAL_REMOTE selects the variable frequency drive (VFD) operating mode.

- When input is TRUE, VFD is in local mode.
- When input is FALSE, VFD is in remote mode.

EXT_FAULT

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The input EXT_FAULT simulates the external fault condition.

- When input is TRUE, external fault condition is simulated.
- When input is FALSE, no simulation.

RESET

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

The input RESET resets the variable frequency drive (VFD) fault and the value in output ERNO.

- When input is TRUE, softfill fault is reset.
- When input is FALSE, no reset.

ANTIJAM_STATUS

Data type	Default value	Range	Unit
ARRAY[1..8] of INT	0	0 ... 2	-

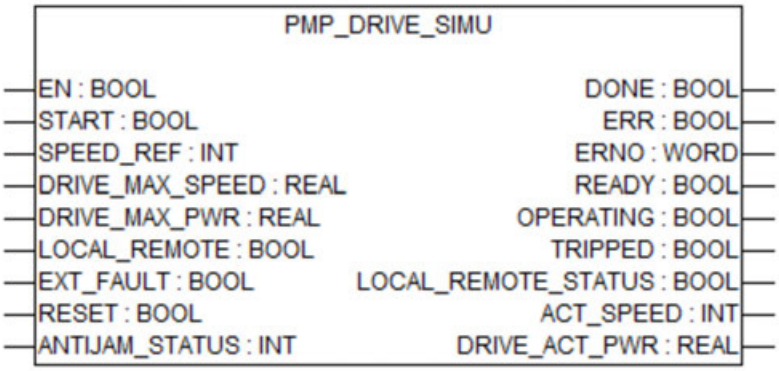
The input ANTIJAM_STATUS receives the status of antijam operation. The input comes from the output ANTIJAM_STATUS of the function block PMP_ANTIJAM.

0 = Not started (Antijam operation has not started)

1 = Busy (Antijam operation has started)

2 = Done (Antijam operation is completed)

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

READY

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output READY indicates that the used system (DOL/VFD) is ready for operation.

- When output is TRUE, system is ready for operation.
- When output is FALSE, system is not ready for operation.

OPERATING

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output OPERATING indicates that the device is operating.

- When output is TRUE, device is operating.
- When output is FALSE, device is not operating.

TRIPPED

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output TRIPPED indicates that the used system (DOL/VFD) has tripped.

- When output is TRUE, system tripped.
- When output is FALSE, system is healthy.

LOCAL_ REMOTE_ STATUS

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The output LOCAL_REMOTE_STATUS indicates the operating mode of the pump system (DOL/VFD).

- When output is TRUE, pump system is in local mode.
- When output is FALSE, pump system is in remote mode.

ACT_SPEED

Data type	Default value	Range	Unit
INT	10	> 1	rpm

The input ACT_SPEED indicates the actual speed of the variable frequency drive (VFD) in rpm.

DRIVE_ACT_ PWR

Data type	Default value	Range	Unit
REAL	0.0	> 0.0	kW

The output DRIVE_ACT_PWR indicates the actual power of the variable frequency drive (VFD) in kW.

Error codes

The error codes of function block PMP_DRIVE_SIMU are listed below:

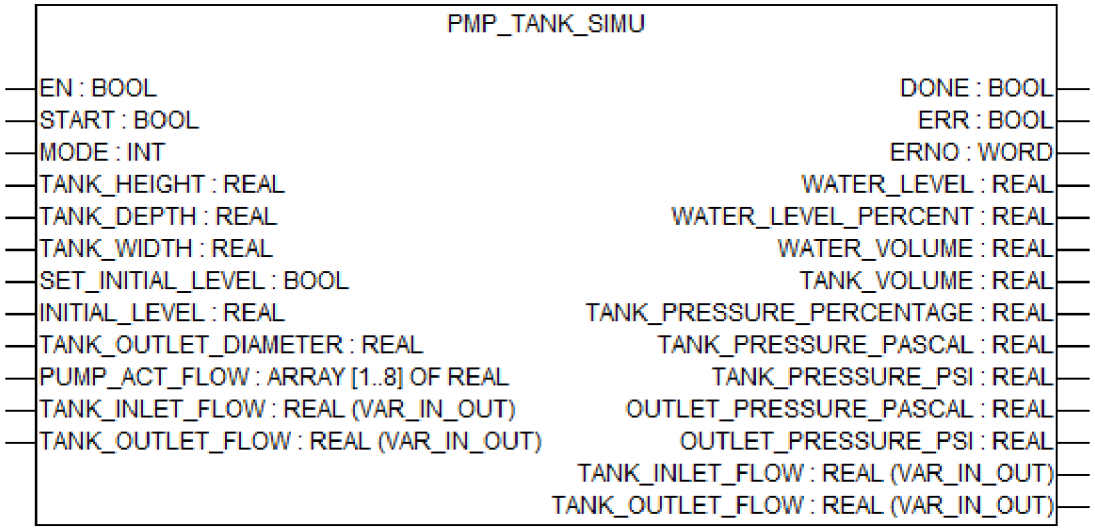
Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16449	4041	Value in input DRIVE_MAX_SPEED is less than or equal to 0
16465	4051	Value in input DRIVE_MAX_PWR is less than or equal to 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.2.21 PMP_TANK_SIMU



The function block PMP_TANK_SIMU is used to simulate the water tank.

This simulation can be used in all the three process control modes:

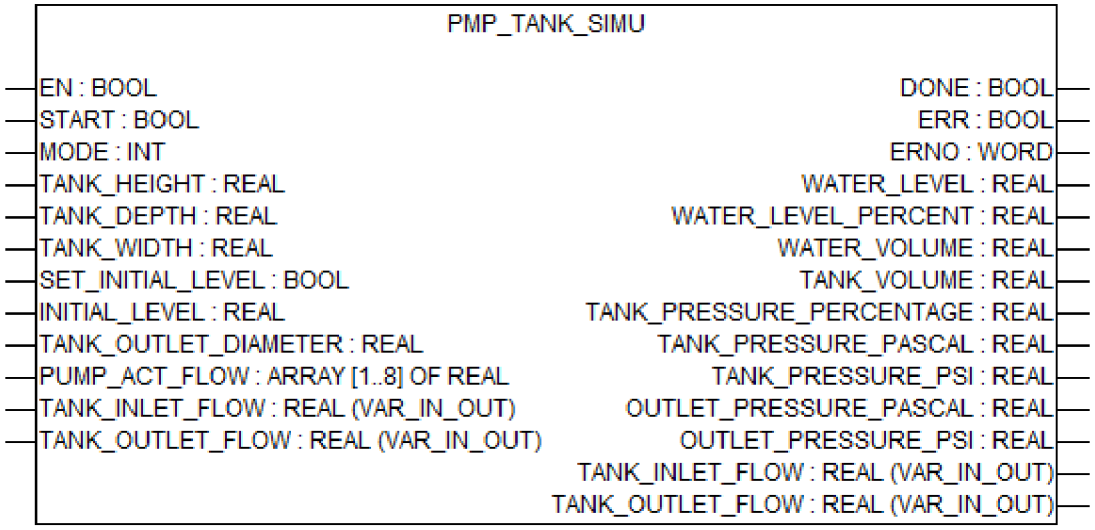
- Pressure
- Flow
- Level

> *Use this function block only for simulation testing and not with critical hardware in loop.*

General information

Runtime system version	V2.5 and later
Library version	**PMP_AC500_V25.lib**
Function block type	Function block without historical values

Input description



EN

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input EN has to be driven in level triggered mode.

In order to enable the execution of the function block the input EN has to be set to value TRUE. It keeps on executing until the input EN is set back to value FALSE.

While it is executed its inputs are continuously evaluated.

MODE

Data type	Default value	Range	Unit
INT	1	1 ... 2	-

The input MODE selects the pumping operation mode.

1 = Emptying.

2 = Filling.

TANK_HIGHT

Data type	Default value	Range	Unit
REAL	20.0	> 1.0	m

The input TANK_HEIGHT defines the height of the tank in meters.

TANK_DEPTH

Data type	Default value	Range	Unit
REAL	10.0	> 1.0	m

The input TANK_DEPTH defines the depth of the tank in meters.

TANK_WIDTH

Data type	Default value	Range	Unit
REAL	5.0	> 1.0	m

The input TANK_WIDTH defines the width of the tank in meters.

SET_INITIAL_LEVEL

Data type	Default value	Range	Unit
BOOL	FALSE	TRUE/FALSE	-

The input SET_INITIAL_LEVEL allows to set the initial level for tank simulation.

- When input is TRUE, initial level setting for tank simulation is allowed.
- When input is FALSE, initial level setting for tank simulation is not allowed.

INITIAL_LEVEL

Data type	Default value	Range	Unit
REAL	10.0	0.0 ... TANK_HEIGHT	m

The input INITIAL_LEVEL defines the initial level of tank in meters.

TANK_OUTLET_DIAMETER

Data type	Default value	Range	Unit
REAL	0.25	> 1.00	m

The input TANK_OUTLET_DIAMETER defines the outlet diameter of the tank pipe in meters.

PUMP_ACT_FLOW

Data type	Default value	Range	Unit
ARRAY[1..8] of REAL	-	> 0.0	m^3/h

The input PUMP_ACT_FLOW is an array of actual flow in cubic meters per hour.

At the first index of array PUMP_ACT_FLOW[1] connect the actual flow of the pump with PUMP ID = 1.

This pattern must be followed for all pump IDs.

TANK_INLET_FLOW

Data type	Default value	Range	Unit
REAL	-	-	m^3/h

The variable input/output TANK_INLET_FLOW defines the inlet flow of the tank in cubic meters per hour.

The inflow is assumed to be constant.

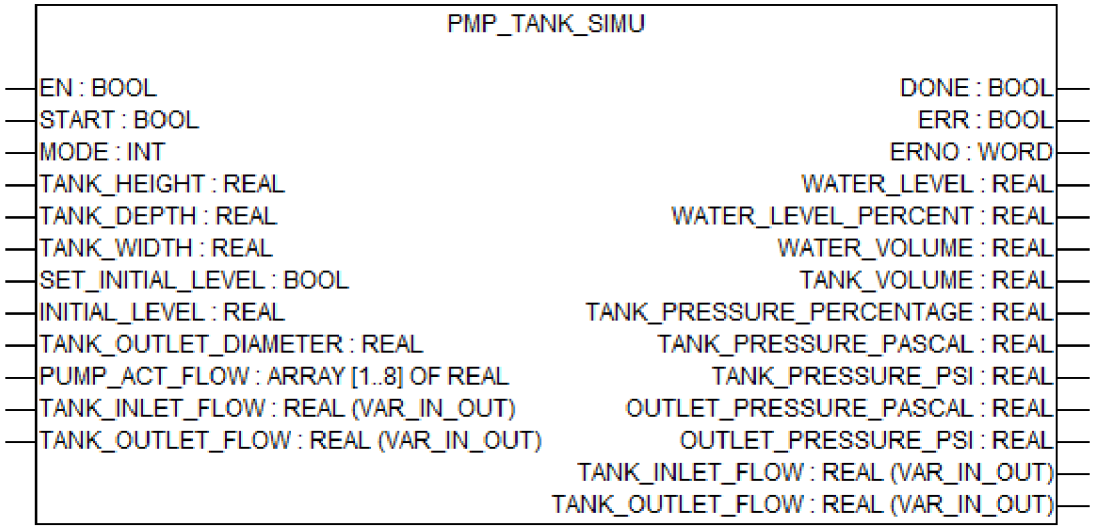
TANK_OUTLET_FLOW

Data type	Default value	Range	Unit
REAL	-	-	m^3/h

The variable input/output TANK_OUTLET_FLOW defines the outlet flow of the tank in cubic meters per hour.

The outflow is assumed to be constant.

Output description



DONE

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output DONE indicates the state of execution.

It has the value TRUE when the execution is finished and the outputs are valid. When DONE becomes FALSE other outputs are not valid any more.

ERR

Data type	Default value	Range	Unit
BOOL	-	TRUE/FALSE	-

Output ERR indicates whether an error occurred during function block execution.

It is only valid when DONE is TRUE. The kind of occurred error is specified by the error identifier at output ERNO.

ERNO

Data type	Default value	Range	Unit
WORD	-	0 ... 65535	-

At output ERNO the identifier of an occurred error is provided. It is only valid when DONE and ERR is TRUE (see *Chapter 1.7.3.5 “Error messages of the AC500 V2 function block libraries” on page 6529*).

WATER_LEVEL

Data type	Default value	Range	Unit
REAL	0.0	-	m

The output WATER_LEVEL indicates the water level of the tank in meters.

WATER_LEVEL_PERCENT

Data type	Default value	Range	Unit
REAL	0.0	-	%

The output WATER_LEVEL_PERCENT indicates the percentage of water level in the tank.

WATER_ VOLUME

Data type	Default value	Range	Unit
REAL	0.0	-	m^3

The output WATER_VOLUME indicates the actual water volume in cubic meters.

TANK_VOLUME

Data type	Default value	Range	Unit
REAL	0.0	-	m^3

The output TANK_VOLUME indicates the total tank volume in cubic meters.

TANK_ PRESSURE_ PERCENTAGE

Data type	Default value	Range	Unit
REAL	0.0	0.0 ... 100	%

The output TANK_PRESSURE_PERCENTAGE indicates the pressure at the bottom of the tank in percentage.

TANK_ PRESSURE_ PASCAL

Data type	Default value	Range	Unit
REAL	0.0	-	Pa

The output TANK_PRESSURE_PASCAL indicates the pressure at the bottom of the tank in Pascal.

TANK_ PRESSURE_PSI

Data type	Default value	Range	Unit
REAL	0.0	-	psi

The output TANK_PRESSURE_PSI indicates the pressure at the bottom of the tank in psi (pounds per square inch).

OUTLET_ PRESSURE_ PASCAL

Data type	Default value	Range	Unit
REAL	0.0	-	Pa

The output OUTLET_PRESSURE_PASCAL indicates the pressure at the outlet pipe in Pascal.

OUTLET_ PRESSURE_PSI

Data type	Default value	Range	Unit
REAL	0.0	-	psi

The output OUTLET_PRESSURE_PSI indicates pressure at the outlet pipe in PSI.

Error codes

The error codes of function block PMP_TANK_SIMU are listed below:

Error code		Error Description
Decimal	**Hexadecimal**	
0	0000	No error occurred
16417	4021	Value in input MODE is less than 1
16418	4022	Value in input MODE is greater than 2
16433	4031	Value in input TANK_HEIGHT is less than or equal to 0
16449	4041	Value in input TANK_DEPTH is less than or equal to 0
16465	4051	Value in input TANK_WIDTH is less than or equal to 0
16497	4071	Value in input INITIAL_LEVEL is less than 0
16499	4073	Value in input INITIAL_LEVEL is greater than TANK_HEIGHT
16513	4081	Value in input TANK_OUTLET_DIAMETER is less than or equal to 0
16529	4091	Value in input PUMP1_ACT_FLOW is less than 0

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.5.14.3 Structures

PMP_STATION_TYPE

The structure PMP_STATION_TYPE contains information of pump station configuration parameters.

All function blocks will receive some data, process it and write it back to the structure.

See the parameters listed below:

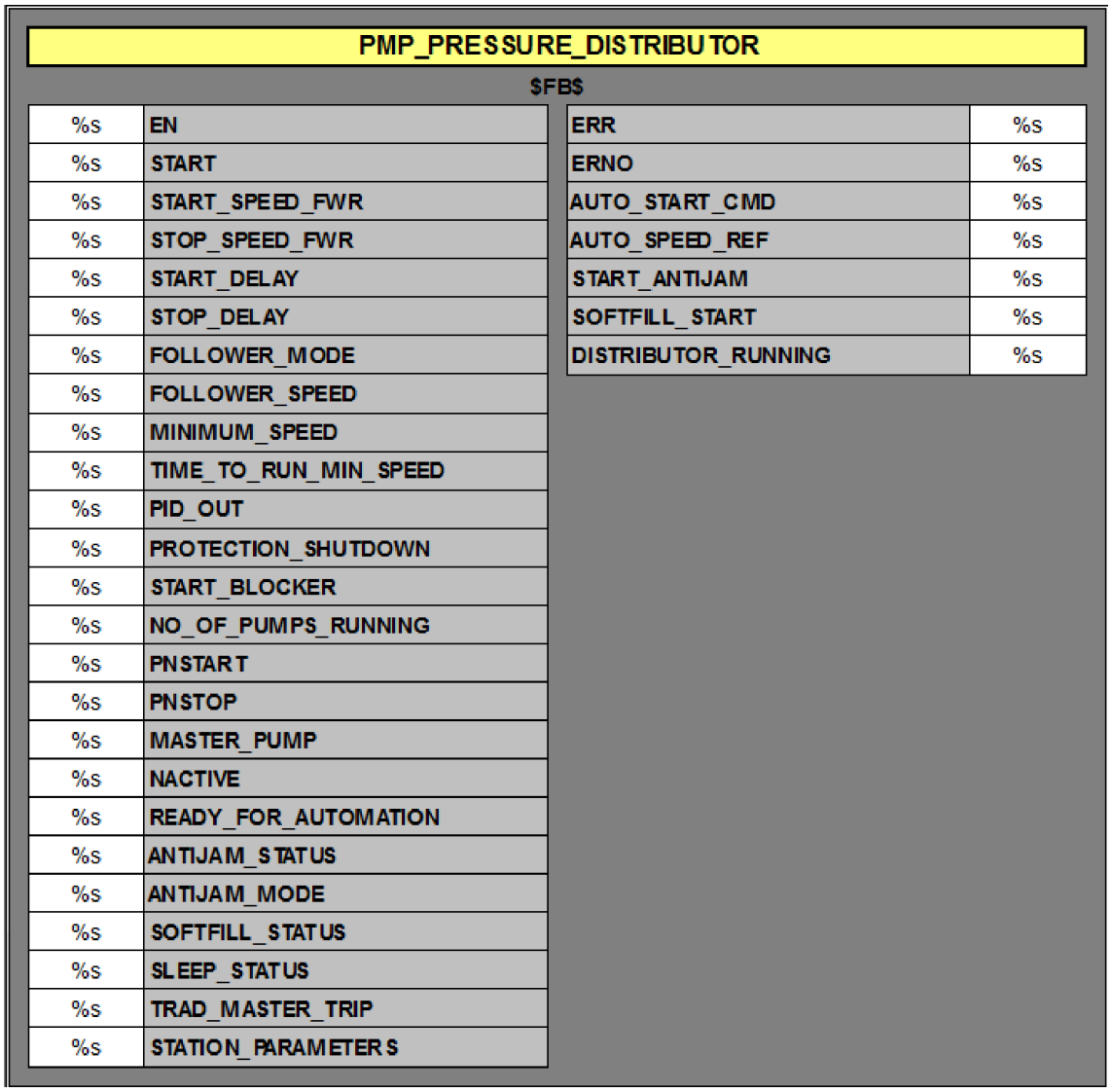
Parameter	Data type	Default value	Description
usiStation_Number	USINT	1	Corresponding station number
sStation_Name	STRING(20)	Station 1	Station name for visualizations
iNumber_of_Pumps	INT	1	Number of pumps in the pumping station
iProcess_Mode	INT	1	Selected process mode: 1 - Pressure control 2 - Flow control 3 - Level control
iPump_Combination	INT	1	Pump combination mode: 1 - Multimode 2 - Traditional mode 3 - DOL mode
aiNominalSpeed	ARRAY[1..8] of INT	8 (1500)	Array to store pump nominal speed values

1.5.14.4 Visualization

PMP_PRESSURE_DISTRIBUTOR_VISU_PH

Visualization element PMP_PRESSURE_DISTRIBUTOR_VISU_PH can be used to show the actual values of all inputs and outputs of the PMP_PRESSURE_DISTRIBUTOR function block. The visualization could also be used to control the function block by those inputs which are not connected inside the program.

The following figure demonstrates visualization in the offline mode:



For all the other function blocks in the library, similar visualizations are available.

All these visualization works same like the function block visualization defined above.

1.6 PLC integration (hardware)

1.6.1 PLC introduction

1.6.1.1 Safety instructions

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variants and requirements associated with any particular installation, ABB cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by ABB with respect to use of information, circuits, equipment or software described in this manual. No liability is assumed for the direct or indirect consequences of the improper use, improper application or inadequate maintenance of these devices. In no event will ABB be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

PLC specific safety notices

The product family AC500 control system is designed according to EN 61131-2 IEC 61131-2 standards. Data, different from IEC 61131, are caused by the higher requirements of Maritime Services. Other differences are described in the technical data description of the devices.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

NOTICE!

PLC damage due to operation conditions

Protect the devices from dampness, dirt and damage during transport, storage and operation!

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

Cleaning instruction

Do not use cleaning agent for cleaning the device.

Use a damp cloth instead.

Connection plans and user software must be created so that all technical safety aspects, legal regulations and standards are observed. In practice, possible shortcircuits and breakages must not be able to lead to dangerous situations. The extent of resulting errors must be kept to a minimum.

Do not operate devices outside of the specified, technical data!

Trouble-free functioning cannot be guaranteed outside of the specified data.

NOTICE!

PLC damage due to missing grounding

– Ensure to earth the devices.
– The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

In the description for the devices (operating manual or AC500 system description), reference is made at several points to grounding, galvanic isolation and EMC measures. One of the EMC measures consists of discharging interference voltages into the grounding via Y-type capacitors. Capacitor discharge currents must basically be able to flow off to the grounding (in this respect, see also VBG 4 and the relevant VDE regulations).

CAUTION!

Do not obstruct the ventilation for cooling!

The ventilation slots on the upper and lower side of the devices must not be covered.

CAUTION!

Run signal and power wiring separately!

Signal and supply lines (power cables) must be laid out so that no malfunctions due to capacitive and inductive interference can occur (EMC).

WARNING!

Labels on or inside the device alert people that dangerous voltage may be present or that surfaces may have dangerous temperatures.

WARNING!

Splaying of strands can cause hazards!

During wiring of terminals with stranded conductors, splaying of strands shall be avoided.

– Ferrules can be used to prevent splaying.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

Information on batteries

CAUTION!

Use only ABB approved lithium battery modules!

At the end of the battery's lifetime, always replace it only with a genuine battery module.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Environment considerations

Recycle exhausted batteries. Dispose batteries in an environmentally conscious manner, in accordance to local-authority regulations.

Environment and enclosure information

This equipment is intended for use in a Pollution Degree 2 industrial environment, in overvoltage Category II applications (as defined in IEC publication 60664-1), at altitudes up to 2.000 meters without derating.

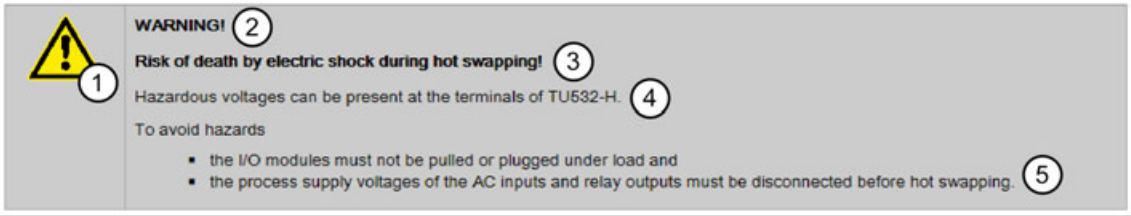
This equipment is considered Group 1, Class A industrial equipment according to IEC/CISPR Publication 11. Without appropriate precautions, there may be potential difficulties ensuring electromagnetic compatibility in other environments due to conducted as well as radiated disturbance.

This equipment is supplied as "open type" equipment. It must be mounted within an enclosure that is suitably designed for those specific environmental conditions that will be present and appropriately designed to prevent personal injury resulting from accessibility to live parts. The interior of the enclosure must be accessible only by the use of a tool. Subsequent sections of this publication may contain additional information regarding specific enclosure type ratings that are required to comply with certain product safety certifications.

Refer to NEMA Standards publication 250 and IEC publication 60529, as applicable, for explanations of the degrees of protection provided by different types of enclosure. Also see the appropriate sections in this manual.

1.6.1.1.1 Safety notice

Throughout the documentation we use the following types of safety and information notices according to ANSI Z535 make you aware of safety considerations or advice on AC500 products usage.



1 **Safety alert symbol** indicates the danger.
2 **Signal word** classifies the danger.
3 **Type and source of the risk** are mentioned.
4 **Possible consequences** of the risk are described.
5 **Measures to avoid these consequences** (enumerations).

Signal words

DANGER!
DANGER indicates a hazardous situation which, if not avoided, will result in death or serious injury.

Ensure to take measures to prevent the described impending danger.

WARNING!
WARNING indicates a hazardous situation which, if not avoided, could result in death or serious injury.

Ensure to take measures to prevent the described dangerous situation.

CAUTION!
CAUTION indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

Ensure to take measures to prevent the described dangerous situation.

NOTICE!
NOTICE is used to address practices not related to physical injury but might lead to property damage for example damage of the product.

Ensure to take measures to prevent the described dangerous situation.

NOTE provides additional information on the product, e.g. advices for configuration or best practice scenarios.

1.6.1.2 Cyber security

Cyber security disclaimer

This product is designed to be connected to and to communicate information and data via a network interface. It is your sole responsibility to provide and continuously ensure a secure connection between the product and your network or any other network (as the case may be). You shall establish and maintain any appropriate measures (such as but not limited to the installation of firewalls, application of authentication measures, encryption of data, installation of anti-virus programs, etc.) to protect the product, the network, its system and the interface against any kind of security breaches, unauthorized access, interference, intrusion, leakage and/or theft of data or information. ABB Ltd and its affiliates are not liable for damages and/or losses related to such security breaches, any unauthorized access, interference, intrusion, leakage and/or theft of data or information.

Although ABB provides functionality testing on the products and updates that we release, you should institute your own testing program for any product updates or other major system updates (to include but not limited to code changes, configuration file changes, third party software updates or patches, hardware exchanges, etc.) to ensure that the security measures that you have implemented have not been compromised and system functionality in your environment is as expected. This also applies to the operating system. Security measures (such as but not limited to the installation of latest patches, installation of firewalls, application of authentication measures, installation of anti-virus programs, etc.) are in your responsibility. You have to be aware that operating systems provide a considerable number of open ports that should be monitored carefully for any threats.

It has to be considered that online connections to any devices are not secured. It is your responsibility to assure that connections are established to the correct device (and e.g. not to an unknown device pretending to be a known device type). Furthermore you have to take care that confidential data exchanged with the PLC is either compiled or encrypted.

Security related deployment guidelines for industrial automation

Security details for industrial automation is provided in a *whitepaper* on ABB website.

Secure communication

Whenever possible, use an encrypted communication between AC500 V3 devices and third party devices, such as HMI devices. This is necessary to protect passwords and other data.

Frequently asked questions

For more information around cyber security please see our *FAQ*.

1.6.1.2.1 Defense in depth

The defense in depth approach implements multi-layer IT security measures. Each layer provides its special security measures. All deployed security mechanisms in the system must be updated regularly. It is also important to follow the system vendor's recommendations on how to configure and use these mechanisms. As a basis, the components must include security functions such as:

- Virus protection
- Firewall protection
- Strong and regularly changed passwords
- User management
- Using VPN tunnels for connections between networks

Additional security components such as routers and switches with integrated firewalls should be available. A defined user and rights concept managing access to the controllers and their networks is mandatory. Finally, the manufacturer of the components should be able to quickly discover weaknesses and provide patches.

Only used services/ports should be enabled (e.g. to enable the functionality of an FTPS server).

References: *CODESYS Security Whitepaper*

Security zones

IT resources vary in the extent to which they can be trusted. A common security architecture is therefore based on a layered approach that uses zones of trust to provide increasing levels of security according to increasing security needs. Less-trusted zones contain more-trusted zones and connections between the zones are only possible through secure interconnections such as firewallsFig. 719. All resources in the same zone must have the same minimum level of trust. The inner layers, where communication interaction needs to flow freely between nodes, must have the highest level of trust. This is the approach described in the IEC 62443 series of standards.

Firewalls, gateways, and proxies are used to control network traffic between zones of different security levels, and to filter out any undesirable or dangerous material. Traffic that is allowed to pass between zones should be limited to what is absolutely necessary because each type of service call or information exchange translates into a possible route that an intruder may be able to exploit. Different types of services represent different risks. Internet access, incoming e-mail and instant messaging, for example, represent very high risks.

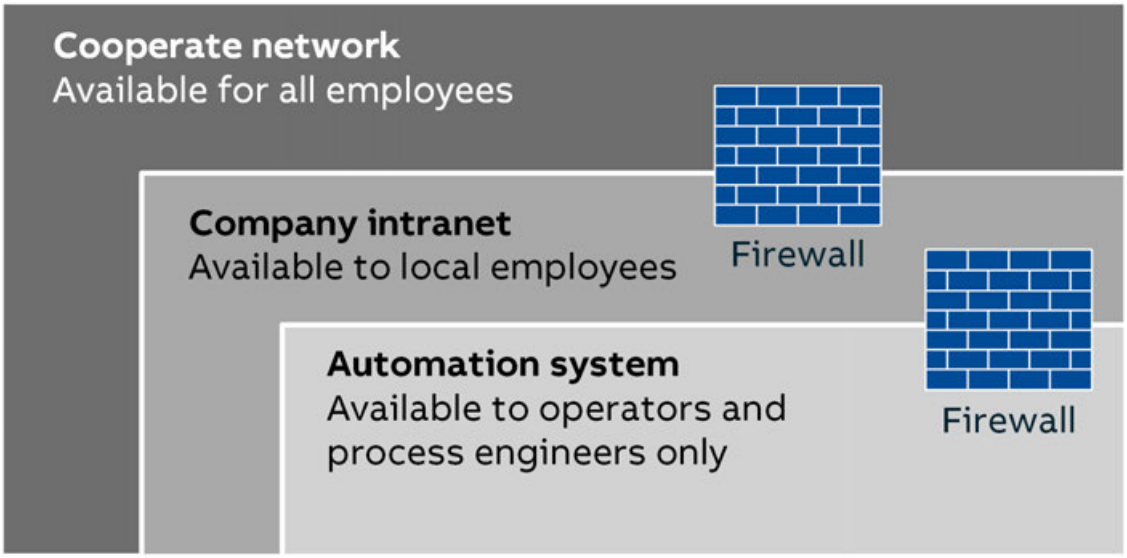


Fig. 719: Security zones

Fig. 719 shows three security zones, but the number of zones does not have to be as many or as few as three. The use of multiple zones allows access between zones of different trust levels to be controlled to protect a trusted resource from attack by a less trusted one.

High-security zones should be kept small and independent. They need to be physically protected, i.e. physical access to computers, network equipment and network cables must be limited by physical means to authorized persons only. A high-security zone should obviously not depend on resources in a less secure zone for its security. Therefore, it should form its own domain that is administered from the inside, and not depend on, e.g., a domain controller in a less secure network.

Even if a network zone is regarded as trusted, an attack is still possible: by a user or compromised resource that is inside the trusted zone, or by an outside user or resource that succeeds to penetrate the secure interconnection. Trust therefore depends also upon the types of measures taken to detect and prevent compromise of resources and violation of the security policy.

References: *Security for Industrial Automation and Control Systems*

1.6.1.2.2 Secure operation

The controller must be located in a protected environment in order to avoid accidental or intended access to the controller or the application.

A protected environment can be:

- Locked control cabinets without connection from outside
- No direct internet connection
- Use firewalls and VPN to separate different networks
- Separate different production areas with different access controls

To increase security, physical access protection measures such as fences, turnstiles, cameras or card readers can be added.

Follow these rules for the protected environment:

- Keep the trusted network as small as possible and independent from other networks.
- Protect the cross-communication of controllers and the communication between controllers and field devices via standard communication protocols (fieldbus systems) using appropriate measures.
- Protect such networks from unauthorized physical access.
- Use fieldbus systems only in protected environments. They are not protected by additional measures, such as encryption. Open physical or data access to fieldbus systems and their components is a serious security risk.

- Physically protect all equipment, i.e., ensure that physical access to computers, network equipment and cables, controllers, I/O systems, power supplies, etc., is limited to authorized persons
- When connecting a trusted network zone to outer networks, make sure that all connections are through properly configured secure interconnections only, such as a firewall or a system of firewalls, which is configured for “deny by default”, i.e., blocks everything except traffic that is explicitly needed to fulfill operational requirements.
- Allow only authorized users to log on to the system, and enforce strong passwords that are changed regularly.
- Continuously maintain the definitions of authorized users, user groups, and access rights, to properly reflect the current authorities and responsibilities of all individuals at all times. Users should not have more privileges than they need to do their job.
- Do not use the system for e-mail, instant messaging, or internet browsing. Use separate computers and networks for these functions if they are needed.
- Do not allow installation of any unauthorized software in the system.
- Restrict temporary connection of portable computers, USB memory sticks and other removable data carriers. Computers that can be physically accessed by regular users should have ports for removable data carriers disabled.
- If portable computers need to be connected, e.g., for service or maintenance purposes, they should be carefully scanned for viruses immediately before connection.
- All CDs, DVDs, USB memory sticks and other removable data carriers, and files with software or software updates, should also be checked for viruses before being introduced into the trusted zone.
- Continuously monitor the system for intrusion attempts.
- Define and maintain plans for incident response, including how to recover from potential disasters.
- Regularly review the organization as well as technical systems and installations with respect to compliance with security policies, procedures and practices.

A protected local control cabinet could look like in figure 720, page 3705. This network is not connected to any external network. Security is primarily a matter of physically protecting the automation system and preventing unauthorized users from accessing the system and from connecting or installing unauthorized hardware and software.

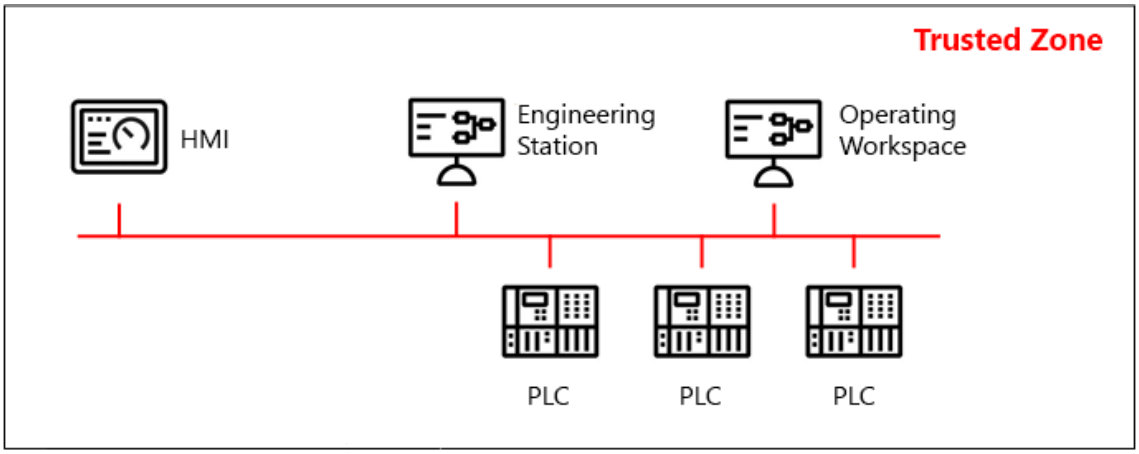


Fig. 720: Isolated automation system

Servers and workplaces that are not directly involved in the control and monitoring of the process should preferably be connected to a subnet that is separated from the automation system network by means of a router/firewall. This makes it possible to better control the network load and to limit access to certain servers on the automation system network. Note that servers and workplaces on this subnet are part of the trusted zone and thus need to be subject to the same security precautions as the nodes on the automation system network.

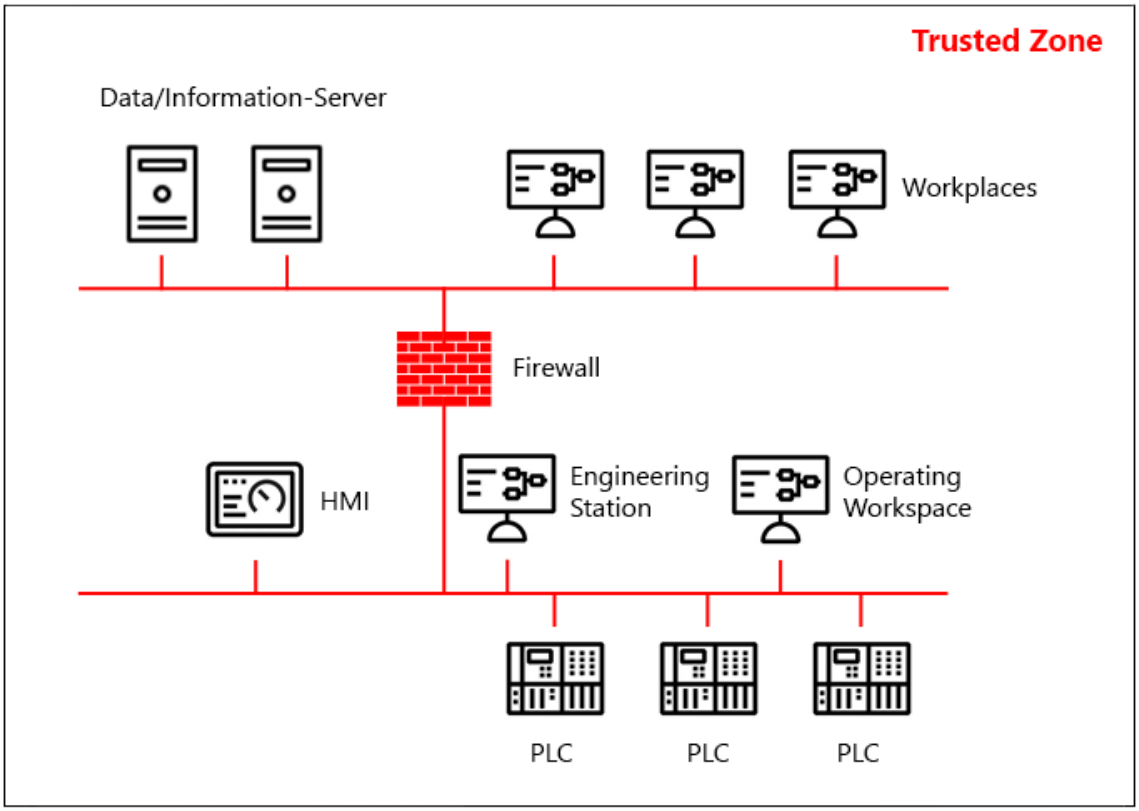


Fig. 721: Plant information network connected to an automation system

For the purposes of process control security, a general-purpose information system (IS) network should not be considered a trusted network, not the least since such networks are normally further connected to the Internet or other external networks. The IS network is therefore a different lower-security zone, and it should be separated from the automation system by means of a firewall. The IS and automation system networks should form separate domains.

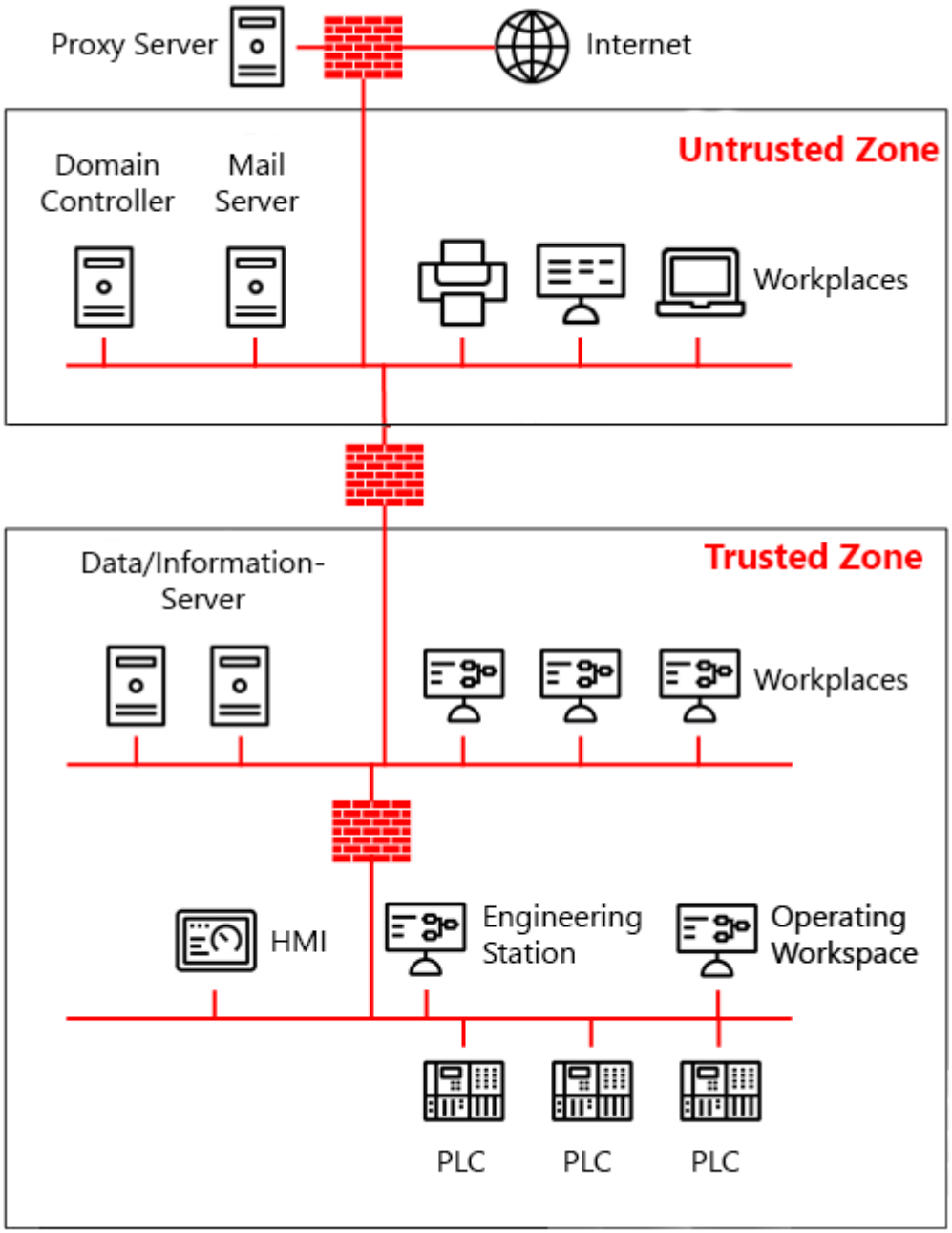


Fig. 722: Automation system and IS network

1.6.1.2.3 Hardening

System hardening means to eliminate as many security risks as possible. Hardening your system is an important step to protect your personal data and information. This process intends to eliminate attacks by patching vulnerabilities and turning off inessential services. Hardening a system involves several steps to form layers of protection.

Commissioning phase

- Protect the hardware from unauthorized access
- Be sure the hardware is based on a secure environment
- Disable unused software and services (network ports)
- Install firewalls
- Disallow file sharing among programs
- Install virus and spyware protection

- Use containers or virtual machines
- Create strong passwords by applying a strong password policy
- Create and keep backups
- Use encryption when possible
- Disable weak encryption algorithms
- Separate data and programs
- Enable and use disk quotas
- Strong logical access control
- Adjust default settings, especially passwords

Verification phase

- Verification of antivirus - Check antivirus is active and updated
- Verification of the identification - Check that test and unauthorized accounts are removed
- Verification of intrusion detection systems - Check malicious traffic is blocked
- Verification of audit logging - Check audit log is enabled
- You can use the checklist out of the *cyber security white paper*

Operation phase

- Keep software up-to-date, especially by applying security patches
- Keep antivirus up and running
- Keep antivirus definitions up-to-date
- Delete unused user accounts
- Lock an active session whenever it is unattended, e.g., lock the screen of the PC or of the control panel (HMI)

Decommissioning phase

- Delete all credentials stored in the device like certificates and user data *Chapter 1.6.3.4.6 “Decommissioning” on page 5233*.

References: *Hardening in Wikipedia (2021)*

1.6.1.2.4 Open Ports and Services

Overview of minimum cyber security requirements for open ports and services settings.

Port	Protocol	Description
1217	TCP	CODESYS Gateway V3
1210	TCP	CODESYS Gateway V2
1211	TCP	CODESYS Gateway V2
22350	TCP/UDP	CodeMeter License Server (runtime) – license
22352	HTTP	CodeMeter License Server (runtime) – WebAdmin
22353	HTTPS	CodeMeter License Server (runtime) – WebAdmin
11030	HTTP	Python editor server

1.6.1.3 License and third party information

Information on Automation Builder licensing and Third Party software can be found in the "About" window of the Automation Builder Installation Manager.

1.6.1.4 Regulations

Appropriate system setup

The following regulations have to be taken into due consideration:

- DIN VDE 0100: "Regulations for the Setting up of Power Installations"
- DIN VDE 0110 Part 1 and Part 2: "The Rating of Creepage Distances and Clearances"
- DIN VDE 0160 and DIN VDE 0660 Part 500: "The Equipment of Power Installations with Electrical Components"

To ensure project success and proper installation of all systems, customers must be familiar and proficient with the following standards and must comply with their directives:

- DIN VDE 0113 Part 1 & Part 200: "Working & Process Machinery"
- DIN VDE 0106 Part 100: "Close proximity to dangerous voltages"
- DIN VDE 0160, DIN VDE 0110 Part 1: "Protection against direct contact"

The user has to guarantee that the devices and the components are mounted following these regulations. For operating the machines and installations, other national and international relevant regulations, concerning prevention of accidents and using technical working means, also have to be met.

AC500 devices are designed according to IEC 1131 Part 2 under overvoltage category II per DIN VDE 0110 Part 2.

For direct connection of AC Category III overvoltages provide protection measures for overvoltage category II according to IEC-Report 664/1980 and DIN VDE 0110 Part 1.

Equivalent standards:

- DIN VDE 0110 Part 1 ↔ IEC 664
- DIN VDE 0113 Part 1 ↔ EN 60204 Part 1
- DIN VDE 0660 Part 500 ↔ EN 60439-1 ↔ IEC 439-1

All rights reserved to change design, size, weight, etc.

Qualified personnel

Both the control system AC500 and other components in the vicinity are operated with dangerous contact voltages. Touching parts, which are under such voltages, can cause grave damage to health.

In order to avoid such risks and the occurrence of material damage, persons involved with the assembly, starting up and servicing must possess pertinent knowledge of the following:

- Automation technology sector
- Dealing with dangerous voltages
- Using standards and regulations, in particular VDE, accident prevention regulations and regulations concerning special ambient conditions (e.g. areas potentially endangered by explosive materials, heavy pollution or corrosive influences).

1.6.1.5 Definitions: PLC system start-up

Cold start

The AC500-eCo V3 does not use a battery for buffering the operand areas specified below, hence the “cold start” mode does not exist in this product.

- A cold start is performed by switching power OFF/ON if no battery is connected.
- All RAM memory modules are checked and erased.
- If no user program is stored in the Flash EPROM, the default values (as set on delivery) are applied to the interfaces.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

Warm start

- A warm start is performed by switching power OFF/ON with a battery connected.
- All RAM memory modules are checked and erased except of the buffered operand areas and the RETAIN variables .
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

RUN -> STOP

- RUN -> STOP means pressing the RUN function key on the PLC while the PLC is in run mode (AC500 PLC display "run", AC500-eCo PLC "RUN LED" is ON).
- If a user program is loaded into RAM, execution is stopped.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP", AC500-eCo "RUN LED" changes from ON to OFF.

START -> STOP

- START -> STOP means stopping the execution of the user program in the PLC's RAM using the menu item "Online/Stop" in the programming system.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP".

Reset

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (exception: RETAIN variables) are set to their initialization values.
- Reset is performed using the menu item "Online/Reset" in the programming system or pressing the function key RUN for $\geq$ 5 s in STOP mode.

Reset (cold)

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (also RETAIN variables) are set to their initialization values.
- Reset (cold) is performed using the menu item "Online/Reset (cold)" in the programming system.

Reset (original)

- Resets the controller to its original state (deletion of Flash, SRAM (%M, area, %R area, RETAIN, RETAIN PERSISTENT), Communication Module configurations and user program!).
- Reset (original) is performed using the menu item "Online/Reset (original)" in the programming system.

STOP -> RUN

- STOP -> RUN means short pressing the RUN function key on the PLC while the PLC is in STOP mode (AC500 PLC display "StoP", AC500-eCo "RUN LED" is ON). "RUN LED" is OFF of the toggle switch of an AC500-eCo CPU.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo "RUN LED" changes from OFF to ON.

STOP -> START

- STOP -> START means continuing the execution of the user program in the PLC's RAM using the menu item "Online/Start" in the programming system.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo PLC "RUN LED" changes from OFF to ON.

Download

- Download means loading the complete user program into the PLC's RAM. This process is started by selecting the menu item "Online/Download" in the programming system or after confirming a corresponding system message when switching to online mode (menu item "Online/Login").
- Execution of the user program is stopped.
- In order to store the user program to the Flash memory, the menu item "Online/Create boot project" must be called after downloading the program.
- Variables are set to their initialization values according to the initialization table.
- RETAIN variables can have wrong values as they can be allocated to other memory addresses in the new project!
- A download is forced by the following:
  - changed PLC configuration
  - changed task configuration
  - changed library management
  - changed compile-specific settings (segment sizes)
  - execution of the commands "Project/Clean all" and "Project/Rebuild All".

Online change

- After a project has changed, only these changes are compiled when pressing the key <F11> or calling the menu item "Project/Build". The changed program parts are marked with a blue arrow in the block list.
- The term Online Change means loading the changes made in the user program into the PLC's RAM using the programming system (after confirming a corresponding system message when switching to online mode, menu item "Online/Login").
- Execution of the user program is not stopped. After downloading the program changes, the program is re-organized. During re-organization, no further online change command is allowed. The storage of the user program to the Flash memory using the command "Online/Create boot project" cannot be initiated until re-organization is completed.
- Online Change is not possible after:
  - changes in the PLC configuration
  - changes in the task configuration
  - changes in the library management
  - changed compile-specific settings (segment sizes)
  - performing the commands "Project/Clean all" and "Project/Rebuild All".

Data buffering

- Data buffering, i.e., maintaining data after power ON/OFF, is only possible, if a battery is connected for AC500 CPU and the buffering will take place in FLASH with AC500-eCo V3 CPU. The following data can be buffered completely or in parts:
  - Data in the addressable flag area (%M area)
  - RETAIN variable
  - PERSISTENT variable (number is limited, no structured variables)
  - PERSISTENT area (%R area)
- In order to buffer particular data, the data must be excluded from the initialization process (see *Chapter 1.6.4.1.1.8 “Data backup and initialization” on page 5410*).

1.6.1.6 Definitions: RCOM

Terms

Do not confuse the RCOM services cold start / warm start and the corresponding PLC commands referring to the hardware state. In this section, the terms "cold start" and "warm start" always refer to the RCOM system services and therefore only affect the protocol state.

Cold start

A cold start service has to be performed after the initialization of the RCOM master. The cold start can be transmitted either by broadcast to all slaves simultaneously or to each slave individually.

A cold start requires reinitialization of the entire protocol mechanism and clearing of the event queue contents. For this, a special cold start event is triggered in the addressed RCOM slaves. This event is required when operating ABB MasterPiece systems. In case of pure Advant Controller networks, the event is only indicated when polling.

After a cold start, always normalization has to be performed. Otherwise it is not possible to transmit data sets.

Warm start

By executing a warm start service, it is possible to clear the event queue of a slave (or all slaves). A warm start can be used to resume communication after transmission errors. This permits master and slave to resynchronize.

After a warm start, always normalization has to be performed. Otherwise it is not possible to transmit data sets.

Normalization

A slave has to be normalized after a cold start or a warm start. Normalization enables the transmission of data sets and events. If a slave is not normalized, it cannot trigger events. The RCOM_TRANSMIT connection element then displays a corresponding error message.

If a master polls a non-normalized slave, the RCOM_POLL connection element signals a corresponding error.

1.6.1.7 Device lists

1.6.1.7.1 Device list: Terminal bases

Terminal bases for AC500 (Standard):

V2 products

Type	Description
TB511-ARCNET ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB511-ARCNET, terminal base AC500, slots: 1 processor module, 1 communication module, ARCNET COAX connector
TB511-ETH ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB511-ETH, terminal base AC500, slots: 1 processor module, 1 communication module, Ethernet RJ45 connector
TB511-ETH-XC ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB511-ETH-XC, terminal base AC500, slots: 1 processor module, 1 communication module, Ethernet RJ45 connector, XC version
TB521-ARCNET ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB521-ARCNET, terminal base AC500, slots: 1 processor module, 2 communication modules, ARCNET COAX connector
TB521-ETH ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB521-ETH, terminal base AC500, slots: 1 processor module, 2 communication modules, Ethernet RJ45 connector
TB521-ETH-XC ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB521-ETH-XC, terminal base AC500, slots: 1 processor module, 2 communication modules, Ethernet RJ45 connector, XC version

Type	Description
TB523-2ETH ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB523-2ETH, terminal base AC500, slots: 1 processor module, 2 communication modules, 2 Ethernet RJ45 connector
TB541-ETH ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB541-ETH, terminal base AC500, slots: 1 processor module, 4 communication modules, Ethernet RJ45 connector
TB541-ETH-XC ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TB541-ETH-XC, terminal base AC500, slots: 1 processor module, 4 communication modules, Ethernet RJ45 connector, XC version
TF501-CMS ↳ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*	TF501-CMS, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 1 communication module, Ethernet RJ45 connector
TF501-CMS-XC ↳ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*	TF501-CMS-XC, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 1 communication module, Ethernet RJ45 connector, XC version
TF521-CMS ↳ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*	TF521-CMS, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 2 communication modules, Ethernet RJ45 connector
TF521-CMS-XC ↳ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*	TF521-CMS-XC, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 2 communication modules, Ethernet RJ45 connector, XC version

1.6.1.7.2 Device list: Processor modules (CPUs)

Processor modules for AC500-eCo

V2 products

Type	Description
PM554-TP ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM554-TP, processor module, 128 kB memory, 8 DI, 6 DO-T, 24 V DC, with pluggable I/O terminal blocks
PM554-TP-ETH ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM554-TP-ETH, processor module, 128 kB memory, 8 DI, 6 DO-T, 24 V DC, onboard Ethernet, with pluggable I/O terminal blocks
PM554-RP ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM554-RP, processor module, 128 kB memory, 8 DI, 6 DO-R, 24 V DC, with pluggable I/O terminal blocks

Type	Description
PM554-RP-AC ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM554-RP-AC, processor module, 128 kB memory, 8 DI, 6 DO-R, 100 V AC...240 V AC, with pluggable I/O terminal blocks
PM556-TP-ETH ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM556-TP-ETH, processor module, 512 kB memory, 8 DI, 6 DO-T, 24 V DC, onboard Ethernet, with pluggable I/O terminal blocks
PM564-TP ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-TP, processor module, 128 kB memory, 6 DI, 6 DO-T, 2 AI, 1 AO, 24 V DC
PM564-TP-ETH ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-TP-ETH, processor module, 128 kB memory, 6 DI, 6 DO-T, 2 AI, 1 AO, 24 V DC, Ethernet interface
PM564-RP ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-RP, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI, 1 AO, 24 V DC
PM564-RP-AC ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-RP-AC, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI, 1 AO, 100 V AC...240 V AC
PM564-RP-ETH ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-RP-ETH, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI, 1 AO, 24 V DC, Ethernet interface
PM564-RP-ETH-AC ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM564-RP-ETH-AC, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI, 1 AO, 100 V AC...240 V AC, Ethernet interface
PM566-TP-ETH ↳ *Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804*	PM566-TP-ETH, processor module, 512 kB memory, 6 DI, 6 DO-T, 2 AI, 1 AO, 24 V DC, Ethernet interface

Processor modules for AC500 (Standard)

V2 products

Type	Description
PM572 *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM572, processor module, memory 128 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display
PM573-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM573-ETH, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM573-ETH-XC *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM573-ETH-XC, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version
PM582 *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM582, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display
PM582-XC *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM582-XC, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, XC version
PM583-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM583-ETH, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM583-ETH-XC *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM583-ETH-XC, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBPP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version
PM585-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM585-ETH, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM590-ARCNET *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM590-ARCNET, processor module, memory 2 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, integrated communication module ARCNET
PM590-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM590-ETH, processor module, memory 2 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM591-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM591-ETH, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBPP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM591-2ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*	PM591-2ETH, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 1 RS-232/485 (programming, Modbus/ CS31), display, 2 onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols

Type	Description
PM591-ETH-XC ↳ *Chapter 1.6.2.3.2.1 "PM57x (-y), PM58x (-y) and PM59x (-y)" on page 3848*	PM591-ETH-XC, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/ CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC Version
PM592-ETH ↳ *Chapter 1.6.2.3.2.1 "PM57x (-y), PM58x (-y) and PM59x (-y)" on page 3848*	PM592-ETH, processor module, memory 4 MB / 4 GB flash disk, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols
PM592-ETH-XC ↳ *Chapter 1.6.2.3.2.1 "PM57x (-y), PM58x (-y) and PM59x (-y)" on page 3848*	PM592-ETH-XC, processor module, memory 4 MB / 4 GB flash disk, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version
PM595-4ETH-F ↳ *Chapter 1.6.2.3.2.2 "PM595-4ETH" on page 3863*	PM595-4ETH-F, processor module, user progr./data memory 16 MB / 16 MB, 1.3 GHz, 24 V DC, memory card slot, interfaces 2 RS232-485, 2 independent Ethernet interfaces (Progr., web server, IEC60870-5-104 protocol), 2 independent Ethernet based interfaces with 2-port switch (between fieldbus protocols PROFINET IO, EtherCAT and Ethernet)
PM595-4ETH-M-XC ↳ *Chapter 1.6.2.3.2.2 "PM595-4ETH" on page 3863*	PM595-4ETH-M-XC, processor module, user progr./data memory 16 MB / 16 MB, 1.0 GHz, 24 V DC, memory card slot, interfaces 2 RS232-485, 2 independent Ethernet interfaces (Progr., web server, IEC60870-5-104 protocol), 2 independent Ethernet based interfaces with 2-port switch (between fieldbus protocols PROFINET IO, EtherCAT and Ethernet), XC version

1.6.1.7.3 Device list: Communication modules

Type	Description
CM574-RCOM ↳ *Chapter 1.6.2.4.3.1 "CM574-RCOM for RCOM/RCOM+ " on page 4044*	CM574-RCOM, communication module, 2 serial RS-232/485, RCOM/RCOM+ protocol
CM574-RS ↳ *Chapter 1.6.2.4.4.1 "CM574-RS with 2 serial interfaces" on page 4049*	CM574-RS, communication module, 2 serial RS232/485, free configurable serial interface module
CM579-ETHCAT ↳ *Chapter 1.6.2.4.6.1 "CM579-ETHCAT - EtherCAT master" on page 4066*	CM579-ETHCAT, EtherCAT communication module
CM579-PNIO ↳ *Chapter 1.6.2.4.9.1 "CM579-PNIO - PROFINET IO RT controller" on page 4084*	CM579-PNIO, PROFINET communication module
CM579-PNIO-XC ↳ *Chapter 1.6.2.4.9.1 "CM579-PNIO - PROFINET IO RT controller" on page 4084*	CM579-PNIO-XC, PROFINET communication module, XC version

Type	Description
CM582-DP *Chapter 1.6.2.4.8.1 “CM582-DP - PROFIBUS DP slave” on page 4075*	CM582-DP, communication module PROFIBUS DP slave 12 Mbit/s
CM582-DP-XC *Chapter 1.6.2.4.8.1 “CM582-DP - PROFIBUS DP slave” on page 4075*	CM582-DP-XC, communication module PROFIBUS DP slave 12 Mbit/s, XC version
CM588-CN *Chapter 1.6.2.4.5.1 “CM588-CN - CANopen slave” on page 4053*	CM588-CN, communication module, CANopen slave
CM588-CN-XC *Chapter 1.6.2.4.5.1 “CM588-CN - CANopen slave” on page 4053*	CM588-CN-XC, communication module, CANopen slave, XC version
CM589-PNIO *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089*	CM589-PNIO, PROFINET communication module
CM589-PNIO-XC *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089*	CM589-PNIO-XC, PROFINET communication module, XC version
CM589-PNIO-4 *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089*	CM589-PNIO-4, PROFINET communication module
CM589-PNIO-4-XC *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089*	CM589-PNIO-4-XC, PROFINET communication module, XC version
CM592-DP *Chapter 1.6.2.4.8.2 “CM592-DP - PROFIBUS DP master” on page 4079*	CM592-DP, communication module PROFIBUS DP master 12 Mbit/s
CM592-DP-XC *Chapter 1.6.2.4.8.2 “CM592-DP - PROFIBUS DP master” on page 4079*	CM592-DP-XC, communication module PROFIBUS DP master 12 Mbit/s, XC version
CM597-ETH *Chapter 1.6.2.4.7.1 “CM597-ETH - Communication module Ethernet” on page 4070*	CM597-ETH, communication module Ethernet TCP/IP with integrated 2-port switch

Type	Description
CM597-ETH-XC ↳ *Chapter 1.6.2.4.7.1 “CM597-ETH - Communication module Ethernet” on page 4070*	CM597-ETH-XC, communication module Ethernet TCP/IP with integrated 2-port switch, XC version
CM598-CN ↳ *Chapter 1.6.2.4.5.2 “CM598-CN - CANopen master” on page 4060*	CM598-CN, communication module, CANopen master
CM598-CN-XC ↳ *Chapter 1.6.2.4.5.2 “CM598-CN - CANopen master” on page 4060*	CM598-CN-XC, communication module, CANopen master, XC version

1.6.1.7.4 Device list: Terminal units

Type	Description
TU507-ETH ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*	TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals
TU508-ETH ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*	TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals
TU508-ETH-XC ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*	TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version
TU509 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099*	TU509, communication interface module terminal unit, 24 V DC, screw terminals
TU510 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099*	TU510, communication interface module terminal unit, 24 V DC, spring terminals
TU510-XC ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099*	TU510-XC, communication interface module terminal unit, 24 V DC, spring terminals, XC version

Type	Description
TU515 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU515, I/O terminal unit, 24 V DC, screw terminals
TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU516, I/O terminal unit, 24 V DC, spring terminals
TU516-XC *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
TU516-H *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
TU516-H-XC *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version
TU517 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*	TU517, terminal unit for communication interface modules, 24 V DC, screw terminals
TU518 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*	TU518, terminal unit for communication interface modules, 24 V DC, spring terminals
TU518-XC *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*	TU518-XC, terminal unit for communication interface modules, 24 V DC, spring terminals, XC version
TU520-ETH *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*	TU520-ETH, PROFINET I/O terminal unit, 24 V DC, spring terminals
TU520-ETH-XC *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*	TU520-ETH-XC, PROFINET I/O terminal unit, 24 V DC, spring terminals, XC version
TU531 *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*	TU531, I/O terminal unit, 230 V AC, relays, screw terminals

Type	Description
TU532 *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*	TU532, I/O terminal unit, 230 V AC, relays, spring terminals
TU532-XC *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*	TU532-XC, I/O terminal unit, 230 V AC, relays, spring terminals, XC version
TU532-H *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*	TU532-H, I/O terminal unit, hot swap, 230 V AC, relays, spring terminals
TU532-H-XC *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*	TU532-H-XC, I/O terminal unit, hot swap, 230 V AC, relays, spring terminals, XC version
TU541 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU541, I/O terminal unit, 24 V DC, screw terminals
TU542 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU542, I/O terminal unit, 24 V DC, spring terminals
TU542-XC *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
TU542-H *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
TU542-H-XC *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*	TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version
TU551-CS31 *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*	TU551-CS31, CS31 bus terminal unit, 24 V DC, screw terminals

Type	Description
TU552-CS31 ↳ *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*	TU552-CS31, CS31 bus terminal unit, 24 V DC, spring terminals
TU552-CS31-XC ↳ *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*	TU552-CS31-XC, CS31 bus terminal unit, 24 V DC, spring terminals, XC version

1.6.1.7.5 Device list: S500-eCo I/O modules

Type	Description
AI561 ↳ *Chapter 1.6.2.6.2.1.1 “AI561 - Analog input module” on page 4351*	AI561, analog input module, 4 AI, U/I
AI562 ↳ *Chapter 1.6.2.6.2.1.2 “AI562 - Analog input module” on page 4362*	AI562, analog input module, 2 AI, RTD
AI563 ↳ *Chapter 1.6.2.6.2.1.3 “AI563 - Analog input module” on page 4373*	AI563, analog input module, 4 AI, thermocouple
AO561 ↳ *Chapter 1.6.2.6.2.1.4 “AO561 - Analog output module” on page 4385*	AO561, analog output module, 2 AO, U/I
AX561 ↳ *Chapter 1.6.2.6.2.1.5 “AX561 - Analog input/output module” on page 4394*	AX561, analog input/output module, 4 AI, 2AO, U/I
DC561 ↳ *Chapter 1.6.2.6.1.1.1 “DC561 - Digital input/output module” on page 4125*	DC561, digital input/output module, 16 configurable inputs/outputs, transistor output, interfast connection
DC562 ↳ *Chapter 1.6.2.6.1.1.2 “DC562 - Digital input/output module” on page 4133*	DC562, digital input/output module, 16 configurable inputs/outputs
DI561 ↳ *Chapter 1.6.2.6.1.1.3 “DI561 - Digital input module” on page 4144*	DI561, digital input module, 8 DI, 24 V DC
DI562 ↳ *Chapter 1.6.2.6.1.1.4 “DI562 - Digital input module” on page 4151*	DI562, digital input module, 16 DI, 24 V DC

Type	Description
DI571 ↳ *Chapter 1.6.2.6.1.1.5 “DI571 - Digital input module” on page 4159*	DI571, digital input module, 8 DI, 120 V AC...240 V AC
DI572 ↳ *Chapter 1.6.2.6.1.1.6 “DI572 - Digital input module” on page 4168*	DI572, digital input module, 16 DI, 100 V AC...240 V AC
DO561 ↳ *Chapter 1.6.2.6.1.1.7 “DO561 - Digital output module” on page 4177*	DO561, digital output module, 8 DO, transistor output
DO562 ↳ *Chapter 1.6.2.6.1.1.8 “DO562 - Digital output module” on page 4186*	DO562, digital output module, 16 DO, transistor output
DO571 ↳ *Chapter 1.6.2.6.1.1.9 “DO571 - Digital output module” on page 4195*	DO571, digital output module, 8 DO, relay output
DO572 ↳ *Chapter 1.6.2.6.1.1.10 “DO572 - Digital output module” on page 4205*	DO572, digital output module, 8 DO, triac output
DO573 ↳ *Chapter 1.6.2.6.1.1.11 “DO573 - Digital output module” on page 4215*	DO573, digital output module, 16 DO, relay output
DX561 ↳ *Chapter 1.6.2.6.1.1.12 “DX561 - Digital input/output module” on page 4227*	DX561, digital input/output module, 8 DI 24 V DC, 8 DO 24 V DC, transistor output
DX571 ↳ *Chapter 1.6.2.6.1.1.13 “DX571 - Digital input/output module” on page 4239*	DX571, digital input/output module, 8 DI 24 V DC, 8 DO, relay output

1.6.1.7.6 Device list: S500 I/O modules

Type	Description
AI523 ↳ *Chapter 1.6.2.6.2.2.2 “AI523 - Analog input module” on page 4433*	AI523, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires
AI523-XC ↳ *Chapter 1.6.2.6.2.2.2 “AI523 - Analog input module” on page 4433*	AI523-XC, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires, XC version
AI531 ↳ *Chapter 1.6.2.6.2.2.3 “AI531 - Analog input module” on page 4455*	AI531, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires

Type	Description
AI531-XC ↯ *Chapter 1.6.2.6.2.2.3 “AI531 - Analog input module” on page 4455*	AI531-XC, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires, XC version
AO523 ↯ *Chapter 1.6.2.6.2.2.4 “AO523 - Analog output module” on page 4487*	AO523, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires
AO523-XC ↯ *Chapter 1.6.2.6.2.2.4 “AO523 - Analog output module” on page 4487*	AO523-XC, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires, XC version
AX521 ↯ *Chapter 1.6.2.6.2.2.5 “AX521 - Analog input/output module” on page 4502*	AX521, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires
AX521-XC ↯ *Chapter 1.6.2.6.2.2.5 “AX521 - Analog input/output module” on page 4502*	AX521-XC, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version
AX522 ↯ *Chapter 1.6.2.6.2.2.6 “AX522 - Analog input/output module” on page 4525*	AX522, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires
AX522-XC ↯ *Chapter 1.6.2.6.2.2.6 “AX522 - Analog input/output module” on page 4525*	AX522-XC, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version
DA501 ↯ *Chapter 1.6.2.6.3.1.1 “DA501 - Digital/Analog input/output module” on page 4550*	DA501, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO
DA501-XC ↯ *Chapter 1.6.2.6.3.1.1 “DA501 - Digital/Analog input/output module” on page 4550*	DA501-XC, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO, XC version
DA502 ↯ *Chapter 1.6.2.6.3.1.2 “DA502 - Digital/Analog input/output module” on page 4585*	DA502, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO
DA502-XC ↯ *Chapter 1.6.2.6.3.1.2 “DA502 - Digital/Analog input/output module” on page 4585*	DA502-XC, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO, XC version
DC522 ↯ *Chapter 1.6.2.6.1.2.1 “DC522 - Digital input/output module” on page 4253*	DC522, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires
DC522-XC ↯ *Chapter 1.6.2.6.1.2.1 “DC522 - Digital input/output module” on page 4253*	DC522-XC, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires, XC version

Type	Description
DC523 *Chapter 1.6.2.6.1.2.2 “DC523 - Digital input/output module” on page 4264*	DC523, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire
DC523-XC *Chapter 1.6.2.6.1.2.2 “DC523 - Digital input/output module” on page 4264*	DC523-XC, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire, XC version
DC532 *Chapter 1.6.2.6.1.2.3 “DC532 - Digital input/output module” on page 4276*	DC532, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire
DC532-XC *Chapter 1.6.2.6.1.2.3 “DC532 - Digital input/output module” on page 4276*	DC532-XC, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire, XC version
DI524 *Chapter 1.6.2.6.1.2.5 “DI524 - Digital input module” on page 4298*	DI524, digital input module, 32 DI, 24 V DC, 1-wire
DI524-XC *Chapter 1.6.2.6.1.2.5 “DI524 - Digital input module” on page 4298*	DI524-XC, digital input module, 32 DI, 24 V DC, 1-wire, XC version
DO524 *Chapter 1.6.2.6.1.2.6 “DO524 - Digital output module” on page 4307*	DO524, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire
DO524-XC *Chapter 1.6.2.6.1.2.6 “DO524 - Digital output module” on page 4307*	DO524-XC, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire, XC version
DO526 *Chapter 1.6.2.6.1.2.7 “DO526 - Digital output module” on page 4317*	DO526, digital output module, 8 DO, 24 V DC / 2 A, 1-wire
DO526-XC *Chapter 1.6.2.6.1.2.7 “DO526 - Digital output module” on page 4317*	DO526-XC, digital output module, 8 DO, 24 V DC / 2 A, 1-wire, XC version
DX522 *Chapter 1.6.2.6.1.2.8 “DX522 - Digital input/output module” on page 4327*	DX522, digital input/output module, 8 DI, 24 V DC, 8 DO relays
DX522-XC *Chapter 1.6.2.6.1.2.8 “DX522 - Digital input/output module” on page 4327*	DX522-XC, digital input/output module, 8 DI, 24 V DC, 8 DO relays, XC version
DX531 *Chapter 1.6.2.6.1.2.9 “DX531 - Digital input/output module” on page 4339*	DX531, digital input/output module, 8 DI, 230 V AC, 4 DO relay, 2-wires

1.6.1.7.7 Device list: Function modules

Type	Description
CD522 ↳ *Chapter 1.6.2.7.2.1 “CD522 - Encoder, counter and PWM module” on page 4635*	CD522, encoder & PWM module, 2 encoder inputs, 2 PWM outputs, 2 digital inputs 24 V DC, 8 digital outputs 24 V DC
CD522-XC ↳ *Chapter 1.6.2.7.2.1 “CD522 - Encoder, counter and PWM module” on page 4635*	CD522-XC, encoder & PWM module, 2 encoder inputs, 2 PWM outputs, 2 digital inputs 24 V DC, 8 digital outputs 24 V DC, XC version
DC541-CM ↳ *Chapter 1.6.2.6.1.2.4 “DC541-CM - Digital input/output module” on page 4290*	DC541-CM, digital input/output module, 8 DC, 24 V DC / 0.5 A, 1-wire
DC541-CM-XC ↳ *Chapter 1.6.2.6.1.2.4 “DC541-CM - Digital input/output module” on page 4290*	DC541-CM-XC, digital input/output module, 8 DC, 24 V DC / 0.5 A, 1-wire, XC version
FM502-CMS ↳ *Chapter 1.6.2.7.2.2 “FM502-CMS - Analog measurements” on page 4658*	FM502-CMS for condition monitoring systems, 16 AI, 2 DI, 2 DC, 1 encoder (A, B, Z)
FM502-CMS-XC ↳ *Chapter 1.6.2.7.2.2 “FM502-CMS - Analog measurements” on page 4658*	FM502-CMS-XC for condition monitoring systems, 16 AI, 2 DI, 2 DC, 1 encoder (A, B, Z), XC version
AC500-eCo only:	
FM562 ↳ *Chapter 1.6.2.7.1.1 “FM562 for pulse train output” on page 4617*	FM562, pulse-train output module, 2 axes, RS-422, 4 DI 24 V DC

1.6.1.7.8 Device list: Communication interface modules

Table 254: CANopen

Type	Description
CI581-CN ↳ *Chapter 1.6.2.8.2.2 “CI581-CN” on page 4685*	CI581-CN, CANopen communication interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI581-CN-XC ↳ *Chapter 1.6.2.8.2.2 “CI581-CN” on page 4685*	CI581-CN-XC, CANopen communication interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version

Type	Description
CI582-CN ↳ *Chapter 1.6.2.8.2.3 “CI582-CN” on page 4723*	CI582-CN, CANopen communication interface module, 8 DI, 8 DO and 8 DC
CI582-CN-XC ↳ *Chapter 1.6.2.8.2.3 “CI582-CN” on page 4723*	CI582-CN-XC, CANopen communication interface module, 8 DI, 8 DO and 8 DC, XC version

Table 255: ***CS31***

Type	Description
CI590-CS31-HA ↳ *Chapter 1.6.2.8.3.1 “CI590-CS31-HA” on page 4745*	CI590-CS31-HA, CS31 redundant communucation interface module, 16 DC
CI590-CS31-HA-XC ↳ *Chapter 1.6.2.8.3.1 “CI590-CS31-HA” on page 4745*	CI590-CS31-HA-XC, CS31 redundant communucation interface module, 16 DC, XC version
CI592-CS3 ↳ *Chapter 1.6.2.8.3.2 “CI592-CS31 - Digital and analog inputs and outputs” on page 4764*	CI592-CS31, CS31 communucation interface module, 8 DI, 8 DC, 4 AI, 2 AO
CI592-CS31-XC ↳ *Chapter 1.6.2.8.3.2 “CI592-CS31 - Digital and analog inputs and outputs” on page 4764*	CI592-CS31-XC, CS31 communucation interface module, 8 DI, 8 DC, 4 AI, 2 AO, XC version
DC551-CS31 ↳ *Chapter 1.6.2.8.3.3 “DC551-CS31 - Digital inputs and outputs” on page 4797*	DC551-CS31, CS31 communucation interface module, 8 DI, 16 DC
DC551-CS31-XC ↳ *Chapter 1.6.2.8.3.3 “DC551-CS31 - Digital inputs and outputs” on page 4797*	DC551-CS31-XC, CS31 communucation interface module, 8 DI, 16 DC, XC version

Table 256: EtherCAT

Type	Description
CI511-ETHCAT ↳ *Chapter 1.6.2.8.4.1 “CI511-ETHCAT” on page 4814*	CI511-ETHCAT, EtherCAT communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI512-ETHCAT ↳ *Chapter 1.6.2.8.4.2 “CI512-ETHCAT” on page 4846*	CI512-ETHCAT, EtherCAT communucation interface module, 8 DI, 8 DO and 8 DC

Table 257: Modbus

Type	Description
CI521-MODTCP ↳ *Chapter 1.6.2.8.5.1 “CI521-MODTCP” on page 4864*	CI521-MODTCP, Modbus TCP communucation interface module, 4 AI, 2 AO, 8 DI and 8 DO
CI521-MODTCP-XC ↳ *Chapter 1.6.2.8.5.1 “CI521-MODTCP” on page 4864*	CI521-MODTCP-XC, Modbus TCP communucation interface module, 4 AI, 2 AO, 8 DI and 8 DO, XC version
CI522-MODTCP ↳ *Chapter 1.6.2.8.5.2 “CI522-MODTCP” on page 4904*	CI522-MODTCP, Modbus TCP communucation interface module, 8 DC, 8 DI and 8 DO
CI522-MODTCP-XC ↳ *Chapter 1.6.2.8.5.2 “CI522-MODTCP” on page 4904*	CI522-MODTCP-XC, Modbus TCP communucation interface module, 8 DC, 8 DI and 8 DO, XC version

Table 258: PROFIBUS

Type	Description
CI541-DP ↳ *Chapter 1.6.2.8.6.1 “CI541-DP” on page 4930*	CI541-DP, PROFIBUS DP communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI541-DP-XC ↳ *Chapter 1.6.2.8.6.1 “CI541-DP” on page 4930*	CI541-DP-XC, PROFIBUS DP communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC versioncommunucation interface
CI542-DP ↳ *Chapter 1.6.2.8.6.2 “CI542-DP” on page 4969*	CI542-DP, PROFIBUS DP communucation interface module, 8 DI, 8 DO and 8 DC
CI542-DP-XC ↳ *Chapter 1.6.2.8.6.2 “CI542-DP” on page 4969*	CI542-DP-XC, PROFIBUS DP communucation interface module, 8 DI, 8 DO and 8 DC, XC version

Table 259: PROFINET

Type	Description
CI501-PNIO (V3) ↳ *Chapter 1.6.2.8.7.2 “CI501-PNIO” on page 4995*	CI501-PNIO (V3), PROFINET communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO
CI501-PNIO-XC (V3) ↳ *Chapter 1.6.2.8.7.2 “CI501-PNIO” on page 4995*	CI501-PNIO-XC (V3), PROFINET communucation interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version
CI502-PNIO ↳ *Chapter 1.6.2.8.7.3 “CI502-PNIO” on page 5035*	CI502-PNIO, PROFINET communucation interface module, 8 DI, 8 DO and 8 DC
CI502-PNIO-XC ↳ *Chapter 1.6.2.8.7.3 “CI502-PNIO” on page 5035*	CI502-PNIO-XC, PROFINET communucation interface module, 8 DI, 8 DO and 8 DC, XC version

Type	Description
CI504-PNIO *Chapter 1.6.2.8.7.4 “CI504-PNIO” on page 5060*	CI504-PNIO, PROFINET communucation interface module with 3 serial interfaces
CI504-PNIO-XC *Chapter 1.6.2.8.7.4 “CI504-PNIO” on page 5060*	CI504-PNIO-XC, PROFINET communucation interface module with 3 serial interfaces, XC version
CI506-PNIO *Chapter 1.6.2.8.7.5 “CI506-PNIO” on page 5076*	CI506-PNIO, PROFINET communucation interface module with 2 serial interfaces and 1 CANopen master interface
CI506-PNIO-XC *Chapter 1.6.2.8.7.5 “CI506-PNIO” on page 5076*	CI506-PNIO-XC, PROFINET communucation interface module with 2 serial interfaces and 1 CANopen master interface, XC version

1.6.1.7.9 Device list: Accessories

Type	Description
Automation Builder	DM-TOOL, Automation Builder software suite, programming software (multilanguage) *www.abb.com/automationbuilder*
MC502 *Chapter 1.6.2.9.1.2 “MC502 - Memory card” on page 5096*	MC502, memory card
MC5102 *Chapter 1.6.2.9.1.4 “MC5102 - Micro memory card with micro memory card adapter” on page 5103*	MC5102, micro memory card with micro memory card adapter
MC5141 *Chapter 1.6.2.9.1.5 “MC5141 - Memory card” on page 5108*	MC5141, memory card
MC503 *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*	MC503, memory card expansion module for PM554-x and PM564-x
TA521 *Chapter 1.6.2.9.2.4 “TA521 - Battery” on page 5175*	TA521, lithium battery
TA523 *Chapter 1.6.2.9.4.3 “TA523 - Pluggable label mounting” on page 5209*	TA523, pluggable label mounting (10 pcs)
TA524 *Chapter 1.6.2.9.2.5 “TA524 - Dummy communication module” on page 5179*	TA524, dummy communication module

Type	Description
TA525 ↳ *Chapter 1.6.2.9.4.4 “TA525 - Plastic labels” on page 5210*	TA525, set of 10 white plastic markers
TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*	TA526, wall mounting accessory, 10 pcs
TA527 ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TA527, power plug, spare part
TA528 ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*	TA528, COM1 plug, spare part
TA536 ↳ *Chapter 1.6.2.4.5.1 “CM588-CN - CANopen slave” on page 4053*	TA536, spring plug, spare part
TA535 ↳ *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*	TA535, protective caps for XC devices
↳ *Chapter 1.6.2.6.1.2.4 “DC541-CM - Digital input/output module” on page 4290*	TA536, spring plug, spare part
TA540 ↳ *Chapter 1.6.2.3.2.2 “PM595-4ETH” on page 3863*	TA540, protective plastic cap for PM595, spare part
TA541 ↳ *Chapter 1.6.2.9.2.7 “TA541 - Battery” on page 5180*	TA541, lithium battery for PM595
TA543 ↳ *Chapter 1.6.2.9.2.8 “TA543 - Screw mounting accessory” on page 5184*	TA543, screw mounting accessory for processor module PM595 without DIN rail
TA561-RTC ↳ *Chapter 1.6.2.9.1.6 “TA561-RTC - Real-time clock adapter” on page 5113*	TA561-RTC, real-time clock adapter for PM55x-xP and PM56x-xP, lithium battery TA522 included
TA562-RS ↳ *Chapter 1.6.2.9.1.7 “TA562-RS - Serial RS-485 adapter ” on page 5120*	TA562-RS, serial RS-485 adapter for PM55x-xP and PM56x-xP
TA562-RS-RTC ↳ *Chapter 1.6.2.9.1.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5125*	TA562-RS-RTC, serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP
TA566 ↳ *Chapter 1.6.2.9.3.2 “ TA566 - Wall mounting accessory” on page 5205*	TA566, wall mounting accessory for S500-eCo I/O modules and AC500-eCo processor modules without DIN rail

Type	Description
TA569-RS-ISO *Chapter 1.6.2.9.1.9 “TA569-RS-ISO - Serial RS-485 isolated adapter” on page 5131*	TA569-RS-ISO, serial RS-485 adapter for PM55x-xP and PM56x-xP
TA570 *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*	TA570, spare part set for AC500-eCo processor modules
TA571-SIM *Chapter 1.6.2.9.1.11 “TA571-SIM - Input simulator” on page 5137*	TA571-SIM, input simulator for PM55x and PM56x
TK501 *Chapter 1.6.2.9.2.9 “TK501 - Programming cable” on page 5186*	TK501, programming cable, D-sub / D-sub
TK502 *Chapter 1.6.2.9.2.10 “TK502 - Programming cable” on page 5188*	TK502, programming cable, terminal block / D-sub
TK503 *Chapter 1.6.2.9.2.11 “TK503 - COM1 USB programming cable” on page 5190*	TK503, COM1 USB programming cable-> D-sub (RS-485), length 3 m
TK504 *Chapter 1.6.2.9.1.12 “TK504 - COM2 USB programming cable” on page 5143*	TK504, COM2 USB programming cable -> D-sub (RS-485), length 3 m

Table 260: Accessories for FieldBusPlugs

Type	Description
PDP22-FBP.025	PDP22-FBP.025 PROFIBUS DP-V0/V1-FBP 0.25 m, modular (FieldBusPlug)
PDP22-FBP.050	PDP22-FBP.050 PROFIBUSc DP-V0/V1-FBP 0.5 m, modular (FieldBusPlug)
PDP22-FBP.100	PDP22-FBP.100 PROFIBUS DP-V0/V1-FBP 1 m, modular (FieldBusPlug)
PDP22-FBP.200	PDP22-FBP.200 PROFIBUS DP-V0/V1-FBP 2 m, modular (FieldBusPlug)
PDP22-FBP.500	PDP22-FBP.500 PROFIBUS DP-V0/V1-FBP 5 m, modular (FieldBusPlug)
PDV11-FBP.0	Feeding connector 24 V DC, Code B-A
PDV12-FBP.0	Feeding connector 24 V DC, Code A-A
PDA11-FBP.050	Adapter M12-D-sub 9-M12, cable length 0.50 m
PDA12-FBP.050	Adapter M12-D-sub 9-M12, cable length 2 x 0.50 m
PDR11-FBP.150	Terminating resistor 150 Ω

1.6.1.8 PLC system description

1.6.1.8.1 AC500 product family

AC500 programmable logic controllers (PLCs)

The AC500 (Standard), AC500-eCo, AC500-S and AC500-XC scalable PLC ranges provide solutions for small, middle and high-end applications. Our AC500 platform offers different performance levels and is the ideal choice for high availability, extreme environments or safety solutions. Our AC500 PLC platform offers interoperability and compatibility in hardware and software from compact PLCs up to high-end and safety PLCs.

Due to the flexible combinations of AC500 devices and components, AC500 PLCs can be used for controlling a wide variety of applications to fulfill your automation needs.

Features of AC500 PLCs

- Scalable and consistently expandable system
- Different performance classes of processor modules (CPUs) available
- Several field busses available
- Parallel connection to several field busses which can be combined arbitrarily

The AC500 product family consists of the product groups:

- AC500 (standard):
  AC500 standard PLCs offer a wide range of performance levels and scalability. The PLCs are highly capable of communication and extension for flexible application.
- AC500-eCo:
  AC500-eCo PLCs are cost-effective, high-performance compact PLCs that offer total interoperability with the core AC500 range and provide battery-free uninterrupted output. All I/O modules can be freely connected in a simple, stable and reliable manner.
- AC500-S:
  AC500-S PLCs are designed for safety applications involved in factory, process or machinery automation area.
- AC500-XC:
  AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. These variants operate according to their product group and can, in addition, be operated under extreme conditions. AC500-XC PLCs can be used at high altitudes, extended operating temperature and in humid condition. Further, the devices provide immunity to vibration and hazardous gases. The AC500-XC series is consistent with standard devices in the overall dimensions, control function and software compatibility. *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*.

The AC500 product family is characterized by functional modularity. As the complete AC500 product family shares the same hardware platform and programming software tool, the devices of the AC500 product groups can be flexibly combined.

S500 devices represent the I/O modules of the product group AC500 (standard), whereas S500-eCo devices represent the I/O modules of the product group AC500-eCo. Both S500 and S500-eCo devices can be combined with devices of the AC500 product family in a flexible way.

Extensions in the product name

AC500 devices support different protocols and technologies (e.g. Ethernet, PROFIBUS etc.) in variable number. AC500 devices with onboard interfaces for support of a certain protocol or technology can be identified easily by the extension in the product name of the AC500 device. For example the AC500 Communication Module PM592**-DP** provides onboard support for PROFIBUS DP, the AC500 processor module PM595**-4ETH** provides onboard support for four provided Ethernet interfaces.

Further extensions in AC500 device names:

- -ETH: Ethernet
- -ARC: ARCNET
- -DP: PROFIBUS DP
- -CAN: CAN/CANopen
- -ETHCAT: EtherCAT
- -PNIO: PROFINET

- -RCOM: RCOM/RCOM+
- -RS: Serial interface

1.6.1.8.2 AC500/S500 system structure

The AC500 product family provides a variety of modules and pluggable components for expanding the capabilities of the CPU with additional I/Os or other communication protocols. Depending on the features and functions of the processor module (CPU) compatible components can be added to a complete AC500 PLC system.

Example of an AC500 PLC system:

1 Processor module. In the example, without a LCD display or keypad.
2 Plug-in communication module (AC500-S)
3 Plug-in communication module (AC500 Standard)
4 Plug-in I/O module (AC500-S)
5 Plug-in I/O module (AC500 Standard)
6 Plug-in function module (AC500-eCo)

Centralized I/O extension

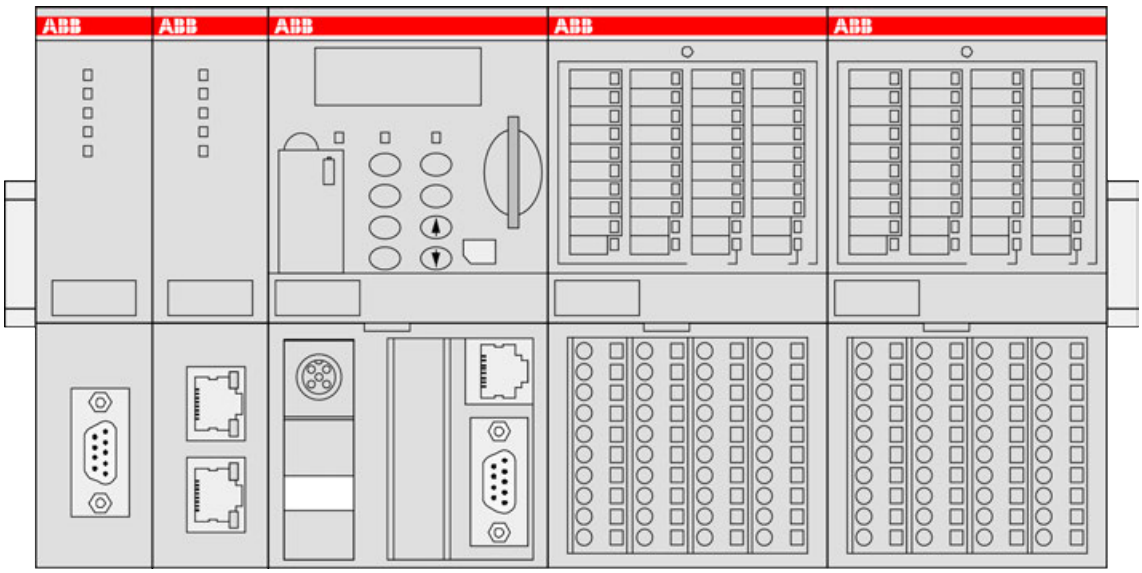


Fig. 723: S500 I/O modules directly connected to a processor module

Decentralized I/O extension

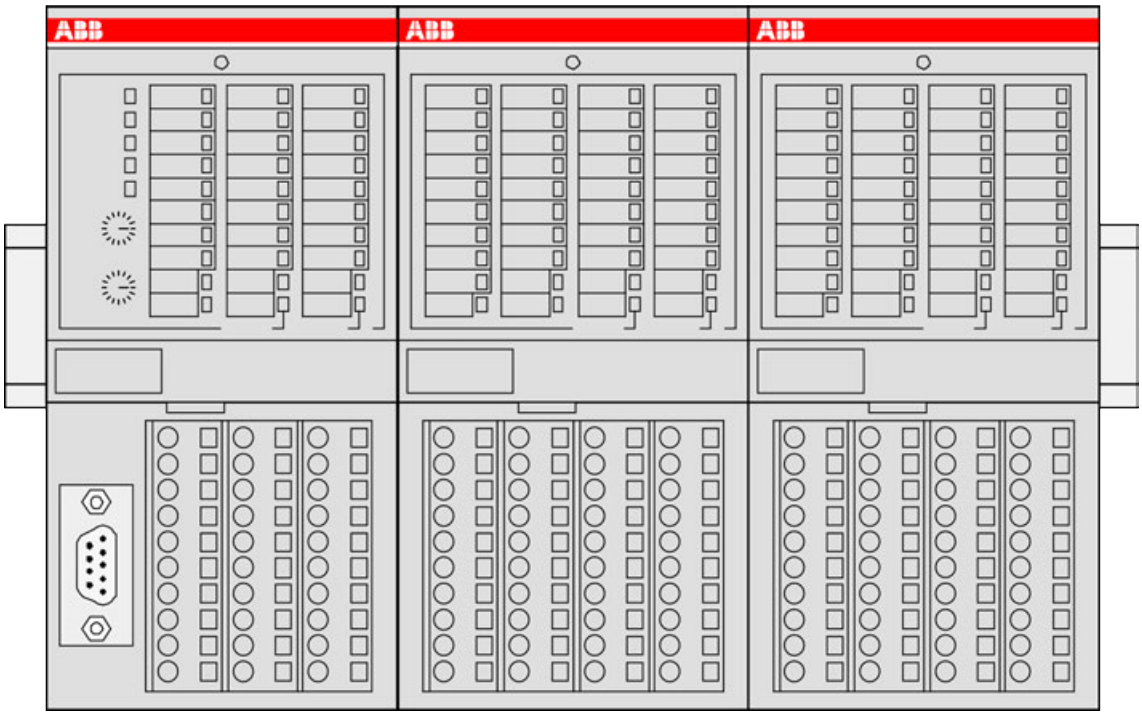


Fig. 724: S500 I/O modules connected via communication interface module

1.6.1.8.3 AC500-eCo/S500-eCo system structure

AC500-eCo/S500-eCo serie is fully compatible with AC500/S500. Customers can flexibly select S500-eCo I/O modules and S500 I/O modules to realize economical and upgradeable solutions.

The AC500-eCo processor module can run independently but also carry out centralized I/O extension, decentralized I/O extension and network extension.

Centralized I/O extension

Centralized I/O extension means that the processor module is connected with the I/O module via I/O bus on the side to increase control points and functions.

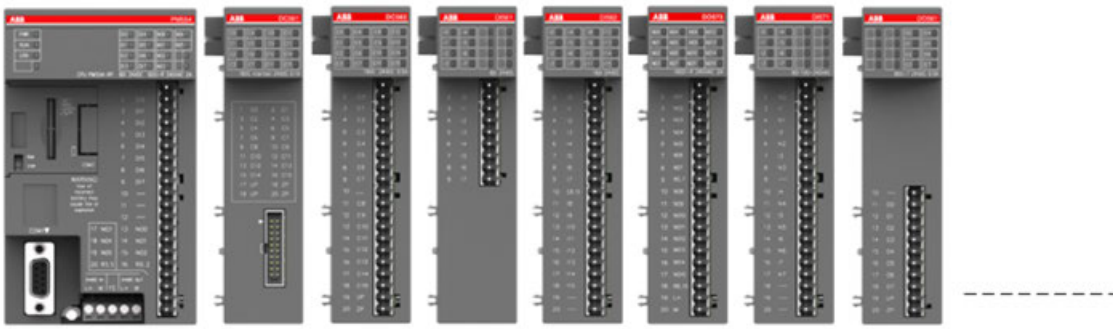


Fig. 725: I/O modules (S500-eCo) directly connected to a AC500-eCo processor module

Decentralized I/O extension

Usually, an AC500-eCo decentralized I/O station is realized with an CS31 module. As a master station, the processor module communicates with the CS31 slave station module (such as DC551) via the serial port COM1 connected to the bus.

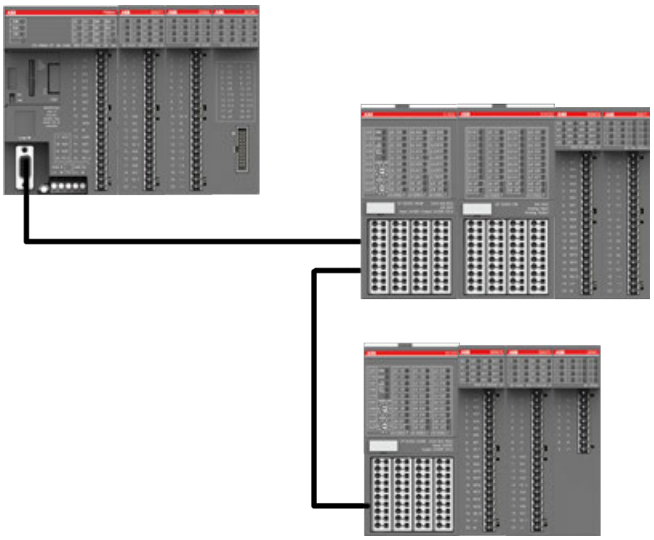
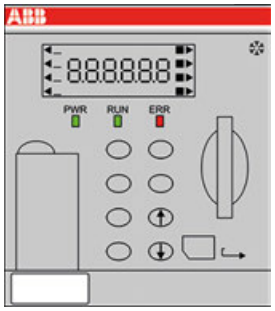


Fig. 726: I/O modules (S500-eCo) connected to a AC500-eCo processor module via CS31 bus

1.6.1.8.4 AC500/S500: Short description hardware

Processor modules



AC500 processor modules contains the CPU with the core component of the PLC. The CPU is connected with the user memory, input and output module, communication port and other units via system bus and performs tasks by means of system programs preset in the system memory. The CPU adopts the function preset by the system program to command the PLC for operation.

Its functions include:

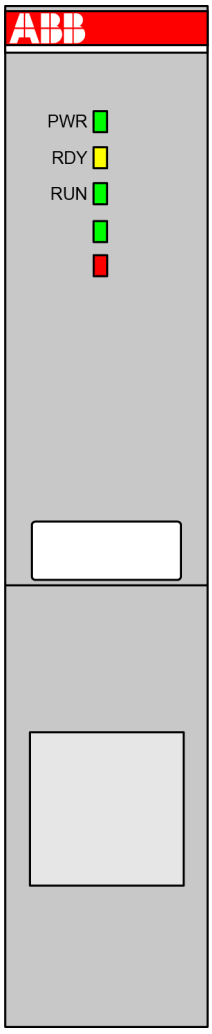
- To receive user program and data entered
- To diagnose work faults of the power supply and PLC circuit as well as syntax error in programming
- To receive the state or data of the site via the input interface and save it into the shadow register or data register
- To read the user program in the memory one by one and execute it after interpretation
- To update the state of related flag bits and output shadow register contents according to execution results and realize output control by means of output unit.

Processor modules are available in different performance classes. Only one processor module is required for a valid system architecture.

There are different types of processor module available that differ in the features and functions they provide, e.g. performance, LED display etc.

If required, processor modules are also available with an integrated Ethernet communication module (TCP/IP).

Communication modules



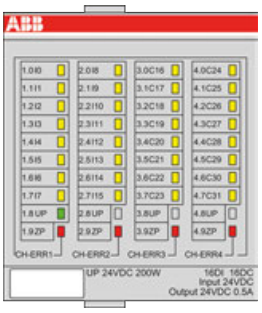
AC500 communication modules are required for

- a connection to standard field bus systems and
- for integration into existing networks.

AC500 communication modules

- enable communication on different field buses.
- are mounted on the left side of the processor module on the same terminal base.
- are directly powered via the internal communication module bus of the terminal base. A separate voltage source is not required.

I/O modules



The I/O modules are the input / output unit which connects the PLC with the industrial production site. The PLC can detect controlled object data via the input interface and the data is taken as the basis for PLC control on the controlled object. In addition, the PLC sends processing results via the output interface to the controlled object to realize the control purpose.

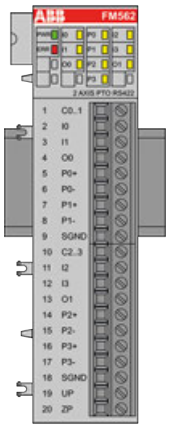
External input equipment and output equipment need various signal levels while the information processed by the CPU in the PLC only can be the standard level. In order to realize such conversion, the I/O interface generally shall perform optical isolation and filtering to improve interference immunity of the PLC. In addition, the I/O interface generally can indicate the working state to facilitate maintenance.

The PLC provides multiple I/O interface for operation level and drive capability to users for selection such as digital input, digital output, analog input, analog output, etc. I/O interfaces of the PLC have the number of input / output signals taken as the number of PLC I/O points. The number of I/O points is an important basis for PLC selection. If the system is insufficient in the I/O points, it can be expanded via the I/O extension interface of the PLC.

The I/O modules for digital and/or analog inputs and outputs are available in different versions and allow flexible use thanks to configurable channels.

The modules can be simply plugged onto a terminal unit for a centralized I/O extension or for a decentralized I/O extension via communication interface modules.

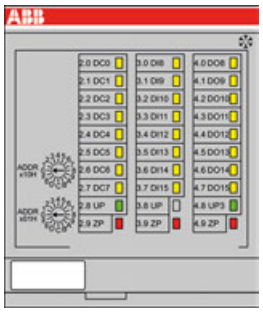
Function modules



Function modules extend the PLC system to perform special task control. Those modules often provide independent components such as a CPU, system programs, memory and interfaces connected with the PLC system bus.

It is connected with the PLC via the I/O bus to exchange data and independently work under cooperative management of the PLC.

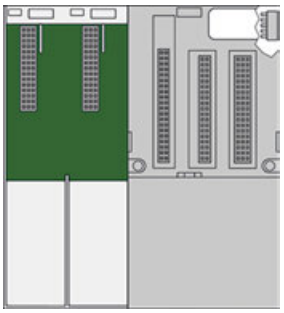
Communication interface modules



Communication interface modules enable a decentralized I/O station. It contains embedded digital I/Os and a fieldbus interface.

Communication interface modules act as I/O slave devices within a master-slave-arrangement.

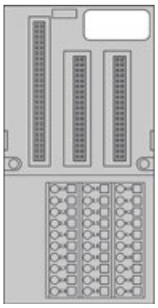
Terminal bases



On a terminal base the processor module and the communication modules are plugged.

> *For AC500-eCo processor modules and special AC500 (Standard) processor modules the terminal base cannot be removed.*

Terminal units



On a terminal unit the I/O modules are plugged.

Terminal units enable simple prewiring without electronics and are available for 24 V DC and 230 V AC, optionally for spring or screw-type terminals.

Memory

In the PLC, the memory is mainly used for saving system programs, user programs and work data. The following memory types can be distinguished:

- Volatile memory:
  All saved data will be lost after power failure of the memory but the memory can provide high access rate and unlimited programming cycles. Common volatile memories mainly include SRAM and DRAM (including common memories such as SDRAM).
- Nonvolatile memory:
  All saved data will not be lost after power failure of the memory, but the memory is subject to low read-write rate and limited rewrite cycles. Common nonvolatile memories mainly include NORflash, NANDflash, EEPROM, memory card, etc.

AC500 PLCs store all user programs in the nonvolatile memory to get protected from power failure. The programs are exported to the volatile memory under operation of the PLC to ensure high-speed and efficient operation. If user program debugging is finished, the programs can be fixed in the nonvolatile memory when they need no change. The work data is subject to frequent change and access in the PLC operation. It is saved in the volatile memory to meet the requirements for random access.

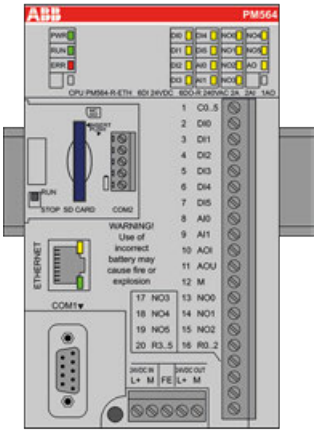
The work data memory of PLC has the memory area for input and output relay, auxiliary relay, timer, counter and other logic devices. The state of these devices depends on initial setting and operation of the user programs. Some data maintains existing state by using built-in supercapacitors or backup batteries in case of power failure. The memory area for data saving in case of power failure is called the data retention area.

Power supply

The PLC is equipped with a switch power supply for internal circuit. Compared with ordinary power supply, the PLC power supply has the higher stability and interference immunity. A number of PLC products provide 24 V DC stabilized voltage supply to meet external sensors.

1.6.1.8.5 AC500-eCo/S500-eCo: Short description hardware

Processor modules



AC500-eCo processor modules contains the CPU with the core microprocessor of the PLC. It is integrated with power supply, input channel and communication interface.

Functions:

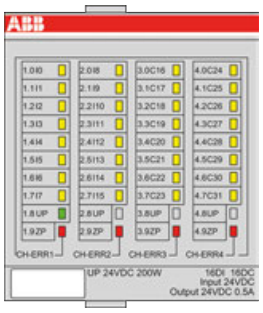
- To download user programs
- To run the CPU
- To execute user programs in loops
- To monitor program input and output devices.

Processor modules are available in different performance classes and provide different amounts onboard I/Os.

Only one processor module is required for a valid system architecture.

If required, processor modules are also available with Ethernet interface.

I/O modules



If the amount of onboard I/Os provided on the processor module is insufficient for a certain use case, the PLC can be expanded with I/O modules to meet the control requirements.

Memory

In the PLC, the memory is mainly used for saving system programs, user programs and work data. The following memory types can be distinguished:

- Volatile memory:
  All saved data will be lost after power failure of the memory but the memory can provide high access rate and unlimited programming cycles. Common volatile memories mainly include SRAM and DRAM (including common memories such as SDRAM).
- Nonvolatile memory:
  All saved data will not be lost after power failure of the memory, but the memory is subject to low read-write rate and limited rewrite cycles. Common nonvolatile memories mainly include NORflash, NANDflash, EEPROM, memory card, etc.

AC500 PLCs store all user programs in the nonvolatile memory to get protected from power failure. The programs are exported to the volatile memory under operation of the PLC to ensure high-speed and efficient operation. If user program debugging is finished, the programs can be fixed in the nonvolatile memory when they need no change. The work data is subject to frequent change and access in the PLC operation. It is saved in the volatile memory to meet the requirements for random access.

The work data memory of PLC has the memory area for input and output relay, auxiliary relay, timer, counter and other logic devices. The state of these devices depends on initial setting and operation of the user programs. Some data maintains existing state by using built-in supercapacitors or backup batteries in case of power failure. The memory area for data saving in case of power failure is called the data retention area.

Power supply

The CPU is equipped with a switch power supply for internal circuit. Compared with ordinary power supply, the PLC power supply has the higher stability and interference immunity. A number of PLC products provide 24VDC stabilized voltage supply to meet external sensors.

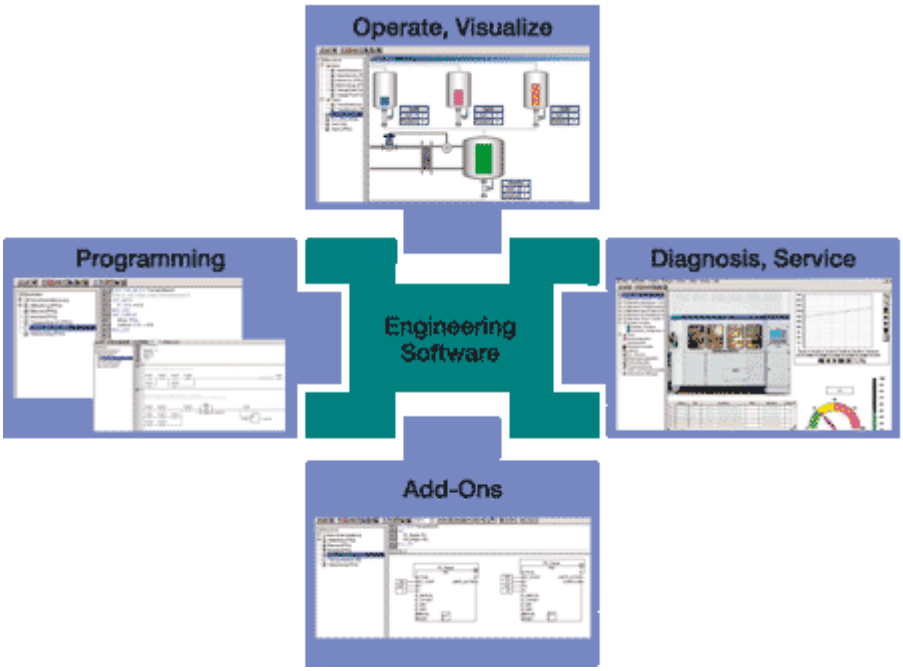
1.6.1.8.6 Short description software

Configuration and programming

Configuration and programming of all AC500 control systems (CPUs) is done by using Automation Builder software.

Features:

- Standardized programming according to IEC 61131-3 - five programming languages (Structured Text (ST), Function Block Diagram (FBD), Instruction List (IL), Ladder Diagram (LD), Sequential Function Chart (SFC)), free graphical function chart (CFC), debugging functions for program test
- Application programming in C/C++
- Online diagnosis
- Debugging functions for the program test: Single step, Single cycle, Breakpoint
- Offline simulation - simulate commands without PLC being connected
- Sampling trace - timing diagrams for process variables
- Recipe management and watch lists
- Visualization
- Configuration of the communication interface modules (for PROFINET, EtherCAT, CANopen, Ethernet, Modbus)
- Programming - serial or via Ethernet networks
- Comprehensive libraries
- Export and import interfaces for devices, signals, applications, visualization, etc.
- Multi-user support and project compare
- Project scripting



Offline simulation IEC 61131-3 commands can be simulated without a PLC being connected, including the relevant malfunctions. After the program test, the application can be downloaded to the control system.

Sampling trace Timing diagrams for process variables and storage of data in a ring buffer with event trigger.

Recipe management and watch lists Values of selected variables are displayed. Pre-defined values can be assigned to variables which can then be downloaded to the control system all at once ("Write recipe"). Actual values from the control system can also be pre-assigned for reading into the Watch and Recipe Manager, and stored in memory there ("Read recipe). These functions are also helpful, for example, for setting and entering control parameters.

Visualization Includes color change, moving elements, bitmaps, text display, allows input of setpoint values and display of process variables read from the PLC, dynamic bar diagrams, alarm and event management, function keys and ActiveX elements.

Programming Serial or via Ethernet networks.

Engineering interface Provides access from the programming system to an external project database in which the program source code of one or several automation projects is managed. Optionally, a version control system, such as Visual Source Safe, can be used in order to ensure data consistency of the program code for several different users and projects.

1.6.1.8.7 Control panels (HMI)

ABB control panels offer a wide range of features and functionalities for maximum operability. The panels are distinguished by their robustness and easy usability, providing all the relevant information from production plants and machines at a single touch.

HMI - human control and operation of machines and processes.

Individual solutions for each application - this enables an operator at any time to have an overview on a profitable production and intervene manually if necessary.

Control panels with TFT graphical display and touch screen.

Available in various resolutions.

1.6.1.9 AC500-S

The AC500-S safety user manual must be read and understood before using safety configuration and programming tools of Automation Builder / PS501 Control Builder Plus. Only qualified personnel shall be allowed to work with AC500-S safety PLCs.

In order to have always the latest version and due to a different lifecycle compared to Automation Builder help, the *AC500-S safety user manual* is only available on our website.

The AC500-S safety PLC includes the following safety-relevant hardware components.

- SM560-S / SM560-S-FD-1 / SM560-S-FD-4
- DI581-S
- DX581-S
- AI581-S
- TU582-S

1.6.1.10 AC500-eCo starter kit

This AC500-eCo Starter kit helps you to get familiar with ABB AC500 PLC offerings and the engineering tool. For that purpose, this manual explains how to connect and setup the components provided in the starter kit and how to program the AC500-eCo CPU by means of several simple example applications.

Contents of the AC500-eCo starter kit:

- 1 x AC500-eCo CPU PM554-TP-ETH with 2 terminal blocks plugged on the CPU: 1 x 11 pin, 1 x 9 pin
- 1 x digital input simulator
- 1 x programming cable

1.6.1.10.1 Preparing the AC500 CPU

Enclosed to the CPU, you find the installation instructions. Refer to these installation instructions for assembling the CPU.

Connecting the input simulator

Enclosed to the input simulator, you find the installation instructions. Refer to these installation instructions to connect the input simulator to the CPU.

Connection of the AC500 CPU

DANGER!

Risk of death by electric shock.

Before using the CPU refer to the “Regulations Concerning the Setting up of Installations” for safety instructions. *http://search-ext.abb.com/library/Download.aspx?DocumentID=3ADR025003*&Action=Launch*

CAUTION!

Risk of damaging the PLC modules.

The CPU can be damaged by overvoltages and short circuits. Make sure that all voltage sources (supply and process voltage) are switched off before you begin with operations on the system.

The CPU needs to be powered by 24 V DC. Use the 5-pin screw-type terminal block for connecting the power.

CAUTION!

Risk of damaging the CPU and the connected modules.

Voltages > 35 V DC can destroy the CPU and the connected modules. Make sure that the supply voltage never exceeds 35 V DC.

Connecting the programming cable

Plug in the provided programming cable to the CPU and your PC.

Set-up communication parameters in Windows

Set-up communication parameters

To set-up the communication between the PC and the PLC, e.g., for downloading the compiled program, you have to set-up the communication parameters.

The IP address of your PC must be in the same class as the IP address of the CPU.

The factory setting of the IP address of the CPU is 192.168.0.10.

The IP address of your PC should be 192.168.0.X. Avoid X = 10 in order to prevent an IP conflict with the CPU.

Subnet mask should be 255.255.255.0.

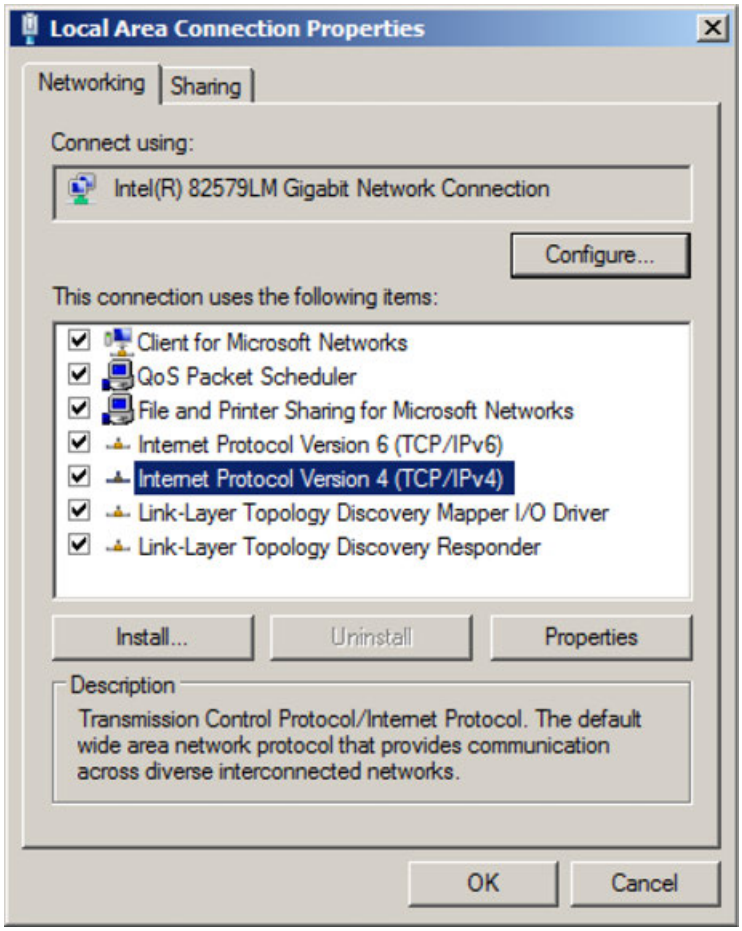
Change the IP address

1. Open Windows **Control Panel**. Click *“Network and Internet ➔ Network and Sharing Center”*.
2. Click **Change adapter settings**.

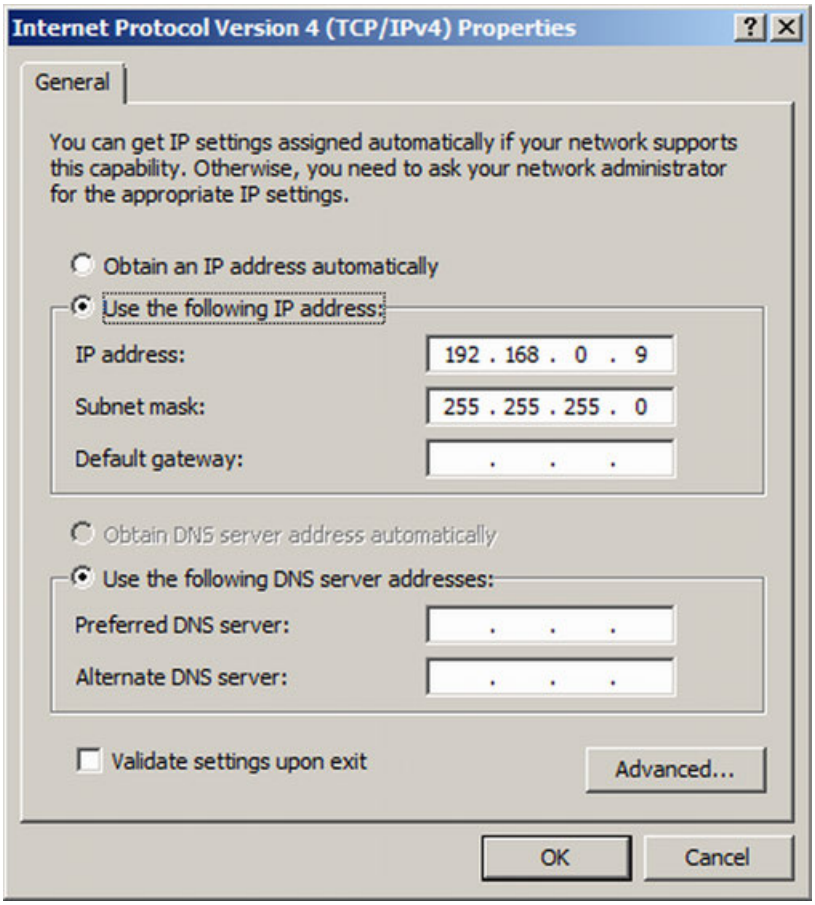
⇨

If using existing network with several devices, please pay attention on given network rules or contact your system administrator.

3. Right-click **Local Area Connection (Ethernet)** and select **Properties**.



4. Double-click **Internet Protocol Version 4 (TCP/IPv4)**.



5. Enter your desired IP address and subnet mask.

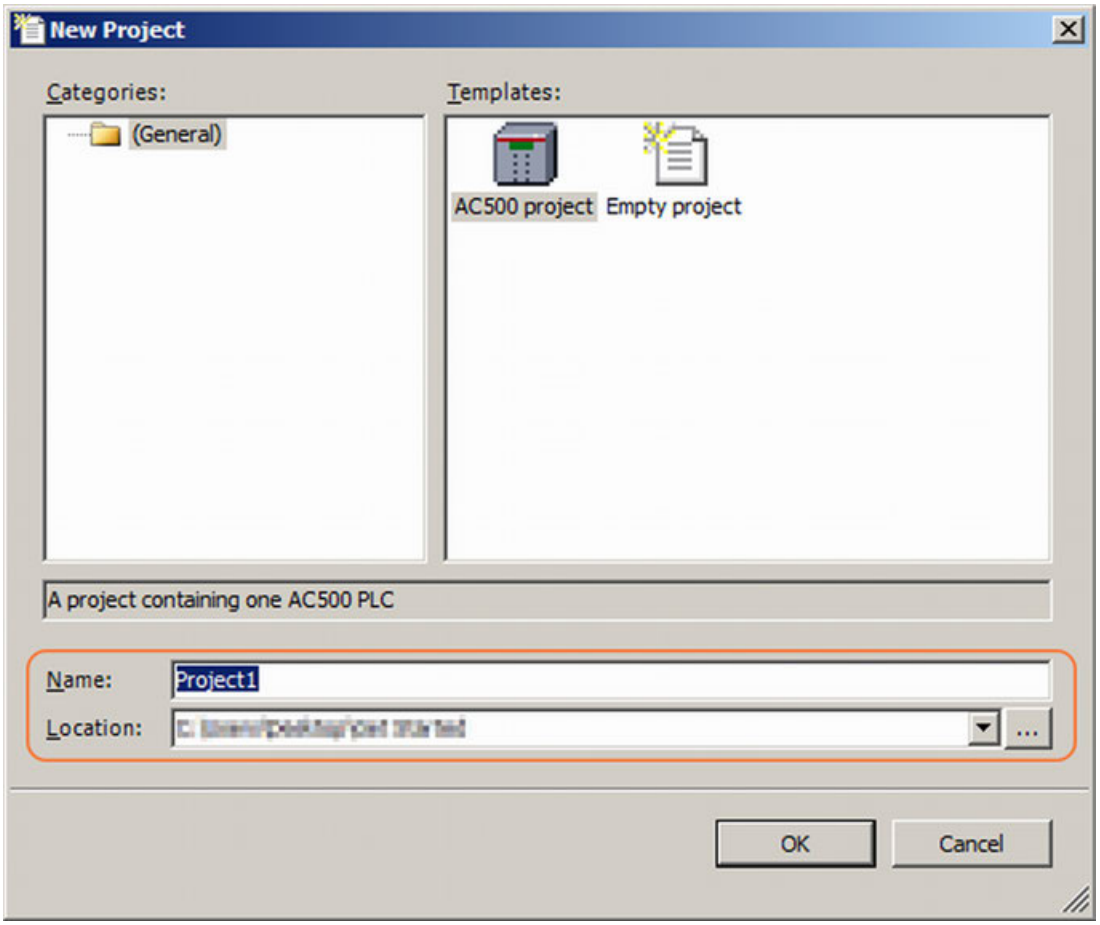
1.6.1.10.2 Step-by-Step introduction to Automation Builder

The following example gives you a brief step-by-step introduction to Automation Builder software and to the programming basics for AC500 PLCs. You will develop and start up a very simple application project with a logical AND function in the programming language Function Block Diagram (FBD).

Configurating the hardware

1. In Automation Builder, from the **File** menu, select **New Project**.
2. Under **Name** enter the project name.

   Under **Location** enter where you want to store the project.



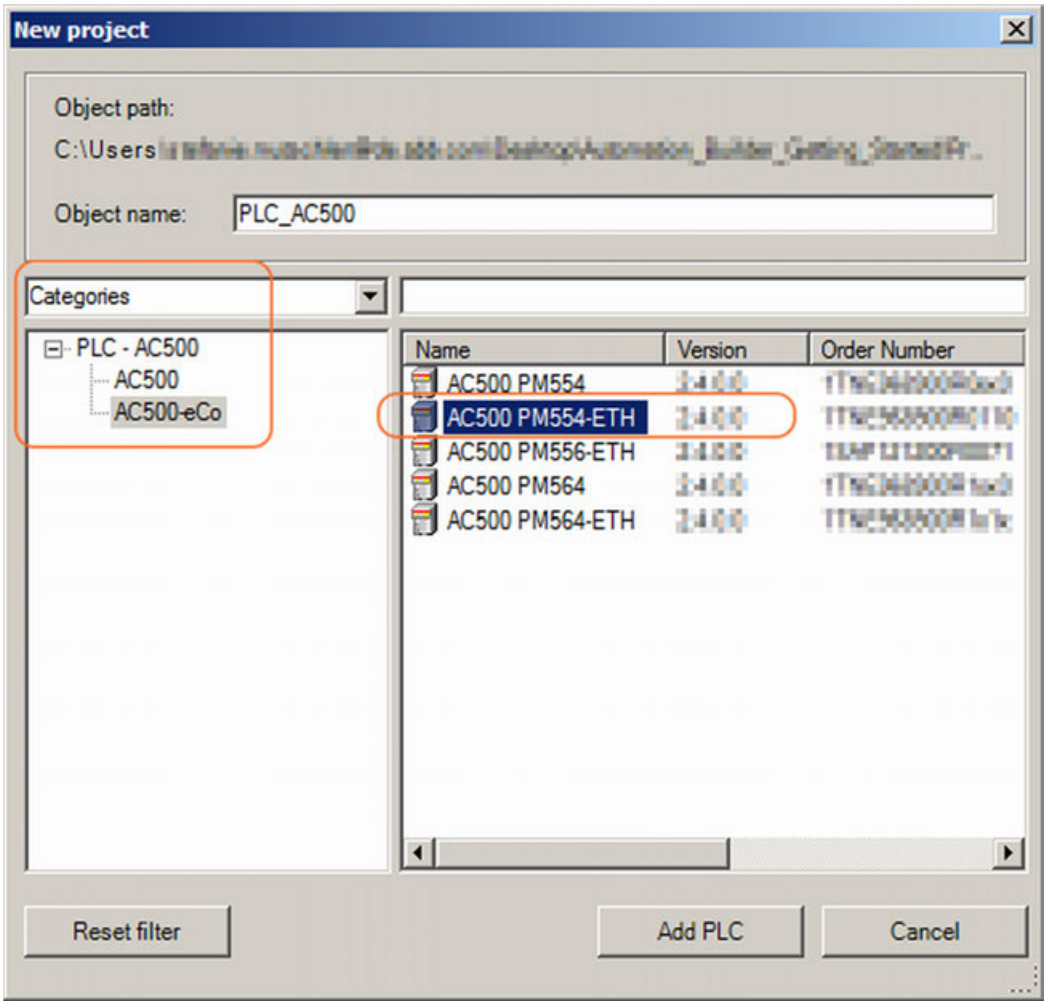
3. Click **OK**.

4. Under **Categories -> PLC - AC500** select **AC500-eCo**.

> *To expand the list, click on the plus sign.*

Select **AC500 PM554-ETH**.

Click **Add PLC**.



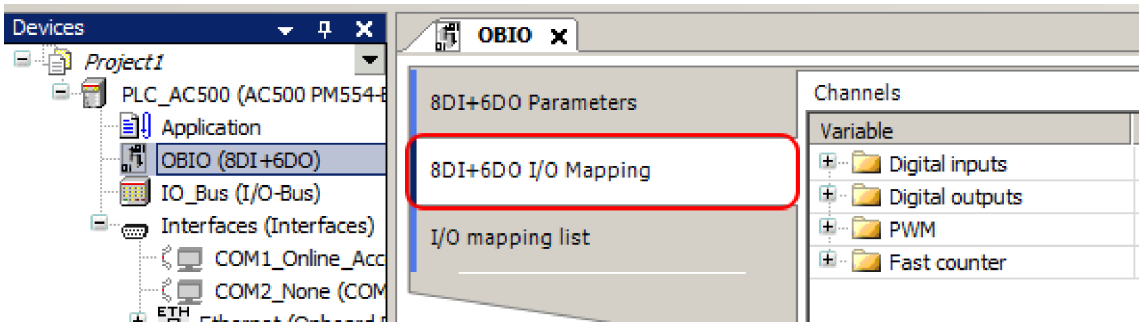
Creating I/O variables

To specify the hardware configuration, you have to define the I/Os and their symbolic names.

> *If you made a mistake during the process, you can always undo. In the CODESYS menu select "Edit ➔ Undo".*

1. Double-click **OBIO** (onboard inputs and outputs).
   ⇨ The **OBIO** tab opens on the right side with several child tabs.

2. Select the **8DI+6DO I/O Mapping** tab.

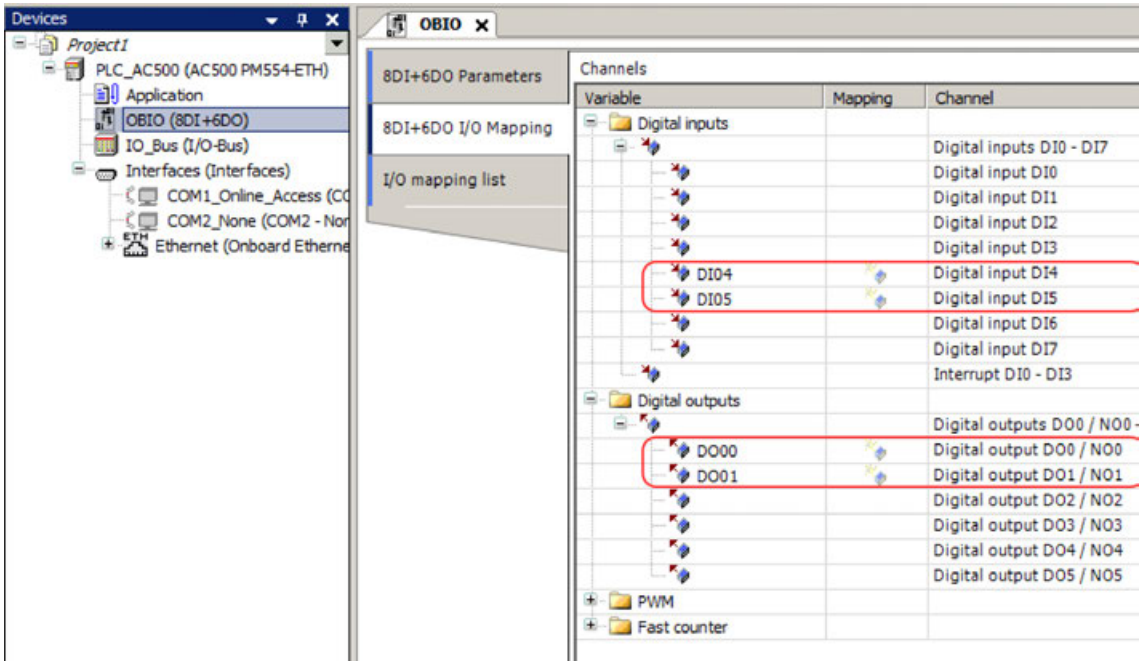


⇨

> *To expand the list, click on the plus sign.*

3. In the **Variable** column, insert the text **DI04** and **DI05** at the corresponding inputs and insert the text **DO00** and **DO01** at the corresponding outputs.

> *To see the number of the inputs and outputs, expand the width of the **Channel** column.*

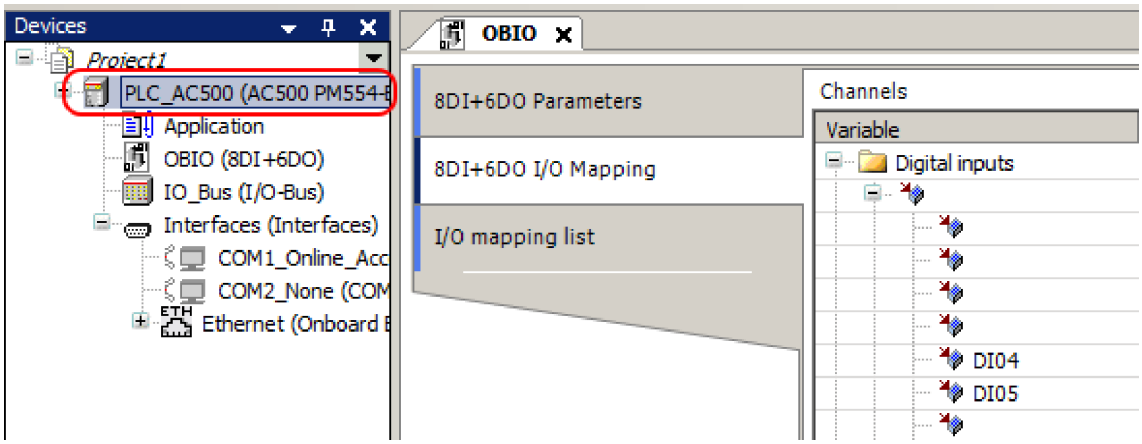


⇨ You added two digital inputs and two digital outputs to your AC500-eCo CPU.

Behavior of the error LED

If the Automation Builder software version and the AC500-eCo CPU firmware version are different, the red Error LED will go on. This has no influence on running your example program on your AC500-eCo CPU. The example program can be executed with any firmware version of the AC500-eCo CPU. In Automation Builder you can define if the Error LED is enabled or disabled.

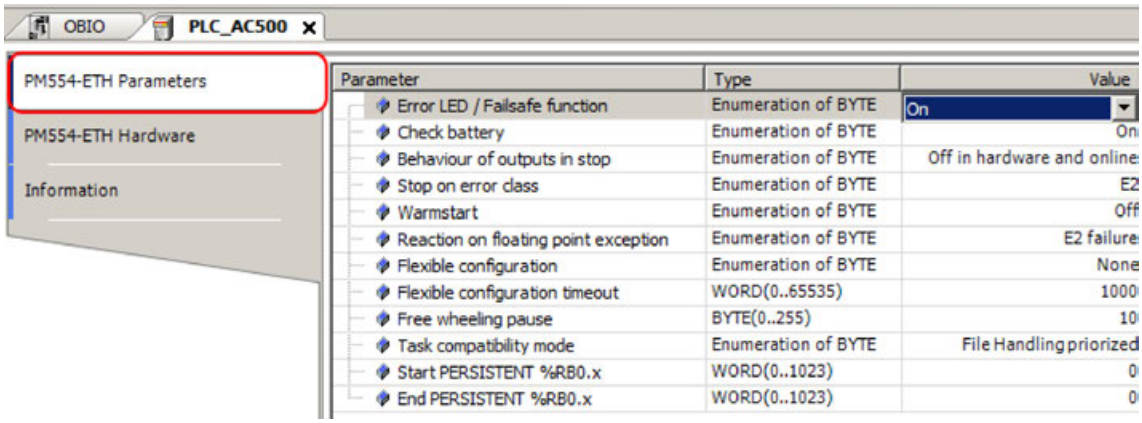
1. Double-click **PLC_AC500**.



⇨ The **PLC_AC500** tab opens on the right side with several child tabs.

2. Select the **PM554-ETH Configuration** tab.

   In the **Error LED / Failsafe function** row, double-click on the entry of the **Value** column.



3. Select **Off by E4** to disable the Error LED.

   Select **On** to enable the Error LED.

4. Close the **PLC_AC500** tab.

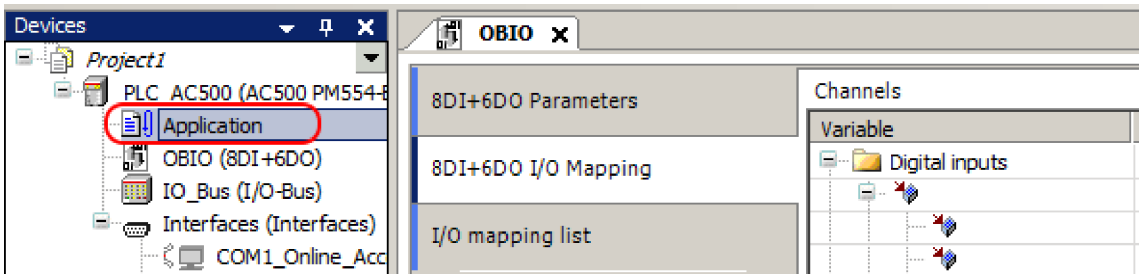
Programming your project

You write the program code in a separate IEC 61131-3 editor (CODESYS). You can open CODESYS out of Automation Builder.

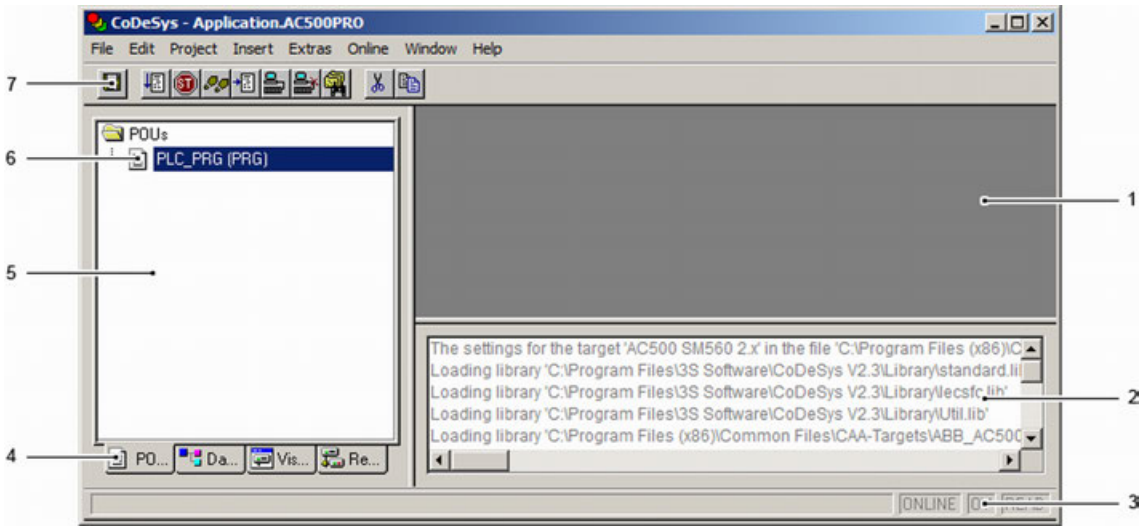
Supported programming languages:

- ST (Structured Text)
- IL (Instruction List)
- FBD (Function Block Diagram)
- LD (Ladder Diagram)
- SFC (Sequential Function Chart)
- CFC (Continuous Function Chart)

▷ To open CODESYS, double-click **Application**.



⇨ CODESYS opens in a new window.



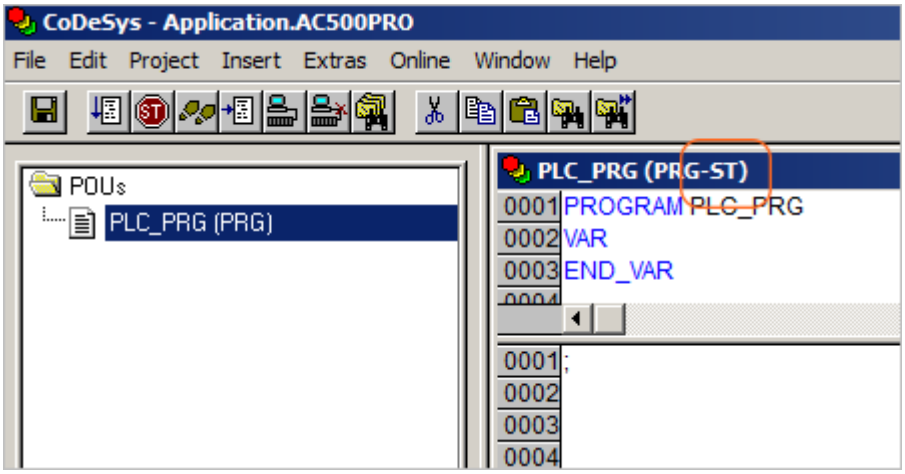
1 Work space
2 Message window
3 Status bar
4 Register cards: POUs (Program Organization Units), Data types, Visualizations, Resources
5 Object organizer
6 Automatically generated POU
7 Toolbar

Change the programming Language into FBD

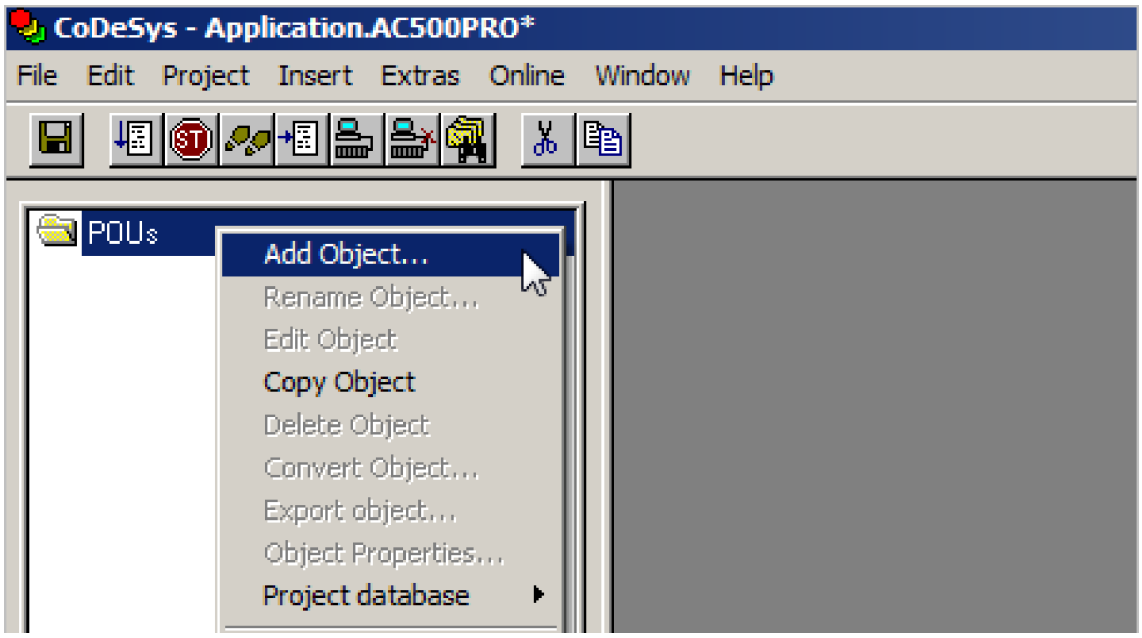
Default programming language is ST.

1. Select *“PLC_PRG (PRG)”* and press *[DELETE]*.

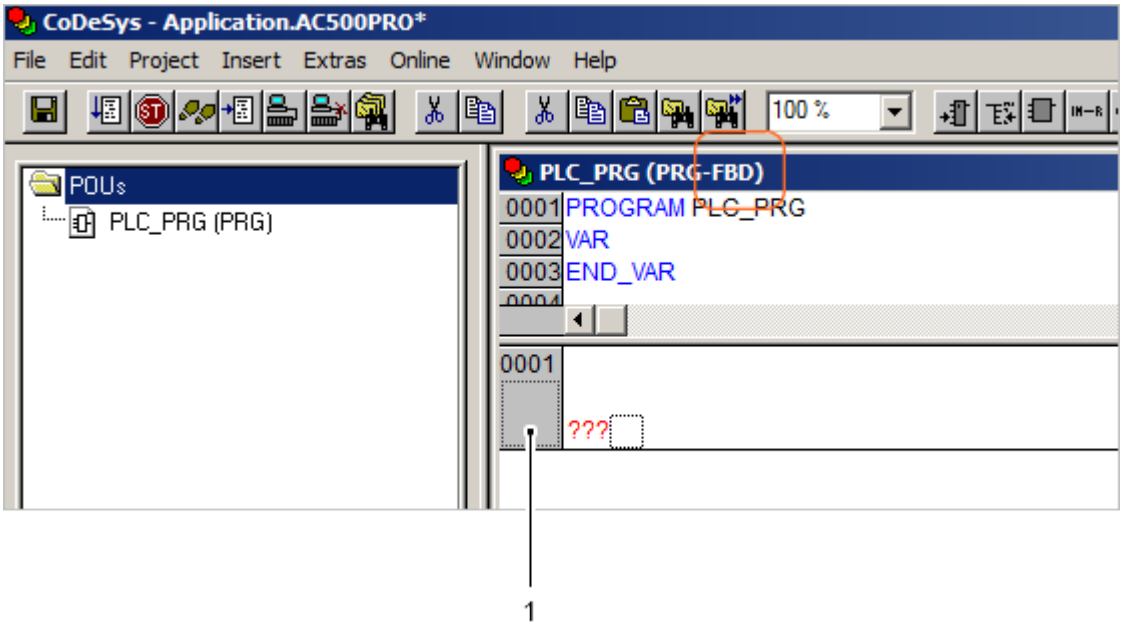
> *If you made a mistake during the process, you can always undo. In the CODESYS menu select “Edit ➔ Undo”.*



2. Right-click **POUs** and select **Add Object**.



3. Under **Language of the POU**, select **FBD**. Click **OK**.
   ⇨ A newly generated POU opens. The programming language of the POU is FBD.

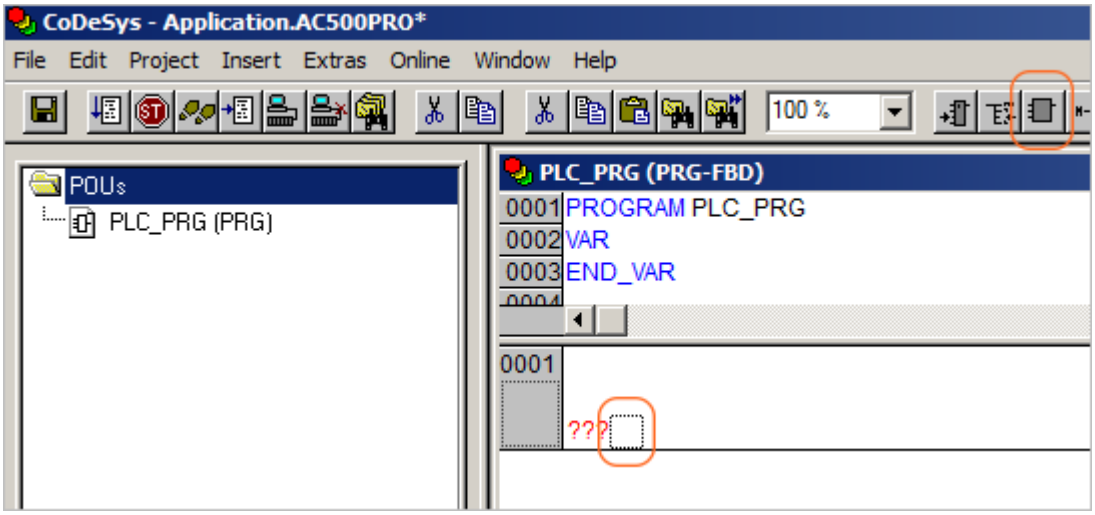


1 Network (here network number 0001)

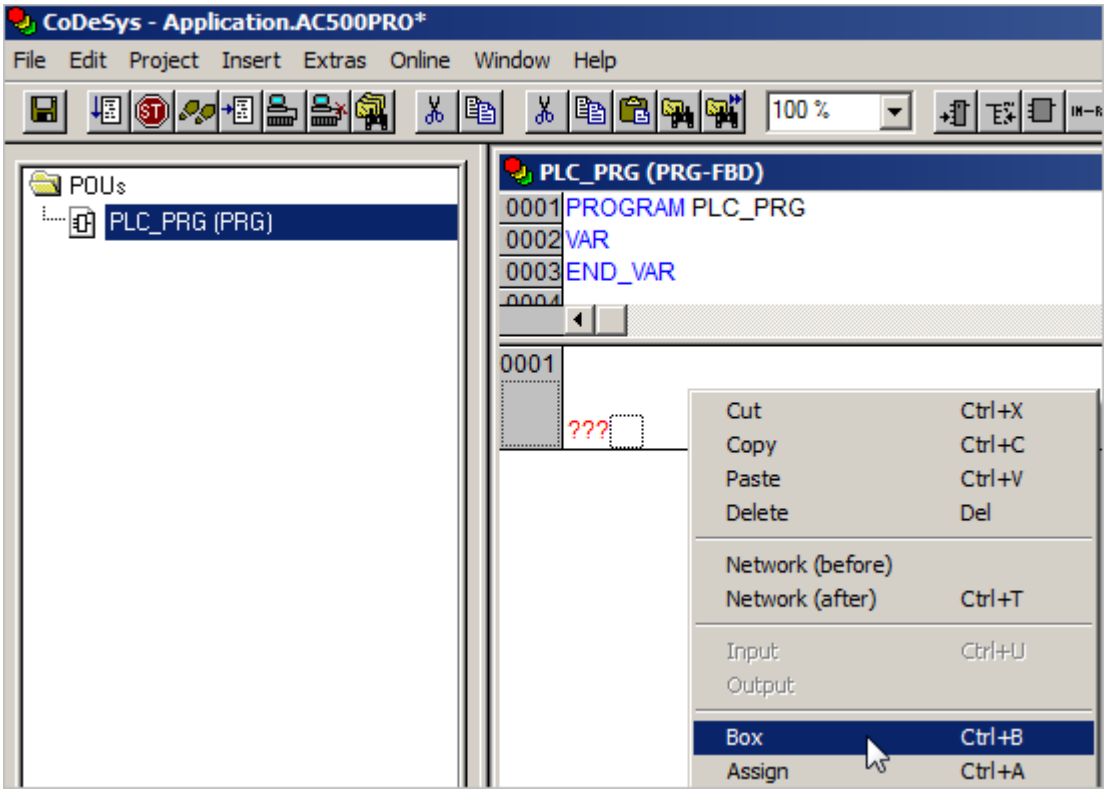
4. Click on the **Save** button.

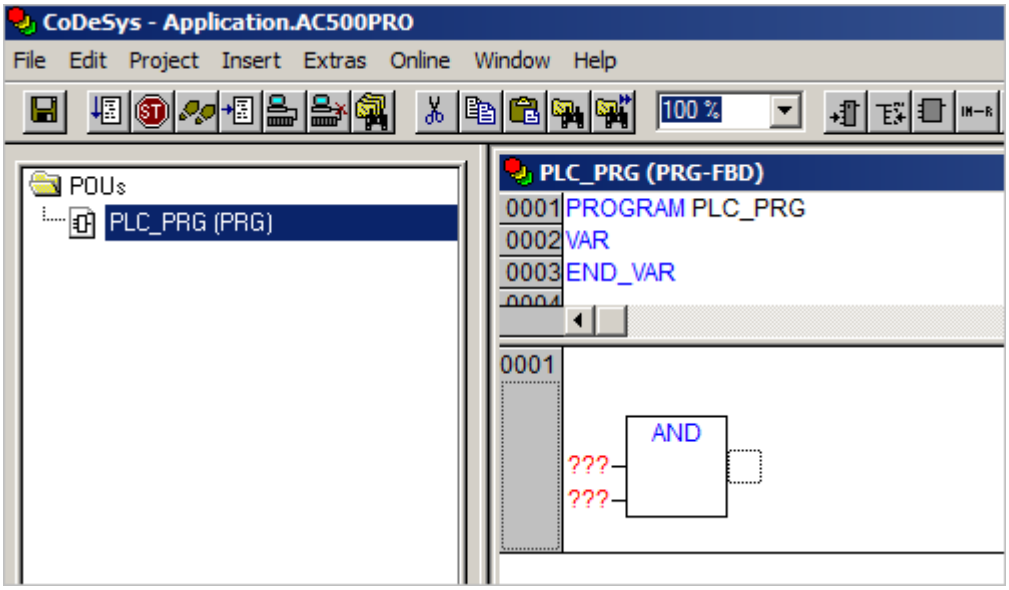
Creating an AND operator

1. Click on the dotted rectangle behind *"???"*. From the toolbar, click on the **Box** icon.
   The dotted rectangle represents the cursor position in this network.



2. Alternatively, right-click on the dotted rectangle and select **Box**.



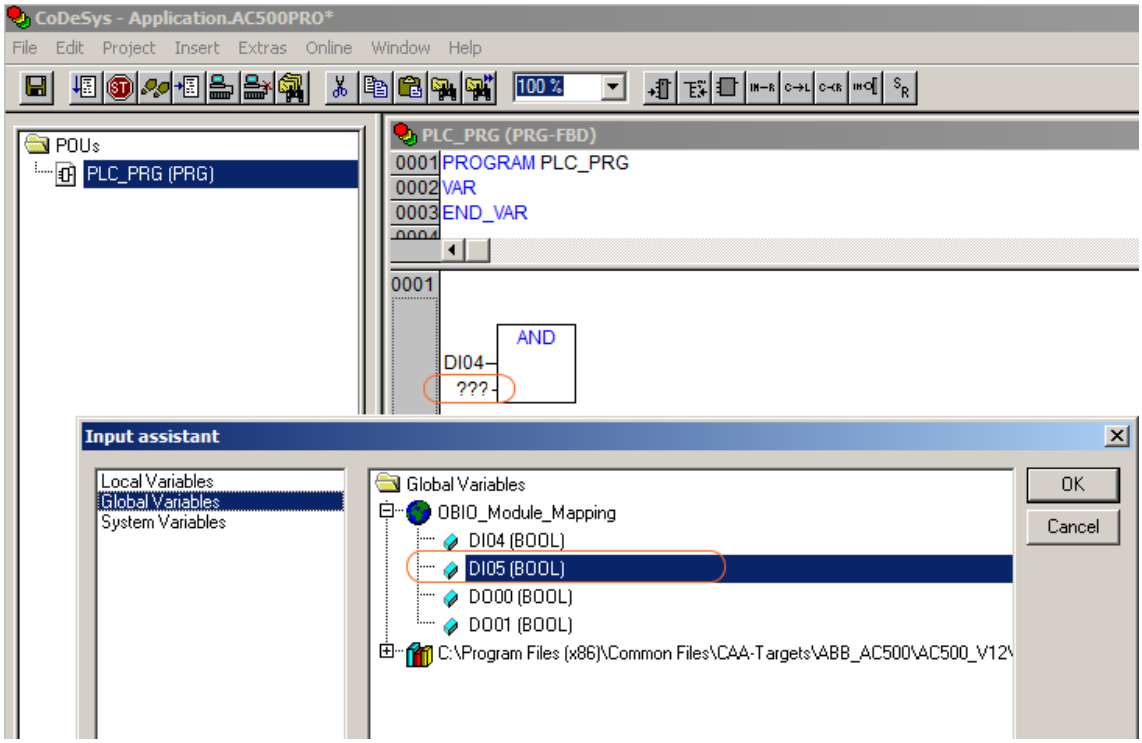


⇨

> *When inserting a new box, it always appears as an AND operator. You can change the type of the box at any time. Click on the AND text and type another name. Or click on the AND text and press F2. You will get an overview of all accepted operators, functions and function blocks.*

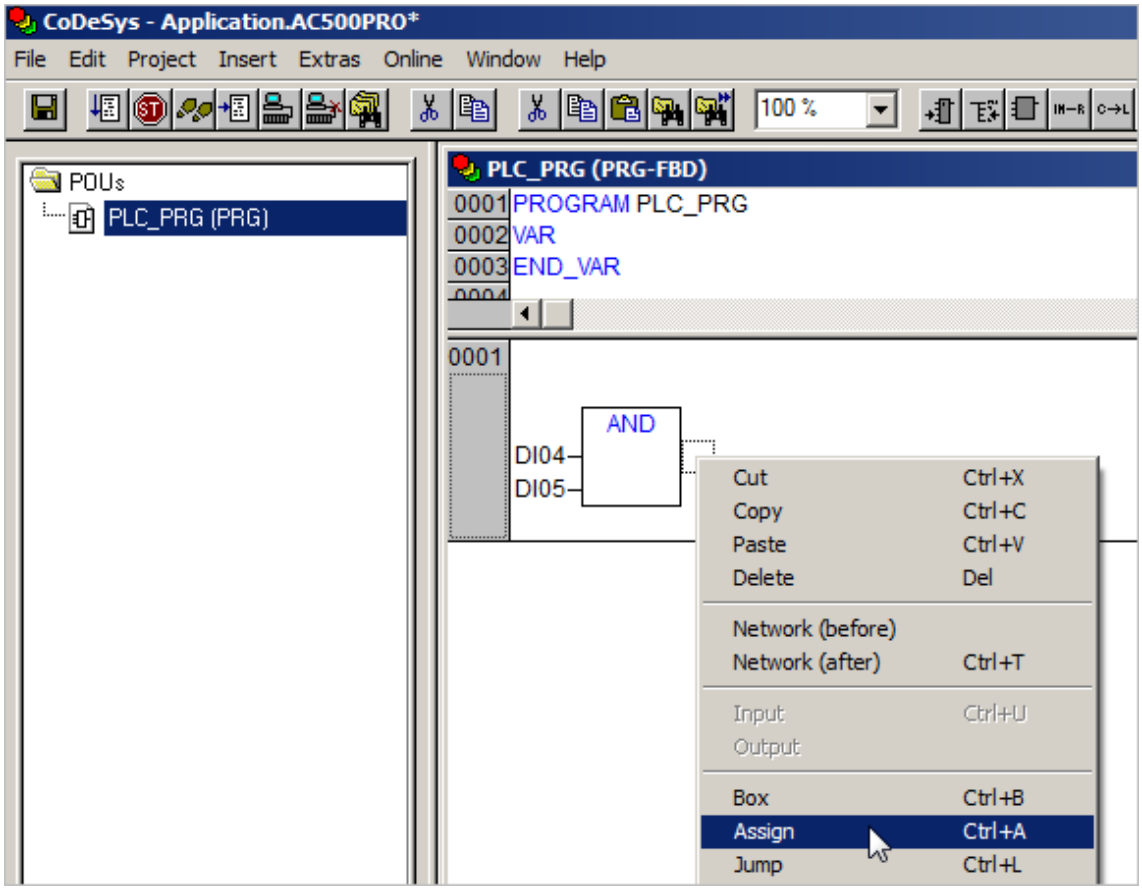
3. Click on *"???"* and press F2.
   ⇨ The input assistant opens.

4. Select **Global Variables**. Select the variable you want to assign.

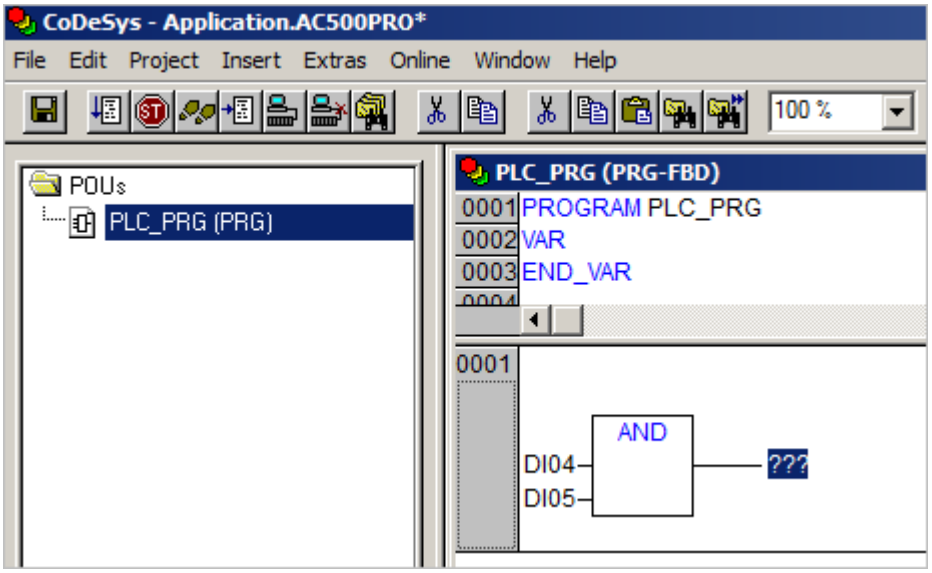


5. Alternatively, click on *"???"* and type DI04 for input 1 and DI05 for input 2.

6. To assign an output: Click on the right side of the AND operator. Right-click on the dotted rectangle and select **Assign**.



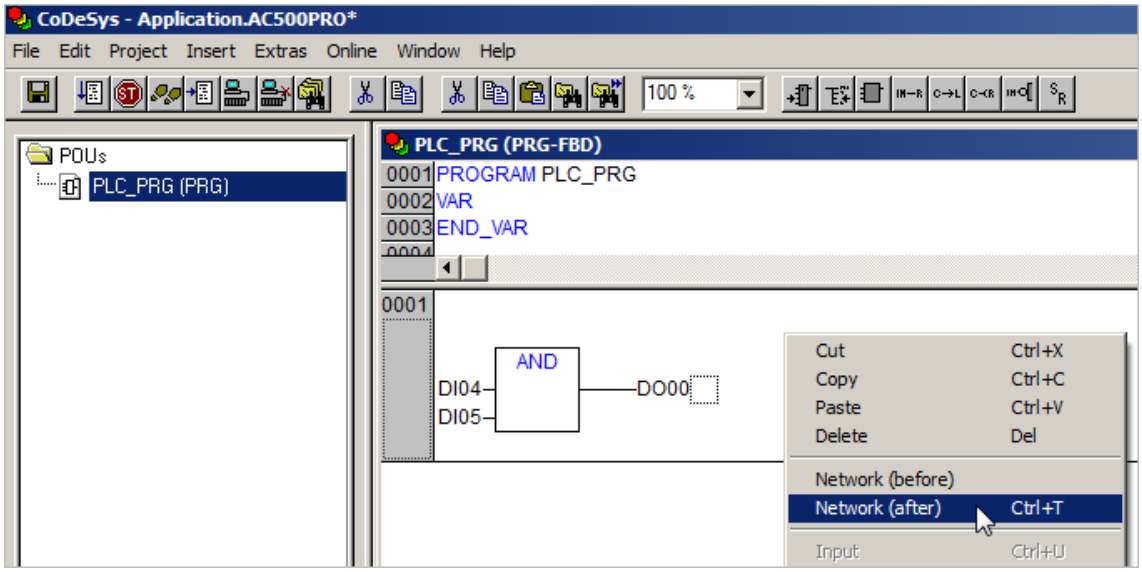
⇨ You assigned an output.



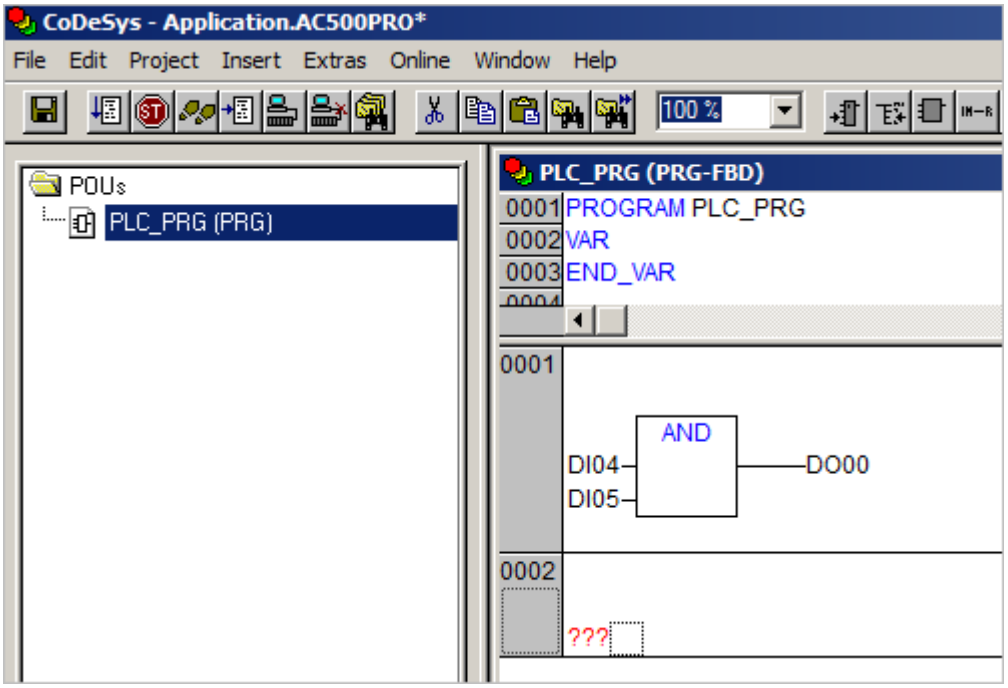
7. Type DO00 for the output. Or press F2 for opening the input assistant.

Add a new network

▷ Right-click on network 0001 and select *“Network (after)”*.



⇨ You added a new network after network 0001. The new network number is 0002.

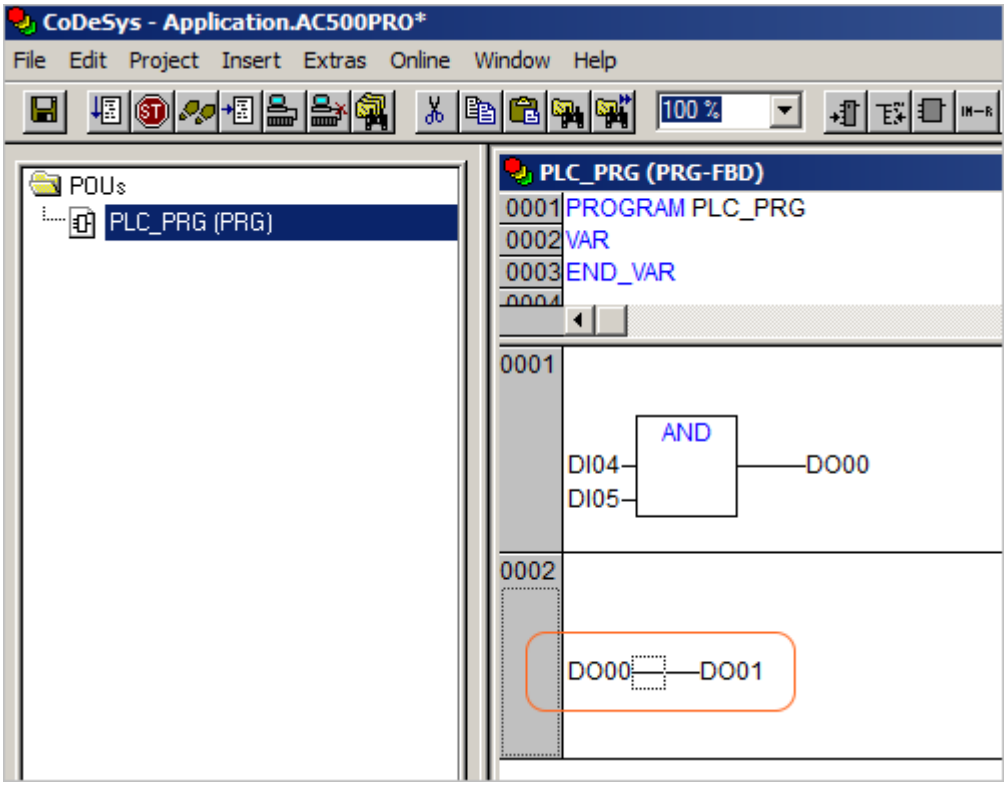


Insert a Negation

1. Right-click on the dotted rectangle and select *“Assign”*.
2. Type DO00 for input and DO01 for output.

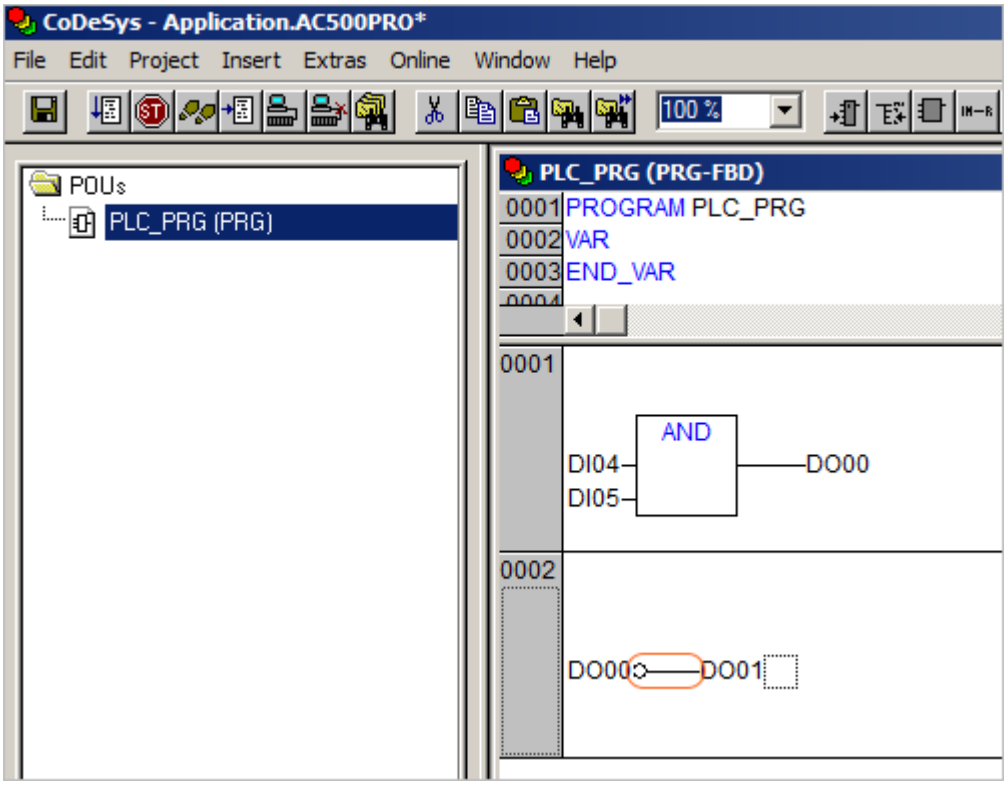
3. Click behind the input DO00.

   ⇨ The dotted rectangle represents the cursor position. The cursor position is behind DO00.



4. Right-click on the dotted rectangle and select **Negate**.

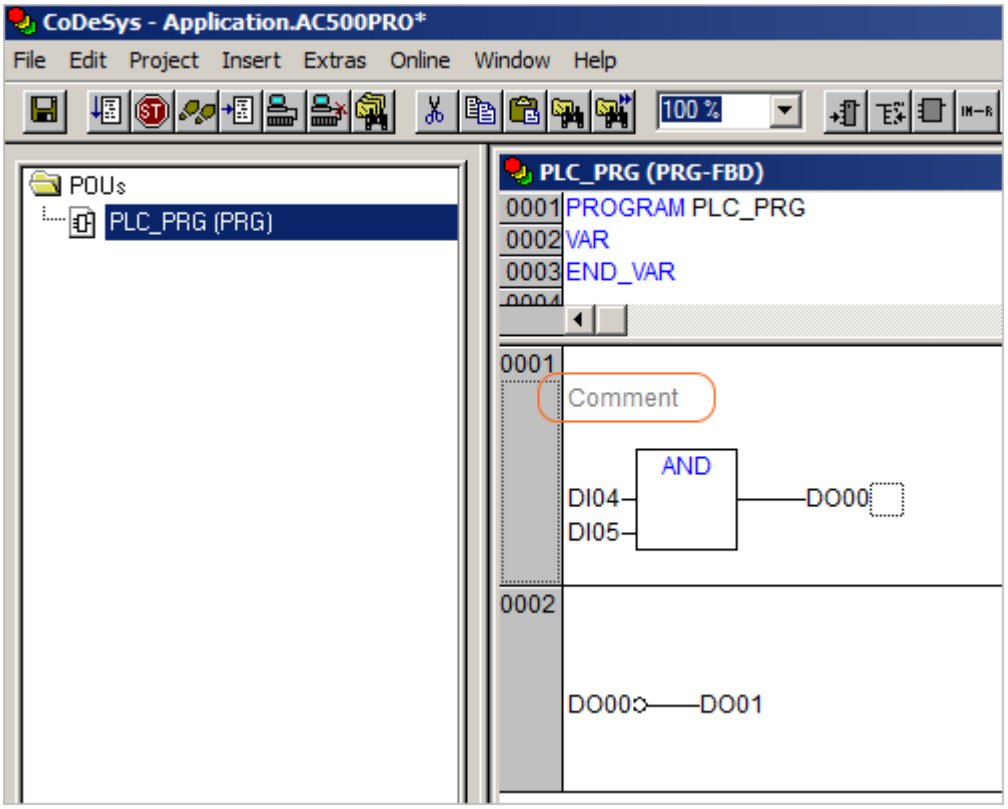
   ⇨ You inserted a negation between DO00 and DO01.



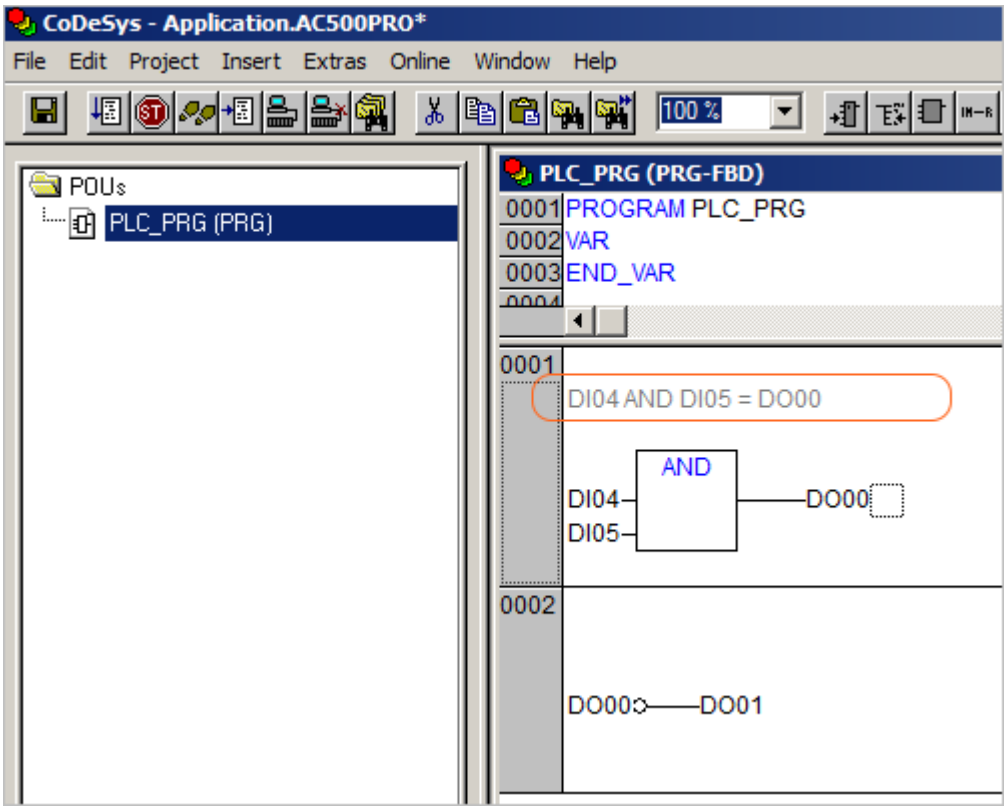
Insert a Comment

1. Right-click on network 0001 and select *"Comment"*.

   ⇨ The text *Comment* appears gray colored on first position in network 0001.

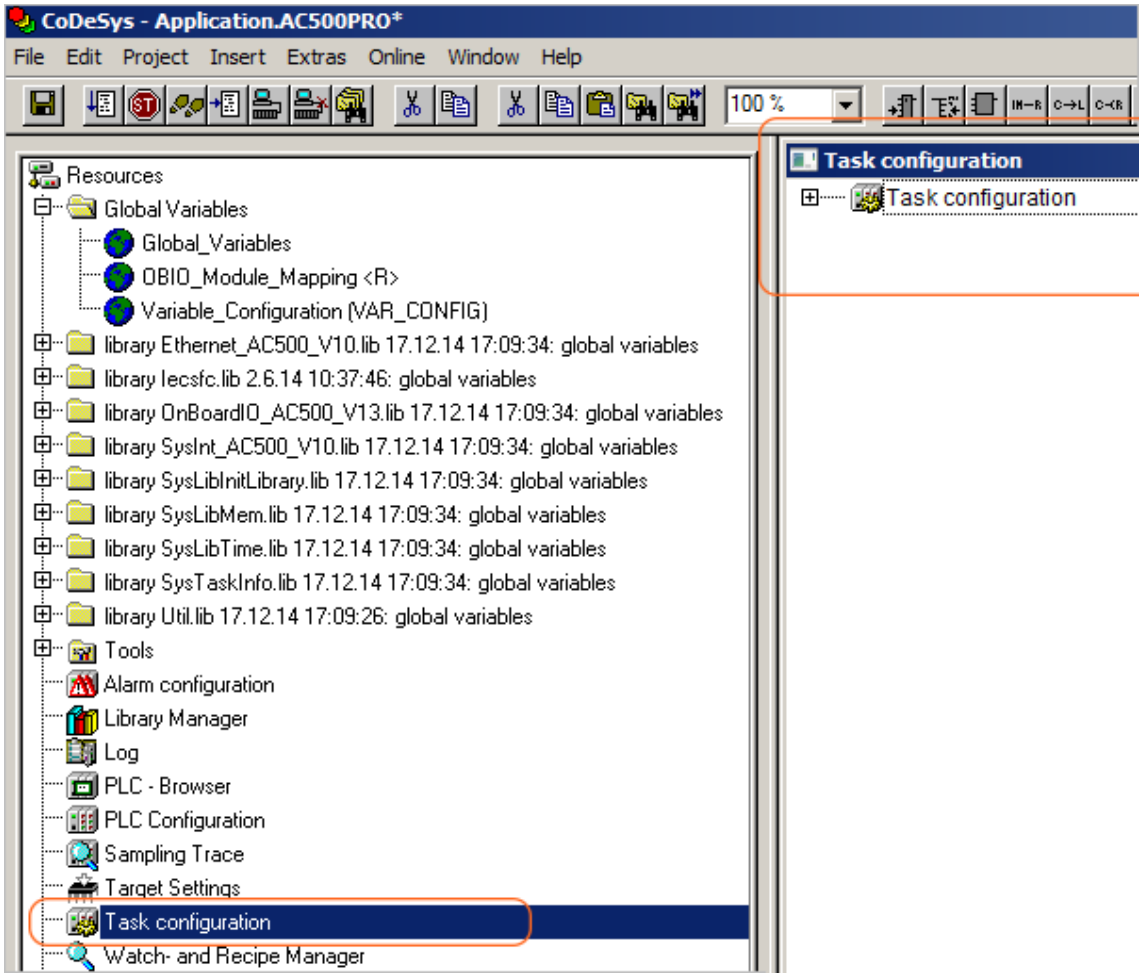


2. Double-click on *Comment* and insert *DI04 AND DI05 = DO00*.

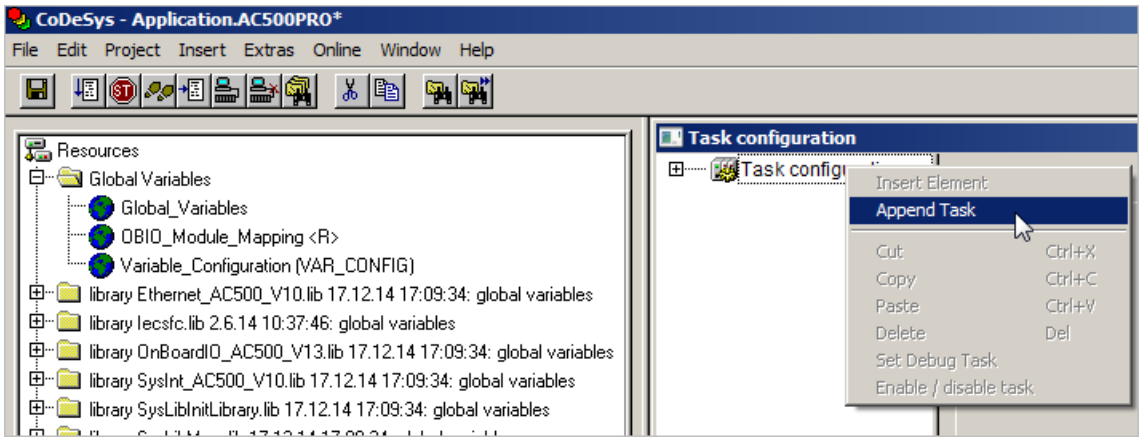


Create a Task Configuration

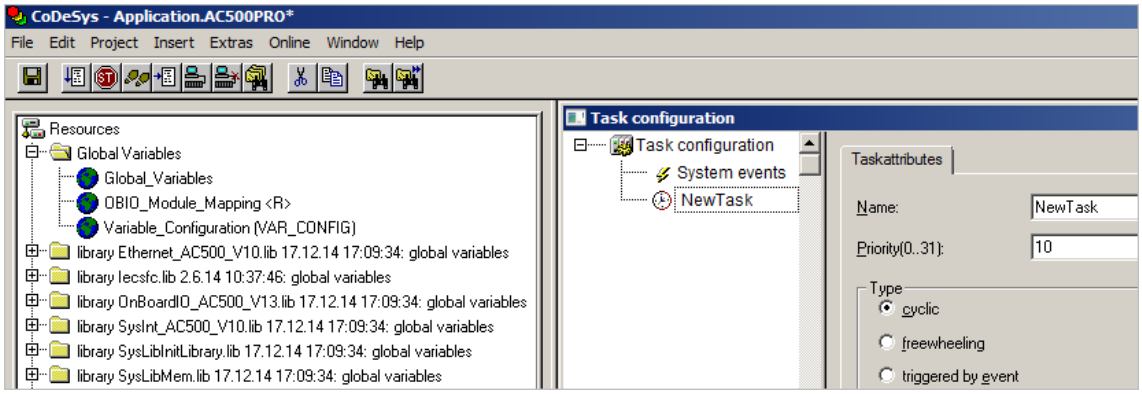
1. In the **Resources** register card, double-click **Task configuration**.
   ⇨ In work space the **Task configuration** window opens.



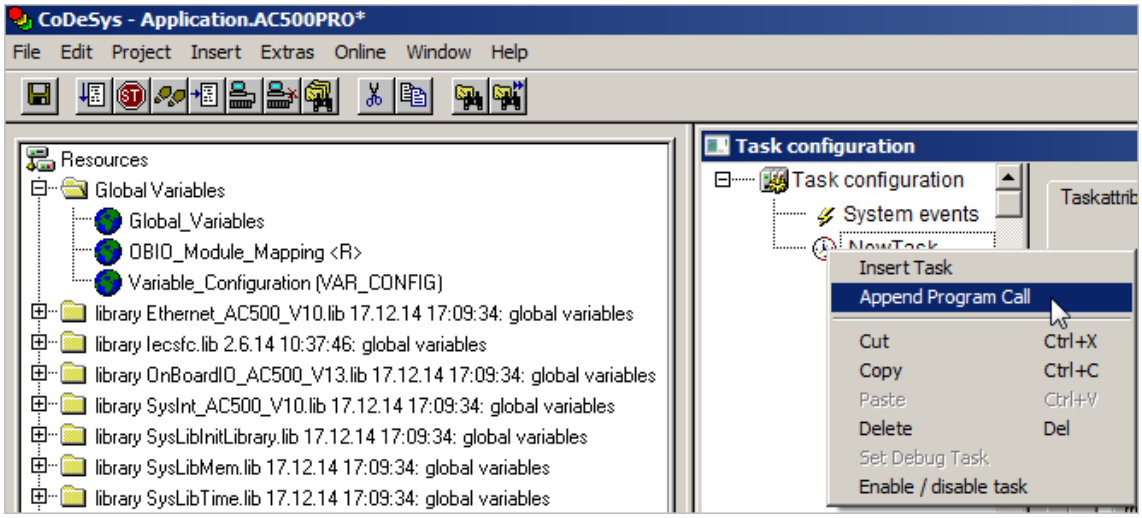
2. Right-click on **Task configuration** and select **Append Task**.



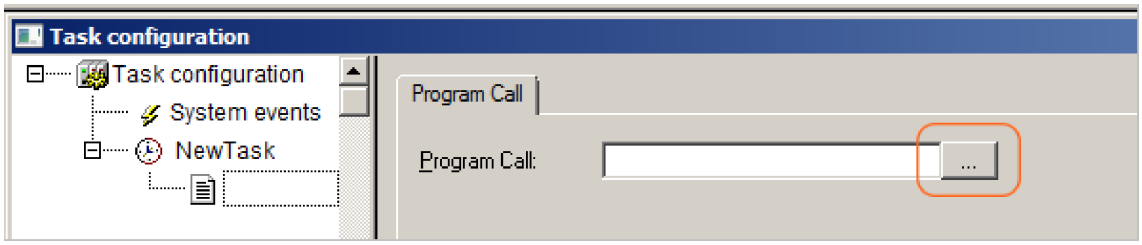
3. Under *"Properties ➔ Interval"*, enter *10*.



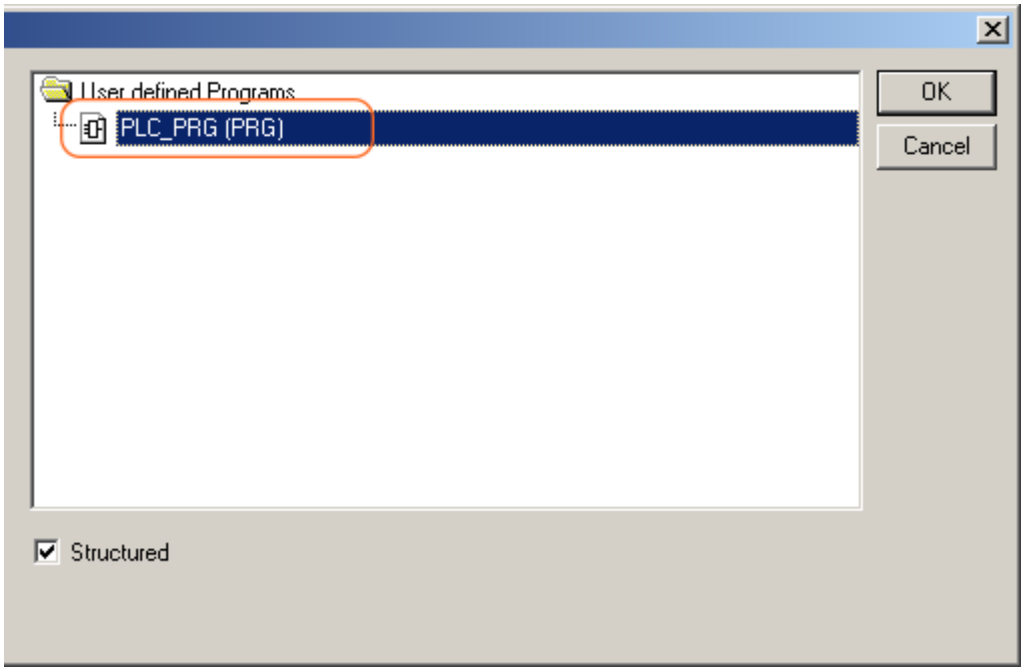
4. Right-click on the watch icon and select **Append Program Call**.

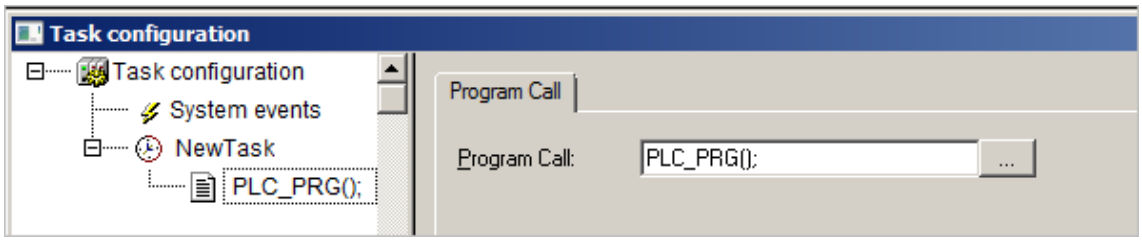


5. Click on **...** to open the input assistant.



6. Double-click on **PLC_PRG (PRG)**.





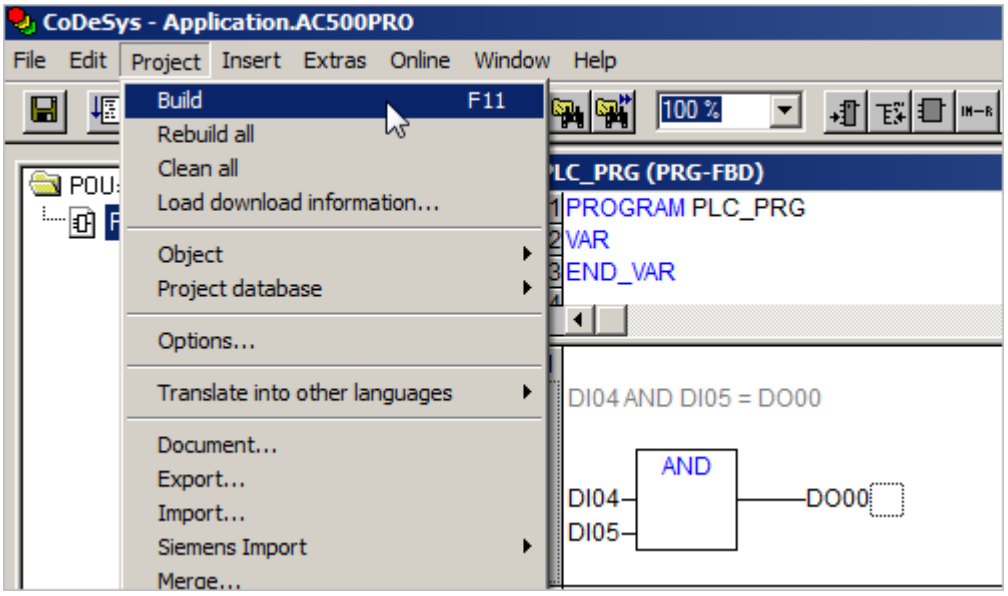
7. You finished the task configuration.

   You can close the **Task configuration** window.

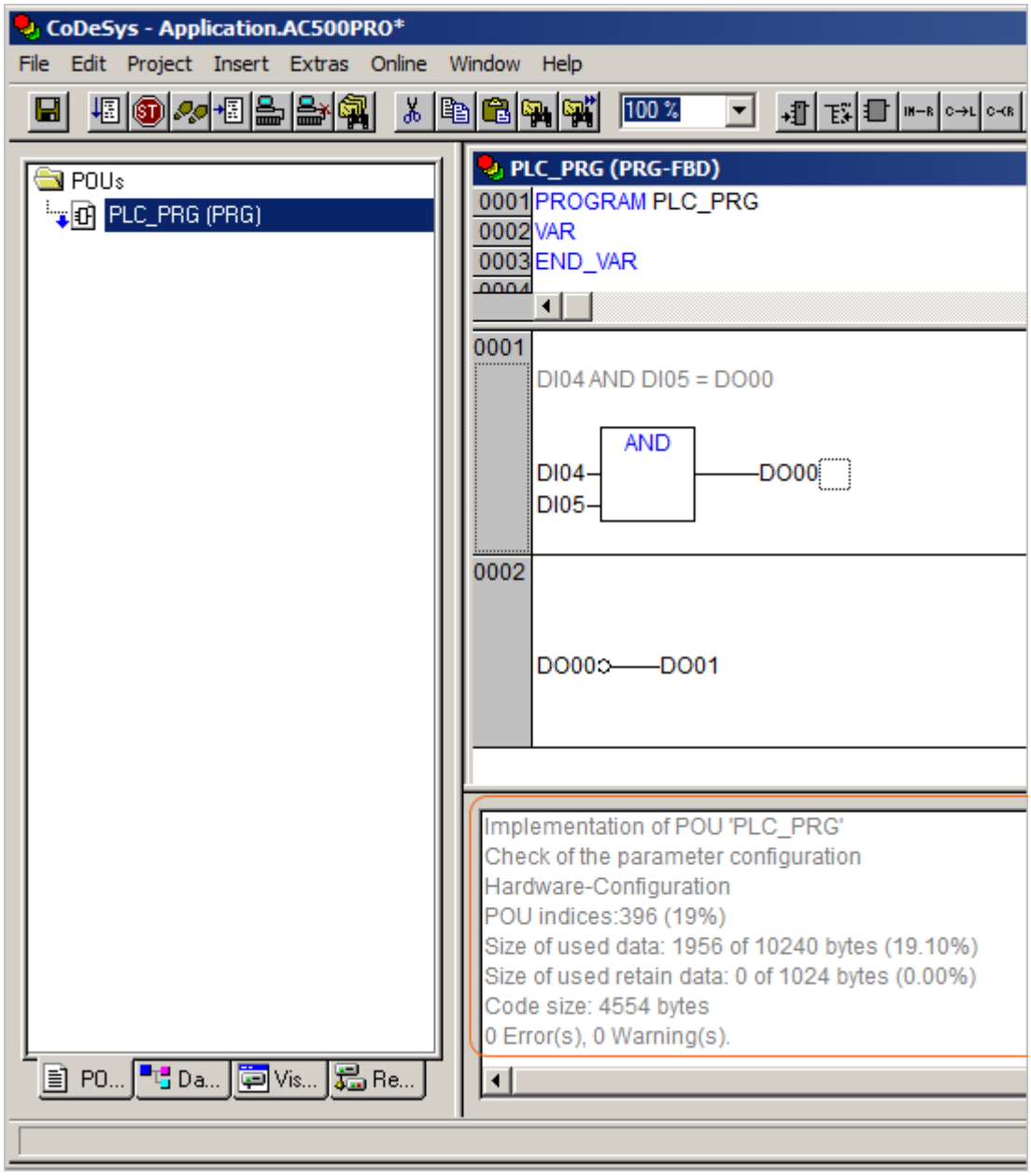
1. **Compile your Project.**

   Click on the **Save** button.

2. From the **Project** menu, click **Build**.



⇨ During compilation a message window opens. The message window shows the progress of the compilation process and any errors and warnings. Double-click on the error message to open the window that contains the error.

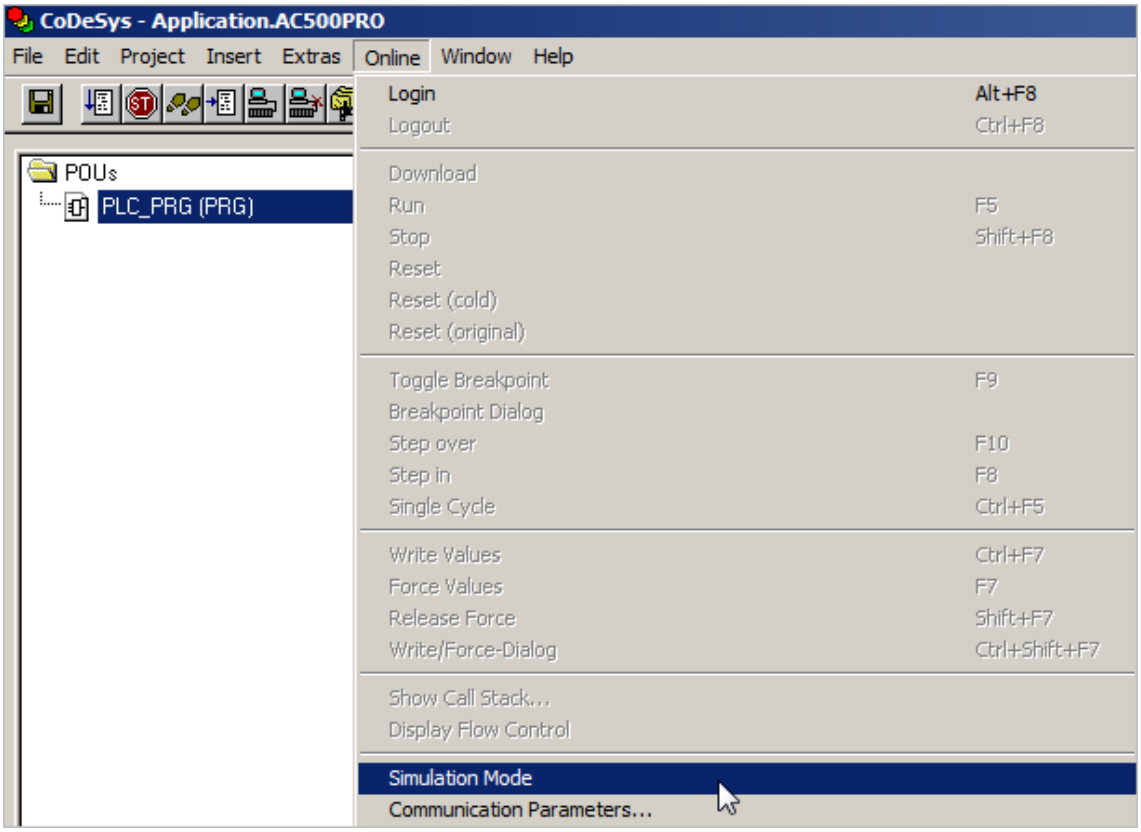


Size of used data:	Total variable that you have declared and used in your program.
Size of used retain data:	Total retain variable that you have declared and used in your program.

Testing your project without connecting the hardware

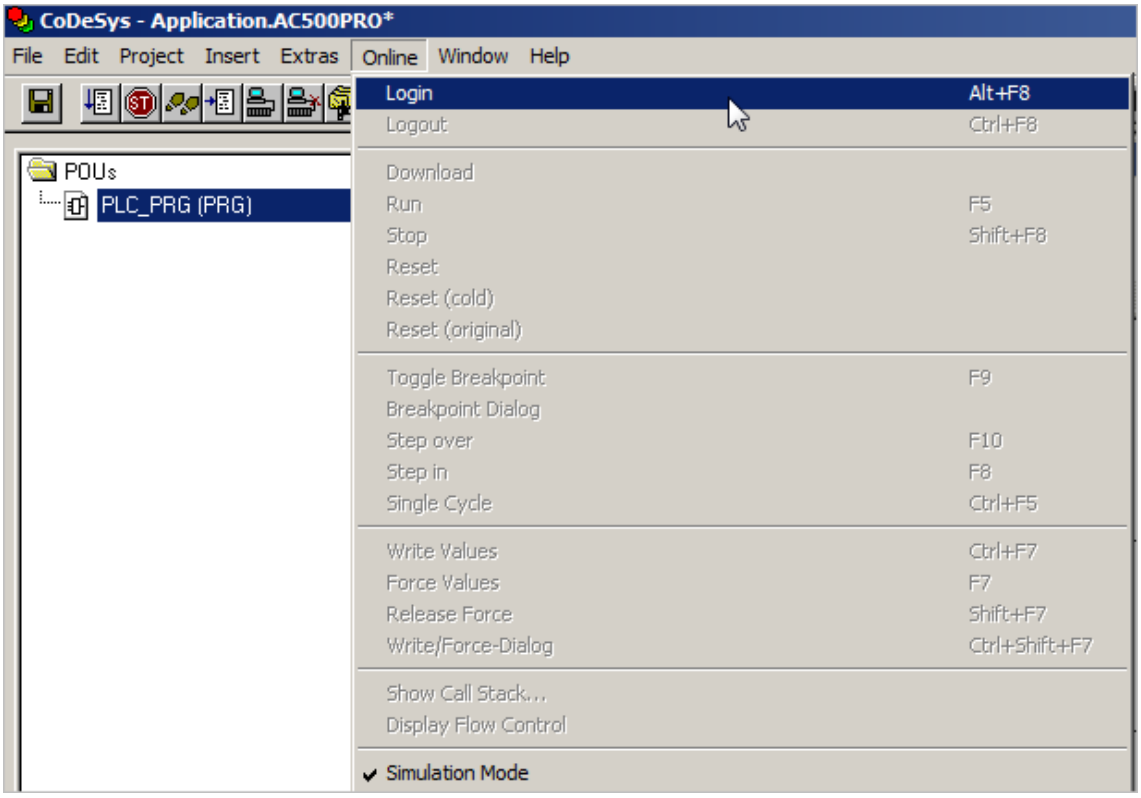
You can test your program in simulation mode. No hardware is required for simulation mode.

1. From the **Online** menu, select **Simulation Mode**.



2. From the **Online** menu, select **Login**.

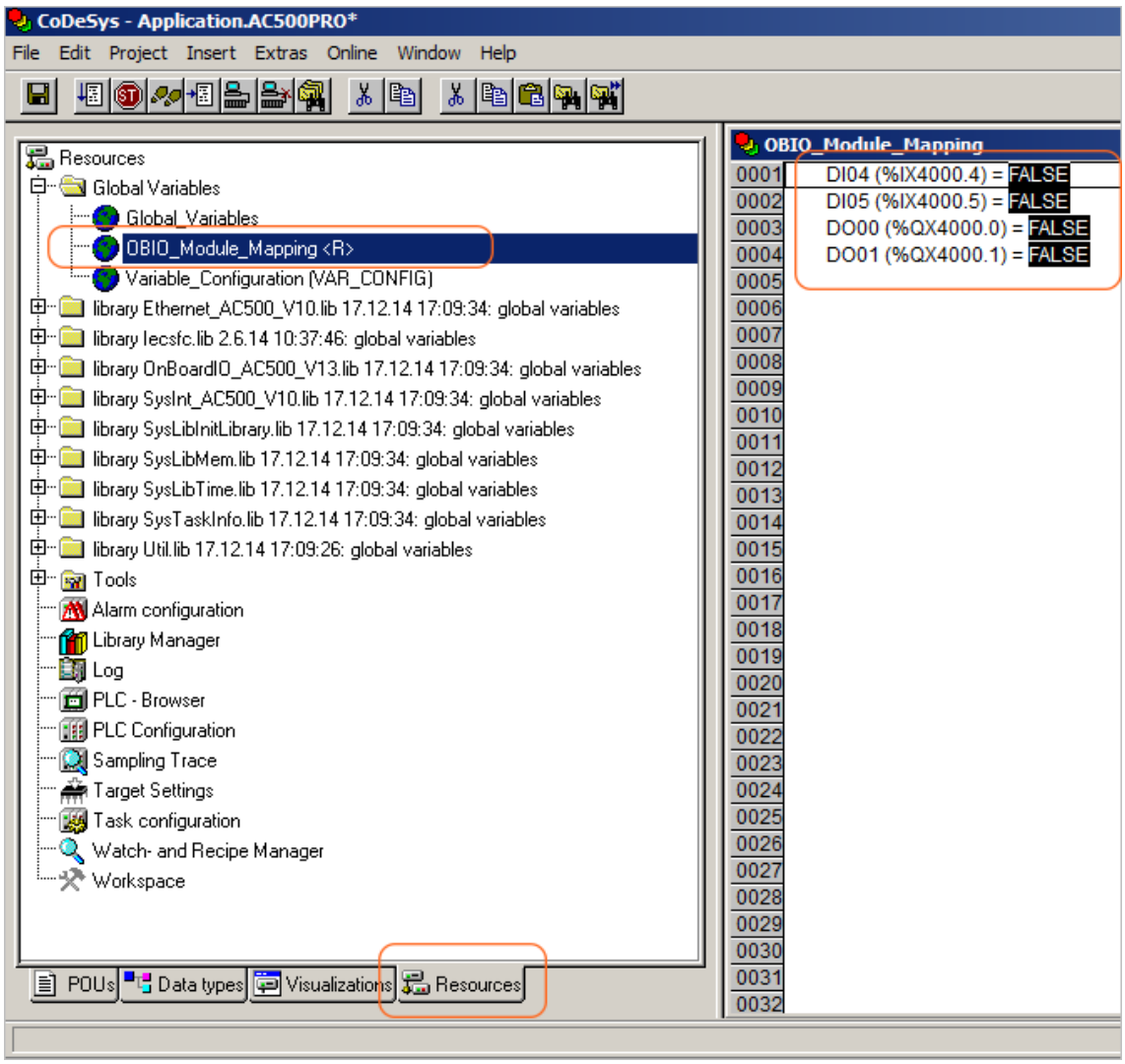
   You can see that **Simulation Mode** is checked.



   ⇨ In the status bar ONLINE and SIM are highlighted in black color.

3. From the **Resources** register card, double-click *“Global Variables ➔ OBIO_Module_Mapping <R>”*.

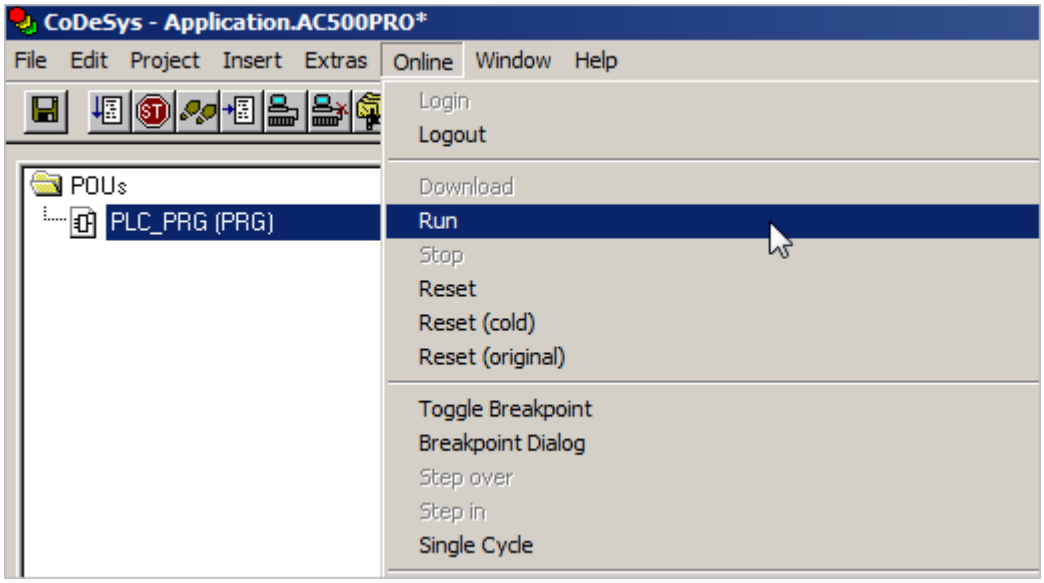
   ⇨ In work space a window opens. The window shows all declared inputs and outputs.



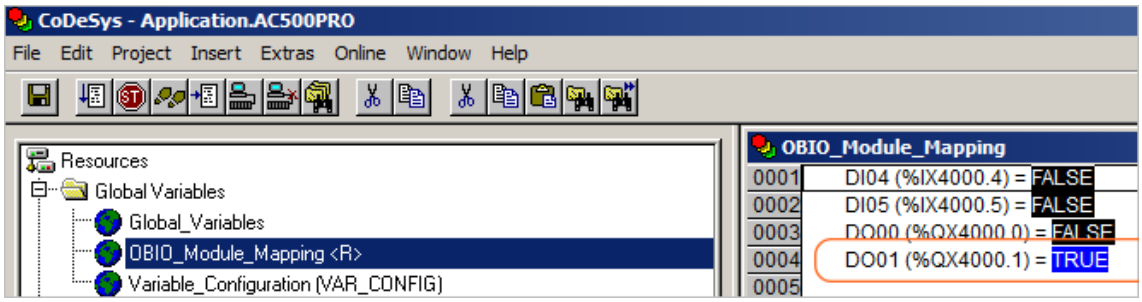
You see the status of every input and output. FALSE is highlighted on black background. TRUE is highlighted on blue background.

4. **Simulate the Starting of the CPU.**

   From the **Online** menu, select **Run**.



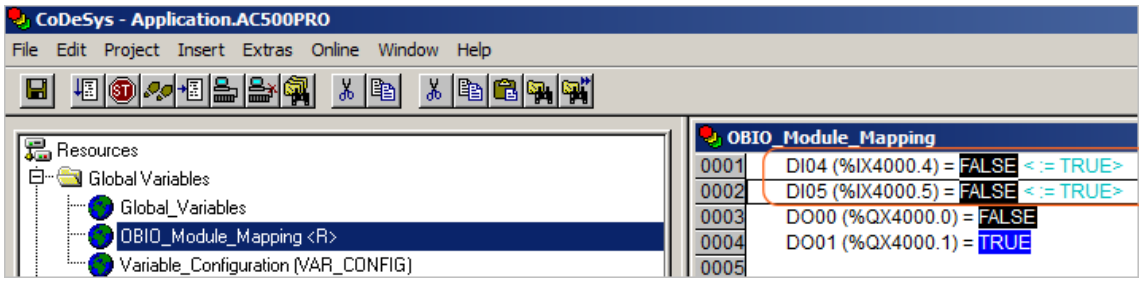
   ⇨ The status of DO01 changes from FALSE to TRUE.



5. **Change the Status of an Input.**

   Double-click FALSE of input DI04 and input DI05.

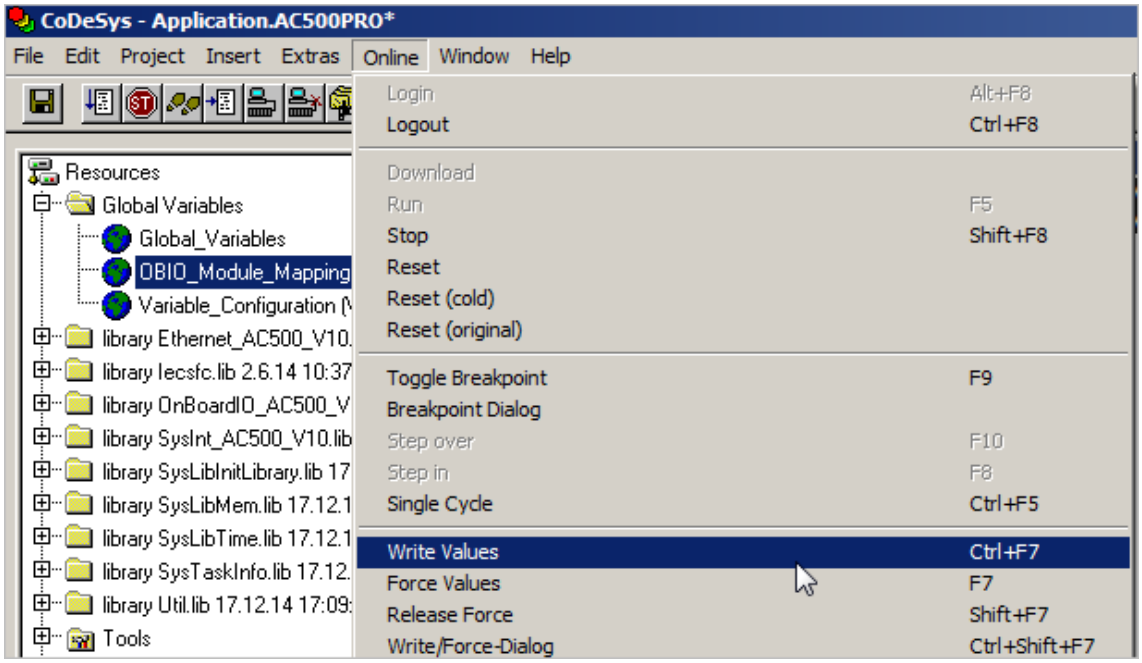
   ⇨ The change value is shown in green color behind FALSE.



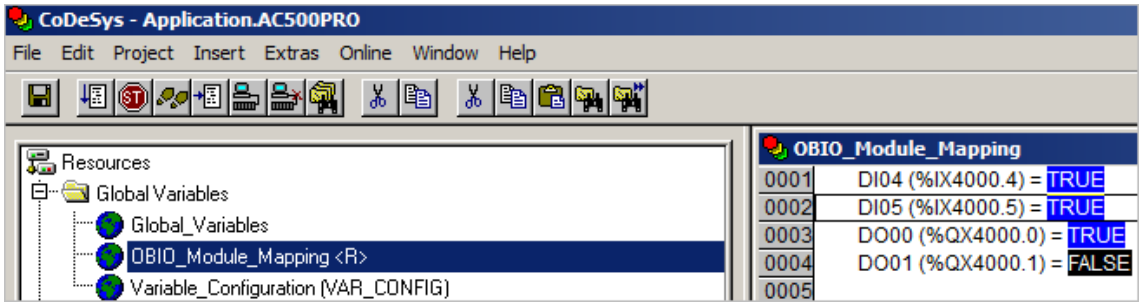
6. From the **Online** menu, select **Write Values** or **Force Values**.

   Write Values: One time effect.

   Force Values: Writes the value in every program cycle.



   ⇨ The status of the inputs and outputs is changed.



Stop simulation mode

1. From the **Online** menu, select **Logout**.
2. From the **Online** menu, select **Simulation Mode**.

   ⇨ **Simulation Mode** is unchecked.

Running your program on the AC500-eCo CPU

☑ Preconditions:

☑ The input simulator is connected to the AC500-eCo CPU ↳ *Chapter 1.6.1.10.1.1 “Connecting the input simulator” on page 3742*.

☑ The AC500-eCo CPU is connected to power supply ↳ *Chapter 1.6.1.10.1.2 “Connection of the AC500 CPU” on page 3742*.

☑ The programming cable is connected between AC500-eCo CPU and your PC ↳ *Chapter 1.6.1.10.1.3 “Connecting the programming cable” on page 3742*.

☑ The RUN/STOP switch of the AC500-eCo CPU is in RUN position. See the installation instruction which is enclosed to the AC500-eCo CPU.

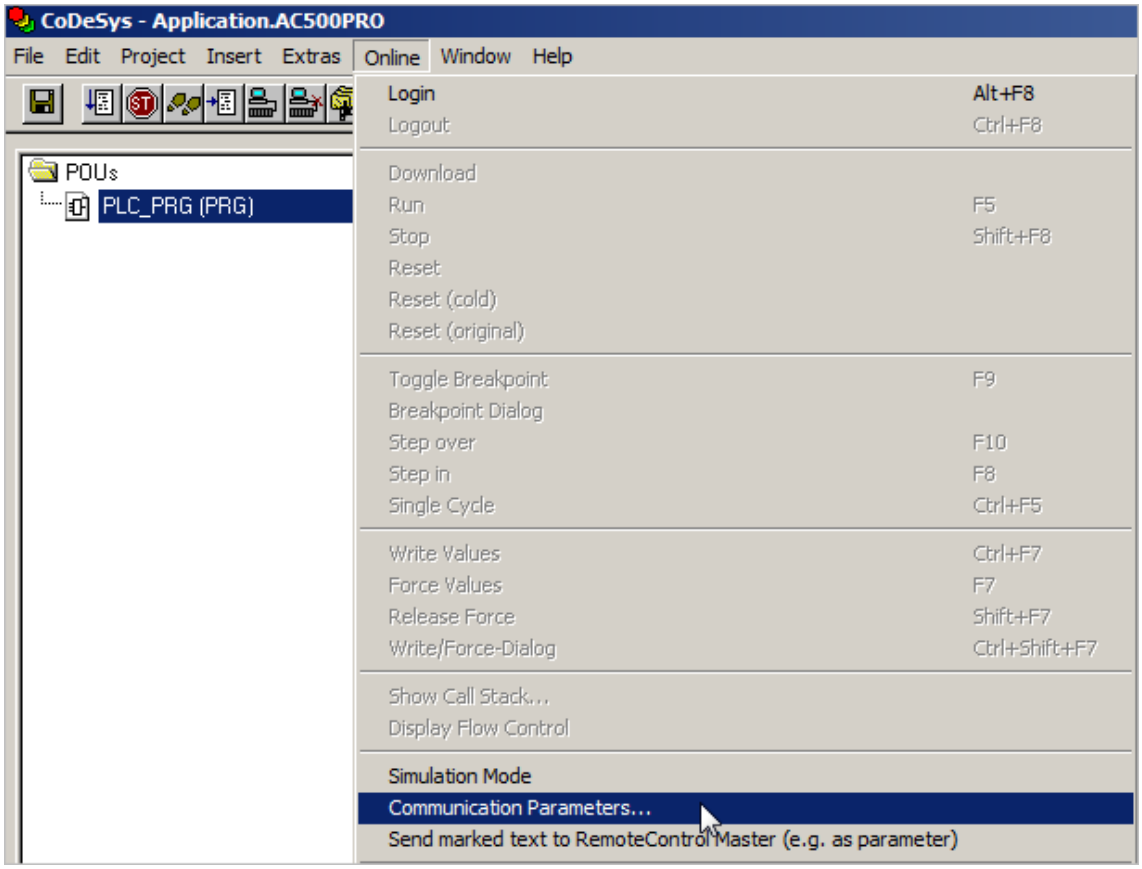
☑ The communication parameters in Windows are set ↳ *Chapter 1.6.1.10.1.4 “Set-up communication parameters in Windows” on page 3742*.

☑ The project is compiled successfully ↳ *Instruction on page 3749*.

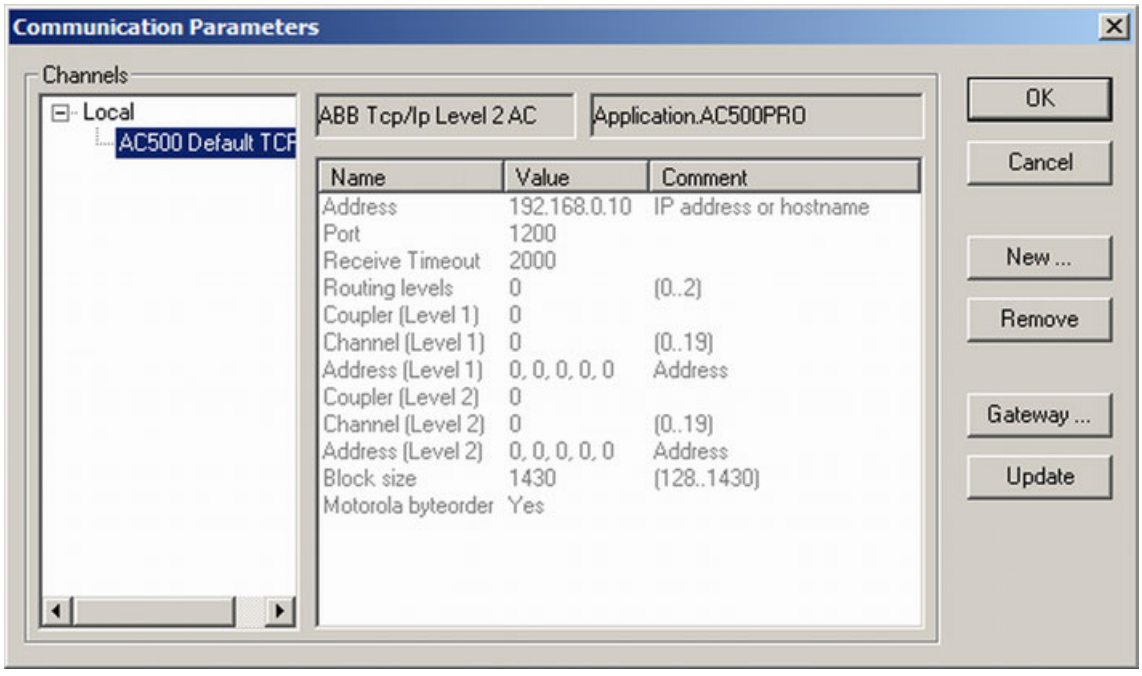
☑ The **Simulation Mode** is unchecked ↳ *“Stop simulation mode” on page 3766*.

1. Set the Communication Parameters in CODESYS

   From the **Online** menu, select **Communication Parameters**.



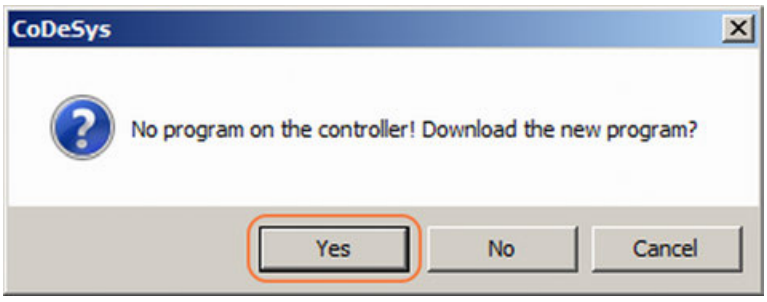
2. From **Channels**, select **AC500 Default TCP-IP**. Click **OK**.



3. **Download your Program to the AC500-eCo CPU.**

   From the **Online** menu, select **Login**.

4. The upcoming dialog informs you that no program is on the AC500-eCo CPU. You are asked if you want to download your program to the AC500-eCo CPU. Click **Yes**.



   ⇨ If the Automation Builder software version and the AC500-eCo CPU firmware version are different, a message is displayed in the status bar. This has no influence on running your example program on your AC500-eCo CPU. The example program can be executed with any firmware version of the AC500-eCo CPU.

5. **Create a Boot Project: Save RAM to ROM.**

   The application project is stored in the volatile memory of a CPU. You can enable the CPU to automatically load and execute the application project after a restart. Then the downloaded program will be stored in a non-volatile memory (flash memory). Otherwise, the program has to be reloaded manually (here: downloaded) each time the CPU is powered up.

   From the **Online** menu, select **Create boot project**.

> *Once the program is stored in the non-volatile memory, it can only be overwritten by another program or deleted with the delappl command*
> *↳ "Delete a program" on page 3768.*

6. From the **Online** menu, select **Run**.

   ⇨ In the status bar, ONLINE and RUNNING are highlighted in black color.

7. Use the toggles of the input simulator to change the status of input DI04 and DI05.

   ⇨ The LEDs of the inputs and outputs go on and off.

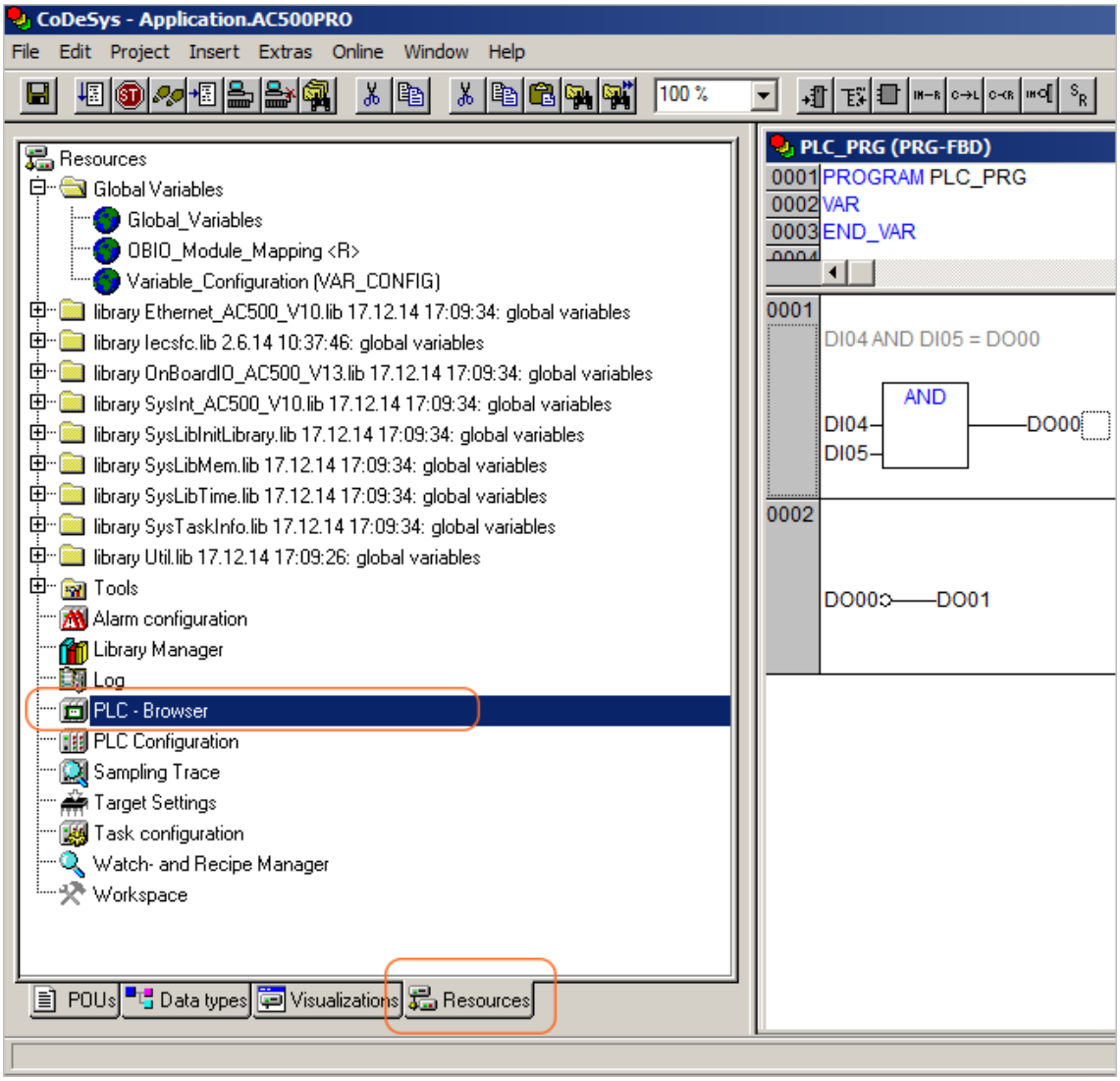
8. Stop the Connection to the AC500-eCo CPU.

   From the **Online** menu, select **Logout**.

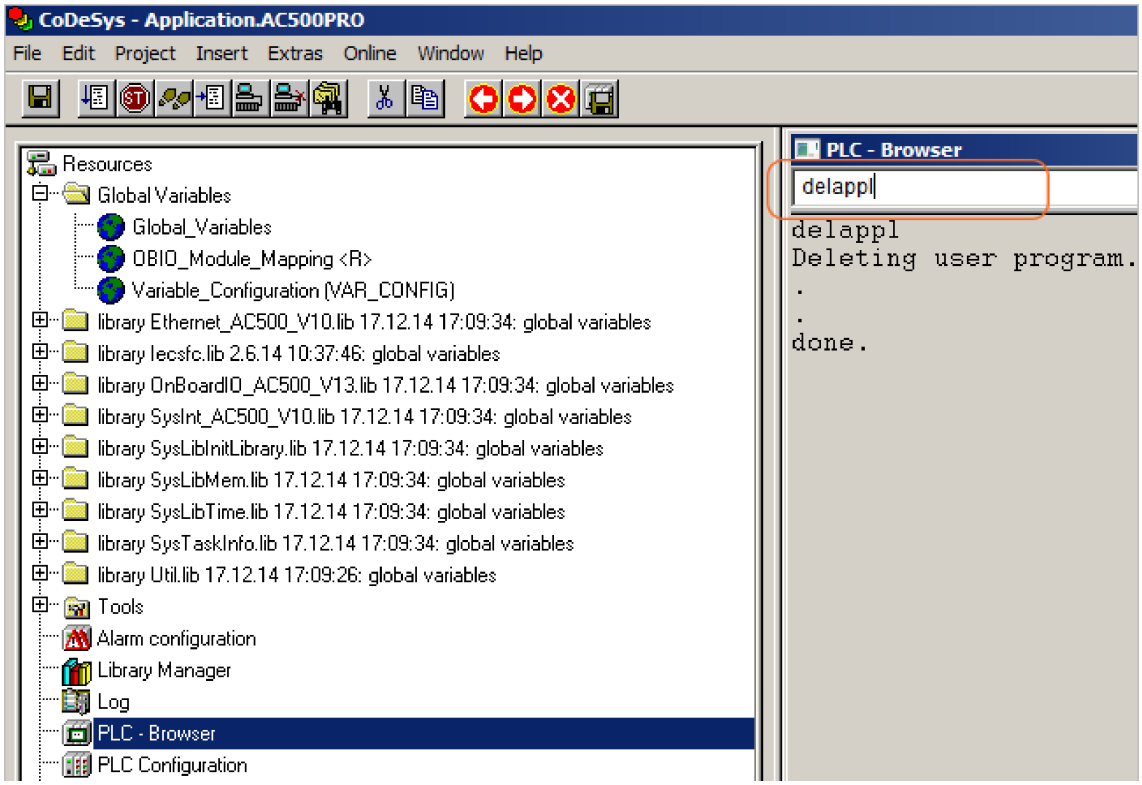
Delete a program

Once the program is stored in the non-volatile memory, it can only be overwritten by another program or deleted with the delappl command.

1. In the **Resources** register card, double-click **PLC-Browser**.



2. In the command line type *delappl*. Press ENTER.



3. The deletion is effective after power cycling.

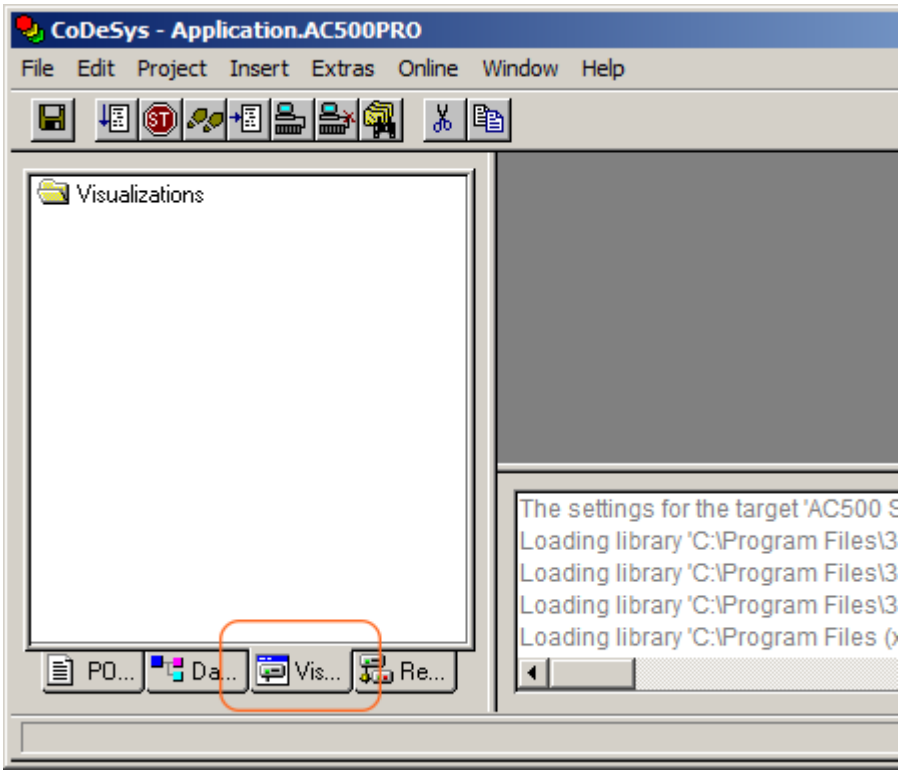
Program visualization

The visualization allows designing a graphical representation of project variables. In online mode, the graphical elements can change, for example, their color, size or position according to the actual variable status.

☑ Preconditions:

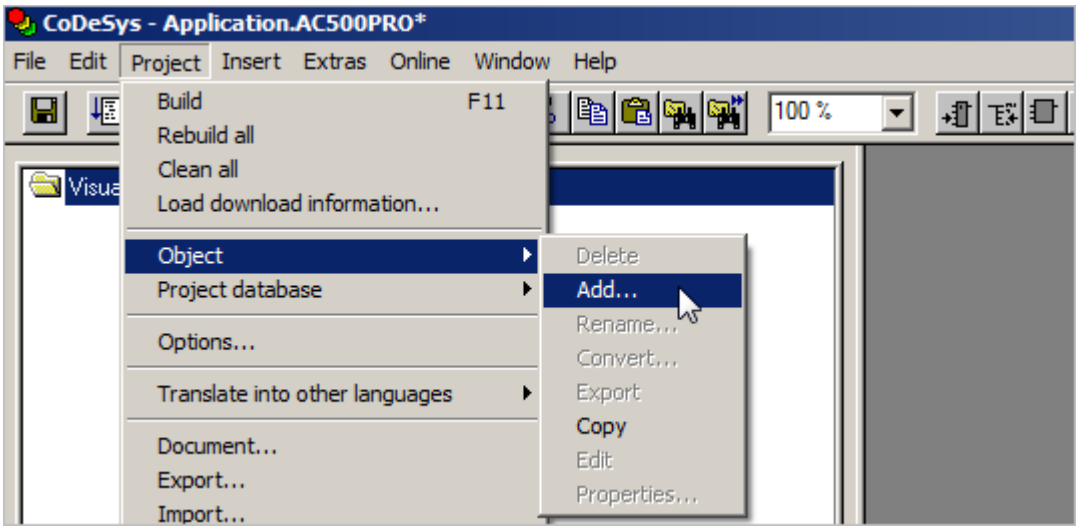
☑ The AC500-eCo CPU is disconnected. In the status bar, ONLINE is gray colored
↳ *Chapter 1.6.1.10.2.3 "Testing your project without connecting the hardware" on page 3762*.

1. Select the **Visualizations** register card.

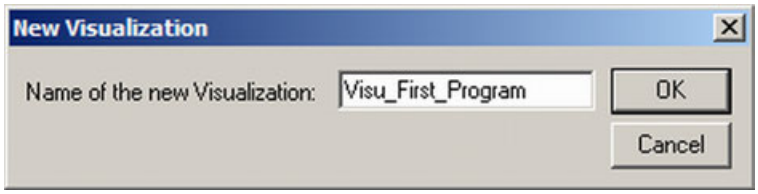


2. **Insert a new Visualization Object.**

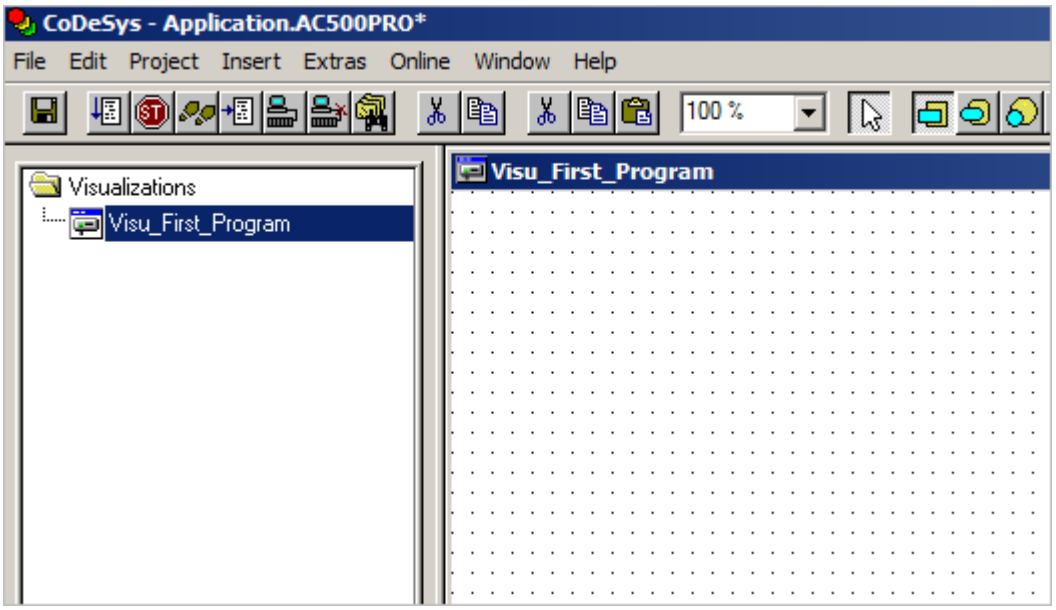
   From the **Project** menu, select *"Object ➔ Add"*.



3. Under **Name of the new Visualization**, enter *Visu_First_Program*.



⇨ The new visualization object is inserted.

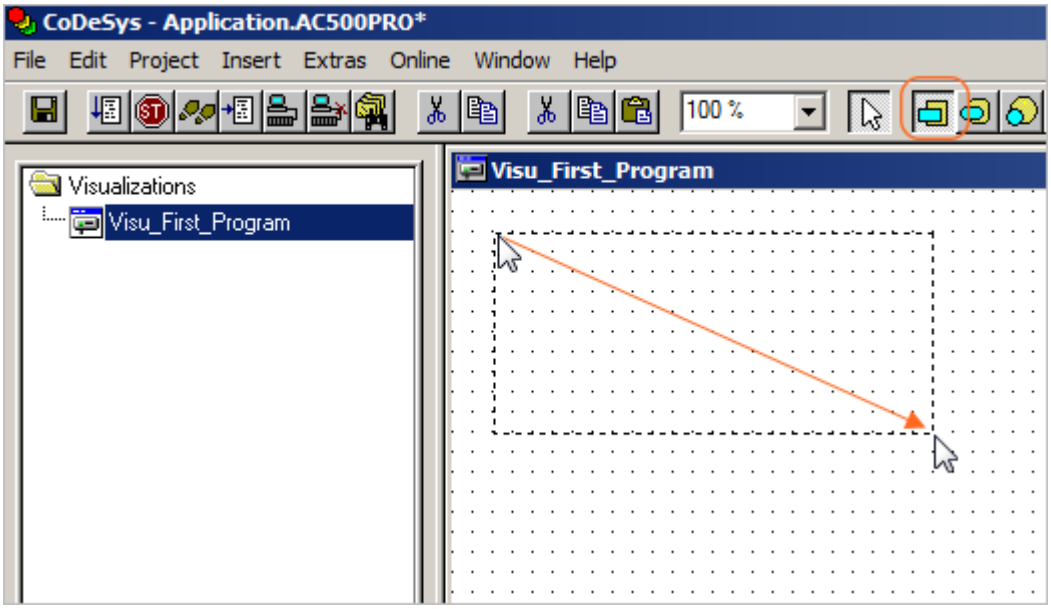


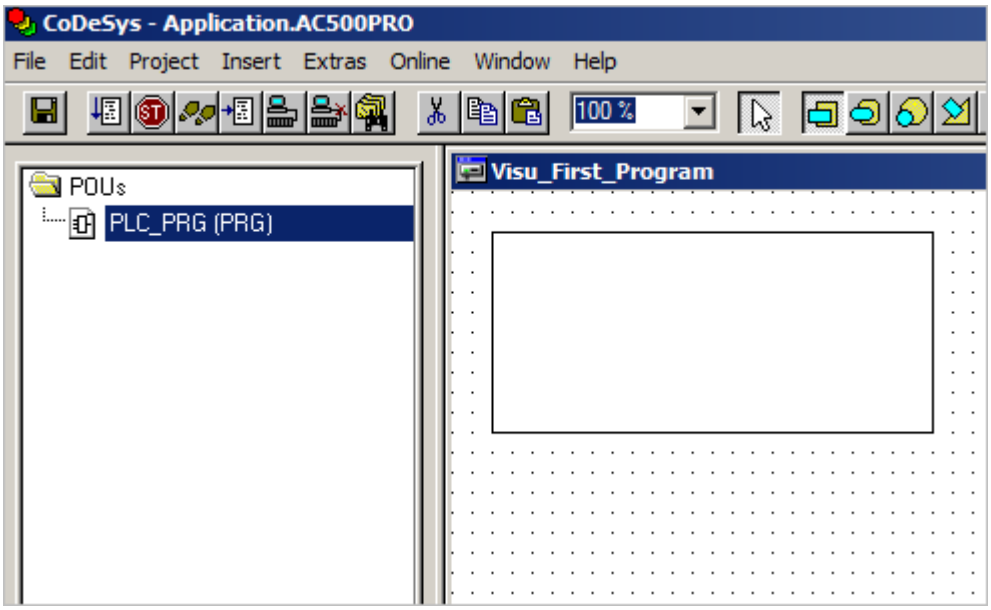
Create an element

In the toolbar, graphical elements are available for designing the inputs and outputs of your program.

1. Click on the **Rectangle** icon. Click

   and hold to draw a rectangle.



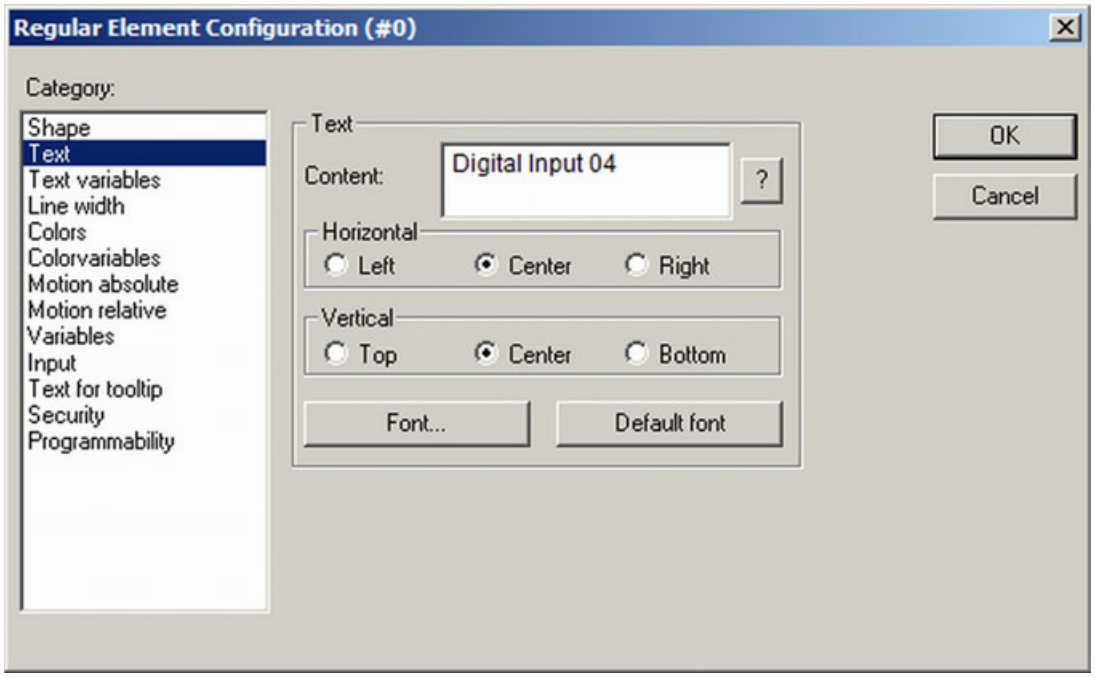


2. Double-click on the rectangle.

Insert Text

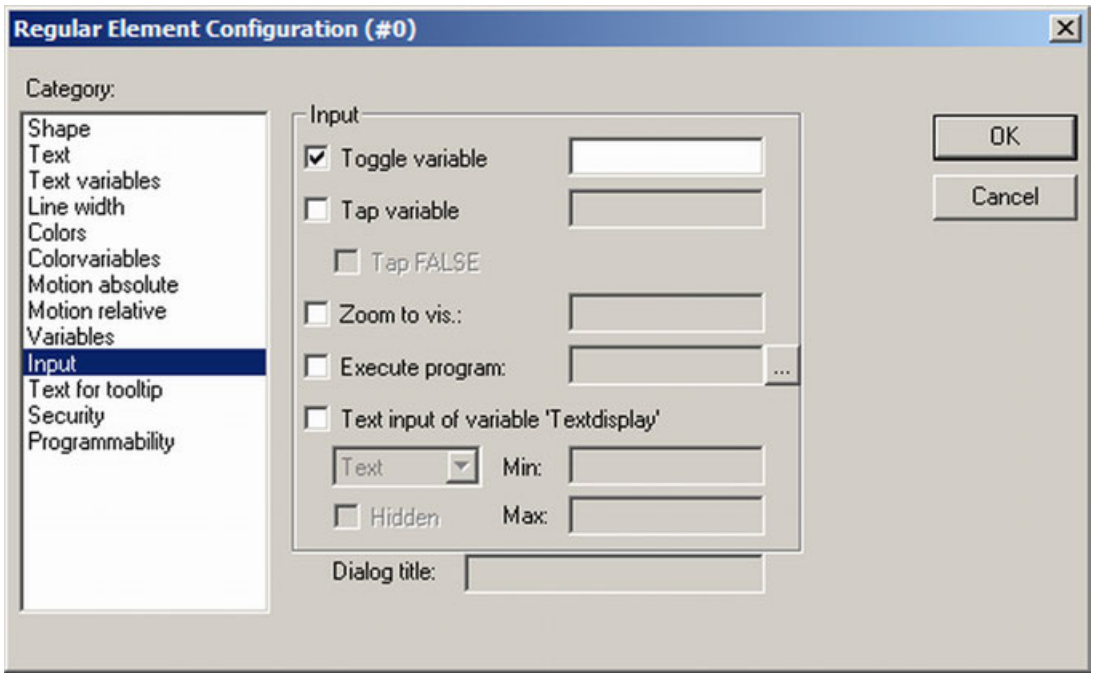
▷ Under **Category**, select **Text**. Under **Content**, enter *Digital Input 04*.

Optional, you can change the position and font of the text.



Define the Relation between the Rectangle and the Digital Input 04

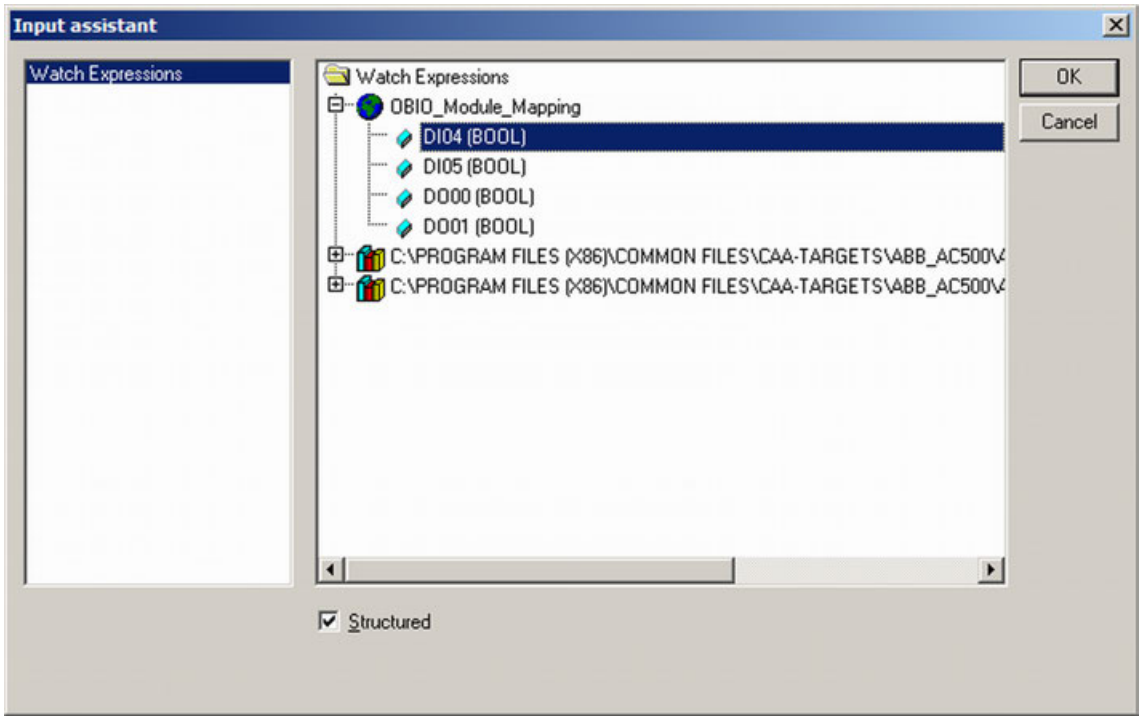
1. Under **Category**, select **Input**. Select the *Toggle variable* check box.



2. Click in the *Toggle variable*'s text box. Press F2.

   ⇨ The input assistant opens.

3. Under **OBIO_Module_Mapping**, select **DI04**.



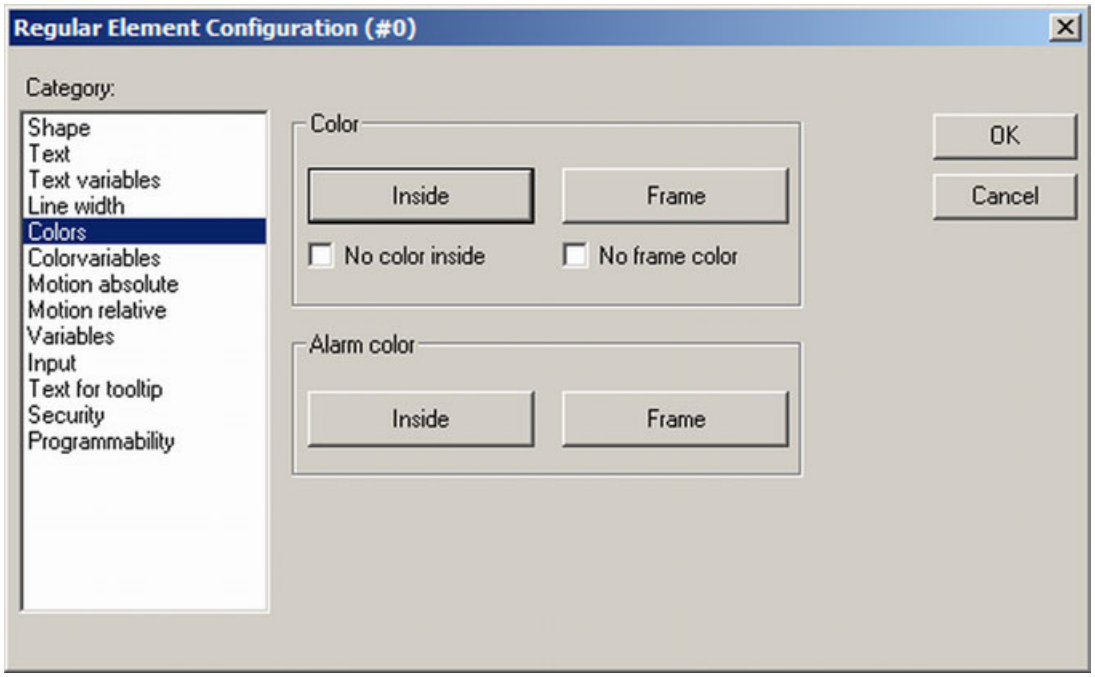
Define Color Changing depening on the Input's Status

1. Under **Category**, select **Variables**. Click in the *Change color*'s text box. Press F2.
   ⇨ The input assistant opens.
2. Under **OBIO_Module_Mapping**, select **DI04**.
   ⇨ The rectangle will change its color in online mode, depending on the status of the input.

Define Colors for each Status (TRUE and FALSE)

1. Under **Category**, select **Colors**.

   Color represents the FALSE status. **Alarm color** represents the TRUE status.



2. Under **Color**, select **Inside**. Select a color for the status FALSE, for example white. Click **OK**.

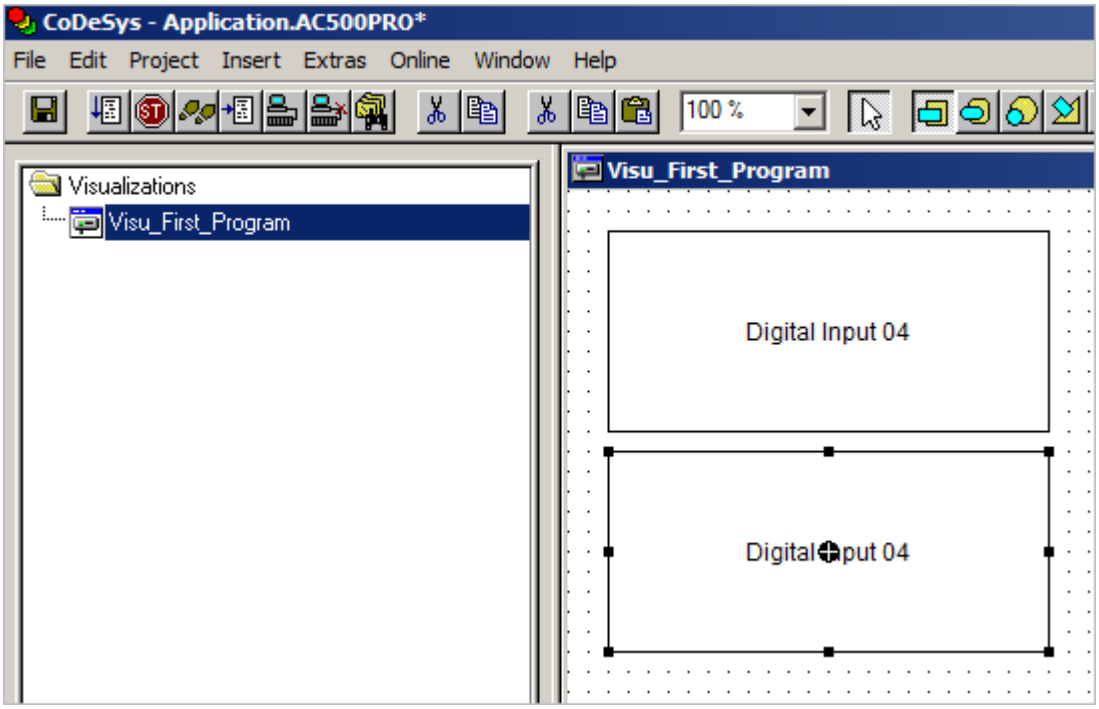
3. Under **Alarm color**, select **Inside**. Select a color for the status TRUE, for example orange. Click **OK**.

4. Click **OK**.

Create an element DI05

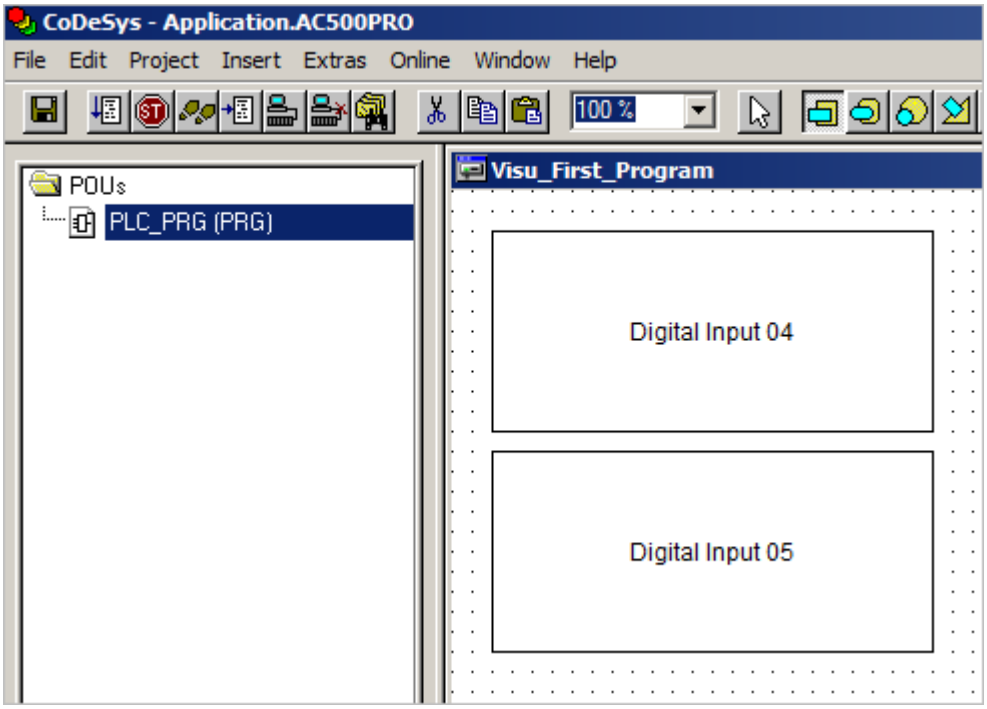
1. Copy and paste the rectangle of Digital Input 04.

   Place the duplicated rectangle next to the rectangle of Digital Input 04.



2. Double-click on the duplicated rectangle.
3. Under **Category**, select **Text**. Under **Content**, change the text into *Digital Input 05*.
4. Under **Category**, select **Input**. In the *Toggle variable*'s text box, change the text into *.DI05*.

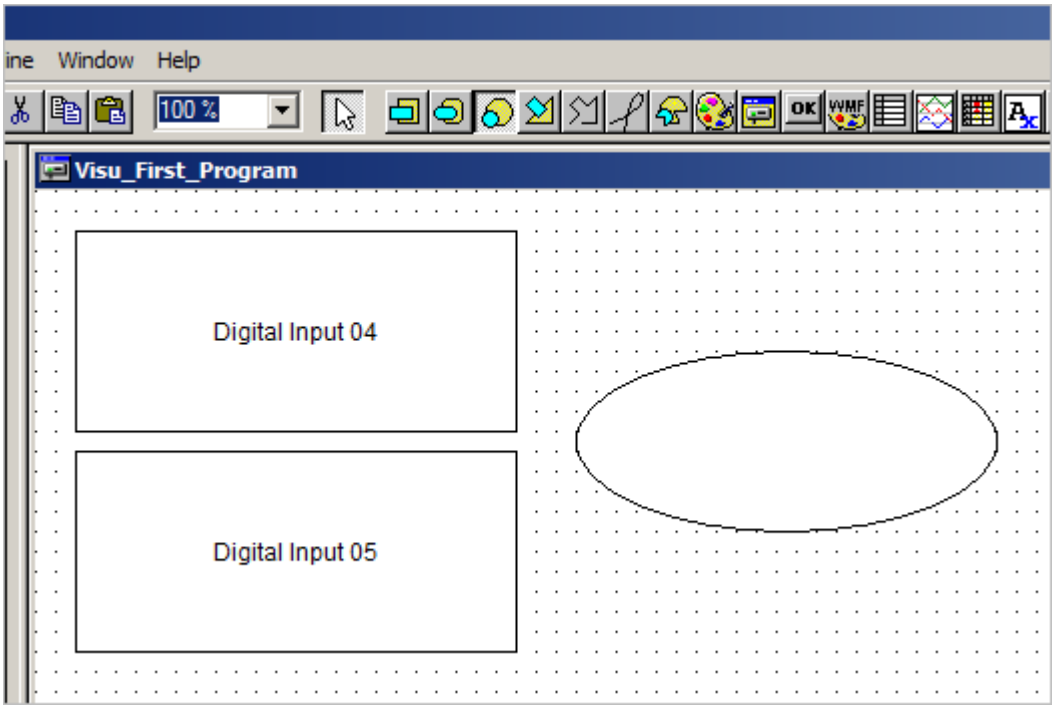
5. Under **Category**, select **Variables**. In the *Change color*'s text box, change the text into *.DI05*.



Create an element DO00

The creation of the visualization element for a digital output is very similar to a digital input.

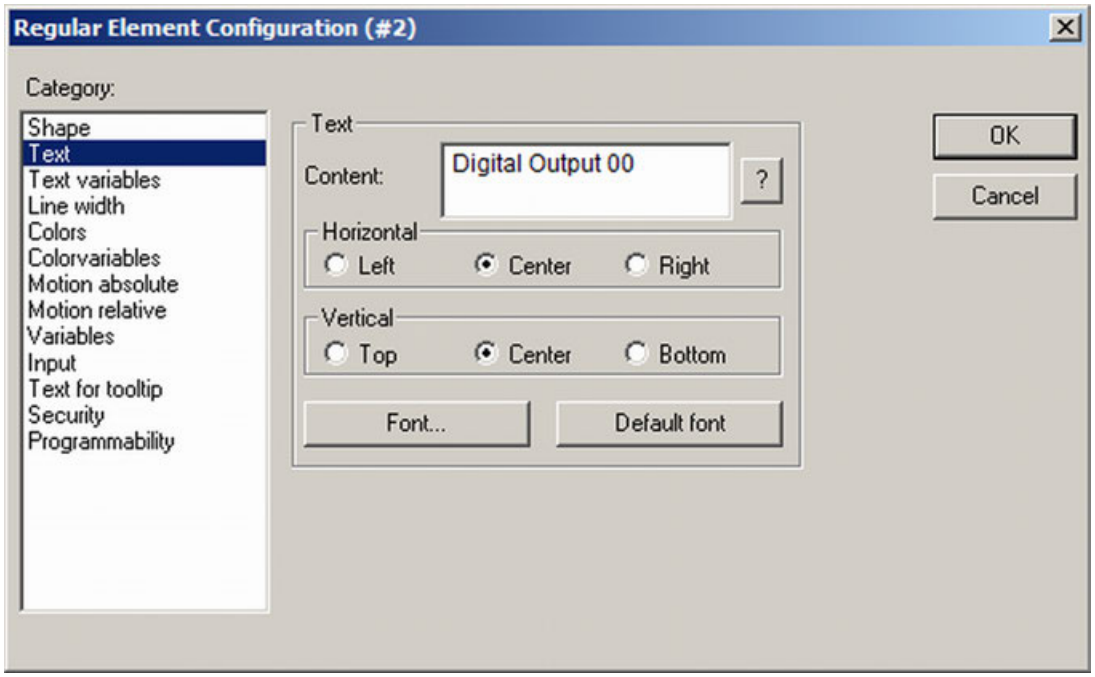
1. Click on the **Ellipse** icon. Click and hold to draw an ellipse.



2. Double-click on the ellipse.

3. Insert Text.

   Under **Category**, select **Text**. Under **Content**, enter *Digital Output 00*.



Define Color Changing depending on the Output's Status

1. Under **Category**, select **Variables**. Click in the *Change color*'s text box. Press F2.

   ⇨ The input assistant opens.

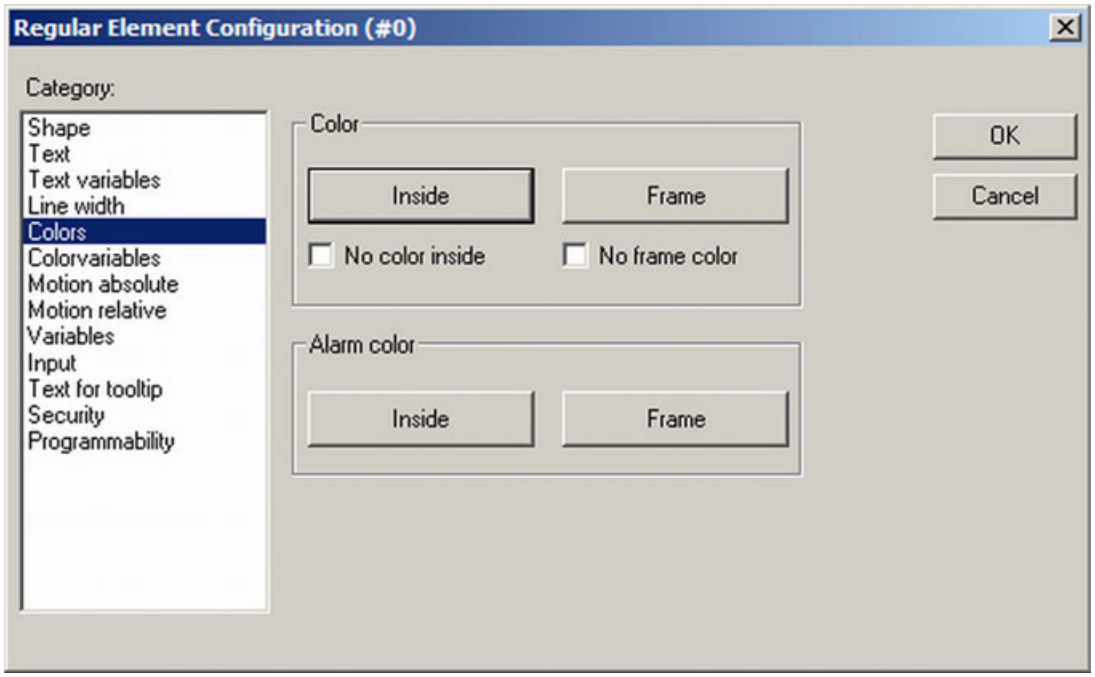
2. Under **OBIO_Module_Mapping**, select **DO00**.

   ⇨ The ellipse will change its color in online mode, depending on the status of the output.

Define Colors for each Status (TRUE and FALSE).

1. Under **Category**, select **Colors**.

   Color represents the FALSE status. **Alarm color** represents the TRUE status.



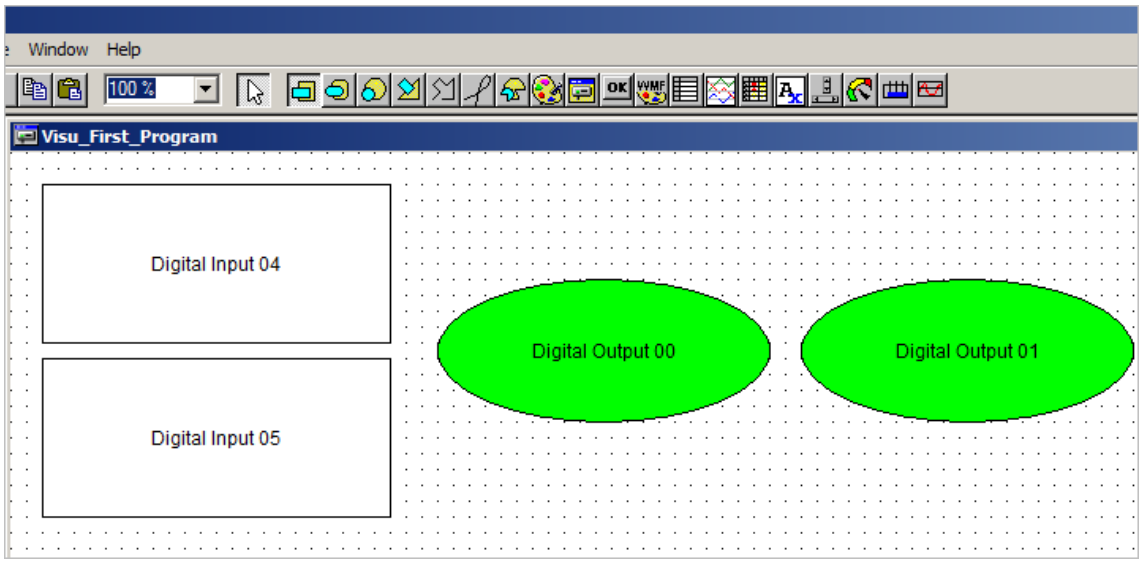
2. Under **Color**, select **Inside**. Select a color for the status FALSE, for example green.

3. Under **Alarm color**, select **Inside**. Select a color for the status TRUE, for example orange.

Create an element DO01

1. Copy and paste the ellipse of Digital Output 00.

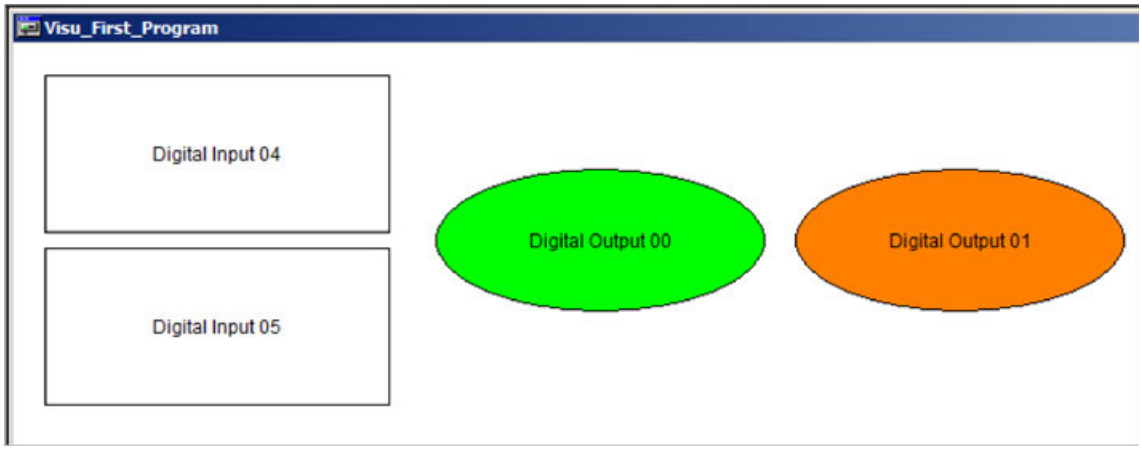
   Place the duplicated ellipse next to the ellipse of Digital Output 00.
2. Double-click on the duplicated rectangle.
3. Under **Category**, select **Text**. Under **Content**, change the text into *Digital Output 01*.
4. Under **Category**, select **Variables**. In the *Change color*'s text box, change the text into *.DO01*.



Test in simulation mode

1. Under **Online**, select **Simulation Mode**.
2. Under **Online**, select **Login**.

3. Under **Online**, select **Run**.
   ⇨ Digital output 01 changes its color because the status has been changed.



4. Click Digital Input 04 or Digital Input 05 to change the status of the inputs.

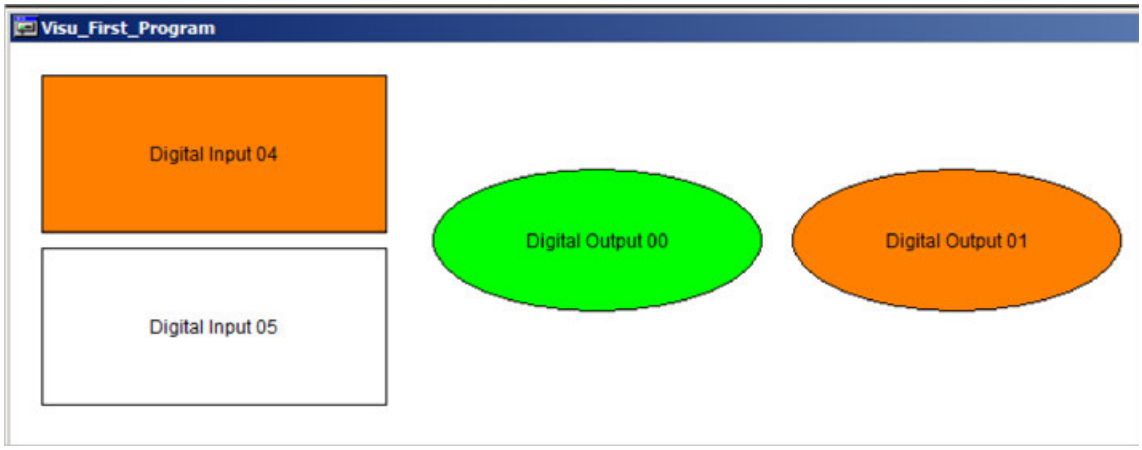


Fig. 727: Example: Digital Input 04 = TRUE, Digital Input 05 = FALSE.

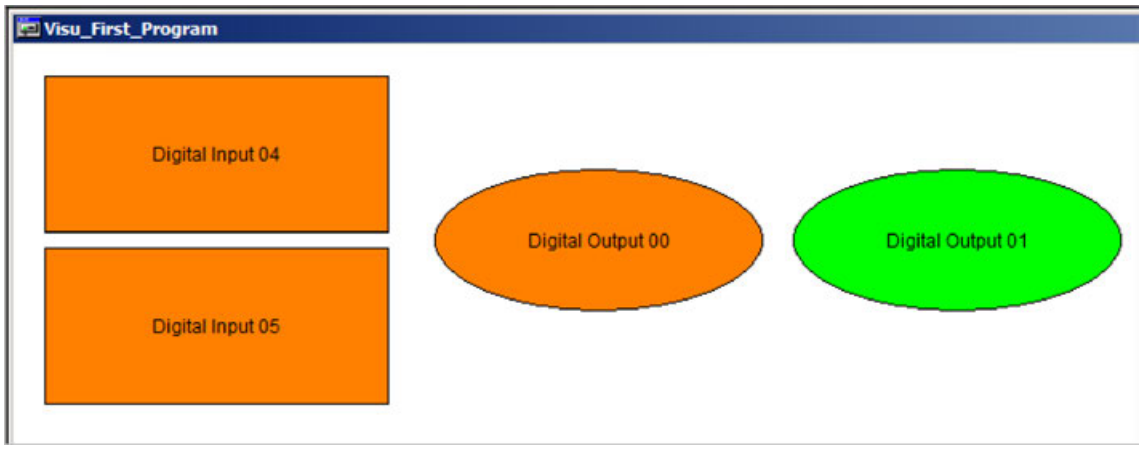


Fig. 728: Example: Digital Input 04 = TRUE, Digital Input 05 = TRUE.

5. **Stop Simulation Mode.**
   From the **Online** menu, select **Logout**.
6. From the **Online** menu, select **Simulation Mode**.
   ⇨ **Simulation Mode** is unchecked.

Visualization during connection

You can use visualization to show the status of the inputs and outputs of your AC500-eCo CPU during execution of the program on the AC500-eCo CPU.

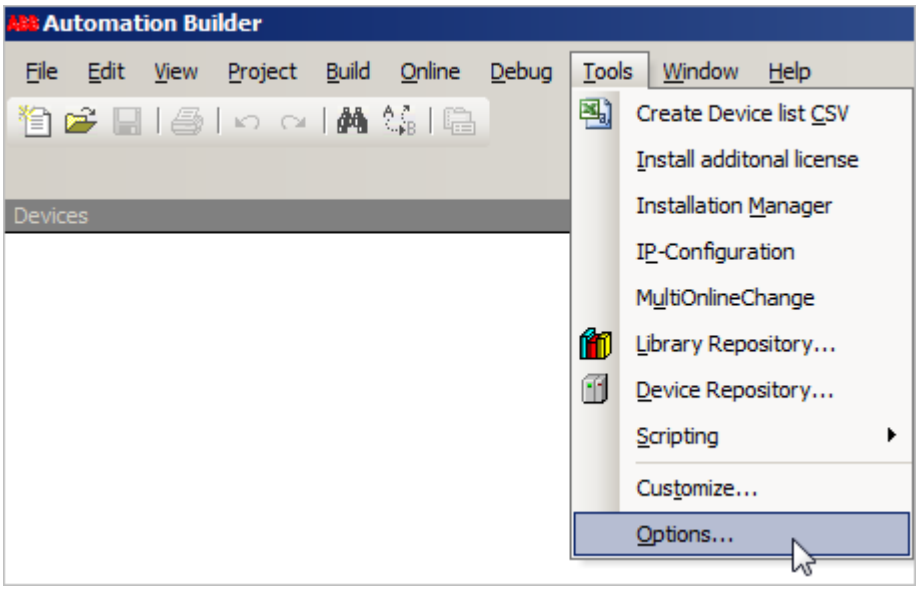
▷ Run your program on the AC500-eCo CPU *Chapter 1.6.1.10.2.4 “Running your program on the AC500-eCo CPU” on page 3766*.

⇨ The visualization shows the status of the inputs and outputs of the AC500-eCo CPU.

Changing the user interface language

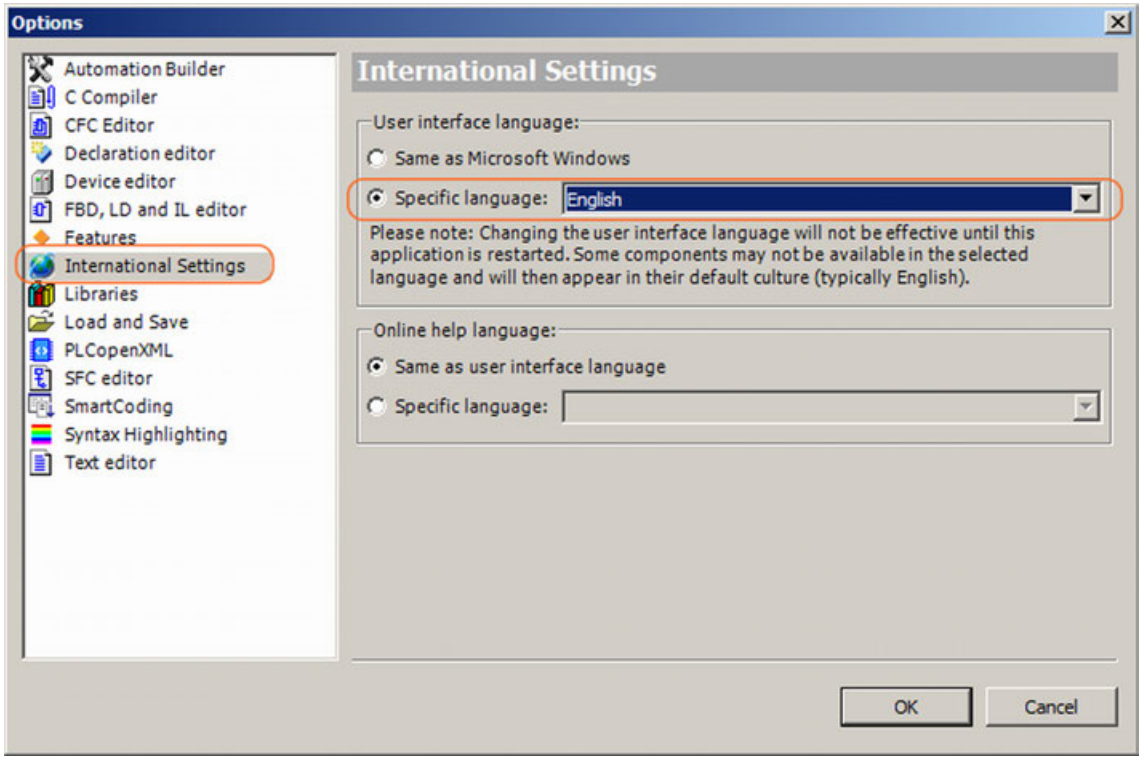
Changing the language in Automation Builder

Changing the user interface language will not be effective until you restart Automation Builder.



1. In the Automation Builder select *“Tools ➔ Options”*

⇨ This will open the Options tab

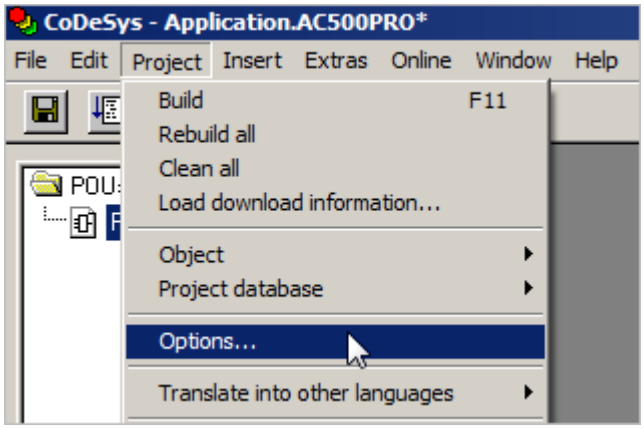


2. Select *“International Settings”*.
3. Under *“User interface language”*, select *“Specific language”*.
4. Select a language.
5. Select *“OK”*.
6. Close and restart Automation Builder.

   ⇨ The user interface language has been changed.

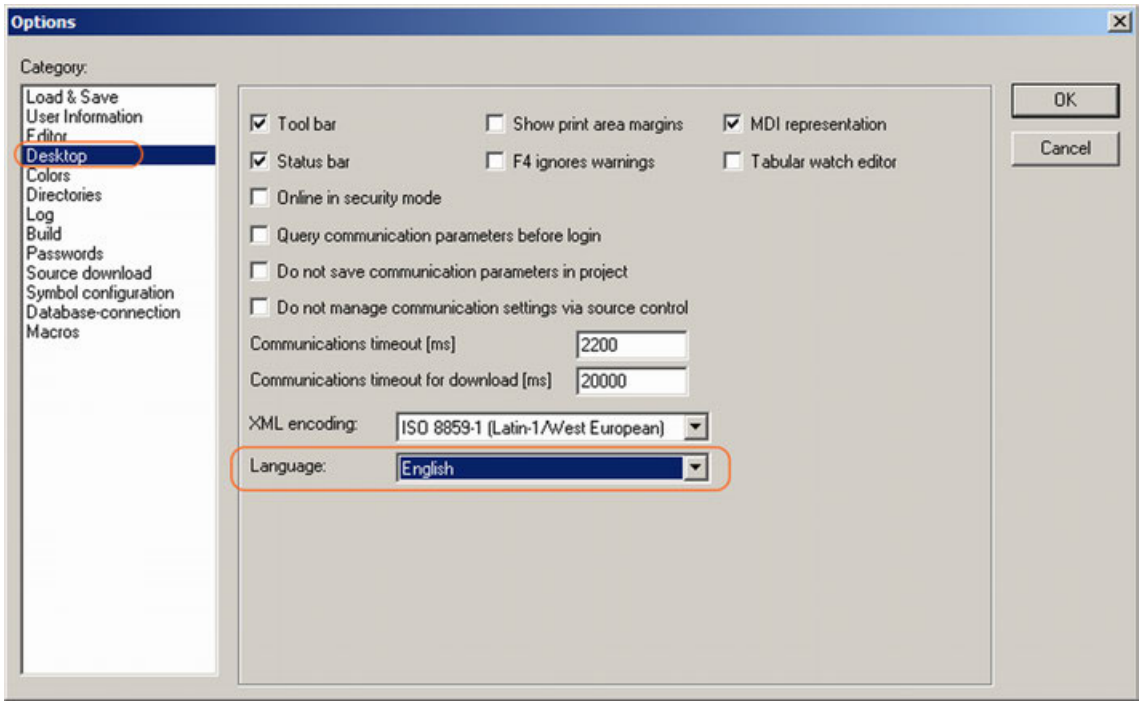
Changing the language in CODESYS

1. In CODESYS select *“Project ➔ Options”*



2. Under *“Category”*, select *“Desktop”*.

   Under *“Language”*, select a language.



3. Select *“OK”*

   ⇨ The user interface language has been changed.

1.6.1.11 Converting an AC500 V2 project to an AC500 V3 project

A project that has been configured for an AC500 V2 PLC can be converted to a project for an AC500 V3 PLC.

Essentially, the conversion is done in Automation Builder, however, some additional actions have to be executed manually. The complete procedure is described in the application example

Instructions on how to convert a V2 project to a V3 project and differences between V2 and V3.

1.6.2 Device specifications

1.6.2.1 Status LEDs, display and control elements

Depending on the device type, various operating elements provided on the front panel can be used to control the devices of the PLC system and/or to change the operating mode.

Operating elements:

- Status LEDs:
  Indicates the availability of devices/components such as communication modules, communication interface modules or function modules. Functionality and diagnosis of the status LEDs depends on the specific module and is described in the device description of the appropriate module. Possible status: on/off/blinking
- I/O LEDs:
  Displays the status of the the inputs and outputs.

- LED display:
  Available for some processor modules. It can be used for simple configurations and for reading out diagnosis information.
  ↳ *Chapter 1.6.4.1.5 “LEDs, display and function keys on the front panel” on page 5422*
  ↳ *Chapter 1.7.1.2 “Diagnosis in CPU display” on page 6365*
- Function keys and switches:
  Allows to change the current operating modes/status manually ↳ *Chapter 1.6.4.1.5.4 “Description of the function keys” on page 5426*.

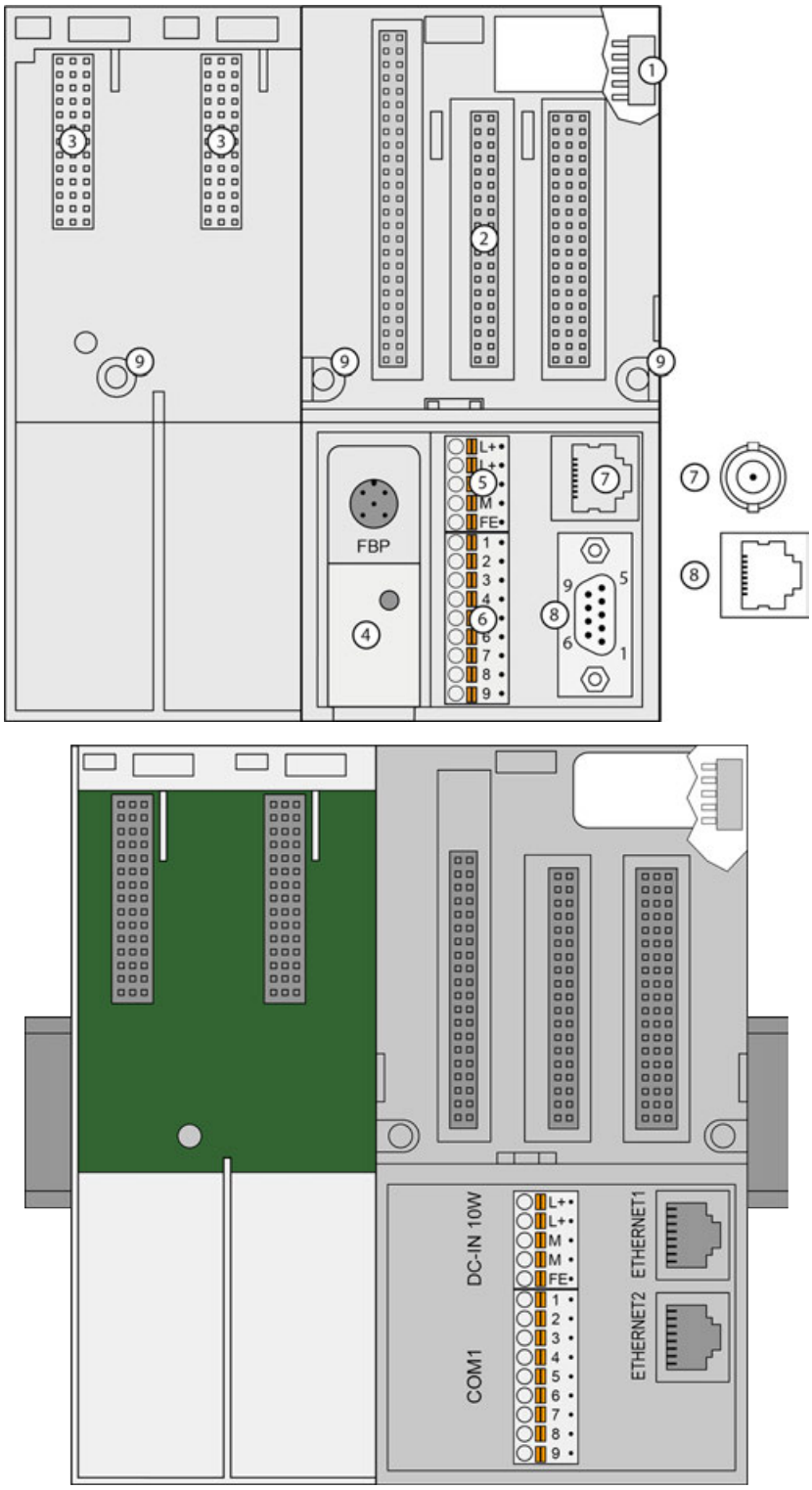
1.6.2.2 Terminal bases (AC500 standard)

For AC500-eCo processor modules and special AC500 (Standard) processor modules the terminal base cannot be removed.

1.6.2.2.1 TB51x-TB54x

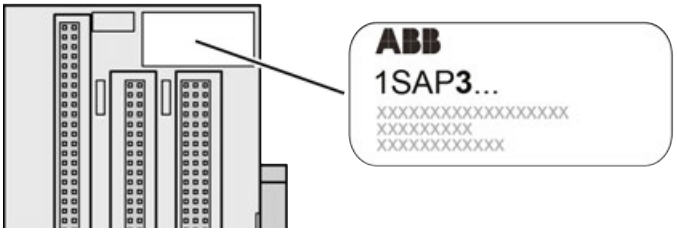
- TB511-ARCNET: 1 processor module, 1 communication module, with network interface ARCNET BNC
- TB511-ETH: 1 processor module, 1 communication module, with network interface Ethernet RJ45
- TB521-ARCNET: 1 processor module, 2 communication modules, with network interface ARCNET BNC
- TB521-ETH: 1 processor module, 2 communication modules, with network interface Ethernet RJ45
- TB523-2ETH: 1 processor module, 2 communication modules, with 2 network interface Ethernet RJ45
- TB541-ETH: 1 processor module, 4 communication modules, with network interface Ethernet RJ45
- XC version for use in extreme ambient conditions available (-ETH versions only)

The following figure shows the TB521-ETH as example.



1 I/O bus (10-pin, female) to connect the I/O terminal units
2 One available slot for the processor module
3 Slots for communication modules (TB511-xxx: 1 slot, TB521-xxx: 2 slots, TB541-xx: 4 slots)
4 Interface for FieldBusPlug, not for terminal base TB523-2ETH
5 Power supply (5-pin terminal block, removable)
6 Serial interface COM1 (9-pin terminal block, removable)
7 Network interfaces: TB5xx-ETH: Ethernet, TB5xx-ARCNET: ARCNET
8 TB5x1: Serial interface COM2 (D-sub 9, female), TB523-2ETH: second Ethernet network interface
9 Holes for screw mounting

XC version **XC** = e**X**treme **C**onditions



Extreme conditions

Terminal bases for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **3** in the Part no. 1SAP**3**... (label) identifies the XC version.*

Short description

Terminal bases TB5xx are used as sockets for AC500 CPUs and communication modules.

Up to 10 I/O terminal units for I/O expansion modules can be added to these terminal bases.

The terminal bases have slots for one processor module and for communication modules as well as terminals and interfaces for power supply, expansion and networking.

Terminal Base	TB51x	TB52x	TB54x
Slots for processor modules	1	1	1
Slots for communication modules	1	2	4

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

- Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↯ *Chapter 1.6.2.9.2.5 “TA524 - Dummy communication module” on page 5179*.
- I/O bus connectors must not be touched during operation.

Terminal Base		TB511-		TB521-		TB523-	TB541-
		ETH	ARCNET	ETH	ARCNET	2ETH	ETH
I/O bus	I/O interface for direct connection of up to 10 I/O terminal units	x	x	x	x	x	x
Power supply	removable 5-pin terminal block	x	x	x	x	x	x

Terminal Base		TB511-		TB521-		TB523-	TB541-
		ETH	ARCNET	ETH	ARCNET	2ETH	ETH
COM1	Serial interface, removable 9-pin terminal block	x	x	x	x	x	x
COM2	Serial interface, 9-pin D-sub connector (female)	x	x	x	x	-	x
Network interface [1])	Ethernet RJ45	x	-	x	-	-	x
	ARCNET BNC	-	x	-	x	-	-
	2 Ethernet RJ45	-	-	-	-	x	-
Neutral FBP interface	Neutral FBP interface (M12, 5-pin, male, fastening with screw)	x	x	x	x	-	x
CAN interface	CAN 2 A/B	-	-	-	-	-	-
[1]) Type must be equal to the type of the used processor module.							

PM57x-ETH, PM58x-ETH and PM59x-ETH with part No. 1SAPxxxxxxR0271 can only be used with terminal bases with part No. 1SAPxxxxxxR0270.

PM5xx-2ETH can only be used with TB5x3-2ETH terminal bases.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Connections

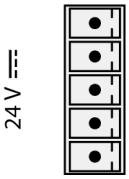
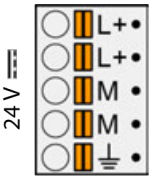
I/O Bus

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules can be added (see description for I/O bus in the system assembly chapter ↳ *Chapter 1.6.3.4.1 “Serial I/O bus” on page 5218*).

Power supply

The supply voltage of 24 V DC is connected to a removable 5-pin terminal block. L+/M exist twice. It is therefore possible to feed e.g. external sensors (up to 8 A max. with 1.5 mm^2 conductor) via these terminals.

Pin assignment

Pin Assignment		Label	Function	Description
 Terminal block removed	 Terminal block inserted	L+	+24 V DC	Positive pin of the power supply voltage
		L+	+24 V DC	Positive pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		⏚	FE	Functional earth

Faulty wiring on power supply terminals

NOTICE!

Risk of damaging the processor module and terminal base!

Exceeding the maximum voltage could lead to unrecoverable damage to the system.

The system might be destroyed.

NOTICE!

Risk of damaging the terminal base and power supply!

Short circuits might damage the terminal base and power supply.

Make sure that the four clamps L+ and M (two of each) are not wrongly connected (e. g. +/- of power supply is connected to both L+/L+ or both M/M).

NOTICE!

Risk of damaging the terminal base!

Terminal base can be damaged by connecting the power supply terminal block (L+/M) to COM1.

Make sure that the COM1 terminal block is always connected to the terminal base even if you do not use COM1 to prevent this.

NOTICE!

Risk of damaging the terminal base!

Excessive current might damage the clamp and terminal base.

Make sure that the current flowing through the removable clamps never exceeds 8 A (with 1.5 mm^2 conductor).

Serial interfaces COM1/COM2

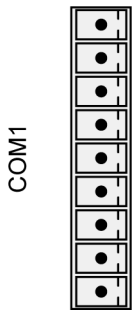
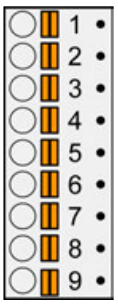
Serial interface COM1

The serial interface COM1 is connected to a removable 9-pin terminal block. It is configurable for RS-232 and RS-485 and can be used (depending on the processor module) for:

- Online access (RS-232 programming interface for Automation Builder)
- A free protocol

- Modbus RTU, client and server
- CS31 bus (RS-485), as master only ↳ *Chapter 1.6.3.6.4.8.2 “Wiring” on page 5347*

Pin assignment (RS-485 / RS-232)

		Pin	Signal	Interface	Description
 Terminal block removed	 Terminal block inserted	1	Terminator P	RS-485	Terminator P
		2	RxD/TxD-P	RS-485	Receive/Transmit, positive
		3	RxD/TxD-N	RS-485	Receive/Transmit, negative
		4	Terminator N	RS-485	Terminator N
		5	RTS	RS-232	Request to send (output)
		6	TxD	RS-232	Transmit data (output)
		7	SGND	Signal Ground	Signal Ground
		8	RxD	RS-232	Receive data (input)
		9	CTS	RS-232	Clear to send (input)

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

For further information on connection and wiring please refer to ↳ *Chapter 1.6.3.6.4.6 “Serial interface COM1 of the terminal bases” on page 5343.*

Serial interface COM2

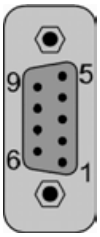
The serial interface COM2 is connected to a 9-pin D-sub connector. It is configurable for RS-232 and RS-485 and can be used (depending on the processor module) for:

- Online access (RS-232 programming interface for Automation Builder)
- A free protocol
- Modbus RTU, client and server

COM2 is not intended to establish a CS31 bus.

TB5x3-2ETH terminal bases have no COM2 D-sub.

Pin assignment

Serial Interface	Pin	Signal	Interface	Description	
	1	FE	-	Functional earth	
	2	TxD	RS-232	Transmit data	Output
	3	RxD/TxD-P	RS-485	Receive/Transmit	Positive
	4	RTS	RS-232	Request to send	Output
	5	SGND	Signal ground	0 V supply out	
	6	+5 V	-	5 V supply out	
	7	RxD	RS-232	Receive data	Input
	8	RxD/TxD-N	RS-485	Receive/Transmit	Negative
	9	CTS	RS-232	Clear to send	Input
	Shield	FE	-	Functional earth	

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

For further information on connection and wiring please refer to Chapter 1.6.3.6.4.7 “Serial interface COM2 of the terminal bases ” on page 5345.

ARCNET interface

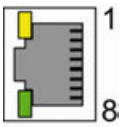
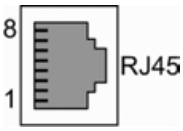
Ethernet interface

This interface is used for the connection of processor modules with onboard Ethernet e.g. AC500 CPU with an Ethernet interface.

Terminal bases TB5x3-2ETH for processor modules PM5xx-2ETH provide 2 independent Ethernet interfaces.

For structured Ethernet cabling only use cables in accordance with TIA/EIA-568-A, ISO/IEC 11801 or EN 50173.

Pin assignment

Interface	Pin	Signal	Description
 or 	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

See supported protocols and used Ethernet ports for AC500 V2products: *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.9 “Communication with Modbus TCP/IP” on page 5488*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

Neutral FieldBusPlug interface

Via a 5-pin neutral FBP interface, a processor module can be connected as a slave to a fieldbus master. The FieldBusPlug is fastened using a screw.

Pin assignment in serial mode

FieldBusPlug	Pin	Signal	Description
	1	+24 V	Standard power supply
	2	Diagnosis pin	
	3	0 V	Standard power supply
	4	Serial data	
	5	Serial data	

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Terminal bases TB5x3-2ETH for processor modules PM5xx-2ETH do not provide an neutral FBP interface.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Connection of the supply voltage 24 V DC at the terminal base of the processor module	Removable 5-pin terminal block spring type
Max. current consumption from 24 V DC	TB511: 0.35 A [1]) TB521: 0.4 A [1]) TB523: 0.4 A [1]) TB541: 0.6 A [1])
Melting integral of a fuse at 24 V DC	Min. 1 A^2s [2])
Peak inrush current from 24 V DC	55 A [2])
Slots	TB511: 1 processor module, 1 communication module
	TB521: 1 processor module, 2 communication modules
	TB523: 1 processor module, 2 communication modules
	TB541: 1 processor module, 4 communication modules
Processor module interfaces at TB5x1	I/O bus, COM1, COM2, FBP
Processor module interfaces at TB5x3	I/O bus, COM1
Processor module network interfaces	TB5x1-ETH / AC500 CPU with Ethernet interface

Parameter	Value
	TB523-2ETH / PM591-2ETH: 2x Ethernet
	TB5x1-ARCNET / AC500 CPU with ARCNET interface
Net weight (terminal base without processor module)	TB511: 175 g
	TB521: 200 g
	TB541: 250 g
Mounting position	Horizontal or vertical

[1]) Including processor modules, communication modules and communication interface modules

[2]) The inrush current and the melting integral depends on the internal power supply of the processor module and the number and type of communication modules and I/O modules connected to the I/O bus.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 111 100 R0260	TB511-ARCNET, terminal base AC500, slots: 1 processor module, 1 communication module, ARCNET COAX connector	Active
1SAP 111 100 R0270	TB511-ETH, terminal base AC500, slots: 1 processor module, 1 communication module, Ethernet RJ45 connector	Active
1SAP 311 100 R0270	TB511-ETH-XC, terminal base AC500, slots: 1 processor module, 1 communication module, Ethernet RJ45 connector, XC version	Active
1SAP 112 100 R0260	TB521-ARCNET, terminal base AC500, slots: 1 processor module, 2 communication modules, ARCNET COAX connector	Active
1SAP 112 100 R0270	TB521-ETH, terminal base AC500, slots: 1 processor module, 2 communication modules, with network interface Ethernet RJ45	Active
1SAP 312 100 R0270	TB521-ETH-XC, terminal base AC500, slots: 1 processor module, 2 communication modules, with network interface Ethernet RJ45, XC version	Active
1SAP 112 300 R0277	TB523-2ETH, teminal base AC500, slots: 1 processor module, 2 communication modules, with 2 network interfaces Ethernet RJ45	Active

Part no.	Description	Product life cycle phase *)
1SAP 114 100 R0270	TB541-ETH, slots: 1 processor module, 4 communication modules, with network interface Ethernet RJ45	Active
1SAP 314 100 R0270	TB541-ETH-XC, slots: 1 processor module, 4 communication modules, with network interface Ethernet RJ45, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

Processor modules PM57x-ETH(-XC), PM58x-ETH(-XC) and PM59x-ETH(-XC) with ordering No. 1SAPxxxxxxR0271 can only be used with terminal bases TB5x1-ETH(-XC) with ordering No. 1SAPxxxxxxR0270.

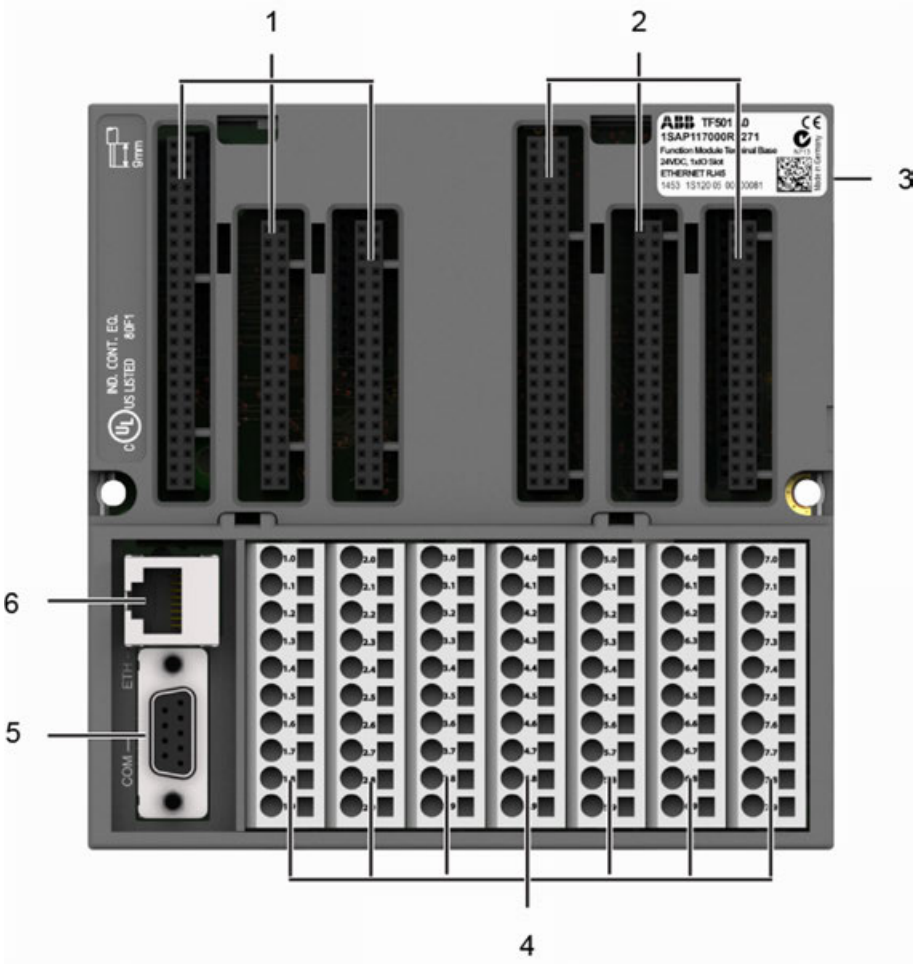
Processor module PM591-2ETH can only be used with TB523-2ETH.

Table 261: Accessories

Part no.	Description
1SAP 180 200 R0001	TK501, programming cable D-sub / D-sub, length: 5 m
1SAP 180 200 R0101	TK502, programming cable terminal block / D-sub, length: 5 m
1TNE 968 901 R1100	TK503, COM1 USB programming cable / D-sub (RS-485), length 3 m
1SAP 180 800 R0001	TA526, wall mounting accessory

1.6.2.2.2 TF501-CMS and TF521-CMS - Function module terminal bases

- For function module FM502-CMS
- TF501-CMS: 1 processor module, 1 FM502-CMS, with network interface Ethernet RJ45
- TF521-CMS: 1 processor module, 1 FM502-CMS, 2 communication modules, with network interfaces Ethernet RJ45
- XC version for use in extreme ambient conditions available



1 Slots for PM592-ETH
2 Slots for FM502-CMS
3 I/O bus to galvanically connect the terminal units
4 Terminal blocks for analog/digital inputs/outputs
5 Serial interface COM1
6 Ethernet network interface

> *All I/O channels (digital and analog) are protected against reverse polarity, reverse supply and continuous overvoltage up to 30 V DC.*

The TF5x1-CMS are used as terminal bases for FM502-CMS, PM592-ETH and communication modules ↳ *Chapter 1.6.2.7.2.2 “FM502-CMS - Analog measurements” on page 4658*
↳ *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*.

Short description

The function module terminal bases have slots for one FM502-CMS, one processor module and for communication modules as well as terminals and interfaces for power supply, expansion, networking and IO. The number of slots differs depending on the type of terminal base.

Table 262: Number of slots

Slot	TF501-CMS	TF521-CMS
Slots for processor modules	1	1
Slots for function modules	1	1
Slots for communication modules	0	2

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

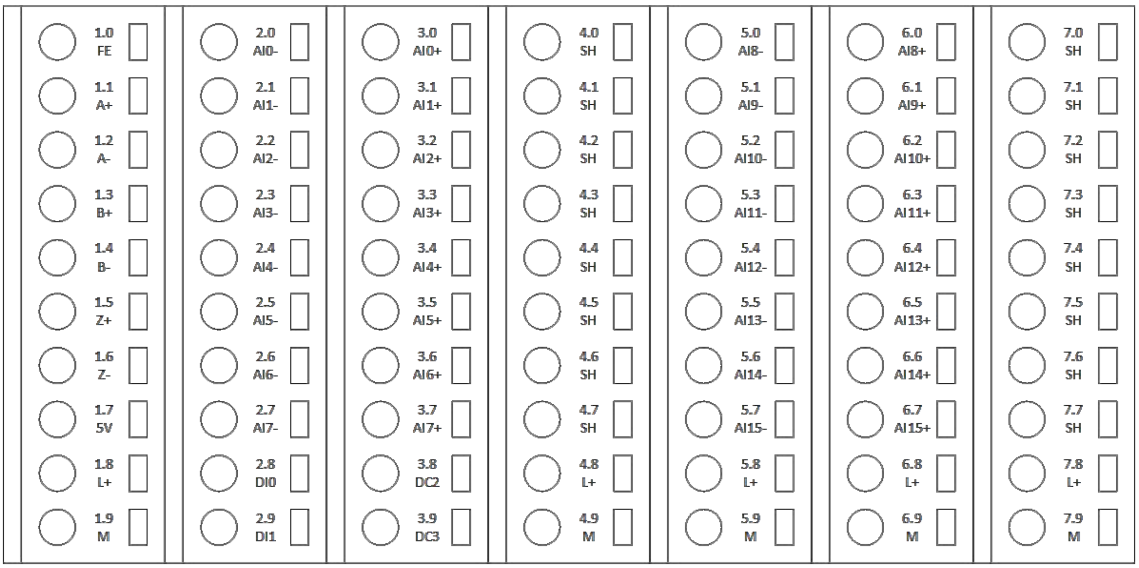
– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↳ *Chapter 1.6.2.9.2.5 “TA524 - Dummy communication module” on page 5179*.
– I/O bus connectors must not be touched during operation.

Connections

The connection is set up using the terminals of the TF5x1-CMS.

Mounting, disassembling and connection for the terminal function block and the I/O modules are described in the system assembly chapter, as well as the serial I/O bus ↳ Chapter 1.6.3.4 “Overall information (valid for complete AC500 product family)” on page 5218.

Terminal assignment of the TF5x1-CMS



Terminal	Signal	Description
1.0	FE	Functional earth for encoder shield connection
1.1	A+	Input signal A of encoder 0
1.2	A-	Inverted input signal A of encoder 0
1.3	B+	Input signal B of encoder 0

Terminal	Signal	Description
1.4	B-	Inverted input signal B of encoder 0
1.5	Z+	Input signal Z of encoder 0
1.6	Z-	Inverted input signal Z of encoder 0
1.7	5 V	+5 V DC power supply output for encoder
1.8	L+	Process voltage L+ (24 V DC)
1.9	M	Process voltage M (0 V DC)
2.0...2.7	AI0-...AI7-	Negative input signal AI0...AI7 for analog channel 0...7
2.8/2.9	DI0/DI1	Input signal I0/I1 (standard digital input)
3.0...3.7	AI0+...AI7+	Positive input signal AI0...AI7 for analog channel 0...7
3.8/3.9	DC2/DC3	Signal of configurable digital input/output C2/C3
4.0...4.7	SH	Shield connection
4.8	L+	Process voltage L+ (24 V DC)
4.9	M	Process voltage M (0 V DC)
5.0...5.7	AI8-...AI15-	Negative input signal AI0AI7 for analog channel 8...15
5.8	L+	Process voltage L+ (24 V DC)
5.9	M	Process voltage M (0 V DC)
6.0...6.7	AI8+...AI15+	Positive input signal AI0...AI7 for analog channel 8...15
6.8	L+	Process voltage L+ (24 V DC)
6.9	M	Process voltage M (0 V DC)
7.0...7.7	SH	Shield connection
7.8	L+	Process voltage L+ (24 V DC)
7.9	M	Process voltage M (0 V DC)

CAUTION!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you remove or replace a module.

Analog signals must be transmitted through shielded cables. The analog cable shield must only be connected to the side of the module (SH terminals) to avoid isothermal relaxation currents influencing the measuring results and for optimal robustness against external noise. The shield connection must be as short as possible (< 3 cm). The analog shield is capacitive and internally coupled with the functional earth (FE). To avoid unacceptable potential differences between different parts of the installation, low-resistance equipotential bonding conductors must be laid.

CAUTION!

Risk of damaging the processor module and terminal base!

Voltages surpassing the permitted range might damage the processor module and terminal base.

Never connect supply and process voltages > 30 V DC to the terminal base.

NOTICE!

Risk of damaging the terminal base and power supply!

Short circuits might damage the terminal base and power supply.

Make sure that the four clamps L+ and M (two of each) are not wrongly connected (e. g. +/- of power supply is connected to both L+/L+ or both M/M).

NOTICE!

Risk of damaging terminal base!

Excessive current might damage the clamp and terminal base.

Make sure that the current flowing through the spring terminals never exceeds 10 A.

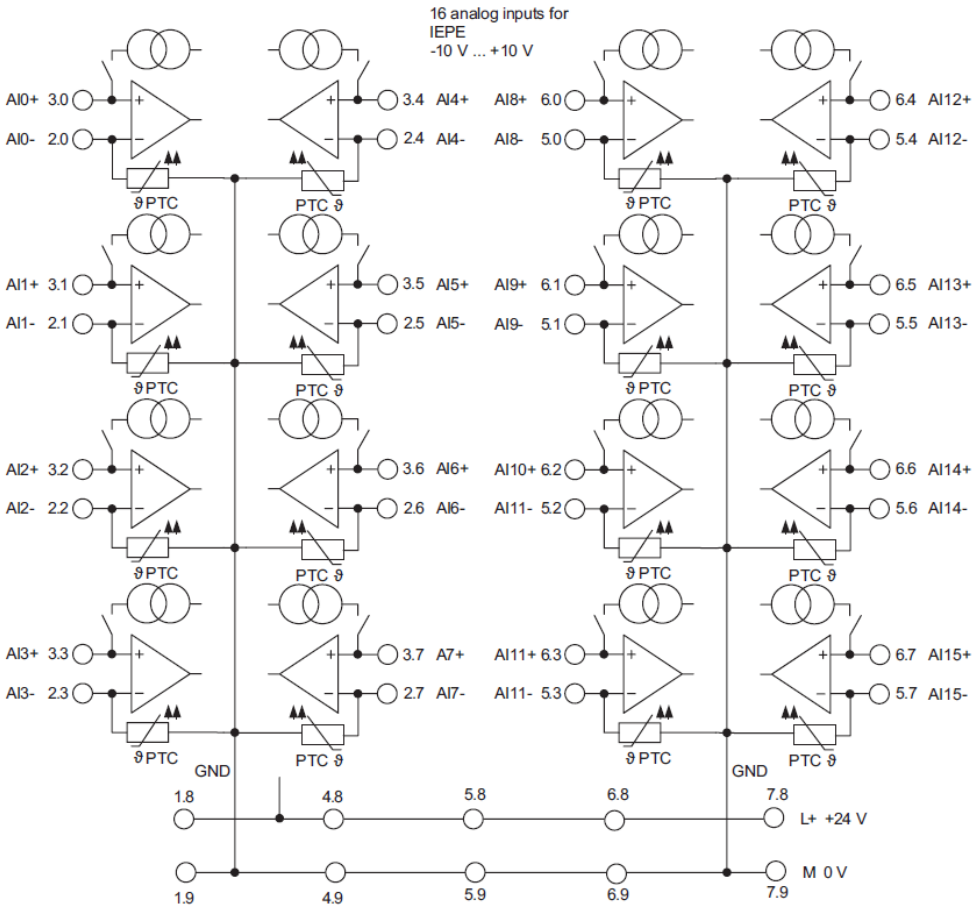


Fig. 729: Terminal assignment and connection

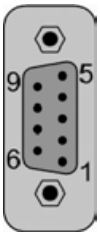
Serial interface COM1

The serial interface COM1 can be used for:

- Online access (RS-232 programming interface for Automation Builder software)
- Free protocol
- Modbus RTU, client and server
- CS31 bus (RS-485), as master only

↳ *Chapter 1.6.3.6.4.6 “Serial interface COM1 of the terminal bases” on page 5343*.

Pin assignment

Serial Interface	Pin	Signal	Interface	Description	
	1	FE	-	Functional earth	
	2	TxD	RS-232	Transmit data	Output
	3	RxD/TxD-P	RS-485	Receive/Transmit	Positive
	4	RTS	RS-232	Request to send	Output
	5	SGND	Signal ground	0 V supply out	
	6	+5 V	-	5 V supply out	
	7	RxD	RS-232	Receive data	Input
	8	RxD/TxD-N	RS-485	Receive/Transmit	Negative
	9	CTS	RS-232	Clear to send	Input
	Shield	FE	-	Functional earth	

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. ↳ *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

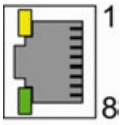
Ethernet interface

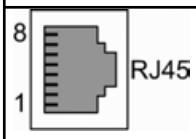
This interface is the connection to the internal Ethernet communication module of the processor modules.

Applications:

- TCP/IP for PC/Automation Builder (programming)
- UDP: communication via function blocks
- Modbus on TCP/IP, master and slave

Pin assignment

Interface	Pin	Signal	Description
 or	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used

Interface	Pin	Signal	Description
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

See supported protocols and used Ethernet ports for AC500 V2products: *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.9 “Communication with Modbus TCP/IP” on page 5488*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Connection of the supply voltage 24 V DC at the TF5x1-CMS		The terminals 1.8, 4.8...7.8, 1.9, 4.9...7.9, 4.0...4.7, 7.0...7.7 are electrically interconnected within the TF5x1-CMS. Terminals 1.8, 4.8...7.8: process voltage L+ = +24 V DC Terminals 1.9, 4.9...7.9: process voltage M = 0 V Terminals 4.0...4.7, 7.0...7.7: analog shield clamps SH Terminal 1.0: FE shield clamp of encoder
Rated voltage		24 V DC
Max. permitted total current		10 A (between terminals 1.8, 4.8...7.8 and 1.9, 4.9...7.9)
Slots		
	TF501-CMS	1 function module FM502-CMS, 1 processor module PM592-ETH, 0 communication modules
	TF521-CMS	1 function module FM502-CMS, 1 processor module PM592-ETH, 2 communication modules

Parameter	Value
Processor module interfaces	I/O bus, COM1
Weight	TF501-CMS: 350 g
	TF521-CMS: 400 g
Mounting position	Horizontal or vertical

Table 263: Connection of the TF5x1-CMS

Parameter	Value
I/O bus	I/O interface for directly adding up to 10 terminal units
Terminal block	70 clamps for I/O, shield and power supply connection
COM1	Serial interface, 9-pin D-sub connector, female
Network interface (type must be equal to the type of the used processor module)	Ethernet RJ45

Ordering data

Part No.	Scope of delivery	Product life cycle status
1SAP 117 000 R0271	TF501-CMS, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 1 communication module, Ethernet RJ45 connector	Active
1SAP 317 000 R0271	TF501-CMS-XC, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 1 communication module, Ethernet RJ45 connector, XC version	Active
1SAP 117 200 R0271	TF521-CMS, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 2 communication modules, Ethernet RJ45 connector	Active
1SAP 317 200 R0271	TF521-CMS-XC, function module terminal base, slots: 1 function module FM502-CMS, 1 processor module PM592-ETH, 2 communication modules, Ethernet RJ45 connector, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.3 Processor modules

The AC500 product family consists of the product groups:

- AC500 (standard):
  AC500 standard PLCs offer a wide range of performance levels and scalability. The PLCs are highly capable of communication and extension for flexible application.
- AC500-eCo:
  AC500-eCo PLCs are cost-effective, high-performance compact PLCs that offer total interoperability with the core AC500 range and provide battery-free uninterrupted output. All I/O modules can be freely connected in a simple, stable and reliable manner.
- AC500-S:
  AC500-S PLCs are designed for safety applications involved in factory, process or machinery automation area.
- AC500-XC:
  AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. These variants operate according to their product group and can, in addition, be operated under extreme conditions. AC500-XC PLCs can be used at high altitudes, extended operating temperature and in humid condition. Further, the devices provide immunity to vibration and hazardous gases. The AC500-XC series is consistent with standard devices in the overall dimensions, control function and software compatibility. *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*.

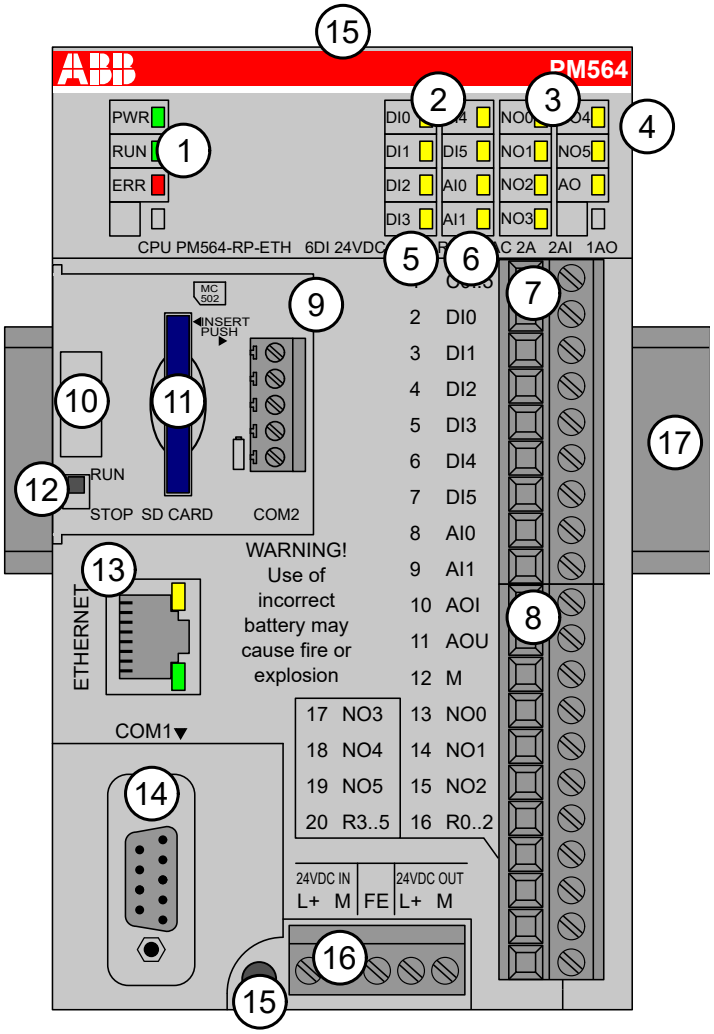
The AC500 product family is characterized by functional modularity. As the complete AC500 product family shares the same hardware platform and programming software tool, the devices of the AC500 product groups can be flexibly combined.

S500 devices represent the I/O modules of the product group AC500 (standard), whereas S500-eCo devices represent the I/O modules of the product group AC500-eCo. Both S500 and S500-eCo devices can be combined with devices of the AC500 product family in a flexible way.

1.6.2.3.1 AC500-eCo

PM55x-xP and PM56x-xP

- PM55x-xP: Processor module with integrated digital inputs and outputs
- PM56x-xP: Processor module with integrated digital and analog inputs and outputs



1 3 LEDs to display the states of the processor module
2 **PM55x-xP**: 8 yellow LEDs to display the states of the digital input signals.
PM56x-xP: 6 yellow LEDs to display the states of the digital input signals, 2 yellow LEDs to display the states of the analog input signals.
3 **PM55x-xP**: 6 yellow LEDs to display the states of the digital output signals.
PM56x-xP: 6 yellow LEDs to display the states of the digital output signals, 1 yellow LED to display the state of the analog output signal
4 I/O bus for connecting additional I/O modules
5 Terminal number
6 Signal name according to terminal number
7 Terminal block for input/output signals (9-pin)
8 Terminal block for input/output signals (11-pin)
9 Removable 5-pin connector for COM2 (optional)
10 Recess for opening the option board slot cover
11 Memory card slot (optional)
12 RUN/STOP switch
13 Ethernet interface (depending on model)
14 9-pin D-sub jack (COM1) for RS-485 connection
15 2 holes for wall-mounting with screws
16 Removable 5-pin connector for power supply (24 V DC or 100-240 V AC - depending on model)
17 DIN rail

The processor module is shown with pluggable terminal blocks mounted. These terminal blocks must be ordered separately.

Short description

The processor modules PM55x-xP and PM56x-xP are the central units of AC500-eCo. Their main characteristics are:

- 128 kB (PM554-xP and PM564-xP types) program memory, 512 kB (PM556-xP and PM566-xP types) program memory
- I/O bus (for expansion with max. 10 I/O modules)
- COM1 (serial RS-485 interface)
- 8 digital inputs (PM55x-xP), 6 digital inputs (PM56x-xP)
- 6 digital outputs
- 2 analog inputs (PM56x-xP only; the 2 analog inputs can be configured as digital inputs)
- 1 analog output (PM56x-xP only)

The various processor module variants differ in the following characteristics:

- Power supply (24 V DC or 100-240 V AC)
- Type of the digital outputs (transistor or relays)
- Ethernet interface (only models with suffix -ETH) - Analog inputs/outputs (only type PM56x-xP)

All processor module variants can be expanded to include an memory card slot, a second serial RS-485 interface (COM2) and an RTC (real-time clock).

Details and technical data are provided in the technical data section *Chapter 1.6.2.3.1.1.8 “Technical data” on page 3814*.

Assortment

Processor Module	Ethernet interface	Other interfaces	Type of digital outputs	Power supply
PM554-TP	-	Serial RS-485 interface (COM1) Serial RS-485 interface (COM2, optional) I/O bus Memory card slot (optional)	Transistor	24 V DC
PM554-TP-ETH	x		Transistor	24 V DC
PM554-RP	-		Relays	24 V DC
PM554-RP-AC	-		Relays	100-240 V AC
PM556-TP-ETH	x		Transistor	24 V DC
PM564-TP	-		Transistor	24 V DC
PM564-TP-ETH	x		Transistor	24 V DC
PM564-RP	-		Relays	24 V DC
PM564-RP-AC	-		Relays	100-240 V AC
PM564-RP-ETH	x		Relays	24 V DC
PM564-RP-ETH-AC	x		Relays	100-240 V AC
PM566-TP-ETH	x		Transistor	24 V DC

Connections

I/O bus

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules can be added (see description for I/O bus in the system assembly chapter *Chapter 1.6.3.4.1 “Serial I/O bus” on page 5218*).

Serial interface COM1

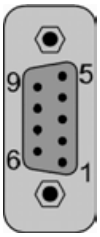
The serial non-isolated COM1 interface provides communication via RS-485 and is carried out as a 9-pin D-sub jack. The COM1 interface can be used

- for online connection with Automation Builder software (via a RS-485 programming cable. e. g. TK503 *Chapter 1.6.2.9.2.11 “TK503 - COM1 USB programming cable” on page 5190*)
- as Modbus RTU (master and slave)
- for ASCII serial protocols
- as CS31 bus (master only).

COM1 does not support communication via RS-232. The programming cable TK501 cannot be used.

Serial interface COM1

Table 264: Pin assignment

Serial Interface	Pin	Signal	Description
Serial Interface 	1	FE	Functional earth
	2	SGND	0 V power supply, internally connected to M terminal
	3	RxD/TxD-P	Receive/Transmit positive
	4	Reserved	Reserved, not connected
	5	SGND	0 V power supply, internally connected to M terminal
	6	+3.3 V	3.3 V power supply
	7	Reserved	Reserved, not connected
	8	RxD/TxD-N	Receive/Transmit negative
	9	Reserved	Reserved, not connected
	Shield	Cable shield	Functional earth

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

The internal power supply voltage, which is connected to pin 6 of the D-sub, must not be short-circuited or connected to any other voltages.

Serial interface COM2 (optional)

The optional serial COM2 interface provides communication via RS-485 and is carried out as a removable 5-pin terminal with screw connection. The COM2 interface can be used

- for online connection with Automation Builder software (via a RS-485 programming cable. e. g. TK504 *Chapter 1.6.2.9.1.12 “TK504 - COM2 USB programming cable” on page 5143*
- as Modbus RTU (master and slave)
- for ASCII serial protocols

The serial RS-485 interface is not galvanically isolated using TA562-RS or TA562-RS-RTC.

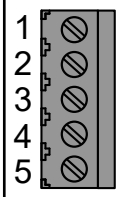
Using TA569-RS-ISO the RS-485 serial interface has galvanic isolation.

Communication via CS31 bus is not possible.

Additional information for installing the accessory modules can be found in *Chapter 1.6.2.9.1.7 “TA562-RS - Serial RS-485 adapter ” on page 5120*, *Chapter 1.6.2.9.1.9 “TA569-RS-ISO - Serial RS-485 isolated adapter” on page 5131* and *Chapter 1.6.2.9.1.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5125*.

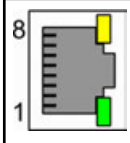
Additional information for wiring the COM2 interface can be found in serial interface COM2 (PM55x, PM56x) *Chapter 1.6.3.5.4.3 “Serial interface COM2” on page 5254*.

Table 265: Pin assignment

Serial Interface	Pin	Description
	1	Terminator P
	2	TxD/RxD-P
	3	TxD/RxD-N
	4	Terminator N
	5	Functional earth

Ethernet interface

The Ethernet interface is carried out via a RJ45 jack. The pin assignment of the Ethernet interface:

Interface	Pin	Description	
	1	Tx+	Transmit Data +
	2	Tx-	Transmit Data -
	3	Rx+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	Rx-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

The supported protocols and used Ethernet ports can be found in a separate chapter *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

Communication via Modbus TCP/IP is described in detail in a separate chapter ↳ *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

Connections

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

NOTICE!

Risk of damaging the PLC modules!

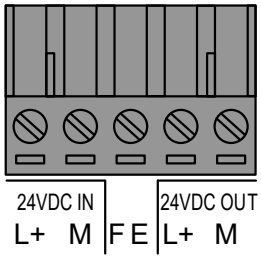
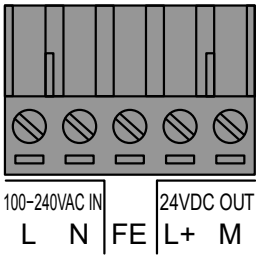
Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Power supply

Power supply

Depending on the variant, the processor modules can be connected to the following supply voltages:

	
24 V DC	100 - 240 V AC

The connection is established via a removable 5-pin terminal block. As the terminal block is also available as a spare part (inside TA570 Spare Part Set for AC500-eCo processor modules).

The 24 V DC variant contains 2 L+ and M terminals. The L+ terminal on the left side is the input and the right side is the output. The M terminals are internally interconnected. The supply can be easily looped through to the onboard digital inputs.

CAUTION!

Risk of damaging the processor module and the connected modules!

Voltages > 35 V DC (DC variants only) or > 288 V AC (AC variants only) might damage the processor module and the connected modules.

Make sure that the supply voltage never exceeds 35 V DC / 288 V AC.

CAUTION!

Risk of damaging the processor module!

Excess currents at 24 V DC output (24 V DC processor module variant) will damage the processor module.

Use an appropriate fuse ↳ *Chapter 1.6.2.3.1.1.8 “Technical data” on page 3814* within 24 V DC input connection.

The 100-240 V AC variant contains an internal power supply with a wide-range input. It provides a 24 V DC output at the terminals L+ and M which can be used to supply the onboard digital inputs.

The voltage output at 100 V AC ... 240 V AC variants can provide 180 mA max. The output is protected against overload by a self-resetting fuse (PTC).

Onboard I/Os

For connection of the onboard inputs and outputs, both a 9-pin and an 11-pin terminal block are needed and must be ordered separately. Compatible terminal blocks can be found in TA563-TA565 terminal blocks ↳ Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204.

Processor module PM55x

The processor module PM55x provides 8 onboard digital inputs (24 V DC) and 6 onboard digital outputs (depending on variant 24 V DC transistor outputs or relay outputs).

Table 266: Numbers and types of the onboard I/Os

Processor module	Power supply	No. and type of digital inputs	No. and type of digital outputs	No. and type of analog inputs	No. and type of analog outputs
PM55x-T(P), PM55x-T(P)-ETH	24 V DC	8 x 24 V DC	6 x 24 V DC, 0.5 A max. (transistor)	none	none
PM55x-R(P)	24 V DC	8 x 24 V DC	6 x relay output, 2 A max.	none	none
PM55x-R(P)-AC	120 to 240 V AC	8 x 24 V DC	6 x relay output, 2 A max.	none	none

All inputs (DI0...DI7) belong to 1 group. All outputs (DO0...DO5 / NO0...NO5) belong to 1 group. The inputs and outputs are group-wise galvanically isolated.

Processor module PM56x

The processor module PM56x provides 6 onboard digital inputs (24 V DC), 6 onboard digital outputs (depending on variant 24 V DC transistor outputs or relay outputs), 2 onboard analog inputs (voltage 0 V...10 V) and 1 onboard analog output (voltage 0 V...10 V or current 0 mA...20 mA / 4 mA...20 mA). The onboard analog inputs can be configured as digital inputs, so 8 onboard digital inputs may be available if no analog inputs are needed.

Table 267: Numbers and types of the onboard I/Os

Processor module	Power supply	No. and type of digital inputs	No. and type of digital outputs	No. and type of analog inputs	No. and type of analog outputs
PM56x-T(P), PM56x-T(P)-ETH	24 V DC	6 x 24 V DC *)	6 x 24 V DC, 0.5 A max. (transistor)	2 x voltage *)	1 x voltage or current
PM56x-R(P), PM56x-R(P)-ETH	24 V DC	6 x 24 V DC *)	6 x relay output, 2 A max.	2 x voltage *)	1 x voltage or current
PM56x-R(P)-AC, PM56x-R(P)-ETH-AC	100-240 V AC	6 x 24 V DC *)	6 x relay output, 2 A max.	2 x voltage *)	1 x voltage or current

*) PM56x has 2 analog inputs which can be configured as digital inputs. If the analog inputs are configured as digital inputs, 8 digital inputs are available overall.

All digital inputs (DI0...DI5) belong to 1 group. All digital outputs (DO0...DO5 / NO0...NO5) belong to 1 group. These inputs and outputs are group-wise galvanically isolated.

The 2 analog inputs are not galvanically isolated from the 24 V power supply of the processor module.

For more information on the onboard I/Os, refer to onboard I/Os in processor module PM55x ↳ *Chapter 1.6.2.3.1.2 “Onboard I/Os in processor module PM55x” on page 3819* and onboard I/Os in processor module PM56x ↳ *Chapter 1.6.2.3.1.3 “Onboard I/Os in processor module PM56x” on page 3831*.

Diagnosis

The AC500 processor module can display various errors according to the error classes. The following error classes are possible. The reaction of the processor module is different for each type of error.

Error class	Type	Description	Example
E1 ERR-LED is ON	Fatal error	A safe function of the operating system is no longer guaranteed.	Checksum error in the system Flash or RAM error
E2 ERR-LED is ON	Severe error	The operating system is functioning without problems, but the error-free processing of the user program is no longer guaranteed.	Checksum error in the user Flash, independent of the task duration
E3 ERR-LED is ON/OFF *)	Minor error	It depends on the application if the user program should be stopped by the operating system or not. The user should determine which reaction is necessary.	Flash could not be programmed, I/O module has failed

Error class	Type	Description	Example
E4 ERR-LED is ON/OFF *)	Warning	Error in the periphery (e.g. I/O) which may show an impact in the future. The user should determine which reaction is necessary.	Short-circuit at an I/O module, the battery is run down or not inserted
*) The behaviour if the ERR-LED lights up at error classes E3 or E4 is configurable.			

Occurred errors can be displayed with the commands diagshow all in the PLC-Browser of Automation Builder software.

State LEDs and operating elements

RUN/STOP switch

The processor modules PM55x-xP and PM56x-xP contain a RUN/STOP switch which can be set with a small screwdriver. In the RUN position, the program loaded in the processor module will be executed and in the STOP position it will be stopped.

When COM1 and COM2 are not in online access mode, the user program can only be changed, uploaded and downloaded if the RUN/STOP switch is in STOP position.

State LEDs

The processor modules PM55x-xP and PM56x-xP indicate their states of operation via 3 LEDs located on the upper left edge of the processor module.

LED	State	Color	LED = ON	LED = OFF	LED flashing
PWR	Power supply	Green	Power supply present	Power supply missing	--
RUN	RUN/STOP state	Green	Processor module is in state RUN	Processor module is in state STOP	Fast flashing (4 Hz): The processor module is reading/ writing data from/to the memory card. If the ERR-LED is also flashing, data is being written to the flash EEPROM. Slow flashing (1 Hz): The firmware update from the memory card has been completed successfully.
ERR	Error indication	Red	An error occurred	No errors or only warnings encountered (E4-errors). The LED behavior for the error classes 2 to 4 is configurable.	With 4 Hz (fast): displays together with the RUN LED a currently running a firmware-upgrade or writing data to the Flash-EPROM.

I/O LEDs

Each processor module contains up to 15 LEDs (depending on type) to display the states of the inputs and outputs.

Processor module	LED	State	Color	LED = ON	LED = OFF
PM55x-xP PM56x-xP	I0...I7 (PM55x-xP) I0...I5 (PM56x-xP)	Digital input	Yellow	Input is ON	Input is OFF
	O0...O5	Digital output	Yellow	Output is ON	Output is OFF
PM56x-xP	AI0, AI1 *)	Analog input	Yellow	Input is ON	Input is OFF
	AO	Analog output	Yellow	Output is ON	Output is OFF
*) The analog inputs AI0 and AI1 can be configured as digital input or analog input.					

State LEDs

Table 268: State LEDs at Ethernet connector (-ETH models only)

LED	Color	OFF	ON	Flashing
Activity	Yellow	No activity	---	Activity
Link	Green	No link	Link	---

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

General data

Power supply	24 V DC	100 - 240 V AC
Connection of power supply	Via removable 5-pin screw terminal	
Current consumption from power supply (max.)	PM554-TP: 180 mA PM554-TP-ETH: 190 mA PM554-RP: 220 mA PM556-TP-ETH: 190 mA PM564-TP: 210 mA PM564-TP-ETH: 220 mA PM564-RP: 240 mA PM564-RP-ETH: 250 mA PM566-TP-ETH: 220 mA	PM554-RP-AC: 200 mA at 100 V AC, 110 mA at 240 V AC *) PM564-RP-AC: 210 mA at 100 V AC, 125 mA at 240 V AC *) PM564-RP-ETH-AC: 220 mA at 100 V AC, 130 mA at 240 V AC *)
Current consumption from power supply (typ.)	PM554-TP: 60 mA PM554-TP-ETH: 70 mA PM554-RP: 80 mA PM556-TP-ETH: 70 mA PM564-TP: 95 mA PM564-TP-ETH: 100 mA PM564-RP: 110 mA PM564-RP-ETH: 120 mA PM566-TP-ETH: 100 mA	PM554-RP-AC: 20 mA at 100 V AC, 12 mA at 240 V AC *) PM564-RP-AC: 20 mA at 100 V AC, 11 mA at 240 V AC *) PM564-RP-ETH-AC: 23 mA at 100 V AC, 14 mA at 240 V AC *)
Inrush current at nominal voltage	Typ. 3.9 A^2s	Typ. 0.3 A^2s
Required fuse	3 A fast	Max. 10 A

Power supply	**24 V DC**	**100 - 240 V AC**
Max. power dissipation within the processor module	PM554-TP: 3.0 W PM554-TP-ETH: 3.3 W PM554-RP: 3.5 W PM556-TP-ETH: 3.3 W PM564-TP: 3.9 W PM564-TP-ETH: 4.4 W PM564-RP: 4.5 W PM564-RP-ETH: 4.9 W PM566-TP-ETH: 4.4 W	PM554-RP-AC: 4.8 W PM564-RP-AC: 4.8 W PM564-RP-ETH-AC: 5.3 W
Processor module interfaces	I/O bus, COM1, COM2 (optional), Ethernet (depending on model)	
Connection system	see System Assembly, Construction and Connection *Chapter 1.6.3.5 “AC500-eCo” on page 5233*	
Weight	PM554-TP: 300 g PM554-TP-ETH: 300 g PM554-RP: 350 g PM556-TP-ETH: 300 g PM564-TP: 300 g PM564-TP-ETH: 300 g PM564-RP: 350 g PM564-RP-ETH: 350 g PM566-TP-ETH: 300 g	PM554-RP-AC: 400 g PM564-RP-AC: 400 g PM564-RP-ETH-AC: 400 g
Mounting position	horizontal or vertical	

*) These values show the value of the apparent current (sum of active and reactive current)

Detailed data

Program memory	128 kB Flash EPROM (PM554-xP and PM564-xP types) 512 kB Flash EPROM (PM556-xP and PM566-xP types)
Data memory	
- VAR data	10 kB
- VAR_RETAIN data	1 kB, always buffered in flash
- %RB data (persistent)	1 kB, can be buffered in flash (depending on configuration)
- %MB data	2 kB (PM554 and PM564 types) 64 kB (PM556 and PM566 types)
Data buffering	In flash memory
Real-time clock (RTC)	Optional
Battery low indication	Warning

Programming languages	- Instruction List (IL) - Function Block Diagram (FBD) - Ladder Diagram (LD) - Sequential Function Chart (SFC) - Structured Text (ST) - Continuous Function Chart (CFC)
Cycle time for 1000 instructions	
Binary	0.08 ms
Word	0.1 ms
Floating point	1.2 ms
Program execution	
Cyclic	Yes
Time-controlled	Yes
Multitasking	Yes
Interruption	1 interrupted with up or down edge detection
LEDs	Power, Run, Error, Status of I/Os
RUN/STOP switch	Yes
Protection of the user program by password	Possible
Usable accessories	MC503: Memory card TA561-RTC: Real-time clock TA562-RS: Serial RS-485 TA569-RS-ISO: Serial RS-485 isolated TA562-RS-RTC: Real-time clock and serial RS-485

Detailed data of the interfaces

Serial interface COM1	
Physical link	RS-485
Galvanic isolation	none
Transmission rate	Configurable from 1.2 to 187.5 kBit/s
Connection	9-pin D-sub female connector
Common mode range	Typ. -8 V / +12 V (CAUTION: The interface can be damaged if the signal exceeds the common mode range.)
Usage	- Programming port - Modbus (master and slave) - Serial ASCII communication - CS31 (master only)

Serial interface COM2 (optional)	
Physical link	RS-485
Galvanic isolation	none (TA562-RS or TA562-RS-RTC) 500 V DC (TA569-RS-ISO)
Baudrate	Configurable from 1.2 to 115.2 kBit/s
Connection	Removable 5-pin terminal block
Common mode range	Typ. -8 V / +12 V (CAUTION: The interface can be damaged if the signal exceeds the common mode range.)
Usage	- Programming port - Modbus (master and slave) - Serial ASCII communication

Data of I/Os

	PM55x-xP	**PM56x-xP**
Max. number of I/O modules	10	10
Digital inputs	320 + 8	320 + 8
Digital outputs	240 + 6	240 + 6
Type of digital outputs	PM554-TP PM554-TP-ETH PM554-RP PM554-RP-AC PM556-TP-ETH PM564-TP PM564-TP-ETH PM564-RP PM564-RP-AC PM564-RP-ETH PM564-RP-ETH-AC PM566-TP-ETH	Transistor Transistor Relays Relays Transistor Transistor Transistor Relays Relays Relays Relays Transistor
Analog inputs	160	160 + 2
Analog outputs	160	160 + 1
Number of decentralized inputs and outputs	On CS31 Bus: up to 31 stations with up to 120 digital inputs / 120 digital outputs each	
Detailed data of the onboard I/O	Onboard I/Os in PM55x and Onboard I/Os in PM56x *Chapter 1.6.2.3.1.2 “Onboard I/Os in processor module PM55x” on page 3819* *Chapter 1.6.2.3.1.3 “Onboard I/Os in processor module PM56x” on page 3831*	

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Ordering data

Table 269: Processor modules for AC500-eCo

Part no.	Description	Product life cycle phase *)
1SAP 120 600 R0001	PM554-TP, processor module, 128 kB memory, 8 DI, 6 DO-T, 24 V DC, with pluggable I/O terminal blocks	Active
1SAP 120 600 R0071	PM554-TP-ETH, processor module, 128 kB memory, 8 DI, 6 DO-T, 24 V DC, onboard Ethernet, with pluggable I/O terminal blocks	Active
1SAP 120 700 R0001	PM554-RP, processor module, 128 kB memory, 8 DI, 6 DO-R, 24 V DC, with pluggable I/O terminal blocks	Active
1SAP 120 800 R0001	PM554-RP-AC, processor module, 128 kB memory, 8 DI, 6 DO-R, 100 V AC...240 V AC, with pluggable I/O terminal blocks	Active
1SAP 121 200 R0071	PM556-TP-ETH, processor module, 512 kB memory, 8 DI, 6 DO-T, 24 V DC, onboard Ethernet, with pluggable I/O terminal blocks	Active
1SAP 120 900 R0001	PM564-TP, processor module, 128 kB memory, 6 DI, 6 DO-T, 2 AI and 1 AO, 24 V DC	Active
1SAP 120 900 R0071	PM564-TP-ETH, processor module, 128 kB memory, 6 DI, 6 DO-T 2 AI and 1 AO, 24 V DC, Ethernet interface	Active
1SAP 121 000 R0001	PM564-RP, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI and 1 AO, 24 V DC	Active
1SAP 121 100 R0001	PM564-RP-AC, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI and 1 AO, 100 V AC...240 V AC	Active
1SAP 121 000 R0071	PM564-RP-ETH, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI and 1 AO, 24 V DC, Ethernet interface	Active
1SAP 121 100 R0071	PM564-RP-ETH-AC, processor module, 128 kB memory, 6 DI, 6 DO-R, 2 AI and 1 AO, 100 V AC...240 V AC, Ethernet interface	Active
1SAP 121 500 R0071	PM566-TP-ETH, processor module, 512 kB memory, 6 DI, 6 DO-T, 2 AI and 1 AO, 24 V DC, Ethernet interface	Active

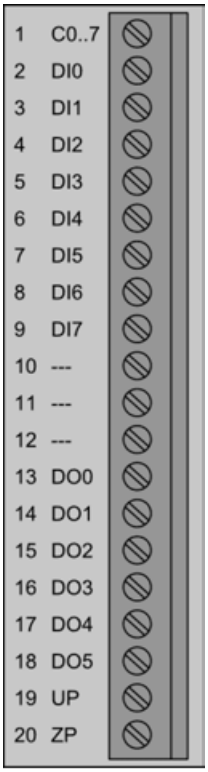
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 270: Accessories

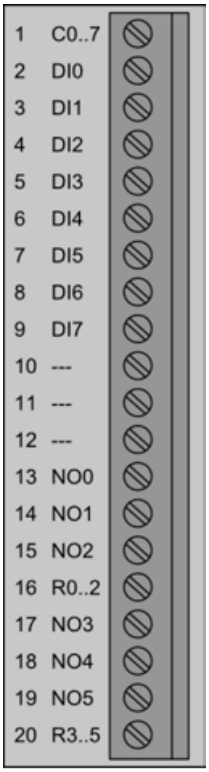
Part no.	Description
1TNE 968 901 R3101	Terminal Block TA563-9, 9-pin, screw front, cable side, 6 pieces per unit
1TNE 968 901 R3102	Terminal Block TA563-11, 11-pin, screw front, cable side, 6 pieces per unit
1TNE 968 901 R3103	Terminal Block TA564-9, 9-pin, screw front, cable front, 6 pieces per unit
1TNE 968 901 R3104	Terminal Block TA564-11, 11-pin, screw front, cable front, 6 pieces per unit
1TNE 968 901 R3105	Terminal Block TA565-9, 9-pin, spring front, cable front, 6 pieces per unit
1TNE 968 901 R3106	Terminal Block TA565-11, 11-pin, spring front, cable front, 6 pieces per unit
1SAP 180 100 R0001	MC502: Memory card
1TNE 968 901 R0100	MC503: Memory card adapter for PM55x-xP and PM56x-xP
1TNE 968 901 R1100	TK503: COM1 USB programming cable
1TNE 968 901 R2100	TK504: COM2 USB programming cable
1TNE 968 901 R3200	TA561-RTC: real-time clock adapter for PM55x-xP and PM56x-xP
1TNE 968 901 R4300	TA562-RS: serial RS-485 adapter for PM55x-xP and PM56x-xP
1SAP 186 400 R0001	TA569-RS-ISO: serial RS-485 adapter with galvanic isolation for PM55x-XP and PM56x-xP
1TNE 968 901 R5210	TA562-RS-RTC: serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP
1TNE 968 901 R3107	TA566: wall mounting accessory, 100 pieces
1TNE 968 901 R3203	TA570: spare part set for AC500-eCo processor modules

Onboard I/Os in processor module PM55x

- 8 DI 24 V DC
- PM55x-T(P): 6 DO (24 V DC, 0.5 A max. transistor outputs)
- PM55x-R(P) and PM55x-R(P)-AC: 6 DO (24 V DC or 120/240 V AC, 2 A max. relay outputs)



Terminals of onboard I/Os for PM55x-T



Terminals of onboard I/Os for PM55x-R and PM55x-R-AC

AC500-eCo processor modules are equipped with non-removable terminals.

AC500-eCo processor modules are equipped with removable terminal blocks which must be ordered separately.

The electrical functionality of both processor module types is identical.

Intended purpose

Processor module PM55x

The processor module PM55x provides 8 onboard digital inputs (24 V DC) and 6 onboard digital outputs (depending on variant 24 V DC transistor outputs or relay outputs).

Table 271: Numbers and types of the onboard I/Os

Processor module	Power supply	No. and type of digital inputs	No. and type of digital outputs	No. and type of analog inputs	No. and type of analog outputs
PM55x-T(P), PM55x-T(P)-ETH	24 V DC	8 x 24 V DC	6 x 24 V DC, 0.5 A max. (transistor)	none	none
PM55x-R(P)	24 V DC	8 x 24 V DC	6 x relay output, 2 A max.	none	none
PM55x-R(P)-AC	120 to 240 V AC	8 x 24 V DC	6 x relay output, 2 A max.	none	none

All inputs (DI0...DI7) belong to 1 group. All outputs (DO0...DO5 / NO0...NO5) belong to 1 group. The inputs and outputs are group-wise galvanically isolated.

Functionality

Parameter	Value
Digital inputs	8 (24 V DC), can be used as source inputs or as sink inputs
Interrupt inputs	4 (DI0...DI3), configurable
Interrupt response time	Max. 0.8 ms when input delay is set to 0.1 ms
Fast counter	2 (DI0 and DI1), configurable
Digital outputs	6 transistor outputs (24 V DC, 0.5 A max.) or relay outputs (2 A max.), (depending on processor module)
PWM outputs	2 (DO2 and DO3), configurable
LED displays	For signal states
Internal power supply	Via processor module
External power supply	Via UP and ZP terminal

Connections

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove or replace a module.

NOTICE!

Risk of damaging the PLC modules!

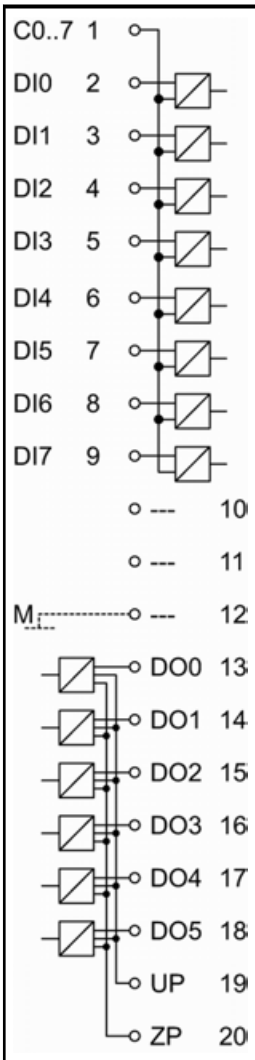
Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

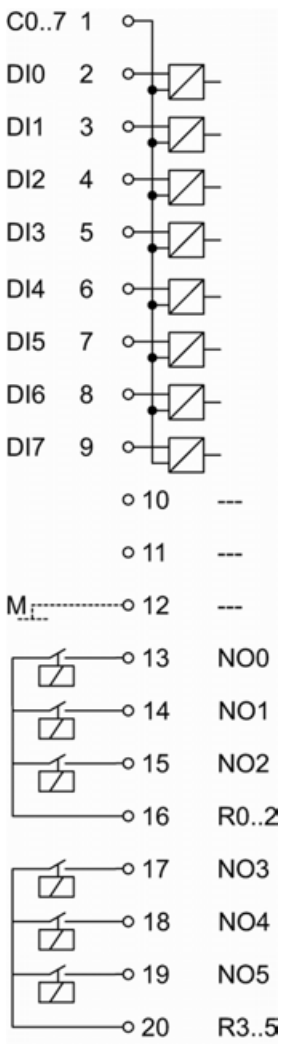
When replacing a processor module, it is recommended to mark each wire connected to the onboard I/O terminal block before disconnecting it. This should make sure that the wires can be reconnected in the same order.

The connection is carried out by using a non-removable 20-pin terminal block.

The following block diagram shows the internal structure of the onboard I/Os:



PM55x-T(P)



PM55x-R(P)

The assignment of the terminals for PM55x-T(P):

Terminal	Signal	Description
1	C0...7	Input common for digital input signals DI0 to DI7
2	DI0	Digital input signal DI0
3	DI1	Digital input signal DI1
4	DI2	Digital input signal DI2
5	DI3	Digital input signal DI3
6	DI4	Digital input signal DI4
7	DI5	Digital input signal DI5
8	DI6	Digital input signal DI6
9	DI7	Digital input signal DI7
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	DO0	Digital output signal O0

Terminal	Signal	Description
14	DO1	Digital output signal O1
15	DO2	Digital output signal O2
16	DO3	Digital output signal O3
17	DO4	Digital output signal O4
18	DO5	Digital output signal O5
19	UP	Process supply voltage UP +24 V DC
20	ZP	Process supply voltage ZP 0 V DC

The assignment of the terminals for PM55x-R(P):

Terminal	Signal	Description
1	C0...7	Input common for digital input signals DI0 to DI7
2	DI0	Digital input signal DI0
3	DI1	Digital input signal DI1
4	DI2	Digital input signal DI2
5	DI3	Digital input signal DI3
6	DI4	Digital input signal DI4
7	DI5	Digital input signal DI5
8	DI6	Digital input signal DI6
9	DI7	Digital input signal DI7
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	NO0	Normally-open relay contact of the output NO0
14	NO1	Normally-open relay contact of the output NO1
15	NO2	Normally-open relay contact of the output NO2
16	R0..2	Output common for signals NO0 to NO2
17	NO3	Normally-open relay contact of the output NO3
18	NO4	Normally-open relay contact of the output NO4
19	NO5	Normally-open relay contact of the output NO5
20	R3...5	Output common for signals NO3 to NO5

Connection of the digital inputs

The digital inputs can be used as source inputs or as sink inputs.

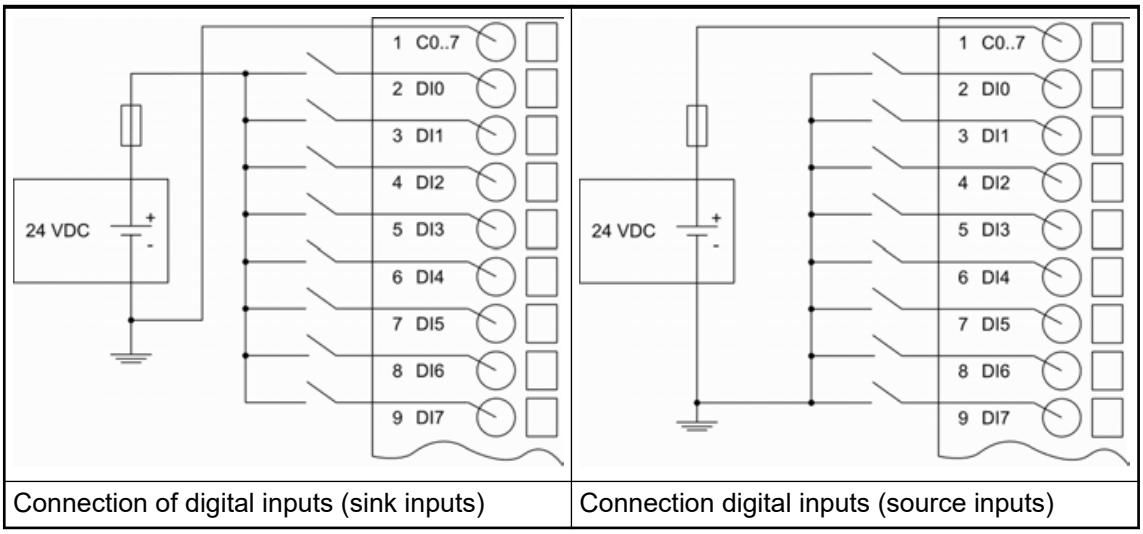
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital inputs to the PM55x processor modules:



Connection of digital inputs (sink inputs) | Connection digital inputs (source inputs)

Connection of the digital transistor outputs (PM55x-T(P) only)

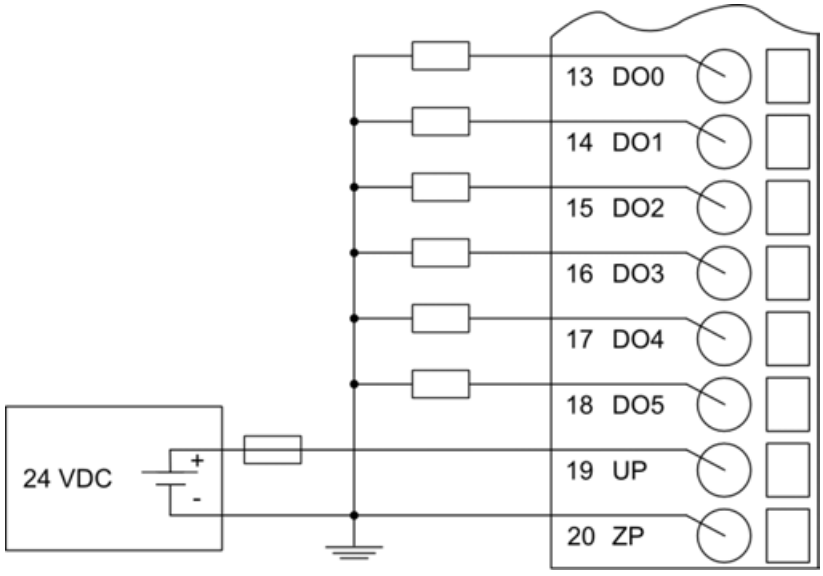


Fig. 730: Connection of digital transistor outputs

NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

CAUTION!

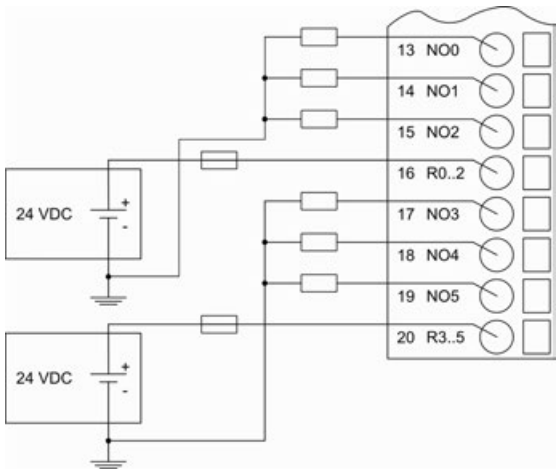
Risk of damaging the processor module!

The outputs are not protected against short circuit and overload.

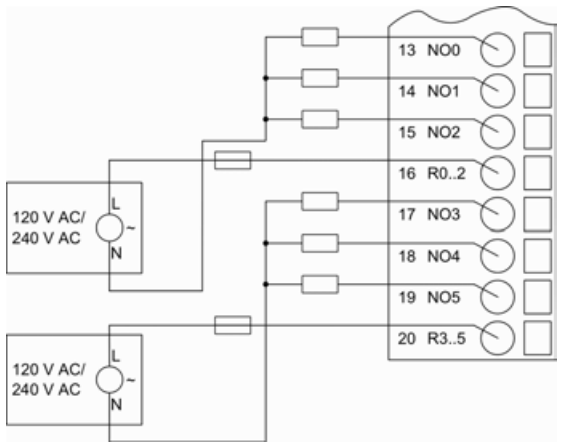
- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast protection fuse for the outputs.

Connection of the digital relay outputs (PM55x-R(P) only)

The following figures show the connection of the digital relay outputs to the processor modules:



Connection of digital relay outputs (24 V DC)



Connection of digital relay outputs (100-240 V AC)

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

CAUTION!

Risk of damaging the processor module!

– Never short-circuit or overload the outputs.
– Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
– Never connect voltages > 240 V. All outputs must be fed from the same phase.
– Use an external 5 A fast acting fuse to protect the outputs.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..2 and R3..5) does not exceed 6 A.

Never connect total currents > 6 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1

I/O configuration

The configuration data of the onboard I/Os is stored in the processor module PM55x.

Parameterization

For information about parameterization, refer to the description for onboard I/Os for processor module PM55x. *Chapter 1.6.5.2.3.8 “AC500-eCo onboard I/Os” on page 5852*

Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Class	Inter-face	Device	Module	Channel	Error-Identifier	Error message	Remedy
Errors for Onboard I/O							
Minor errors							

E1...E 4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**	**Remedy**
3	8	255	2	0	3	MaxWaitRun for onboard I/O module has expired, when PLC is put into RUN state	Reboot and try it again. If the error still exists, replace processor module for testing
3	8	255	3	0	26	Invalid configuration of onboard I/O module, e. g. 2 input channels are configured as fast counter and interrupt input at the same time.	Correct PLC configuration
Warnings							
4	8	1	2	1	2	Invalid configuration value for PWM channel. Frequency / cycle time for the PWM channel of the 8DI+6DO and 8DI+6DO+2AI+1AO module are common and if both channels are configured for PWM, the frequency of the second channel must be set to 0.	Correct frequency
4	8	1	2	0..1	4	PWM channel frequency or cycle time too high	Correct frequency or cycle time
4	8	1	2	0..1	7	PWM channel frequency or cycle time too low	Correct frequency or cycle time
4	8	1	2	0	52	Frequency on interrupt input pin too high and interrupt events are missed	Correct frequency
4	8	255	2	0	26	PLC was put into RUN state, although a configuration error is present, because parameter Run on config fault is set to YES	Correct PLC configuration
4	8	255	0	0	43	Unspecified or internal error occured	Replace processor module

Displays

LED	**Status**	**Color**	**LED = ON**	**LED = OFF**
DI0...DI7	Digital input	yellow	Input is ON	Input is OFF
DO0...DO5	Digital output	yellow	Output is ON	Output is OFF

Technical data

Technical data of the digital inputs

Parameter		Value	
Number of channels per module		8 transistor inputs (24 V DC)	
Distribution of the channels into groups		1 group for 8 channels	
Galvanic isolation		Yes, per group	
Connections of the channels I0 to I7		Terminals 2 to 9	
Reference potential for the channels I0 to I7		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1) and the module's logic is in operation	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
Signal 0		-5 V...+3 V	-3 V...+5 V
Undefined signal		-15 V...- 5 V	+5 V...+15 V
Signal 1		-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		Within -5 V...+3 V	Within -3 V...+5 V
Ripple with signal 1		Within -30 V...-15 V	Within +15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 0.1 to 32 ms (configurable via software), default: 8 ms	
Max. cable length			
	Shielded	500 m	
	Unshielded	300 m	

Technical data of the fast counter

Parameter	Value
Used inputs for the traces A and B	DI0 / DI1
Used output	DO0 / NO0
Counting frequency	On request

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

Technical data of the digital transistor outputs

Parameter		Value
Number of channels per module		6 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups		1 group of 6 channels
Galvanic isolation		Yes, per group
Connection of the channels DO0 to DO5		Terminals 13 to 18
Common power supply voltage		Terminals 19 (+24 V DC, signal name UP) and 20 (0 V DC, signal name ZP)
Reference potential for the channels DO0 to DO5		Terminal 20 (negative pole of the process voltage, name ZP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1)
Way of operation		Non-latching type
Min. output voltage at signal 1		20 V DC at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Rated protection fuse (per group)		3 A fast
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	3 A
	Rated current (all channels together, max.)	3 A
Lamp load (max.)		5 W
Max. leakage current with signal 0		0.5 mA
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 1 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Technical data of the digital relay outputs

Parameter		Value
Number of channels per module		6 normally-open relay outputs
Distribution of the channels into groups		2 groups for 3 channels
Galvanic isolation		Yes, per group
Connection of the channels NO0 to NO2		Terminals 13 to 15
Connection of the channels NO3 to NO5		Terminals 17 to 19
Reference potential for the channels NO0 to NO2		Terminal 16
Reference potential for the channels NO3 to NO5		Terminal 20
Relay output voltage		
	Rated value	24 V DC or 120/240 V AC
	Range	5 to 30 V DC or 5 to 250 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered through the I/O bus
Way of operation		Non-latching type
Output delay		
	0 to 1	Typ. 10 ms
	1 to 0	Typ. 10 ms
Rated protection fuse		5 A
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	6 A
	Rated current (all channels together, max.)	12 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)
Demagnetization when inductive loads are switched off		A free-wheeling diode must be circuited in parallel to the inductive load
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	Not possible
	With lamp loads	Max. 1 Hz
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Overload message		No

Parameter		Value
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

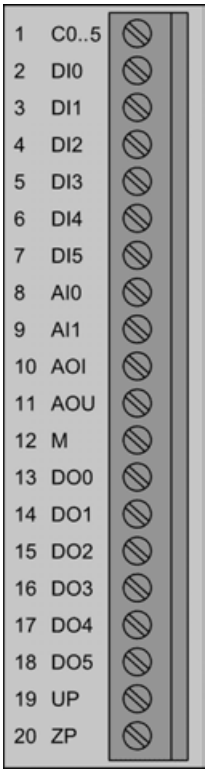
[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Technical data of the PWM outputs

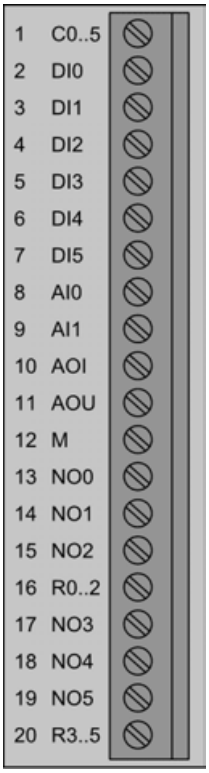
Parameter	Value
Used outputs for PWM	O2 and O3
Output frequency	125 Hz 30 kHz

Onboard I/Os in processor module PM56x

- 6 DI 24 V DC
- PM56x-T(P): 6 DO (24 V DC, 0.5 A max. transistor outputs)
- PM56x-R(P) and PM56x-R(P)-AC: 6 DO (24 V DC or 120/240 V AC, 2 A max. relay outputs)
- 2 AI (voltage 0 V...10 V)
- 1 AO (voltage 0 V...10 V or current 0 mA...20 mA / 4 mA...20 mA)



Terminals of onboard I/Os for PM56x-T(P)



Terminals of onboard I/Os for PM56x-R(P) and PM56x-R(P)-AC

AC500-eCo processor modules are equipped with non-removable terminals.

AC500-eCo processor modules are equipped with removable terminal blocks which must be ordered separately.

The electrical functionality of both processor modules is identical.

Intended purpose

Processor module PM56x

The processor module PM56x provides 6 onboard digital inputs (24 V DC), 6 onboard digital outputs (depending on variant 24 V DC transistor outputs or relay outputs), 2 onboard analog inputs (voltage 0 V...10 V) and 1 onboard analog output (voltage 0 V...10 V or current 0 mA...20 mA / 4 mA...20 mA). The onboard analog inputs can be configured as digital inputs, so 8 onboard digital inputs may be available if no analog inputs are needed.

Table 272: Numbers and types of the onboard I/Os

Processor module	Power supply	No. and type of digital inputs	No. and type of digital outputs	No. and type of analog inputs	No. and type of analog outputs
PM56x-T(P), PM56x-T(P)-ETH	24 V DC	6 x 24 V DC *)	6 x 24 V DC, 0.5 A max. (transistor)	2 x voltage *)	1 x voltage or current
PM56x-R(P), PM56x-R(P)-ETH	24 V DC	6 x 24 V DC *)	6 x relay output, 2 A max.	2 x voltage *)	1 x voltage or current
PM56x-R(P)-AC, PM56x-R(P)-ETH-AC	100-240 V AC	6 x 24 V DC *)	6 x relay output, 2 A max.	2 x voltage *)	1 x voltage or current

*) PM56x has 2 analog inputs which can be configured as digital inputs. If the analog inputs are configured as digital inputs, 8 digital inputs are available overall.

All digital inputs (DI0...DI5) belong to 1 group. All digital outputs (DO0...DO5 / NO0...NO5) belong to 1 group. These inputs and outputs are group-wise galvanically isolated.

The 2 analog inputs are not galvanically isolated from the 24 V power supply of the processor module.

Functionality

Parameter	Value
Digital inputs	6 (24 V DC), can be used as source inputs or as sink inputs
Interrupt inputs	4 (DI0...DI3), configurable
Interrupt response time	Max. 0.8 ms when input delay is set to 0.1 ms
Fast counter	2 (DI0 and DI1), configurable
Digital outputs	6 transistor outputs (24 V DC, 0.5 A max) or relay outputs (2 A max), (depending on processor module)
PWM outputs	2 (DO2 and DO3), configurable
Analog inputs	2, voltage input 0 V DC...10 V DC, can be configured as digital inputs
Analog outputs	1, voltage output 0 V DC...10 V DC or current output 0 mA...20 mA / 4 mA...20 mA
LED displays	For signal states
Internal power supply	Via processor module
External power supply	Via processor module

Connections

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

Risk of damaging the PLC modules!

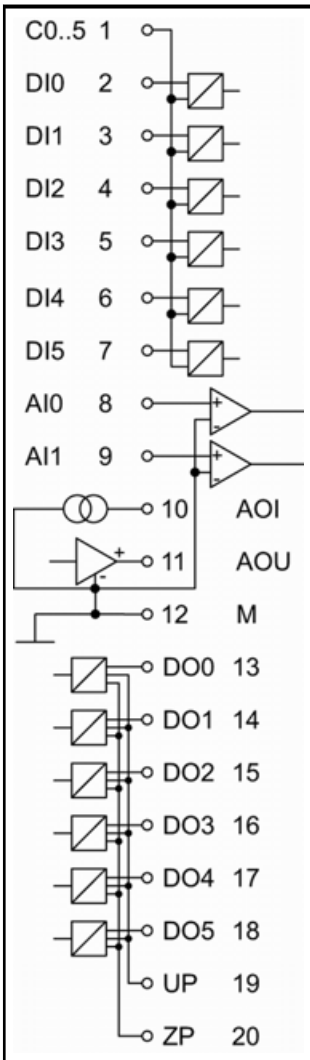
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

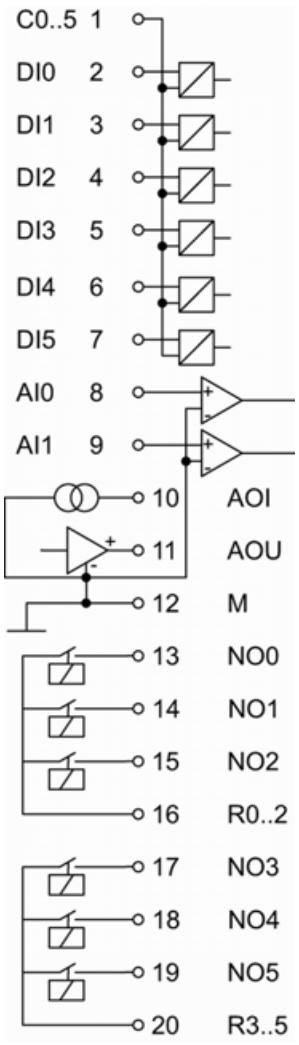
When replacing a processor module, it is recommended to mark each wire connected to the onboard I/O terminal block before disconnecting it. This should make sure that the wires can be reconnected in the same order.

The connection is carried out by using a non-removable 20-pin terminal block.

The following block diagram shows the internal structure of the onboard I/Os:



PM56x-T(P)



PM56x-R(P)

Assignment of the Terminals for PM56x-T(P)

Terminal	Signal	Description
1	C0...5	Input common for digital input signals DI0 to DI5
2	DI0	Digital input signal DI0
3	DI1	Digital input signal DI1
4	DI2	Digital input signal DI2
5	DI3	Digital input signal DI3
6	DI4	Digital input signal DI4
7	DI5	Digital input signal DI5
8	AI0	Analog voltage input signal AI0
9	AI1	Analog voltage input signal AI1
10	AOU	Analog voltage output
11	AOI	Analog current output
12	M	Input/output common for analog signals
13	DO0	Digital output signal O0
14	DO1	Digital output signal O1
15	DO2	Digital output signal O2
16	DO3	Digital output signal O3
17	DO4	Digital output signal O4
18	DO5	Digital output signal O5
19	UP	Process supply voltage UP +24 V DC
20	ZP	Process supply voltage ZP 0 V DC

Assignment of the Terminals for PM56x-R(P)

Terminal	Signal	Description
1	C0...5	Input common for digital input signals DI0 to DI5
2	DI0	Digital input signal DI0
3	DI1	Digital input signal DI1
4	DI2	Digital input signal DI2
5	DI3	Digital input signal DI3
6	DI4	Digital input signal DI4
7	DI5	Digital input signal DI5
8	AI0	Analog voltage input signal AI0
9	AI1	Analog voltage input signal AI1
10	AOU	Analog voltage output
11	AOI	Analog current output
12	M	Input/output common for analog signals
13	NO0	Normally-open relay contact of the output NO0
14	NO1	Normally-open relay contact of the output NO1
15	NO2	Normally-open relay contact of the output NO2
16	R0..2	Output common for signals NO0 to NO2
17	NO3	Normally-open relay contact of the output NO3
18	NO4	Normally-open relay contact of the output NO4

Terminal	Signal	Description
19	NO5	Normally-open relay contact of the output NO5
20	R3...5	Output common for signals NO3 to NO5

Connection of the digital inputs

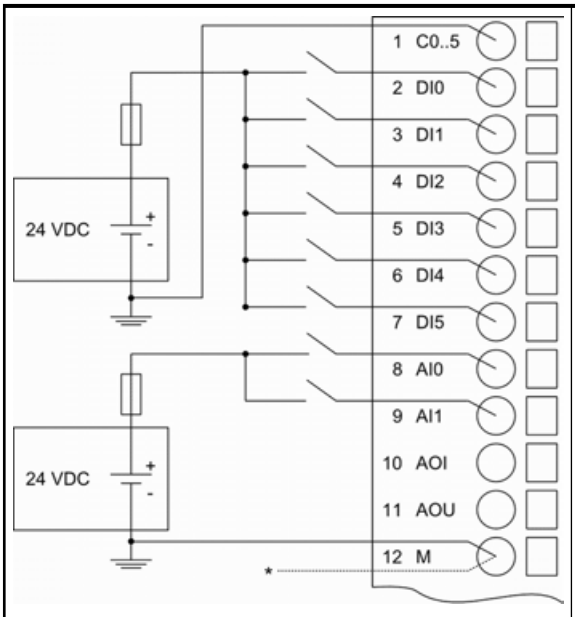
The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

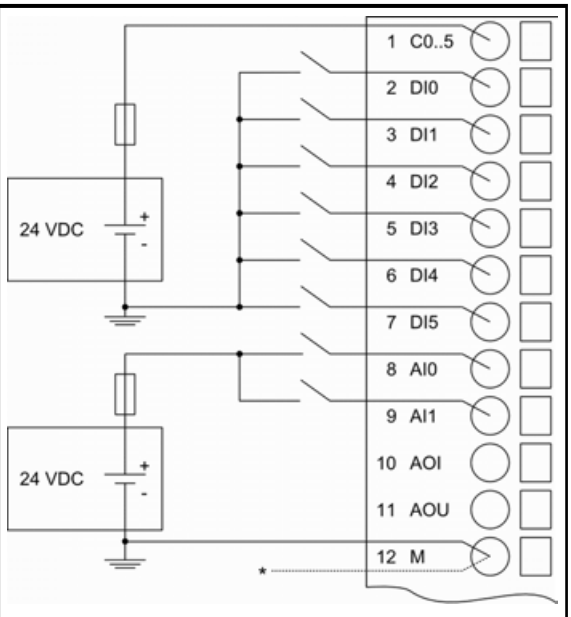
A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.



*) The M-terminal of the onboard I/Os is internally connected to the M-terminal of the processor module supply voltage

Connection of digital inputs (sink inputs)



*) The M-terminal of the Onboard I/Os is internally connected to the M-terminal of the processor module supply voltage

Connection of digital inputs (source inputs)

If the inputs AI0 and AI1 are to be used as digital inputs, they must be configured as digital inputs.

The inputs AI0 and AI1 can only be used as sink inputs.

Connection of the digital transistor outputs (PM56x-T(P) only)

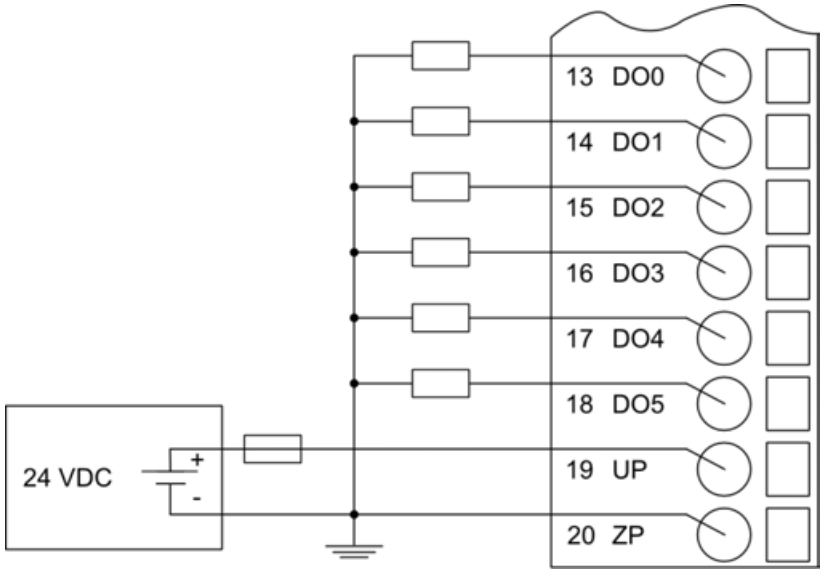


Fig. 731: Connection of digital transistor outputs

NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

CAUTION!

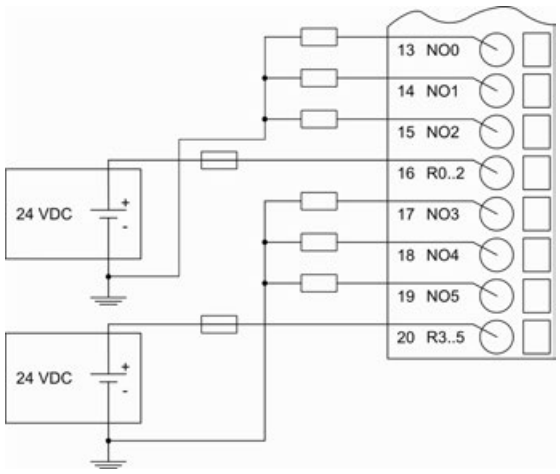
Risk of damaging the processor module!

The outputs are not protected against short circuit and overload.

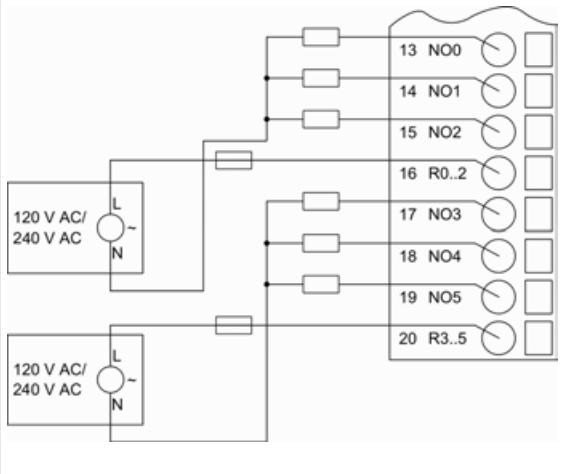
- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast protection fuse for the outputs.

Connection of the digital relay outputs (PM56x-R(P) only)

The following figures show the connection of the digital relay outputs to the processor modules:



Connection of digital relay outputs (24 V DC)



Connection of digital relay outputs (100-240 V AC)

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

CAUTION!

Risk of damaging the processor module!

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be fed from the same phase.
- Use an external 5 A fast acting fuse to protect the outputs.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

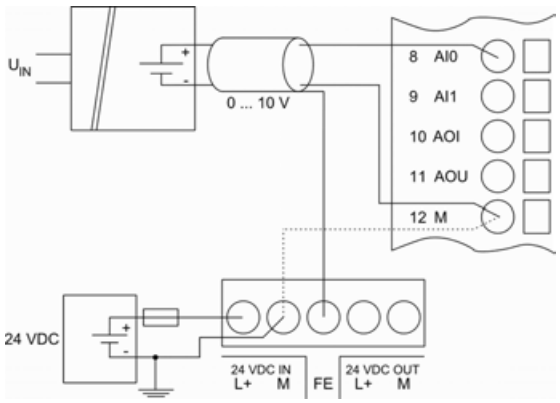
Make sure that the total current of each output common terminal (R0..2 and R3..5) does not exceed 6 A.

Never connect total currents > 6 A per group.

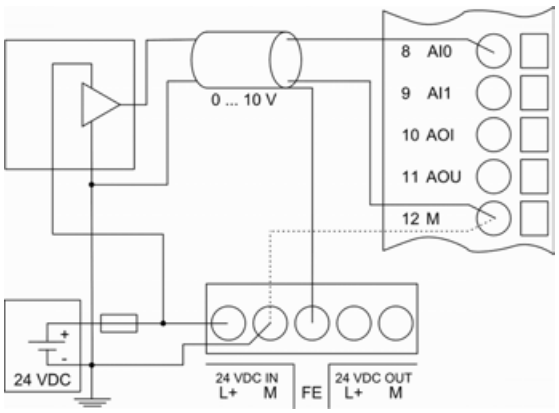
If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

Connection of the analog inputs

The following figures show an example of the connection of analog sensors (voltage) to the input AI0 of PM56x processor modules. Proceed with the input AI1 in the same way:



Connection of active-type analog sensors (voltage)

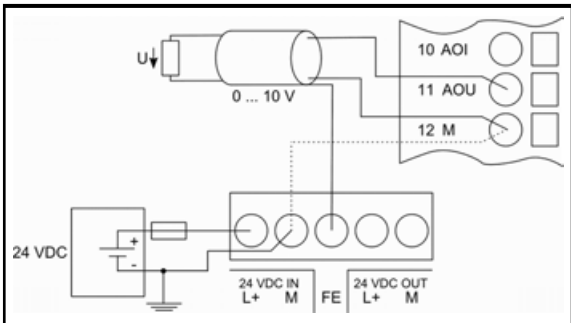


Connection of passive-type analog sensors (voltage)

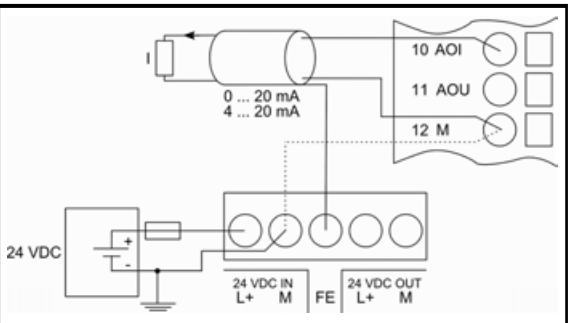
The inputs AI0 and AI1 must be configured as analog inputs.

Connection of the analog output

The following figures show the connection of analog actuators (voltage and current) to the output AO of PM56x processor modules:



Connection of analog actuator (voltage)



Connection of analog actuator (current)

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1
Analog inputs (bytes)	4
Analog outputs (bytes)	2

I/O configuration

The configuration data of the onboard I/Os is stored in the processor module PM56x.

Parameterization

For information about parameterization, refer to the description for onboard I/Os for processor module PM56x. *Chapter 1.6.5.2.3.8 “AC500-eCo onboard I/Os” on page 5852*

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**		**Remedy**
Errors for Onboard I/O								
Minor errors								
3	8	255	2	0	3	MaxWaitRun for onboard I/O module has expired, when PLC is put into RUN state		Reboot and try it again. If the error still exists, replace processor module for testing
3	8	255	3	0	26	Invalid configuration of onboard I/O module, e. g. 2 input channels are configured as fast counter and interrupt input at the same time.		Correct PLC con-figuration
Warnings								
4	8	1	2	1	2	Invalid configuration value for PWM channel. Frequency / cycle time for the PWM channel of the 8DI+6DO and 8DI+6DO+2AI+1AO module are common and if both channel are configured for PWM, the frequency of the second channel must be set to 0.		Correct frequency
4	8	1	2	0..1	4	PWM channel fre-quency or cycle time too high		Correct frequency or cycle time

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**	**Remedy**
4	8	1	2	0..1	7	PWM channel frequency or cycle time too low	Correct frequency or cycle time
4	8	1	2	0	52	Frequency on interrupt input pin too high and interrupt events are missed	Correct frequency
4	8	4	2	0..1	48	Analog input value too high	Correct value
4	8	5	2	0	48	Analog output value too high	Correct value
4	8	255	2	0	26	PLC was put into RUN state, although a configuration error is present, because parameter Run on config fault is set to YES	Correct PLC configuration
4	8	255	0	0	43	Unspecified or internal error occured	Replace processor module

Displays

LED	Status	Color	LED = ON	LED = OFF
DI0...DI5	Digital input	yellow	Input is ON	Input is OFF
DO0...DO5	Digital output	yellow	Output is ON	Output is OFF
AI0, AI1*)	Analog input	yellow	Input is ON	Input is OFF
AO	Analog output	yellow	Output is ON	Output is OFF
*) The analog inputs can be configured as digital inputs				

Measuring ranges

Risk of invalid analog input values!

The analog input values may be invalid if they exceed the measuring range of the inputs.

Make sure that the analog signal at the connection terminals is always within the signal range.

Range	0 V...10 V	Digital value	
		Decimal	Hex.
Overflow	> 11.7589	32767	7FFF
Measured value too high	11.7589 : 10.0004	32511 : 27649	7EFF : 6C01
Normal range	10.0000 : 0.0004	27648 : 1	6C00 : 0001
	0.0000	0	0000

The represented resolution corresponds to 10 bits.

Output ranges

Range	0 V...+10 V	0 mA...20 mA	Digital value	
			Decimal	Hex.
Overflow	11.75 11.75	23.50 23.50	32767 : 32512	7FFF : 7F03
Output value too high	11.75 : 10.01	23.50 : 20.02	32480 : 27680	7EE0 : 6C20
Normal range	10.00 : 0.01	20.00 : 0.02	27648 : 32	6C00 : 20
	0.00	0.00	0	0000
Output value too low or underflow		 0.00	-32 : -6912 : -32768	FFE0 : E500 : 8000

Range	4 mA...20 mA	Digital value	
		Decimal	Hex.
Overflow	22.80 : 22.80	32767 : 32520	7FFF : 7F08
Output value too high	22.80 : 20.02	32480 : 27668	7EE0 : 6C28

Range	4 mA...20 mA	Digital value	
		Decimal	Hex.
Normal range	20.00 : 4.02	27648 : 40	6C00 : 28
	4	0	0
Output value too low or underflow	3.98 : 0.00 : 0.00	-40 : -6920 : -32768	FFD8 : E4F8 : 8000

The represented resolution corresponds to 10 bits.

Technical data

Technical data of the digital inputs

Parameter		Value	
Number of channels per module		8 transistor inputs (24 V DC)	
Distribution of the channels into groups		1 group for 8 channels	
Galvanic isolation		Yes, per group	
Connections of the channels I0 to I7		Terminals 2 to 9	
Reference potential for the channels I0 to I7		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
Signal 0		-5 V...+3 V	-3 V...+5 V
Undefined signal		-15 V...- 5 V	+5 V...+15 V
Signal 1		-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		Within -5 V...+3 V	Within -3 V...+5 V
Ripple with signal 1		Within -30 V...-15 V	Within +15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 0.1 to 32 ms (configurable via software), default: 8 ms	
Max. cable length			

Parameter		Value
	Shielded	500 m
	Unshielded	300 m

Technical data of the fast counter

Parameter	Value
Used inputs for the traces A and B	DI0 / DI1
Used output	DO0 / NO0
Counting frequency	On request

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

Technical data of the digital transistor outputs

Parameter		Value
Number of channels per module		6 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups		1 group of 6 channels
Galvanic isolation		Yes, per group
Connection of the channels DO0 to DO5		Terminals 13 to 18
Common power supply voltage		Terminals 19 (+24 V DC, signal name UP) and 20 (0 V DC, signal name ZP)
Reference potential for the channels DO0 to DO5		Terminal 20 (negative pole of the process voltage, name ZP)
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1)
Way of operation		Non-latching type
Min. output voltage at signal 1		20 V DC at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Rated protection fuse (per group)		3 A fast
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	3 A
	Rated current (all channels together, max.)	3 A
Lamp load (max.)		5 W
Max. leakage current with signal 0		0.5 mA
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz

Parameter		Value
	With lamp loads	Max. 1 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Technical data of the digital relay outputs

Parameter		Value
Number of channels per module		6 normally-open relay outputs
Distribution of the channels into groups		2 groups for 3 channels
Galvanic isolation		Yes, per group
Connection of the channels NO0 to NO2		Terminals 13 to 15
Connection of the channels NO3 to NO5		Terminals 17 to 19
Reference potential for the channels NO0 to NO2		Terminal 16
Reference potential for the channels NO3 to NO5		Terminal 20
Relay output voltage		
	Rated value	24 V DC or 120/240 V AC
	Range	5 to 30 V DC or 5 to 250 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered through the I/O bus
Way of operation		Non-latching type
Output delay		
	0 to 1	Typ. 10 ms
	1 to 0	Typ. 10 ms
Rated protection fuse		5 A
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	6 A
	Rated current (all channels together, max.)	12 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)

Parameter		Value
Demagnetization when inductive loads are switched off		A free-wheeling diode must be circuited in parallel to the inductive load
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	Not possible
	With lamp loads	Max. 1 Hz
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Overload message		No
Output current limitation		No
Resistance to feedback against 24 V DC		No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Technical data of the PWM outputs

Parameter	Value
Used outputs for PWM	O2 and O3
Output frequency	125 Hz 30 kHz

Technical data of the analog inputs

Parameter		Value
Number of channels per module		2 voltage inputs
Distribution of channels into groups		1 group for 2 channels
Galvanic isolation		None
Power Supply Voltage		Via the L+ and the M terminal of the processor module power supply
Resolution		Voltage 0 V DC...10 V DC: 10 bits
Connection of the signals AI0 and AI1		Terminals 8 and 9
Input type		Unipolar
Data word format		
	Unipolar, full-scale range	0 to 27648

Parameter		Value
	Indication of the input signals	No
Channel input resistance		Voltage: > 100 kΩ
Accuracy		
	Typical (25 °C)	±1 %
	Worst case (at 0 °C...60 °C or EMC disturbances)	±2.5 % of full-scale
Time constant of the input filter		Typ. 1 ms
Relationship between input signal and hex code		*Chapter 1.6.2.3.1.3.9 “Measuring ranges” on page 3841*
Analog to digital conversion time		Typ. 6.2 ms
Unused inputs		Can be left open and should be configured as "unused"
Overvoltage protection		Yes, up to 30 V DC
Max. cable length		Conductor cross section > 0.14 mm²
	Unshielded wire	On request
	Shielded wire	100 m

Technical data of the analog output

Parameter	Value
Number of channels per module	1 configurable voltage or current outputs
Distribution of channels into groups	1 group for 1 channel
Galvanic isolation	None
Connection of the signal AOU	Terminal 11
Connection of the signal AOI	Terminal 10
Power supply voltage	Via the L+ and the M terminal of the processor module power supply
Output type	Unipolar (voltage and current)
Resolution	10 bits
Indication of the output signals	Yes, one LED per channel
Output Resistance (load) as current output	0 Ω...500 Ω
Output load ability as voltage output	+2 mA max.
Accuracy for current and voltage output	
Typical (25 °C)	±1 % of full-scale
Worst case (at 0 °C...60 °C or EMC disturbances)	±2.5 % of full-scale
Relationship between input signal and hex code	*Chapter 1.6.2.3.1.3.10 “Output ranges” on page 3842*
Unused inputs	Can be left open and should be configured as "unused"
Overvoltage protection	Yes, up to 30 V DC
Max. cable length	Conductor cross section > 0.14 mm²

Parameter		Value
	Unshielded wire	On request
	Shielded wire	100 m

1.6.2.3.2 AC500 (standard)

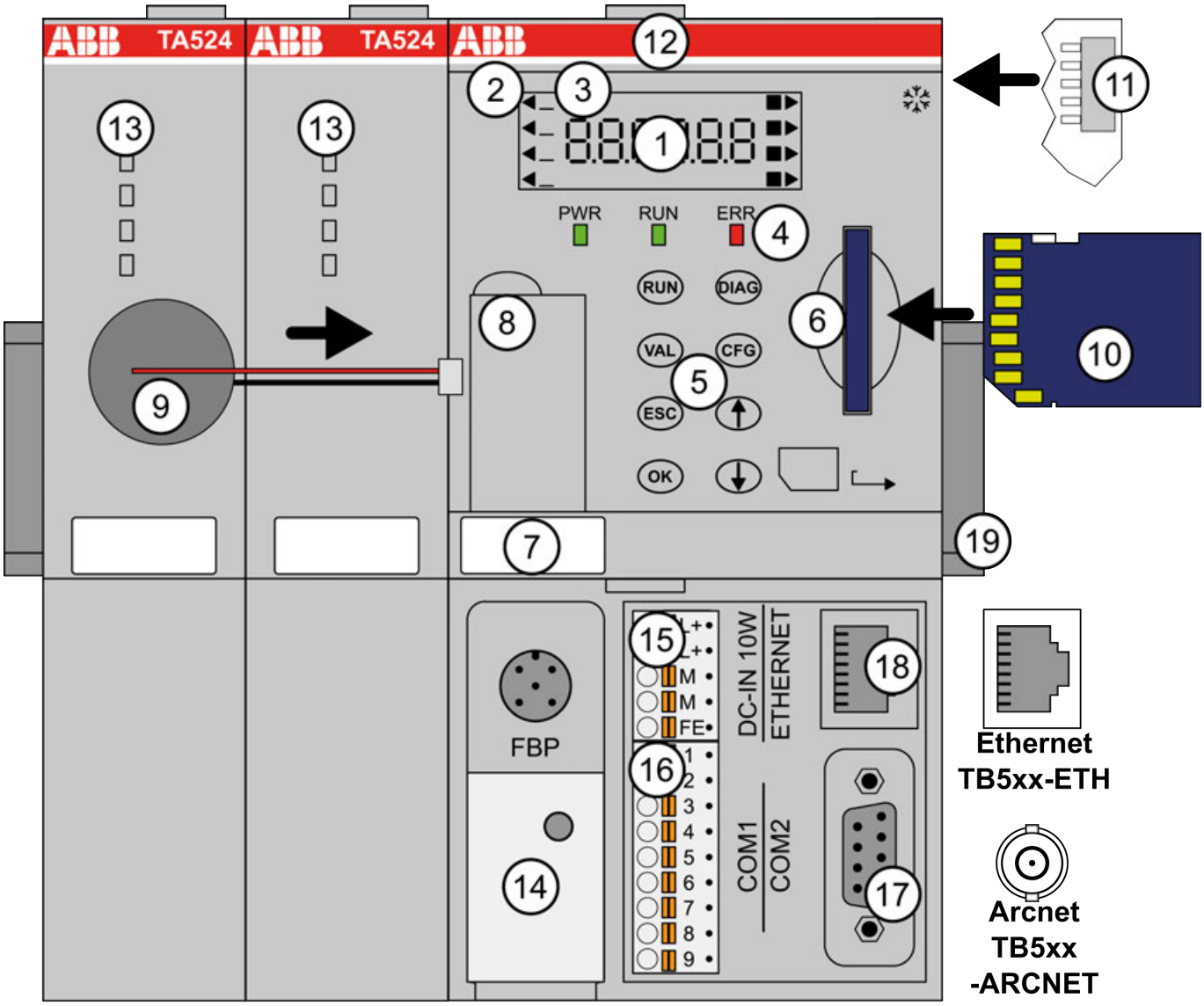
PM57x (-y), PM58x (-y) and PM59x (-y)

Processor modules without onboard interfaces:

- PM57x, PM58x, PM59x: processor module without Ethernet support
- The processor module PM595 is described in a seperate device description *Chapter 1.6.2.3.2.2 “PM595-4ETH” on page 3863*
- XC version for usage in extreme ambient conditions available (some models versions only)

Processor modules with onboard interfaces:

- PM5xy-ETH: processor module with Ethernet support (onboard Ethernet) - 1 network interface RJ45 on the terminal base
- PM5xy-2ETH: processor module with Ethernet support (onboard Ethernet) - 2 network interfaces RJ45 on the terminal base
- PM5xy-ARC: processor module with ARCNET support (onboard ARCNET) - 1 network interface ARCNET BNC on the terminal base



1 6 7-segment state displays with backlight
2 "Triangle" displays for "item"
3 "Square" displays for "state"
4 3 state LEDs
5 8 function keys
6 Slot for memory card
7 Label
8 Compartment for lithium battery TA521
9 Lithium battery TA521
10 Memory card
11 I/O bus for connection of I/O modules
12 Slot for processor module (processor module mounted on terminal base)
13 Slots for communication modules (multiple, depending on terminal base; unused slots must be covered with TA524)
14 Interface for FieldBusPlug
15 Power supply (5-pin terminal block, removable)
16 Serial interface COM1 (9-pin terminal block, removable)
17 PM5xy-ETH and PM5xy-ARCNET: D-sub 9 for serial interface COM2. PM5xy-2ETH: RJ45 female connector for 2nd Ethernet connection
18 RJ45 female connector for Ethernet connection / BNC female connector for ARCNET connection (depending on terminal base)
19 DIN rail
❄ Sign for XC version

Short description

The processor modules are the central units of the control system AC500. The types differ in their performance (memory size, speed etc.). Each processor module must be mounted on a suitable terminal base.

The terminal base type (TB5xx) depends on the number of communication modules which are used together with the processor module and on the processor module's network interface type (1 Ethernet, 2 Ethernet or ARCNET).

Each processor module can operate multiple communication modules through its communication module interface (defined by the terminal base).

The communication modules are mounted on the left side of the processor module on the same terminal base.

On the right side of the processor module, up to 10 digital or analog I/O expansion modules can be connected to the I/O bus. Each I/O module requires a suitable terminal unit depending on the module type.

Terminal bases, terminal units, I/O modules, communication modules and accessories have their own technical descriptions.

Each processor module can be used as:

- Stand-alone processor module
- Stand-alone processor module with local I/Os
- Remote IO server
- Remote IO client

All processor modules V2 (except PM591-2ETH) have a FieldBusPlug interface (FBP).

This interface is no more supported and in limited state.

The processor modules are powered with 24 V DC.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

The processor module PM595 is described in a separate device description
↳ Chapter 1.6.2.3.2.2 “PM595-4ETH” on page 3863.

Assortment

Processor Module	Suitable Terminal Base	Network Interface		Other Interfaces
		Ethernet	ARCNET	
PM572	TB5x1-ETH	-	-	[3])
PM573-ETH	TB5x1-ETH (1SAP11x100R0270 only)	Onboard Ethernet	-	[3])
PM582	TB5x1-ETH	-	-	[3])
PM583-ETH	TB5x1-ETH (1SAP11x100R0270 only)	Onboard Ethernet	-	[3])
PM585-ETH	TB5x1-ETH (1SAP11x100R0270 only)	Onboard Ethernet	-	[3])
PM590-ETH [1])	TB5x1-ETH	Integrated communication module	-	[3])
PM590-ARCNET (R0261)	TB5x1-ARCNET	-	Integrated communication module	[3])
PM591-ETH	TB5x1-ETH	Integrated communication module	-	[3])
PM591-ETH	TB5x1-ETH (1SAP11x100R0270 only)	Onboard Ethernet	-	[3])
PM591-2ETH	TB5x3-2ETH	2x Onboard Ethernet	-	[2])
PM592-ETH	TB5x1-ETH (1SAP11x100R0270 only)	Onboard Ethernet	-	[3])

Remarks:

[1]) The processor modules PM59x-ETH can only be used with terminal bases with product index C6 or higher. Otherwise, they should be updated to that index. ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*

[2]) Serial interface COM1, Communication Interface Module, I/O bus

[3]) Serial interface COM1, Serial interface COM2, Communication Interface Module, FieldBus-Plug (FBP), I/O bus

Connections

All terminals for connection are available on the terminal base. For information on connection and available interfaces see the descriptions for

- ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*.

Processor modules PM57x-ETH, PM58x-ETH and PM59x-ETH with ordering No. 1SAPxxxxxxR0271 can only be used with terminal bases with ordering No. 1SAPxxxxxxR0270.

Processor modules PM5xx-2ETH can only be used with TB5x3-2ETH terminal bases.

Storage elements

Lithium battery

The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.4.1.4.2 “AC500 battery” on page 5419*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

The technical data, handling instructions and the insertion/replacement of the battery is described in detail in the chapter TA521 lithium battery *Chapter 1.6.2.9.2.4 “TA521 - Battery” on page 5175*.

Memory card

AC500 processor modules are supplied without memory card. It must be ordered separately.

The memory card can be used

- to read and write user files
- for firmware updates

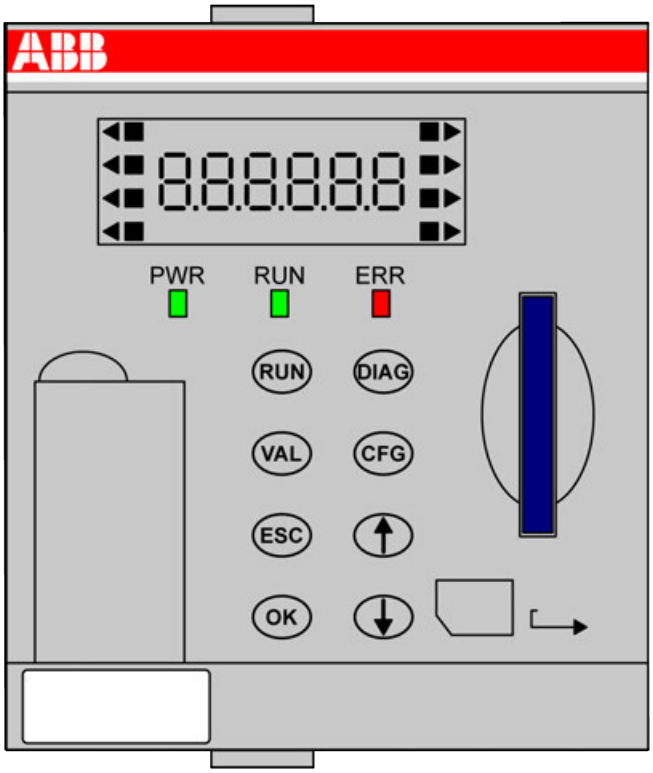
Detailed information can be found in the system technnology chapter. *Chapter 1.6.4.1.2 “System processing” on page 5412*

AC500 processor modules can be operated with and without memory cards. The processor module uses a standard file system (FAT; filenames stored in 8.3 format, on memory card). This allows standard card readers to read and write the memory cards.

Only genuine MC502 memory cards are supported.

For more information on the technical data, handling instructions and the insertion/replacement of the memory card, please refer to the chapter memory card MC502 *Chapter 1.6.2.9.1.2 “MC502 - Memory card” on page 5096*.

LEDs, display and function keys on the front panel



Detailed information on using the LEDs, display and the function keys such as startup procedure and error coding is described in the system technology section ↳ Chapter 1.6.4.1.5 “LEDs, display and function keys on the front panel” on page 5422.

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Processor module and terminal base

Parameter	Value
Connection of the supply voltage 24 V DC at the terminal base of the processor module	Removable 5-pin terminal block with spring connection
Current consumption from 24 V DC	PM57x: 50 mA PM57x-ETH: 110 mA
	PM58x: 50 mA PM58x-ETH: 110 mA PM58x-ARCNET: 110 mA

Parameter	Value
	PM59x: 90 mA PM59x-ETH: 150 mA PM59x-2ETH: 150 mA PM59x-ARCNET: 150 mA
Slots on the terminal bases	TB511: 1 processor module, 1 communication module
	TB521: 1 processor module, 2 communication modules
	TB523: 1 processor module, 2 communication modules
	TB541: 1 processor module, 4 communication modules
Processor module interfaces at the terminal bases TB5x1	I/O bus, COM1, COM2, FBP
Processor module interfaces at the terminal bases TB5x3	I/O bus, COM1
Processor module network interfaces at the terminal bases	TB5x1-ETH *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786* / AC500 CPU with Ethernet interface
	TB5x3-ETH *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*/ AC500CPU with two Ethernet interfaces
	TB5x1-ARCNET *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*/ AC500 CPU with ARCNET
Connection system	See *Chapter 1.6.3.6.4 “Connection and wiring” on page 5337*
Weight (processor module without terminal base)	PM582: 135 g PM58x-ETH: 150 g
	PM59x: 135 g PM59x-ETH: 150 g PM59x-2ETH: 150 g PM59x-ARCNET: 160 g
Mounting position	Horizontal or vertical

Detailed data

Table 273: PM57x

Processor Module		PM572	PM573-ETH
Program memory flash EPROM and RAM		128 kB	512 kB
Data memory, integrated		128 kB, incl. 12 kB buffered	512 kB, incl. 288 kB buffered
Expandable memory		None	None
Integrated mass storage memory		None	None
Pluggable memory card for:			
	User data storage	x	x
	Program storage	x	x
	Firmware update	x	x

Processor Module		**PM572**	**PM573-ETH**
Cycle time for 1 instruction:			
	Binary	Min. 0.06 µs	Min. 0.06 µs
	Word	Min. 0.09 µs	Min. 0.09 µs
	Floating point	Min. 0.70 µs	Min. 0.70 µs
Max. number of central inputs and outputs (up to 7 exp. modules): ([1])			
	Digital inputs	224	224
	Digital outputs	224	224
	Analog inputs	112	112
	Analog outputs	112	112
Max. number of central inputs and outputs (10 exp. modules):			
	Digital inputs	320	320
	Digital outputs	320	320
	Analog inputs	160	160
	Analog outputs	160	160
Number of decentralized inputs and outputs		Depends on the fieldbus used (as an info on the CS31 bus: up to 31 stations with up to 120 DI / 120 DO each)	
Data backup		Battery	
Data buffering time at 25 °C		Typ. 3 years without power supply	
Battery low indication		Warning issued about 2 weeks before the state of charge becomes critical	
Real-time clock:			
	With battery backup	x	x
	Accuracy	Typ. ± 2 s / day at 25 °C	
Program execution:			
	Cyclic	x	x
	Time-controlled	x	x
	Multitasking	x	x
Protection of the user program by a password		x	x
Serial interface COM1:			
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) pluggable terminal block, spring connection for programming, as Modbus (master/slave), as serial ASCI communication, as CS31 Master	
	Connection		
	Usage		
Serial interface COM2 (not for PM5xy-2ETH models):			
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) D-sub for programming, as Modbus (master/slave), as serial ASCII communication	
	Connection		
	Usage		
Integrated communication module:			
	ETH = Ethernet	-	ETH onboard with web server, SNTP and IEC60870-5-104 protocol
	RJ45	-	

Processor Module		PM572	PM573-ETH
	ARCNET = ARCNET BNC	-	
Number of external communication modules		Up to 4 communication modules like PROFIBUS DP, Ethernet, CANopen. There are no restrictions concerning the communication module types and communication module combinations (e.g. up to 4 PROFIBUS DP communication modules are possible)	
Ethernet		-	10/100 base-TX, 1x RJ45 socket, provided on TB5x1-ETH
LEDs, LCD display, 8 function keys		For RUN/STOP switchover, status displays and diagnosis	
Number of timers		Unlimited	
Number of counters		Unlimited	
Programming languages:			
	Structured Text ST	x	x
	Instruction List IL	x	x
	Function Block Diagram FBD	x	x
	Ladder Diagram LD	x	x
	Sequential Function Chart SFC	x	x
	Continuous Function Chart CFC	x	x
[1]): up to 7 I/O terminal units before PS501 V1.2 and processor module firmware before V1.2.0.			

Table 274: PM58x

Processor Module		PM582	PM583-ETH	PM585-ETH
Program memory flash EPROM and RAM		512 kB	1024 kB	1024 kB
Data memory, integrated		416 kB, incl. 288 kB buffered	1024 kB, incl. 288 kB buffered	1536 kB, incl. 512 kB buffered
Expandable memory		None	None	None
Integrated mass storage memory		None	None	None
Pluggable memory card for:				
	User data storage	x	x	x
	Program storage	x	x	x
	Firmware update	x	x	x
Cycle time for 1 instruction:				
	Binary	Min. 0.05 µs		Min. 0.004 µs
	Word	Min. 0.06 µs		Min. 0.008 µs
	Floating point	Min. 0.50 µs		Min. 0.008 µs
Max. number of central inputs and outputs (up to 7 exp. modules): [1])				
	Digital inputs	224		
	Digital outputs	224		
	Analog inputs	112		

Processor Module		**PM582**	**PM583-ETH**	**PM585-ETH**
	Analog outputs	112		
Max. number of central inputs and outputs (10 exp. modules):				
	Digital inputs	320		
	Digital outputs	320		
	Analog inputs	160		
	Analog outputs	160		
Number of decentralized inputs and outputs		Depends on the fieldbus used (as an info on the CS31 bus: up to 31 stations with up to 120 DI / 120 DO each)		
Data backup		Battery		
Data buffering time at 25 °C		Typ. 3 years without power supply		
Battery low indication		Warning issued about 2 weeks before the state of charge becomes critical		
Real-time clock:				
	With battery backup	x		
	Accuracy	Typ. ±2 s / day at 25 °C		
Program execution:				
	Cyclic	x		
	Time-controlled	x		
	Multitasking	x		
Protection of the user program by a password		x		
Serial interface COM1:				
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) pluggable terminal block, spring connection for programming, as Modbus (master/slave), as serial ASCI communication, as CS31 master		
	Connection			
	Usage			
Serial interface COM2 (not for PM5xy-2ETH models):				
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) D-sub for programming, as Modbus (master/slave), as serial ASCII communication		
	Connection			
	Usage			
Integrated communication module:				
	ETH = Ethernet	-	ETH onboard with web server, SNTP and IEC60870-5-104 protocol	
	RJ45	-		
	ARCNET = ARCNET BNC	-		
Number of external communication modules		Up to 4 communication modules like PROFIBUS DP, Ethernet, CANopen. There are no restrictions concerning the communication module types and communication module combinations (e.g. up to 4 PROFIBUS DP communication modules are possible)		
Ethernet		-	10/100 base-TX, 1x RJ45 socket, provided on TB5x1-ETH	

Processor Module		PM582	PM583-ETH	PM585-ETH
LEDs, LCD display, 8 Function Keys		For RUN/STOP switchover, status displays and diagnosis		
Number of timers		Unlimited		
Number of counters		Unlimited		
Programming languages:				
	Structured Text ST	x		
	Instruction List IL	x		
	Function Block Diagram FBD	x		
	Ladder Diagram LD	x		
	Sequential Function Chart SFC	x		
	Continuous Function Chart (CFC)	x		
[1]): up to 7 I/O terminal units before PS501 V1.2 and processor module firmware before V1.2.0.				

Table 275: PM59x [2])

Processor Module		PM59x-ETH	PM59x-ARCNET	PM59x-ETH PM59x-2ETH
Program memory flash EPROM and RAM		PM590: 2048 kB PM591/PM592: 4096 kB		
Data memory, integrated		PM590: 2560 kB, PM591: 3584 kB, incl. 1536 kB buffered		PM590: 3072 kB, PM591/592: 5632 kB, incl. 1536 kB buffered
Expandable memory		None	None	None
Integrated mass storage memory		None	None	PM592-ETH: 4 GB flash disk
Pluggable memory card for:				
	User data storage	x	x	x
	Program storage	x	x	x
	Firmware update	x	x	x
Cycle time for 1 instruction:				
	Binary	Min. 0.002 µs	Min. 0.002 µs	Min. 0.002 µs
	Word	Min. 0.004 µs	Min. 0.004 µs	Min. 0.004 µs
	Floating point	Min. 0.004 µs	Min. 0.004 µs	Min. 0.004 µs
Max. number of central inputs and outputs (up to 7 exp. modules): [1])				
	Digital inputs	224	224	224
	Digital outputs	224	224	224
	Analog inputs	112	112	112
	Analog outputs	112	112	112
Max. number of central inputs and outputs (10 exp. modules):				
	Digital inputs	320	320	320

Processor Module		**PM59x-ETH**	**PM59x-ARCNET**	**PM59x-ETH** **PM59x-2ETH**
	Digital outputs	320	320	320
	Analog inputs	160	160	160
	Analog outputs	160	160	160
Number of decentralized inputs and outputs		Depends on the fieldbus used (as an info on the CS31 bus: up to 31 stations with up to 120 DI / 120 DO each)		
Data backup		Battery		
Data buffering time at 25 °C		Typ. 3 years without power supply		
Battery low indication		Warning issued about 2 weeks before the state of charge becomes critical		
Real-time clock:				
	With battery backup	x	x	x
	Accuracy	Typ. ±2 s / day at 25 °C	Typ. ±2 s / day at 25 °C	Typ. ±2 s / day at 25 °C
Program execution:				
	Cyclic	x	x	x
	Time-controlled	x	x	x
	Multitasking	x	x	x
Password protection of user program		x	x	x
Serial interface COM1:				
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) pluggable terminal block, spring connection for programming, as Modbus (master/slave), as serial ASCII communication, as CS31 master		
	Connection			
	Usage			
Serial interface COM2 (not for PM5xy-2ETH models):				
	Physical link	Configurable for RS-232 or RS-485 (from 0.3 to 187.5 kB/s) D-sub for programming, as Modbus (master/slave), as serial ASCII communication		
	Connection			
	Usage			
Integrated communication module:				
	ETH = Ethernet	ETH	ARCNET	ETH onboard with web server, SNTP and IEC60870-5-104 protocol
	RJ45	ETH	ARCNET	
	ARCNET = ARCNET BNC	ETH	ARCNET	
Number of external communication modules		Up to 4 communication modules like PROFIBUS DP, Ethernet, CANopen. There are no restrictions concerning the communication module types and communication module combinations (e.g. up to 4 PROFIBUS DP communication modules are possible)		

Processor Module		PM59x-ETH	PM59x-ARCNET	PM59x-ETH PM59x-2ETH
Ethernet		10/100 base-TX, 1x RJ45 socket	-	PM59x-ETH: 10/100 base-TX, 1x RJ45 socket, provided on TB5x1-ETH PM591-2ETH: 10/100 base-TX, independent interfaces, 2x RJ45 socket, provided on TB521-2ETH
LEDs, LCD display, 8 Function Keys		For RUN/STOP switchover, status displays and diagnosis		
Number of timers		Unlimited	Unlimited	Unlimited
Number of counters		Unlimited	Unlimited	Unlimited
Programming languages:				
	Structured Text ST	x	x	x
	Instruction List IL	x	x	x
	Function Block Diagram FBD	x	x	x
	Ladder Diagram LD	x	x	x
	Sequential Function Chart SFC	x	x	x
	Continuous Function Chart (CFC)	x	x	x
[1]): up to 7 I/O terminal units before PS501 V1.2 and processor module firmware before V1.2.0.				
[2]): For PM595 see device description for PM595 *Chapter 1.6.2.3.2.2 “PM595-4ETH” on page 3863*.				

Ordering data

Processor modules for AC500 (Standard) V2 products

Part no.	Description	Product life cycle phase *)
1SAP 130 200 R0200	PM572, processor module, memory 128 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display	Active
1SAP 130 300 R0271	PM573-ETH, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active

Part no.	Description	Product life cycle phase *)
1SAP 330 300 R0271	PM573-ETH-XC, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version	Active
1SAP 140 200 R0201	PM582, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display	Active
1SAP 340 200 R0201	PM582-XC, processor module, memory 512 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, XC version	Active
1SAP 140 300 R0271	PM583-ETH, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active
1SAP 340 300 R0271	PM583-ETH-XC, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version	Active
1SAP 140 500 R0271	PM585-ETH, processor module, memory 1024 kB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active
1SAP 150 000 R0261	PM590-ARCNET, processor module, memory 2 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, integrated communication module ARCNET	Active
1SAP 150 000 R0271	PM590-ETH, processor module, memory 2 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active
1SAP 150 100 R0271	PM591-ETH, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active

Part no.	Description	Product life cycle phase *)
1SAP 150 100 R0277	PM591-2ETH, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 1 RS-232/485 (programming, Modbus/CS31), display, 2 onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active
1SAP 350 100 R0271	PM591-ETH-XC, processor module, memory 4 MB, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version	Active
1SAP 150 200 R0271	PM592-ETH, processor module, memory 4 MB / 4 GB flash disk, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols	Active
1SAP 350 200 R0271	PM592-ETH-XC, processor module, memory 4 MB / 4 GB flash disk, 24 V DC, memory card slot, interfaces 2 RS-232/485 (programming, Modbus/CS31), 1 FBP, display, onboard Ethernet TCP/IP with web server, SNTP, IEC60870-5-104 protocols, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 276: Accessories

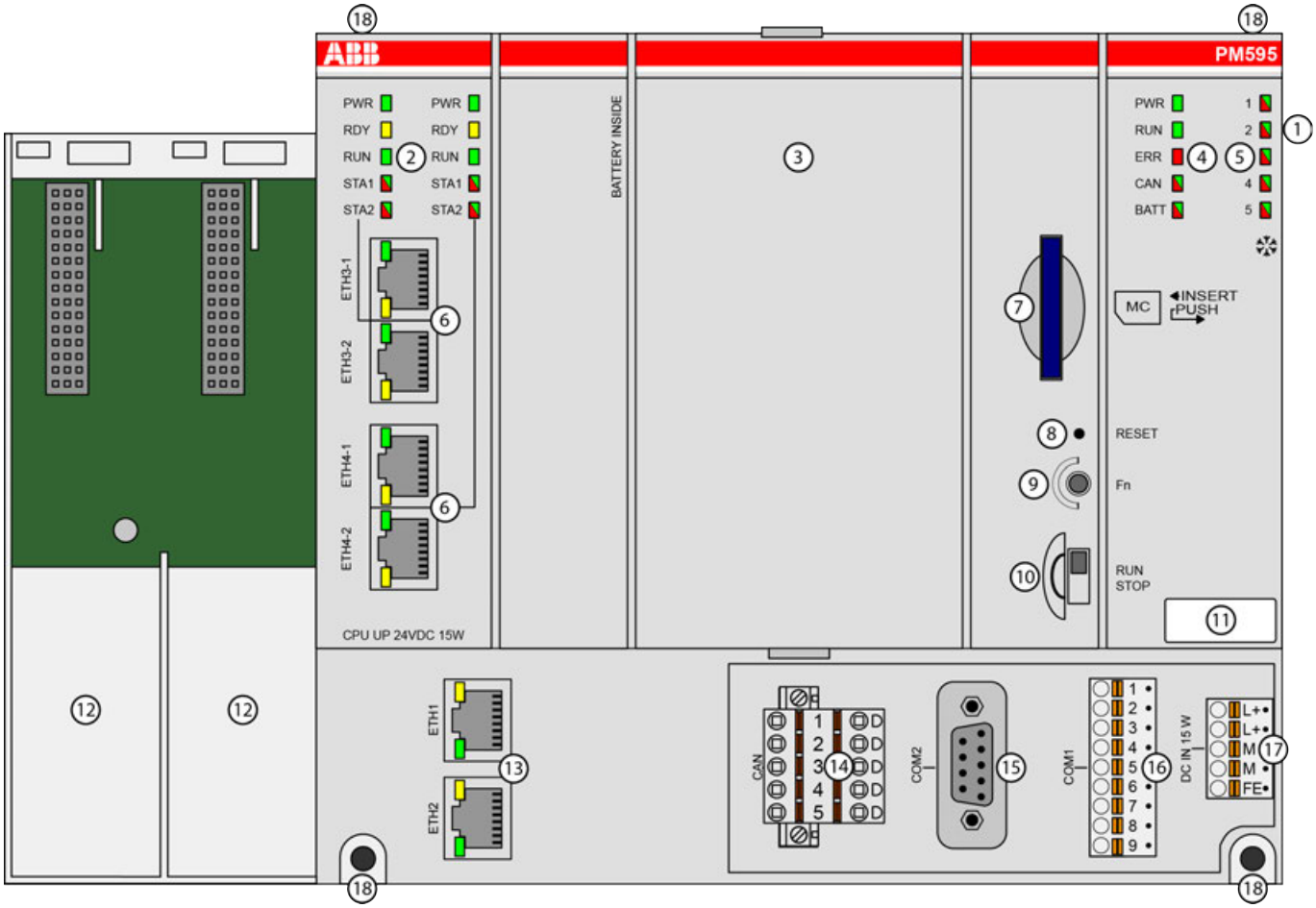
Part no.	Description
1SAP 180 300 R0001	TA521, lithium battery
1SAP 180 100 R0001	MC502, memory card

Processor modules PM57x-ETH(-XC), PM58x-ETH(-XC) and PM59x-ETH(-XC) with ordering No. 1SAPxxxxxxR0271 can only be used with terminal bases TB5x1-ETH(-XC) with ordering No. 1SAPxxxxxxR0270.

Processor module PM591-2ETH can only be used with TB523-2ETH.

PM595-4ETH

- High-performance processor module with 1.3 GHz
- XC version with 1 GHz for use in extreme ambient conditions available (maintenance free)



1 I/O bus for connection of I/O modules
2 2x 5 LEDs to display the states of the fieldbuses
3 Cover for battery and display
4 5 LEDs to display the states of the processor module
5 5 LEDs (reserved)
6 2x2 RJ45 interfaces for fieldbuses
7 Slot for memory card
8 Reset button
9 Button (reserved)
10 RUN/STOP switch
11 Label
12 Slots for communication modules (max 2; unused slots must be covered with TA524)
13 2 RJ45 interfaces for Ethernet connection
14 5-pin terminal block (reserved)
15 Serial interface COM2 (D-sub 9)
16 Serial interface COM1 (9-pin terminal block, removable)
17 Power supply (5-pin terminal block, removable)
18 5 holes for screw mounting
❄ Sign for XC version

Short description

The processor module is a central unit for AC500 with high performance.

Each processor module can operate up to 2 communication modules via its communication module interface. The communication modules are mounted on the left side of the processor module. On the right side of the processor module, up to 10 digital or analog I/O expansion modules can be connected to the I/O bus. Each I/O module requires a suitable terminal unit depending on the module type.

Terminal bases, terminal units, I/O modules, communication modules and accessories have their own technical descriptions.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

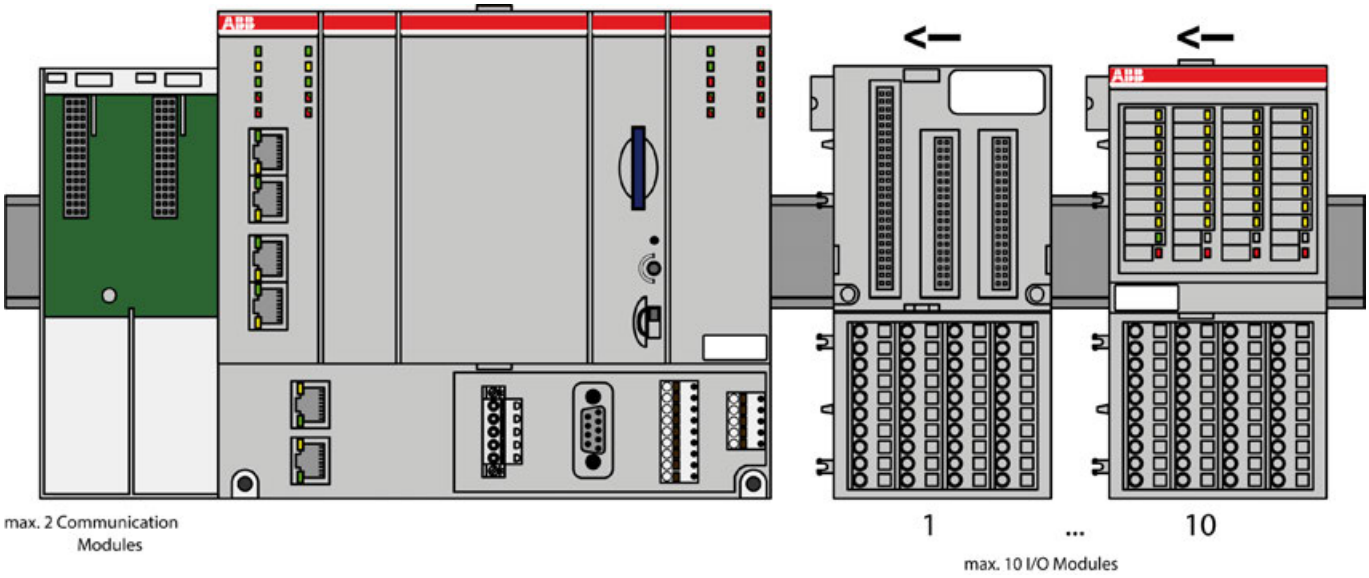


Fig. 732: Processor module, communication modules and I/O modules

For EtherCAT and PROFINET support make sure the following firmware is installed:

– *PROFINET: V 2.8.1.2 or newer*
– *EtherCAT: V 4.2.23 (2) or newer*

To update the Firmware of PM595-4ETH, please follow the instructions in the chapter ↳ Chapter 1.6.5.1.7.4 “Update PM595 firmware” on page 5792.

Assortment

Processor Module	Ethernet Interfaces	Other Interfaces
PM595-4ETH-F PM595-4ETH-M-XC	ETH1 and ETH2 for Ethernet-based system communication ETH3.1 and ETH3.2 for Ethernet-based fieldbuses with switch functionality ETH4.1 and ETH4.2 for Ethernet-based fieldbuses with switch functionality	Serial interface COM1 Serial interface COM2 Communication module interface I/O bus

Connections

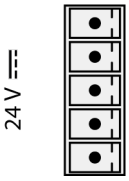
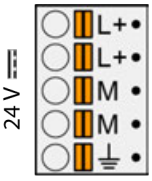
I/O bus

The I/O bus is the I/O data bus for the I/O modules. Through this bus, I/O and diagnosis data are transferred between the processor module and the I/O modules. Up to 10 I/O modules can be added (see description for I/O bus in the system assembly chapter *Chapter 1.6.3.4.1 “Serial I/O bus” on page 5218*).

Power supply

The supply voltage of 24 V DC is connected to a removable 5-pin terminal block. L+/M exist twice. It is therefore possible to feed e.g. external sensors (up to 8 A max. with 1.5 mm^2 conductor) via these terminals.

Pin assignment

Pin Assignment		Label	Function	Description
 Terminal block removed	 Terminal block inserted	L+	+24 V DC	Positive pin of the power supply voltage
		L+	+24 V DC	Positive pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		⏚	FE	Functional earth

Faulty wiring on power supply terminals

NOTICE!

Risk of damaging the processor module and terminal base!

Exceeding the maximum voltage could lead to unrecoverable damage to the system.

The system might be destroyed.

NOTICE!

Risk of damaging the terminal base and power supply!

Short circuits might damage the terminal base and power supply.

Make sure that the four clamps L+ and M (two of each) are not wrongly connected (e. g. +/- of power supply is connected to both L+/L+ or both M/M).

NOTICE!

Risk of damaging the terminal base!

Terminal base can be damaged by connecting the power supply terminal block (L+/M) to COM1.

Make sure that the COM1 terminal block is always connected to the terminal base even if you do not use COM1 to prevent this.

NOTICE!

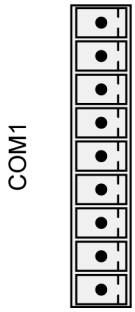
Risk of damaging the terminal base!

Excessive current might damage the clamp and terminal base.

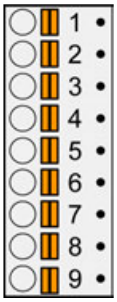
Make sure that the current flowing through the removable clamps never exceeds 8 A (with 1.5 mm^2 conductor).

Serial interface COM1

Pin assignment (RS-485 / RS-232)



Terminal block removed



Terminal block inserted

Pin	Signal	Interface	Description
1	Terminator P	RS-485	Terminator P
2	RxD/TxD-P	RS-485	Receive/Transmit, positive
3	RxD/TxD-N	RS-485	Receive/Transmit, negative
4	Terminator N	RS-485	Terminator N
5	RTS	RS-232	Request to send (output)
6	TxD	RS-232	Transmit data (output)
7	SGND	Signal Ground	Signal Ground
8	RxD	RS-232	Receive data (input)
9	CTS	RS-232	Clear to send (input)

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

The serial interface COM1 is connected to a removable 9-pin terminal block. It is configurable for RS-232 and RS-485.

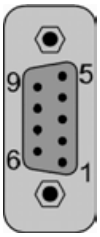
For a detailed description of COM1, please refer to Serial interface COM1 *Chapter 1.6.3.6.4.6 “Serial interface COM1 of the terminal bases” on page 5343*.

Serial interface COM2

The serial interface COM2 is connected to a D-sub 9. It is configurable for RS-232 and RS-485.

COM2 cannot be used for communication via CS31 bus. For a detailed description of COM2, please refer to Serial interface COM2. *Chapter 1.6.3.6.4.7 “Serial interface COM2 of the terminal bases ” on page 5345*

Pin assignment

Serial Interface	Pin	Signal	Interface	Description	
	1	FE	-	Functional earth	
	2	TxD	RS-232	Transmit data	Output
	3	RxD/TxD-P	RS-485	Receive/Transmit	Positive
	4	RTS	RS-232	Request to send	Output
	5	SGND	Signal ground	0 V supply out	
	6	+5 V	-	5 V supply out	
	7	RxD	RS-232	Receive data	Input
	8	RxD/TxD-N	RS-485	Receive/Transmit	Negative
	9	CTS	RS-232	Clear to send	Input
	Shield	FE	-	Functional earth	

NOTICE!

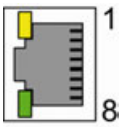
Risk of corrosion!

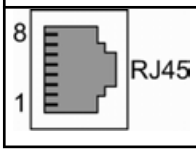
Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Network interfaces Ethernet (ETHx)

Pin assignment

Interface	Pin	Signal	Description
 or	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -

Interface	Pin	Signal	Description
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

See supported protocols and used Ethernet ports for AC500 V2products: *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.9 “Communication with Modbus TCP/IP” on page 5488*.

See communication via Modbus for AC500 V2 products: *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

MAC addresses

The MAC addresses of the network interfaces of the PM595-4ETH are printed on the label in the following way:

MAC ETH1

MAC ETH2

MAC ETH3

MAC ETH4

The figure below shows the assignment of the MAC addresses to the corresponding interface.

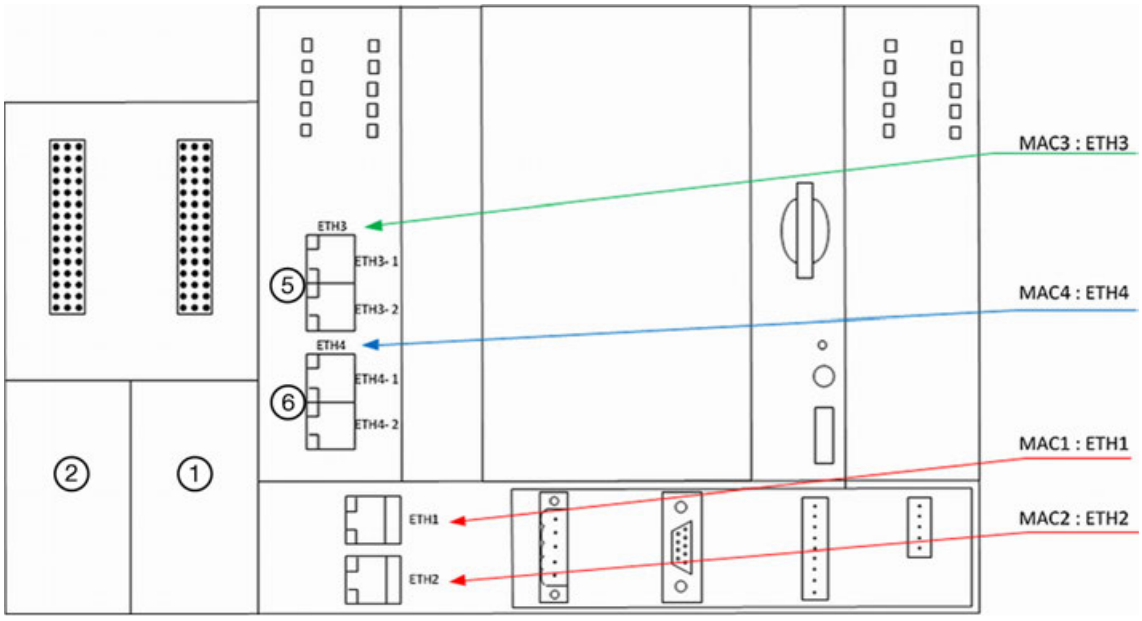


Fig. 733: Assignment of the MAC addresses to the corresponding interface

The figure above also shows the assigned SLOT-Numbers 1, 2, 5 and 6.

Storage elements

Lithium battery

AC500 processor modules are supplied without a lithium battery. It must therefore be ordered separately. The TA541 lithium battery is used to save SRAM contents of processor modules (PM595-4ETH-F only) and back up the real-time clock in case of power failures. Even if the processor modules can work without a battery, its use is still recommended in order to prevent process data being lost in case of power failures (PM595-4ETH-F only).

The processor module monitors the battery's state of charge. If the processor module signals a low state of charge (via the diagnosis system and LED), the battery has to be replaced immediately.

For technical data, handling instructions and a description of the insertion/replacement of the battery, please refer to the chapter TA541 Lithium Battery ↳ *Chapter 1.6.2.9.2.7 “TA541 - Battery” on page 5180*.

The processor module PM595-4ETH-M-XC is maintenance-free. The lithium battery TA541 in this processor module type is used only for back-up of the real-time clock (RTC) in case of no power supply. If the RTC is not used, there is no need to install a TA541 lithium battery.

Memory card

AC500 processor modules are supplied without memory card. It must be ordered separately.

The memory card can be used

- to read and write user files
- for firmware updates

Detailed information can be found in the system technnology chapter. ↳ *Chapter 1.6.4.1.2 “System processing” on page 5412*

AC500 processor modules can be operated with and without memory cards. The processor module uses a standard file system (FAT; filenames stored in 8.3 format, on memory card). This allows standard card readers to read and write the memory cards.

Only genuine MC502 memory cards are supported.

For more information on the technical data, handling instructions and the insertion/replacement of the memory card, please refer to the chapter memory card MC502 ↳ *Chapter 1.6.2.9.1.2 “MC502 - Memory card” on page 5096*.

Operating elements on the front panel

Status LEDs

Table 277: Meaning of the status LEDs (left part)

LED	Color	Status	Description
PWR *)	Green	On	Power supply available
		Blinking	---
		Off	Power supply not available or defective hardware
RDY *)	Yellow	On	Boot procedure
		Blinking	Boot failure
		Off	---
RUN *)	Green	On	Communication module is operational
		Blinking	---

LED	Color	Status	Description
		Off	Communication module is not operational
STA1 *)	Red	On	Depending on used fieldbus
		Blinking	Depending on used fieldbus
		Off	Depending on used fieldbus
	Green	On	Depending on used fieldbus
		Blinking	Depending on used fieldbus
		Off	Depending on used fieldbus
STA2 *)	Red	On	Depending on used fieldbus
		Blinking	Depending on used fieldbus
		Off	Depending on used fieldbus
	Green	On	Depending on used fieldbus
		Blinking	Depending on used fieldbus
		Off	Depending on used fieldbus

*) These LEDs exist twice.

LED	Color	Status	Description
PWR	Green	On	Power supply available
		Blinking	---
		Off	Power supply not available or defective hardware
RUN	Green	On	Processor module is in RUN mode
		Blinking	---
		Off	Processor module is in STOP mode
ERR	Red/green	On	An error has occurred
		Blinking	Flashing fast (4 Hz): Indicates together with RUN a firmware update process and a flash EEPROM write.
		Off	No errors are encountered or only warnings (E4 errors). This is configurable (for errors 2 - 4, the LED behavior is configurable.
-	Red/green	On	Reserved
		Blinking	Reserved
		Off	Reserved
Batt	Red/green	On	TA541 lithium battery is not installed or is weak
		Blinking	---
		Off	TA541 lithium battery is installed and has sufficient capacity
1	Red/green	On	Reserved
		Blinking	Reserved
		Off	Reserved
2	Red/green	On	Reserved

LED	Color	Status	Description
		Blinking	Reserved
		Off	Reserved
3	Red/green	On	Reserved
		Blinking	Reserved
		Off	Reserved
4	Red/green	On	Reserved
		Blinking	Reserved
		Off	Reserved
5	Red/green	On	Reserved
		Blinking	Reserved
		Off	Reserved

Buttons and switches

The processor module can be operated manually using the buttons and switches at the front panel. Meaning of the buttons and switches:

Button	Description
RESET	If pressed during power-on: Enter serial download of firmware. This is signalized by blinking of the RUN LED with a frequency of 1 Hz. If pressed during normal operation: reserved for future implementation.
Fn	If pressed during power-on: Boot project will not be loaded. This is signalized by blinking of the RUN LED with a frequency of 1 Hz. If pressed during normal operation: reserved for future implementation.
RUN/STOP	Switches the processor module from RUN to STOP mode.

The AC500 processor module can display various errors according to the error classes. The reaction of the Processor Module is different for each type of error. See System Technology ↳ *Chapter 1.6.4.1.5 “LEDs, display and function keys on the front panel” on page 5422*.

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

General data of the processor modules

Parameter		Value
Connection of the supply voltage 24 V DC at the removable terminal block of the processor module		at a removable 5-pin terminal block with spring connection
	Current consumption from 24 V DC	0.4 A
	Inrush current at 24 V DC	1 A^2s ∗)
	Max. power dissipation within the module	15 W

Parameter		Value
	Slots for communication modules	2
	Processing module's interfaces	I/O bus, COM1, COM2
	Processing module's network interfaces	ETH1 and ETH2 for Ethernet-based system communication ETH3.1 and ETH3.2 for Ethernet-based fieldbuses with switch functionality ETH4.1 and ETH4.2 for Ethernet-based fieldbuses with switch functionality
Connection system		see *Chapter 1.6.3.6.4 “Connection and wiring” on page 5337*
Weight		1070 g
Mounting position		horizontal or vertical with derating (50 % output load, reduction of temperature to 40 °C)

∗) The melting integral of the processor module depends on the processor module's integrated power supply, and the number and type of communication modules and I/O modules connected to the I/O bus.

Detailed data

Parameter		Value
Flash memory for boot projects, symbols and web pages		32768 kB
SDRAM for user program		16384 kB
SDRAM for user data		16384 kB
Expandable memory		None
Integrated mass storage memory		4 GB non rotating flashdisk
Pluggable memory card for:		x
	User data storage	
	Program source code storage	
	Firmware update	
Cycle time for 1 instruction		
	Binary	Min. 0.0006 µs
	Word	Min. 0.001 µs
	Floating point	Min. 0.001 µs
Max. number of central inputs and outputs (10 exp. modules):		
	Digital inputs	320
	Digital outputs	240
	Analog inputs	160
	Analog outputs	160
Number of decentralized inputs and outputs		Depends on the field bus used (as an info on the CS31 bus: up to 31 stations with up to 120 DI / 120 DO each)

Parameter	Value
Data backup	Battery for PM595-4ETH-F, MRAM for PM595-4ETH-M-XC without battery
Data buffering time at 25 °C	About 3 years
Battery low indication	Warning issued about 2 weeks before the state of charge becomes critical
Real-time clock	
With battery backup	x
Accuracy	Typ. ± 2 s / day at 25° C
Integrated Communication Module, ETH = Ethernet RJ45	2x Ethernet, 2x Ethernet interfaces with downloadable protocol e.g. PROFINET IO, EtherCAT (in preparation)
Number of external communication modules	Up to 2 communication modules like PROFIBUS DP, Ethernet, CANopen or functional safety module, e.g., SM560-S. There are no restrictions concerning the communication module types and communication module combinations (e.g. up to 2 PROFIBUS DP communication modules are possible)
LEDs	5 to display states, rest of LEDs reserved
LCD display	Optional
Buttons and switches	1 button for Reset (Reserved) 1 Button (Reserved) 1 Switch for RUN/STOP

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 155 500 R0279	PM595-4ETH-F, processor module, user progr./data memory 16 MB / 16 MB, 1.3 GHz, 24 V DC, memory card slot, interfaces 2 RS232-485, 2 independent Ethernet interfaces (progr., web server, IEC60870-5-104 protocols), 2 independent Ethernet based interfaces with 2-port switch (between fieldbus protocols PROFINET IO, EtherCAT and Ethernet)	Active
1SAP 351 500 R0279	PM595-4ETH-M-XC, processor module, user progr./data memory 16 MB / 16 MB, 1.0 GHz, 24 V DC, memory card slot, interfaces 2 RS232-485, 2 independent Ethernet interfaces (progr., web server, IEC60870-5-104 protocols), 2 independent Ethernet based interfaces with 2-port switch (between fieldbus protocols PROFINET IO, EtherCAT and Ethernet), XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Table 278: Accessories

Part no.	Description
1SAP 182 700 R0001	TA541, lithium battery
1SAP 180 100 R0001	MC502, memory card
1SAP 180 200 R0001	TK501, programming cable D-sub / D-sub, length: 5 m
1SAP 180 200 R0101	TK502, programming cable terminal block / D-sub, length: 5 m
1TNE 968 901 R1100	TK503, COM1 USB programming cable / D-sub (RS-485), length 3 m
1SAP 182 300 R0001	TA535, protective caps for XC devices
1SAP 182 600 R0001	TA540, front cover as spare part (3 pieces)
1SAP 182 800 R0001	TA543, screw mounting accessory (20 pieces)

1.6.2.3.3 AC31 adapters

Introduction

Replacement devices for AC31

The modular product line of the AC31 adapter series includes modular exchange components for control systems of the Advant Controller 31 (90 series). The simple exchange of individual components allows existing customers to maintain their PLCs in a quick and cost-effective manner. Extensive software modifications are not required.

Each replacement device is based on trend setting technologies of the AC500 series. Therefore, by exchanging components it is not only possible to replace the existing device, but also to profit from new functions and improved product quality.

Note regarding product documentation

During the development of the AC31 adapter series, care was taken to keep the device configuration identical to the configuration of the AC31 devices. Consequently, the technical documents for the AC31 devices are still valid and serve as reference:

- *Software description (only available in English)*
- *System description Advant Controller 31*

Only unavoidable deviations, for example due to technical limitations, are described in this document.

CAUTION!
Installation and maintenance work on the device must be performed by qualified personnel in line with the recognized technical rules, regulations and relevant standards such as EN 60204-1.

For safety instructions, please refer to Regulations for the erection of installations.

Overview of AC31 adapters (replacement devices)

An AC31 adapter (replacement device) is available for the following AC31 devices of the 90 series (existing devices):

Existing devices: AC31 (90 series)	Replacement devices: AC31 adapters	Replacement device is based on the following AC500 device
CPU devices:		
07KT94-ARC	07KT94-ARC-AD *)	PM590, DA501 and DA502
07KT98-ARC	07KT98-ARC-AD	
07KT98-ARC-DP	07KT98-ARC-DP-AD	
07KT98-ARC-ETH	07KT98-ARC-ETH-AD	
07KT98-ETH-DP	07KT98-ETH-DP-AD	
--	07KT98-ARC-ETH-DP-AD	

*) Customer specific product not available for current sales

Existing devices: AC31 (90 series)	Replacement devices: AC31 adapters	Replacement device is based on the following AC500 device
I/O modules:		
07DC91	07DC91-AD	DC532
07DC92	07DC92-AD	DO524
07AC91	07AC91-AD (8-Bit)	AO523
07AC91	07AC91-AD2 (12-Bit)	AX522
07AI91	07AI91-AD	AI523
DC501-CS31	DC501-CS31-AD	DC532

System data and CS31 bus system data

The system data described in this chapter are valid for the following replacement devices:

- 07KT94-ARC-AD
- 07KT98-ARC-AD
- 07KT98-ARC-DP-AD
- 07KT98-ARC-ETH-AD
- 07KT98-ETH-DP-AD
- 07KT98-ARC-ETH-DP-AD
- 07AC91-AD
- 07AC91-AD2
- 07AI91-AD
- 07DC91-AD
- 07DC92-AD
- DC501-CS31-AD

Please also observe the CS31 bus system data *Chapter 1.6.2.3.3.3.2 “CS31 bus system data” on page 3882*.

The devices of the AC31 adapter series do not have marine approval.

NOTICE!
AC31 adapter I/O modules must only be used with an ABB CPU with master CS31 bus (e.g. AC31 07KT9x, AC31-Adpater 07KT9x-x-x-AD or AC500 CPU).

System data of the AC31 adapters

Operating and environmental conditions

Table 279: Supply voltages

Voltages according to IEC 61131-2:		
24 V DC	Process and supply voltage	24 V DC (-15 %, +20 % without residual ripple)
	Absolute limits	19.2 V ... 30 V incl. residual ripple
	Residual ripple	≤ 5 %
	Polarity reversal protection	10 s (test duration), permanently present on AC31 adapters
Bridging time for power interruptions according to IEC 61131-2:		
	DC supply	Interruption < 10 ms Time between 2 interruptions > 1 s

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Table 280: Operating and environmental conditions

Temperature:	
-> Operation	0 °C ... +55 °C (vertical mounting position, terminals upward and downward)
-> Storage	-40 °C ... +75 °C
-> Transport	-40 °C ... +75 °C
Humidity	max. 95 %, without condensation
Air pressure:	
-> Operation	> 800 hPa / < 2000 m
-> Storage	> 660 hPa / < 3500 m

Creepage distances and clearances

The creepage distances and clearances correspond to overvoltage category II, pollution degree 2.

Test voltages for type test

Test voltages for type test according to IEC 61131-2:

Table 281: Impulse testing

Data	Voltage	Duration
24 V circuits (supply, 24 V inputs/outputs), when galvanically isolated from other circuitry	500 V	1.2 / 50 µs
CS31 interface from other circuitry	500 V	1.2 / 50 µs
Ethernet	500 V	1.2 / 50 µs
ARCNET	500 V	1.2 / 50 µs
COM interfaces, galvanically isolated	500 V	1.2 / 50 µs
Enabling input, galvanically isolated	500 V	1.2 / 50 µs

Table 282: AC voltage tests

Data	Voltage	Duration
24 V circuits (supply, 24 V inputs/outputs), when galvanically isolated from other circuitry	350 V AC	60 s
CS31 interface from other circuitry	350 V AC	60 s
Ethernet	350 V AC	60 s
ARCNET	350 V AC	60 s
COM interfaces, galvanically isolated	350 V AC	60 s
Enabling input, galvanically isolated	350 V AC	60 s

Power supply units

For the supply of devices, use power supply units according to PELV specification.

Electromagnetic compatibility

Table 283: Immunity

Data	Value
Immunity against electrostatic discharge (ESD)	According to EN 61000-4-2, zone B, criterion B
-> Interference voltage with air discharge	8 kV
-> Interference voltage with contact discharge	4 kV
ESD with communication connectors	Ensure that any electrostatic charge is discharged prior to contact with the communication connectors (e.g. by touching an grounded metal object). Otherwise malfunctions may occur.
ESD module carrier connectors	Do not touch the plug connecting the module carrier on the bottom side of the device.
ESD external communication module interface	Do not touch the plug to the flat ribbon cable.
Immunity against the influence of radiated interference (CW radiated)	According to EN 61000-4-3, zone B, criterion A
-> Test field strength	10 V/m (except ITU transmission bands 87…108 MHz, 174…230 MHz and 470…790 MHz -> 3 V/m)
-> Maximum temporary deviation during irradiation	Analog current output signals max. 1.5 %. Devices affected: 07AC91-AD, 07AC91-AD2, 07KT94-ARC-AD, 07KT98-ARC-AD, 07KT98-ARC-DP-AD, 07KT98-ARC-ETH-AD, 07KT98-ETH-DP-AD, 07KT98-ARC-ETH-DP-AD
Immunity against transient interference voltages (burst)	According to EN 61000-4-4, zone B, criterion B
-> Voltage supply	2 kV
-> Enabling input	2 kV
-> Digital inputs/outputs	1 kV
-> Analog inputs/outputs	1 kV
-> CS31 bus	1 kV
-> Serial RS-232 interfaces (COM)	1 kV
-> ARCNET	1 kV
-> Ethernet	1 kV
-> I/O supply, DC out	1 kV
Immunity against the influence of power related interference (CW radiated):	According to EN 61000-4-6, zone B, criterion A
-> Test voltage	Zone B, also according to 10 V
Immunity against transient interference voltages with high energy (surge)	According to EN 61000-4-5, zone B, criterion B
-> Voltage supply DC, enabling input	0.5 kV CM / 0.5 kV DM *)
-> I/O supply, DC out	0.5 kV CM / 0.5 kV DM *)
-> Shielded buses	1 kV CM *)

Data	Value
-> I/O analog, I/O DC unshielded	1 kV CM / 0.5 kV DM *)
Emitted interference (radiation):	-
-> From radiated interferences	According to EN 55011, group 1, class A

*) CM = Common Mode, DM = Differential Mode

The devices of the AC31 adapter series do not have marine approval.

Mechanical data

Data	Value
Degree of protection	IP20
Housing	According to UL 94
Vibration resistance according to EN 61131-2	All three axes 2 Hz ... 15 Hz, continuous 3.5 mm 15 Hz ... 150 Hz, continuous 1 g
Vibration resistance with memory card plugged	15 Hz ... 150 Hz, continuous 1 g
Shock resistance	All three axes 15 g, 11 ms, semi-sinusoidal

Grounding

The AC31 adapter devices can be grounded as follows:

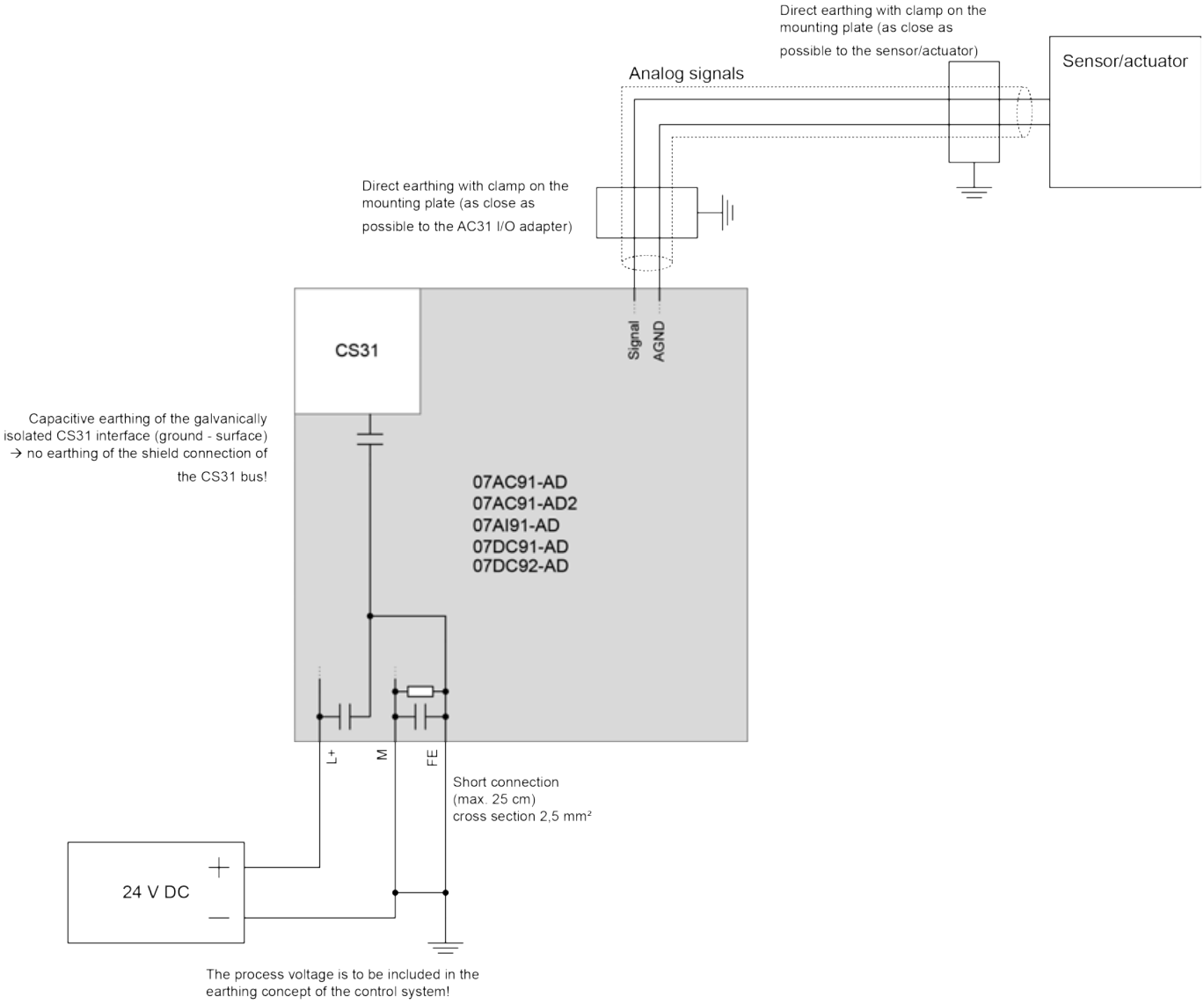


Fig. 734: Grounding of devices 07AC91-AD, 07AC91-AD2, 07AI91-AD, 07DC91-AD and 07DC92-AD

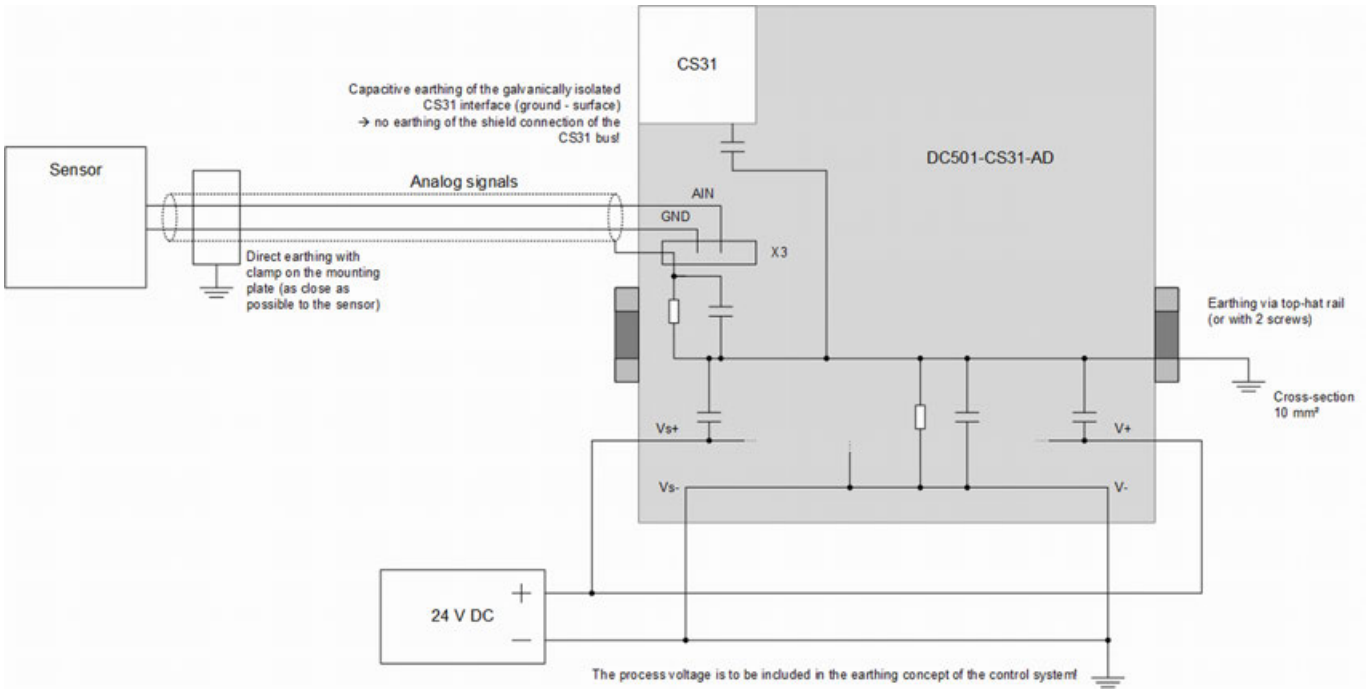


Fig. 735: Grounding of device DC501-CS31-AD

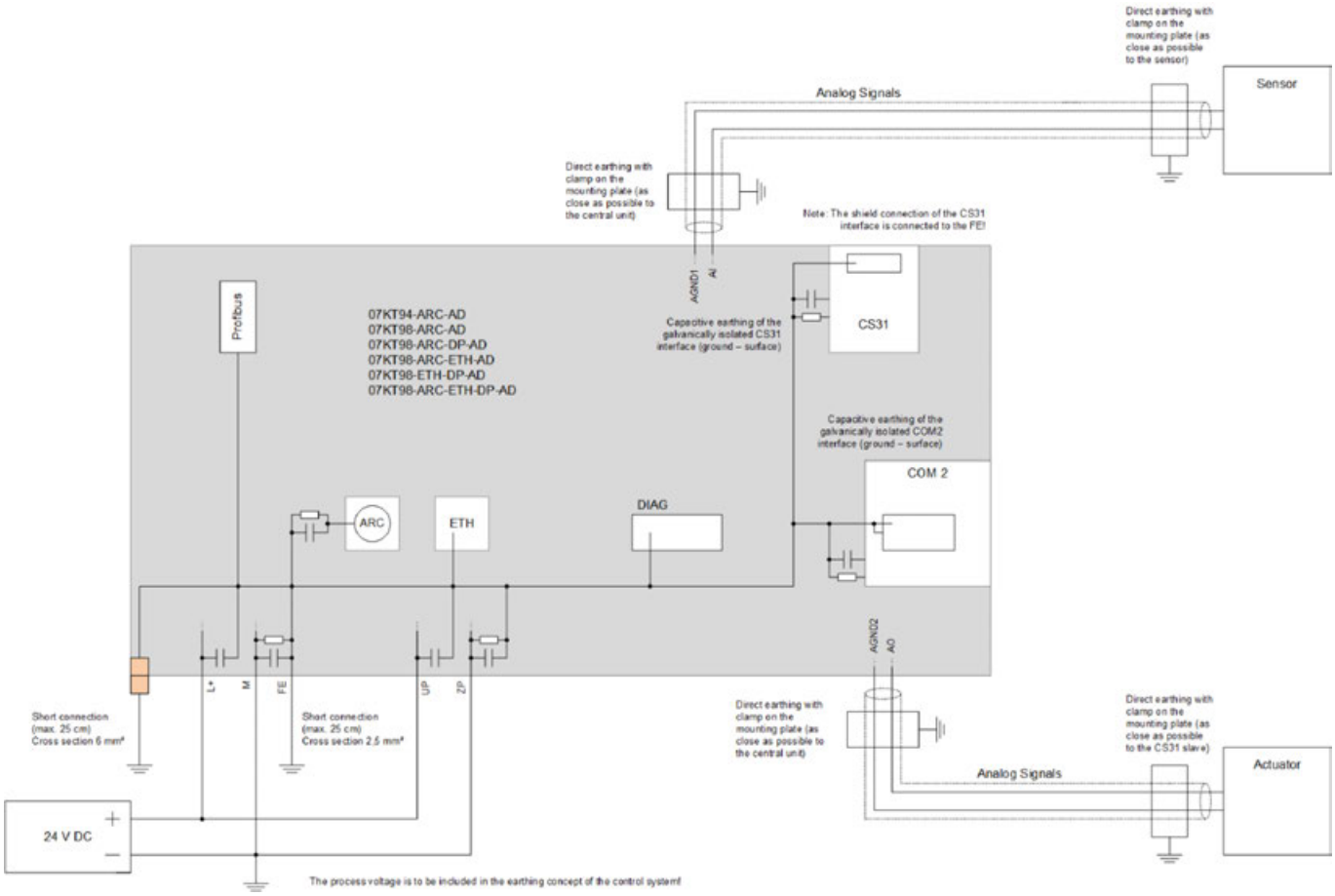


Fig. 736: CPU grounding

When grounding the replacement devices, observe the following:

- Install the AC31 adapter devices onto an grounded mounting plate to ensure a uniform reference potential of all equipment.
- Implement the connections between switchgear cabinet, mounting plate, PE rail and shield rail with low impedance.

- Install the lines in groups (power lines, power supply lines, signal lines, data lines).
- Use lines with braided cable shield for analog signals. Ground the shield on both sides and make sure that no compensation currents flow through the cable shield. For this purpose, use a potential equalization line with current carrying capacity, for instance on systems consisting of several switchgear cabinets.

Further information concerning CS31 bus grounding: ↳ *Chapter 1.6.2.3.3.3.2.3 “Grounding” on page 3884*

CS31 bus system data

Wiring

Table 284: Bus line

Data	Value
Configuration	2 cores, twisted, with common shield
Cross section	> 0.22 mm² (24 AWG) Recommendation: 0.5 mm² corresponds to Ø 0.8 mm
Twist rate	> 10/m (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	approx. 120 Ω (100 ... 150 Ω)
Capacitance between the cores	< 55 nF/km (in case of higher capacitance values, the maximum possible bus length is reduced)
Terminating resistors	120 Ω ¼ W at both ends
Notes	Cables with PVC core insulation and core diameter of 0.8 mm can be used up to a length of approx. 250 m. In this case, the terminating resistor is 100 Ω. Cables with PE core insulation can be used up to a length of approx. 500 m.

The transmission rate used on the CS31 bus is 187.5 kBaud.

Bus topology

A CS31 bus always contains only one CS31 bus master to control the bus. Up to 31 CS31 slaves can be controlled by one bus. The CS31 bus master has no address, whereas the CS31 slaves can accept addresses in the range from 0 - 61, depending on CS31 slave type.

Possible CS31 bus masters:

- 07KT94-ARC-AD, 07KT94
- 07KT98-ARC-AD, 07KT98
- 07KT98-ARC-DP-AD
- 07KT98-ARC-ETH-AD
- 07KT98-ETH-DP-AD
- 07KT98-ARC-ETH-DP-AD

Possible CS31 slaves:

- 07AC91-AD, 07AC91
- 07AC91-AD2
- 07AI91-AD, 07AI91
- 07DC91-AD, 07DC91
- 07DC92-AD, 07DC92
- DC501-CS31-AD, DC501-CS31

The following diagram shows the bus topology without shielding and grounding treatment:

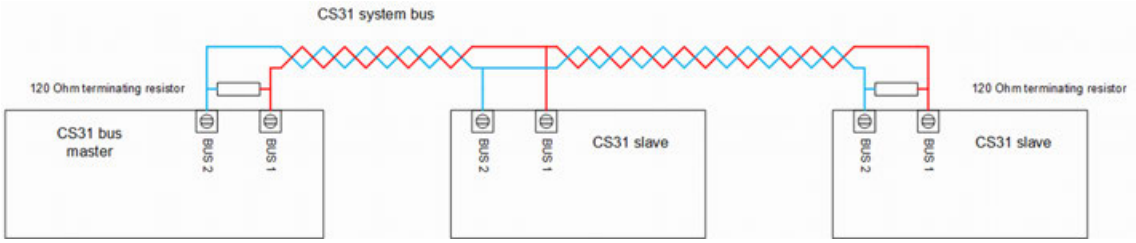


Fig. 737: Bus topology with CS31 bus master on the side

The CS31 slave DC501-CS31-AD has an internal 120 Ω terminating resistor which can be connected by using a DIP switch. On the other CS31 slaves and the CS31 bus master, the terminating resistor must be installed externally by the user.

The following diagram shows the bus topology without shielding and grounding treatment:

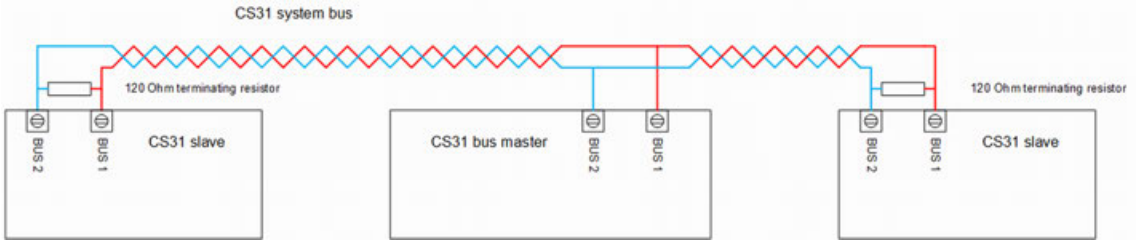
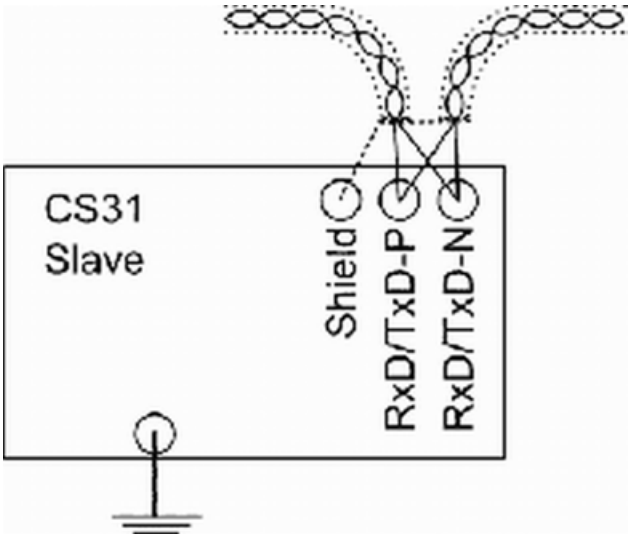


Fig. 738: Bus topology with CS31 bus master in the middle

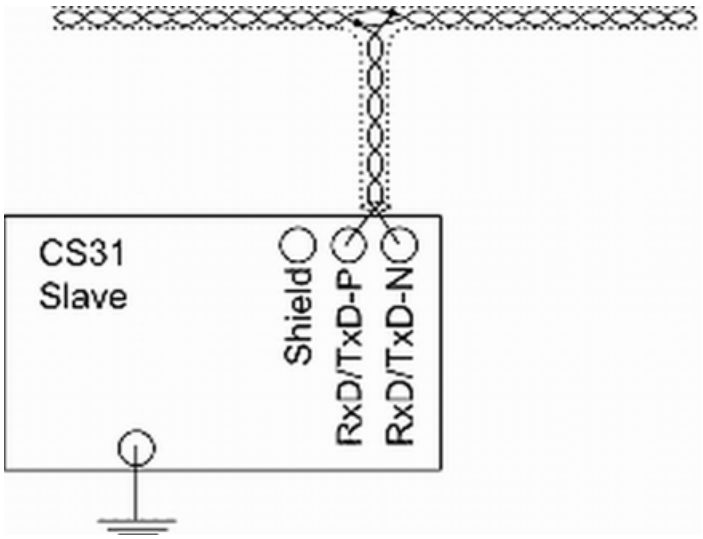
CAUTION!
Risk of malfunctions!
Spur lines are not allowed within the CS31 bus. Loop the bus line from module to module.

CS31 cable laying

Correct cable laying:



Incorrect cable laying:



Grounding

In order to avoid disturbances, ground the cable shields directly.

Current carrying capacity

Choose direct grounding if it can be ensured by means of current carrying metal connections (steel constructions, ground bars, etc.) that no potential differences can occur.

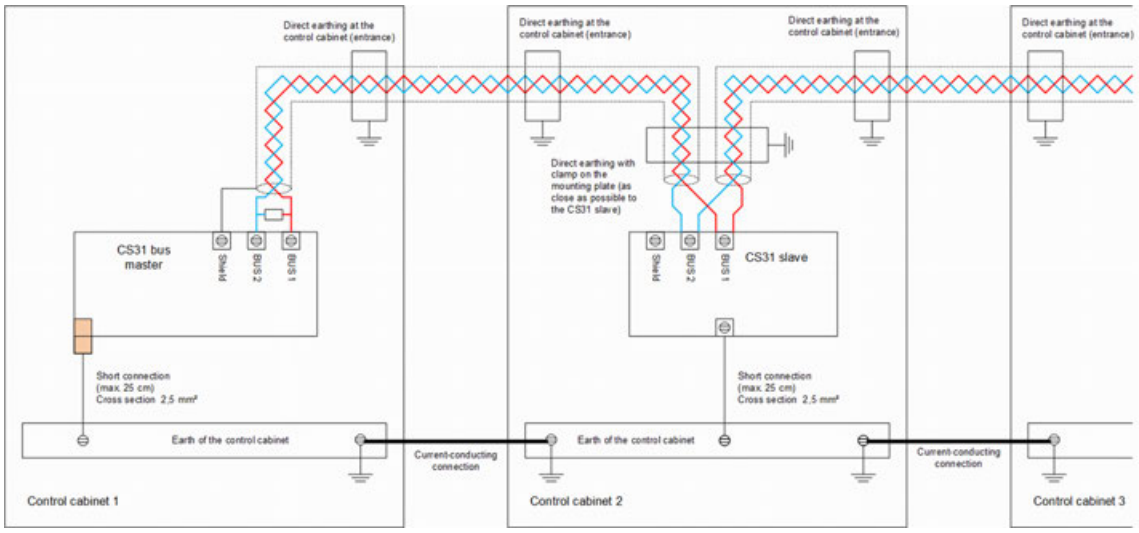


Fig. 739: Direct grounding of CS31 bus master and CS31 slave

> *The shield connection of the CS31 bus master is internally connected to the ground terminal.*

No current carrying capacity

Apply capacitative grounding if system parts are not connected to each other in terms of their current carrying capacity. This prevents the flow of compensation currents through the cable shields.

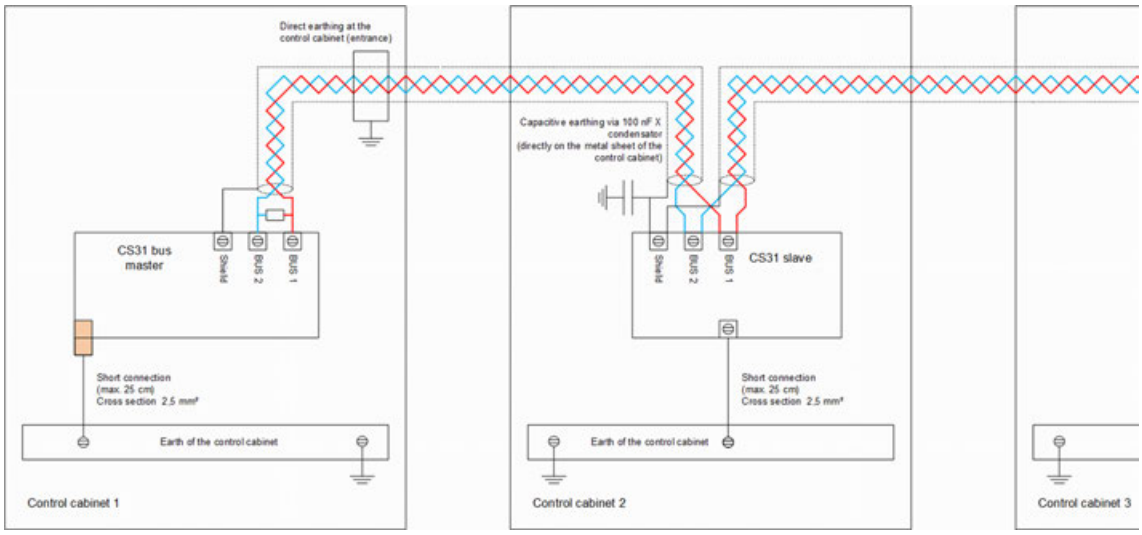


Fig. 740: Direct grounding of CS31 bus master and capacitative CS31 slave

> *On the CS31 slave, the shield connection is not connected internally and thus not grounded. The shield connection can be used to connect the shields of two cables.*

> *VDE 0160 requires that the system's shield is grounded directly at least once.*

Bus cycle time and data security

The communication via the CS31 bus is cyclic and controlled by the CS31 bus master.

Fig. 741: Format of request telegram of a CS31 bus master

In each cycle, the CS31 bus master successively polls all existing CS31 slaves at regular intervals, performs a diagnosis on one of the existing CS31 slaves and sends a request to search for added CS31 slaves. Thus, on one hand it is possible to maintain a continuous diagnosis of the proper network function and on the other hand to take all the newly added CS31 slaves into account.

Fig. 742: Format of response telegram of a CS31 slave

The CS31 slaves respond to the telegrams of the CS31 bus master with a response telegram (see diagram above). The data are indicated in the documentation of the individual devices (e.g. 07AC91-AD2). The telegram is ignored when a CS31 slave or a CS31 bus master detects a deviation between the received CRC and the self-calculated CRC. A CS31 bus error exists when 10 faulty telegrams are issued successively.

The bus cycle time is composed of a base time, the bus transmission times of the data of the individual CS31 slaves and the bus idle times between the individual telegrams.

During the base time, the CS31 bus master performs a diagnosis and searches for newly added CS31 slaves. This time depends on the control system (PLC / central unit) and is partially configurable:

- Devices 07KT94 and 07KT98: base time 2 ms
- Device 07KT94-ARC-AD: base time 10 ms *)
- Devices 07KT98-ARC-AD, 07KT98-ARC-DP-AD, 07KT98-ARC-ETH-AD, 07KT98-ETH-DP-AD,
  07KT98-ARC-ETH-DP-AD:
  Base time 5 ms to 100 ms (configurable in Automation Builder, parameter "Min update time")

*) The base time of device 07KT94-ARC-AD cannot be configured since the old programming environment (907 PC 331) must be used.

The bus transmission times of the data of the individual CS31 slaves can be determined as follows:

- Duration for the transmission of 1 byte = (1/187.5 kBaud) x 8 = 43 µs
- Determine number of data bytes (sending + receiving) from existing documentation
- Add 3 bytes for the transmission of the address and CRCs

Per CS31 slave, approx. 0.5 ms can be assumed as bus idle time. The CS31 bus master needs this time to process the data. This time depends on the computing power and on the implementation of the CS31 bus master. This time can vary between various firmware versions.

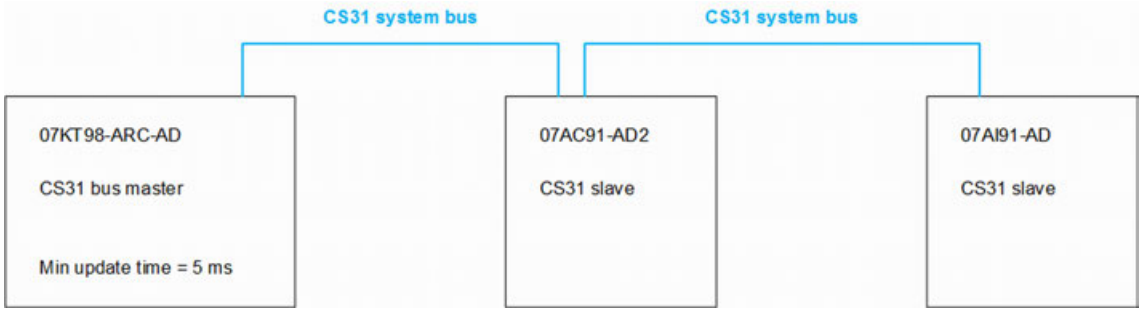


Fig. 743: Example bus cycle time

Table 285: Example: Bus cycle time

Base time	Min. update time = 5 ms		5000 µs
Bus transmission time 07AC91-AD2	Receiving 16 byte data	16 x 43 µs	688 µs
	Sending 16 byte data	16 x 43 µs	688 µs
	3 byte address + CRCs	3 x 43 µs	129 µs
Bus idle time	-	-	500 µs
Bus transmission time 07AI91-AD	Sending 16 byte data	16 x 43 µs	688 µs
	3 byte address + CRCs	3 x 43 µs	129 µs
Bus idle time	-	-	500 µs
Bus cycle time (sum)	-	-	8322 µs ≈ 8500 µs

Configuration

Below is a description of the configuration of the devices 07KT98-ARC-AD, 07KT98-ARC-DP-AD, 07KT98-ARC-ETH-AD and 07KT98-ETH-DP-AD, 07KT98-ARC-ETH-DP-AD in the Automation Builder software. For further information on Automation Builder, please refer to ↳ *Chapter 1.6.5.2 “PLC devices and components” on page 5811*.

The configuration of the CS31 slaves takes place only by means of DIP switches (see existing documentation), whereby the configuration of the CS31 bus topology is carried out in the CS31 bus master.

> *The configuration of the devices 07KT94 and 07KT94-ARC-AD is carried out with the DOS program "907 PC 331". Further information on configuration is available in the existing documentation.*

Configure the COM1 interface as CS31 bus master:

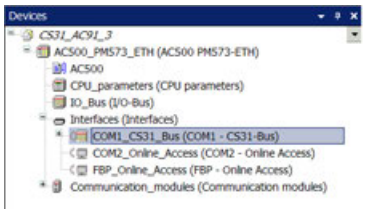


Fig. 744: CS31 bus master

The "Min update time" parameter can also be set on the CS31 bus master:

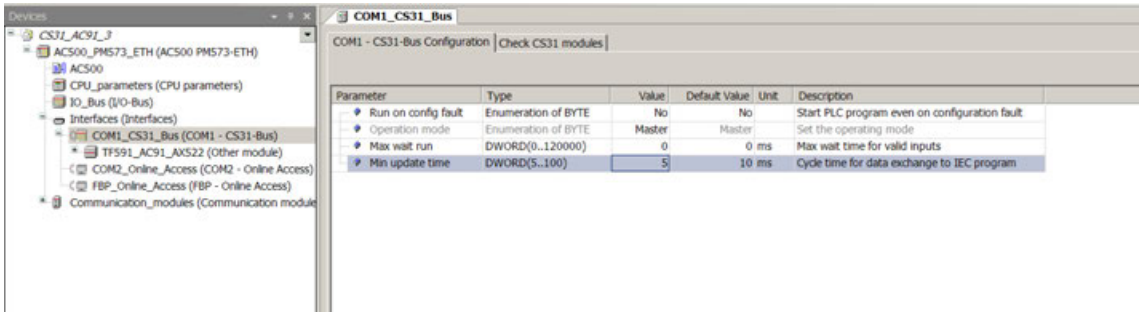


Fig. 745: Parameter configuration

The individual CS31 slaves must be configured in the tree structure under the CS31 bus master:

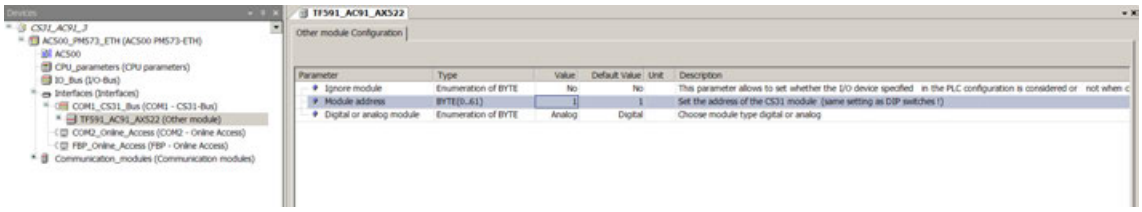


Fig. 746: CS31 slave

The module address must be set on each CS31 slave. Specify the same module address that has been selected with the DIP switches.

Set the CS31 slave type (analog/digital):

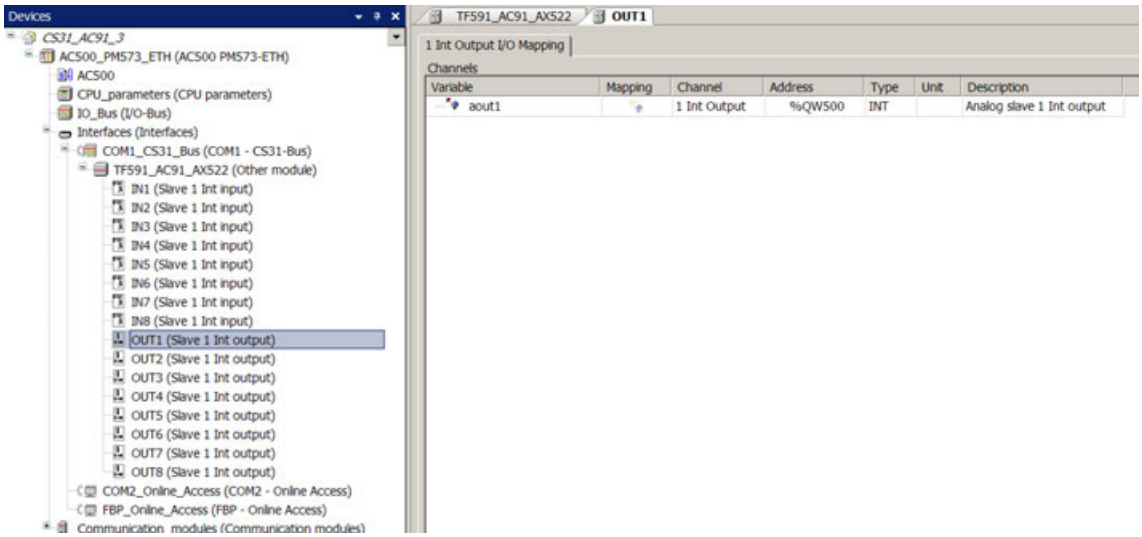


Fig. 747: CS31 bus slave configuration

The data must be configured in the tree structure under the CS31 bus slave. Information about the number of input and output data can be obtained from the respective documentation of the CS31 bus slaves.

> *If the data represent bipolar values (e.g. voltage from -10 V…+10 V), the use of the data type INT is appropriate. In case of unipolar values (e.g. current from 0 mA…20 mA), the data type WORD can be used.*

Diagnosis

For the diagnosis of the CS31 bus, various mechanisms are available in the CS31 bus master of the devices 07KT98-ARC-AD, 07KT98-ARC-DP-AD, 07KT98-ARC-ETH-AD, 07KT98-ETH-DP-AD and 07KT98-ARC-ETH-DP-AD:

- Diagnosis via the function block `CS31_DIAG`
- Diagnosis system of the AC500 series

For further information on both mechanisms, please refer to *Chapter 1.7 “Diagnosis and debugging for AC500 V2 products” on page 6365*. Below, only a few special diagnosis functions of the AC31 adapter are addressed.

Function block `CS31_DIAG`:

In the 'State' column, the variable `byStateDiag` of the structure `strCS31_DiagOneModule` is indicated for every CS31 bus slave.

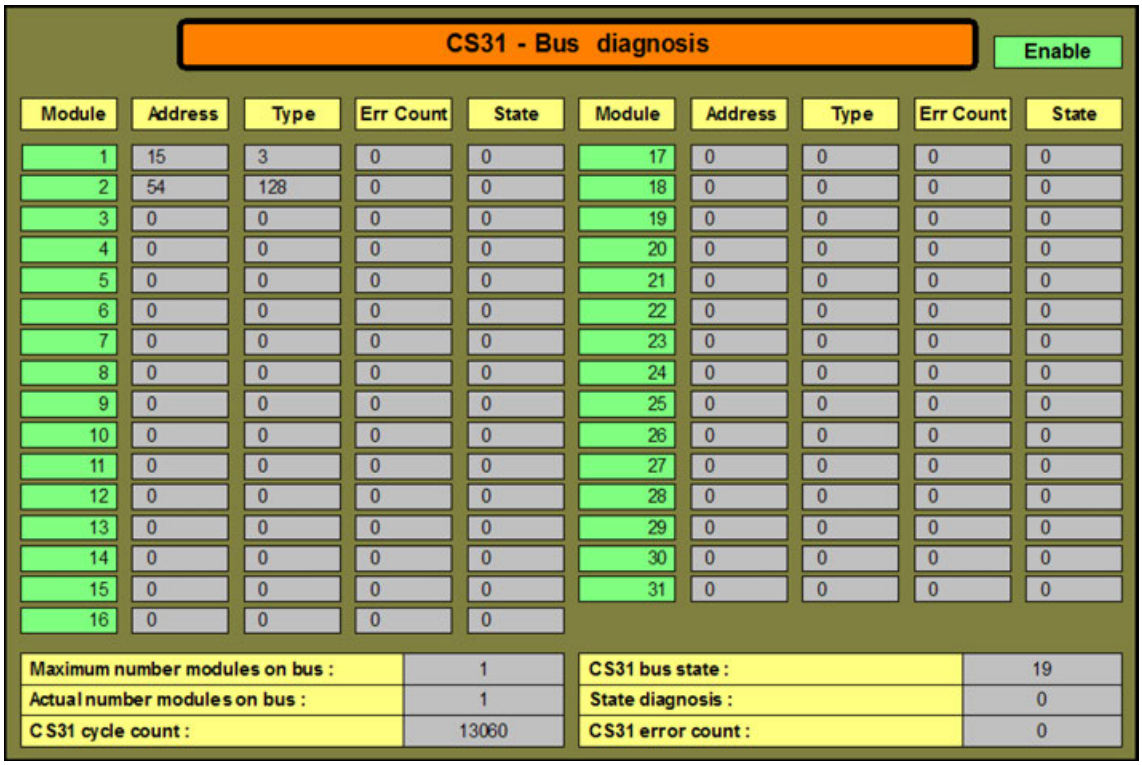


Fig. 748: Visualization: CS31 bus diagnosis

Table 286: Interpretation of variable `byStateDiag`

Bit	Value	Description
0	1	CS31 bus slave disconnected
1	2	Not used
2	4	slave on CS31 bus bus not configured
3	8	Difference in the number of data bytes between configuration and CS31 bus
4	16	Internal device error
5	32	Channel error
6	64	Not used
7	128	Not used

All bits of `byStateDiag` equal 0 -> no error in CS31 bus slave.

The variables `byDiagChannel` and `byDiagErr` in the structure `strCS31_DiagOneModule` include the error channel and code. The possible values of these variables are indicated in the documentation of the respective CS31 bus slave.

Diagnosis system

The Diagnosis system of the AC500 series provides the errors in the following format:

Table 287: Error messages AC500 series

Format	e.g. name of PLC browser command diagshow all	Description
Error class	Class	1 to 4 (see *Chapter 1.7 “Diagnosis and debugging for AC500 V2 products” on page 6365*)
Faulty component	Comp	11 (COM1 interface, here for the CS31 bus)
Faulty device	Dev	Address of CS31 bus slave with error
Faulty module	Mod	CS31 bus type of CS31 bus slave with error (e.g. 5 for analog input/output)
Faulty channel	Ch	See existing documentation of CS31 bus slave
Error code	Err	See existing documentation of CS31 bus slave

A CS31 bus slave error is indicated by an error LED on the CS31 bus slave. The error LED remains on even after elimination of the error and is switched off only after the error has been acknowledged by the CS31 bus master.

The acknowledgment of a CS31 bus slave error can take place via the CS31 bus master by means of the function block `CS31QU_EXT` (see AC500 documentation).

Replacement devices: CPU

For AC31 devices of the 90 series, AC31 adapters (replacement devices) are available for the exchange of the CPU.

Replacement device 07KT9x-AD

Introduction

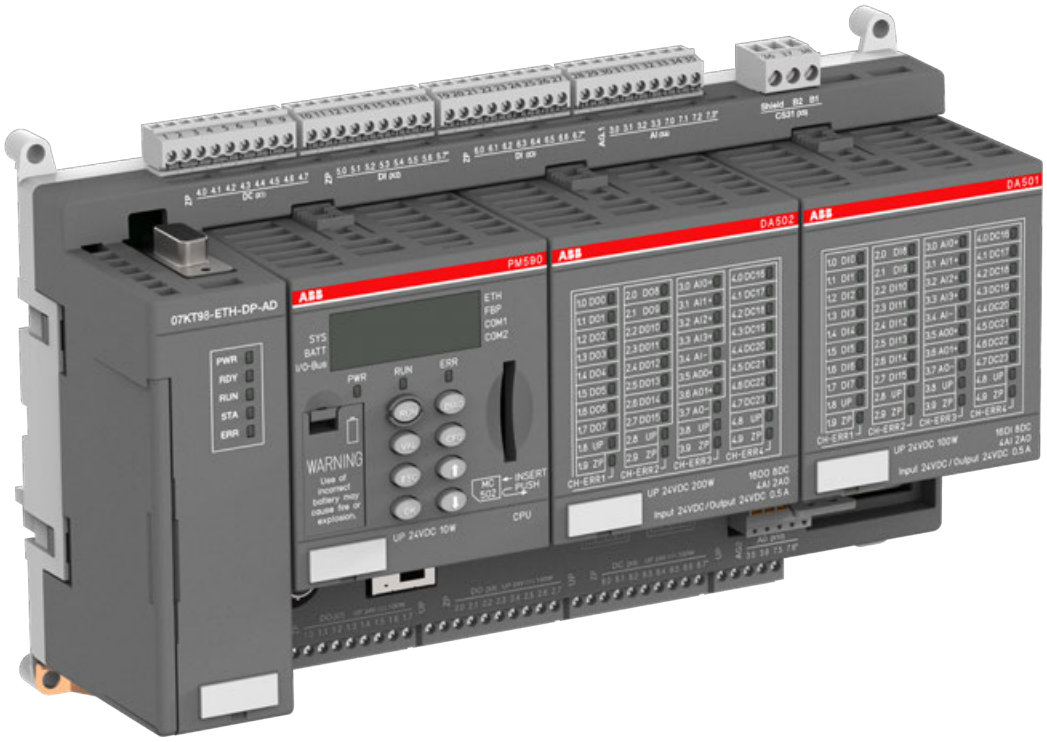


Fig. 749: 3ADR331183S0015

The replacement device versions 07KT9x-AD of the AC31 adapter series replace the existing devices 07KT94 and 07KT98 of the AC31 devices of the 90 series.

Versions:

- 07KT94-ARC-AD: I/O module DA501, I/O module DA502, CPU EC581 *)
- 07KT98-ARC-AD: I/O module DA501, I/O module DA502, CPU PM590-ARC
- 07KT98-ARC-DP-AD: I/O module DA501, I/O module DA502, CPU PM590-ARC
- 07KT98-ARC-ETH-AD: I/O module DA501, I/O module DA502, CPU PM590-ARC-ETH
- 07KT98-ETH-DP-AD: I/O module DA501, I/O module DA502, CPU PM590-ETH
- 07KT98-ARC-ETH-DP-AD: I/O module DA501, I/O module DA502, PM590-ARC-ETH

During the development of the replacement devices, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07KT98 remains valid and serves as reference *(system description Advant Controller 31)*. The document structure of this document is based on the document structure of the existing documentation.

*) Customer specific product not for standard use

This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement devices 07KT9x-AD can be found in the operating and assembly instructions of device 07KT9x-AD: 3ADR020082M0401.

Please observe the system data for CS31 bus *Chapter 1.6.2.3.3.3 "System data and CS31 bus system data" on page 3876*.

For general information on the CPU, please refer to the AC500 documentation *Chapter 1.7 "Diagnosis and debugging for AC500 V2 products" on page 6365*.

In addition to the CPU, the replacement devices 07KT9x-AD are based on the modules DA501 and DA502 of the AC500 series. All I/O channels are protected against reverse polarity, reverse supply, short circuit and continuous overvoltages up to 30 V DC. For further information on these modules, please refer to the AC500 documentation.

The description of the protective functions, error indications and diagnosis options contained in the existing documentation are no longer valid. Please refer to the AC500 documentation (DA501-/ DA502 modules and CPU) concerning this information.

Central unit 07KT98

Short description

The central unit 07KT9x-AD acts as

- bus master in the decentralized automation system.
  Slave operation is not possible.
- Advant Controller 31 or as stand-alone central unit.

Main features

- 16 digital inputs with LED display.
  Caution! Galvanic isolation/potential reference has changed.
- 16 digital outputs with LED display.
  Caution! Galvanic isolation/potential reference has changed.
- 16 digital inputs/outputs with LED display.
  Caution! Galvanic isolation/potential reference has changed.
- 8 individually configurable analog inputs. Available modes *Chapter 1.6.2.3.3.4.1.3.1.7 "Connection of the 8 configurable analog inputs" on page 3905*.
  Caution! Galvanic isolation/potential reference has changed.
- 4 individually configurable analog outputs.
  Caution! Galvanic isolation/potential reference has changed.
- 2 counters for counting frequencies up to 50 kHz, configurable in 10 different modes.
  Caution! Each counting input requires an external resistor of 470 Ω / 1 W that is connected upstream. The potential reference has changed.
- 1 serial interface COM2
  - Modbus RTU, master and slave
  - An online access (RS-232 programming interface for PC/Automation Builder)
  - A free protocol (communication via the blocks COM_SEND and COM_REC)
- 1 serial diagnosis interface DIAG
  Caution! No galvanic isolation to supply voltage L+/M.
- LED LCD display to indicate operating conditions and error messages
- Fastening by screws or snapping onto top-hat rail
- Lithium battery TA521
- Various operating buttons for user input
- Comprehensive diagnosis functions
- Integrated Flash EPROM, RAM and memory for storing programs and data
- Exchangeable memory card

Planning/ commissioning

Software Automation Builder (see AC500 documentation):

- 07KT98-ARC-AD
- 07KT98-ARC-DP-AD
- 07KT98-ARC-ETH-AD
- 07KT98-ETH-DP-AD
- 07KT98-ARC-ETH-DP-AD

Software 907PC331

- 07KT94-ARC-AD

Functionality

Table 288: Existing device vs. replacement device

Designation	Existing device: 07KT98	Replacement device: 07KT9x-AD	Note
User program	1 MB	CPU PM590: 2 MB storage, memory card slot	-
User data	1 MB + 256 kB RETAIN + 128 kB (Flash EPROM)	CPU PM590: 2 MB storage, memory card slot	-
Digital inputs	24 in 3 groups (8 each), galvanically isolated	16 in 2 groups (8 each). Caution: Potential reference/galvanic isolation	Potential reference/galvanic isolation has changed *).
Digital outputs	16 transistor outputs in 2 groups (8 each), galvanically isolated	16 in 2 groups (8 each). Caution: Potential reference/galvanic isolation	Potential reference/galvanic isolation has changed *).
Digital inputs/outputs	8 in 1 group, galvanically isolated	16 in 2 groups (8 each). Caution: Potential reference/galvanic isolation	Potential reference/galvanic isolation has changed *).
Analog inputs	8 in 1 group, individually configurable to 0 ... 10 V, 0 ... 5 V, ±10 V, ±5 V, 0 ... 20 mA, 4 ... 20 mA, Pt100 (2-wire or 3-wire), differential inputs, digital inputs	8 in 1 group, individually configurable 0..10 V, ±10 V, 0..20 mA, 4 ... 20 mA, Pt100/ PT1000/ Ni1000 (2-wire or 3-wire), differential inputs, digital inputs	Potential reference has changed *). Some wiring adjustments are required in part. 5 V measuring ranges can be shown with 10 V measuring range.
Analog inputs (can also be configured as digital inputs)	Yes	Yes	Caution: AGND reference to ZP no longer M
Analog outputs	4 in 1 group, individually configurable to ± 10 V, 0 ... 20 mA, 4 ... 20 mA	4 in 1 group, individually configurable to ± 10 V, 0 ... 20 mA, 4 ... 20 mA	Caution: AGND reference to ZP no longer M *). Some wiring adjustments are required in part.

Designation	Existing device: 07KT98	Replacement device: 07KT9x-AD	Note
Serial Interfaces	COM1, COM2 as Modbus interfaces, for programming and test functions as well as freely programmable interfaces	COM2 (programming function, test function, free protocol) DIAG (diagnosis interface)	The serial COM1 interface of 07KT9x is no longer available. The serial diagnosis interface DIAG has a reduced range of functions and is not galvanically isolated from the supply voltage L+/M.
Parallel interface	For connection to communication module	For connection to communication module	Additional information upon request.
System bus interface	CS31	CS31	Caution: Terminal "Shield" is internally connected to FE (functional earth).
High-speed counter	Integrated, many functions configurable	Integrated, many configurable operating modes	At the counting input, an external resistor of 470 Ω / 1 W must always be connected upstream. For further information on high-speed counters, please refer to the AC500 documentation.
Real-time clock	Integrated	Integrated	-
Memory card	SmartMedia Card: Storage medium for operating system, user program and user data	Memory card: for the backup of user data, storage of the user program and update of the internal CPU firmware	-
Display LEDs	For signal states, operating conditions and error messages	Indication on LEDs and LCD display	-
Supply voltage	24V	24V	-
Data buffering	With lithium battery 07 LE 90	With lithium battery TA521	-
Programming software	907 AC 1131 as of V 4.1 (07KT98 with ARCNET interface) 907 AC 1131 as of V 4.3 (07KT98 with PROFIBUS DP interface)	Automation Builder as of V1.2	-
Processing time	Processing time: 65% bit, 35% word, for 1 kB program, typ. 0.07 ms	Cycle time for 1 instruction (CPU PM590). Binary: min. 0.002 µs, word: min. 0.004 µs, floating point: min. 0.004 µs	-

*) ↳ *Chapter 1.6.2.3.3.4.1.3.1 “Connections” on page 3897*

Table 289: Comparison: Replacement device versions

	07KT94-ARC-AD	**07KT98-ARC-AD**	**07KT98-ARC-DP-AD**	**07KT98-ARC-ETH-AD**	**07KT98-ETH-DP-AD**	**07KT98-ARC-ETH-DP-AD**
ARCNET	x	x	x	x	-	x
PROFIBUS	-	-	x	-	x	x
Ethernet	-	-	-	x	x	x
CS31	x	x	x	x	x	x
Parallel interface for connection to communication module	-	x	x	x	x	x
Cycle time for 1 instruction	CPU EC581: n.a.	*)	*)	*)	*)	*)

*) CPU PM590: -> Binary: min. 0.002 µs, -> word: min. 0.004 µs, -> floating point: min. 0.004 µs

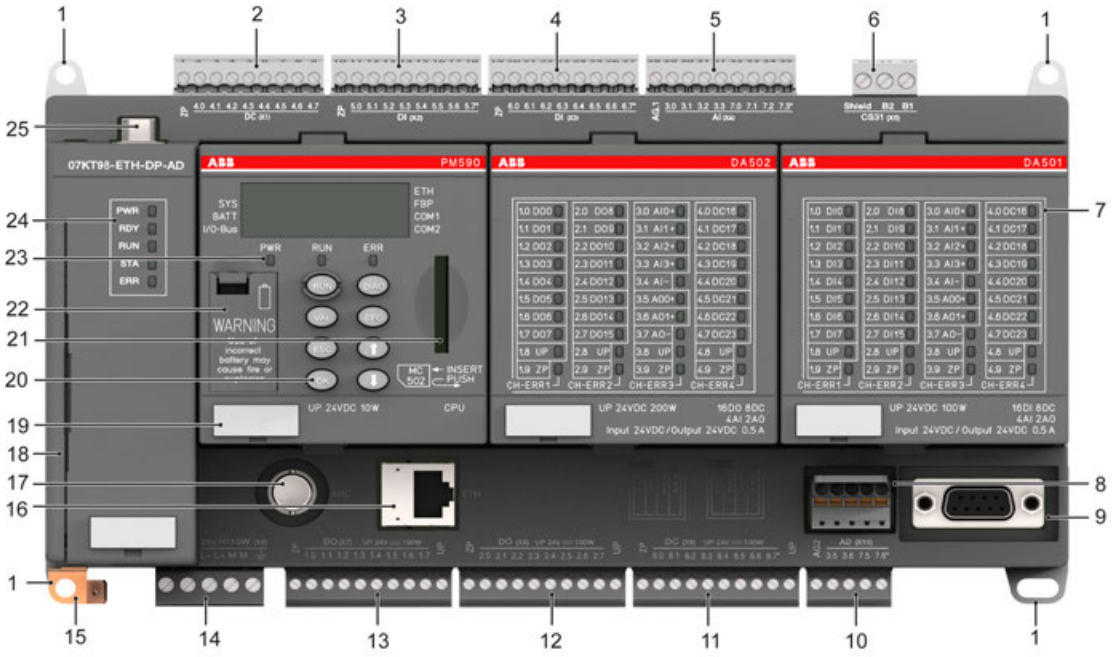
Available versions

To get an overview of the the available versions for 07 KT 98 central units, please refer to previous chapter ↳ *Table 289 “Comparison: Replacement device versions” on page 3895*.

Suitable SmartMedia cards

The 07KT9x-AD systems use memory cards of the type "SD Memory Card MC502".

Device configuration



1 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
2 Digital inputs/outputs for DA502
3 Digital inputs for DA501
4 Digital inputs for DA501
5 Analog inputs for DA501/DA502
6 CS31 bus Interface
7 Status LEDs for DA501/DA502
8 DIAG: Serial interface (diagnosis)
9 COM2: Serial interface (thread UNC 4-40)
10 Analog outputs for DA501/DA502. ±10 V, 0 ... 20 mA, 4 ... 20 mA in one group
11 Digital inputs/outputs for DA501
12 Digital outputs for DA502
13 Digital outputs for DA502
14 Supply voltage connection 24 V DC (CPU and communication module)
15 Ground connection (FE). Connection for 6.3 mm Faston.
16 Ethernet: Network interface (function depends on device version)
17 Interface for ARCNET (BNC)
18 External network interface
19 TA525: Label
20 8 operating buttons
21 Memory card
22 Battery compartment for lithium battery TA521
23 3 system LEDs
24 5 status LEDs (only for PROFIBUS)
25 Connection for PROFIBUS (optional) (function depends on device version)

For information on the available I/O modules DA501 and DA502, please refer to the AC500 documentation. The CPU module used (here: PM590) depends on the model version.

Connections

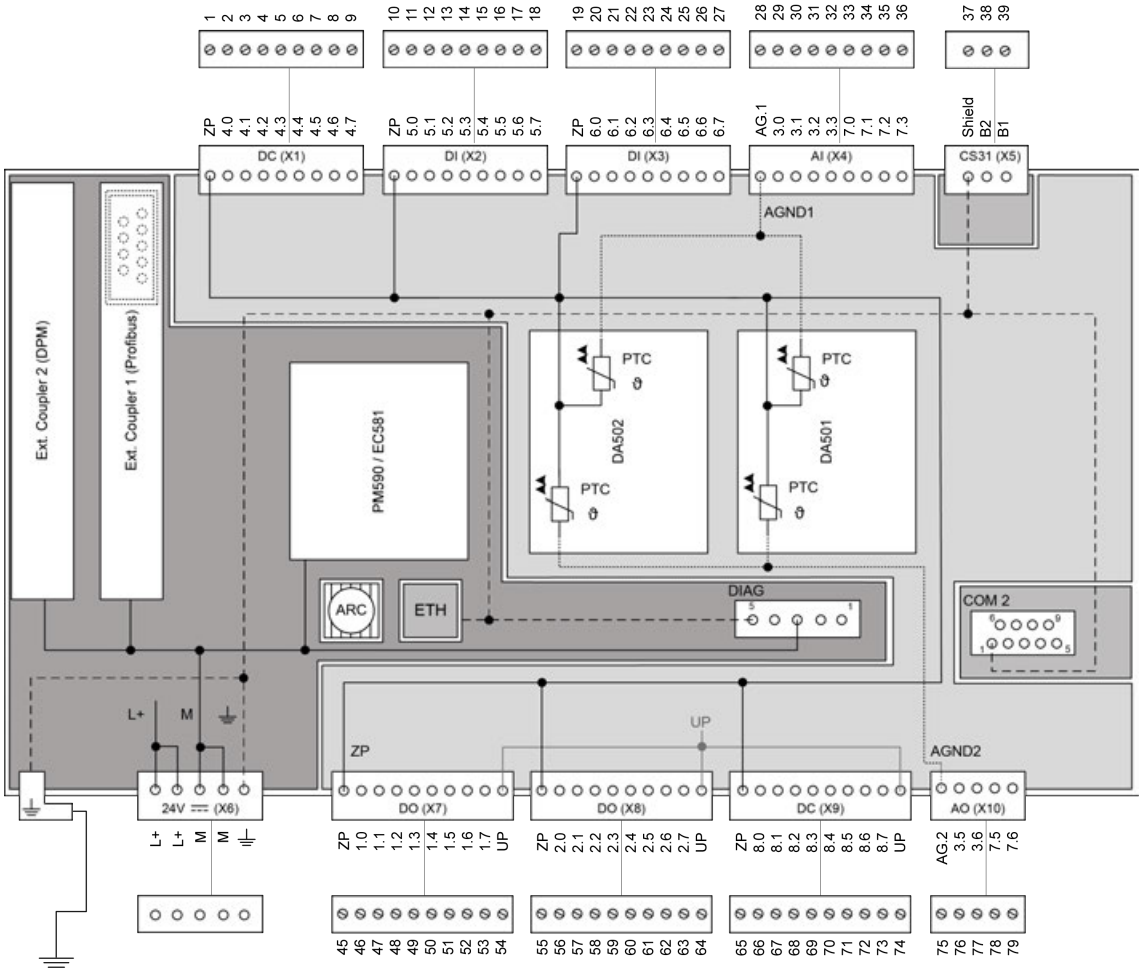


Fig. 750: Terminal assignment 07KT9x-AD

DIAG	No galvanic isolation (M)
COM2	Galvanically isolated
CS31 bus	Galvanically isolated
Ethernet	Galvanically isolated
ARCNET	Galvanically isolated
DA501/DA502	Galvanically isolated

Further information on grounding: ↬ *Chapter 1.6.2.3.3.3.1.7 “Grounding” on page 3879*.

Application example for connecting the inputs and outputs

Please observe the following information: ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*

Connection of the supply voltage

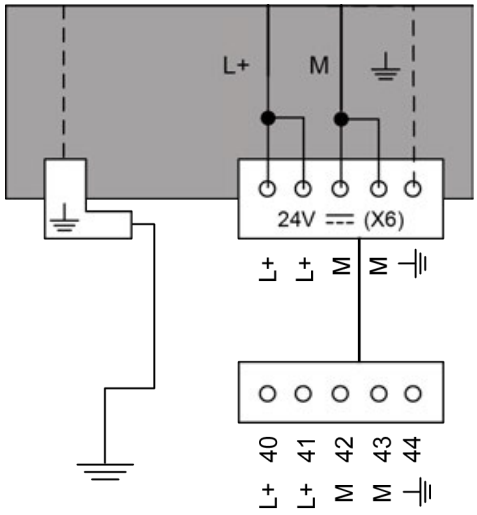


Fig. 751: Connection of the supply voltage

Table 290: Connector (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / L+	40	Supply voltage +24 V DC
X6 / L+	41	Supply voltage +24 V DC
X6 / M	42	Ground connection (0 V)
X6 / M	43	Ground connection (0 V)
X6 / functional earth	44	The functional earth (FE) is connected to the Faston terminal inside the device. Ensure that no ground loops are created and that FE and Faston are connected to the same ground potential.

NOTICE!

- In addition to connecting the supply voltage (L+/M) to X6, the supply voltage (UP/ZP) must be connected to all connectors.
- ZP must be connected to all connectors (X1, X2, X3, X7, X8, X9).
- UP must be connected to all connectors (X7, X8, X9).
- L+/M and UP/ZP must always be supplied with voltage.

Connection for CS31 bus

Table 291: Connector (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / shield	37	Shield (functional earth)
X5 / B2	38	BUS2
X5 / B1	39	BUS1

Terminal "Shield" is internally connected to FE. The previous grounding measures, e.g. with clip at the switchgear cabinet, are still required. ↳ Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876

If 07KT9x-AD is connected to one of the bus ends, a 120 Ω resistor must be connected for bus termination. The device 07KT9x-AD always functions as master. Slave operation is not possible. Further information on CS31 bus: ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*

Connection of digital inputs

See ↳ *Chapter 1.6.2.3.3.4.1.3.1 “Connections” on page 3897* .

Table 292: Connector X2

Connector / Terminal	Pin	Assignment / Signal
X2 / ZP	10	ZP
X2 / 5.0	11	DA501 / DI0
X2 / 5.1	12	DA501 / DI1
X2 / 5.2	13	DA501 / DI2
X2 / 5.3	14	DA501 / DI3
X2 / 5.4	15	DA501 / DI4
X2 / 5.5	16	DA501 / DI5
X2 / 5.6	17	DA501 / DI6
X2 / 5.7	18	DA501 / DI7

Table 293: Connector (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / ZP	19	ZP
X3 / 6.0	20	DA501 / DI8
X3 / 6.1	21	DA501 / DI9
X3 / 6.2	22	DA501 / DI10
X3 / 6.3	23	DA501 / DI11
X3 / 6.4	24	DA501 / DI12
X3 / 6.5	25	DA501 / DI13
X3 / 6.6	26	DA501 / DI14
X3 / 6.7	27	DA501 / DI15

In contrast to the existing device 07KT98, the function of the digital inputs is only possible if voltage UP is connected.

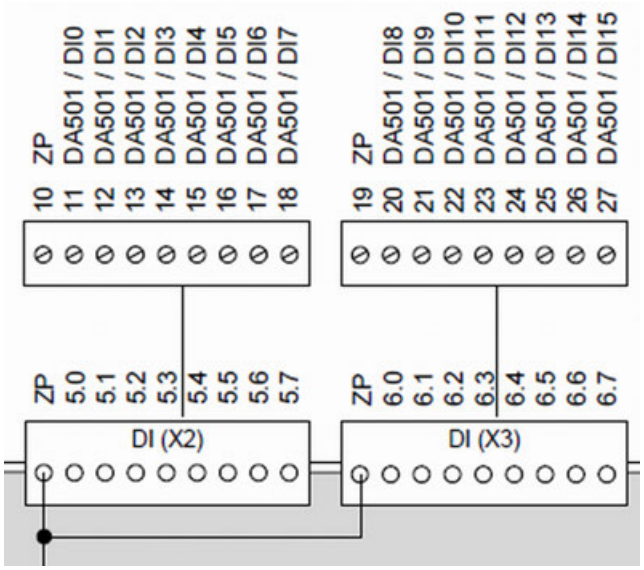


Fig. 752: Arrangement of the 16 digital inputs

The digital input states are always indicated by the LEDs DI0-DI15:

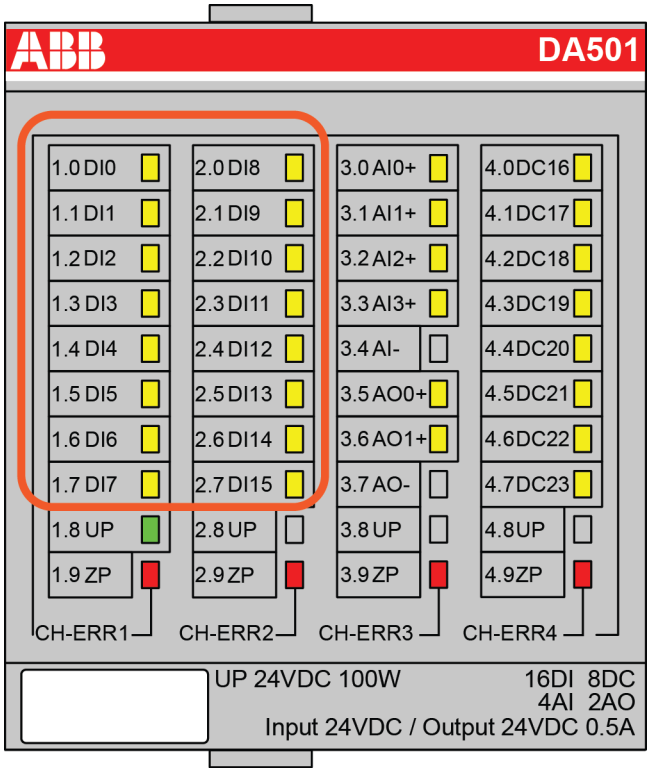


Fig. 753: DA501 LED status indication

Characteristics of the digital inputs:

- All 16 inputs have the same potential ZP as all other inputs/outputs. The galvanic isolation included in the existing devices is no longer available.
- Input delay (0->1 or 1->0): Typically 0.1 ms, configurable from 0.1 to 32 ms.

> *The signal coupling of the input signals is no longer realized via optocoupler. All channels of the DA501 and DA502 modules have reference to ZP. The AGND1/AGND2 of the analog channels are internally connected to ZP via PTC resistors. For information on terminal assignment, refer to figure Fig. 750).*

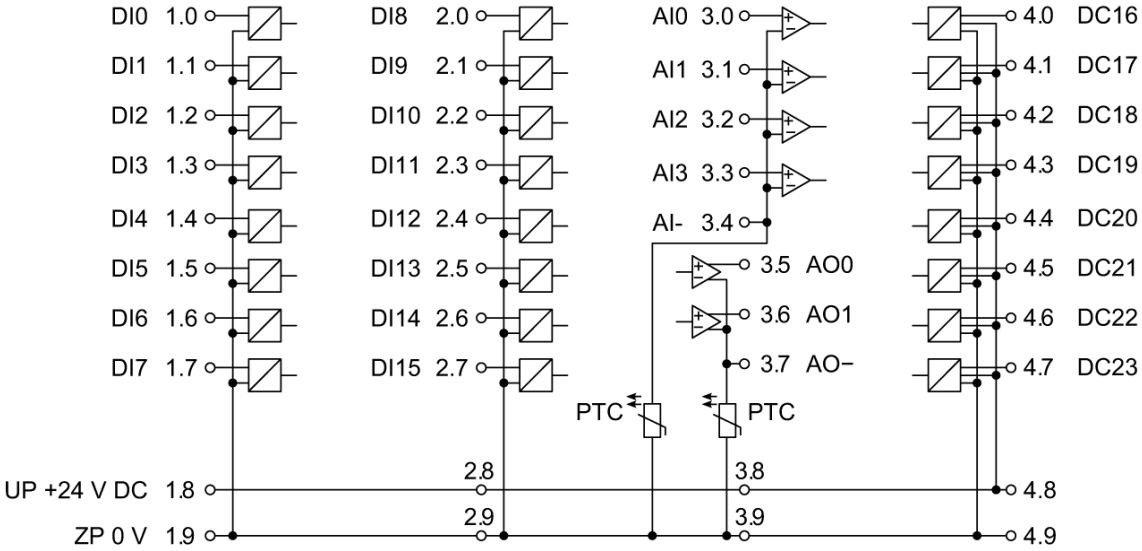


Fig. 754: Circuit arrangement of DA501 module

Connection of the digital outputs

See *Chapter 1.6.2.3.3.4.1.3.1 “Connections” on page 3897*.

Table 294: Connector (X7)

Connector / Terminal	Pin	Assignment / Signal
X7 / ZP	45	ZP
X7 / 1.0	46	DA502 / DO0
X7 / 1.1	47	DA502 / DO1
X7 / 1.2	48	DA502 / DO2
X7 / 1.3	49	DA502 / DO3
X7 / 1.4	50	DA502 / DO4
X7 / 1.5	51	DA502 / DO5
X7 / 1.6	52	DA502 / DO6
X7 / 1.7	53	DA502 / DO7
X7 / UP	54	UP

Table 295: Connector (X8)

Connector / Terminal	Pin	Assignment / Signal
X8 / ZP	55	ZP
X8 / 2.0	56	DA502 / DO8
X8 / 2.1	57	DA502 / DO9
X8 / 2.2	58	DA502 / DO10
X8 / 2.3	59	DA502 / DO11
X8 / 2.4	60	DA502 / DO12
X8 / 2.5	61	DA502 / DO13
X8 / 2.6	62	DA502 / DO14

Connector / Terminal	Pin	Assignment / Signal
X8 / 2.7	63	DA502 / DO15
X8 / UP	64	UP

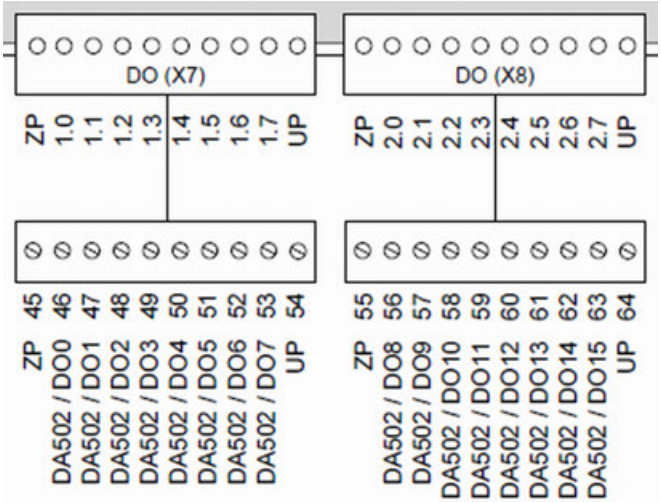


Fig. 755: Arrangement of digital outputs

Characteristics of the digital outputs

- The digital output states are always indicated by the LEDs DO0-DO15 on DA501 module.
- All 16 outputs have the same potential ZP as all other inputs/outputs. The galvanic isolation included in the existing devices is no longer available.
- Diagnosis: Stored errors are indicated via an LED and can be accessed by the CPU (see AC500 documentation).

Circuit arrangement of digital outputs

- Fig. 755
- *Further information on page 3904*

Connection of the digital inputs/outputs

Table 296: Connector (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / ZP	1	ZP
X1 / 4.0	2	DA502 / DC16
X1 / 4.1	3	DA502 / DC17
X1 / 4.2	4	DA502 / DC18
X1 / 4.3	5	DA502 / DC19
X1 / 4.4	6	DA502 / DC20
X1 / 4.5	7	DA502 / DC21
X1 / 4.6	8	DA502 / DC22
X1 / 4.7	9	DA502 / DC23

Table 297: Connector (X9)

Connector / Terminal	Pin	Assignment / Signal
X9 / ZP	65	ZP
X9 / 8.0	66	DA501 / DC16
X9 / 8.1	67	DA501 / DC17
X9 / 8.2	68	DA501 / DC18
X9 / 8.3	69	DA501 / DC19
X9 / 8.4	70	DA501 / DC20
X9 / 8.5	71	DA501 / DC21
X9 / 8.6	72	DA501 / DC22
X9 / 8.7	73	DA501 / DC23
X9 / UP	74	UP

The arrangement of the 16 digital inputs/outputs is shown below:

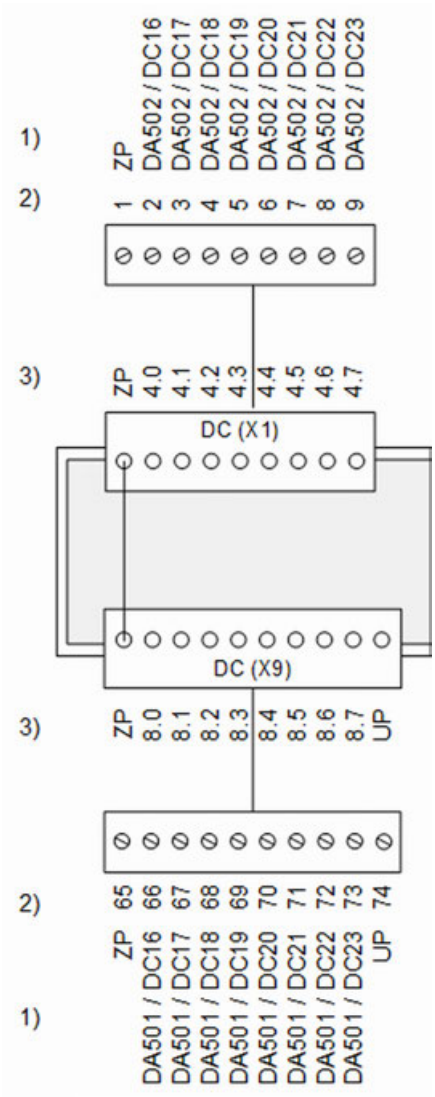


Fig. 756: Digital inputs/outputs

1) Module assignment
2) Terminal number
3) Terminal

Characteristics of the digital inputs/outputs

- The digital input/output states are always indicated via the LEDs DC16 - DC23 on DA501 or DA502.
- All 16 inputs/outputs have the same potential ZP as all other inputs/outputs. The galvanic isolation included in the existing devices is no longer available.
- Diagnosis: Stored errors are indicated via an LED and can be accessed by the CPU (see AC500 documentation).
- The inputs/outputs can be configured as input and as output. The outputs can also be read back.
- Input delay (0->1 or 1->0): Typically 0.1 ms, configurable from 0.1 ms to 32 ms.
- The total current consumption of all 16 DC channels must not exceed 4 A.
- The total current consumption of all 16 DO and 16 DC channels must not exceed 12 A.

The following shows a diagram of the circuitry of a digital input/output with varistors for demagnetization when switching off inductive loads:

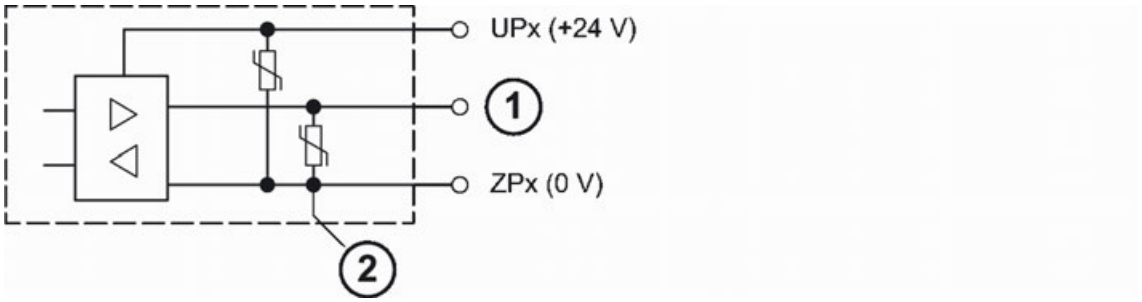


Fig. 757: Circuit arrangement

1 Digital input/output
2 For demagnetization when switching off inductive loads

Data	Value
Input signal voltage	24 V DC
0 signal	-3 V ... +5 V
Undefined signal state	> +5 V ... < +15 V
1 signal	+15 V ... +30 V

The technical input data contained in the existing documentation are no longer valid.

The varistor protection circuit has changed. The varistors for demagnetization are no longer located between UP and the respective channel, but rather between ZP and the respective channel. It is no longer possible to connect the voltage supply UP to connector X5 and thus use the input voltage range from -30 V to 30 V. At the inputs, only voltages from -3 V to +30 V may be applied. UP must always be connected to all connectors (X7, X8, X9).

Connection of the 8 configurable analog inputs

Table 298: Connector (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / AG.1	28	AGND1
X4 / 3.0	29	DA502 / AI0+
X4 / 3.1	30	DA502 / AI1+
X4 / 3.2	31	DA502 / AI2+
X4 / 3.3	32	DA502 / AI3+
X4 / 7.0	33	DA501 / AI0+
X4 / 7.1	34	DA501 / AI1+
X4 / 7.2	35	DA501 / AI2+
X4 / 7.3	36	DA501 / AI3+

To be able to use the analog inputs, UP must be connected. L+/M and UP/ZP must always be supplied with voltage.

To be able to use the analog inputs, UP must be connected. L+/M and UP/ZP must always be supplied with voltage.

The analog channels offer self-protective functions and diagnosis options in the following situations:

- Above range of analog value (input)
- Above range of analog value (output)
- Below range of analog value (input)
- Below range of analog value (output)
- Wire breakage
- Short circuit

For further information on behavior and indication of these errors, please refer to the AC500 documentation. The arrangement of the 8 analog inputs is shown below on X4.

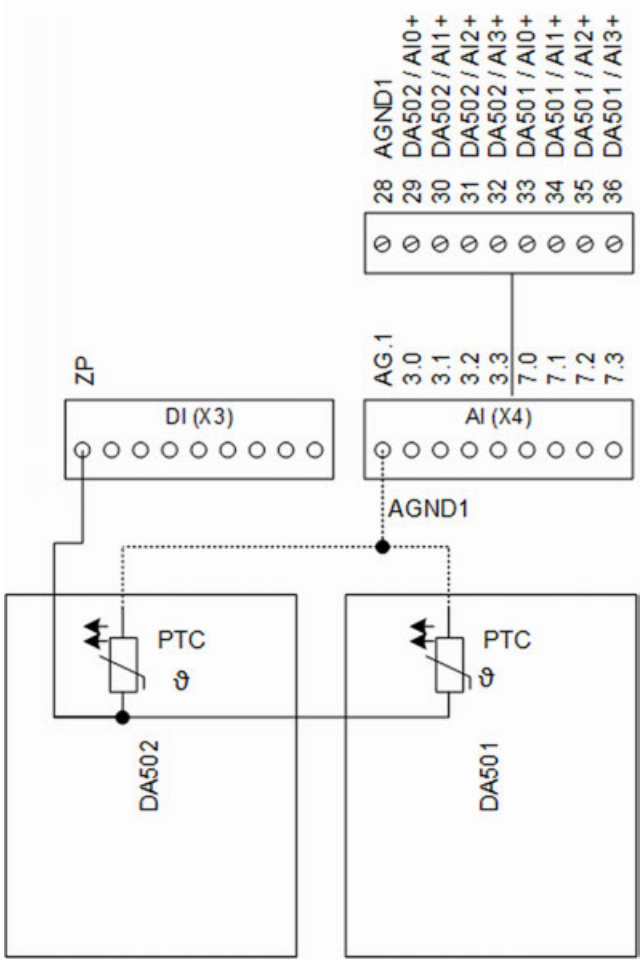


Fig. 758: Arrangement of the analog inputs

> *Reference to earth ZP: connect ZP to several connectors. In the example, ZP is connected to connector X3.*

Characteristics of the analog inputs:

- The 8 analog inputs are not galvanically isolated. The internal PTC connection is connected to earth ZP (existing device: earth M). Depending on sensor type or measuring principle, this may result in wiring adjustments.
- Resolution:
  - Range 0 ... 10 V: 12 bit
  - Range -10 ... +10 V: 12 bit + sign
  - Range 0 ... 20 mA: 12 bit
  - Range 4 ... 20 mA: 12 bit
  - Range RTD (Pt100, PT1000, Ni1000): 0.1 °C

Connection examples for analog transmitters are shown below.

Measuring ranges

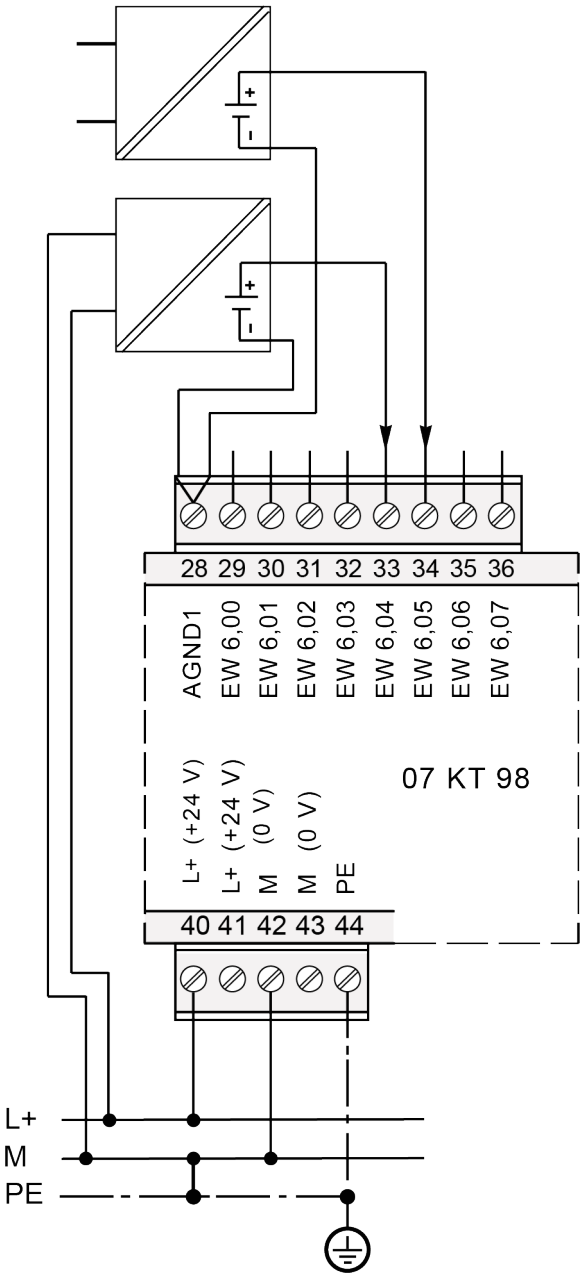


Fig. 759: 07KT98_Measuring_Ranges ± 10 V / 0 ... 10 V

Due to the internal galvanic isolation of the sensor voltage supply, no change to the wiring is necessary.

> *UP must be connected to connectors X7, X8 and X9. The internal voltage supply to the ADC channels is no longer provided by L+ but by UP in the modules DA501 and DA502.*

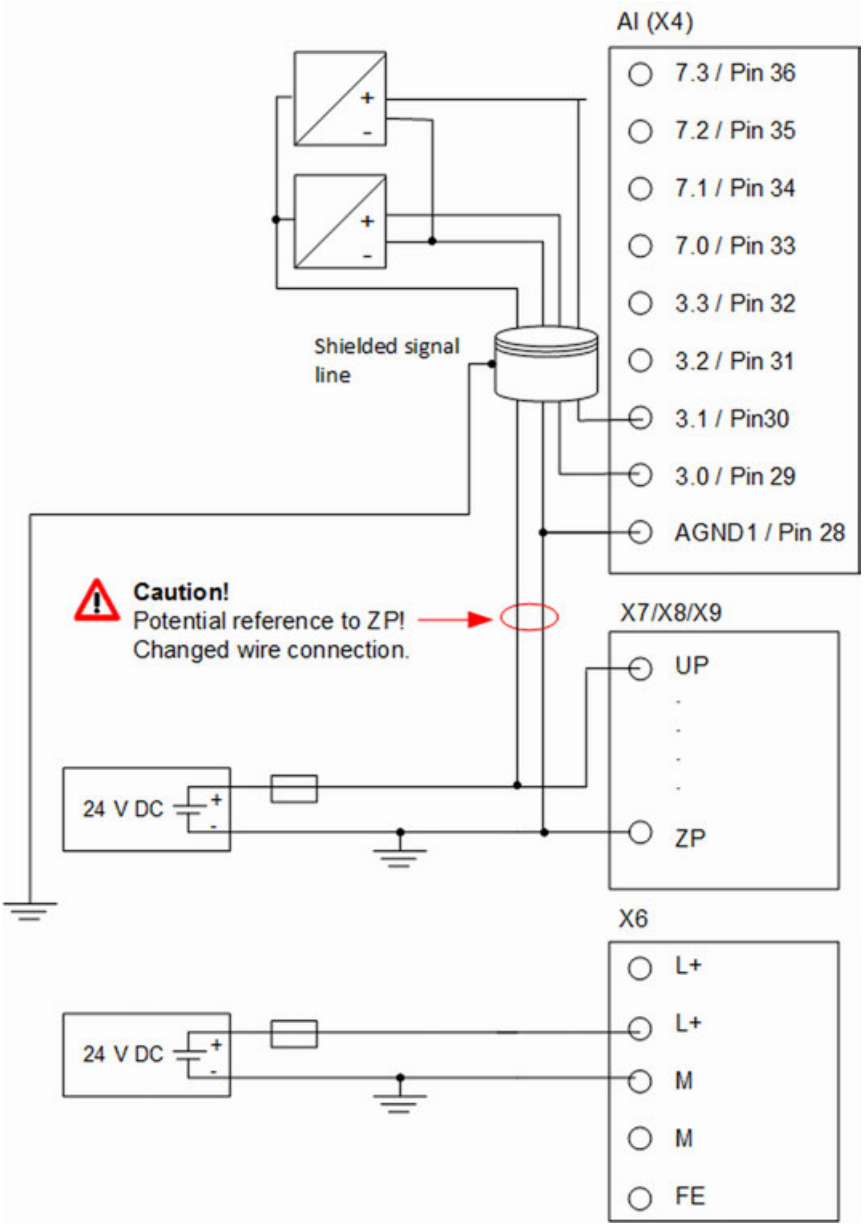


Fig. 760: Voltage input with externally supplied 3-wire voltage sensors

Measuring ranges (passive two pole sensors)

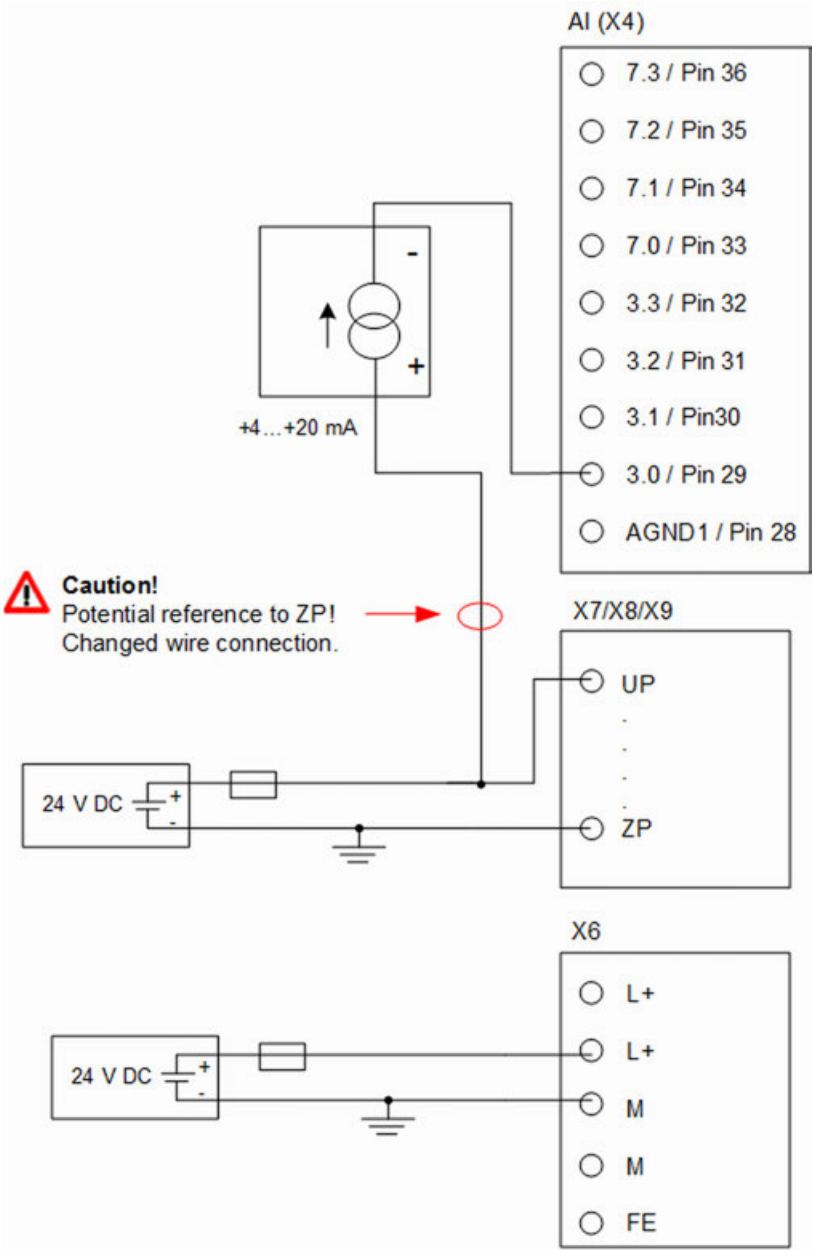


Fig. 761: Connection of current sensors 4 ... 20 mA to the analog inputs

If the analog current sensors 4 ... 20 mA are supplied from a separate power supply unit, the 0 V/GND connection of the power supply unit must be connected to the ZP connection of the 07KT9x-AD.

Protective functions

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies a current in excess of 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or sensors without current peaks higher than 25 mA. If this is not possible, protect the input by connecting a 10-volt zener diode in parallel to I+ and I-.

For further information on protective function, error indication and diagnosis, please refer to the AC500 documentation.

Measuring range (active sensors with external supply)

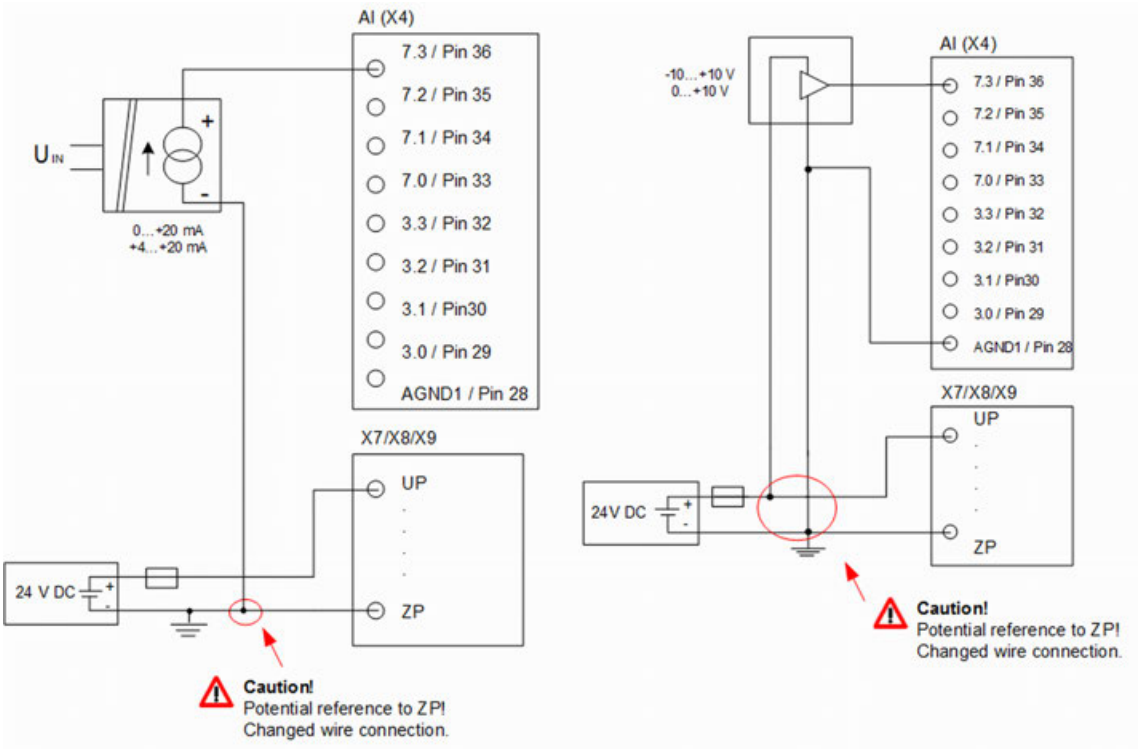


Fig. 762: Connection of current sensors 0 ... 20 mA to the analog inputs

Please note that in the example the 0 V supply (ZP) must be used as reference potential.

For further information on protective functions, error indication and diagnosis, please refer to the AC500 documentation.

Measuring ranges ± 10 V / 0 ... 10 V as differential inputs

Differential inputs are very useful when applying analog sensors with non-isolated installation at the site (e.g. if the minus terminal is grounded on site). The measurement via differential inputs considerably improves the measuring accuracy and prevents ground loops.

When configuring differential inputs, always two adjacent analog channels belong together (e.g. the channels 3.0 and 3.1). In this case, both channels are configured according to the desired operating mode. The channel with the lower channel number must be the one with the even number (e.g. channel 3.0).

The converted analog value is available at the odd channel (e.g. channel 3.1) and can be determined by means of the Automation Builder. The analog value is calculated by subtracting the input values: input value at the channel with the higher channel number minus input value on channel with lower channel number.

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too much potential difference with respect to ZP (max. ± 1 V within the full signal range).

- Ensure that the potential difference never exceeds ± 1 V.
- No change to the wiring is necessary. The connection of the sensor corresponds to the one of the existing device 07KT98.

For further information on protective function, error indication and diagnosis, please refer to the AC500 documentation.

Measuring range with Pt100 2-wire

Table 299: Figure range

Range	Assigned figure range
-50 C ... 400°C	-500 ... +4000
-50 C ... 70°C	-500 ... +700

The following measuring ranges can be configured:

Table 300: Measuring ranges

Pt100	-50 °C ... +400 °C	2-wire configuration, 1 channel used
Pt100	-50 °C ... +70 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C ... +400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C ... +150 °C	2-wire configuration, 1 channel used

Measuring values above range, below range and wire breaks are monitored and indicated.

For further information on protective function, error indication and diagnosis, please refer to the AC500 documentation.

Measuring range with Pt100 3-wire

Table 301: Figure range

Range	Assigned figure range
-50 C ... 400°C	-500 ... +4000
-50 C ... 70°C	-500 ... +700

The following measuring ranges can be configured:

Table 302: Measuring ranges

Pt100	-50 °C ... +400 °C	3-wire configuration, 2 channels used
Pt100	-50 °C ... +70 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C ... +400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C ... +150 °C	3-wire configuration, 2 channels used

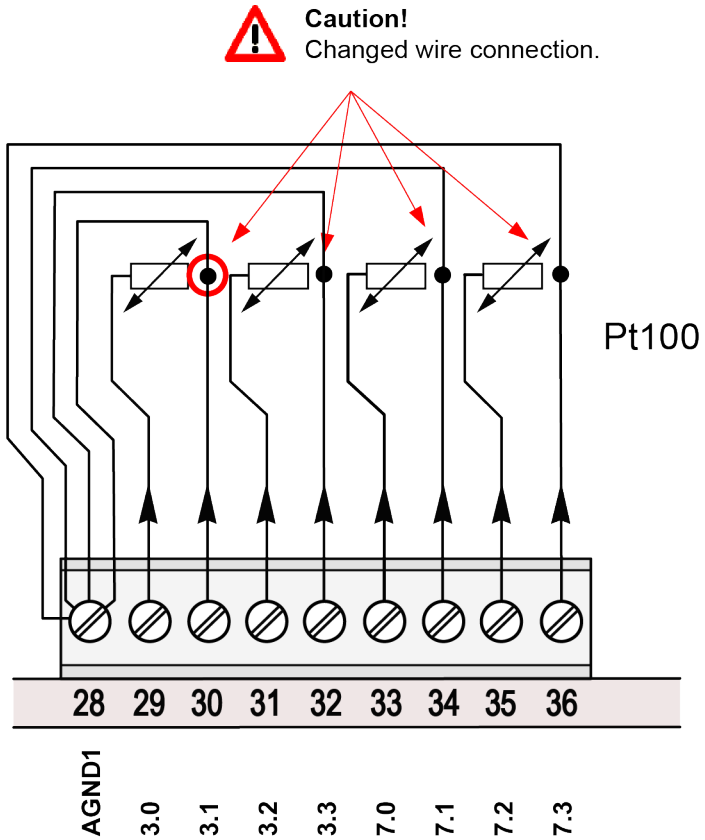


Fig. 763: Connection of Pt100 temperature sensors in 3-wire configuration

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Measuring values above range, below range and wire breaks are monitored and indicated.

For further information on protective function, error indication and diagnosis, please refer to the AC500 documentation.

Use of analog inputs as digital inputs

Data	Value
Input signal voltage • Signal 0 • Undefined signal state • Signal 1	24 V DC • -30 V ... +5 V • +5 V ... +13 V • +13 V ... +30 V
Input resistance	approx. 3.5 kΩ
Conversion cycle	1 ms (for 4 inputs + 2 outputs) 1 s when measuring with resistance thermometer Pt/Ni

ZP serves as reference signal for the inputs.

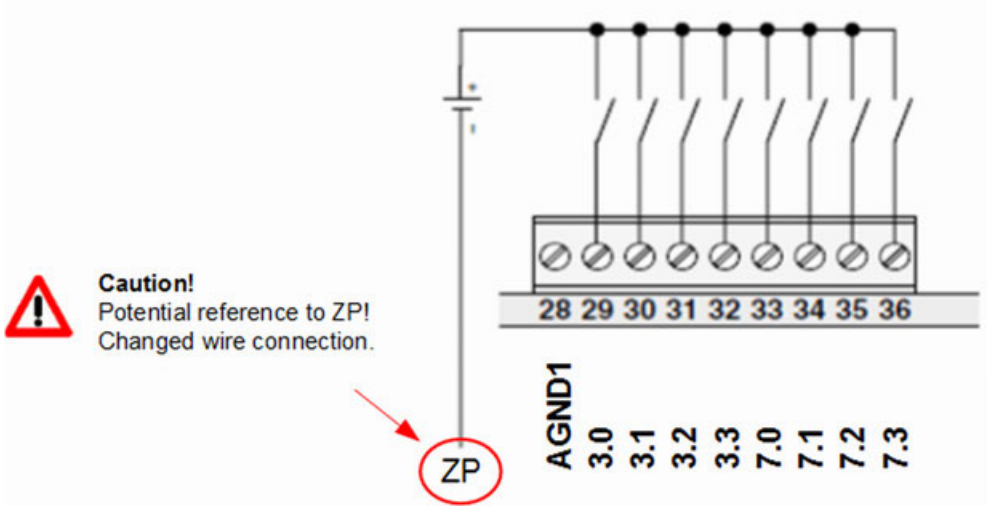


Fig. 764: Use of analog inputs as digital inputs

Connection of the 4 configurable analog outputs

Connector / Terminal	Pin	Assignment / Signal
X10 / AG.2	75	AGND2
X10 / 3.5	76	DA502 / AO0+
X10 / 3.6	77	DA502 / AO1+
X10 / 7.5	78	DA501 / AO0+
X10 / 7.6	79	DA501 / AO1+

UP must be connected to connectors X7, X8 and X9. The internal voltage supply to the ADC channels is no longer provided by L+ but by UP in the modules DA501 and DA502.

The arrangement of the 4 analog outputs is shown below:

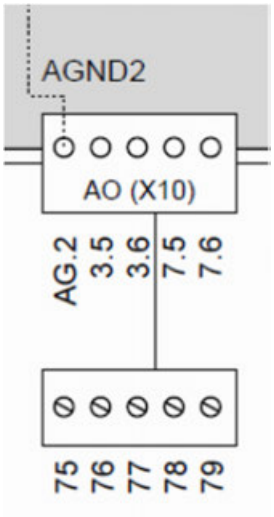


Fig. 765: Arrangement of the analog outputs

Resolution: 12 bit (+ sign)

The 4 analog outputs are not galvanically isolated and have a reference to ZP internally via PTC resistors.

Output areas ±10 V / 0 ... 20 mA / 4 ... 20 mA

No change to the wiring is necessary. The sensor is connected the same way as with the existing device 07KT98. Output load capability of voltage output: max. ±10 mA.

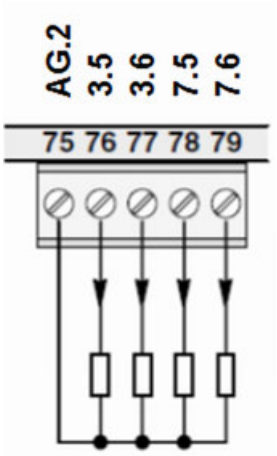


Fig. 766: Connection of output loads (voltage and current) to analog outputs

Battery and battery replacement

The AC31 adapters use another battery (lithium battery TA521).

For further information, please refer to the AC500 documentation.

Serial interface COM1

The serial interface COM1 is no longer available.

Programming can be performed via the serial interface COM2.

Serial interface DIAG

The serial interface DIAG is used for diagnosis and configuration. The DIAG interface is not galvanically isolated and thus only intended for connection with the Automation Builder.

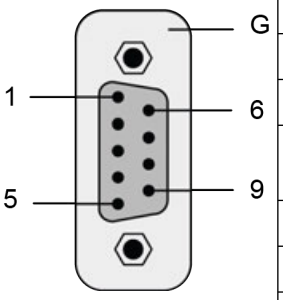
In the CPU or Automation Builder, the DIAG interface is accessed via the neutral FBP interface. Consequently, the information of the DIAG interface appears on the CPU display under the neutral FBP interface.

Connector / Pin	Assignment / Signal
DIAG / 1	Not connected
DIAG / 2	TX
DIAG / 3	M
DIAG / 4	RX
DIAG / 5	FE

Serial interface COM2

Connector / Pin	Assignment / Signal
COM2 / 1	FE
COM2 / 2	TX
COM2 / 3	RX
COM2 / 4	RTS
COM2 / 5	CTS
COM2 / 6	Not connected
COM2 / 7	Signal Ground
COM2 / 8	Signal Ground
COM2 / 9	+5 V

The assignment of the serial interface COM2 has not changed.

	PIN	Signal	Description
COM2 	G	Housing	FE
	1	FE	FE (shield)
	2	TxD	Transmit data (output)
	3	RxD	Receive data (input)
	4	RTS	Request to send (output)
	5	CTS	Clear to send (input)
	6	NC	-
	7	SGND	Signal ground (SGND)
	8	0 V out	-
	9	+5 V out	Reserved

Network interface

The existing device 07KT9x-AD has a parallel interface for connection to the communication module. Additional information upon request.

SmartMedia Card 07 MC 90

The content of this chapter is invalid. Another memory card is used in the CPU (Memory card MC502).

For further information on the memory card, please refer to the AC500 documentation.

High-speed counter

DA502

The standard fast counter input in 07KT9x-AD devices is located on connector X1 terminal X1/4.0/4.1 (DA502 /DC16/DC17). When using the counter inputs (X1/ 4.0/4.1), an external resistor 470 Ω / 1 W must be connected upstream. There are 10 operating modes available. The fast counter output is located on connector X1 terminal X1/4.2 (DA502 /DC18).

See also connection of the digital inputs/outputs *Table 296 “Connector (X1)” on page 3902*, Technical Data, *Table 303 “Data of the high-speed hardware counter installed (DA502)” on page 3926* and connection, *Table 303 “Data of the high-speed hardware counter installed (DA502)” on page 3926* .

DA501

From configuration point of view that is not forbidden to use also the fast counter coming from DA501 connector X9 terminal X9/8.0/8.1/8.2 (DA501 /DC16/DC17/DC18). When using the counter inputs (X9/ 8.0/8.1), an external resistor 470 Ω / 1 W must be connected upstream. There are 10 operating modes available. The fast counter output is located on connector X9 terminal X9/8.2 (DA501 /DC18).

See also connection of the digital inputs/outputs *Table 296 “Connector (X1)” on page 3902*, Technical Data, *Table 304 “Data of the high-speed hardware counter installed (DA501)” on page 3926* and connection, *Table 304 “Data of the high-speed hardware counter installed (DA501)” on page 3926*.

For further information on high-speed counters, please refer to the AC500 documentation.

See *Chapter 1.6.4.1.10 “Fast counters” on page 5498* and *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*.

Technical data 07KT9x-AD

The technical data described in the existing documentation (chapter 2.2.7) are invalid for the AC31 adapter and are replaced by the following data.

Further information: *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*

General data

Data	Value
Number of digital inputs	16
Number of digital outputs	16
Number of digital inputs/outputs	16
Number of analog inputs	8
Number of analog outputs	4
Supply voltages:	
-> UP	-> X7 / UP (pin 54), X7 / ZP (pin 45) -> X8 / UP (pin 64), X8 / ZP (pin 55) -> X9 / UP (pin 74), X9 / ZP (pin 65) See Fig. 750
-> Fuse for UP	16 A
-> Power consumption for UP	300 W (per 100W on X7, X8 and X9)

Data	Value
-> L+	X6 / L+ (pin 40), X6 / L+ (pin 41) X6 / M (pin 42), X6 / M (pin 43) See Fig. 750
-> Fuse for L+	10 A
-> Power consumption for L+	10 A
-> Galvanic isolation between UP and L+	Yes
Number of serial interfaces	1 COM2 (for diagnosis and programming with the Automation Builder software)
Number of serial interfaces (diagnosis)	1 DIAG (for diagnosis with the Automation Builder software)
Number of parallel interfaces	1 special interface for connection of an external communication module
Ethernet	10/100 base-TX, 1x RJ45 socket
Program memory	PM590 2MB
Resolution of the integrated real-time clock	1 s
Data of the high-speed hardware counter installed:	
-> Number of operating modes	-> 10
-> Counting range	-> 0 ... 4,294,967,295 (double word format, 32 bit)
-> Counting frequency	-> Depending on operating mode Note: At the counting input, an external resistor of 470 Ω / 1 W must always be connected upstream.
Cycle time for 1 instruction	Binary: min. 0.002 µs, word: min. 0.004 µs, floating point: min. 0.004 µs
Operating and error indications	Display via LEDs and CPU display. For detailed information, please refer to the AC500 documentation.
Connection technology	Detachable screw-type terminal blocks
Supply terminals, CS31 bus	max. 1 x 2.5 mm² or max. 2 x 1.5 mm²
All other terminals	max. 1 x 1.5 mm²

For further information, please refer to the existing documentation System description Advant Controller 31.

Supply of devices

Data	Value
Rated supply voltage	24 V DC
Supply voltages:	

Data	Value
-> UP	X7 / UP (pin 54), X7 / ZP (pin 45) X8 / UP (pin 64), X8 / ZP (pin 55) X9 / UP (pin 74), X9 / ZP (pin 65) See Fig. 750
-> Fuse for UP	10 A
-> Power consumption for UP	300 W (per 100W on X7, X8 and X9)
-> L+	X6 / L+ (pin 40), X6 / L+ (pin 41) X6 / M (pin 42), X6 / M (pin 43) See Fig. 750
-> Fuse for L+	10 A
-> Power consumption for L+	10 A
-> Protection against reversed voltage	Yes
-> Galvanic isolation between UP and L+	Yes

For further information, please refer to the existing documentation System description Advant Controller 31.

Lithium battery

Data	Value
Battery for buffering RAM contents and real-time clock	Lithium battery TA521
Buffer time at 25 °C	Typ. 3 years

Digital inputs

Data	Value
Number of channels per device	16
Connections	Connector X2 (terminals X5.0…X5.7) Connector X3 (terminals X6.0…X6.7)
Division of channels in groups	2 groups with 8 channels (not galvanically isolated!)
Voltage supply	UP (supplies module DA501 and 502)
Common reference potential:	
-> for group 1 (8 channels)	ZP (terminals 5.0 ... 5.07)
-> for group 2 (8 channels)	ZP (terminals 6.0 ... 6.07)

Data	Value
Galvanic isolation:	● Galvanic isolation from group to group is no longer available. ● Galvanic isolation from DA501 and DA502 (reference ZP) to the rest of the device (reference M) is available. ● On DA501 and DA502, all channels have the same potential ZP. Voltage supply UP/ZP. ● AGND1 and AGND2 of the analog channels are internally connected to ZP via PTC resistors Fig. 750.
Configurability of the inputs	Input delay configurable (0.1 ms, 1 ms, 8 ms and 32 ms). Default: 0.1 ms.
Channels for high-speed counters	*Chapter 1.6.2.3.3.4.1.3.4.6 “Digital inputs/ outputs” on page 3920* Channels for high-speed counters are implemented with the inputs/outputs (channels: 4.0 and 4.1).
Indication of the input signals	One yellow LED each per channel. The LED corresponds functionally to the input signal.
Input signal voltage:	24 V DC
-> 0 signal	-3 V…+5 V
-> Undefined signal state	+5 V ... + 15 V
-> 1 signal	+15 V ... + 30 V
Input current per channel:	
-> Input voltage = +24 V	Typ. 5.0 mA
-> Input voltage = + 5 V	> 1 mA
-> Input voltage = +13 V	> 2 mA
-> Input voltage = +30 V	< 8.0 mA
Max. cable length unshielded	600 m
Max. cable length shielded	1000 m

For further information, please refer to the existing documentation System description Advant Controller 31.

Digital outputs

Data	Value
Number of channels per device	16 high-side switches
Connections	Connector X7 (terminals 1.0 ... 1.7) Connector X8 (terminals 2.0 ... 2.7)
Division of channels in groups	2 groups with 8 channels (not galvanically isolated!)
Common voltage supply	UP (supplies module DA501 and 502)
Common reference potential ZP:	

Data	Value
-> for group 1	ZP (terminals 1.0 ... 1.7)
-> for group 2	ZP (terminals 2.0 ... 2.7)
Galvanic isolation	● Galvanic isolation from group to group is no longer available. ● Galvanic isolation from DA501 and DA502 (reference ZP) to the rest of the device (reference M). ● On DA501 and DA502, all channels have the same potential ZP. Voltage supply UP/ZP. ● AGND1 and AGND2 of the analog channels are internally connected to ZP via PTC resistors Fig. 750.
Indication of the output signals	One yellow LED each per channel. The LED corresponds functionally to the output signal.
Output current:	
-> Rated value	500 mA at UP = 24 V
-> Residual current at 0 signal	< 0.5 mA
Demagnetization with inductive load	Internally via varistor
Switching frequency with inductive load	max. 0.5 Hz
Switching frequency with lamp load	max. 11 Hz at max. 5 W
Max. cable length	1000 m (shielded) 600 m (unshielded)
Short-circuit proof / overload proof	Yes
Protection against reversed voltage of process supply voltage	Yes
Resistance to feedback against 24 V signals	Yes
Total load current (all DO channels, 1.0...1.7 max. 4A and 2.0…2.7)	max. 4 A
Total load current (all DC channels, 4.0..4.7 max. 8A and 8.0…8.7)	max. 8 A
Total load current (via UP) 16 DO channels and 16 DC channels	max. 12 A (all UP terminals must be connected)

For further information, please refer to the existing documentation System description Advant Controller 31.

Digital inputs/outputs

Data	Value
Number of channels per device	16 inputs/outputs
Connections	Connector X1 (terminals 4.0 ... 4.7) Connector X9 (terminals 8.0 ... 8.7)

Data	Value
Division of channels in groups	2 groups of 8 channels each Group 1: terminals 4.0 ... 4.7 Group 2: terminals 8.0 ... 8.7
Common reference potential ZP	All digital I/O channels of the DA501 and DA502 module
Common voltage supply	UP (supplies DA501 and DA502 module)
Galvanic isolation	Galvanic isolation from group to group is no longer available. Galvanic isolation from DA501 and DA502 (reference ZP) to the rest of the device (reference M). On DA501 and DA502, all digital channels have the same potential ZP. AGND1 and AGND2 of the analog channels are internally connected to ZP via PTC resistors. Fig. 750
Configurability of the inputs:	
-> Input delay	Typically 0.1 ms, configurable from 0.1 ms to 32 ms
Indication of the input/output signals	1 yellow LED per channel. The LED is ON in "High" signal state (1 signal)
Input signal voltage (when used as input)	*Further information on page 3904*.
-> 0 signal	-3 V ... + 5 V
-> 1 signal	+15 V ... + 30 V
Input current per channel	*Chapter 1.6.2.3.3.4.1.3.4.4 “Digital inputs” on page 3918*.
Output current / switching frequency / inductive loads	*Chapter 1.6.2.3.3.4.1.3.4.5 “Digital outputs” on page 3919*.
Total load current (all DC channels, 4.0..4.7 max. 8A and 8.0…8.7)	max. 8 A
Total load current (all DO channels, 1.0...1.7 max. 4A and 2.0…2.7)	max. 4 A
Total load current (via UP) 16 DO channels and 16 DC channels	max. 12 A (all UP terminals must be connected)
Max. cable length	*Chapter 1.6.2.3.3.4.1.3.4.4 “Digital inputs” on page 3918* *Chapter 1.6.2.3.3.4.1.3.4.5 “Digital outputs” on page 3919*

For further information, please refer to the existing documentation System description Advant Controller 31.

Analog inputs

Data	Value
Number of channels per device	8
Connections	Connector X4 (terminals 3.0 … 3.3 and 7.0… 7.3)
Division of channels in groups	1 group with 8 channels (evenly distributed among the modules DA501 and DA502 internally)
Common reference potential for analog inputs (8 channels)	AGND1 (terminals 3.0 … 3.3 and 7.0…7.3) Caution: internal reference to ZP via PTC resistors Fig. 750
Galvanic isolation	No Fig. 750
Max. permissible potential difference between terminal ZP (minus the supply voltage) and terminals AGND (minus the analog inputs and outputs)	± 1 V Caution: The internal reference is no longer M but ZP. *Chapter 1.6.2.3.3.4.1.3.1 “Connections” on page 3897*
Indication of the input signals	8 yellow LEDs to indicate the signal statuses of the analog inputs (4 LEDs per DA501 module and DA502 module)
Configurability (optional per channel) *Chapter 1.6.2.3.3.4.1.3.1.2 “Connection of the supply voltage” on page 3898*	0 ... 10 V, ± 10 V (also with differential signal), 0 ... 20 mA, 4 ... 20 mA Pt100 -50 ... +400 °C and -50 ... +70°C Pt1000 -50…+400 °C (2-wire and 3-wire configuration) Digital input
Input resistance per channel:	
-> Voltage input	> 100 kΩ
-> Current input	approx. 330 kΩ
-> Digital input	approx. 3.5 kΩ
Time constant of the input filter	Voltage: 100 µs, current: 100 µs
Conversion cycle	1 ms (for 4 inputs and 2 outputs) 1 s when measuring with resistance thermometer Pt/Ni

The "Examples for the conversion cycle" from the existing documentation 07KT98 are no longer valid.

Data	Value
Resolution in bits:	
-> Ranges	±10 V, 0 ... 10 V 12 bit plus sign
-> Ranges	0 ... 20 mA, 4 ... 20 mA 12 bit without sign
-> Range	Pt100, Pt1000, Ni1000: 0.1 °C

Data	Value
Resolution in mV, µA:	
-> Range	±10 V approx. 2.5 mV
-> Range	0 ... 10 V approx. 2.5 mV
-> Range	0 ... 20 mA approx. 5 µA
-> Range	4 ... 20 mA approx. 4 µA
Relationship between input signal and hex code	-100 % ... 0 ... +100 % = 9400H ... 0000H ... 6C00H (-27648 ... 0 ... 27648 decimal)
Conversion error of the analog values due to non-linearity. Adjustment error on delivery and resolution in the nominal range	Typ. 0.5 %, max. 1 %
Use as digital input:	
-> Signal 0	-30 V ... +5 V
-> Undefined signal state	+5 V ... +13 V
-> Signal 1	+13 V ... +30 V
Max. cable length 2-core shielded and conductor cross section > 0.14 mm²	100 m

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies a current in excess of 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or sensors without current peaks higher than 25 mA. If this is not possible, protect the input by connecting a 10-volt zener diode in parallel to I+ and I-.

For further information, please refer to the existing documentation System description Advant Controller 31.

Analog outputs

Data	Value
Number of channels per device	4
Connections	Connector X10 (terminals 3.5, 3.6, 7.5 and 7.6)
Reference potential	AGND2 (terminals 3.5, 3.6, 7.5 and 7.6)
Galvanic isolation	No Fig. 750

Data	Value
Max. permissible potential difference between terminal ZP (minus the supply voltage) and terminals AGND (minus the analog inputs and outputs)	± 1 V Caution: The internal reference is no longer M but ZP. Fig. 750
Indication of output signal	4 yellow LEDs to indicate the signal statuses of the analog outputs (2 LEDs each at DA501 and DA502)
Output signal ranges (configurable)	-10 V ... 0, 0 ... +10 V 0 ... 20 mA 4 ... 20 mA
Output load capability of voltage output	max. ±10 mA
Resolution	12 bit (+ sign)
Resolution (1 LSB), range 10 V ... 0, 0 ... +10 V	approx. 5 mV
Relationship between output signal and hex code	-100 % ... 0 ... +100 % = 9400H ... 0000H ... 6C00H (-27648 ... 0 ... 27648 decimal)
Conversion cycle	1 ms (for 4 inputs + 2 outputs) 1 s when measuring with resistance thermometer Pt/Ni
Conversion error of the analog values due to non-linearity Adjustment error on delivery and resolution in the nominal range	Typ. 0.5 %, max. 1 %
Max. cable length, 2-core shielded and conductor cross section > 0.14 mm²	100 m

For further information, please refer to the existing documentation System description Advant Controller 31.

Connection of the serial interfaces COM2

The COM1 interface is no longer available. The assignment of the COM2 interface remains the same as in the existing device. Programming in Automation Builder can be performed via the COM2 interface.

Data	Value
Interface standard	EIA RS-232
Programming	07KT94-ARC-AD: 907 PC 331 07KT98-ARC-AD: Automation Builder
Program change	07KT94-ARC-AD: 907 PC 331 07KT98-ARC-AD: Automation Builder
Man-Machine Communication	Yes, e.g. via Automation Builder

Data	Value
Galvanic isolation	Fig. 750
Potential differences	In order to avoid potential differences between the replacement device 07KT98-AD and the peripheral devices connected to COM2, these devices are supplied by the socket in the switchgear cabinet.
Terminal assignment and description of the COM2 interface	↳ *Chapter 1.6.2.3.3.4.1.3.1.11 “Serial interface COM2” on page 3915*

For further information, please refer to the existing documentation System description Advant Controller 31.

Serial interface DIAG

Data	Value
Programming	07KT94-ARC-AD: 907 PC 331 07KT98x-AD: Automation Builder
Program change	07KT94-ARC-AD: 907 PC 331 07KT98x-AD: Automation Builder
Galvanic isolation	No Fig. 750

Connection to the CS31 bus

When configuring the CS31 bus interface (connector X5), select the COM1 interface of CPU PM590 in Automation Builder.

The shield connection must be internally connected to FE.

Data	Value
Interface standard	EIA RS-485
Connection:	
-> as master PLC	Yes
-> as slave PLC	No
Setting of the CS31 bus module address	No, the master has no module address
Galvanic isolation	Yes Fig. 750
Terminal assignment and description of the CS31 bus interface	↳ *Chapter 1.6.2.3.3.4.1.3.1.3 “Connection for CS31 bus” on page 3898* Note that the shield connection is internally connected to FE.

LED display

Data	Value
LEDs for signaling:	
-> State of digital inputs	1 yellow LED per channel
-> State of digital outputs	1 yellow LED per channel
-> State of digital inputs/outputs	1 yellow LED per channel
-> Supply voltage available (Supply)	1 green LED
-> Battery	1 red LED (name: ERR) at the CPU
-> Program is running (RUN)	1 green LED
-> Controller-specific errors	1 red LED (name: ERR) at the CPU
-> CS31 bus	Indication on CPU display under COM1 (CS31 bus is assigned to COM1 within the CPU)
-> Overload / short circuit of digital outputs	Red LEDs on modules DA501/ DA502 and at the CPU via ERR-LED. An indication on the display is possible.

High-speed hardware counter

At the counting input, an external resistor of 470 Ω / 1 W must always be connected upstream. For further information on high-speed counters, please refer to the AC500 documentation.

Table 303: Data of the high-speed hardware counter installed (DA502)

Data	Value
Number of operating modes	10
Counting range	0 ... 4,294,967,295 (double word format, 32 bit)
Counting frequency	Depending on operating mode
Used inputs	Connector X1, terminals 4.0 and 4.1
Used outputs	Connector X1, terminal 4.2

Table 304: Data of the high-speed hardware counter installed (DA501)

Data	Value
Number of operating modes	10
Counting range	0 ... 4,294,967,295 (double word format, 32 bit)
Counting frequency	Depending on operating mode
Used inputs	Connector X9, terminals 8.0 and 8.1
Used outputs	Connector X9, terminal 8.2

Mechanical data

Data	Value
Width x height x depth	Replacement device: 239.5 x 138 x approx. 80.9 mm Existing device: 240 x 140 x 85 mm
Weight	Replacement device 07KT94-ARCNET: 910 g Replacement device 07KT98-ARCNET: 945 g Existing device: 1.6 kg
Dimensions for mounting	See operating and assembly instructions of the replacement device (3ADR020082M0401)

Ordering data

Order No.	Scope of delivery
1SAP 801 000 R0061	CPU: 07KT94-ARC-AD
1SAP 801 400 R0060	CPU: 07KT98-ARC-AD
1SAP 801 100 R0062	CPU: 07KT98-ARC-DP-AD
1SAP 801 200 R0067	CPU: 07KT98-ARC-ETH-AD
1SAP 801 300 R0072	CPU: 07KT98-ETH-DP-AD
1SAP 801 500 R0062	CPU: 07KT98-ARC-ETH-DP-AD

ARCNET communication module

Central units with integrated ARCNET communication module (Attached Resource Computer Network):

- 07KT94-ARC-AD
- 07KT98-ARC-AD
- 07KT98-ARC-DP-AD
- 07KT98-ARC-ETH-AD
- 07KT98-ARC-ETH-DP-AD

Technical data

In the replacement device, addresses cannot be set via DIP switch. Instead, the ARCNET interface is configured in the Automation Builder. The ARCNET address can also optionally be set via the display.

Data	Value
Connector	ARC (BNC connector)
ARCNET interface	For coaxial cable connection

Data	Value
Recommended system cable	Cable RG 62 A/U (characteristic impedance 93 Ω) Cable length 300 m in case of ARCNET bus with 8 stations. For further information, please refer to the AC500 documentation (chapter ARCNET).
Signaling	Indication on CPU display
Galvanic isolation	Yes Fig. 750

ARCNET short description

The ARCNET interface is configured in the Automation Builder. For further information on the ARCNET interface for the respective CPU, please refer to the AC500 documentation.

ARCNET system

The general information about the ARCNET system is still valid. For further information on ARCNET, please refer to the AC500 documentation.

PROFIBUS DP communication module

Central units with an integrated PROFIBUS communication module:

- 07KT98-ARC-DP-AD
- 07KT98-ETH-DP-AD
- 07KT98-ARC-ETH-DP-AD

Technical data

Data	Value
Connector	9 pin D-sub socket
PROFIBUS interface	EIA RS-485 according to EN 50170
Recommended system cable	Dual twisted, shielded pair cable (characteristic impedance 135 to 165 Ω) Max. line length 1000 m with a transmission rate of 187.5 Kbps For further information, please refer to the AC500 documentation (chapter PROFIBUS).
Signaling	With 5 LEDs
Galvanic isolation	Yes Fig. 750

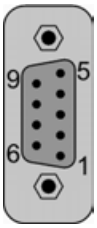
PROFIBUS short description

The PROFIBUS interface is configured in the Automation Builder. For further information on the PROFIBUS interface for the respective CPU used, please refer to theAC500 documentation.

The PROFIBUS system

The general information about the PROFIBUS system is still valid. For further information on PROFIBUS, please refer to the AC500 documentation.

Pin Assignment

	Pin	Signal	Description
	1	NC	Not connected
	2	NC	Not connected
	3	RxD/TxD-P	Receive/Transmit positive
	4	CNTR-P	Control signal for repeater, positive
	5	DGND	Reference potential for data exchange and +5 V
	6	VP	+5 V (power supply for the bus terminating resistors)
	7	NC	Not connected
	8	RxD/TxD-N	Receive/Transmit negative
	9	NC	Not connected

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

Ethernet communication module

Central units with an integrated Ethernet communication module:

- 07KT98-ARC-ETH-AD
- 07KT98-ETH-DP-AD
- 07KT98-ARC-ETH-DP-AD

Technical data

Data	Value
Connector	RJ45 socket
Ethernet interface	10/100 Base-TX
Recommended system cable	For detailed information, please refer to the AC500 documentation (Ethernet chapter).
Signaling	Indication on the CPU display
Galvanic isolation	Yes Fig. 750

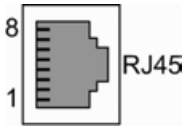
Ethernet short description

The Ethernet interface is configured in the Automation Builder. For further information on the Ethernet interface for the respective CPU used, please refer to the AC500 documentation.

Ethernet system

The general information about the Ethernet system is still valid. For further information on Ethernet, please refer to the AC500 documentation.

Pin assignment

	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Replacement devices: I/O modules

For AC31 devices of the 90 series, AC31 adapters (replacement devices) are available for the exchange of individual I/O modules.

Replacement device 07AC91-AD

Introduction

Fig. 767: 3ADR331193S0015_07AC91-AD

The replacement device 07AC91-AD of the AC31 adapter series replaces the existing device 07AC91 of the AC31/90 series in operating mode **8 bit**. The replacement device 07AC91-AD2 is available for operating mode **12 bit**.

During the development of the replacement device, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07AC91 remains valid and serves as reference *(system description Advant Controller 31)*.

The document structure of this document is based on the document structure of the existing documentation.

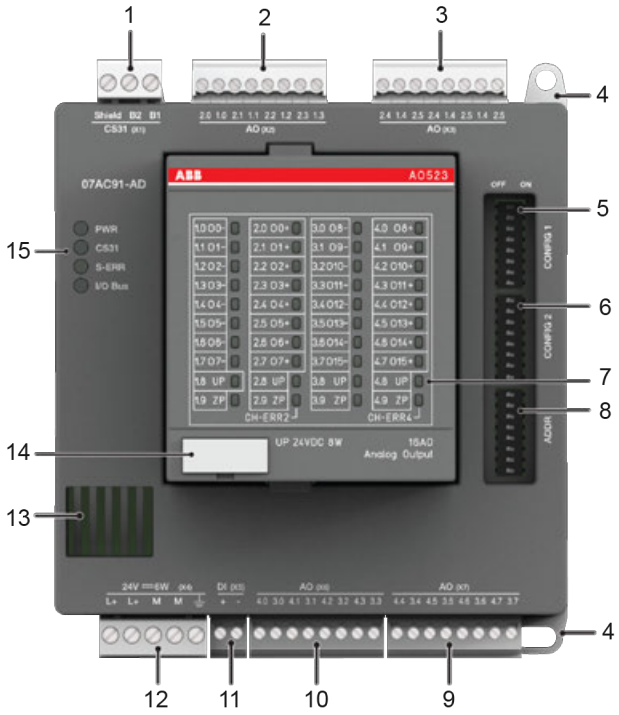
This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement device 07DC91-AD can be found in the operating and assembly instructions of device 07DC91-AD: 3ADR020084M0401. Please note that for the existing device 07AI91 no separate operating and assembly instructions are available.

Please also observe the system data as well as the information on CS31 bus ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Device configuration



1 Connection for CS31 bus (X1)
2 Analog outputs (X2): 0 ... 10 V, 0 ... 20 mA
3 Analog outputs (X3): 0 ... 10 V
4 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
5 DIP switch for CONFIG1
6 DIP switch for CONFIG2
7 Status LEDs for AO523
8 DIP switch for ADDR
9 Analog outputs (X7): 0 ... 10 V
10 Analog outputs (X6): 0 ... 10 V, 0 ... 20 mA
11 Enabling input for analog outputs (X5)
12 Supply 24 V DC (incl. AO523)
13 Ventilation
14 TA525: Label
15 4 Status LEDs

LED display

The LED display on the replacement device is changed:

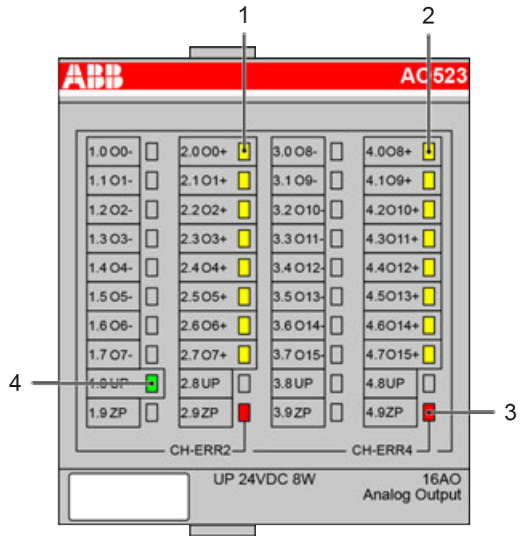


Fig. 768: AO523

No.	Display of module
1	8 yellow LEDs to indicate the signal status of the analog inputs (X2 and X3)
2	8 yellow LEDs to indicate the signal status of the analog inputs (X6 and X7)
3	2 red LEDs to indicate errors (of AO523 module)
4	1 green LED to indicate the status of the supply voltage of the AO523 module (is supplied via X4)

The replacement device does not provide a test button to measure functionality.

Connections

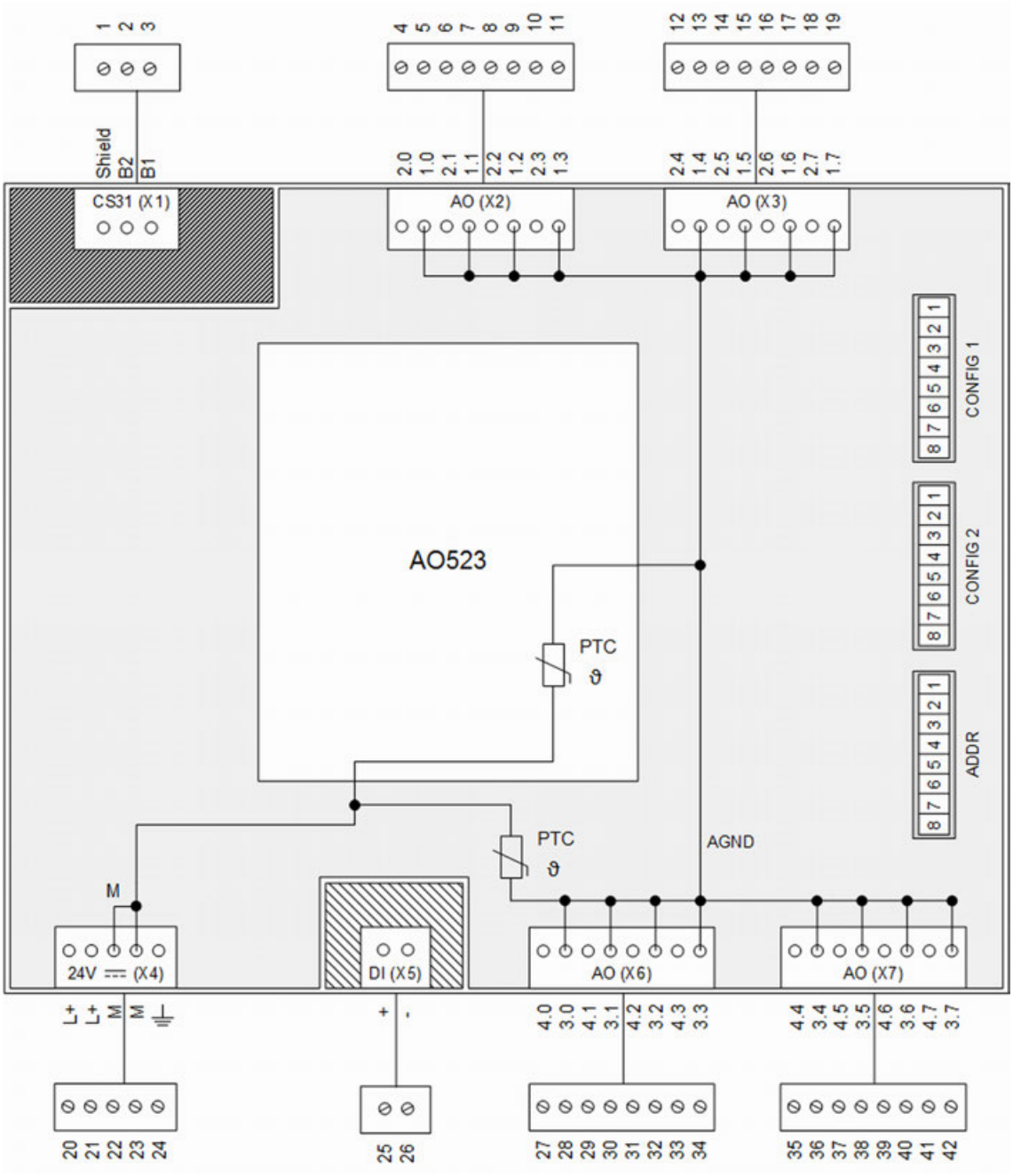


Fig. 769: Connection

Please observe the information contained in the existing documentation. In section "Fig. 5.4-2: Connection of the analog input/output module 07AC91", only the information concerning operating mode 8 bit is relevant for the replacement device 07AC91-AD.

Table 305: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	No internal connection
X1 / B2	2	BUS 2
X1 / B1	3	BUS 1

Table 306: Pin assignment AO (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / 2.0	4	AO523 / O0+
X2 / 1.0	5	AO523 / O0- (AGND)
X2 / 2.1	6	AO523 / O1+
X2 / 1.1	7	AO523 / O1- (AGND)
X2 / 2.2	8	AO523 / O2+
X2 / 1.2	9	AO523 / O2- (AGND)
X2 / 2.3	10	AO523 / O3+
X2 / 1.3	11	AO523 / O3- (AGND)

Table 307: Pin assignment AO (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / 2.4	12	AO523 / -
X3 / 1.4	13	AO523 / O4- (AGND)
X3 / 2.5	14	AO523 / -
X3 / 1.5	15	AO523 / O5- (AGND)
X3 / 2.6	16	AO523 / -
X3 / 1.6	17	AO523 / O6- (AGND)
X3 / 2.7	18	AO523 / -
X3 / 1.7	19	AO523 / O7- (AGND)

Table 308: Pin assignment 24 V DC 9W (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / L+	20	L+
X4 / L+	21	L+
X4 / M	22	M
X4 / M	23	M
X4 / FE	24	FE

Table 309: Pin assignment DI (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / +	25	IN+
X5 / -	26	IN- (galvanic isolated ground)

Table 310: Pin assignment AO (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / 4.0	27	AO523 / O8+
X6 / 3.0	28	AO523 / O8- (AGND)
X6 / 4.1	29	AO523 / O9+
X6 / 3.1	30	AO523 / O9- (AGND)
X6 / 4.2	31	AO523 / O10+
X6 / 3.2	32	AO523 / O10- (AGND)
X6 / 4.3	33	AO523 / O11+
X6 / 3.3	34	AO523 / O11- (AGND)

Table 311: Pin assignment AO (X7)

Connector / Terminal	Pin	Assignment / Signal
X7 / 4.4	35	AO523 / O12+
X7 / 3.4	36	AO523 / O12- (AGND)
X7 / 4.5	37	AO523 / O13+
X7 / 3.5	38	AO523 / O13- (AGND)
X7 / 4.6	39	AO523 / O14+
X7 / 3.6	40	AO523 / O14- (AGND)
X7 / 4.7	41	AO523 / O15+
X7 / 3.7	42	AO523 / O15- (AGND)

The signals Ox- are internally linked to an AGND area. The potential AGND is connected to the potential M via PTC resistors. Potential difference AGND to M ±1 V (max.).

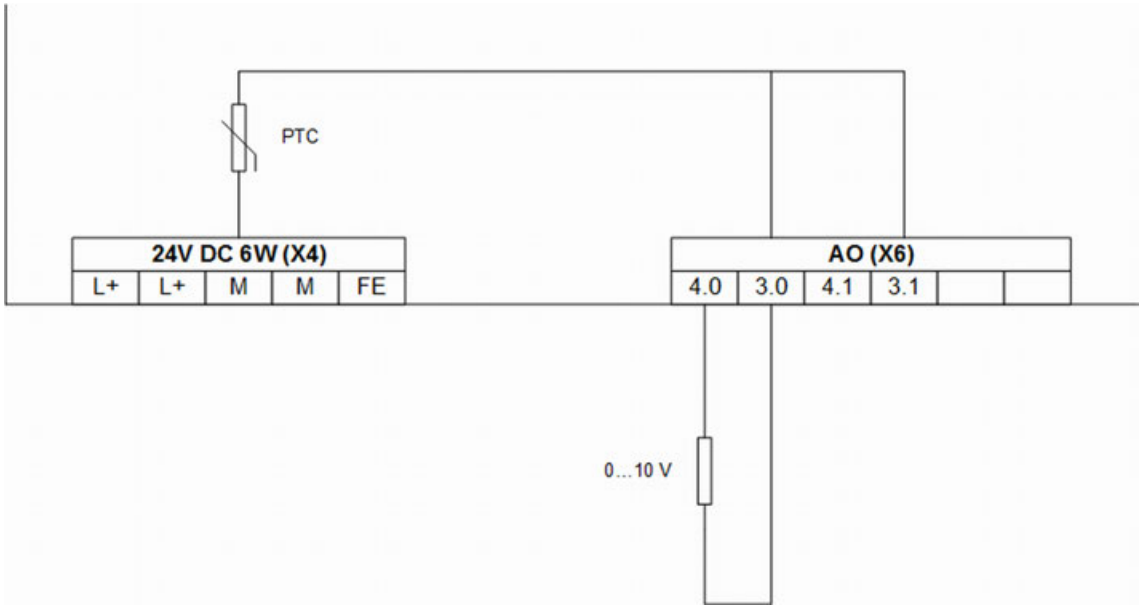


Fig. 770: Voltage output

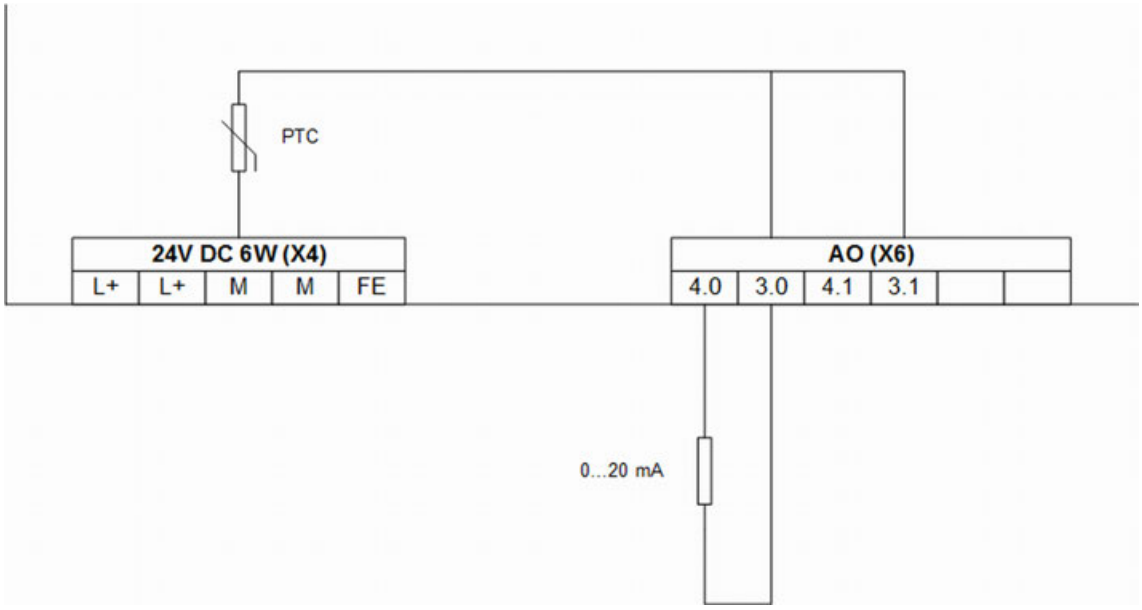


Fig. 771: Current output

Analog signal lines must be routed in shielded cables. The shield must be grounded on both sides and should be grounded to replacement device and signal source / signal sink as close as possible.

Configuration

The existing device had a DIP switch on the upper printed circuit board. Since the replacement device is not equipped with an upper printed circuit board, the white DIP switch is arranged on the lower printed circuit board instead.

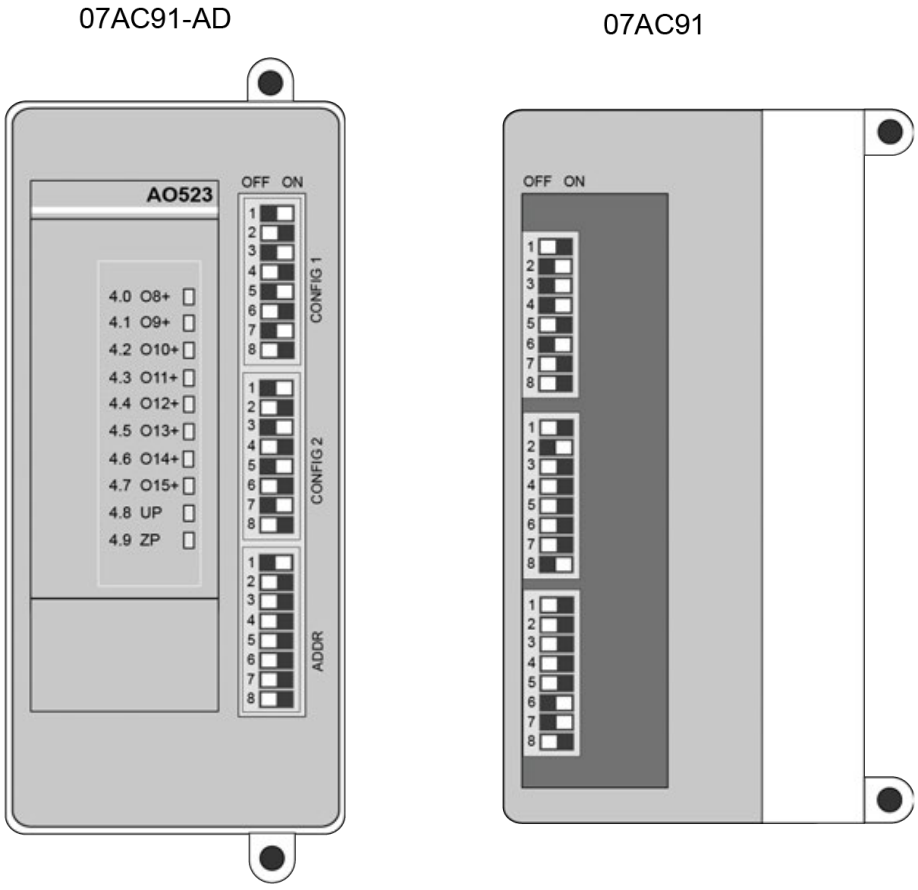


Table 312: Example configuration for 07AC91-AD:

Config 1	All output channels on voltage.
Config 2	All output channels on voltage.
ADDR	8-bit mode, without range monitoring, CS31 address 0 and channel number ≤ 7.

Configuration areas with (white) DIP switches

Please observe the following:

- All channels must be configured as outputs.
- The position of the DIP switches are read by the device only once after the supply voltage has been connected.

Config 1	The DIP switches for the channels 1, 3, 5 and 7 must be set to ON (configuration as outputs). **A configuration as inputs is not permitted.**
	The DIP switches for the channels 2 and 4 can be set as desired. The outputs 0..3 may be set to OFF (voltage) or ON (current).
	The DIP switches for channels 6 and 8 must be set to OFF. The outputs 4..7 must be set to OFF (voltage). **The setting to ON (current) is not permitted.**
Config 2	The DIP switches for the channels 1, 3, 5 and 7 must be set to ON (configuration as outputs). **A configuration as inputs is not permitted.**
	The DIP switch position for the channels 2 and 4 can be set as desired. The outputs 8..11 may be set to OFF (voltage) or ON (current).
	The DIP switches for the channels 6 and 8 must be set to OFF. The outputs 12..15 must be set to OFF (voltage). **The setting to ON (current) is not permitted.**
ADDR	The DIP switch for channel 1 must be set to ON (8-bit mode).
	The DIP switch for channel 2 can be set as desired (no functionality).
	The DIP switch for channel 3 can be set as desired for range monitoring.
	The DIP switches for the channels 4-7 can be set as desired for the CS31 address.
	The DIP switch for channel 8 must be set to OFF for CS31 channels ≤ 7. **Channels > 7 are not supported. The outputs on connector X3 and X7 cannot be configured as current outputs.**

For further information, please refer to the existing documentation System description Advant Controller 31.

Measuring ranges of the analog channels

For the replacement device 07AC91-AD, only the operating mode "8 bit" is relevant.

The outputs of the S500 module AO523 have a 12 bit resolution. The values that are to be transmitted via the CS31 bus bus and output by the replacement device have only a 8 bit resolution. For this reason, the overall resolution achieved is only 8 bits.

Addressing

The function of the address DIP switch 8 (channel No. ≤ 7 or channel No. > 7) is no longer supported.

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

Table 313: CS31 bus

Type	Byte	Position in WORD	Connector / Terminal
WORD output (received) 0	1	High	X2 / 2.1
	2	Low	X2 / 2.0
WORD output (received) 1	3	High	X2 / 2.3
	4	Low	X2 / 2.2
WORD output (received) 2	5	High	X3 / 2.5
	6	Low	X3 / 2.4
WORD output (received) 3	7	High	X3 / 2.7
	8	Low	X3 / 2.6
WORD output (received) 4	9	High	X6 / 4.1
	10	Low	X6 / 4.0
WORD output (received) 5	11	High	X6 / 4.3
	12	Low	X6 / 4.2
WORD output (received) 6	13	High	X7 / 4.5
	14	Low	X7 / 4.4
WORD output (received) 7	15	High	X7 / 4.7
	16	Low	X7 / 4.6

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.
- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

The RAM is checked during the initialization of the device. In addition, the firmware in the Flash memory is checked by means of a checksum during initialization. When the control system (PLC/central unit) is stopped during normal operation, the outputs of the device are switched off. The outputs are also switched off in case of a malfunction of the CS31 bus bus.

Diagnosis and display

LEDs are used for diagnosis and display purposes. In addition, some diagnosis information can be transmitted via the CS31 bus.

The replacement device does not provide a test button to measure functionality.

Table 314: Diagnosis information of the CS31 bus bus

Channel	Error code (CODESYS)	Error code (CS31 bus)	Description
Device error:			
0	43	1	Internal error

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

Table 315: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.
- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 316: S500 module AO523 LEDs

LED	Status	Color	LED off	LED on	LED flashes
O0+…O7+ O8+…O15+ (see No. 1 + 2 in the following figure)	Analog outputs	Yellow	Output is not activated	Output is activated (brightness depends on value of analog signal).	-
Error indication left (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	-

LED	Status	Color	LED off	LED on	LED flashes
Error indication right (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	-
Indication supply voltage (see No. 4 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-

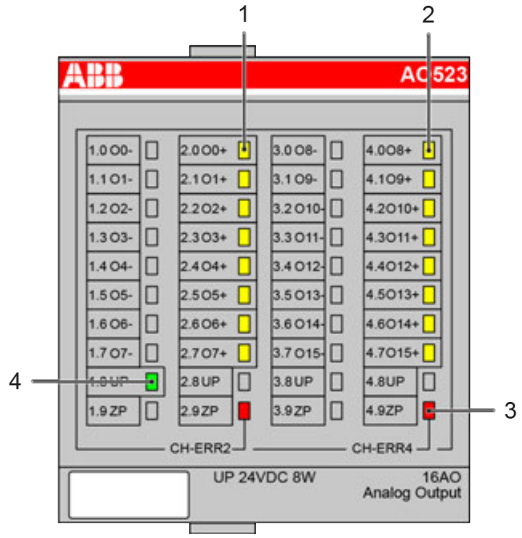


Fig. 772: AO523

Technical data

This section provides additional information on section ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*. In case of doubt, the following information applies.

For the device 07AC91-AD, only the operating mode "8 bit" is relevant.

Technical data of the complete device

Data	Value
Process voltage:	
-> Connections	X4/L+ (pin 20), X4/L+ (pin 21), X4/M (pin 22), X4/M (pin 23)
-> Fuse for L+	10 A, fast acting
- Galvanic isolation	No
Current consumption:	
-> via L+	0.19 A + output load

Data	Value
- Inrush current via L+ (when voltage is switched on)	0.18 A²s
Power consumption	Replacement device: 9 W Existing device: 5 W

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Technical data of the binary input

Data	Value
Input current at input voltage +24 V	Typ. 6 mA
Protection against reversed voltage	Yes
Overvoltage protection	No

The enabling input is a proprietary input.

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical data of the analog outputs

Data	Value
Connections	X2 / 2.0, X2 / 2.1, X2 / 2.2, X2 / 2.3, X3 / 2.4, X3 / 2.5, X3 / 2.6, X3 / 2.7, X6 / 4.0, X6 / 4.1, X6 / 4.2, X6 / 4.3, X7 / 4.4, X7 / 4.5, X7 / 4.6, X7 / 4.7
Reference connections (AGND)	X2 / 1.0, X2 / 1.1, X2 / 1.2, X2 / 1.3, X3 / 1.4, X3 / 1.5, X3 / 1.6, X3 / 1.7, X6 / 3.0, X6 / 3.1, X6 / 3.2, X6 / 3.3, X7 / 3.4, X7 / 3.5, X7 / 3.6, X7 / 3.7
Type of outputs	Voltage unipolar, current unipolar
Configurability	No inputs are available Replacement device: 8 current outputs Existing device: 16 current outputs
Output load capability, as voltage output	Replacement device: ± 10 mA Existing device: +20 mA, -10 mA

Data	Value
Short-circuit-proof	Yes
External supply protection	Up to 30 V DC

For further information, please refer to the existing documentation System description Advant Controller 31.

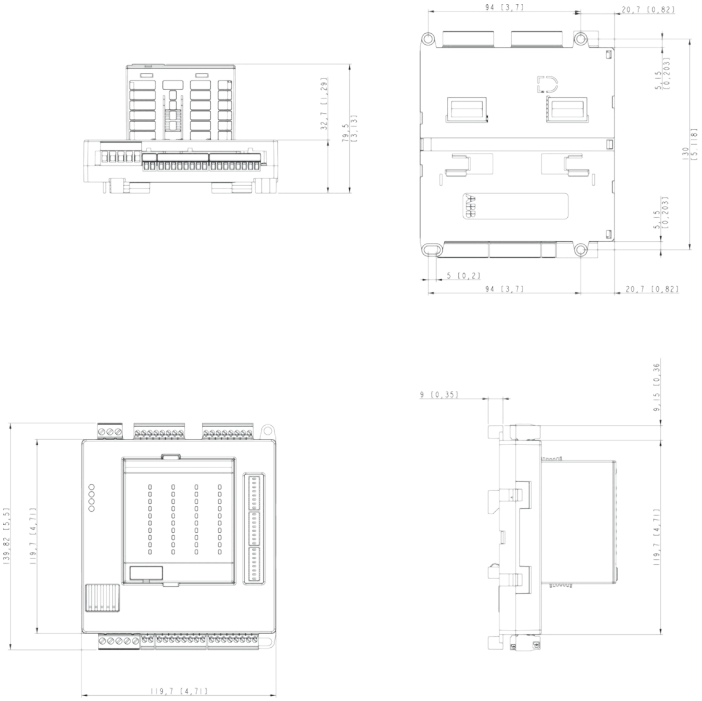
Connection to the CS31 bus

Data	Value
Connections	X1/B2, X1/B1
CS31 bus type	03 (analog output)
Terminating resistor	Not available (must be provided externally if needed)

Mechanical data

Data	Value
Width x height x depth	Replacement device: 120 x 140 x approx. 80 mm Existing device: 120 x 140 x 85 mm
Weight	Replacement device: 363 g Existing device: 450 g
Dimensions for mounting	See assembly instructions 07AC91-AD (3ADR020084M0401)

Mounting information



The dimensions are in mm and in brackets in inch.

The dimensions for the assembly holes are the same for the replacement device and the existing device.

To assemble or disassemble the replacement device, grab the device at the housing and not directly at the S500 module.

Ordering data

Order No.	Scope of delivery
1SAP 800 000 R0010	Analog output module 07AC91-AD 1 2-pole terminal block (3.81 mm grid space) 1 3-pole terminal block (5.08 mm grid space) 1 5-pole terminal block (5.08 mm grid space) 4 8-pole terminal blocks (3.81 mm grid space)

Replacement device 07AC91-AD2

Introduction

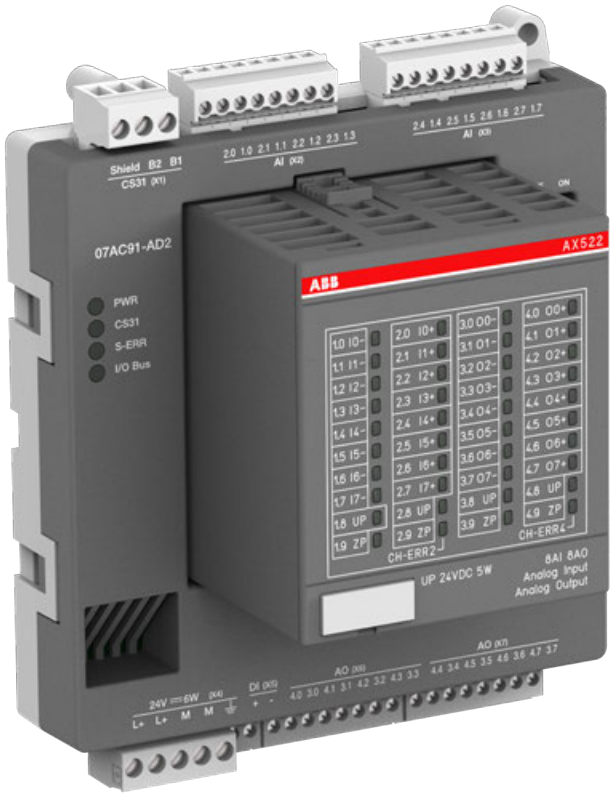


Fig. 773: 3ADR331194S0015_07AC91-AD2

The replacement device 07AC91-AD2 of the AC31 adapter series replaces the existing device 07AC91 of the AC31/90 series in operating mode **12 bit**. The replacement device 07AC91-AD is available for operating mode **8 bit**.

During the development of the replacement device, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07AC91 remains valid and serves as reference *(system description Advant Controller 31)*. The document structure of this document is based on the document structure of the existing documentation.

This document adds the following points to the still valid existing documentation:

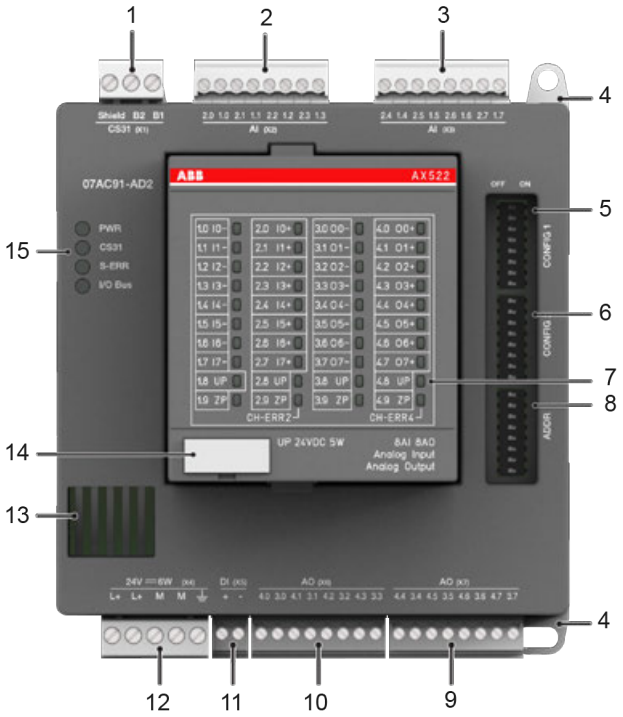
- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement device 07AC91-AD2 can be found in the operating and assembly instructions of device 07AC91-AD2: 3ADR020085M0401. Please note that for the existing device 07AI91 no separate operating and assembly instructions are available.

Please also observe the system data as well as the information on CS31 bus ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Please observe the information contained in the existing documentation. In section "Fig. 5.4-2: Connection of the analog input/output module 07AC91", only the information concerning operating mode 12 bit is relevant for the replacement device 07AC91-AD2.

Device configuration



1 Connection for CS31 bus (X1)
2 Analog inputs (X2): -10 V…+10 V, 0…20 mA
3 Analog inputs (X3): -10 V…+10 V, 0…20 mA
4 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
5 DIP switch for CONFIG1
6 DIP switch for CONFIG2
7 Status LEDs for AX522
8 DIP switch for ADDR
9 Analog outputs (X7): -10 V…+10 V
10 Analog outputs (X6): -10 V…+10 V, 0…20 mA
11 Enabling input for analog outputs (X5)
12 Supply 24 V DC (incl. AX522)
13 Ventilation
14 TA525: Label
15 4 Status LEDs

LED display

The LED display on the replacement device is changed:

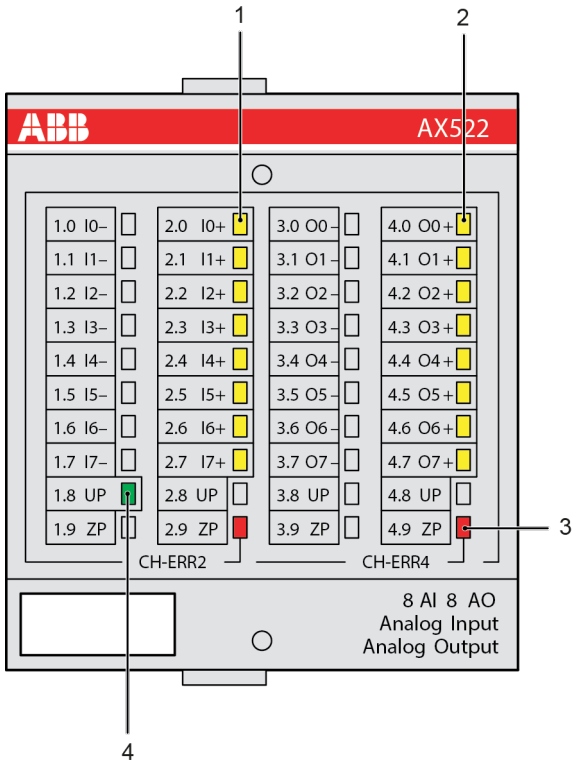


Fig. 774: AX522

No.	Display of module
1	8 yellow LEDs to indicate the signal status of the analog inputs (X2 and X3)
2	8 yellow LEDs to indicate the signal status of the analog inputs (X6 and X7)
3	2 red LEDs to indicate errors (of AX522 module)
4	1 green LED to indicate the status of the supply voltage of the AX522 module (is supplied via X4)

The replacement device does not provide a test button to measure functionality.

Connections

Please observe the information contained in the existing documentation. In section "Fig. 5.4-2: Connection of the analog input/output module 07AC91", only the information concerning operating mode 12 bit is relevant for the replacement device 07AC91-AD2.

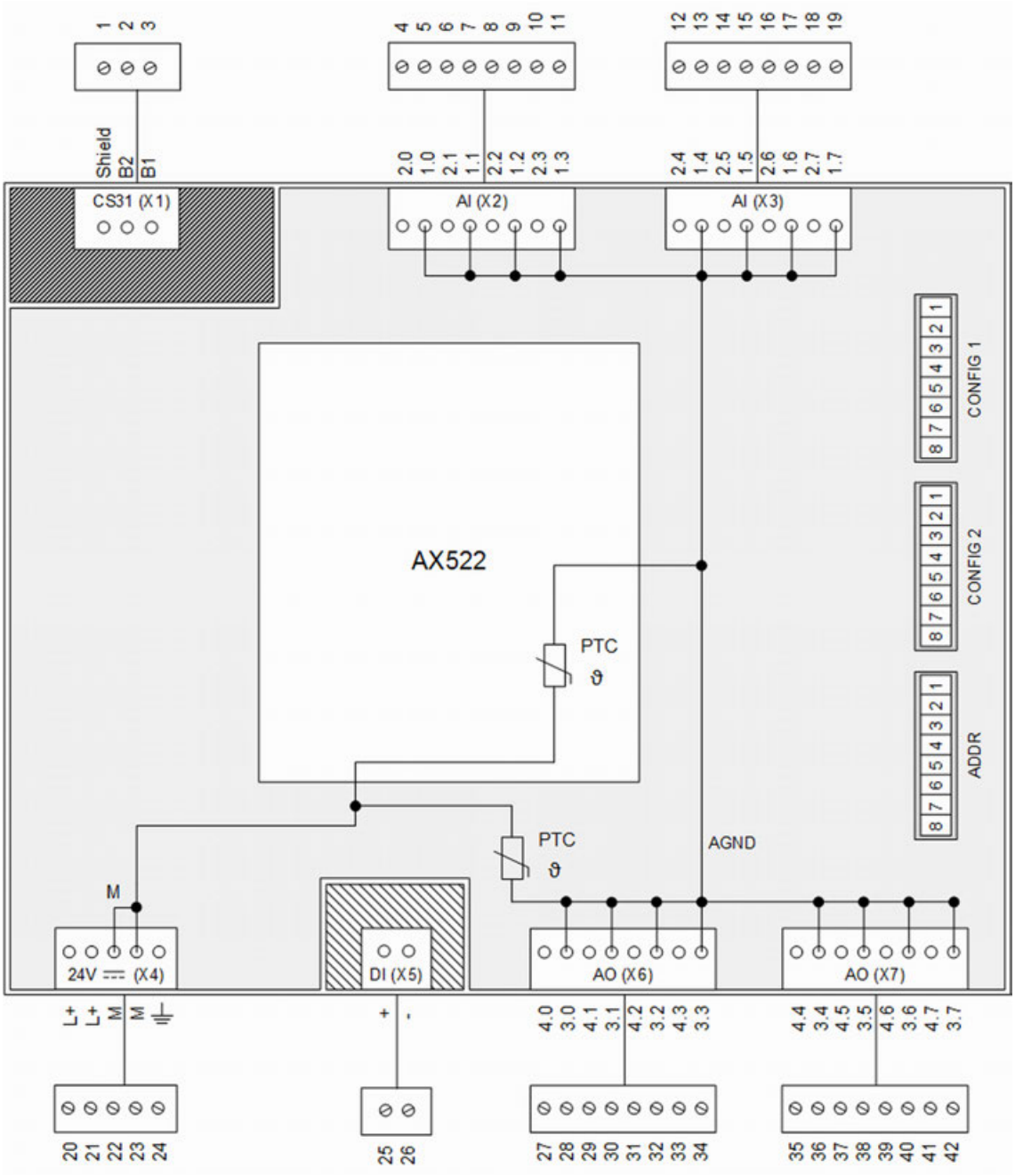


Fig. 775: Connection

Table 317: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	No internal connection
X1 / B2	2	BUS 2
X1 / B1	3	BUS 1

Table 318: Pin assignment AI (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / 2.0	4	AX522 / I0+
X2 / 1.0	5	AX522 / I0- (AGND)

Connector / Terminal	Pin	Assignment / Signal
X2 / 2.1	6	AX522 / I1+
X2 / 1.1	7	AX522 / I1- (AGND)
X2 / 2.2	8	AX522 / I2+
X2 / 1.2	9	AX522 / I2- (AGND)
X2 / 2.3	10	AX522 / I3+
X2 / 1.3	11	AX522 / I3- (AGND)

Table 319: Pin assignment AI (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / 2.4	12	AX522 / I4+
X3 / 1.4	13	AX522 / I4- (AGND)
X3 / 2.5	14	AX522 / I5+
X3 / 1.5	15	AX522 / I5- (AGND)
X3 / 2.6	16	AX522 / I6+
X3 / 1.6	17	AX522 / I6- (AGND)
X3 / 2.7	18	AX522 / I7+
X3 / 1.7	19	AX522 / I7- (AGND)

Table 320: Pin assignment 24 V DC 6W (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / L+	20	L+
X4 / L+	21	L+
X4 / M	22	M
X4 / M	23	M
X4 / FE	24	FE

Table 321: Pin assignment DI (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / +	25	IN+
X5 / -	26	IN- (galvanic isolated earth)

Table 322: Pin assignment AO (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / 4.0	27	AX522 / O0+
X6 / 3.0	28	AX522 / O0- (AGND)
X6 / 4.1	29	AX522 / O1+
X6 / 3.1	30	AX522 / O1- (AGND)
X6 / 4.2	31	AX522 / O2+
X6 / 3.2	32	AX522 / O2- (AGND)
X6 / 4.3	33	AX522 / O3+
X6 / 3.3	34	AX522 / O3- (AGND)

Table 323: Pin assignment AO (X7)

Connector / Terminal	Pin	Assignment / Signal
X7 / 4.4	35	AX522 / O4+
X7 / 3.4	36	AX522 / O4- (AGND)
X7 / 4.5	37	AX522 / O5+
X7 / 3.5	38	AX522 / O5- (AGND)
X7 / 4.6	39	AX522 / O6+
X7 / 3.6	40	AX522 / O6- (AGND)
X7 / 4.7	41	AX522 / O7+
X7 / 3.7	42	AX522 / O7- (AGND)

The outputs on connector X7 cannot be configured as current outputs.

The signals Ix- and Ox- are internally linked to an AGND area. The potential AGND is connected to the potential M via PTC resistors. Potential difference AGND to M ± 1 V maximal.

To enable wire-break detection, each input is internally pulled to "plus" by means of a high-impedance resistor. As a result, the maximum voltage is read when nothing is connected. Do not replace the AX522 module while voltage is connected.

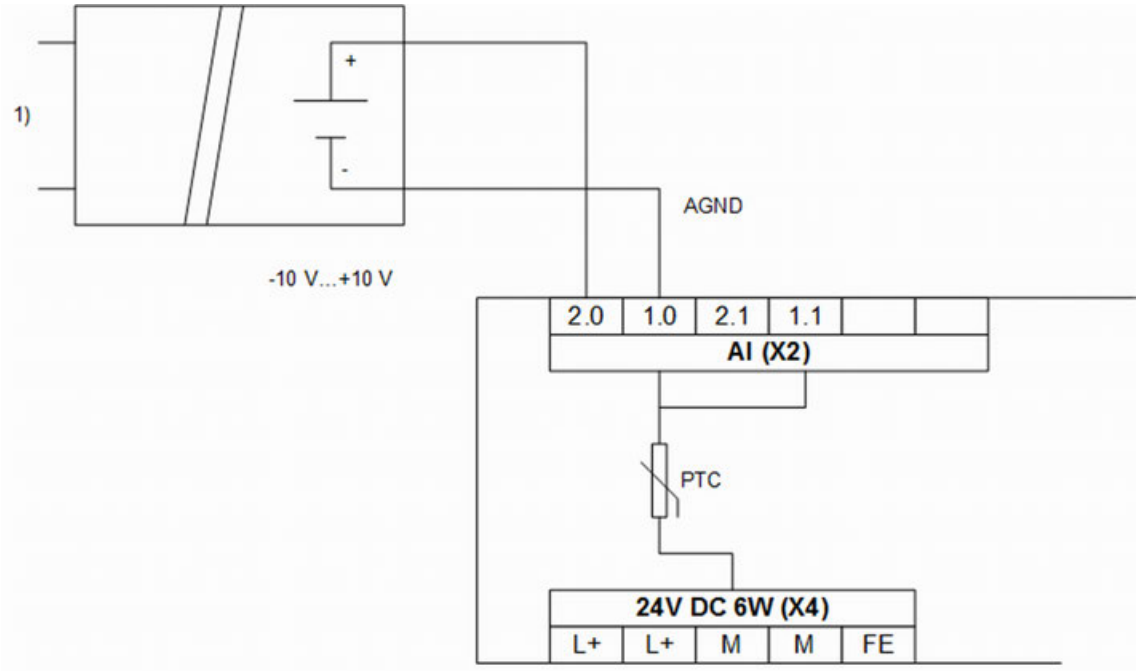


Fig. 776: Voltage input

1) Galvanically isolated power supply of analog sensor

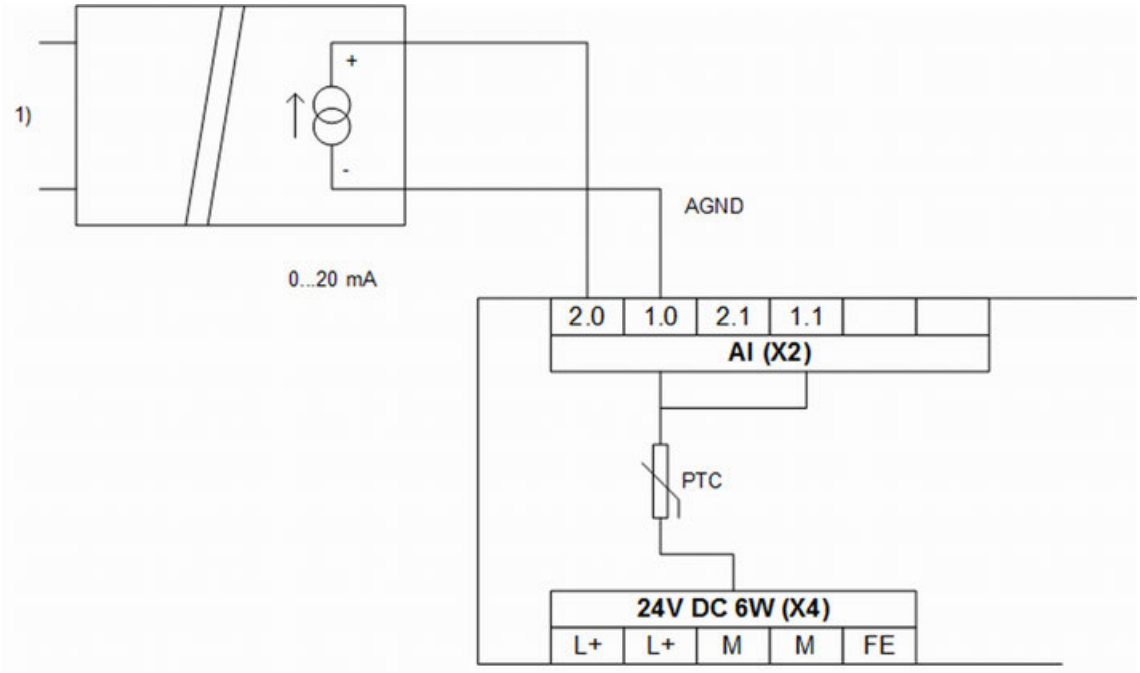


Fig. 777: Current input

1) Galvanically isolated power supply of analog sensor

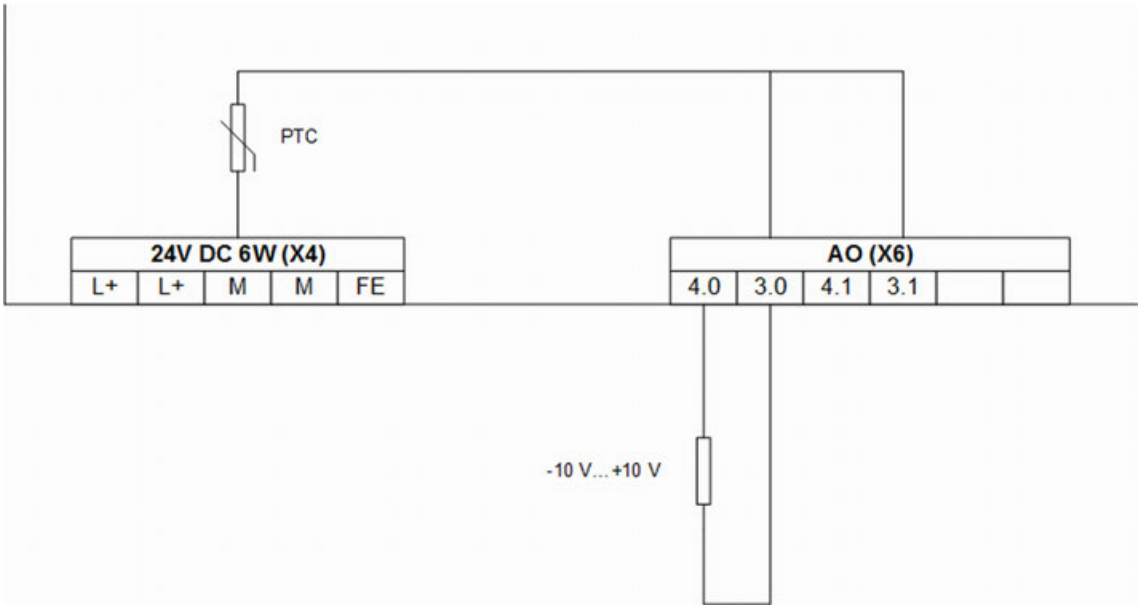


Fig. 778: Voltage output

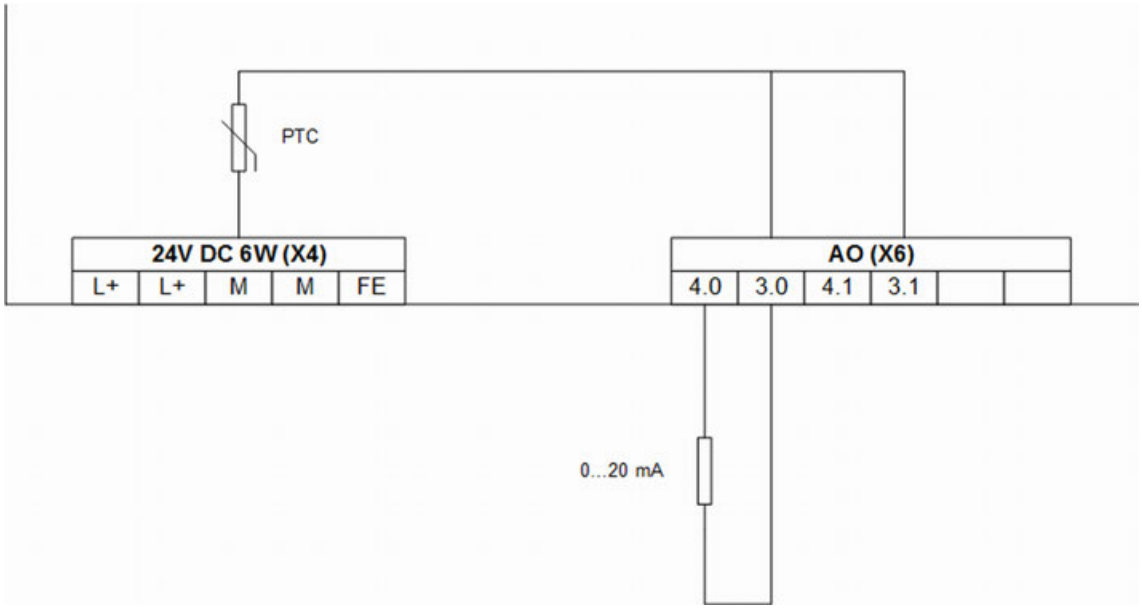


Fig. 779: Current output

Analog signal lines must be routed in shielded cables. The shield must be grounded on both sides and should be grounded to replacement device and signal source / signal sink as close as possible.

Configuration

The existing device had a DIP switch on the upper printed circuit board. Since the replacement device is not equipped with an upper printed circuit board, the white DIP switch is arranged on the lower printed circuit board instead.

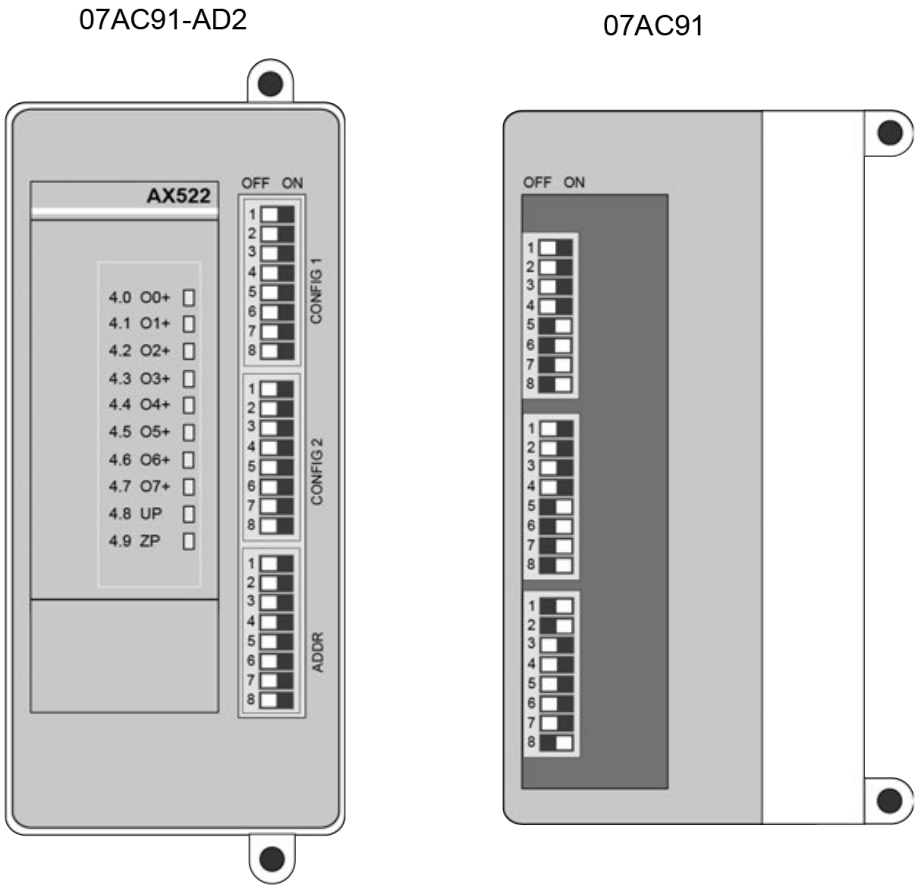


Table 324: Example configuration for 07AC91-AD2:

Config 1	All input channels set to ON (voltage).
Config 2	All output channels set to ON (voltage).
ADDR	12-bit mode, without range monitoring, CS31 address 0 and channel number ≤ 7.

Configuration areas with (white) DIP switches

Please observe the following:

- Unused voltage inputs must be configured as current inputs (due to wire-break detection AX522 S500 module).
- The DIP switches are read by the device only once after the supply voltage has been connected.

Config 1	The DIP switches for all 8 channels (inputs) may be set to ON (current) or OFF (voltage).
Config 2	The DIP switches for the channels 1-4 (outputs 0..3) may be set to ON (current) or OFF (voltage).
	The DIP switches for the channels 5-8 (outputs 4..7) must be set to OFF (voltage). **The setting ON (current) is not permitted.**
ADDR	The DIP switch for channel 1 (operating mode) **must be set to OFF (12-bit mode)**.
	The DIP switch for channel 2 can be set as desired (no functionality).
	The DIP switch for channel 3 can be set as desired for range monitoring.

	The DIP switches for the channels 4-7 can be set as desired for the CS31 address.
	The DIP switch for channel 8 must be set to OFF for CS31 channels ≤ 7. **Channels > 7 are not supported. The outputs on connector X7 cannot be configured as current outputs.**

For further information, please refer to the existing documentation System description Advant Controller 31.

Measuring ranges of the analog channels

For the replacement device 07AC91-AD2, only the operating mode "12 bit" is relevant.

Measuring range:

- Inputs: ± 10 V and 0..20 mA
- Outputs for X6 (AW1.0..AW1.3): ± 10 V and 0..20 mA
- Outputs for X7 (AW1.4..AW1.7): ± 10V

Addressing

The function of the address DIP switch 8 (channel No. ≤ 7 or channel No. > 7) is no longer supported.

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

When the measuring values are bipolar, it is advisable to use the data type "INT input/output" instead of "WORD input/output".

Table 325: CS31 bus

Type	Byte	Connector / Terminal
WORD (send) 0	1	X2 / 2.0
	2	
WORD input (send) 1	3	X2 / 2.1
	4	
WORD input (send) 2	5	X2 / 2.2
	6	
WORD input (send) 3	7	X2 / 2.3
	8	
WORD input (send) 4	9	X3 / 2.4

Type	Byte	Connector / Terminal
	10	
WORD input (send) 5	11	X3 / 2.5
	12	
WORD input (send) 6	13	X3 / 2.6
	14	
WORD input (send) 7	15	X3 / 2.7
	16	
WORD output (received) 8	17	X6 / 4.0
	18	
WORD output (received) 9	19	X6 / 4.1
	20	
WORD output (received) 10	21	X6 / 4.2
	22	
WORD output (received) 11	23	X6 / 4.3
	24	
WORD output (received) 12	25	X7 / 4.4
	26	
WORD output (received) 13	27	X7 / 4.5
	28	
WORD output (received) 14	29	X7 / 4.6
	30	
WORD output (received) 15	31	X7 / 4.7
	32	

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.
- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

The RAM is checked during the initialization of the device. In addition, the firmware in the Flash memory is checked by means of a checksum during initialization. When the control system (PLC/central unit) is stopped during normal operation, the outputs of the device are switched off. The inputs remain active. The outputs are also switched off in case of a malfunction of the CS31 bus.

Diagnosis and display

LEDs are used for diagnosis and display purposes. In addition, some diagnosis information can be transmitted via the CS31 bus.

The replacement device does not provide a test button to measure functionality.

Table 326: Diagnosis information of the CS31 bus

Channel	Error code (CODESYS)	Error code (CS31 bus)	Description
Device error:			
0	43	1	Internal error
Channel error:			
0 ... 7	49	10	Analog value is out of measuring range (on analog inputs)

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

An exceedance of the measuring range is signaled even if nothing is connected to an analog voltage input.

Table 327: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.

- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 328: S500 module AX522 LEDs

LED	Status	Color	LED off	LED on	LED flashes
I0+…I7+ (see No. 1 in the following figure)	Analog inputs	Yellow	Input is not activated	Input is activated (brightness depends on value of analog signal).	-
O0+…O7+ (see No. 2 in the following figure)	Analog outputs	Yellow	Output is not activated	Output is activated (brightness depends on value of analog signal).	-
Error indication left (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	-
Error indication right (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	-
Indication supply voltage (see No. 4 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-

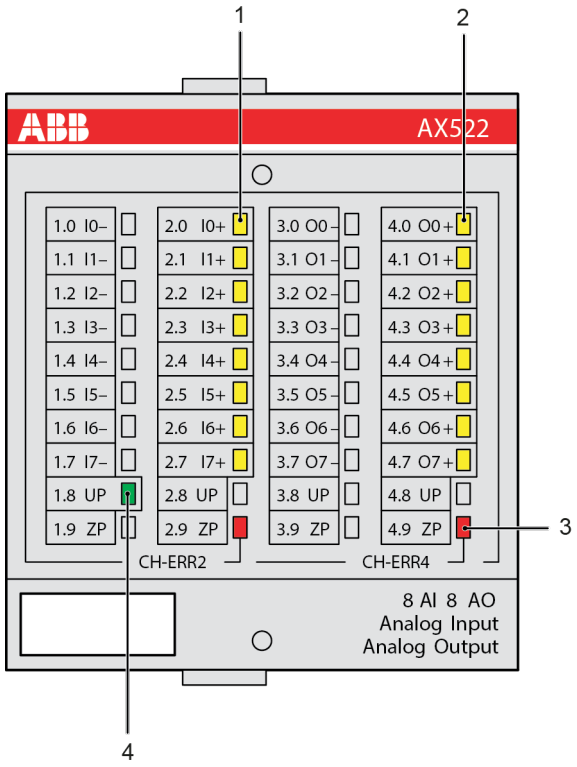


Fig. 780: AX522

Technical data

This section provides additional information on section *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*. In case of doubt, the following information applies.

For the device 07AC91-AD2, only the operating mode 12 bit is relevant.

Technical data of the complete device

Data	Value
Process voltage:	
-> Connections	X4/L+ (pin 20), X4/L+ (pin 21), X4/M (pin 22), X4/M (pin 23)
-> Fuse for L+	10 A, fast acting
- Galvanic isolation	No
Current consumption:	
-> via L+	0.19 A + output load
- Inrush current via L+ (when voltage is switched on)	0.16 A^2s
Power consumption	Replacement device: 6 W Existing device: 5 W

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Technical data of the binary input

Data	Value
Input current at input voltage +24 V	Typ. 6 mA
Protection against reversed voltage	Yes
Overvoltage protection	No

The enabling input is a proprietary input.

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical data of the analog inputs

Data	Value
Connections	X2 / 2.0, X2 / 2.1, X2 / 2.2, X2 / 2.3, X3 / 2.4, X3 / 2.5, X3 / 2.6, X3 / 2.7
Reference connections (AGND)	X2 / 1.0, X2 / 1.1, X2 / 1.2, X2 / 1.3, X3 / 1.4, X3 / 1.5, X3 / 1.6, X3 / 1.7
Type of inputs	Voltage bipolar, current unipolar
Time constant of the input filter	Voltage Replacement device: 100 µs Existing device: 470 µs
Conversion cycle *)	Replacement device: 2 ms (over 8 inputs + 8 outputs) Existing device: 8 ms
Resolution: range ± 10 V	Replacement device: 2.4 mV, 12 bit + sign Existing device: 5 mV, 11 bit + sign
Protection against reversed voltage	Yes
Overvoltage protection	Up to 30 V DC

*) Conversion cycle of S500 module AX522. The transmission via serial buses is slower.

Unused voltage inputs must be configured as current inputs (due to wire-break detection AX522 S500 module).

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical data of the analog outputs

Data	Value
Connections	X6 / 4.0, X6 / 4.1, X6 / 4.2, X6 / 4.3, X7 / 4.4, X7 / 4.5, X7 / 4.6, X7 / 4.7
Reference connections (AGND)	X6 / 3.0, X6 / 3.1, X6 / 3.2, X6 / 3.3, X7 / 3.4, X7 / 3.5, X7 / 3.6, X7 / 3.7
Type of outputs	Voltage bipolar, current unipolar
Configurability	Replacement device: 4 current outputs available Existing device: 8 current outputs available
Output load capability, as voltage output	Replacement device: ± 10 mA Existing device: +20 mA, -10 mA
Short-circuit-proof	Yes
External supply protection	Up to 30 V DC

For further information, please refer to the existing documentation System description Advant Controller 31.

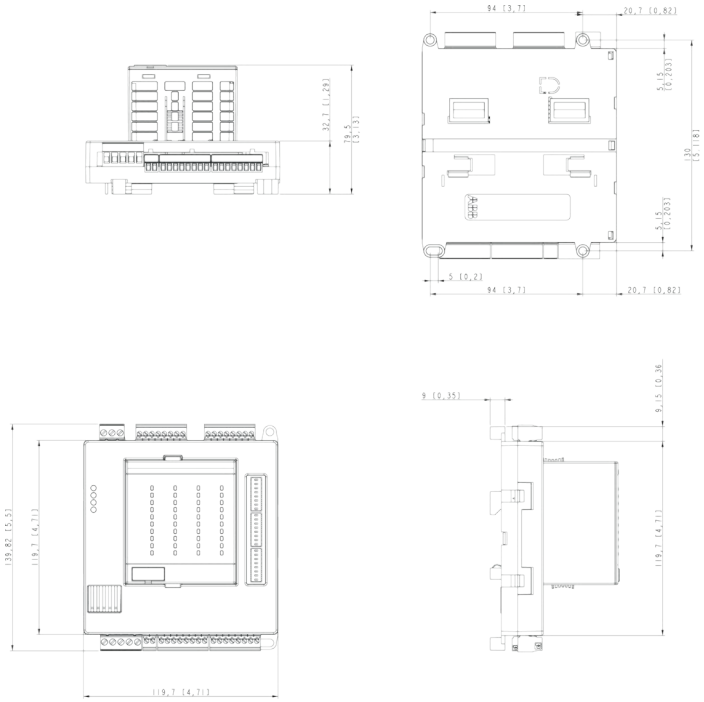
Connection to the CS31 bus

Data	Value
Connections	X1/B2, X1/B1
CS31 bus type	05 (analog input/output)
Termination resistor	Not available (must be provided externally if needed)

Mechanical data

Data	Value
Width x height x depth	Replacement device: 120 x 140 x approx. 80 mm Existing device: 120 x 140 x 85 mm
Weight	Replacement device: 362 g Existing device: 450 g
Dimensions for mounting	See assembly instructions 07AC91-AD2 (3ADR020085M0401)

Mounting information



The dimensions are in mm and in brackets in inch.

The dimensions for the assembly holes are the same for the replacement device and the existing device.

To assemble or disassemble the replacement device, grab the device at the housing and not directly at the S500 module.

Ordering data

Order No.	Scope of delivery
1SAP 800 100 R0010	Analog input/output module 07AC91-AD2 1 2-pole terminal block (3.81 mm grid space) 1 3-pole terminal block (5.08 mm grid space) 1 5-pole terminal block (5.08 mm grid space) 4 8-pole terminal blocks (3.81 mm grid space)

Replacement device 07AI91-AD

Introduction

Fig. 781: 3ADR331191S0015_07AI91-AD

The replacement device 07AI91-AD from the AC31 adapter series replaces the existing device 07DC91 from the 90 series.

During the development of the replacement device, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07AI91 remains valid and serves as a reference *(system description Advant Controller 31)*. The document structure of this document is based on the document structure of the existing documentation.

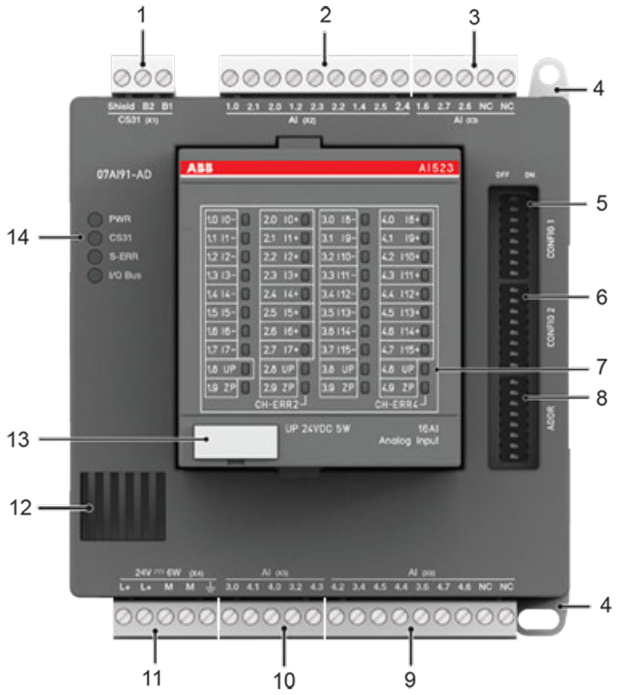
This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement device 07AI91-AD can be found in the operating and assembly instructions of device 07AI91-AD: 3ADR020086M0401. Please note that for the existing device 07AI91 no separate operating and assembly instructions are available.

Please also observe the system data as well as the information on CS31 bus *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Device configuration



1 Connection for CS31 bus (X1)
2 Analog inputs (X2). 2.5 AI (± 10 V differential, ± 5 V differential, temperature measurement PT100 / PT1000, 4…20 mA and 0…20 mA with external resistor)
3 Analog inputs (X3). 1.5 AI (± 10 V differential, ± 5 V differential, temperature measurement PT100 / PT1000, 4…20 mA and 0…20 mA with external resistor)
4 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
5 DIP switch for CONFIG1
6 DIP switch for CONFIG2
7 Status LEDs for AI523
8 DIP switch for ADDR
9 Analog inputs (X6). 2.5 AI (± 10 V differential, ± 5 V differential, temperature measurement PT100 / PT1000, 4…20 mA and 0…20 mA with external resistor)
10 Analog inputs (X5). 1.5 AI (± 10 V differential, ± 5 V differential, temperature measurement PT100 / PT1000, 4…20 mA and 0…20 mA with external resistor)
11 Supply 24 V DC (incl. AI523)
12 Ventilation
13 TA525: Label
14 4 Status LEDs of complete device

In contrast to the existing device, the following measuring ranges are not available in the replacement device: ± 500 mV, ± 50 mV. Temperature measurement with thermocouples is also not possible.

The replacement device does not perform a self-calibration.

LED display

The LED display on the replacement device is changed:

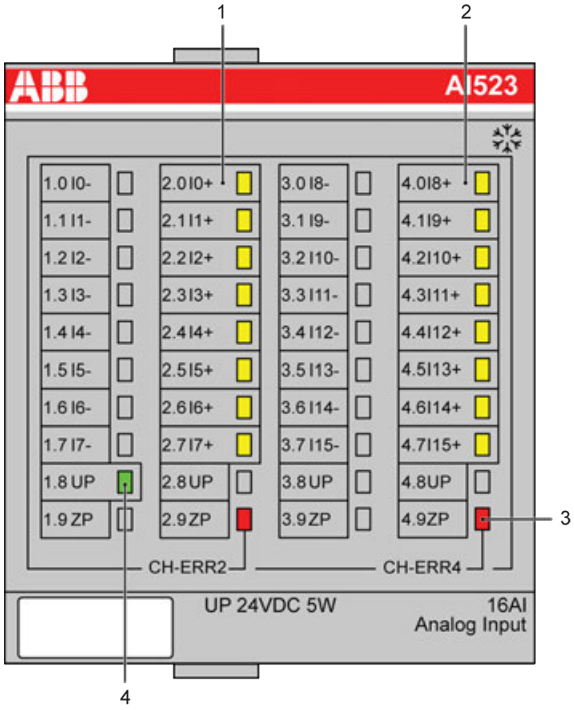


Fig. 782: Front view: 07AI91-AD

No.	Display of module
1	8 yellow LEDs to indicate the signal status of the analog inputs (X2 and X3)
2	8 yellow LEDs to indicate the signal status of the analog inputs (X5 and X6)
3	2 red LEDs to indicate errors (of AI523 module)
4	1 green LED to indicated the status of the supply voltage of the AI523 module (is supplied via X4)

The replacement device does not provide a test button to measure functionality.

Connections

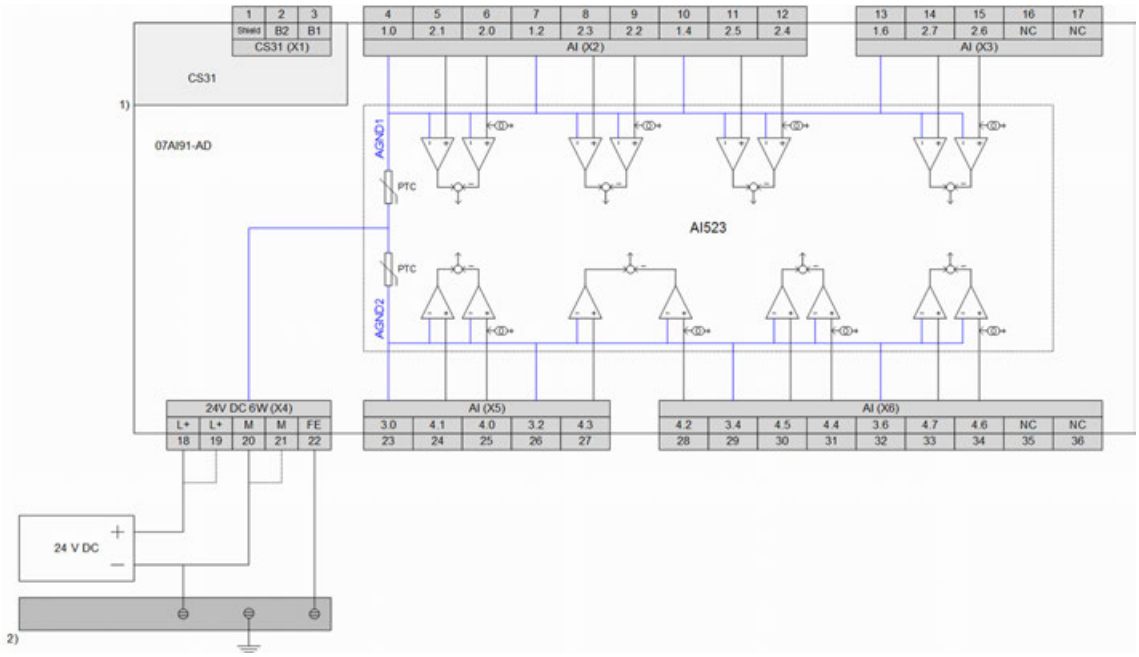


Fig. 783: Connection

1) Galvanic isolation
2) Switchgear cabinet grounding

Please observe the following information:

- The **Shield** connections of the CS31 bus and **FE** of the supply voltage have no connection within the device.
- The process voltage must be included in the grounding concept of the control system (e.g. grounding of the negative pole).
- The connections of all sensors must be galvanically isolated from the mounting environment of the sensors. The cable shields of the temperature sensors are grounded to the switchgear cabinet at the entry into the cabinet. The setting of the module address as well as the configuration of the analog channels are performed by means of DIP switches (see next pages).
- Unused inputs must be configured as "not evaluated" (DIP switch).
- The current sources in AI523 are configurable and therefore not always active. The current sources are connected alternately with the multiplex method. Consequently, the device does not have 8 current sources.
- The module address and the analog channels are set with DIP switches.

Table 329: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	No internal connection
X1 / B2	2	BUS 2
X1 / B1	3	BUS 1

Table 330: Pin assignment AI (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / 1.0	4	AI523 / I0- (AGND1)
X2 / 2.1	5	AI523 / I1+
X2 / 2.0	6	AI523 / I0+

Connector / Terminal	Pin	Assignment / Signal
X2 / 1.2	7	AI523 / I2- (AGND1)
X2 / 2.3	8	AI523 / I3+
X2 / 2.2	9	AI523 / I2+
X2 / 1.4	10	AI523 / I4- (AGND1)
X2 / 2.5	11	AI523 / I5+
X2 / 2.4	12	AI523 / I4+

Table 331: Pin assignment AI (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / 1.6	13	AI523 / I6- (AGND1)
X3 / 2.7	14	AI523 / I7+
X3 / 2.6	15	AI523 / I6+
X3 / NC	16	Not connected
X3 / NC	17	Not connected

In module AI523, the signals I0-, I2-, I4- and I6- are internally connected to an analog ground. The potential difference of the analog ground to M is ±1 V (max.). The replacement device has no current sources on pins 16 and 17. If necessary, these current sources can be connected to individual measurement channels via the configuration (DIP switch).

Table 332: Pin assignment 24 V DC 6W (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / L+	18	L+
X4 / L+	19	L+
X4 / M	20	M
X4 / M	21	M
X4 / FE	22	FE

Table 333: Pin assignment AI (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / 3.0	23	AI523 / I8- (AGND2)
X5 / 4.1	24	AI523 / I9+
X5 / 4.0	25	AI523 / I8+
X5 / 3.2	26	AI523 / I10- (AGND2)
X5 / 4.3	27	AI523 / I11+

Table 334: Pin assignment AI (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / 4.2	28	AI523 / I10+
X6 / 3.4	29	AI523 / I12- (AGND2)
X6 / 4.5	30	AI523 / I13+
X6 / 4.4	31	AI523 / I12+
X6 / 3.6	32	AI523 / I14- (AGND2)
X6 / 4.7	33	AI523 / I15+
X6 / 4.6	34	AI523 / I14+

Connector / Terminal	Pin	Assignment / Signal
X6 / NC	35	Not connected
X6 / NC	36	Not connected

In module AI523, the signals I8-, I10-, I12- and I14- are internally connected to an analog ground. The potential difference of the analog ground to M is ±1 V (max.). The replacement device does not have current sources on pins 35 and 36. If necessary, these current sources can be connected to individual measurement channels via the configuration (DIP switch).

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

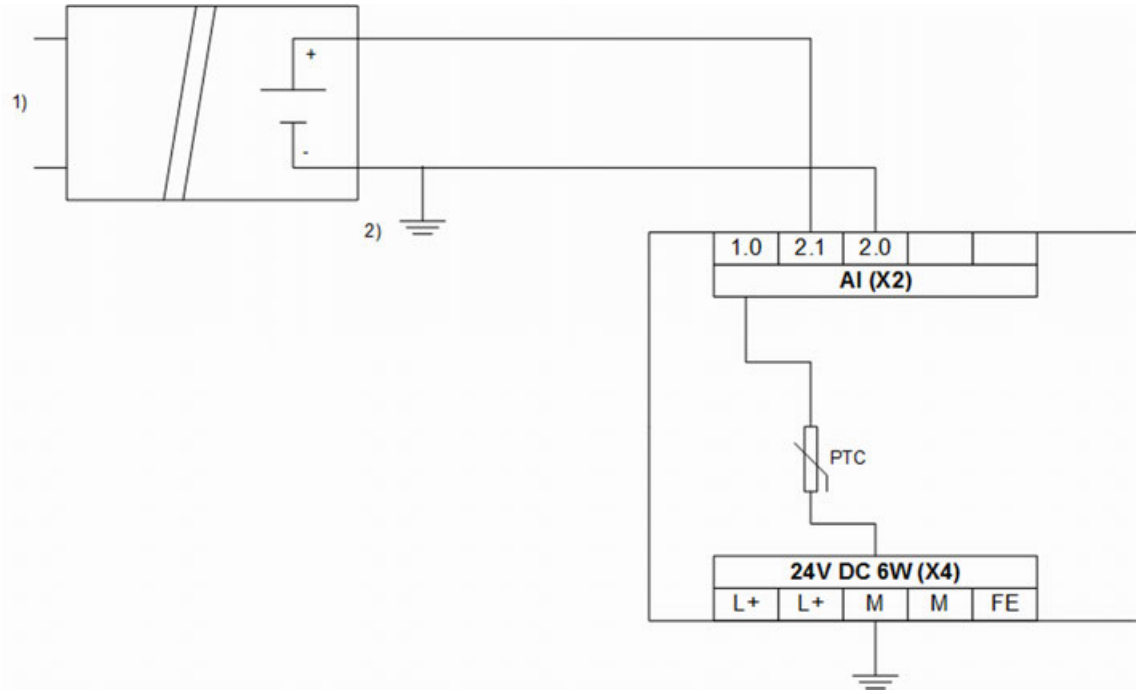


Fig. 784: Differential voltage input

1) Galvanically isolated power supply of analog sensor
2) Grounding at sensor
   ±10 V or ±5 V at differential inputs

On the replacement devices, the wire-break detection is also active in case of a differential voltage measurement. For this purpose, each measuring channel is internally pulled to "plus" by means of a high-impedance resistor. As a result, the individual potentials of the differential voltage measurement must also be referenced to M. Completely isolated voltages are **not** symmetrized to M by the inputs.

The potential difference of the grounding at the sensor to M must not be too big (max. ± 1 V for the whole signal range). Otherwise problems can occur concerning the common-mode input voltages of the involved analog inputs.

Analog signal lines must be routed in shielded cables. The shield must be grounded on both sides and should be grounded to replacement device and signal source / signal sink as close as possible.

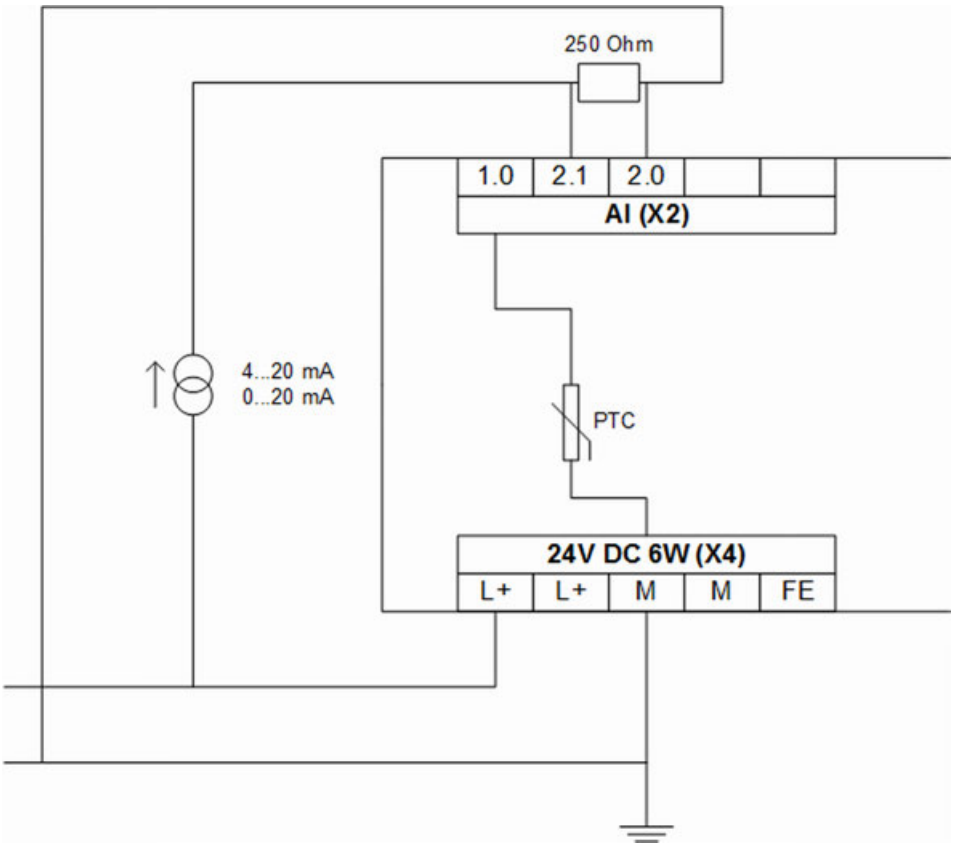


Fig. 785: Current input with external resistor

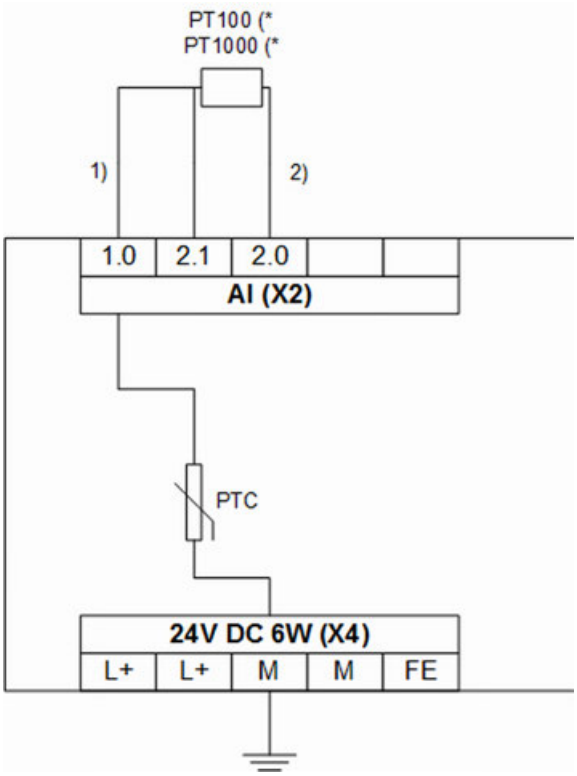


Fig. 786: Resistance thermometer

1) Return conductor
2) Twisted wire pair in the cable
(*) 3-wire

For temperature measurements with PT100/PT1000 resistors, the wiring to the existing device must be changed. A 4-wire temperature measurement is not possible with the replacement device. Based on the above figure, a 3-wire temperature measurement can be implemented.

Configuration

The existing device had a DIP switch on the upper printed circuit board. Since the replacement device is not equipped with an upper printed circuit board, the white DIP switch is arranged on the lower printed circuit board instead.

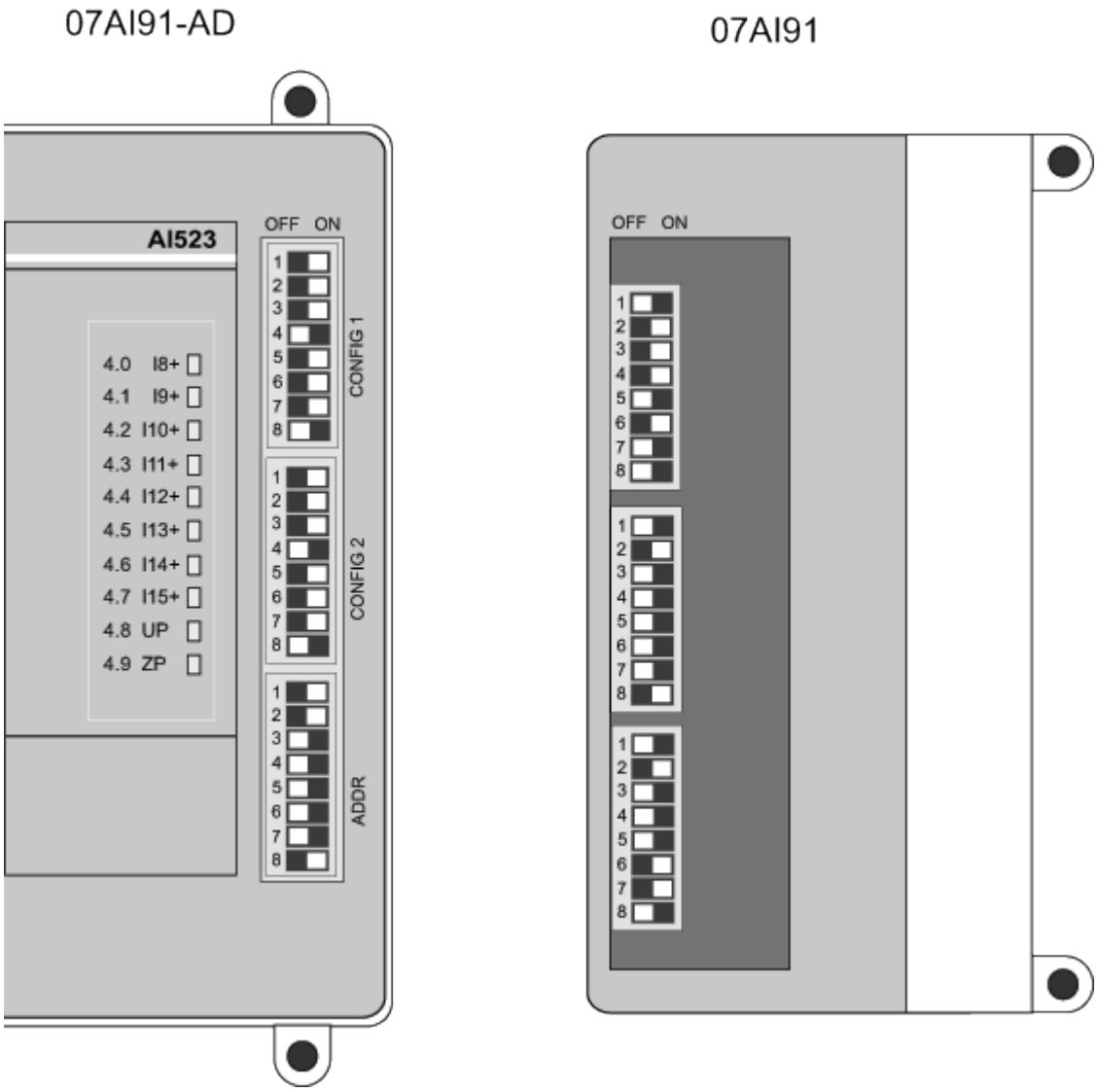


Fig. 787: DIP switch for 07AI91-AD

The function of the address DIP switch 8 (channel No. ≤ 7 or channel No. > 7) is not supported for the replacement device. This DIP switch must be switched off.

On address DIP switch 3 (assignment of analog value), only the CS31 bus format is supported in the replacement device. This DIP switch must be switched on. The setting of the line frequency suppression (address DIP switch 1 and 2) has no effect on the existing device 07AI91.

The following settings of DIP switches CONFIG 1 and CONFIG 2 are not implemented in the replacement device and must not be selected:

- *± 500 mV*
- *± 50 mV*
- *J-type thermocouple with linearization*
- *K-type thermocouple with linearization*
- *S-type thermocouple with linearization*

For further information, please refer to the existing documentation System description Advant Controller 31.

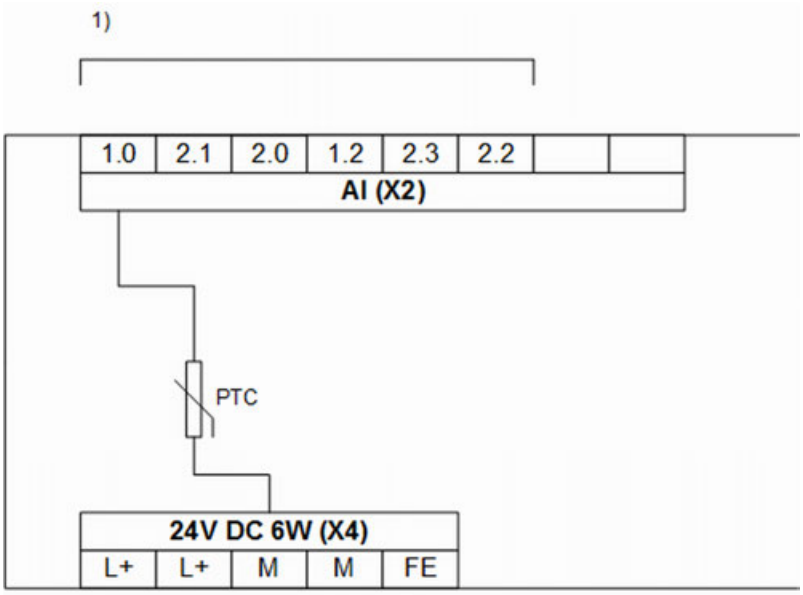


Fig. 788: "Configuration pair" not used

1) Channel 0 and channel 1 are not used -> DIP switch "No evaluation of channels"

If both channels of a "configuration pair" are not used, set the DIP switches to "No evaluation of channels".

The DIP switches are read by the device only once after the supply voltage has been connected.

Measuring ranges of the input channels

All input signals are not evaluated as differential signals. Two input channels are used to implement a differential measurement.

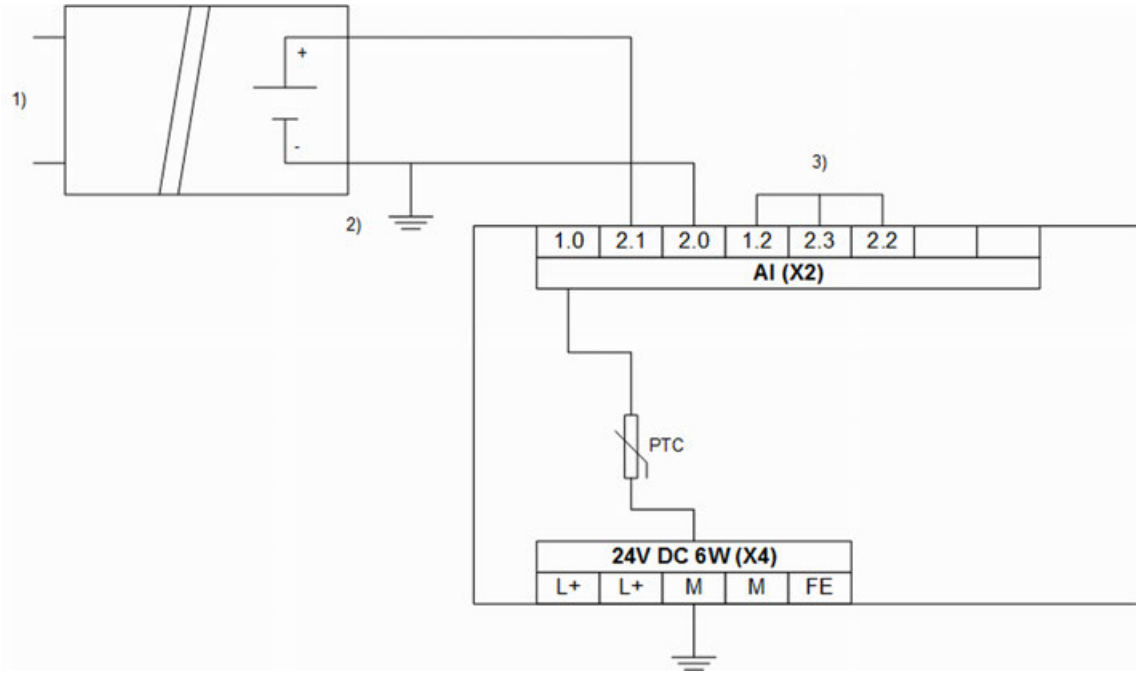


Fig. 789: Only one channel of a "configuration pair" is used

1) Galvanically isolated power supply of analog sensor
2) Grounding at sensor
3) Channel not used
   ± 10 V, ± 5 V at differential inputs

> *If only one channel of a "configuration pair" is used (e.g. channel 0 and 1), then the other channel must be short-circuited during a voltage measurement. Short-circuited in this context means that for instance the connections 1.2, 2.3 and 2.2 are connected. Otherwise the channel not used reports that the range has been exceeded.*

Measuring ranges

- Measuring ranges ± 10 V / ± 5 V / ± 500 mV and ± 50 mV no longer exist.
- Measuring ranges 4 ... 20 mA / 0 ... 20 mA not changed to existing documentation.

Pt 100 / Pt 1000

To measure the temperature by means of resistors, a constant current is supplied by the replacement device. This imprint no longer occurs at terminals 16, 17, 35 and 36. Therefore the wiring must be changed for the temperature measurement.

Further information:

- Fig. 783
- Fig. 786
- Figures 5.2-4 and 5.2-5 from the existing documentation of the 07AI91 are not valid for the replacement device *System description Advant Controller 31*.
- Terminals 7, 10, 13, 26, 29 and 32 can no longer be used as connection bases. The terminals are only used for the 3-wire temperature measurement *System description Advant Controller 31*.

> ***Wire-breakage***
> *In case of a wire-breakage, the numerical value +32767 is output. This is followed by an error message via the CS31 bus.*

Channel use

If only one channel of a "configuration pair" is used (e.g. channel 0 and 1), then the other channel must be connected with a resistor (e.g. 120 Ω Pt100 measuring range, 1200 Ω Pt1000 measuring range). Otherwise an error message is indicated.

NOTICE!

Temperature-dependent resistors

Other temperature-dependent resistors cannot be used for the replacement device.

NOTICE!

Thermocouples type J, type K, type S

Thermocouples cannot be evaluated with the replacement device. The respective section in the existing documentation (incl. figure 5.2-6) is not valid for device 07AI91.

Configuration for unused channels

See existing documentation 07AI91 System description Advant Controller 31.

Relationship between the measuring values and the location of the bits in a 16 bit WORD

- *The measuring ranges ± 500 mV and ± 50 mV no longer exist.*
- *Measuring range ±5 V:*
  - *Replacement device: 11 bit resolution plus sign*
  - *Existing device: 12 bit resolution plus sign*
- *All measuring ranges for thermocouples are no longer available.*

Addressing

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

The function of the address DIP switch 8 (channel No. ≤ 7 or channel No. > 7) is no longer supported.

Table 335: CS31 bus

Type	Byte	Connector / Terminal
WORD input (send) 0	1	X2 / 2.1, X2 / 2.0
	2	
WORD input (send) 1	3	X2 / 2.3, X2 / 2.2
	4	
WORD input (send) 2	5	X2 / 2.5, X2 / 2.4
	6	
WORD input (send) 3	7	X3 / 2.7, X3 / 2.6
	8	
WORD input (send) 4	9	X5 / 4.1, X5 / 4.0
	10	
WORD input (send) 5	11	X5 / 4.3, X6 / 4.2
	12	
WORD input (send) 6	13	X6 / 4.5, X6 / 4.4
	14	
WORD input (send) 7	15	X6 / 4.7, X6 / 4.6
	16	

When the measuring values are bipolar, use data type "INT input" instead of "WORD input".

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.
- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

The RAM is checked during the initialization of the device. In addition, the firmware in the Flash memory is checked by means of a checksum during initialization. When the control system (PLC/central unit) is stopped during normal operation, the inputs remain active.

Diagnosis and display

LEDs are used for diagnosis and display purposes. In addition, some diagnosis information can be transmitted via the CS31 bus.

The replacement device does not provide a test button to measure functionality.

Table 336: Diagnosis information of the CS31 bus

Channel	Error code (CODESYS)	Error code (CS31 bus bus)	Description
Device error:			
0	43	1	Internal error
Channel error:			
0 ... 7	45	9	Cut wire (is also indicated if the current in measuring range 4 ... 20 mA is less than 2 mA)
0 ... 7	49	10	Analog value is out of measuring range

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

An exceedance of the measuring range is signaled even if nothing is connected to an analog voltage input.

Table 337: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.
- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 338: LEDs of the S500 module AI523

LED	Status	Color	LED off	LED on	LED flashes
I1+, I3+, I5+, I7+ (see No. 1 in the following figure)	Analog inputs	Yellow	Input is not activated	Input is activated (brightness depends on value of analog signal).	-
I9+, I11+, I13+, I15+ (see no. 2 in the following figure)	Analog inputs	Yellow	Input is not activated	Input is activated (brightness depends on value of analog signal).	-
Error indication left (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	Cut wire on a channel of the corresponding group
Error indication right (see No. 3 in the following figure)	Error indication	Red	No error	Internal error	Cut wire on a channel of the corresponding group
Indication supply voltage (see No. 4 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-

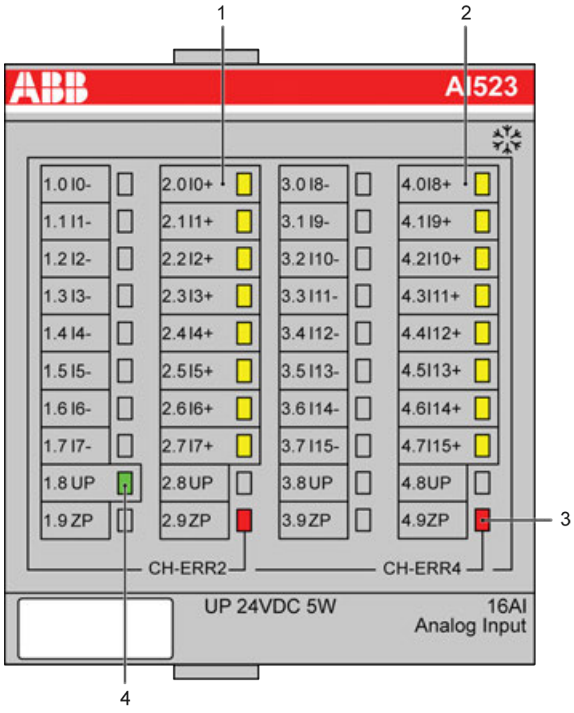


Fig. 790: 07AI91-AD_Front

Technical data

This section provides additional information on section ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*. In case of doubt, the following information applies.

Technical data of the complete device

Data	Value
Process voltage:	
-> Connections	X4/L+ (pin 18), X4/L+ (pin 19), X4/M (pin 20), X4/M (pin 21)
-> Fuse for L+	10 A, fast acting
- Galvanic isolation	No
Current consumption:	
-> via L+	0.19 A
- Inrush current via L+ (when voltage is switched on)	0.22 A^2s
Power consumption	Replacement device: 6 W Existing device: 3 W
Address setting and configuration	DIP switch right side of housing
Max. line length of analog lines, line cross section > 0.14 mm²	100 m

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Technical data of the analog inputs

Data	Value
Connections	[X2 / 2.1, X2 / 2.0], [X2 / 2.3, X2 / 2.2], [X2 / 2.5, X2 / 2.4], [X3 / 2.7, X3 / 2.6], [X5 / 4.1, X5 / 4.0], [X5 / 4.3, X6 / 4.2], [X6 / 4.5, X6 / 4.4], [X6 / 4.7, X6 / 4.6]
Reference connections (AGND1)	X2 / 1.0, X2 / 1.2, X2 / 1.4, X3 / 1.6
Reference connections (AGND2)	X5 / 3.0, X5 / 3.2, X6 / 3.4, X6 / 3.6
Max. potential difference AGND1/2 <-> M	± 1 V
Type of inputs	Voltage bipolar, current unipolar, temperature measurement
Line frequency suppression	Not available
Time constant of the input filter	Replacement device: Voltage: 100 µs, current 100 µs Existing device: no RC combination available
Conversion cycle	Replacement device: 2 ms over 8 inputs, 1 s during temperature measurement Existing device: 30 ms to 150 ms, depending on configuration
Protection against reversed voltage	Yes
Overvoltage protection	Up to 30 V DC

For further information, please refer to the existing documentation System description Advant Controller 31.

Analog voltage input

Data	Value
Input resistance	Replacement device: > 100 kΩ Existing device: > 1 MΩ
Measuring ranges nominal values	Replacement device: ± 10 V, ± 5 V Existing device: ± 10 V, ± 5 V, ± 500 mV, ± 50 mV

Data	Value
Resolution	12 bit + sign (measuring range ± 10 V) 11 bit + sign (measuring range ± 5 V)
Total error	Replacement device: ± 1 % of full range value Existing device: ± 0.5 % of full range value
Common mode input voltage range (e.g. X2 / 2.1, reference e.g. X2 / 1.0 (AGND1))	-10 V ... +10 V

For further information, please refer to the existing documentation System description Advant Controller 31.

Current input 0 … 20 mA / 4 … 20 mA

Total error:

Replacement device: ±1 % of full range value ± tolerance of current-sensing resistor

Existing device: ±0.5 % of full range value + tolerance of current-sensing resistor

Pt100/Pt1000 input

Data	Value
Measurement method	Replacement device: 3-wire configuration Existing device: 4-wire configuration. It is no longer possible to connect sensors in series.
Evaluation errors in measuring range -50… +400 °C	Replacement device: ± 1 % of full range value Existing device: ± 0.5 % of full range value at Pt100, ± 1 % of full range value at Pt1000
Current source for Pt100/Pt1000 resistors	The replacement device has a constant current source that is alternately connected to up to 8 analog channels (depending on configuration).

Unused input channels

See existing documentation 07AI91.

Connection of other temperature-dependent resistors

Other temperature-dependent resistors cannot be used in the replacement device.

Input with thermocouples

Thermocouples cannot be used in the replacement device. The existing documentation is no longer valid.

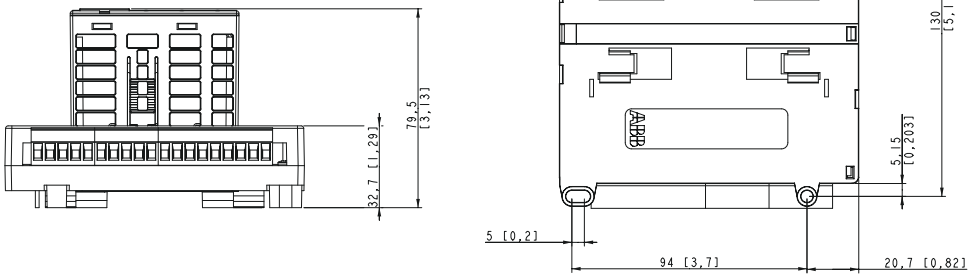
Connection to the CS31 bus

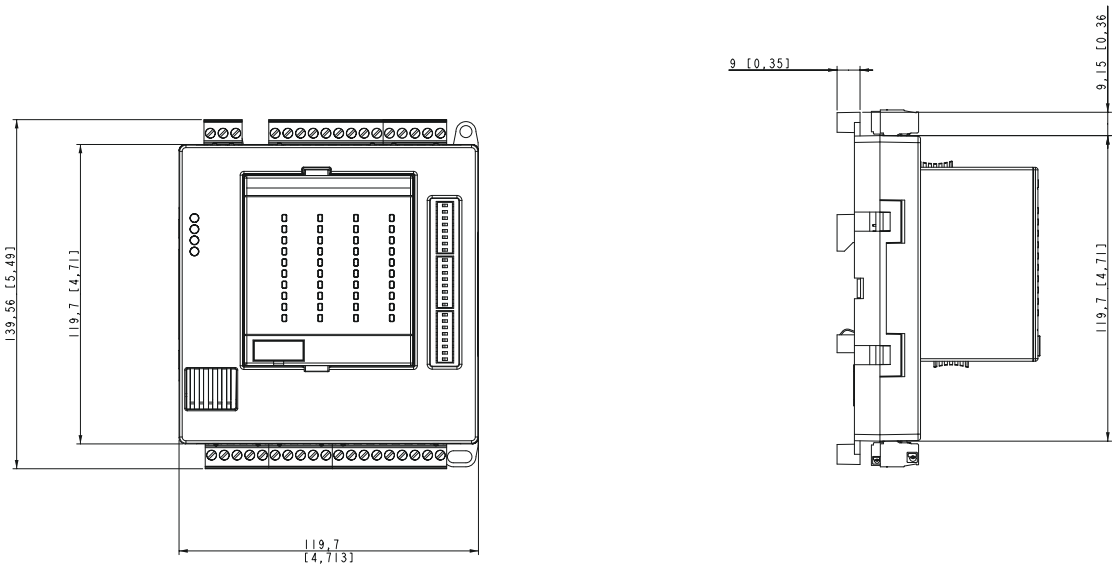
Data	Value
Connections	X1 / B2, X1 / B1
CS31 bus type	01 (analog input)
Terminating resistor	Not available (must be provided externally if needed)

Mechanical data

Data	Value
Width x height x depth	Replacement device: 120 x 140 x approx. 80 mm Existing device: 120 x 140 x 85 mm
Weight	Replacement device: 384 g Existing device: 450 g
Dimensions for mounting	See operating and assembly instructions of the replacement device (3ADR020086M0401)

Mounting information





The dimensions are in mm and in brackets in inch.

The dimensions for the assembly holes are the same for the replacement device and the existing device.

To assemble or disassemble the replacement device, grab the device at the housing and not directly at the S500 module.

Ordering data

Order No.	Scope of delivery
1SAP 800 200 R0010	Analog input module 07AI91-AD 1 3-pole terminal block 3 5-pole terminal blocks 2 9-pole terminal blocks

Replacement device 07DC91-AD

Fig. 791: 3ADR331192S0015_07DC91-AD

The replacement device 07DC91-AD of the AC31 adapter series replaces the existing device 07DC91 of the 90 series.

During the development of the replacement device, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07DC91 remains valid and serves as reference *(system description Advant Controller 31)*. The document structure of this document is based on the document structure of the existing documentation.

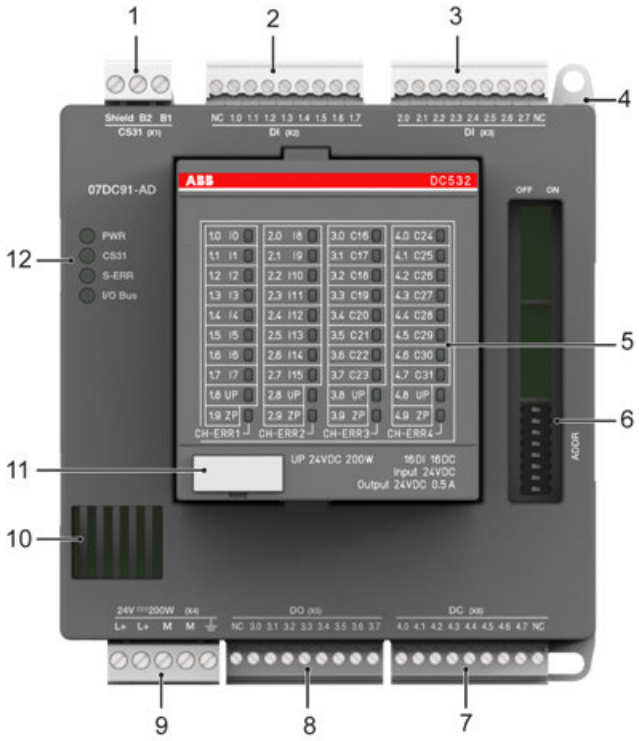
This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement devices 07DC91-AD can be found in the operating and assembly instructions of device 07DC91-AD: 3ADR020083M0401. Please note that for device 07DC91 no separate operating and assembly instructions are available.

Please also observe the system data as well as the information on CS31 bus ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Device configuration



1 Connection for CS31 bus (X1)
2 8 digital inputs 24 V DC (X2)
3 8 digital inputs 24 V DC (X3)
4 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
5 Status LEDs for DC532
6 DIP switch for address setting (ADDR)
7 8 digital inputs/outputs 24 V DC / 0.5 A (X6)
8 8 digital outputs (X5)
9 Supply 24 V DC (X4)
10 Ventilation
11 TA525: Label
12 4 Status LEDs

LED display

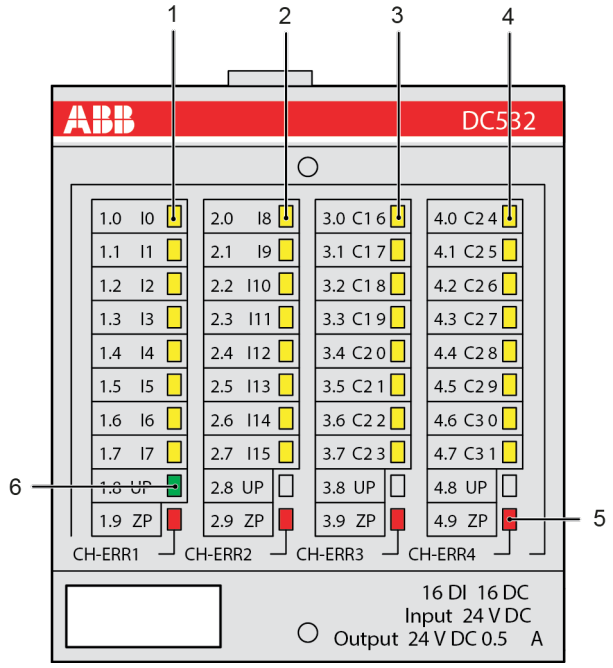


Fig. 792: Front view: DC532

No.	Displays of module
1	8 yellow LEDs to indicate the signal status of the digital inputs (X2).
2	8 yellow LEDs to indicate the signal status of the digital inputs (X3).
3	8 yellow LEDs to indicate the signal status of the digital outputs (X5).
4	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X6).
5	4 red LEDs to indicate errors (of DC532 module).
6	1 green LED to indicated the status of the supply voltage of the DC532 module (is supplied via X4).

The replacement device does not provide a test button to measure functionality.

Connections

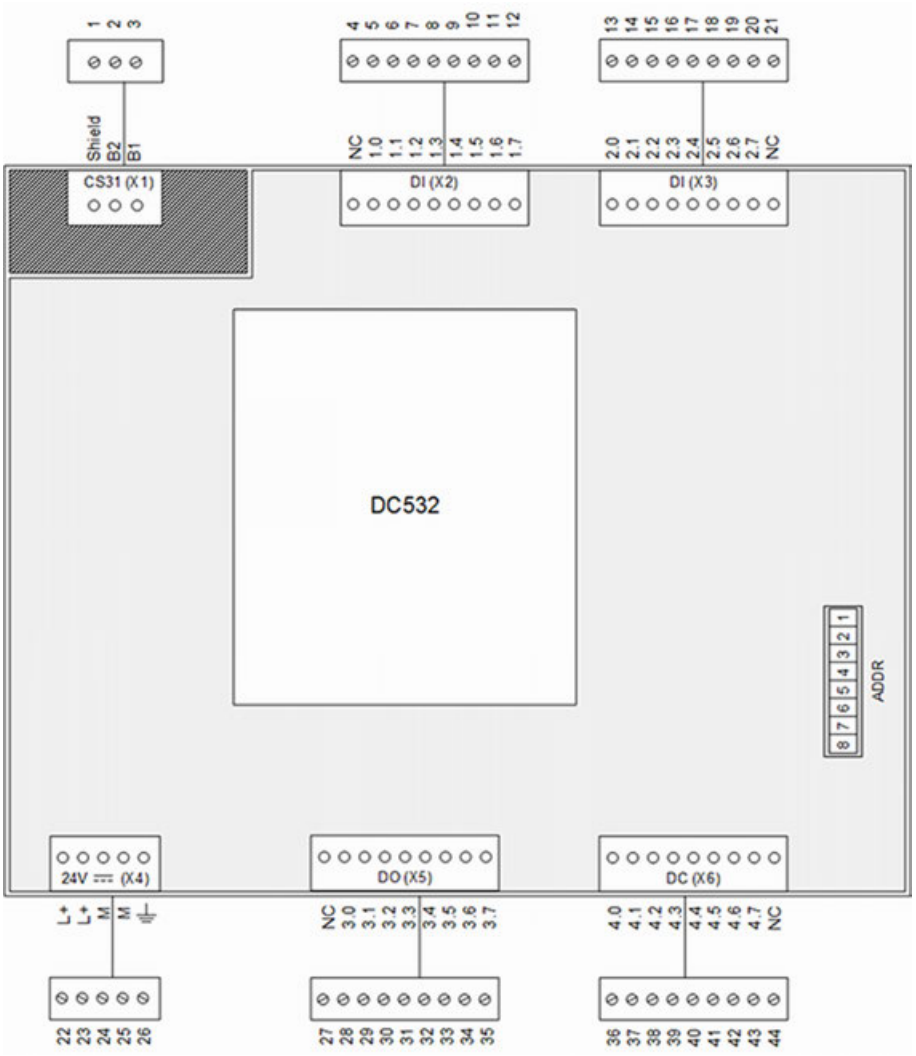


Fig. 793: Connection

Table 339: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	No internal connection
X1 / B2	2	BUS 2
X1 / B1	3	BUS 1

The shield connection of the CS31 bus is not galvanically connected to the functional earth of the supply voltage.

Table 340: Pin assignment DI (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / NC	4	No internal connection
X2 / 1.0	5	DC532 / I0
X2 / 1.1	6	DC532 / I1

Connector / Terminal	Pin	Assignment / Signal
X2 / 1.2	7	DC532 / I2
X2 / 1.3	8	DC532 / I3
X2 / 1.4	9	DC532 / I4
X2 / 1.5	10	DC532 / I5
X2 / 1.6	11	DC532 / I6
X2 / 1.7	12	DC532 / I7

Table 341: Pin assignment DI (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / 2.0	13	DC532 / I8
X3 / 2.1	14	DC532 / I9
X3 / 2.2	15	DC532 / I10
X3 / 2.3	16	DC532 / I11
X3 / 2.4	17	DC532 / I12
X3 / 2.5	18	DC532 / I13
X3 / 2.6	19	DC532 / I14
X3 / 2.7	20	DC532 / I15
X3 / NC	21	No internal connection

Table 342: Pin assignment DC (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / 4.0	36	DC532 / C24
X6 / 4.1	37	DC532 / C25
X6 / 4.2	38	DC532 / C26
X6 / 4.3	39	DC532 / C27
X6 / 4.4	40	DC532 / C28
X6 / 4.5	41	DC532 / C29
X6 / 4.6	42	DC532 / C30
X6 / 4.7	43	DC532 / C31
X6 / NC	44	No internal connection

Table 343: Pin assignment DO (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / NC	27	No internal connection
X5 / 3.0	28	DC532 / C16
X5 / 3.1	29	DC532 / C17
X5 / 3.2	30	DC532 / C18
X5 / 3.3	31	DC532 / C19
X5 / 3.4	32	DC532 / C20
X5 / 3.5	33	DC532 / C21
X5 / 3.6	34	DC532 / C22
X5 / 3.7	35	DC532 / C23

Table 344: Pin assignment 24 V DC 200 W (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / L+	22	L+
X4 / L+	23	L+
X4 / M	24	M
X4 / M	25	M
X4 / FE	26	FE

The device 07DC91-AD has 16 digital outputs, each with 0.5 A output current. This results in a maximum output current of 8 A. With an output current of 4 A and higher, both terminals (L+) of connector X4 must be used.

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

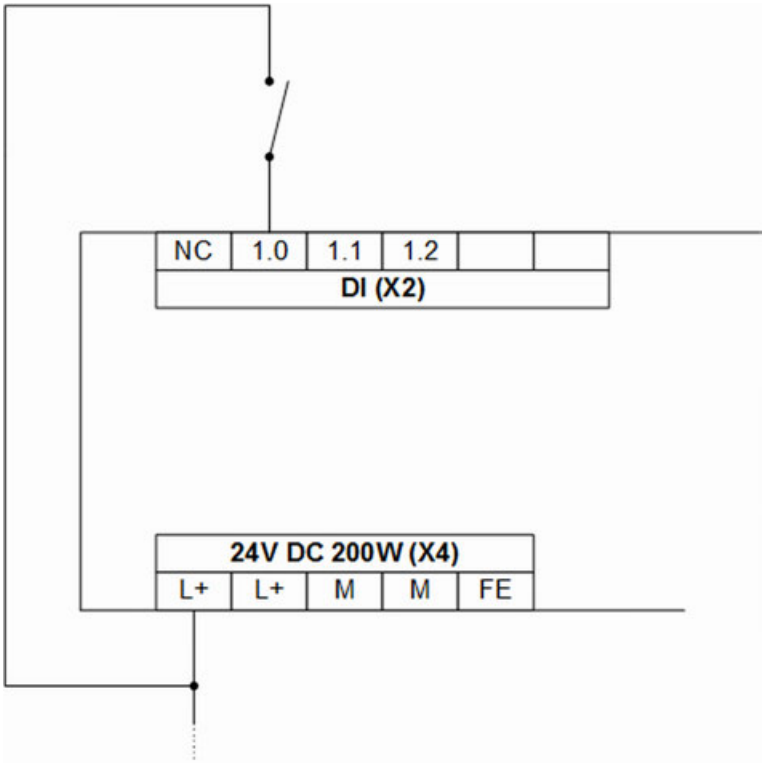


Fig. 794: Connection example: digital input

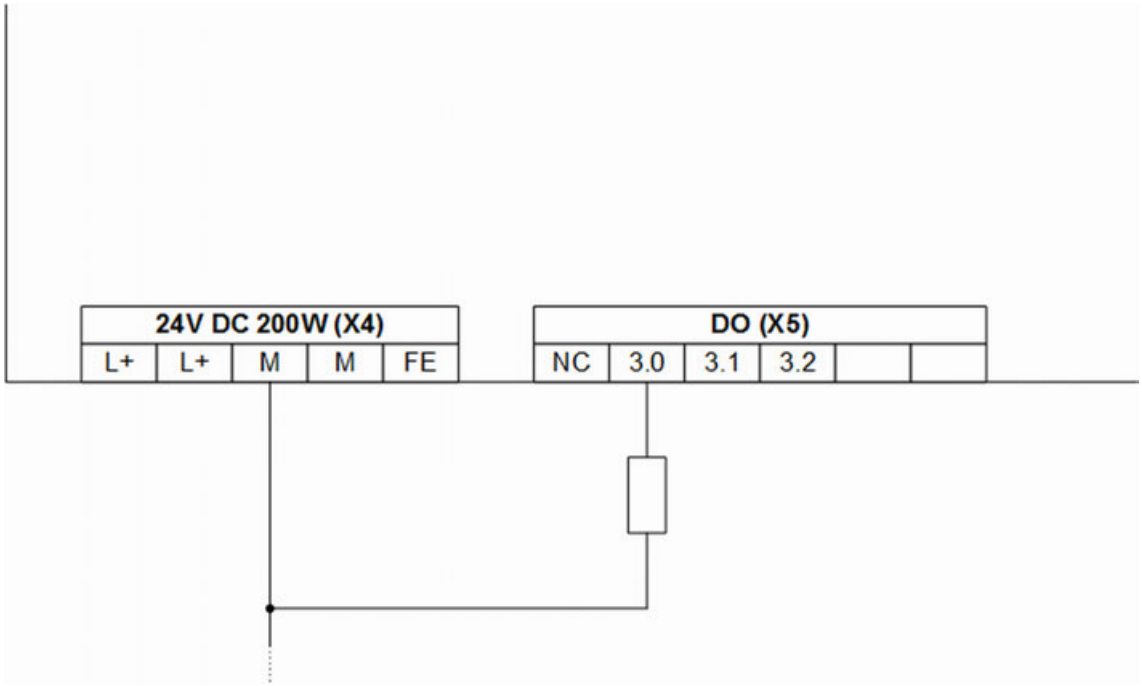


Fig. 795: Connection example: digital output

Addressing

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data ↬ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

Table 345: CS31 bus (16 inputs / 16 outputs)

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X2 / 1.0
			X2 / 1.1
			X2 / 1.2
			X2 / 1.3
			X2 / 1.4
			X2 / 1.5
			X2 / 1.6
			X2 / 1.7
2	8 bit input (send)	0 ... 7	X3 / 2.0
			X3 / 2.1
			X3 / 2.2
			X3 / 2.3
			X3 / 2.4
			X3 / 2.5
			X3 / 2.6
			X3 / 2.7

Byte	Type	Bit	Connector / Terminal
3	8 bit output (receive)	0 ... 7	X5 / 3.0
			X5 / 3.1
			X5 / 3.2
			X5 / 3.3
			X5 / 3.4
			X5 / 3.5
			X5 / 3.6
			X5 / 3.7
4	8 bit output (receive)	0 ... 7	X6 / 4.0
			X6 / 4.1
			X6 / 4.2
			X6 / 4.3
			X6 / 4.4
			X6 / 4.5
			X6 / 4.6
			X6 / 4.7

Table 346: CS31 bus (24 inputs / 16 outputs)

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X2 / 1.0 ... 1.7
2	8 bit input (send)	0 ... 7	X3 / 2.0 ... 2.7
3	8 bit output (receive)	0 ... 7	X5 / 3.0 ... 3.7
4	8 bit input (send)	0 ... 7	X6 / 4.0 ... 4.7
5	8 bit output (receive)	0 ... 7	X6 / 4.0 ... 4.7

I/O configuration

The existing device had a DIP switch on the upper printed circuit board. Since the replacement device is not equipped with an upper printed circuit board, the white DIP switch is arranged on the lower printed circuit board instead.

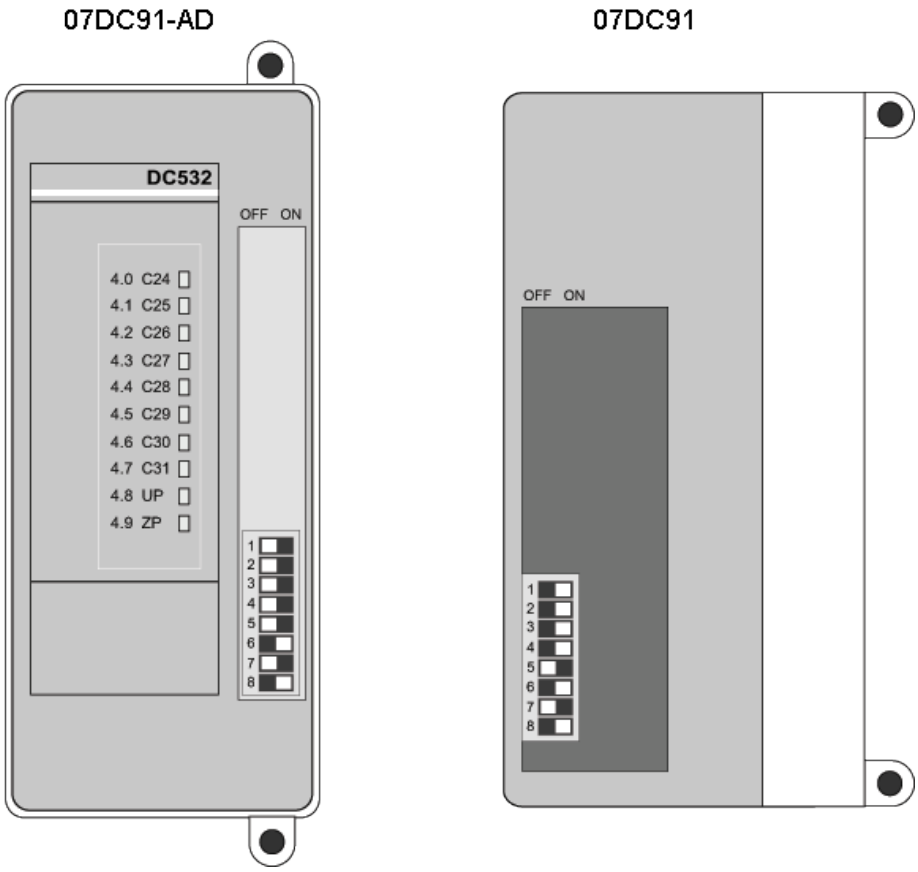


Fig. 796: DIP switch for 07DC91-AD

The DIP switches are read by the device only once after the supply voltage has been connected.

For further information, please refer to the existing documentation System description Advant Controller 31.

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.
- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

The RAM is checked during the initialization of the device. In addition, the firmware in the Flash memory is checked by means of a checksum during initialization. When the control system (PLC/central unit) is stopped during normal operation, the outputs of the device 07DC91-AD are switched off. The inputs remain active. The outputs are also switched off in case of a malfunction of the CS31 bus.

Diagnosis and displays

LEDs are used for diagnosis and display purposes. In addition, some diagnosis information can be transmitted via the CS31 bus.

The replacement device does not provide a test button to measure functionality.

Table 347: Diagnosis information of the CS31 bus

Error description	Channel	Error code (CODESYS)	Error code (CS31 bus)	Description
Device error	0	43	1	Internal error
Channel error	0, 4, 8, 12 *)	46	4	Overload or short circuit on a digital output

*) The channel numbers are grouped as follows:

0 - for X5/3.0, X5/3.1, X5/3.2, X5/3.3

4 - for X5/3.4, X5/3.5, X5/3.6, X5/3.7

8 - for X6/4.0, X6/4.1, X6/4.2, X6/4.3

12 - for X6/4.4, X6/4.5, X6/4.6, X6/4.7

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

Table 348: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.
- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 349: LEDs of the S500 module DC532

LED	Status	Color	LED off	LED on	LED flashes
I0…I7 (see No. 1 in the following figure)	Digital inputs	Yellow	Input is not activated	Input is activated (input voltage is indicated even if supply is switched off)	-
- I8…I15 (see No. 2 in the following figure)	Digital inputs	Yellow	Input is not activated	Input is activated (input voltage is indicated even if supply is switched off)	-
C16…C23 (see No. 3 in the following figure)	Digital outputs	Yellow	Output is not activated	Output is activated	-
C24…C31 (see No. 4 in the following figure)	Digital inputs or digital outputs	Yellow	Input or output is not activated	Input is activated (input voltage is indicated even if supply is switched off)	-
Error indications left (see No. 5 in the following figure)	Error indication	Red	No error	Internal error	-
Error indications right (see No. 5 in the following figure)	Error indication	Red	No error	Internal error	Overload or short circuit on a channel of the corresponding group
Indication supply voltage (see No. 6 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-

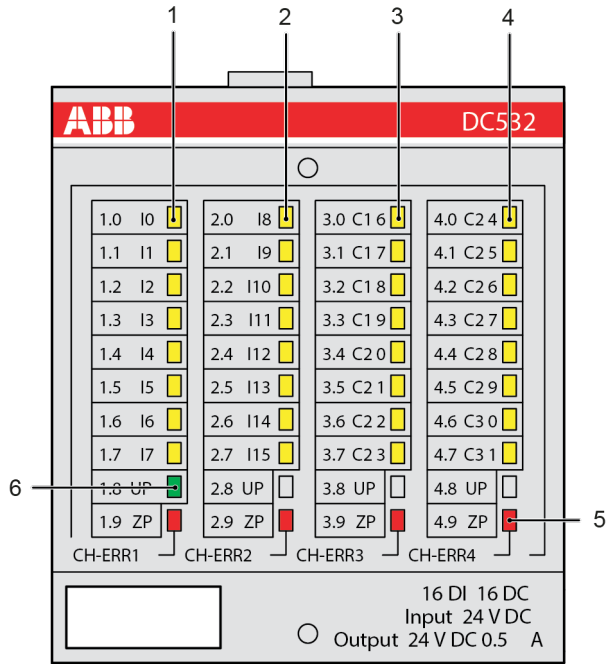


Fig. 797: Front view: DC532

Technical data

This section provides additional information on section *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*. In case of doubt, the following information applies.

Technical Data of the complete device

Data	Value
Process voltage:	
-> Connections	X4/L+ (pin 22), X4/L+ (pin 23), X4/M (pin 24), X4/M (pin 25)
-> Fuse for L+	10 A, fast acting
- Galvanic isolation	No
Current consumption:	
-> via L+	0.19 A and max. 0.5 A per output
- Inrush current via L+ (when voltage is switched on)	0.17 A^2s
Power consumption	Replacement device: 200 W Existing device: 202 W
Max. power dissipation within the module (outputs unloaded)	Replacement device: 6 W Existing device: 5 W
Address setting and configuration	DIP switch on right side of the housing
Operating and error indications	Replacement device: 41 LEDs Existing device: 33 LEDs

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Technical data of the digital inputs

Data	Value
Connections	X2/1.0, X2/1.1, X2/1.2, X2/1.3, X2/1.4, X2/1.5, X2/1.6, X2/1.7, X3/2.0, X3/2.1, X3/2.2, X3/2.3, X3/2.4, X3/2.5, X3/2.6, X3/2.7
Input type according to EN 61131-2	Type 1 (realized through current sink)
Input delay: 0 -> 1 or 1 -> 0 *)	Replacement device: Typ. 8 ms Existing device: Typ. 7 ms
Indication of the input signals	Replacement device: One yellow LED per channel. The LED corresponds functionally to the input signal. Existing device: One green LED per channel. The LED corresponds functionally to the input signal.
Input signal voltage:	24 V DC
-> 0 signal	Replacement device: -3 V…+5 V Existing device: -30 V…+5 V
-> Undefined signal	Replacement device: > +5 V…< +15 V Existing device: > +5 V…< +13 V
-> 1 signal	Replacement device: +15 V…+30 V Existing device: +13 V…+30 V
-> Residual ripple at 0 signal	Replacement device: within -3 V…+5 V Existing device: within -30 V…+5 V
-> Residual ripple at 1 signal	Replacement device: within +15 V…+30 V Existing device: within +13 V…+30 V
Input current per channel:	
Input voltage +24 V	Replacement device: Typ. 5 mA Existing device: Typ. 7 mA
Input voltage +5 V	Replacement device: > 1 mA Existing device: ≥ 1 mA
Input voltage +15 V	Replacement device: > 5 mA Existing device: ≥ 2 mA (at input voltage +13 V)

Data	Value
Input voltage +30 V	Replacement device: < 8 mA Existing device: ≤ 9 mA
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m
Protection against reversed voltage	Yes
Overvoltage protection	Up to 30 V DC

*) Input delay of the S500 module DC532. The transmission rate via serial buses has not been taken into account.

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical data of the digital outputs

Data	Value
Connections	X5/3.0, X5/3.1, X5/3.2, X5/3.3, X5/3.4, X5/3.5, X5/3.6, X5/3.7
Type of digital outputs	High-side switch
Demagnetization with inductive load	With a varistor inside the device (with other circuitry)
Switching frequency with ohmic load	On request
Output voltage at signal 1	X4 / L+ (typ. 24 V) -0.8 V
Output delay: 0 -> 1 or 1 -> 0	On request
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical data of the configurable inputs/outputs

Data	Value
Connections	X6/4.0, X6/4.1, X6/4.2, X6/4.3, X6/4.4, X6/4.5, X6/4.6, X6/4.7
Use as digital input	See *Chapter 1.6.2.3.3.5.4.8.2 “Technical data of the digital inputs” on page 3993*
Use as digital output	See *Chapter 1.6.2.3.3.5.4.8.3 “Technical data of the digital outputs” on page 3994*

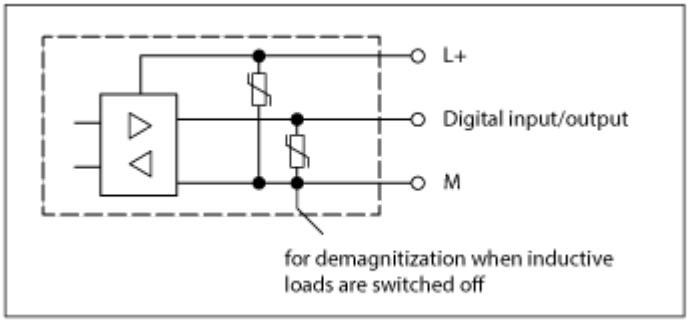


Fig. 798: Protective circuits inputs/outputs

Due to the changed protective circuit on the inputs and outputs, the restrictions concerning the input signal voltage described in the existing documentation no longer apply.

When the channels of connector X6 are to be used as inputs, the respective outputs (high-end switches) must be switched off.

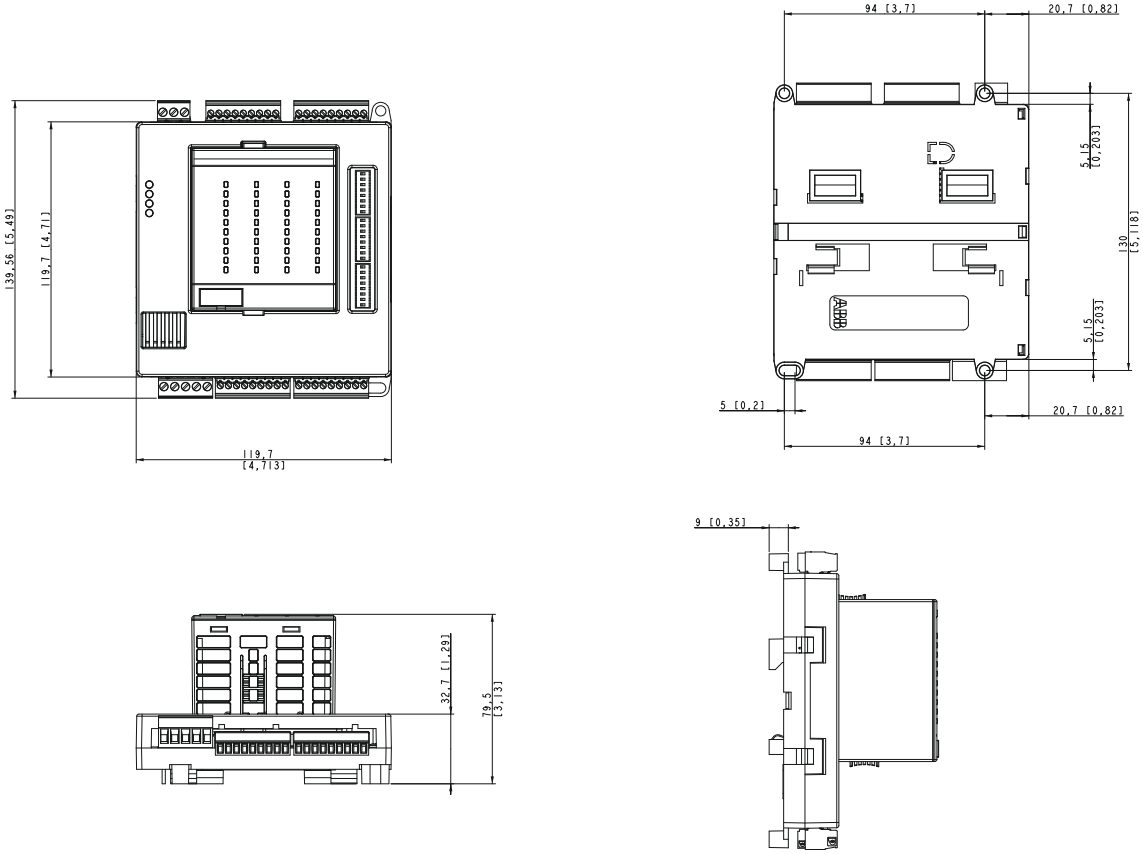
Connection to the CS31 bus

Data	Value
Connections	X1/B2, X1/B1
CS31 bus type	04 (digital input/output)
Terminating resistor	Not available (must be provided externally if needed)

Mechanical data

Data	Value
Width x height x depth	Replacement device: 120 x 140 x approx. 80 mm Existing device: 120 x 140 x 85 mm
Weight	Replacement device: 351 g (incl. terminals) Existing device: 450 g
Dimensions for mounting	See operating and assembly instructions of the replacement device (3ADR020083M0401)

Assembly / Disassembly



The dimensions are in mm and in brackets in inch.

The dimensions for the assembly holes are the same for the replacement device and the existing device.

To assemble or disassemble the replacement device, grab the device at the housing and not directly at the S500 module.

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

Ordering data

Order No.	Scope of delivery
1SAP 800 300 R0010	Digital input/output module 07DC91-AD 1 5-pin terminal block (5.08 mm grid space) 1 3-pin terminal block (5.08 mm grid space) 4 9-pin terminal blocks (3.81 mm grid space)

Replacement device 07DC92-AD

Fig. 799: 3ADR333196F0015_07DC92-AD

The replacement device 07DC92-AD of the AC31 adapter series replaces the existing device 07DC92 of the 90 series.

During the development of the replacement device, care was taken to keep the device configuration identical to the configuration of the existing device. Thus, the existing documentation of device 07DC92 remains valid and serves as a reference *(system description Advant Controller 31*). The document structure of this document is based on the document structure of the existing documentation.

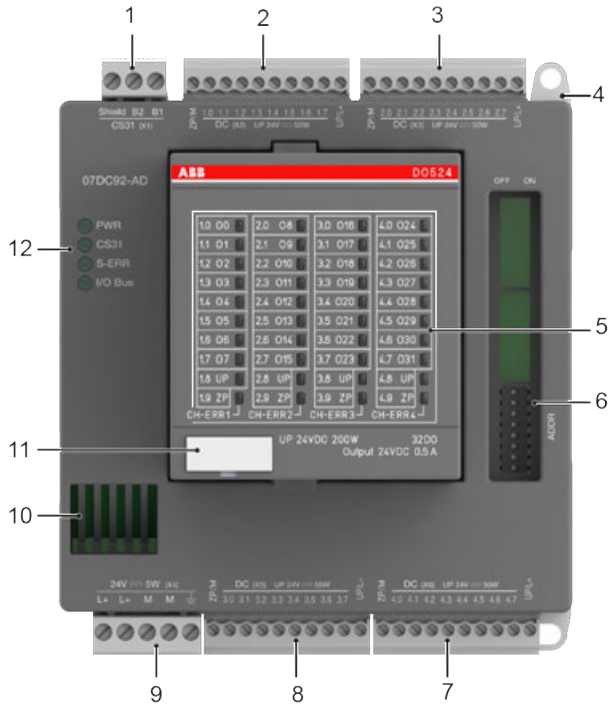
This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement device 07DC92-AD can be found in the operating and assembly instructions of device 07DC92-AD: 3ADR020151M0401. Please note that no separate operating and assembly instructions are available for device 07DC92.

Please also observe the system data as well as the information on CS31 bus ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Device configuration



1 Connector X1: CS31 bus
2 Connector X2: 8 DC + voltage supply (incl. DO524)
3 Connector X3: 8 DC + voltage supply (incl. DO524)
4 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
5 Status LEDs for DO524
6 DIP switch for address setting (ADDR)
7 Connector X6: 8 DC + voltage supply (incl. DO524)
8 Connector X5: 8 DC + voltage supply (incl. DO524)
9 Connector X4: Voltage supply (incl. DO524)
10 Ventilation
11 TA525: Label
12 4 LEDs to display the status of the complete 07DC92-AD device

LED display

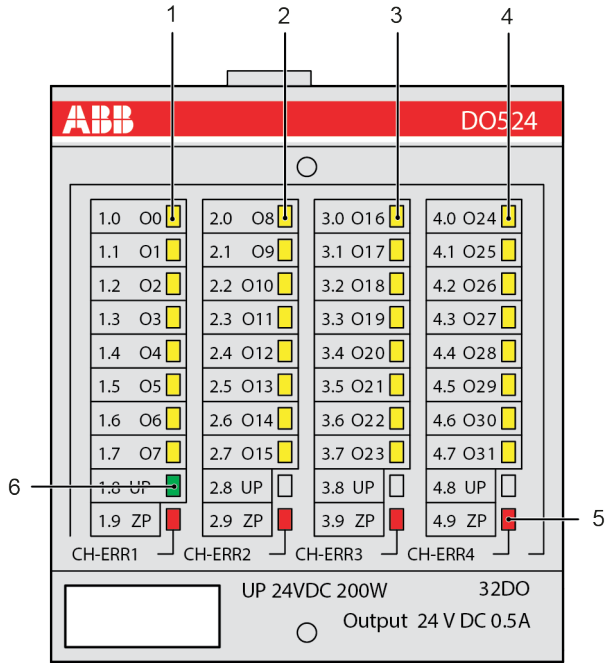


Fig. 800: LEDs DO524

No.	Displays of module
1	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X2).
2	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X3).
3	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X5).
4	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X6).
5	4 red LEDs to indicate errors (from the DO524 module).
6	1 green LED to indicate the status of the supply voltage of the DO524 module (is supplied via UP/L+).

The replacement device does not provide a test button to measure functionality.

Connections

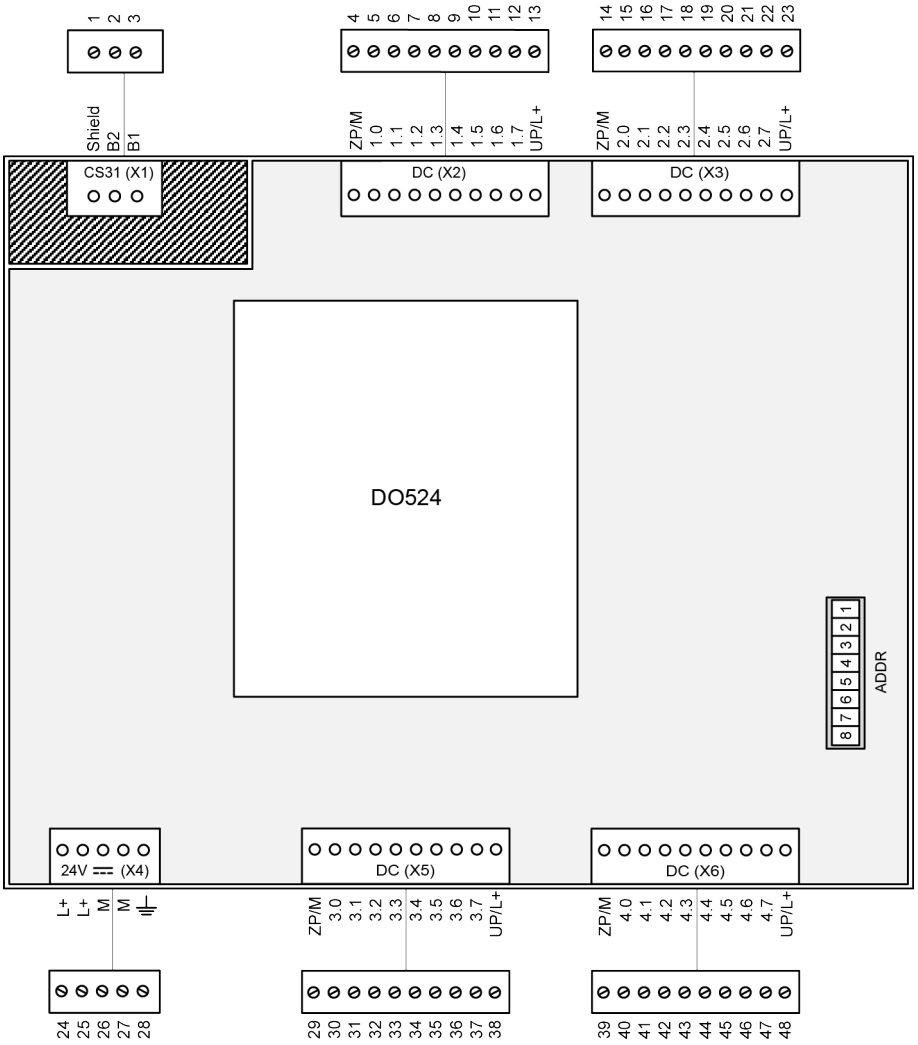


Fig. 801: Connection

Table 350: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	No internal connection
X1 / B2	2	BUS 2
X1 / B1	3	BUS 1

The shield connection of the CS31 bus is not galvanically connected to the functional earth of the supply voltage.

Table 351: Pin assignment DC (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / ZP/M	4	ZP/M
X2 / 1.0	5	DO524 / O0
X2 / 1.1	6	DO524 / O1

Connector / Terminal	Pin	Assignment / Signal
X2 / 1.2	7	DO524 / O2
X2 / 1.3	8	DO524 / O3
X2 / 1.4	9	DO524 / O4
X2 / 1.5	10	DO524 / O5
X2 / 1.6	11	DO524 / O6
X2 / 1.7	12	DO524 / O7
X2 / UP/L+	13	UP/L+

Table 352: Pin assignment DC (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / ZP/M	14	ZP/M
X3 / 2.0	15	DO524 / O8
X3 / 2.1	16	DO524 / O9
X3 / 2.2	17	DO524 / O10
X3 / 2.3	18	DO524 / O11
X3 / 2.4	19	DO524 / O12
X3 / 2.5	20	DO524 / O13
X3 / 2.6	21	DO524 / O14
X3 / 2.7	22	DO524 / O15
X3 / UP/L+	23	UP/L+

Table 353: Pin assignment 24 V DC (X4)

Connector / Terminal	Pin	Assignment / Signal
X4 / L+	24	L+
X4 / L+	25	L+
X4 / M	26	M
X4 / M	27	M
X4 / FE	28	FE

Table 354: Pin assignment DC (X5)

Connector / Terminal	Pin	Assignment / Signal
X5 / ZP/M	29	ZP/M
X5 / 3.0	30	DO524 / O16
X5 / 3.1	31	DO524 / O17
X5 / 3.2	32	DO524 / O18
X5 / 3.3	33	DO524 / O19
X5 / 3.4	34	DO524 / O20
X5 / 3.5	35	DO524 / O21
X5 / 3.6	36	DO524 / O22
X5 / 3.7	37	DO524 / O23
X5 / UP/L+	38	UP/L+

Table 355: Pin assignment DC (X6)

Connector / Terminal	Pin	Assignment / Signal
X6 / ZP/M	39	ZP/M
X6 / 4.0	40	DO524 / O24
X6 / 4.1	41	DO524 / O25
X6 / 4.2	42	DO524 / O26
X6 / 4.3	43	DO524 / O27
X6 / 4.4	44	DO524 / O28
X6 / 4.5	45	DO524 / O29
X6 / 4.6	46	DO524 / O30
X6 / 4.7	47	DO524 / O31
X6 / UP/L+	48	UP/L+

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

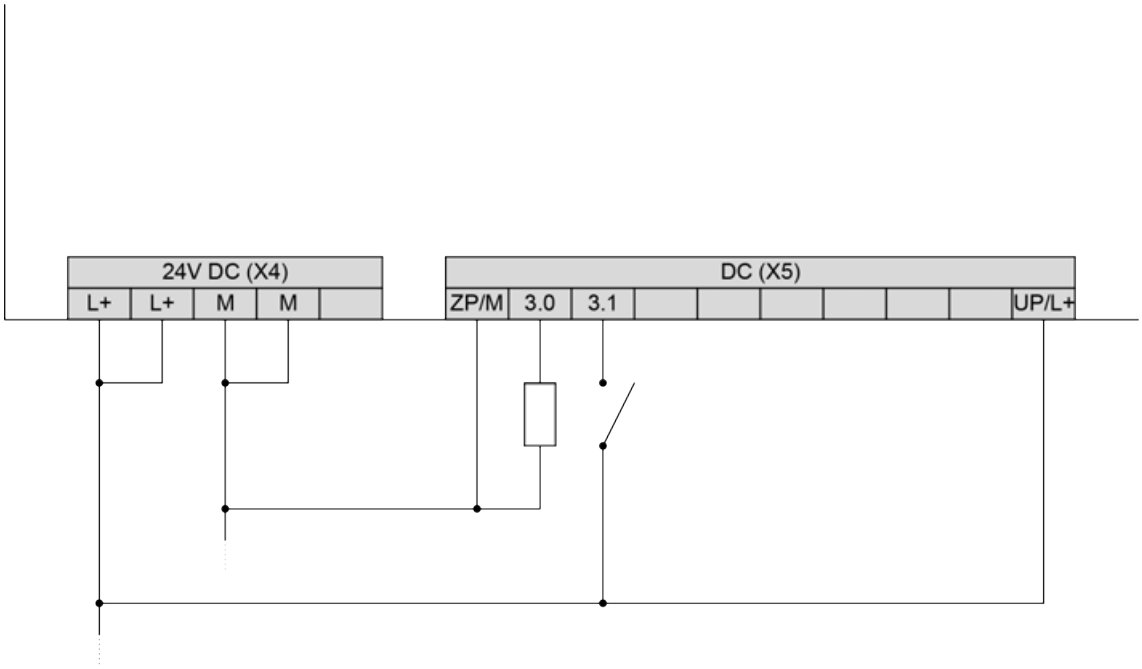


Fig. 802: Connection example: digital output

Addressing

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

Table 356: CS31 bus (32 inputs / 32 outputs)

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X2 / 1.0 ... 1.7
2	8 bit input (send)	0 ... 7	X3 / 2.0 to 2.7
3	8 bit input (send)	0 ... 7	X5 / 3.0 to 3.7
4	8 bit input (send)	0 ... 7	X6 / 4.0 to 4.7
5	8 bit output (receive)	0 ... 7	X2 / 1.0 ... 1.7
6	8 bit output (receive)	0 ... 7	X3 / 2.0 to 2.7
7	8 bit output (receive)	0 ... 7	X5 / 3.0 to 3.7
8	8 bit output (receive)	0 ... 7	X6 / 4.0 to 4.7

Table 357: CS31 bus (32 outputs)

Byte	Type	Bit	Connector / Terminal
1	8 bit output (receive)	0 ... 7	X2 / 1.0 ... 1.7
2	8 bit output (receive)	0 ... 7	X3 / 2.0 to 2.7
3	8 bit output (receive)	0 ... 7	X5 / 3.0 to 3.7
4	8 bit output (receive)	0 ... 7	X6 / 4.0 to 4.7

NOTICE!

In case of overloading or a short-circuit, the output limits the electricity and switches off thermally. The LED of the overloaded output is also switched off and the corresponding error indication of the DO524 flashes.

I/O configuration

The existing device had a DIP switch on the upper printed circuit board. Since the replacement device is not equipped with an upper printed circuit board, the white DIP switch is arranged on the lower printed circuit board instead.

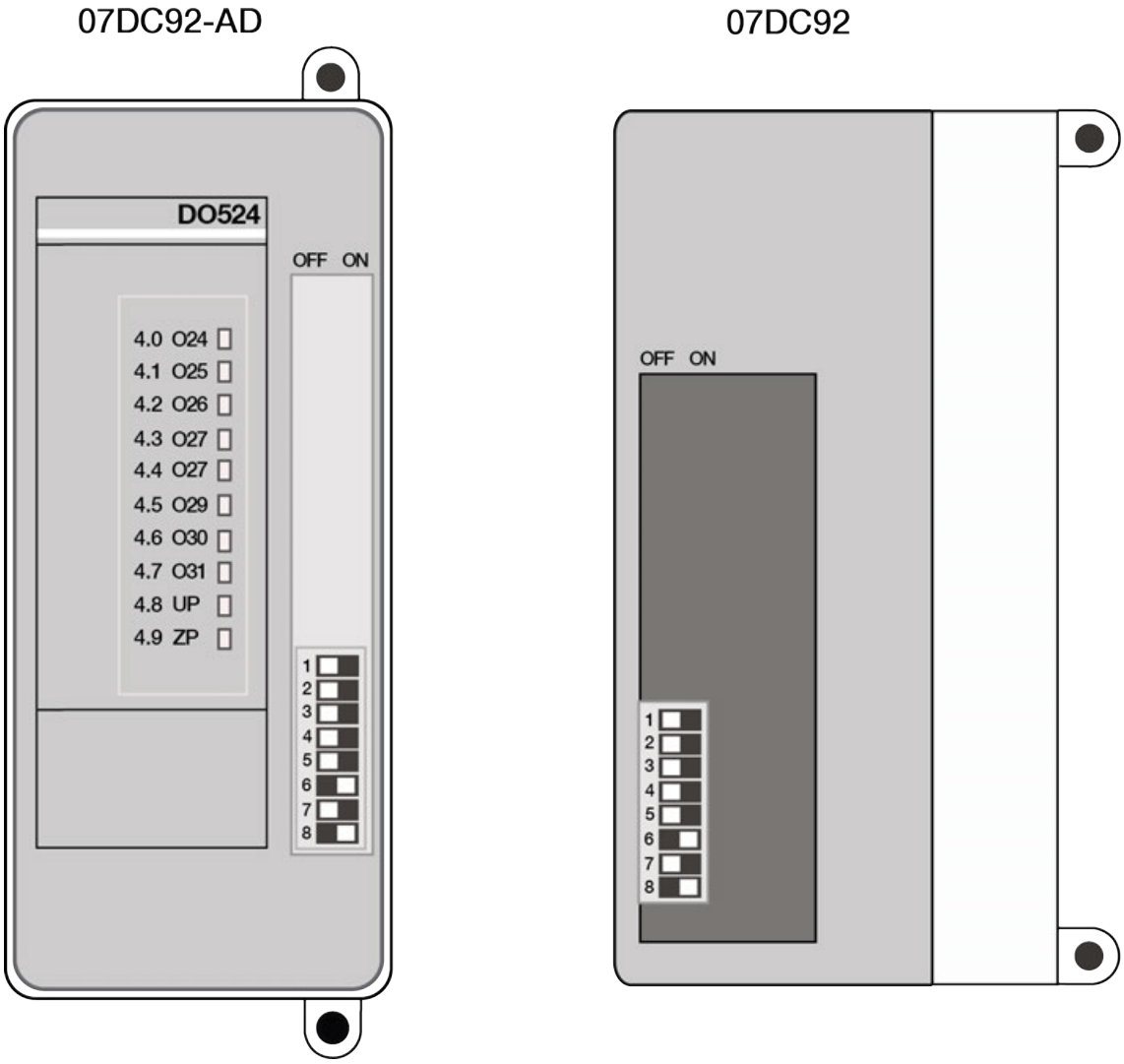


Fig. 803: DIP switch for 07DC92-AD:

The DIP switches are read by the device only once after the supply voltage has been connected.

For further information, please refer to the existing documentation System description Advant Controller 31.

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.

- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

Diagnosis and Displays

LEDs are used for diagnosis and display purposes. In addition, some diagnosis information can be transmitted via the CS31 bus.

The replacement device does not provide a test button to measure functionality.

Table 358: Diagnosis information of the CS31 bus

Error description	Channel	Error code (CODESYS)	Error code (CS31 bus)	Description
Device error	0	43	1	Internal error
Channel error	0, 8, 15 *)	46	4	Overload or short circuit on a digital output

*) The channel numbers are grouped as follows:

0 - for X2 / 1.0 to 1.7

8 - for X2 / 2.0 to 2.7

15 - for X5 / 3.0 to 3.7 and X6 / 4.0 to 4.7

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

Table 359: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.
- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 360: LEDs of the S500 module DO524

LED	Status	Color	LED off	LED on	LED flashes
O0…O7 (see No. 1 in the following figure)	Digital inputs/ outputs	Yellow	Input/output is not activated	Input/output is activated (input voltage is indicated even if supply is switched off)	-
I8 to I15 (see No. 2 in the following figure)	Digital inputs/ outputs	Yellow	Input/output is not activated	Input/output is activated (input voltage is indicated even if supply is switched off)	-
O16 to O23 (see No. 3 in the following figure)	Digital inputs/ outputs	Yellow	Input/output is not activated	Input/output is activated (input voltage is indicated even if supply is switched off)	-
C24 to C31 (see No. 4 in the following figure)	Digital inputs/ outputs	Yellow	Input/output is not activated	Input/output is activated (input voltage is indicated even if supply is switched off)	-
Error indications right (see No. 5 in the following figure)	Error indication	Red	No error	Internal error	Overload or short circuit on a channel of the corresponding group
Indication supply voltage (see No. 6 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-

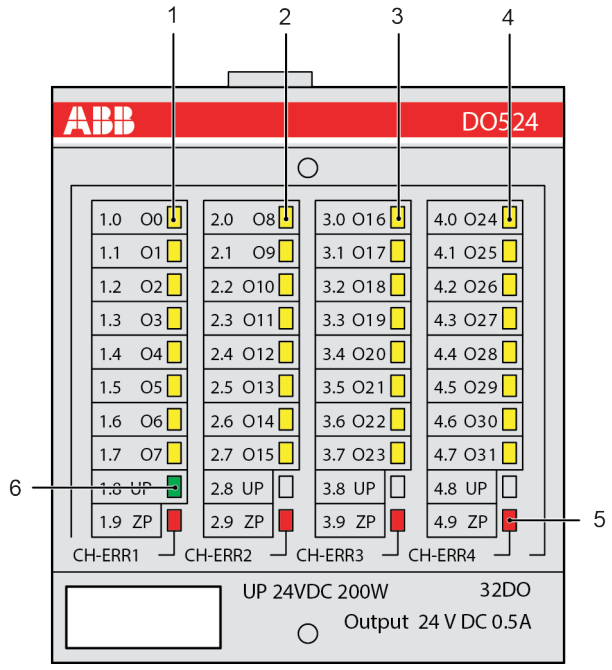


Fig. 804: LEDs DO524

Technical data

This section provides additional information on section *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*. In case of doubt, the following information applies.

Technical data of the complete device

Data	Value
Process voltage:	
-> Connections L+	X2 (pin 13) X3 (pin 23), X4 (pin 24, pin 25) X5 (pin 38), X6 (pin 48)
-> Connections M	X2 (pin 4) X3 (pin 14) X4 (pin 26, pin 27) X5 (pin 29) X6 (pin 39)
-> Fuse for L+	10 A, fast acting
- Galvanic isolation	None (07DC92: Group against group, all groups in relation to the rest of the device
Current consumption:	
-> via L+	0.19 A and max. 0.5 A per output
- Inrush current via L+ (when voltage is switched on)	0.17 A²s

Data	Value
Power consumption	Replacement device: 200 W Existing device: 394 W
Max. power dissipation within the module (outputs unloaded)	Replacement device: 6 W Existing device: 5 W
Address setting and configuration	DIP switch right side of housing
Operating and error indications	Replacement device: 41 LEDs Existing device: 33 LEDs

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Changes to the process voltage connections

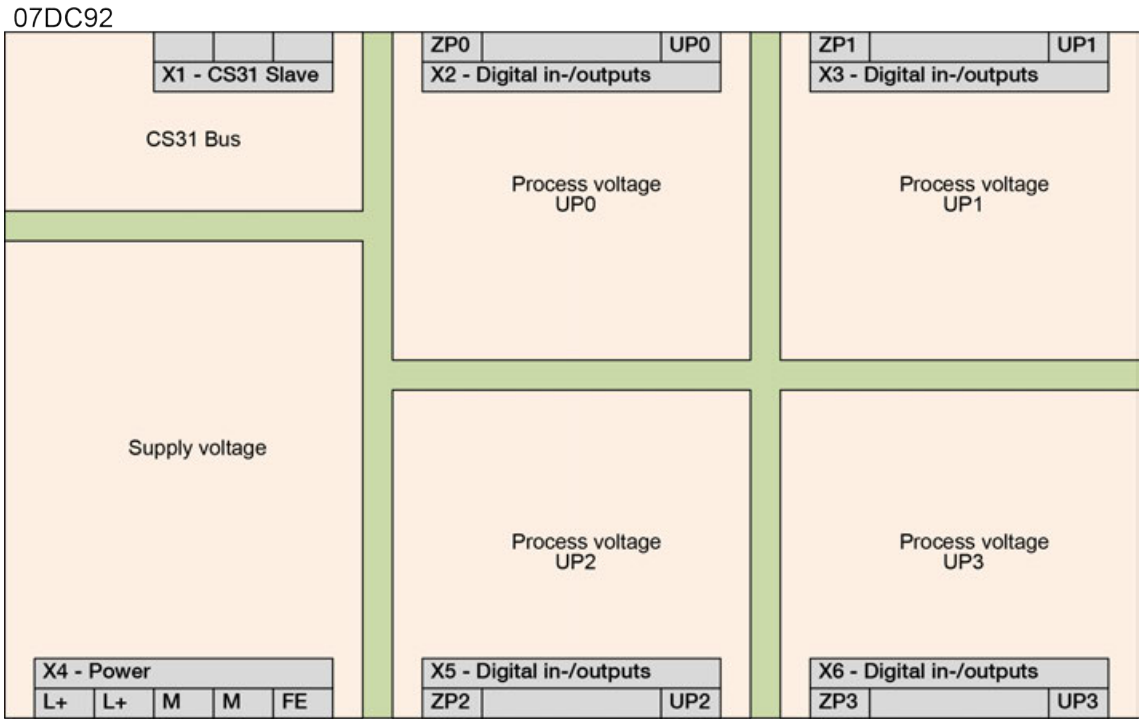


Fig. 805: Process voltage connections - 07DC92

CAUTION!
System damage caused by voltage!
Changed potential ranges!

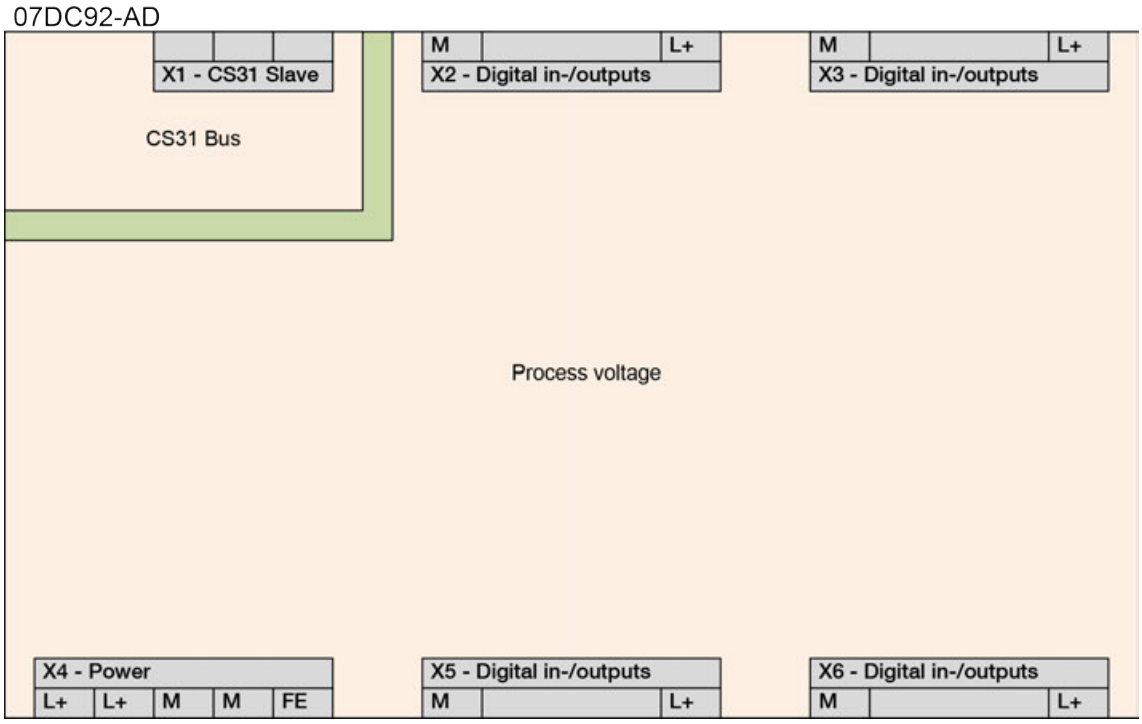


Fig. 806: Process voltage connections - 07DC92-AD

NOTICE!
Process voltage must always be connected to connector X4 on device 07DC92-AD.

Connector X4 also supplies the internal electronics for the device 07DC92-AD with 0.15 A.

Process voltage connections (X2, X3, X5, X6):

- *Maximum current for digital outputs X2 + X3: 4 A / 4 to*
- *Maximum current for digital outputs X5 + X6: 4 A / 4 to*

- *Input currents > 4 A require the connection of the second L+ contact of connector X4.*
- *For input currents > 8 A, additional L+ contacts from X2, X3, X5 or X6 must be used.*
- *The L+ contacts for the connectors X2, X3, X5 or X6 may be loaded with a maximum of 4 A.*

Technical details of the I/O channels as binary inputs

Data	Value
Connections	X2 / 1.0 to 1.7 X3 / 2.0 to 2.7 X5 / 3.0 to 3.7 X6 / 4.0 to 4.7
Input type according to EN 61131-2	Type 1 (realized through resistors)

Data	Value
Input delay: 0 -> 1 or 1 -> 0 *)	Replacement device: Type. 8 ms Existing device: Type. 7 ms
Indication of the input signals	Replacement device: One yellow LED per channel. The LED corresponds functionally to the input signal. Existing device: One green LED per channel. The LED corresponds functionally to the input signal.
Input signal voltage:	24 V DC
-> 0 signal	Replacement device: -3 V…+5 V Existing device: -6 V…+5 V
-> Undefined signal	Replacement device: > +5 V…< +15 V Existing device: > +5 V…< +13 V
-> 1 signal	Replacement device: +15 V…+30 V Existing device: +13 V…+30 V
-> Residual ripple at 0 signal	Replacement device: within -3 V…+5 V Existing device: within -6 V…+5 V
-> Residual ripple at 1 signal	Replacement device: within +15 V…+30 V Existing device: within +13 V…+30 V
Input current per channel:	
Input voltage +24 V	Replacement device: Type. 3.5 mA / 4 to Existing device: Type. 7 mA / 4 to
Input voltage +5 V	Replacement device: > 0.5 mA Existing device: ≥ 0.2 mA
Input voltage +15 V	Replacement device: > 2 mA Existing device: ≥ 2 mA (at input voltage +13 V)
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m
Protection against reversed voltage	Yes
Overvoltage protection	Up to 30 V DC

*) Input delay of the S500 module DO524. The transmission rate via serial buses has not been taken into account.

For further information, please refer to the existing documentation System description Advant Controller 31.

Technical details of the I/O channels as digital outputs

Data	Value
Connections	X2 / 1.0 to 1.7 X3 / 2.0 to 2.7 X5 / 3.0 to 3.7 X6 / 4.0 to 4.7
Type of digital outputs	High-side switch
Reference potentials for the outputs	M (07DC92: ZP0, ZP1, ZP2 and ZP3)
Supply voltage for the outputs	L+ (07DC92: UP0, UP1, UP2 and UP3)
Galvanic isolation	No (07DC92: Group against group, all groups in relation to the rest of the device
Output current (maximum value)	X2 + X3 = 4 A, X5 + X6 = 4 A (07DC92: 4 A per group)
Demagnetization with inductive load	Internally with a varistor (with other circuitry)
Switching frequency with ohmic load	On request
Output voltage at signal 1	X4 / L+ (typ. 24 V) -0.8 V
Output delay: 0 -> 1 or 1 -> 0	On request
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m

For further information, please refer to the existing documentation System description Advant Controller 31.

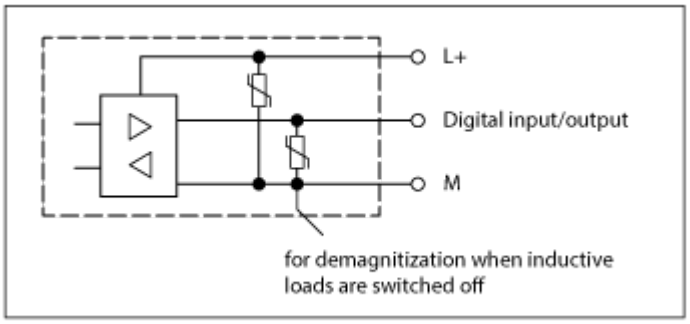


Fig. 807: Protective circuits inputs/outputs

Due to the changed protective circuit on the inputs and outputs, the restrictions concerning the input signal voltage described in the existing documentation no longer apply.

If the channels are to be used as inputs, the respective outputs (high-side switches) must be switched off.

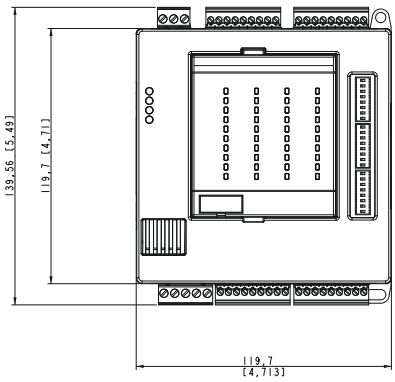
Connection to the CS31 bus

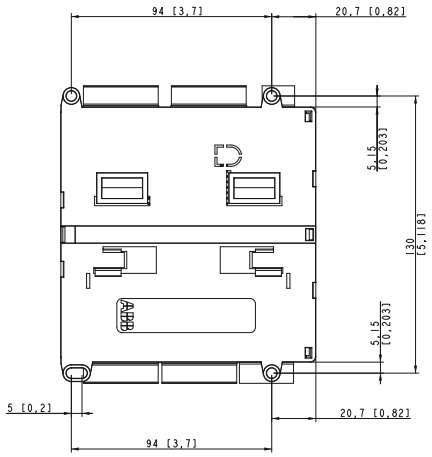
Data	Value
Connections	X1/B2, X1/B1
CS31 bus type	04 (digital input/output)
Terminating resistor	Not available (must be provided externally if needed)

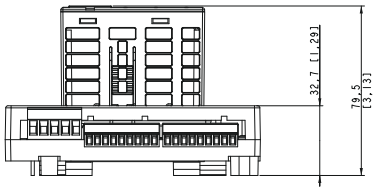
Mechanical data

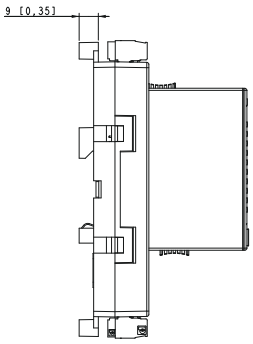
Data	Value
Width x height x depth	Replacement device: 120 x 140 x approx. 80 mm Existing device: 120 x 140 x 85 mm
Weight	Replacement device: 351 g (incl. terminals) Existing device: 450 g
Dimensions for mounting	See operating and assembly instructions of the replacement device (3ADR020151M0401)

Assembly / Disassembly









The dimensions are in mm and in brackets in inch.

The dimensions for the assembly holes are the same for the replacement device and the existing device.

To assemble or disassemble the replacement device, grab the device at the housing and not directly at the S500 module.

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

Ordering data

Order No.	Scope of delivery
1SAP 800 500 R0010	Digital input/output module 07DC92-AD 1 5-pole terminal block (5.08 mm grid space) 1 3-pole terminal block (5.08 mm grid space) 4 10-pole terminal blocks (3.81 mm grid space)

Replacement unit DC501-CS31-AD

Introduction

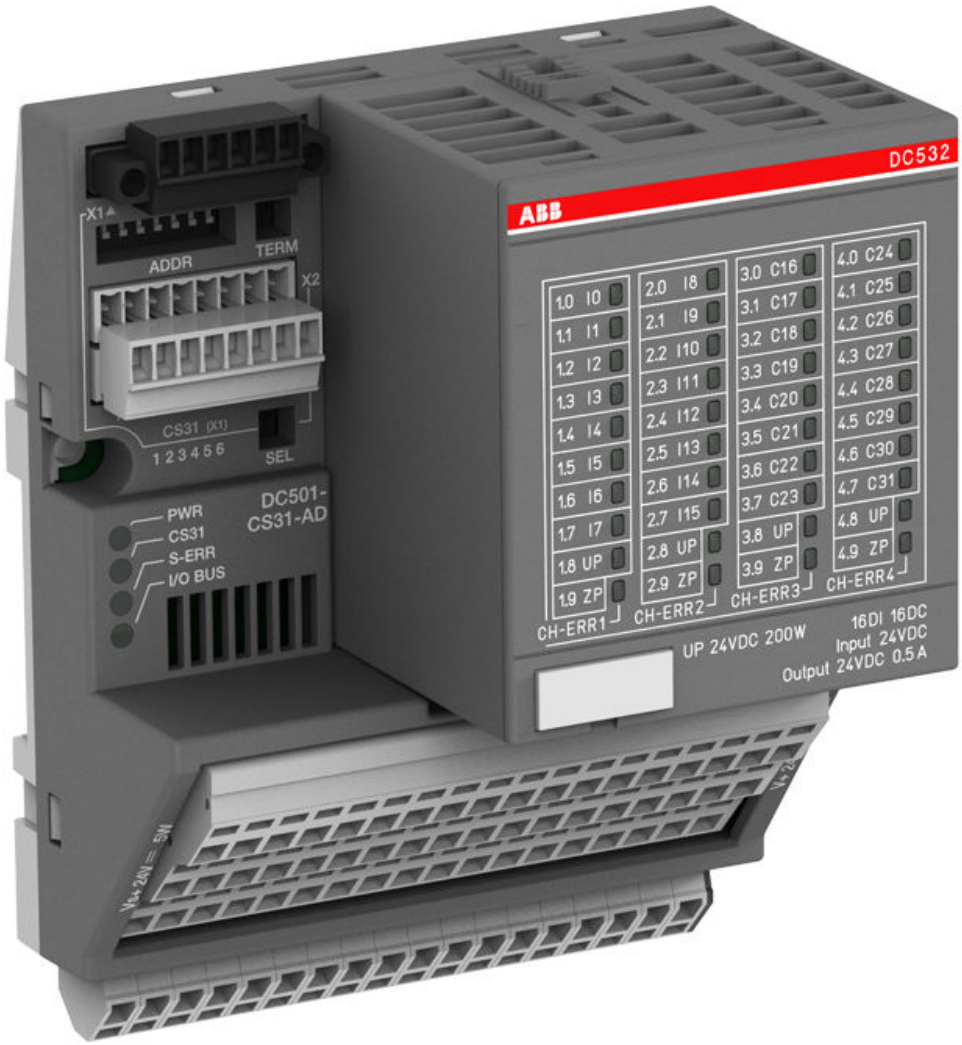


Fig. 808: 3ADR331189S0015_DC501-CS31-AD

The replacement device DC501-CS31-AD of the AC31 adapter series replaces the existing device DC501-CS31.

The existing device DC501-CS31 supported the use of so-called extension box modules to increase I/O functionality. The following modules were supported:

- Module AX501 for analog signals: 3 analog inputs, 1 analog output
- Module DI501 for digital signals: 4 digital inputs
- Module DO501 for relay output: 8 relays

The replacement device DC501-CS31-AD does not support the use of extension box modules. Instead, the functionality of modules AX501 and DI501 is integrated in the replacement device. The functionality of module DO501 is not supported.

This document describes only changes that have been integrated in the replacement device and expansions to the existing device DC501-CS31. Thus, *the existing documentation of device DC501-CS31 remains valid and serves as reference*. The extension box modules are documented in the existing documentation of the I/O-S500 hardware. This description is replaced by this document.

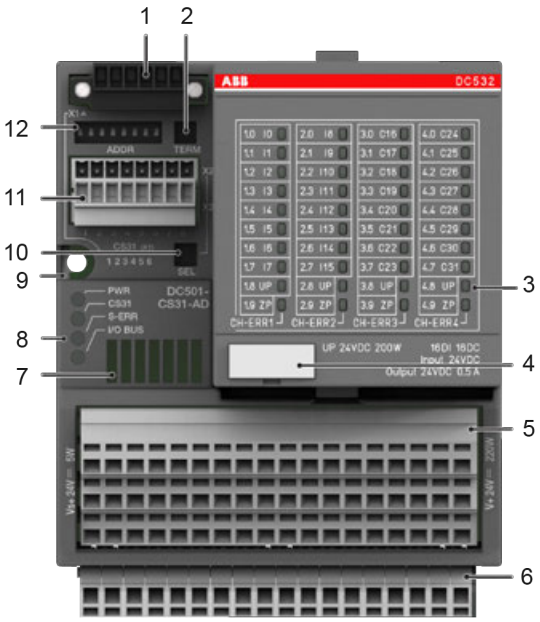
This document adds the following points to the still valid existing documentation:

- Unavoidable device deviations, e.g. due to technical limitations.
- Expansion of documentation as a result of normative requirements.
- Additional contents not described in the existing documentation.

Further information on replacement device DC501-CS31-AD can be found in the operating and assembly instructions of device DC501-CS31-AD: 3ADR020087M0401. Please note that for device DC501-CS31 no separate operating and assembly instructions are available.

Please also observe the system data as well as the information on CS31 bus *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Device configuration



1 Connection for CS31 bus (X1)
2 Bus termination (CS31 bus)
3 Status LEDs for DC532
4 TA525: Label
5 Terminals signal level (X4). 16 digital inputs, 8 digital outputs, 8 DC voltage supply (incl. DC532)
6 Terminals signal level (plug-in power bus)
7 Ventilation
8 4 Status LEDs
9 Hole for screw mounting (screw diameter 4 mm, extension torque 1.2 Nm)
10 Function selector switch for I/O extension
11 4 digital inputs (X2): 24 V DC. 3 analog inputs, 1 analog output (X3): 0 V … +10 V.
12 DIP switch for ADDR (X1)

LED display

The LED display on the replacement device is changed:

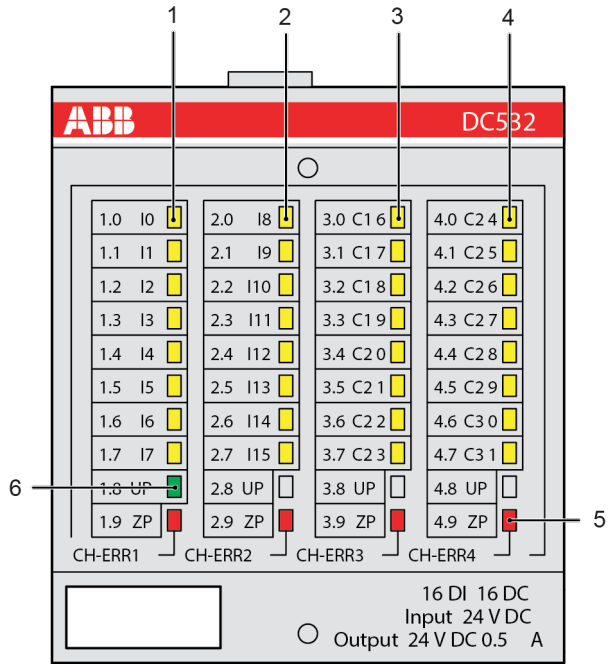


Fig. 809: Front view: DC532

No.	Displays of module
1	8 yellow LEDs to indicate the signal status of the digital inputs (X2).
2	8 yellow LEDs to indicate the signal status of the digital inputs (X3).
3	8 yellow LEDs to indicate the signal status of the digital outputs (X5).
4	8 yellow LEDs to indicate the signal status of the digital inputs/outputs (X6).
5	4 red LEDs to indicate errors (of DC532 module).
6	1 green LED to indicated the status of the supply voltage of the DC532 module (is supplied via X4).

Connections

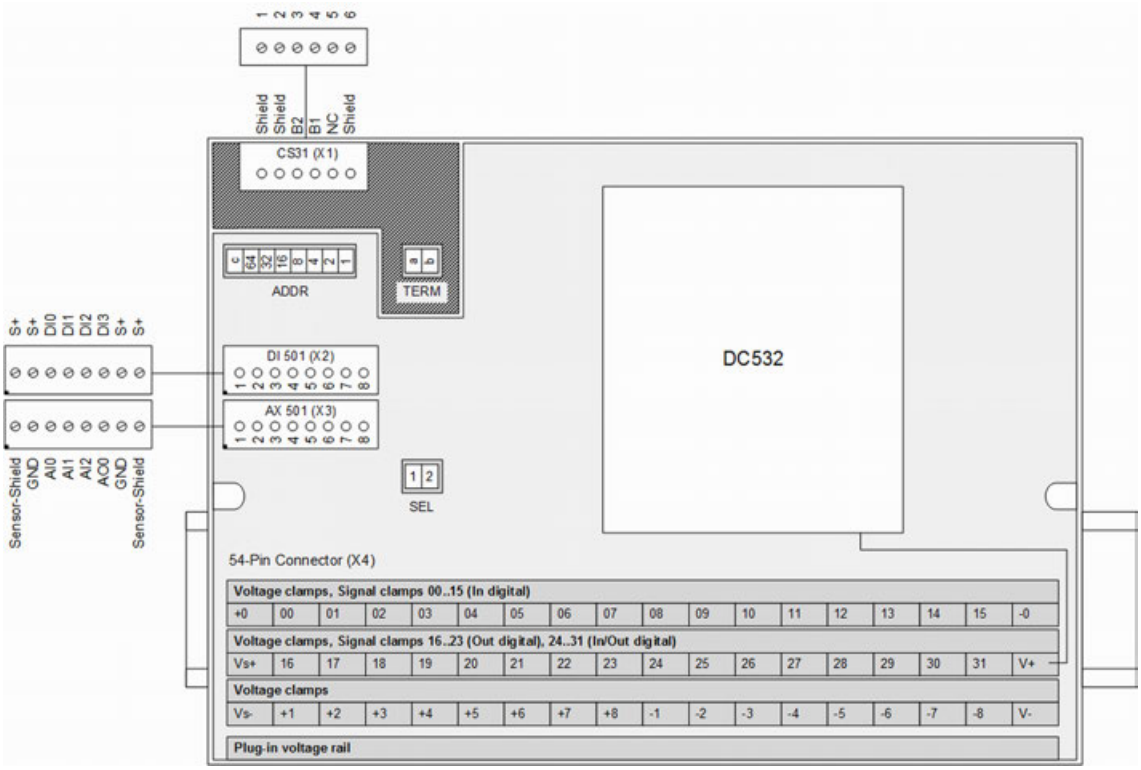


Fig. 810: Connection

Table 361: Pin assignment CS31 bus (X1)

Connector / Terminal	Pin	Assignment / Signal
X1 / Shield	1	Shield (internally connected to pins 2 and 6. No internal connection to functional earth)
X1 / Shield	2	Shield (internally connected to pins 1 and 6. No internal connection to functional earth)
X1 / B2	3	BUS 2
X1 / B1	4	BUS 1
X1 / NC	5	Not connected
X1 / Shield	6	Shield (internally connected to pins 1 and 2. No internal connection to functional earth)

Correction to existing documentation

In the existing documentation, connection X1 / 2 is incorrectly documented as "free / not connected". On the replacement device DC501-CS31-AD, the selection of the pin assignment of connector X1 is identical to the realization of device DC501-CS31. Thus, the pin assignment described in this document is valid for the replacement device and the existing device.

Table 362: Pin assignment DI501 (X2)

Connector / Terminal	Pin	Assignment / Signal
X2 / S+	1	Auxiliary voltage (max. 32 mA total load of S+ permitted) for DI0 - DI3. Voltage derived from input voltage Vs+ (X4)
X2 / S+	2	Auxiliary voltage (max. 32 mA total load of S+ permitted) for DI0 - DI3. Voltage derived from input voltage Vs+ (X4)

Connector / Terminal	Pin	Assignment / Signal
X2 / DI0	3	Digital extension input 0
X2 / DI1	4	Digital extension input 1
X2 / DI2	5	Digital extension input 2
X2 / DI3	6	Digital extension input 3
X2 / S+	7	Auxiliary voltage (max. 32 mA total load of S+ permitted) for DI0 - DI3. Voltage derived from input voltage Vs+ (X4)
X2 / S+	8	Auxiliary voltage (max. 32 mA total load of S+ permitted) for DI0 - DI3. Voltage derived from input voltage Vs+ (X4)

Table 363: Pin assignment AX501 (X3)

Connector / Terminal	Pin	Assignment / Signal
X3 / Sensor shield	1	Sensor shield
X3 / GND	2	GND
X3 / AI0	3	Analog extension input 0
X3 / AI1	4	Analog extension input 1
X3 / AI2	5	Analog extension input 2
X3 / AO0	6	Analog extension output 0
X3 / GND	7	GND
X3 / Sensor shield	8	Sensor shield

The connections X3 / 2 and X3 / 7 (GND) are directly connected to X4 / Vs-, X4 / V-. There is no AGND potential in accordance with module AX501. In module AX501, AGND is connected to GND via a resistor.

Both sensor shield connections of X3 are interconnected and jointly connected to FE via 10 MΩ || 4 nF.

The connections X3 / 2 and X3 / 7 (GND) are directly connected to X4 / Vs-, X4 / V-. There is no AGND potential in accordance with module AX501. In module AX501, AGND is connected to GND via a resistor.

Both sensor shield connections of X3 are interconnected and jointly connected to FE via 10 MΩ || 4 nF.

The terminal blocks of X2 and X3 have the following connection data:

- Conductor cross section, single wire/ flexible: 0.14 mm² to 1.5 mm²
- Conductor cross section, flexible with wire-end ferrule (without plastic ferrule): 0.25 mm² to 1.5 mm²
- Conductor cross section, flexible with wire-end ferrule (with plastic ferrule): 0.25 mm² to 0.5 mm²

Table 364: Pin assignment 54 pin connector (X4)

Connector / Block	Pin	Assignment / Signal
X4 / 1	+0	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 1	00	DC532 / I0
X4 / 1	01	DC532 / I1

Connector / Block	Pin	Assignment / Signal
X4 / 1	02	DC532 / I2
X4 / 1	03	DC532 / I3
X4 / 1	04	DC532 / I4
X4 / 1	05	DC532 / I5
X4 / 1	06	DC532 / I6
X4 / 1	07	DC532 / I7
X4 / 1	08	DC532 / I8
X4 / 1	09	DC532 / I9
X4 / 1	10	DC532 / I10
X4 / 1	11	DC532 / I11
X4 / 1	12	DC532 / I12
X4 / 1	13	DC532 / I13
X4 / 1	14	DC532 / I14
X4 / 1	15	DC532 / I15
X4 / 1	-0	GND
X4 / 2	Vs+	Voltage supply for electronics system (also for functionality of AX501 and DI501)
X4 / 2	16	DC532 / C16
X4 / 2	17	DC532 / C17
X4 / 2	18	DC532 / C18
X4 / 2	19	DC532 / C19
X4 / 2	20	DC532 / C20
X4 / 2	21	DC532 / C21
X4 / 2	22	DC532 / C22
X4 / 2	23	DC532 / C23
X4 / 2	24	DC532 / C24
X4 / 2	25	DC532 / C25
X4 / 2	26	DC532 / C26
X4 / 2	27	DC532 / C27
X4 / 2	28	DC532 / C28
X4 / 2	29	DC532 / C29
X4 / 2	30	DC532 / C30
X4 / 2	31	DC532 / C31
X4 / 2	V+	Voltage supply of inputs/outputs (module DC532 and auxiliary voltage)
X4 / 3	Vs-	GND
X4 / 3	+1	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+2	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)

Connector / Block	Pin	Assignment / Signal
X4 / 3	+3	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+4	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+5	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+6	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+7	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	+8	Auxiliary voltage (max. 200 mA total load of +0/ +1/ …/ +7/ +8 permitted). Voltage derived from input voltage V+ (X4)
X4 / 3	-1	GND
X4 / 3	-2	GND
X4 / 3	-3	GND
X4 / 3	-4	GND
X4 / 3	-5	GND
X4 / 3	-6	GND
X4 / 3	-7	GND
X4 / 3	-8	GND
X4 / 3	V-	GND

Connection data of spring terminals (X4):

- Conductor cross section, single wire: 0.2 mm² to 2.5 mm²
- Conductor cross section, flexible: 0.2 mm² to 1.5 mm² (existing device: 2.5 mm² flexible)
- Conductor cross section, flexible with wire-end ferrule: 0.25 mm² to 1.5 mm²

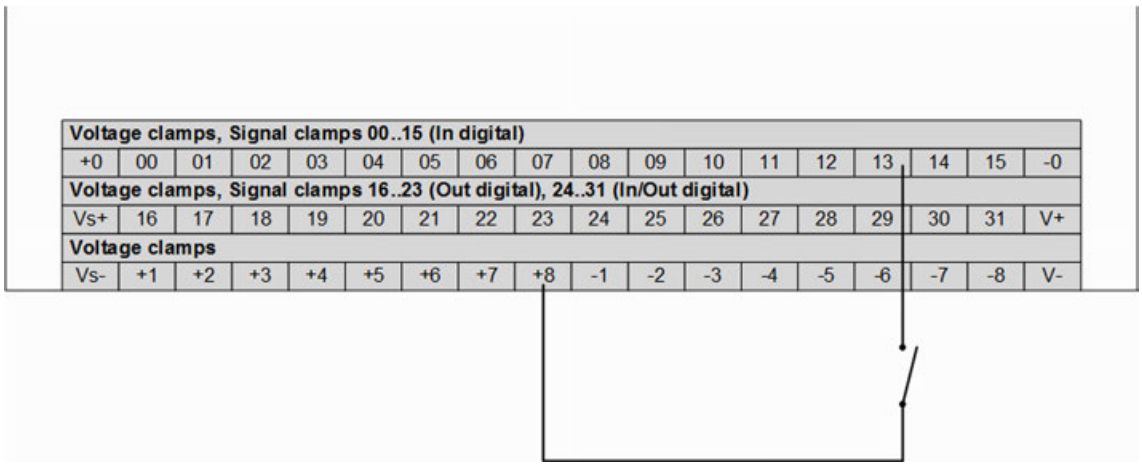


Fig. 811: Connection example: digital input (X4)

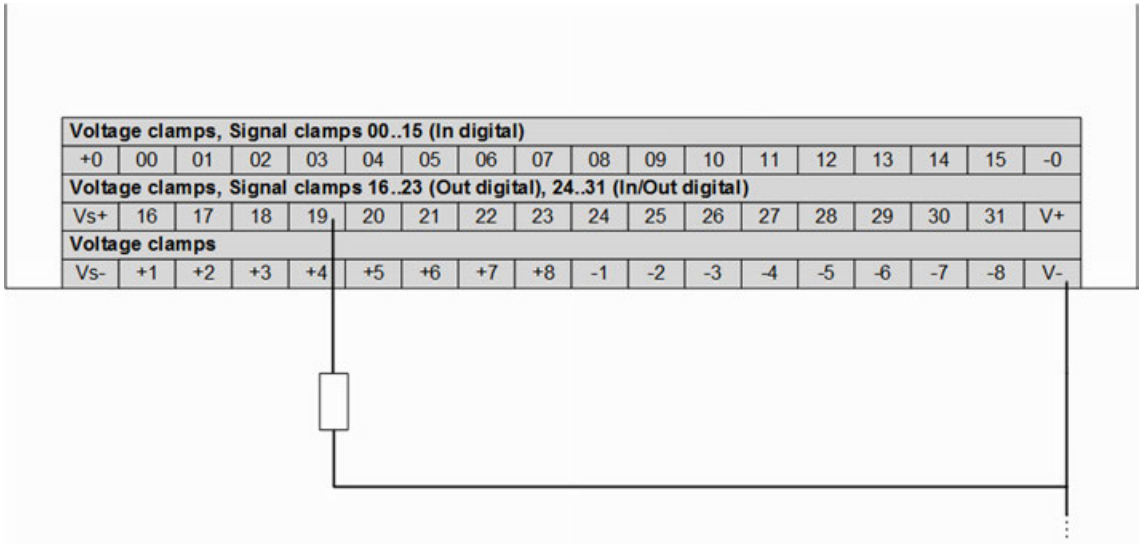


Fig. 812: Connection example: digital output

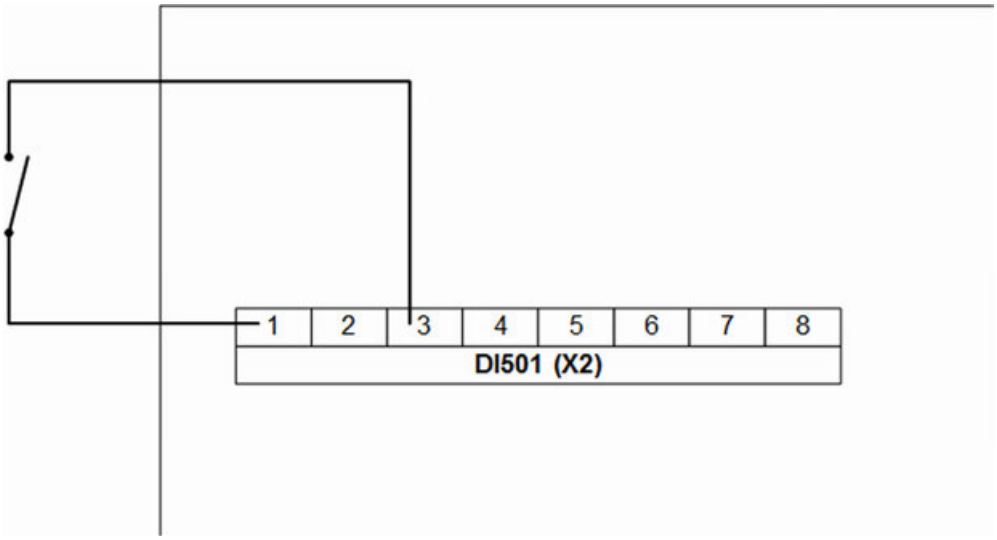


Fig. 813: Connection example: digital input (X2)

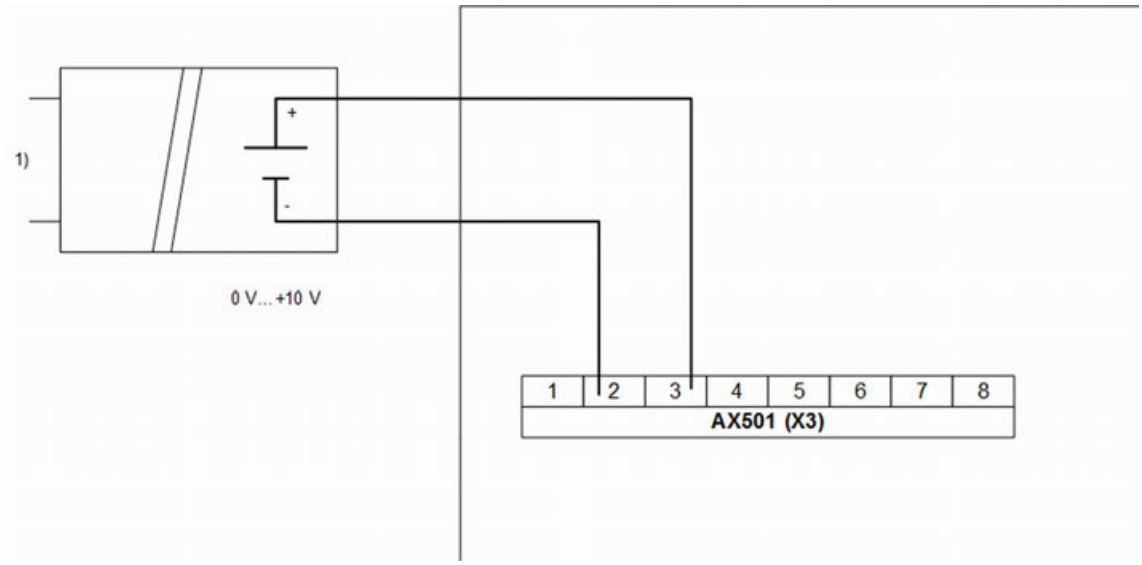


Fig. 814: Connection example: Voltage input

1) Galvanically isolated power supply of analog sensor.

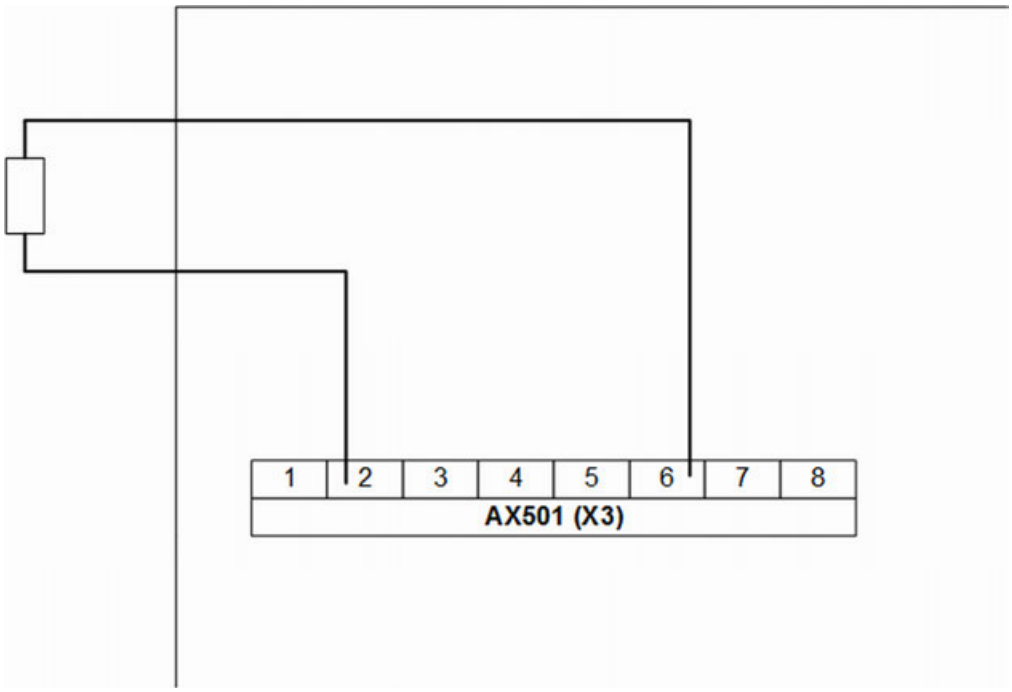


Fig. 815: Connection example: Voltage output

Analog signal lines must be routed in shielded cables. The shield must be grounded on both sides and should be grounded to replacement device and signal source / signal sink as close as possible.

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

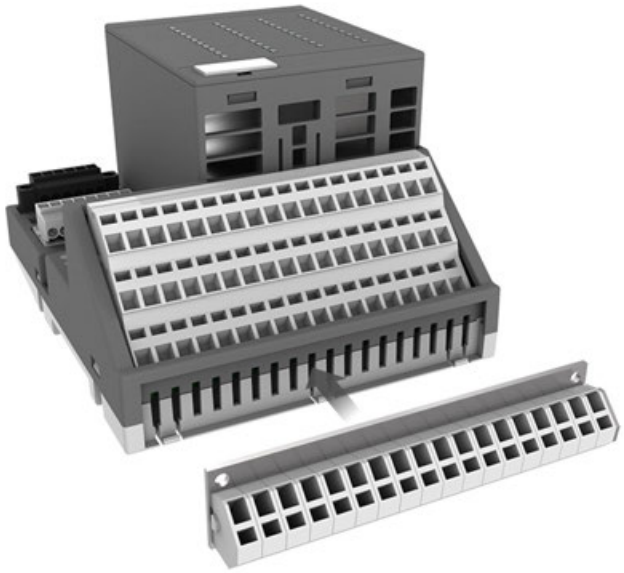


Fig. 816: Plug-in power bus

A power bus can be plugged into the replacement device. The contacts of the power bus have no electrical connection to the electronic system of the replacement device. Furthermore, no FE connection is available.

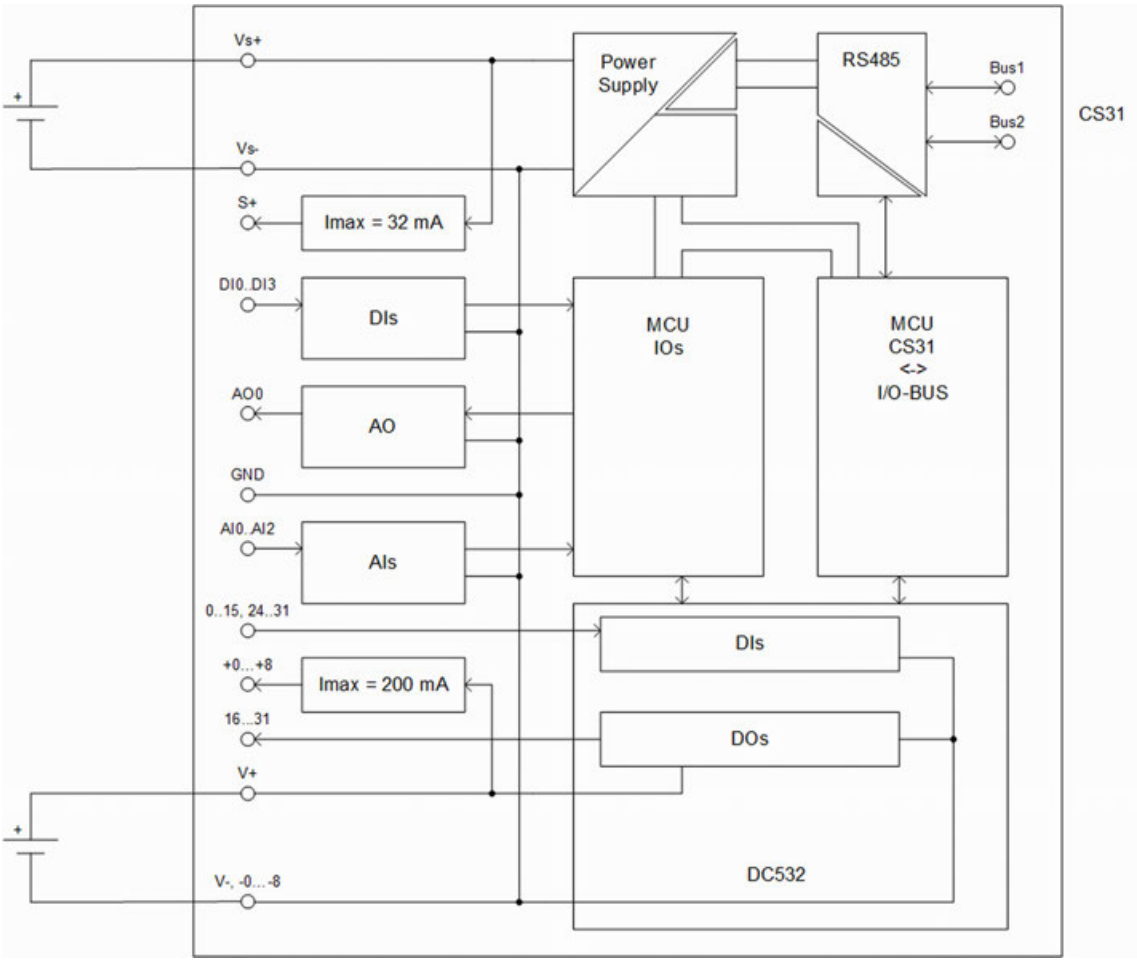


Fig. 817: Schematic diagram

For further information on grounding of the individual connections as well as shielding, please refer to *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Addressing

In the existing device, the address DIP switch was arranged on the top right of the device. In the replacement device, this DIP switch is located on the left side of the device.

An additional DIP switch (SEL) has been implemented for the selection of the extension (AX501, 3AI1AO or DI501/4DI). Please note that only one extension at a time can be used.

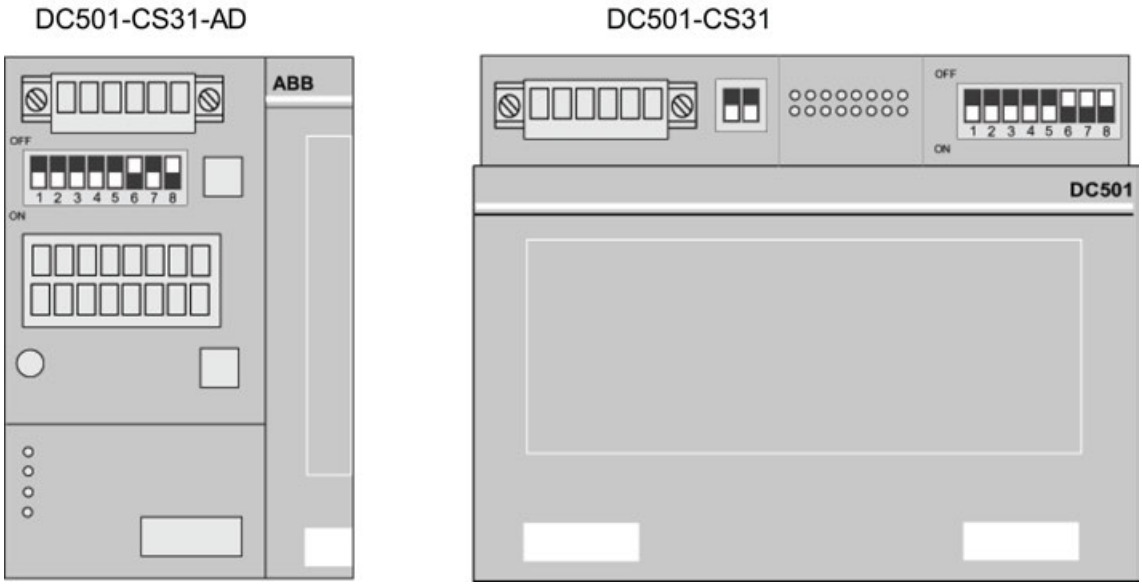


Fig. 818: DIP switch for DC501-CS31-AD

> *The function of the address DIP switch 1 (channel switch) available in the existing device is no longer supported. This DIP switch must be switched off.*

Table 365: Extension DIP switch (SEL)

S1	S2	Description
OFF	OFF	Normal, without extension
OFF	ON	Normal, with 3AI1AO/ AX501 extension
ON	OFF	Normal, with 4DI/ DI501 extension
ON	ON	Version DC501R0100, without extension

The device version DC501R0100 differs only in the data format of the CS31 bus. Further information, ↳ *Chapter 1.6.2.3.3.3.2 “CS31 bus system data” on page 3882*.

> *The DIP switches are read by the device only once after the supply voltage has been connected.*

> *For further information, please refer to the existing documentation System description Advant Controller 31.*

In the following, the information in the "Type" column refers to the data type designation of the Automation Builder (see AC31 system data ↳ *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*). The information in the "Type" column must be interpreted from the viewpoint of the CS31 bus master. The information in brackets must be interpreted from the viewpoint of the replacement device (CS31 bus slave).

Table 366: CS31 bus: 16 DI and 16 DO, normal and version DC501R0100

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
4	8 bit output (receive)	0 ... 7	X4 / 24 ... 31

Table 367: CS31 bus: 24 DI and 16 DO, normal

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit input (send)	0 ... 7	X4 / 24 ... 31
4	8 bit input (send, filling byte)	0 ... 7	-
5	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
6	8 bit output (receive)	0 ... 7	X4 / 24 ... 31

Table 368: CS31 bus: 24 DI and 16 DO, version DC501R0100

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
4	8 bit input (send)	0 ... 7	X4 / 24 ... 31
5	8 bit output (receive)	0 ... 7	X4 / 24 ... 31

Table 369: CS31 bus: 16 DI, 16 DO, 3AI1AO, normal

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit input (send)	0 ... 7	X3 / 3
4	8 bit input (send)	0 ... 7	X3 / 4
5	8 bit input (send)	0 ... 7	X3 / 5
6	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
7	8 bit output (receive)	0 ... 7	X4 / 24 ... 31
8	8 bit output (receive)	0 ... 7	X3 / 6

Table 370: CS31 bus: 16 DI, 16 DO, 4 DI, normal

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit input (send)	0 ... 3	X2 / 3 ... 6
	8 bit input (send)	4 .. .7	-

Byte	Type	Bit	Connector / Terminal
4	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
5	8 bit output (receive)	0 ... 7	X4 / 24 ... 31

Table 371: CS31 bus: 24 DI, 16 DO, 3AI1AO, normal

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit input (send)	0 ... 7	X4 / 24 ... 31
4	8 bit input (send, filling byte)	0 ... 7	-
5	8 bit input (send)	0 ... 7	X3 / 3
6	8 bit input (send)	0 ... 7	X3 / 4
7	8 bit input (send)	0 ... 7	X3 / 5
8	8 bit input (send, filling byte)	0 ... 7	-
9	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
10	8 bit output (receive)	0 ... 7	X4 / 24 ... 31
11	8 bit output (receive)	0 ... 7	X3 / 6
12	8 bit output (receive, filling byte)	0 ... 7	-

Table 372: CS31 bus: 24 DI, 16 DO, 4 DI, normal

Byte	Type	Bit	Connector / Terminal
1	8 bit input (send)	0 ... 7	X4 / 00 ... 07
2	8 bit input (send)	0 ... 7	X4 / 08 ... 15
3	8 bit input (send)	0 ... 7	X4 / 24 ... 31
4	8 bit input (send, filling byte)	0...7	-
5	8 bit input (send)	0 ... 3	X2 / 3 ... 6
		4 ... 7	-
6	8 bit input (send, filling byte)	0 ... 7	-
7	8 bit output (receive)	0 ... 7	X4 / 16 ... 23
8	8 bit output (receive)	0 ... 7	X4 / 24 ... 31

Table 373: CS31 bus: analog values

Nominal range 0…+10 V	Digital value (decimal)	Digital value (hexadecimal)
9.961 V	255	FF
9.922 V	254	FE
…	…	…
0.039 V	1	01
0.000 V	0	00

Relationship between analog voltage and digital representation (applies to analog inputs and analog output):

$$U_{Signal} = U_{Ref} \cdot \frac{Digital\ value_{8\ Bit}}{256}$$
$$U_{Ref} = 10\ V$$

Fig. 819: Formula: Voltage

Documentation change
The replacement device does not have an I/O bus. Communication interface module cannot be connected. For this reason, chapter "1.1.3 Addressing" of the technical description of DC501-CS31 concerning the expansion modules (e.g. DX511, DI511, DO511, AX511, AI511, AI512) is not valid for the replacement device. Possible data structures for the replacement device are indicated in the following table.

Behavior during normal operation

Interpretation of the LEDs:

- The device initializes automatically after the supply voltage is switched on. During this time, the S-ERR LED flashes.
- The PWR LED lights up as soon as the internal supply voltage of the device is present.
- After successful initialization of the I/O bus communication to the S500 module, the I/O bus LED lights up.
- After successful initialization of the CS31 bus communication, the CS31 bus LED lights up. The S-ERR LED goes out.
- During operation, the yellow LEDs indicate the signal statuses of the channels.

The RAM is checked during the initialization of the device. In addition, the firmware in the Flash memory is checked by means of a checksum during initialization. When the control system (PLC/central unit) is stopped during normal operation, the outputs of the device are switched off. The inputs remain active. The outputs are also switched off in case of a malfunction of the CS31 bus.

Diagnosis and display

The replacement device transmits diagnosis information also via the CS31 bus.

Table 374: Diagnosis information CS31 bus

Error description	Channel	Error code (CODESYS)	Error code (CS31 bus)	Description
Device error	0	43	1	Internal error
Device error	1	45	2	No supply voltage V+ available
Channel error	0…15	46	4	Overload or short circuit on a digital output

The error codes that are transferred by the replacement device via the CS31 bus bus are newly displayed in CODESYS. Each error code of the CS31 bus (table column 3) produces the error code in CODESYS (table column 2). As a result, it is possible to operate the replacement device with a new control system (PLC/control unit), e.g. 07KT98-ARC-AD, as well as with an old control system (PLC/central unit), e.g. 07KT98.

Since in the replacement device the functionality of the extension box is integrated in the hardware, error code 6 (failure of extension box) does not occur.

The input/output functions of the extensions (AX501/ 3AI1AO, DI501/ 4DI) have no diagnoses.

Table 375: Device LEDs

LED	Status	Color	LED off	LED on	LED flashes
PWR	Voltage supply	Green	No internal supply voltage	Internal supply voltage	-
CS31 bus	CS31 bus communication	Green	No CS31 bus communication	CS31 bus bus communication	Only diagnosis, no data transfer. Transmission is disturbed.
S-ERR	Error	Red	No error	Static error (must be confirmed by the control system)	No CS31 bus connection or activity
I/O bus	I/O bus communication	Green	No I/O bus communication	I/O bus communication	Error I/O bus communication

The S-ERR LED remains on even if the error no longer occurs. The error must be confirmed by the control system (PLC/central unit), e.g. by means of a function block *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876*.

Special cases with rapidly flashing LEDs (approx. 5 Hz):

- All 4 LEDs flash rapidly: An incorrect S500 module is connected to the device. The device fails to initialize.
- The LEDs of the CS31 bus, S-ERR bus and I/O bus flash rapidly: Invalid position of DIP switches. The device fails to initialize.
- The LEDs of the S-ERR bus and I/O bus flash rapidly: A checksum error occurred in an internal Flash memory.
- The LED of the I/O bus flashes rapidly: An error occurred in an internal RAM.

Table 376: S500 module DC532 LEDs

LED	Status	Color	LED off	LED on	LED flashes
I0…I7 (see No. 1 in the following figure)	Digital inputs	Yellow	Input is not activated	Input is activated (input voltage is indicated even if supply is switched off)	-
I8…I15 (see No. 2 in the following figure)	Digital inputs	Yellow	Input is not activated	Input is activated (input voltage is indicated even if supply is switched off)	-
C16…C23 (see No. 3 in the following figure)	Digital outputs	Yellow	Output is not activated	Output is activated	-

LED	Status	Color	LED off	LED on	LED flashes
C24…C31 (see No. 4 in the following figure)	Digital inputs or digital outputs	Yellow	Input or output is not activated	Input/output is activated (input voltage is indicated even if supply is switched off)	-
Indication supply voltage (see No. 6 in the following figure)	Process voltage	Green	Process voltage not available	Process voltage OK	-
Error indications left (see No. 5 in the following figure)	Error indication	Red	No error	Internal error	
Error indications right (see No. 5 in the following figure)	Error indication	Red	No error	Internal error	Overload or short circuit on a channel of the corresponding group

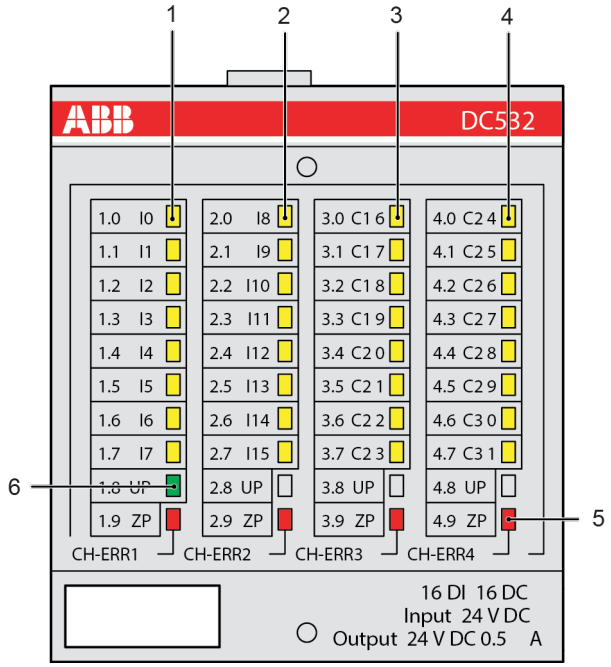


Fig. 820: Front view: DC532

Technical data

This section expands the details provided in the chapter *Chapter 1.6.2.3.3.3 “System data and CS31 bus system data” on page 3876* and contains information on electromagnetic compatibility. The conformity is described in the declaration of conformity, which is available on the ABB website.

To ensure proper function of the replacement device DC501-CS31-AD, both supply voltages Vs+ and V+ must be applied.

Technical data of the complete device

Table 377: Supply voltage Vs

Data	Value
Process voltage: Fuse for Vs+	10 A, fast acting
Current consumption:	
-> via Vs+	Replacement device: 0.15 A Existing device DC501-CS31: approx. 60 ... 230 mA
- Inrush current via Vs+ (when voltage is switched on)	0.06 A²s
Power consumption	5 W

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Table 378: Supply voltage V

Data	Value
Process voltage:	
-> Fuse for V+	10 A, fast acting
-> Additional V-/Vs- connections (GND)	X4 / -0, X4 / -1, X4 / -2, X4 / -3, X4 / -4, X4 / -5, X4 / -6, X4 / -7, X4 / -8
Current consumption:	
-> via V+	Replacement device: 0.15 A incl. load current Existing device DC501-CS31: approx. 100 mA without load current
- Inrush current via V+ (when voltage is switched on)	0.013 A^2s
Power consumption	220 W
Power consumption outputs unloaded	6 W
Sensor supply voltage connections	X4 / +0, X4 / +1, X4 / +2, X4 / +3, X4 / +4, X4 / +5, X4 / +6, X4 / +7, X4 / +8
Current sensor supply voltage (all connections combined)	Replacement device: max. 200 mA Existing device DC501-CS31: Microfuse 8 A, fast acting *)

*) The existing device contained an 8 A fuse to be exchanged by the user. The replacement device has an integrated electronic current limiter instead.

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!
System damage caused by voltage!
Exceeding the maximum supply or process voltage (>30 V DC) results in permanent system damage (destruction).

Connection to the CS31 bus

Data	Value
Connections	X1 / 3, X1 / 4
CS31 bus type	04 (digital input/output)

Expansion interface

The replacement device does not have an expansion interface.

Interface extension box

Table 379: Analog inputs

Data	Value
Number of channels	3
Connections	X3 / 3, X3 / 4, X3 / 5
Reference connections (GND)	X3 / 2, X3 / 7
Type of inputs	Voltage unipolar
Galvanic isolation	Not available
Nominal range	0…10 V
Input resistance per channel	Replacement device: > 100 kΩ Existing device AX501: 95 kΩ
Time constant of the input filter	Replacement device: approx. 8 ms Existing device AX501: approx. 7 ms
Total errors (due to non-linearity, offset, resolution and temperature)	Replacement device: max. 3 % Existing device AX501: 0.6 % ± 1 digit ± 150 ppm/K
Indication of the input signals	Replacement device: not available Existing device AX501: green LED per channel
Conversion cycle *)	Replacement device: 1.5 ms for all three channels Existing device AX501: 1.64 ms for all three channels
Conversion process	Successive approximation

Data	Value
Averaging of measured values	not available
Resolution	8 bit
Unused voltage inputs	Can remain open or be short-circuited after GND or V+ to increase noise immunity
Overvoltage protection	Available
Overload range	± 30 V DC
Max. line length of analog lines, line cross section > 0.14 mm²	100 m

*) Conversion cycle of MCU of I/O processing. The transmission via serial buses is slower.

For further information, please refer to the existing documentation System description Advant Controller 31.

Table 380: Analog output

Data	Value
Number of channels	1
Connections	X3 / 6
Reference connections (GND)	X3 / 2, X3 / 7
Type of outputs	Voltage unipolar
Galvanic isolation	not available
Nominal range	0 ... 10 V
Output load capability	max. ± 5 mA
Indication of the output signals	Replacement device: Not available Existing device AX501: green LED per channel
Resolution	8 bit
Total errors (due to non-linearity, offset, resolution and temperature)	Replacement device: max. 3 % Existing device AX501: 0.6 % ± 1 digit ± 150 ppm/K
Update cycle	1.5 ms
Unused output	remains unconnected
Short-circuit-proof	Yes *)
External supply protection	Up to +30 V DC (no external supply protection available for negative voltages!)
Max. line length of analog lines, line cross section > 0.14 mm²	100 m

For further information, please refer to the existing documentation System description Advant Controller 31.

CAUTION!

System damage caused by short-circuit!

*) A short-circuit can result in up to 2 W additional power dissipation in the device. If this power dissipation cannot be discharged, the replacement device can be damaged.

Table 381: Digital inputs

Data	Value
Number of channels	4
Connections	X2 / 3, X2 / 4, X2 / 5, X2 / 6
Reference connection (GND)	X4 / Vs-
Connections switch supply	X2 / 1, X2 / 2, X2 / 7, X2 / 8
Current switch supply (all connections combined)	Replacement device: max. 32 mA Existing device DI501: max. 30 mA
Input type according to EN 61131-2	Type 1
Galvanic isolation	Not available
Indication of the input signals	Replacement device: Not available Existing device DI501: green LED per channel
Input delay (0->1 or 1->0)	Typ. 3 ms
Scanning cycle	500 µs
Input signal voltage:	
-	24 V DC
-> 0 signal	Replacement device: -3 V … +5 V Existing device DI501: -30 V … +5 V
-> Undefined signal	Replacement device: > +5 V … < +15 V Existing device DI501: > +5 V … < +13 V
-> 1 signal	Replacement device: +15 V … +30 V Existing device DI501: +13 V … +30 V
-> Residual ripple at 0 signal	Within -3 V … +5 V
-> Residual ripple at 1 signal	Within +15 V … +30 V
Input current per channel:	
-> Input voltage +24 V	Typ. 5.5 mA
-> Input voltage +5 V	≥ 0.5 mA
-> Input voltage +15 V	≥ 2 mA
-> Input voltage +30 V	≤ 8 mA
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m
Overvoltage protection	Available
Overload range	±30 V DC

For further information, please refer to the existing documentation System description Advant Controller 31.

Inputs 24 V DC

Data	Value
Connections	X4 / 00, X4 / 01, X4 / 02, X4 / 03, X4 / 04, X4 / 05, X4 / 06, X4 / 07, X4 / 08, X4 / 09, X4 / 10, X4 / 11, X4 / 12, X4 / 13, X4 / 14, X4 / 15, X4 / 24, X4 / 25, X4 / 26, X4 / 27, X4 / 28, X4 / 29, X4 / 30, X4 / 31
Input type according to EN 61131-2	Type 1
Galvanic isolation	Not available
Status display	Replacement device: 1 yellow LED per input Existing device DC501-CS31: 1 green LED per input
Input delay (0-> 1 or 1-> 0)	Replacement device: Typ. 8 ms Existing device DC501-CS31: Typ. 3 ms
Input signal voltage:	
-	24 V DC
-> 0 signal	Replacement device: -3 V … +5 V Existing device DC501-CS31: -30 V … +5 V
-> Undefined signal	Replacement device: > +5 V … < +15 V Existing device DC501-CS31: > +5 V … < +13 V
-> 1 signal	Replacement device: +15 V … +30 V Existing device DC501-CS31: +13 V … +30 V
-> Residual ripple at 0 signal	Within -3 V … +5 V
-> Residual ripple at 1 signal	Within +15 V … +30 V
Input current per channel:	
-> Input voltage +24 V	Replacement device: typ. 5 mA Existing device DC501-CS31: typ. 4 mA
-> Input voltage +5 V	> 1 mA
-> Input voltage +15 V	> 5 mA
-> Input voltage +30 V	< 8 mA
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m
Overvoltage protection	Up to 30 V DC
Marking	Replacement device: not possible Existing device DC501-CS31: with label strip possible

For further information, please refer to the existing documentation System description Advant Controller 31.

Outputs 24 V DC

Data	Value
Connections	X4 / 16, X4 / 17, X4 / 18, X4 / 19, X4 / 20, X4 / 21, X4 / 22, X4 / 23, X4 / 24, X4 / 25, X4 / 26, X4 / 27, X4 / 28, X4 / 29, X4 / 30, X4 / 31
Type of digital outputs	High-side switches
Demagnetization with inductive load	Via internal varistor (see following figure)
Status display	Replacement device: 1 yellow LED per output Existing device DC501-CS31: 1 green LED per output
Output delay (0-> 1 or 1-> 0)	On request
Switching frequency:	
-> With ohmic load	Replacement device: on request Existing device DC501-CS31: ≤ 100 Hz
-> With inductive load	Replacement device: max. 0.5 Hz Existing device DC501-CS31: ≤ 2 Hz
-> With lamp load	Replacement device: max. 11 Hz at max. 5 W Existing device DC501-CS31: ≤ 10 Hz at max. 5 W
Inductive cut-off voltage	Replacement device: Typ. -67 V Existing device DC501-CS31: Typ. (voltage V) -55 V
Maximum cable length:	
-> Shielded	1000 m
-> Unshielded	600 m
Marking	Replacement device: not possible Existing device DC501-CS31: with label strip possible

For further information, please refer to the existing documentation System description Advant Controller 31.

The following figure shows the circuitry of a digital input/output with the varistors for demagnetization when switching off inductive loads.

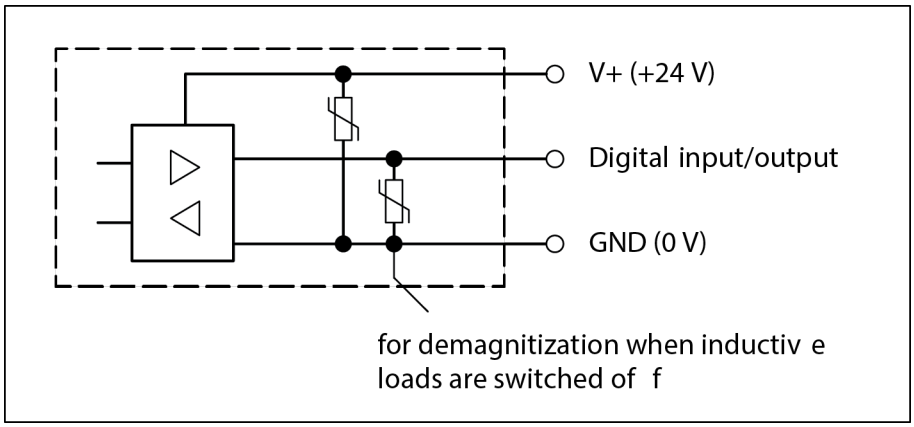


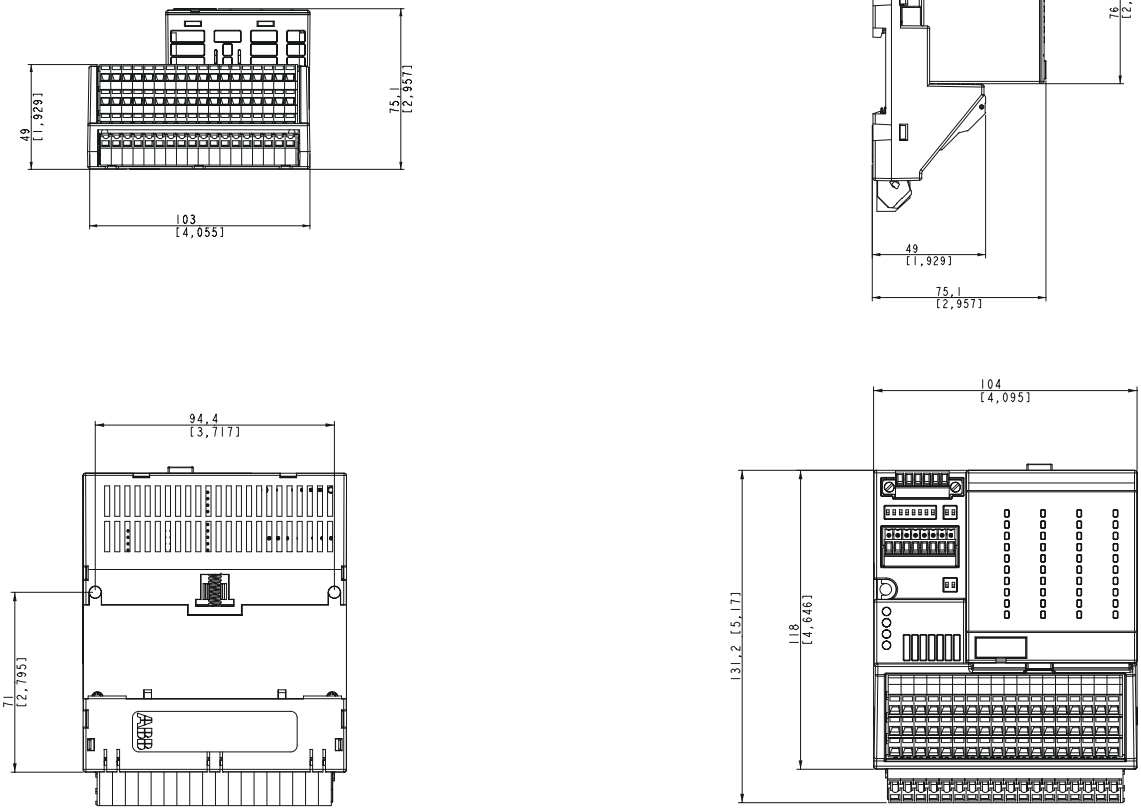
Fig. 821: Protective circuits inputs/outputs

When the channels of X4 / 24 to X4 / 31 are to be used as inputs, the respective outputs (high-end switches) must be switched off.

Mechanical data

Data	Value
Width x height x depth	Replacement device: 104 x 118 x 75.1 mm Existing device DC501-CS31: 102 x 112 x 77 mm
Weight	Replacement device: 354 g Existing device DC501-CS31: approx. 150 g
Dimensions for mounting	See operating and assembly instructions of the replacement device: 3ADR020087M0401

Mounting information



The dimensions are in mm and in brackets in inch.

CAUTION!

System damage caused by voltage!

The exchange of a replacement device under voltage can cause permanent system damage (destruction).

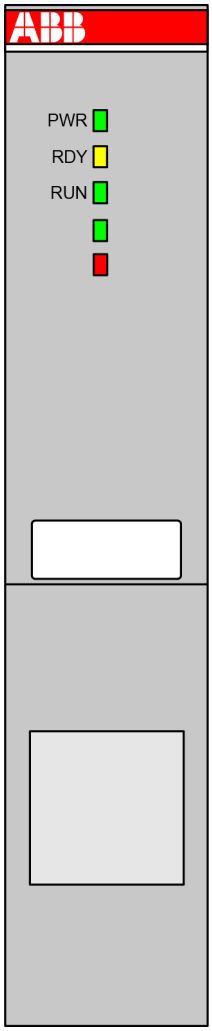
Data	Value
Mounting position	Vertical, terminal block facing downward
Cooling	The natural convection cooling must not be hindered by cable ducts or other switchgear cabinet equipment (clearance between cable duct and device at least 20 mm).

Ordering data

Order No.	Scope of delivery
1SAP 800 400 R0010	Communication interface module CS31 16 DI, 8 DC, 8 DO, DC501-CS31-AD 1 6-pole terminal block 2 8-pole terminal blocks

1.6.2.4 Communication modules (AC500 standard)

1.6.2.4.1 Overview



AC500 communication modules are required for

- a connection to standard field bus systems and
- for integration into existing networks.

AC500 communication modules

- enable communication on different field buses.
- are mounted on the left side of the processor module on the same terminal base.
- are directly powered via the internal communication module bus of the terminal base. A separate voltage source is not required.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

For information on mounting and demounting, please refer to the chapter mounting and demounting the communication modules ↳ Chapter 1.6.3.6.3.6 “Mounting/Demounting the communication modules” on page 5335.

The communication between the processor module and the communication modules takes place via the communication module bus, which is integrated in the terminal base. Depending on the used terminal base up to 6 communication modules can be connected.

- ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*

There are no restrictions concerning which communication modules can be arranged for a processor module.

Within the AC500 control system, the communication modules can be used as

- bus master or
- slave.

It depends on the

- selected protocol,
- the functionality of the communication module and
- the several field buses and networks.

The following name extensions of the device names describe the supported field bus/protocol:

- CMxyz-ETH: Ethernet
- CMxyz-DP: PROFIBUS
- CMxyz-PNIO: PROFINET
- CMxyz-ETHCAT: EtherCAT
- CMxyz-CN: CANopen
- CMxyz-RCOM: RCOM/RCOM+ protocol (and 2 serial interfaces)
- CMxyz-RS: 2 serial interfaces (COM1/COM2)

If a XC version of the device is available, for use in extreme ambient conditions (e.g. wider temperature and humidity range), this is indicated with a snowflake sign.

Compatibility of communication modules and communication interface modules

Table 382: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O
CM597-ETH	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 383: PROFIBUS DP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM592-DP master	CI541-DP CI542-DP	x	x	--	remote I/O
CM592-DP master	CI541-DP CI542-DP	x	--	--	hot-swap I/O

Table 384: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	--	--	hot swap I/O

Table 385: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM598-CN master	CI581-CN CI582-CN	x	x	--	remote I/O

Table 386: EtherCAT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

Table 387: CS31 bus

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard COM1 interface	DC551-CS31 CI592-CS31	x	x	--	remote I/O
Onboard COM1 interface	CI590-CS31-HA	x	--	--	high availability
CM574-RS	DC551-CS31 CI592-CS31	x	x	--	remote I/O
CM574-RS	CI590-CS31-HA	x	--	--	high availability

Technical data (Overview)

Communication module	Field bus	Transmission rate	Field bus connector	Processor	Communication module interface	Current consumption from 24 V DC power supply at the terminal base of the CPU	Internal RAM memory	External RAM memory	External flash memory
CM574-RCOM	RCOM/RCOM+	2.4 ... 19.2 kBit/s	MC 0.5/9-G-2.5, 9-pin, male	PowerPC	Dual-port memory, 8 kB	Typ. 80 mA	256 kB	--	--
CM574-RS	Serial (ASCII/Modbus)	9.6 ... 187.5 kBit/s	MC 0.5/9-G-2.5, 9-pin, male	PowerPC	Dual-port memory, 8 kB	Typ. 80 mA	256 kB	--	512 kB (firmware) + 2 x 64 kB (user data)

Communication module	Field bus	Transmission rate	Field bus connector	Processor	Communication module interface	Current consumption from 24 V DC power supply at the terminal base of the CPU	Internal RAM memory	External RAM memory	External flash memory
CM579-ETHCAT	EtherCAT	10 or 100 MBit/s	2 x RJ45	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 85 mA	128 kB	8 MB	4 or 8 MB
CM582-DP CM592-DP	PROFIBUS DP	9.6 kBit/s ... 12 MBit/s	D-sub, 9-pin, female, bended	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 65 mA	128 kB	8 MB	8 MB
CM598-CN CM588-CN	CANopen	10 ... 1 MBit/s	COMBICON 2x 5-pin, bended	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 65 mA	128 kB	8 MB	8 MB
CM589-PNIO(-4) CM579-PNIO	PROFINET	100 MBit/s	2 x RJ45	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 85 mA	128 kB	8 MB	4 or 8 MB
CM597-ETH	2 x Ethernet	10 or 100 MBit/s	2 x RJ45	Hilscher NETX 100	Dual-port memory, 16 kB	Typ. 85 mA	128 kB	8 MB	8 MB

1.6.2.4.2 Compatibility of communication modules and communication interface modules

Table 388: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O
CM597-ETH	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 389: PROFIBUS DP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM592-DP master	CI541-DP CI542-DP	x	x	--	remote I/O
CM592-DP master	CI541-DP CI542-DP	x	--	--	hot-swap I/O

Table 390: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	--	--	hot swap I/O

Table 391: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM598-CN master	CI581-CN CI582-CN	x	x	--	remote I/O

Table 392: EtherCAT

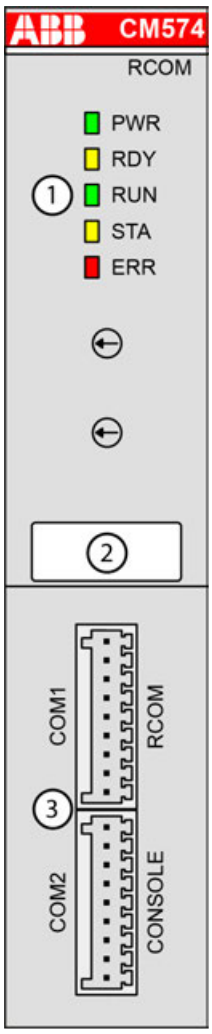
Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

Table 393: CS31 bus

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard COM1 interface	DC551-CS31 CI592-CS31	x	x	--	remote I/O
Onboard COM1 interface	CI590-CS31-HA	x	--	--	high availability
CM574-RS	DC551-CS31 CI592-CS31	x	x	--	remote I/O
CM574-RS	CI590-CS31-HA	x	--	--	high availability

1.6.2.4.3 RCOM / RCOM+

CM574-RCOM for RCOM/RCOM+



1 5 LEDs for state display
2 Label
3 2 interfaces: 1 RCOM protocol interface, 1 CONSOLE

Purpose

Communication module CM574-RCOM is equipped with 2 serial interfaces (RCOM protocol communication and Console) which provide the remote protocol RCOM/RCOM+.

Depending on the connection, the physical interface of the RCOM protocol interface and of the debugging terminal interface is either RS-232 or RS-485.

Connections

Serial interfaces

PIN assignment The serial interface connectors (COM1/COM2) have the following pin assignment:

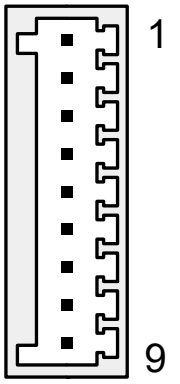
Pin		Signal	Interface	Description
	1	Term. P	RS-485	Terminator P
	2	RxD/TxD-P	RS-485	Receive/Transmit, positive
	3	RxD/TxD-N	RS-485	Receive/Transmit, negative
	4	Term. N	RS-485	Terminator N
	5	RTS	RS-232	Request to send (output)
	6	TxD	RS-232	Transmit data (output)
	7	SGND	Signal Ground	Signal Ground
	8	RxD	RS-232	Receive data (input)
	9	CTS	RS-232	Clear to send (input)

Table 394: Protocols:

No.	Protocol	Description
COM1		
1	Online access	Online access for IEC 61131-3 programming via serial driver
2	Modbus	Modbus RTU, master or slave
3	ASCII	Any protocol with FB COM_SEND, COM_REC
4	SysLibCom	Support for blocks contained in the SysLibCom.lib library
5	Multi	Switch between two protocols (Online access, Modbus, ASCII, SysLibCom) using the block COM_SET_PROT
6	CS31 bus	CS31 bus master
7	RCOM/RCOM+	ABB remote protocol RCOM or RCOM+ (only available as separate communication module CM574-RCOM)
COM2		
1	Online access	Online access for IEC 61131-3 programming with serial driver
2	Modbus	Modbus RTU, master or slave
3	ASCII	Any protocol with FB COM_SEND, COM_REC
4	SysLibCom	Support for SysLibCom.lib library blocks
5	Multi	Switch between two protocols (Online access, Modbus, ASCII, SysLibCom) using the block COM_SET_PROT

Bus cable for RS-485

Bus cable

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm^2 (24 AWG)
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends
Remarks	Commonly used telephone cables with PE insulation and a core diameter of > 0.8 mm are usually sufficient.
	Cables with PVC core insulation and core diameter of 0.8 mm can be used up to a length of approx. 250 m. In this case, the bus terminating resistor is approx. 100 Ω.

Cable lengths

The maximum possible cable length of a serial connection subnet within a segment depends on the transmission rate (transmission rate).

COM1 - RCOM:

Parameter	**Value**
Transmission rate	2.4 kBaud to 19.2 kBaud
Maximum cable length	On request

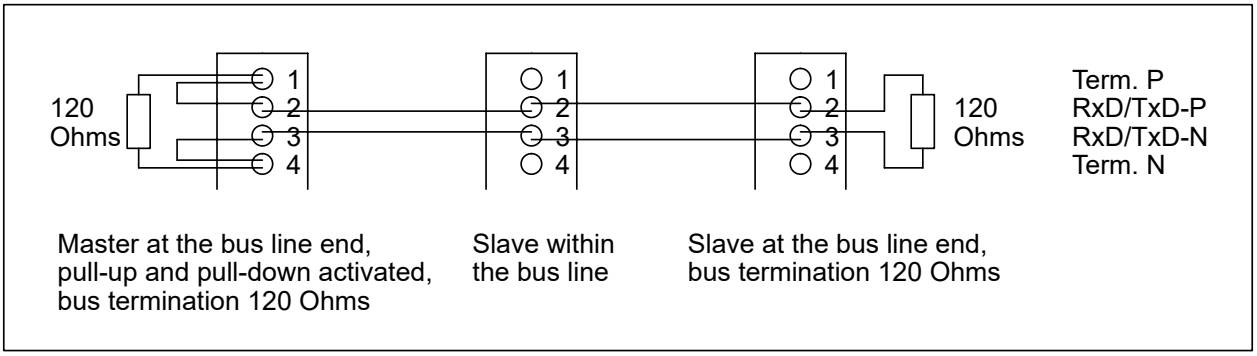
COM2 - CONSOLE:

Parameter	**Value**
Transmission rate	19.2 kBaud
Maximum cable length	On request

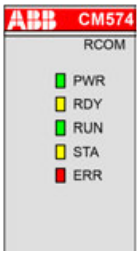
Transmission rate	Maximum cable length
19.2 kBaud	On request

Bus termination (RS-485 only)

The line ends of the bus segment must be equipped with bus terminating resistors. Normally, these resistors are integrated in the interface connectors.



State LEDs

LED		Color	State	Description
	PWR	Green	ON	Voltage is present
			OFF	Voltage is missing
	RDY	Yellow	ON	Communication module is ready
			Flashes cyclically	Event queue blocked (slave devices only)
			OFF	Hardware defective
	RUN	Green	ON	Normal operation
			Flashes cyclically	Protocol error occurred
			OFF	No communication
	STA	Yellow	Flashes	Traffic detected
	ERR	Red	ON	Error
			OFF	No error

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Interface	Serial interface
Transmission rate	2.4 kbit/s to 19.2 kbit/s
Protocol	RCOM/RCOM+
Interface connector	MC 0.5/9-G-2.5, 9-pin, male
Processor	PowerPC

Parameter	Value
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Ambient temperature	see: System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System Data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Communication module bus	Dual-port memory, 8 kB
Internal power supply	Through the communication module bus of the terminal base
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 80 mA
Internal RAM memory	256 kB
External RAM memory	-
External flash memory	512 kB (firmware)
State display	PWR, RDY, RUN, STA, ERR
Weight	Ca. 150 g

Table 395: Technical data of the interfaces

Parameter	Value
Serial interface standard	EIA RS-232 or EIA RS-485
Interface connector	Pluggable 9-pin terminal block
Potential separation	Yes, from the CPU, 500 V DC
Serial interface parameters	Protocol interface configurable via PLC configuration. Preset configuration for debugging the terminal interface.
Modes of operation	Data exchange
Protocols supported	RCOM/RCOM+

The pin assignment of the serial interfaces RCOM and OPERATOR is identical to the serial interface COM1 of the processor modules PM57x, PM58x and PM59x.

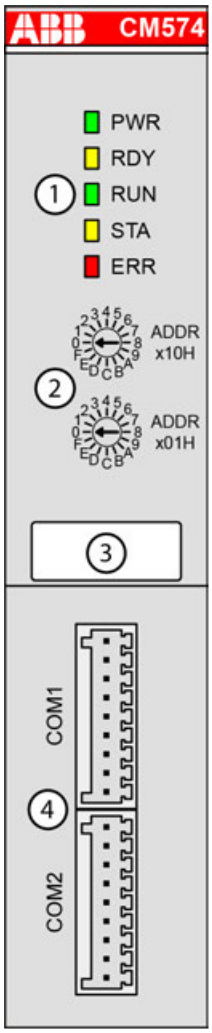
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 170 401 R0201	CM574-RCOM, communication module, 2 serial RS-232/485, RCOM/RCOM+ protocol	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.4.4 Serial

CM574-RS with 2 serial interfaces



1 5 LEDs for state display
2 2 rotary switches for address setting
3 Label
4 2 serial communication interfaces

Purpose

Communication module CM574-RS is equipped with 2 serial interfaces (COM1 and COM2) which can be used as programming interface or for communication e.g. for communication via Modbus or ASCII.

The CM574-RS can be a CS31 master at COM1 and COM2.

Depending on the connection, the physical interface of COM1 and COM2 is either RS-232 or RS-485.

Connections

Serial interfaces

PIN assignment The serial interface connectors (COM1/COM2) have the following pin assignment:

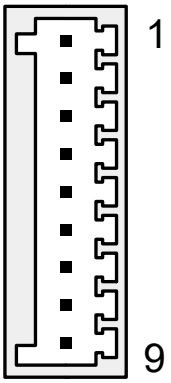
Pin		Signal	Interface	Description
	1	Term. P	RS-485	Terminator P
	2	RxD/TxD-P	RS-485	Receive/Transmit, positive
	3	RxD/TxD-N	RS-485	Receive/Transmit, negative
	4	Term. N	RS-485	Terminator N
	5	RTS	RS-232	Request to send (output)
	6	TxD	RS-232	Transmit data (output)
	7	SGND	Signal Ground	Signal Ground
	8	RxD	RS-232	Receive data (input)
	9	CTS	RS-232	Clear to send (input)

Table 396: Protocols:

No.	Protocol	Description
COM1		
1	Online access	Online access for IEC 61131-3 programming via serial driver
2	Modbus	Modbus RTU, master or slave
3	ASCII	Any protocol with FB COM_SEND, COM_REC
4	SysLibCom	Support for blocks contained in the SysLibCom.lib library
5	Multi	Switch between two protocols (Online access, Modbus, ASCII, SysLibCom) using the block COM_SET_PROT
6	CS31 bus	CS31 bus master
7	RCOM/RCOM+	ABB remote protocol RCOM or RCOM+ (only available as separate communication module CM574-RCOM)
COM2		
1	Online access	Online access for IEC 61131-3 programming with serial driver
2	Modbus	Modbus RTU, master or slave
3	ASCII	Any protocol with FB COM_SEND, COM_REC
4	SysLibCom	Support for SysLibCom.lib library blocks
5	Multi	Switch between two protocols (Online access, Modbus, ASCII, SysLibCom) using the block COM_SET_PROT

Bus cable for RS-485

Bus cable

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm² (24 AWG)
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends

Bus line	
Remarks	Commonly used telephone cables with PE insulation and a core diameter of > 0.8 mm are usually sufficient.
	Cables with PVC core insulation and core diameter of 0.8 mm can be used up to a length of approx. 250 m. In this case, the bus terminating resistor is approx. 100 Ω.

Cable lengths

The maximum possible cable length of a serial connection subnet within a segment depends on the transmission rate (transmission rate).

RS-232 (for point-to-point connection):

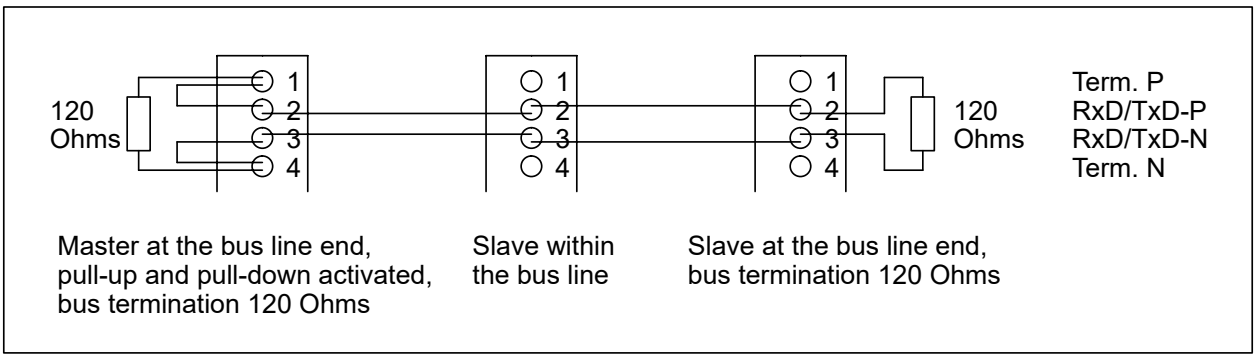
Parameter	**Value**
Transmission rate	9.6 kBaud to 187.5 kBaud
Maximum cable length	On request

RS-485 (for point-to-point or bus connection):

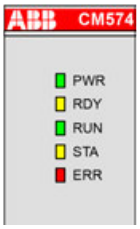
Parameter	**Value**
Transmission rate	9.6 kBaud to 187.5 kBaud
Maximum cable length	On request

Bus termination (RS-485 only)

The line ends of the bus segment must be equipped with bus terminating resistors. Normally, these resistors are integrated in the interface connectors.



State LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Voltage is present
			OFF (dark)	Voltage is missing
	RDY	Yellow	Programmable	Depends on user program
	RUN	Green	Programmable	Depends on user program
	STA	Yellow	Programmable	Depends on user program
	ERR	Red	Programmable	Depends on user program

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	Programmable with Automation Builder e.g. Modbus / ASCII via serial interfaces
Interface	Serial interface
Serial interface standard	EIA RS-232 or EIA RS-485
Potential separation	Yes, from the CPU, 500 V DC
Serial interface parameters	Configurable via software
Modes of operation	Programming or data exchange
Transmission rate	9.6 kbit/s to 187.5 kbit/s
Protocol	Programmable
Interface connector	MC 0.5/9-G-2.5, 9-pin, male
Processor	PowerPC
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Ambient temperature	see: System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System Data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Communication module bus	Dual-port memory, 8 kB
Internal power supply	Through the communication module bus of the terminal base
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 80 mA

Parameter	Value
Internal RAM memory	256 kB
External RAM memory	-
External Flash memory	512 kB (firmware) + 2 x 64 kB (user data)
Status display	PWR, RDY, RUN, STA, ERR
Weight	Ca. 150 g

Ordering data

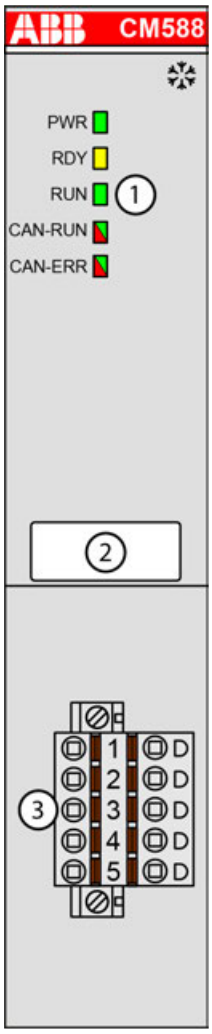
Part no.	Description	Product life cycle phase *)
1SAP 170 400 R0201	CM574-RS, communication module, 2 serial RS232/485, free configurable serial interface module	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.4.5 CANopen

CM588-CN - CANopen slave

- CANopen slave 1 Mbit/s
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 Label
3 Communication interface, 5-pin, Combicon, male, removable plug with spring terminals
❄ Sign for XC version

Purpose

Communication module CM588-CN enables communication via the CANopen field bus. CM588-CN *Chapter 1.6.2.4.5.1 “CM588-CN - CANopen slave” on page 4053* is a slave in a CANopen network. It is connected to the processor module via an internal communication bus. CM588-CN allows communicating of multiple CPUs in a CANopen network.

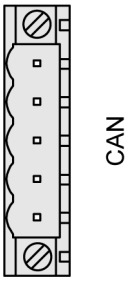
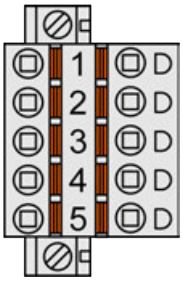
Connections

Field bus interface

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transfer rate (transmission rate)	10 kbit/s, 20 kbit/s, 50 kbit/s, 100 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s,

The CANopen connector has the following pin assignment:

Pin assignment

Interface		PIN	Signal	Description
 Terminal block removed	 Terminal block inserted	1	CAN_GND	CAN reference potential
		2	CAN_L	Bus line, receive/transmit line, LOW
		3	CAN_SHLD	Shield of the bus line
		4	CAN_H	Bus line, receive/transmit line, HIGH
		5	NC	Not connected

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
62.5 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Types of bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

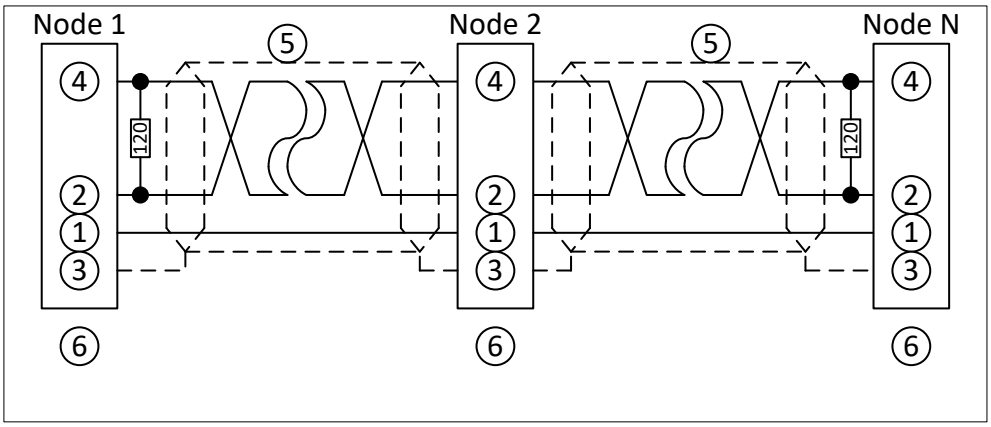


Fig. 822: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

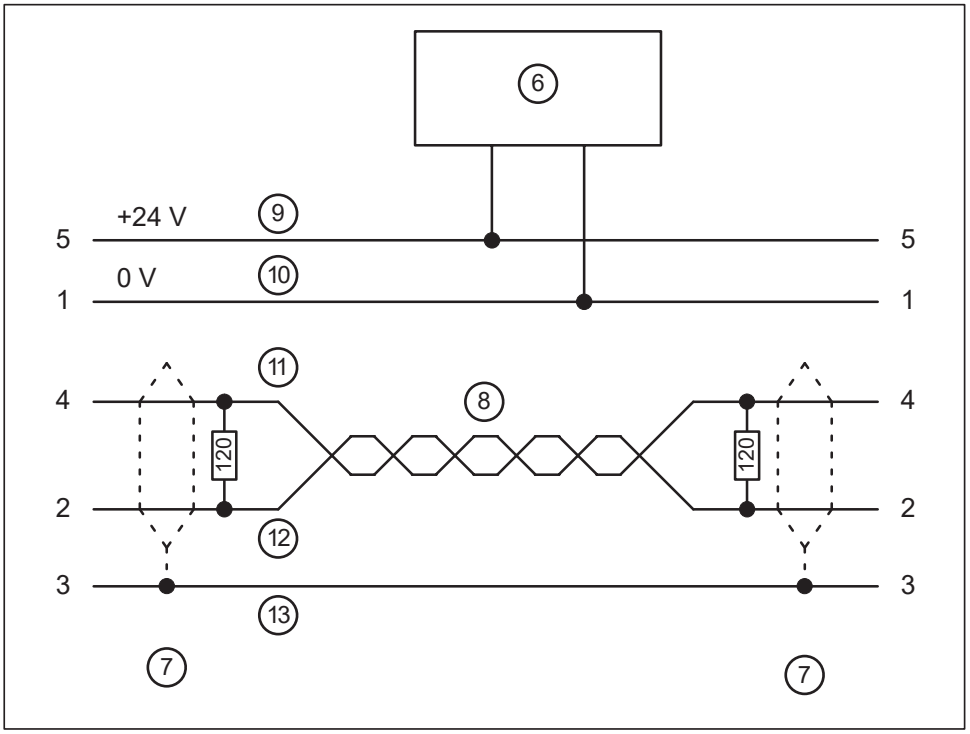


Fig. 823: DeviceNet interface, bus terminating resistors connected to the line ends

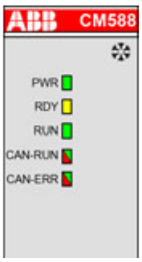
6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to Chapter 1.6.3.6.1 “System data AC500” on page 5313.

State LEDs

The state of the CANopen communication module is displayed by means of 5 state LEDs.

Table 397: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Power supply available
			OFF (dark)	Power supply not available or defective hardware
	RDY	Yellow	ON	Boot procedure
			Blinking	Boot failure
	RUN	Green	ON	Communication module is operational
			OFF	Communication module is not operational
	CAN-RUN	Green	ON	Device configured, CANopen bus in OPERATIONAL state and cyclic data exchange running
			Blinking	CANopen bus in PRE-OPERATIONAL state and slave are being configured
	CAN-ERR	Red	ON	CANopen bus is off
			Blinking	Configuration error
			Single flash	Error counter overflow due to too many error frames
			Double flash	A node-guard or a heartbeat event occurred
			OFF	No error
	CAN-RUN	Yellow	Blinking (synchronously)	No production data available, No bus communication possible.
	CAN-ERR	Yellow		
LED state during firmware update	CAN-RUN	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	CAN-ERR	Red		
	CAN-RUN	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	CAN-ERR	Red		

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	CANopen slave
Technology	Hilscher NETX 100

Parameter	Value
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Bus connection	Pluggable connector COMBICON, 2x5-pin
Internal power supply	Via the communication module Interface of the terminal base
Transfer rate	10 kbit/s to 1 Mbit/s
Transfer method	According to CAN standard
Bus length (segment length max.)	According to table: Maximum cable length within a CANopen field bus
Indicators	5 LEDs
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 65 mA
Weight	Ca. 150 g
Ambient temperature	see: System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System Data AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Adjusting elements	None
Quantity of input and output data per I/O device	Max. 512 byte (respectively for input and output)
Supported protocol services	NMT slave PDO SDO server Heartbeat Nodeguard
Min. bus cycle	1 ms

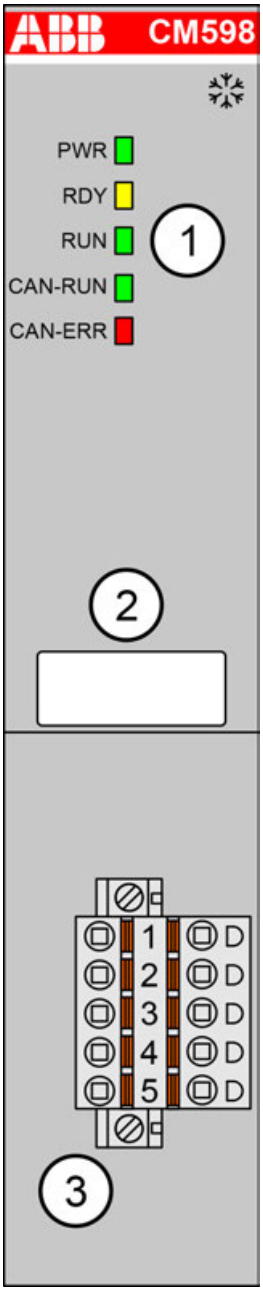
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 172 800 R0001	CM588-CN, communication module CANopen slave	Active
1SAP 372 800 R0001	CM588-CN-XC, communication module CANopen slave, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CM598-CN - CANopen master

- CANopen master 1 Mbit/s
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 Label
3 Communication interface, 5-pin, Combicon, male, removable plug with spring terminals
❄ Sign for XC version

Purpose

Communication module CM598-CN enables communication over the CANopen field bus.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

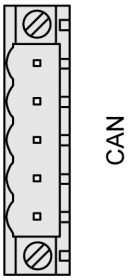
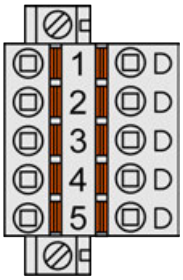
Connections

Field bus interface

Interface socket	5-pin COMBICON
Transmission standard	ISO 11898, potential-free
Transmission protocol	CANopen (CAN), 1 Mbaud max.
Transfer rate (transmission rate)	10 kbit/s, 20 kbit/s, 50 kbit/s, 100 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s,

The CANopen connector has the following pin assignment:

Pin assignment

Interface		PIN	Signal	Description
 Terminal block removed	 Terminal block inserted	1	CAN_GND	CAN reference potential
		2	CAN_L	Bus line, receive/transmit line, LOW
		3	CAN_SHLD	Shield of the bus line
		4	CAN_H	Bus line, receive/transmit line, HIGH
		5	NC	Not connected

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
62.5 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Types of bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

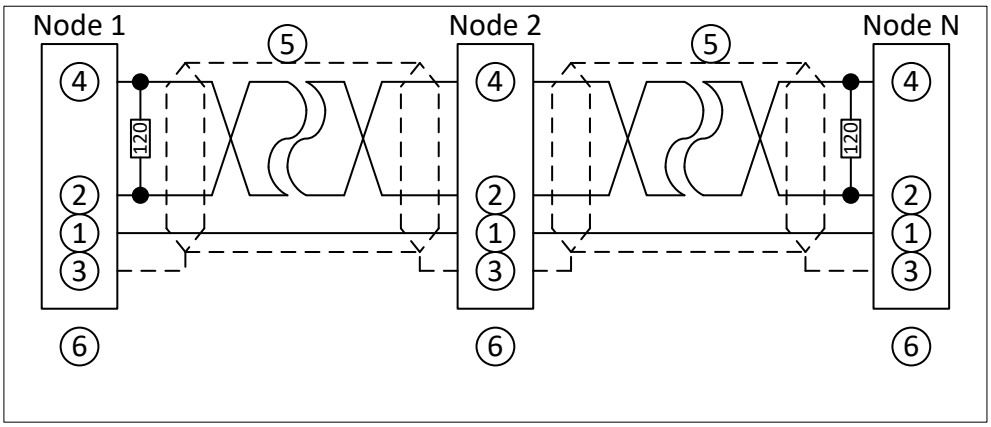


Fig. 824: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

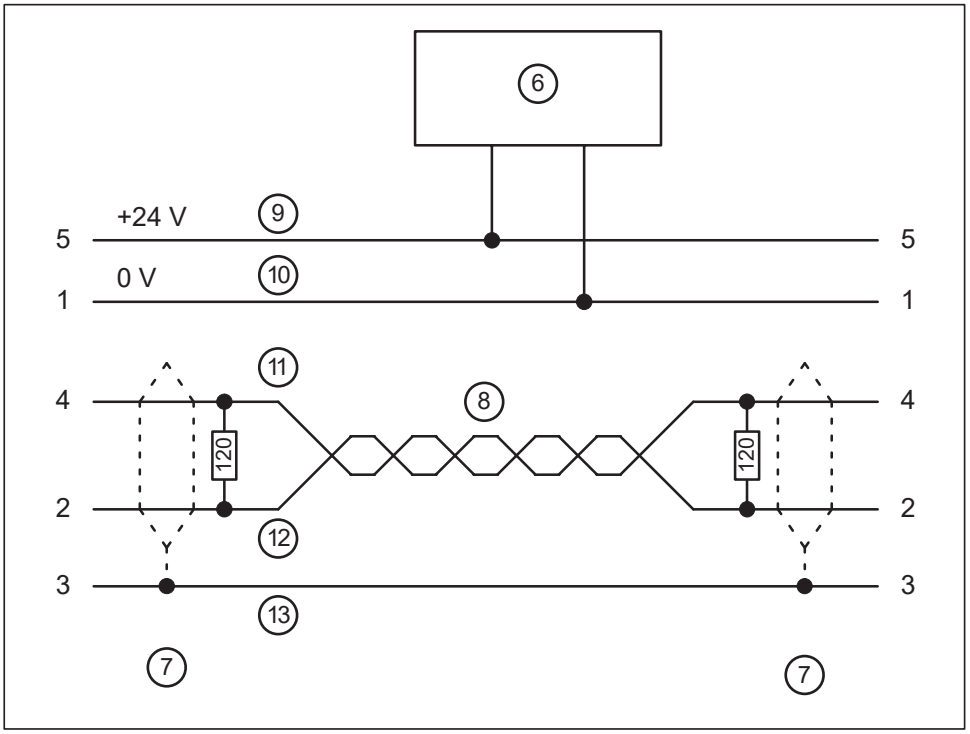


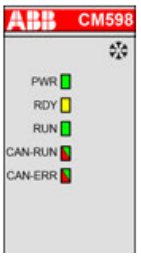
Fig. 825: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to *Chapter 1.6.3.6.1 “System data AC500” on page 5313.*

State LEDs

Table 398: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Power supply available
			OFF (dark)	Power supply not available or defective hardware
	RDY	Yellow	ON	Boot procedure
			Blinking	Boot failure
			OFF	---
	RUN	Green	ON	Communication module is operational
			Blinking	---
			OFF	Communication module is not operational
	CAN-RUN	Green	ON	Operational: Device is in the OPERATIONAL state
			Single Flash	Stopped: Device is in STOPPED state
			Blinking	Pre-operational: Device is in the PREOPERATIONAL state
			OFF	No communication or no power supply
	CAN-ERR	Red	ON	CANopen bus is off
			Single flash	Warning limit reached: At least one of the error counters of the CAN controller has reached or exceeded the warning level (too many error frames)
			Double flash	Error control event: A guard event (NMT Slave or NMTmaster) or a heartbeat event (Heartbeat consumer) has occurred
			OFF	No Error: Device is in working condition
	CAN-RUN	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	CAN-ERR	Yellow		
LED state during firmware update	CAN-RUN	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	CAN-ERR	Red		
	CAN-RUN	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	CAN-ERR	Red		

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	CANopen master, CAN2A, CAN2B
Transmission rate	10 kbit/s to 1 Mbit/s
Ambient temperature	see: System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System Data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Field bus connector	Pluggable connector COMBICON, 5-pin
Technology	Hilscher NETX 100
Indicators	5 LEDs
Internal power supply	Via the communication module interface of the terminal base
Current consumption from 24 V DC power supply at the Terminal Base of the CPU	Typ. 65 mA
Number of Slaves	Max. 126
Number of receive/transmit PDOs	Max. 512 (respectively for receive and transmit)
Total quantity of input and output data	Max. 3584 byte (respectively for input and output)
Weight	Ca. 150 g

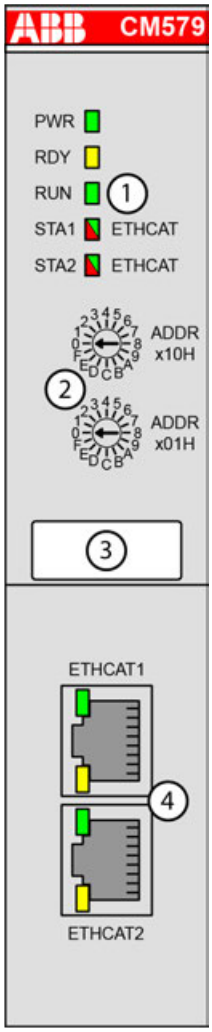
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 173 800 R0001	CM598-CN, communication module CANopen master	Active
1SAP 373 800 R0001	CM598-CN-XC, communication module CANopen master, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.4.6 EtherCAT

CM579-ETHCAT - EtherCAT master



1 5 LEDs for state display
2 2 rotary switches for address setting (not used)
3 Label
4 2 communication interfaces RJ45 (ETHCAT1 and ETHCAT2)

Intended purpose

Communication module CM579-ETHCAT is for EtherCAT communication.

The comunication module is configured via the dual-port memory by means of a system configurator. The configuration is saved on a non-volatile Flash EPROM memory.

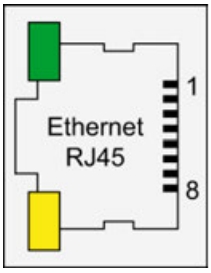
Chapter 1.7.3.4.3 “CM579-ETHCAT” on page 6500

Connections

Field bus interfaces

The EtherCAT communication module provides 2 RJ45 interfaces with the following pin assignment. The pin assignment is used for the EtherCAT slaves (communication interface modules CI5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

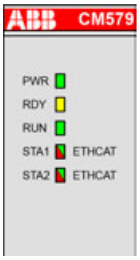
At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

State LEDs

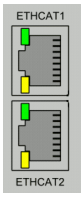
The EtherCAT state is shown by the EtherCAT communication module's LEDs. Some LEDs are two-colored.

Table 399: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Blinking	---
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
			Off	---
	RUN	Green	On	Communication module is operational
			Blinking	---
			Off	Communication module is not operational
	STA1	Green	On	No bus error, communication running
			Blinking	Establishing communication
			Off	System error
	STA2	Red	On	Configuration error
			Blinking	---
			Off	No error
	STA1	Yellow	Blinking	No production data available,
	STA2	Yellow	(synchronously)	no bus communication possible.
LED state during firmware update	STA1	Green	Blinking	Firmware file transfers during communication module firmware update.
	STA2	Red	(synchronously)	
	STA1	Green	Blinking	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red	(alternately)	

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 400: Meaning of the diagnosis LEDs

LED		Color	State	Description
	ETHCAT1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	ETHCAT1 LED "RX/TX"	Yellow	On	Device sends/receives frames
			Off	No Ethernet connection
	ETHCAT2 LED "Link"	Green		Connector ETHCAT2 is not used
	ETHCAT2 LED "RX/TX"	Yellow		

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Internal Supply	Via the communication module interface of the terminal base
Protocol	EtherCAT
Field bus connector	2 x RJ45 (ETHCAT1 and ETHCAT2)
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Bus length (segment length max.)	100 m at 100 Mbit/s
Indicators	5 LEDs
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Ambient temperature	System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System Data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal supply	Via the communication module interface of the terminal base
Number of slaves	Limited to 200
Quantity of input and output data for a single slave	Max. 5760 bytes (respectively for input and output)
Total quantity of input and output data	Max. 5760 bytes (only valid for asynchronous operation, for synchronous operation the reachable values depends on the additional load of SoE, CoE and EoE, typical reachable values are 1024 bytes).
Supported protocols	RTC - Real-time cyclic protocol, class 1 RTA - Real-time acyclic protocol
Acyclic services	● CoE upload ● CoE download (1500 bytes max.) ● Emergency
Min. bus cycle	1 ms

Parameter	Value
Max. size of the bus configuration file	2 MB
Weight	Ca. 170 g

Ordering data

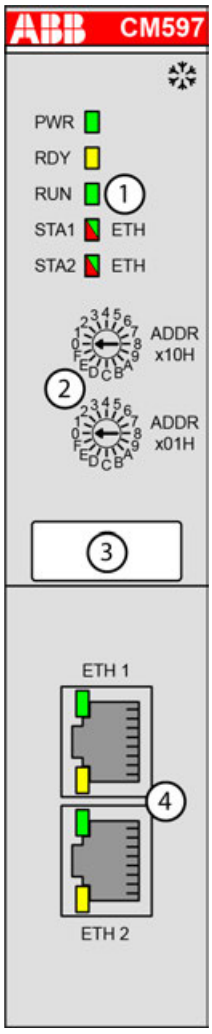
Part no.	Description	Product life cycle phase *)
1SAP 170 902 R0101	CM579-ETHCAT, EtherCAT communication module	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.4.7 Ethernet

CM597-ETH - Communication module Ethernet

- TCP/IP with integrated 2-port switch
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 2 rotary switches for address setting
3 Label
4 2 communication interfaces Ethernet RJ45
Sign for XC version

Purpose

The communication module provides communication via the Ethernet bus. Ethernet connection can be established directly to the communication module, an additional switch is not necessary.

The Ethernet communication module is an intelligent 100Base-T-Ethernet communication interface based on the highly integrated netX100 microcontroller. The complete TCP/IP protocol and the application layers are supported.

The user interface is based on a dual-port memory. The Ethernet communication runs via RJ45 interfaces.

The communication module is configured via the dual-port memory, the diagnosis interface or a TCP/IP connection by means of a system configurator.

It is not possible to close a RSTP ring by using the two ports of the communication module.

Applications:

- TCP/IP for PC/ Automation Builder (programming)
- UDP (communication via the function blocks ETH_UDP_SEND *Chapter 1.5.4.13.1.11 “ETH_UDP_SEND” on page 1225* and ETH_UDP_REC *Chapter 1.5.4.13.1.10 “ETH_UDP_REC” on page 1222*
- Modbus on TCP/IP (Modbus on TCP/IP, client and server)

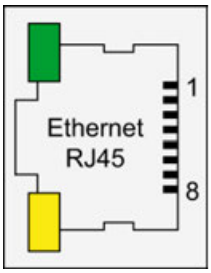
For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Connections

Field bus interfaces

The Ethernet communication module has 2 RJ45 interfaces:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

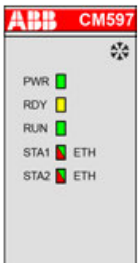
Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.*

State LEDs

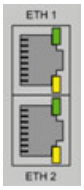
The Ethernet state is shown by the Ethernet communication module's LEDs.

Table 401: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
	RUN	Green	On	Communication module is operational
			Off	Communication module is not operational
	STA1	Green	Blinking (1 Hz)	Device ready
			Blinking (5 Hz)	Device configured / UDP traffic
			On	Modbus communication established
	STA2	Red	On	Modbus communication error
			Off	No error
	STA1	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	STA2	Yellow		
LED state during firmware update	STA1	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	STA2	Red		
	STA1	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red		

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 402: Meaning of the diagnosis LEDs

LED		Color	State	Description
	ETH1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	ETH1 LED "RX/TX"	Yellow	On	---
			Blinking	Device sends/receives frames
			Off	---
	ETH2 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	ETH2 LED "RX/TX"	Yellow	On	---
			Blinking	Device sends/receives frames
			Off	---

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Field bus	2 x Ethernet
Transmission rate	10 Mbit/s or 100 Mbit/s
Protocol	Ethernet TCP/IP, UDP/IP, Modbus TCP, ICMP (Ping), DNS, SMTP (email)
Field bus connectors	2 x RJ45, with integrated 2-port switch
Processor	Hilscher NETX 100
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Communication module interface	Dual-port memory, 16 kB
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal power supply	Via the communication module interface of the terminal base
External RAM memory	8 MB
External flash memory	8 MB
State display	PWR, RDY, RUN, STA, ERR, 2 x LINK, 2 x ACT
Ethernet	10/100 Base-TX, internal switch, 2 x RJ45 socket
LED indication	State indication via 5 LEDs
Station identification	Rotary switch, 0...255 (00...FFhex)
Transmission mode	Half or full-duplex operation, adjustable
Transmission rate	10 or 100 Mbit/s, adjustable
Auto negotiation	Optionally adjustable
MAC address	Optionally configurable
Ethernet frame types	Ethernet II (RFC 894), IEEE 802.3 receive only (RFC 1042)
Weight	Ca. 170 g

Ordering data

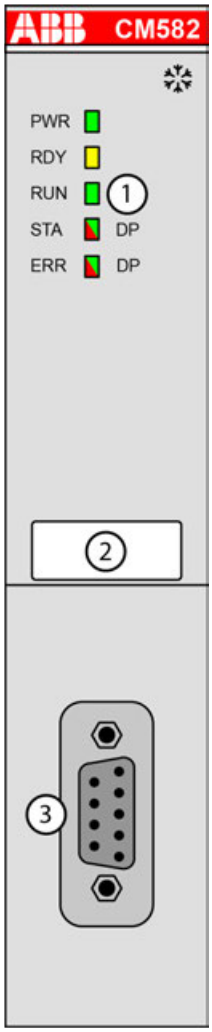
Part no.	Description	Product life cycle phase *)
1SAP 173 700 R0001	CM597-ETH, communication module Ethernet TCP/IP with integrated 2-port switch	Active
1SAP 373 700 R0001	CM597-ETH-XC, communication module Ethernet TCP/IP with integrated 2-port switch, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

1.6.2.4.8 PROFIBUS

CM582-DP - PROFIBUS DP slave

- PROFIBUS DP slave 12 Mbit/s
- Compatible with Automation Builder version starting from V2.0.2, and with CPU firmware version starting from V2.6
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 Label
3 Communication interface PROFIBUS DP D-sub, 9-pin, female
❄ Sign for XC version

Purpose

Communication module CM582-DP enables communication over the PROFIBUS DP field bus.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Connections

Field bus interface

The PROFIBUS DP connector (9-pin, female) has the following pin assignment:

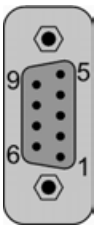
Pin		Signal	Description
	1	NC	Not connected
	2	NC	Not connected
	3	RxD/TxD-P	Receive/Transmit positive
	4	CNTR-P	Control signal for repeater, positive
	5	DGND	Reference potential for data exchange and +5 VI
	6	VP	+5 V (power supply for the bus terminating resistors)
	7	NC	Not connected
	8	RxD/TxD-N	Receive/Transmit negative
	9	NC	Not connected

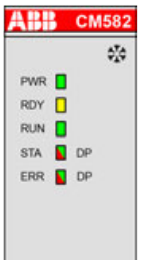
Table 403: Correlation of transmission rate, bit time and cable length:

Tranmission rate in [kbit/s]	Bit time [tBit]	Max. cable length in [m]
9.6	104.2 µs	1200
19.2	52.1 µs	1200
31.25	32 µs	1200
45.45	22 µs	1200
93.75	10.7 µs	1200
187.5	5.3 µs	1000
500	2 µs	400
1500	666.7 ns	200
3000	333.3 ns	100
6000	166.7 ns	100
12000	83.3 ns	100

State LEDs

The PROFIBUS state is shown by state LEDs.

Table 404: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Power supply available.
			OFF (dark)	Power supply not available or defective hardware
	RDY	Yellow	ON	Boot procedure
			Blinking	Boot failure
			OFF	---
	RUN	Green	ON	Communication module is operational
			Blinking	---
			OFF	Communication module is not operational
	STA	Green	ON	Communication to all slaves is established
			Flashes cyclic	---
			Flashes non-cyclic	No configuration or stack error
			OFF	No communication
	ERR	Red	Blinking	No data exchange to the master module or the cable is disconnected
			OFF	No error
	STA	Yellow	Blinking	No production data available,
	ERR	Yellow	(synchronously)	no bus communication possible.
LED state during firmware update	STA	Green	Blinking	Firmware file transfers during communication module firmware update.
	ERR	Red	(synchronously)	
	STA	Green	Blinking	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	ERR	Red	(alternately)	

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
State indication	By 5 LEDs PWR, RDY, RUN, STA, ERR
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 65 mA
Internal power supply	Through the communication module interface of the terminal base
Maximum number of cyclic input data	244 bytes
Maximum number of cyclic output data	244 bytes
Maximum number of acyclic read/write	240 bytes
Configuration data	max. 244 bytes
Parameter data	237 bytes application specific parameters
Processor	Hilscher NETX 100
Internal RAM memory	8 MB
External Flash memory	8 MB
Weight	Ca. 150 g

Technical data of the interface

Parameter	Value
Interface socket	9-pin, D-sub socket
Transmission standard	EIA RS-485 acc. to IEC 61158/61784, potential-free
Transmission protocol	PROFIBUS DP
Transmission rate	9.6 kbit/s up to 12 Mbit/s

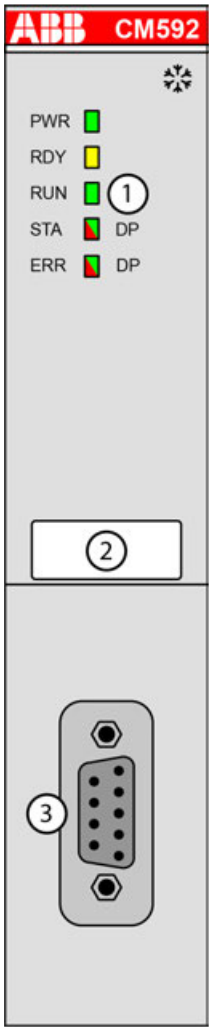
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 172 200 R0001	CM582-DP, communication module PROFIBUS DP slave, 12 MBit/s	Active
1SAP 372 200 R0001	CM582-DP, communication module PROFIBUS DP slave, 12 MBit/s, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CM592-DP - PROFIBUS DP master

- Master 12 Mbit/s
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 Label
3 Communication interface PROFIBUS DP D-sub, 9-pin, female
❄ Sign for XC version

Purpose

Communication module CM592-DP enables communication over the PROFIBUS DP field bus.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Connections

Field bus interface

The PROFIBUS DP connector (9-pin, female) has the following pin assignment:

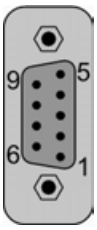
Pin		Signal	Description
	1	NC	Not connected
	2	NC	Not connected
	3	RxD/TxD-P	Receive/Transmit positive
	4	CNTR-P	Control signal for repeater, positive
	5	DGND	Reference potential for data exchange and +5 VI
	6	VP	+5 V (power supply for the bus terminating resistors)
	7	NC	Not connected
	8	RxD/TxD-N	Receive/Transmit negative
	9	NC	Not connected

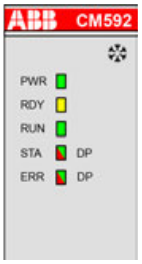
Table 405: Correlation of transmission rate, bit time and cable length:

Tranmission rate in [kbit/s]	Bit time [tBit]	Max. cable length in [m]
9.6	104.2 µs	1200
19.2	52.1 µs	1200
31.25	32 µs	1200
45.45	22 µs	1200
93.75	10.7 µs	1200
187.5	5.3 µs	1000
500	2 µs	400
1500	666.7 ns	200
3000	333.3 ns	100
6000	166.7 ns	100
12000	83.3 ns	100

State LEDs

The PROFIBUS state is shown by state LEDs.

Table 406: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	ON (light)	Power supply available
			OFF (dark)	Power supply not available or defective hardware
	RDY	Yellow	ON	Boot procedure
			Blinking	Boot failure
			OFF	---
	RUN	Green	ON	Communication module is operational
			Blinking	---
			OFF	Communication module is not operational
	STA	Green	ON	Communication to all slaves is established
			Flashes cyclic	---
			Flashes non-cyclic	No configuration or stack error
			OFF	No communication
	ERR	Red	ON	Communication to one/all slaves is disconnected
			Flashes cyclic	Communication to at least one slave is disconnected
			OFF	No error
	STA	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	ERR	Yellow		
LED state during firmware update	STA	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	ERR	Red		
	STA	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	ERR	Red		

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
State indication	By 5 LEDs PWR, RDY, RUN, STA, ERR
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 65 mA
Internal power supply	Through the communication module interface of the terminal base
Maximum number of supported slaves	125 (DPV0/DPV1)
Maximum number of total cyclic input data	5712 bytes (Status information is separately managed)
Maximum number of total cyclic output data	5760 bytes
Maximum number of cyclic intput data	244 bytes/slave
Maximum number of cyclic output data	244 bytes/slave
Configuration data	max. 244 bytes per slave
Parametrization data per slave	7 bytes/slave standard parameters 237 bytes/slave application specific parameters
Maximum number of acyclic read/write	240 bytes per slave and telegram
Processor	Hilscher NETX 100
Internal RAM memory	8 MB
External Flash memory	8 MB
Weight	Ca. 150 g

Technical data of the interface

Parameter	Value
Interface socket	9-pin, D-sub socket
Transmission standard	EIA RS-485 acc. to IEC 61158/61784, potential-free
Transmission protocol	PROFIBUS DP
Transmission rate	9.6 kbit/s up to 12 Mbit/s

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 173 200 R0001	CM592-DP, communication module PROFIBUS DP master, 12 MBit/s	Active
1SAP 373 200 R0001	CM592-DP, communication module PROFIBUS DP master, 12 MBit/s, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

PROFIBUS connection details

Attachment plug for the bus cable 9-pin D-sub connector, male

Parameter	Value
Fastening torque	0.4 Nm

Assignment

Pin	Signal	Description
1	Shield	Shielding, protective ground
2	not used	-
3	RxD/TxD-P	Reception / transmission line, positive
4	CBTR-P	Control signal for repeater, positive (optional)
5	DGND	Reference potential for data lines and +5 V
6	VP	+5 V, supply voltage for bus terminating resistors
7	not used	-
8	RxD/TxD-N	Reception / transmission line, negative
9	CNTR-N	Control signal for repeater, negative (optional)

Bus cable

Parameter	Value
Type	Twisted pair (shielded)
Characteristic impedance	135 Ω...165 Ω
Cable capacitance	< 30 pF/m
Conductor diameter of the cores	≥ 0.64 mm
Conductor cross section of the cores	≥ 0.34 mm²
Cable resistance per core	≤ 55 Ω/km
Loop resistance (resistance of two cores)	≤ 110 Ω/km

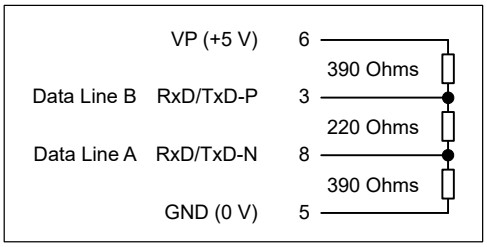
Cable lengths The maximum possible cable length of a PROFIBUS subnet within a segment depends on the tranmission rate (baud rate).

Transmission Rate	Maximum Cable Length
9.6 / 19.2 / 93.75 kBaud	1200 m
187.5 kBaud	1000 m
500 kBaud	400 m
1.5 MBaud	200 m
3 MBaud to 12 MBaud	100 m

Branch lines are generally permissible for transmission rates of up to 1500 kbit/s. But in fact they should be avoided for transmission rates higher than 500 kbit/s.

Bus terminating resistors

The line ends (of the bus segments) have to be terminated using bus terminating resistors according to the drawing below. The bus terminating resistors are usually placed inside the bus connector.



Repeaters

One bus segment can have up to 32 subscribers. Using repeaters a system can be expanded to up to 126 subscribers. Repeaters are also required for longer transfer lines. Please note that a repeater's load to the bus segment is the same as the load of a normal bus subscriber. The sum of normal bus subscribers and repeaters in one bus segment must not exceed 32.

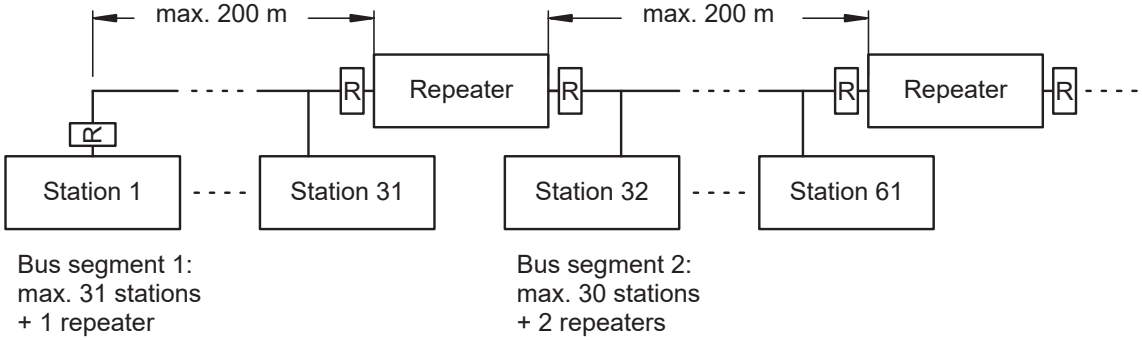
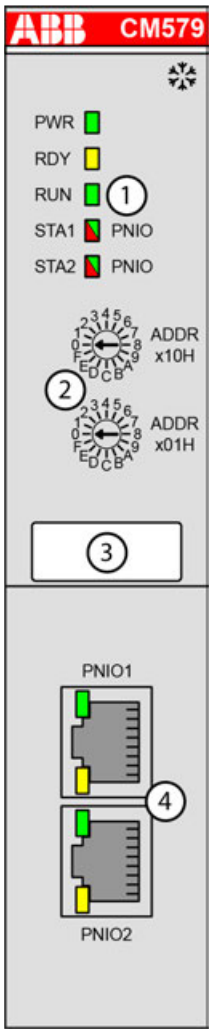


Fig. 826: Principle example for a PROFIBUS-DP system with repeaters (1500 kbit/s baud rate)

1.6.2.4.9 PROFINET

CM579-PNIO - PROFINET IO RT controller

- PROFINET IO controller
- Integrated 2-port switch
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 2 rotary switches for address setting (not used)
3 Label
4 2 communication interfaces RJ45 (PNIO1 and PNIO2)
❄ Sign for XC version

Intended purpose

The communication module is for PROFINET RT communication.

The PROFINET communication module includes an internal Ethernet switch. The connection to the Ethernet can be established directly to the communication module. An additional switch is not necessary.

The communication module is configured via the dual-port memory by means of a system configurator. The configuration is saved on a non-volatile Flash EPROM memory.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

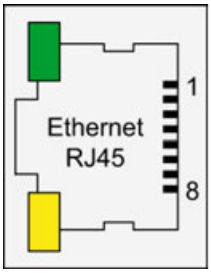
Parameter	Value
Protocol	PROFINET IO RT
Usable CPUs	PM57x, PM58x, PM59x ↳ *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Field bus connector	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Internal supply	Via the communication module interface of the terminal base

Connections

Field bus interfaces

The communication module provides 2 RJ45 interfaces.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

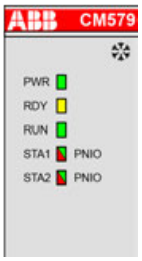
Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet ↳ Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

State LEDs

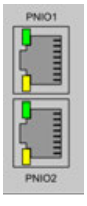
The PROFINET state is shown by the state LEDs.

Table 407: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Blinking	---
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
			Off	---
	RUN	Green	On	Communication module is operational
			Blinking	---
			Off	Communication module is not operational
	STA1	Red	On	Diagnosis alarm reported. At least one device is having a diagnosis alarm. In incorporation with STA2 PNIO: License fault.
			Blinking	System error
			Off	No system error
	STA2	Red	On	No connection; in incorporation with STA1 PNIO: license fault
			Blinking	Configuration fault: some configured I/O modules are not connected
			Off	No bus error, communication is running
	STA1	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	STA2	Yellow		
LED state during firmware update	STA1	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	STA2	Red		
	STA1	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red		

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 408: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PNIO1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	PNIO1 LED "RX/TX"	Yellow	On	---
			Blinking	PROFINET device sends/receives frames
			Off	---
	PNIO2 LED "Link"	Green	On	Ethernet connection established

LED		Color	State	Description
			Off	No Ethernet connection
	PNIO2 LED "RX/TX"	Yellow	On	---
			Blinking	PROFINET device sends/receives frames
			Off	---

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	PROFINET IO RT
Bus connection	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Bus length (segment length max.)	100 m
Indicators	5 LEDs
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Supported alarm types	Process alarm, diagnostic alarm, return of Sub-Module, plug alarm, pull alarm
Alarm processing	Requires handling in application program
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal supply	Via the communication module interface of the terminal base
Weight	Ca. 170 g

Parameter		Value
Supported protocols		RTC - real-time cyclic protocol, class 1 RTA - real-time acyclic protocol DCP - discovery and configuration protocol *) CL-RPC - connectionless remote procedure call Since revision FW 2.4.8.0 additionally LLDP - link layer discovery protocol SNMP - simply network management protocol (SNMP v1)
Acyclic services		PNIO read / write (max. 1392 bytes per telegram, max. 4096 bytes per service request)
Total quantity of input and output data		
	CM579-PNIO < FW 2.4.8.0	1024 bytes per I/O module 3072 bytes in total
	CM579-PNIO = FW 2.4.8.0	1024 bytes per I/O module 4096 bytes in total
	CM579-PNIO > FW 2.4.8.0	1440 bytes per I/O module 4096 bytes in total
Min. bus cycle		1 ms
Conformance class		CC A

*) CM579-PNIO does not allow setting "Station name" by using PROFINET service "DCP SET NameOfStation".

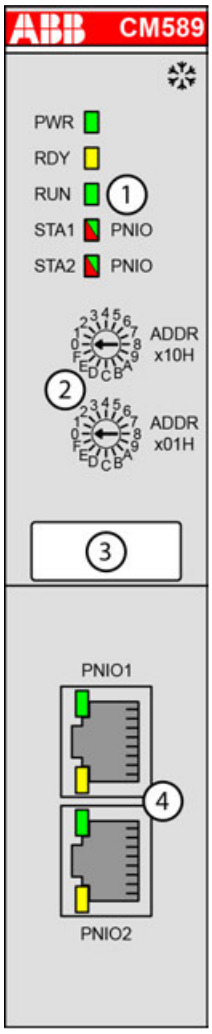
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 170 901 R0101	CM579-PNIO, PROFINET communication module	Active
1SAP 370 901 R0101	CM579-PNIO-XC, PROFINET communication module, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CM589-PNIO(-4) - PROFINET IO RT with 4 devices

- PROFINET IO device
- Integrated 2-port switch
- XC version for use in extreme ambient conditions available



1 5 LEDs for state display
2 2 rotary switches for setting the IO device identifier
3 Label
4 2 communication interfaces RJ45 (PNIO1 and PNIO2)
Sign for XC version

The communication module is for PROFINET RT communication.

The PROFINET communication module includes an internal Ethernet switch. The connection to the Ethernet can be established directly to the communication module. An additional switch is not necessary.

The communication module is configured via the dual-port memory by means of a system configurator. The configuration is saved on a non-volatile Flash EPROM memory.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

CM589-PNIO(-4)

CM589-PNIO supports one application relation to communicate to one single PROFINET IO controller.

CM589-PNIO-4 supports 4 application relations to communicate to up to 4 PROFINET IO controllers in parallel using PROFINET Shared Device technology.

Functionality

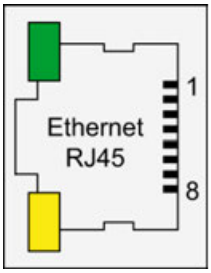
Parameter	Value
Protocol	PROFINET IO RT
Usable CPUs	PM57x, PM58x, PM59x *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
Usable terminal bases	All TB5xx *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Field bus connector	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Internal supply	Via the communication module interface of the terminal base

Connections

Field bus interfaces

The PROFINET communication module provides 2 RJ45 interfaces:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

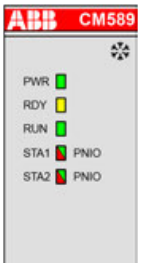
Addressing

> *The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.*

State LEDs

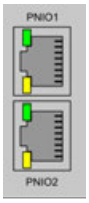
The PROFINET state is shown by the state LEDs.

Table 409: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PWR	Green	On	Power supply available
			Blinking	---
			Off	Power supply not available or defective hardware
	RDY	Yellow	On	Boot procedure
			Blinking	Boot failure
			Off	---
	RUN	Green	On	Communication module is operational
			Blinking	---
			Off	Communication module is not operational
	STA1	Red	On	System error; watchdog timeout
			Blinking	
			Off	No system error
	STA2	Red	On	No connection; no configuration
			Blinking	No data exchange
			Off	No bus error, communication is running
	STA1	Yellow	Blinking (synchronously)	No production data available, no bus communication possible.
	STA2	Yellow		
LED state during firmware update	STA1	Green	Blinking (synchronously)	Firmware file transfers during communication module firmware update.
	STA2	Red		
	STA1	Green	Blinking (alternately)	Communication module writes the firmware file to the internal flash. Do not power off the PLC!
	STA2	Red		

The RJ45 Ethernet connector contains two LEDs showing the current Ethernet port connection state.

Table 410: Meaning of the diagnosis LEDs

LED		Color	State	Description
	PNIO1 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	PNIO1 LED "RX/TX"	Yellow	On	PROFINET device sends/receives frames
			Blinking	PROFINET device sends/receives frames
			Off	---
	PNIO2 LED "Link"	Green	On	Ethernet connection established
			Off	No Ethernet connection
	PNIO2 LED "RX/TX"	Yellow	On	PROFINET device sends/receives frames
			Blinking	PROFINET device sends/receives frames
			Off	---

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Protocol	PROFINET IO RT
Bus connection	2 RJ45 (PNIO1 and PNIO2), with integrated 2-port switch
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Bus length (segment length max.)	100 m
Indicators	5 LEDs
Usable terminal bases	All TB5xx ↬ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*
Supported alarm types	Process alarm, diagnostic alarm, return of SubModule, plug alarm, pull alarm
Current consumption from 24 V DC power supply at the terminal base of the CPU	Typ. 85 mA
Internal supply	Via the communication module interface of the terminal base

Parameter	Value
Setting of the I/O device identifier	With 2 rotary switches at the front side of the module
Weight	Ca. 170 g
Supported protocols	RTC - real-time cyclic protocol, class 1 RTA - real-time acyclic protocol DCP - discovery and configuration protocol *) CL-RPC - connectionless remote procedure call LLDP - link layer discovery protocol SNMP - simply network management protocol MRP - MRP Client
Acyclic services	PNIO read / write CM589-PNIO < FW 1.4.0: max. 1024 bytes CM589-PNIO ≥ FW 1.4.0: max. 8096 bytes CM589-PNIO-4: max. 8096 bytes
Total quantity of input and output data	CM589-PNIO < FW 1.4.0 (respectively for input and output): max. 1024 byte CM589-PNIO ≥ FW 1.4.0 (respectively for input and output): max. 1440 byte CM589-PNIO-4 (respectively for input and output): max. 1440 byte
Min. bus cycle	1 ms
Conformance class	CC B

*) Setting NameOfStation via service "DCP SET NameOfStation" is enabled only if rotary switches are adjusted to position "00".

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 172 900 R0011	CM589-PNIO, PROFINET communication module	Active
1SAP 372 900 R0011	CM589-PNIO-XC, PROFINET communication module, XC version	Active
1SAP 172 900 R0111	CM589-PNIO-4, PROFINET communication module	Active
1SAP 372 900 R0111	CM589-PNIO-4-XC, PROFINET communication module, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.5 Terminal units (AC500 standard)

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

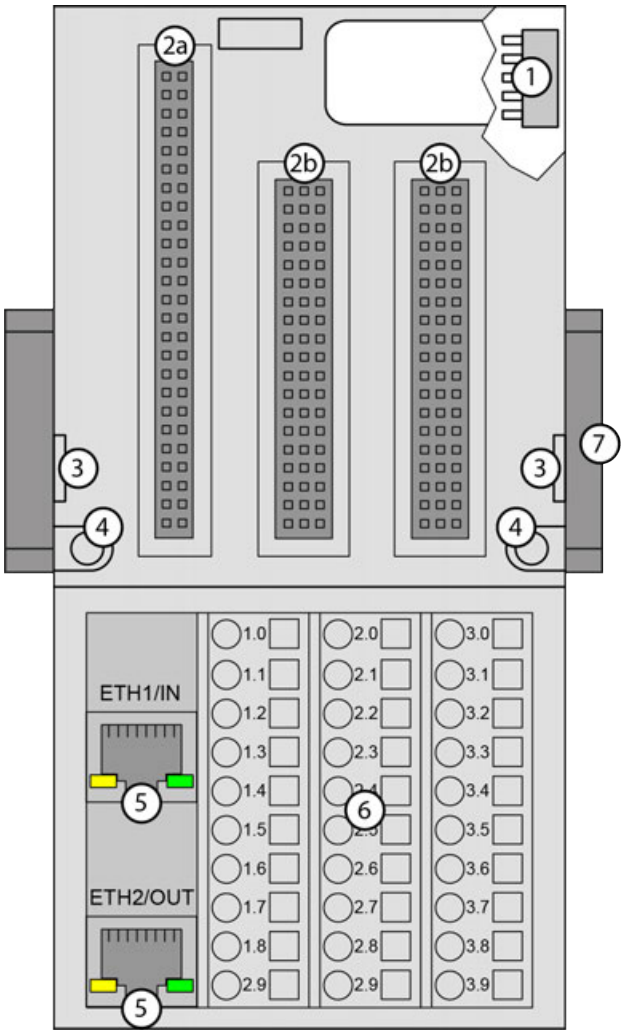
- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

Further information about hot swap: ↬ Chapter 1.6.4.1.7 “Hot swap” on page 5463.

1.6.2.5.1 TU507-ETH and TU508-ETH for Ethernet communication interface modules

- TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals
- TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals
- TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version



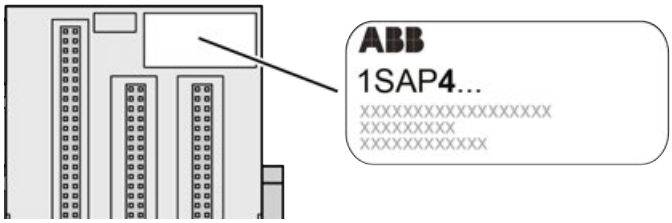
1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2x 25 pins) to connect the inserted Ethernet communication interface module
2b Plug (3x 19 pins) to connect the inserted Ethernet communication interface module
3 With a screwdriver, inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
4 2 holes for wall mounting
5 2 RJ45 interfaces with indication LEDs for connection with the Ethernet network
6 30 terminals for signals and process supply voltages (UP and UP3)
7 DIN rail

The Ethernet communication interface modules plug into the Ethernet terminal unit. When properly seated, they are secured with two mechanical locks. All the connections are made through the Ethernet terminal unit, which allows removal and replacement of the Ethernet communication interface modules without disturbing the wiring at the Ethernet terminal unit.

The Ethernet terminal units TU507-ETH and TU508-ETH are specifically designed for use with AC500/S500 Ethernet communication interface modules (e. g. CI501-PNIO).

XC version

XC = e**X**treme **C**onditions

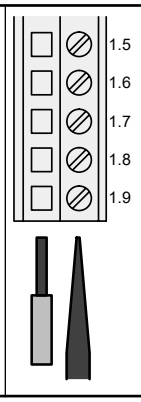
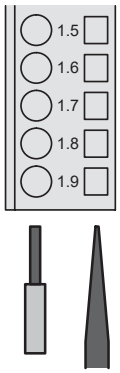


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

The assignment of the other terminals is dependent on the inserted communication interface module.

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of I/O channels per module	Max. 24 (depending on the inserted communication interface module)
Distribution of the channels into groups	3 groups of max. 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7), the allocation of the channels is given by the inserted Ethernet bus module
Network interface connector	2 RJ45, 8-pole
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP, UP3 and ZP)
Ethernet	10/100 base-TX or 100 base-TX (depending on CI5xx module plugged in), 2 RJ45 socket
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring-type terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

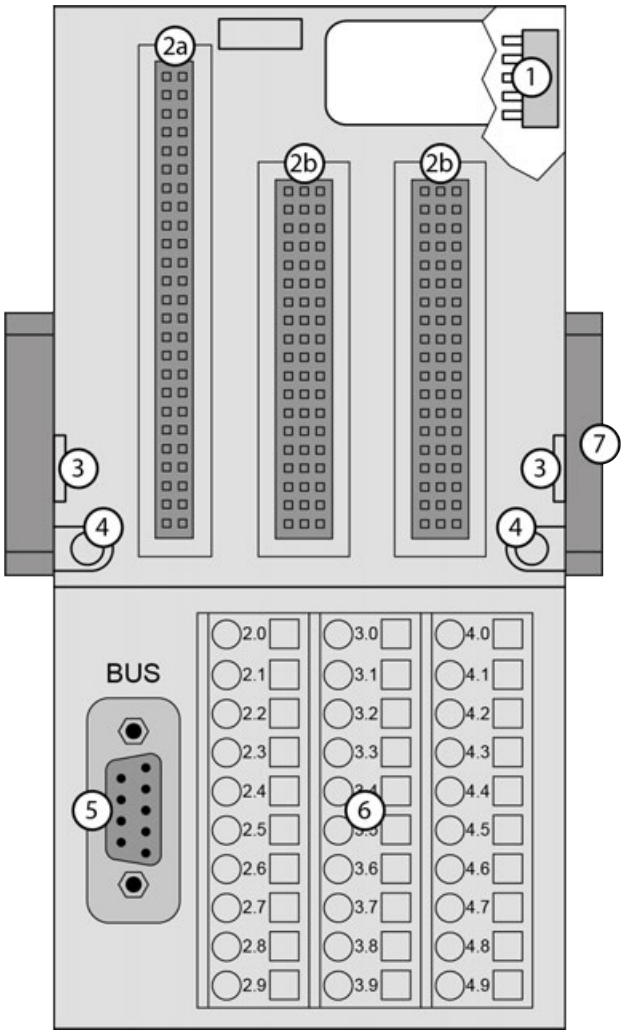
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 214 200 R0001	TU507-ETH, Ethernet terminal unit, 24 V DC, screw terminals	Active
1SAP 214 000 R0001	TU508-ETH, Ethernet terminal unit, 24 V DC, spring terminals	Active
1SAP 414 000 R0001	TU508-ETH-XC, Ethernet terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.5.2 TU509 and TU510 for communication interface modules

- TU509, terminal unit, 24 V DC, screw terminals
- TU510, terminal unit, 24 V DC, spring terminals
- TU510-XC, terminal unit, 24 V DC, spring terminals, XC version



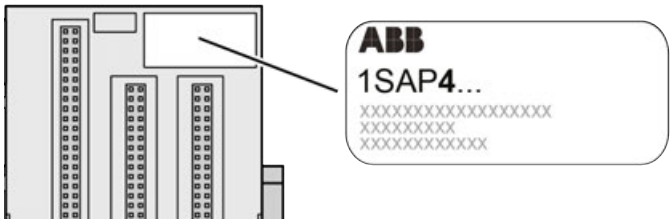
1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2 25 pins) to connect the inserted communication interface module
2b Plug (3 19 pins) to connect the inserted communication interface module
3 With a screwdriver, inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
4 2 holes for wall mounting
5 D-sub 9 (female) for connection with the PROFIBUS network
6 30 terminals for signals and process supply voltages (UP and UP3)
7 DIN rail

The communication interface modules plug into the terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the communication interface modules without disturbing the wiring at the terminal unit.

The terminal units TU509 and TU510 are specifically designed for use with AC500/S500 communication interface modules (e. g. CI451-DP).

XC version

XC = e**X**treme **C**onditions

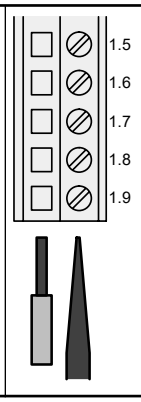
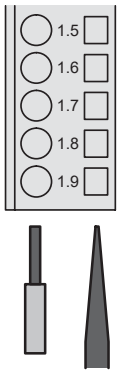


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC

Terminal 4.8: process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see communication interface modules for CANopen and PROFIBUS).

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of I/O channels per module	Max. 24 (depending on the inserted communication interface module)
Distribution of the channels into groups	3 groups of max. 8 channels each (2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted communication interface module
Network interface connector	9-pin D-sub connector, female
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP, UP3 and ZP)
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

Ordering data

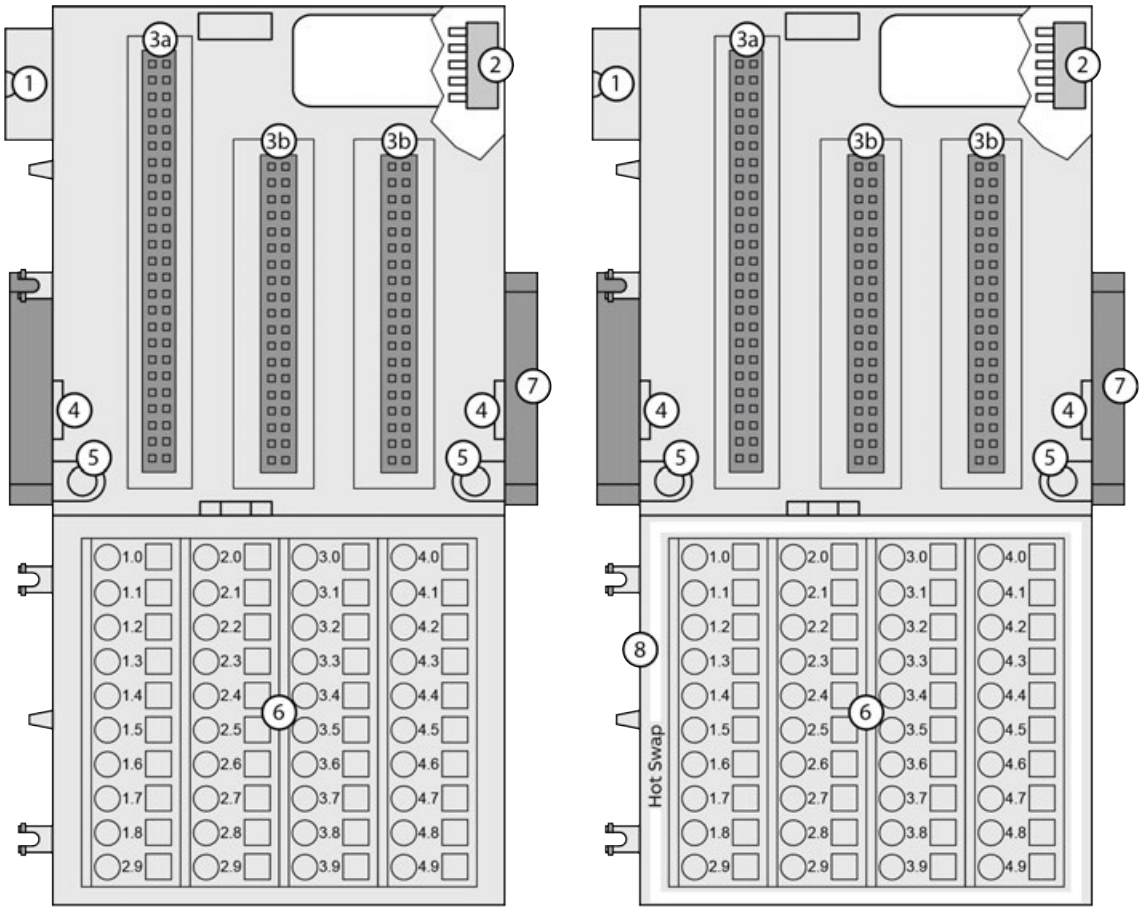
Part no.	Description	Product life cycle phase *)
1SAP 211 000 R0001	TU509, terminal unit, 24 V DC, screw terminals	Active
1SAP 210 800 R0001	TU510, terminal unit, 24 V DC, spring terminals	Active
1SAP 410 800 R0001	TU510-XC, terminal unit, 24 V DC, spring terminals, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

1.6.2.5.3 TU515, TU516, TU541 and TU542 for I/O modules

- TU515, I/O terminal unit, 24 V DC, screw terminals
- TU516, I/O terminal unit, 24 V DC, spring terminals
- TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
- TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
- TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version
- TU541, I/O terminal unit, 24 V DC, screw terminals
- TU542, I/O terminal unit, 24 V DC, spring terminals
- TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version
- TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals
- TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version

The input/output modules plug into the I/O terminal unit. When properly seated, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the I/O modules without disturbing the wiring at the terminal unit.



1 I/O bus (10 pins, male) to connect the previous terminal unit, the CPU terminal base or the communication interface module to the terminal unit
2 I/O bus (10 pins, female) to connect other terminal units
3a Plug (2 x 25 pins) to connect the inserted I/O modules

3b Plug (2 x 19 pins) to connect the inserted I/O modules
4 With a screwdriver inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
5 Holes for screw mounting
6 40 terminals for signals and process supply voltage
7 DIN rail
8 White border signifies hot swap capability of the terminal unit

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!

Electric shock due to negligent behavior during hot swapping!

To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

Hot swap

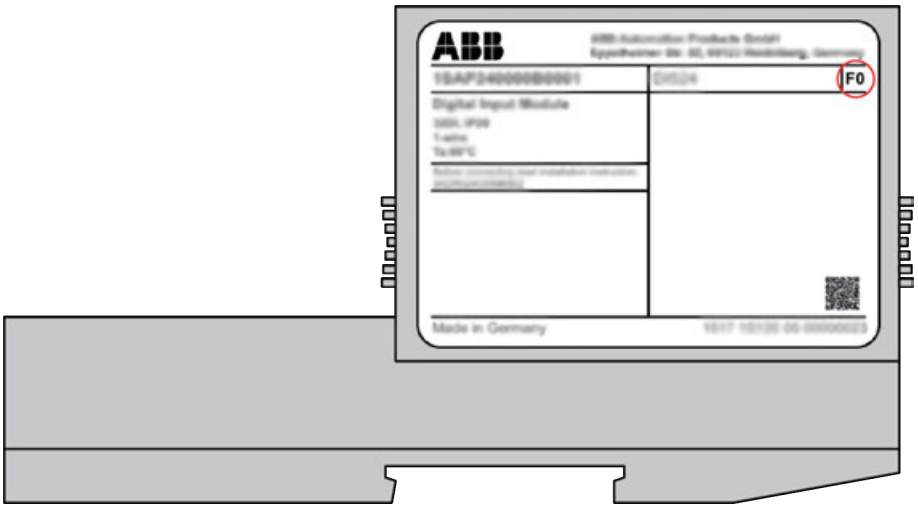
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

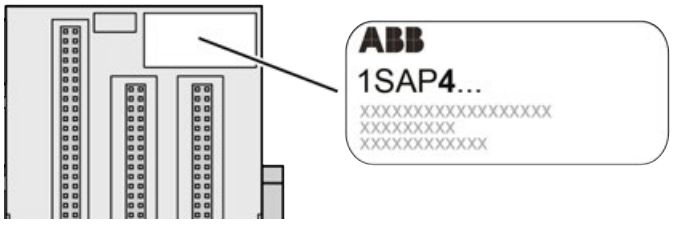
Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1

Device	Min. required device index for I/O module as of FW Version 3.0.14
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

XC version **XC** = e**X**treme **C**onditions

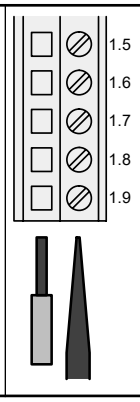
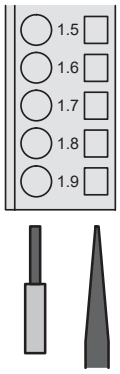


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (lable) identifies the XC version.*

Terminals

Screw terminals			Spring terminals		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

– *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
– *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
– *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The following terminals are used for connection of the process supply voltage.

	Terminals							
Type	1.8	2.8	3.8	4.8	1.9	2.9	3.9	4.9
TU515, TU516 and TU516-H	These terminals are internally connected with assignment: process supply voltage UP = +24 V DC				These terminals are internally connected with assignment: process supply voltage ZP = 0 V			
TU541, TU542 and TU542-H	These terminals are internally connected with assignment: process voltage UP = +24 V DC		Separate process supply voltage UP3 = +24 V DC	Separate process supply voltage UP4 = +24 V DC	These terminals are internally connected with assignment: process supply voltage ZP = 0 V		Separate process supply voltage ZP = 0 V	Separate process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see the description of the respective module used).

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of channels per module	Max. 32
Distribution of the channels into groups	4 groups of 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted I/O module
Rated voltage	24 V DC
Max. permitted total current	10 A, per separated process voltage terminal or for internal connection of process voltages
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

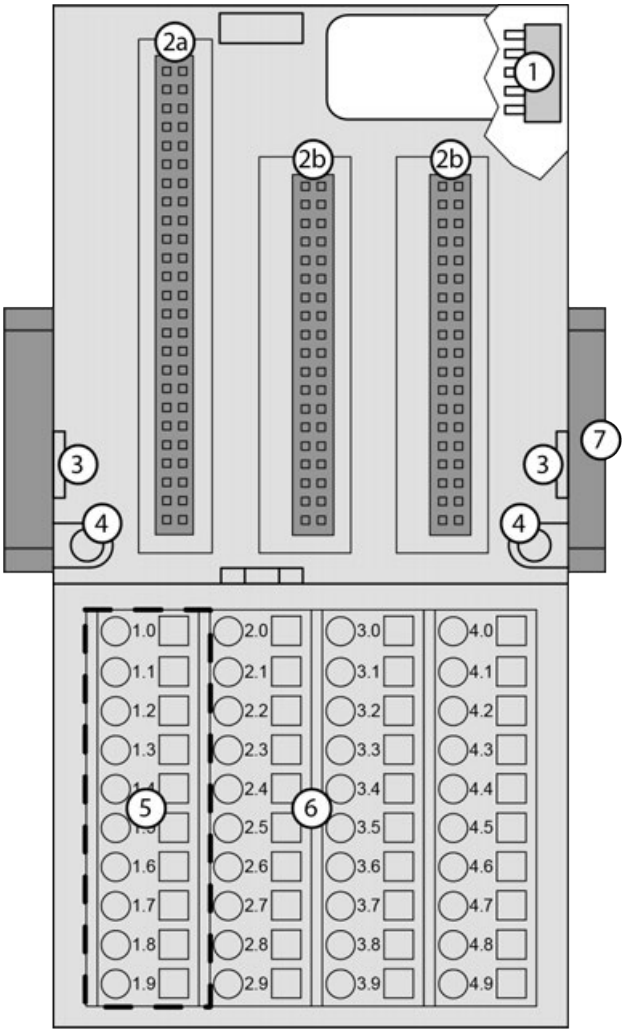
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 212 200 R0001	TU515, I/O terminal unit, 24 V DC, screw terminals	Active
1SAP 212 000 R0001	TU516, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 412 000 R0001	TU516-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active
1SAP 215 000 R0001	TU516-H, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version	Active
1SAP 415 000 R0001	TU516-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals	Active
1SAP 213 000 R0001	TU541, I/O terminal unit, 24 V DC, screw terminals	Active
1SAP 213 200 R0001	TU542, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 413 200 R0001	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active
1SAP 215 200 R0001	TU542-H, I/O terminal unit, hot swap, 24 V DC, spring terminals	Active
1SAP 415 200 R0001	TU542-H-XC, I/O terminal unit, hot swap, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.5.4 TU517 and TU518 for communication interface modules

- TU517, terminal unit, 24 V DC, screw terminals
- TU518, terminal unit, 24 V DC, spring terminals
- TU518-XC, terminal unit, 24 V DC, spring terminals, XC version



1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2 25 pins) to connect the inserted communication interface module
2b Plug (2 19 pins) to connect the inserted communication interface module
3 With a screwdriver, inserted in this place, the terminal unit and the adjacent I/O terminal unit can be shoved from each other
4 2 holes for wall mounting
5 10 terminals for connection with the bus system
6 30 terminals for signals and process supply voltages (UP and UP3)
7 DIN rail

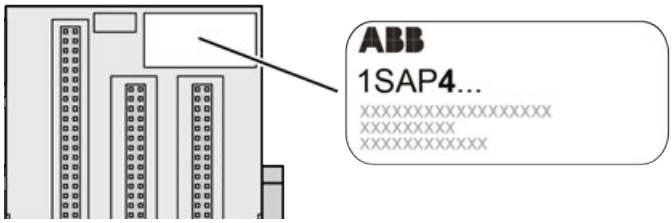
The communication interface modules plug into the terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the communication interface modules without disturbing the wiring at the terminal unit.

The terminal units TU517 and TU518 are specifically designed for use with AC500/S500 communication interface modules (e. g. CI581-CN, CI541-DP):

- CANopen communication interface modules
- DeviceNet modules
- PROFIBUS DP communication interface modules

XC version

XC = e**X**treme **C**onditions

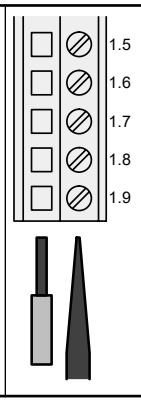
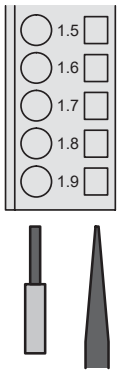


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted communication interface module:

- Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC
- Terminal 4.8: process supply voltage UP3 = +24 V DC
- Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see communication interface modules for CANopen and PROFIBUS).

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of I/O channels per module	Max. 24 (depending on the inserted communication interface module)
Distribution of the channels into groups	3 groups of max. 8 channels each (2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted communication interface module
Network interface connector	10 screw or spring terminals (1.0...1.9)
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP, UP3 and ZP)
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

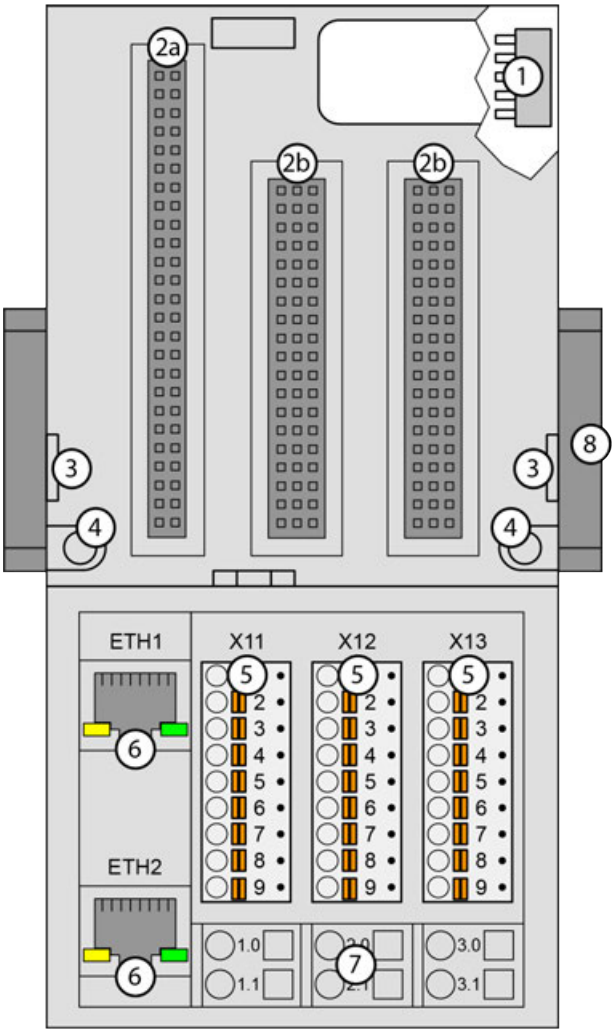
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 211 400 R0001	TU517, terminal unit, 24 V DC, screw terminals	Active
1SAP 211 200 R0001	TU518, terminal unit, 24 V DC, spring terminals	Active
1SAP 411 200 R0001	TU518-XC, terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.5.5 TU520-ETH for PROFINET communication interface modules

- TU520-ETH, 2 RJ45 interfaces for connection to PROFIBUS network, 3 removable connectors for bus systems
- TU520-ETH-XC, 2 RJ45 interfaces for connection to PROFIBUS network, 3 removable connectors for bus systems, XC version



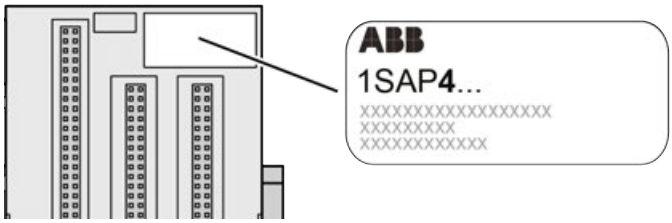
1 I/O bus (10 pins, female) to connect the first terminal unit
2a Plug (2 25 pins) to connect the inserted PROFINET communication interface module
2b Plug (3 19 pins) to connect the inserted PROFINET communication interface module
3 With a screwdriver, inserted in this place, the PROFINET I/O terminal unit and the adjacent I/O terminal unit can be shoved from each other
4 2 holes for wall mounting
5 3 removable connectors to connect the subordinated bus systems
6 2 RJ45 interfaces with indication LEDs for connection with the PROFINET network
7 6 spring terminals for process supply voltage (UP)
8 DIN rail

The PROFINET communication interface modules plug into the PROFINET IO terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the PROFINET IO terminal unit, which allows removal and replacement of the communication interface modules without disturbing the wiring at the PROFINET IO terminal unit.

The PROFINET IO terminal unit TU520-ETH are specifically designed for use with AC500/S500 PROFINET communication interface modules (e. g. CI504-PNIO, CI506-PNIO).

XC version

XC = e**X**treme **C**onditions



Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

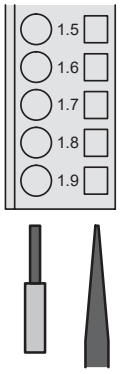
NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Spring terminals

Conductor		Screwdriver (opens terminal)

For information about wiring specifications see the description for the terminal unit Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.

For a detailed description of the mounting, disassembly and connection of the terminal units and the I/O modules, please refer to the "System Assembly, Construction and Connection" chapter Chapter 1.6.3.6.3 “Mounting and demounting” on page 5325.

The terminals 1.0, 2.0, 3.0, 1.1, 2.1 and 3.1 are electrically interconnected within the PROFINET IO terminal unit and always have the same assignment, irrespective of the inserted PROFINET communication interface module:

- Terminals 1.0, 2.0 and 3.0: process supply voltage UP = +24 V DC
- Terminals 1.1, 2.1 and 3.1: process supply voltage ZP = 0 V

The assignment of the bus system terminals depends on the inserted PROFINET communication interface module (see Ethernet communication interface modules overview).

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Ethernet	10/100 base-TX or 100 base-TX (depending on the plugged CI5xx module), 2 RJ45 socket
Number of bus system connectors	3 (the type of bus system depends on the PROFINET IO communication interface module)
Rated voltage	24 V DC
Max. permitted total current	10 A via the supply terminals (UP and ZP)
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

Ordering data

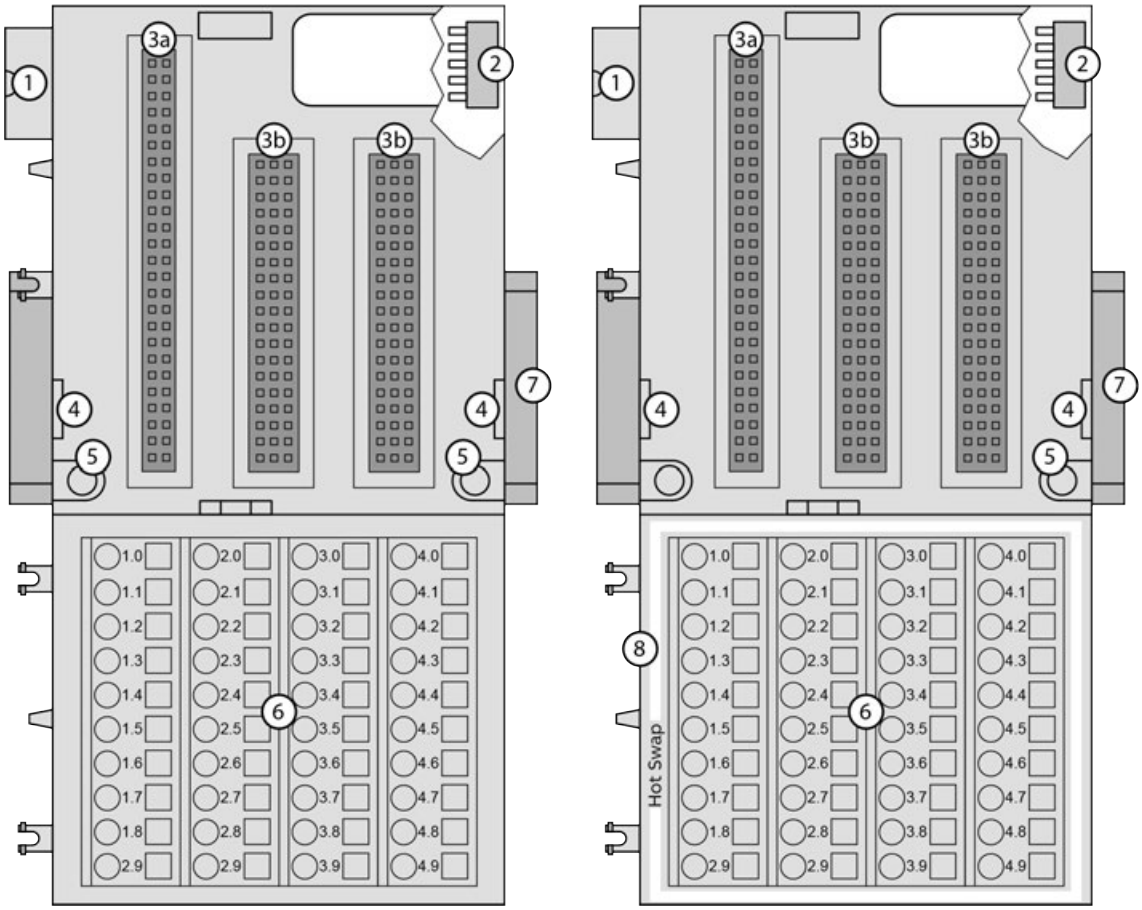
Part no.	Description	Product life cycle phase *)
1SAP 214 400 R0001	TU520-ETH, PROFINET I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 414 400 R0001	TU520-ETH-XC, PROFINET I/O terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.5.6 TU531 and TU532 for I/O modules

- TU531, I/O terminal unit, 230 V AC, screw terminals
- TU532, I/O terminal unit, 230 V AC, spring terminals

- TU532-XC, I/O terminal unit, 230 V AC, spring terminals, XC version
- TU532-H, I/O terminal unit, hot swap, 230 V AC, spring terminals
- TU532-H-XC, I/O terminal unit, hot swap, 230 V AC, spring terminals, XC version



1 I/O bus (10 pins, male) to connect the previous terminal unit, the CPU terminal base or the communication interface module to the terminal unit
2 I/O bus (10 pins, female) to connect other terminal units
3a Plug (2 x 25 pins) to connect the inserted I/O modules
3b Plug (3 x 19 pins) to connect the inserted I/O modules
4 With a screwdriver inserted in this place, the terminal unit and the adjacent I/O terminal unit can be shoved from each other
5 Holes for screw mounting
6 40 terminals for signals and process supply voltage
7 DIN rail
8 White border signifies hot swap capability of the terminal unit

The input/output modules (I/O modules) plug into the I/O terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the I/O modules without disturbing the wiring at the terminal unit.

The terminal units TU531 and TU532 are specifically designed for use with AC500/S500 I/O modules that incorporate 115-230 V AC inputs and/or 230 V AC relay outputs.

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!

Electric shock due to negligent behavior during hot swapping!

To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

Hot swap

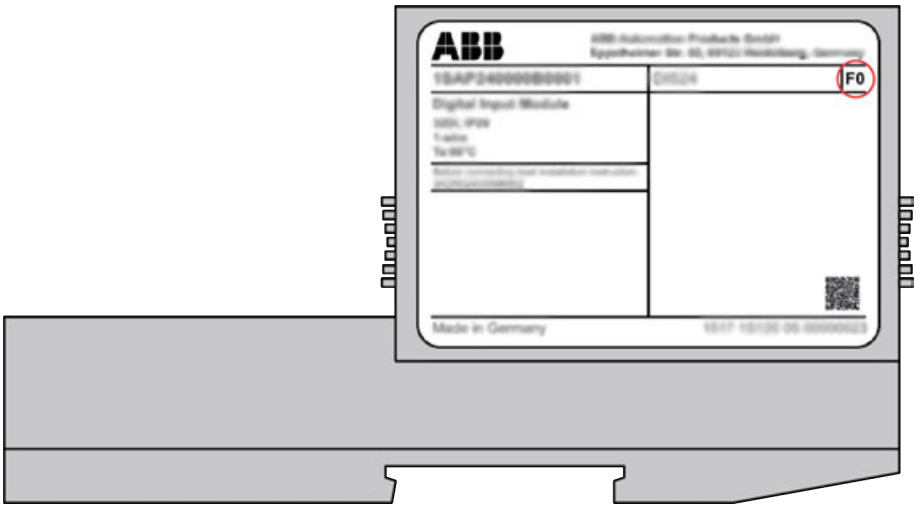
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx**-H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

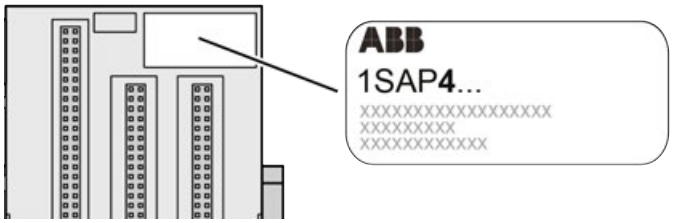
Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1

Device	Min. required device index for I/O module as of FW Version 3.0.14
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

XC version **XC** = e**X**treme **C**onditions

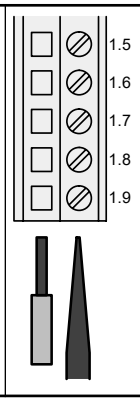
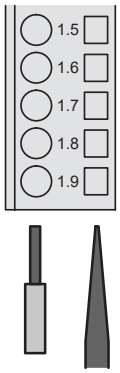


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			Spring terminals		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

– *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
– *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
– *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the terminal unit and always have the same assignment, independent of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V

The assignment of the other terminals depends on the inserted communication interface module (see the description of the respective module used).

The supply voltage of 24 V DC for the module's circuitry comes from the I/O expansion bus (I/O bus).

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Number of channels per module		32
Distribution of the channels into groups		4 groups of 8 channels each (1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted I/O module
Terminals 1.8...4.8 and 1.9...4.9		
	Max. voltage	30 V DC
	Max. permitted total current	10 A
Terminals 1.0...1.7, 2.0...2.7, 3.0...3.7, 4.0...4.7		
	Max. voltage	300 V AC [1])
	Max. permitted current	3 A [2])
Grounding		Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals		Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals		Front terminal, conductor connection vertically with respect to the printed circuit board
Weight		200 g
Mounting position		Horizontal or vertical

[1]) Only when the voltage is not limited by the specification of the I/O channel or the supply input which is internally connected to the terminal.

[2]) The terminals are connected to the electronic module via internal connectors (X22 (or 3b), X23 (or 3b), X32, X33 and X34). The current per terminal is limited by the permitted current of these connectors.

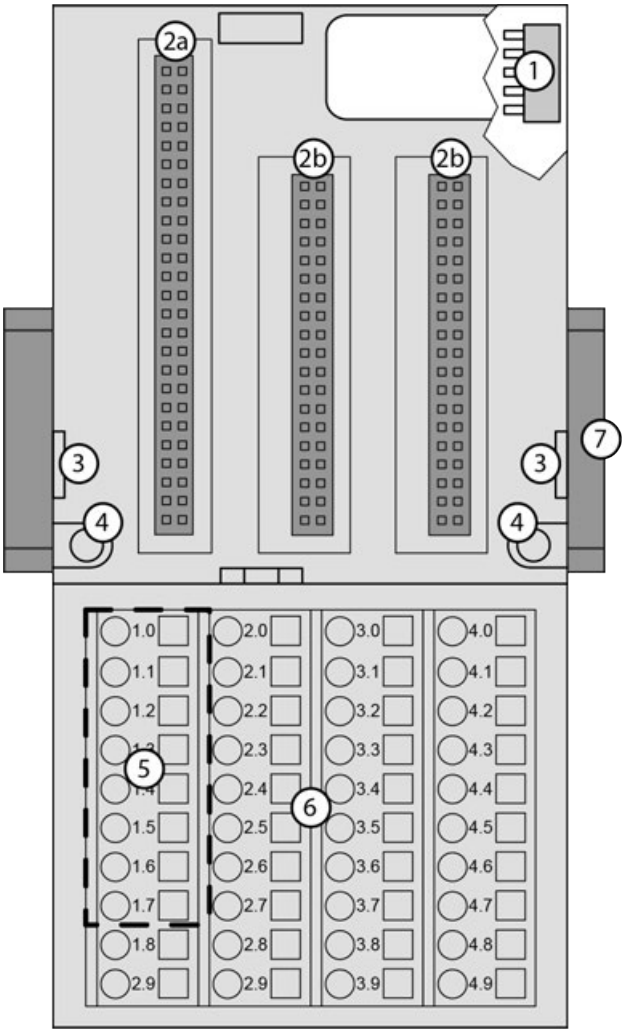
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 217 200 R0001	TU531, terminal unit, 230 V AC, relays, screw terminals	Active
1SAP 217 000 R0001	TU532, terminal unit, 230 V AC, relays, spring terminals	Active
1SAP 417 000 R0001	TU532-XC, terminal unit, 230 V AC, relays, spring terminals, XC version	Active
1SAP 215 100 R0001	TU532-H, terminal unit, hot swap, 230 V AC, relays, spring terminals	Active
1SAP 415 100 R0001	TU532-H-XC, terminal unit, hot swap, 230 V AC, relays, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

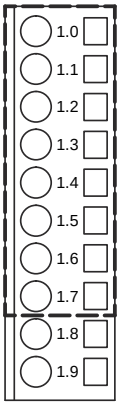
1.6.2.5.7 TU551-CS31 and TU552-CS31 for CS31 communication interface modules

- TU551-CS31, CS31 bus terminal unit, 24 V DC, screw terminals
- TU552-CS31, CS31 bus terminal unit, 24 V DC, spring terminals
- TU552-CS31-XC, CS31 bus terminal unit, 24 V DC, spring terminals, XC version



1 I/O bus (10 pins, female) to connect other terminal units
2a Plug (2 25 pins) to connect the inserted I/O modules
2b Plug (2 19 pins) to connect the inserted I/O modules
3 With a screwdriver inserted in this place, the terminal unit and the adjacent terminal unit can be shoved from each other
4 2 holes for wall mounting
5 CS31 bus interface
6 30 terminals for signals and process supply voltage
7 DIN rail

PIN assignment for bus interface



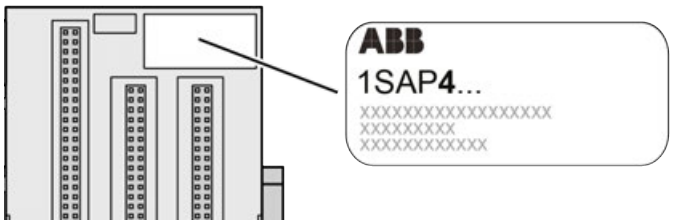
R1	Resistor + (end-of-line)
R2	Resistor - (end-of-line)
B1	CS31 bus +
B2	CS31 bus -
FE	Functional earth
B1	CS31 bus +
B2	CS31 bus -
FE	Functional earth
UP	24 V DC process voltage
ZP	0 V process voltage

The CS31 communication interface modules plug into the terminal unit. When properly plugged-in, they are secured with two mechanical locks. All the connections are established via the terminal unit, which allows removal and replacement of the CS31 communication interface modules without disturbing the wiring at the terminal unit.

The terminal units TU551-CS31 and TU552-CS31 are specifically designed for use with S500 CS31 communication interface modules that incorporate only 24 V DC inputs/outputs or interface signals.

XC version

XC = e**X**treme **C**onditions

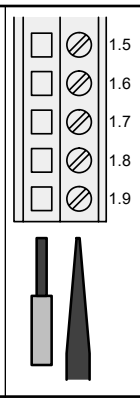
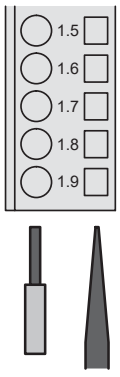


Extreme conditions

Terminal units for use in extreme ambient conditions have no ❄ sign for XC version.

*The figure **4** in the Part no. 1SAP**4**... (label) identifies the XC version.*

Terminals

Screw terminals			**Spring terminals**		
Conductor		Screwdriver	Conductor		Screwdriver (opens terminal)

- *For information about wiring specifications see the description of the terminal units ↳ Chapter 1.6.3.6.4.3 “Terminals at the terminal unit” on page 5338.*
- *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*
- *For information about mechanical dimensions, please refer to the Mechanical dimensions S500 chapter ↳ Chapter 1.6.3.6.2.3 “Mechanical dimensions S500” on page 5323*

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process voltage ZP = 0 V

The assignment of the other terminals depends on the inserted CS31 bus module.

The supply voltage of 24 V DC for the module's circuitry comes from ZP and UP.

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Number of channels per module	24
Distribution of the channels into groups	3 groups of 8 channels each (2.0...2.7, 3.0...3.7, 4.0...4.7), the allocation of the channels is given by the inserted CS31 communication interface module
CS31 field bus connector	Terminals 1.0 to 1.7
Rated voltage	24 V DC
Max. permitted total current	10 A (between the terminals 1.8...4.8 and 1.9...4.9)
Grounding	Direct connection to the grounded DIN rail or via the screws with wall mounting
Screw terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Spring terminals	Front terminal, conductor connection vertically with respect to the printed circuit board
Weight	200 g
Mounting position	Horizontal or vertical

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 210 600 R0001	TU551-CS31, CS31 bus terminal unit, 24 V DC, screw terminals	Active
1SAP 210 400 R0001	TU552-CS31, CS31 bus terminal unit, 24 V DC, spring terminals	Active
1SAP 410 400 R0001	TU552-CS31-XC, CS31 bus terminal unit, 24 V DC, spring terminals, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.6 I/O modules

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

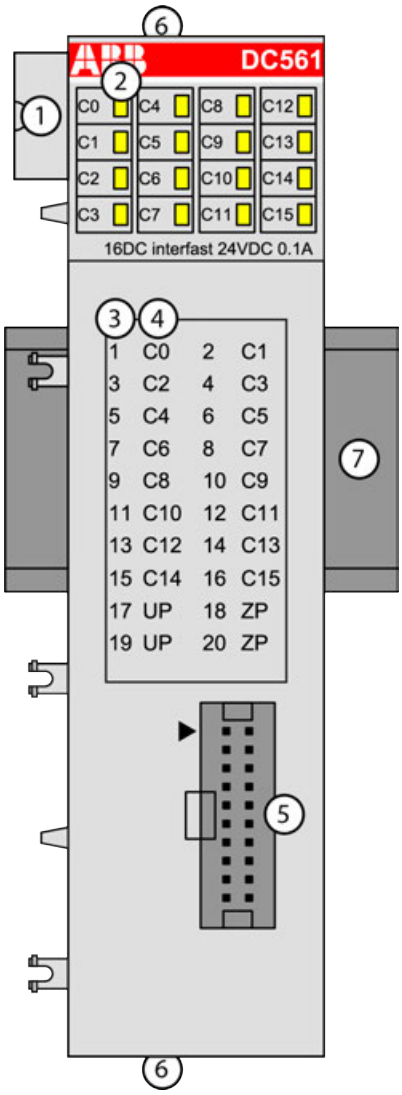
Further information about hot swap: Chapter 1.6.4.1.7 “Hot swap” on page 5463.

1.6.2.6.1 Digital I/O modules

S500-eCo

DC561 - Digital input/output module

- 16 configurable digital inputs/outputs 24 V DC,
- Connection via Interfast
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the states of the inputs/outputs C0 to C15
3 Terminal number
4 Allocation of signal name
5 Interfast connector (20-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The digital I/O module DC561 can be connected to the following devices via the I/O bus connector:

- S500 communication interface modules (e. g. CI501-PNIO, CI541-DP, CI581-CN)
- AC500 CPUs
- other AC500 I/O modules

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

The module contains 16 digital channels in 1 group, each channel can be used as a digital 24 V DC input or 24 V DC output.

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

Functionality

Parameter	Value
Digital inputs	Max. 16 (24 V DC), can be used as sink inputs
Digital outputs	Max. 16 (transistor outputs 24 V DC, max. 0.1 A)
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

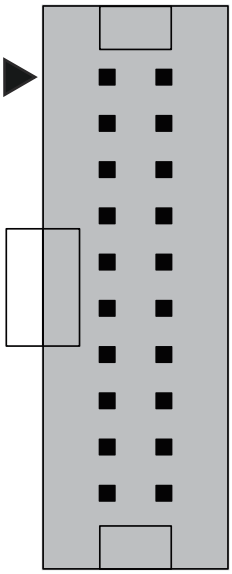
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is established out by using the 20-pin Interfast connector. For further information, refer to the Interfast documentation.

The assignment of the terminals:

Table 411: Assignment of the terminals for DC561

	PIN	Signal	Description
	1	C0	Input/output signal C0
	2	C1	Input/output signal C1
	3	C2	Input/output signal C2
	4	C3	Input/output signal C3
	5	C4	Input/output signal C4
	6	C5	Input/output signal C5
	7	C6	Input/output signal C6
	8	C7	Input/output signal C7
	9	C8	Input/output signal C8
	10	C9	Input/output signal C9
	11	C10	Input/output signal C10
	12	C11	Input/output signal C11
	13	C12	Input/output signal C12
	14	C13	Input/output signal C13
	15	C14	Input/output signal C14
	16	C15	Input/output signal C15
	17	UP	Process voltage UP +24 V DC
	18	ZP	Process voltage ZP 0 V DC
	19	UP	Process voltage UP +24 V DC
	20	ZP	Process voltage ZP 0 V DC

The arrow located next to the Interfast connector marks terminal 1.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DC561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Process supply voltage must be connected to UP/ZP of the module. The inputs and UP/ZP must use the same power supply.

If DC561 with index A0 is used, the process supply voltage must stem from the same source as the power supply voltage of the CPU. The index consists of 1 letter, followed by 1 digit, and can be found on the type plate of the module next to the type designator "DC561".

The module provides several diagnosis functions *Chapter 1.6.2.6.1.1.1.6 “Diagnosis” on page 4130*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.2.6.1.1.1.7 “State LEDs” on page 4130*.

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6100 [1])	WORD	6100 0x17D4	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

Remarks:

[1])	With CS31 and addresses smaller than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0x25, 0x17, 0x00;

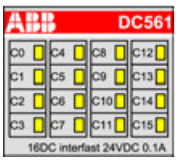
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error DI571							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs/outputs C0...C15	Digital input or digital output	Yellow	Input/output is OFF	Input/output is ON (the LEDs are only operating if the module's circuitry is supplied via the I/O bus)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process voltage UP		
	Connections	Terminals 17 and 19 for UP (+24 V DC); terminals 18 and 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption via UP terminal	10 mA + 0.1 A per output (max.)
	Max. ripple	5 %
	Inrush current	0.000001 A^2s
	Protection against reversed voltage	Yes
	Protection fuse on UP	Recommended; the outputs must be protected by an 1 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input/output group and the rest of the module
Isolated groups		1 group for 16 channels
Surge voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		On request
Input data length		2 bytes
Output data length		2 bytes
Weight		Ca. 115 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	16 configurable inputs (24 V DC)
Distribution of the channels into groups	1 (16 channels per group)
Connections of the channels C0 to C15	Terminals 1 to 16
Reference potential for the channels C0 to C15	Terminals 18 and 20 (negative pole of the process voltage, name ZP)

Parameter		Value
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered via the I/O bus.
Input type according to EN 61131-2		Type 1 sink
Input signal range		+24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		-3 V...+5 V
Ripple with signal 1		+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	> 2.5 mA
	Input voltage +30 V	< 8 mA
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Input delay (0->1 or 1->0)		Typ. 8 ms
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		16 configurable transistor outputs
Distribution of the channels into groups		1 (16 channels per group)
Connections of the channels C0 to C15		Terminals 1 to 16
Reference potential for the channels C0 to C15		Terminals 18 and 20 (negative pole of the process voltage, signal name ZP)
Common power supply voltage		Terminals 17 and 19 (positive pole of the process voltage, signal name UP)
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered via the I/O bus.
Way of operation		Non-latching type
Output voltage at signal 1		UP -0.3 V at max. current
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output current		
	Rated current per channel (max.)	0.1 A at UP 24 V DC
	Rated current per group (max.)	1.6 A

Parameter		Value
	Rated current (all channels together, max.)	1.6 A
	Lamp load (max.)	Not applicable
	Max. leakage current with signal 0	< 0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		1 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to load specification
Switching frequency		
	With inductive loads	Max. 0.5 Hz
Short-circuit-proof / overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC signals	Yes
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

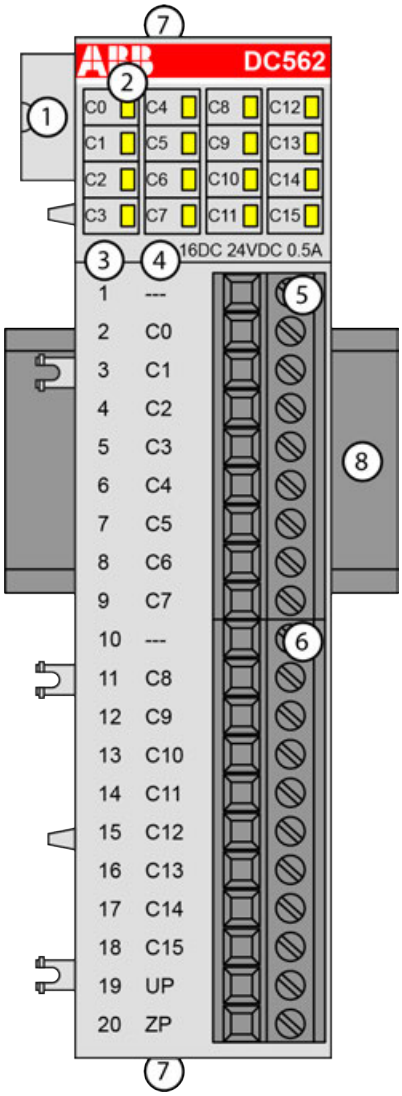
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2001	DC561, digital input/output module, 16 configurable inputs/outputs, transistor output, interfast connector	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC562 - Digital input/output module

- 16 configurable digital inputs/outputs in 1 group, 24 V DC
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the states of the inputs/outputs C0 to C15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input and output signals (9-pin)
6 Terminal block for input and output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs/outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs/outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs and outputs:

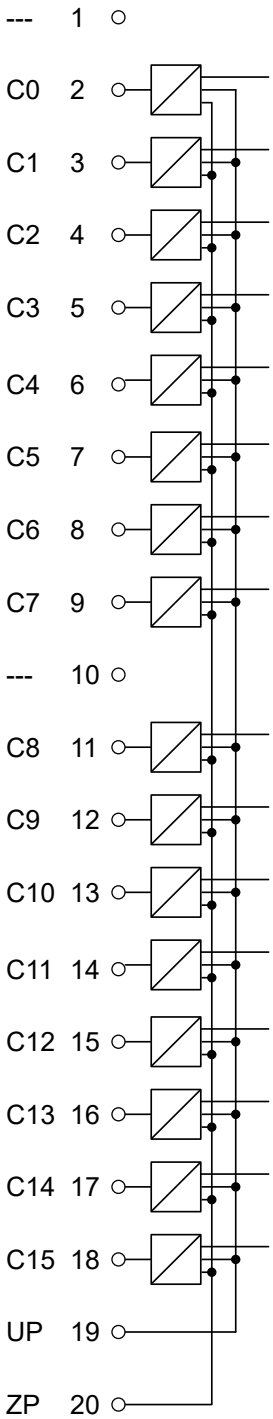


Table 412: Assignment of the terminals:

Terminal	Signal	Description
1	---	Reserved
2	C0	Input/output signal C0
3	C1	Input/output signal C1
4	C2	Input/output signal C2
5	C3	Input/output signal C3
6	C4	Input/output signal C4
7	C5	Input/output signal C5

Terminal	Signal	Description
8	C6	Input/output signal C6
9	C7	Input/output signal C7
10	---	Reserved
11	C8	Input/output signal C8
12	C9	Input/output signal C9
13	C10	Input/output signal C10
14	C11	Input/output signal C11
15	C12	Input/output signal C12
16	C13	Input/output signal C13
17	C14	Input/output signal C14
18	C15	Input/output signal C15
19	UP	Process voltage UP +24 V DC
20	ZP	Process voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DC562.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

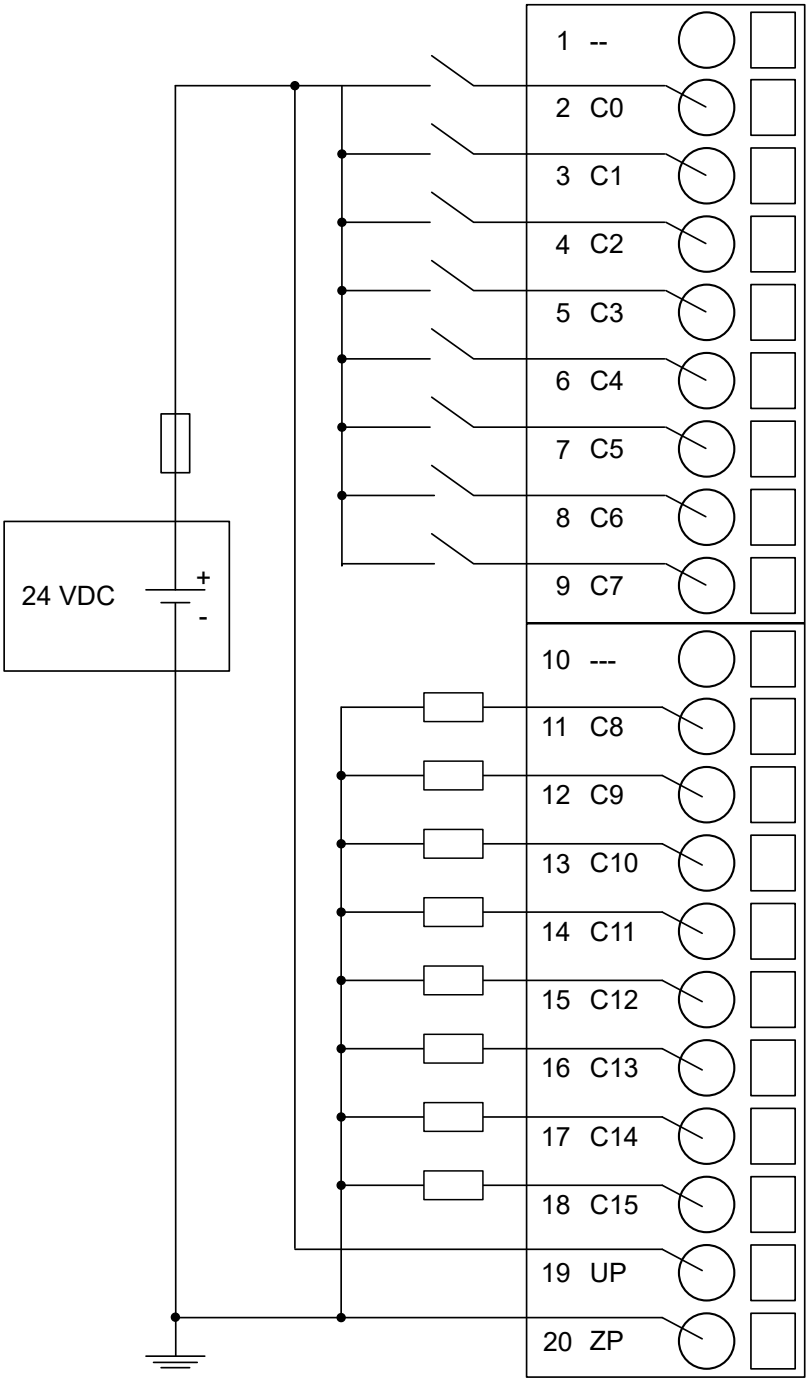
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Process supply voltage must be connected to UP/ZP of the module. The inputs and UP/ZP must use the same power supply.

The following figure shows the connection of the digital input/output module DC562:



In this connection example, the inputs/outputs C0...C7 are connected as inputs and the inputs/outputs C8...C15 are connected as outputs.

The module provides several diagnosis functions *Chapter 1.6.2.6.1.1.2.6 “Diagnosis” on page 4140*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.2.6.1.1.2.7 “State LEDs” on page 4140*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6155 [1])	WORD	6155 0x180B	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x06
Ext_User_Prm_Data_Const(0) =	0x18, 0x0C, 0x00, 0x02, 0x00, 0x00;

Diagnosis

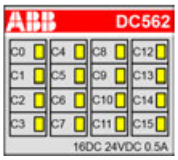
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block	

Class	Inter-face	Device	Module	Channel	Error-Identifier	Error message	Remedy
	[1])	[2])	[3])	[4])			
Module error DC562							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs/outputs C0...C15	Digital input or digital output	Yellow	Input/output is OFF	Input/output is ON (the LEDs are only operating if the module's circuitry is supplied via the I/O bus)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption via UP terminal	90 mA + 0.5 A per output (max.)
	Max. ripple	5 %
	Inrush current	0.000001 A^2s
	Protection against reversed voltage	Yes
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input/output group and the rest of the module
	Isolated groups	1 group for 16 channels
Surge voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		4.8 W
Input data length		2 bytes
Output data length		2 bytes
Weight		Ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	16 configurable inputs (24 V DC)
Distribution of the channels into groups	1 (16 channels per group)
Connections of the channels C0 to C15	Terminals 1 to 16
Reference potential for the channels C0 to C15	Terminal 20 (negative pole of the process voltage, name ZP)
Indication of the input signals	1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.
Input type according to EN 61131-2	Type 1 sink

Parameter		Value
Input signal range		+24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		-3 V...+5 V
Ripple with signal 1		+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	> 2.5 mA
	Input voltage +30 V	< 8 mA
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Input delay (0->1 or 1->0)		Typ. 8 ms
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		16 configurable transistor outputs
Distribution of the channels into groups		1 (16 channels per group)
Connections of the channels C0 to C15		Terminals 1 to 16
Reference potential for the channels C0 to C15		Terminal 20 (negative pole of the process voltage, signal name ZP)
Common power supply voltage		Terminal 19 (positive pole of the process voltage, signal name UP)
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.
Way of operation		Non-latching type
Output voltage at signal 1		UP -0.3 V at max. current
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	8 A
	Rated current (all channels together, max.)	8 A
	Lamp load (max.)	5 W

Parameter		Value
	Max. leakage current with signal 0	< 0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching frequency		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC signals	Yes
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

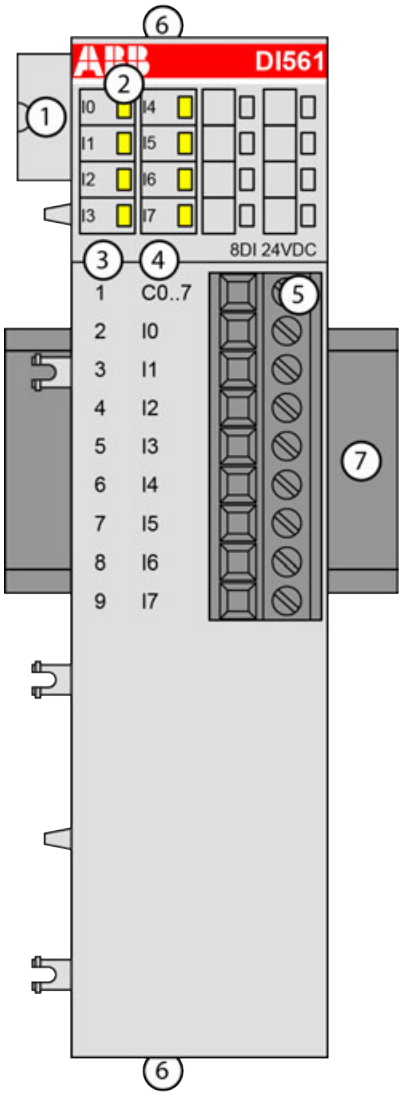
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 231 900 R0000	DC562, digital input/output module, 16 configurable inputs/outputs, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI561 - Digital input module

- 8 digital inputs 24 V DC / 24 V AC (I0 to I7) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using a removable 9-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs:

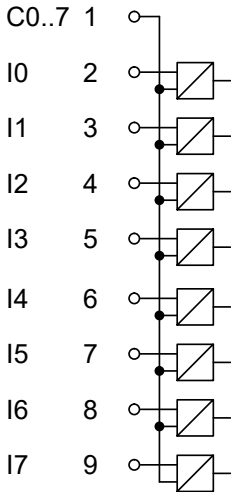


Table 413: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0

Terminal	Signal	Description
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI561.

An external power supply connection is not needed.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

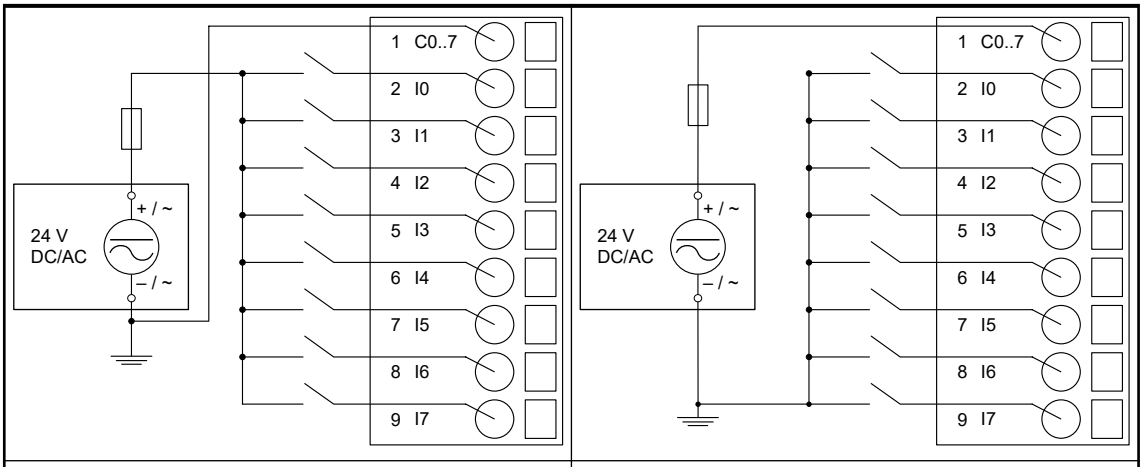
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital input module DI561:



Connection of DI561 - sink inputs	Connection of DI561 - source inputs

The module provides several diagnosis functions *Chapter 1.6.2.6.1.1.3.6 “Diagnosis” on page 4148*.

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.2.6.1.1.3.7 “State LEDs” on page 4149*.

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

> *If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.*

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6105 [1])	WORD	6105 0x17D9	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDA, 0x17, 0x00;

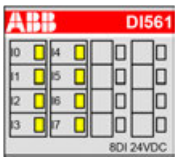
Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input group and the rest of the module
Isolated groups	1 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	1.6 W
Weight	Ca. 110 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter		Value		
Number of channels per module		8 inputs (24 V DC / 24 V AC)		
Distribution of the channels into groups		1 (8 channels per group)		
Connections of the channels I0 to I7		Terminals 2 to 9		
Reference potential for the channels I0 to I7		Terminal 1 (plus or negative pole of the process supply voltage, signal name C0..7)		
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.		
Monitoring point of input indicator		LED is part of the input circuitry		
Input type according to EN 61131-2		Type 1 source	Type 1 sink	Type 1 AC [1])
Input signal range		-24 V DC	+24 V DC	24 V AC 50/60 Hz
	Signal 0	-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
	Undefined signal	-15 V...-5 V	+5 V...+15 V	5 V AC...14 V AC
	Signal 1	-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel				
	Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
	Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
	Input voltage 14 V			Typ. 2.7 mA r.m.s.
	Input voltage 15 V	> 2.5 mA		
	Input voltage 27 V			Typ. 5.5 mA r.m.s.
	Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)		1 mA		Typ. 1 mA r.m.s.
Input delay (0->1 or 1->0)		Typ. 8 ms		
Input data length		1 byte		
Max. cable length				
	Shielded	500 m		
	Unshielded	300 m		

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

Refer to *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233* for details

Ordering data

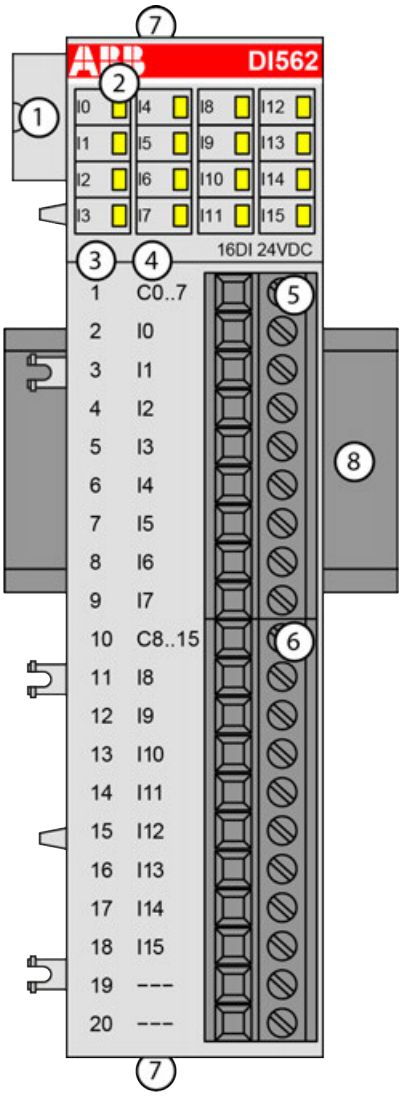
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2101	DI561, digital input module, 8 DI, 24 V DC / 24 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI562 - Digital input module

- 16 digital inputs 24 V DC / 24 V AC (I0 to I15) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the inputs I0 to I15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)

6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

The other electronic circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

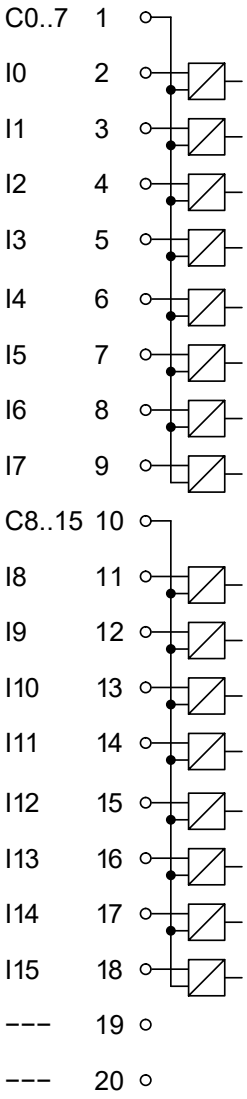
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw-type terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	C8...15	Input common for signals I8 to I15
11	I8	Input signal I8
12	I9	Input signal I9
13	I10	Input signal I10
14	I11	Input signal I11

Terminal	Signal	Description
15	I12	Input signal I12
16	I13	Input signal I13
17	I14	Input signal I14
18	I15	Input signal I15
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI562.

An external power supply connection is not needed.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.2.6.1.1.4.6 “Diagnosis” on page 4156*.

The digital inputs can be used as source inputs or as sink inputs.

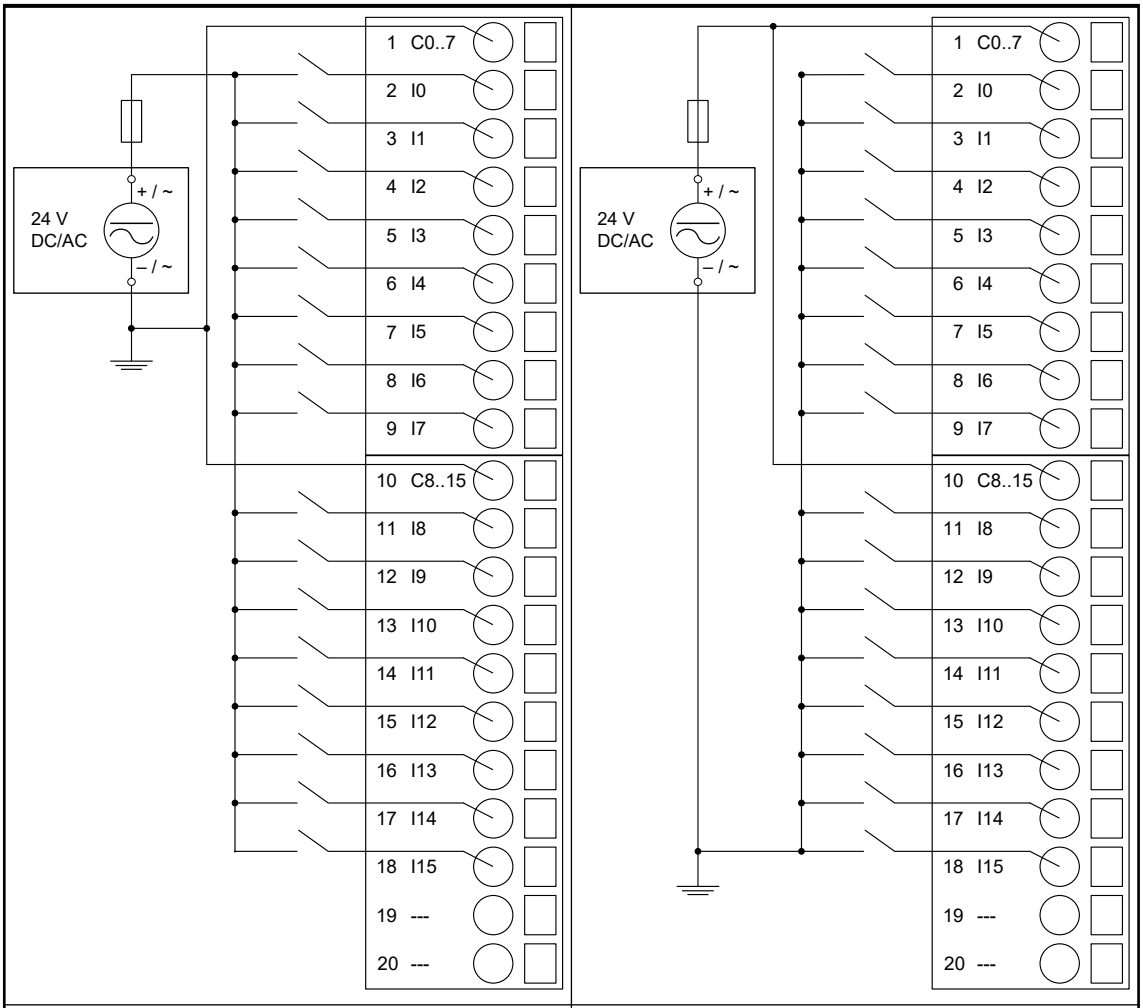
NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the digital input module DI562:



Connection of DI562 - sink inputs	Connection of DI562 - source inputs

The meaning of the LEDs is described in section State LEDs *Chapter 1.6.2.6.1.1.4.7 “State LEDs” on page 4157*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

> *If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.*

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6110 [1])	WORD	6110 0x17DE	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

Remarks:

[1])	With CS31 and addresses less than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDF, 0x17, 0x00;

Diagnosis

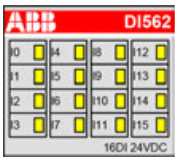
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<− Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error DI562								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Inputs I0...I15	Digital input	Yellow	Input is OFF	Input is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input groups and the rest of the module
Isolated groups	2 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	3.2 W
Weight	Ca. 115 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter	Value		
Number of channels per module	16 inputs (24 V DC / 24 V AC)		
Distribution of the channels into groups	2 (8 channels per group)		
Connections of the channels I0 to I7	Terminals 2 to 9		
Connections of the channels I8 to I15	Terminals 11 to 18		
Reference potential for the channels I0 to I7	Terminal 1 (positive or negative pole of the process supply voltage, signal name C0..7)		
Reference potential for the channels I8 to I15	Terminal 10 (positive or negative pole of the process supply voltage, signal name C8..15)		
Indication of the input signals	1 yellow LED per channel; the LED is ON when the input signal is high (signal 1). The module is powered through the I/O bus.		
Monitoring point of input indicator	LED is part of the input circuitry		
Input type according to EN 61131-2	Type 1 source	Type 1 sink	Type 1 AC [1])
Input signal range	-24 V DC	+24 V DC	24 V AC 50/60 Hz
Signal 0	-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
Undefined signal	-15 V...-5 V	+5 V...+15 V	5 V AC...14 V AC
Signal 1	-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel			
Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
Input voltage 14 V			Typ. 2.7 mA r.m.s.
Input voltage 15 V	> 2.5 mA		
Input voltage 27 V			Typ. 5.5 mA r.m.s.
Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)	1 mA		Typ. 1 mA r.m.s.
Input delay (0->1 or 1->0)	Typ. 8 ms		

Parameter		Value
Input data length		2 bytes
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

Refer to *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233* for details

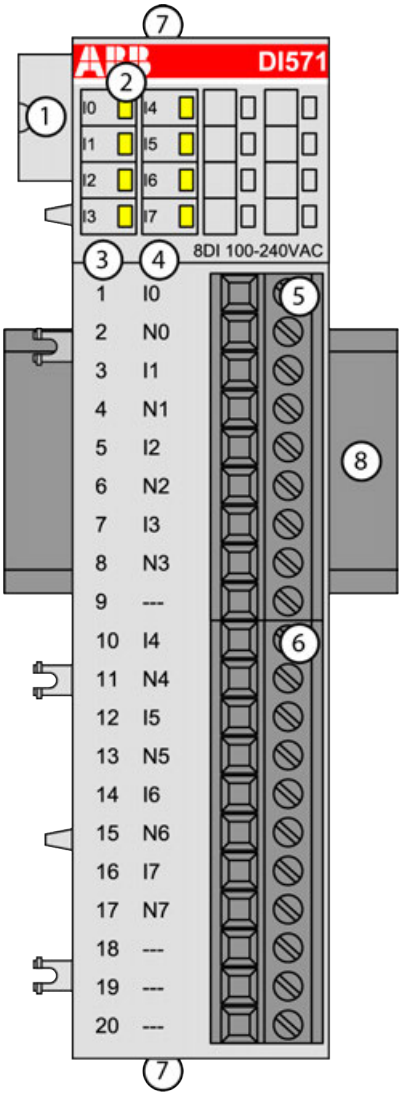
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2102	DI562, digital input module, 16 DI, 24 V DC / 24 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI571 - Digital input module

- 8 digital inputs 100-240 V AC (I0 to I7) in 8 groups
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs:

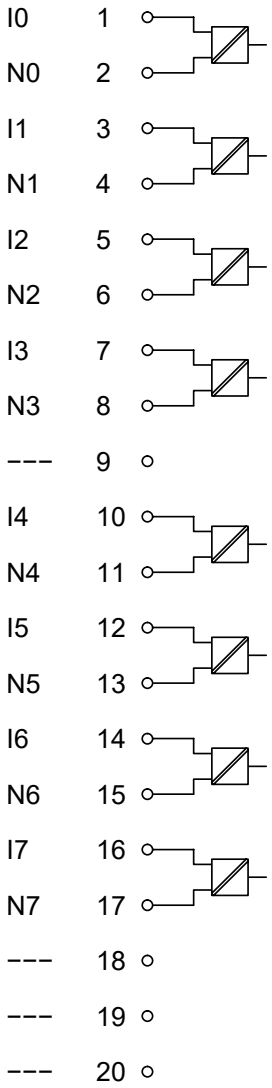


Table 414: Assignment of the terminals:

Terminal	Signal	Description
1	I0	Input signal I0
2	N0	Neutral conductor for the input signal I0
3	I1	Input signal I1
4	N1	Neutral conductor for the input signal I1
5	I2	Input signal I2
6	N2	Neutral conductor for the input signal I2
7	I3	Input signal I3
8	N3	Neutral conductor for the input signal I3
9	---	Reserved
10	I4	Input signal I4
11	N4	Neutral conductor for the input signal I4
12	I5	Input signal I5
13	N5	Neutral conductor for the input signal I5
14	I6	Input signal I6

Terminal	Signal	Description
15	N6	Neutral conductor for the input signal I6
16	I7	Input signal I7
17	N7	Neutral conductor for the input signal I7
18	---	Reserved
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI571.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

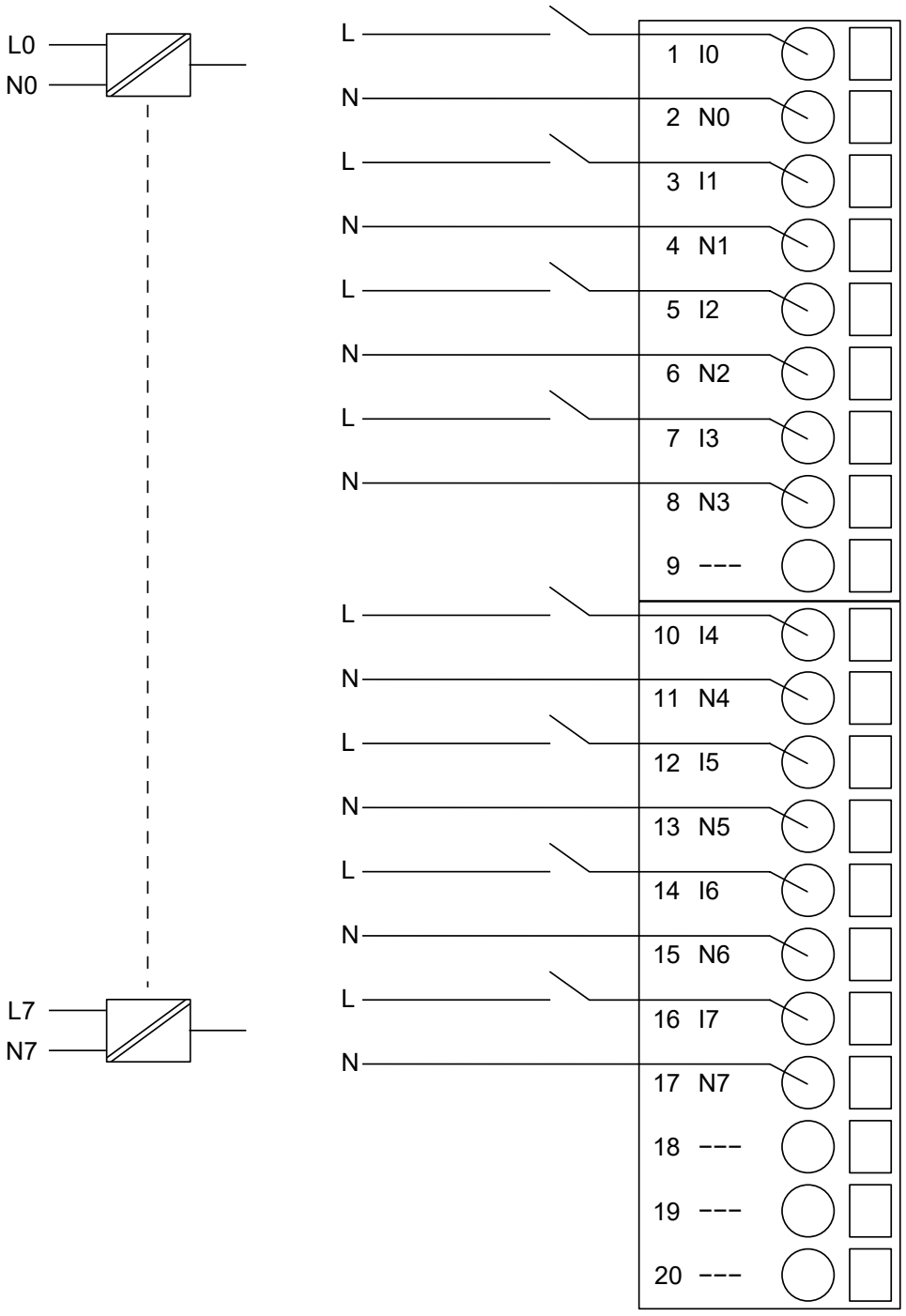
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital input module DI571:



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions ↳ *Chapter 1.6.2.6.1.1.5.7 “Diagnosis” on page 4166*.

The meaning of the LEDs is described in the section State LEDs ↳ *Chapter 1.6.2.6.1.1.5.8 “State LEDs” on page 4166*.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	0

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of the modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6115 [1])	WORD	6115 0x17E3	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) the module has no additional user-configurable parameters

[3]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xDF, 0x17, 0x00;

Diagnosis

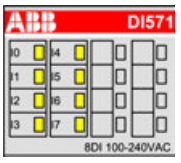
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block	

Class	Interface	Device	Module	Channel	Error Identifier	Error message	Remedy
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON (the input voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Galvanic isolation		Yes, between the channels and the rest of the module
	Isolated groups	8 (1 channel per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Max. power dissipation within the module		On request
Weight		Ca. 135 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8 AC inputs (100-240 V AC)
Distribution of the channels into groups		8 (1 channel per group)
Input voltage range		0 V AC..264 V AC (47 Hz...63 Hz)
Input current per channel (typically at 25 °C)		<5 mA (at 40 V AC) >6 mA (at 159 V AC, 50 Hz) >7 mA (at 159 V AC, 60 Hz)
Connections of the channels I0 to I7		Terminals 1, 3, 5, 7, 10, 12, 14, 16
Reference potential for the channels I0 to I7		Terminals 2, 4, 6, 8, 11, 13, 15, 17
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)
Input type according to EN 61131-2		Type 1
Input signal range		
	Signal 0 (max.)	20 V AC
	Undefined signal	20 V AC < U < 79 V AC
	Signal 1 (min.)	79 V AC
Input delay		
	Signal 0 -> 1	Typ. 15 ms
	Signal 1 -> 0	Typ. 30 ms
Input data length		1 byte
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Max. cable length		

Parameter		Value
	Shielded	500 m
	Unshielded	300 m

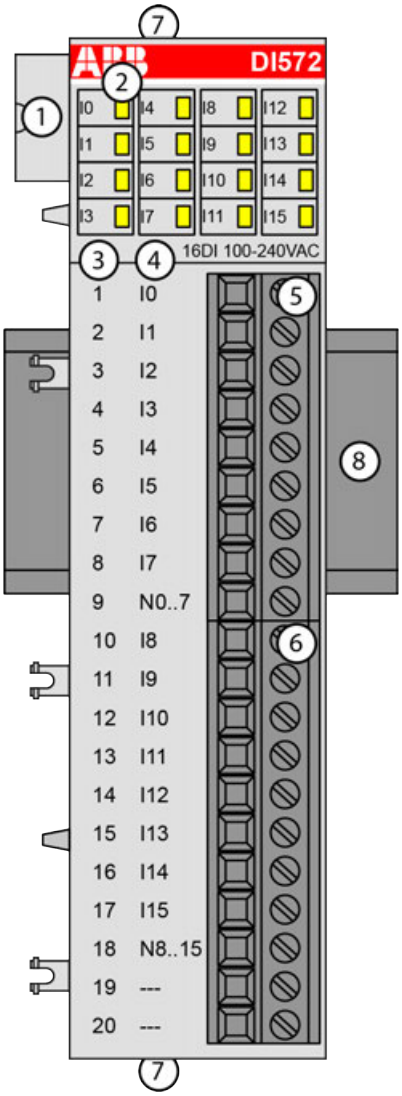
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2103	DI571, digital input module, 8 DI, 100 V AC...240 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DI572 - Digital input module

- 16 digital inputs 100-240 V AC (I0 to I15) in 2 groups
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the inputs I0 to I15
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

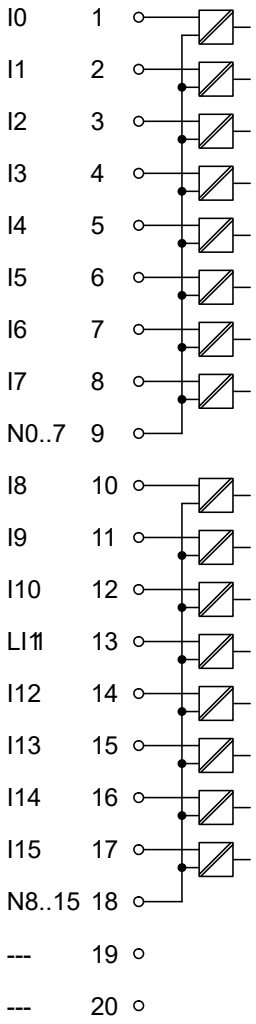


Fig. 827: Block diagram for the internal construction of the digital inputs.

Table 415: Assignment of the terminals

Terminal	Signal	Description
1	I0	Input signal I0
2	I1	Input signal I1
3	I2	Input signal I2
4	I3	Input signal I3
5	I4	Input signal I4
6	I5	Input signal I5
7	I6	Input signal I6
8	I7	Input signal I7
9	N0...7	Neutral conductor for the input signals I0...I7
10	I8	Input signal I8
11	I9	Input signal I9
12	I10	Input signal I10
13	I11	Input signal I11
14	I12	Input signal I12
15	I13	Input signal I13
16	I14	Input signal I14
17	I15	Input signal I15
18	N8...15	Neutral conductor for the input signals I8...I15
19	---	Reserved
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DI572.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

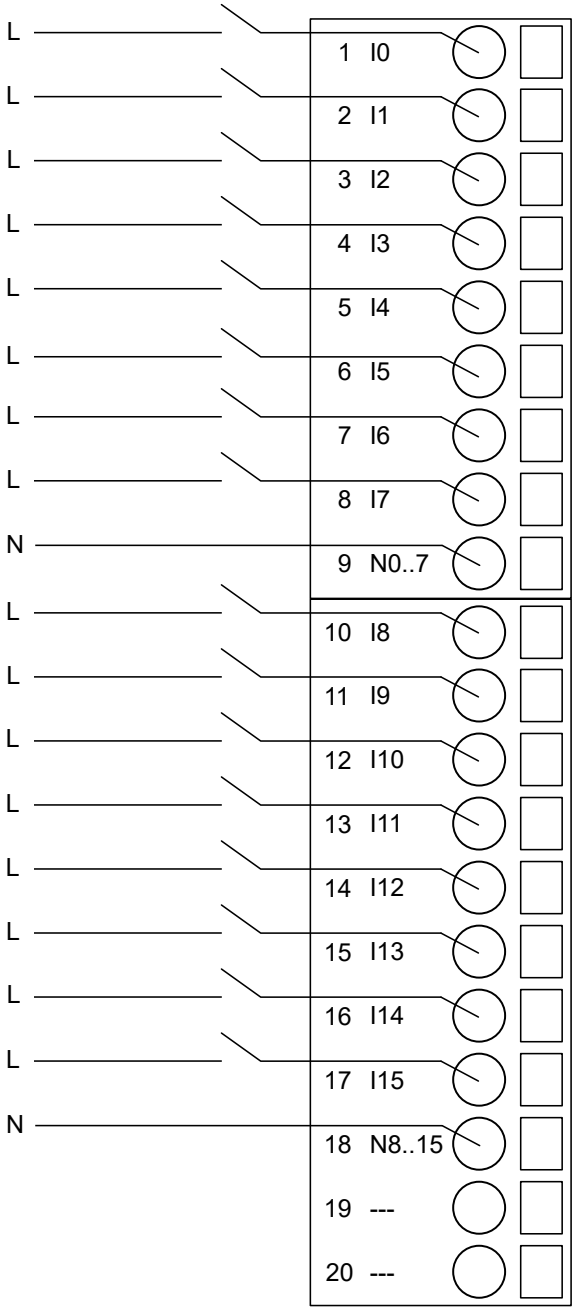
The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions *Chapter 1.6.2.6.1.1.6.6 “Diagnosis” on page 4175*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Parameter name	Value	Internal value	Data type of internal value	Default value	Min.	Max.	EDS Slot Index
Module ID	Internal	6160 [1])	WORD	6160 0x1810	0	65535	xx01 [2])
Ignore module	No	0	BYTE	No 0x00	-	-	-
	Yes	1					
Parameter length	Internal	3	BYTE	3	0	255	xx02 [2])
Input delay	20 ms	0	BYTE	20 ms 0x00	0	1	-
	100 ms	1					

[1]) With CS31 and addresses less than 70, the value is increased by 1.

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n).

GSD file:

Ext_Module_Prm_Data_Len =	7
Ext_User_Prm_Data_Const(0) =	0x18, 0x11, 0x00, 0x03, 0x00, 0x00, 0x00;

Diagnosis

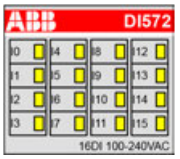
E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block	

Class	Interface	Device	Module	Channel	Error Identifier	Error message	Remedy
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				

Remarks:

Parameter	Remark
[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I15	Digital input	Yellow	Input is OFF	Input is ON (the input voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the input groups and the rest of the module
Isolated groups	2 (8 channels per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	6 W
Weight	Ca. 222 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		16 AC inputs (100-240 V AC)
Distribution of the channels into groups		2 (8 channels per group)
Input voltage range		0 V AC...264 V AC (47 Hz...63 Hz)
Input current per channel (typically at 25 °C)		< 3 mA (at 40 V AC) > 6 mA (at 164 V AC) > 8 mA (at 240 V AC)
Connections of the channels I0..I7		Terminals 1...8
Connections of the channels I8...I15		Terminals 10...17
Reference potential for the channels I0...I7		Terminal 9
Reference potential for the channels I8...I15		Terminal 18
Indication of the input signals		1 yellow LED per channel. The LED is on when the input signal is high (signal 1).
Input type according to EN 61131-2		Type 1
Input signal range		
	Signal 0 (max.)	40 V AC
	Undefined signal	40 V AC < U < 79 V AC
	Signal 1 (min.)	79 V AC
Input delay		
	Signal 0 -> 1	Typ. 24 ms
	Signal 1 -> 0	Typ. 24 ms
Input data length		2 bytes

Parameter		Value
Max. permissible leakage current (at 2-wire proximity switches)		1 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

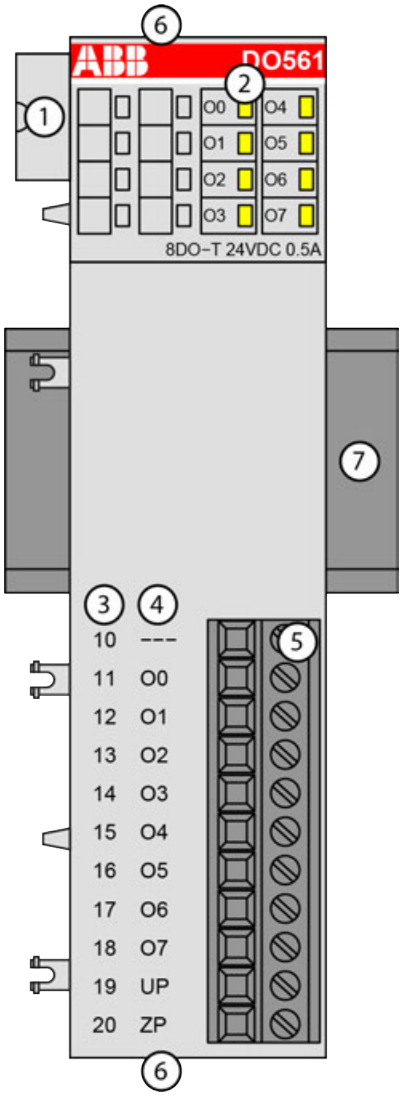
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 230 500 R0000	DI572, digital input module, 16 DI, 100 V AC...240 V AC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO561 - Digital output module

- 8 digital outputs 24 V DC (O0 to O7) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital outputs:

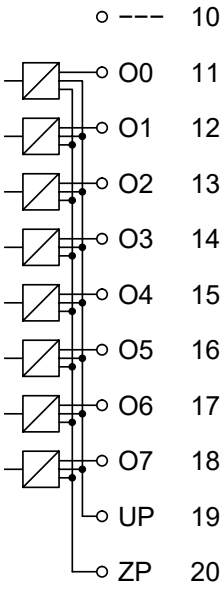


Table 416: Assignment of the terminals:

Terminals	Signal	Description
10	---	Reserved
11	O0	Output signal O0
12	O1	Output signal O1
13	O2	Output signal O2
14	O3	Output signal O3

Terminals	Signal	Description
15	O4	Output signal O4
16	O5	Output signal O5
17	O6	Output signal O6
18	O7	Output signal O7
19	UP	Process supply voltage UP +24 V DC
20	ZP	Process supply voltage ZP 0 V

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

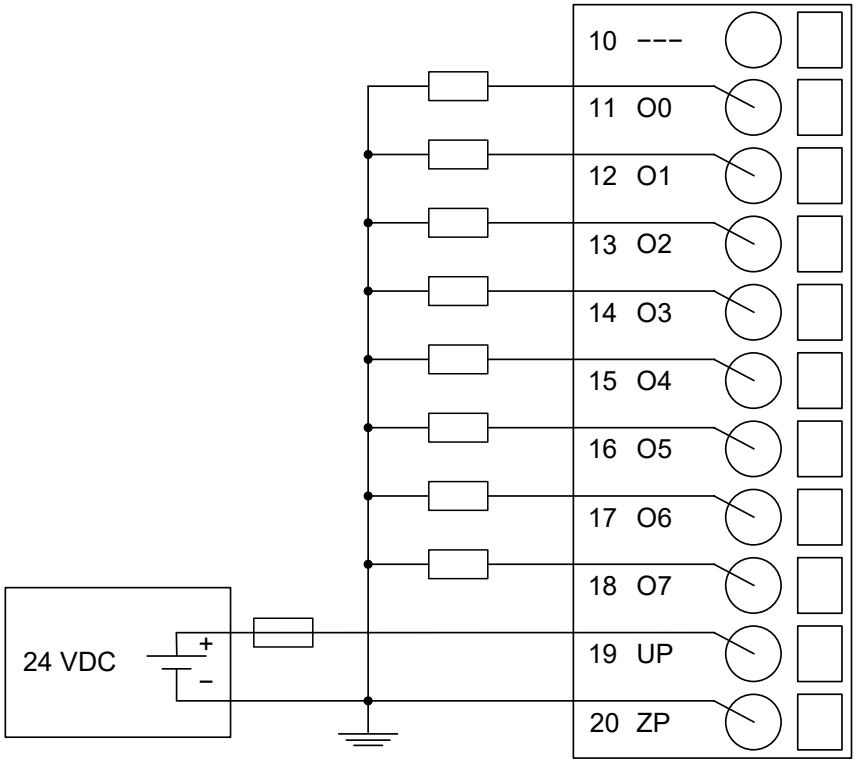
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital output module DO561:



NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.2.6.1.1.7.6 “Diagnosis” on page 4182*).

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.2.6.1.1.7.7 “State LEDs” on page 4183*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6120 [1])	WORD	6120 0x17E8	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xE9, 0x17, 0x00;

Diagnosis

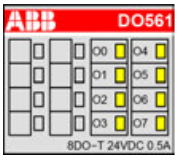
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error DO561								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	Interface	Device	Module	Channel	Error Identifier	Error message		Remedy
	[1])	[2])	[3])	[4])				
Module error DO561								
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		1 (8 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Power dissipation within the module (max.)		1.6 W
Weight		Ca. 115 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 (8 channels per group)
Connection of the channels O0 to O7	Terminals 11 to 18
Common power supply voltage	Terminal 19 (positive pole of the process voltage, signal name UP)
Reference potential for the channels O0 to O7	Terminal 20 (negative pole of the process voltage, signal name ZP)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus

Parameter		Value
Way of operation		Non-latching type
Min. output voltage at signal 1		20 V DC at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	4 A
	Lamp load (max.)	5 W
Max. leakage current with signal 0		0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

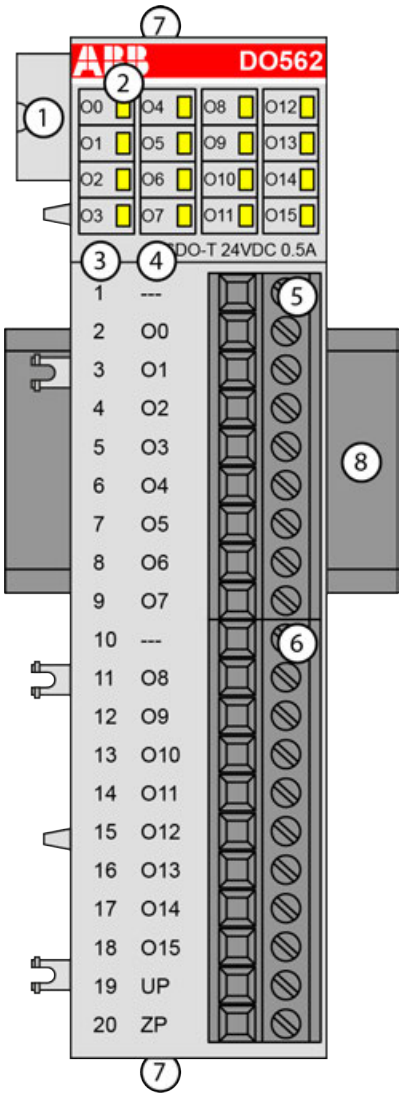
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2201	DO561, digital output module, 8 DO, transistor output	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DO562 - Digital output module

- 16 digital outputs 24 V DC (O0 to O15) in 1 group
- Module-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the outputs O0 to O15
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital outputs:

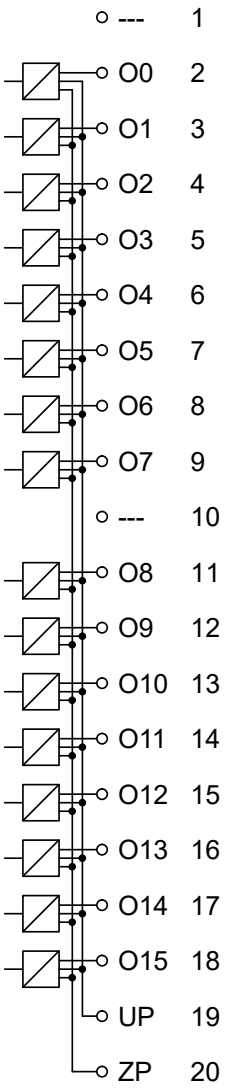


Table 417: Assignment of the terminals:

Terminal	Signal	Description
1	---	Reserved
2	O0	Output signal O0
3	O1	Output signal O1
4	O2	Output signal O2
5	O3	Output signal O3
6	O4	Output signal O4
7	O5	Output signal O5
8	O6	Output signal O6
9	O7	Output signal O7
10	---	Reserved
11	O8	Output signal O8
12	O9	Output signal O9
13	O10	Output signal O10
14	O11	Output signal O11

Terminal	Signal	Description
15	O12	Output signal O12
16	O13	Output signal O13
17	O14	Output signal O14
18	O15	Output signal O15
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO562.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

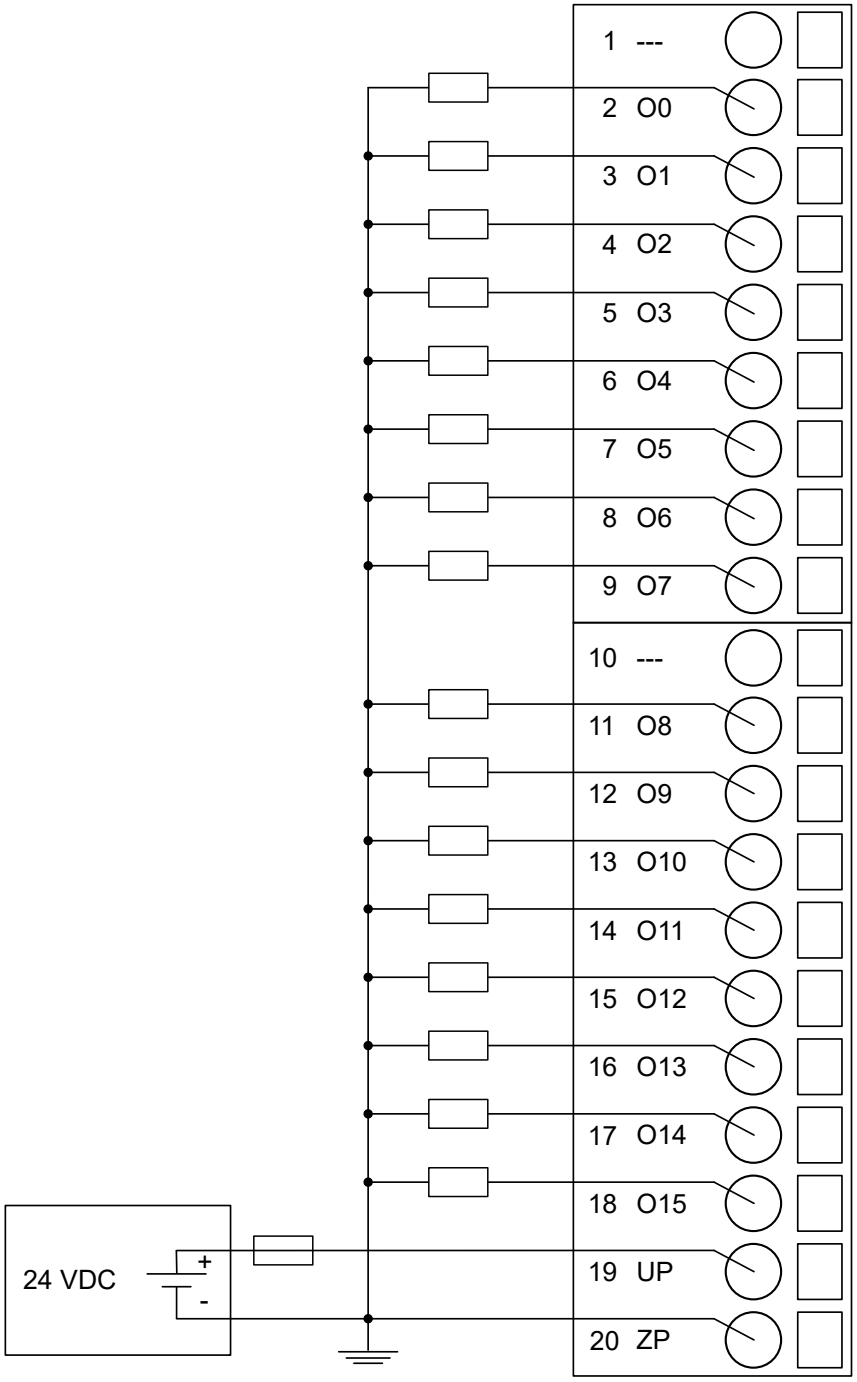
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the digital output module DO562:



NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

– Never short-circuit or overload the outputs.
– Never connect the outputs to other voltages.
– Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.2.6.1.1.8.6 “Diagnosis” on page 4192*).

The meaning of the LEDs is described in the section Status LEDs *Chapter 1.6.2.6.1.1.8.7 “State LEDs” on page 4192*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6145 [1])	WORD	6145 0x1801	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x06
Ext_User_Prm_Data_Const(0) =	0x18, 0x02, 0x00, 0x02, 0x00, 0x00;

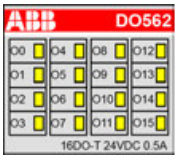
Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Inter- face**	**Device**	**Module**	**Channel**	**Error-Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O15	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	20 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		1 (16 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		1.4 W
Weight		Ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	16 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 (16 channels per group)
Connection of the channels O0 to O7	Terminals 1 to 9
Connection of the channels O8 to O15	Terminals 11 to 18
Common power supply voltage	Terminal 19 (positive pole of the process voltage, signal name UP)
Reference potential for the channels O0 to O15	Terminal 20 (negative pole of the process voltage, signal name ZP)

Parameter		Value
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation		Non-latching type
Min. output voltage at signal 1		UP -0.3 V at max. current consumption
Output delay (max. at rated load)		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		2 bytes
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	8 A
	Lamp load (max.)	5 W
Max. leakage current with signal 0		0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

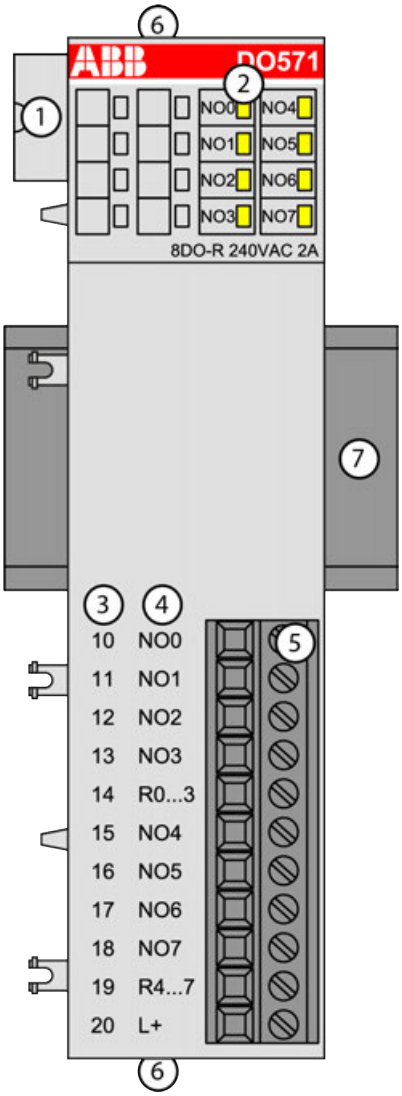
Part no.	Description	Product life cycle phase *)
1SAP 230 900 R0000	DO562, digital output module, 16 DO, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO571 - Digital output module

- 8 digital normally open relay outputs 24 V DC / 24 V AC or 100-240 V AC, 2 A max. (NO0 to NO7) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminal L+ (process voltage 24 V DC). The negative pole is provided by the I/O bus.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital outputs:

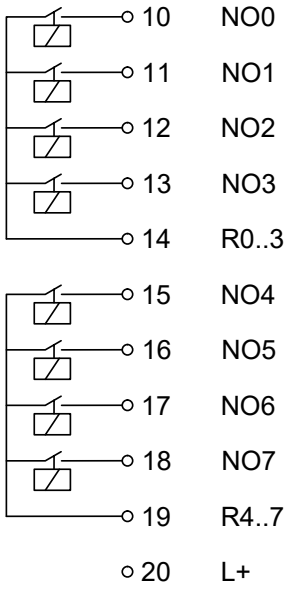


Table 418: Assignment of the terminals:

Terminal	Signal	Description
10	NO0	Normally-open contact of the output NO0
11	NO1	Normally-open contact of the output NO1
12	NO2	Normally-open contact of the output NO2
13	NO3	Normally-open contact of the output NO3
14	R0..3	Output common for signals NO0 to NO3

Terminal	Signal	Description
15	NO4	Normally-open contact of the output NO4
16	NO5	Normally-open contact of the output NO5
17	NO6	Normally-open contact of the output NO6
18	NO7	Normally-open contact of the output NO7
19	R4..7	Output common for signals NO4 to NO7
20	L+	Process voltage L+ +24 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DO571.

The external power supply connection is carried out via the L+ (+24 V DC) terminal. The negative pole of the external power supply is realized via the I/O bus. Therefore, the CPU/ communication interface module and the DO571 must have a common power supply.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..3 and R4..7) does not exceed 8 A.

Never connect total currents > 8 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The following figure shows the connection of the module:

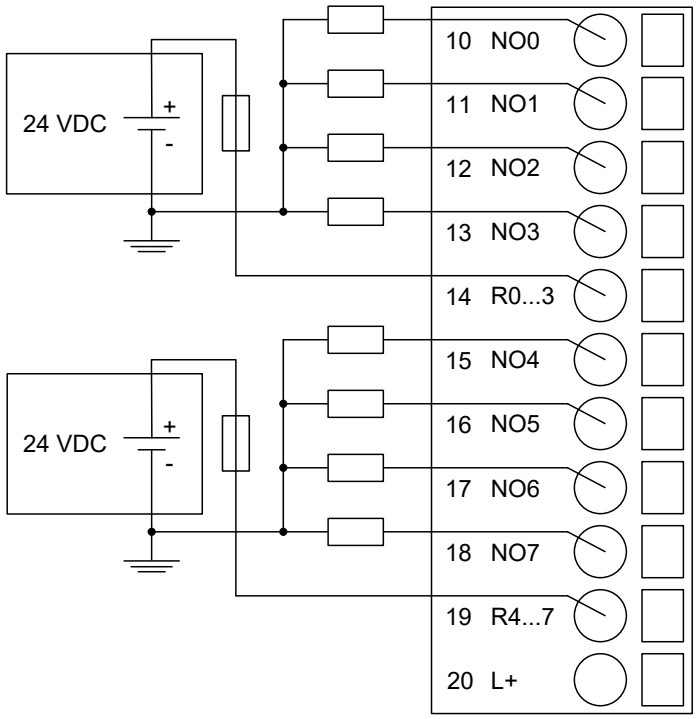


Fig. 828: Connection of 24 V DC actuators

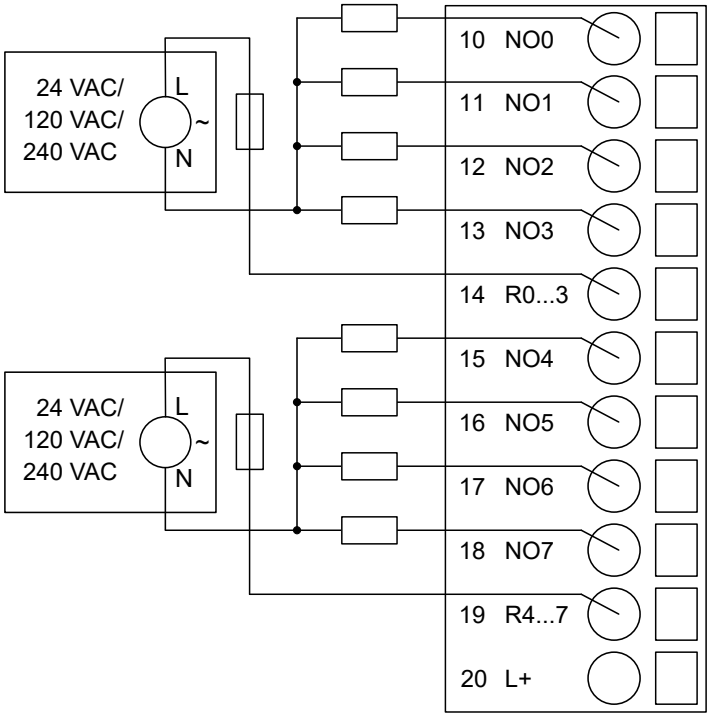


Fig. 829: Connection of 24 V AC or 100-240 V AC actuators

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

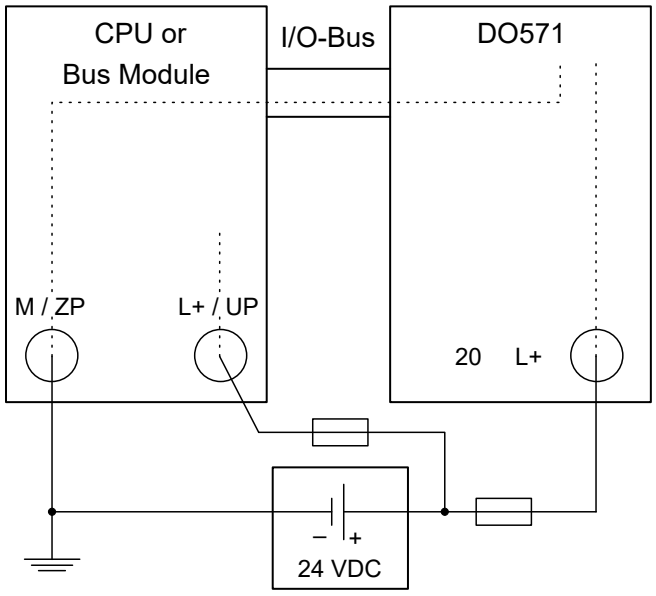


Fig. 830: Power supply - the negative connection is realized via the I/O bus

The L+ connection of the DO571 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.2.6.1.1.9.6 “Diagnosis” on page 4202*).

The meaning of the LEDs is described in the section Status LEDs *Chapter 1.6.2.6.1.1.9.7 “State LEDs” on page 4203*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6125 [1])	WORD	6125 0x17ED	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])
Check supply	Off On	0 1	BYTE	On 0x01			

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x04
Ext_User_Prm_Data_Const(0) =	0xEF, 0x17, 0x00,\ 0x01;

Diagnosis

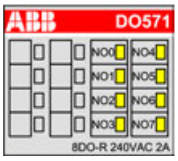
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e. g. of the DC551-CS31)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1..10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 20 for L+ (+24 V DC). The negative pole is provided by the I/O bus.
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Inrush current (at power-up)	0.0035 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by a 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the output group and the rest of the module
Isolated groups		2 (4 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.0 W
Weight		Ca. 150 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 normally-open relay outputs
Distribution of the channels into groups		2 (4 channels per group)
Connection of the channels O0 to O3		Terminals 10 to 13
Connection of the channels O4 to O7		Terminals 15 to 18
Reference potential for the channels O0 to O3		Terminal 14 (signal name R0..3)
Reference potential for the channels O4 to O7		Terminal 19 (signal name R4..7)
Relay coil power supply		Terminal 20 (positive pole of the process supply voltage, signal name L+). The negative pole is provided by the I/O bus.
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation		Non-latching type
Relay output voltage		
	Rated value	24 V DC / 24 V AC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		1 byte
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	8 A
	Lamp load (max.)	200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No

Parameter		Value
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

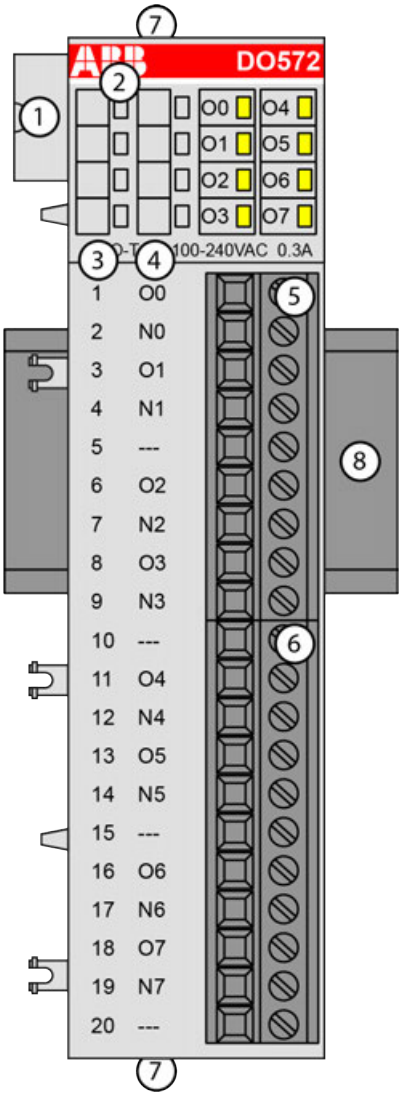
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2202	DO571, digital output module, 8 DO, relay output	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO572 - Digital output module

- 8 digital triac outputs (O0 to O7) in 8 groups
- 240 V AC
- Module-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the outputs O0 to O7
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Not necessary

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital outputs:

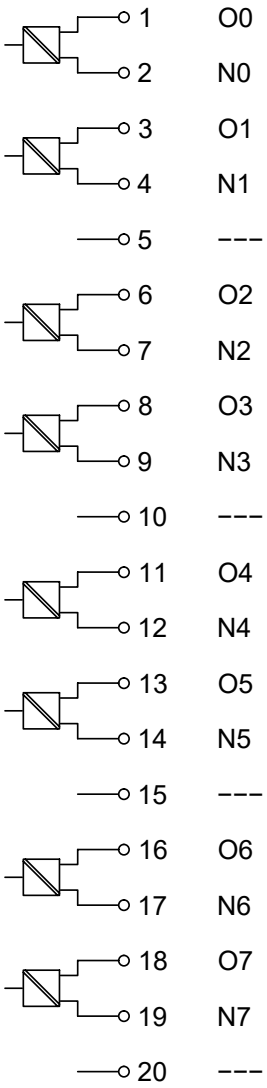


Table 419: Assignment of the terminals:

Terminal	Signal	Description
1	O0	Output signal O0
2	N0	Neutral conductor for the output signal O0
3	O1	Output signal O1
4	N1	Neutral conductor for the output signal O1
5	---	Reserved
6	O2	Output signal O2
7	N2	Neutral conductor for the output signal O2
8	O3	Output signal O3
9	N3	Neutral conductor for the output signal O3
10	---	Reserved
11	O4	Output signal O4

Terminal	Signal	Description
12	N4	Neutral conductor for the output signal O4
13	O5	Output signal O5
14	N5	Neutral conductor for the output signal O5
15	---	Reserved
16	O6	Output signal O6
17	N6	Neutral conductor for the output signal O6
18	O7	Output signal O7
19	N7	Neutral conductor for the output signal O7
20	---	Reserved

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DO572.

An external power supply connection is not needed.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

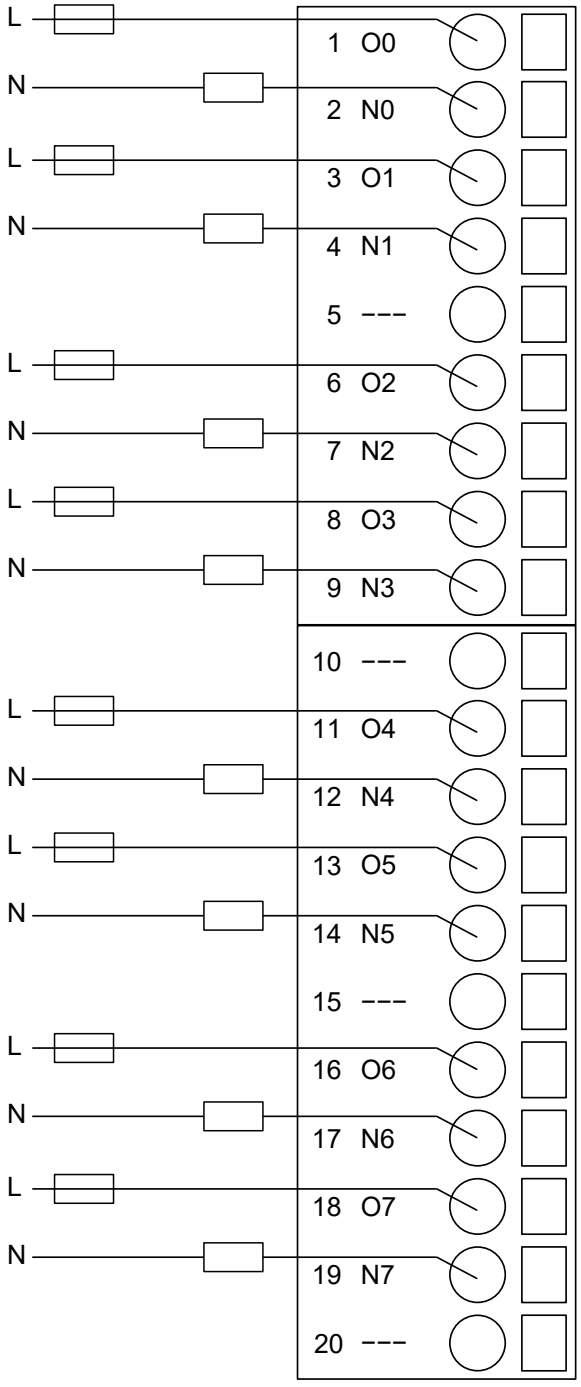
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the module:



NOTICE!

Risk of damaging the PLC modules!

The PLC modules will be irreparably damaged if a voltage > 240 V is connected.

Make sure that all inputs are fed from the same phase. The module must not be connected to a 400 V voltage.

The module provides several diagnosis functions (see chapter Diagnosis *Chapter 1.6.2.6.1.1.10.6 “Diagnosis” on page 4212*).

The meaning of the LEDs is described in the section State LEDs *Chapter 1.6.2.6.1.1.10.7 “State LEDs” on page 4213*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6130 [1])	WORD	6130 0x17F2	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length [2])	Internal	1 - CPU	BYTE	0	0	255	xx02 [3])

[1])	With CS31 and addresses smaller than 70, the value is increased by 1
[2])	The module has no additional user-configurable parameters
[3])	Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xF3, 0x17, 0x00;

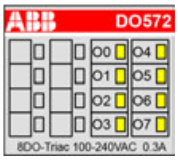
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter	Value
Galvanic isolation	Yes, between the channels and the rest of the module
Isolated groups	8 (1 channel per group)
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 10 mA
Max. power dissipation within the module	On Request
Weight	ca. 120 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 triac outputs
Distribution of the channels into groups		8 groups (1 channel per group)
Connection of the channels O0 to O7		Terminals 1, 3, 5, 7, 10, 12, 14, 16
Reference potential for the channels O0 to O7		Terminals 2, 4, 6, 8, 11, 13, 15, 17
Output voltage for signal 1		On Request
Max. leakage current with signal 0		1.1 mA root mean square at 132 V AC and 1.8 mA root mean square at 264 V AC
Output voltage		
	Rated value	120 V AC or 240 V AC
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus

Parameter		Value
Way of operation		Non-latching type
Output delay		On Request
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.3 A
	Rated current per group (max.)	0.3 A
Surge current (max.)		On request
Lamp load (max.)		On request
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 10 Hz
	With inductive loads	Not applicable
	With lamp loads	Max. 10 Hz
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse		2 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Resistance to feedback against 230 V AC		No
Connection of 2 outputs in parallel		Not applicable
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

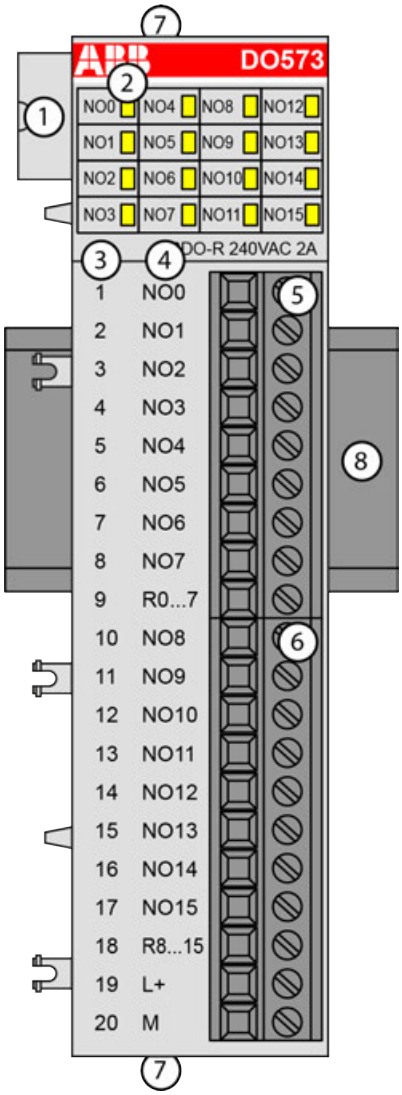
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2203	DO572, digital output module, 8 DO, triac output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO573 - Digital output module

- 16 digital normally open relay outputs 24 V DC or 100-240 V AC (NO0 to NO15) in 2 groups, 2 A max.
- Group-wise galvanically isolated



1 I/O bus
2 16 yellow LEDs to display the signal states of the outputs O0 to O15
3 Terminal number

4 Allocation of signal name
5 Terminal block for output signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital outputs:

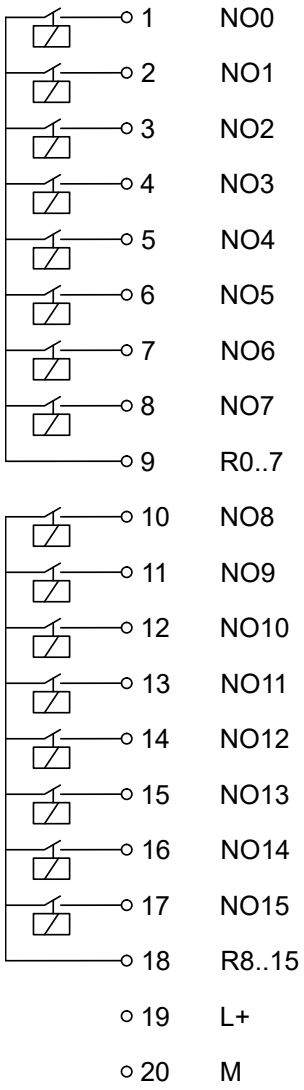


Table 420: Assignment of the terminals:

Terminal	Signal	Description
1	NO0	Normally-open contact of the output NO0
2	NO1	Normally-open contact of the output NO1
3	NO2	Normally-open contact of the output NO2
4	NO3	Normally-open contact of the output NO3
5	NO4	Normally-open contact of the output NO4
6	NO5	Normally-open contact of the output NO5
7	NO6	Normally-open contact of the output NO6
8	NO7	Normally-open contact of the output NO7
9	R0..7	Output common for signals NO0 to NO7
10	NO8	Normally-open contact of the output NO8
11	NO9	Normally-open contact of the output NO9
12	NO10	Normally-open contact of the output NO10
13	NO11	Normally-open contact of the output NO11
14	NO12	Normally-open contact of the output NO12

Terminal	Signal	Description
15	NO13	Normally-open contact of the output NO13
16	NO14	Normally-open contact of the output NO14
17	NO15	Normally-open contact of the output NO15
18	R8..15	Output common for signals NO8 to NO15
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DO573.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is electrically interconnected to the M/ZP terminal of the CPU/ communication interface module.

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..7 and R8..15) does not exceed 10 A.

Never connect total currents > 10 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The following figure shows the connection of the module:

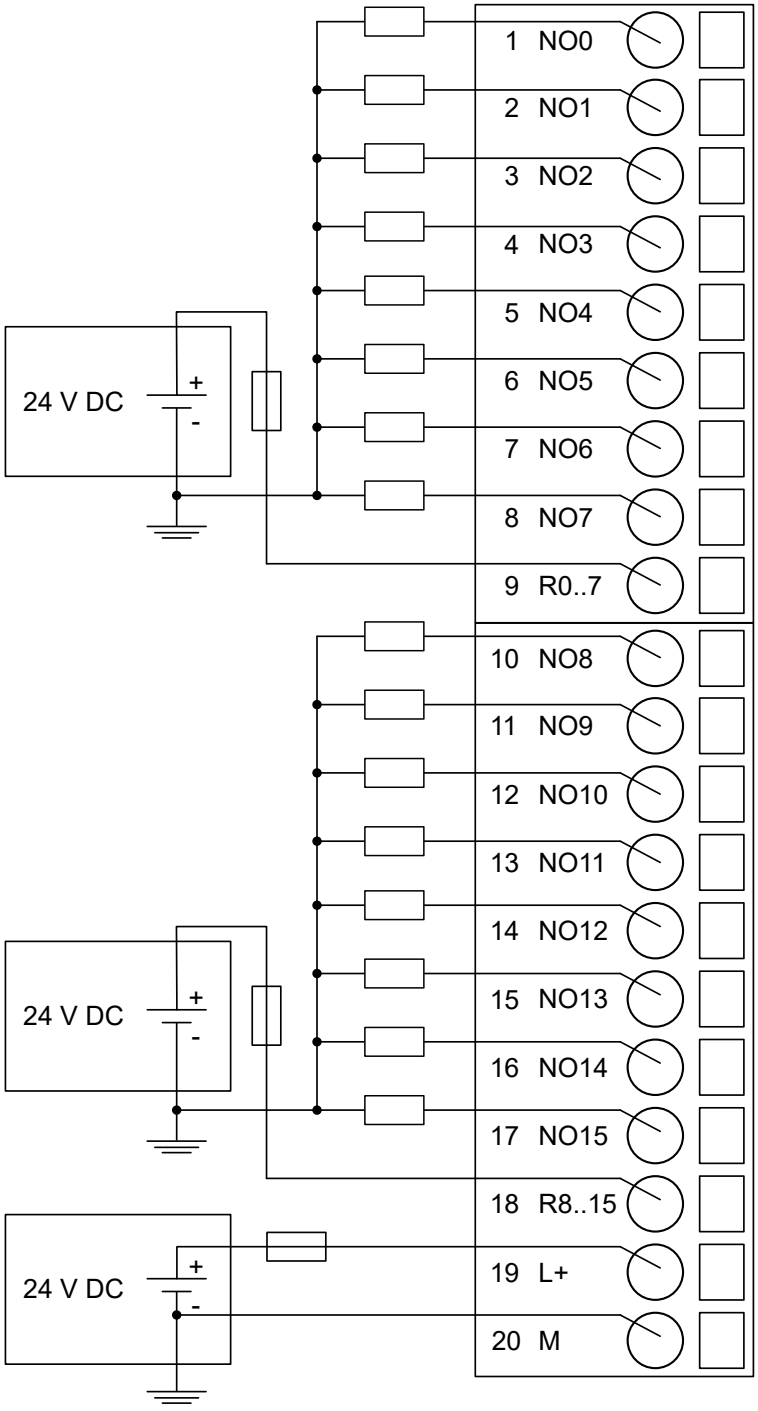


Fig. 831: Connection of 24 V DC actuators

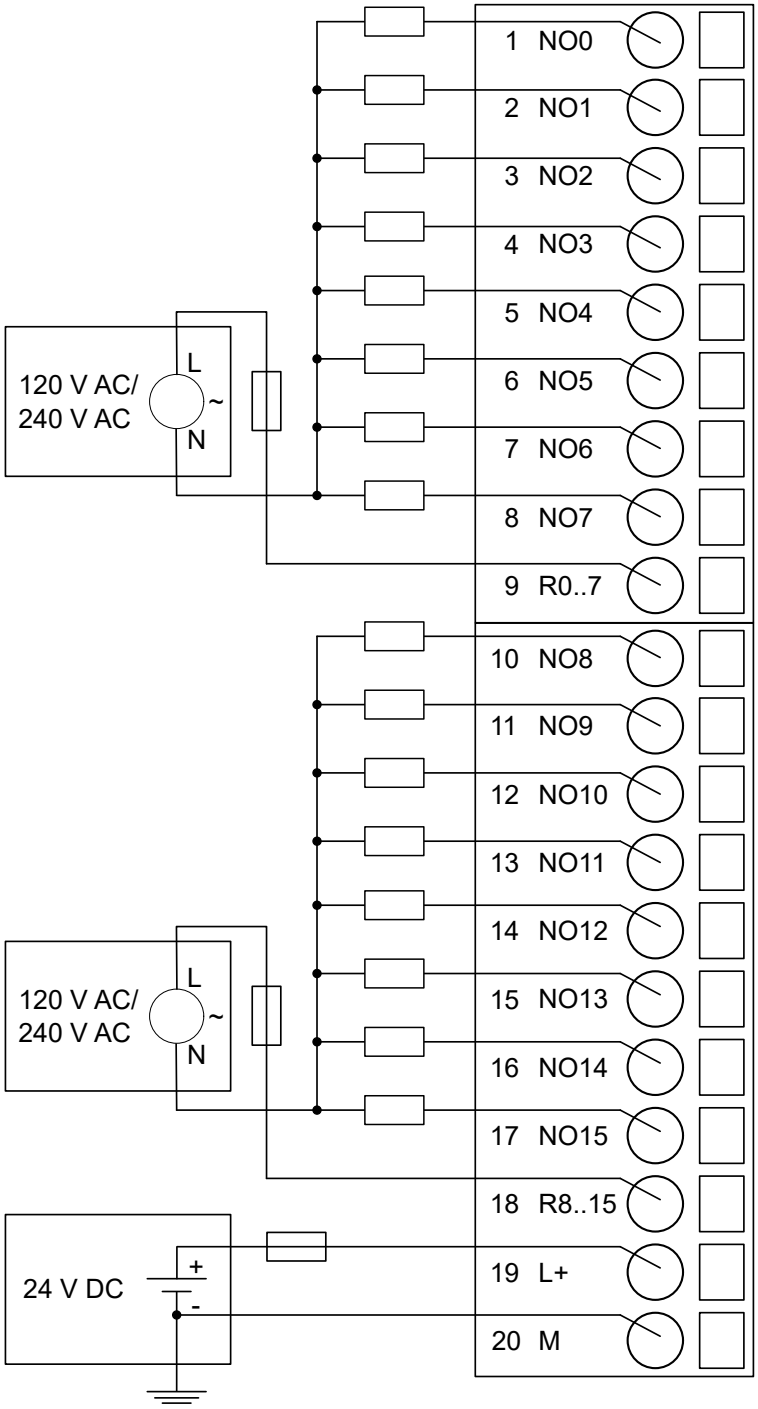


Fig. 832: Connection of 100-240 V AC actuators

The module provides several diagnosis functions (see section Diagnosis ↳ *Chapter 1.6.2.6.1.1.11.6 “Diagnosis” on page 4223*).

The meaning of the LEDs is described in the section State LEDs ↳ *Chapter 1.6.2.6.1.1.10.7 “State LEDs” on page 4213*.

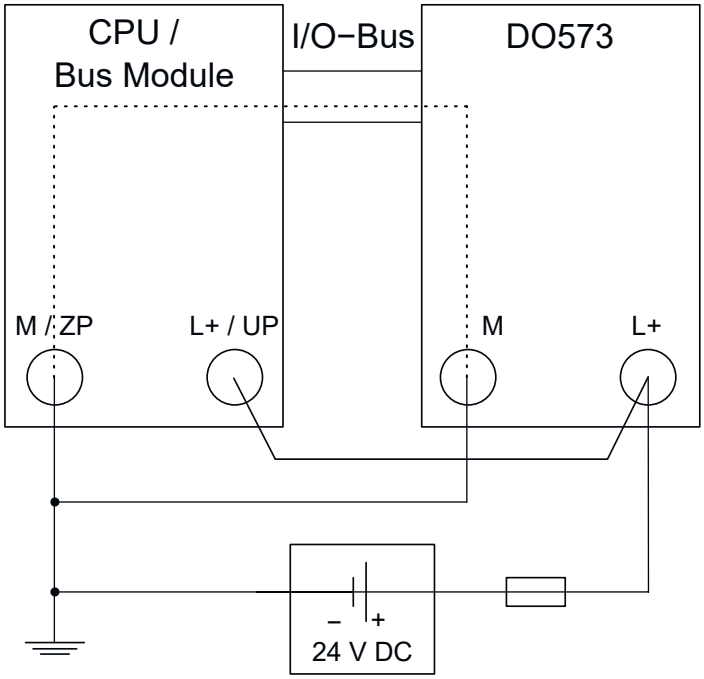


Fig. 833: Power supply - the negative connection is realized via the I/O bus

The L+ connection of the DO573 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply .

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6150 [1])	WORD	6150 0x1806	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])
Check supply	Off On	0 1	BYTE	On 0x01			

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	0x07 0x18, 0x07, 0x00, 0x03, 0x01, 0x00, 0x00;

Diagnosis

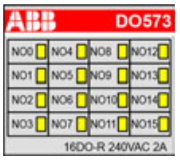
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**
	Outputs NO0...NO15	Digital output	Yellow	Output is OFF	Output is ON (the output voltage is only displayed if the supply voltage of the module is ON)

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminals 19 for L+ (+24 V DC) and 20 for M (0 V DC)
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for L+	Recommended; the outputs must be protected by an 5 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the output groups and the rest of the module
Isolated groups		2 (8 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.0 W
Weight		Ca. 160 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter	Value
Number of channels per module	16 normally-open relay outputs
Distribution of the channels into groups	2 (8 channels per group)
Connection of the channels NO0 to NO7	Terminals 1 to 8
Connection of the channels NO8 to NO15	Terminals 10 to 17
Reference potential for the channels NO0 to NO7	Terminal 9 (signal name R0..7)
Reference potential for the channels NO8 to NO15	Terminal 18 (signal name R8..15)
Relay coil power supply	Terminals 19 and 20 (signal names L+ and M)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Way of operation	Non-latching type

Parameter		Value
Relay output voltage		
	Rated value	24 V DC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		2 bytes
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	10 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		
	With resistive loads	Max. 1 Hz
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Ordering data

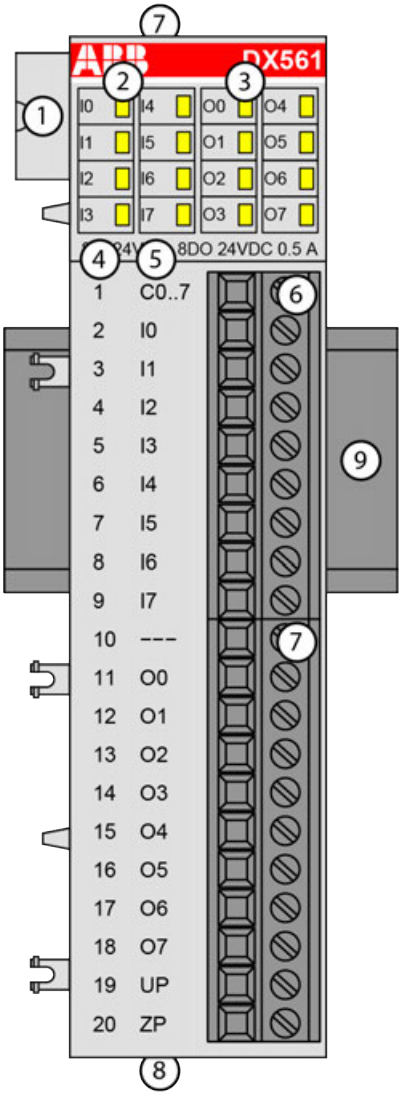
Part no.	Description	Product life cycle phase *)
1SAP 231 300 R0000	DO573, digital output module, 16 DO, relay output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DX561 - Digital input/output module

- 8 digital inputs 24 V DC (I0 to I7) in 1 group
- 8 digital transistor outputs 24 V DC (O0 to O7) in 1 group
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 8 yellow LEDs to display the signal states of the outputs O0 to O7
4 Terminal number
5 Allocation of signal name
6 Terminal block for input signals (9-pin)
7 Terminal block for output signals (11-pin)
8 2 holes for wall-mounting with screws
9 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs and outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs and outputs:

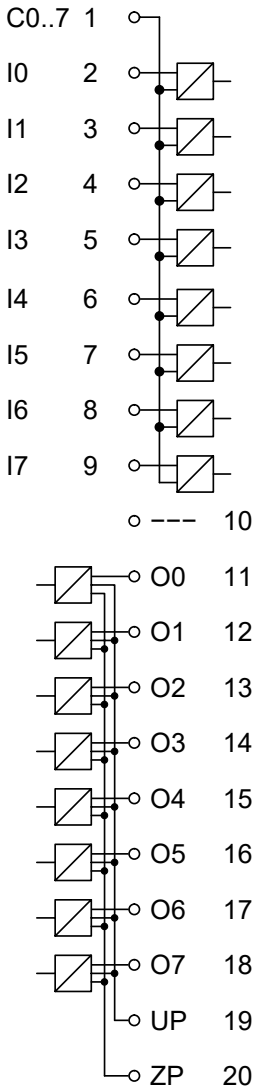


Table 421: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	---	Reserved
11	O0	Output signal O0
12	O1	Output signal O1
13	O2	Output signal O2

Terminal	Signal	Description
14	O3	Output signal O3
15	O4	Output signal O4
16	O5	Output signal O5
17	O6	Output signal O6
18	O7	Output signal O7
19	UP	Process voltage UP +24 V DC
20	ZP	Process voltage ZP 0 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per DX561.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the inputs to the digital input/output module DX561:

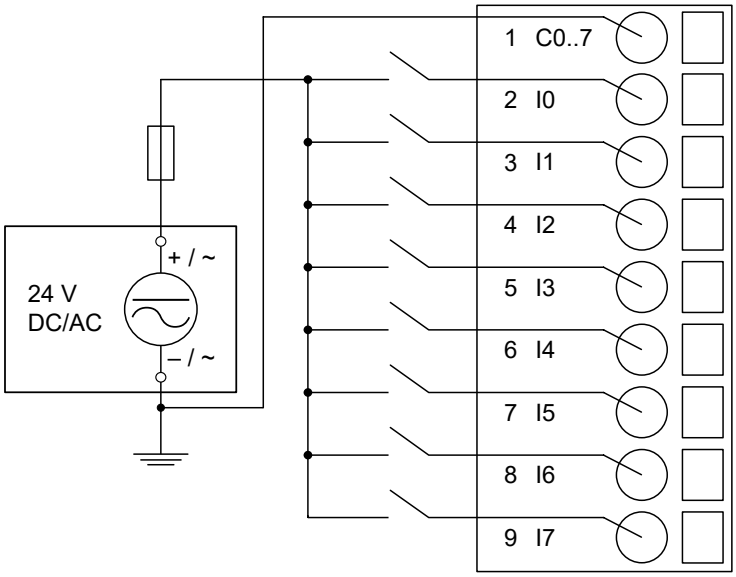


Fig. 834: Connection of inputs - sink inputs

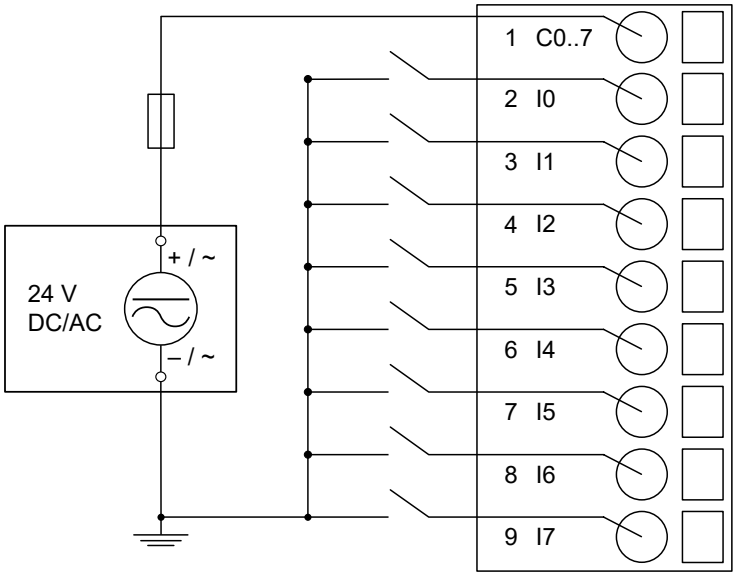


Fig. 835: Connection of inputs - source inputs

The following figure shows the connection of the outputs to the module:

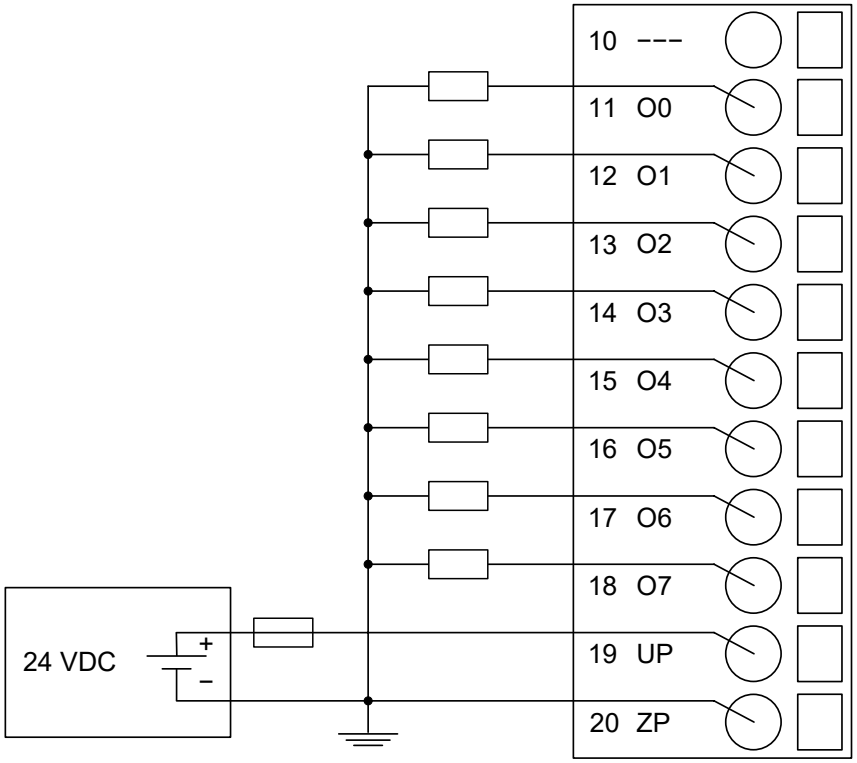


Fig. 836

NOTICE!

Risk of malfunctions in the plant!

The outputs may switch on for a period of 10 to 50 µs if the process supply voltage UP/ZP is switched on.

This must be considered in the planning of the application.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuits and overload.

- Never short-circuit or overload the outputs.
- Never connect the outputs to other voltages.
- Use an external 3 A fast-protection fuse for the outputs.

The module provides several diagnosis functions (see chapter Diagnosis ↬ *Chapter 1.6.2.6.1.1.12.6 “Diagnosis” on page 4235*).

The meaning of the LEDs is described in the Displays section ↬ *Chapter 1.6.2.6.1.1.12.7 “State LEDs” on page 4236* chapter.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6135 [1])	WORD	6135 0x17F7	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x03
Ext_User_Prm_Data_Const(0) =	0xF8, 0x17, 0x00,\
(0) =	0x01;

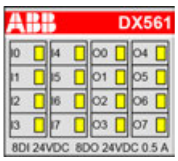
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON
	Outputs O0...O7	Digital output	Yellow	Output is OFF	Output is ON

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V DC)
	Rated value	24 V DC
	Current consumption via UP terminal	5 mA + max. 0.5 A per output
	Max. ripple	5 %
	Inrush current	0.000002 A^2s
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Recommended; the outputs must be protected by an 3 A fast-acting fuse
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		Yes, between the input group and the output group and the rest of the module
Isolated groups		2 groups (1 group for 8 input channels, 1 group for 8 output channels)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.3 W
Weight		ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter		Value	
Number of channels per module		8	
Distribution of the channels into groups		1 group for 8 channels	
Connections of the channels I0 to I7		Terminals 2 to 9	
Reference potential for the channels I0 to I7		Terminal 1	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)	
Monitoring point of input indicator		LED is part of the input circuitry	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
Signal 0		-5 V...+3 V	-3 V...+5 V
Undefined signal		-15 V...+ 5 V	+5 V...+15 V
Signal 1		-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		-5 V...+3 V	-3 V...+5 V
Ripple with signal 1		-30 V...-15 V	+15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 8 ms	
Input data length		1 byte	
Max. cable length			
	Shielded	500 m	
	Unshielded	300 m	

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8 transistor outputs (24 V DC, 0.5 A max.)
Distribution of the channels into groups	1 group of 8 channels
Connection of the channels O0 to O7	Terminals 11 to 18
Reference potential for the channels O0 to O7	Terminal 20 (negative pole of the process voltage, name ZP)
Common power supply voltage	Terminal 19 (positive pole of the process voltage, name UP)
Indication of the output signals	1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered via the I/O bus
Monitoring point of output indicator	Controlled together with transistor

Parameter		Value
Way of operation		Non-latching type
Max. output voltage at signal 1		20 V DC at max. current consumption
Output delay		
	0 to 1	50 µs
	1 to 0	200 µs
Output data length		1 byte
Output current		
	Rated current per channel (max.)	0.5 A at UP 24 V DC
	Rated current per group (max.)	4 A
	Rated current (all channels together, max.)	4 A
	Lamp load (max.)	5 W
	Max. leakage current with signal 0	0.5 mA
Output type		Non-protected
Protection type		External fuse on each channel
Rated protection fuse (for each channel)		3 A fast
Demagnetization when inductive loads are switched off		Must be performed externally according to driven load specification
Switching Frequencies		
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz at max. 5 W
Short-circuit-proof / Overload-proof		No
	Overload message	No
	Output current limitation	No
	Resistance to feedback against 24 V DC	No
Connection of 2 outputs in parallel		Not possible
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

Ordering data

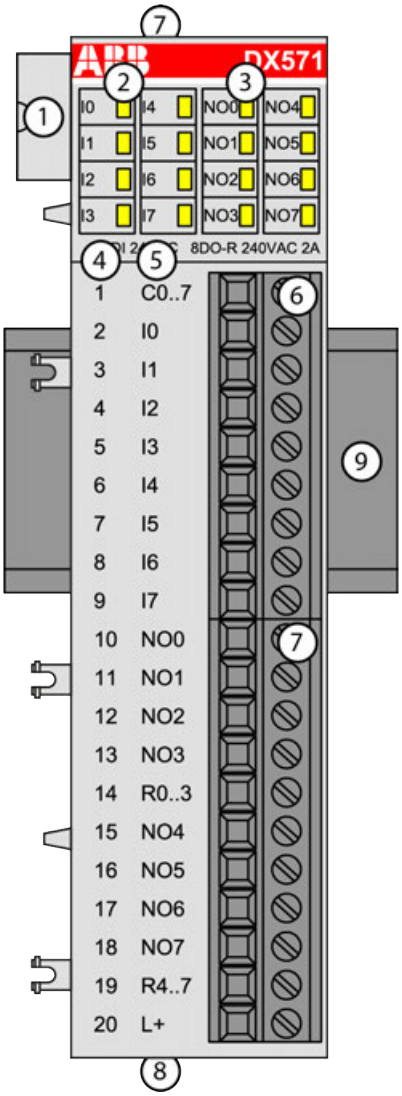
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2301	DX561, digital input/output module, 8 DI 24 V DC, 8 DO 24 V DC, transistor output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DX571 - Digital input/output module

- 8 digital inputs 24 V DC / 24 V AC (I0 to I7) in 1 group
- 8 digital normally open relay outputs 24 V DC / 24 V AC or 100-240 V AC, 2 A max. (NO0 to NO7) in 2 groups
- Group-wise galvanically isolated



1 I/O bus
2 8 yellow LEDs to display the signal states of the inputs I0 to I7
3 8 yellow LEDs to display the signal states of the outputs NO0 to NO7
4 Terminal number
5 Allocation of signal name
6 Terminal block for input signals (9-pin)
7 Terminal block for output signals (11-pin)
8 2 holes for wall-mounting with screws
9 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs and outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

Parameter	Value
LED displays	For signal states
Internal power supply	Via I/O bus
External power supply	Via the terminal L+ (process voltage 24 V DC). The negative pole is provided by the I/O bus.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the digital inputs and outputs:

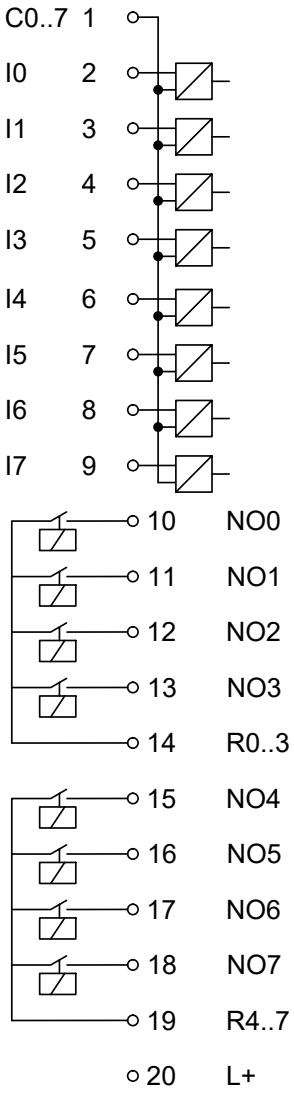


Table 422: Assignment of the terminals:

Terminal	Signal	Description
1	C0...7	Input common for signals I0 to I7
2	I0	Input signal I0
3	I1	Input signal I1
4	I2	Input signal I2
5	I3	Input signal I3
6	I4	Input signal I4
7	I5	Input signal I5
8	I6	Input signal I6
9	I7	Input signal I7
10	NO0	Normally-open contact of the output 0
11	NO1	Normally-open contact of the output 1
12	NO2	Normally-open contact of the output 2

Terminal	Signal	Description
13	NO3	Normally-open contact of the output 3
14	R0...3	Output common for signals O0 to O3
15	NO4	Normally-open contact of the output 4
16	NO5	Normally-open contact of the output 5
17	NO6	Normally-open contact of the output 6
18	NO7	Normally-open contact of the output 7
19	R4...7	Output common for signals O4 to O7
20	L+	Process voltage +24 V DC

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per DX571.

The external power supply connection is carried out via the L+ (+24 V DC) terminal. The negative pole of the external power supply is realized via the I/O bus. Therefore, the CPU/ communication interface module and the DX571 must have a common power supply.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overload.

Make sure that the total current of each output common terminal (R0..3 and R4..7) does not exceed 8 A.

Never connect total currents > 8 A per group.

If the group fuse protection is not sufficient, then individual fuse protection of the outputs should be used.

The module provides several diagnosis functions (see Diagnosis *Chapter 1.6.2.6.1.1.13.6 “Diagnosis” on page 4248*).

The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figures show the connection of the inputs to the digital input/output module DX571:

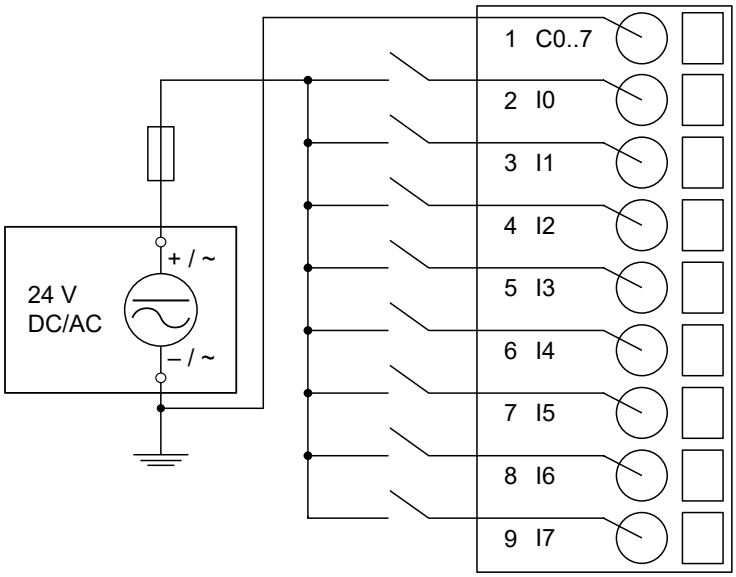


Fig. 837: Connection of inputs - sink inputs

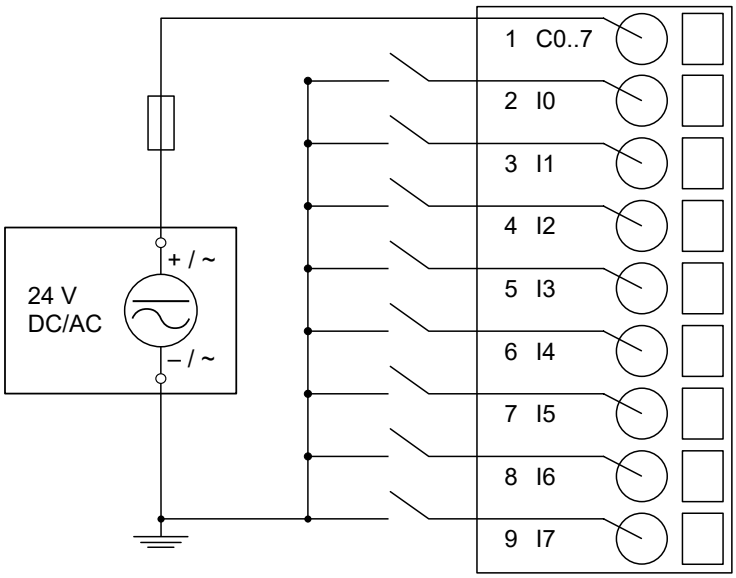


Fig. 838: Connection of inputs - source inputs

The following figures show the connection of the outputs to the module:

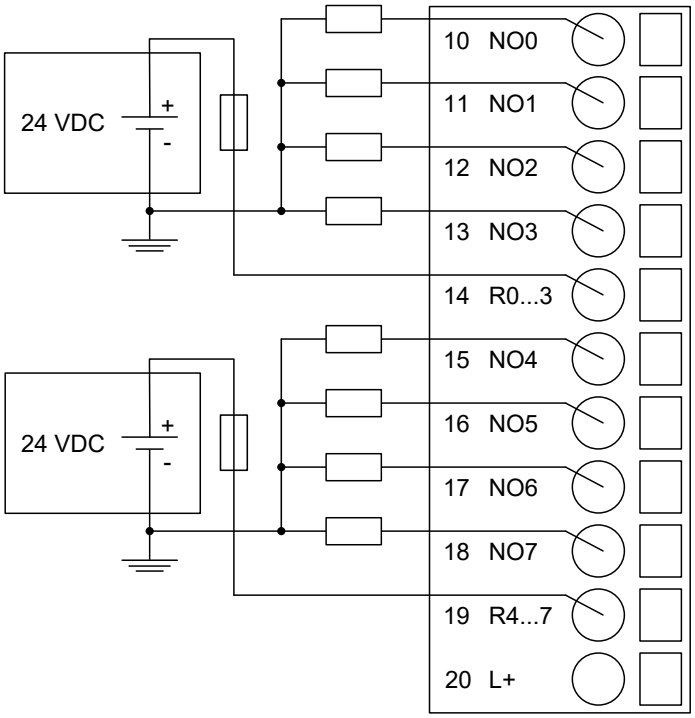


Fig. 839: Connection of 24 V DC actuators

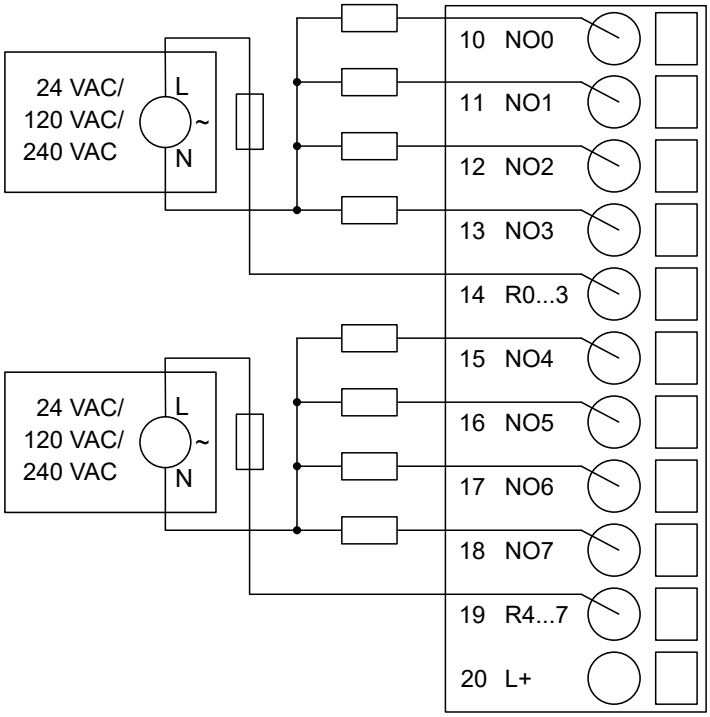


Fig. 840: Connection of 24 V AC or 100-240 V AC actuators

The L+ connection of the DX571 and the 24 V supply of the CPU/communication interface module must be connected to the same 24 V power supply.

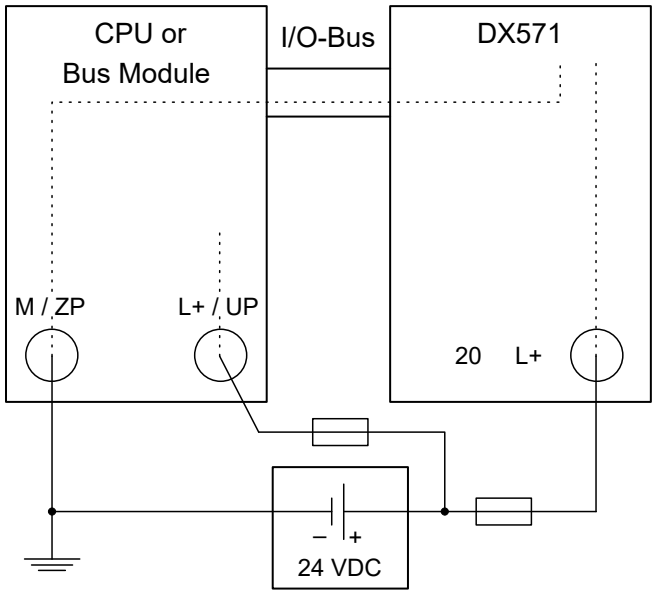


Fig. 841: Power supply - the minus connection is realized via the I/O bus

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For screw-type terminals only:

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

NOTICE!

Risk of damaging the I/O module!

The outputs are not protected against short circuit and overload.

- Never short-circuit or overload the outputs.
- Never connect inductive loads without an external suppression against voltage peaks due to inductive kickback.
- Never connect voltages > 240 V. All outputs must be supplied from the same phase.
- Use an external 5 A fast protection fuse for the outputs.

The meaning of the LEDs is described in the Displays section *Chapter 1.6.2.6.1.1.13.7 “State LEDs” on page 4249*.

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6140 [1])	WORD	6140 0x17FC	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No (0x00)			
Parameter length	Internal	1	BYTE	0	0	255	xx02 [2])

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.	EDS Slot Index
Check supply	Off On	0 1	BYTE	On 0x01			
[1]) with CS31 and addresses smaller than 70, the value is increased by 1							
[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x04
Ext_User_Prm_Data_Const(0) =	0xFD, 0x17, 0x00,\
(0) =	0x01;

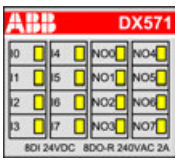
Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500-Display	<− Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Inter face**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
4	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = Module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON
	Inputs I0...I7	Digital input	Yellow	Input is OFF	Input is ON
	Outputs NO0...NO7	Digital output	Yellow	Output is OFF	Output is ON

In the undefined signal range, the state LED for the inputs can be ON although the input state detected by the module is OFF.

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 20 for L+ (+24 V DC). The negative pole is provided by the I/O bus.
	Rated value	24 V DC
	Current consumption via L+	50 mA
	Inrush current (at power-up)	0.0035 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for L+	Recommended; the outputs must be protected by a 3 A fast-acting fuse

Parameter	Value
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 5 mA
Galvanic isolation	Yes, between the input group and the output group and the rest of the module
Isolated groups	3 groups (1 group for 8 input channels, 2 groups for 8 output channels)
Surge-voltage (max.)	35 V DC for 0.5 s
Max. power dissipation within the module	2.3 W
Weight	Ca. 150 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter		Value		
Number of channels per module		8		
Distribution of the channels into groups		1 group for 8 channels		
Connections of the channels I0 to I7		Terminals 2 to 9		
Reference potential for the channels I0 to I7		Terminal 1		
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)		
Monitoring point of input indicator		LED is part of the input circuitry		
Input type according to EN 61131-2		Type 1 source	Type 1 sink	Type 1 AC [1])
Input signal range		-24 V DC	+24 V DC	24 V AC 50/60 Hz
Signal 0		-5 V...+3 V	-3 V...+5 V	0 V AC...5 V AC
Undefined signal		-15 V...+ 5 V	+5 V...+15 V	5 V AC...14 V AC
Signal 1		-30 V...-15 V	+15 V...+30 V	14 V AC...27 V AC
Input current per channel				
	Input voltage 24 V	Typ. 5 mA		Typ. 5 mA r.m.s.
	Input voltage 5 V	Typ. 1 mA		Typ. 1 mA r.m.s.
	Input voltage 14 V			Typ. 2.7 mA r.m.s.
	Input voltage 15 V	> 2.5 mA		
	Input voltage 27 V			Typ. 5.5 mA r.m.s.
	Input voltage 30 V	< 8 mA		
Max. permissible leakage current (at 2-wire proximity switches)		1 mA		Typ. 1 mA r.m.s.

Parameter		Value
Input delay (0->1 or 1->0)		Typ. 8 ms
Input data length		1 byte
Max. cable length		
	Shielded	500 m
	Unshielded	300 m

[1]) When inputs are used with 24 V AC, external surge limiting filters are required.

Refer to *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233* for details

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 normally-open relay outputs
Distribution of the channels into groups		2 (4 channels per group)
Connection of the channels O0 to O3		Terminals 10 to 13
Connection of the channels O4 to O7		Terminals 15 to 18
Reference potential for the channels O0 to O3		Terminal 14 (signal name R0..3)
Reference potential for the channels O4 to O7		Terminal 19 (signal name R4..7)
Relay coil power supply		Terminal 20 (positive pole of the process supply voltage, signal name L+). The negative pole is provided by the I/O bus.
Indication of the output signals		1 yellow LED per channel; the LED is on when the output signal is high (signal 1) and the module is powered through the I/O bus
Monitoring point of output indicator		Controlled together with relay
Way of operation		Non-latching type
Relay output voltage		
	Rated value	24 V DC / 24 V AC or 120/240 V AC
Output delay		
	Switching 0 to 1 (max.)	Typ. 10 ms
	Switching 1 to 0 (max.)	Typ. 10 ms
Output data length		1 byte
Output current		
	Rated current per channel (max.)	2.0 A (24 V DC / 24 V AC / 48 V AC / 120 V AC / 240 V AC, only resistive loads) 2.0 A (24 V AC / 48 V AC / 120 V AC, only pilot duty) 1.5 A (240 V AC, only pilot duty)
	Rated current per group (max.)	8 A
Lamp load (max.)		200 W (230 V AC), 30 W (24 V DC)
Spark suppression with inductive AC loads		Must be performed externally according to driven load specification
Switching Frequencies		

Parameter		Value
	With resistive loads	Max. 1 Hz
	With inductive loads	On Request
	With lamp loads	Max. 1 Hz
Output type		Non-protected
Protection type		External fuse [1])
Rated protection fuse		5 A fast
Short-circuit-proof / Overload-proof		No, should be provided by an external fuse or circuit breaker
	Overload message	No
	Output current limitation	No
Connection of 2 outputs in parallel		Not possible
Lifetime of relay contacts (cycles)		100.000 at rated load
Max. cable length		
	Shielded	500 m
	Unshielded	150 m

[1]) Per group in case of group fuse protection. For each channel in case of channel-by-channel fuse protection. The maximum current per group must not be exceeded.

Ordering data

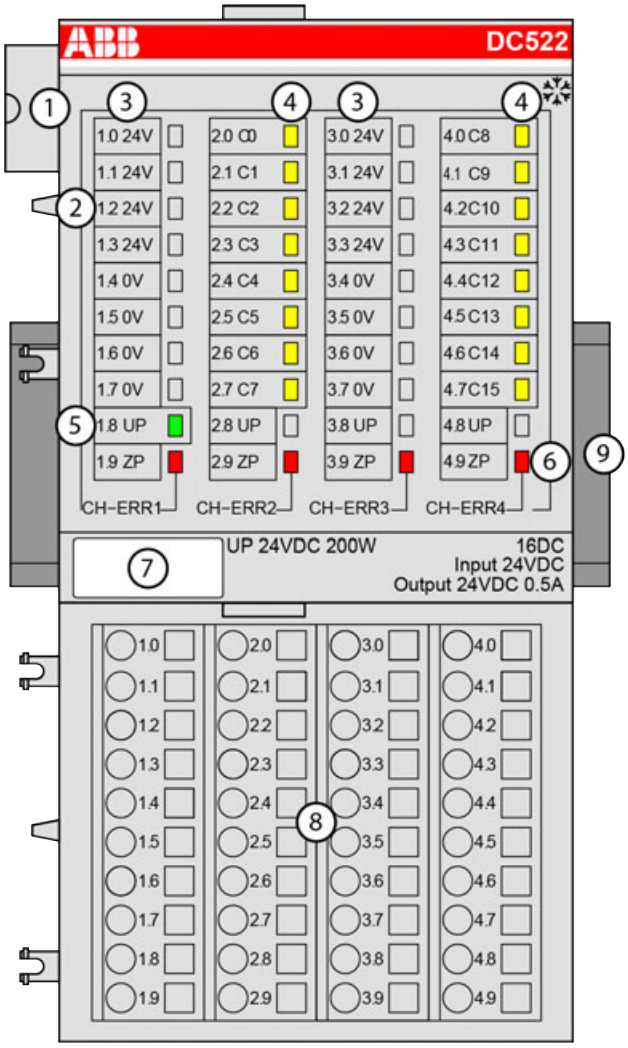
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R2302	DX571, digital input/output module, 8 DI 24 V DC / 24 V AC, 8 DO, relay output	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

S500

DC522 - Digital input/output module

- 16 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 Sensor power supply 24 V DC / 0.5 A
4 16 yellow LEDs to display the signal states at the digital inputs/outputs (C0 - C15)
5 1 green LED to display the state of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 2 sensor supply voltages 24 V DC, 0.5 A, with short-circuit and overload protection
- 16 digital configurable inputs/outputs 24 V DC (C0 to C15) in 1 group (2.0...2.7 and 4.0...4.7), each of which can be used
  - as an input,
  - as a transistor output with short-circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

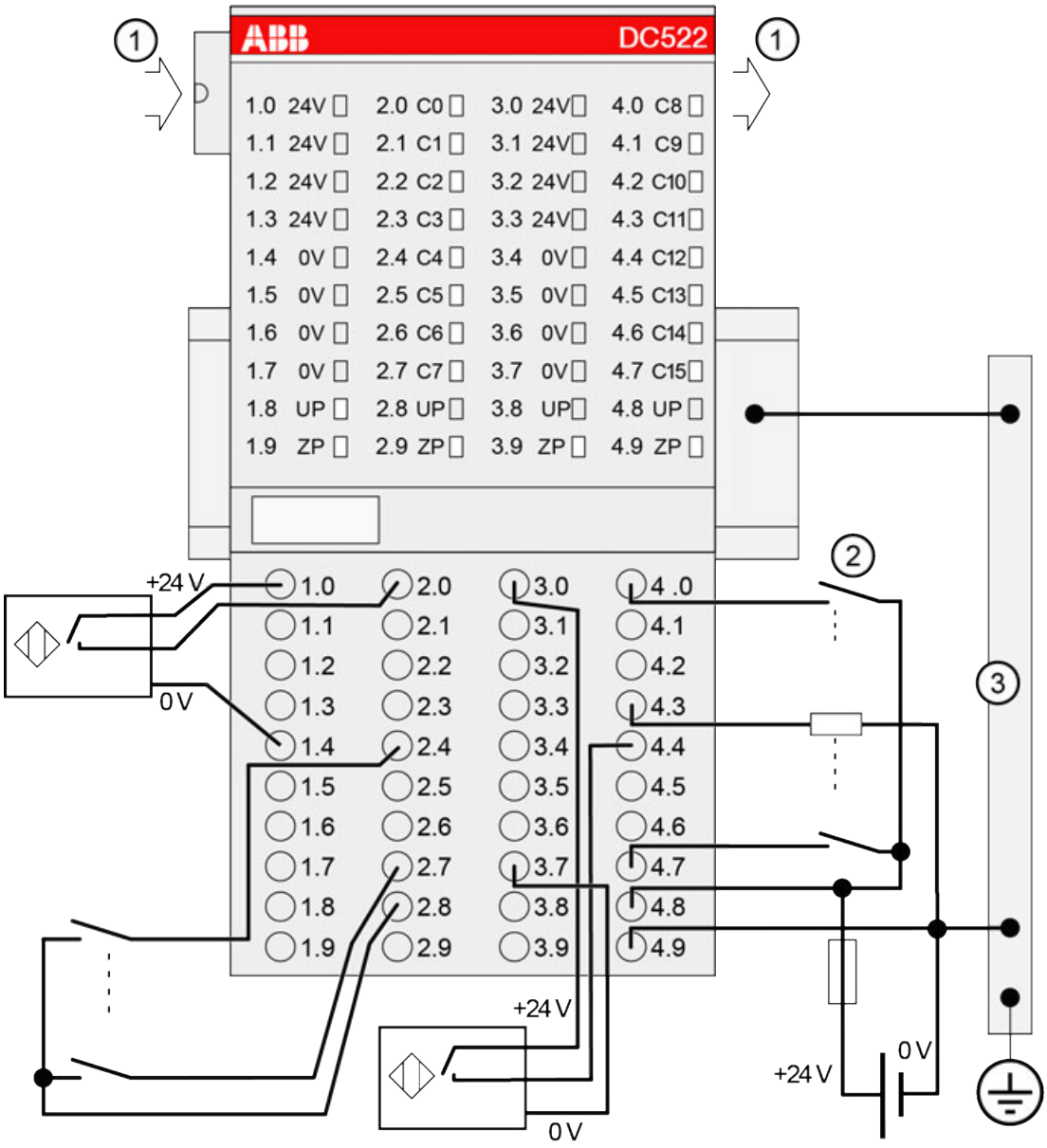
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
1.4 to 1.7	0 V	0 V (reference potential)
2.0 to 2.7	C0 to C7	8 digital inputs/outputs
3.0 to 3.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
3.4 to 3.7	0 V	0 V (reference potential)
4.0 to 4.7	C8 to C15	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC522.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!
Removal/Insertion under power
Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC522.

Connect a 470 Ω / 1 W resistor in series to inputs C8/C9 if they are used as fast counter inputs to avoid any influences.

The modules provide several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	2	4
Digital outputs (bytes)	2	4
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O Configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1220 [1])	Word	1220 0x04C4	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	7	Byte	7-CPU 6-FBP	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Short-circuit detection of output or sensor supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs Bit 15 = Output 15 Bit 0 = Output 0	0... 65535	0... 0xffff	Word	0 0x0000	0	65535	0x0Y07

Remarks:

[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
[4])	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	9
Ext_User_Prm_Data_Const(0) =	0x04, 0xc5, 0x06, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00;

Diagnosis

In case of overload or short-circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	2	0...15	47	Short-circuit at an output		Check connection
	11 / 12	ADR	1...10					

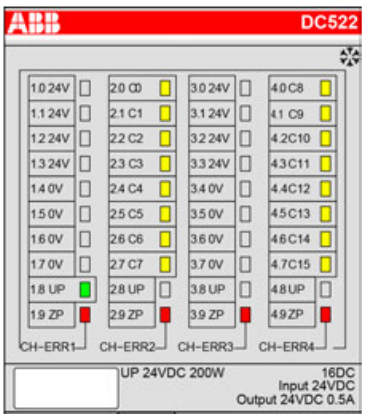
Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs C0...C15	Digital input or digital output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)

Parameter		Value
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.005 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Sensor power supply		
	Connections	Terminals 1.0...1.3 = +24 V, 1.4...1.7 = 0 V Terminals 3.0...3.3 = +24 V, 3.4...3.7 = 0 V
	Voltage	24 V DC with short-circuit and overload protection
	Loadability	Terminals 1.0...1.3, in total max. 0.5 A Terminals 3.0...3.3, in total max. 0.5 A
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		16 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
If the channels are used as inputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8...C15	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8 C15	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Galvanic isolation		From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		Max. 16 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 16 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

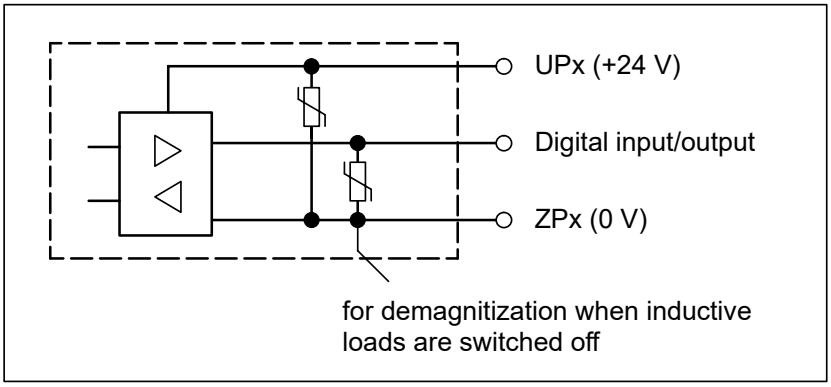


Fig. 842: Digital input/output (circuit diagram)

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C8 / C9
Used outputs	C10
Counting frequency	Max. 50 kHz

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

Ordering data

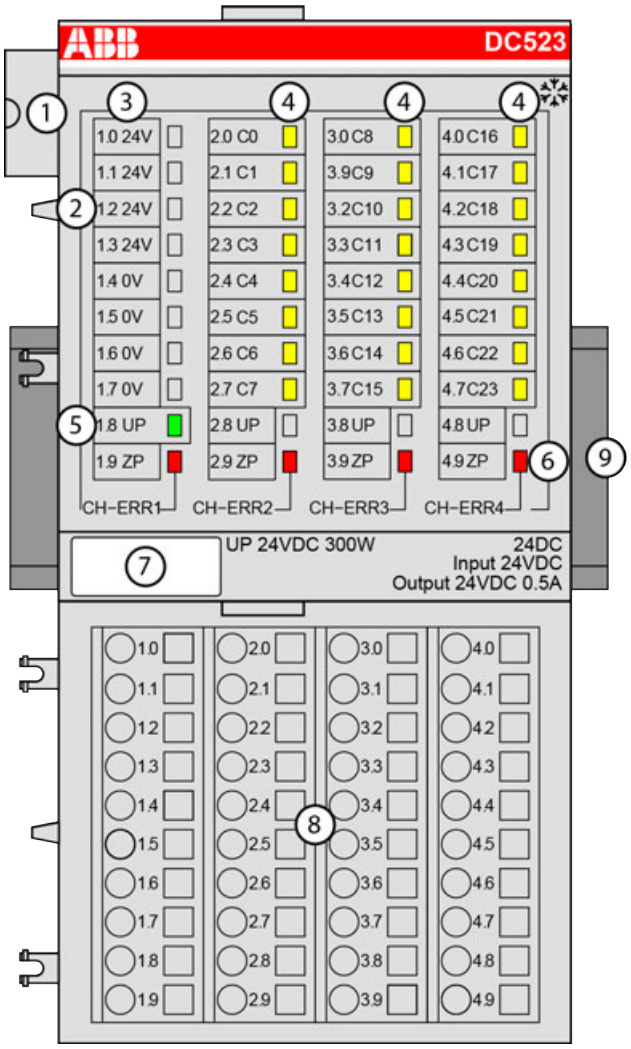
Part no.	Description	Product life cycle phase *)
1SAP 240 600 R0001	DC522, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires	Active
1SAP 440 600 R0001	DC522-XC, digital input/output module, 16 DC, 24 V DC / 0.5 A, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC523 - Digital input/output module

- 24 configurable digital inputs/outputs
- Module-wise galvanically isolated

- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 Sensor power supply 24 V DC / 0.5 A
4 24 yellow LEDs to display the signal states at the digital inputs/outputs (C0 - C23)
5 1 green LED to display the status of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 1 sensor supply voltage 24 V DC, 0.5 A, with short circuit and overload protection
- 24 digital configurable inputs/outputs 24 V DC (C0 to C23) in 1 group (2.0...2.7, 3.0...3.7 and 4.0...4.7), of which each can be used
  - as an input,
  - as a transistor output with short circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

The device is plugged on a terminal unit *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

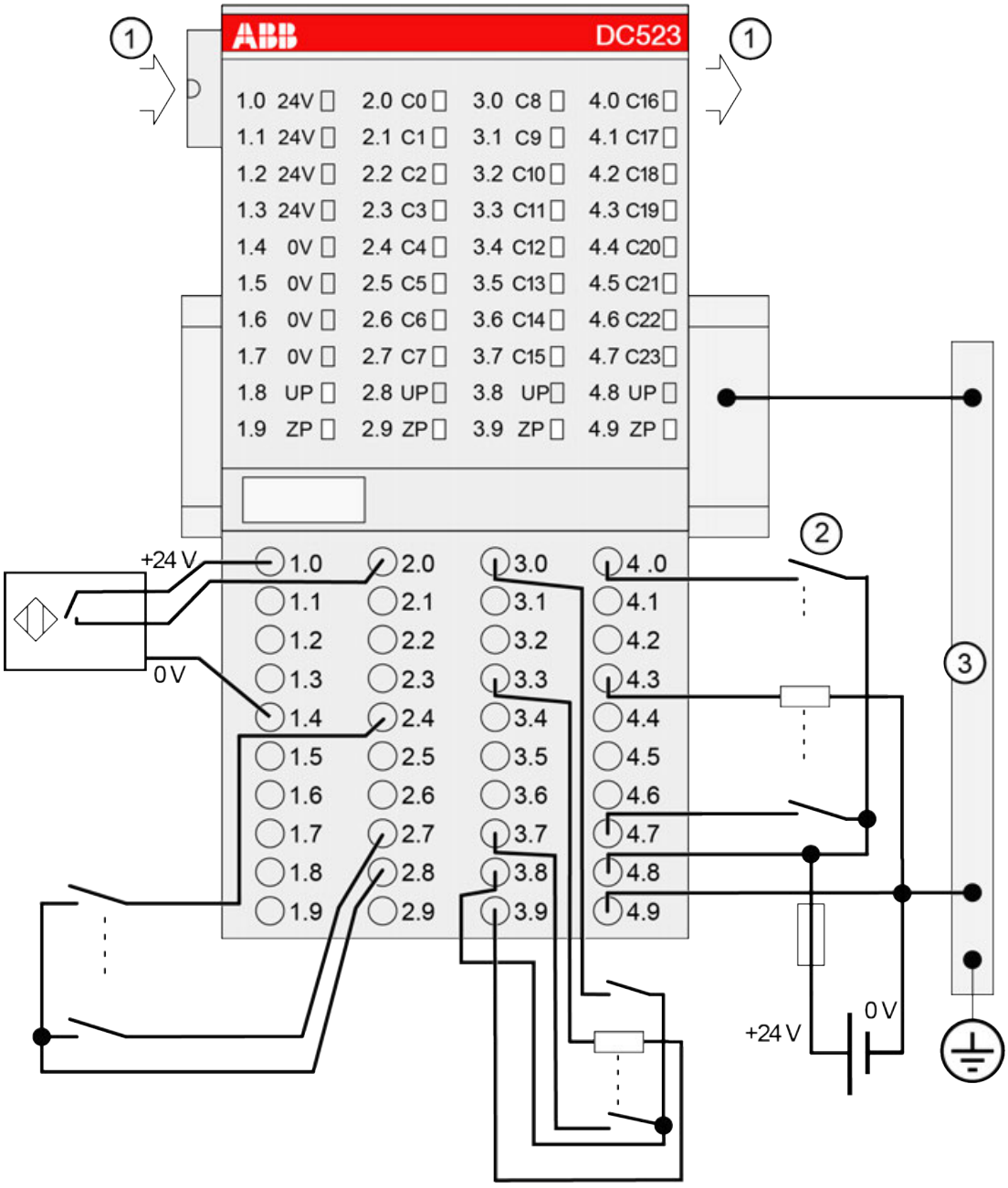
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	+24 V	4 x sensor power supply sources (loadable with 0.5 A in total)
1.4 to 1.7	0 V	0 V (reference potential)
2.0 to 2.7	C0 to C7	8 digital inputs/outputs

Terminals	Signal	Description
3.0 to 3.7	C8 to C15	8 digital inputs/outputs
4.0 to 4.7	C16 to C23	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!
Removal/Insertion under power
Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC523.

Connect a 470 Ω / 1 W resistor in series to inputs C16/C17 if they are used as fast counter inputs to avoid any influences.

The modules provide several diagnosis functions ↬ *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	3	5
Digital outputs (bytes)	3	5
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1215 [1])	Word	1215 0x04BF	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	9	Byte	9-CPU 8-FBP	0	255	0x0Y02

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x=Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Short circuit detection of output or sensor supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs B23 = Output 23 Bit 0 = Output 0	0... 16777215	0... 0x00ff-ffff	DWord	0 0x0000 -0000	0	224−1	0x0Y07

Remarks:

[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section ↬ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
[4])	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	11
Ext_User_Prm_Data_Const(0) =	0x04, 0xc0, 0x08, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00;

Diagnosis

In case of overload or short circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

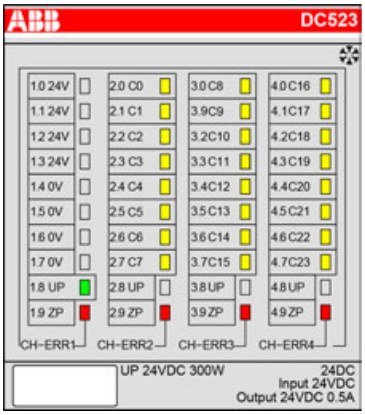
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	2	0...23	47	Short circuit at an output	Check connection
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = Module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs C0...C23	Digital input or digital output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 "System data AC500" on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 "System data AC500-XC" on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.1 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.008 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Sensor power supply		
	Connections	Terminals 1.0...1.3 = +24 V, 1.4...1.7 = 0 V
	Voltage	24 V DC with short circuit and overload protection
	Loadability	Terminals 1.0...1.3, in total max. 0.5 A
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		24 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 24 channels
If the channels are used as inputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8...C15	Terminals 3.0...3.7
	Channels C16...C23	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels C0...C7	Terminals 2.0...2.7
	Channels C8 C15	Terminals 3.0...3.7
	Channels C16...C23	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Galvanic isolation		From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		Max. 24 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA

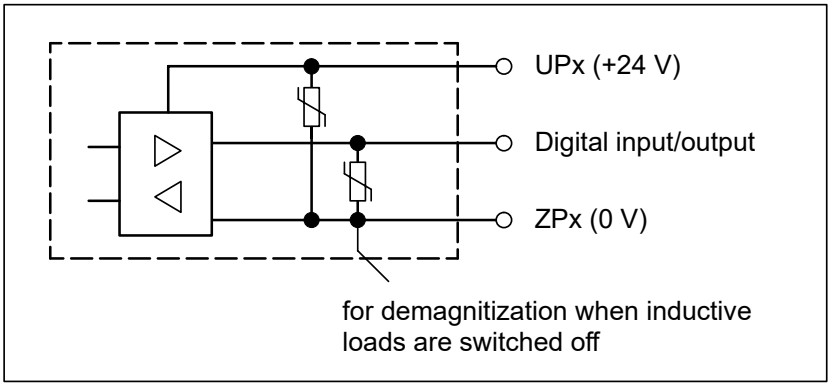
Parameter		Value
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 24 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C16 / C17
Used outputs	C18
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

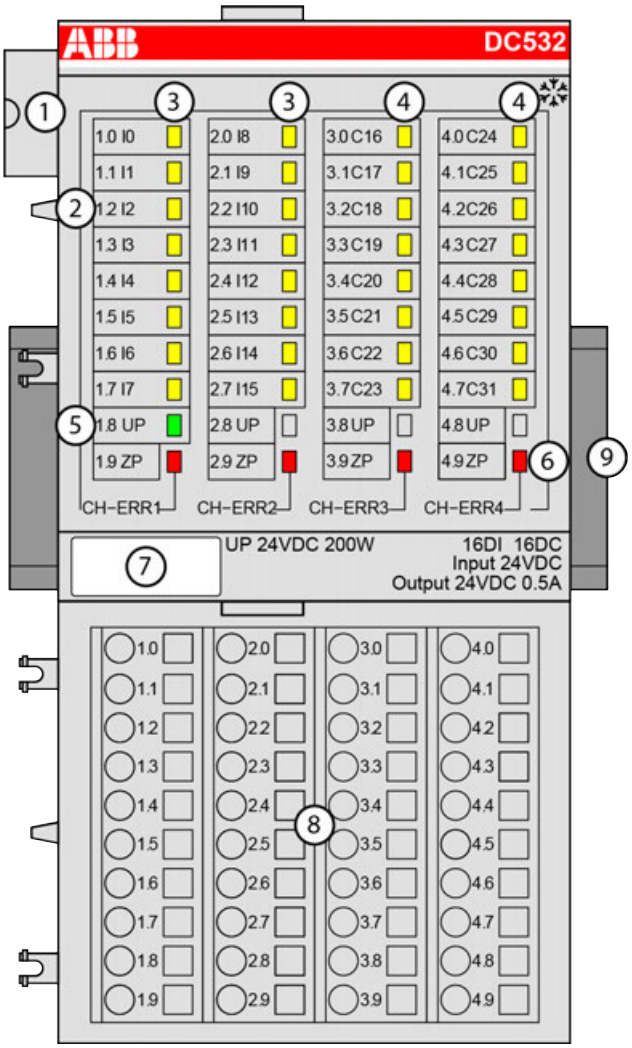
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 500 R0001	DC523, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 500 R0001	DC523-XC, digital input/output module, 24 DC, 24 V DC / 0.5 A, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC532 - Digital input/output module

- 16 digital inputs 24 V DC, 16 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the digital inputs (I0 - I15)
4 16 yellow LEDs to display the signal states at the digital inputs/outputs (C16 - C31)
5 1 green LED to display the state of the process supply voltage UP
6 4 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input / output unit.

- 16 digital inputs 24 V DC in 2 groups (1.0...1.7 and 2.0...2.7)
- 16 digital configurable inputs/outputs 24 V DC (C16 to C31) in 1 group (3.0...3.7 and 4.0...4.7), of which each can be used
  - as an input,
  - as a transistor output with short circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.
- Optional with fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Digital inputs	16 (24 V DC)
Digital inputs/outputs	16 (24 V DC)
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

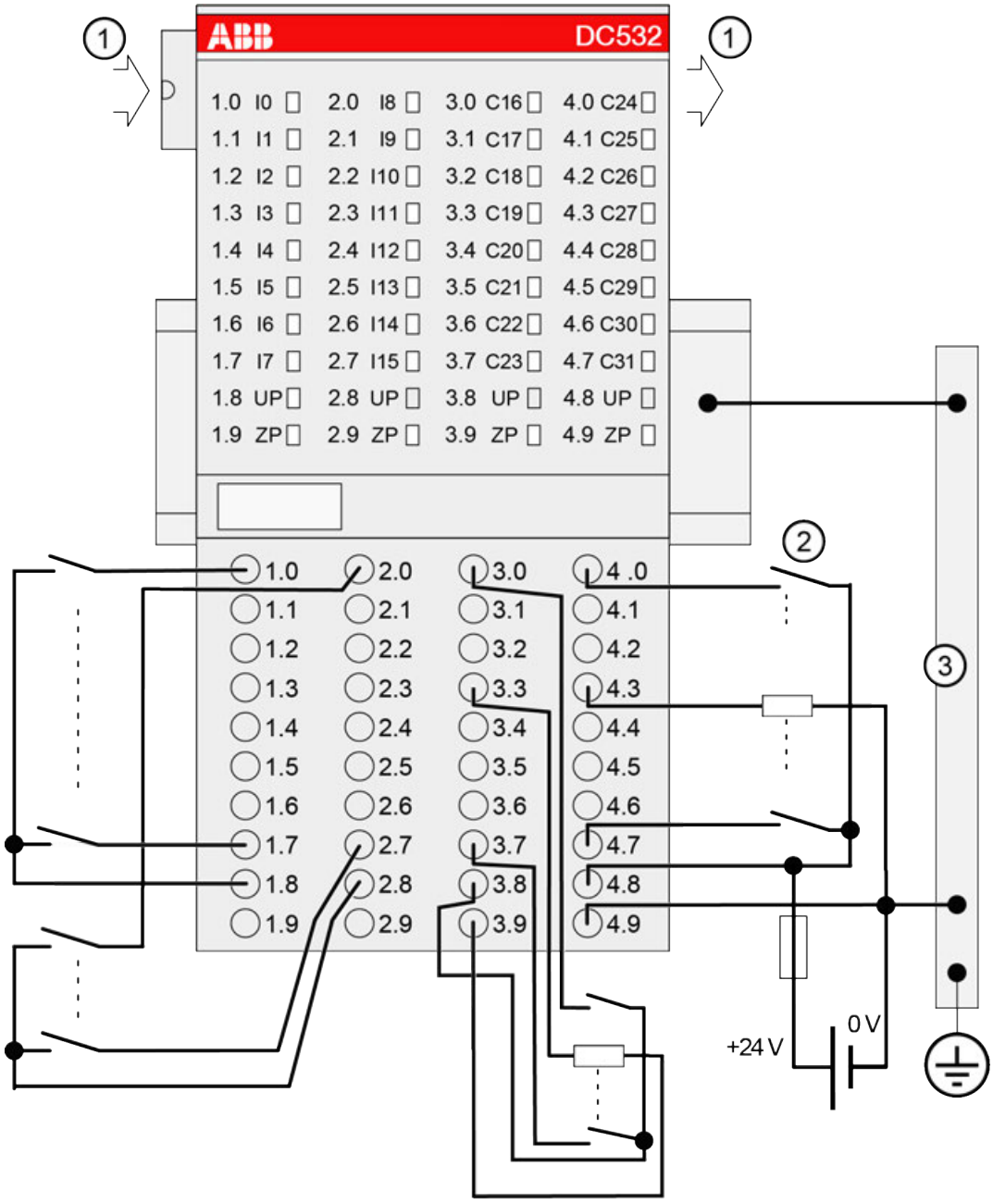
Connections

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	8 digital inputs
2.0 to 2.7	I8 to I15	8 digital inputs
3.0 to 3.7	C16 to C23	8 digital inputs/outputs
4.0 to 4.7	C24 to C31	8 digital inputs/outputs

CAUTION!

The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DC532.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DC532.

Connect a 470 Ω / 1 W resistor in series to inputs C24/C25 if using them as fast counter inputs to avoid any influences.

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	4	6
Digital outputs (bytes)	2	4
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	
Module ID	Internal	1200 [1])	Word	1200 0x04B0	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	7	Byte	7-CPU 6-FBP	0	255	0x0Y02
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Output short circuit detection	Off On	0 1	Byte	On 0x01	0	1	0x0Y05
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y06
Substitute value at outputs Bit 15 = Output 15 Bit 0 = Output 0	0... 65535	0... 0xffff	Word	0 0x0000	0	65535	0x0Y07

Remarks:

[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
[4])	With FBP or CS31 without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	9
Ext_User_Prm_Data_Const(0) =	0x04, 0xb1, 0x06, \ 0x01, 0x02, 0x01, 0x00, 0x00, 0x00;

Diagnosis

In case of overload or short circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore, an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**FBP diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firm-ware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				

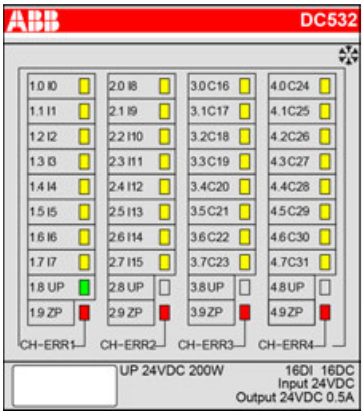
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**FBP diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error DC532								
4	14	1...10	2	16...31	47	Short circuit at a digital output		Check connec-tion
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I15	Digital input	Yellow	Input = OFF	Input = ON [1])	--
	Inputs/ outputs C16...C31	Digital input/ output	Yellow	Input/output = OFF	Input/output = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module Error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA

Parameter		Value
	From UP at normal operation / with outputs	0.15 A + max. 0.5 A per output
	Inrush current from UP (at power up)	0.007 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		16
Distribution of the channels into groups		1 group of 16 channels
Terminals of the channels I0 to I7		1.0 to 1.7
Terminals of the channels I8 to I15		2.0 to 2.7
Reference potential for all inputs		Terminals 1.9, 2.8, 3.8 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module (I/O bus)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	> +5 V...< +15 V Parameter
	Signal 1	+15 V...+30 V

Parameter		Value
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		16 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
If the channels are used as inputs		
	Channels I16...I23	Terminals 3.0...3.7
	Channels I24...I31	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels Q16...Q23	Terminals 3.0...3.7
	Channels Q24...Q31	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Galvanic isolation		From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	Max. 16 digital inputs
Reference potential for all inputs	Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Input current, per channel	See Technical Data of the Digital Inputs ↬ *Chapter 1.6.2.6.1.2.3.9.1 “Technical data of the digital inputs” on page 4286*
Input type acc. to EN 61131-2	Type 1
Input delay (0->1 or 1->0)	Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage	24 V DC

Parameter		Value
Signal 0		-3 V...+5 V *)
undefined signal		> +5 V...< +15 V
Signal 1		+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

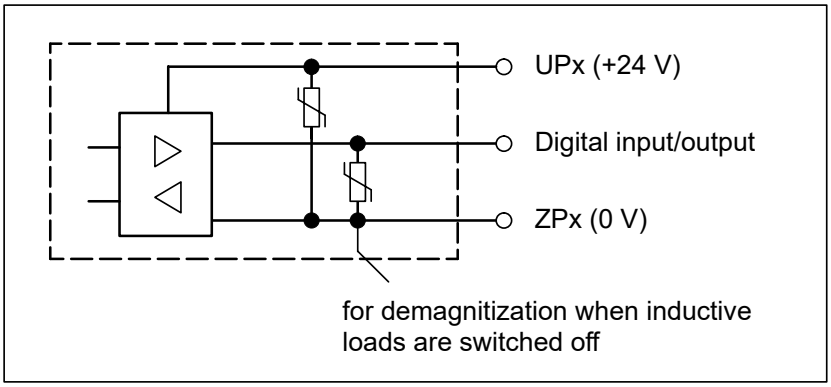
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 16 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		

Parameter		Value
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	C24 / C25
Used outputs	C26
Counting frequency	Max. 50 kHz

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

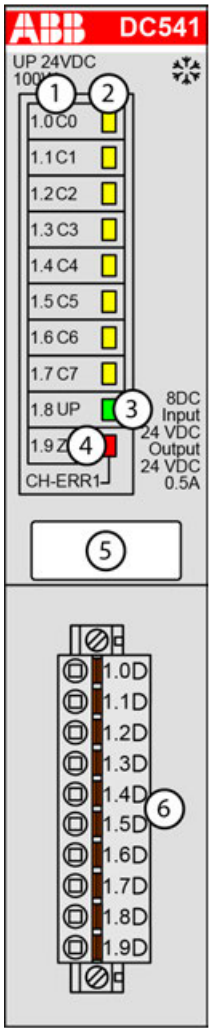
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 100 R0001	DC532, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 100 R0001	DC532-XC, digital input/output module, 16 DI, 16 DC, 24 V DC / 0.5 A, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC541-CM - Digital input/output module

- 8 configurable digital inputs/outputs 24 V DC, in a communication module housing
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 Allocation between terminal number and signal name
2 8 yellow LEDs to display the signal states at the inputs/outputs C0 to C7
3 1 green LED to display the state of the process supply voltage UP
4 1 red LED to display errors (CH-ERR1)
5 Label
6 Terminal block with 10 terminals for 8 inputs/outputs and process power supply (ZP/UP)
❄ Sign for XC version

Intended purpose

In contrast to other I/O modules, the digital I/O module (multi-function module) DC541-CM is connected to a communication module slot to the left of the AC500 CPU. It contacts the internal communication module bus. This way, the full functionality of the communication module bus is available for the module DC541-CM. Depending on the terminal base TB5x1 used, up to 4 DC541-CM modules can be connected.

The multi-function module DC541-CM can optionally (not at the same time) be configured as an interrupt module or as a fast counter module for 24 V signals (e.g. 24 V incremental encoder).Automation Builder is used for the configuration.

The module contains 8 fast channels (C0...C7) with the following features:

- 8 digital inputs/outputs in one group (1.0...1.7), of which each can be used
  - as an input,
  - as a transistor output with short-circuit and overload protection, 0.5 A rated current or
  - as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.

The states of the inputs/outputs are indicated by yellow LEDs (one per channel). There is no potential separation between the channels.

Functionality

Parameter	Value
Digital inputs/outputs	8 (24 V DC)
Fast counter	Integrated, many configurable operating modes
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the communication module bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

In the operating mode Interrupt I/O module, the channels can be configured as follows:

- Input
- Output
- Interrupt input

In this way, important input information can be evaluated independently of the program cycle and outputs can be set.

In the operating mode Counter, the channels can be configured as follows:

- Input
- Output
- 32-bit bidirectional counter (uses C0...C3) as a 32-bit-counter without limit
- 32-bit periodic counter as a 32-bit counter with a limit
- Limiter for a 32-bit counter (limit channel 0)
- 32-bit count up counter (forward counter) with the frequencies 50 kHz, 5 kHz and 2.5 kHz
- Pulse-width modulation (PWM) with a resolution of 10 kHz
- Time and frequency measurement
- Frequency output

Used as a fast counter module, the 8 channels of the multi-function module DC541-CM can be configured and combined individually, easily and versatilely in the PLC configuration. The module is therefore also excellent for universal high-frequency counting tasks up to 50 kHz. In addition, it has measuring functions for rotational speed, time and frequency.

These different channel configurations can now be combined flexibly on-board.

Example 1: 32-bit bidirectional counter incl. zero trace and touch-trigger for max. 50 kHz plus 4 accompanying limiting values (comparison values). When the counter reaches one of the comparison values, the corresponding output can be set in order to trigger control functions at the machine or installation directly.

Example 2: 2 counters for 50 kHz plus frequency measurement with a resolution of 200 µs plus 4 digital I/Os.

Further examples and a detailed description of the fields of application are contained in the chapter "System Technology of *Chapter 1.6.4.4.1 “DC541-CM interrupt and counter module” on page 5685*.

Commissioning is carried out via the user program by using the appropriate function blocks.

Connections

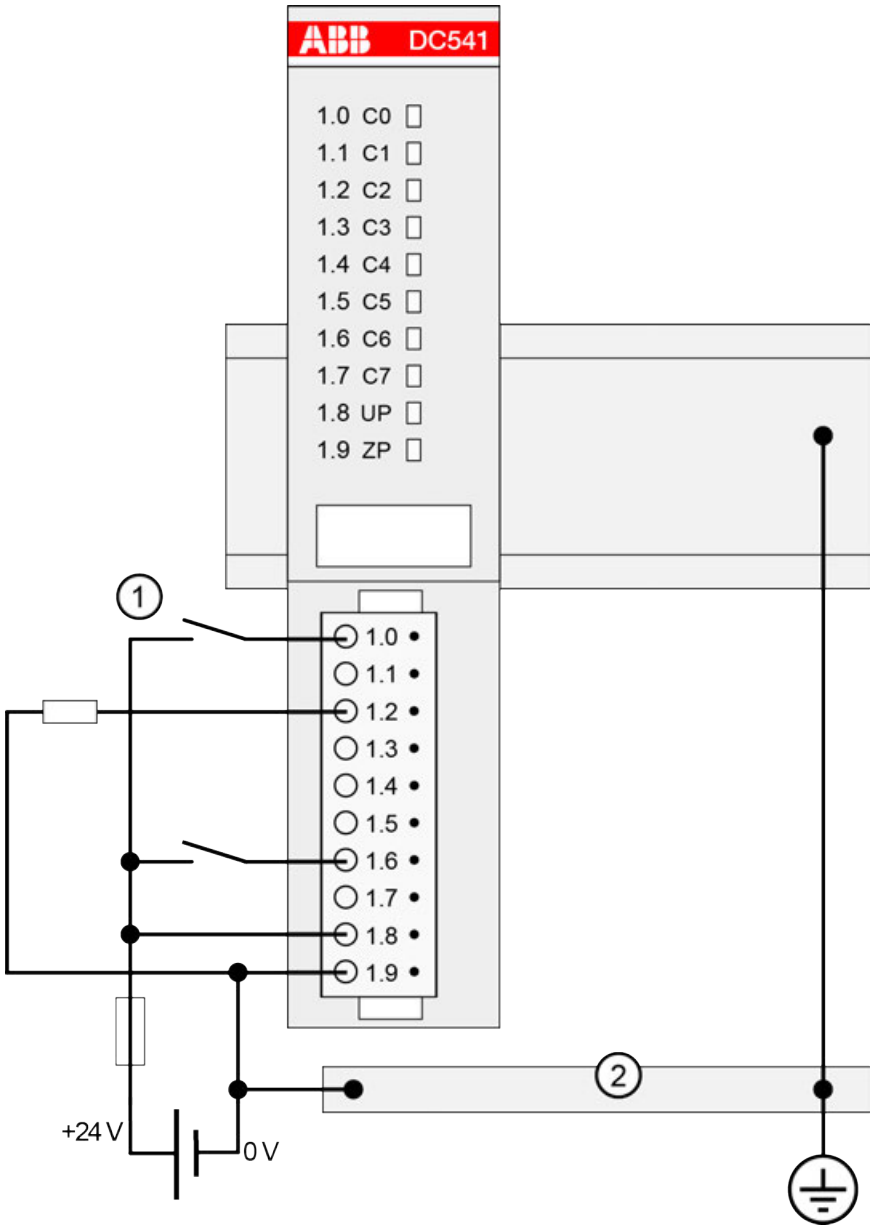
The I/O module DC541-CM is mounted to the left of an AC500 CPU on the same terminal base. The connection to the communication module bus is automatically established while mounting.

The connection of the I/O channels is carried out using the 10 terminals of the removable terminal block. I/O modules can be replaced without re-wiring.

The process voltage is connected in the following way:

Terminal 1.8: process voltage UP = +24 V DC

Terminal 1.9: process voltage ZP = 0 V DC



1 1.0 - 1.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
2 Switchgear cabinet earth

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	C0 to C7	8 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The internal supply voltage for the module's circuitry comes from the communication module bus. The process voltage for the inputs/outputs is provided via ZP and UP.

WARNING!
Removal/Insertion under power
The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!
Risk of damaging the PLC modules!
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC522.

Connect a 470 Ω / 1 W resistor in series to inputs C8/C9 if they are used as fast counter inputs to avoid any influences.

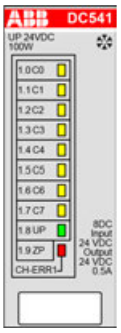
The module provides several diagnostic functions *Chapter 1.6.2.6.1.2.4.5 “State LEDs” on page 4294*).

I/O configuration and parameterization

The DC541-CM module does not store configuration data itself. Configuration and parameterization are performed with Automation Builder software. *Chapter 1.6.5.2.8.1 “DC541-CM interrupt and counter module” on page 5974*

State LEDs

In case of overload or short-circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore, an acknowledgement of the outputs is not necessary.

LED		State	Color	LED = OFF	LED = ON
	Inputs/ outputs C0...C7	Digital input or digital output	Yellow	Input/output = OFF	Input/output = ON
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK and initialization terminated
	CH-ERR1	Module Error	Red	No error	Error

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8 for +24 V (UP) and 1.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Absolute limits at XC version	Above 60 °C: 20 V DC...30 V DC
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
Galvanic isolation		Yes, per module
Current consumption		
	From 24 V DC power supply at the Terminal Base of the CPU	10 mA

Parameter		Value
	Current consumption from UP at normal operation / with outputs	10 mA + 5 mA per input
	Inrush current from UP (at power up)	0.002 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Max. power dissipation within the module		On request
Weight (without terminal block)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Altitude		> 2000 m: On request

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter	Value
Number of channels per module	8 inputs/outputs (with transistors)
Distribution of the channels into groups	1 group of 8 channels
If the channels are used as inputs	
Channels C0...C7	Terminals 1.0...1.7
If the channels are used as outputs	
Channels C0...C7	Terminals 1.0...1.7
Reference potential for all inputs/outputs	Terminal 1.9 (ZP = Negative pole of the process supply voltage)
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	From the rest of the module

Technical data of the digital inputs/outputs if used as inputs

Parameter	Value
Number of channels per module	Max. 8 digital inputs
Reference potential for all inputs	Terminal 1.9 (negative pole of the process supply voltage, signal name ZP)
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA
Input voltage +15 V	> 5 mA
Input voltage +30 V	< 8 mA
Input type acc. to EN 61131-2	Type 1
Input delay (0 -> 1 or 1 -> 0)	Typ. 2 µs
Input signal voltage	24 V DC
Signal 0	-3 V...+5 V *)
Undefined signal	> +5 V...< +15 V
Signal 1	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V *)
Ripple with signal 1	Within +15 V...+30 V
Max. cable length	
Shielded	1000 m
Unshielded	600 m

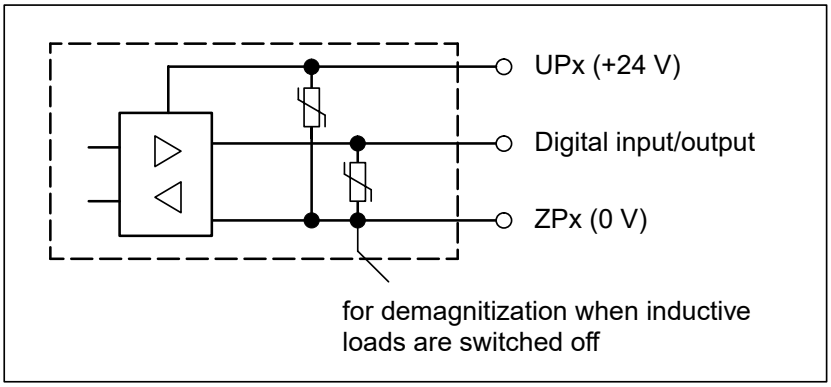
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	Max. 8 transistor outputs
Common power supply voltage	For all outputs: terminal 1.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1	UP (-0.8 V)
Output delay (0 -> 1 or 1 -> 0)	Typ. 10 µs
Output current	
Rated value, per channel	500 mA at UP = 24 V
Maximum value (all channels together)	8 A
Leakage current with signal 0	< 0.5 mA
Rated protection fuse for UP	10 A fast
De-magnitization when inductive loads are switched off	With varistors integrated in the module (see figure below)
Switching frequency	
With resistive load	On request

Parameter	Value
With inductive loads	Max. 0.5 Hz
With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



Technical data of the fast counters

Parameter	Value
Used inputs for the traces A and B	C0 / C1
Used input for the zero trace, touch trigger	C2 / C3
Used outputs	C4 to C7, if needed
Operating modes	↳ *Chapter 1.6.2.6.1.2.4.2 “Functionality” on page 4291*

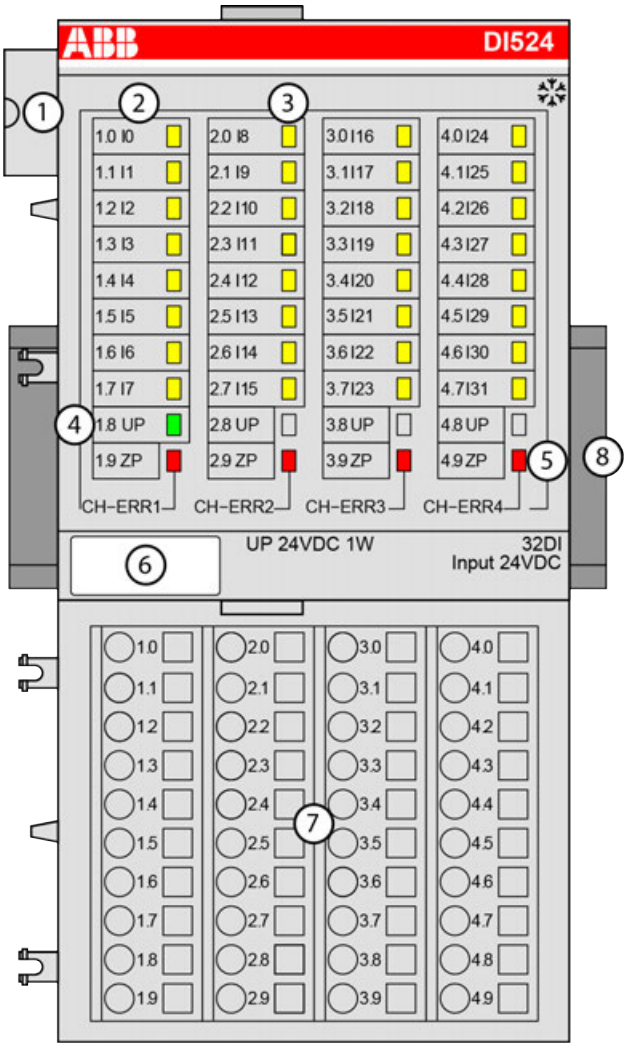
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 270 000 R0001	DC541-CM, digital input/output module, 8 DC, 24 V DC / 0.5 A, 1-wire	Active
1SAP 470 000 R0001	DC541-CM-XC, digital input/output module, 8 DC, 24 V DC / 0.5 A, 1-wire, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DI524 - Digital input module

- 32 digital inputs 24 V DC in 4 groups (1.0...1.7, 2.0...2.7, 3.0...3.7 and 4.0...4.7)
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 32 yellow LEDs to display the signal states at the digital inputs (I0 - I31)
4 1 green LED to display the state of the process supply voltage UP
5 4 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Via the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal units	TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

The device is plugged on a terminal unit *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, irrespective of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

Table 423: Assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	8 digital inputs
2.0 to 2.7	I8 to I15	8 digital inputs
3.0 to 3.7	I16 to I23	8 digital inputs
4.0 to 4.7	I24 to I31	8 digital inputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DI524.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

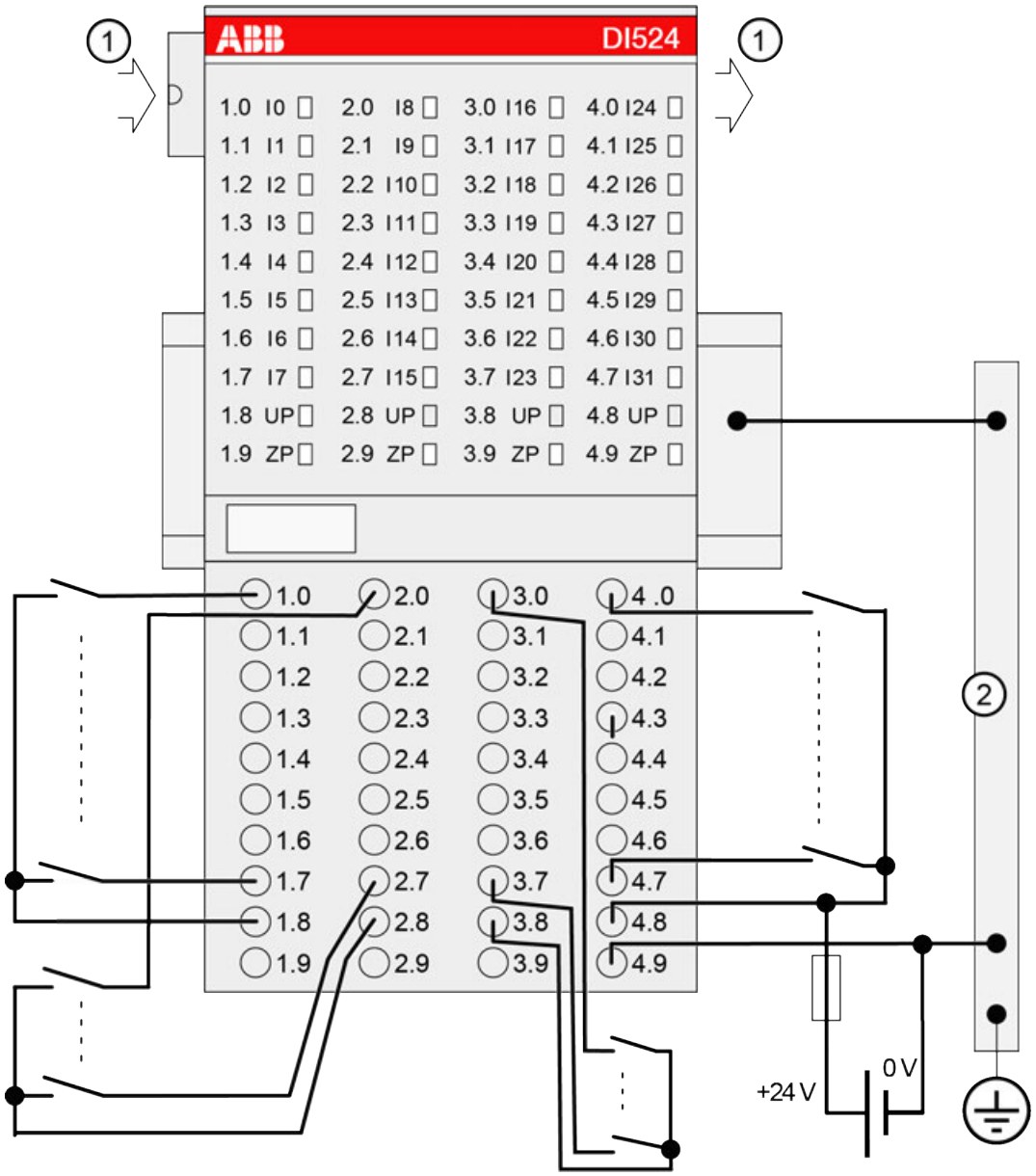
The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.



1 I/O bus
2 switchgear cabinet earth

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	4	6
Digital outputs (bytes)	0	2

	Without the fast counter	With the fast counter (only with AC500)
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1000 [1])	Word	1000 0x03E8	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length	Internal	3-CPU 2-FBP	Byte	3 2	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
5	Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
6	Fast counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP

Remarks:

[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
[4])	With FBP or CS31 without the parameter Fast counter

GSD file:

Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	5 0x03, 0xe9, 0x02, \ 0x01, 0x02;

Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diag-nosis block		
Class	Interface	Device	Module	Channel	Error identifier	Error message		Remedy
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	

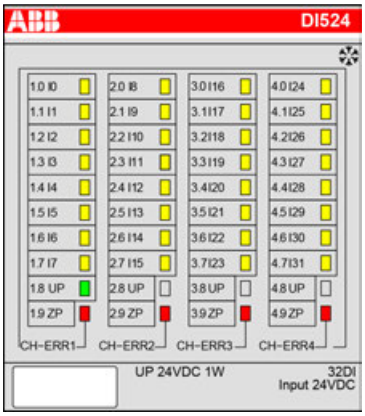
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I31	Digital input	Yellow	Input = OFF	Input = ON [1])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [2])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	ca. 2 mA
	From UP at normal operation	0.15 A
	Inrush current from UP (at power up)	0.008 A^2s
Weight (without terminal unit)		ca. 105 g

Parameter	Value
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		32
Distribution of the channels into groups		1 group of 32 channels
Terminals of the channels I0 to I7		1.0 to 1.7
Terminals of the channels I8 to I15		2.0 to 2.7
Terminals of the channels I16 to I23		3.0 to 3.7
Terminals of the channels I24 to I31		4.0 to 4.7
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module (I/O bus)
Indication of the input signals		One yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0 -> 1 or 1 -> 0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		

Parameter		Value
	Shielded	1000 m
	Unshielded	600 m

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	I24 / I25
Used outputs	None
Counting frequency	Max. 50 kHz

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

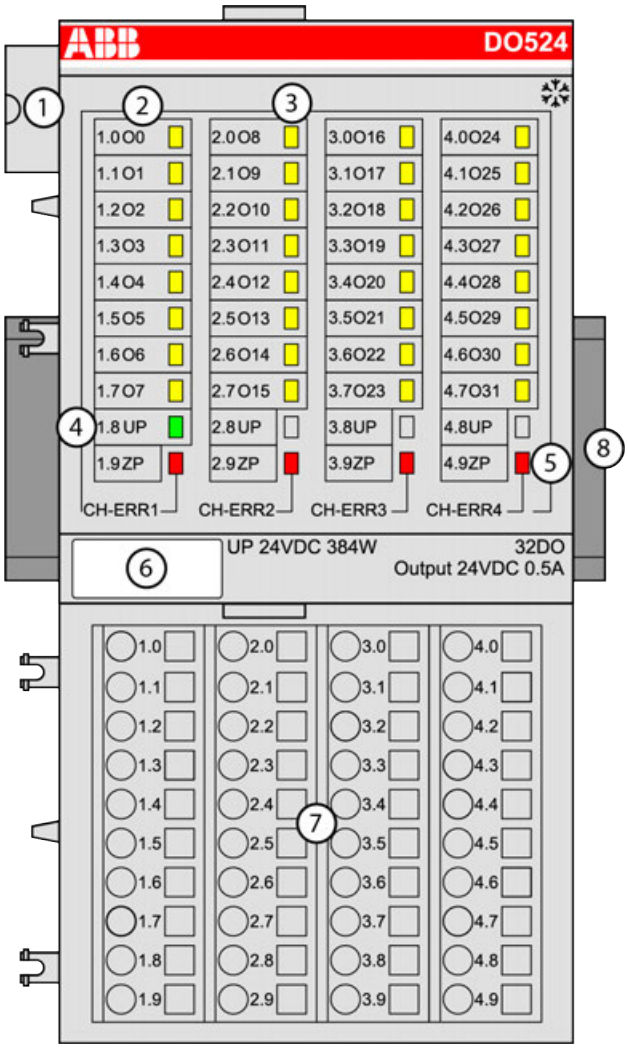
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 000 R0001	DI524, digital input module, 32 DI, 24 V DC, 1-wire	Active
1SAP 440 000 R0001	DI524-XC, digital input module, 32 DI, 24 V DC, 1-wire, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DO524 - Digital output module

- 32 digital outputs 24 V DC / 0.5 A in 4 groups (1.0...4.7) with short circuit and overload protection
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 32 yellow LEDs to display the signal states at the digital outputs (O0 - O31)
4 1 green LED to display the state of the process supply voltage UP
5 4 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltage
Internal power supply	Via the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit	TU515 or TU516 ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

The device is plugged on a terminal unit ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	O0 to O7	8 digital outputs
2.0 to 2.7	O8 to O15	8 digital outputs
3.0 to 3.7	O16 to O23	8 digital outputs
4.0 to 4.7	O24 to O31	8 digital outputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DO524.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

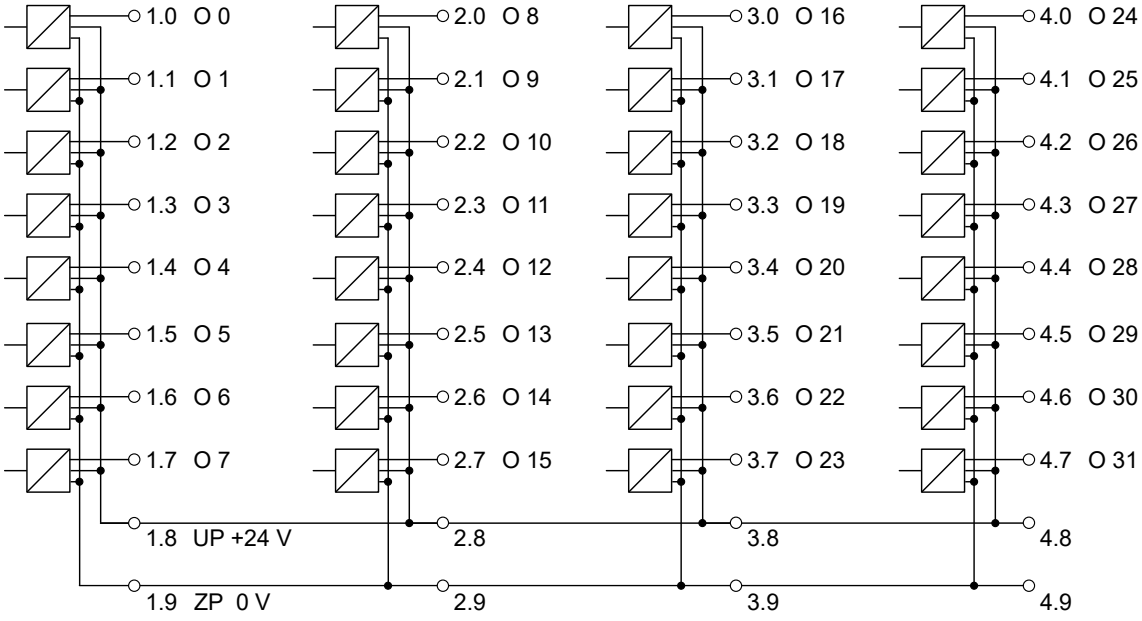
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following block diagram shows the internal construction of the digital outputs:



The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	4

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Module ID	Internal	1101 [1])	WORD	1101 0x044D	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	BYTE	No 0x00			not for FBP
Parameter length	Internal	7	BYTE	7-CPU 7-FBP	0	255	0x0Y02
Check supply	Off on	0 1	BYTE	On 0x01	0	1	0x0Y03
Output short circuit detection	Off On	0 1	BYTE	On 0x01	0	1	0x0Y04

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	BYTE	Off 0x00	0	2	0x0Y05
Substitute value at outputs Bit 31 = Output 31 Bit 0 = Output 0	0... 4294967295	0... 0xffffffff	DWORD	0 0x00000000	0	4294967295	0x0Y06

[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1

[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	10
Ext_User_Prm_Data_Const(0) =	0x04, 0x4d, 0x07, \ 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00;

Diagnosis

In case of overload or short circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore, an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	1)	**2)**	**3)**	**4)**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module

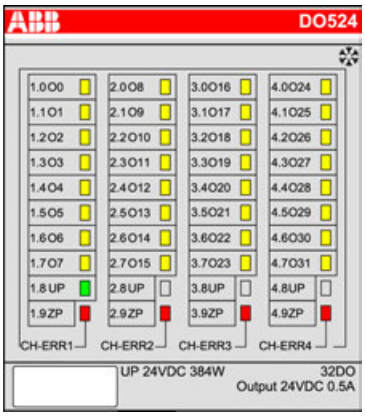
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	2	0...31	47	Short circuit at a digital output		Check connection
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = Hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O31	Digital output	Yellow	Output = OFF	Output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.10 A + max. 0.5 A per output

Parameter		Value
	Inrush current from UP (at power up)	0.005 A^2s
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		Ca. 100 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

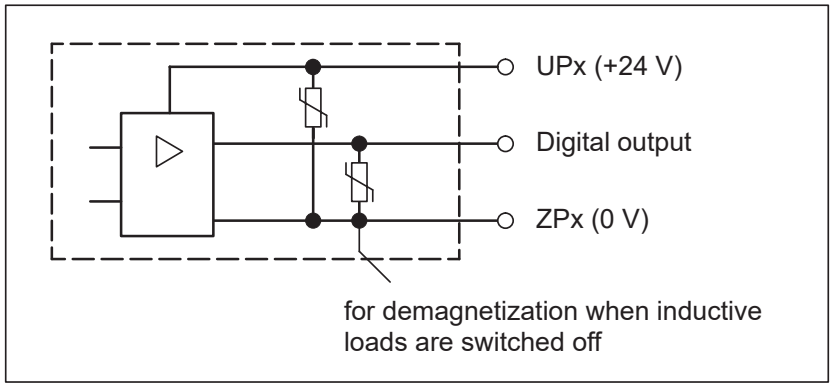
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		32 outputs (with transistors)
Distribution of the channels into groups		1 group of 32 channels
Connection of the channels		
	O0 to O7	Terminals 1.0 to 1.7
	O8 to O15	Terminals 2.0 to 2.7
	O16 to O23	Terminals 3.0 to 3.7
	O24 to O31	Terminals 4.0 to 4.7
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0 -> 1 or 1 -> 0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (channels O0 to O15)	4 A
	Maximum value (channels O16 to O31)	4 A

Parameter		Value
	Maximum value (all channels together)	8 A
Max. leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit proof / overload proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short-circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital output with the varistors for demagnetization when inductive loads are switched off.



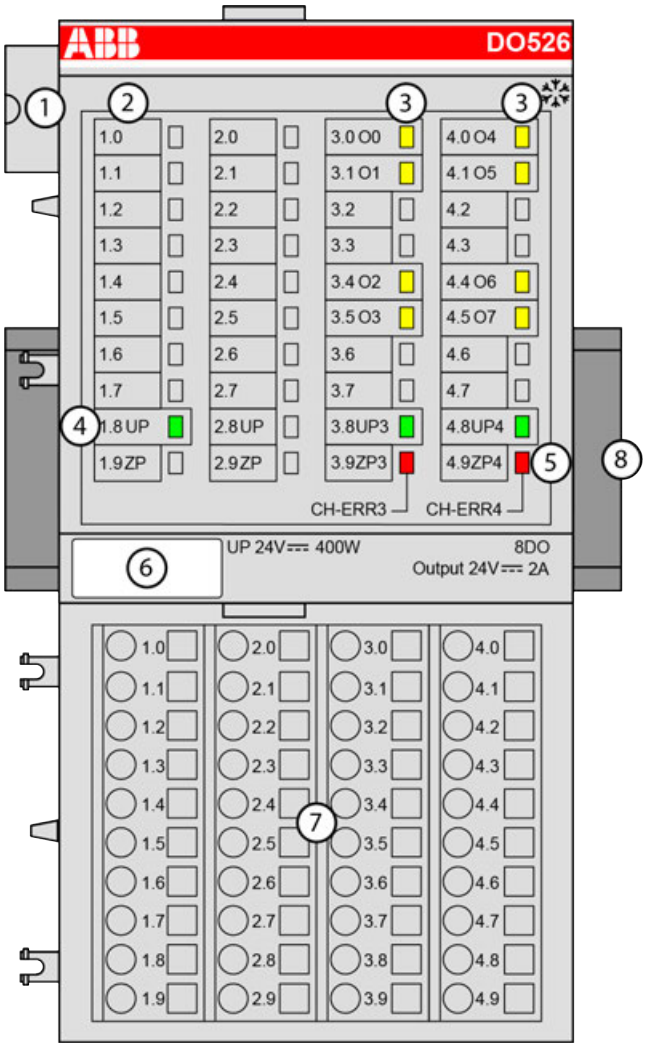
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 700 R0001	DO524, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire	Active
1SAP 440 700 R0001	DO524-XC, digital output module, 32 DO, 24 V DC / 0.5 A, 1-wire, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

DO526 - Digital output module

- 8 digital outputs 24 V DC (O0 to O7) in 2 groups without short circuit and without overload protection.
- Module and group-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the outputs O0 to O7
4 3 green LEDs to display the states of the process supply voltage UP, UP3 and UP4
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN-rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are group-wise galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the outputs.

Potential separation between the channel groups.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltages
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP, ZP3, ZP4, UP, UP3 and UP4 (process voltage 24 V DC)
Required terminal unit	TU542 ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

The output module is plugged on the terminal unit TU542. Properly position the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 2.8 and 1.9 to 2.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 1.8 to 2.8:	Process voltage UP = +24 V DC
Terminals 1.9 to 2.9:	Process voltage ZP = 0 V
Terminal 3.8:	Process voltage UP3 = +24 V DC
Terminal 3.9:	Process voltage ZP3 = 0 V
Terminal 4.8:	Process voltage UP4 = +24 V DC
Terminal 4.9:	Process voltage ZP4 = 0 V

Terminals	Signal	Description
3.0, 3.1, 3.4, 3.5	O0 to O3	4 digital outputs
4.0, 4.1, 4.4, 4.5	O4 to O7	4 digital outputs

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DO526.

The external power supply connection is carried out via the UP, UP3, UP4 (+24 V DC) and the ZP, ZP3, ZP4 (0 V DC) terminals.

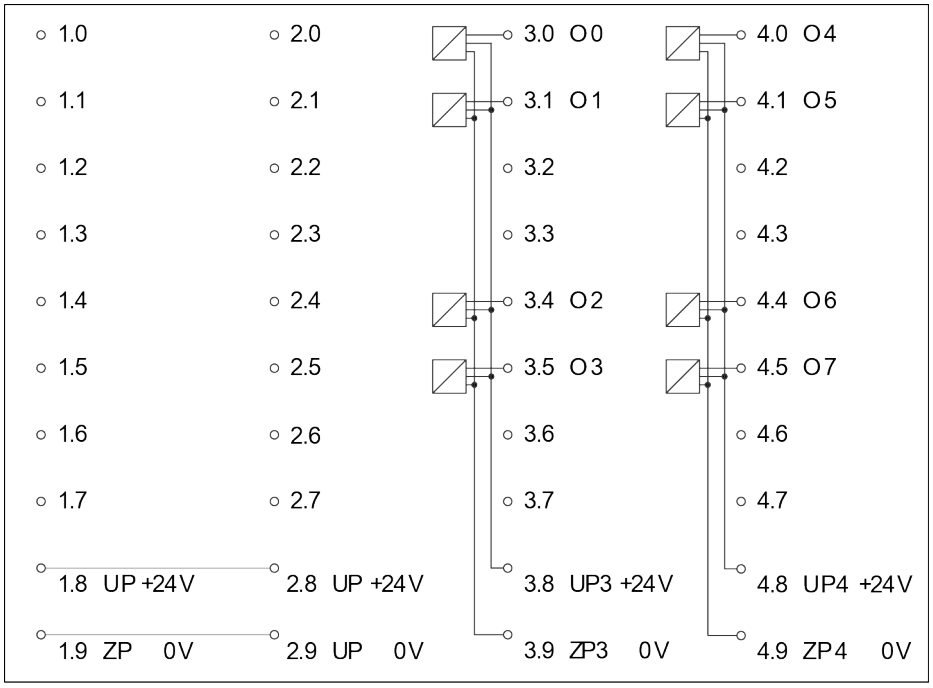
NOTICE!

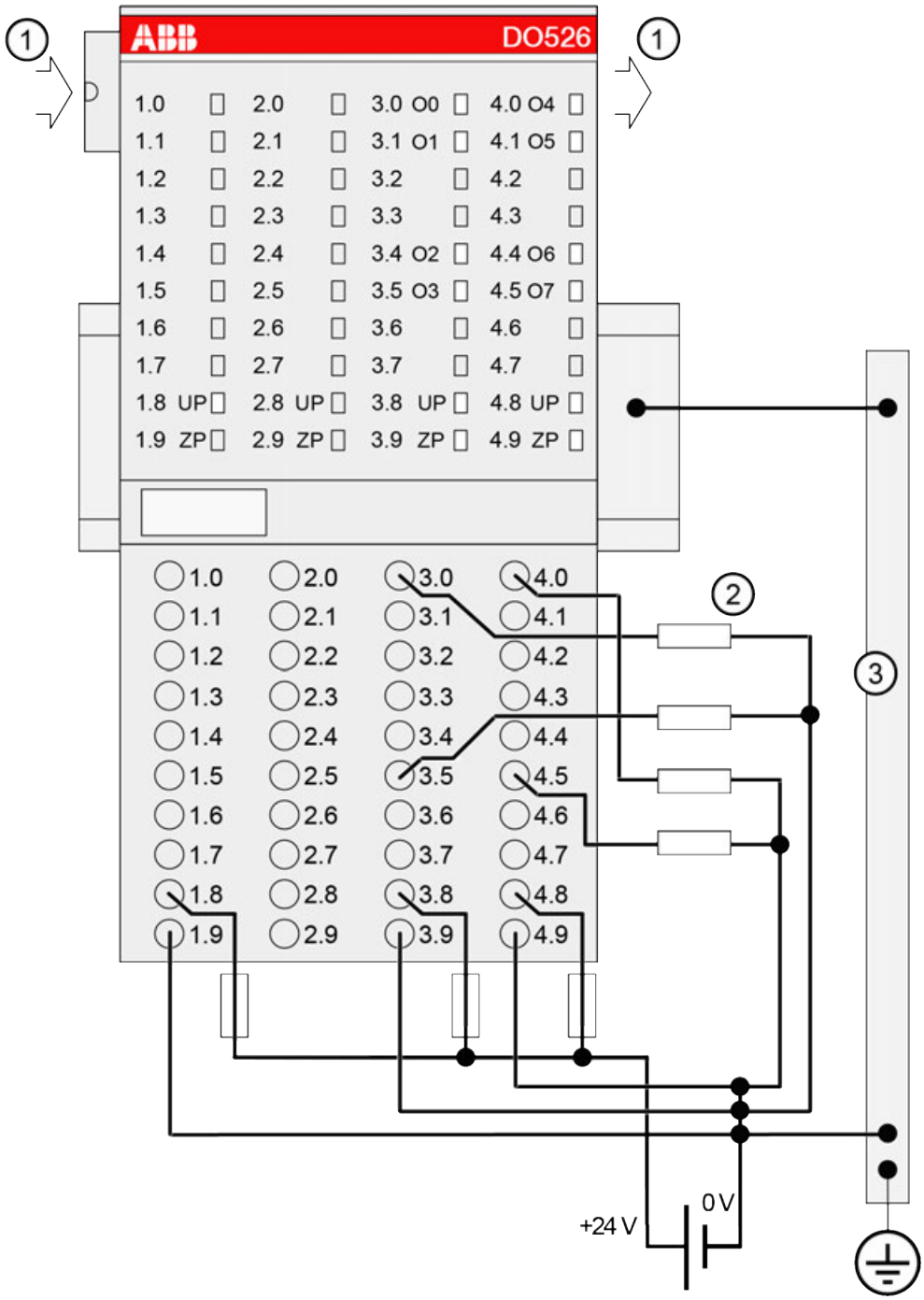
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following block diagram shows the internal construction of the digital outputs:





1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	1

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software, versions ≥ 1.2.3.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...7

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Module ID	Internal	1105 [1])	WORD	1105 0x0451	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	BYTE	No 0x00			not for FBP
Parameter length	Internal	6	BYTE	6-CPU 6-FBP	0	6	0x0Y02
Check supply	Off on	0 1	BYTE	On 0x01	0	1	0x0Y03
Reserve	0...255	0...0xff	BYTE	On 0x01	0	1	0x0Y04
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	BYTE	Off 0x00	0	2	0x0Y05

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	Max.
Substitute value at outputs Bit 7 = Output 7 Bit 0 = Output 0	0...255	0...0xff	BYTE	0x00	0	255	0x0Y06
Reserve	0...255	0...0xff	BYTE	0x00	0	255	0x0Y07
Reserve	0...255	0...0xff	BYTE	0x00	0	255	0x0Y08
[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1 [2]) Not with FBP							

GSD file:

Ext_User_Prm_Data_Len =	10
Ext_User_Prm_Data_Const(0) =	0x04, 0x51, 0x00, 0x06, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					

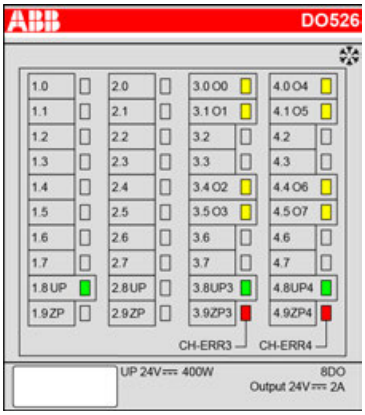
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage UP3 and/or UP4 too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage UP is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	31	0(UP3)	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10	4(UP4)				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O7	Digital output	Yellow	Output = OFF	Output = ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	UP3	Process supply voltage outputs 0...3 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	UP4	Process supply voltage outputs 4...7 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR3	Channel Error, error messages in groups (digital outputs combined into the groups 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on in the corresponding group
	CH-ERR4		Red			
	CH-ERR [1])	Module Error	Red	--	Internal error	--
	[1]) All of the LEDs CH-ERR3 to CH-ERR4 light up together [2]) The state of the LEDs corresponds to the logic state of the output. In case of missing or low process supply voltage UP3 or UP4, the signal on the output terminal is off even though the LED is on.					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP, UP3 and UP4		
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) as well as 1.9 and 2.9 0 V (ZP) Terminals 3.8 for +24 V (UP3) as well as 3.9 for 0 V (ZP3) Terminals 4.8 for +24 V (UP4) as well as 4.9 for 0 V (ZP4)

Parameter		Value
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP, UP3 and UP4	10 A fast (for each process supply voltage)
	Galvanic isolation	Yes, per module and per output channel groups
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	Ca. 20 mA + 1.5 mA per output
	From UP3 or UP4 at normal operation / with outputs	Ca. 0.01 A + max. 2 A per output
	Inrush current from UP (at power up)	0.015 A^2s
	Inrush current from UP3 or UP4 (at power up)	0.005 A^2s (without output load)
Max. power dissipation within the module		6 W
Weight (without terminal unit)		Ca. 135 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8 outputs (with transistors, non-latching type)
Distribution of the channels into groups		2 groups of 4 channels
Connection of the channels		
	O0 to O3	Terminals 3.0, 3.1, 3.4, 3.5
	O4 to O7	Terminals 4.0, 4.1, 4.4, 4.5
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)

Parameter		Value
Power supply voltage for the module		Terminals 1.8 and 2.8 (positive pole of the process supply voltage, signal name UP)
Reference potential for module power supply		Terminals 1.9 and 2.9 (negative pole of the process supply voltage, signal name ZP)
Power supply voltage for the outputs O0 to O3		Terminal 3.8 (positive pole of the process supply voltage, signal name UP3)
Reference potential for the outputs O0 to O3		Terminal 3.9 (negative pole of the process supply voltage, signal name ZP3)
Power supply voltage for the outputs O4 to O7		Terminal 4.8 (positive pole of the process supply voltage, signal name UP4)
Reference potential for the outputs O4 to O7		Terminal 4.9 (negative pole of the process supply voltage, signal name ZP4)
Output voltage for signal 1		UP (-0.4 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	2 A at UP3 or UP4 = 24 V
	Maximum value (channels O0 to O3)	8 A
	Maximum value (channels O4 to O7)	8 A
Leakage current with signal 0		< 0.1 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With clamp diode in output high side driver
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 2 Hz
	With lamp loads	Max. 11 Hz with max. 48 W
Short-circuit proof / overload proof		No (should be done externally)
Overload message		No
Output current limitation		No (should be done externally)
Resistance to feedback against 24 V signals		Yes to UP3 or UP4. No to outputs in same group.
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

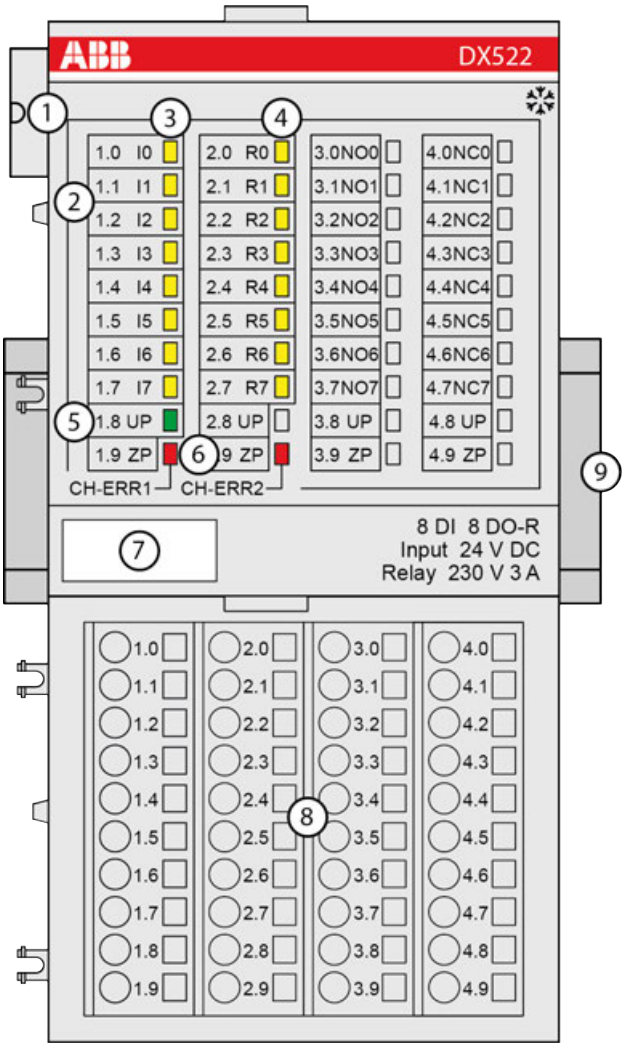
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 240 800 R0001	DO526, digital output module, 8 DO, 24 V DC / 2 A, 1-wire	Active
1SAP 440 800 R0001	DO526-XC, digital output module, 8 DO, 24 V DC / 2 A, 1-wire, XC version	Active

Part no.	Description	Product life cycle phase *)
1SAP 213 200 R0001	TU542, I/O terminal unit, 24 V DC, spring terminals	Active
1SAP 413 200 R0001	TU542-XC, I/O terminal unit, 24 V DC, spring terminals, XC version	Active

DX522 - Digital input/output module

- 8 digital inputs 24 V DC, module-wise galvanically isolated
- 8 relay outputs
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the digital inputs (I0 - I7)
4 8 yellow LEDs to display the signal states at the digital relay outputs (R0 - R7)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input/output unit.

- 8 digital inputs 24 V DC in 1 group (1.0...1.7)
- 8 digital relay outputs with one change-over contact each (R0...R7). All output channels are galvanically isolated from each other.
- Fast counter

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes (only with AC500)
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)
Required terminal units	TU531 or TU532 ↳ *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*

The device is plugged on a terminal unit ↳ *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

Connections

WARNING!
Risk of death by electric shock!
Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and have always the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V DC

Table 424: Assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0 to I7	Input signals of the 8 digital inputs
1.8 to 4.8	UP	Process supply voltage +24 V DC
1.9 to 4.9	ZP	Reference potential for the 8 digital inputs and the process supply voltage
2.0	R0	Common contact of the first relay output
3.0	NO 0	Normally-open contact of the first relay output
4.0	NC 0	Normally-closed contact of the first relay output
2.1	R1	Common contact of the second relay output
3.1	NO 1	Normally-open contact of the second relay output
4.1	NC 1	Normally-closed contact of the second relay output
:	:	:
2.7	R7	Common contact of the eighth relay output
3.7	NO 7	Normally-open contact of the eighth relay output
4.7	NC 7	Normally-closed contact of the eighth relay output

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DX522.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions (see Diagnosis and State LEDs *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*).

The following figure shows the connection of the digital input/output module DX522.

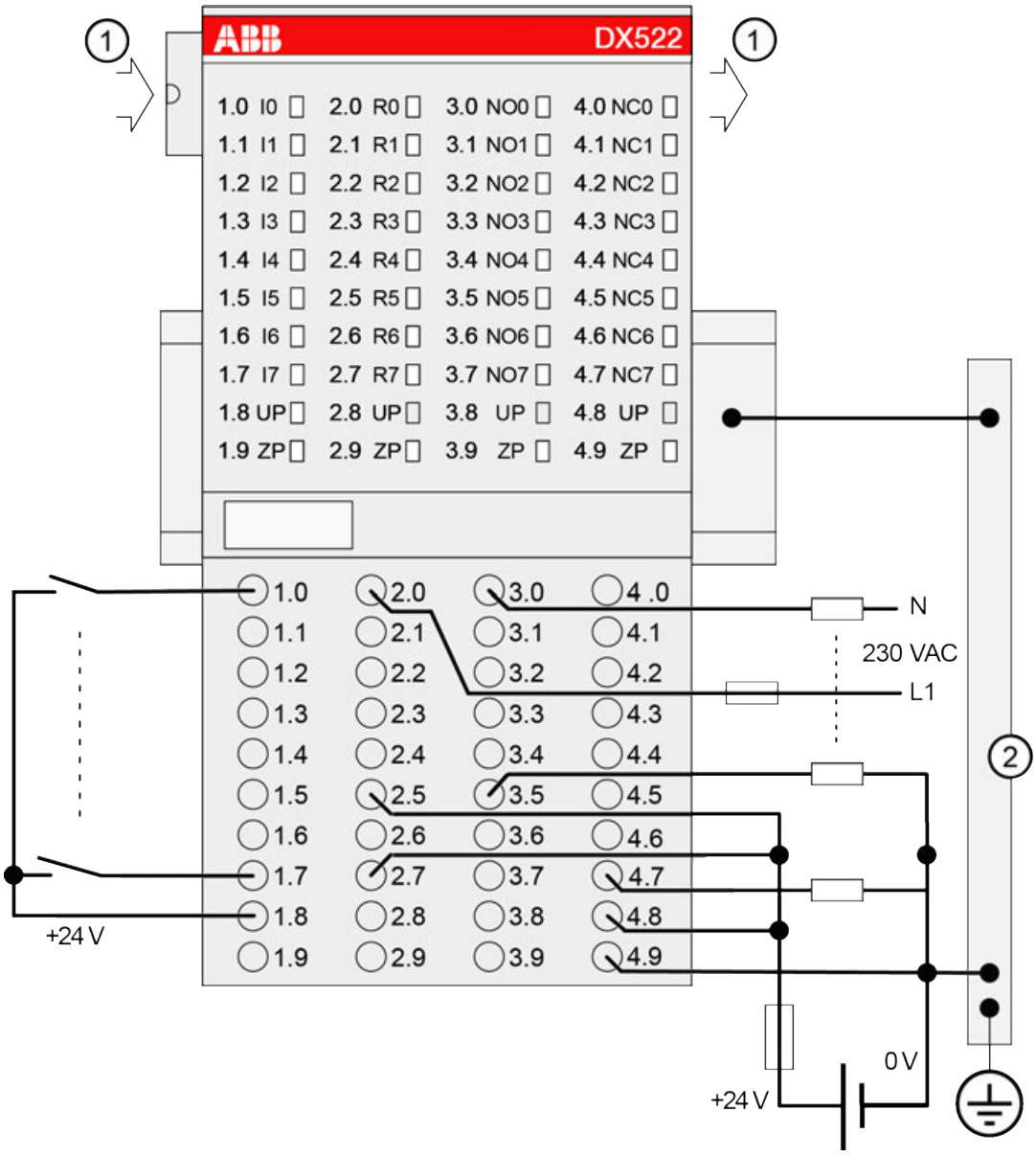


Fig. 843: Connection of the module

1 I/O bus
2 Switchgear cabinet earth

NOTICE!

– If the relay outputs have to switch inductive **DC loads**, free-wheeling diodes must be circuited in parallel to these loads.
– If the relay outputs have to switch inductive **AC loads**, spark suppressors are required.

CAUTION!

The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

NOTICE!

Risk of damaging the PLC module!

The following has to be considered when connecting input and output voltages to the module:

- All 230 V AC feeds must be single-phase from the same supply system.
- Connection of 2 or more relay contacts in series is possible; however, voltages above 230 V AC and 3-phase loads are not allowed.
- The 8 change-over contacts of the relays are galvanically isolated from channel to channel. This allows to connect loads of 24 V DC and 230 V AC to relay outputs of the same module. In such cases it is necessary that both supply voltages are grounded to prevent unsafe floating grounds.

NOTICE!

Risk of damaging the PLC module!

There is no internal short-circuit or overload protection for the relay outputs.

Protect the relay contacts by back-up fuses of 6 A max. (characteristic gG/gL). Depending on the application, fuses can be used for single channels or module-wise.

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	1	3
Digital outputs (bytes)	1	3
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1210 [1])	Word	1210 0x04BA	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length	Internal	5	Byte	5-CPU 4-FBP	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02	0	3	0x0Y04
Fast Counter [4])	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00			Not for FBP
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
Substitute value at outputs) Bit 7 = Output 7 Bit 0 = Output 0	0... 255	0... 0xff	Byte	0 0x00	0	255	0x0Y06

Remarks:

[1])	With CS31 and addresses smaller than 70 and FBP, the value is increased by 1
[2])	Not with FBP
[3])	For a description of the counter operating modes, please refer to the 'Fast Counter' section ↬ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
[4])	With FBP and without the parameter Fast Counter

GSD file:

Ext_User_Prm_Data_Len =	7
Ext_User_Prm_Data_Const (0) =	0x04, 0xbb, 0x04, \ 0x01, 0x02, 0x00, 0x00;

Diagnosis

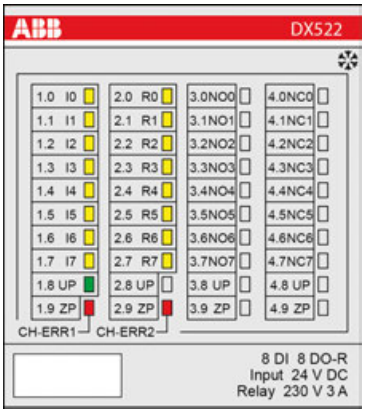
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process supply voltage too low		Check process supply voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process supply voltage is switched off (ON −> OFF)		Process supply voltage ON
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Digital input	Yellow	Input = OFF	Input = ON [1])	--
	Outputs R0...R7 (relays)	Digital output	Yellow	Relay output = OFF	Relay output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel Error, error messages in groups (digital inputs/ outputs combined into the groups 1 and 2)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR2		Red			
	CH-ERR [2])	Module Error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) All of the LEDs CH-ERR1 to CH-ERR2 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module	ca. 2 mA
	From UP at normal operation / with outputs	0.05 A + output loads
	Inrush current from UP (at power up)	0.010 A^2s
Max. power dissipation within the module		6 W (outputs OFF)
Weight (without terminal unit)		ca. 300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels

Parameter		Value
Terminals of the channels I0 to I7		1.0 to 1.7
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module (I/O bus)
Indication of the input signals		One yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
Signal 0		-3 V...+5 V
Undefined signal		> +5 V...< +15 V
Signal 1		+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the relay outputs

Parameter	Value
Number of channels per module	8 relay outputs
Distribution of channels into groups	8 groups of 1 channel each
Connection of the channel R0	Terminal 2.0 (common), 3.0 (NO) and 4.0 (NC)
Connection of the channel R1	Terminal 2.1 (common), 3.1 (NO) and 4.1 (NC)
Connection of the channel R6	Terminal 2.6 (common), 3.6 (NO) and 4.6 (NC)
Connection of the channel R7	Terminal 2.7 (common), 3.7 (NO) and 4.7 (NC)
Galvanic isolation	Between the channels and from the rest of the module
Indication of the output signals	One yellow LED per channel, the LED is ON when the relay coil is energized
Monitoring point of output indicator	LED is controlled by process CPU
Way of operation	Non-latching type
Output delay (0->1 or 1->0)	On request
Relay power supply	By UP process supply voltage
Relay outputs	

Parameter		Value
	Output short circuit protection	Should be provided externally with a fuse or circuit breaker
Rated protection fuse		6 A gL/gG per channel
Min. switching current		10 mA
Output switching capacity		
	Resistive load, max.	3 A; 3 A (230 V AC), 2 A (24 V DC)
	Inductive load, max.	1.5 A; 1.5 A (230 V AC), 1.5 A (24 V DC)
	Lamp load	60 W (230 V AC), 10 W (24 V DC)
Output switching capacity (XC version above 60 °C)		On request
Lifetime (cycles)		Mechanical: 300 000; Under load: 300 000 (24 V DC at 2 A), 200 000 (120 V AC at 2 A), 100 000 (230 V AC at 3 A)
Spark suppression with inductive AC load		Must be performed externally according to driven load specifications
Demagnetization with inductive DC load		A free-wheeling diode must be circuited in parallel to the inductive load
Switching frequency		
	With resistive load	Max. 10 Hz
	With inductive load	Max. 2 Hz
	With lamp load	On request
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to a

- *FBP interface module*
- *CS31 bus module*
- *CANopen communication interface module*

Parameter	Value
Used inputs	I0 / I1
Used outputs	None
Counting frequency	50 kHz max.
Detailed description	See *Chapter 1.6.4.1.10 “Fast counters” on page 5498*
Operating modes	See *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

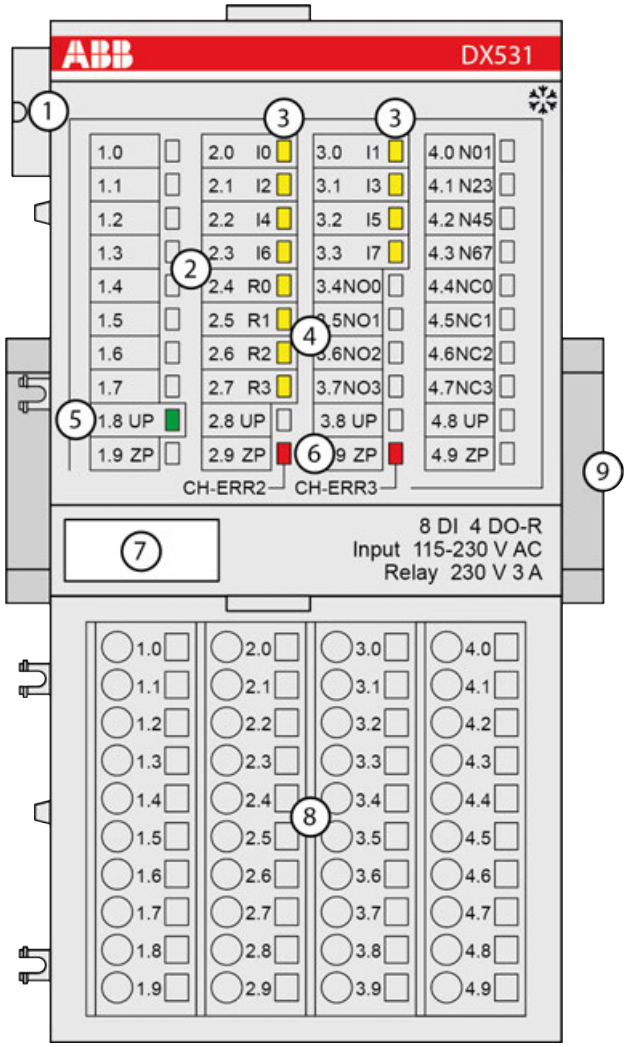
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 245 200 R0001	DX522, digital input/output module, 8 DI, 24 V DC, 8 DO relays	Active
1SAP 445 200 R0001	DX522-XC, digital input/output module, 8 DI, 24 V DC, 8 DO relays, XC version	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DX531 - Digital input/output module

- 8 digital inputs 120/230 V AC
- 4 relay outputs with one change-over contacts each
- Module-wise galvanically isolated



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the digital inputs (I0 - I7)

4 4 yellow LEDs to display the signal states at the digital relay outputs (R0 - R3)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Digital configurable input / output unit.

- 8 digital inputs 120/230 V AC in 1 group (2.0...2.3 and 3.0...3.3)
- 4 digital relay outputs with one change-over contact each (R0...R3). All output channels are galvanically isolated from each other.

The configuration is performed by software. The modules are supplied with a process supply voltage of 24 V DC.

All available inputs/outputs are galvanically isolated from all other circuitry of the module. There is no potential separation between the channels within the same group.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
LED displays	For signal states, errors and supply voltage
Internal power supply	Through the I/O bus interface (I/O bus)
External power supply	Via the terminals ZP and UP (process supply voltage 24 V DC)
Required terminal units	TU531 or TU532 ↳ *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*

The device is plugged on a terminal unit ↳ *Chapter 1.6.2.5.6 “TU531 and TU532 for I/O modules” on page 4114*. Position the module properly and press until it locks in place. The terminal unit is either mounted on a DIN rail or to the wall using 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

Connections

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal unit and always have the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process supply voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process supply voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	unused	
2.0 and 3.0	I0 and I1	Input signals for the digital inputs I0 and I1
4.0	N01	Neutral conductor for the digital inputs I0 and I1
2.1 and 3.1	I2 and I3	Input signals for the digital inputs I2 and I3
4.1	N23	Neutral conductor for the digital inputs I2 and I3
2.2 and 3.2	I4 and I5	Input signals for the digital inputs I4 and I5
4.2	N45	Neutral conductor for the digital inputs I4 and I5
2.3 and 3.3	I6 and I7	Input signals for the digital inputs I6 and I7
4.3	N67	Neutral conductor for the digital inputs I6 and I7
2.4	R0	Common contact of the first relay output
3.4 and 4.4	NO0 and NC0	NO and NC contacts of the first relay output
2.5	R1	Common contact of the second relay output
3.5 and 4.5	NO1 and NC1	NO and NC contacts of the second relay output

Terminals	Signal	Description
2.6	R2	Common contact of the third relay output
3.6 and 4.6	NO2 and NC2	NO and NC contacts of the third relay output
2.7	R3	Common contact of the fourth relay output
3.7 and 4.7	NO3 and NC3	NO and NC contacts of the fourth relay output

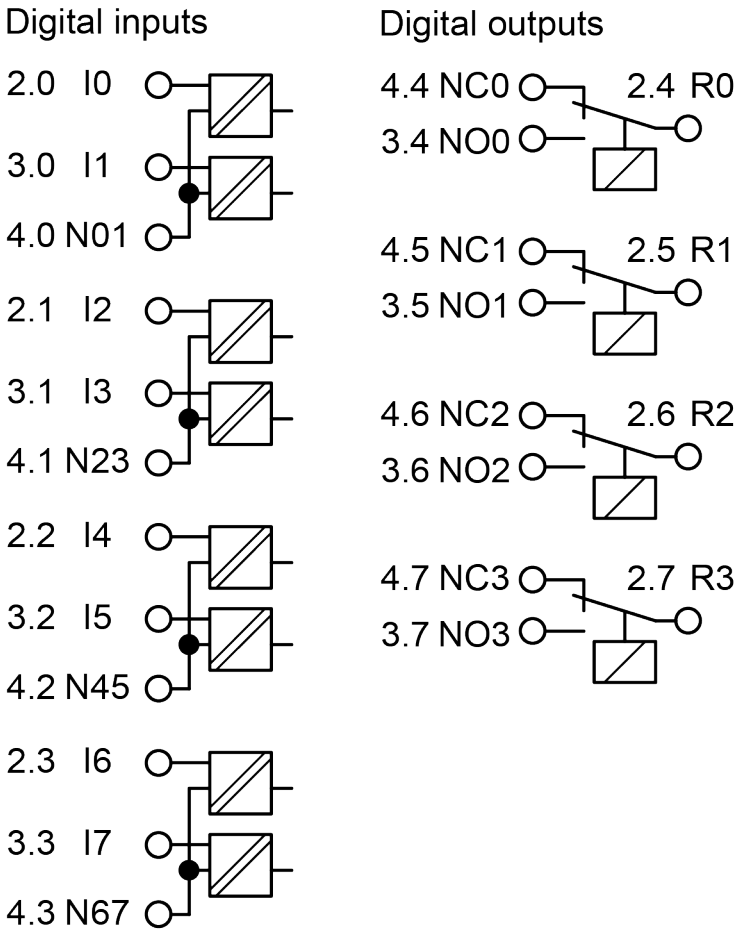


Fig. 844: Internal construction

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DX531. The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

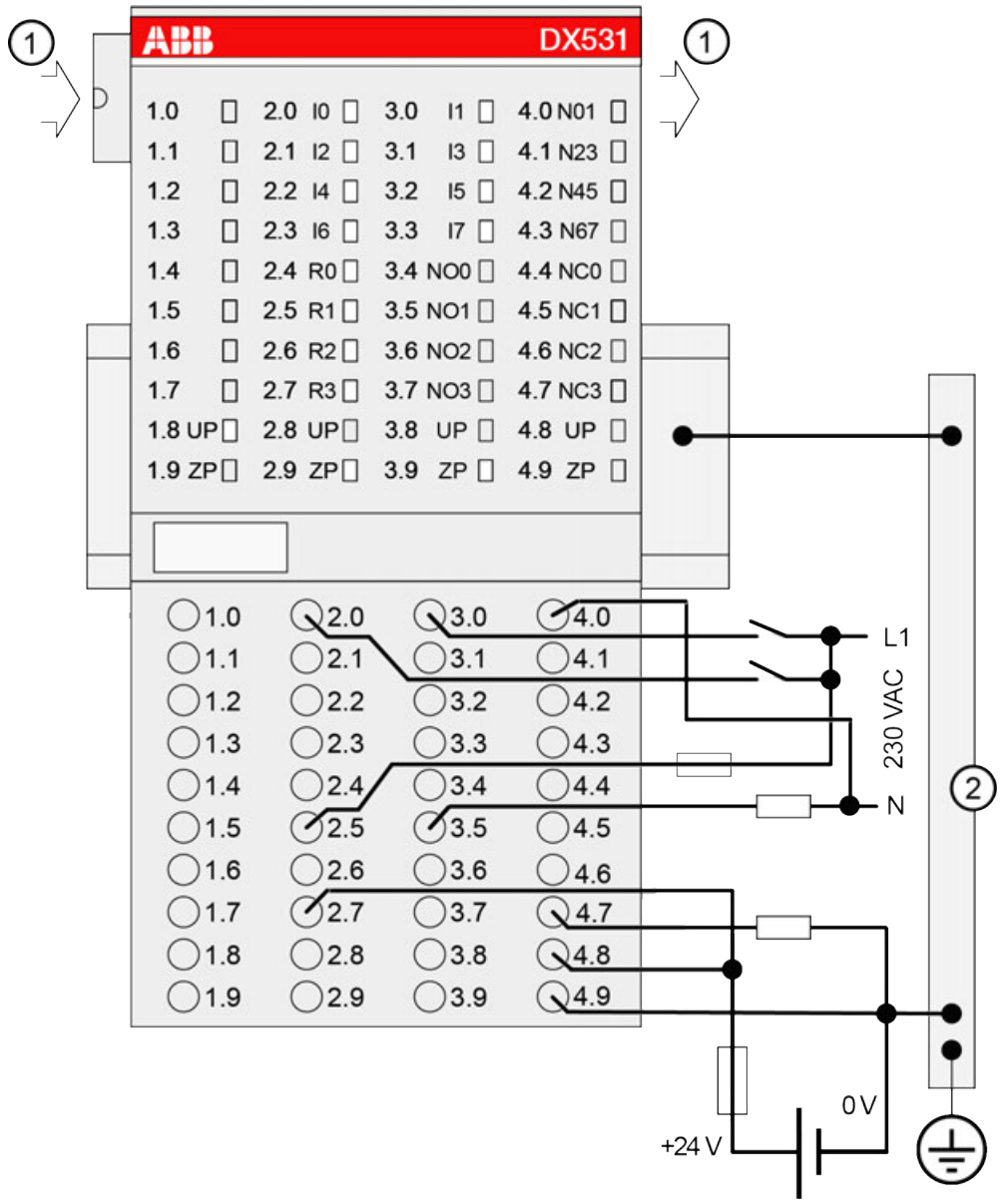
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the module:



1 I/O bus
2 Switchgear cabinet earth

NOTICE!

- If the relay outputs have to switch inductive **DC loads**, free-wheeling diodes must be circuited in parallel to these loads.
- If the relay outputs have to switch inductive **AC loads**, spark suppressors are required.

CAUTION!

The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

NOTICE!

Risk of damaging the PLC module!

The following has to be considered when connecting input and output voltages to the module:

- All 230 V AC feeds must be single phase from the same supply system.
- Connection of 2 or more relay contacts in series is possible; however, voltages above 230 V AC and 3-phase loads are not allowed.
- The 4 change-over contacts of the relays are galvanically isolated from channel to channel. This allows to connect loads of 24 V DC and 230 V AC to relay outputs of the same module. In such cases it is necessary that both supply voltages are grounded to prevent unsafe floating grounds.
- All input signals must come from the same phase of the same supply system (together with the used neutral conductor). The module is designed for 120/230 V AC max., not for 400 V AC, not even between two input terminals.
- All neutral conductor connections must be common to the same supply system, since the terminals 4.0 to 4.3 are interconnected within the module. Otherwise, accidental energization could occur.

NOTICE!

Risk of damaging the PLC module!

There is no internal short-circuit or overload protection for the relay outputs.

Protect the relay contacts by back-up fuses of 6 A max. (characteristic gG/gL). Depending on the application, fuses can be used for single channels or module-wise.

The module provides several diagnosis functions (see chapter Diagnosis and State LEDs ↳ *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*).

Internal data exchange

Digital inputs (bytes)	1
Digital outputs (bytes)	1
Counter input data (words)	0
Counter output data (words)	0

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1205 [1])	Word	1205 0x04B5	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			not for FBP
Parameter length	Internal	4	Byte	4-CPU 4-FBP	0	255	0x0Y02
Check supply	Off on	0 1	Byte	On 0x01	0	1	0x0Y03
Input delay	20 ms 100 ms	0 1	Byte	20 ms 0x00	0	1	0x0Y04
Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
Substitute value at outputs Bit 3 = Output 3 Bit 0 = Output 0	0...15	0... 0x0f	Byte	0 0x00	0	15	0x0Y06
[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1 [2]) Not with FBP							

GSD file:

Ext_User_Prm_Data_Len =	7
Ext_User_Prm_Data_Const (0) =	0x04, 0xb6, 0x04, \ 0x01, 0x00, 0x00, 0x00;

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process supply voltage too low		Check process supply voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process supply voltage is switched off (ON −> OFF)		Process supply voltage ON
	11 / 12	ADR	1...10					

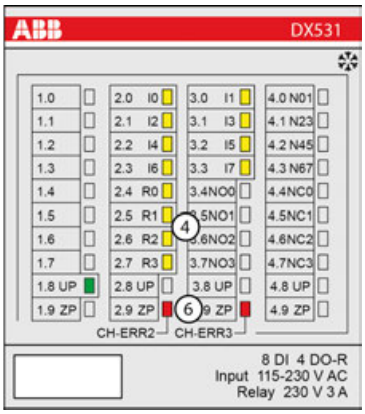
Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Digital input	Yellow	Input = OFF	Input = ON	--
	Outputs R0...R3 (relays)	Digital output	Yellow	Relay output = OFF	Relay output = ON	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR2	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 2 and 3)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Error on one channel of the corresponding group
	CH-ERR3		Red			
	CH-ERR *)	Module Error	Red	--	Internal error	--
	*) All of the LEDs CH-ERR2 to CH-ERR3 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage UP		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V DC (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V DC (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + output loads
Inrush current from UP (at power up)		0.004 A^2s
Max. power dissipation within the module		6 W (outputs OFF)
Weight (without terminal unit)		Ca. 300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	4 groups of 2 channels each
Terminals of the channels I0 to I7	*Chapter 1.6.2.6.1.2.9.3 “Connections” on page 4341*
Galvanic isolation	2500 V AC from the rest of the module (I/O bus)
Indication of the input signals	1 yellow LED per channel The LEDs are only operating if the module is initialized

Parameter		Value
Monitoring point of input indicator		LED is controlled by process CPU
Input type acc. to EN 61131-2		Type 2
Input delay (0->1 or 1->0)		Typ. 20 ms
Input signal voltage		230 V AC or 120 V AC
Input signal range		0 V AC...265 V AC
Input signal frequency		47 Hz...63 Hz
Input characteristic		According EN 61132-2 Type 2
Signal 0		0 V AC...40 V AC
Undefined signal		> 40 V AC...< 74 V AC
Signal 1		74 V AC...265 V AC
Input current per channel		
	Input voltage = 159 V AC	> 7 mA
	Input voltage = 40 V AC	< 5 mA
Overvoltage protection		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the relay outputs

Parameter		Value
Number of channels per module		4 relay outputs
Distribution of channels into groups		4 groups of 1 channel each
Connection of the four relays		*Chapter 1.6.2.6.1.2.9.3 “Connections” on page 4341*
Galvanic isolation		Between the channels and from the rest of the module
Indication of the output signals		1 yellow LED per channel, the LED is ON when the relay coil is energized
Monitoring point of output indicator		LED is controlled by process CPU
Way of operation		Non-latching type
Output delay (0->1 or 1->0)		On request
Relay power supply		By UP process supply voltage
Relay outputs		
	Output short circuit protection	Must be provided externally with a fuse or circuit breaker
	Rated protection fuse	6 A gL/gG per channel
Output switching capacity		
	Resistive load, max.	3 A; 3 A (230 V AC), 2 A (24 V DC)
	Inductive load, max.	1.5 A; 1.5 A (230 V AC), 1.5 A (24 V DC)
	Lamp load	60 W (230 V AC), 10 W (24 V DC)

Parameter		Value
Lifetime (cycles)		Mechanical: 300 000; Under load: 300 000 (24 V DC at 2 A), 200 000 (120 V AC at 2 A), 100 000 (230 V AC at 3 A)
Spark suppression with inductive AC load		Must be performed externally according to driven load specifications
Demagnetization with inductive DC load		A free-wheeling diode must be circuited in parallel to the inductive load
Switching frequency		
	With resistive load	Max. 10 Hz
	With inductive load	Max. 2 Hz
	With lamp load	On request
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 245 000 R0001	DX531, digital input/output module, 8 DI, 230 V AC, 4 DO relays, 2-wires	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Fast counter

More information can be found in the Automation Builder chapter, *"Fast counters in AC500 devices"*.

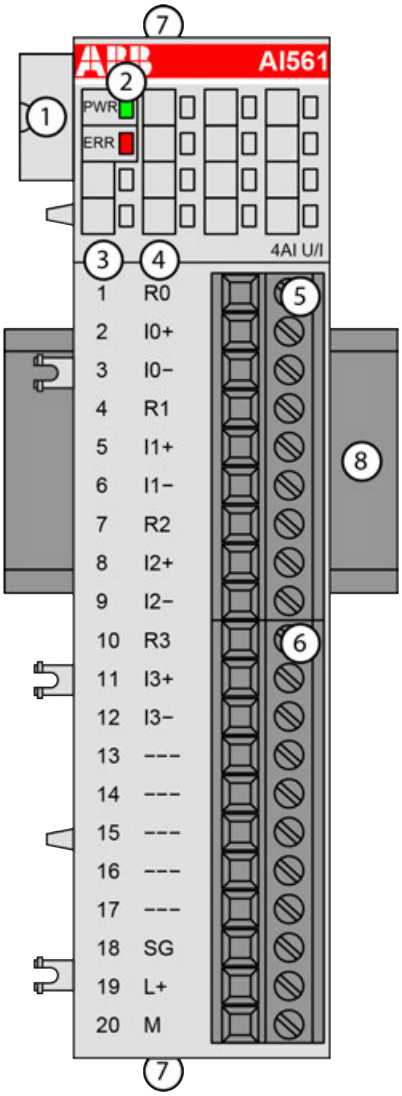
↳ *Chapter 1.6.4.1.10 "Fast counters" on page 5498*

1.6.2.6.2 Analog I/O modules

S500-eCo

AI561 - Analog input module

- 4 configurable analog inputs (I0 to I3) in 1 group
- Resolution: 11 bits plus sign or 12 bits



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

All other circuitry of the module is not galvanically isolated from the inputs or from the I/O bus.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog inputs, individually configurable for

- Not used (default setting)
- -2.5 V...+2.5 V
- -5 V...+5 V
- 0 V...+5 V
- 0 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-2.5 V...+2.5 V; -5 V...+5 V)	11 bits plus sign
	Voltage unipolar (0 V...5 V; 0 V...10 V)	12 bits
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages
Internal supply		Via I/O bus
External supply		Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

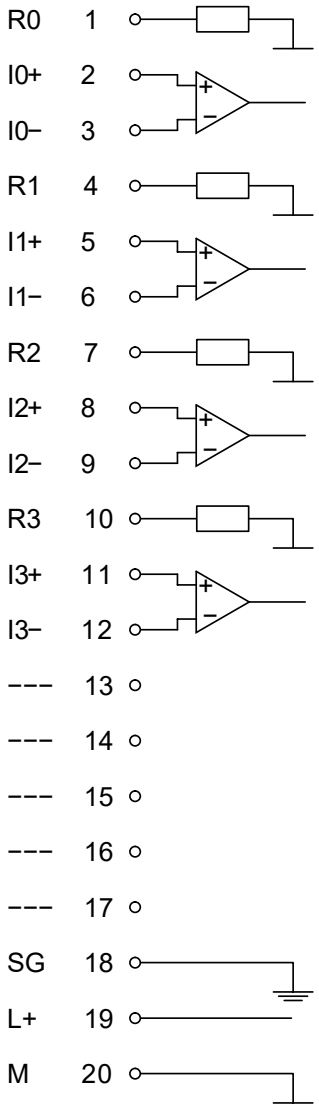
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	R0	Burden resistor for input signal 0 for current sensing
2	I0+	Positive pole of input signal 0
3	I0-	Negative pole of input signal 0
4	R1	Burden resistor for input signal 1 for current sensing
5	I1+	Positive pole of input signal 1
6	I1-	Negative pole of input signal 1
7	R2	Burden resistor for input signal 2 for current sensing
8	I2+	Positive pole of input signal 2
9	I2-	Negative pole of input signal 2
10	R3	Burden resistor for input signal 3 for current sensing
11	I3+	Positive pole of input signal 3

Terminal	Signal	Description
12	I3-	Negative pole of input signal 3
13	---	Reserved
14	---	Reserved
15	---	Reserved
16	---	Reserved
17	---	Reserved
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 10 mA per AI561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is interconnected to the M/ZP terminal of the CPU/communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

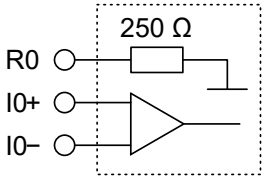
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

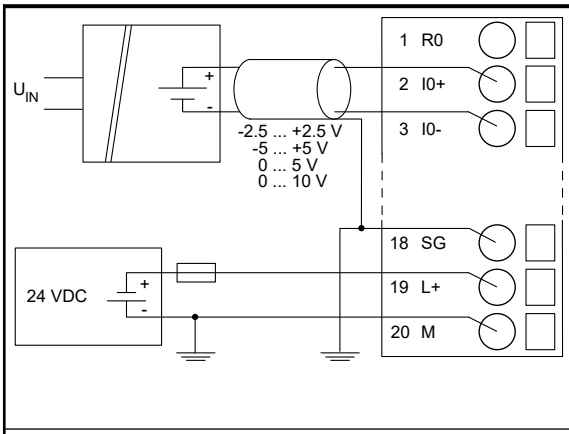
The module provides several diagnosis functions *Chapter 1.6.2.6.2.1.1.6 “Diagnosis” on page 4358*.

The following figure is an example of the internal construction of the analog input AI0. The analog inputs AI1...AI3 are designed in the same way.

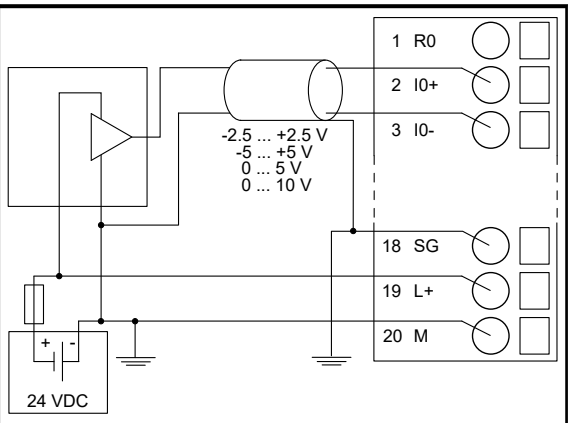


CAUTION!
Risk of damaging the analog input!
The 250 Ω input resistor can be damaged by overcurrent.
Make sure that the current through the resistor never exceeds 30 mA.

The following figures are an example of the connection of analog sensors (voltage) to the input I0 of the analog input module AI561. Proceed with the inputs I1 to I3 in the same way.

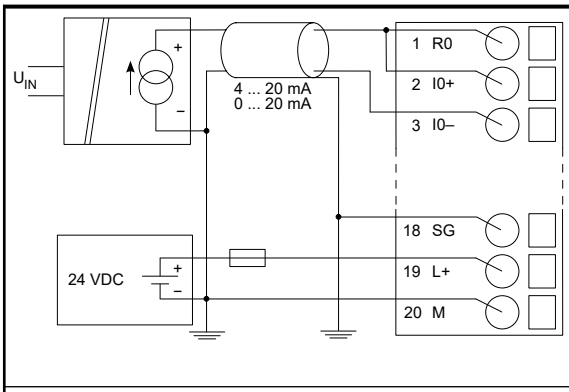


Connection of active-type analog sensors (voltage)

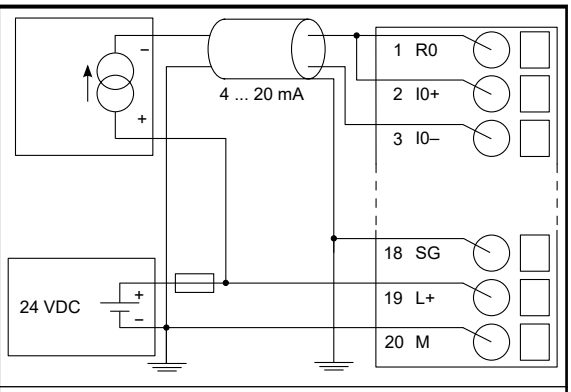


Connection of passive-type analog sensors (voltage)

The following figures are an example of the connection of analog sensors (current) to the input I0 of the analog input module AI561. Proceed with the inputs I1 to I3 in the same way.



Connection of active-type analog sensors (current)



Connection of passive-type analog sensors (current)

The meaning of the LEDs is described in the Displays section *Chapter 1.6.2.6.2.1.1.7 “State LEDs” on page 4359*.

I/O configuration

The analog input module AI561 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6500 [1])	WORD	0x1964	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Internal	6	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	

[1]) with CS31 and addresses smaller than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0 ... 7), LowByte is index (1...n)

GSD file:	Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	0x09 0x65, 0x19, 0x06, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00	0	65535

Table 425: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
1	0 V...10 V
3	0 mA...20 mA
4	4 mA...20 mA
6	0 V...5 V

Internal value	Operating modes for the analog inputs, individually configurable
7	-5 V...+5 V
20	-2,5 V...+2,5 V

Diagnosis

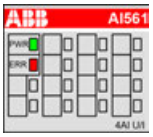
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	1	0...3	48	Analog value overflow at an analog input		Check input value or terminal
	11 / 12	ADR	1...0					
4	14	1...10	1	0...3	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...0					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Overflow	>2.9397	>5.8795	>5.8795	>11.7589	>23.5178	>22.8142	32767	7FFF
Measured value too high	2.9397 : 2.5014	5.8795 : 5.0029	5.8795 : : : 5.0015	11.7589 : : : 10.0029	23.5178 : : : 20.0058	22.8142 : : 20.0058	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range	2.5000 : 0.0014	5.0000 : 0.0029	5.0000 : : : 0.0015	10.0000 : : : 0.0029	20.0000 : : : 0.0058	20.0000 : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Normal range or measured value too low	0.0000	0.0000	0.0000	0.0000	0	4	0	0000
	:	:				3.9942	-10	FFF6
	-0.0014	-0.0029				:	-16	FFF0
	:	:				:	-4864	ED00
	:	:				0	-6912	E500
	:	:					:	:
	-2.5000	-5.0000					-27648	9400
Measured value too low	-2.5014	-5.0029					-27664	93F0
	:	:					:	:
	-2.9398	-5.8795					-32512	8100
Underflow	<-2.9398	<-5.8795	<-0.0300	<-0.0600	<-0.1200	<-0.1200	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption via L+ terminal	0.1 A
	Inrush current (at power up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 10 mA
Galvanic isolation		No
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.7 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 individually configurable voltage or current inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Unipolar	Voltage: 0 V...+5 V; 0 V...+10 V: 12 bits Current 0 mA...20 mA; 4 mA...20 mA: 12 bits	
	Bipolar	Voltage -2.5 V...+2.5 V; -5 V...+5 V: 11 bits plus sign	
Connection of the signals I0- to I3-		Terminals 3, 6, 9, 12	
Connection of the signals I0+ to I3+		Terminals 2, 5, 8, 11	
Input type		Differential	
Galvanic isolation		No galvanic isolation between the inputs and the I/O bus	
Common mode input range		Signal voltage plus common mode voltage must be within ±12 V	
Indication of the input signals		No	
Channel input resistance		Voltage: > 1 MΩ Current: ca. 250 Ω	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25 °C
		Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Time constant of the input filter		Voltage: 300 µs Current: 300 µs	
Relationship between input signal and hex code		↳ *Chapter 1.6.2.6.2.1.1.8 “Measuring ranges” on page 4359*	
Analog to digital conversion time		Typ. 500 µs per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC only for voltage input	

Parameter		Value
Max. cable length (conductor cross section > 0,14 mm²)		
	Unshielded wire	10 m
	Shielded wire	100 m

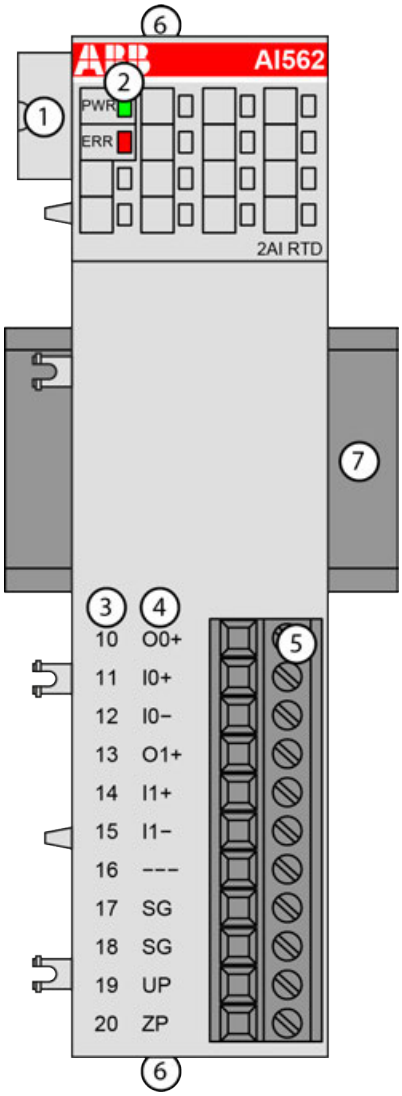
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1101	AI561, analog input module, 4 AI, U/I	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI562 - Analog input module

- 2 configurable analog resistance temperature detector (RTD) inputs (I0 and I1) in 1 group
- Resolution: 15 bits plus sign



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

All other circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

2 analog RTD-inputs, individually configurable for

- Not used (default)
- Pt100, -50 °C...+400 °C, 2-wire
- Pt100, -50 °C...+400 °C, 3-wire
- Pt1000, -50 °C...+400 °C, 2-wire
- Pt1000, -50 °C...+400 °C, 3-wire
- Ni1000, -50 °C...+150 °C, 2-wire
- Ni1000, -50 °C...+150 °C, 3-wire
- Ni100, -50 °C...+150 °C, 2-wire
- Ni100, -50 °C...+150 °C, 3-wire
- Analog input resistance 0 Ω...150 Ω
- Analog input resistance 0 Ω...300 Ω

Parameter	Value
Resolution of the analog channels	
Temperature	0.1 °C
LED displays	2 LEDs for process voltage and error messages
Internal supply	Via I/O bus
External supply	Via the terminals UP (process voltage 24 V DC) and ZP (0 V DC)

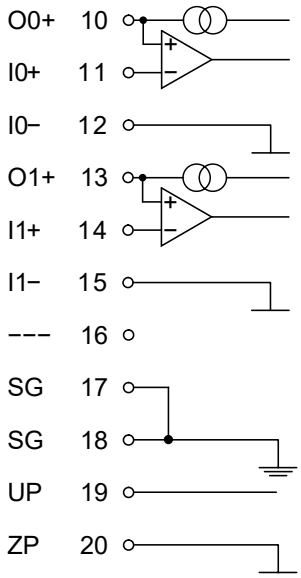
Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
10	O0+	Current source of channel 0
11	I0+	Sense input of channel 0
12	I0-	Return input of channel 0
13	O1+	Current source of channel 1
14	I1+	Sense input of channel 1
15	I1-	Return input of channel 1
16	---	Reserved
17	SG	Shield grounding
18	SG	Shield grounding
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per AI562.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove or replace a module.

NOTICE!

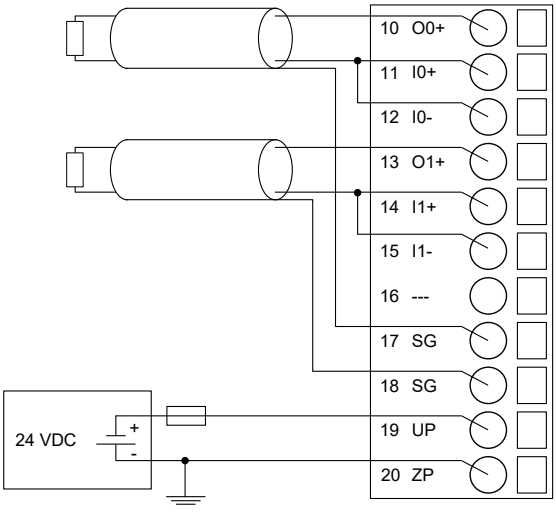
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

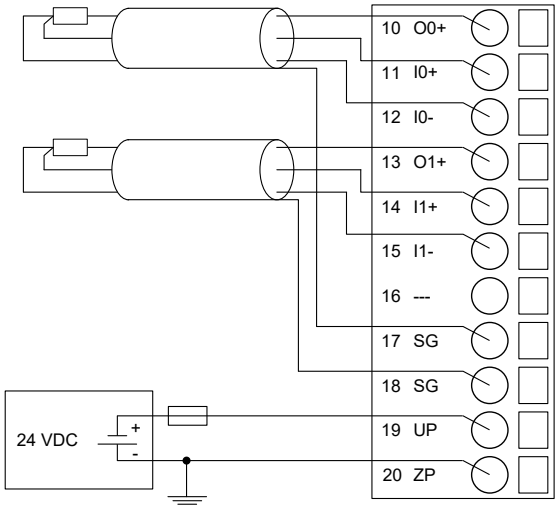
- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.2.6.2.1.2.6 "Diagnosis" on page 4368*.

The following figures show the connection of RTDs to the inputs of the analog input module AI562.



2-wires input



3-wires input

With 2-wires connection, the resistance of the connection wires influences the accuracy of the measured value. Use 3-wires connection to achieve the guaranteed measuring accuracy.

The meaning of the LEDs is described in the Displays section *Chapter 1.6.2.6.2.1.2.7 "State LEDs" on page 4369*.

I/O configuration

The analog input module AI562 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6505 [1])	WORD	0x1969	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	4	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	

[1]) with CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x07
Ext_User_Prm_Data_Const(0) =	0x6A, 0x19, 0x04, \ 0x01, 0x00, \ 0x00, 0x00;

Input channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [3])	0	65535

Table 426: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default) [3])
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C
16	2-wire Pt1000, -50 °C...+400 °C

Internal value	Operating modes for the analog inputs, individually configurable
17	3-wire Pt1000, -50 °C...+400 °C
18	2-wire Ni1000 -50 °C...+150 °C
19	3-wire Ni1000 -50 °C...+150 °C
22	2-wire Ni100, -50 °C...+150 °C
23	3-wire Ni100, -50 °C...+150 °C
32	Analog input resistor 0 Ω...150 Ω
33	Analog input resistor 0 Ω...300 Ω

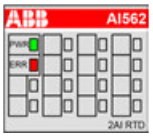
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	1	0...1	48	Analog value overflow at an analog input		Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...1	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Resistance temperature detectors

Range	Pt100 / Pt1000 -50 ... +400 °C	Ni1000 / Ni100 -50 ... +150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1

Range	Pt100 / Pt1000 -50 ... +400 °C	Ni1000 / Ni100 -50 ... +150 °C	Digital value	
			Decimal	Hex.
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
Normal range	400.0 °C : : : : 0.1 °C	 150.0 °C : : 0.1 °C	4000 2000 1500 700 : 1	0FA0 07D0 05DC 02BC : 1
	0,0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500 -2000	FFFF : FE0C F830
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Resistances

Range	Resistance 0 ... 150 Ω	Resistance 0 ... 300 Ω	Digital value	
			Decimal	Hex.
Overflow	>176.383	>352.767	32767	7FFF
Measured value too high	176.383 150.005	352.767 300.011	32511 27649	7EFF 6C01
Normal range	150.000 : 0.005	300.000 : 0.011	27648 : 1	6C00 : 0001
	0	0	0	0000

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption	0.04 A
	Inrush current (at power-up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for UP	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the input group and the rest of the module
	Isolated groups	1 (2 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		1.1 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value
Number of channels per module		2 configurable RTD (resistance temperature detector) inputs
Distribution of channels into groups		1 (2 channels per group)
Resolution		
	RTD	0.1 °C / 0.1 °F
	Resistance	15 bits + sign
Connection of the signals O0+ and O1+		Terminals 10 and 13
Connection of the signals I0- and I1-		Terminals 11 and 14
Connection of the signals I0+ and I1+		Terminals 12 and 15
Input type		Module ground referenced RTD for 2-wire and 3-wire resistance temperature detectors

Parameter		Value	
Galvanic isolation		Against internal power supply and other modules	
Input ranges		Pt100, Pt1000, Ni100, Ni1000	
		150 Ω, 300 Ω	
Indication of the input signals		No	
Module update time		All channels: < 1 s	
Channel input resistance		> 100 kΩ	
Input filter attenuation		-3 dB at 3.6 kHz	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	Depending on RTD max. ±0.6 % of full scale (guaranteed for 3-wires connection only) at 25 °C
		Max.	±2 % of full scale (guaranteed for 3-wires connection only) at 0 °C...60 °C or EMC disturbances
Measuring range		↬ *Chapter 1.6.2.6.2.1.2.8 “Measuring ranges” on page 4369*	
Analog to digital conversion time		Typ. 140 ms per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		4 bytes	
Power dissipation inside the sensor (max.)		1 mW	
Suppression of interference		On request	
Maximum input voltage		30 V DC (sense), 5 V DC (source)	
Basic error (resistance)		0.1 % of full-scale	
Repeatability		0.05 % of full-scale	
Overvoltage protection		Yes, up to 30 V DC	
Wire loop resistance		< 20 Ω	
Max. cable length (conductor cross section > 0.14 mm^2)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Ordering data

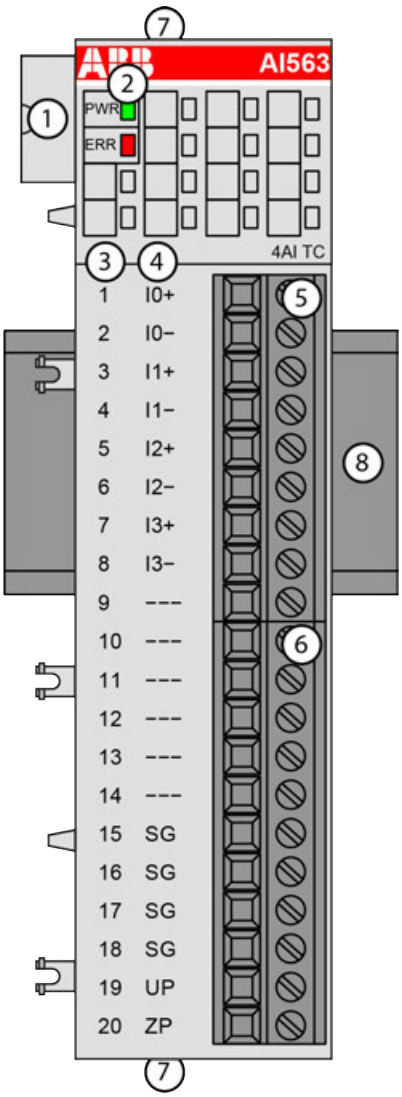
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1102	AI562, analog input module, 2 AI, RTD	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI563 - Analog input module

- 4 configurable thermocouple (TC) / -80 mV...+80 mV inputs (I0 to I3) in 1 group
- Resolution: 15 bits plus sign



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number

4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for input signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are group-wise galvanically isolated from each other.

The other electronic circuitry of the module is galvanically isolated from the inputs.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog TC inputs, individually configurable for

- Not used (default)
- Voltage -80 mV ... + 80 mV
- Thermocouple J-type -210 °C...+1200 °C
- Thermocouple K-type -270 °C...+1372 °C
- Thermocouple R-type -50 °C...+1768 °C
- Thermocouple S-type -50 °C...+1768 °C
- Thermocouple T-type -270 °C...+400 °C
- Thermocouple E-type -270 °C...+1000 °C
- Thermocouple N-type -270 °C...+1300 °C

Parameter	Value
Resolution of the analog channels	
Temperature	0.1 °C
LED displays	2 LEDs for process voltage and error messages
Internal supply	Via I/O bus
External supply	Via the terminals UP (process voltage 24 V DC) and ZP (0 V DC)

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

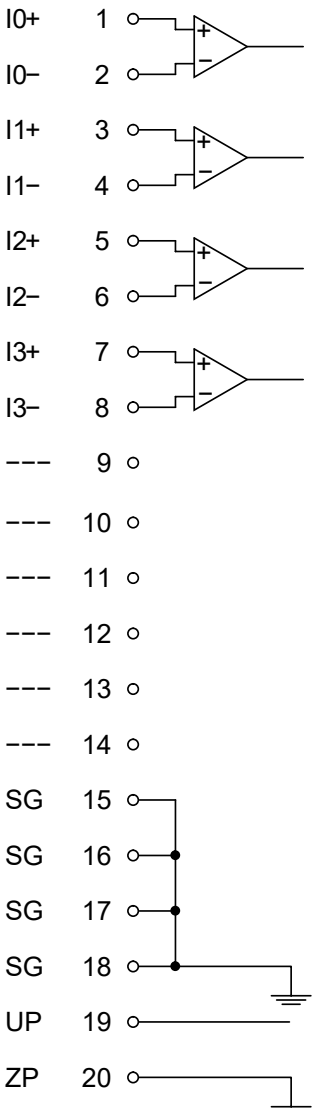
After powering up the system, input channels, which are configured will have undefined values /diagnosis message for typically 45 seconds, if the wires of all configured channels are broken.

If the AI563 is connected to a PROFINET communication interface module, the firmware version of PROFINET communication interface module must be 1.2 or above.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204

The following block diagram shows the internal construction of the analog inputs:



The assignment of the terminals:

Terminal	Signal	Description
1	I0+	Positive pole of channel 0
2	I0-	Negative pole of channel 0
3	I1+	Positive pole of channel 1
4	I1-	Negative pole of channel 1
5	I2+	Positive pole of channel 2
6	I2-	Negative pole of channel 2
7	I3+	Positive pole of channel 3
8	I3-	Negative pole of channel 3
9	---	Reserved
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	---	Reserved
14	---	Reserved
15	SG	Shield grounding
16	SG	Shield grounding
17	SG	Shield grounding
18	SG	Shield grounding
19	UP	Process voltage UP (24 V DC)
20	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AI563.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

NOTICE!

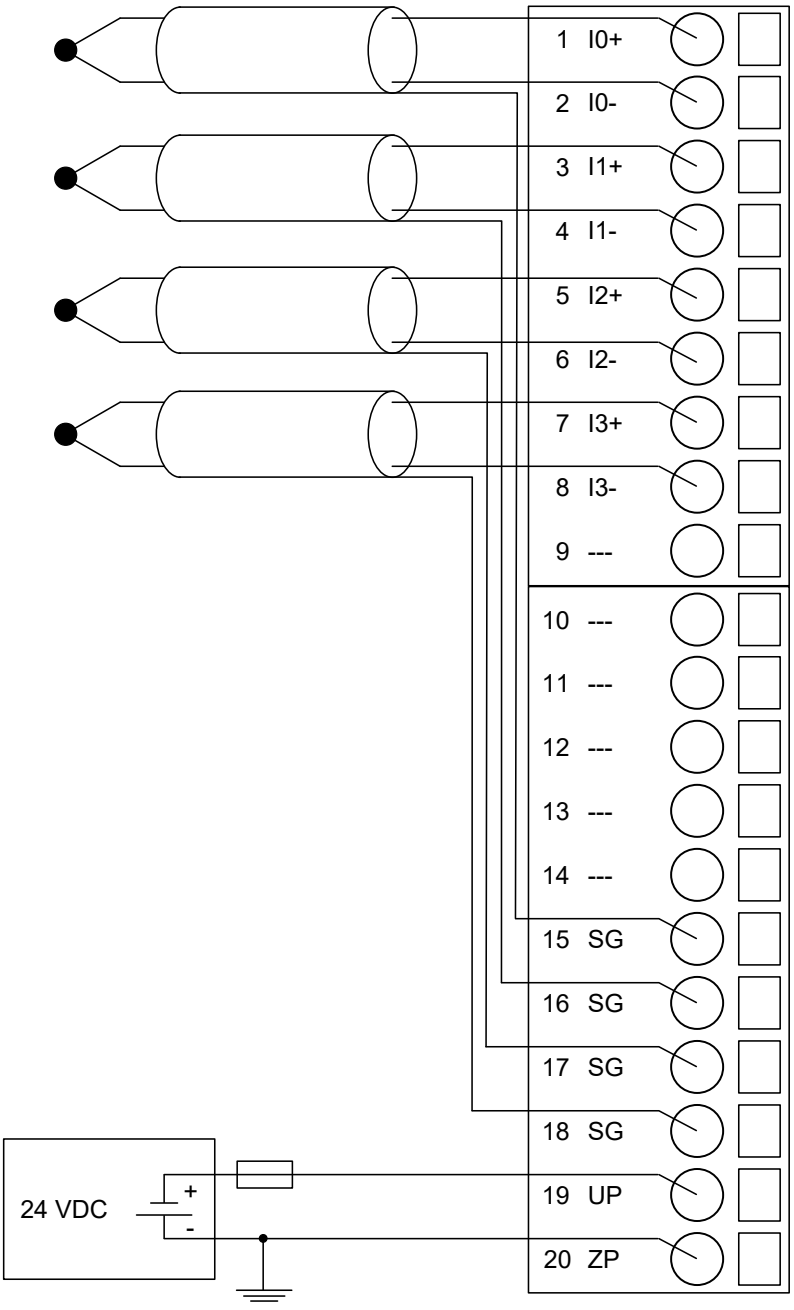
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.2.6.2.1.3.6 “Diagnosis” on page 4379*.

The following figure shows the connection of thermocouples to the inputs of the module:



The meaning of the LEDs is described in Displays *Chapter 1.6.2.6.2.1.3.7 “State LEDs” on page 4380* chapter.

I/O configuration

The analog input module AI563 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6510 [1])	WORD	0x196E	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	6	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	
[1]) with CS31 and addresses less than 70, the value is increased by 1 [2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x09
Ext_User_Prm_Data_Const(0) =	0x6F, 0x19, 0x06, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 427: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
21	Voltage -80 mV...+80 mV
24	Thermocouple J-type -210 °C...+1200 °C
25	Thermocouple K-type -270 °C...+1372 °C
26	Thermocouple R-type -50 °C...+1768 °C
27	Thermocouple S-type -50 °C...+1768 °C
28	Thermocouple T-type -270 °C...+400 °C
29	Thermocouple E-type -270 °C...+1000 °C
30	Thermocouple N-type -270 °C...+1300 °C

Diagnosis

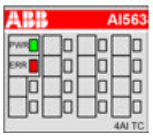
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...3	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...3	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31-Bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Measuring ranges

AI563 needs typ. 6 to 8 seconds for initialization after applying the process supply voltage to clamp UP/ZP. During this time, the accuracy of the measurement values is not within specification. After that, valid measurement values are provided by the module. After that, valid measurement values are provided by the module.

After an interruption of the process supply voltage > 10 ms, a re-initialization is performed by AI563.

Risk of invalid analog input values!

The analog input values may be invalid if the measuring range of the inputs is exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

When a wire break occurs on a sensor wire, the temperature measurement value of the corresponding channel changes to Overflow (Hexadecimal 7FFF).

Range	**Type J** **-210 ... +1200 °C**	**Type K** **-270 ... +1372 °C**	**Type N** **-270 ... +1300 °C**	**Type T** **-270 ... +400 °C**	**Digital value**	
					Decimal	**Hex.**
Overflow	> 1200.0 °C	> 1372.0 °C	> 1300.0 °C	> 400.0 °C	32767	7FFF
Normal range					17680	4510
		1372.0 °C			13720	3598
		:	1300.0 °C		13000	32C8
	1200.0 °C	:	:		12000	2EE0
	:	:	:	400.0 °C	4000	0FA0
	:	:	:	:	:	:
	0.1 °C	0.1 °C	0.1 °C	0.1 °C	1	1
	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:	:
	:	:	:	:	-500	FE0C
	-210.0 °C	:	:	:	-2100	F7CC
		-270.0 °C	-270.0 °C	-270.0 °C	-2700	F574
Underflow	< -210.0 °C	< -270.0 °C	< -270.0 °C	< -270.0 °C	-32768	8000

Range	**-80 mV ... +80 mV**	**Type E** **-270 ... +1000 °C**	**Types R, S** **-50 ... +1768 °C**	**Digital value**	
				Decimal	**Hex.**
Overflow	> +90 mV	> 1000.0 °C	> 1768.0 °C	32767	7FFF
Normal range	+80 mV			27648	6C00
			1768.0 °C	17680	4510
		1000.0 °C		10000	2710
				9000	2328
	:	:	:	:	:
	3 µV	0.1 °C	0.1 °C	1	1
	0 µV	0.0 °C	0.0 °C	0	0000
	-3 µV	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:
	:	:	-50.0 °C	-500	FE0C
	:	-270.0 °C		-2700	F574

Range	-80 mV ... +80 mV	Type E -270 ... +1000 °C	Types R, S -50 ... +1768 °C	Digital value	
				Decimal	Hex.
	-80 mV			-27648	9400
Underflow	< -90 mV	< -270.0 °C	< -50.0 °C	-32768	8000

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage UP		
	Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
	Rated value	24 V DC
	Current consumption	0.10 A
	Inrush current (at power-up)	0.07 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse for UP	Not necessary
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		Yes, between the channels and the rest of the module
	Isolated groups	1 (4 channels per group)
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		2.6 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 configurable thermocouple (TC) inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Temperature	0.1 °C	
	Voltage	15 bits plus sign	
Connection of the signals I0+ to I3+		Terminals 1, 3, 5 and 7	
Connection of the signals I0- to I3-		Terminals 2, 4, 6 and 8	
Input type		Floating thermocouple	
Galvanic isolation		Against internal power supply and other modules	
Common mode rejection		> 120 dB at 120 V AC	
Indication of the input signals		No	
Module update time		All channels: < 1.6 s	
Channel input resistance		On request	
Input filter attenuation		-3 dB at 15 kHz	
Cold junction error		±1.5 °C	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	0.1 % of full-scale (voltage) Depending on thermocouple, see table *Chapter 1.6.2.6.2.1.3.9.1.1 “Accuracy of thermocouple ranges at 25 °C (with cold junction compensation)” on page 4384* at 25 °C
		Max.	±2 % of full scale (T-Type: ±3 % for -240 °C...-270 °C) at 0 °C...60 °C
Relationship between input signal and hex code		*Chapter 1.6.2.6.2.1.3.8 “Measuring ranges” on page 4380*	
Analog to digital conversion time		400 ms per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC	
Repeatability		On request	
Wire loop resistance		< 100 Ω	
Max. cable length (conductor cross section > 0.14 mm²)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Accuracy of thermocouple ranges at 25 °C (with cold junction compensation)

Thermocouple Type	Range	Accuracy
E	-270 °C...-220 °C	±2 %
	-220 °C...+1000 °C	±0.6 %
J	-210 °C...+1200 °C	±0.6 %
K	-270 °C...-220 °C	±1.5 %
	-220 °C...+1372 °C	±0.6 %
N	-270 °C...-150 °C	±2 %
	-150 °C...+1300 °C	±0.6 %
R	-50 °C...+150 °C	±1.5 %
	+150 °C...+1768 °C	±0.6 %
S	-50 °C...+150 °C	±1.5 %
	+150 °C...+1768 °C	±0.6 %
T	-270 °C...-240 °C	±3 %
	-240 °C...-0 °C	±2 %
	0 °C...+400 °C	±0.6 %

These accuracy values are valid only for stable module temperatures.

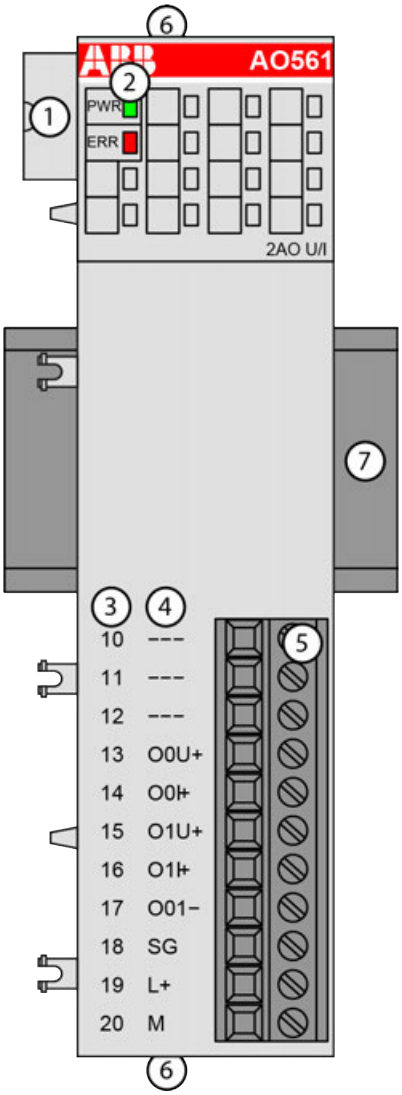
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1103	AI563, analog input module, 4 AI, thermocouple	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AO561 - Analog output module

- 2 configurable analog outputs (O0 and O1) in 1 group
- Resolution: 11 bits plus sign or 12 bit



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error
3 Terminal number
4 Allocation of signal name
5 Terminal block for output signals (11-pin)
6 2 holes for wall-mounting with screws
7 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The outputs are not galvanically isolated from each other.

The other electronic circuitry of the module is not galvanically isolated from the outputs or from the I/O bus.

The I/O module must not be used as communication interface module at CI590-CS31-HA bus modules.

Functionality

2 analog outputs, individually configurable for

- Not used (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-10 V...+10 V)	11 bits plus sign
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages
Internal supply		Via I/O bus
External supply		Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

Connections

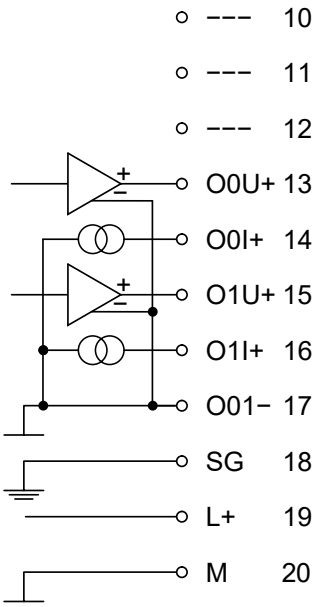
For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↬ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

If the output is configured as not used, the voltage and current output signals are undefined and must not be connected.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↬ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the analog outputs:



The assignment of the terminals:

Terminal	Signal	Description
10	---	Reserved
11	---	Reserved
12	---	Reserved
13	O0U+	Voltage output of channel 0
14	O0I+	Current output of channel 0
15	O1U+	Voltage output of channel 1
16	O1I+	Current output of channel 1
17	O01-	Negative pole of channels O0 and O1
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AO561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is electrically interconnected to the M/ZP terminal of the CPU/ communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove or replace a module.

NOTICE!

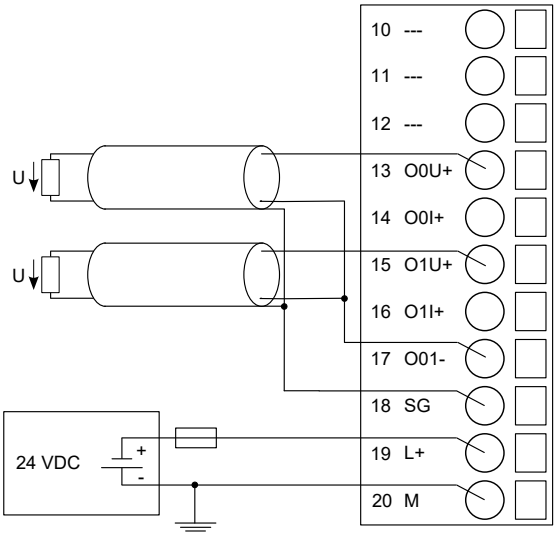
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

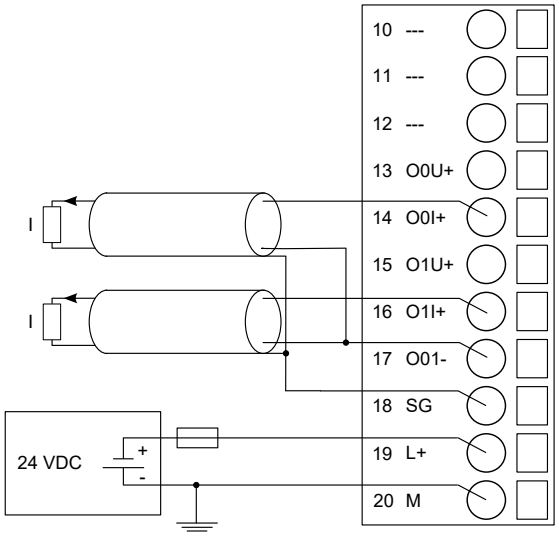
- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.2.6.2.1.4.6 “Diagnosis” on page 4390*.

The following figures show the connection of analog actuators to the analog output module AO561.



Connection of analog voltage actuators



Connection of analog current actuators

The output signal is undefined if the supply voltage at the L+ terminal is below 10 V. This can, for example, occur if the supply voltage has a slow ramp-up / ramp-down behavior and must be foreseen when planning the installation.

If the output is configured in current mode, the voltage output signal is undefined and must not be connected.

If the output is configured in voltage mode, the current output signal is undefined and must not be connected.

I/O configuration

The analog output module AO561 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	6515 [1])	WORD	0x1973	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Intern	4	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00		255	
[1]) with CS31 and addresses less than 70, the value is increased by 1 [2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)							

GSD file:

Ext_User_Prm_Data_Len =	0x07
Ext_User_Prm_Data_Const(0) =	0x74, 0x19, 0x04, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Output channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 428: Channel configuration [2])

Internal value	Operating modes for the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

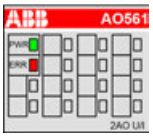
Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	3	0...1	48	Analog value overflow at an analog output	Check output value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	3	0...1	7	Analog value underflow at an analog output	Check output value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more channels of the module

Output ranges

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	>11.7589	>23.5178	>22.8142	32767	7FFF
Value too high	11.7589	23.5178	22.8142	32511	7EFF
	:	:	:	:	:
	10.0058	:	:	27664	6C10
	:	:	20.0058	27658	6C0A
	:	20.0058	:	27656	6C08
Normal range	10.0000	20.0000	20.0000	27648	6C00
Normal range or value too low	:	:	:	:	:
	0.0058	:	:	16	0010
	:	:	4.0058	10	000A
	:	0.0058		8	0008
	0.0000	0	4	0	0000

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
	: -0.0058 : : : -10.0000		3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Value too low	-10.0058 : -11.7589			-27664 : -32512	93F0 : 8100
Underflow	<-11.7589		<0.0000	-32768	8000

The represented resolution corresponds to 12 bit respectively 11 bit plus sign.

Technical data

The System Data of AC500-eCo apply ↬ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption	0.1 A + output load
	Inrush current (at power-up)	0.05 A^2s
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module		Ca. 5 mA
Galvanic isolation		No
Surge-voltage (max.)		35 V DC for 0.5 s
Max. power dissipation within the module		3.1 W
Weight		Ca. 120 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog outputs

Parameter		Value	
Number of channels per module		2 configurable voltage or current outputs	
Distribution of channels into groups		1 (2 channels per group)	
Connection of the signals O0U- and O1U+		Terminals 13 and 15	
Connection of the signals O0I+ and O1I+		Terminals 14 and 16	
Output type		Bipolar with voltage, unipolar with current	
Resolution		12 bits or 11 bits plus sign	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.5 % of full scale at 25 °C
		Max.	±2 % of full scale at 0 °C...+60 °C or EMC disturbances
Indication of the output signals		No	
Output Resistance (load) as current output		0 Ω...500 Ω	
Output load ability as voltage output		±2 mA max.	
Output data length		4 bytes	
Relationship between output signal and hex code		*Chapter 1.6.2.6.2.1.4.8 “Output ranges” on page 4391*	
Unused outputs		Must not be connected and must be configured as "unused"	
Overvoltage protection		Yes, up to 30 V DC	
Max. cable length (conductor cross section > 0.14 mm^2)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Ordering data

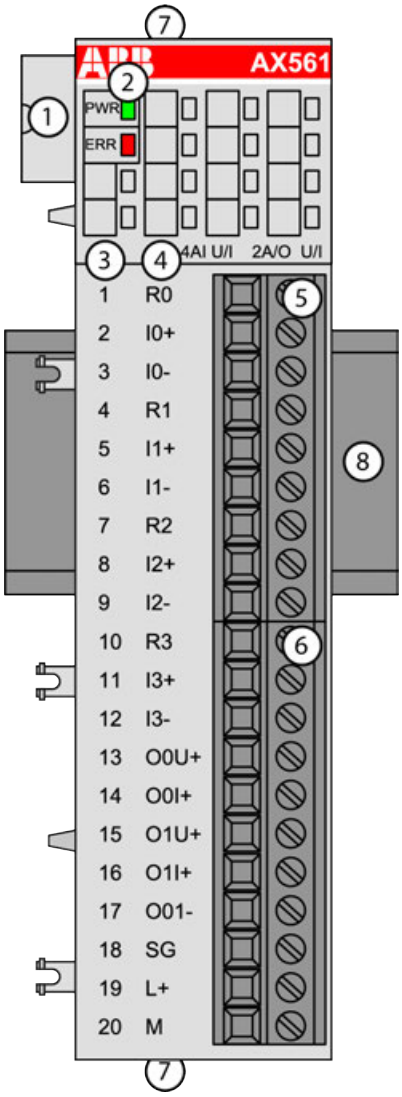
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1201	AO561, analog output module, 2 AO, U/I	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX561 - Analog input/output module

- 4 configurable analog inputs (I0 to I3) in 1 group
- 2 configurable analog outputs (O0 and O1) in 1 group
- Resolution: 11 bits plus sign or 12 bits



1 I/O bus
2 1 green LED to display power supply, 1 red LED to display error

3 Terminal number
4 Allocation of signal name
5 Terminal block for input signals (9-pin)
6 Terminal block for output signals (11-pin)
7 2 holes for wall-mounting with screws
8 DIN rail

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The inputs are not galvanically isolated from each other.

The outputs are not galvanically isolated from each other.

All other circuitry of the module is not galvanically isolated from the inputs/outputs or from the I/O bus.

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

Functionality

4 analog inputs, individually configurable for

- Not used (default)
- -2.5 V...+2.5 V
- -5 V...+ 5 V
- 0 V...+5 V
- 0 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

2 analog outputs, individually configurable for

- Not used (default)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage bipolar (-2.5 V...+2.5 V; -5 V...+5 V)	11 bits plus sign
	Voltage unipolar (0 V...5 V; 0 V...10 V)	12 bits
	Current (0 mA...20 mA; 4 mA...20 mA)	12 bits
LED displays		2 LEDs for process voltage and error messages

Parameter	Value
Internal supply	Via I/O bus
External supply	Via the terminals L+ (process voltage 24 V DC) and M (0 V DC); the M terminal is connected to the M terminal of the CPU via the I/O bus

Connections

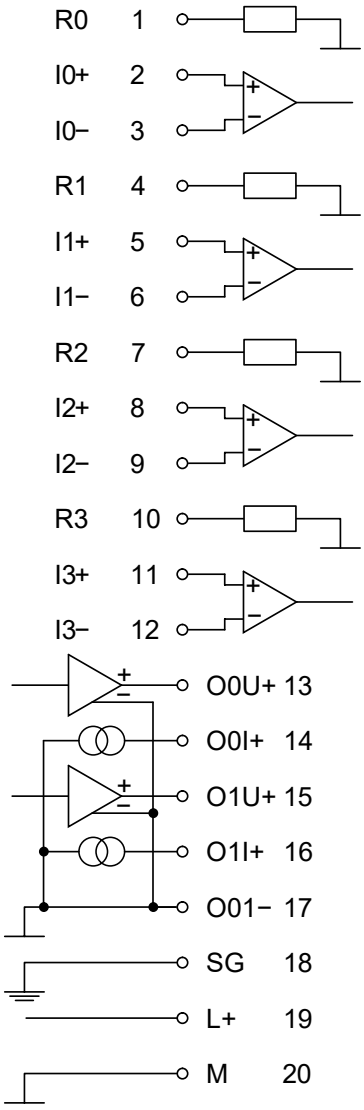
For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter ↳ Chapter 1.6.3.5 “AC500-eCo” on page 5233.

If the output is configured as not used, the voltage and current output signals are undefined and must not be connected.

The connection is carried out by using a removable 9-pin and 11-pin terminal block. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*

The following block diagram shows the internal construction of the analog inputs and outputs:



The assignment of the terminals:

Terminal	Signal	Description
1	R0	Burden resistor for input signal 0 for current sensing
2	I0+	Positive pole of input signal 0
3	I0-	Negative pole of input signal 0
4	R1	Burden resistor for input signal 1 for current sensing
5	I1+	Positive pole of input signal 1
6	I1-	Negative pole of input signal 1
7	R2	Burden resistor for input signal 2 for current sensing
8	I2+	Positive pole of input signal 2
9	I2-	Negative pole of input signal 2
10	R3	Burden resistor for input signal 3 for current sensing
11	I3+	Positive pole of input signal 3
12	I3-	Negative pole of input signal 3
13	O0U+	Voltage output of channel 0
14	O0I+	Current output of channel 0

Terminal	Signal	Description
15	O1U+	Voltage output of channel 1
16	O1I+	Current output of channel 1
17	O01-	Negative pole of channels O0 and O1
18	SG	Shield grounding
19	L+	Process voltage L+ (24 V DC)
20	M	Process voltage M (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module increases by 5 mA per AX561.

The external power supply connection is carried out via the L+ (+24 V DC) and the M (0 V DC) terminals. The M terminal is interconnected to the M/ZP terminal of the CPU/communication interface module.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

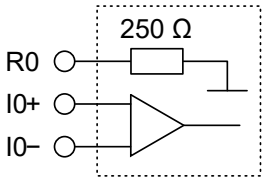
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The module provides several diagnosis functions *Chapter 1.6.2.6.2.1.5.6 “Diagnosis” on page 4402*.

The following figure is an example of the internal construction of the analog input AI0. The analog inputs AI1...AI3 are designed in the same way.



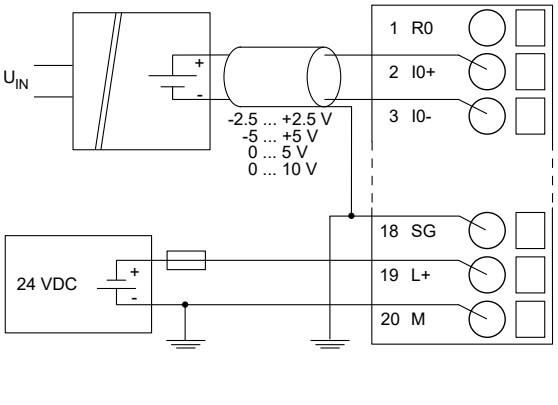
CAUTION!

Risk of damaging the analog input!

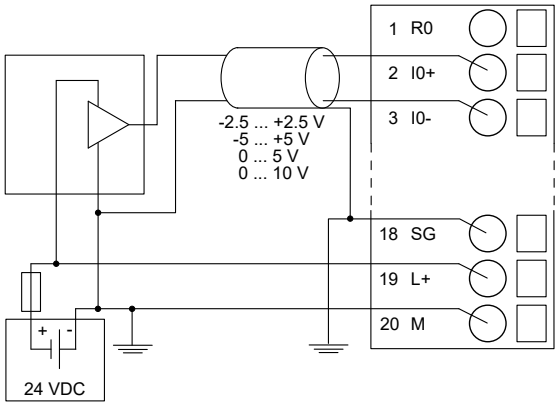
The 250 Ω input resistor can be damaged by overcurrent.

Make sure that the current through the resistor never exceeds 30 mA.

The following figures are an example of the connection of analog sensors (voltage) to the input I0 of the analog input/output module AX561. Proceed with the inputs I1 to I3 in the same way.

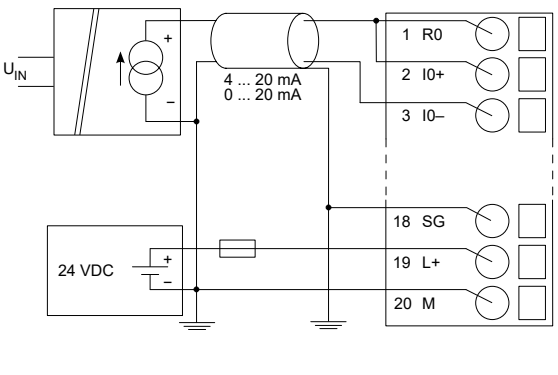


Connection of active-type analog sensors (voltage)

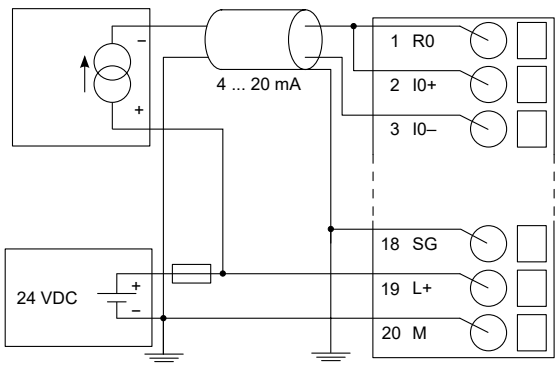


Connection of passive-type analog sensors (voltage)

The following figures are an example of the connection of analog sensors (current) to the input I0 of the analog input/output module AX561. Proceed with the inputs I1 to I3 in the same way.

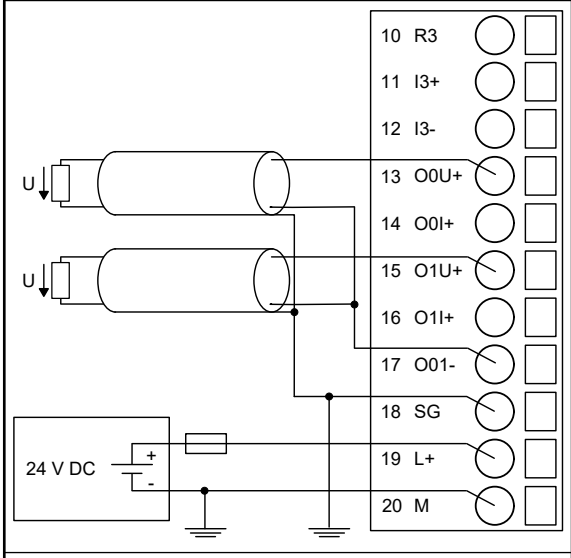


Connection of active-type analog sensors (current)

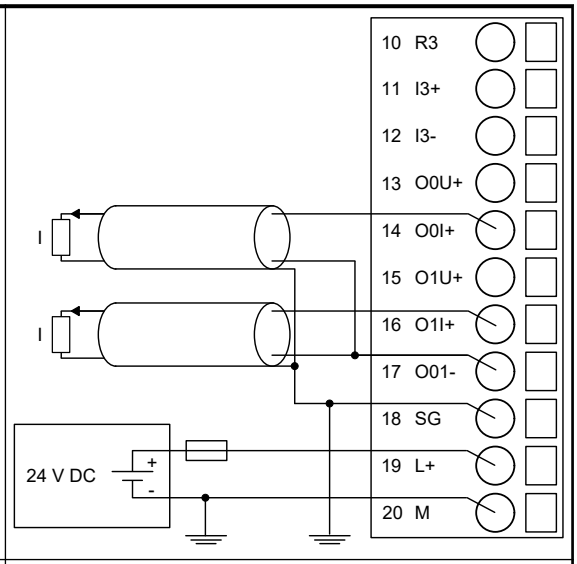


Connection of passive-type analog sensors (current)

The following figures are an example of the connection of analog actuators to the analog input/ output module AX561.



Connection of analog voltage actuators



Connection of analog current actuators

The output signal is undefined if the supply voltage at the L+ terminal is below 10 V. This can, for example, occur if the supply voltage has a slow ramp-up / ramp-down behavior and must be foreseen when planning the installation.

If the output is configured in current mode, the voltage output signal is undefined and must not be connected.

If the output is configured in voltage mode, the current output signal is undefined and must not be connected.

The meaning of the LEDs is described in the displays chapter ↳ *Chapter 1.6.2.6.2.1.5.7 “State LEDs” on page 4403*.

I/O configuration

The I/O module does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Internal	6520 [1])	WORD	0x1978	0	65535	xx01
Ignore module	No Yes	0 1	BYTE	No 0x00			
Parameter length	Internal	8	BYTE	0	0	255	xx02 [2])
Check Supply	Off On	0 1	BYTE	On 0x01			
Analog Data Format	Default	0	BYTE	Default 0x00			

[1]) With CS31 and addresses less than 70, the value is increased by 1

[2]) Value is hexadecimal: HighByte is slot (xx: 0...7), LowByte is index (1...n)

GSD file:

Ext_User_Prm_Data_Len =	0x0B
Ext_User_Prm_Data_Const(0) =	0x79, 0x19, 0x08, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00;

Input channel (4x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configuration	see table [2])	see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 429: Channel configuration [2])

Internal value	Operating modes for the analog inputs, individually configurable
0	Not used (default)
1	0 V...+10 V
3	0 mA...20 mA
4	4 mA...20 mA
6	0 V...+5 V
7	-5 V...+5 V
20	-2.5 V...+2.5 V

Output channel (2x)

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.
Channel configura-tion	see see table [2])	see see table [2])	BYTE	0 0x00 see table [2])	0	65535

Table 430: Channel configuration [2])

Internal value	Operating modes for the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+ 10 V
129	0 mA...20 mA
130	4 mA...20 mA

Diagnosis

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<− Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC Browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	PNIO diagnosis block		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		Restart
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	1	0...3	48	Analog value overflow at an analog input		Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...10					

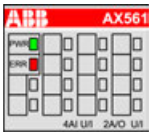
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	14	1...10	3	0...1	48	Analog value overflow at an analog output	Check output value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	3	0...1	7	Analog value underflow at an analog output	Check output value
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e. g. of the DC551-CS31)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or PNIO = module type (1 = AI, 3 = AO); COM1/ COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		**State**	**Color**	**LED = OFF**	**LED = ON**	**LED flashes**
	PWR	Process voltage 24 V DC via terminal	Green	CPU module voltage or external 24 V DC supply voltage is missing	3.3 V system voltage (I/O bus) and external 24 V DC supply voltage are present	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Error on 1 or more chan-nels of the module

Measuring ranges

CAUTION!

Risk of wrong analog input values!

The analog input values may be wrong if the measuring range of the inputs are exceeded.

Make sure that the analog signal at the connection terminals is always within the signal range.

Range	-2.5 ... +2.5 V	-5 ... +5 V	0 ... 5 V	0 ... 10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
							Decimal	Hex.
Overflow	>2.9397	>5.8795	>5.8795	>11.7589	>23.5178	>22.8142	32767	7FFF
Measured value too high	2.9397 : 2.5014	5.8795 : 5.0029	5.8795 : : : 5.0015	11.7589 : : : 10.0029	23.5178 : : : 20.0058	22.8142 : : 20.0058	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range Normal range or measured value too low	2.5000 : 0.0014	5.0000 : 0.0029	5.0000 : : : 0.0015	10.0000 : : : 0.0029	20.0000 : : : 0.0058	20.0000 : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008
	0.0000	0.0000	0.0000	0.0000	0	4	0	0000
	: -0.0014 : : : -2.5000	: -0.0029 : : : -5.0000				3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Measured value too low	-2.5014 : -2.9398	-5.0029 : -5.8795					-27664 : -32512	93F0 : 8100
Underflow	<-2.9398	<-5.8795	<-0.0300	<-0.0600	<-0.1200	<-0.1200	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Output ranges

Range	-10 ... +10 V	0 ... 20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589	> 23.5178	> 22.8142	32767	7FFF
Output value too high	11.7589 : 10.0058 : :	23.5178 : : : 20.0058	22.8142 : : 20.0058 :	32511 : 27664 27658 27656	7EFF : 6C10 6C0A 6C08
Normal range Normal range or output value too low	10.0000 : 0.0058 : :	20,0000 : : : 0.0058	20.0000 : : 4.0058	27648 : 16 10 8	6C00 : 0010 000A 0008
	0.0000	0	4	0	0000
	: -0.0058 : : : -10.0000		3.9942 : : 0	-10 -16 -4864 -6912 : -27648	FFF6 FFF0 ED00 E500 : 9400
Output value too low	-10.0058 : -11.7589			-27664 : -32512	93F0 : 8100
Underflow	< -11.7589		<0.0000	-32768	8000

The represented resolution corresponds to 12 bits respectively 11 bits plus sign.

Technical data

The System Data of AC500-eCo apply *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter		Value
Process supply voltage L+		
	Connections	Terminal 19 for L+ (+24 V DC) and terminal 20 for M (0 V)
	Rated value	24 V DC
	Current consumption via L+ terminal	0.14 A + output load
	Inrush current (at power-up)	0.05 A
	Max. ripple	5 %
	Protection against reversed voltage	Yes

Parameter	Value
Protection fuse for L+	Recommended
Current consumption from 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 5 mA
Galvanic isolation	No
Surge-voltage (max.)	35 V DC for 0.5 s
Max. power dissipation within the module	4.9 W
Weight	Ca. 120 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter		Value	
Number of channels per module		4 individually configurable voltage or current inputs	
Distribution of channels into groups		1 (4 channels per group)	
Resolution			
	Unipolar	Voltage: 0 V...+5 V; 0 V...+10 V: 12 bits Current 0 mA...20 mA; 4 mA...20 mA: 12 bits	
	Bipolar	Voltage -2.5 V...+2.5 V; -5 V...+5 V: 11 bits plus sign	
Connection of the signals I0- to I3-		Terminals 3, 6, 9, 12	
Connection of the signals I0+ to I3+		Terminals 2, 5, 8, 11	
Input type		Differential	
Galvanic isolation		No galvanic isolation between the inputs and the I/O bus	
Common mode input range		Signal voltage plus common mode voltage must be within ±12 V	
Indication of the input signals		No	
Channel input resistance		Voltage: >1 MΩ Current: ca. 250 Ω	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25 °C

Parameter		Value	
		Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Time constant of the input filter		Voltage: 300 µs Current: 300 µs	
Relationship between input signal and hex code		↳ *Table on page 4404*	
Analog to digital conversion time		Typ. 500 µs per channel	
Unused inputs		Can be left open and should be configured as "unused"	
Input data length		8 bytes	
Overvoltage protection		Yes, up to 30 V DC only for voltage input	
Max. cable length (conductor cross section > 0.14 mm^2)			
	Unshielded wire	10 m	
	Shielded wire	100 m	

Technical data of the analog outputs

Parameter	Value	
Number of channels per module	2 configurable voltage or current outputs	
Distribution of channels into groups	1 (2 channels per group)	
Connection of the signals O0U- and O1U+	Terminals 13 and 15	
Connection of the signals O0I+ and O1I+	Terminals 14 and 16	
Output type	Bipolar with voltage, unipolar with current	
Resolution	12 bits or 11 bits plus sign	
Indication of the output signals	No	
Output resistance (load) as current output	0 Ω...500 Ω	
Output load ability as voltage output	2 mA max.	
Relationship between input signal and hex code	Table Output Ranges ↳ *Table on page 4405*	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale (voltage) ±0.5 % of full scale (current 0 mA...20 mA) ±0.7 % of full scale (current 4 mA...20 mA) at 25°C
	Max.	±2 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Unused outputs	Can be left open and should be configured as "unused"	
Output data length	4 bytes	
Overvoltage protection	Yes, up to 30 V DC	

Parameter		Value
Max. cable length (conductor cross section > 0.14 mm²)		
	Unshielded wire	10 m
	Shielded wire	100 m

Ordering data

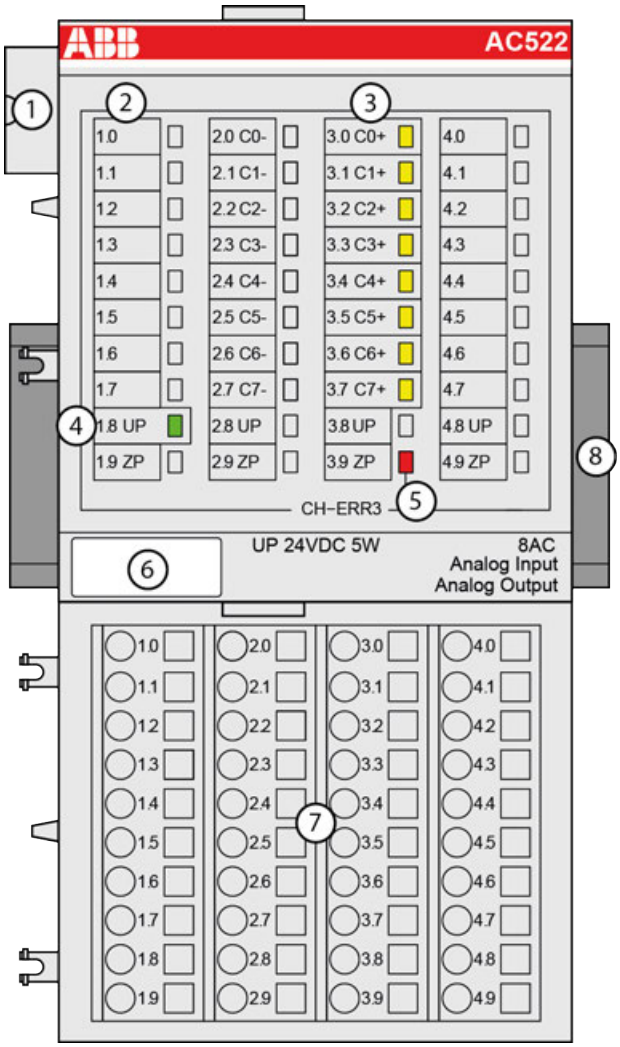
Part no.	Description	Product life cycle phase *)
1TNE 968 902 R1301	AX561, analog input/output module, 4 AI, 2 AO, U/I	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

S500

AC522 - Analog input/output module

- 8 configurable analog inputs/outputs in one group (2.0...2.7 and 3.0...3.7)
- Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the analog inputs/outputs (C0 - C7)
4 1 green LED to display the state of the process supply voltage UP
5 1 red LED to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

The configuration is performed by software. The modules are supplied with a process voltage of 24 V DC.

The inputs and outputs are galvanically isolated from all other circuitry of the module.

Functionality

8 analog inputs (I0...I7), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

4 analog outputs (O4...O7), individually configurable for

- Unused (default setting)
- -10 V...+10 V

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		10 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The modules are plugged on an I/O terminal unit ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	Unused	Unused
2.0 to 2.7	C0- to C7-	Negative poles of the 8 analog inputs/outputs
3.0 to 3.7	C0+ to C7+	Positive poles of the analog inputs/outputs
4.0 to 4.7	Unused	Unused

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module. The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

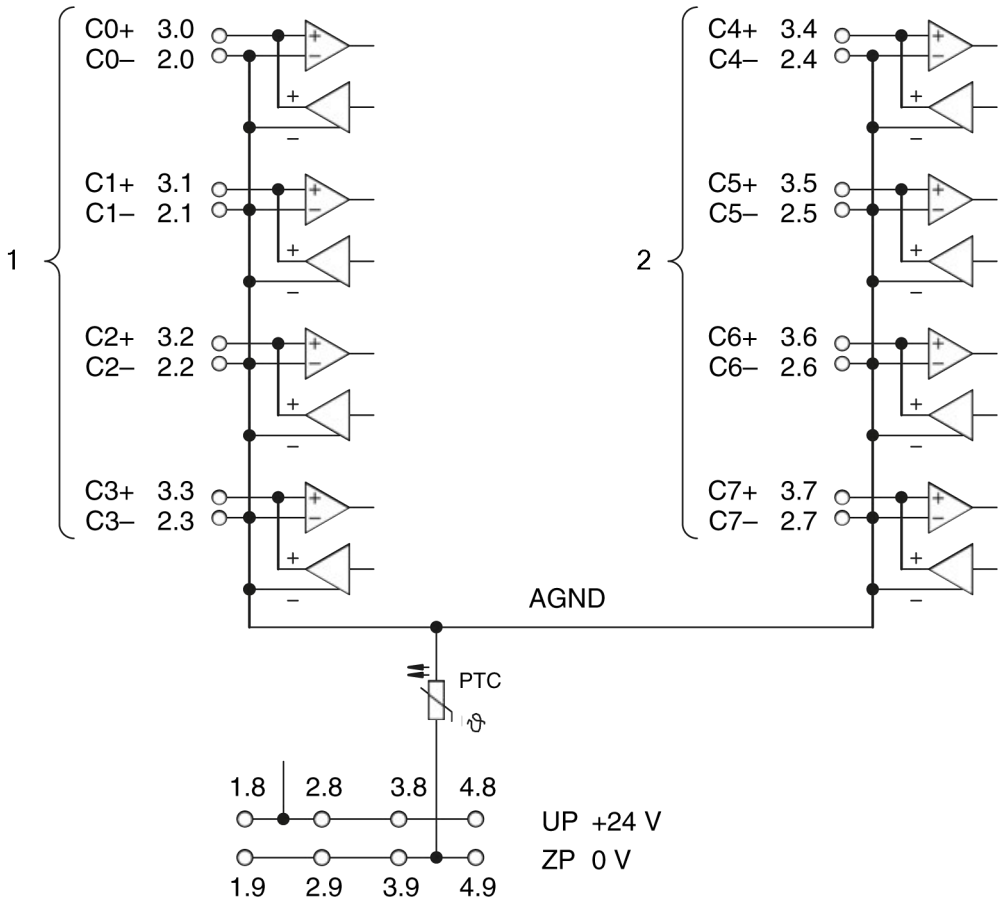
Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.



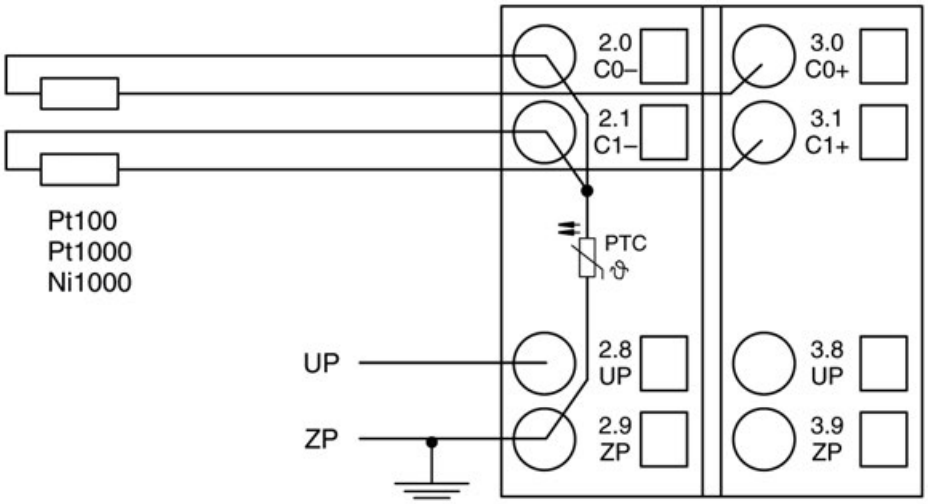
1 4 analog I/O channels
as inputs for 0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/Pt1000/Ni1000
digital signals
as outputs for -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA
2 4 analog I/O channels
as inputs for 0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/Pt1000/Ni1000
digital signals
as outputs for -10 V...+10 V

The process voltage must be included in the grounding concept of the control system (e.g. grounding the negative pole).

By installing equipotential bonding conductors between the different parts of the system, it must be made ensured that the potential difference between ZP and AGND never exeeds 1 V.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.



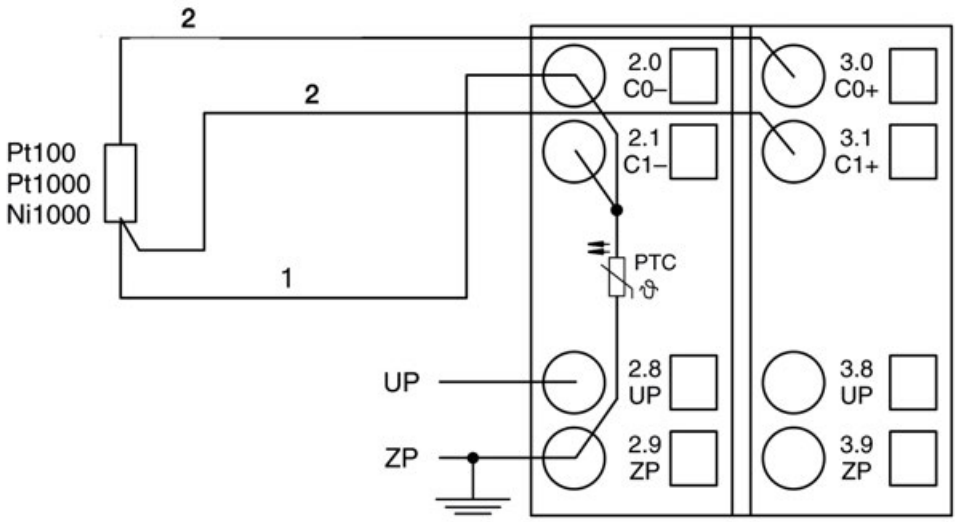
Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.



1 Return line
2 Twisted pair within the cable

> *If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.*

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. C1).

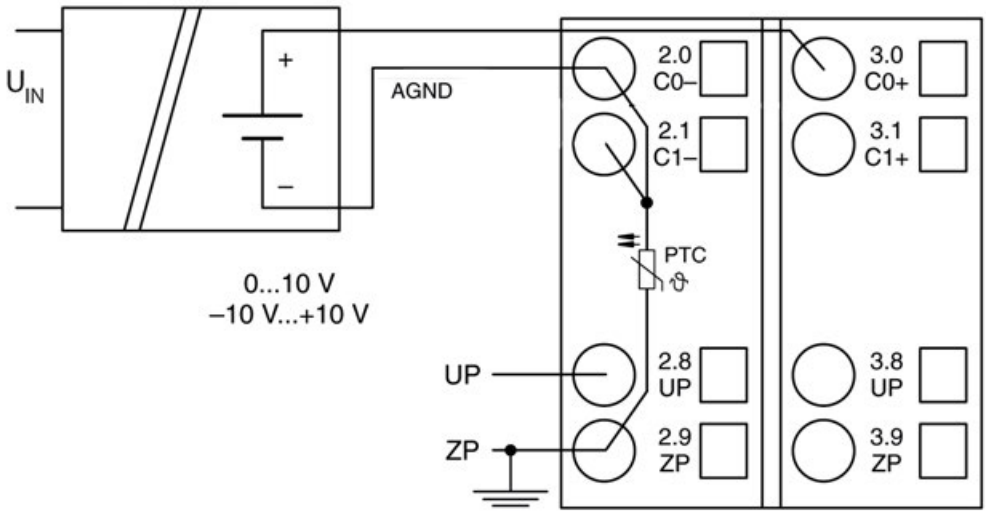
In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply



By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

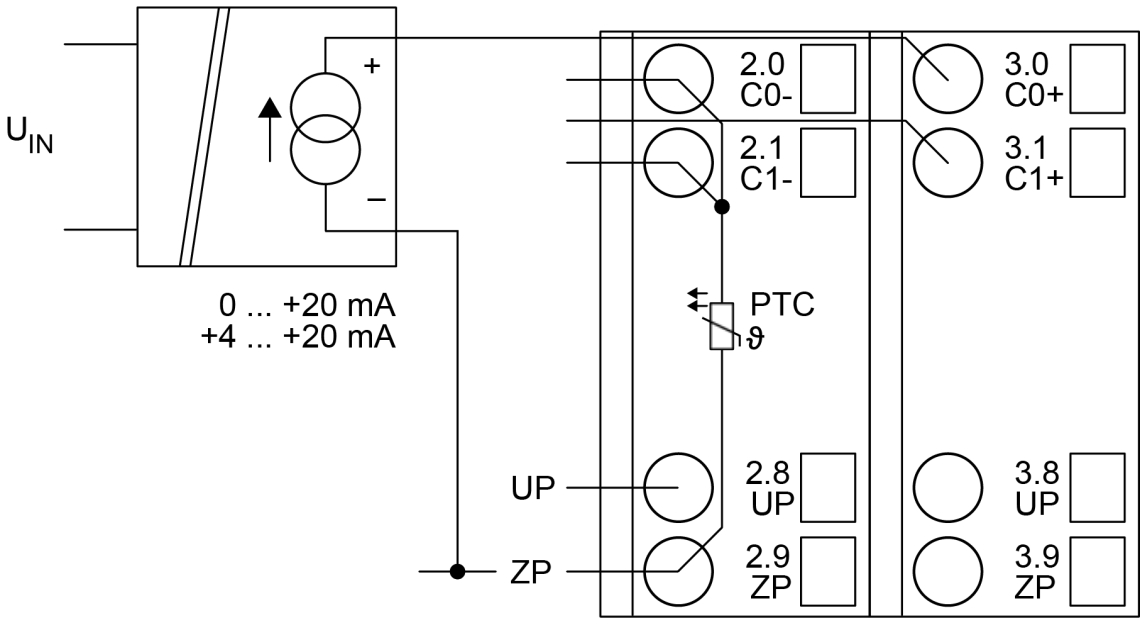
By connecting to AGND the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

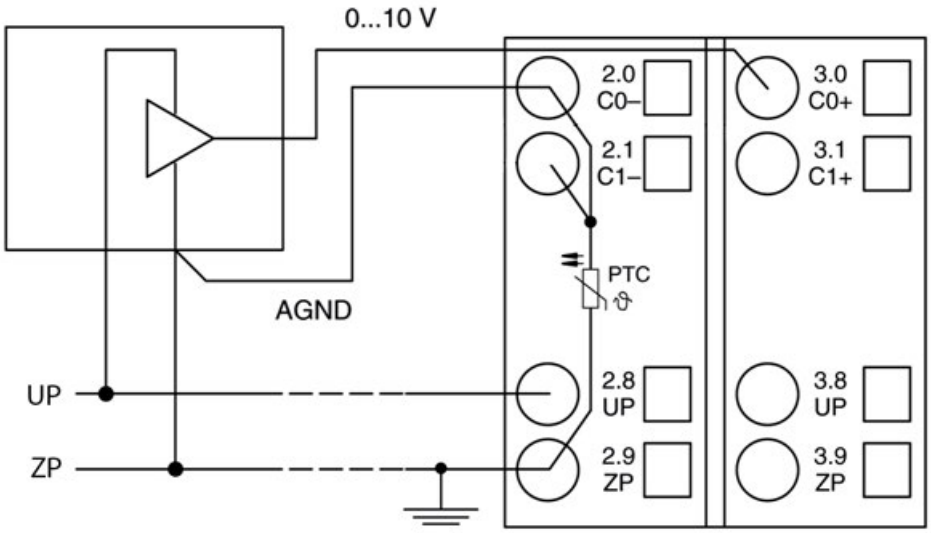


The following measuring ranges can be configured:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply



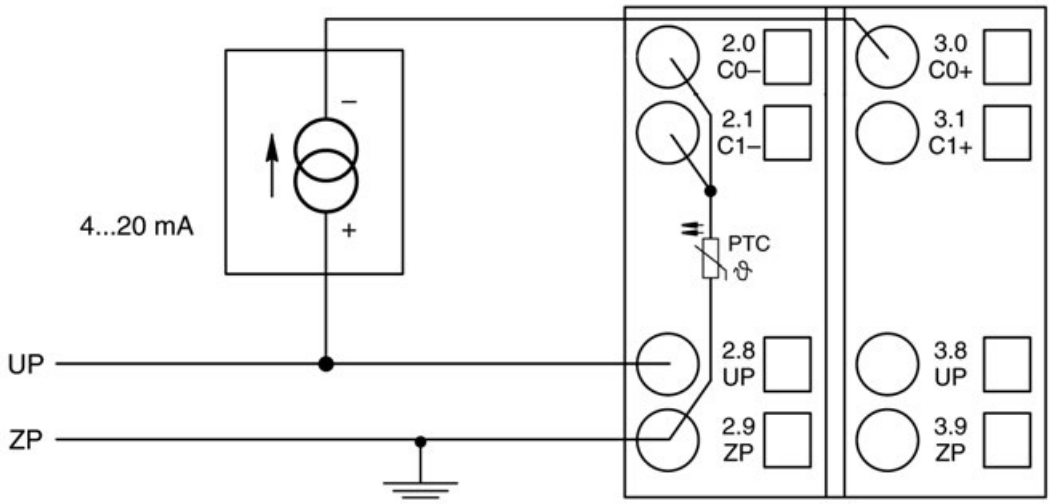
CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used
*) if the sensor can provide this signal range		

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)



Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

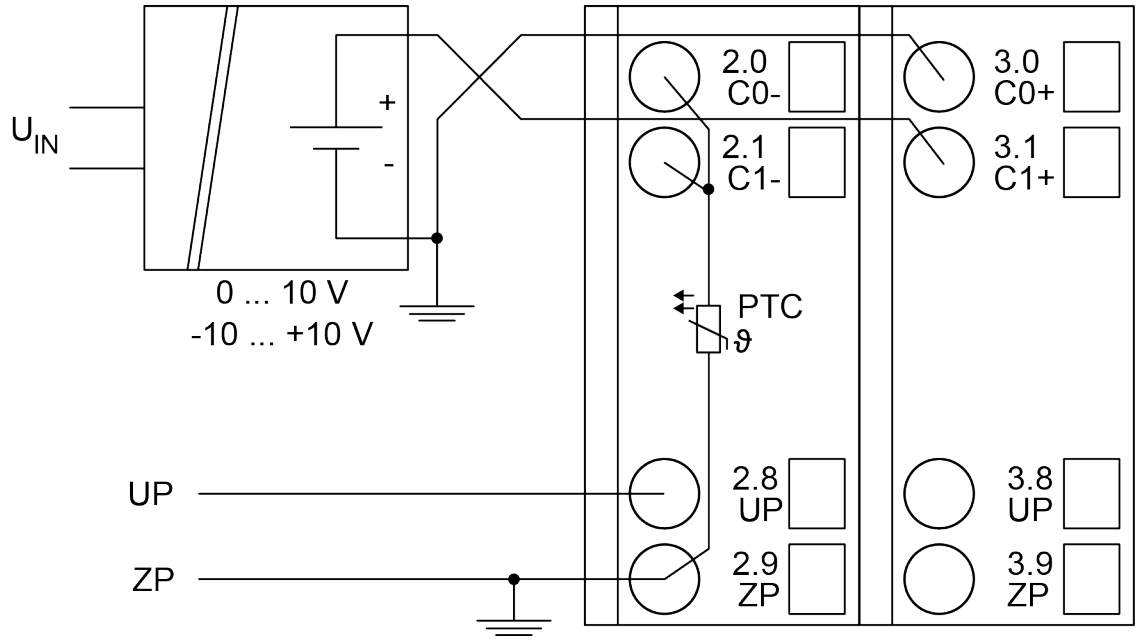
The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.



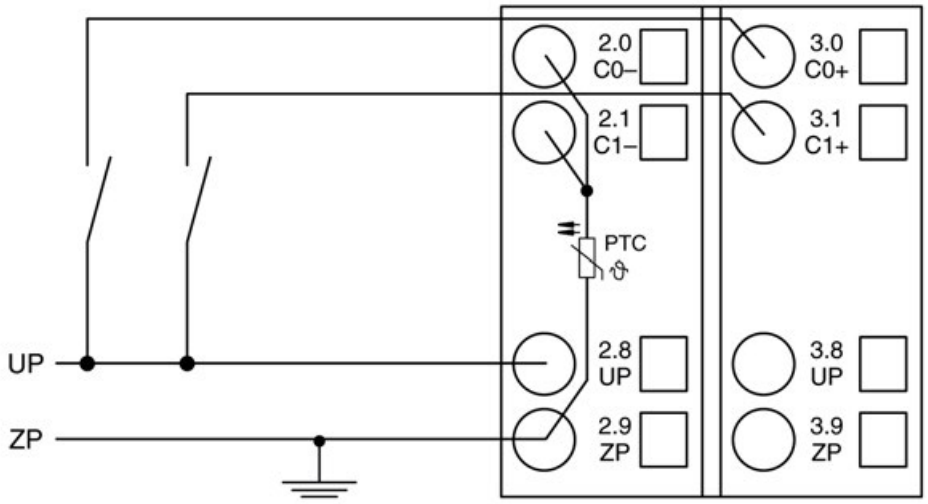
The negative pole of the sensor must be grounded next to the sensor.

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

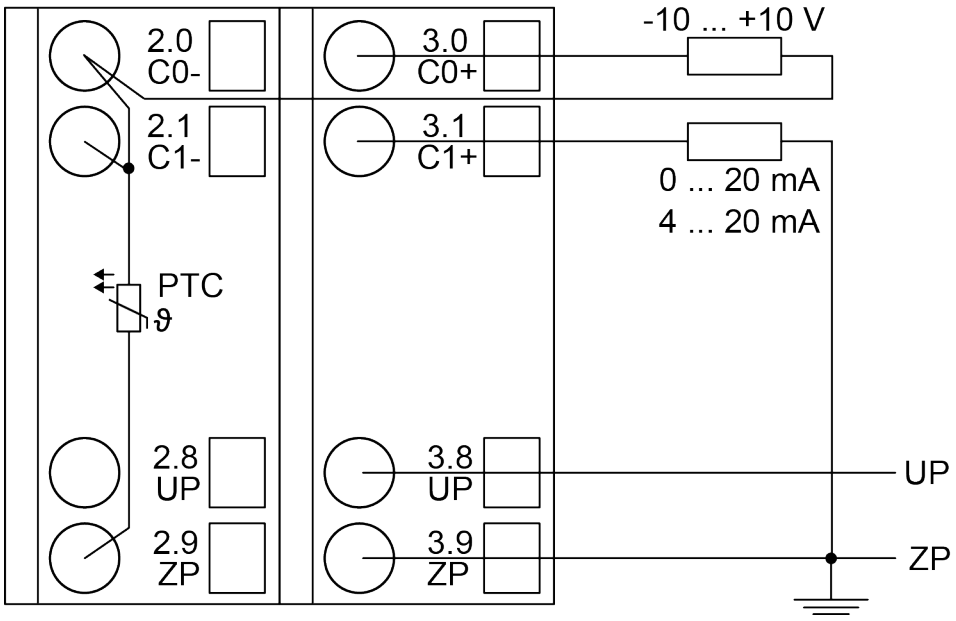
Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.



Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)



Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Analog inputs (words)	8
Analog outputs (words)	8

I/O configuration

The module does not store configuration data itself. The 8 configurable analog channels are defined as inputs or outputs by the configuration, i.e. each of the configurable channels can used as input or output (or re-readable output in case of voltage input/output).

When a channel is used as input, the corresponding output must be configured unused.

When a channel is used as output, the corresponding input must be configured unused.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1520 [1])	Word	1520 0x05f0	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	37	Byte	37-CPU 37-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel configuration Input channel 0	see table Channel configuration		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y07
9 to 22	Channel configuration and channel monitoring of the input channels 1 to 7	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y15
23	Channel configuration Output channel 0	see table Channel configuration		Byte	Default 0x00	0	130	0x0Y16
24	Channel monitoring Output channel 0	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y17
25	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y18
26 to 31	Channel configuration and channel monitoring of the output channels 1 to 3	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y19 to 0x0Y1E

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
32	Channel configuration Output channel 4	see table Channel configuration		Byte	Default 0x00	0	128	0x0Y1F
33	Channel monitoring Output channel 4	see table Channel monitoring		Byte	Default 0x00	0	3	0x0Y20
34 to 39	Channel configuration and channel monitoring of the output channels 5 to 7	see tables channel configuration and channel monitoring		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y21 to 0x0Y26

[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1

[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	40
Ext_User_Prm_Data_Const(0) =	0x05, 0xf1, 0x25, \ 0x01, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 431: Input channel (8x)

No.	Name	Internal value, type	Default
1	Channel configuration see table [2])	Byte	0 0x00 see table [2])
2	Channel monitoring see table [3])	Byte	0 0x00 see table [3])

Table 432: Channel configuration [2])

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 433: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
1	Open-circuit and short-circuit
2	Plausibility
3	No monitoring

Table 434: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 435: Output channels 1...7 (7x)

No.	Name	Internal value, type	Default
1	Channel configuration see table [4])	Byte	see table [4])
2	Channel monitoring see table [5])	Byte	see table [5])

Table 436: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7)
130	Analog output 4 mA...20 mA (not with the channels 4...7)

Table 437: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
1	Open-circuit (broken wire) and short-circuit
2	Plausibility
3	No monitoring

Table 438: Substitute value [6])

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value	Last value	0
Substitute value	Off or last value	1...65535

Diagnosis

Table 439: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value underflow	Input value overflow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 440: Content of diagnosis messages

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				

E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])				
3	14	1...10	31	31		26	Parameter error	Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31		11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31		45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

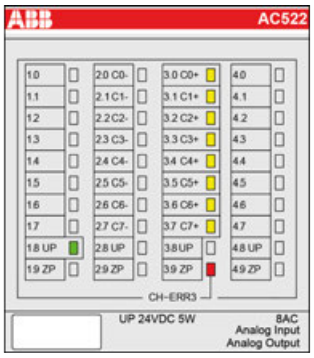
Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs/ outputs 00...07	Analog input/ output	Yellow	Input/output is OFF	Input/output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR3	Channel error, error messages combined into group 3	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	OFF	0	0000

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The System Data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

Only additional details are therefore documented below.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 VDC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 VDC power supply at the terminals UP/L+ and ZP/M of the CPU/bus module	Ca. 2 mA
	From UP at normal operation	0.10 A + output loads
Inrush current from UP (at power up)		0.040 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switch-gear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	8	
Distribution of channels into groups	1 group of 8 channels	
Connections of the channels C0- to C7-	Terminals 2.0 to 2.7	
Connections of the channels C0+ to C7+	Terminals 3.0 to 3.7	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	See table *Chapter 1.6.2.6.2.2.1.9.1 "Input ranges of voltage, current and digital input" on page 4428*	

Parameter	Value
Unused inputs	Must be configured as "unused".
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 8
Distribution of channels into groups		1 group of 8 channels
Connections of the channels C0+ to C7+		Terminals 3.0 to 3.7
Reference potential for the inputs		Terminals 1.9 to 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 VDC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		8, all channels for voltage, the first 4 channels also for current
Distribution of channels into groups		1 group of 8 channels
	Channels C0-...C7-	Terminals 2.0...2.7
	Channels C0+...C7+	Terminals 3.0...3.7
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table *Chapter 1.6.2.6.2.2.1.9.3 “Output ranges voltage and current” on page 4430*	
Unused outputs	Must be configured as "unused".	

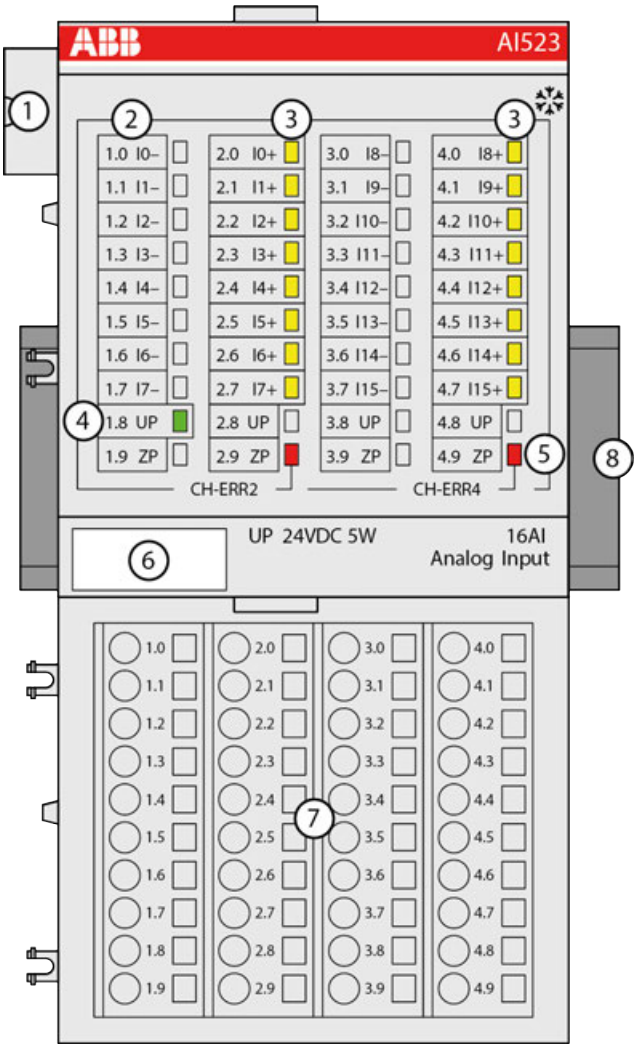
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 500 R0001	AC522, analog input/output module, 8 AC, U/I/RTD, 12 bits + sign, 2-wires	Active
1SAP 450 500 R0001	AC522-XC, analog input/output module, 8 AC, U/I/RTD, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AI523 - Analog input module

- 16 configurable analog inputs (I0 to I15) in 2 groups (1.0...2.7 and 3.0...4.7)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the analog inputs (I0 - I15)
4 1 green LED to display the state of the process supply voltage UP
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

16 analog inputs, individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V... +10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		19 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

The modules are plugged on an I/O terminal unit ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0- to I7-	Negative poles of the first 8 analog inputs
2.0 to 2.7	I0+ to I7+	Positive poles of the first 8 analog inputs
3.0 to 3.7	I8- to I15-	Negative poles of the following 8 analog inputs
4.0 to 4.7	I8+ to I15+	Positive poles of the following 8 analog inputs

CAUTION!
The negative poles of the analog inputs are galvanically connected to each other. They form an "Analog Ground" signal for the module. The negative poles of the analog outputs are also galvanically connected to each other to form an "Analog Ground" signal.

CAUTION!
There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

CAUTION!
Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AI523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the module:

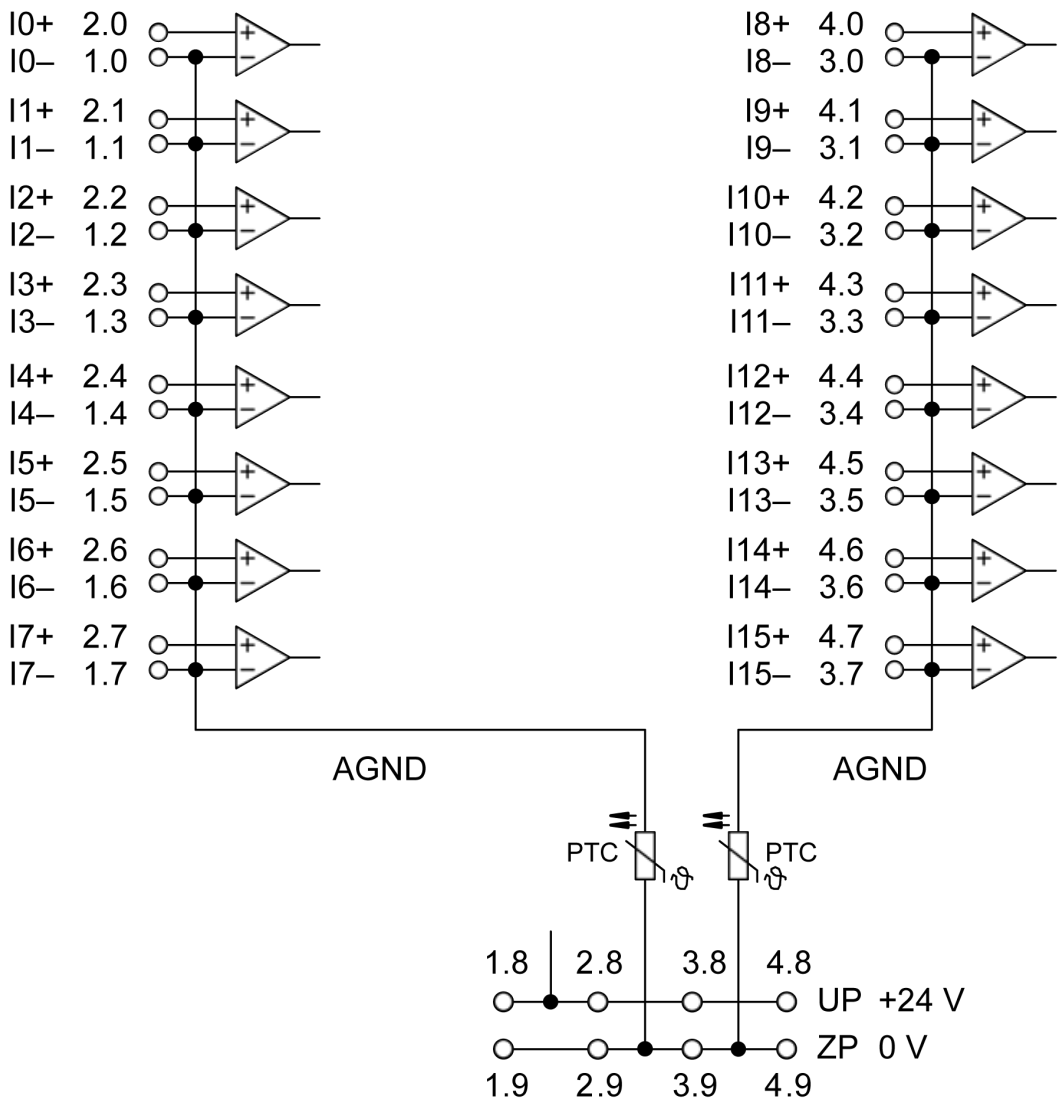


Fig. 845: 16 analog inputs in two groups, individually configurable *Chapter 1.6.2.6.2.2.2.2 “Functionality” on page 4434*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The modules provide several diagnosis functions *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI523 provides a constant current source which is multiplexed over the 8 analog channels.

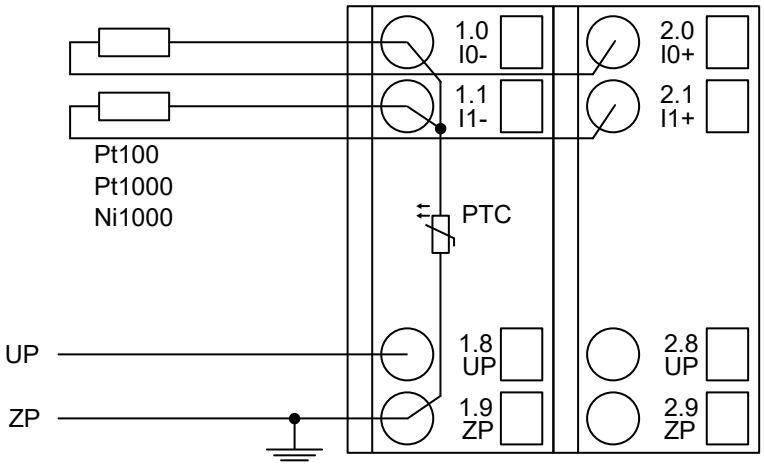


Fig. 846: Connection example

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446*.

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The function of the LEDs is described under Displays *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

The module AI523 performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI523 provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

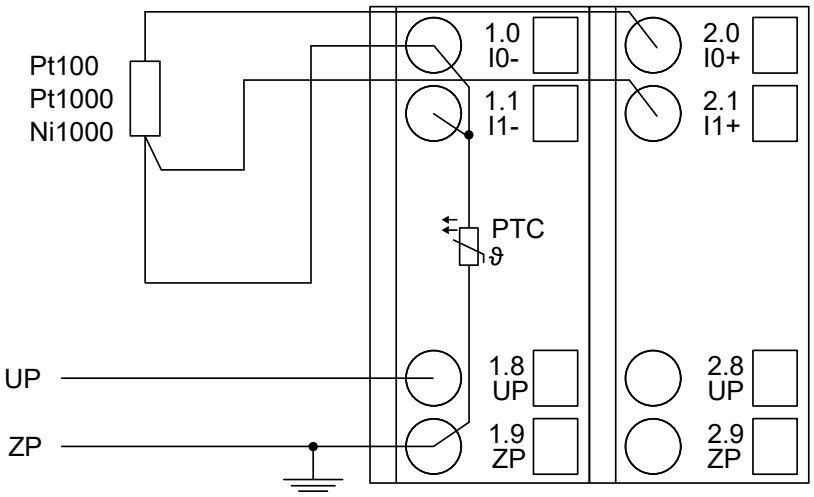


Fig. 847: Connection example

> *If several measuring points are adjacent to each other, the return line is necessary only once. This saves wiring costs.*

With 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446*

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The function of the LEDs is described under Displays *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

The module AI523 performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

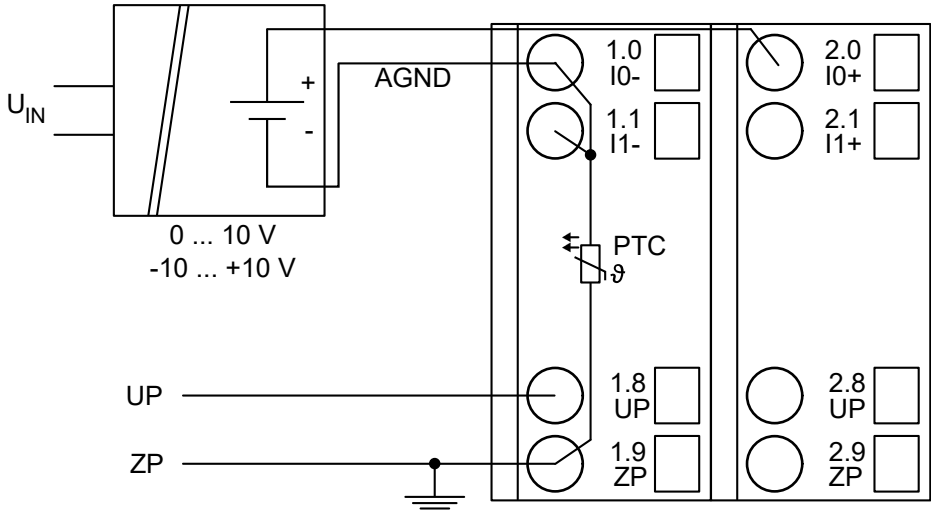


Fig. 848: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446* ↳ *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

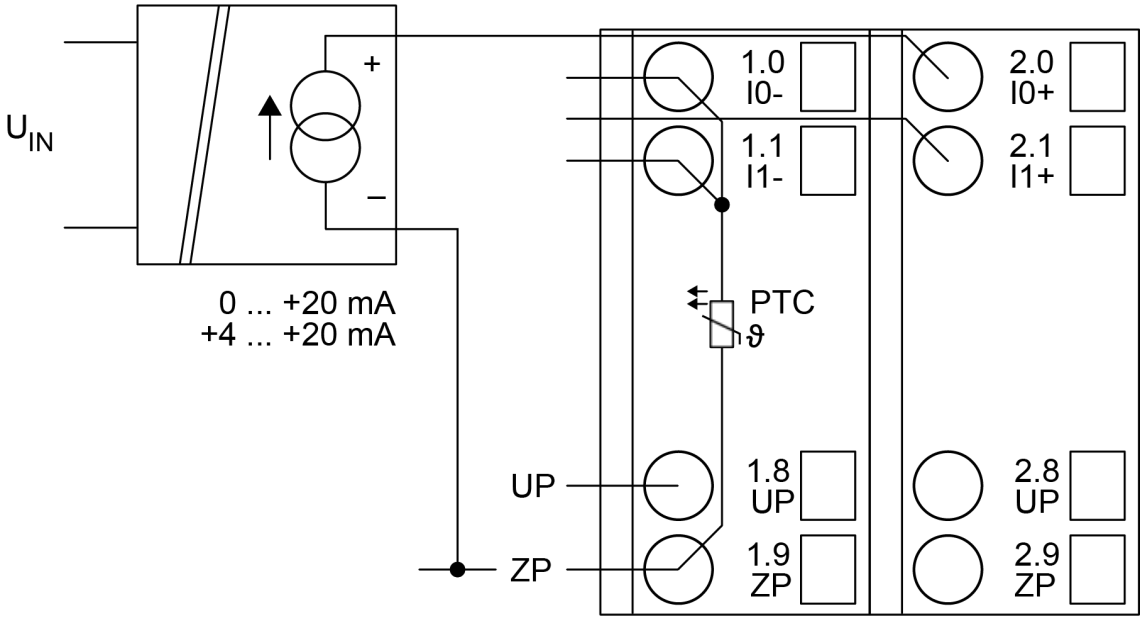


Fig. 849: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446* ↳ *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

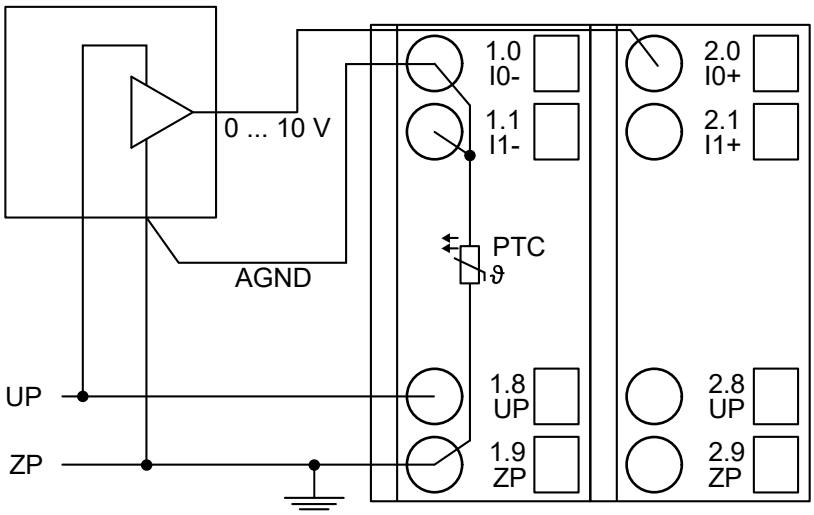


Fig. 850: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1 V, not even in case of long lines .

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very low current flows over the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method has to be preferred.

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used
*) if the sensor can provide this signal range		

The function of the LEDs is described under Displays *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

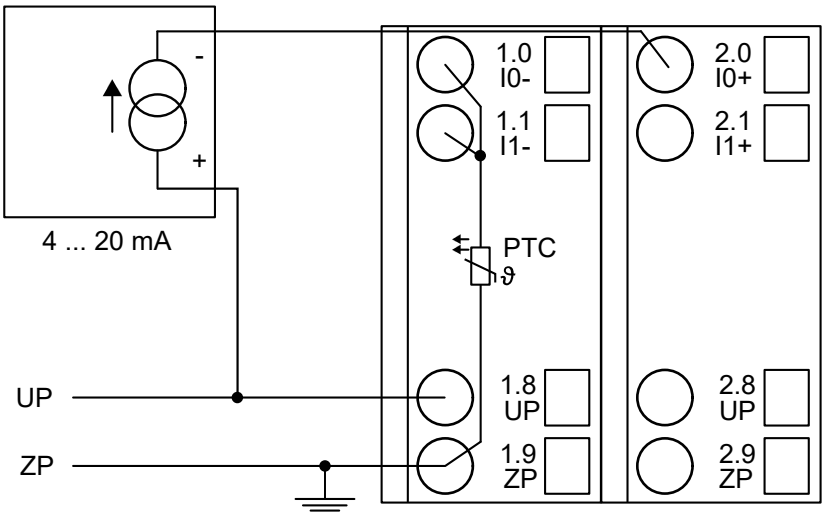


Fig. 851: Connection example

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446* *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*

Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Displays *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second into an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10 volt Zener diode (in parallel to I+ and I-). But, in general, it is a better solution to use sensors with fast initialization or without current peaks higher than 25 mA.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise problems can occur concerning the common-mode input voltages of the involved analog inputs.

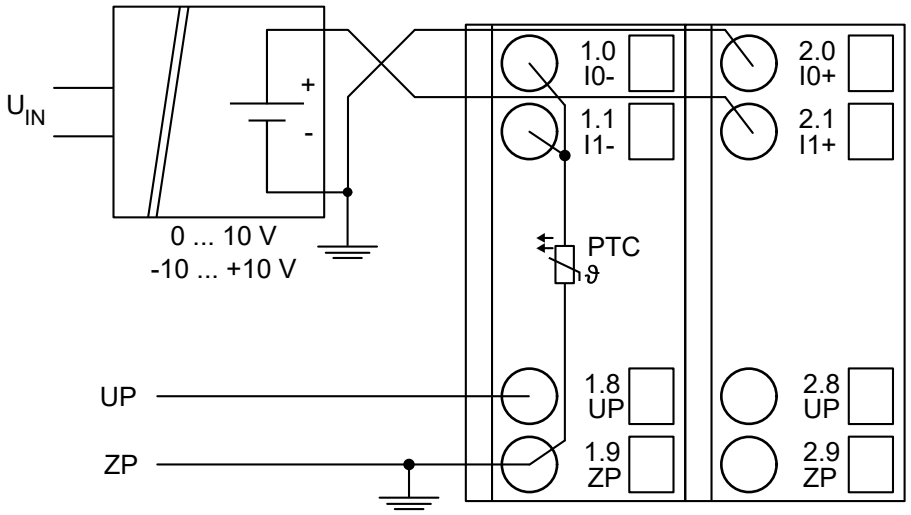


Fig. 852: Connection example

The negative pole of the sensor must be grounded next to the sensor.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446* ↳ *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

The function of the LEDs is described under Displays ↳ *Chapter 1.6.2.6.2.2.2.7 “Diagnosis” on page 4449*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

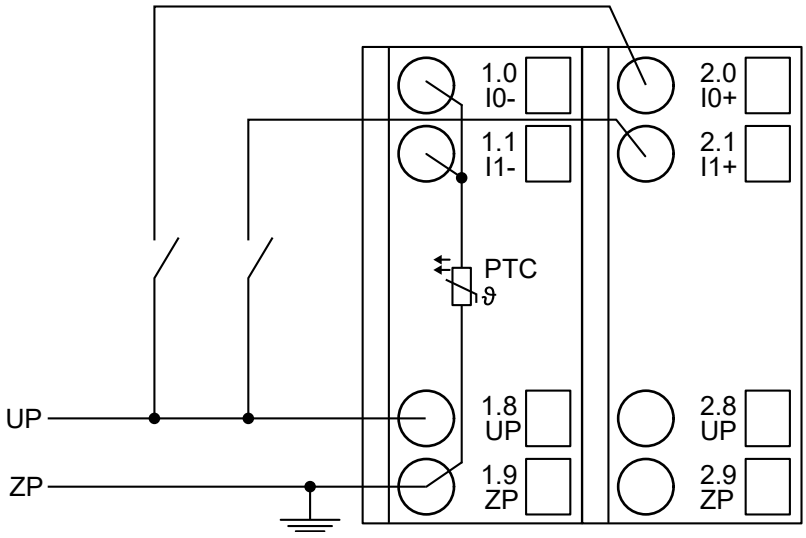


Fig. 853: Connection example

The following operating mode can be configured ↳ *Chapter 1.6.2.6.2.2.2.6 “Parameterization” on page 4446* ↳ *Chapter 1.6.2.6.2.2.2.9 “Measuring ranges” on page 4451*

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

The function of the LEDs is described under Displays.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	16
Counter output data (words)	0

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

That means replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1515 [1)]	Word	1515 0x05eb	0	65535	0x0Y01
2	Ignore module [2)]	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	34	Byte	34-CPU 34-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Channel configuration Input channel 0	See *Table 441 “Channel configuration [2)]” on page 4448*		Byte	Default 0x00	0	19	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel monitoring Input channel 0	See *Table 442 “Channel monitoring [4])” on page 4448*		Byte	Default 0x00	0	3	0x0Y06
8 to 35	Channel configuration and channel monitoring of the input channels 1 to 14	See *Table 441 “Channel configuration [2])” on page 4448* and *Table 442 “Channel monitoring [4])” on page 4448*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y07 to 0x0Y22
36	Channel configuration Input channel 15	See *Table 441 “Channel configuration [2])” on page 4448*		Byte	Default 0x00	0	19	0x0Y23
37	Channel monitoring Input channel 15	See *Table 442 “Channel monitoring [4])” on page 4448*		Byte	Default 0x00	0	3	0x0Y24
[1]) With CS31 and addresses less than 70 and FBP, the value is increased by 1 [2]) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len =	37
Ext_User_Prm_Data_Const(0) =	0x05, 0xec, 0x22, \ 0x01, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Input channel (16 x with AI523)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [2])	see table [2])	Byte	0 0x00 see [3])
2	Channel monitoring	see table [4])	see table [4])	Byte	0 0x00 see [5])

Table 441: Channel configuration [2])

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default) [3])
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 442: Channel monitoring [4])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit [5])
1	Open-circuit and short circuit
2	Plausibility
3	No monitoring

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	

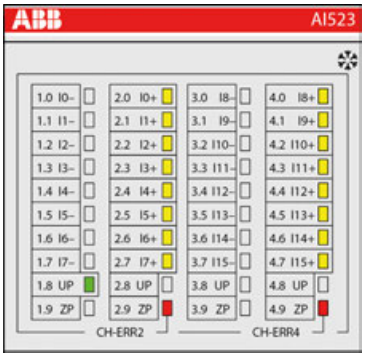
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON -> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...15	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...15	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	0...15	47	Short circuit at an analog input	Check ter-minal
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1..10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI); COM1/COM2: 1..10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7 and I8...I15	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	OFF	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	< -1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

The resolution corresponds to 16 bits.

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation / with outputs	0.15 A + output loads
Inrush current from UP (at power up)		0.050 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm^2	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	16	
Distribution of channels into groups	2 groups of 8 channels each	
Connections of the channels I0- to I7- Connections of the channels I0+ to I7+	Terminals 1.0 to 1.7 Terminals 2.0 to 2.7	
Connections of the channels I8- to I15- Connections of the channels I8+ to I15+	Terminals 3.0 to 3.7 Terminals 4.0 to 4.7	
Input type	Bipolar (not with current or Pt100/ Pt1000/ Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0/4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	1 LED per channel	
Conversion cycle	2 ms (for 16 inputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits	
	Range -10 V...+10 V: 12 bits + sign	
	Range 0 mA...20 mA: 12 bits	
	Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C

Parameter	Value	
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	↳ *Chapter 1.6.2.6.2.2.2.9.1 “Input ranges of voltage, current and digital input” on page 4451* ↳ *Chapter 1.6.2.6.2.2.2.9.2 “Input ranges resistance temperature detector” on page 4451*	
Unused voltage inputs	Are configured as "unused"	
Unused current inputs	Have a low resistance, can be left open-circuited	
Overvoltage protection	Yes	

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 16
Distribution of channels into groups		2 groups of 8 channels each
Connections of the channels I0+ to I7+		Terminals 2.0 to 2.7
Connections of the channels I8+ to I15+		Terminals 4.0 to 4.7
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

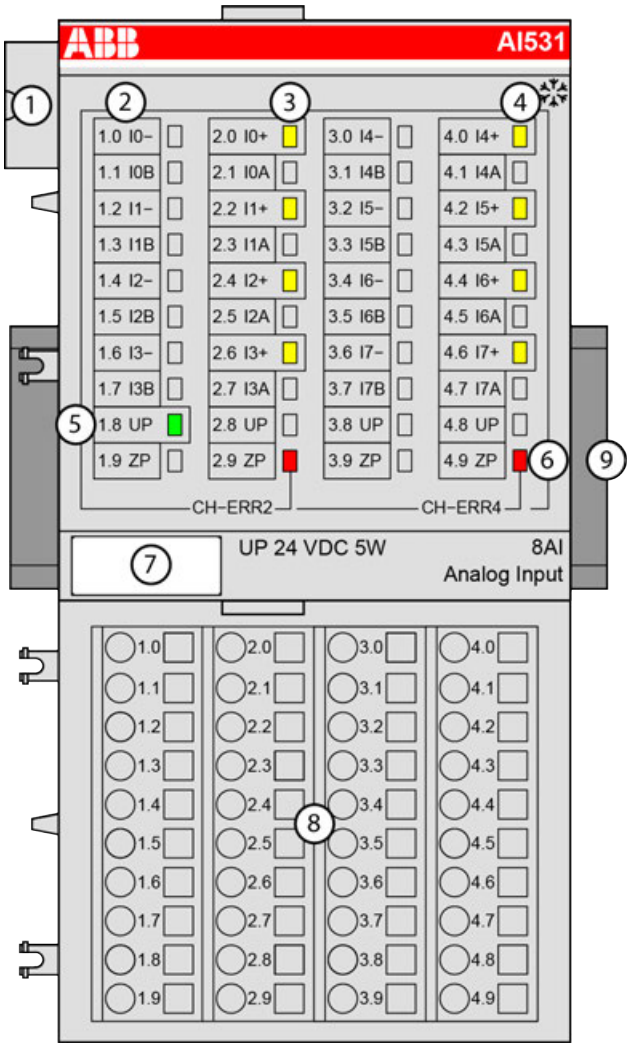
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 300 R0001	AI523, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 300 R0001	AI523-XC, analog input module, 16 AI, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

AI531 - Analog input module

- 8 configurable analog inputs (I0 to I7) in 2 groups (1.0...1.7 and 2.0...2.7 as well as 3.0...3.7 and 4.0...4.7)
  Resolution 15 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal names
3 4 yellow LEDs to display the states at the inputs I0 to I3
4 4 yellow LEDs to display the states at the inputs I4 to I7
5 1 green LED to display the process supply voltage UP
6 2 red LEDs to display errors (CH-ERR2 and CH-ERR4)
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

8 analog inputs, individually configurable for

- Unused (default setting)
- 0 V...5 V, 0 V...10 V
- -50 mV...+50 mV, -500 mV...+500 mV
- -1 V...+1 V, -5 V...+5 V, -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- -20 mA...20 mA
- Pt100, -50 °C...+70 °C or 400 °C (2-, 3- and 4-wire)
- Pt100, -200 °C...+850 °C (2-, 3- and 4-wire)
- Pt1000, -50 °C...+400 °C (2-, 3- and 4-wire)
- Ni1000, -50 °C...+150 °C (2-, 3- and 4-wire)
- Cu50 (1.426): -50 °C...+200 °C (2-, 3- and 4-wire)
- Cu50 (1.428): -200 °C...+200 °C (2-, 3- and 4-wire)
- 0 Ω...50 kΩ
- Thermocouples of types J, K, T, N, S
- Resistance measuring bridge
- Digital signals (digital input)

Parameter		Value
Resolution of the analog channels		
	Voltage and current, bipolar	15 bits plus sign
	Voltage and current, unipolar	15 bits
	Temperature	0.1 °C (0,01°C at Pt100 -50 °C...+70 °C)
LED displays		11 LEDs for signals and error messages
Internal power supply		through the I/O bus interface (I/O bus)
External power supply		via terminals (process voltage UP = 24 V DC)
Required terminal unit		TU515 or TU516 ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The modules are plugged on an I/O terminal unit *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8, 4.8, 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V

The assignment of the other terminals:

Terminals	Signal	Description
2.0, 2.2, 2.4, 2.6	I0+ to I3+	Positive poles of the first 4 analog inputs
1.0, 1.2, 1.4, 1.6	I0- to I3-	Negative poles of the first 4 analog inputs
2.1, 2.3, 2.5, 2.7	I0A to I3A	Connections A (supply) of the first 4 analog inputs
1.1, 1.3, 1.5, 1.7	I0B to I3B	Connections B (analog ground) of the first 4 analog inputs
4.0, 4.2, 4.4, 4.6	I4+ to I7+	Positive poles of the following 4 analog inputs
3.0, 3.2, 3.4, 3.6	I4- to I7-	Negative poles of the following 4 analog inputs
4.1, 4.3, 4.5, 4.7	I4A to I7A	Connections A (supply) of the following 4 analog inputs
3.1, 3.3, 3.5, 3.7	I4B to I7B	Connections B (analog ground) of the following 4 analog inputs

CAUTION!
Analog sensors must be galvanically isolated against the ground. In order to avoid inaccuracy with the measuring results, the analog sensors should also be isolated against the power supply.

The "IxB" clamps (x=0..7) of the analog inputs are galvanically connected to each other. They form an "Analog Ground Signal" (AGND) for the module.

The negative poles of the analog inputs Ix- may accept a potential difference up to ±20 V DC with regard to the common reference potential IxB (AGND, ZP). Observing this maximum voltage difference, analog current inputs of one module can be switched in series to each other and also with current inputs of other modules.

For the open-circuit detection (cut wire), each positive analog input channel Ix+ is pulled up to "plus" by a high-resistance resistor and each negative analog input channel Ix- is pulled down to "minus" by a resistor. If cut wire occurs, a maximum voltage (overflow or underflow) will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AI531.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

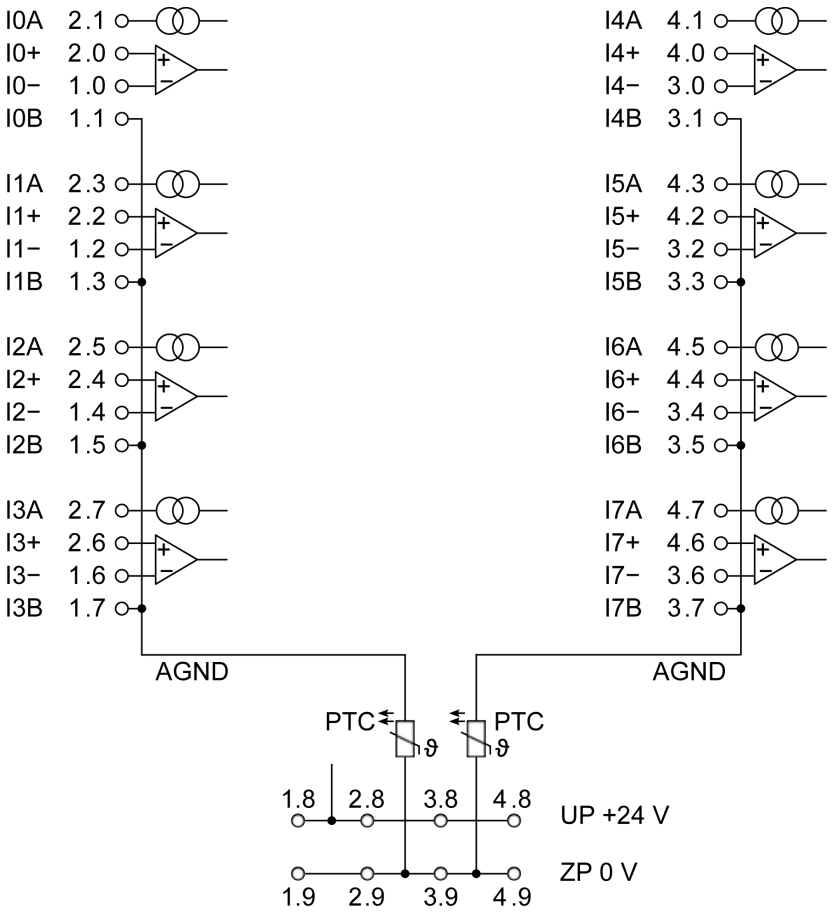


Fig. 854: 8 analog inputs in two groups, individually configurable ↬ *Chapter 1.6.2.6.2.2.3.2 “Functionality” on page 4456*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The module provides several diagnosis functions ↬ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

Standard ranges

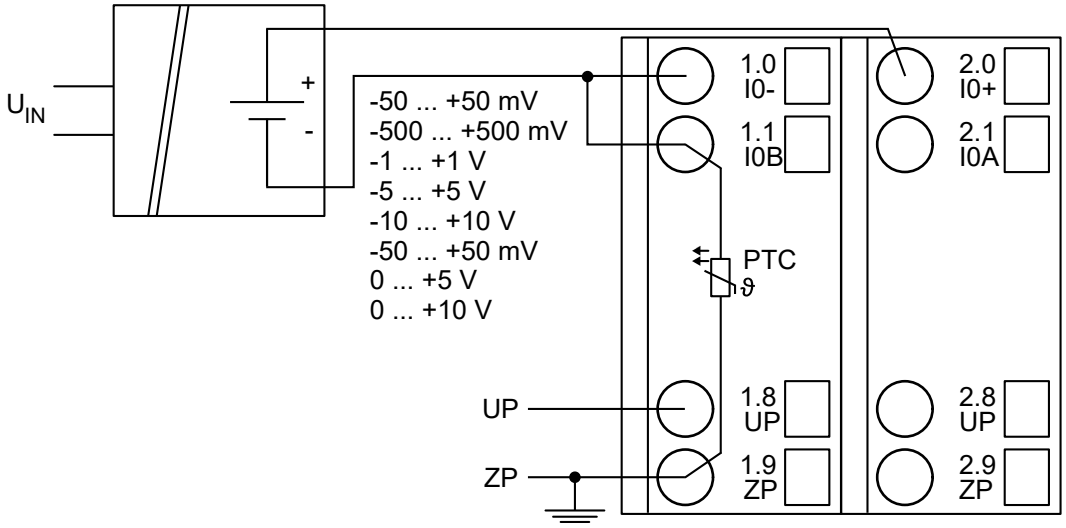


Fig. 855: Connection example

The measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Voltage	-50 mV...+50 mV	1 channel used
Voltage	-500 mV...+500 mV	1 channel used
Voltage	-1 V...+1 V	1 channel used
Voltage	-5 V...+5 V	1 channel used
Voltage	-10 V...+10 V	1 channel used
Voltage	0 V...+5 V	1 channel used
Voltage	0 V...+10 V	1 channel used

Common mode range (+/-20 V)

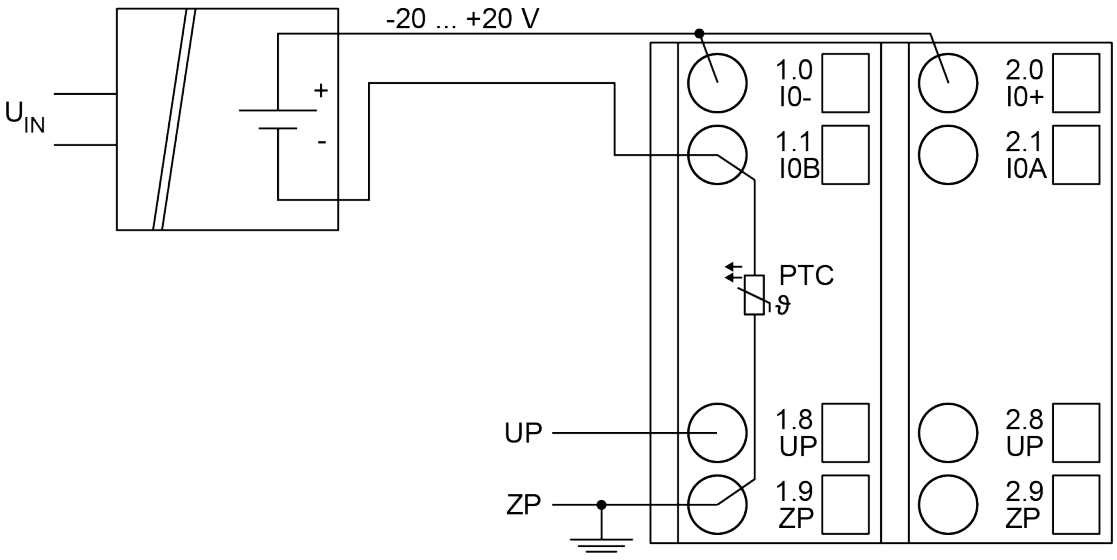


Fig. 856: Connection example

The measuring range can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Voltage	Common mode voltage	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

Standard ranges

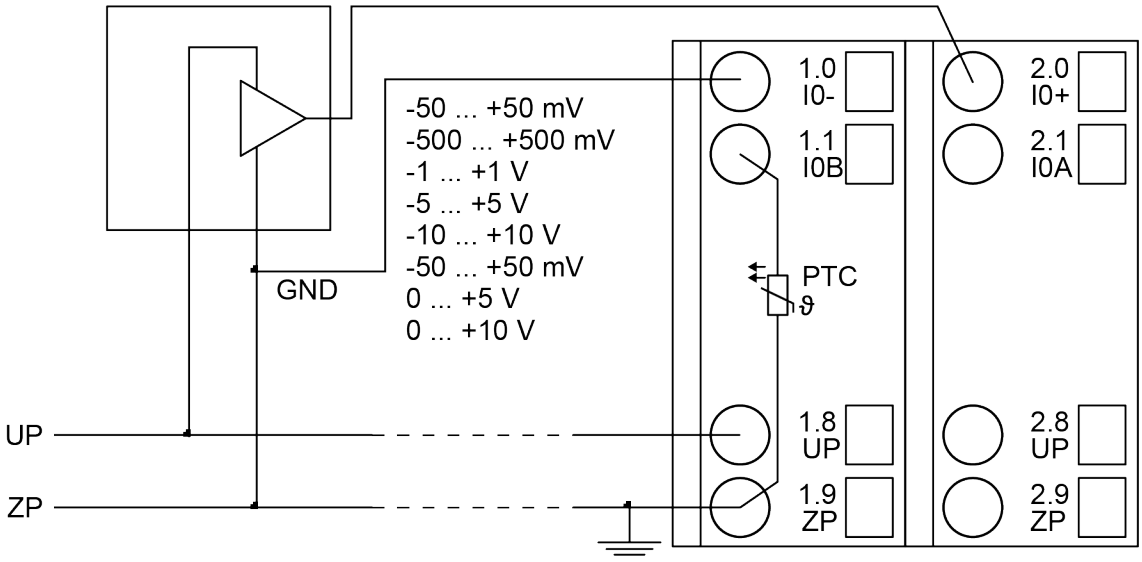


Fig. 857: Connection example

CAUTION!
If GND is not directly connected to ZP at the sensor, the supply current flows via the GND line to ZP. Measuring errors can only occur caused by voltage differences higher than ±20 V DC between GND and ZP.

The measuring ranges can be configured *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474* :

Voltage	-50 mV...+50 mV	1 channel used
Voltage	-500 mV...+500 mV	1 channel used
Voltage	-1 V...+1 V	1 channel used
Voltage	-5 V...+5 V	1 channel used
Voltage	-10 V...+10 V	1 channel used
Voltage	0 V...+5 V	1 channel used
Voltage	0 V...+10 V	1 channel used

Common mode range (+/-20 V)

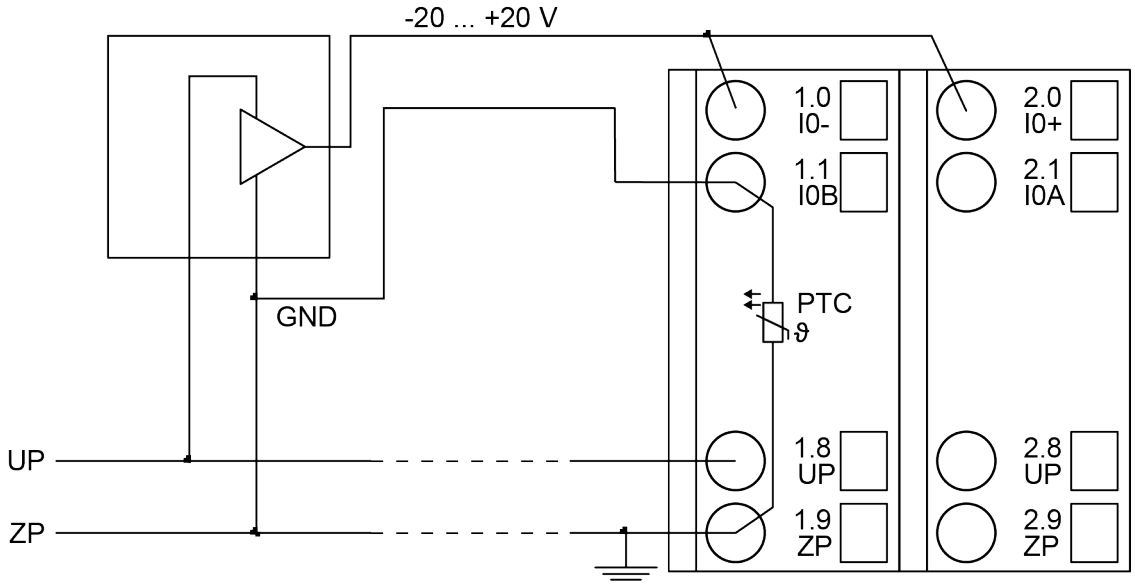


Fig. 858: Connection example

CAUTION!
If GND is not directly connected to ZP at the sensor, the supply current flows via the GND line to ZP. Measuring errors can only occur caused by voltage differences higher than ±20 V DC between GND and ZP.

The measuring range can be configured *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Voltage	Common mode voltage	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

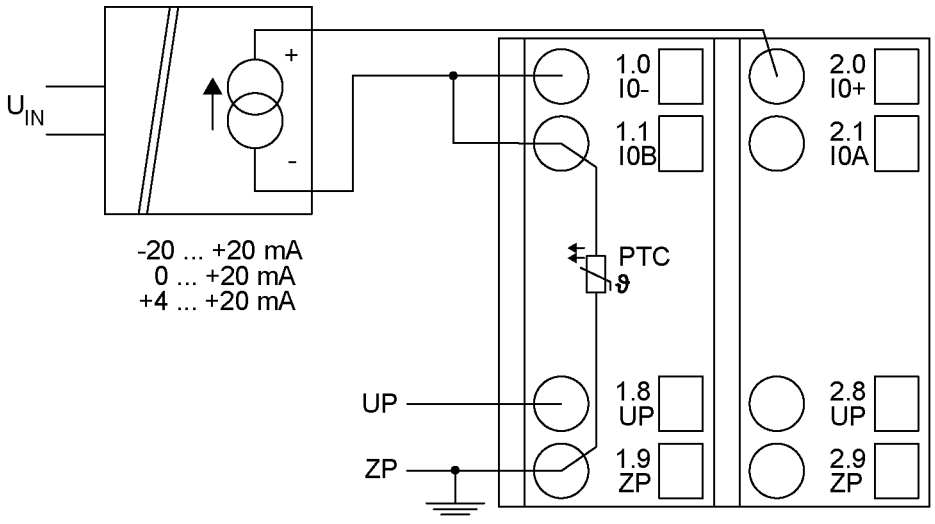


Fig. 859: Connection example

Figure:

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Current	-20 mA...20 mA	1 channel used
Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

Unused input channels can be left open, because they are of low resistance.

Connection of active-type analog sensors (Current) with galvanically isolated power supply and series-connection of an additional input

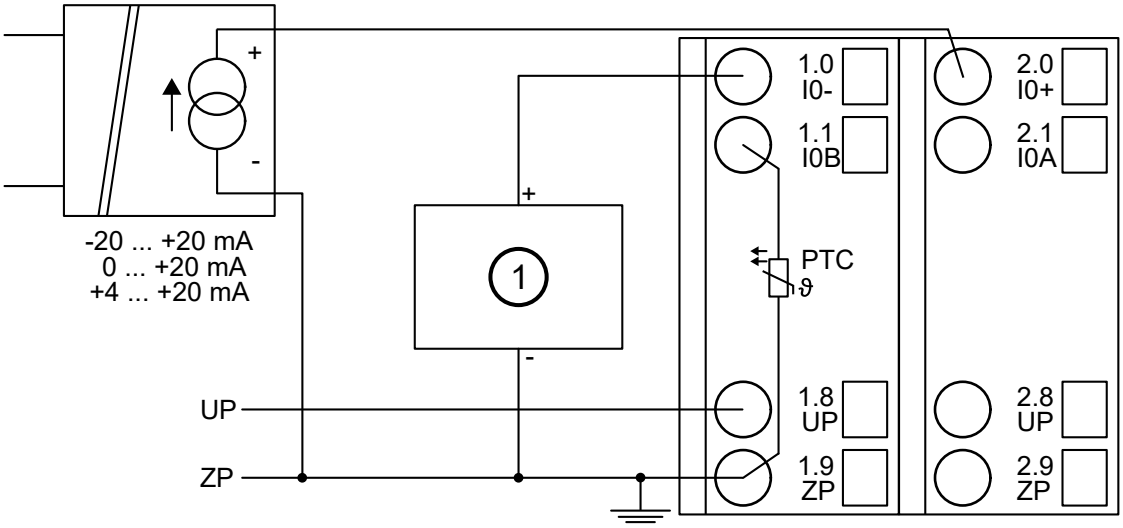


Fig. 860: Connection example

1 Analog input of the second device

If series-connection of an additional input is used, the input resistance of the module (ca. 330 Ω) must be added to the input resistance of the second device. Make sure that the maximum permitted load resistance of the analog sensor is not exceeded (see the data sheet of the analog sensor).

The input of the module is not related to ZP. If the input of the second device is related to ZP, the order of sequence in the series-connection must be observed by all means (from the sensor to the module and then to the input of the second device).

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Current	-20 mA...20 mA	1 channel used
Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

For a description of the functions of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

Unused input channels can be left open, because they are of low resistance.

Connection of passive-type analog sensors (Current)

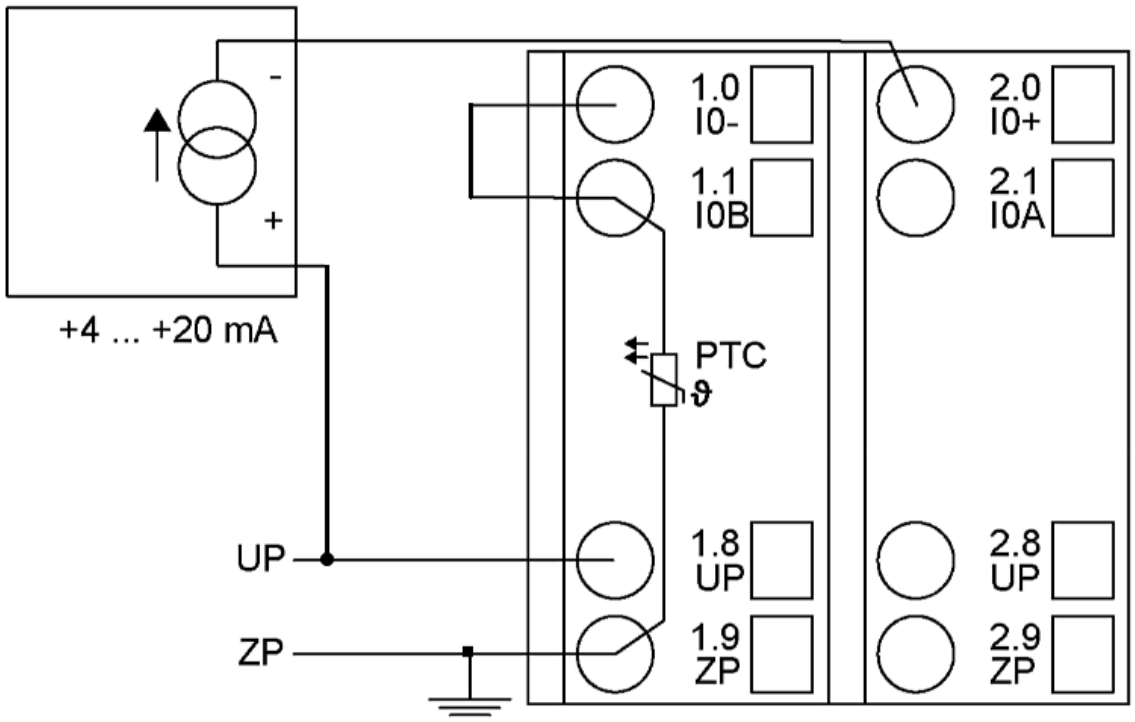


Fig. 861: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Current	-20 mA... 20 mA *)	1 channel used
Current	0 mA... 20 mA *)	1 channel used
Current	4 mA... 20 mA	1 channel used
*) This setting is not applicable with passive-type analog sensors (current).		

The function of the LEDs is described under Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

Unused input channels can be left open, because they are of low resistance.

Connection of passive-type analog sensors (Current) and series-connection of an additional analog sensor

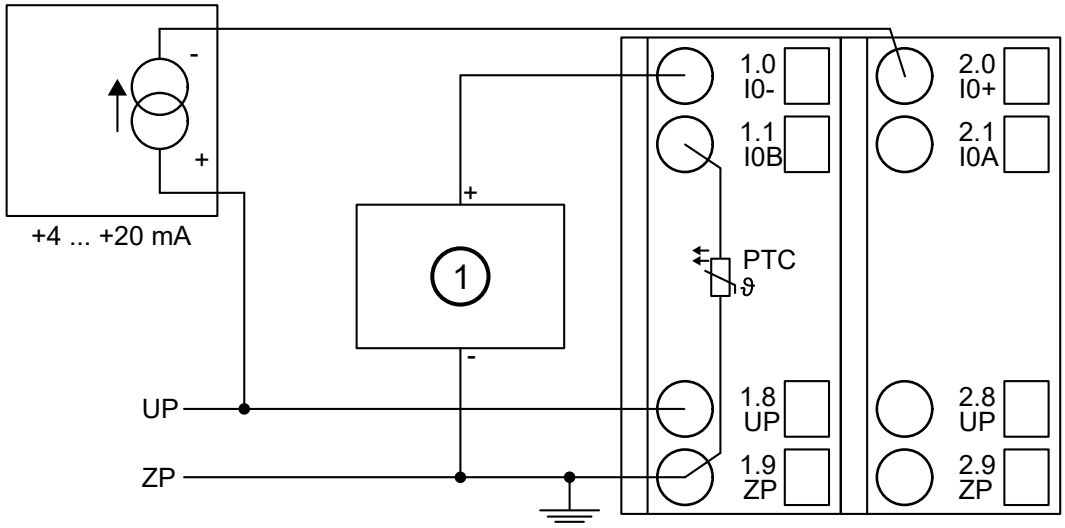


Fig. 862: Connection example

1 Analog input of the second device

> *If series-connection of an additional input is used, the input resistance of the module (ca. 330 Ω) must be added to the input resistance of the second device. Make sure that the maximum permitted load resistance of the analog sensor is not exceeded (see the data sheet of the analog sensor).*

> *The input of the module is not related to ZP. If the input of the second device is related to ZP, the order of sequence in the series-connection must be observed by all means (from the sensor to the module and then to the input of the second device).*

The following measuring ranges can be configured ↬ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Current	-20 mA...20 mA *)	1 channel used
Current	0 mA...20 mA *)	1 channel used
Current	4 mA...20 mA	1 channel used
*) This setting is not applicable with passive-type analog sensors (current).		

The function of the LEDs is described under Diagnosis and displays / displays ↬ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

Unused input channels can be left open, because they are of low resistance.

Connection of digital signal sources at analog inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

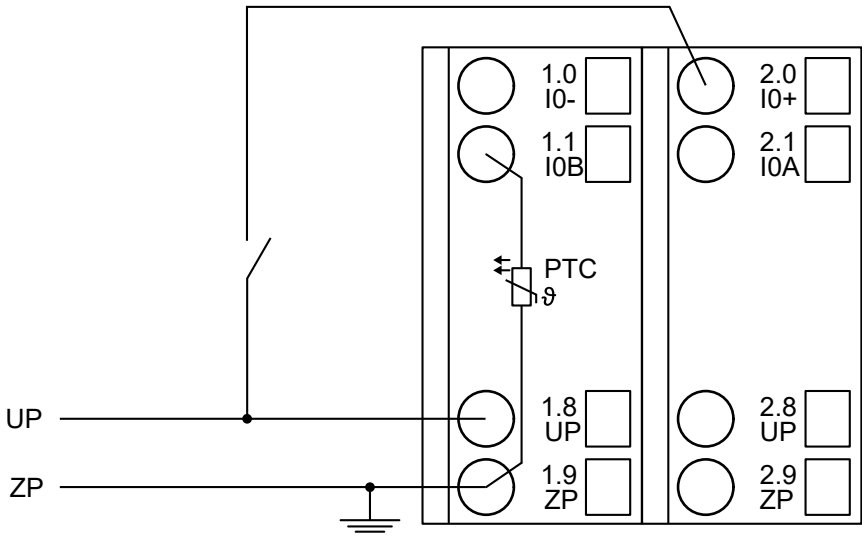


Fig. 863: Connection example

The following operating mode can be configured *Chapter 1.6.2.6.2.2.3.6 "Parameterization" on page 4474* :

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays *Chapter 1.6.2.6.2.2.3.7 "Diagnosis" on page 4477*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

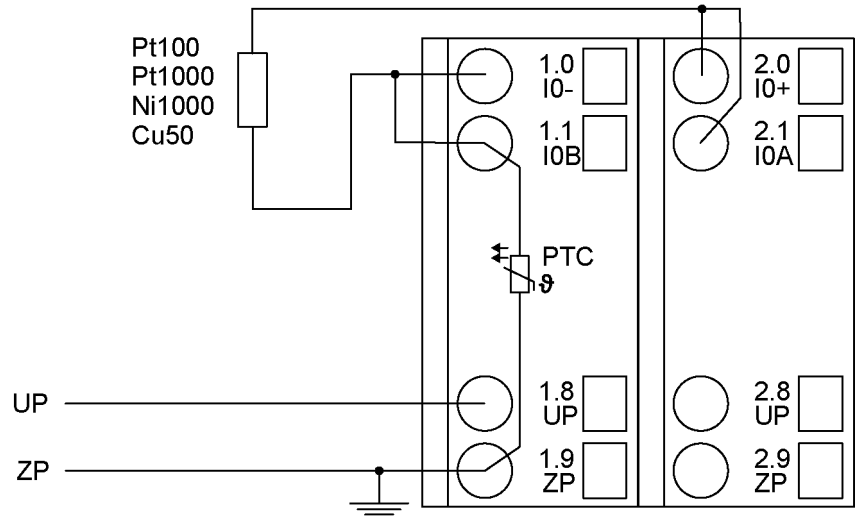


Fig. 864: Connection example

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.3.6 "Parameterization" on page 4474*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C...+850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

The module linearizes the resistance thermometer characteristics.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

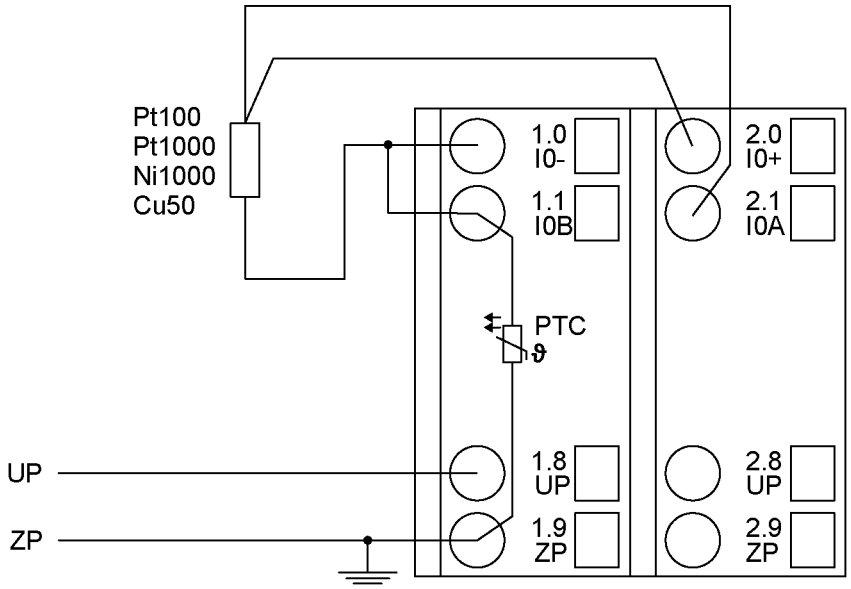


Fig. 865: Connection example

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C ... +850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

The module linearizes the resistance thermometer characteristics. In order to keep measuring errors as small as possible, it is necessary by all means to have all the involved conductors in the same cable. All the conductors must have the same cross section.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 4-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000, Cu50) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

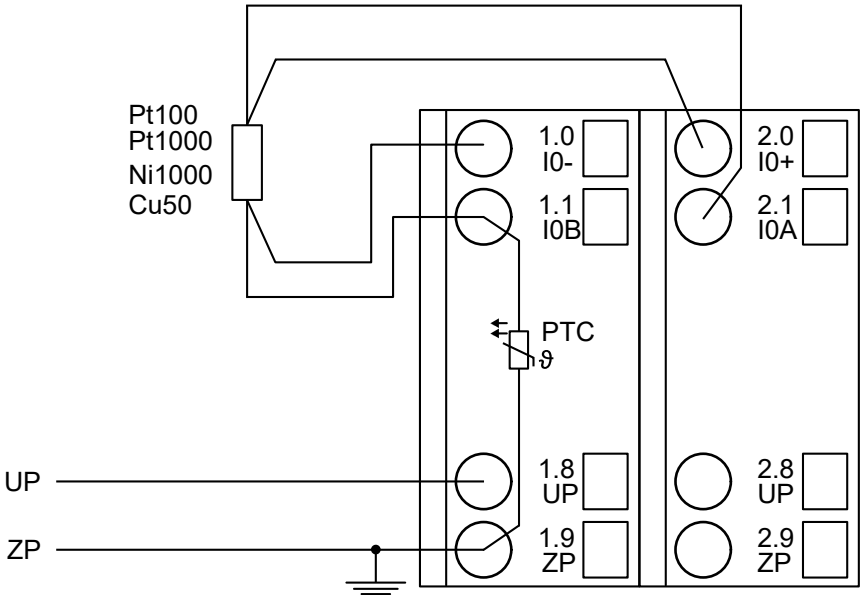


Fig. 866: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*:

Pt100	-50 °C...+70 °C / +400 °C; -200 °C...+850 °C	1 channel used
Pt1000	-50 °C...+400 °C	1 channel used
Ni1000	-50 °C...+150 °C	1 channel used
Cu50	-50 °C...+200 °C (1.426); -200 °C...+200 °C (1.428)	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

The module linearizes the resistance thermometer characteristics. In order to keep measuring errors as small as possible, it is necessary by all means, to have all the involved conductors in the same cable.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistors in 2-wire configuration

For evaluating resistors, a constant current must flow through them to build the necessary voltage drop. For this, the module AI531 provides a constant current source which is multiplexed over the 4 analog channels.

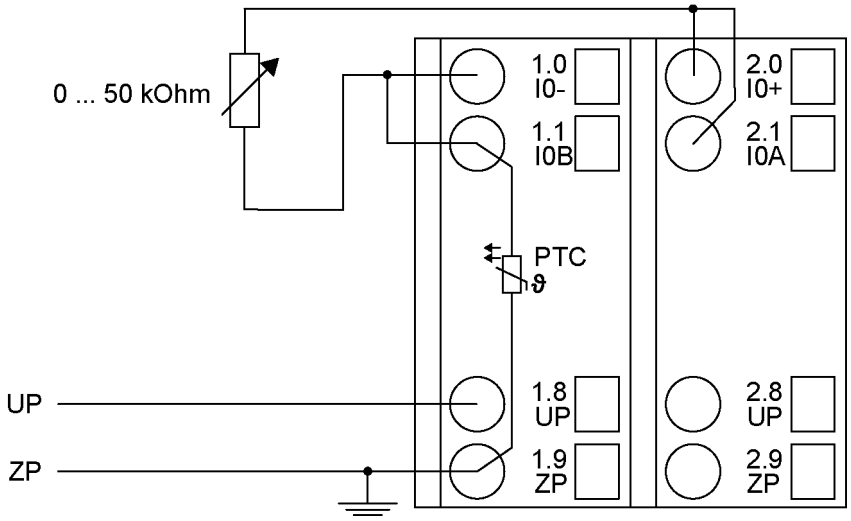


Fig. 867: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474* :

Resistor	50 kΩ	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of a resistance measuring bridge with internal supply

When resistance measuring bridges are connected, the short-circuit-proof voltage output (internal supply) at pin I0A (or I2A, I4A, I6A) must be used. This supply voltage is activated as soon as "Voltage Measurement" is configured for the relevant channel.

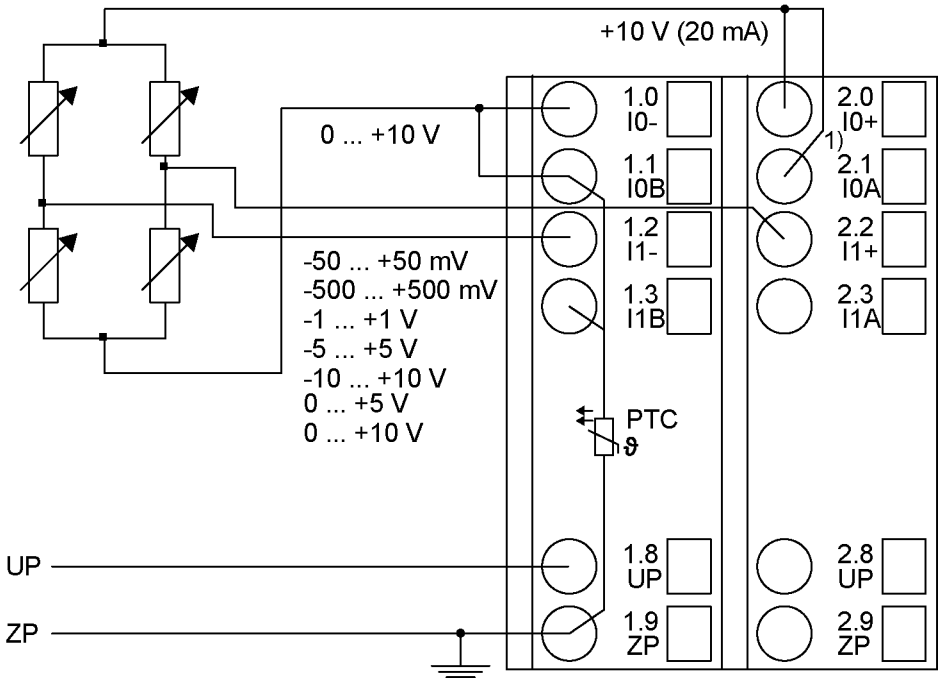


Fig. 868: Connection example

1 Internal supply

All voltage measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474*.

The calculation of the resistor deviation must be performed via the bridge voltage by the PLC user program.

Connection of a resistance measuring bridge with external supply

With the connection of a resistance measuring bridge with external supply, the supply voltage is provided separately.

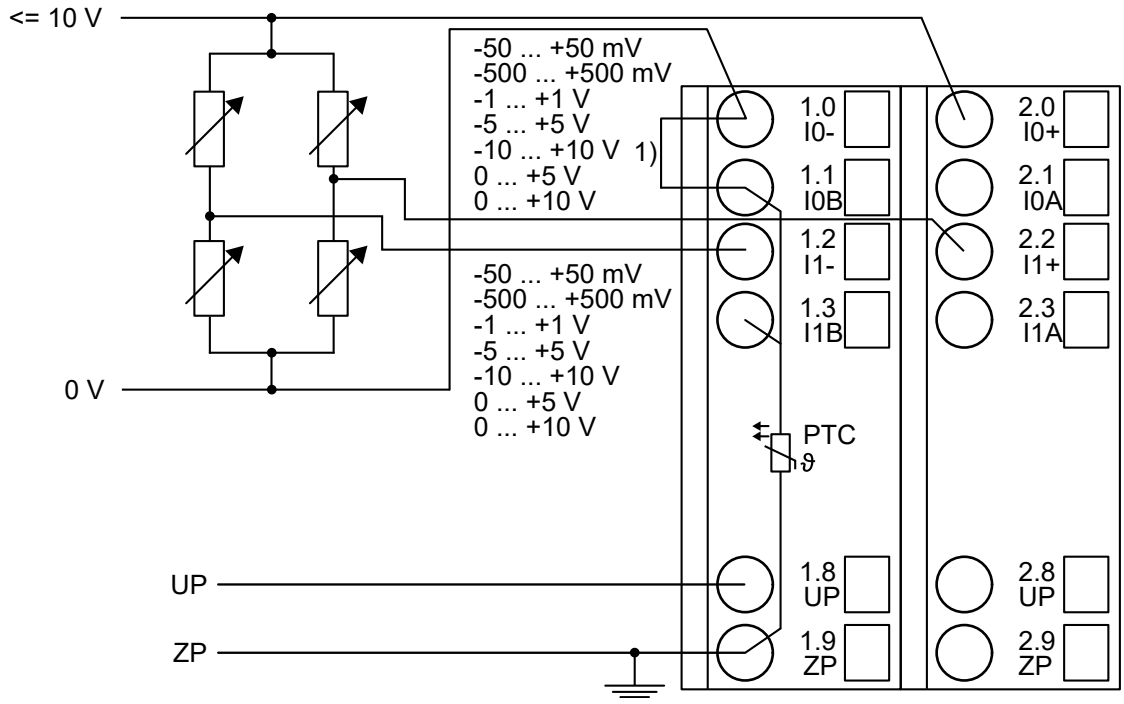


Fig. 869: Connection example

1 Bridge to IxB necessary with galvanically isolated supply

All voltage measuring ranges can be configured ↬ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474* .

The calculation of the resistor deviation must be performed via the bridge voltage by the PLC user program.

Connection of thermocouples

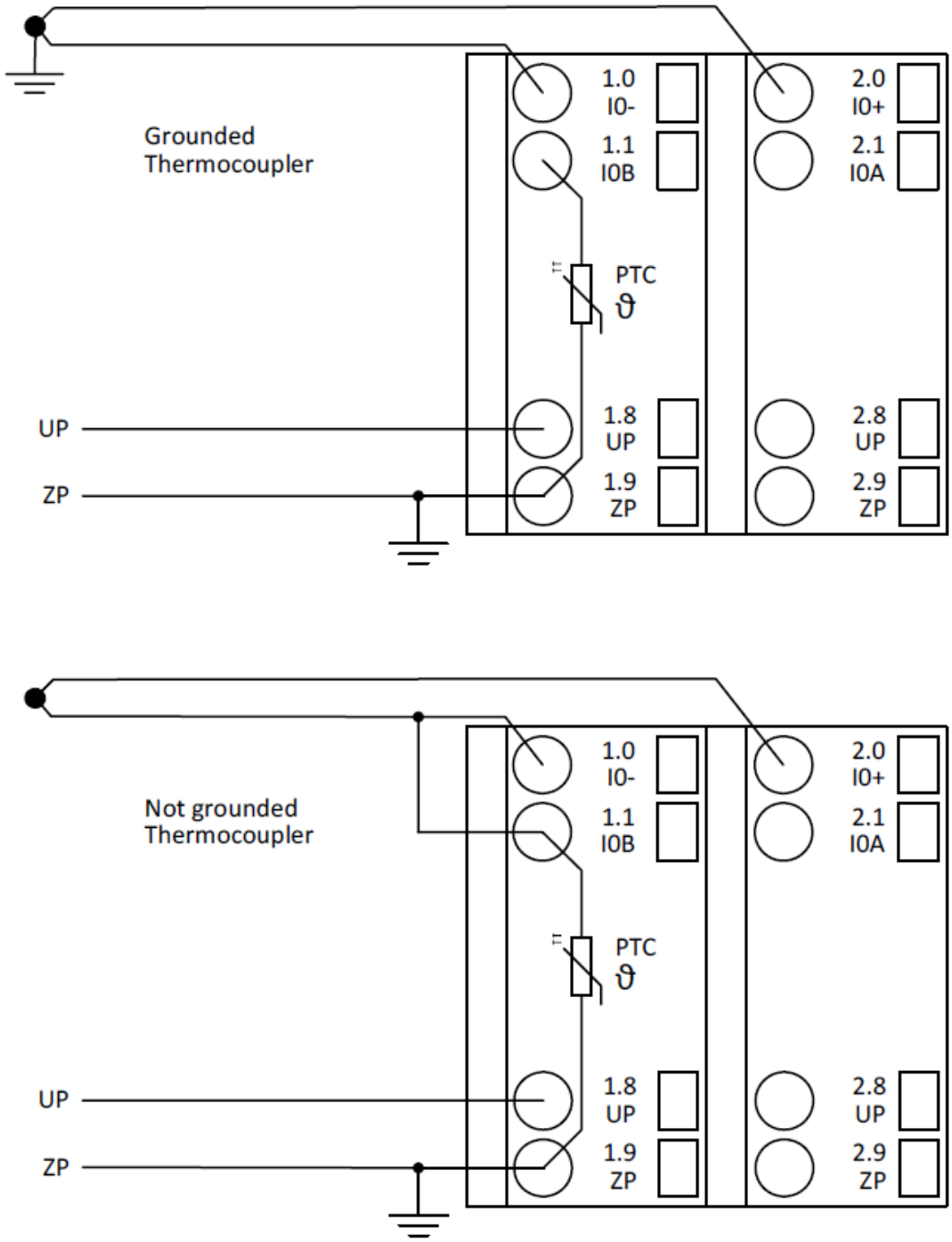


Fig. 870: Connection example

The following measuring ranges can be configured *Chapter 1.6.2.6.2.2.3.6 "Parameterization" on page 4474* :

J type	-210 °C...1200 °C	Fe-CuNi	1 channel used
K type	-270 °C...1372 °C	Ni-CrNi	1 channel used
N type	-270 °C...1300 °C	NiCrSi-NiSi	1 channel used
S type	-50 °C...1768 °C	Pt10Rh-Pt	1 channel used
T type	-270 °C...400 °C	Cu-CuNi	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / displays ↳ *Chapter 1.6.2.6.2.2.3.7 “Diagnosis” on page 4477*.

The module linearizes the thermocouple characteristics. It supports the following possibilities of temperature compensation and handling with cold junctions:

Internal compensation

An internal temperature sensor which is located next to the terminal unit is used to detect the temperature of the cold junction. So the compensating cables must be connected directly to the terminal unit, where the cold junction is located.

The setting "Internal compensation (default)" for the parameter "Compensation channel" should be selected.

To get more precise temperature measurements, the use of an external compensation method is recommended.

External compensation with temperature input

The temperature for the cold junction can be determinated externally.

A measured or known temperature value (e.g. ambient temperature in the cabinet) is transferred to the module via the output data word to all required channels. The possible temperature range is from -25 °C to +60 °C and is monitored by the AI531.

The setting "External with temperature value" for the parameter "Compensation channel" should be selected.

External compensation with compensation box

A compensation box balances the temperature difference between the cold junction and the reference temperature by generating a bridge voltage. The reference temperature is transferred via the output data word.

The compensation box must fit to the type of thermocouple and is located at the end of the compensating cables, where the cold junction is located. The cabling to the AI531 can be carried out with normal cables. The operating manual of the compensation box also has to be considered.

The setting "External with temperature value" for the parameter "Compensation channel" should be selected.

External compensation with flanking channel

A flanking channel of the same input group can be used for compensation, e. g. for channel 3, the channels 0, 1 and 2 can be selected as reference channels. The type of sensor for the reference channel can be selected in the parameters for the flanking channel. For example, a RTD sensor which is located next to the thermocouple terminal can be used as reference point for other channels.

The setting "Channel x" for the parameter "Compensation channel" should be selected. Refer to Channel configuration ↳ *Chapter 1.6.2.6.2.2.3.6 “Parameterization” on page 4474* for possible settings.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Analog inputs (words)	8
Analog outputs (words)	1

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

This means that replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, Type	Default	Min.	Max.	EDS Slot/ Index
Module ID	Internal	1535 [1])	Word	1535 0x05ff	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
Parameter length in bytes	Internal	36	Byte	36	0	255	0x0Y02
Check supply	Off On	0 1	Byte	On 0x01			0x0Y03
Analog data format	Default	0	Byte	Default 0x00			0x0Y04

[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1

[2]) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	39
Ext_User_Prm_Data_Const(0) =	0x05, 0xff, 0x24, \ 0x01, 0x00, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Input channel (8x)

No.	Name	Value	Internal value	Internal value, Type	Default	EDS Slot Index
1	Channel configuration	see *Table 443 "Channel configuration" on page 4475*	see *Table 443 "Channel configuration" on page 4475*	Byte	0 0x00	0x0Y07
2	Channel monitoring	see *Table 444 "Channel monitoring" on page 4477*	see *Table 444 "Channel monitoring" on page 4477*	Byte	0 0x03	
3	Line frequency suppression	see *Further information on page 4477*	see *Further information on page 4477*	Byte	0 0x00	
4	Compensation channel	see *Further information on page 4477*	see *Further information on page 4477*	Byte	0 0x00	

Table 443: Channel configuration

Internal value	Operating modes for the analog inputs, individually configurable
0	Unused (default)
2	Digital input
34	Analog input -50 mV...+50 mV
35	Analog input -500 mV...+500 mV
36	Analog input -1 V...+1 V
7	Analog input -5 V...+5 V
5	Analog input -10 V...+10 V
6	Analog input 0 V...+5 V

Internal value	Operating modes for the analog inputs, individually configurable
1	Analog input 0 V...+10 V
37	Analog input -20 mA...+20 mA
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
14	Analog input Pt100 (2-wire), -50 °C...+70 °C
15	Analog input Pt100 (3-wire), -50 °C...+70 °C
48	Analog input Pt100 (4-wire), -50 °C...+70 °C
57	Analog input Pt100 (2-wire), -50 °C...+70 °C (resolution: 0,01 K)
58	Analog input Pt100 (3-wire), -50 °C...+70 °C (resolution: 0,01 K)
59	Analog input Pt100 (4-wire), -50 °C...+70 °C (resolution: 0,01 K)
8	Analog input Pt100 (2-wire), -50 °C...+400 °C
9	Analog input Pt100 (3-wire), -50 °C...+400 °C
49	Analog input Pt100 (4-wire), -50 °C...+400 °C
45	Analog input Pt100 (2-wire), -200 °C...+850 °C
46	Analog input Pt100 (3-wire), -200 °C...+850 °C
47	Analog input Pt100 (4-wire), -200 °C...+850 °C
16	Analog input Pt1000 (2-wire), -50 °C...+400 °C
17	Analog input Pt1000 (3-wire), -50 °C...+400 °C
50	Analog input Pt1000 (4-wire), -50 °C...+400 °C
18	Analog input Ni1000 (2-wire), -50 °C...+150 °C
19	Analog input Ni1000 (3-wire), -50 °C...+150 °C
51	Analog input Ni1000 (4-wire), -50 °C...+150 °C
39	Analog input Cu50 1.426 (2-wire) -50 °C...+200 °C
40	Analog input Cu50 1.426 (3-wire) -50 °C...+200 °C
41	Analog input Cu50 1.426 (4-wire) -50 °C...+200 °C
42	Analog input Cu50 1.428 (2-wire) -200 °C...+200 °C
43	Analog input Cu50 1.428 (3-wire) -200 °C...+200 °C
44	Analog input Cu50 1.428 (4-wire) -200 °C...+200 °C
24	Analog input J-type thermocouple -210 °C...+1200 °C
25	Analog input K-type thermocouple -270 °C...+1372 °C
30	Analog input N-type thermocouple -270 °C...+1300 °C
27	Analog input S-type thermocouple -50 °C...+1768 °C
28	Analog input T-type thermocouple -270 °C...+400 °C
38	Analog input resistor 50 kΩ
52	Temperature-internal reference point
53	Common mode voltage

Table 444: Channel monitoring

Internal value	Monitoring
0	Plausibility, open-circuit (cut wire) and short circuit (default)
3	No monitoring

Table 445: Line frequency suppression

Internal value	Line frequency suppression
0	50 Hz
1	60 Hz
2	No line frequency suppression

Table 446: Compensation channel

Internal value	Compensation channel
0	Internal compensation (default)
1	Channel 0 (possible with channels 1, 2, 3)
2	Channel 1 (possible with channels 0, 2, 3)
3	Channel 2 (possible with channels 0, 1, 3)
4	Channel 3 (possible with channels 0, 1, 2)
5	Channel 4 (possible with channels 5, 6, 7)
6	Channel 5 (possible with channels 4, 6, 7)
7	Channel 6 (possible with channels 4, 5, 7)
8	Channel 7 (possible with channels 4, 5, 6)
9	External with temperature value

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module, e.g. internal analog voltage is not correct	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched OFF (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error							
4	14	1...10	1	0...7	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	0...7	1	Possibly wrong measured value caused by inadmissible temperature of the compensation channel	Check the temperature compensation channel
	11 / 12	ADR	1...10				

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
4	14	1...10	1	0...7	2	Invalid measured value of the channel caused by overly high voltage difference		Check voltage difference; install equalizing conductors if necessary
	11 / 12	ADR	1...10					
4	14	1...10	1	0...7	11	Output voltage 10 V faulty		Check output load
	11 / 12	ADR	1...10					

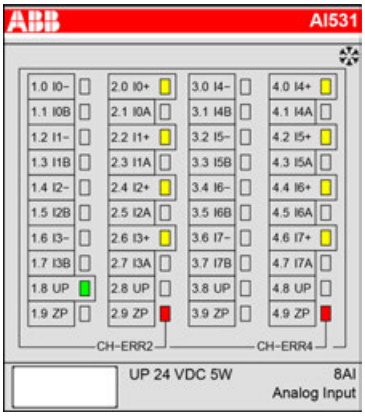
Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

States of the LEDs (see also section Diagnosis LEDs in the S500 system data):

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I3 and I4...I7	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, messages in groups (analog inputs combined into the groups 2 and 4)	Red	No error, or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs CH-ERR2 and CH-ERR4 light up together					

Measuring ranges

Voltage input ranges

Bipolar voltage input range, measuring bridge

The represented resolution corresponds to 16 bits.

Range	-50 ... +50 mV	-500 ... +500 mV	-1 ... +1 V	-5 ... +5 V	-10 ... +10 V	Common Mode Voltage	Digital value	
							Decimal	Hex.
Over-flow	> 58.7945	> 587.9449	> 1.17589	> 5.8794	> 11.7589	> 20.0000	32767	7FFF
Meas-ured value too high	58.7945 : 50.0018	587.9449 : 500.0181	1.17589 : 1.00004	5.8794 : 5.0002	11.7589 : 10.0004		32511 : 27649	7EFF : 6C01
Normal range Normal range or Meas-ured value too low	50.0000 : 0.0018	500.0000 : 0.0181	1.00000 : 0.00004	5.0000 : 0.0002	10.0000 : 0.0004	20.0000 : 0.0008	27648 : 1	6C00 : 0001
	0.0000	0.0000	0.0000	0.00000	0.0000	0.0000	0	0000
	-0.0018 : -50.0000	-0.0181 : -500.0000	-0.00004 : -1.00000	-0.0002 : -5.0000	-0.004 : -10.0000	-0.0008 : -20.0000	-1 : -27648	FFFF : 9400

Range	-50 ... +50 mV	-500 ... +500 mV	-1 ... +1 V	-5 ... +5 V	-10 ... +10 V	Common Mode Voltage	Digital value	
							Decimal	Hex.
Measured value too low	-50.0018 : -58.7945	-500.0181 : -587.9449	-1.00004 : -1.17589	-5.0002 : -5.8794	-10.0004 : -11.7589		-27649 : -32512	93FF : 8100
Underflow	< -58.7945	< -587.9449	< -1.17589	< -5.8794	< -11.7589	< -20.0000	-32768	8000

Unipolar voltage input range, measuring bridge, digital input

Range		0 ... +5 V	0 ... +10 V	Digital input	Digital value	
					Decimal	Hex.
Measured value too high		5.8794 : 5.0002	11.7589 : 10.0004		32511 : 27649	7EFF : 6C01
Normal range		5.0000 : 0.0002	10.0000 : 0.0004	ON	27648 : 1	6C00 : 0001
		0.0000	0.0000	OFF	0	0000
Measured value too low		-0.0002 : -0.8794	-0.0004 : -1.1759		-1 : -4864	FFFF : ED00
Underflow		< -0.8794	< -1.1759		-32768	8000

Current input ranges

Range	-20 ... +20 mA	0 ... +20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Overflow	> 23.5178	> 23.5178	> 22.8142	32767	7FFF
Measured value too high	23.5178 : 20.0007	23.5178 : 20.0007	22.8142 : 20.0006	32511 : 27649	7EFF : 6C01
Normal range	20.0000 : 0.0007	20.0000 : 0.0007	20.0000 : 4.0006	27648 : 1	6C00 : 0001
	0.0000	0.0000	4.0000	0	0000
	-0.0007 : -20.0000			-1 : -27648	FFFF : 9400

Range	-20 ... +20 mA	0 ... +20 mA	4 ... 20 mA	Digital value	
				Decimal	Hex.
Measured value too low		-0.0007 : -3.5178	3.9994 : 1.1852	-1 : -4864	FFFF : ED00
	-20.0007 : -23.5178			-27649 : -32512	93FF : 8100
Underflow	< -23.5178	< -3.5178	< 1.1852	-32768	8000

Resistance thermometer input ranges

The represented resolution corresponds to 16 bits.

Range	Pt100 -50 ... +70 °C [1]	Pt100 / Pt1000 -50 ... +400 °C	Pt100 -200 ... +850 °C	Ni1000 -50 ... +150 °C	Cu50 -200 ... +200 °C	Digital value	
						Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 850 °C	> 160.0 °C	> 200 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C				4500 : 4001	1194 : 0FA1
				160.0 °C : 150.1 °C		1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C					800 : 701	0320 : 02BD
Normal range	: : : : 70.0 °C : 0.1 °C	: 400.0 °C : : : : 0.1 °C	850.0 °C : : : : : 0.1 °C	: : : 150.0 °C : : 0.1 °C	: : 200.0 °C : : : 0.1 °C	8500 4000 2000 1500 700 : 1	2134 0FA0 07D0 05DC 02BC : 1
	0.0 °C	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : : -200 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C [2] -200.0 °C [2]	-1 : -500 -2000	FFFF : FE0C F830

Range	Pt100 -50 ... +70 °C [1])	Pt100 / Pt1000 -50 ... +400 °C	Pt100 -200 ... +850 °C	Ni1000 -50 ... +150 °C	Cu50 -200 ... +200 °C	Digital value	
						Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C		-50.1 °C : -60.0 °C		-501 : -600	FE0B : FDA8
Under-flow	< -60.0 °C	< -60.0 °C	< -200 °C	< -60.0 °C	< -200 °C [2])	-32768	8000

[1]) also possible with resolution 0.01 K

[2]) if Cu50 with 1.426, -50 °C is valid; if Cu50 with 1.428, -200.0 °C is valid

Resistor input range

The represented resolution corresponds to 16 bits.

Range	Resistor [Ω]	Digital value	
		Decimal	Hex.
Overflow	> 55000	32767	7FFF
Measured value too high	55000 : 50001	30413 : 27649	76CD : 6C01
Normal range	50000 : 2 1 0	27648 : 1 1 0	6C00 : 0001 0001 0000

Thermocouple input ranges

The represented resolution corresponds to 16 bits.

Range	Typ J -210 ... +1200 °C	Typ K -270 ... +1372 °C	Typ N -270 ... +1300 °C	Typ S -50 ... +1768 °C	Typ T -270 ... +400 °C	Digital value	
						Decimal	Hex.
Overflow	> 1200.0 °C	> 1372.0 °C	> 1300.0 °C	> 1768.0 °C	> 400.0 °C	32767	7FFF
Normal range				1768.0 °C		17680	4510
		1372.0 °C		:		13720	3598
		:	1300.0 °C	:		13000	32C8
	1200.0 °C	:	:	:		12000	2EE0
	:	:	:	:	400.0 °C	4000	0FA0
	:	:	:	:	:	:	:

Range	Typ J -210 ... +1200 °C	Typ K -270 ... +1372 °C	Typ N -270 ... +1300 °C	Typ S -50 ... +1768 °C	Typ T -270 ... +400 °C	Digital value	
						Decimal	Hex.
	0.1 °C	0.1 °C	0.1 °C	0.1 °C	0.1 °C	1	1
	0.0 °C	0.0 °C	0.0 °C	0.0 °C		0	0000
	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-0.1 °C	-1	FFFF
	:	:	:	:	:	:	:
	:	:	:	-50.0 °C	:	-500	FE0C
	-210.0 °C	:	:	:	:	-2100	F7CC
		-270.0 °C	-270.0 °C		-270.0 °C	-2700	F574
Under-flow	< -210.0 °C	< -270.0 °C	< -270.0 °C	< -50.0 °C	< -270.0 °C	-32768	8000

Temperature-internal reference point ranges

Range	Value	Digital value	
		Decimal	Hex.
Overflow	> +85 °C	32767	7FFF
Normal range	+85 °C	850	0352
	0 °C	0	0000
	-40 °C	-400	FE70
Underflow	< -40 °C	-32768	8000

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		

Parameter		Value
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	Current consumption from UP in normal operation	130 mA
	Inrush current from UP (at power up)	0.056 A^2s
Max. length of analog cables, conductor cross section > 0.14 mm^2		100 m
Weight		130 g
Mounting position		Horizontal or vertical with derating (max. temperature 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value
Number of channels per module	8
Distribution of channels into groups	2 groups of 4 channels each
Connections of the channels I0 to I3	Terminals 1.0 to 1.7 and terminals 2.0 to 2.7
Connections of the channels I4 to I7	Terminals 3.0 to 3.7 and terminals 4.0 to 4.7
Input type	Bipolar (not with current or Pt100/ Pt1000/ Ni1000/ Cu50/ resistor)
Galvanic isolation	Against internal supply and other modules
Common mode input range	±20 V DC plus signal voltage
Configurability	Digital input, -50 mV...+50 mV, -500mV...+500 mV, -1 V...+1 V, -5 V...+5 V, -10 V...+10 V, 0 V...+5 V, 0 V...+10 V, -20 mA...+20 mA, 0 mA...20 mA, 4 mA...20 mA, Pt100, Pt1000, Ni1000, Cu50, resistor, thermocouple types J, K, N, S, T (each input can be configured individually)
Channel input resistance	Voltage: > 100 kΩ, current: ca. 330 Ω
Time constant of the input filter	Line-frequency suppression 50 Hz, 60 Hz, none
Indication of the input signals	1 yellow LED per channel, the brightness depends on the value of the analog signal

Parameter		Value		
Conversion time		1 ms (none), 100 ms (50 Hz / 60 Hz) per channel		
Resolution		Range	unipolar	15 bits
			bipolar	15 bits + sign
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ.	±0.1 % (voltage) ±0.3 % (current, resistor) at 25 °C	
		Max.	±0.7 % (voltage) ±0.9 % (current, resistor) ±0.5 % (thermocouple type J, N, S, T; thermocouple type K > -220 °C) 1.0 K (resistance temperature detectors) at 0 °C...60 °C or EMC disturbance	
Maximum permanent allowed overload (no damage)				
	Current input	When the input current exceeds the overflow value of the measurement range, the input impedance is switched to high impedance for protection. The maximum allowed overload is then 30 V. The digital value corresponds to the overflow value. Periodically, the input impedance is switched to the normal value and the input current is measured. If the input current is within the measurement range, the input impedance remains at the normal level and the digital value corresponds to the measured current.		
	Voltage input	30 V		
Relationship between input signal and hex code		*Table 444 “Channel monitoring” on page 4477*		
Unused voltage inputs		Are configured as "unused"		
Unused current inputs		Have a low resistance, can be left open-circuited		
Overvoltage protection		Yes		

Technical data of the analog inputs if used as digital inputs

Parameter	Value
Number of channels per module	Max. 8
Distribution of channels into groups	2 groups of 4 channels each
Connections of the channels I0+ to I3+	Terminals 2.0, 2.2, 2.4, 2.6
Connections of the channels I4+ to I7+	Terminals 4.0, 4.2, 4.4, 4.6
Reference potential for the inputs	Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)

Parameter		Value
Input delay		Typ. 2 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	Typ. 1 mA
	Input voltage +15 V	Typ. 3.1 mA
	Input voltage +30 V	< 7 mA
Input resistance		Ca. 4.8 kΩ

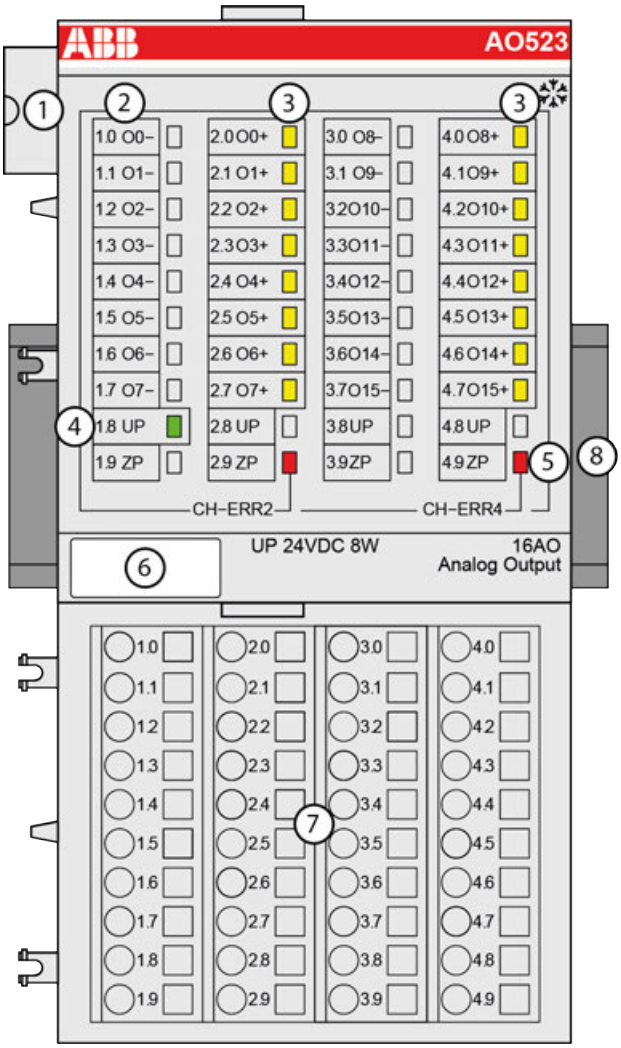
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 600 R0001	AI531, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires	Active
1SAP 450 600 R0001	AI531-XC, analog input module, 8 AI, U/I/Pt100, TC, 15 bits + sign, 4-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AO523 - Analog output module

- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states at the analog outputs (O0 - O15)
4 1 green LED to display the state of the process supply voltage UP
5 2 red LEDs to display errors
6 Label
7 Terminal unit
8 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

- 16 analog outputs in two groups:
  - 8 channels configurable for voltage or currrent output (O0...O3 / O8...O11)
  - 8 channels for voltage output (O4...O7 / O12...O15)

  Resolution 12 bits plus sign

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
LED displays		19 LEDs for signals and error messages
Internal power supply		Through the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The modules are plugged on an I/O terminal unit *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, independent of the inserted module:

Terminals 1.8 to 4.8: process voltage UP = +24 V DC

Terminals 1.9 to 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	O0- to O7-	Negative poles of the first 8 analog outputs
2.0 to 2.7	O0+ to O7+	Positive poles of the first 8 analog outputs
3.0 to 3.7	O8- to O15-	Negative poles of the following 8 analog outputs
4.0 to 4.7	O8+ to O15+	Positive poles of the following 8 analog outputs

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per AO523.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the module:

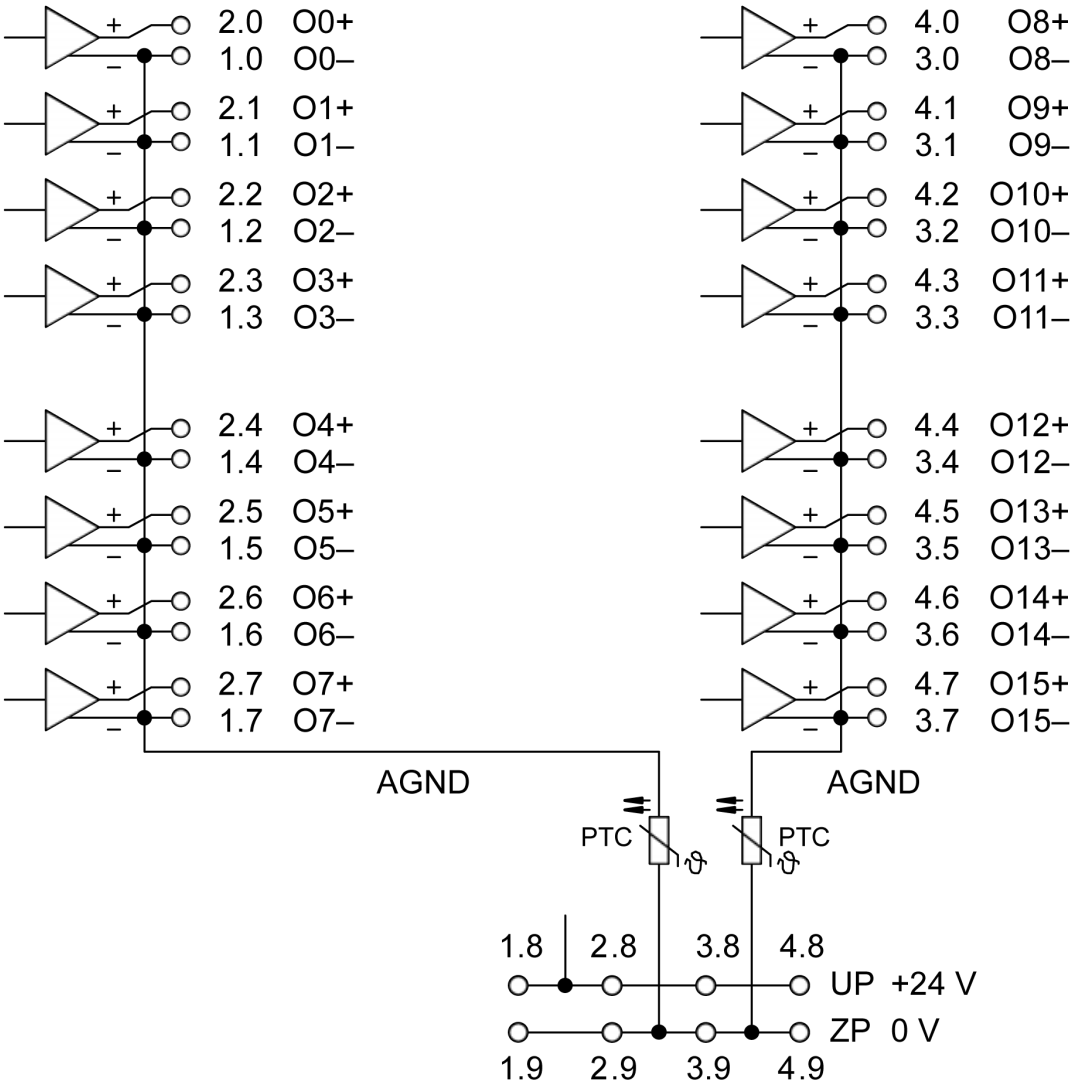


Fig. 871: 16 analog outputs in two groups *Chapter 1.6.2.6.2.2.4.2 “Functionality” on page 4488*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The modules provide several diagnosis functions *Chapter 1.6.2.6.2.2.4.7 “Diagnosis” on page 4497*.

Connection of analog output loads (Voltage, current)

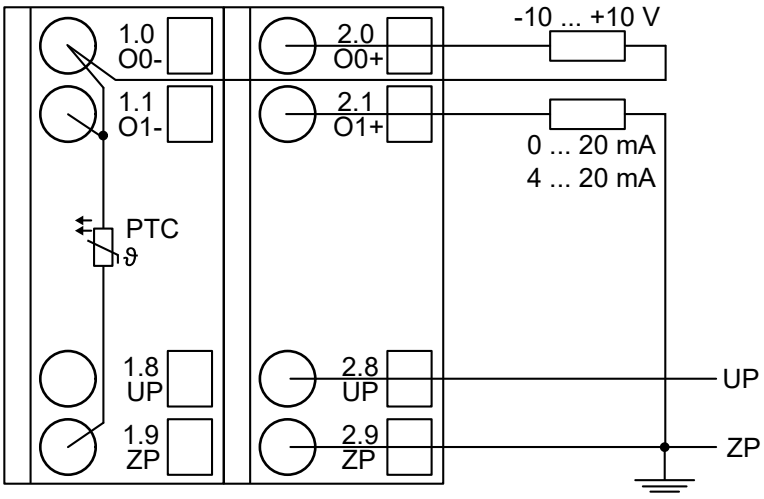


Fig. 872: Connection example

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.2.2.4.6 “Parameterization” on page 4493*:

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 and 8...11 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

The function of the LEDs is described under Displays.

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	0
Counter output data (words)	16

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

That means replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1510 [1])	Word	1510 0x05e6	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length in bytes	Internal	39	Byte	39-CPU 39-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
7	Channel configuration Output channel 0	See *Table 447 "Channel configuration [3])" on page 4496*		Byte	Default 0x00	0	130	0x0Y06
8	Channel monitoring Output channel 0	See *Table 448 "Channel monitoring [4])" on page 4496*		Byte	Default 0x00	0	3	0x0Y07

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
9	Substitute value Output channel 0	Output channel 0!	0...0xffff	Word	Default 0x0000	0	65535	0x0Y08
10 to 15	Channel configuration and channel monitoring of the output channels 1 to 3	See *Table 447 “Channel configuration 3)” on page 4496* and *Table 448 “Channel monitoring 4)” on page 4496*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y09 to 0x0Y0E
16 to 23	Channel configuration and channel monitoring of the output channels 4 to 7	See *Table 447 “Channel configuration 3)” on page 4496* and *Table 448 “Channel monitoring 4)” on page 4496*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y0F to 0x0Y16
24	Channel configuration Output channel 8	See *Table 447 “Channel configuration 3)” on page 4496*		Byte	Default 0x00	0	130	0x0Y17
25	Channel monitoring Output channel 8	See *Table 448 “Channel monitoring 4)” on page 4496*		Byte	Default 0x00	0	3	0x0Y18
26	Substitute value Output channel 8	Output channel 8!	0...0xffff	Word	Default 0x0000	0	65535	0x0Y19

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
27 to 32	Channel configuration and channel monitoring of the output channels 9 to 11	See *Table 447 “Channel configuration 3)” on page 4496* and *Table 448 “Channel monitoring 4)” on page 4496*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y1A to 0x0Y1F
33 to 40	Channel configuration and channel monitoring of the output channels 12 to 15	See *Table 447 “Channel configuration 3)” on page 4496* and *Table 448 “Channel monitoring 4)” on page 4496*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y20 to 0x0Y27
1) With CS31 and addresses less than 70 and FBP, the value is increased by 1 2) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len =	42
Ext_User_Prm_Data_Const(0) =	0x05, 0xe7, 0x27, \ 0x01, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Output channels 0 and 8 (2 channels, AO523)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see below ↳ *Table 447 “ Channel configuration [3])” on page 4496*	see below ↳ *Table 447 “ Channel configuration [3])” on page 4496*	Byte	see below ↳ *Table 447 “ Channel configuration [3])” on page 4496*
2	Channel monitoring	see below ↳ *Table 448 “ Channel monitoring [4])” on page 4496*	see below ↳ *Table 448 “ Channel monitoring [4])” on page 4496* *8)	Byte	see below ↳ *Table 448 “ Channel monitoring [4])” on page 4496*
3	Substitute value ↳ *Table 449 “ Substitute value” on page 4497*	0...65535	0... 0xffff	Word	0

Output channels 1...7 and 9...15 (14 channels, AO523)

No.	Name	Internal value, type
1	Channel configuration see table [3])	Byte
2	Channel monitoring see table [4])	Byte

Table 447: Channel configuration [3])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 448: Channel monitoring [4])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit (default)
1	Open-circuit (broken wire) and short circuit
2	Plausibility
3	No monitoring

Table 449: Substitute value

Intended behavior of channel 0 when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	OFF	0
Last value	Last value	0
Substitute value	OFF or Last value	1...65535

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON -> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
4	14	1...10	3	0...15	48	Analog value overflow at an analog output		Check output value

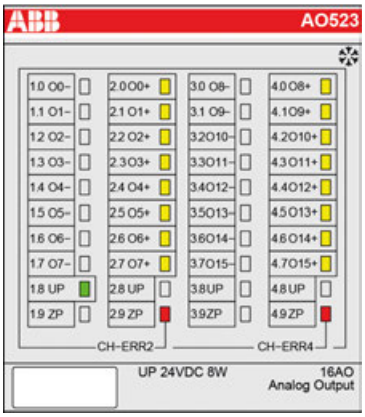
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
	11 / 12	ADR	1...10					
4	14	1...10	3	0...15	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Outputs O0...O7 and O8...O15	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Output ranges

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589 V	> 23.5178 mA	> 22.8142 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	Current consumption from UP at normal operation	0.15 A + output loads
	Inrush current from UP (at power up)	0.040 A^2s
Max. length of analog cables, conductor cross section > 0.14 mm^2		100 m
Weight		300 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog outputs

Parameter		Value	
Number of channels per module		16, of which channnels O0...O3 and O8...O11 for voltage and current, and channels O4...7 and O12...15 only for voltage	
Distribution of channels into groups		2 groups of 8 channels each	
	Channels O0-...O7-	Terminals 1.0...1.7	
	Channels O0+...O7+	Terminals 2.0...2.7	
	Channels O8-...O15-	Terminals 3.0...3.7	
	Channels O8+...O15+	Terminals 4.0...4.7	
Output type		Bipolar with voltage, unipolar with current	
Galvanic isolation		Against internal supply and other modules	
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3 and 8...11	
Output resistance (load), as current output		0 Ω...500 Ω	
Output loadability, as voltage output		Max. ±10 mA	
Indication of the output signals		One LED per channel	
Resolution		12 bits (+ sign)	
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C	
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance	
Relationship between output signal and hex code		*Chapter 1.6.2.6.2.2.4.9 “Output ranges” on page 4499*	
Unused outputs		Can be left open-circuited	

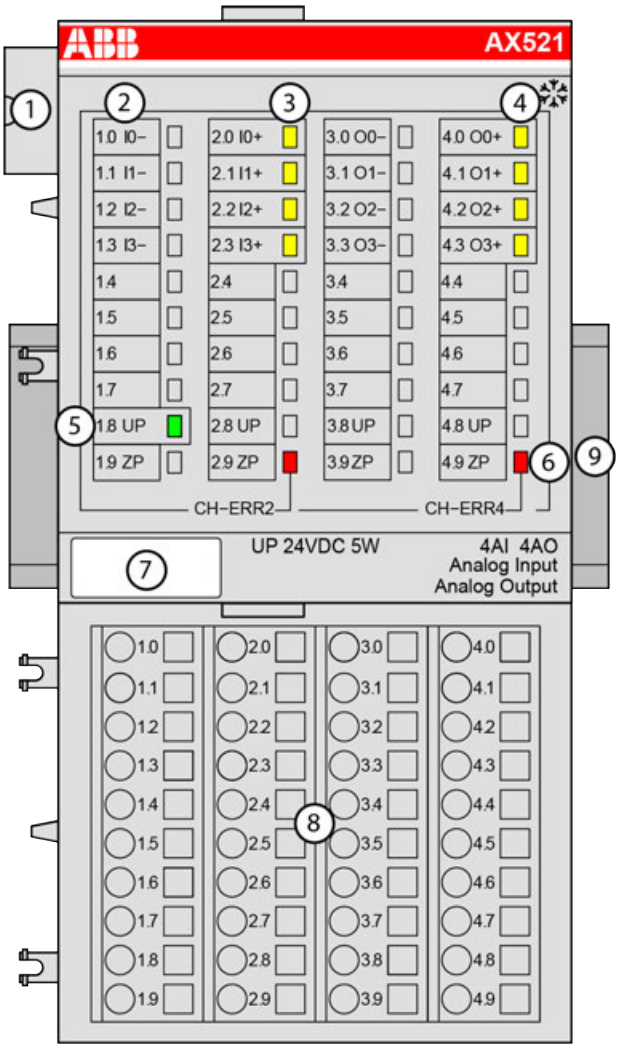
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 200 R0001	AO523, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires	Active
1SAP 450 200 R0001	AO523-XC, analog output module, 16 AO, U/I, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX521 - Analog input/output module

- 4 configurable analog inputs (I0 to I3) in 1 group (1.0...2.3)
  Resolution 12 bits plus sign
- 4 configurable analog outputs (O0 to O3) in 1 group (3.0...4.3)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 4 yellow LEDs to display the signal states at the analog inputs (I0 - I3)
4 4 yellow LEDs to display the signal states at the analog outputs (O0 - O3)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

AX521

4 analog inputs (I0...I3), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

Parameter		Value
Resolution of the analog channels		
	Voltage -10 V... +10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		11 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The modules are plugged on an I/O terminal unit ↳ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9, 3.9 and 4.9 are electrically interconnected within the I/O terminal units and have always the same assignment, irrespective of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.3	I0- to I3-	Negative poles of the 4 analog inputs
2.0 to 2.3	I0+ to I3+	Positive poles of the 4 analog inputs
3.0 to 3.3	O0- to O3-	Negative poles of the 4 analog outputs
4.0 to 4.3	O0+ to O3+	Positive poles of the 4 analog outputs

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.

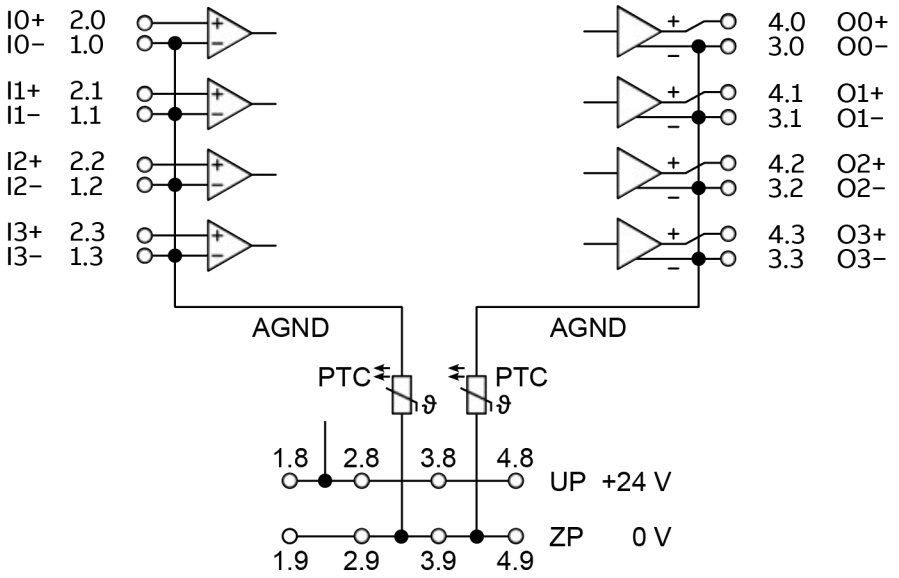


Fig. 873: 4 analog inputs and 4 analog outputs, individually configurable ↬ *Chapter 1.6.2.6.2.2.5.2 “Functionality” on page 4503*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.

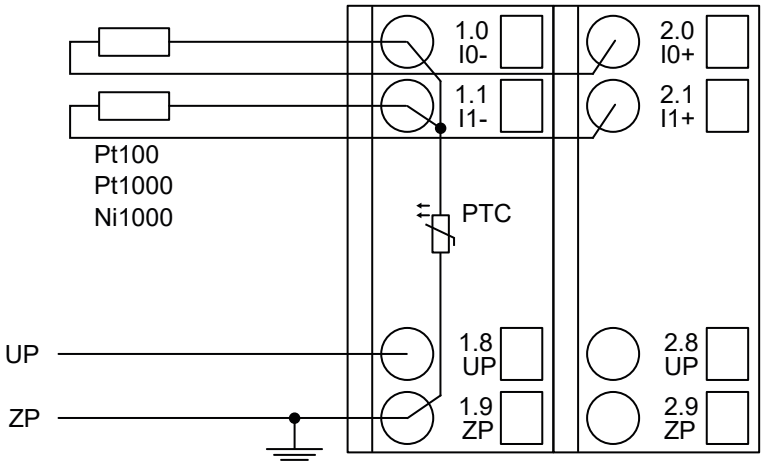


Fig. 874: Connection example

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

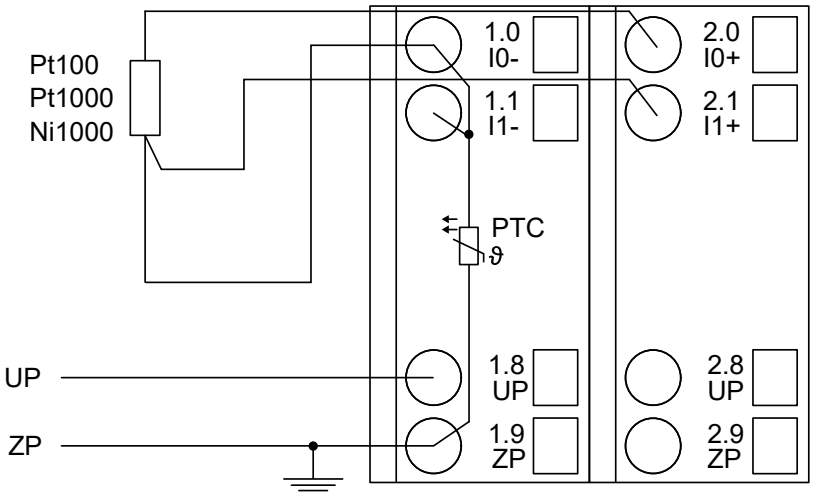


Fig. 875: Connection example

If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

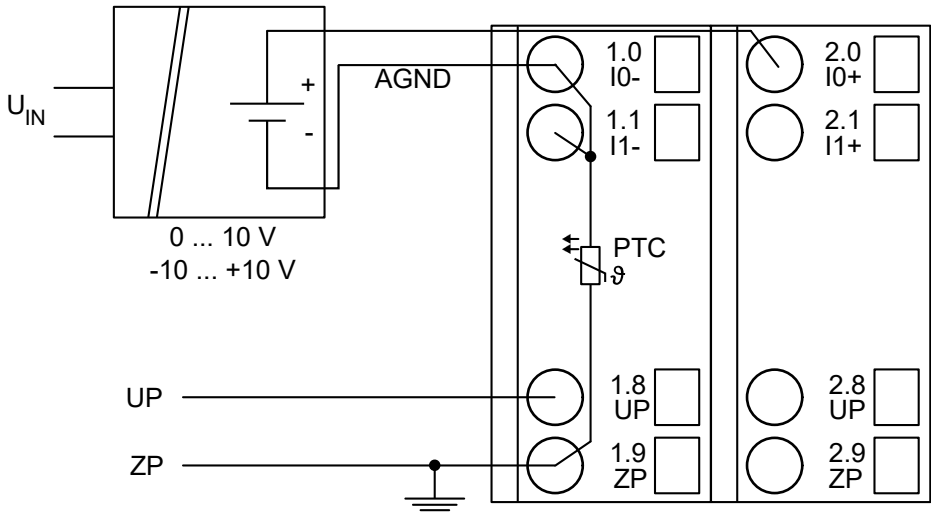


Fig. 876: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured for AX521 *Chapter 1.6.2.6.2.2.5.6 “Parameterization” on page 4513* and for AX522 *Chapter 1.6.2.6.2.2.6.6 “Parameterization” on page 4538*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

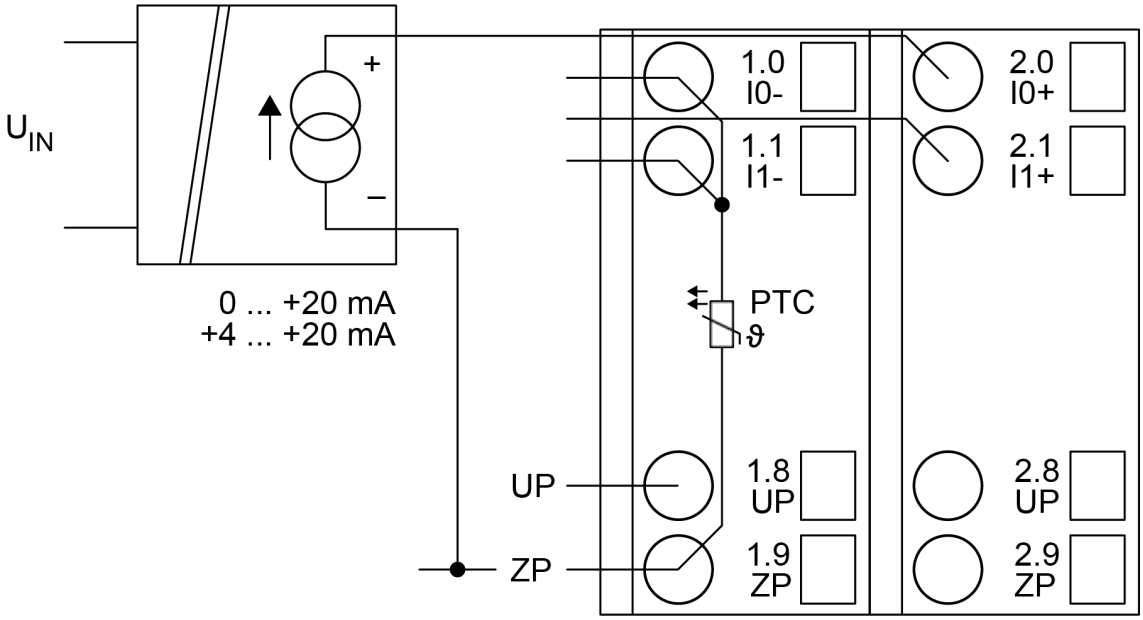


Fig. 877: Connection example

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

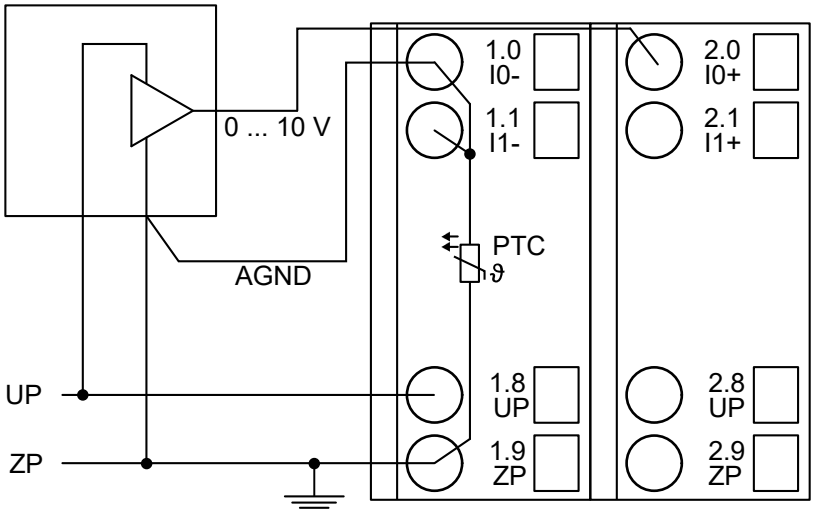


Fig. 878: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

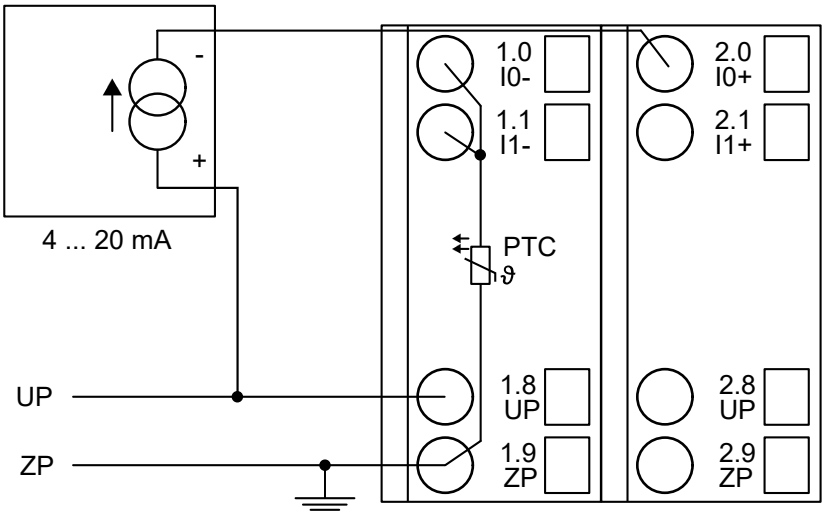


Fig. 879: Connection example

Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.

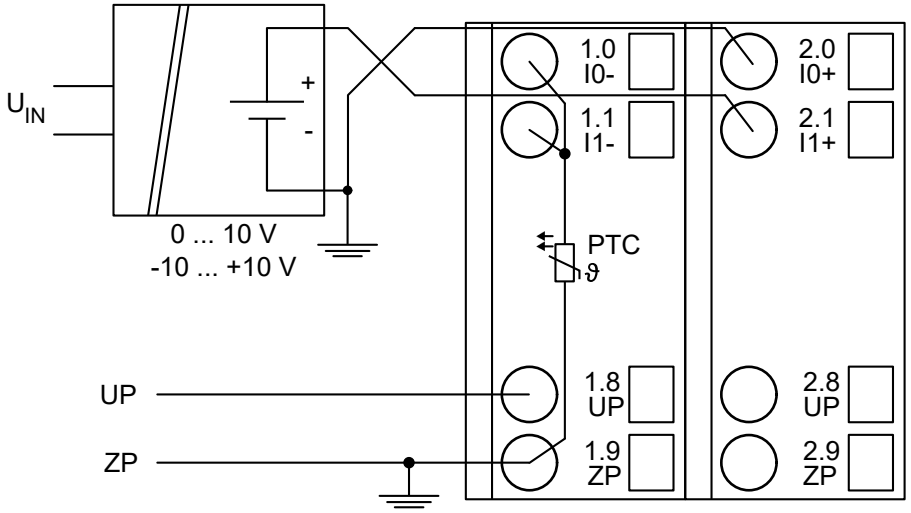


Fig. 880: Connection example

The negative pole of the sensor must be grounded next to the sensor.

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

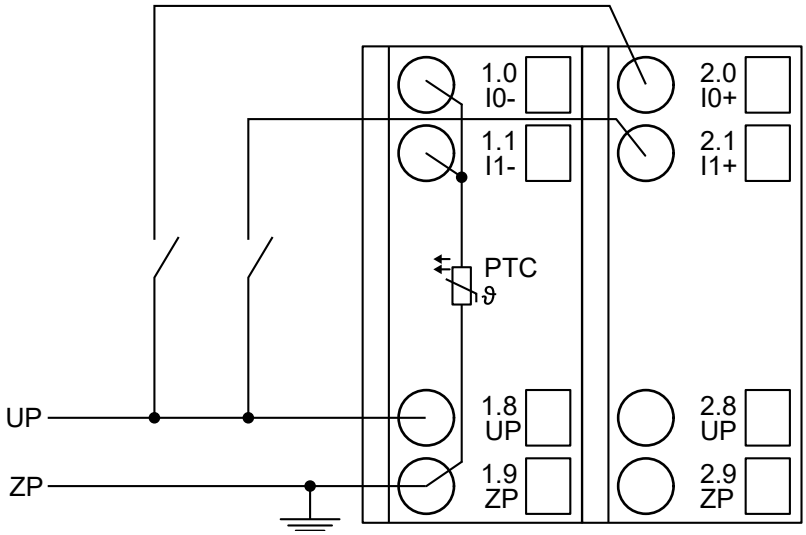


Fig. 881: Connection example

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)

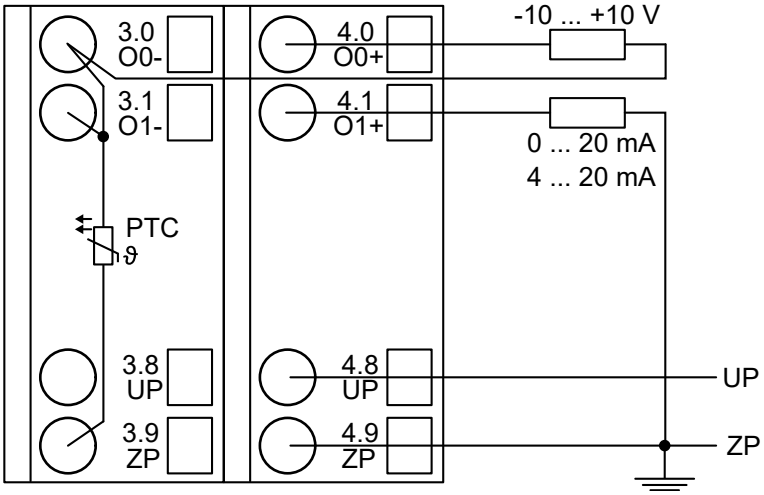


Fig. 882: Connection example

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	4
Counter output data (words)	4

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1505 [1])	Word	1505 0x05E1	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			Not for FBP
3	Parameter length in bytes	Internal	21	Byte	21-CPU 21-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
7	Channel configuration Input channel 0	See table *Table 451 “Channel configuration [2])” on page 4515*		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	See table *Table 452 “Channel monitoring [3])” on page 4516*		Byte	Default 0x00	0	3	0x0Y07
9 to 14	Channel configuration and channel monitoring of the input channels 1 to 3	See tables *Table 451 “Channel configuration [2])” on page 4515* and *Table 452 “Channel monitoring [3])” on page 4516*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y0D
15	Channel configuration Output channel 0	See table *Table 451 “Channel configuration [2])” on page 4515*		Byte	Default 0x00	0	130	0x0Y0E
16	Channel monitoring Output channel 0	See table *Table 452 “Channel monitoring [3])” on page 4516*		Byte	Default 0x00	0	3	0x0Y0F
17	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y10
18 to 21	Channel configuration and channel monitoring of the output channels 1 to 2	See tables *Table 451 “Channel configuration [2])” on page 4515* and *Table 452 “Channel monitoring [3])” on page 4516*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y11 to 0x0Y14

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
22	Channel configu-ration Output channel 3	See table ↳ *Table 451 “Chan nel configuration 2)” on page 4515*		Byte	Default 0x00	0	130	0x0Y15
23	Channel moni-toring Output channel 3	See table ↳ *Table 452 “Chan nel monitoring 3)” on page 4516*		Byte	Default 0x00	0	3	0x0Y16
1) With CS31 and addresses less than 70 and FBP, the value is increased by 1 2) Not with FBP								

GSD file:

Ext_User_Prm_Data_Len = Ext_User_Prm_Data_Const(0) =	24 0x05, 0xe2, 0x15, \ 0x01, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 450: Input channel (4x)

No.	Name	Internal value, type	Default
1	Channel configuration see table 2)	Byte	0 0x00 see table 2)
2	Channel monitoring see table 3)	Byte	0 0x00 see table 3)

Table 451: Channel configuration 2)

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)

Internal value	Operating modes of the analog inputs, individually configurable
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 452: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
3	No monitoring

Table 453: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 454: Output channels 1...3 (3x)

No.	Name	Internal value, type
1	Channel configuration see table [4])	Byte
2	Channel monitoring see table [6])	Byte

Table 455: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V

Internal value	Operating modes of the analog outputs, individually configurable
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 456: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
3	No monitoring

Table 457: Substitute value [6])

Intended behaviour of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

Table 458: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value underflow	Input value overflow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 459: Content of diagnosis messages

E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31		19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		3	Timeout in the I/O module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		40	Different hard-/firm-ware versions in the module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		36	Internal data exchange failure	Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31		9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31		26	Parameter error	Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31		11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31		45	Process voltage is switched off (ON –> OFF)	Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					

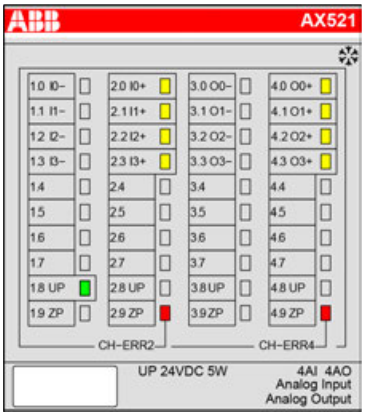
E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I3	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	Outputs O0...O3	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 . . 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 . . 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
		0.0000	0	4	OFF	0	0000
	0.0000	-0.0004 : :		3.9994		-1 -4864 -6912	FFFF ED00 E500

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 : -1.7593	: -10.0000				: -27648	: 9400
Meas-ured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<-1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation	0.15 A + output loads
Inrush current from UP (at power up)		0.020 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	4	
Distribution of channels into groups	1 group of 4 channels	
Connections of the channels I0- to I3-	Terminals 1.0 to 1.3	
Connections of the channels I0+ to I3+	Terminals 2.0 to 2.3	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs Current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between input signal and hex code	See tables *Chapter 1.6.2.6.2.2.5.9.1 “Input ranges of voltage, current and digital input” on page 4520*	

Parameter	Value
Unused voltage inputs	Are configured as "unused"
Unused current inputs	Have a low resistance, can be left open-circuited
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels I0+ to I3+		Terminals 2.0 to 2.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		4, all channels for voltage and current
Distribution of channels into groups		1 group of 4 channels
	Channels O0-...O3-	Terminals 3.0...3.3
	Channels O0+...O3+	Terminals 4.0...4.3
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table *Chapter 1.6.2.6.2.2.5.9.3 "Output ranges voltage and current" on page 4521*	
Unused outputs	Can be left open-circuited	

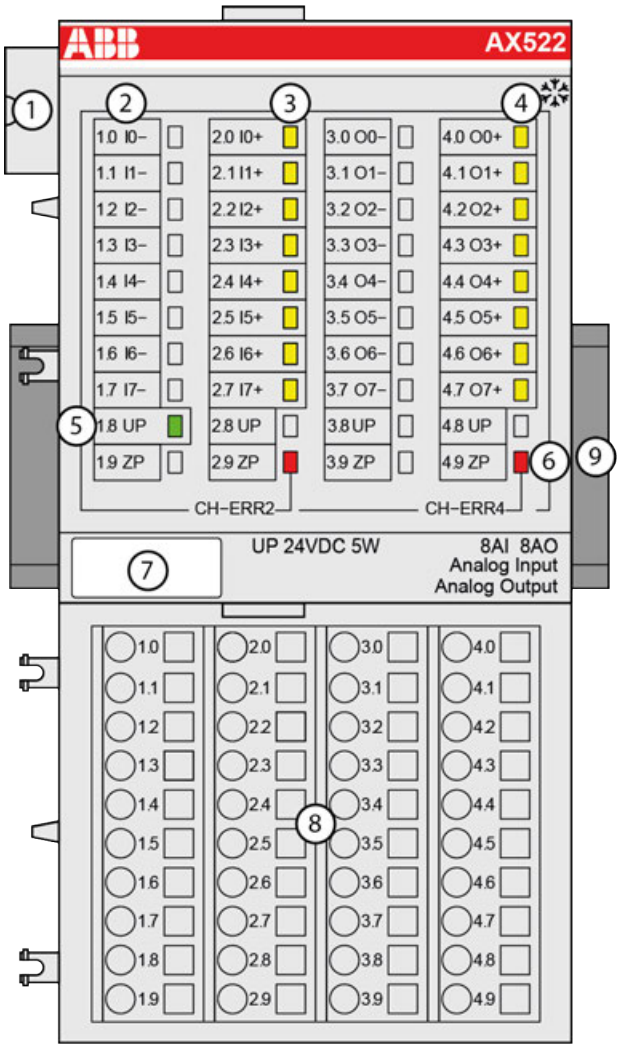
Ordering Data

Part no.	Description	Product life cycle phase *)
1SAP 250 100 R0001	AX521, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 100 R0001	AX521-XC, analog input/output module, 4 AI, 4 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

AX522 - Analog input/output module

- 8 configurable analog inputs (I0 to I7) in 1 group (1.0...2.7)
  Resolution 12 bits plus sign
- 8 configurable analog outputs (O0 to O7) in 1 group (3.0...4.7)
  Resolution 12 bits plus sign
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states at the analog inputs (I0 - I7)
4 8 yellow LEDs to display the signal states at the analog outputs (O0 - O7)
5 1 green LED to display the state of the process supply voltage UP
6 2 red LEDs to display errors
7 Label
8 Terminal unit
9 DIN rail
Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

8 analog inputs (I0...I7), individually configurable for

- Unused (default setting)
- 0 V...10 V
- -10 V...+10 V
- 0 mA...20 mA

- 4 mA...20 mA
- Pt100, -50 °C...+400 °C (2-wire)
- Pt100, -50 °C...+400 °C (3-wire), requires 2 channels
- Pt100, -50 °C...+70 °C (2-wire)
- Pt100, -50 °C...+70 °C (3-wire), requires 2 channels
- Pt1000, -50 °C...+400 °C (2-wire)
- Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels
- Ni1000, -50 °C...+150 °C (2-wire)
- Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels
- 0 V...10 V with differential inputs, requires 2 channels
- -10 V...+10 V with differential inputs, requires 2 channels
- Digital signals (digital input)

4 analog outputs (O0...O3), individually configurable for

- Unused (default setting)
- -10 V...+10 V
- 0 mA...20 mA
- 4 mA...20 mA

4 analog outputs (O4...O7), individually configurable for

- Unused (default setting)
- -10 V...+10 V

Parameter		**Value**
Resolution of the analog channels		
	Voltage -10 V...+10 V	12 bits plus sign
	Voltage 0 V...10 V	12 bits
	Current 0 mA...20 mA, 4 mA...20 mA	12 bits
	Temperature	0.1 °C
LED displays		19 LEDs for signals and error messages
Internal power supply		Via the I/O bus interface (I/O bus)
External power supply		Via the terminals ZP and UP (process voltage 24 V DC)
Required terminal unit		TU515 or TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The modules are plugged on an I/O terminal unit ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*. Properly position the modules and press until they lock in place. The terminal units are mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8, 2.8, 3.8 and 4.8 as well as 1.9, 2.9,3.9 and 4.9 are electrically interconnected within the I/O terminal units and always have the same assignment, independent of the inserted module:

Terminals 1.8, 2.8, 3.8 and 4.8: process voltage UP = +24 V DC

Terminals 1.9, 2.9, 3.9 and 4.9: process voltage ZP = 0 V DC

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	I0- to I7-	Negative poles of the 8 analog inputs
2.0 to 2.7	I0+ to I7+	Positive poles of the 8 analog inputs
3.0 to 3.7	O0- to O7-	Negative poles of the 8 analog outputs
4.0 to 4.7	O0+ to O7+	Positive poles of the 8 analog outputs

The negative poles of the analog inputs are connected to each other to form an "Analog Ground" signal for the module.

The negative poles of the analog outputs are connected to each other to form an "Analog Ground" signal for the module.

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per I/O module.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figure shows the connection of the I/O module.

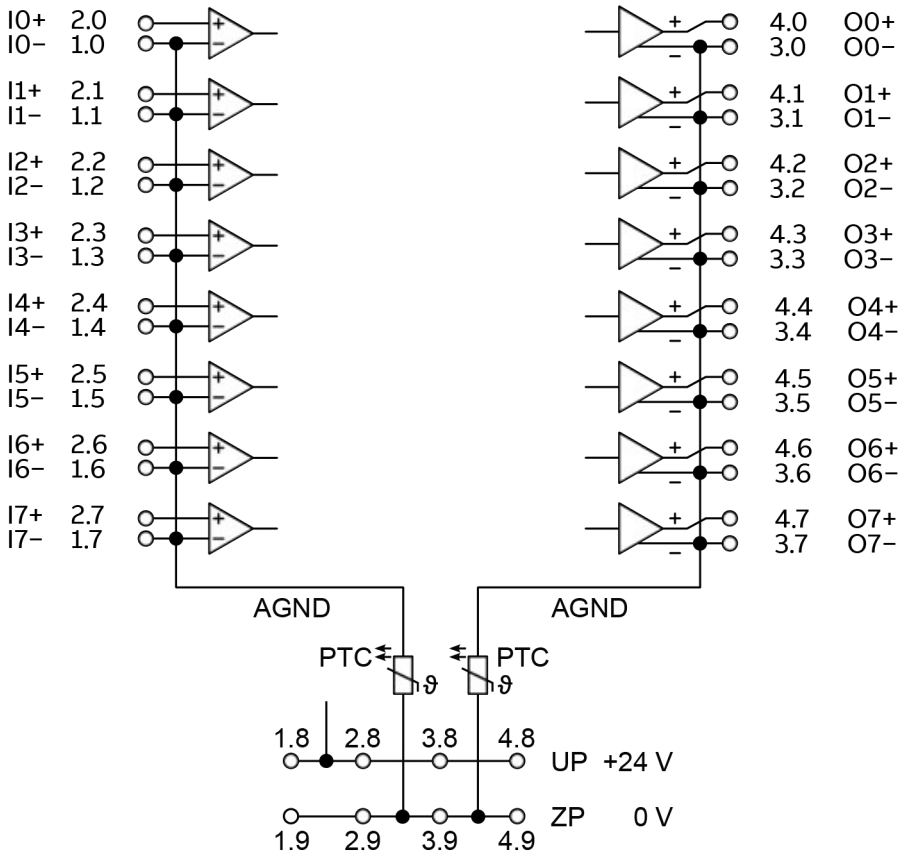


Fig. 883: 8 analog inputs and 8 analog outputs, individually configurable ↬ *Chapter 1.6.2.6.2.2.6.2 “Functionality” on page 4526*

CAUTION!
By installing equipotential bonding conductors between the different parts of the system, it must be ensured that the potential difference between ZP and AGND never can exceed 1 V.

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the 8 analog channels.

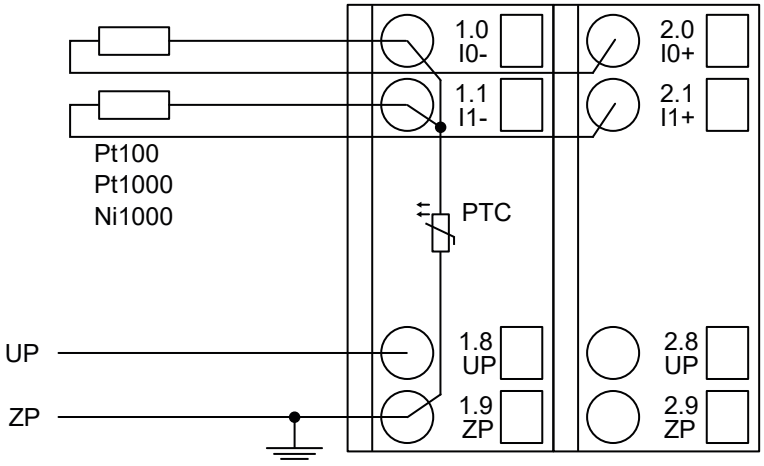


Fig. 884: Connection example

Pt100	-50 °C...+70 °C	2-wire configuration, one channel used
Pt100	-50 °C...+400 °C	2-wire configuration, one channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, one channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, one channel used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the I/O module provides a constant current source which is multiplexed over the max. 8 (depending on the configuration) analog channels.

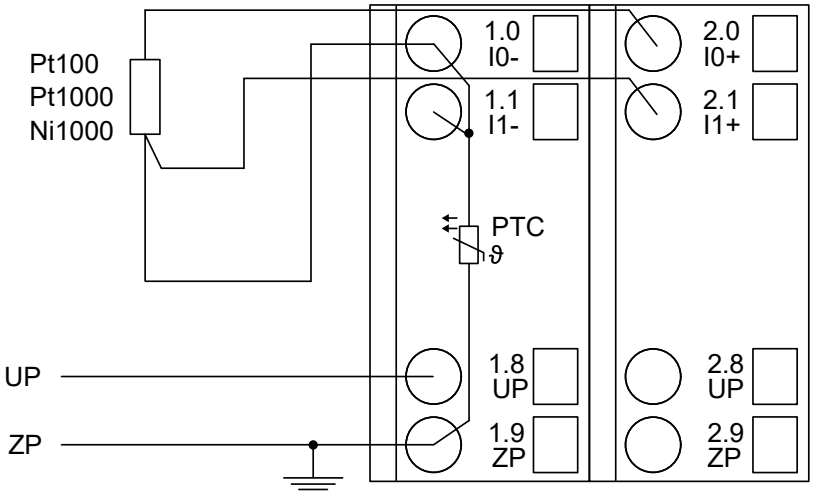


Fig. 885: Connection example

If several measuring points are adjacent to each other, only one return line is necessary. This saves wiring costs.

With the 3-wire configuration, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e.g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+70 °C	3-wire configuration, two channels used
Pt100	-50 °C...+400 °C	3-wire configuration, two channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, two channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, two channels used

The I/O module performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

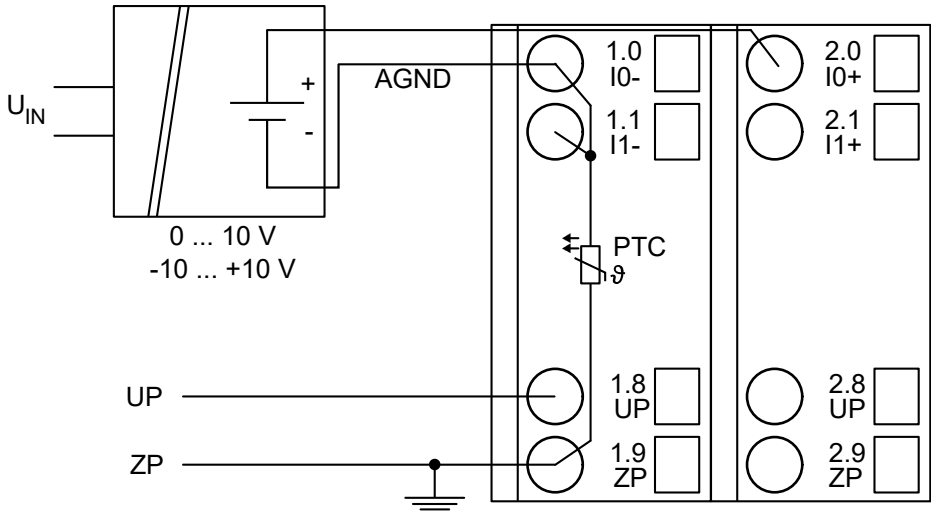


Fig. 886: Connection example

By connecting the sensor's negative pole of the output voltage to AGND, the galvanically isolated voltage source of the sensor is referred to ZP.

The following measuring ranges can be configured for AX521 ↳ *Chapter 1.6.2.6.2.2.5.6 “Parameterization” on page 4513* and for AX522 ↳ *Chapter 1.6.2.6.2.2.6.6 “Parameterization” on page 4538*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

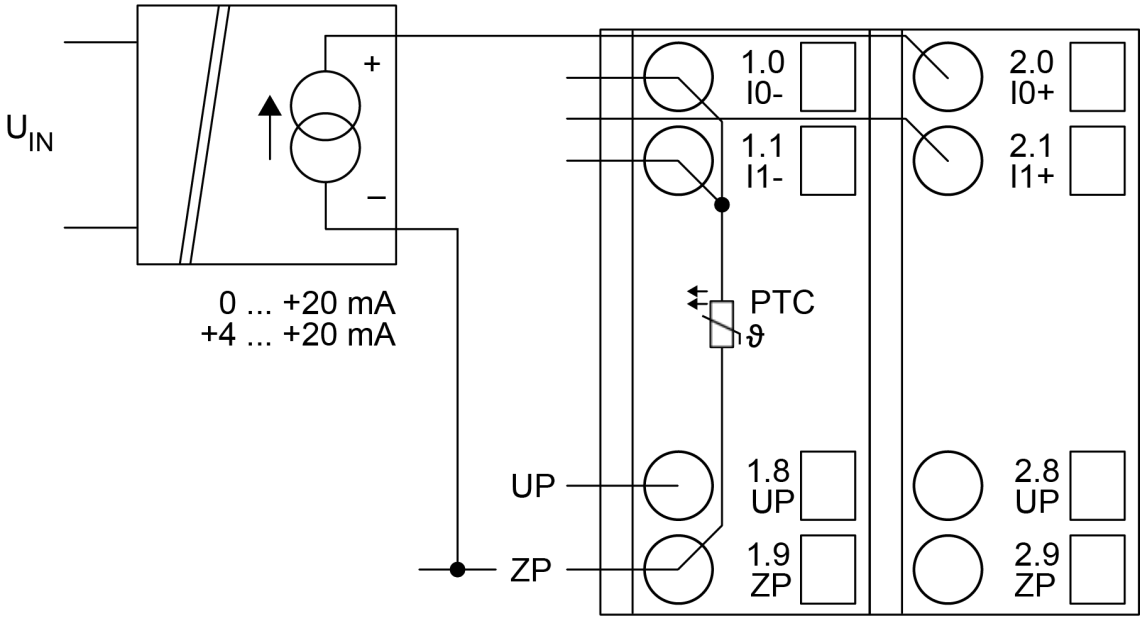


Fig. 887: Connection example

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

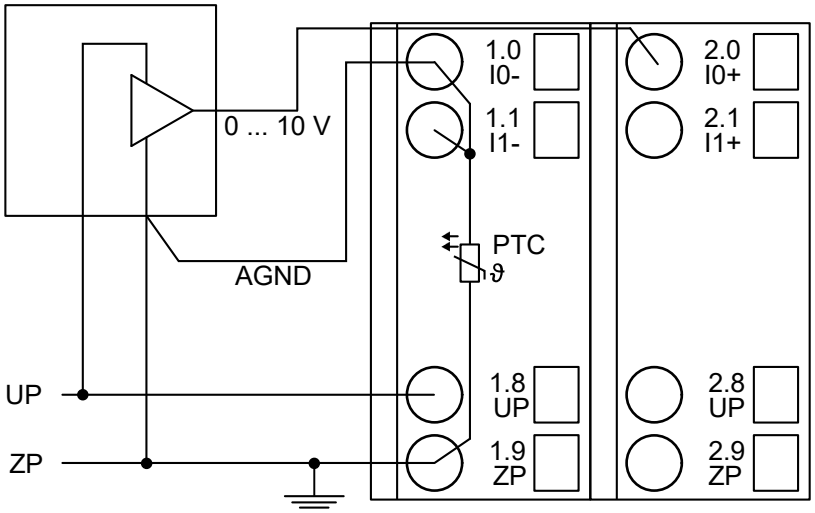


Fig. 888: Connection example

CAUTION!
The potential difference between AGND and ZP at the module must not be greater than 1V, not even in case of long lines (see figure Terminal Assignment).

If AGND does not get connected to ZP, the sensor current flows to ZP via the AGND line. The measuring signal is distorted, as a very small current flows through the voltage line. The total current through the PTC should not exceed 50 mA. This measuring method is therefore only suitable for short lines and small sensor currents. If there are bigger distances, the difference measuring method should be applied.

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

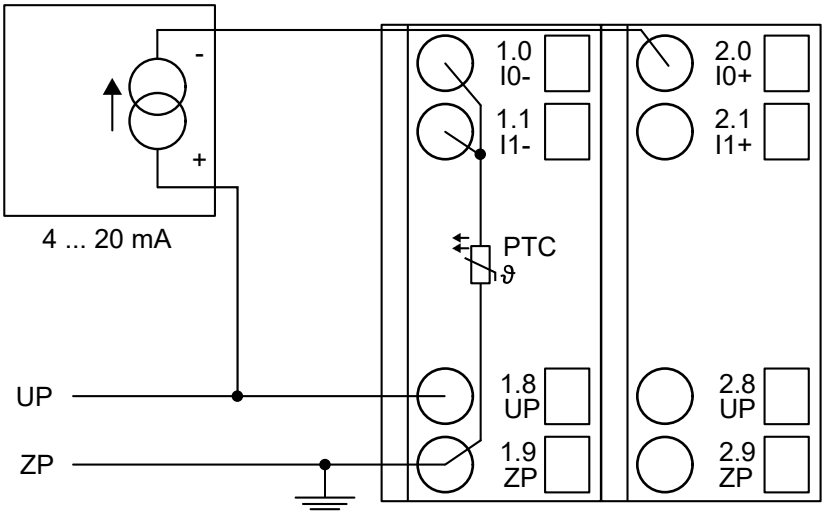


Fig. 889: Connection example

Current	4 mA...20 mA	1 channel used

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second to an analog input, this input is switched off by the module (input protection). In such cases, it is recommended to protect the analog input by a 10-volt Zener diode (in parallel to I+ and I-). But, in general, sensors with fast initialization or without current peaks higher than 25 mA are preferrable.

Unused input channels can be left open-circuited because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The use of differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!
The ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise, problems may occur concerning the common-mode input voltages of the involved analog inputs.

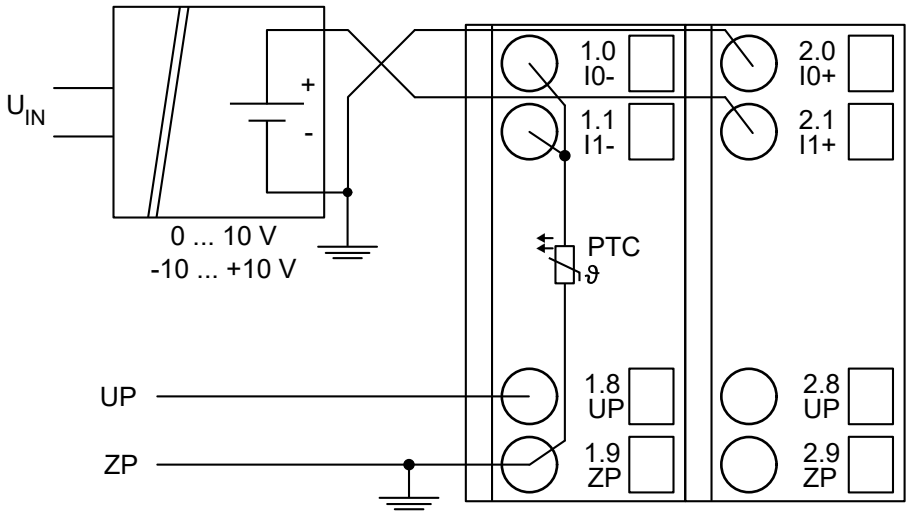


Fig. 890: Connection example

> *The negative pole of the sensor must be grounded next to the sensor.*

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

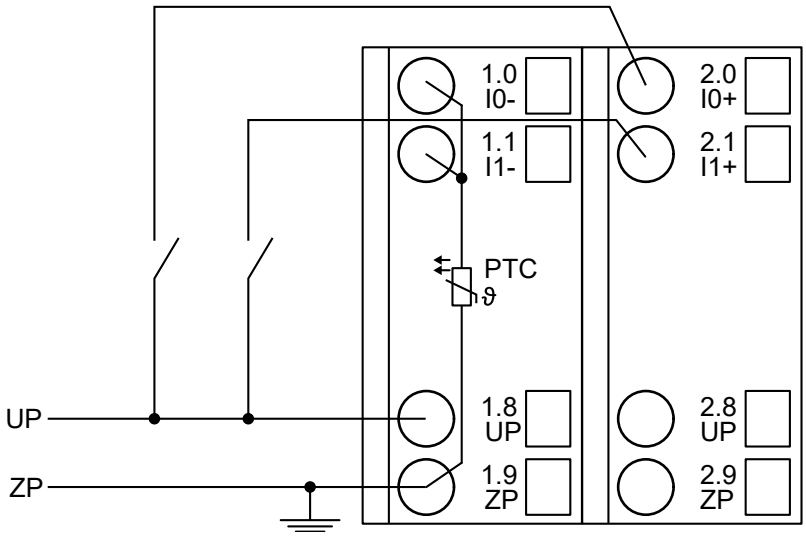


Fig. 891: Connection example

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

Connection of analog output loads (Voltage, current)

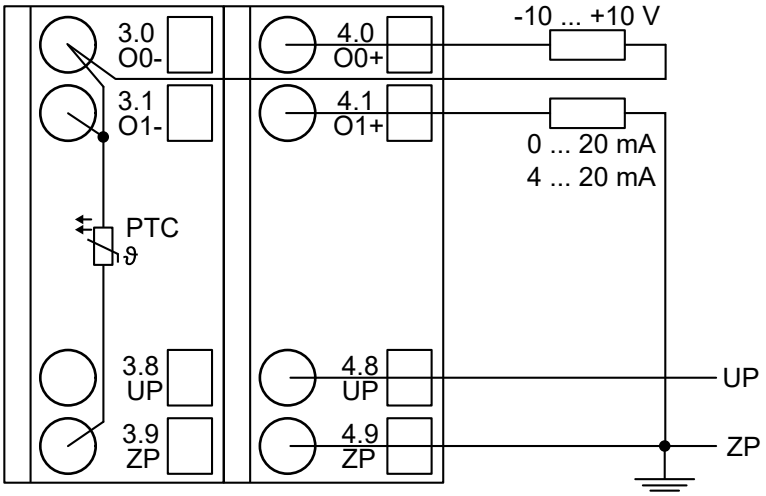


Fig. 892: Connection example

Voltage	-10 V...+10 V	Load max. ±10 mA	1 channel used
Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

Only the channels 0...3 can be configured as current output (0 mA...20 mA or 4 mA...20 mA).

Unused analog outputs can be left open-circuited.

Internal data exchange

Digital inputs (bytes)	0
Digital outputs (bytes)	0
Counter input data (words)	8
Counter output data (words)	8

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module slot address: Y = 1...7

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
1	Module ID	Internal	1500 [1])	Word	1500 0x05dc	0	65535	0x0Y01
2	Ignore module [2])	No Yes	0 1	Byte	No 0x00			not for FBP
3	Parameter length in bytes	Internal	37	Byte	37-CPU 37-FBP	0	255	0x0Y02
4	Check supply	Off On	0 1	Byte	On 0x01	0	1	0x0Y03
5	Analog data format	Default	0	Byte	Default 0x00			0x0Y04
6	Behaviour of outputs at communication errors	Off Last value Substitute value	0 1+(n*5) 2+(n*5), n ≤ 2	Byte	Off 0x00	0	2	0x0Y05
7	Channel configuration Input channel 0	See *Table 461 "Channel configuration [2])" on page 4540*		Byte	Default 0x00	0	19	0x0Y06
8	Channel monitoring Input channel 0	See *Table 462 "Channel monitoring [3])" on page 4541*		Byte	Default 0x00	0	3	0x0Y07

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
9 to 22	Channel configuration and channel monitoring of the input channels 1 to 7	See ↳ *Table 461 “Channel configuration [2])” on page 4540* and ↳ *Table 462 “Channel monitoring [3])” on page 4541*		Byte Byte	Default 0x00 0x00	0 0	19 3	0x0Y08 to 0x0Y15
23	Channel configuration Output channel 0	See ↳ *Table 461 “Channel configuration [2])” on page 4540*		Byte	Default 0x00	0	130	0x0Y16
24	Channel monitoring Output channel 0	See ↳ *Table 462 “Channel monitoring [3])” on page 4541*		Byte	Default 0x00	0	3	0x0Y17
25	Substitute value Output channel 0	only valid for output channel 0	0...0xffff	Word	Default 0x0000	0	65535	0x0Y18
26 to 31	Channel configuration and channel monitoring of the output channels 1 to 3	See ↳ *Table 461 “Channel configuration [2])” on page 4540* and ↳ *Table 462 “Channel monitoring [3])” on page 4541*		Byte Byte	Default 0x00 0x00	0 0	130 3	0x0Y19 to 0x0Y1E
32	Channel configuration Output channel 4	See ↳ *Table 461 “Channel configuration [2])” on page 4540*		Byte	Default 0x00	0	128	0x0Y1F

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.	EDS Slot/ Index
33	Channel monitoring Output channel 4	See *Table 462 “Channel monitoring 3)” on page 4541*		Byte	Default 0x00	0	3	0x0Y20
34 to 39	Channel configuration and channel monitoring of the output channels 5 to 7	See *Table 461 “Channel configuration 2)” on page 4540* and *Table 462 “Channel monitoring 3)” on page 4541*		Byte Byte	Default 0x00 0x00	0 0	128 3	0x0Y21 to 0x0Y26

1) With CS31 and addresses less than 70 and FBP, the value is increased by 1

2) Not with FBP

GSD file:

Ext_User_Prm_Data_Len =	24
Ext_User_Prm_Data_Const(0) =	0x05, 0xe2, 0x15, \ 0x01, 0x00, 0x00 \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, \ 0x00, 0x00, 0x00, 0x00, 0x00, 0x00;

Table 460: Input channel (4x)

No.	Name	Internal value, type	Default
1	Channel configuration see table 2)	Byte	0 0x00 see table 2)
2	Channel monitoring see table 3)	Byte	0 0x00 see table 3)

Table 461: Channel configuration 2)

Internal value	Operating modes of the analog inputs, individually configurable
0	Unused (default)
1	Analog input 0 V...10 V
2	Digital input
3	Analog input 0 mA...20 mA

Internal value	Operating modes of the analog inputs, individually configurable
4	Analog input 4 mA...20 mA
5	Analog input -10 V...+10 V
8	Analog input Pt100, -50 °C...+400 °C (2-wire)
9	Analog input Pt100, -50 °C...+400 °C (3-wire), requires 2 channels *)
10	Analog input 0...10 V via differential inputs, requires 2 channels *)
11	Analog input -10 V...+10 V via differential inputs, requires 2 channels *)
14	Analog input Pt100, -50 °C...+70 °C (2-wire)
15	Analog input Pt100, -50 °C...+70 °C (3-wire), requires 2 channels *)
16	Analog input Pt1000, -50 °C...+400 °C (2-wire)
17	Analog input Pt1000, -50 °C...+400 °C (3-wire), requires 2 channels *)
18	Analog input Ni1000, -50 °C...+150 °C (2-wire)
19	Analog input Ni1000, -50 °C...+150 °C (3-wire), requires 2 channels *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 462: Channel monitoring [3])

Internal value	Monitoring
0	Plausibility, open-circuit (broken wire) and short circuit
3	No monitoring

Table 463: Output channel 0 (1 channel)

No.	Name	Value	Internal value	Internal value, type	Default
1	Channel configuration	see table [4])	see table [4])	Byte	see table [4])
2	Channel monitoring	see table [5])	see table [5])	Byte	see table [5])
3	Substitute value see table [6])	0...65535	0... 0xffff	Word	0

Table 464: Output channels 1...3 (3x)

No.	Name	Internal value, type
1	Channel configuration see table [4])	Byte
2	Channel monitoring see table [6])	Byte

Table 465: Channel configuration [4])

Internal value	Operating modes of the analog outputs, individually configurable
0	Unused (default)
128	Analog output -10 V...+10 V
129	Analog output 0 mA...20 mA (not with the channels 4...7 and 12...15)
130	Analog output 4 mA...20 mA (not with the channels 4...7 and 12...15)

Table 466: Channel monitoring [5])

Internal value	Monitoring
0	Plausibility, open circuit (broken wire) and short circuit (default)
3	No monitoring

Table 467: Substitute value [6])

Intended behaviour of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

Table 468: Possible diagnosis of I/O channels

Output range	Condition	
	Output value in the PLC underflow	Output value in the PLC overflow
0..20 mA	Error identifier = 7	Error identifier = 4
4..20 mA		
-10..+10 V		

Input range	Condition			
	Short circuit	Wire break	Input value under-flow	Input value over-flow
0..20 mA	no diagnosis possible	no diagnosis possible	no diagnosis possible	Error identifier = 48
4..20 mA	Error identifier = 7	Error identifier = 7	Error identifier = 7	Error identifier = 48
-10..+10 V	no diagnosis possible	Error identifier = 48	Error identifier = 7	Error identifier = 48

Table 469: Content of diagnosis messages

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<– Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diag-nosis block		
Class	Interface	Device	Module	Channel	Error Identifier	Error message		Remedy
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firm-ware versions in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON –> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error								
				AX521	AX522			

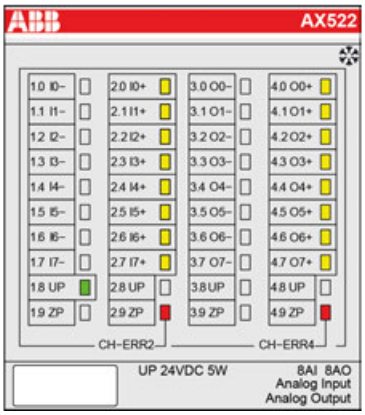
E1...E4	**d1**	**d2**	**d3**	**d4**		**Identifier 000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**		**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**		**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**		**Error Identifier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	1	0...3	0...7	48	Analog value over-flow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	7	Analog value under-flow at an analog input	Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	0...3	0...7	47	Short circuit at an analog input	Check terminal
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	4	Analog value over-flow at an analog output	Check output value
	11 / 12	ADR	1...10					
4	14	1...10	3	4...7	8...15	7	Analog value under-flow at an analog output	Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = expansion module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power ON procedure, the module initializes automatically. All LEDs (except the channel LEDs) are ON during this time.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	Inputs I0...I7	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
	Outputs O0...O7	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
	UP	Process voltage 24 V DC via terminal	Green	Process voltage is missing	Process voltage OK	--
	CH-ERR2	Channel error, error messages in groups (analog inputs or outputs combined into the groups 2 and 4)	Red	No error or process voltage is missing	Severe error within the corresponding group	Error on one channel of the group
	CH-ERR4		Red			
	CH-ERR *)	Module error	Red	--	Internal error	--
	*) Both LEDs (CH-ERR2 and CH-ERR4) light up together					

Measuring ranges

Input ranges of voltage, current and digital input

The represented resolution corresponds to 16 bits.

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 . . 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 . . 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	ON	27648 : 1	6C00 : 0001
		0.0000	0	4	OFF	0	0000
	0.0000	-0.0004 : :		3.9994		-1 -4864 -6912	FFFF ED00 E500

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 : -1.7593	: -10.0000				: -27648	: 9400
Meas-ured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<-1.7593	<-11.7589	<0.0000	<1.1858		-32768	8000

Input ranges resistance temperature detector

Range	Pt100 / Pt 1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : 0.1 °C	: 150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

The represented resolution corresponds to 16 bits.

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for +24 V (UP) as well as 1.9, 2.9, 3.9 and 4.9 for 0 V (ZP)
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Galvanic isolation	Yes, per module
Current consumption		
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	Ca. 2 mA
	From UP at normal operation	0.15 A + output loads
Inrush current from UP (at power up)		0.020 A^2s

Parameter	Value
Max. length of analog cables, conductor cross section > 0.14 mm²	100 m
Weight	300 g
Mounting position	Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the analog inputs

Parameter	Value	
Number of channels per module	8	
Distribution of channels into groups	1 group of 8 channels	
Connections of the channels I0- to I7-	Terminals 1.0 to 1.7	
Connections of the channels I0+ to I7+	Terminals 2.0 to 2.3	
Input type	Bipolar (not with current or Pt100/Pt1000/Ni1000)	
Galvanic isolation	Against internal supply and other modules	
Configurability	0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)	
Channel input resistance	Voltage: > 100 kΩ Current: ca. 330 Ω	
Time constant of the input filter	Voltage: 100 µs current: 100 µs	
Indication of the input signals	One LED per channel	
Conversion cycle	2 ms (for 8 inputs + 8 outputs), with Pt/Ni... 1 s	
Resolution	Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Unused voltage inputs	Are configured as "unused"	

Parameter	Value
Unused current inputs	Have a low resistance, can be left open-circuited
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital Inputs

Parameter		Value
Number of channels per module		Max. 8
Distribution of channels into groups		1 group of 8 channels
Connections of the channels I0+ to I7+		Terminals 2.0 to 2.7
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Input signal delay		Typ. 8 ms, configurable from 0.1 to 32 ms
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 4.3 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		8, all channels for voltage, the first 4 channels also for current
Distribution of channels into groups		1 group of 8 channels
	Channels O0-...O7-	Terminals 3.0...3.7
	Channels O0+...O7+	Terminals 4.0...4.7
Output type		Bipolar with voltage, unipolar with current
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually), current outputs only channels 0...3
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		Max. ±10 mA
Indication of the output signals		One LED per channel
Resolution		12 bits (+ sign)

Parameter	Value	
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms	
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ.	±0.5 % of full scale at 25 °C
	Max.	±1 % of full scale (all ranges) at 0 °C...60 °C or EMC disturbance
Relationship between output signal and hex code	See table, *Chapter 1.6.2.6.2.2.6.9.3 "Output ranges voltage and current" on page 4546*	
Unused outputs	Can be left open-circuited	

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 000 R0001	AX522, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires	Active
1SAP 450 000 R0001	AX522-XC, analog input/output module, 8 AI, 8 AO, U/I/Pt100, 12 bits + sign, 2-wires, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

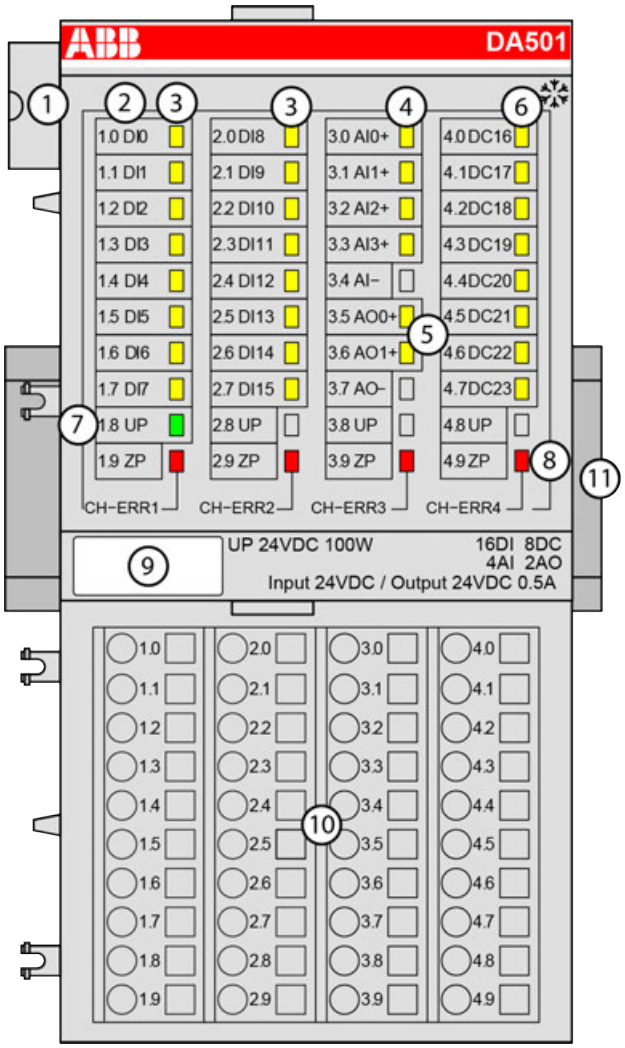
1.6.2.6.3 Digital/Analog I/O modules

S500

DA501 - Digital/Analog input/output module

- 16 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- 4 analog inputs, voltage, current and RTD.
  Resolution 12 bits plus sign
- 2 analog outputs, voltage and current
  Resolution 12 bits plus sign
- Fast counter

- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states of the digital inputs DI0 to DI15
4 4 yellow LEDs to display the signal states of the analog inputs AI0 to AI3
5 2 yellow LEDs to display the signal states of the analog outputs AO0 to AO1
6 8 yellow LEDs to display the signal state of the configurable digital inputs/outputs DC16 to DC23
7 1 green LED to display the state of the process supply voltage UP
8 4 red LEDs to display errors
9 Label
10 Terminal unit
11 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

- 16 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.

- 4 analog inputs, voltage, current and RTD.
  Resolution 12 bits plus sign
- 2 analog outputs, voltage and current
  Resolution 12 bits plus sign
- Fast counter

Parameter	Value
Fast Counter	Integrated, many configurable operating modes
Power supply	From the process supply voltage UP
LED displays	For system displays, signal states, errors and power supply
Internal supply voltage	Via the I/O bus interface (I/O bus)
External supply voltage	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU515 or TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection is carried out by using the 40 terminals of the terminal unit TU515/TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*.

The assignment of the terminals:

Terminal	Signal	Description
1.0	DI0	Signal of the digital input DI0
1.1	DI1	Signal of the digital input DI1
1.2	DI2	Signal of the digital input DI2
1.3	DI3	Signal of the digital input DI3
1.4	DI4	Signal of the digital input DI4
1.5	DI5	Signal of the digital input DI5
1.6	DI6	Signal of the digital input DI6
1.7	DI7	Signal of the digital input DI7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10

Terminal	Signal	Description
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	AI0+	Positive pole of analog input signal 0
3.1	AI1+	Positive pole of analog input signal 1
3.2	AI2+	Positive pole of analog input signal 2
3.3	AI3+	Positive pole of analog input signal 3
3.4	AI-	Negative pole of analog input signals 0 to 3
3.5	AO0+	Positive pole of analog output signal 0
3.6	AO1+	Positive pole of analog output signal 1
3.7	AO-	Negative pole of analog output signals 0 and 1
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	C16	Signal of the configurable digital input/ output C16
4.1	C17	Signal of the configurable digital input/ output C17
4.2	C18	Signal of the configurable digital input/ output C18
4.3	C19	Signal of the configurable digital input/ output C19
4.4	C20	Signal of the configurable digital input/ output C20
4.5	C21	Signal of the configurable digital input/ output C21
4.6	C22	Signal of the configurable digital input/ output C22
4.7	C23	Signal of the configurable digital input/ output C23
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DA501.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

CAUTION!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalization of a low resistance to avoid high potential differences between different parts of the plant.

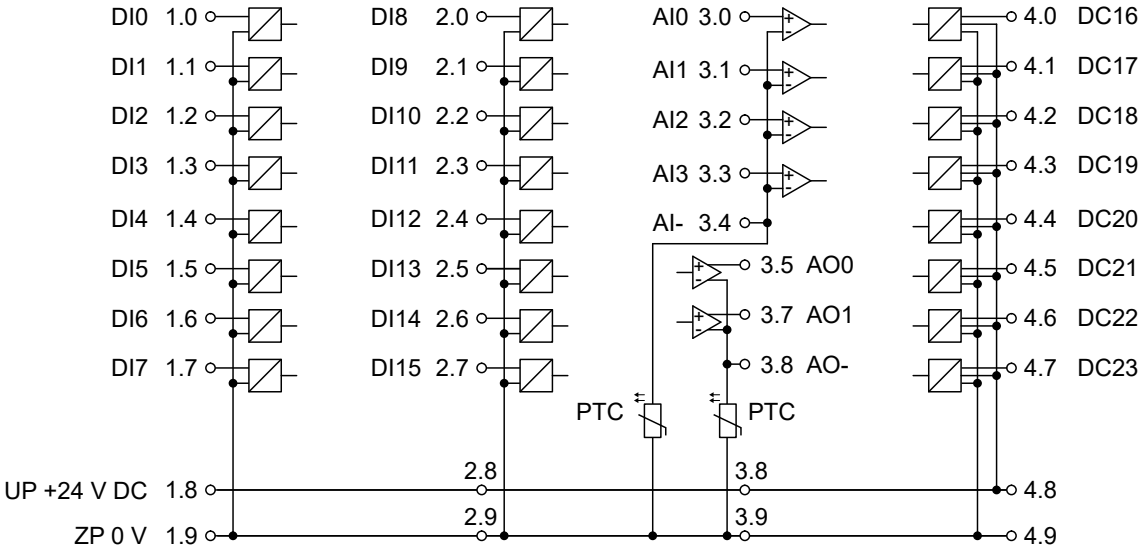


Fig. 893: Terminal assignment of the module

The module provides several diagnosis functions ↳ *Chapter 1.6.2.6.3.1.1.7 “Diagnosis” on page 4571*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI15 in the same way.

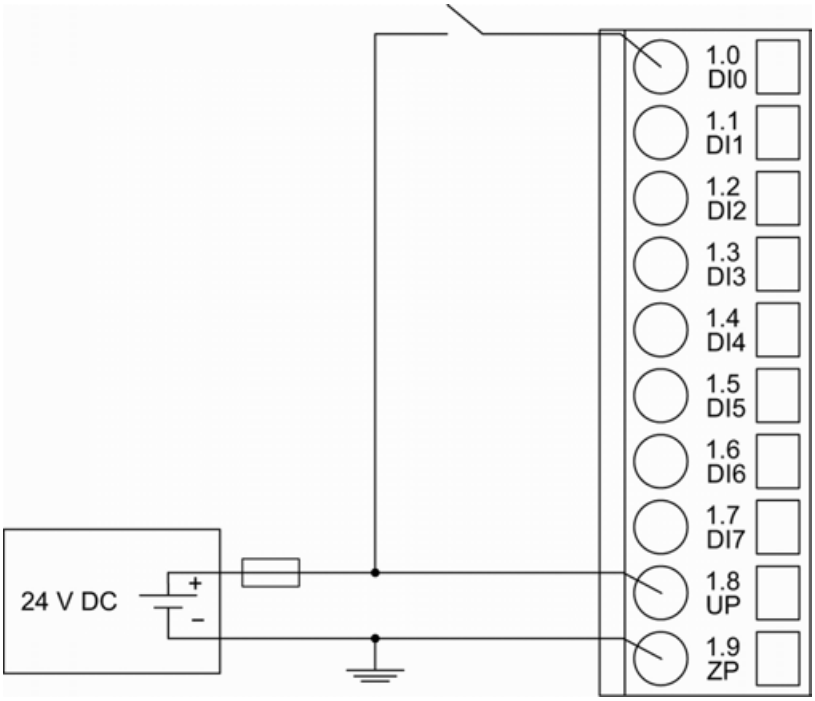


Fig. 894: Connection of the module

The meaning of the LEDs is described in the Displays ↳ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574* chapter.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC16 and DC17. DC16 is connected as an input and DC17 is connected as an output. Proceed with the configurable digital inputs/outputs DC18 to DC23 in the same way.

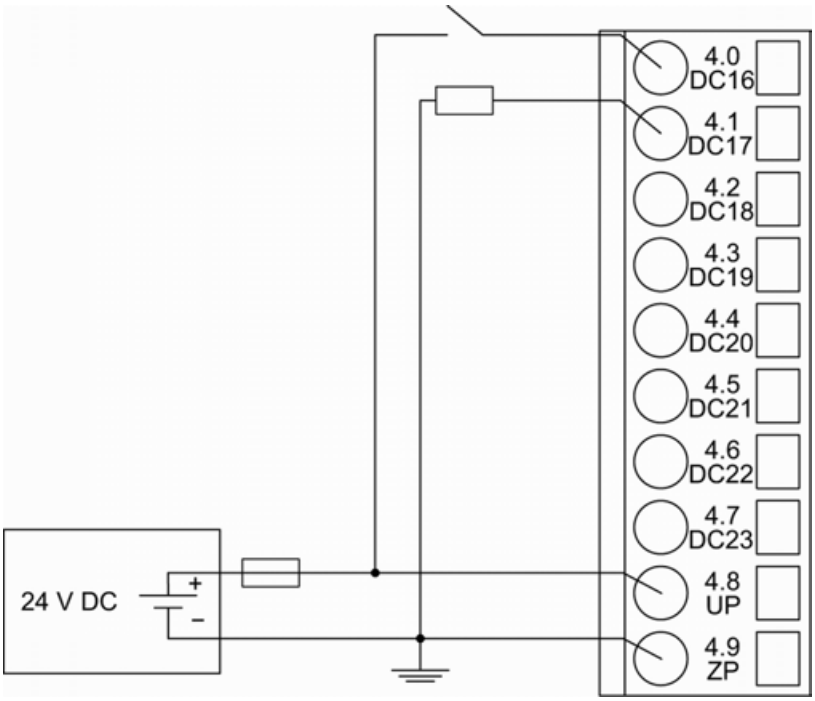


Fig. 895: Connection of configurable digital inputs/outputs to the module

CAUTION!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DA501.

If the inputs are used as fast counter inputs, connect a 470 Ω / 1 W resistor in series to inputs DC16/DC17.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA501 provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

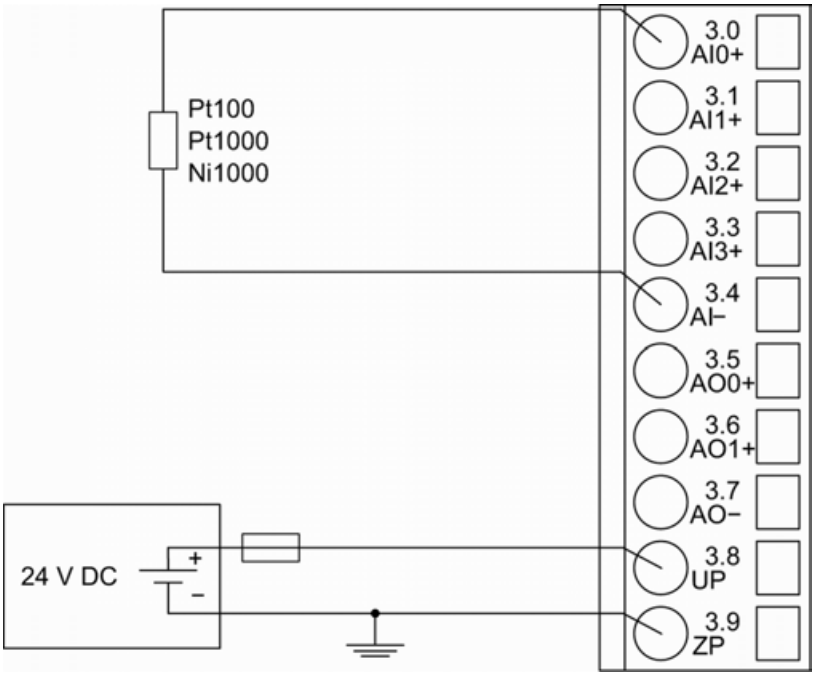


Fig. 896: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured ↬ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↬ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

The module DA501 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA501 provides a constant current source which is multiplexed over the max. 4 analog input channels.

0

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

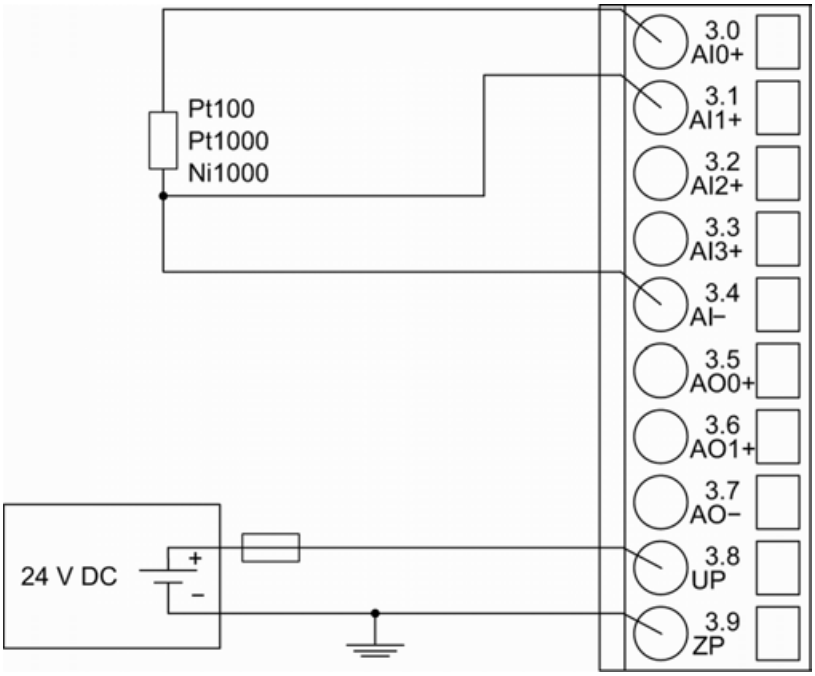


Fig. 897: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.1.7 “Diagnosis” on page 4571*.

0

The module DA501 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

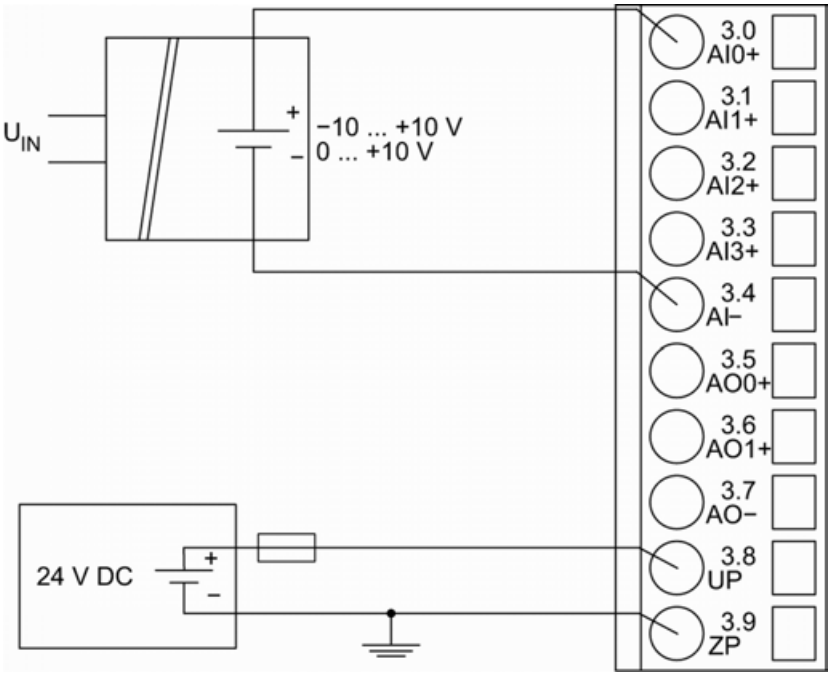


Fig. 898: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

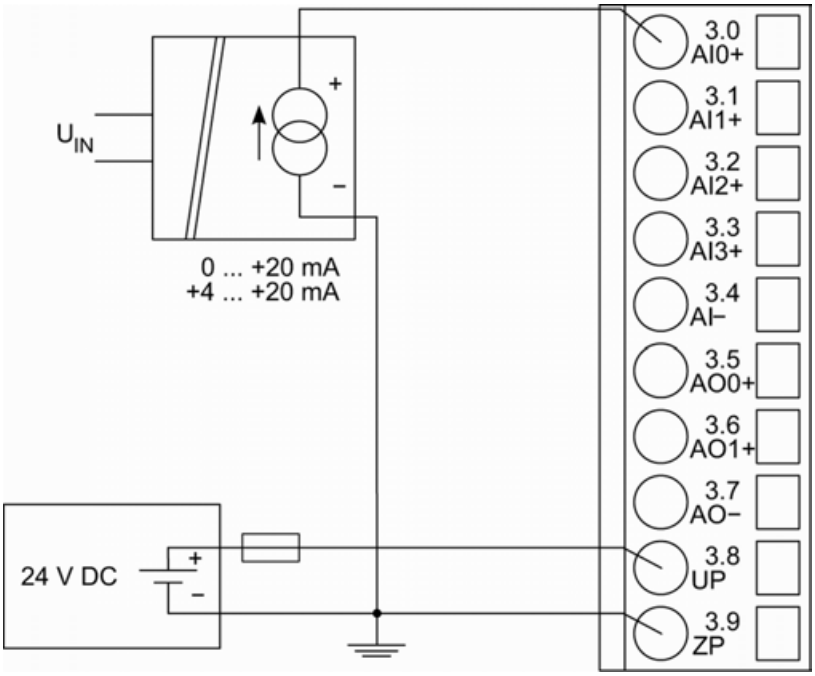


Fig. 899: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

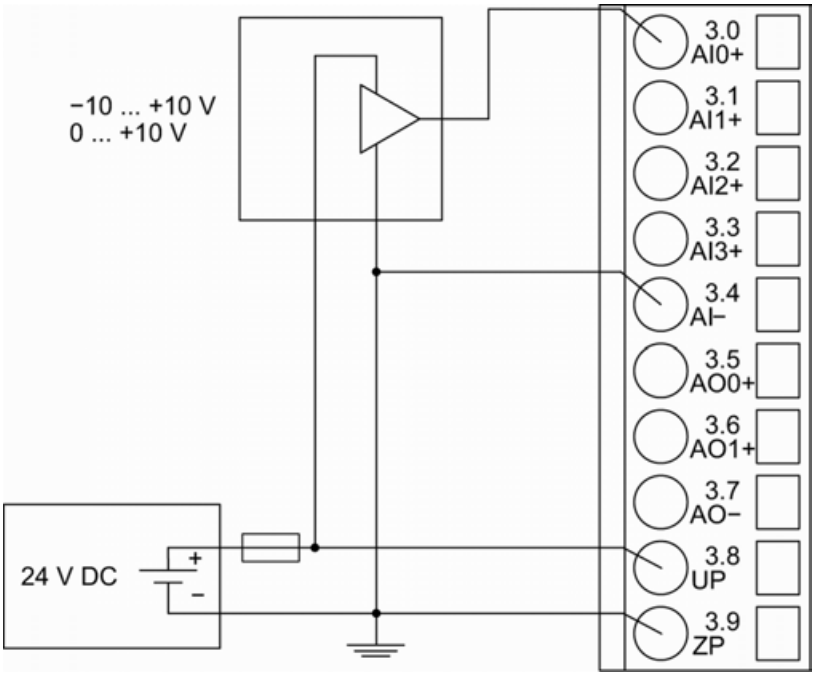


Fig. 900: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

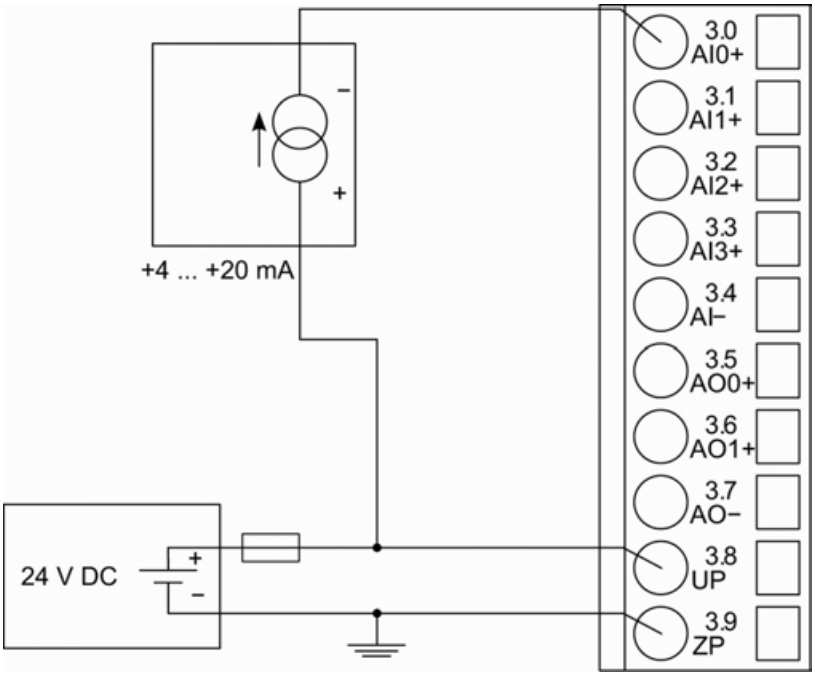


Fig. 901: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.1.6 "Parameterization" on page 4567*:

Current	4 mA...20 mA	1 channel used

For a description of function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↳ *Chapter 1.6.2.6.3.1.1.8 "State LEDs" on page 4574*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Only use sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

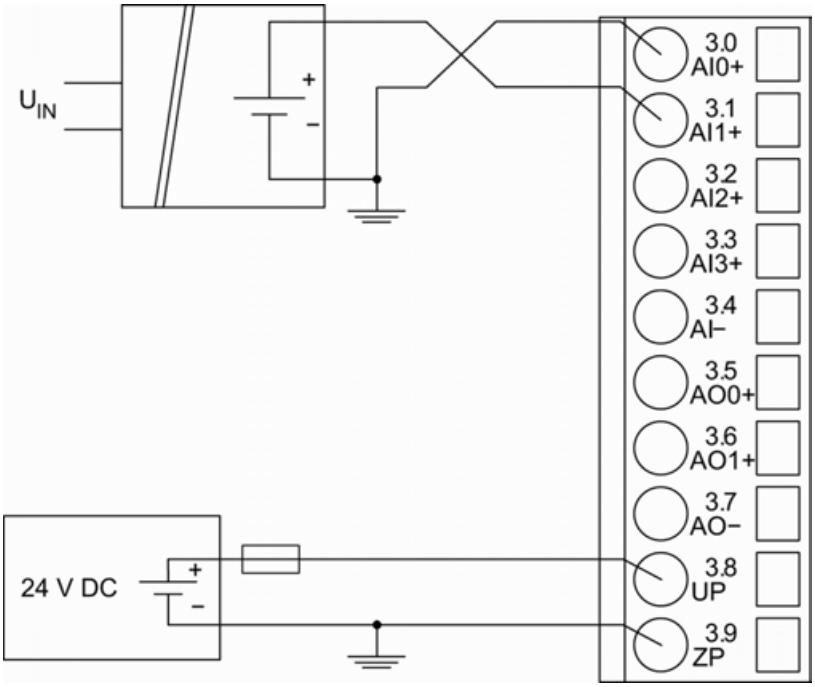


Fig. 902: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

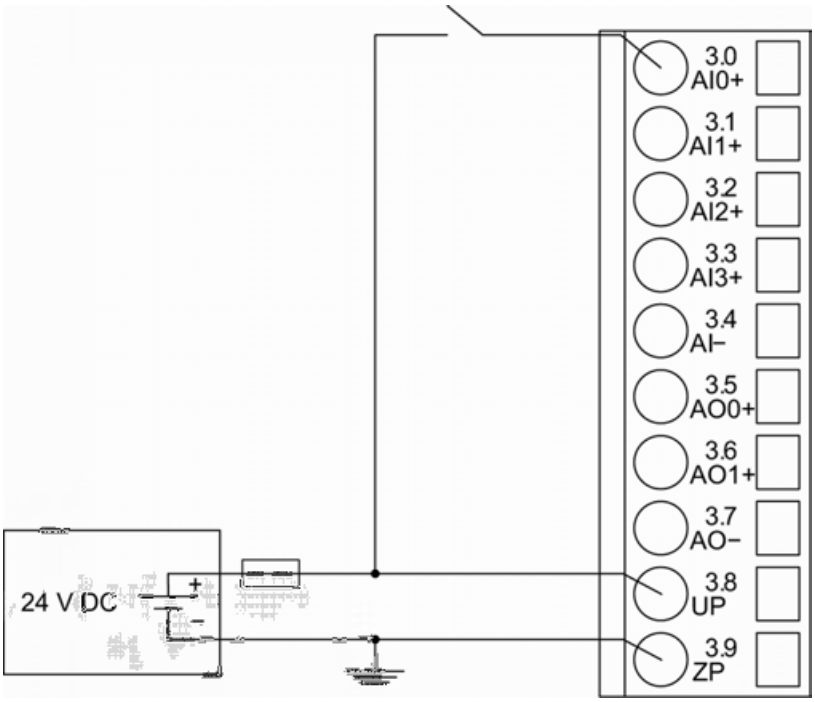


Fig. 903: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↯ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567*:

Digital input	24 V	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↯ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

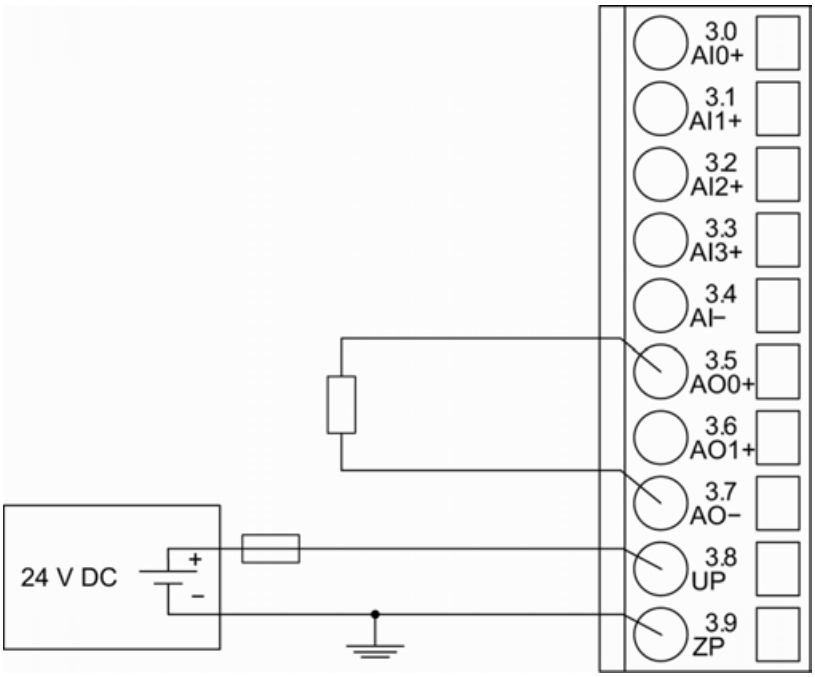


Fig. 904: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↬ *Chapter 1.6.2.6.3.1.1.6 “Parameterization” on page 4567* :

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter ↬ *Chapter 1.6.2.6.3.1.1.8 “State LEDs” on page 4574*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

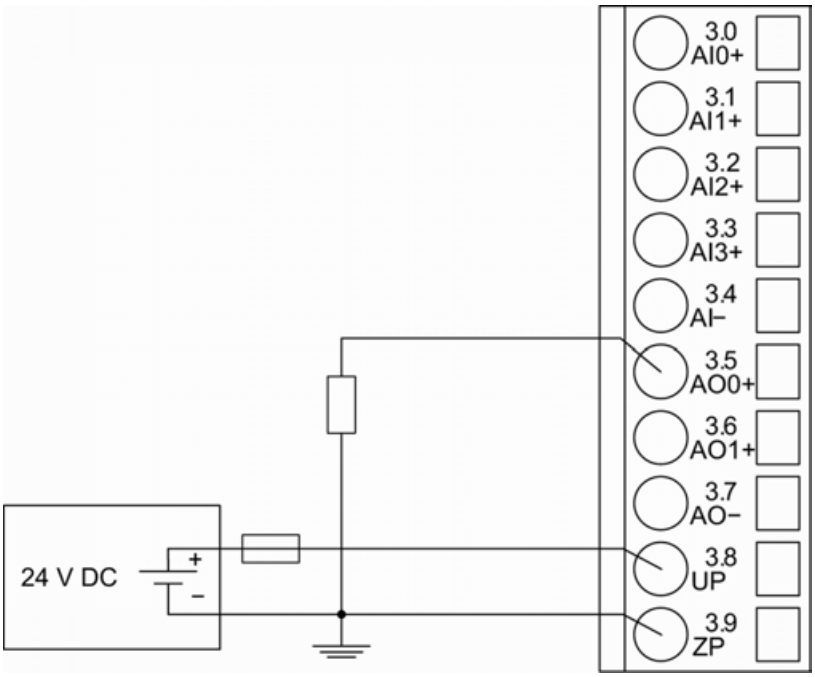


Fig. 905: Connection of analog output loads (current)

The following measuring ranges can be configured *Chapter 1.6.2.6.3.1.1.6 "Parameterization" on page 4567*:

0

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

For a description of the function of the LEDs, please refer to the Diagnosis and displays / Displays chapter *Chapter 1.6.2.6.3.1.1.8 "State LEDs" on page 4574*.

Unused analog outputs can be left open-circuited.

Internal data exchange

	Without the fast counter	**With the fast counter (only with AC500)**
Digital inputs (bytes)	3	5
Digital outputs (bytes)	1	3
Analog inputs (words)	4	4
Digital outputs (words)	2	2
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The module does not store configuration data itself. It gets its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Module ID [1)]	Internal	1810	WORD	1810	0x0Y01
Ignore module see table [2)]	Internal	Yes No	BYTE	No	not for FBP
Parameter length	Internal	8	BYTE	8	0xY02
Check supply	off	0	BYTE	1	0xY03
	on	1			
Fast counter [3)]	0 : 10 [4)]	0 : 10	BYTE	0	not for FBP
Behavior outputs at comm. error [5)]	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00	0x0Y07

[2)]	Setting	Description
	On	Error LED lights up at errors of all error classes, Failsafe mode off
	Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
	Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe mode off
	On +Failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)

[2])	Setting	Description
	Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
	Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

Remarks:

[1]) With a faulty ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission

[2]) Not for FBP

[3]) With FBP or CS31 without the parameter "Fast Counter"

The fast counter of the module does not work if the module is connected to an FBP interface module or CS31 bus module.

[4]) For counter operating modes, please refer to the description of the fast counter ↬ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

[5]) The parameter Behavior outputs at comm. error is only analyzed if the Failsafe-mode is ON.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0x0Y05
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01	0x0Y06
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000	0x0Y08

*) The parameters Behavior DO at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Analog data format	Standard Reserved	0 255	BYTE	0	0x0Y04

*) The parameter Behavior AO at comm. error is only analyzed if the Failsafe mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input 0, Channel configuration	see *Table 470 “ Channel configuration” on page 4569*	see *Table 470 “ Channel configuration” on page 4569*	BYTE	0	0x0Y09
Input 0, Check channel	see *Table 471 “ Channel monitoring” on page 4570*	see *Table 471 “ Channel monitoring” on page 4570*	BYTE	0	0x0Y0A
:	:	:	:	:	
:	:	:	:	:	
Input 3, Channel configuration	see *Table 470 “ Channel configuration” on page 4569*	see *Table 470 “ Channel configuration” on page 4569*	BYTE	0	0x0Y0F
Input 3, Check channel	see *Table 471 “ Channel monitoring” on page 4570*	see *Table 471 “ Channel monitoring” on page 4570*	BYTE	0	0x0Y10

Table 470: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 471: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
0 Output 0, Channel configuration	see *Table 472 “Channel configuration” on page 4571*	see *Table 472 “Channel configuration” on page 4571*	BYTE	0	0x0Y11
Output 0, Check channel	see *Table 473 “Channel monitoring” on page 4571*	see *Table 473 “Channel monitoring” on page 4571*	BYTE	0	0x0Y12
Output 0, Substitute value	see *Table 474 “Substitute value” on page 4571*	see *Table 474 “Substitute value” on page 4571*	WORD	0	0x0Y13
Output 1, Channel configuration	see *Table 472 “Channel configuration” on page 4571*	see *Table 472 “Channel configuration” on page 4571*	BYTE	0	0x0Y14
Output 1, Check channel	see *Table 473 “Channel monitoring” on page 4571*	see *Table 473 “Channel monitoring” on page 4571*	BYTE	0	0x0Y15
Output 1, Substitute value	see *Table 474 “Substitute value” on page 4571*	see *Table 474 “Substitute value” on page 4571*	WORD	0	0x0Y16

Table 472: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 473: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 474: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behavior of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

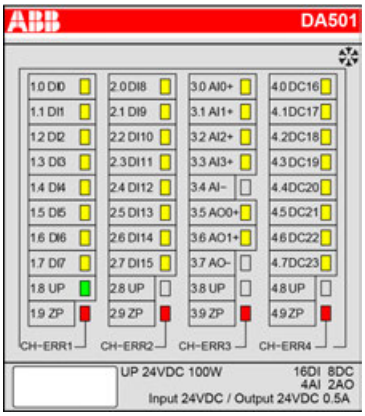
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Module error							
0	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
3	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout in the I/O module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	40	Different hard-/firmware versions in the module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	36	Internal data exchange failure	
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)	Process voltage ON
	11 / 12	ADR	1...10				
Channel error DA501							
4	14	1...10	2	22...29 [5])	47	Short circuit at a digital output	Check connection
	11 / 12	ADR	1...10				
Channel error DA501							
4	14	1...10	1	16...19 [6])	48	Analog value overflow or broken wire at an analog input	Check input value or terminal
	11 / 12	ADR	1...10				
4	14	1...10	1	16...19 [6])	7	Analog value underflow at an analog input	Check input value
	11 / 12	ADR	1...10				
4	14	1...10	1	16...19 [6])	47	Short circuit at an analog input	Check ter-minal
	11 / 12	ADR	1...10				
4	14	1...10	3	20...21 [7])	4	Analog value overflow at an analog output	Check output value
	11 / 12	ADR	1...10				

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	20...21 [7])	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (1 = AI, 3 = AO, 4 = DC); COM1/ COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.
[5])	Ch = 22...29 indicates the digital inputs/outputs DC16...DC23
[6])	Ch = 16...19 indicates the analog inputs AI0...AI3
[7])	Ch = 20...21 indicates the analog outputs AO0...AO1

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	DI0 to DI15	Digital input	Yellow	Input is OFF	Input is ON [1])	--
	DC16 to DC23	Digital input/ output	Yellow	Input/output is OFF	Input/output is ON [1])	--
	AI0 to AI3	Analog input	Yellow	Input is OFF	Input is ON [2])	--
	AO0 to AO1	Analog output	Yellow	Output is OFF	Output is ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Severe error within the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [3])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) Brightness depends on the value of the analog signal					
	[3]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Overflow	> 11.7589	> 11.7589	> 23.5178	> 22.8142	
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006	
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	on
	0.0000	0.0000	0	4	off
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0	

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Measured value too low		-10.0004 : -11.7589			
Underflow	< 0.0000	< -11.7589	< 0.0000	< 0.0000	

Range	Digital value	
	Decimal	Hex.
Overflow	32767	7FFF
Measured value too high	32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	27648 : 1	6C00 : 0001
	0	0000
	-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low	-27649 : -32512	93FF : 8100
Underflow	-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C
Measured value too high		450.0 °C : 400.1 °C	
			160.0 °C : 150.1 °C
	80.0 °C : 70.1 °C		

Range	**Pt100 / Pt1000** **-50...70 °C**	**Pt100 / Pt1000** **-50...400 °C**	**Ni1000** **-50...150 °C**
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C
	0.0 °C	0.0 °C	0.0 °C
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C

Range	**Digital value**	
	Decimal	**Hex.**
Overflow	32767	7FFF
Measured value too high	4500 : 4001	1194 : 0FA1
	1600 : 1501	0640 : 05DD
	800 : 701	0320 : 02BD
Normal range	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0	0000
	-1 : -500	FFFF : FE0C
Measured value too low	-501 : -600	FE0B : FDA8
Underflow	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA
Overflow	>11.7589 V	>23.5178 mA	>22.8142 mA
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA
	0.0000 V	0.0000 mA	4.0000 mA
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA
Underflow	0 V	0 mA	0 mA

Range	Digital value	
	Decimal	**Hex.**
Overflow	> 32511	> 7EFF
Value too high	32511 : 27649	7EFF : 6C01
Normal range	27648 : 1	6C00 : 0001
	0	0000
	-1 -6912 -27648	FFFF E500 9400
Value too low	-27649 : -32512	93FF : 8100
Underflow	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

Technical data of the module

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for UP (+24 V DC) and 1.9, 2.9, 3.9 and 4.9 for ZP (0 V DC)
	Protection against reverse voltage	yes
	Rated protection fuse at UP	10 A fast
	Rated value	24 V DC
	Max. ripple	5 %
Current consumption		
	From UP	0.07 A + max. 0.5 A per output
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	Inrush current from UP (at power-up)	0.04 A^2s
Galvanic isolation		Yes, per module
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal mounting or vertical with derating (output load reduced to 50 % at 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		16
Distribution of the channels into groups		2 groups of 8 channels
Terminals of the channels DI0 to DI7		Terminals 1.0 to 1.7
Terminals of the channels DI8 to DI15		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable digital I/O channels can be defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC16...DC23	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels DC16...DC23	Terminals 4.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)

Parameter	Value
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	Yes, per module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all inputs		Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	shielded	1000 m
	unshielded	600 m

* Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DC16 to DC23	Terminals 4.0 to 4.7

Parameter	Value
Reference potential for all outputs	Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the supply voltage, signal name UP)
Output voltage for signal 1	UP (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
rated value per channel	500 mA at UP = 24 V
max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA
Fuse for UP	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

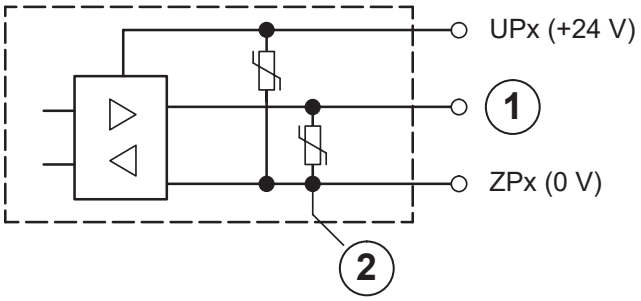


Fig. 906: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

The fast counter of the module does not work if the module is connected to an FBP interface module or CS31 bus module.

Parameter	Value
Used inputs	DC16 / DC17
Used outputs	DC18
Counting frequency	Max. 50 kHz

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for AI0+ to AI3+		Terminal 3.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C

Parameter	Value
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 % For XC version below 0 °C and above 60 °C: on request
Relationship between input signal and hex code	↳ *Chapter 1.6.2.6.3.1.1.9.1 “Input ranges voltage, current and digital input” on page 4574* ↳ *Chapter 1.6.2.6.3.1.1.9.2 “Input ranges resistance temperature detector” on page 4575*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 3.5 and 3.6
Reference potential for AO0+ to AO1+		Terminal 3.7 (AO-) for voltage output Terminals 1.9, 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current

Parameter		Value
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load) as current output		0 Ω...500 Ω
Output loadability as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		↳ *Chapter 1.6.2.6.3.1.1.9.3 “Output ranges voltage and current” on page 4577*
Unused outputs		Are configured as "unused" (default value) and can be left open-circuited

Internal data exchange

	Without the fast counter	With the fast counter (only with AC500)
Digital inputs (bytes)	3	5
Digital outputs (bytes)	1	3
Analog inputs (words)	4	4
Analog outputs (words)	2	2
Counter input data (words)	0	4
Counter output data (words)	0	8

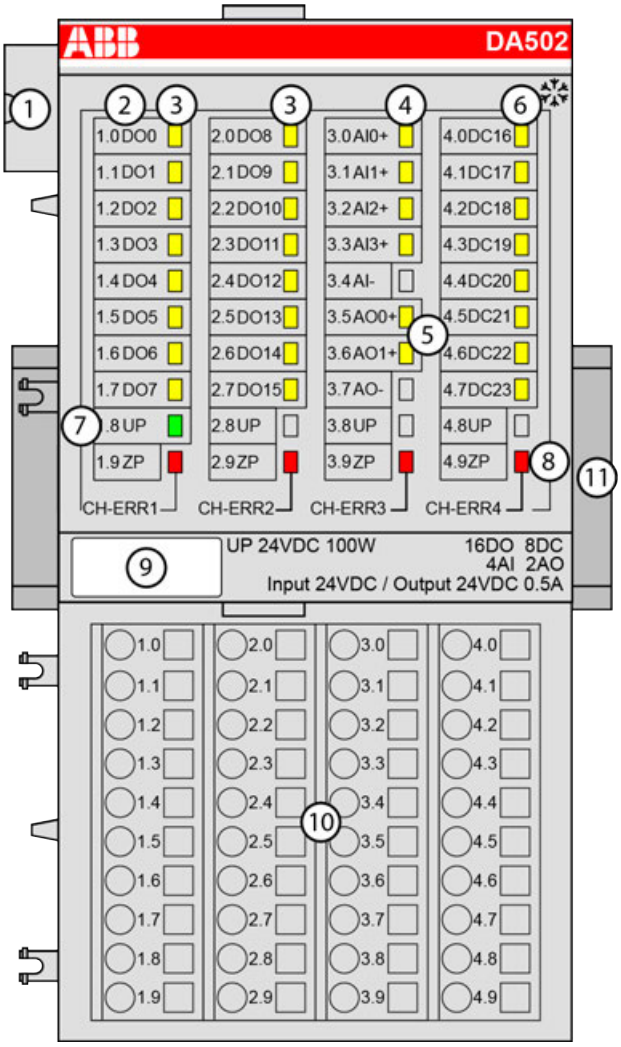
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 700 R0001	DA501, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO	Active
1SAP 450 700 R0001	DA501-XC, digital/analog input/output module, 16 DI, 8 DC, 4 AI, 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DA502 - Digital/Analog input/output module

- 16 digital outputs, 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- 4 analog inputs, voltage, current and RTD, resolution 12 bits plus sign
- 2 analog outputs, voltage and current, resolution 12 bits plus sign
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 16 yellow LEDs to display the signal states of the digital outputs DO0 to DO15
4 4 yellow LEDs to display the signal states of the analog inputs AI0 to AI3
5 2 yellow LEDs to display the signal states of the analog outputs AO0 to AO1
6 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs DC16 to DC23
7 1 green LED to display the state of the process supply voltage UP
8 4 red LEDs to display errors
9 Label
10 Terminal unit
11 DIN rail
❄ Sign for XC version

Intended purpose

The device can be used as a decentralized I/O extension module for S500 communication iInterface modules (e. g. CI592-CS31, CI501-PNIO, CI541-DP, CI581-CN) or as a centralized extension module for AC500 CPUs.

Functionality

Parameter	Value
Fast counter	Integrated, many configurable operating modes
Power supply	From the process supply voltage UP
LED displays	For system displays, signal states, errors and power supply
Internal supply voltage	Via the I/O bus interface (I/O bus)
External supply voltage	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU515 or TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The connection is carried out by using the 40 terminals of the terminal unit TU515/TU516 *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*.

The assignment of the terminals:

Terminal	Signal	Description
1.0	DO0	Signal of the digital output DO0
1.1	DO1	Signal of the digital output DO1
1.2	DO2	Signal of the digital output DO2
1.3	DO3	Signal of the digital output DO3
1.4	DO4	Signal of the digital output DO4
1.5	DO5	Signal of the digital output DO5
1.6	DO6	Signal of the digital output DO6
1.7	DO7	Signal of the digital output DO7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DO8	Signal of the digital output DO8
2.1	DO9	Signal of the digital output DO9

Terminal	Signal	Description
2.2	DO10	Signal of the digital output DO10
2.3	DO11	Signal of the digital output DO11
2.4	DO12	Signal of the digital output DO12
2.5	DO13	Signal of the digital output DO13
2.6	DO14	Signal of the digital output DO14
2.7	DO15	Signal of the digital output DO15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	AI0+	Positive pole of analog input signal 0
3.1	AI1+	Positive pole of analog input signal 1
3.2	AI2+	Positive pole of analog input signal 2
3.3	AI3+	Positive pole of analog input signal 3
3.4	AI-	Negative pole of analog input signals 0 to 3
3.5	AO0+	Positive pole of analog output signal 0
3.6	AO1+	Positive pole of analog output signal 1
3.7	AO-	Negative pole of analog output signals 0 and 1
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DC16	Signal of the configurable digital input/output DC16
4.1	DC17	Signal of the configurable digital input/output DC17
4.2	DC18	Signal of the configurable digital input/output DC18
4.3	DC19	Signal of the configurable digital input/output DC19
4.4	DC20	Signal of the configurable digital input/output DC20
4.5	DC21	Signal of the configurable digital input/output DC21
4.6	DC22	Signal of the configurable digital input/output DC22
4.7	DC23	Signal of the configurable digital input/output DC23
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per DA502.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

Removal or insertion under power is only permissible under conditions described in Hot Swap chapter *Chapter 1.6.2.6 “I/O modules” on page 4124*.

The devices are not designed for removal or insertion under power when Hot Swap conditions do not apply. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

The PLC modules must not be removed while the plant is connected to a power supply.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove or replace a module.

CAUTION!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalization of a low resistance to avoid high potential differences between different parts of the plant.

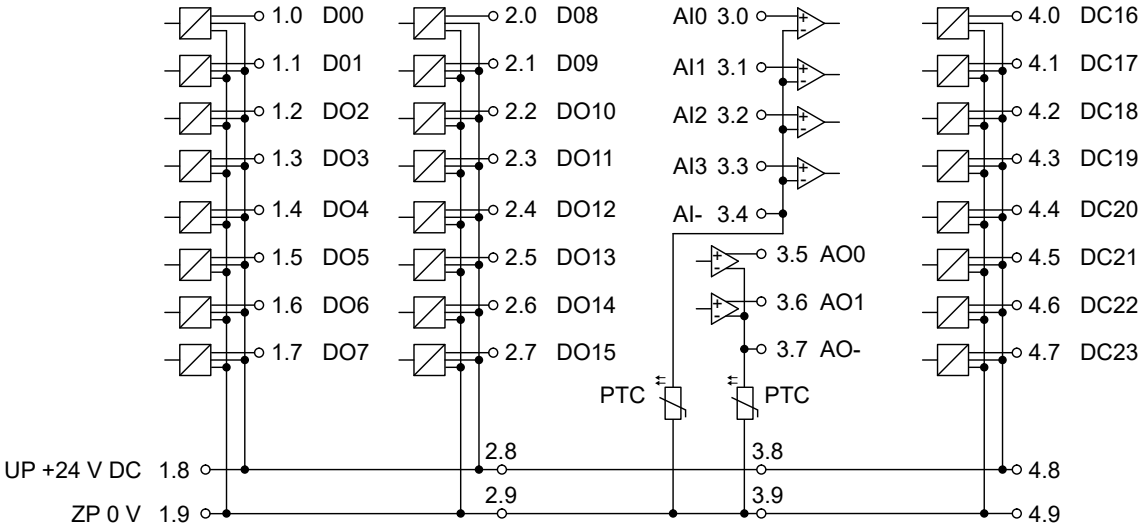
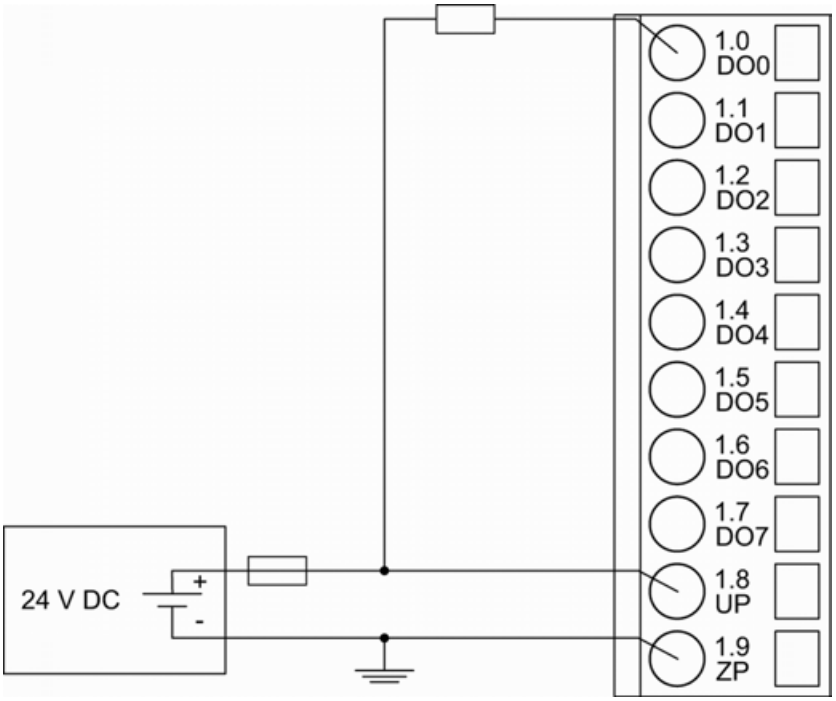


Fig. 907: Terminal assignment of the module

The module provides several diagnosis functions *Chapter 1.6.2.6.3.1.2.7 “Diagnosis” on page 4605*.

Connection of the digital outputs

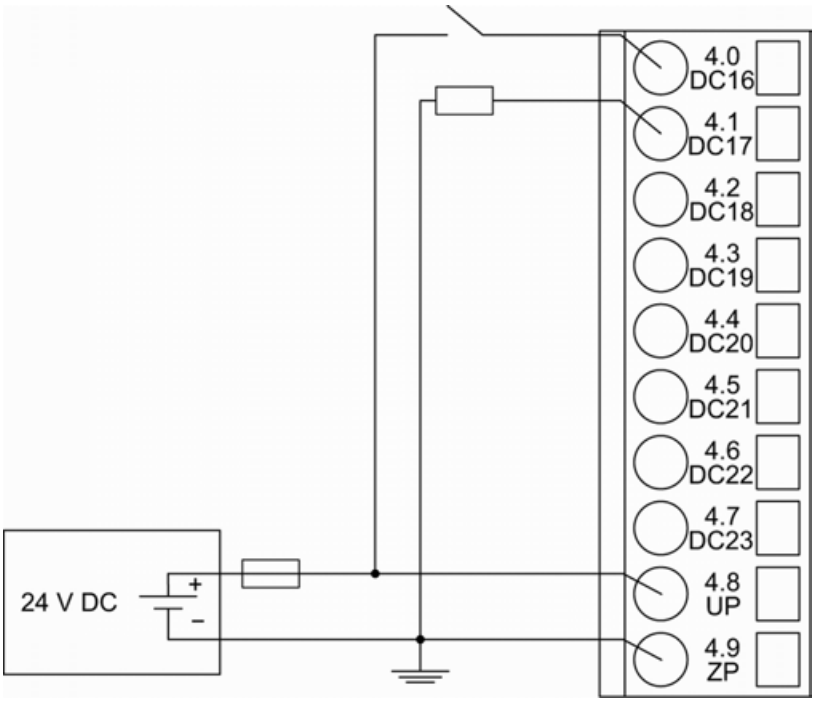
The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 to DO15 in the same way.



For a description of the meaning of the LEDs, please refer to the Displays chapter *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC16 and DC17. DC16 is connected as an input and DC17 is connected as an output. Proceed with the configurable digital inputs/outputs DC18 to DC23 in the same way.



NOTICE!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of DA502.

Connect a 470 Ω / 1 W resistor in series to inputs DC16/DC17 if they are used as fast counter inputs to avoid any influences.

For a description of tthe meaning of the LEDs, please refer to the Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608* chapter.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA502 provides a constant current source which is multiplexed over max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

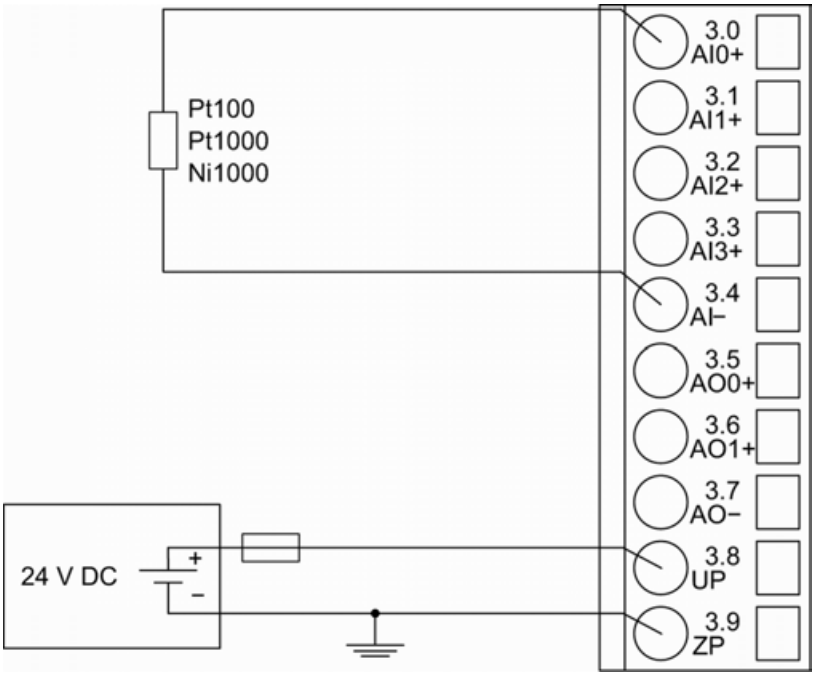


Fig. 908: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

The module DA502 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module DA502 provides a constant current source which is multiplexed over max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

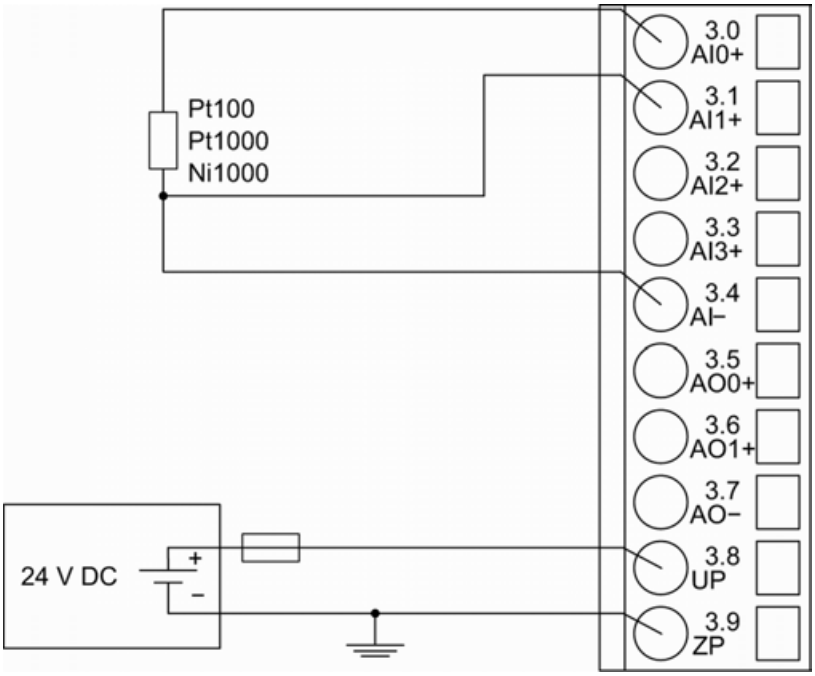


Fig. 909: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

The module DA502 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

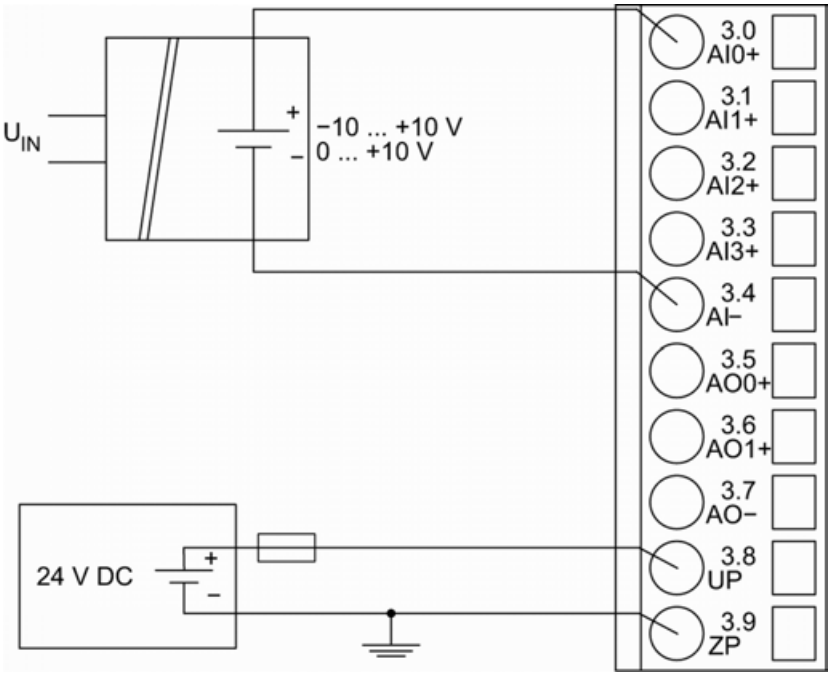


Fig. 910: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

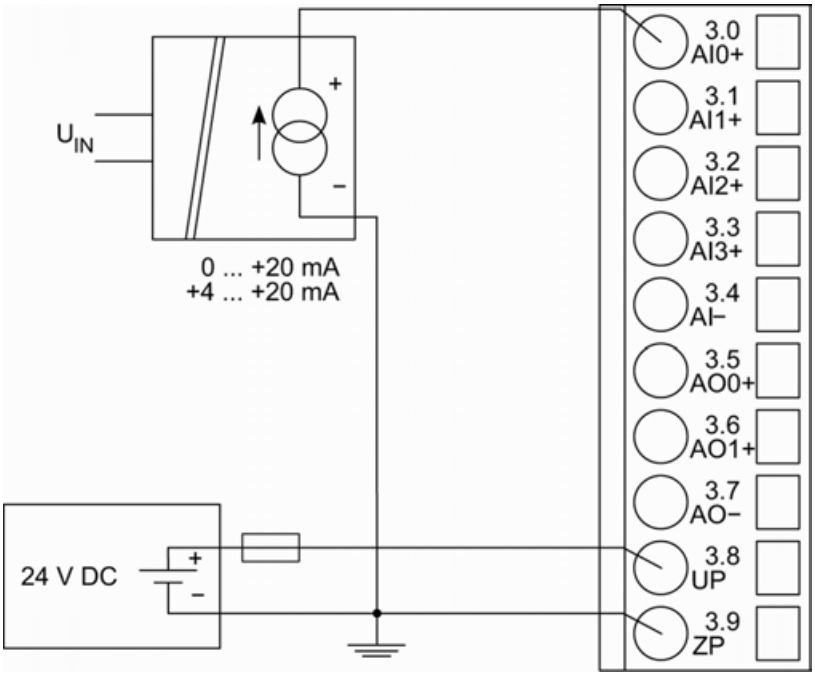


Fig. 911: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

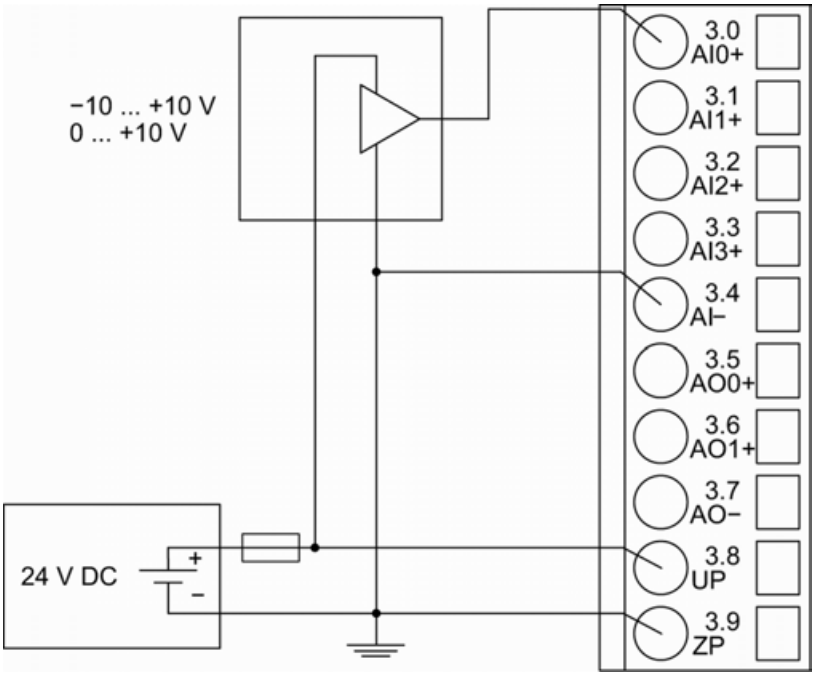


Fig. 912: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following measuring ranges can be configured ↬ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↬ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↬ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

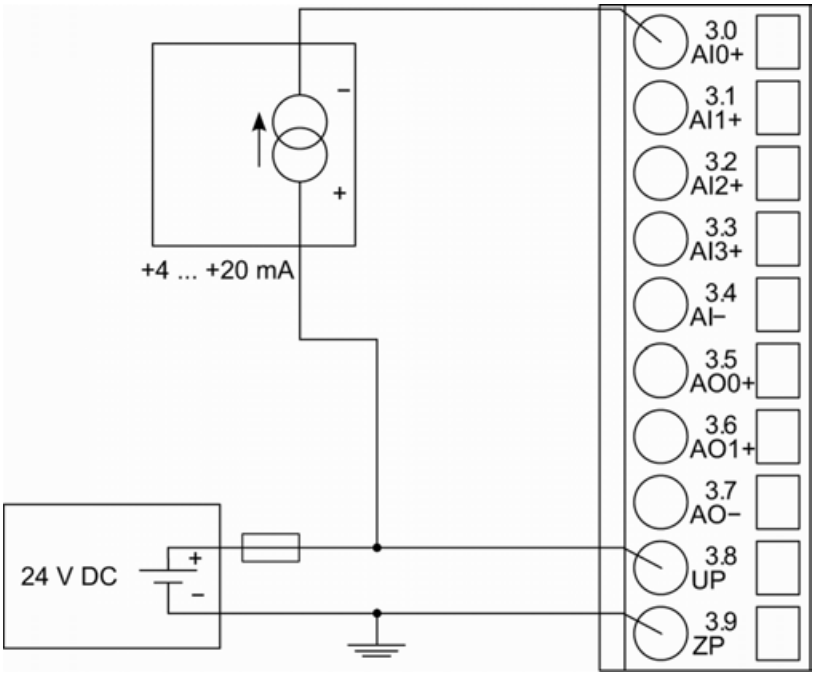


Fig. 913: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Current	4 mA...20 mA	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

NOTICE!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too large a potential difference with respect to ZP (max. ±1 V within the full signal range).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

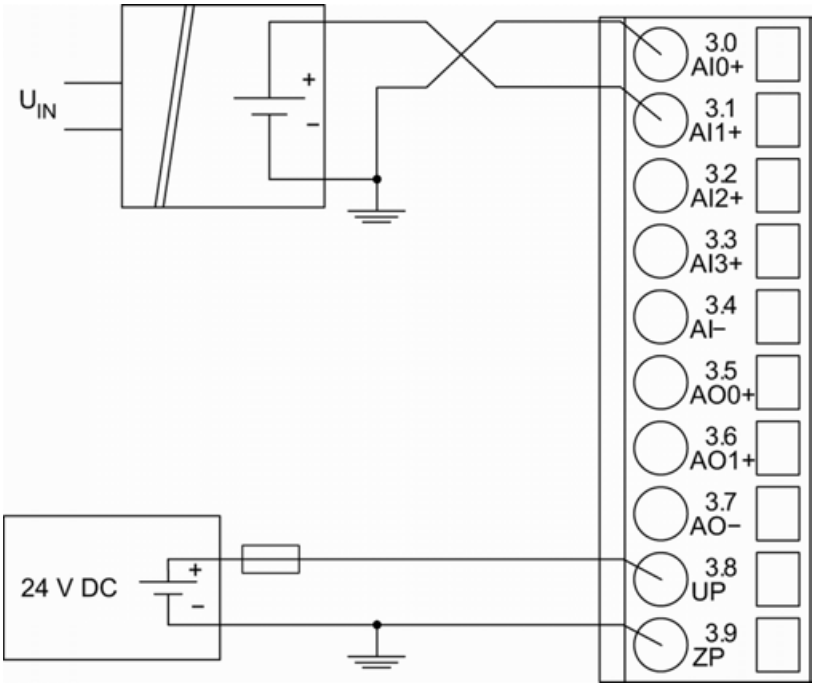


Fig. 914: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

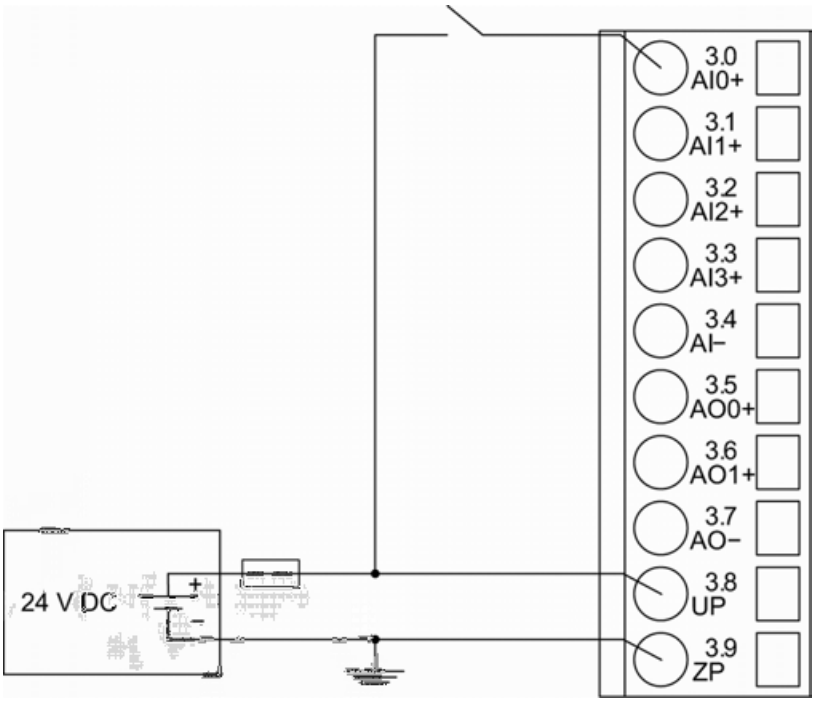


Fig. 915: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608* :

Digital input	24 V	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

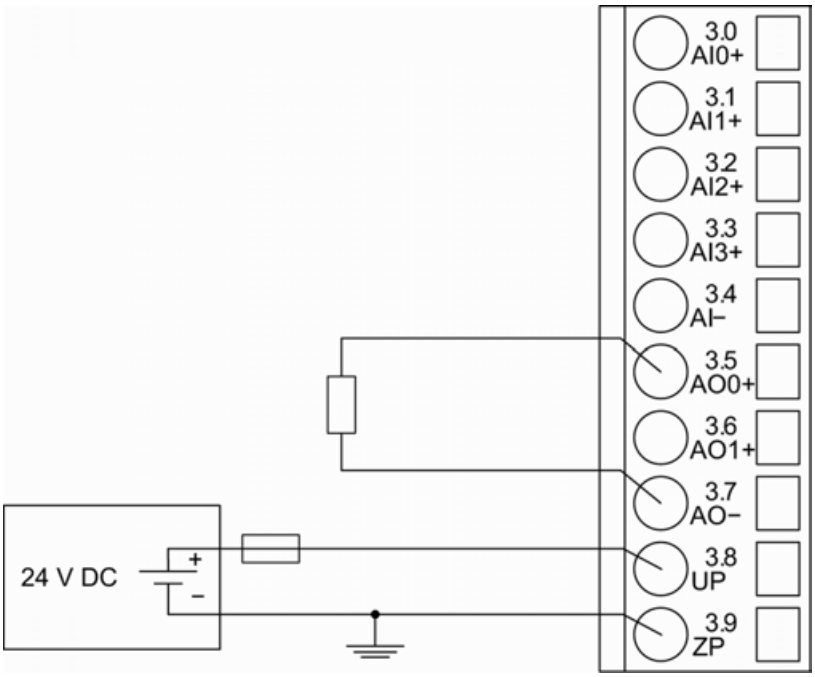


Fig. 916: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

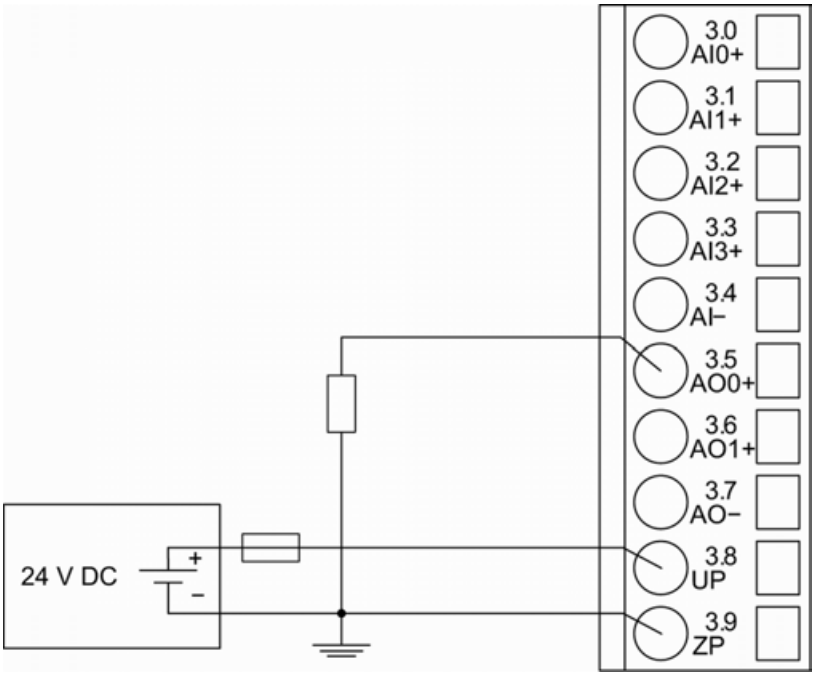


Fig. 917: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.6.3.1.2.6 “Parameterization” on page 4601* ↳ *Chapter 1.6.2.6.3.1.2.9 “Measuring ranges” on page 4608*:

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

For a description of the function of the LEDs, please refer to Diagnosis and displays / Displays ↳ *Chapter 1.6.2.6.3.1.2.8 “State LEDs” on page 4608*.

Unused analog outputs can be left open-circuited.

Internal data exchange

	Without the fast counter	**With the fast counter (only with AC500)**
Digital inputs (bytes)	1	1
Digital outputs (bytes)	3	3
Analog inputs (words)	4	4
Analog outputs (words)	2	2
Counter input data (words)	0	5
Counter output data (words)	0	9

I/O configuration

The module itself does not store configuration data. It draws its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Module ID [1])	Internal	1815	WORD	1815	0x0Y01
Ignore module	Internal	Yes No	BYTE	No	
Parameter length	Internal	8	BYTE	8	0xY02
Check supply	off	0	BYTE	1	0xY03
	on	1			
Fast counter [3])	0 : 10 [2])	0 : 10	BYTE	0	Not for FBP
Behavior outputs at comm. error [5])	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	Off 0x00	0x0Y07

[2])	Setting	Description
	On	Error LED lights up at errors of all error classes, Failsafe mode off
	Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
	Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe mode off

[2])	Setting	Description
	On +Failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)
	Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
	Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission

[2]) For a description of the counter operating modes, please refer to the 'Fast Counter' section
↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

[3]) With CS31 without the parameter "Fast Counter"

The fast counter of the module does not work if the module is connected to a CS31 bus module.

[5]) The parameter Behavior outputs at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0x0Y05
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01	0x0Y06
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000	0x0Y08

*) The parameters Behavior DO at comm. error is only analyzed if the Failsafe mode is ON.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Analog data format	Standard Reserved	0 255	BYTE	0	0x0Y04

*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
Input 0, Channel configuration	see *Table 475 “ Channel configuration” on page 4603*	see *Table 475 “ Channel configuration” on page 4603*	BYTE	0	0x0Y09
Input 0, Check channel	see *Table 476 “ Channel monitoring” on page 4604*	see *Table 476 “ Channel monitoring” on page 4604*	BYTE	0	0x0Y0A
:	:	:	:	:	
:	:	:	:	:	
Input 3, Channel configuration	see *Table 475 “ Channel configuration” on page 4603*	see *Table 475 “ Channel configuration” on page 4603*	BYTE	0	0x0Y0F
Input 3, Check channel	see *Table 476 “ Channel monitoring” on page 4604*	see *Table 476 “ Channel monitoring” on page 4604*	BYTE	0	0x0Y10

Table 475: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 476: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default	EDS Slot / Index
0 Output 0, Channel configuration	see *Table 477 “Channel configuration” on page 4605*	see *Table 477 “Channel configuration” on page 4605*	BYTE	0	0x0Y11
Output 0, Check channel	see *Table 478 “Channel monitoring” on page 4605*	see *Table 478 “Channel monitoring” on page 4605*	BYTE	0	0x0Y12
Output 0, Substitute value	see *Table 479 “Substitute value” on page 4605*	see *Table 479 “Substitute value” on page 4605*	WORD	0	0x0Y13
Output 1, Channel configuration	see *Table 477 “Channel configuration” on page 4605*	see *Table 477 “Channel configuration” on page 4605*	BYTE	0	0x0Y14
Output 1, Check channel	see *Table 478 “Channel monitoring” on page 4605*	see *Table 478 “Channel monitoring” on page 4605*	BYTE	0	0x0Y15
Output 1, Substitute value	see *Table 479 “Substitute value” on page 4605*	see *Table 479 “Substitute value” on page 4605*	WORD	0	0x0Y16

Table 477: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 478: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 479: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behavior of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 s	0
Last value for 10 s and then turn off	Last value 10 s	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 s	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 s	Depending on configuration

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

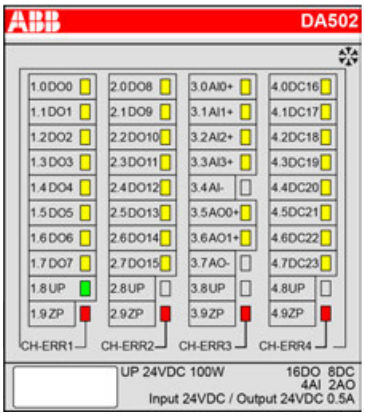
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	14	1...10	31	31	19	Checksum error in the I/O module		Replace I/O module
	11 / 12	ADR	1...10					
3	14	1...10	31	31	3	Timeout in the I/O module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	40	Different hard-/firmware versions in the module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	43	Internal error in the module		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	36	Internal data exchange failure		
	11 / 12	ADR	1...10					
3	14	1...10	31	31	9	Overflow diagnosis buffer		New start
	11 / 12	ADR	1...10					
3	14	1...10	31	31	26	Parameter error		Check master
	11 / 12	ADR	1...10					
3	14	1...10	31	31	11	Process voltage too low		Check process voltage
	11 / 12	ADR	1...10					
4	14	1...10	31	31	45	Process voltage is switched off (ON −> OFF)		Process voltage ON
	11 / 12	ADR	1...10					
Channel error DA502								
4	14	1...10	2	0...15	47	Short-circuit at a digital output		Check connection
	11 / 12	ADR	1...10	22...29 [5])				
Channel error DA502								
4	14	1...10	1	16...19 [6])	48	Analog value overflow or broken wire at an analog input		Check input value or terminal
	11 / 12	ADR	1...10					
4	14	1...10	1	16...19 [6])	7	Analog value underflow at an analog input		Check input value
	11 / 12	ADR	1...10					
4	14	1...10	1	16...19 [6])	47	Short circuit at an analog input		Check ter-minal
	11 / 12	ADR	1...10					
4	14	1...10	3	20...21 [7])	4	Analog value overflow at an analog output		Check output value
	11 / 12	ADR	1...10					

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error Identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
4	14	1...10	3	20...21 [7])	7	Analog value underflow at an analog output		Check output value
	11 / 12	ADR	1...10					

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = communication interface module 1...10, ADR = hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus: 31 = Module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus = module type (1 = AI, 3 = AO, 4 = DC); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = module itself" is output.
[5])	Ch = 22...29 indicate the digital inputs/outputs DC16...DC23
[6])	Ch = 16...19 indicates the analog inputs AI0...AI3
[7])	Ch = 20...21 indicates the analog outputs AO0...AO1

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	DO0 to DO15	Digital output	Yellow	Output is OFF	Output is ON	--
	DC16 to DC23	Digital input/ output	Yellow	Input/output is OFF	Input/output is ON [1])	--
	AI0 to AI3	Analog input	Yellow	Input is OFF	Input is ON [2])	--
	AO0 to AO1	Analog output	Yellow	Output is OFF	Output is ON [2])	--
	UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
	CH-ERR1	Channel error, error messages in groups (digital inputs/ outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Severe error within the corresponding group (e.g. short circuit at an output)
	CH-ERR2		Red			
	CH-ERR3		Red			
	CH-ERR4		Red			
	CH-ERR [3])	Module error	Red	--	Internal error	--
	[1]) Indication LED is ON even if an input signal is applied to the channel and the supply voltage is off. In this case the module is not operating and does not generate an input signal.					
	[2]) Brightness depends on the value of the analog signal					
	[3]) All of the LEDs CH-ERR1 to CH-ERR4 light up together					

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	> 11.7589	> 11.7589	> 23.5178	> 22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	< 0.0000	< -11.7589	< 0.0000	< 0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	: : 70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

Technical data of the module

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for UP (+24 V DC) and 1.9, 2.9, 3.9 and 4.9 for ZP (0 V)
	Protection against reverse voltage	yes
	Rated protection fuse at UP	10 A fast
	Rated value	24 V DC
	Max. ripple	5 %
Current consumption		
	From UP	0.07 A + max. 0.5 A per output
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/communication interface module	ca. 2 mA
	Inrush current from UP (at power-up)	0.04 A^2s
Galvanic isolation		Yes, per module
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal mounting or vertical with derating (output load reduced to 50% at 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital outputs

Parameter		Value
Number of channels per module		16 outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
Connection of the channels		
	DO0 to DO7	Terminals 1.0 to 1.7
	DO8 to DO15	Terminals 2.0 to 2.7

Parameter		Value
Indication of the output signals		1 yellow LED per channel, the LED is ON if the output signal is high (signal 1)
Monitoring point of output indicator		LED is controlled by process CPU
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (channels O0 to O15)	4 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable digital I/O channels can be defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC16...DC23	Terminals 4.0...4.7
If the channels are used as outputs		
	Channels DC16...DC23	Terminals 4.0...4.7

Parameter	Value
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Monitoring point of input/output indicator	LED is part of the input circuitry
Galvanic isolation	Yes, per module

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all inputs		Terminals 1.9...4.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Monitoring point of input/output indicator		LED is part of the input circuitry
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

* Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC16 to DC23		Terminals 4.0 to 4.7
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	rated value per channel	500 mA at UP = 24 V
	max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
Fuse for UP		10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

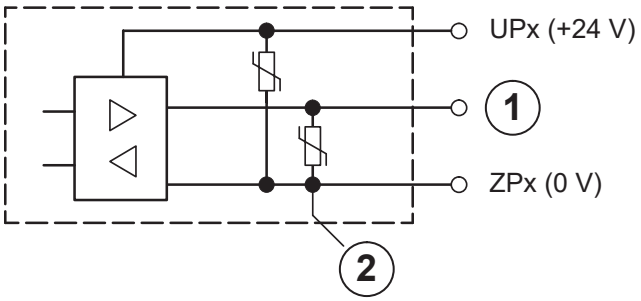


Fig. 918: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

> *The fast counter of the module does not work if the module is connected to a CS31 bus module.*

Parameter	Value
Counting frequency	Max. 50 kHz

Chapter 1.6.4.1.10 “Fast counters” on page 5498

Chapter 1.6.4.4.2.2 “Operating modes” on page 5716

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for AI0+ to AI3+		Terminal 3.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 % For XC version below 0 °C and above 60 °C: on request

Parameter	Value
Relationship between input signal and hex code	↳ *Chapter 1.6.2.6.3.1.2.9.1 “Input ranges voltage, current and digital input” on page 4608* ↳ *Chapter 1.6.2.6.3.1.2.9.2 “Input ranges resistance temperature detector” on page 4609*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 3.5 and 3.6
Reference potential for AO0+ to AO1+		Terminal 3.7 (AO-) for voltage output Terminals 1.9, 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules

Parameter	Value
Configurability	-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output	0 Ω...500 Ω
Output loadability, as voltage output	±10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	*Chapter 1.6.2.6.3.1.2.9.3 “Output ranges voltage and current” on page 4610*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 250 800 R0001	DA502, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO	Active
1SAP 450 800 R0001	DA502-XC, digital/analog input/output module, 16 DO, 8 DC, 4 AI, 2 AO, XC version	Active

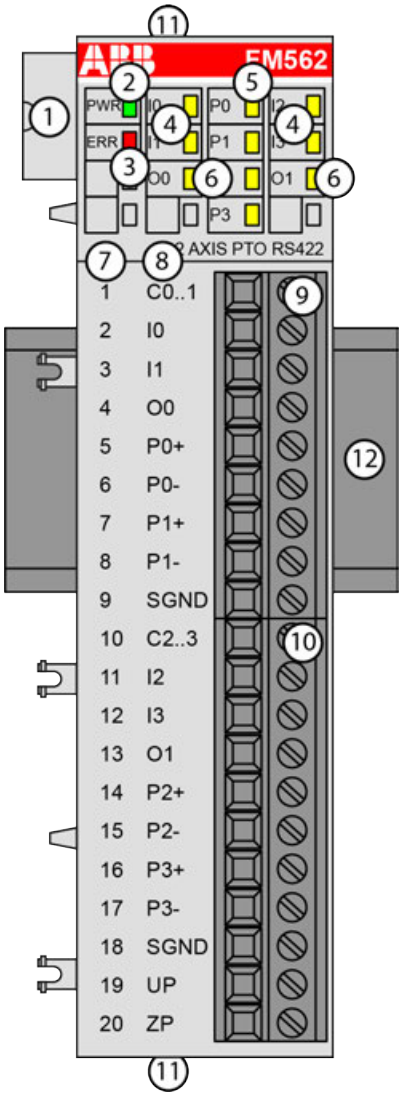
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.7 Function modules

1.6.2.7.1 S500-eCo

FM562 for pulse train output

- 2 axes motion control
- 2 pulse train outputs per axis, RS-422
- 2 configurable digital inputs per axis, 24 V DC
- 32 bits registers for current position, registered position and speed value
- Group-wise galvanically isolated



1 I/O bus
2 1 green LED to display power supply
3 1 red LED to display error
4 4 yellow LEDs to display the signal states of the inputs I0 to I3
5 4 yellow LEDs to display the signal states of the pulse train outputs P0 to P3
6 2 yellow LEDs to display the signal states of O0 to O1 (reserved)
7 Terminal number
8 Allocation of signal name
9 Terminal block for axis signals (9-pin)
10 Terminal block for axis signals and process supply voltage (11-pin)
11 2 holes for wall-mounting with screws
12 DIN rail

Intended purpose

The function module FM562 for pulse train output (PTO) is used for simple positioning tasks with servo drives or stepper drives. FM562 provides 2 axes with 2 inputs and 2 pulse-train outputs each.

It can be used at the following devices:

- Communication interface modules (e. g. CI501-PNIO, CI541-DP)
- Processor modules

It contains the following features:

- 2 axes control
- 2 configurable discrete digital inputs per axis for enable and limit switches signal inputs
- PTO output type: RS-422 differential output (P0, P1, P2 and P3)
- PTO frequency: 10 Hz to 250kHz
- Configurable PTO output mode: CW/CCW (clockwise/counterclockwise), pulse/direction
- Position and speed control with built in motion profile generators. Integration in the application program by PLCopen Motion Control function blocks (PS552-MC-E motion control library is required for programming)

The pulse outputs of the 2 axes are not galvanically isolated from each other.

The other circuitry of the module is galvanically isolated from the inputs/outputs.

Connections

The pulse-train output module FM562 can be connected to the following devices via the I/O bus connector:

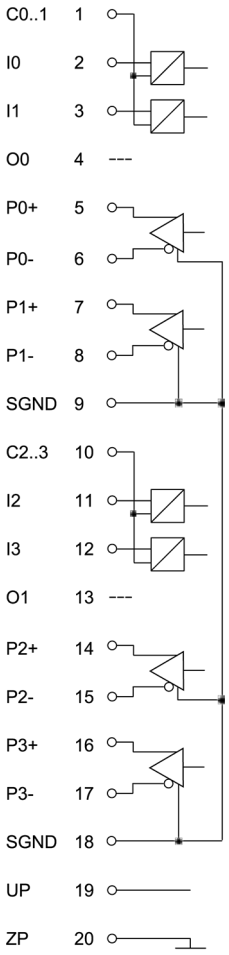
- S500 PROFIBUS and PROFINET communication interface module (e. g. CI501-PNIO, CI541-DP)
- AC500 CPUs
- Other AC500 I/O modules

The I/O module must not be used as a decentralized I/O module with CI590-CS31-HA communication interface modules.

The module must not be used as a communication interface module at CI58x-CN or CI59x-CS31.

The connection is carried out by using removable 9-pin and 11-pin terminal blocks. These terminal blocks differ in their connection system (spring terminals or screw terminals, cable mounting from the front or from the side). For more information, please refer to the chapter terminal blocks for S500-eCo I/O modules ↳ *Chapter 1.6.2.9.3.1 “TA563-TA565 - Terminal blocks” on page 5204*. The terminal blocks are not included in the module's scope of delivery and must be ordered separately.

The following block diagram shows the internal construction of the digital inputs and outputs:



The 2 SGND signals are internally interconnected.

The assignment of the terminals:

Terminal	Signal	Description
1	C0..1	Input common for signals I0 and I1
2	I0	Input signal I0 (axis enable and limit switch)
3	I1	Input signal I1 (stop)
4	O0	Reserved - do not connect
5	P0+	Pulse output P0+ (positive line)
6	P0-	Pulse output P0- (negative line)
7	P1+	Pulse or direction output P1+ (positive line)
8	P1-	Pulse or direction output P1- (negative line)
9	SGND	Signal ground for pulse output
10	C2..3	Input common for signals I2 and I3
11	I2	Input signal I2 (axis enable and limit switch)

Terminal	Signal	Description
12	I3	Input signal I3 (stop)
13	O1	Reserved - do not connect
14	P2+	Pulse output P2+ (positive line)
15	P2-	Pulse output P2- (negative line)
16	P3+	Pulse or direction output P3+ (positive line)
17	P3-	Pulse or direction output P3- (negative line)
18	SGND	Signal ground for pulse output
19	UP	Process voltage UP +24 V DC
20	ZP	Process voltage ZP 0 V DC

When wiring, the motor phase line and power line should be separated in order to avoid signal disturbances between each other.

For cable length ≤ 30 m, unshielded cable can be used with Baldor and BSD servo drives normally.

For cable length > 30 m, shielded cable must be used for surge purpose.

The earthing of the shield should take place at the switchgear cabinet, see chapter System Data AC500 ↳ Chapter 1.6.3.6.1 “System data AC500” on page 5313.

The cable shields must be earthed at both ends of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a CPU). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 5 mA per FM562.

The external power supply connection is carried out via the UP (+24 V DC) and ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of damaging the PLC modules!

Never connect any voltages or signals to reserved terminals (marked with --- or O0 / O1). Reserved terminals may carry internal voltages.

Be sure to connect the pulse output signals in the right order. Otherwise, the pulse number may be wrongly calculated and malfunctions may appear.

The module provides several diagnosis functions (see Diagnosis ↳ *Chapter 1.6.2.7.1.1.6 “Diagnosis” on page 4630*).

The digital inputs can be used as source inputs or as sink inputs.

NOTICE!

Risk of malfunctions in the plant!

A ground fault, e. g. caused by a damaged cable insulation, can bridge switches accidentally.

Use sink inputs when possible or make sure that, in case of error, there will be no risks to persons or plant.

The following figure shows the connection of the inputs to the pulse-train output module FM562:

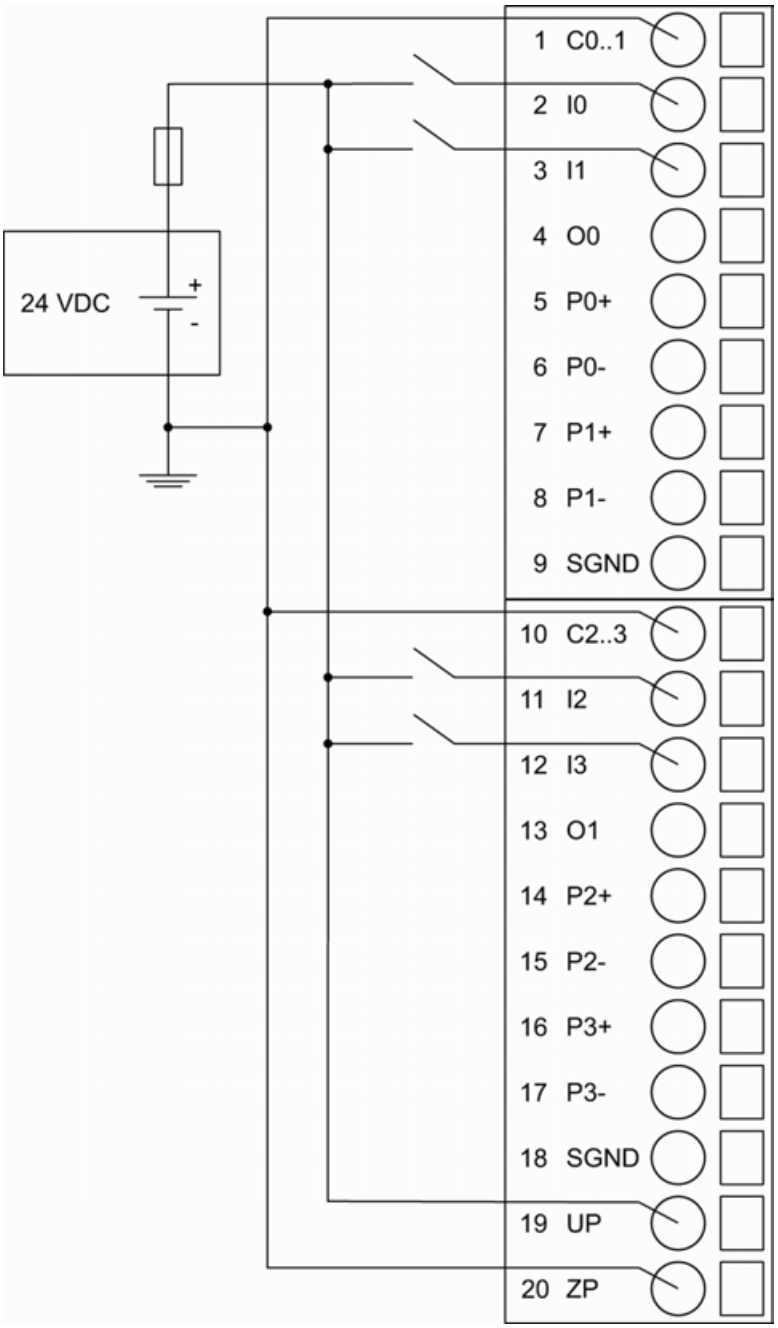


Fig. 919: Connection of inputs to the FM562 - sink inputs

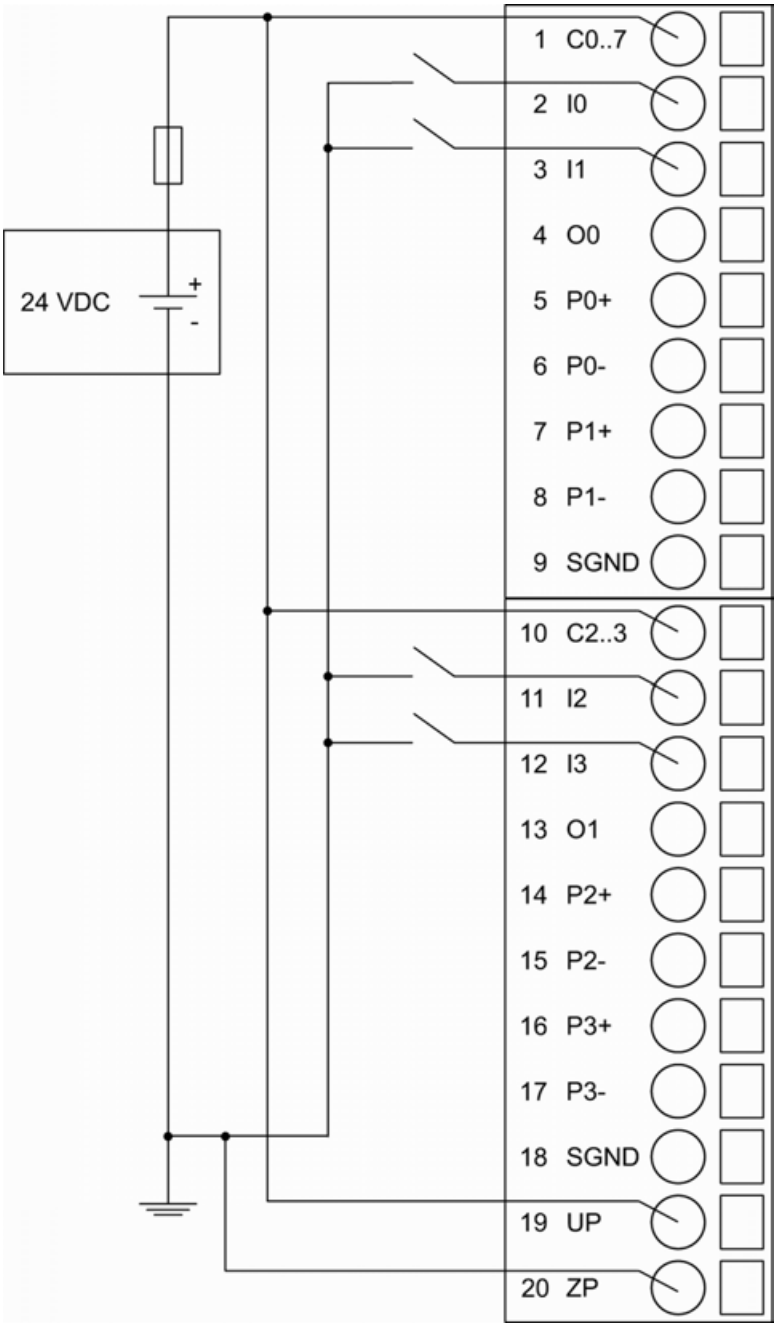


Fig. 920: Connection of inputs to the FM562 - source inputs

The following figure shows the connection of the pulse-train outputs of the FM562 to a servo amplifier:

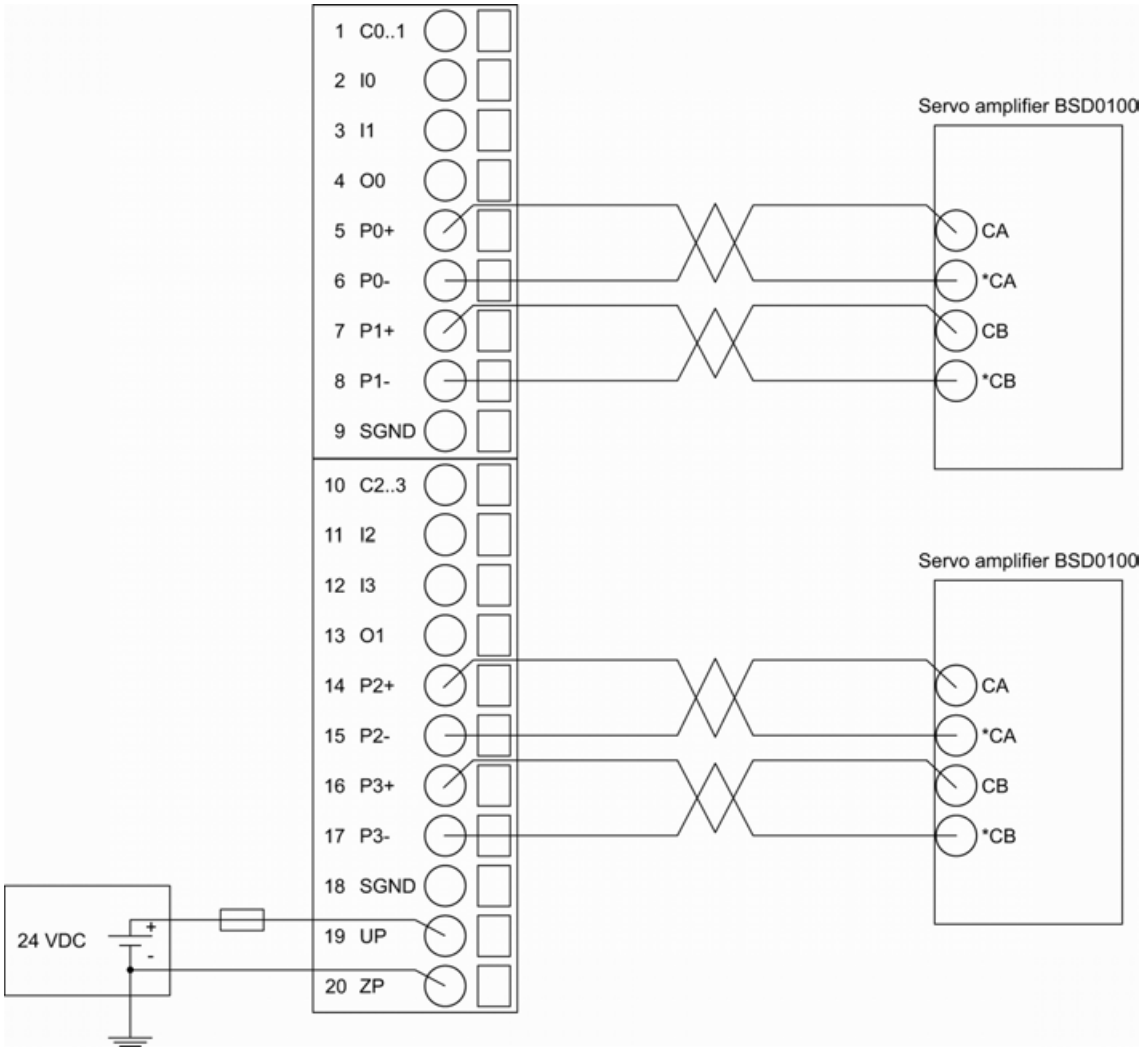


Fig. 921: Connection (differential) of pulse train output

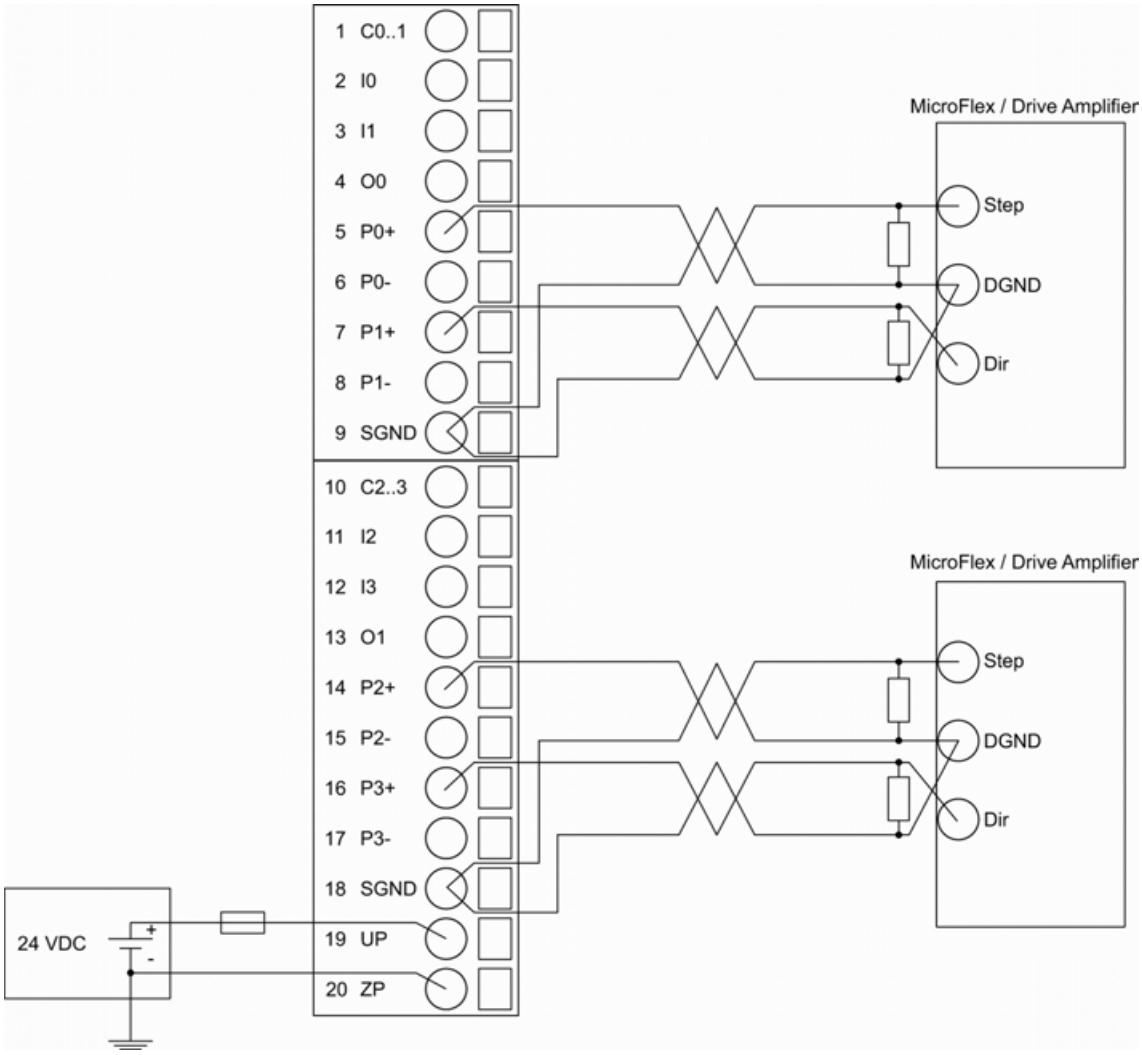


Fig. 922: Connection (single-ended) of pulse train output

For drives/amplifiers with high-impedance pulse input interface like MicroFlex, the cable ends must be equipped with 100 Ω terminating resistors to eliminate signal reflections. Normally, the resistors are integrated in the interface connectors.

Internal data exchange

Parameter	Value
Axes input data (words)	16
Axes output data (words)	16

I/O configuration

The pulse-train output module FM562 does not store configuration data itself.

Parameterization

The arrangement of the parameter data is performed with Automation Builder.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

For programming, the library package PS552-MC-E is required. This library package is not part of Automation Builder and has to be purchased separately.

Module parameters

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Module ID	Internal	1830	WORD	0x0726	0	65535
Ignore module	No Yes	0 1	BYTE	No 0x00		
Parameter length	Internal	19	BYTE	19	0	255
Check Supply	Off On	0 1	BYTE	On 0x01	0	255

Input channels for axis 1

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Input 0, channel configuration	No function Axis enable / limit switch	0 1	BYTE	No function 0x00	0	1
Input 0, input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0	3
Input 1, channel configuration	No function Stop Registration *)	0 1 2	BYTE	No function 0x00	0	2
Input 1, input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0	3

*) Reserved - do not use

Output channel for axis 1

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Output 0, channel configuration	No function	0	BYTE	No function 0x00	0	2

Slot parameters for axis 1

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Output mode	CW/CCW Pulse/Direction	0 1	BYTE	CW/CCW 0x00	0	1
Start frequency *)	0...65535	0...65535	WORD	0 0x00	0	65535

*) Unit is Hz

Input channels for axis 2

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Input 2, channel configuration	No function Axis enable / limit switch	0 1	BYTE	No function 0x00	0	1
Input 2, input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1ms 0x00	0	3
Input 3, channel configuration	No function Stop Registration *)	0 1 2	BYTE	No function 0x00	0	2
Input 3, input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00	0	3

*) Reserved - do not use

Output channel for axis 2

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Output 1, channel configuration	No function	0	BYTE	No function 0x00	0	2

Slot parameters for axis 2

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Output mode	CW/CCW Pulse/Direction	0 1	BYTE	CW/CCW 0x00	0	1
Start frequency *)	0...65535	0...65535	WORD	0 0x00	0	65535

*) Unit is Hz

GSD file:

Ext_User_Prm_Data_Len =	0x17
Ext_User_Prm_Data_Const(0) =	0x07, 0x27, 0x00, 0x13, 0x01\ 0x00, 0x00, 0x00, 0x00, 0x00\ 0x00, 0x00, 0x00, 0x00, 0x00\ 0x00, 0x00, 0x00, 0x00, 0x00\ 0x00, 0x00, 0x00;

Diagnosis

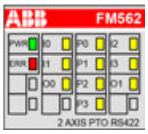
E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
			Module error FM562				
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	43	Internal error in the module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	3	Timeout inside the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	Restart
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
4	14	1...10	31	31	45	Process voltage is switched off (ON => OFF)	Process voltage ON
	11 / 12	ADR	1...10				

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The PNIO diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = decentralized communication interface module 1..10, ADR = hardware address (e. g. of the DC551-CS31)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or PNIO: 31 = module itself; COM1/COM2: 1..10 = expansion 1..10 Channel error: I/O bus or PNIO = module type (2 = DO); COM1/COM2: 1..10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

LED		State	Color	LED = OFF	LED = ON	LED flashes
	PWR	Process voltage 24 V DC via terminal and process voltage via I/O bus	Green	CPU module voltage or external 24 V DC supply voltage is missing	I/O bus voltage and external 24 V DC supply voltage are present (LED is on after startup of the module (approx. 1 s))	---
	ERR	Channel or module error	Red	No error or process voltage is missing	Severe error in the module	Axis related error
	P0...P3	Pulse output	Yellow	Output = OFF	Output = ON	LED follows the state of the outputs, depending on frequency
	I0...I3	Digital Input	Yellow	Input = OFF	Input = ON	---
	O0...O1	Reserved	Yellow	---	---	---

Technical data

The System Data of AC500-eCo apply ↳ *Chapter 1.6.3.5.1 “System data AC500-eCo” on page 5233*

Only additional details are therefore documented below.

Parameter	Value
Digital inputs	4 inputs (2 per axis) 24 V DC, can be used as source inputs or as sink inputs
Input channels 0 and 2	Input signal used for axis enable and limit switch
Input channels 1 and 3	Stop, configurable
Input data length	32 bytes

Parameter	Value
Pulse outputs	Pulse specification ● 2 outputs for each axis, configurable ● Type: RS-422 differential signal ● Mode: CW & CCW or Pulse & Direction ● Frequency: 10 Hz to 250 kHz ● Pulse number: -2147483648 to 2147483647 (32 bits) ● Motion profiles generator
Output data lenth	32 bytes
LED displays	For power supply, errors and signal states
Internal power supply	Via I/O bus
External power supply	Via the terminals ZP and UP (process voltage 24 V DC)

Process supply voltage UP	Value
Connections	Terminal 19 for UP (+24 V DC) and terminal 20 for ZP (0 V)
Rated value	24 V DC
Current consumption via UP terminal	42 mA
Max. ripple	5 %
Inrush current from UP (at power up)	0.067 A^2s
Protection against reversed voltage	Yes
Rated protection fuse for UP	Not necessary
Current consumption from 24 V DC power supply at the L+/UP and M/ZP terminals of the CPU/communication interface module	Ca. 5 mA
Galvanic isolation	Yes, between input groups and the output group and the rest of the module
Isolated groups	5 groups (2 groups for 4 input channels, 1 group for 4 pulse train output channels, 1 group for process supply voltage, 1 group for the rest of the module)
Surge-voltage (max.)	35 V DC for 0.5 s
Max. power dissipation within the module	1.2 W
Weight	Ca. 125 g
Mounting position	Horizontal or vertical
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

No effects of multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an external fuse.

Technical data of the digital inputs

Parameter		Value	
Number of channels per module		4	
Distribution of the channels into axes		1 group of 2 channels for each axis	
	Axis 1	Inputs I0...I1	
	Axis 2	Inputs I2...I3	
Connections of the channels I0 to I1		Terminals 2 to 3	
Connections of the channels I1 to I3		Terminals 11 to 12	
Reference potential for the channels I0 to I1		Terminal 1 (Signal name C0..1)	
Reference potential for the channels I2 to I3		Terminal 10 (Signal name C2..3)	
Galvanic isolation		Yes, per axis	
Indication of the input signals		1 yellow LED per channel; the LED is ON when the input signal is high (signal 1)	
Input type according to EN 61131-2		Type 1 source	Type 1 sink
Input signal range		-24 V DC	+24 V DC
	Signal 0	-5 V...+3 V	-3 V...+5 V
	Undefined signal	-15 V...+ 5 V	+5 V...+15 V
	Signal 1	-30 V...-15 V	+15 V...+30 V
Ripple with signal 0		-5 V...+3 V	-3 V...+5 V
Ripple with signal 1		-30 V...-15 V	+15 V...+30 V
Input current per channel			
	Input voltage +24 V	Typ. 5 mA	
	Input voltage +5 V	Typ. 1 mA	
	Input voltage +15 V	> 2.5 mA	
	Input voltage +30 V	< 8 mA	
Max. permissible leakage current (at 2-wire proximity switches)		1 mA	
Input delay (0->1 or 1->0)		Typ. 0.1 to 32 ms (configurable via software), default: 0.1 ms	
Max. cable length			
	Shielded	500 m	
	Unshielded	300 m	

Technical data of the pulse outputs

Parameter	Value
Number of channels	2 per axis, 4 per module
Output type	RS-422
Output mode	Clockwise and counter-clockwise or pulse and direction
Output frequency	10 Hz to 250 kHz

Parameter		Value
Frequency accuracy		
	From 10 Hz to 500 Hz	± 2 %
	From 501 Hz to 250 kHz	± 1 %
Differential output voltage (at terminal block)		2.8 V at 140 Ω differential load 2.56 V at 100 Ω differential load
Output voltage of positive output (P0+, P1+) referenced to SGND if used for single ended application		Max. 3.3 V without any load Typ. 2.5 V at 100 Ω load
Max. short circuit current		40 mA
Max. cable length		
	Shielded	300 m (at max. frequency, criterion: V ≥ 2 V, tested with 100 Ω termination)
	Unshielded	30 m

Ordering data

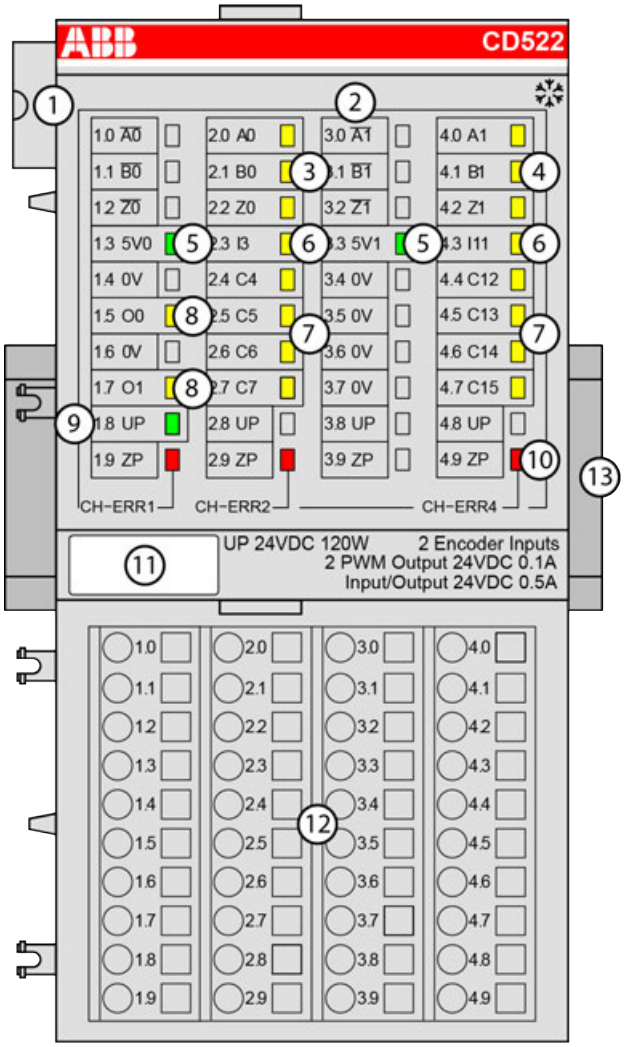
Part no.	Description	Product life cycle phase *)
1SAP 233 100 R0001	FM562, pulse-train output module, 2 axes, RS-422, 4 DI, 24 V DC	Active
1TNE 968 901 R3101	Terminal block TA563-9, 9 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal block TA563-11, 11 pins, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal block TA564-9, 9 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal block TA564-11, 11 pins, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal block TA565-9, 9 pins, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal block TA565-11, 11 pins, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.7.2 S500

CD522 - Encoder, counter and PWM module

- 2 encoder inputs with 2 integrated 5-V-power-supplies for the encoders
- 2 PWM outputs - 2 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC
- Fast counter
- Module-wise galvanically isolated
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation of terminal No. and signal name
3 3 yellow LEDs to display the signal states of the encoder 0 input
4 3 yellow LEDs to display the signal states of the encoder 1 input
5 2 green LEDs to display the 5-V-power-supply states
6 2 yellow LEDs to display the signal state of the digital input I3 and I11
7 8 yellow LEDs to display the input/output signal states
8 2 yellow LEDs to display the signal states of the PWM/pulse outputs
9 1 green LED to display the process voltage UP
10 3 red LEDs to display errors
11 Label
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The encoder and PWM module CD522 can be used at the following devices:

- Communication interface modules (e. g. CI501-PNIO, CI541-DP)
- Processor modules

Features:

- 2 independent counting functions with up to 12 configurable modes (including incremental position encoder and frequency input up to 300 kHz)
- 2 independent PWM (pulse-width modulator) or pulse outputs with push-pull driver
- Dedicated inputs/outputs for specific counting functions (e.g. touch, set, reset)
- All unused inputs/outputs can be used with the specifications of standard inputs/outputs range

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Depending on the configuration used, some inputs and outputs are dedicated to specific counting functions (touch, set, reset…). All unused inputs and outputs can be used with the specification of standard inputs/outputs range.

Functionality

Digital inputs/outputs	24 V DC, dedicated inputs/outputs can be used for specific counting functions: - Catch/touch operation, counter value stored in separate variable on external event (rising or falling edge) - Set input to preset counter register with predefined value - Set input to reset counter register - End value output; the output is set when predefined value is reached - Reference point initialization (RPI) input for incremental encoder initialization All unused inputs/outputs can be used with the specification of standard input/output range. Effect of incorrect input terminal connection: Wrong or no signal detected, no damage up to 35 V.
Fast counter/encoder	integrated, 2 counters (hardware interface with +24 V DC, +5 V DC, differential and 1 Vpp sinus input) with up to 12 configurable operation modes: - 32 bits one counter mode - 16 bits two counter mode - Incremental position encoder - Absolute SSI encoder - Time frequency meter - Frequency input up to 300 kHz

PWM/pulse outputs	2 pulse-width-modulators or pulse outputs Output specification - Push-pull output: 24 V DC, 100 mA max. - Current limitation (thermal and over current) PWM specification - Frequency from 1 Hz to 100 kHz - Value from 0 to 100 % Pulse specification - Frequency from 1 Hz to 15 kHz - Pulse emission from 1 to 65535 pulses - Number of pulses emitted indicator (0 to 100 %) Frequency specification - Frequency output = 100 kHz when duty cycle set to 50 %
Power supply for encoders	2 5V power supplies, max. 100 mA
LED displays	For signal states, errors and supply voltage
Internal power supply	Via I/O bus
External power supply	Via the terminals UP (process voltage 24 V DC) and ZP (0 V DC)
Required terminal unit	TU515 or TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*

Connections

The function module CD522 can be connected to the following devices via the I/O bus connector:

- CS31 bus module DC551-CS31
- AC500 CPU
- OtherAC500 I/O devices.

The connection is carried out by using the 40 terminals of the terminal unit TU515/TU516 ↬ *Chapter 1.6.2.5.3 “TU515, TU516, TU541 and TU542 for I/O modules” on page 4103*.

Table 480: Assignment of the terminals

Terminal	Signal	Description
1.0	/A0	Inverted input signal A of encoder 0
1.1	/B0	Inverted input signal B of encoder 0
1.2	/Z0	Inverted input signal Z of encoder 0
1.3	5V0	+5 V DC power supply output 0 for sensors
1.4	0V	0 V reference input
1.5	O0	Output signal of the fast output O0
1.6	0V	0 V reference input
1.7	O1	Output signal of the fast output O1
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)

Terminal	Signal	Description
2.0	A0	Input signal A of encoder 0
2.1	B0	Input signal B of encoder 0
2.2	Z0	Input signal Z of encoder 0
2.3	I3	Input signal I3 (standard input)
2.4...2.7	C4...C7	Signal of the configurable digital input/output C4...C7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	/A1	Inverted input signal A of encoder 1
3.1	/B1	Inverted input signal B of encoder 1
3.2	/Z1	Inverted input signal Z of encoder 1
3.3	5V1	+5 V DC power supply output 1 for sensors
3.4...3.7	0V	0 V reference input
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	A1	Input signal A of encoder 1
4.1	B1	Input signal B of encoder 1
4.2	Z1	Input signal Z of encoder 1
4.3	I11	Input signal I11 (standard input)
4.4...4.7	C12...C15	Signal of the configurable digital input/output C12...C15
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

The internal power supply voltage for the module's circuitry is carried out via the I/O bus (provided by a communication interface module or a processor module). Thus, the current consumption from 24 V DC power supply at the terminals L+/UP and M/ZP of the CPU/communication interface module increases by 2 mA per CD522.

The external power supply connection is carried out via the UP (+24 V DC) and the ZP (0 V DC) terminals.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

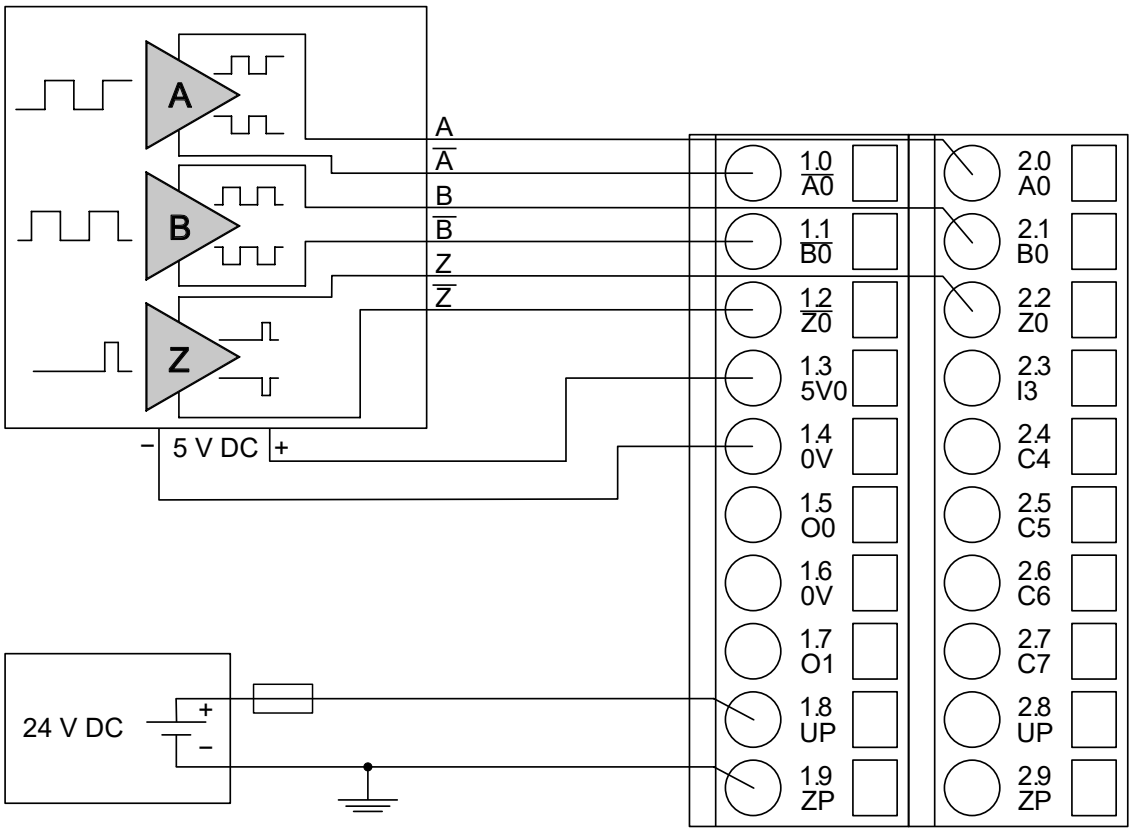
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

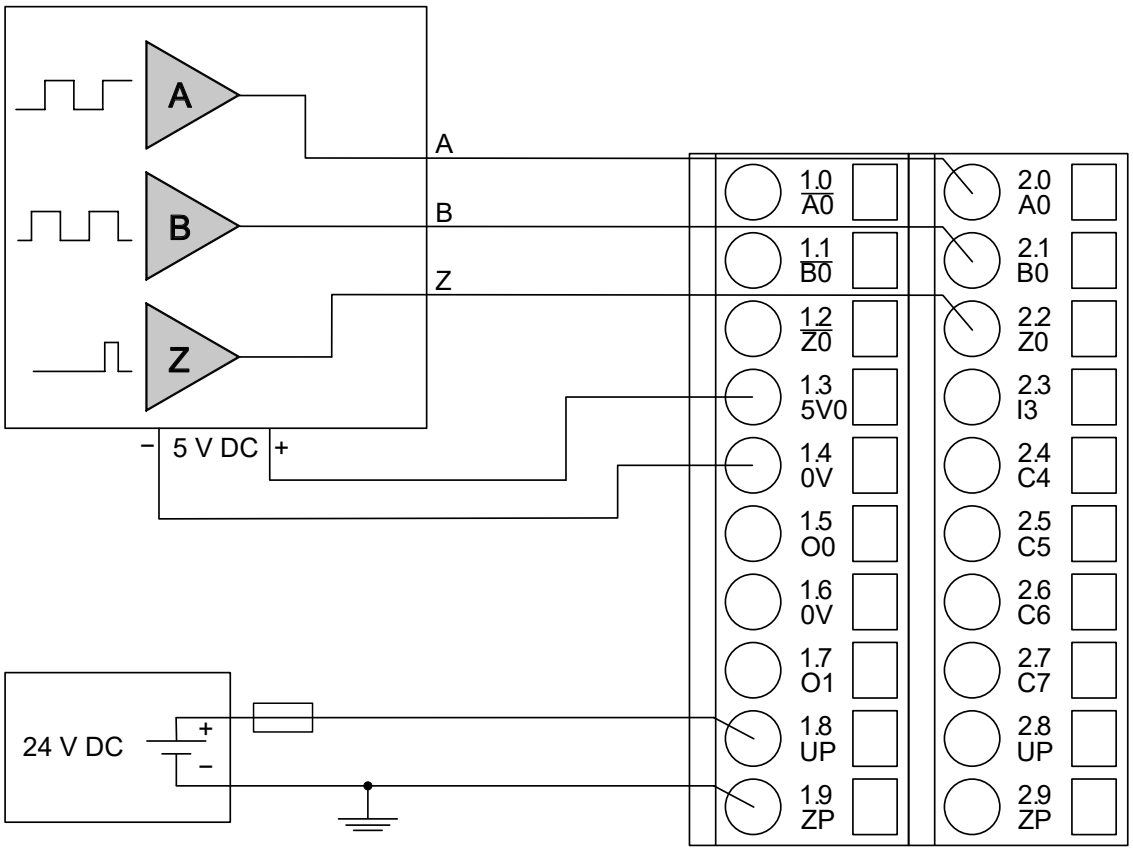
Connection of encoders with differential RS-422 signal

The encoder is powered by the 5 V power supply which is integrated in CD522.

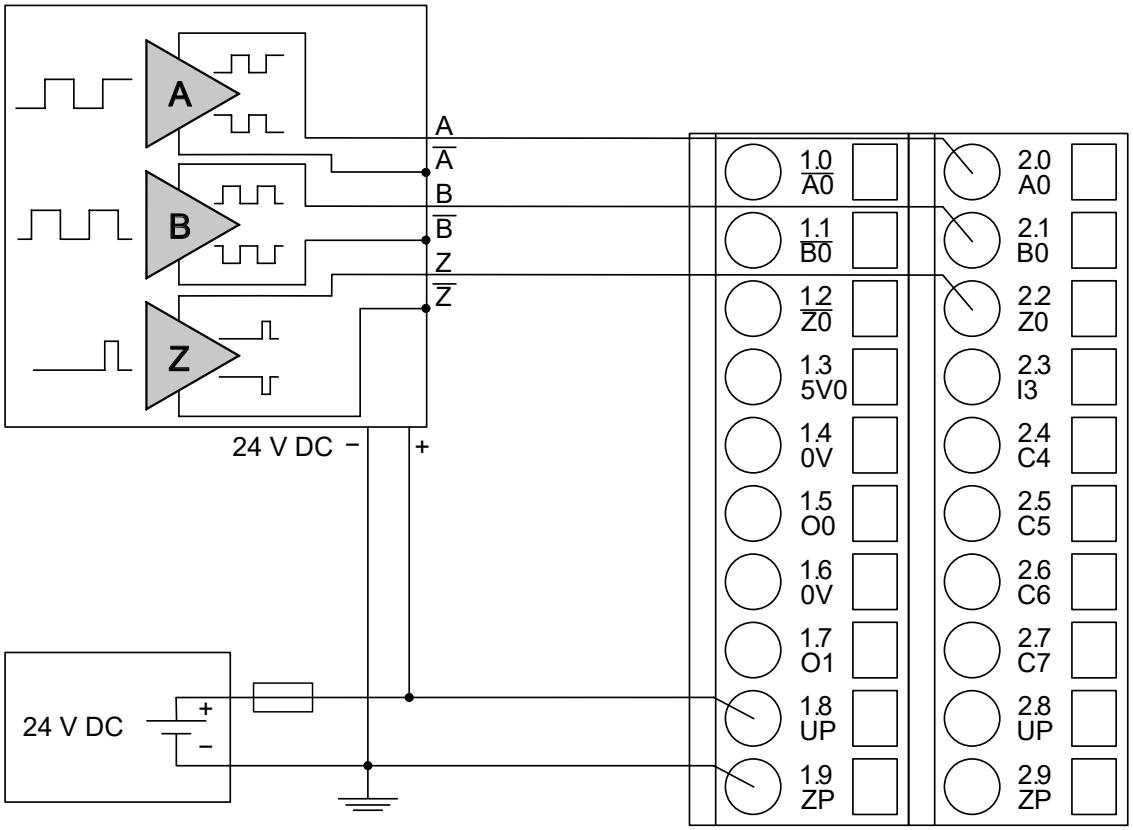


Connection of encoders with 5 V TTL signal

The encoder is powered by the 5 V power supply which is integrated in the CD522.



Connection of encoders with 24 V totem pole signal

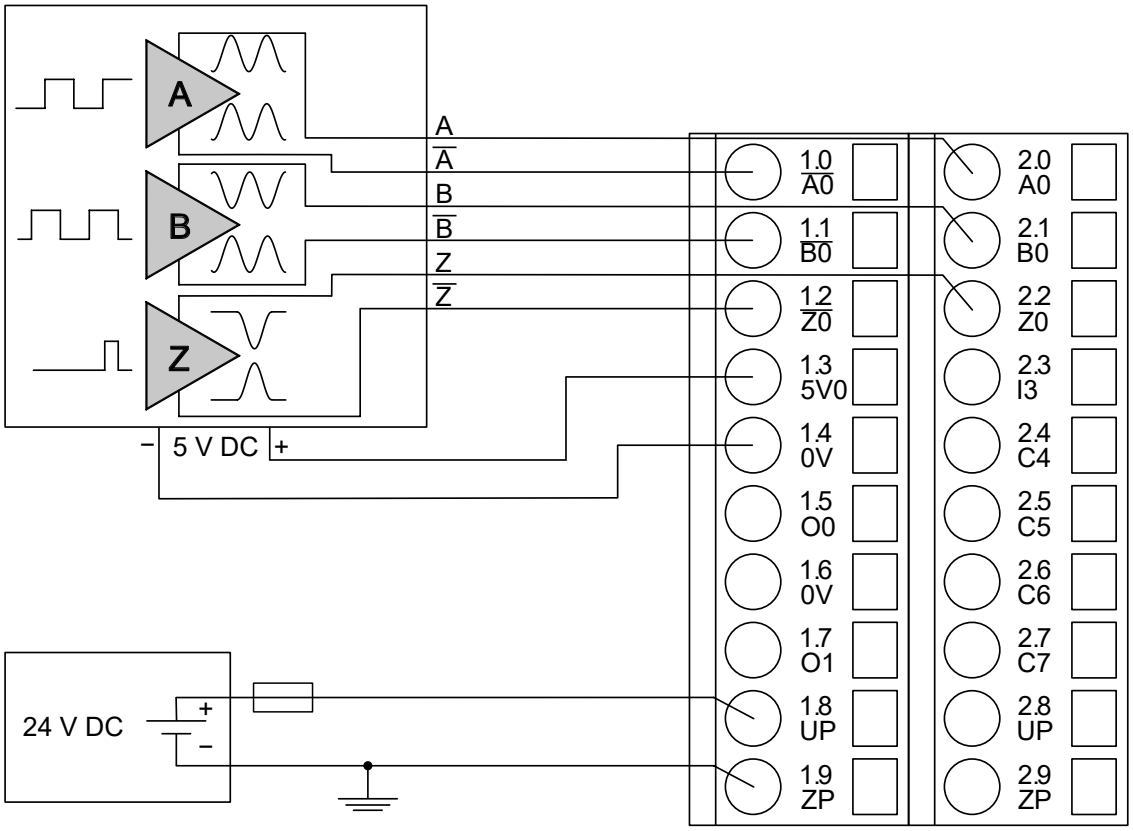


The wires A, B and Z need not to be connected to the module. They are left open.

When using different power supplies for the encoder device and the CD522, make sure that the reference potentials of both power supplies are interconnected.

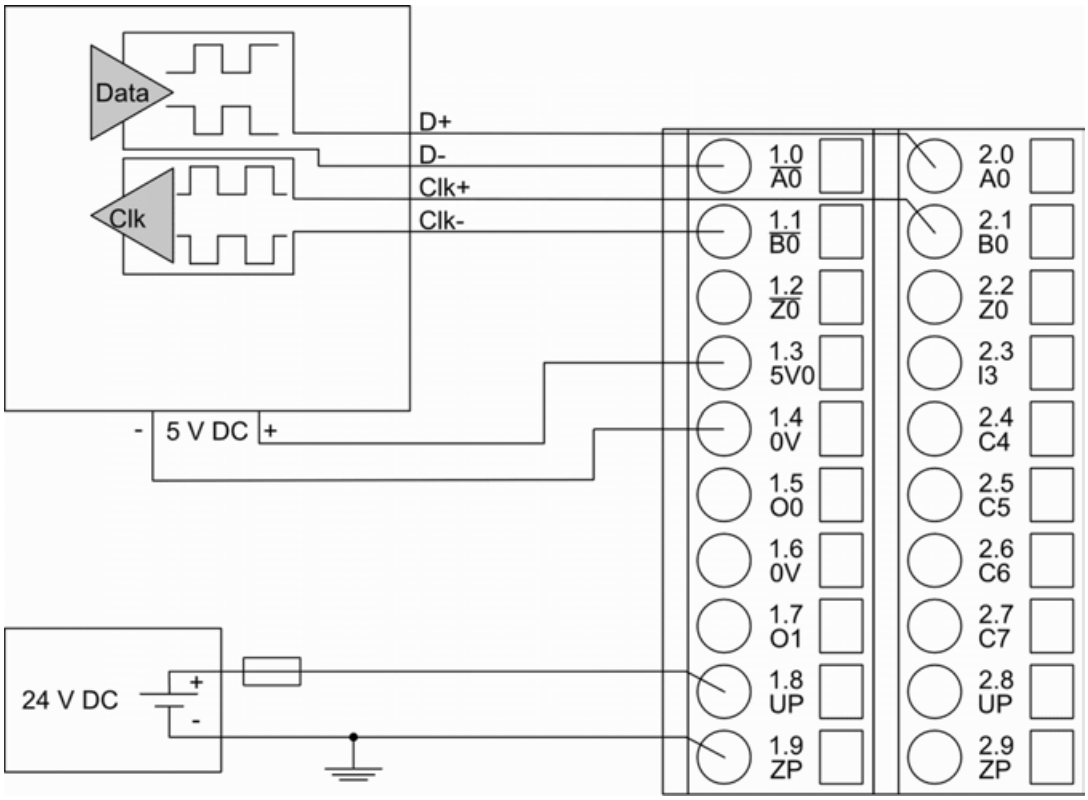
Connection of encoders with 1 Vpp sine signal

The encoder is powered through the 5 V power supply which is integrated in the CD522.



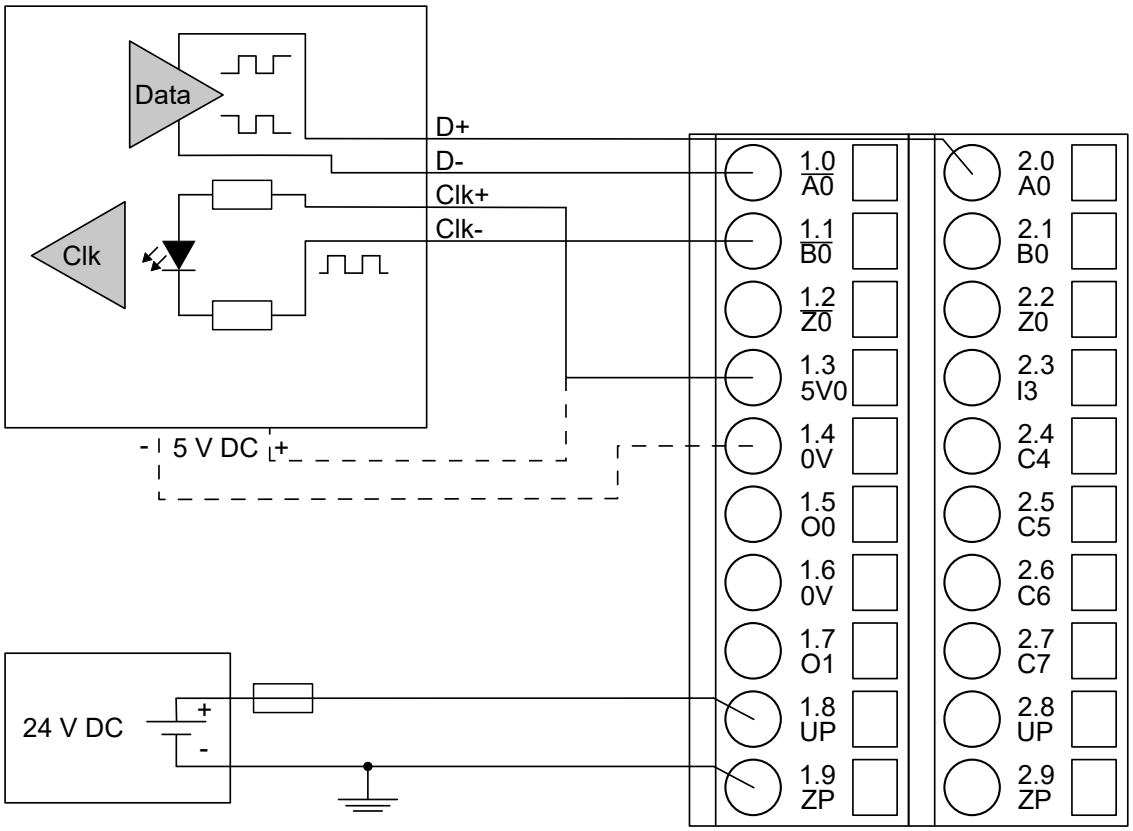
Connection of absolute encoders with SSI interface and differential RS-422 signal

The encoder is powered by the 5 V power supply which is integrated in the CD522.

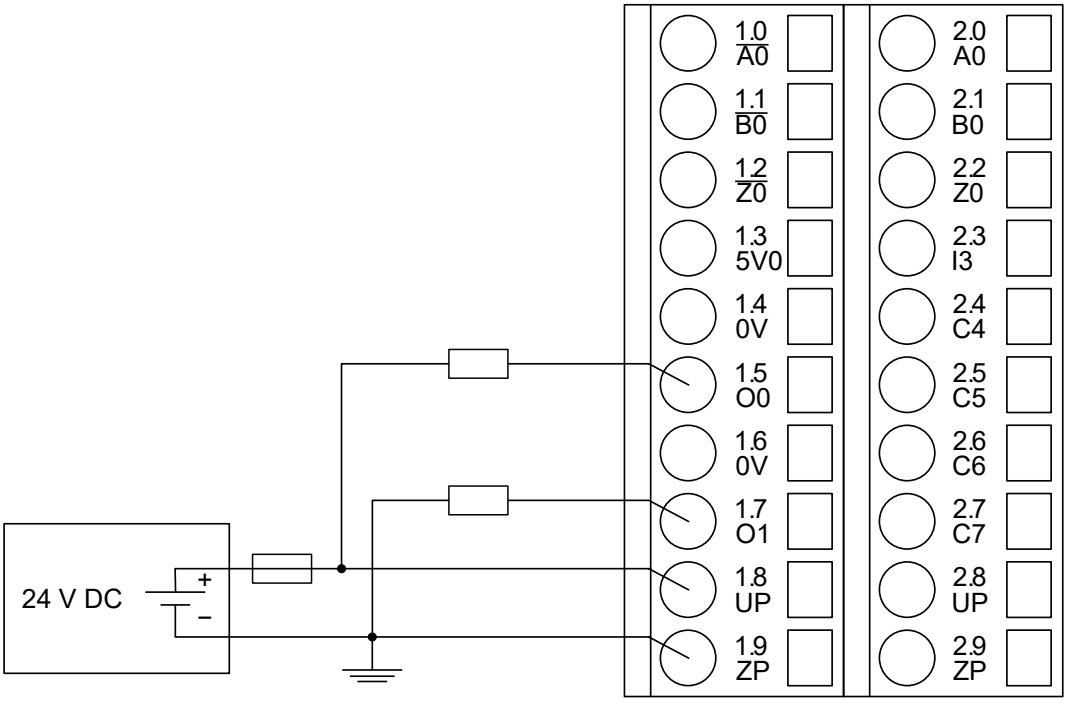


Connection of absolute encoders with an SSI interface and an optocoupler interface at CLK input

The encoder can optionally be powered by the 5 V power supply which is integrated in the CD522.



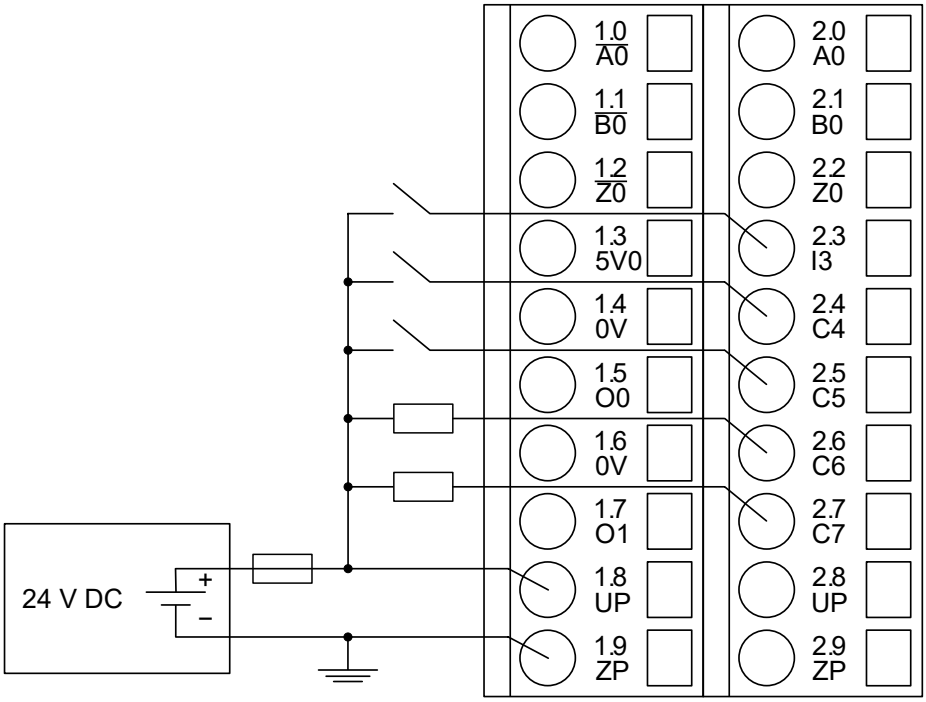
Connection of output loads to the PWM/Pulse putputs



NOTICE!
Risk of damaging the module
The PWM outputs have no protection against reverse polarity.

Connection of standard inputs/ outputs

Proceed with the inputs/outputs I11 and C12-C15 in the same way.



Connection of sensors with frequency out-puts

Proceed with the A0, B0, A1, B1 and Z1 in the same way.

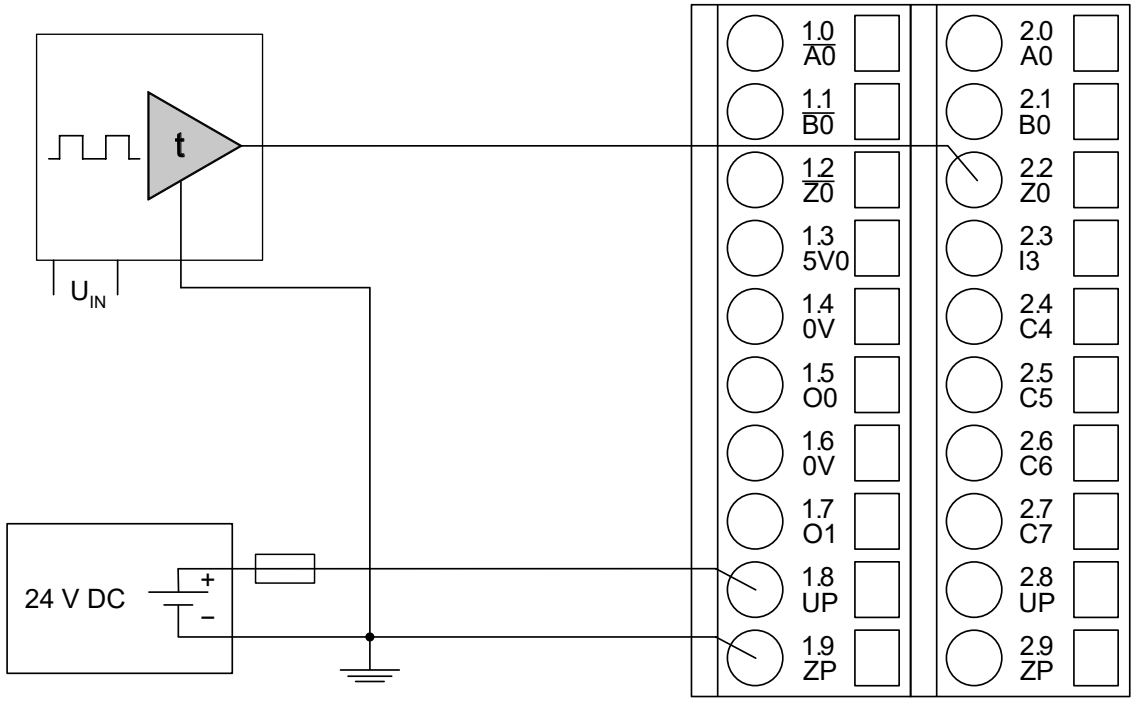


Fig. 923: Example of the connection of sensors with frequency outputs to the input Z0 of the CD522

NOTICE!

Risk of malfunctions!

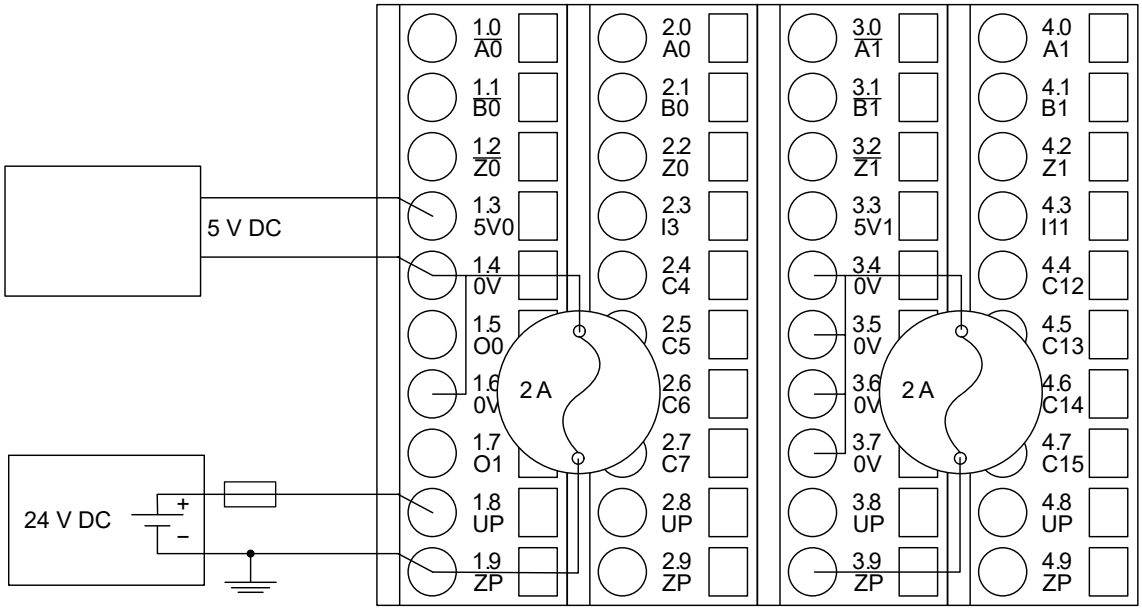
The edges of a signal must be strong enough (0.4 V/µs) to be recognized correctly by the module.

Put a 1 kΩ resistor between 0 V and the Z terminal when using a standard output as time generator.

Connection of sensors to the 5 V power supply

Proceed with the 5 V power supply 1 in the same way.

Each 5-V-power supply provides a current of 100 mA max. It is possible to parallel both integrated power supplies. In this case, the max. current is 200 mA.



NOTICE!

Risk of damaging the module

The integrated 2 A fuse cannot be replaced. If it blows, the module must be replaced.

Ensure that the current per 0 V connection does not exceed 0.5 A.

NOTICE!

Risk of damaging the module

The two 5 V outputs have no protection against reverse polarity.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	0
Digital outputs (bytes)	0
Analog inputs (words)	12
Analog outputs (words)	16

I/O configuration

The module itself does not store configuration data. It receives its parameterization data from the master device of the I/O bus (CPU or communication interface module) during power-up of the system.

Hence, replacing I/O modules is possible without any re-parameterization via software.

If the external power supply voltage via UP/ZP terminals fails, the I/O module loses its configuration data. The whole station has to be switched off and on again to re-configure the module.

Parameterization

Firmware version	Configuration
Firmware version > V2.0.0	The arrangement of the parameter data is performed by Control Builder Plus/ Automation Builder software.

The parameter data directly influences the functionality of modules.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Module: Module slot address: Y = 1...10

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Module ID	Intern	1805 [1])	WORD	0x070D	0	65535	0x0Y01
Ignore module [2])	No Yes	0 1	BYTE	No 0x00			Not for FBP
Parameter length	Internal	42	BYTE	0	0	255	xx02 [3])
Check supply	Off On	0 1	BYTE	On 0x01			0x0Y03
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	8 ms 0x02	0	3	0x0Y04
Mode Counter 0	see table below	0	BYTE	0x00	0	15	0x0Y05
Counter 0 frequency limit	No filter 50 Hz 500 Hz 5 kHz 20 kHz	0 1 2 3 4	BYTE	No filter 0x00	0	4	0x0Y06
Counter 0 input level	0-24 V DC 0-5 V DC Differential 1 Vpp sinus	0 1 2 3	BYTE	0-24 V DC 0x00	0	3	0X0Y07
SSI 0 frequency	200 kHz 500 kHz 1 MHz	2 3 4	BYTE	200 kHz 0x02	0	4	0x0Y08
SSI 0 resolution (in bit)	8 to 32 bit		BYTE	16 bit 16	8	32	0x0Y09
SSI 0 code type	Binary	0	BYTE	Binary 0	0	0	0x0Y0A

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
SSI 0 polling time	10 ms		BYTE	10	1	255	0x0Y0B
5 V sensor 0 supply	Off On	0	BYTE	Off 0x00	0	1	0x0Y0C
Mode Counter 1	see table below	0	BYTE	0x00	0	15	0x0Y0D
Counter 1 frequency limit	No filter 50 Hz 500 Hz 5 kHz 20 kHz	0 1 2 3 4	BYTE	No filter 0x00	0	4	0x0Y0E
Counter 1 input level	0-24 V DC 0-5 V DC Differen-tial 1 Vpp sinus	0 1 2 3	BYTE	0-24 V DC 0x00	0	3	0X0Y0F
SSI 1 fre-quency	200 kHz 500 kHz 1 MHz	2 3 4	BYTE	200 kHz 0x02	2	4	0x0Y10
SSI 1 res-olution (in bit)	8 to 32 bit		BYTE	16 bit 16	8	32	0x0Y11
SSI 1 code type	Binary	0	BYTE	Binary 0	0	0	0x0Y12
SSI 1 polling time	10 ms		BYTE	10	1	255	0x0Y13
5 V sensor 1 supply	Off On	0	BYTE	Off 0x00	0	1	0x0Y14
Detection SC on sensors	Off On	0	BYTE	Off 0x00	0	1	0x0Y15

Name	Value	Internal Value	Internal value, Type	Default	Min.	Max.	EDS Slot Index
Output behaviour com fault	Off Last value Substitute Last value 5s Substitute 5s Last value 10s Sub-stitute 10s	0 1 2 3 4 5 6	BYTE	Off 0x00	0	1	0x0Y16
Substitute value	0	0	WORD	Default 0x0000	0	65536	0x0Y17

[1]) With CS31 and addresses smaller than 70 and FBP, the value is increased by 1

[2]) Not with FBP

[3]) Value is hexadecimal: HighByte is slot (xx: 1...10), LowByte is index (1...n)

Table 481: Operating modes for counters 0 and 1, configuration table

Internal value	Operating modes of counter
0	No counter / No PWM (default value)
1	1-1 UpDown counter (A)
2	2-1 UpDown with release input
3	3-2 UpDown counters (A, B)
4	4-2 UpDown (A, B on falling edges)
5	5-1 UpDown dynamic set (B) / rising edge
6	6-1 UpDown dynamic set (B) / falling edge
7	Not used
8	8-1 UpDown with release (B), 0 cross detection
9 - 19	Not used
20	11-1 Incremental encoder
21	12-2 Incremental encoder X2
22	13-1 Incremental encoder X4
30	14-1 SSI, absolute encoder
40	15-1 Time frequency meter

Table 482: GSD file

Ext_User_Prm_Data_Len =	25
Ext_User_Prm_Data_Const(0) =	0x07, 0x0E, 0x17, \ 0x01, 0x02, \ 0x00, 0x00, 0x00, 0x02, 0x10, 0x00, 0x0A, 0x00, \ 0x00, 0x00, 0x00, 0x02, 0x10, 0x00, 0x0A, 0x00, \ 0x00, 0x00, 0x00, 0x00;

Diagnosis

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**FBP diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	14	1...10	31	31	19	Checksum error in the I/O module	Replace I/O module
	11 / 12	ADR	1...10				
3	14	1...10	31	31	9	Overflow diagnosis buffer	New start
	11 / 12	ADR	1...10				
3	14	1...10	31	31	26	Parameter error	Check master
	11 / 12	ADR	1...10				
3	14	1...10	31	31	11	Process voltage too low	Check process voltage
	11 / 12	ADR	1...10				

Table 483: Channel error CD522

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**FBP diagnosis block**	
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Channel error							
4	14	1...10	1	0...15	47	Output short circuit	Check output connection or terminal
	11 / 12	ADR	1..10				
4	14	1...10	1	0, 1, 8, 9	10	Input frequency too high	Check frequency filter parameter or sensor
	11 / 12	ADR	1...10				
4	14	1...10	1	0, 1	2	PWM frequency too high	Clamp min/max value in program
	11 / 12	ADR	1...10				
4	14	1...10	1	0, 1	10	PWM duty cycle out of range (0-1000)	Clamp min value to 0 in program
	11 / 12	ADR	1...10				
4	14	1...10	1	0, 1	11	5 V sensor supply too low	Check wiring & sensor power
	11 / 12	ADR	1...10				
4	14	1...10	1	0, 1	18	Internal fuse on 0 V has blown, 0 V not connected to GND	Check wiring, replace module
	11 / 12	ADR	1...10				

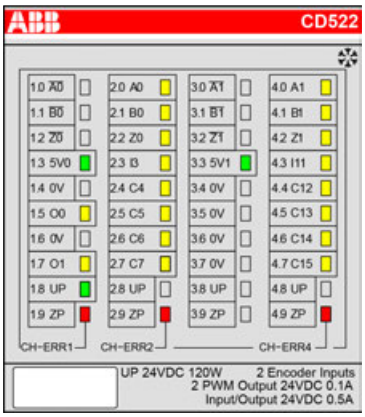
Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1...10 = decentralized communication interface module 1...10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1...10 = expansion 1...10 Channel error: I/O bus or FBP = module type (2 = DO); COM1/COM2: 1...10 = expansion 1...10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power-on procedure, the module initializes automatically. All LEDs (except the LEDs for the signal states) are on during the initialization.

LED		State	Color	LED = OFF	LED = ON	LED flashes
	A0, B0, Z0	Encoder 0 inputs	Yellow	Input ON	Input OFF	LED follows the state of the inputs, depending on frequency
	A1, B1, Z1	Encoder 1 inputs	Yellow	Input ON	Input OFF	LED follows the state of the inputs, depending on frequency
	I3 and I11	Digital inputs	Yellow	Input = ON (the input voltage is even displayed if the supply voltage is OFF).	Input = OFF	---
	C4 to C7 and C12 to C15	Configurable digital inputs/ outputs	Yellow	Input/output = ON (the input voltage is even displayed if the supply voltage is OFF).	Input/output = OFF	---
	O0 and O1	Digital PWM outputs	Yellow	Output = ON	Output = OFF	LED follows the state of the outputs, depending on frequency and operation mode
	5V0 and 5V1	Power supply for encoders	Green	Configuration ON and power 5-V-power ready	Configuration OFF or power failure	Power supply outputs are short-circuited
	UP	Process supply voltage	Green	Process voltage OK	Process voltage is missing	---

LED		State	Color	LED = OFF	LED = ON	LED flashes
	CH-ERR1, CH-ERR2, CH-ERR4		Red	Severe error within the corresponding group	No error or process voltage is missing	Error on one channel of the corresponding group (e.g. short circuit at an output)
	CH-ERR *)	Error indication	Red	Internal error or configuration is not loaded	--	---
	*) All LEDs CH-ERR1, CH-ERR2 and CH-ERR4 light up simultaneously					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter		Value
Process supply voltage		
	Connections	Terminals 1.8, 2.8, 3.8 and 4.8 for UP (+24 V DC) and 1.9, 2.9, 3.9 and 4.9 for ZP (0 V)
	Protection against reverse voltage	Yes
	Rated protection fuse at UP	10 A fast
	Rated value	24 V DC
	Max. ripple	5 %
Current consumption		
	From UP	0.07 A + max. 0.008 A per input + max. 0.5 A per output + 0.01 A for A, B and Z inputs
	Via I/O bus	Ca. 5 mA
	Inrush current from UP (at power-up)	0.04 A^2s
Galvanic isolation		Yes, per module
Max. power dissipation within the module		6 W (outputs unloaded)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal mounting or vertical with derating (output load reduced to 50 % at 40 °C)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs/outputs if used as standard inputs

Parameter		Value
Number of channels		2 + 8 configurable digital inputs/outputs
Reference potential for all inputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the rest of the module
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input data length		24 bytes
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

* Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as standard outputs

Parameter		Value
Number of channels		8 configurable digital inputs/outputs
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8...4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		Typ. 10 µs
Output data length		32 bytes
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together, PWM included)	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

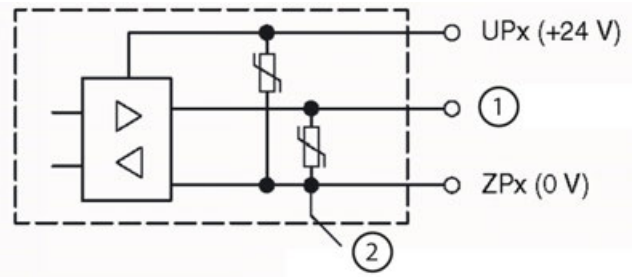


Fig. 924: Circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off

Technical data of the high-speed inputs (A0, B0, Z0; A1, B1, Z1)

Parameter	Value
Number of channels per module	6
Reference potential for all inputs	Terminal 1.9, 2.9, 3.9 and 4.9 (negative pole of the process voltage, signal name ZP)

Parameter		Value	
Input Type		24 V DC	5 V DC / Differential Sinus 1 Vpp
Input current per channel			
	Input voltage +24 V	Typ. 14 mA	
	Input voltage +5 V	> 4.8 mA	
	Input voltage +15 V	> 12 mA	
	Input voltage +30 V	< 15 mA	
Input type acc. to EN 61131-2		Type 1	
Input frequency max. (fast counter)		300 kHz	300 kHz
Input frequency max. (frequency measurement)		5 kHz	5 kHz
Input signal voltage		24 V DC	5 V DC
Signal 0		-3 V...+5 V	-3 V...+0,5 V
Undefined signal		> +5 V...< +15 V	--
Signal 1		+15 V...+30 V	+0,5 V...+30 V
Ripple with signal 0		Within -3 V ... +5 V	Within -3 V...+0.5 V
Ripple with signal 1		Within +15 V...+30 V	Within +0,5 V...+30 V
Max. cable length			
	Shielded	1000 m	
	Unshielded	600 m	

Technical data of the fast outputs O0 and O1

Parameter		Value
Number of channels		2
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8...4.8 (positive pole of the process supply voltage, signal name UP)
Indication of the output signals		Brightness of the LED depends on the number of pulses emitted (0 % to 100 %) (pulse output mode only)
Output voltage for signal 1		UP (-0.1 V)
Output voltage for signal 0		ZP (+0.3 V)
Output delay (0->1 or 1->0)		Typ. 1 µs
Output current		
	Rated value, per channel	100 mA at UP = 24 V
	Maximum value (all channels together, configurable outputs included))	8 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
De-magnetization when inductive loads are switched off		With varistors integrated in the module (see figure above)

Parameter		Value
Switching frequency		PWM: up to 100 kHz (min. step for PWM value: 2 µs) Pulse: up to 15 kHz
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.1x A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short-circuit/overload
Resistance to feedback against 24 V signals		Yes
Resistance to feedback against reverse polarity		No
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the fast outputs (SSI CLK output B0, B1 for optical interface)

Parameter	Value
Number of channels	2
Reference potential for all outputs	Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage	For all outputs: terminals 1.8...4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 0	≤ 1.5 V at 10 mA
Output delay (0->1 or 1->0)	Typ. 0.3 µs
Output current	≤ 10 mA
Switching frequency	< 1 MHz (depending on firmware)
Short-circuit-proof / overload-proof	Yes
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes
Resistance to feedback against reverse polarity	No
Max. cable length (shielded)	Typ. 12.5 m at 500 kHz (depending on sensor)

Technical data of the fast outputs (SSI CLK Output Differential)

Parameter	Value
Number of channels	2
Reference potential for all outputs	Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage	For all outputs: terminals 1.8...4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1	≥ 2.9 V at 10 mA
Output voltage for signal 0	≤ 1.3 V at 10 mA
Output delay (0->1 or 1->0)	Typ. 0.3 µs

Parameter	Value
Output current	≤ 10 mA
Switching frequency	< 1 MHz (depending on firmware)
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.1x A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short-circuit/overload
Resistance to feedback against 24V signals	Yes
Resistance to feedback against reverse polarity	No
Max. cable length (shielded)	100 m

Technical data of the 5 V sensor supply

Parameter	Value
Number of supplies	2, independently configuration
Voltage supply (outputs unloaded)	5 V DC +/- 5%
Resistance to feedback against reverse polarity	No
Output current	100 mA max. (independently) 200 mA max. (parallel use)
Output diagnosis	Yes, with diagnosis LED and error message

Technical data of the 0 V reference input

Parameter		Value
Number of reference inputs (internally connected to ZP through internal fuse)		6
Max. current per connection		0.5 A
Internal fuse protection		
	Terminals 1.4 and 1.6	2 A
	Terminals 3.4 to 3.7	2 A

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 260 300 R0001	CD522, encoder & PWM module, 2 encoder inputs, 2 PWM outputs, 2 digital inputs 24 V DC, 8 digital outputs 24 V DC	Active
1SAP 460 300 R0001	CD522-XC, encoder & PWM module, 2 encoder inputs, 2 PWM outputs, 2 digital inputs 24 V DC, 8 digital outputs 24 V DC, XC version	Active

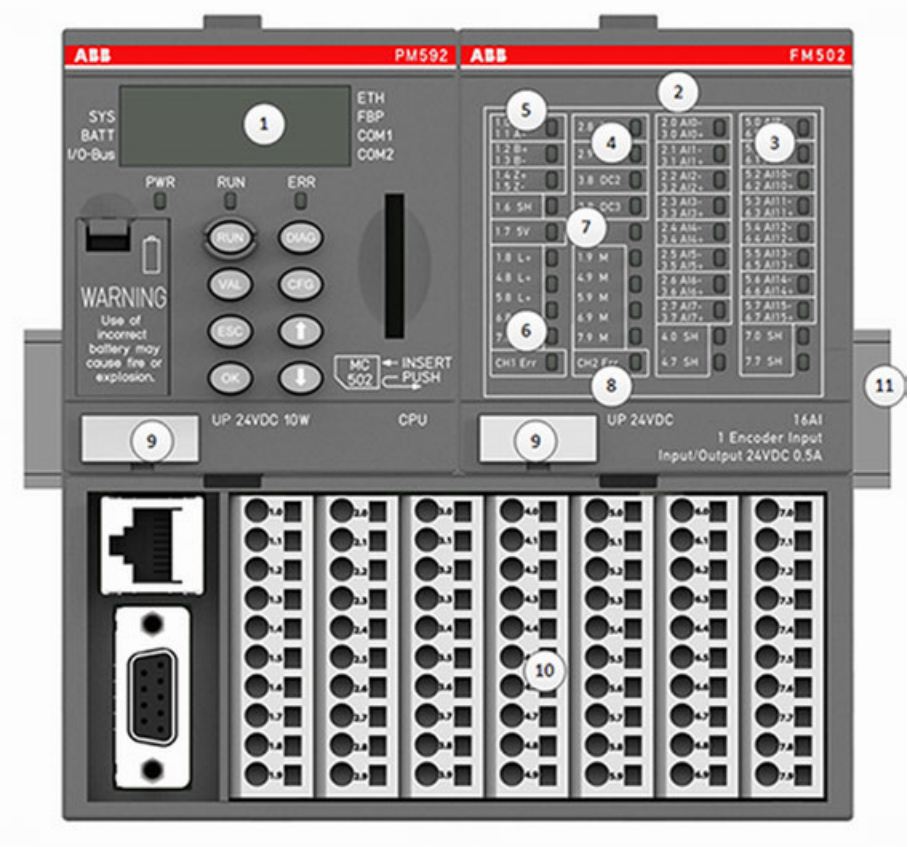
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

FM502-CMS - Analog measurements

- 16 fast analog inputs, up to 50k samples/s.
- Counting functions with different configurable modes, including incremental position encoder and frequency input.
- 4 dedicated inputs/outputs for specific counting measurement functions, e.g. touch, set, reset, start measurement.
- All unused inputs/outputs can be used with the specifications of standard inputs/outputs range.
- Synchronous sampling between all analog channels and the counting input.

FM502-CMS is used for condition monitoring via fast analog signals. For direct connection to processor module PM592-ETH and wiring, the function module terminal bases TF501-CMS or TF521-CMS are available, enabling AC500 communication modules and AC500 I/O modules ↳ *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848* ↳ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*.

For usage in extreme ambient conditions a XC version is available.



1 Processor module PM592-ETH
2 Allocation between terminal no. and signal name
3 16 green/red LEDs to display the signal states at the analog inputs A0-A15
4 4 yellow LEDs to display digital inputs DI0, DI1 and digital inputs/outputs DC2,DC3
5 3 yellow LEDs display encoder/counter inputs
6 1 green LED to display the state of the process supply voltage L+
7 1 green LED to display the state of 5 V supply voltage for encoder
8 2 red LEDs to display errors
9 Label
10 Function module terminal base
11 DIN rail
❄ Sign for XC version

Connections

FM502-CMS is plugged on the TF5x1-CMS together with PM592-ETH. The connection is established using the terminals of the TF5x1-CMS. The FM502-CMS can be replaced without re-wiring the TF5x1-CMS *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V

Connection of IEPE sensors

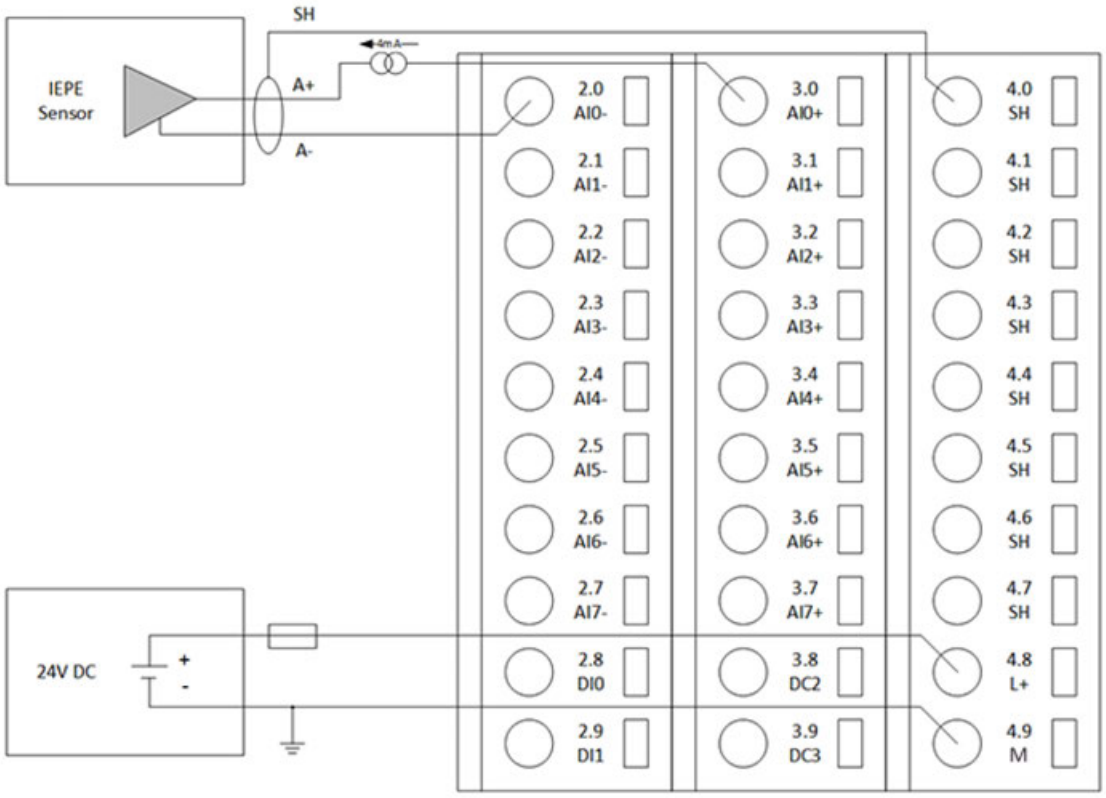
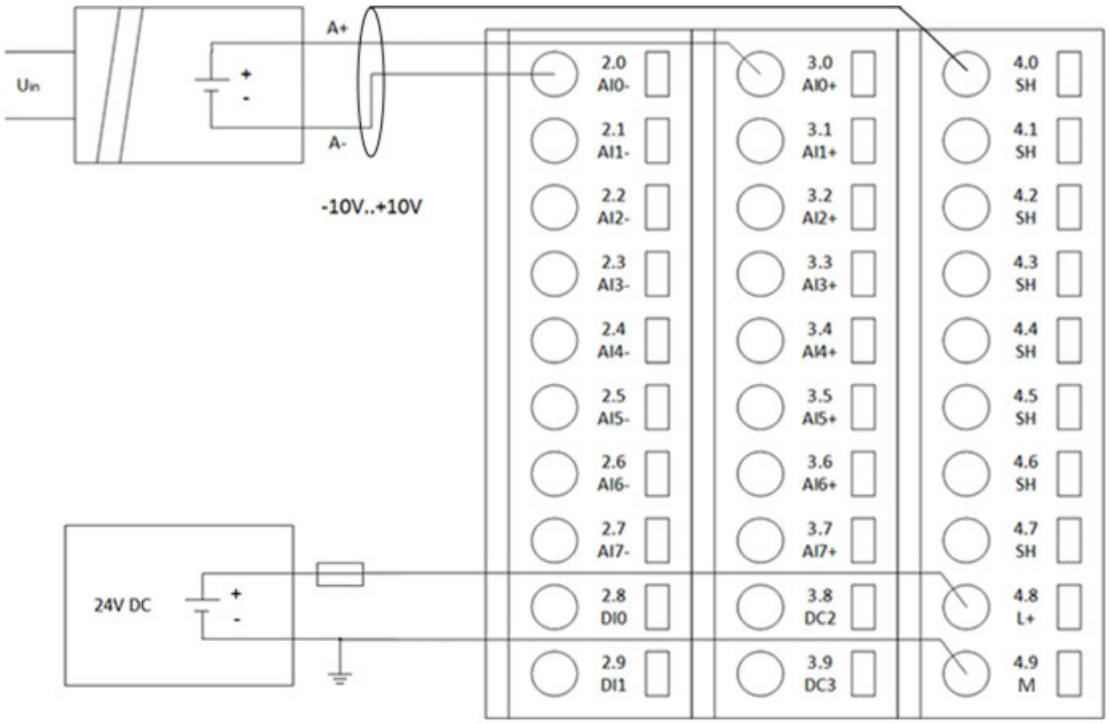


Fig. 925: Connection of IEPE sensor to the FM502-CMS

In order to avoid error messages or long processing times, we recommend to configure unused analog input channels as "unused".

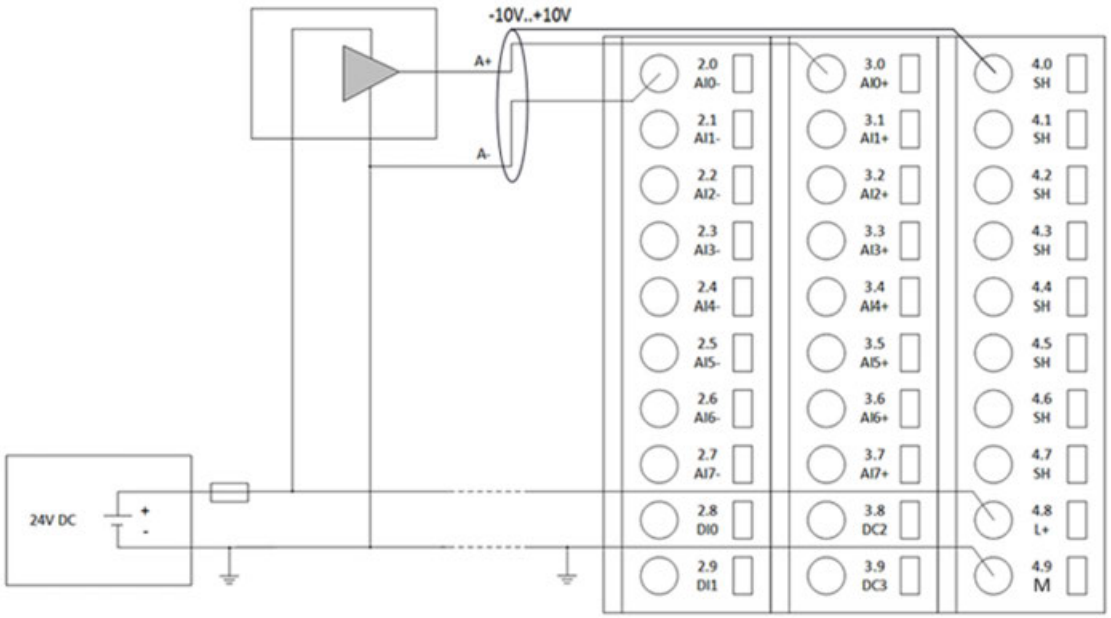
> *For the open-circuit detection (cut wire) in IEPE mode, each channel is pulled up to the positive supply rail by a high impedance. If nothing is connected, the maximum value will be read* *Chapter 1.6.2.7.2.2.5 “Measuring ranges” on page 4674.*

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply



Every negative analog input is internally connected to M (0 V) via an individual low impedance (PTC) return current path for the sensor supply current in IEPE mode. This is important for applications where a high input impedance on the negative analog input is required. Example: Stain gauges, bridge network.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply



NOTICE!

Analog sensors should be galvanically isolated against earth. In order to avoid inaccuracy with the measuring results, the analog sensors should also be isolated against the power supply.

NOTICE!
If A- is not connected directly to M at the sensor, the supply current flows via A- to M. Measuring errors can occur caused by voltage differences between M and A-.

NOTICE!
At system start up, the 4 mA current source on each analog input is active for < 10 s. During this limited time, a positive analog input will drift to < 21 V and no current is flowing, when a high impedance sensor is connected. When a low impedance sensor is connected to the analog input, the current is limited to 4 mA. For analog sensors other than standard IEPE, please make sure that the connected sensor will not be damaged under these conditions.

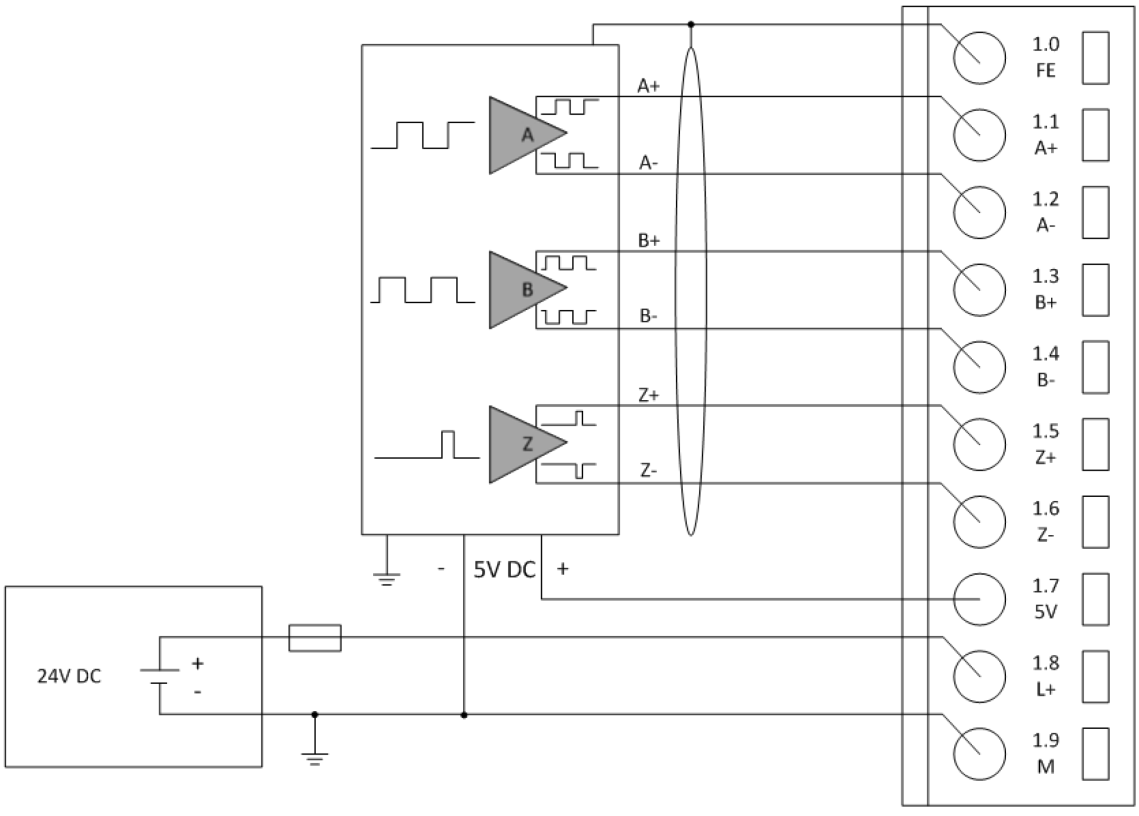
Analog signals must be laid in shielded cables. The analog cable shield must only be connected on the module side (SH terminals) to avoid isothermal relaxation currents influencing the measuring results, and for optimal robustness against external noise. The shield connection must be as short as possible (< 3 cm). The analog shield is capacitive coupled internally with functional earth (FE). Generally to avoid unacceptable potential differences between different parts of the installation, low-resistance equipotential bonding conductors must be laid.

In order to avoid error messages or long processing times, it is recommended to configure unused analog input channels as "unused".

In order to avoid inaccuracy in the analog measurement, the FM502-CMS should be in thermal balance > 15 minutes after power up and start of the PLC application, before measurements are started.

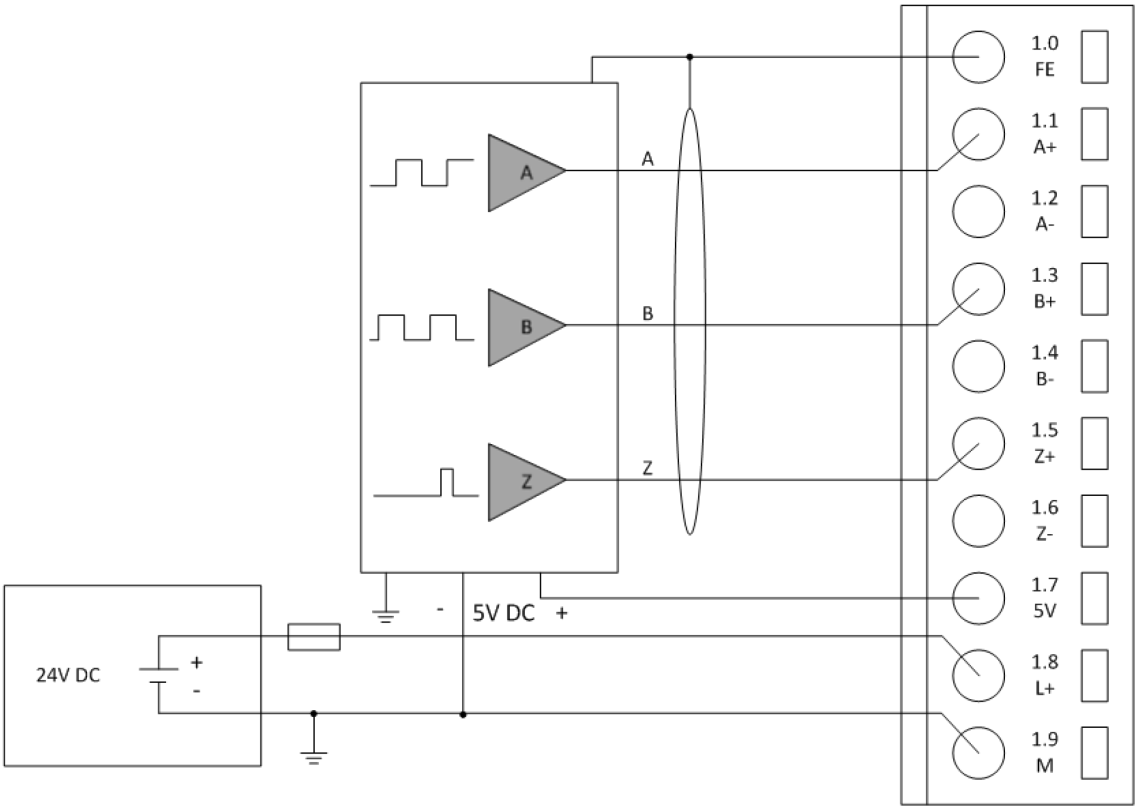
Connection of encoders with differential RS-422 signal

The encoder is powered by the 5 V power supply which is integrated in the FM502-CMS.

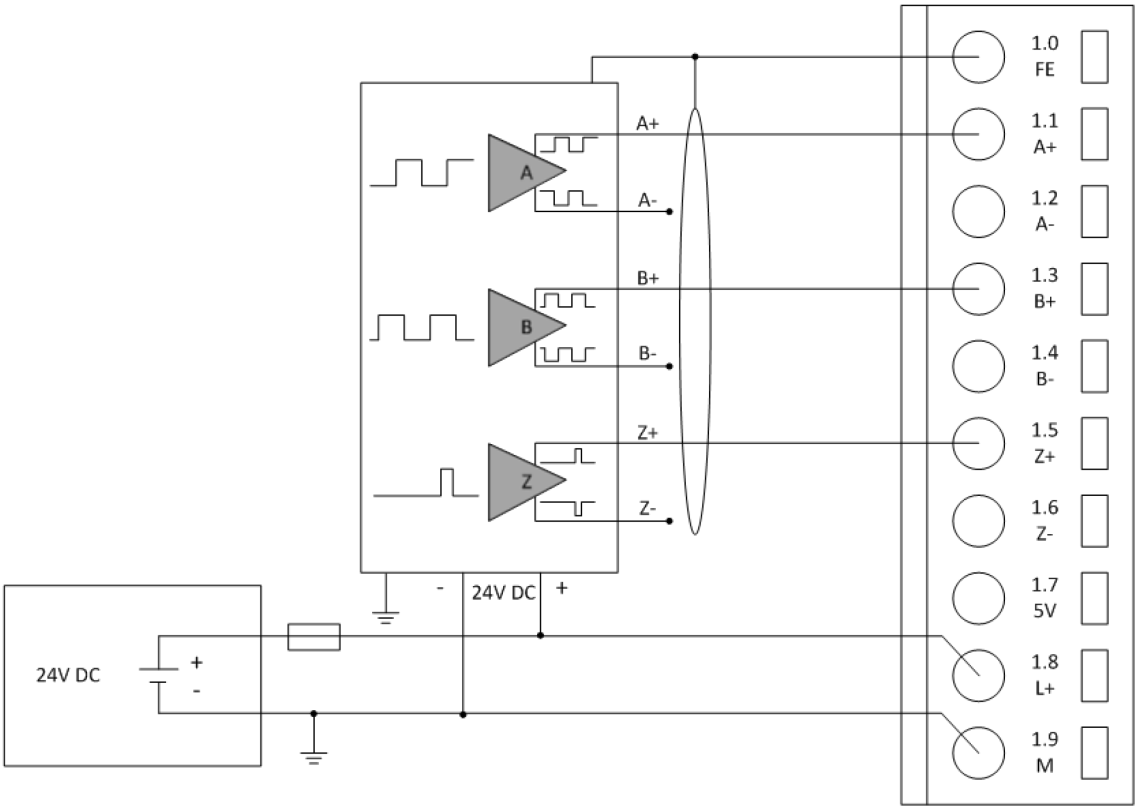


Connection of encoders with 5 V TTL signal

The encoder is powered through the 5 V power supply which is integrated in the FM502-CMS.



Connection of encoders with 24 V totem pole signal

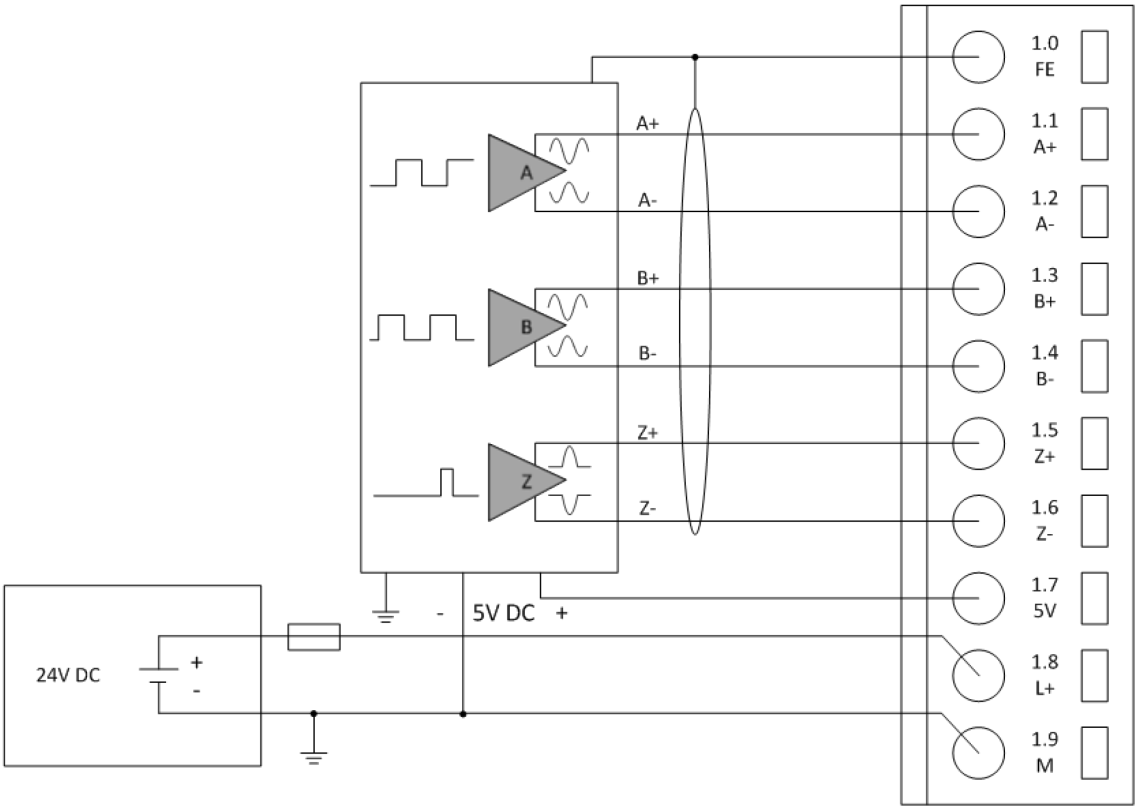


The wires A-, B- and Z- must not be connected to the module for single-ended operation. They are left open.

When using different power supplies for the encoder device and the FM502-CMS, make sure that the reference potentials of both power supplies are interconnected.

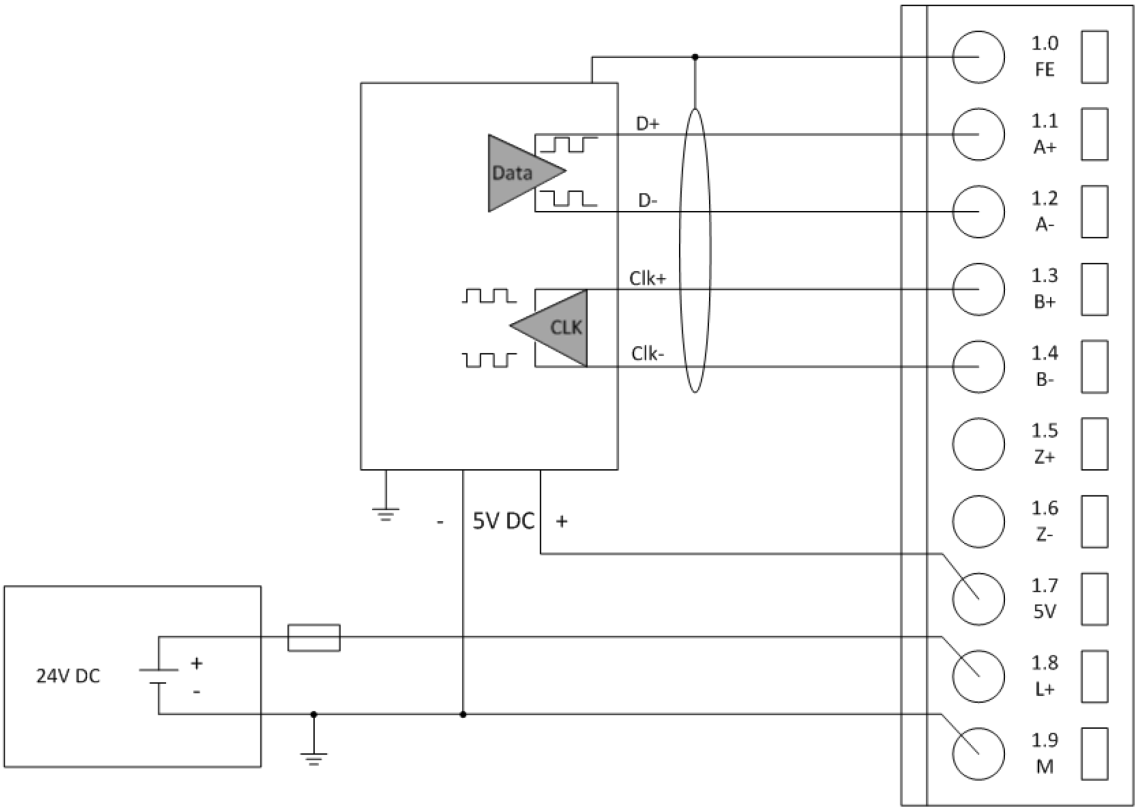
Connection of encoders with 1 Vpp sine signal

The encoder is powered by the 5 V power supply which is integrated in the FM502-CMS.



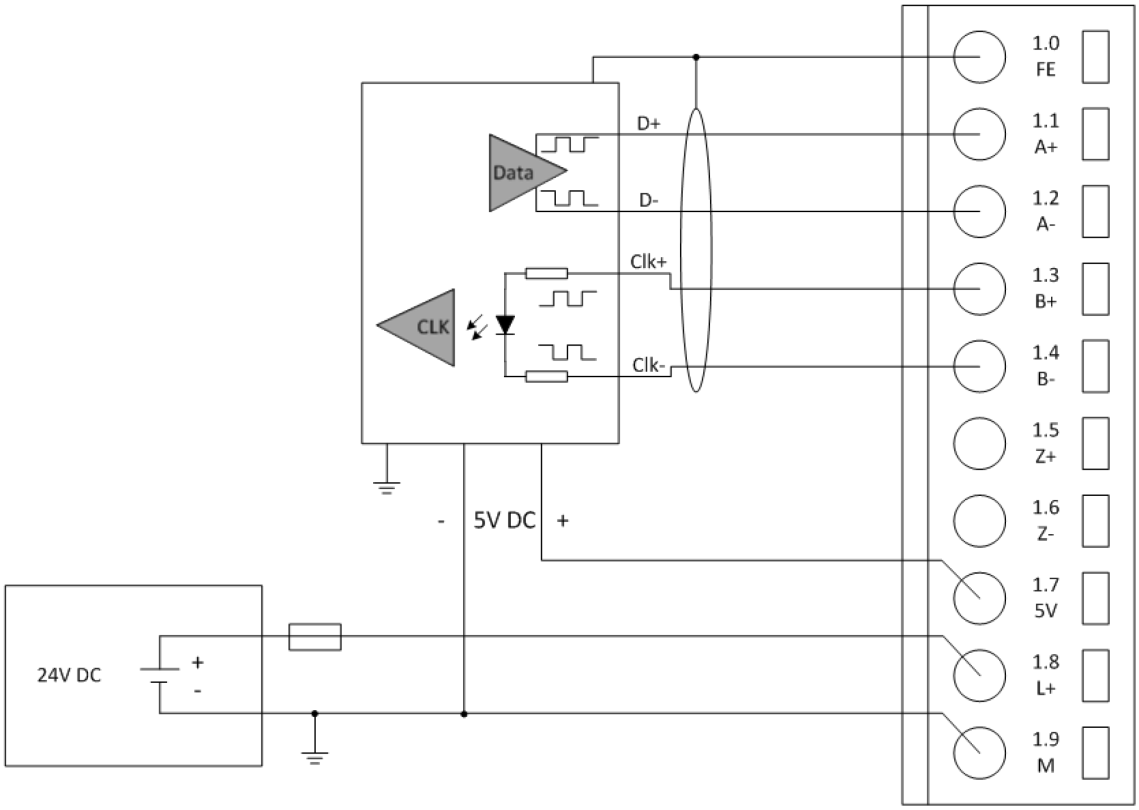
Connection of absolute encoders with RS-422 differential SSI interface

The encoder is powered by the 5 V power supply which is integrated in the FM502-CMS.



Connection of absolute encoders with optical SSI interface (optocoupler at CLK input)

The encoder can optionally be powered by the 5-V-power-supply which is integrated in the FM502-CMS.

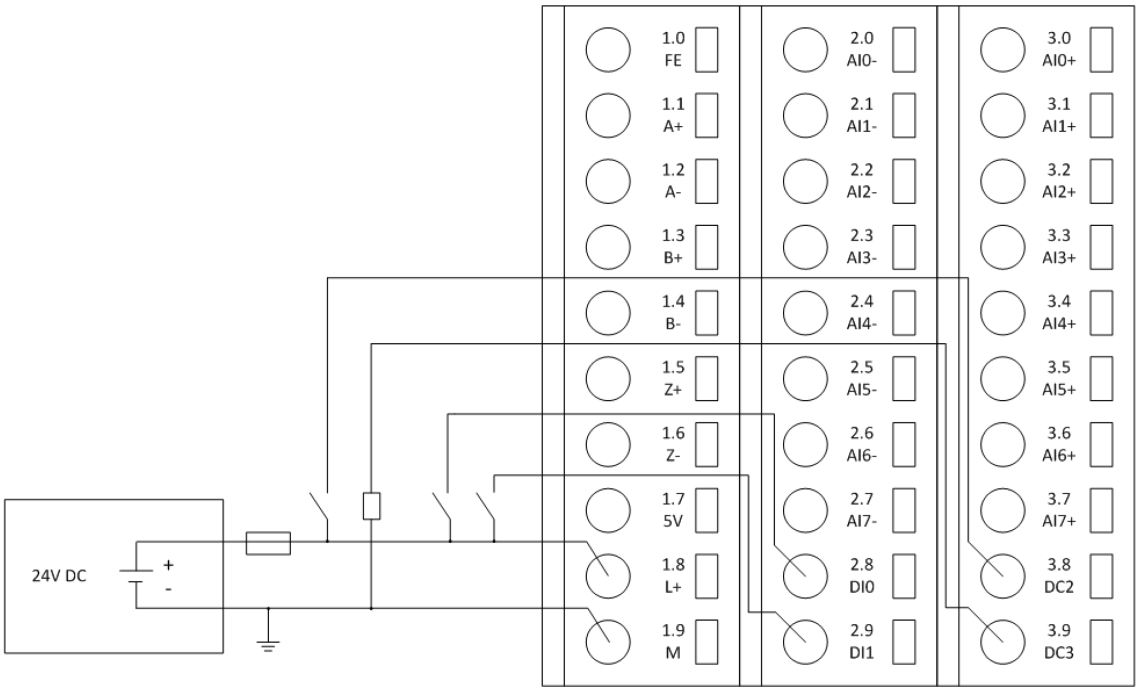


Encoder/counter signals must be laid in shielded cables. The cable shield must be grounded at both sides of the cable. In order to avoid unacceptable potential differences between different parts of the installation, low-resistance equipotential bonding conductors must be laid. Only for applications with low disturbance and/or cables length < 30 m the shield might be omitted.

The 5 V output provides a current of 100 mA max.

NOTICE!
Risk of damaging the FM502-CMS!
The 5 V output has no protection against reverse polarity.

Connection of standard inputs/ outputs



Connection of sensors with frequency outputs

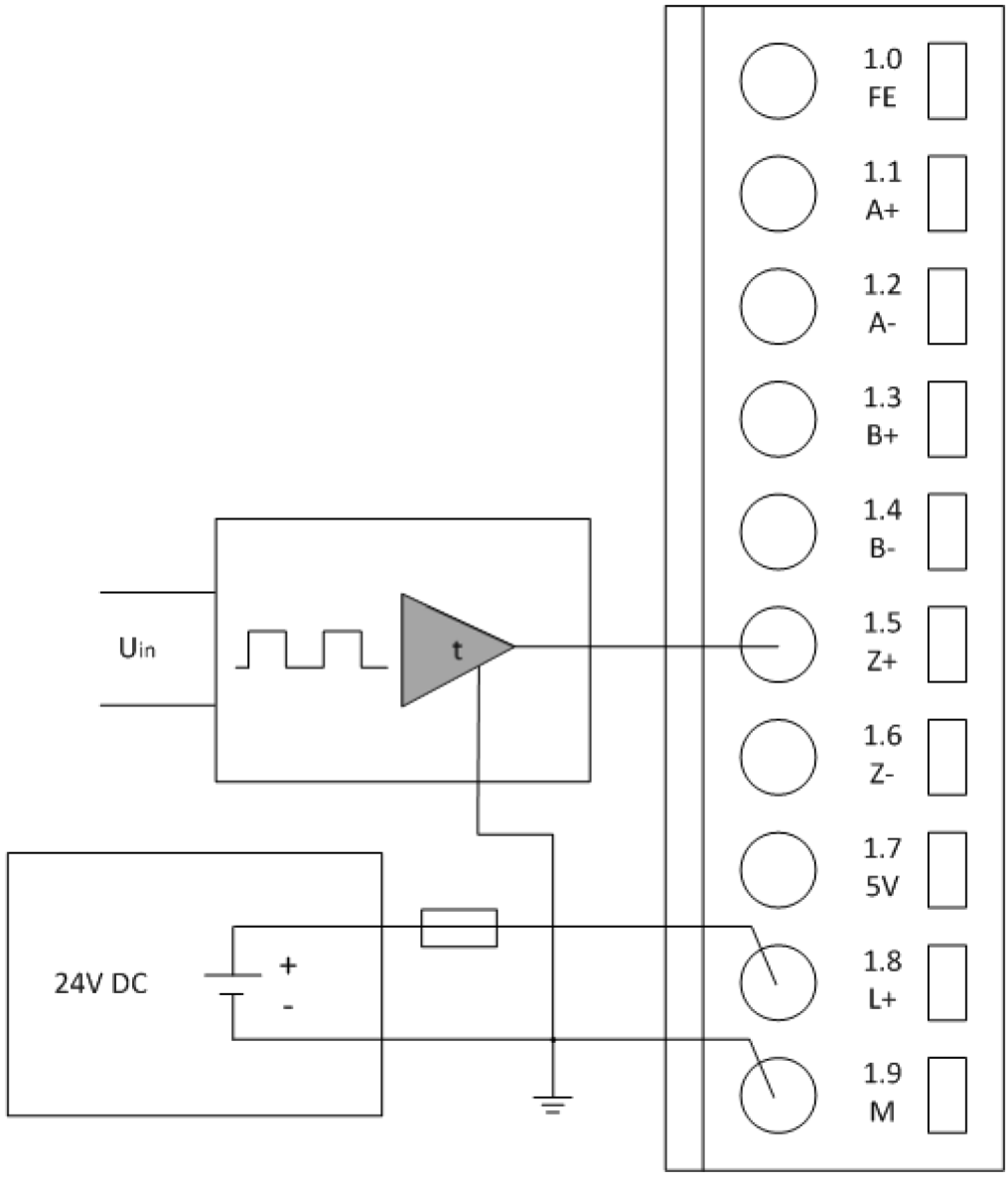


Fig. 926: Example for connection of sensors with frequency outputs to the input Z+

Internal data exchange

Parameter	Value
Digital inputs (bytes)	4
Digital outputs (bytes)	8
Counter inputs (words)	4
Counter outputs (words)	2
Analog inputs (words)	16
Analog outputs (words)	0

Diagnosis

Table 484: Module error FM502-CMS

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000...063**	**AC500 display**	**<-- Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error mes-sage**	**Online number**	**Remedy**
	[1])	[2])	[3])	[4])				
3	5	255	29	31	3	Timeout in the I/O module	184545219	Replace I/O module
3	5	255	29	31	11	Process voltage too low	184545227	Replace I/O module
4	5	255	29	31	13	FW update failed	184545229	Retry FW update
3	5	255	29	31	18	5 V sensor supply too low	184545234	Check wiring & sensor power, Replace I/O module
3	5	255	29	31	19	Checksum error in the I/O module	184545235	Replace I/O module
3	5	255	29	31	36	Internal data exchange failure	184545252	Replace I/O module
3	5	255	29	31	43	Internal error in the module	184545259	Replace I/O module
4	5	255	29	31	52	Production data missing	184545268	Call support

Table 485: Channel error FM502-CMS

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500 display**	**<-- Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**FBP diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Online number**	**Remedy**
	[1])	[2])	[3])	[4])				
4	5	255	29	0..15	5	Analog value overflow at an analog input	184543237, 184543301, 184543365, 184543429, 184543493, 184543557, 184543621, 184543685, 184543749, 184543813, 184543877, 184543941, 184544005, 184544069, 184544133, 184544197	Check input value
4	5	255	29	0..15	7	Analog value underflow at an analog input	184543239, 184543303, 184543367, 184543431, 184543495, 184543559, 184543623	Check input value

E1...E4	d1	d2	d3	d4	Identi-fier 000...063	AC500 display	<-- Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diag-nosis block		
Class	Inter-face	Device	Module	Channel	Error identi-fier	Error mes-sage	Online number	Remedy
	[1])	[2])	[3])	[4])				
							23, 184543687, 184543751, 184543815, 184543879, 184543943, 184544007, 184544071, 184544135, 184544199	
4	5	255	29	0..1	10	Encounter/ counter input frequency too high	184543242, 184543306	Check frequency filter parameter or sensor

E1...E4	d1	d2	d3	d4	Identi-fier 000...063	AC500 display	<-- Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diag-nosis block		
Class	Inter-face	Device	Module	Channel	Error identi-fier	Error mes-sage	Online number	Remedy
	[1])	[2])	[3])	[4])				
4	5	255	29	0..15	45	Cut wire at an analog input (only in IEPE mode)	184543277, 184543341, 184543405, 184543469, 184543533, 184543597, 184543661, 184543725, 184543789, 184543853, 184543917, 184543981, 184544045, 184544109, 184544173, 184544237	Check terminal

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<-- Display in	
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser		
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diagnosis block		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Online number**	**Remedy**
	[1])	[2])	[3])	[4])				
4	5	255	29	0..15	46	Short circuit at an analog input (only in IEPE mode)	184543278, 184543342, 184543406, 184543470, 184543534, 184543598, 184543662, 184543726, 184543790, 184543854, 184543918, 184543982, 184544046, 184544110, 184544174, 184544238	Check terminal
4	5	255	29	2..3	47	Short circuit at an digital output	184543407, 184543471	Check terminal or output connection

Remarks:

[1])	In AC500, the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = communication interface module 1..10, ADR = hardware address (e.g. of the DC551)

[3])	With "Module" the following allocation applies depending on the master: Module error: I/O bus or FBP: 31 = module itself; COM1/COM2: 1..10 = expansion 1..10 channel error: I/O bus or FBP = module type (1 = AI); COM1/COM2: 1..10 = expansion 1..10
[4])	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

During the power-on procedure, the module initializes automatically. All LEDs (except the LEDs for the signal states) are on during the initialization.

LED	State	Color	LED = ON	LED = OFF	LED flashing
AI0 - AI15	Analog channel state	Green	Channel activated and OK	Channel deactivated	CMS measurement running
		Red	Short circuit (only in IEPE mode) over- / undervoltage (only in +-10V mode)	-	Cable break (only in IEPE mode)
A, B, Z	Encoder 0 inputs	Yellow	Input ON	Input OFF	LED follows the state of the inputs, depending on frequency
DI0, DI1, DC2, DC3	Digital inputs	Yellow	Input = ON (the input voltage is even displayed if the supply voltage is OFF).	Input = OFF	-
DC2, DC3	Digital outputs	Yellow	Output = ON	Output OFF	-
5 V	Power supply for encoders	Green	Configuration ON and power 5-V-power ready	Configuration OFF or power failure	Power supply outputs are short-circuited
L+	Process supply voltage	Green	Process voltage OK Initialization finished	Process voltage OFF	Firmware update
CH-ERR1, CH-ERR2		Red	Serious error within the corresponding group	No error or process voltage is missing	Error on one channel of the corresponding group (e.g. short circuit at an output)

Measuring ranges

Table 486: Voltage input ranges

Range	IEPE	Digital value		-10 V...+10 V	Digital value	
		Decimal	Hex.		Decimal	Hex.
Open loop overflow	≥ 7.5	3145728	300000	≥ 12.0000	5033164	4CCCCC
Measured value too high	7.499997616... 6.00000238	3145727... 2516583	2FFFFF... 266667	11.999997 62... 10.000002 38	5033163... 4194305	4CCCCB... 400001
Normal range	6.00000... 0.00000238	2516582... 1	266666... 1	10.0000... 0,00000238	4194304... 1	400000... 1
	0.0000	0	0	0.0000	0	0
	-0.0000023 8... -6.00000	-1... -2516582	-1... -266666	-0.0000023 8... -10.0000	-1... -4194304	-1... -400000
Measured value too low	-6.0000023 8... -7.4999976 16	-2516583... -3145727	-266667... -2FFFFF	-10.000002 38... -11.999997 62	-4194305... -5033163	-400001... -4CCCCB
Short circuit / underflow	≤ -7.5	-3145728	-300000	≤ -12.0000	-5033164	-4CCCCC

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Table 487: Technical data of process supply voltage

Parameter	Value
Connections of terminals	The terminals 1.8, 4.8...7.8, 1.9, 4.9...7.9, 4.0...4.7, 7.0...7.7 are electrically interconnected within the TF5x1-CMS. Terminals 1.8, 4.8...7.8: process voltage L+ = +24 V DC Terminals 1.9, 4.9...7.9: process voltage M = 0 V Terminals 4.0...4.7, 7.0...7.7: analog shield clamps SH Terminal 1.0: FE shield clamp of encoder
Protection against reverse voltage	Yes
Rated protection fuse at UP	10 A fast
Rated value	24 V DC
Max. ripple	5 %

Parameter	Value
Current consumption from L+ (FM502-CMS and PM592-ETH, no communication module)	Max. 0.43 A + max. 0.5 A per output
Inrush current from L+ (at power up, FM502-CMS and PM592-ETH, no communication module)	1.2 A^2s
Galvanic isolation	Yes, PM592-ETH and FM502-CMS to other I/O bus modules
Max. power dissipation within the FM502-CMS	6.5 W (outputs unloaded)

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

For maritime applications a metal cabinet is required

Table 488: Technical data of the device

Parameter	Value
Weight FM502-CMS	215 g
Weight FM502-CMS-XC	220 g
Mounting position	Horizontal Vertical with derating: max. temperature 40 °C
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Deratings for operation of FM502-CMS-XC between +60 °C and +70 °C	No use of 24 V encoder mode. Analog inputs: maximum number of configured input channels limited to 75 % per group AI0...AI7 and AI8...AI15.
Required Terminal Base	TF501 or TF521 ↬ *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*

Table 489: Technical data of the 5 V encoder supply

Parameter	Value
Number of supplies	1
Connections	Terminal 1.7

Parameter	Value
Rated value	5 V DC (+/- 5%)
Resistance to feedback against reverse polarity	No
Resistance to feedback against 24 V signals	Yes
Output current	100 mA max.
Output diagnosis	Yes, with diagnosis LED and error message

Table 490: Technical data of the digital inputs

Parameter		Value
Number of channels		2 + 2 configurable inputs/outputs
Connections		Terminals 2.8, 2.9, 3.8, 3.9
Reference potential		Terminals 1.9, 4.9, 5.9, 6.9, 7.9 for M (0 V)
Indication of the input signals		One yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V Due to the direct connection to the output, the demagnetizing varistor is also effective at the input. This is why the difference between L+ and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. The input voltage must range from -12 V to +30 V when L+ = 24 V and from -6 V to +30 V when L+ = 30 V.
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 5 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
Shielded		1000 m
Unshielded		600 m

Table 491: Technical data of digital outputs

Parameter	Value
Number of channels per module	2 configurable inputs/outputs
Connection	Terminal 3.8, 3.9
Reference potential	Terminals 1.9, 4.9, 5.9, 6.9, 7.9 for M (0 V)

Parameter		Value
Indication of the output signal		One LED per channel
Power supply voltage		Terminals 1.8, 4.8, 5.8, 6.8, 7.8 for L+ (+24 V)
Output voltage for signal 1		L+ (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel: 500 mA at UP = 24 V	500 mA at L+ = 24 V
	Maximum value: 1 A	1 A
Leakage current with signal 0		< 0.5 mA
Demagnetization when inductive loads are switched off		With varistors integrated in the module
Switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit proof / overload proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

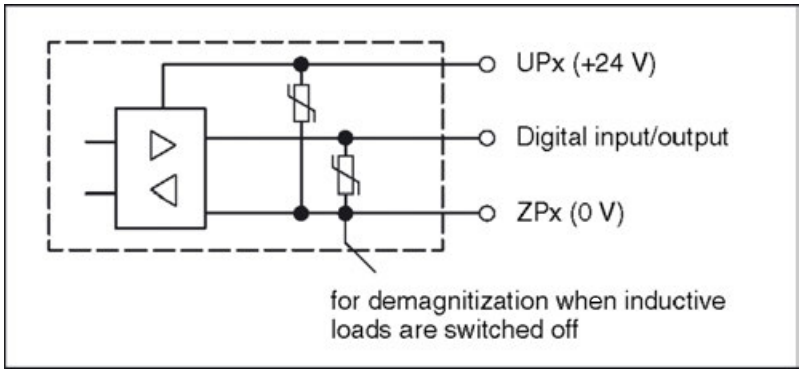


Fig. 927: Circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

Table 492: Technical data of high speed input (Encoder, A/B/Z)

Parameter	Value
Number of channels per module	3 (sampled synchronously with IEPE inputs)
Connection	Terminals 1.1, 1.2, 1.3, 1.4, 1.5, 1.6
Reference potential	Terminals 1.9, 4.9, 5.9, 6.9, 7.9 for M (0 V)
Indication of the input signals	One LED per channel

Parameter		Value		
Resolution		32 bits		
Input type		24 V DC	5 V DC	Differential RS-422 and 1 Vpp sine
Input current per channel				
	Input voltage + 24 V	Typ. 6 mA		
	Input voltage + 5 V	> 1 mA		
	Input voltage + 15 V	> 5 mA		
	Input voltage + 30 V	< 8 mA		
Input type acc. to EN61131-2		Type 1		
Input frequency max. (frequency measurement)		100 kHz (accuracy -0 %/+3 %)		
Input signal voltage		24 V DC	5 V DC	Differential
Input frequence max.		300 kHz	1 MHz	1 MHz
Signal 0		-30 V...+5 V	-30 V...+0.8 V	≤ 200 mV
Undefined signal		> +5 V...< +15 V	> +0.8 V...< +2.0 V	-
Signal 1		+15 V...+30 V	+2.0 V...+30 V	≥ +200 mV
Ripple with signal 0		Within -30 V...+5 V	Within -30 V... +0.8 V	-
Ripple with signal 1		Within +15 V...+30 V	Within +2.0 V... +30 V	-
Max. cable length, shielded (depending on sensor)		300 m	100 m	

Table 493: Technical data of the fast outputs (SI CLK output B for optical interface)

Parameter	Value
Number of channels	1
Connection	Terminals 1.3, 1.4
Reference potential	Terminals 1.9, 4.9, 5.9, 6.9, 7.9 for M (0 V)
Indication of output signal	One LED per channel, the LED is ON when SSI CLK output B is active
Differential output voltage for signal 1	> 2.4 V at 10 mA
Differential output voltage for signal 0	≤ -2.4 V at 10 mA
Output delay (0->1 or 1->0)	Max. 0.35 µs
Output current	≤ 10 mA
Switching frequency (selectable)	200 kHz, 500 kHz and 1 MHz
Short-circuit-proof/overload-proof	Yes
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes

Parameter	Value
Resistance to feedback against reverse polarity	Yes
Max. cable length, shielded (depending on sensor)	Typ. 12.5 m at 1MHz

Table 494: Technical data of the fast outputs (SSI CLK output B, RS-422 differential)

Parameter	Value
Number of channels	1
Connection	Terminals 1.3, 1.4
Reference potential	Terminals 1.9, 4.9, 5.9, 6.9, 7.9 for M (0 V)
Differential output voltage	≥ 2.4 V at 10 mA
Output delay (0->1 or 1->0)	Max. 0.35 µs
Switching frequency (selectable)	200 kHz, 500 kHz, 1 MHz
Short-circuit-proof/overload-proof	Yes
Output current limitation	Yes, automatic reactivation after short-circuit/overload
Resistance to feedback against 24 V signals	Yes
Resistance to feedback against reverse polarity	Yes
Max. cable length, shielded (depending on sensor)	100 m

Table 495: Technical data of analog inputs

Parameter	Value
Number of channels per module	16 (synchronous sampled)
Connection	Terminals 2.0...2.7, 5.0...5.1 for AI-, 3.0...3.7, 6.0...6.7 for AI+
Indication of the input signal	One bicolor LED per channel for signal and error messages.
Measurement resolution	≥ 23 Bit
Resolution	32 bits external use
Accurracy at +25 °C	≤+/-0.1 %
Accurracy over operating temperature and vibration	≤+/-0.5 %

Parameter		Value	
Sample rate/bandwidth high (0 dB)		50 kHz/20 kHz (min. -121 dB/22.5 kHz) 25 kHz/10 kHz (min. -116 dB/11.25kHz) 12.5 kHz/5 kHz (min. -116 dB/5.63 kHz) 6.25 kHz/2.5 kHz (min. -116 dB/2.81 kHz) 3.13 kHz/1.25 kHz (min. -116 dB/1.41 kHz) 1.56 kHz/0.625 kHz (min. -116 dB/0.70 kHz) 0.78 kHz/0.312 kHz (min. -120 dB/0.36 kHz) 0.39 kHz/0.156 kHz (min. -121 dB/0.18 kHz) 0.20 kHz/0.080 kHz (min. -121 dB/0.09 kHz) 0.10 kHz/0.040 kHz (min. -130 dB/0.05 kHz) selectable per channel	
Data storage		128 MB	
Measurement time		Selectable per channel	
Input type default setting		unused	
Input type (selectable per input)		IEPE	-10 V...+10 V
Bandwidth low		min. 3 dB/< 0.1 Hz	min. 3 dB/< 0.1 Hz or DC (selectable)
Dynamic range (SFDR)		> 100 dB	
SINAD (300 Hz/1 kHz sine, 50 k SPS)			
	0 dB from full scale	< -90 dB	< -95 dB
	-20 dB from full scale	< -75 dB	< -80 dB
	-40 dB from full scale	< -55 dB	< -60 dB
Input range		+2 V...+18 V	-10 V...+10 V
Measurement range		+/-6 V (DC coupled)	-10 V...+10 V
Input DC bias range, common mode range		+8 V...+12 V	+/-1 V
Current source per channel		Typ. 4.2 mA (+/- 7 % over temperature)	-
Input resistance AI- to M		Typ. 27 Ohm (PTC)	
Channel input impedance (AI+/AI-)			
	< 1 kHz	> 1 MOhm	> 2 MOhm
	5 kHz	> 100 kOhm	> 40 kOhm
	10 kHz	> 60 kOhm	> 25 kOhm
	20 kHz	> 40 kOhm	> 8 kOhm
Error detection		Short circuit, open wire	-
Max. cable length, shielded (depending on sensor)		100 m	

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP260400R0001	Function module FM502-CMS	Active
1SAP460400R0001	Function module FM502-CMS-XC, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8 Communication interface modules (S500)

Hot swap

System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

Conditions for hot swapping

- *Digital outputs are not under load.*
- *Input/output voltages above safety extra low voltage/ protective extra low voltages (SELV/PELV) are switched off.*
- *Modules are completely plugged on the terminal unit with both snap fit engaged before switching on loads or input/output voltage.*

Hot swap

Further information about hot swap: ↬ Chapter 1.6.4.1.7 “Hot swap” on page 5463.

1.6.2.8.1 Compatibility of communication modules and communication interface modules

Table 496: Modbus TCP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard Ethernet interface	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O
CM597-ETH	CI521-MODTCP CI522-MODTCP	x	x	--	high availability, remote I/O

Table 497: PROFIBUS DP

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM592-DP master	CI541-DP CI542-DP	x	x	--	remote I/O
CM592-DP master	CI541-DP CI542-DP	x	--	--	hot-swap I/O

Table 498: PROFINET IO RT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI501-PNIO CI502-PNIO	x	--	--	hot swap I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	x	x	remote I/O, safety I/O
CM579-PNIO controller	CI504-PNIO CI506-PNIO	x	--	--	hot swap I/O

Table 499: CANopen

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM598-CN master	CI581-CN CI582-CN	x	x	--	remote I/O

Table 500: EtherCAT

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
CM579-ETHCAT master	CI511-ETHCAT CI512-ETHCAT	x	x	--	remote I/O

Table 501: CS31 bus

Communication module	Communication interface module	I/O expansion module S500	I/O expansion module S500-eCo	I/O expansion module S500-S	Applications
Onboard COM1 interface	DC551-CS31 CI592-CS31	x	x	--	remote I/O
Onboard COM1 interface	CI590-CS31-HA	x	--	--	high availability
CM574-RS	DC551-CS31 CI592-CS31	x	x	--	remote I/O
CM574-RS	CI590-CS31-HA	x	--	--	high availability

1.6.2.8.2 CANopen

Comparison CI581 and CI582

CI581/CI582:
Technical data

Parameter	Value
Interface	CAN
Protocol	CANopen
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates	10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection	Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block
Processor	Hilscher NETX 100
Expandability	Max. 10 S500 I/O modules
State display	Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus

Parameter		Value
Adjusting elements		2 rotary switches for generation of the node address
Ambient temperature		System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

The difference of those devices can be found in their input and output characteristics.

CI581-CN: Input/ Output characteristics

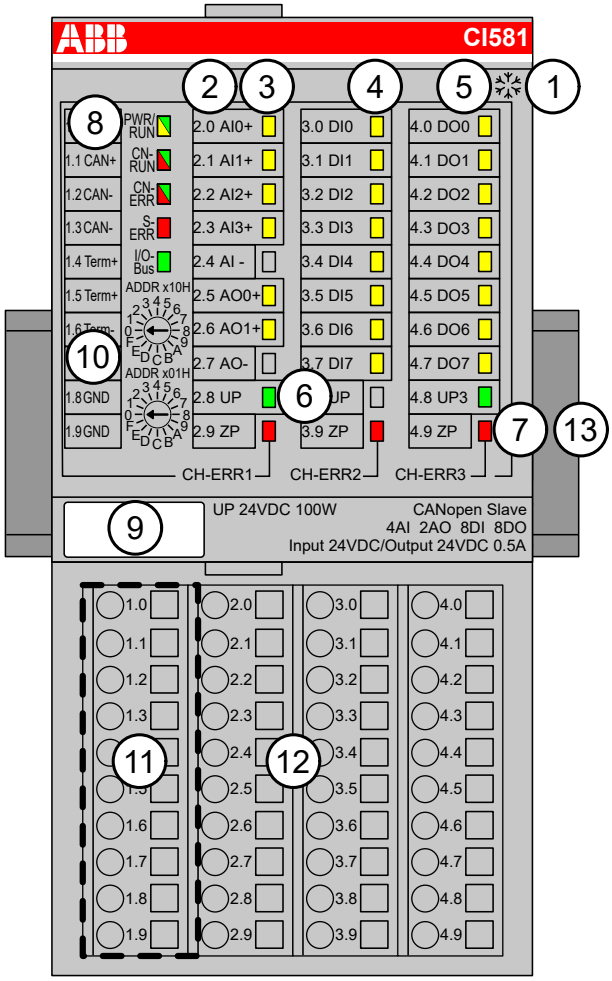
Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC; delay time configurable via software) 8 digital transistor outputs (24 V DC, 0.5 A max.) 4 analog inputs, configurable as: ● -10 V...+10 V ● 0 V...+10 V ● -10 V...+10 V (differential voltage) ● 0 mA...20 mA ● 4 mA...20 mA ● Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire) ● 24 V digital input function 2 analog outputs, configurable as: ● -10 V...+10 V ● 0 mA...20 mA ● 4 mA...20 mA
Resolution of the analog channels	12 bits
Fast counter	Integrated, configurable operating modes

CI582-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC) 8 digital transistor outputs (24 V DC, 0.5 A max.) 8 configurable digital inputs/outputs (24 V DC, 0.5 A max.)

CI581-CN

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal No. and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, CN-RUN, CN-ERR, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the CANopen Node ID
11 10 terminals to connect the CANopen bus signals
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The CANopen communication interface module CI581-CN is used as decentralized I/O module in CANopen networks. Depending on the used terminal unit the network connection is performed either via 9-pin female D-sub or via 10 terminals (screw or spring terminals) which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (2.0...2.3)
- 2 analog outputs (2.5...2.6)
- 8 digital inputs 24 V DC in 1 group (3.0...3.7)
- 8 digital outputs 24 V DC in 1 group (4.0...4.7)

The inputs/outputs are galvanically isolated from the CANopen network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter		Value
Interface		CAN
Protocol		CANopen
Power supply		From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached		Through the I/O bus interface (I/O bus)
Rotary switches		For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays		For system displays, signal states, errors and power supply
External supply voltage		Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates		10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection		Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block
Processor		Hilscher NETX 100
Expandability		Max. 10 S500 I/O modules
State display		Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus
Adjusting elements		2 rotary switches for generation of the node address
Ambient temperature		System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A

Parameter		Value
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

CI581-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC; delay time configurable via software) 8 digital transistor outputs (24 V DC, 0.5 A max.) 4 analog inputs, configurable as: ● -10 V...+10 V ● 0 V...+10 V ● -10 V...+10 V (differential voltage) ● 0 mA...20 mA ● 4 mA...20 mA ● Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire) ● 24 V digital input function 2 analog outputs, configurable as: ● -10 V...+10 V ● 0 mA...20 mA ● 4 mA...20 mA
Resolution of the analog channels	12 bits
Fast counter	Integrated, configurable operating modes

Connections

The CANopen communication interface module is plugged on the I/O terminal units TU517 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* or TU518 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* and accordingly TU509 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* or TU510 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099*. Properly position the module and press until it locks in place.

The connection of the I/O channels is established using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC

Terminal 4.8: process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to the digital outputs!

Reason: External voltages at an output or several outputs may cause other outputs to be supplied via that voltage instead of voltage UP3 (reverse voltage). This ist not the intended use.

CAUTION!

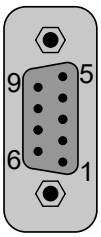
Risk of malfunctions by unintended use!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is connected at the outputs DO0..DO7 and DC0..DC7.

Possibilities of connection

Mounting on terminal units TU509 or TU510

The assignment of the 9-pin female D-sub for the CANopen signals



1	---	Reserved
2	CAN-	Inverted signal of the CAN bus
3	CAN_GND	Ground potential of the CAN bus
4	---	Reserved
5	---	Reserved
6	---	Reserved
7	CAN+	Non-inverted signal of the CAN bus
8	---	Reserved
9	---	Reserved
Shield	Cable shield	Functional earth

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

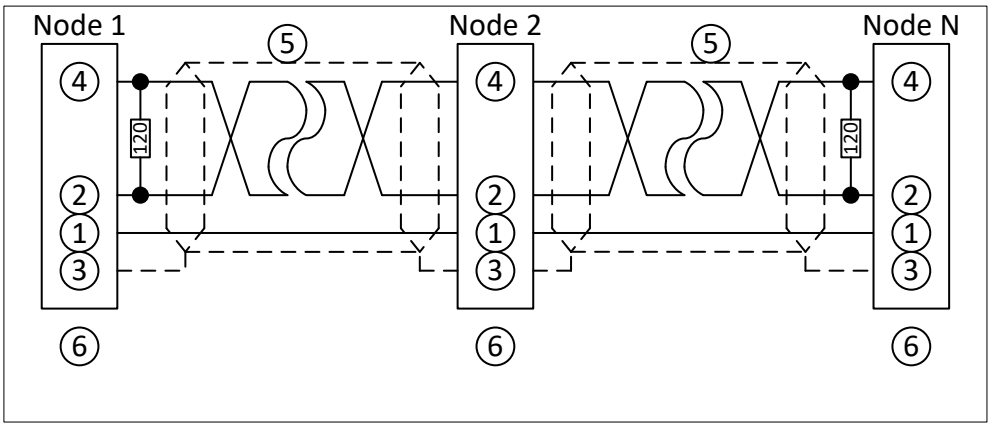


Fig. 928: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

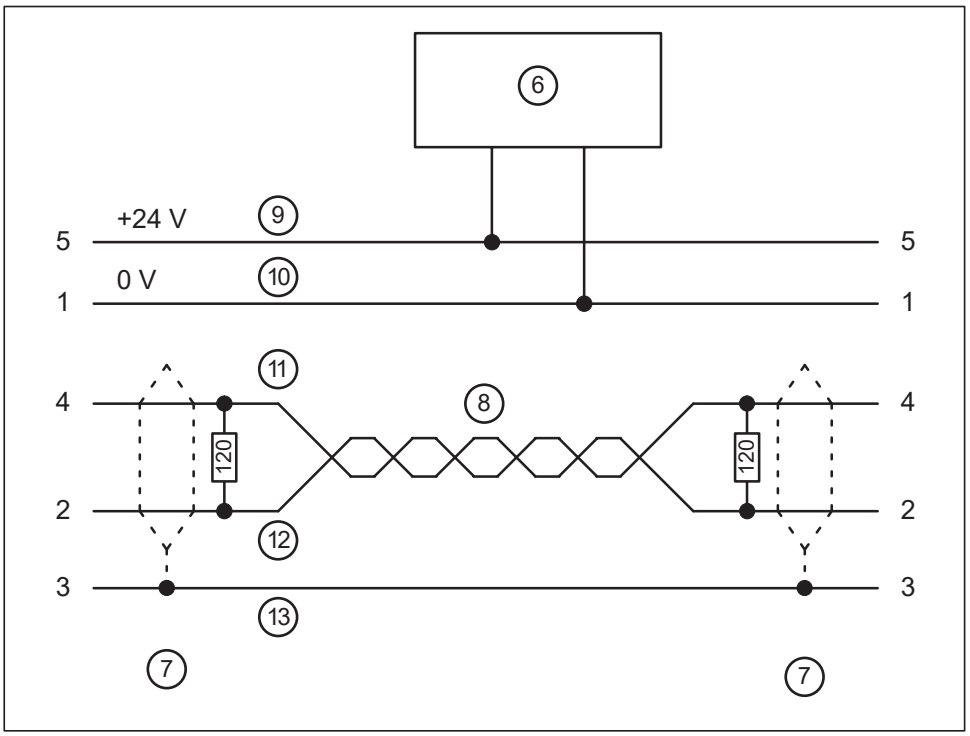


Fig. 929: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to Chapter 1.6.3.6.1 “System data AC500” on page 5313.

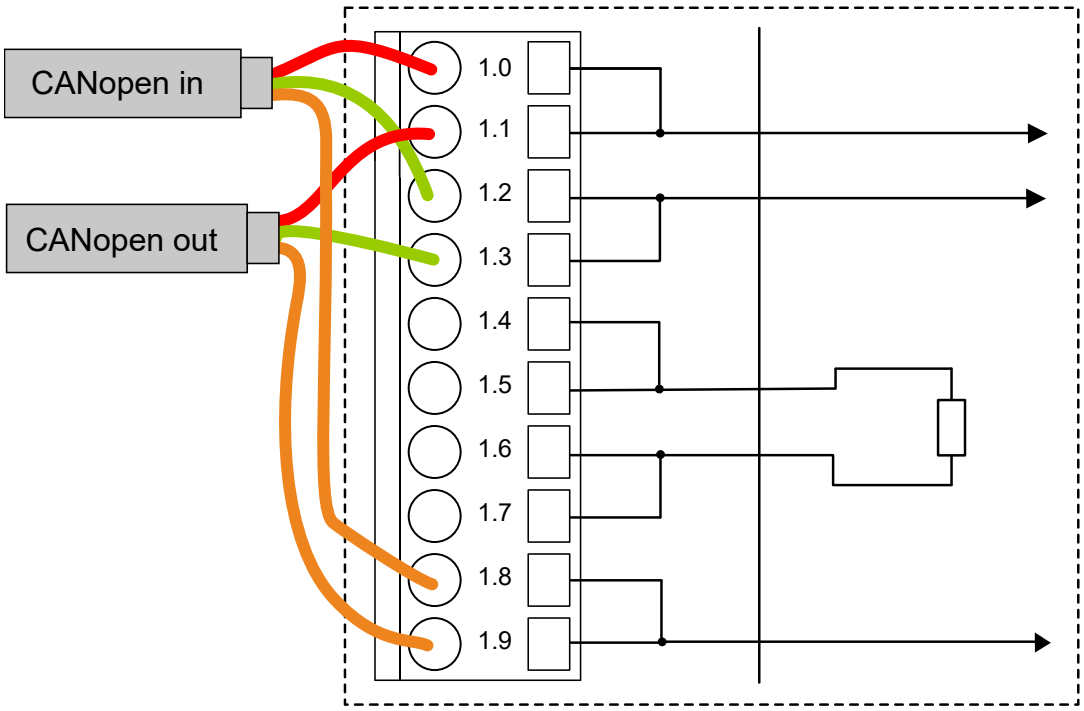
Mounting on terminal units TU517 or TU518

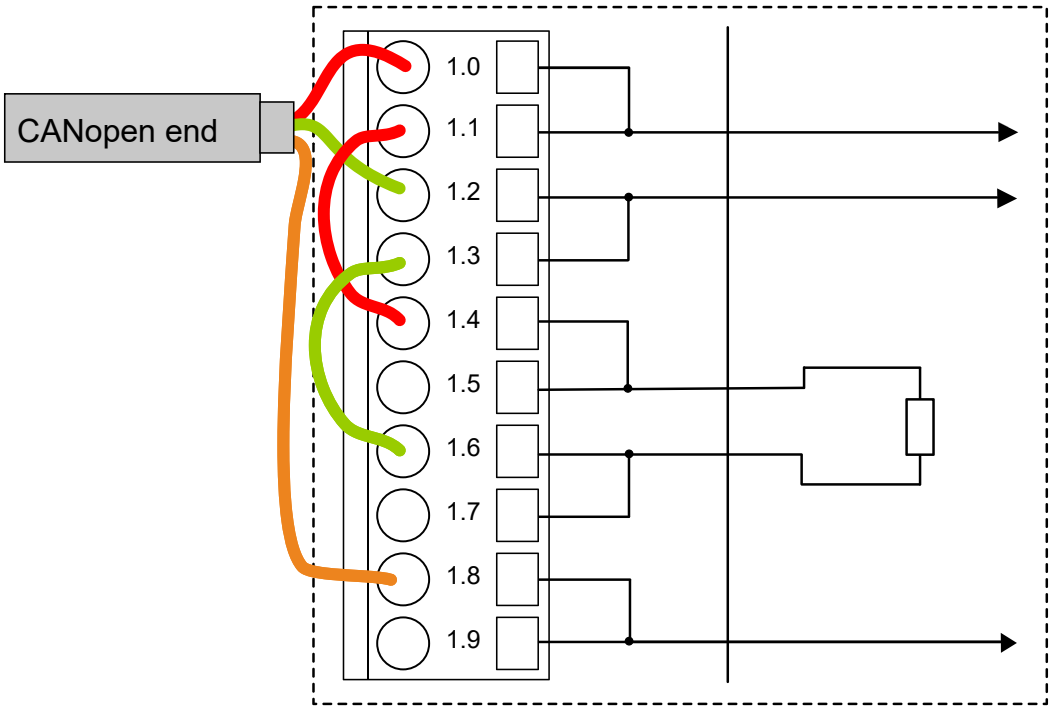
Table 502: Assignment of the terminals

Terminal	Signal	Description
1.0	CAN+	Non-inverted signal of the CAN bus
1.1	CAN+	Non-inverted signal of the CAN bus
1.2	CAN-	Inverted signal of the CAN bus
1.3	CAN-	Inverted signal of the CAN bus
1.4	Term+	CAN bus termination for CAN+ (for bus termination, Term+ must be connected with CAN+)
1.5	Term+	CAN bus termination for CAN+ (connecting alternative for terminal 1.4)
1.6	Term-	CAN bus termination for CAN- (for bus termination, Term- must be connected with CAN-)
1.7	Term-	CAN bus termination for CAN- (connecting alternative for terminal 1.6)
1.8	CAN-GND	Ground potential of the CAN bus
1.9	CAN-GND	Ground potential of the CAN bus

At the line ends of a bus segment, terminating resistors must be connected. If TU517 or TU518 is used, the bus terminating resistors can be enabled by connecting the terminals Term+ and Term- to the data lines CAN+ and CAN- (no external terminating resistors are required, see figure below).

The following figures show the different connection options for the CANopen communication interface module:





In the case of TU517/TU518, the terminating resistors are not located inside the TU but inside the communication interface module CI581-CN. Hence, when removing the device from the TU, the bus terminating resistors are no longer connected to the bus. The bus itself will not be disconnected if a device is removed.

The grounding of the shield should take place at the switchgear cabinet. Please refer to the AC500 System-Data Chapter 1.6.3.6.1 “System data AC500” on page 5313.

Table 503: Assignment of the other terminals

Terminal	Signal	Description
2.0	AI0+	Positive pole of analog input signal 0
2.1	AI1+	Positive pole of analog input signal 1
2.2	AI2+	Positive pole of analog input signal 2
2.3	AI3+	Positive pole of analog input signal 3
2.4	AI-	Negative pole of analog input signals 0 to 3
2.5	AO0+	Positive pole of analog output signal 0
2.6	AO1+	Positive pole of analog output signal 1
2.7	AI-	Negative pole of analog output signals 0 and 1
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI0	Signal of the digital input DI0
3.1	DI1	Signal of the digital input DI1
3.2	DI2	Signal of the digital input DI2
3.3	DI3	Signal of the digital input DI3

Terminal	Signal	Description
3.4	DI4	Signal of the digital input DI4
3.5	DI5	Signal of the digital input DI5
3.6	DI6	Signal of the digital input DI6
3.7	DI7	Signal of the digital input DI7
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO0	Signal of the digital output DO0
4.1	DO1	Signal of the digital output DO1
4.2	DO2	Signal of the digital output DO2
4.3	DO3	Signal of the digital output DO3
4.4	DO4	Signal of the digital output DO4
4.5	DO5	Signal of the digital output DO5
4.6	DO6	Signal of the digital output DO6
4.7	DO7	Signal of the digital output DO7
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

Connection of CANopen communication interface module CI581-CN:

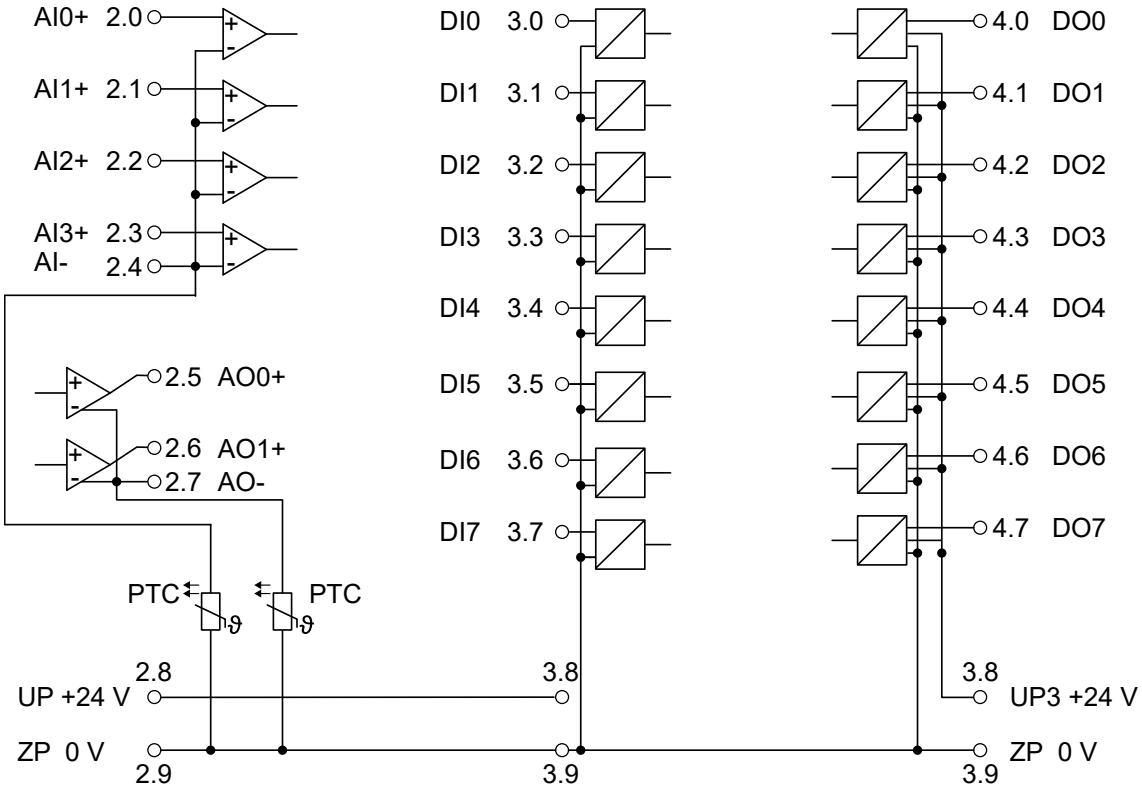


Fig. 930: Connection of the communication interface module CI581-CN

The module provides several diagnosis functions ↳ *Chapter 1.6.2.8.2.2.8 “Diagnosis” on page 4710*.

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

The meaning of the LEDs is described in the section for the state LEDs ↳ *Chapter 1.6.2.8.2.2.9 “State LEDs” on page 4714*.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
62.5 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.

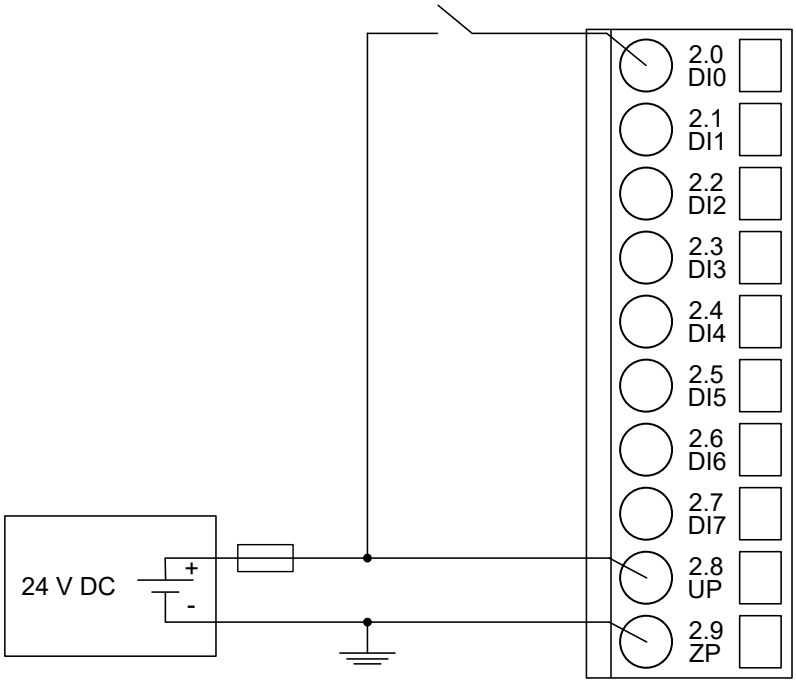


Fig. 931: Connection of the digital inputs to the module CI581-CN

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

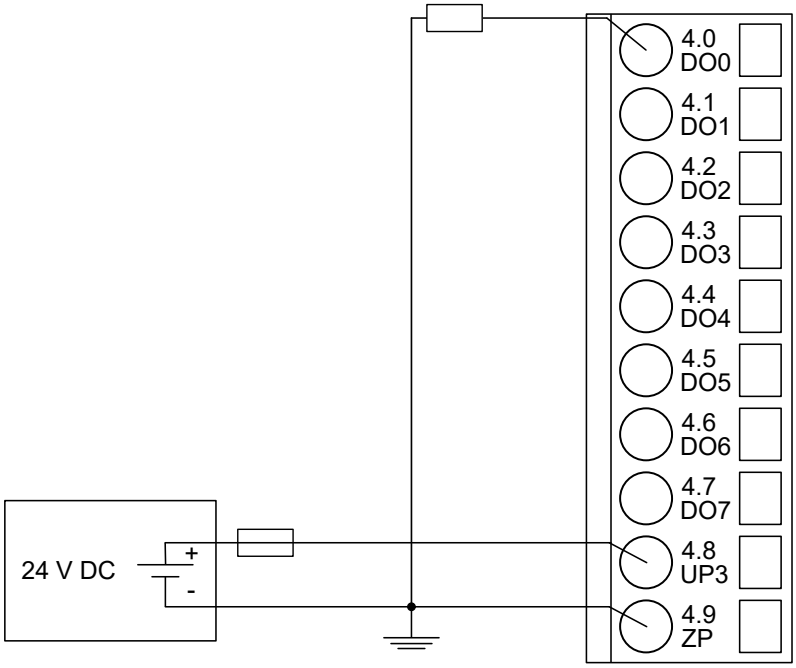


Fig. 932: Connection of configurable digital inputs/outputs to the module CI581-CN

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow to build the necessary voltage drop for the evaluation. For this, the module CI581-CN provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

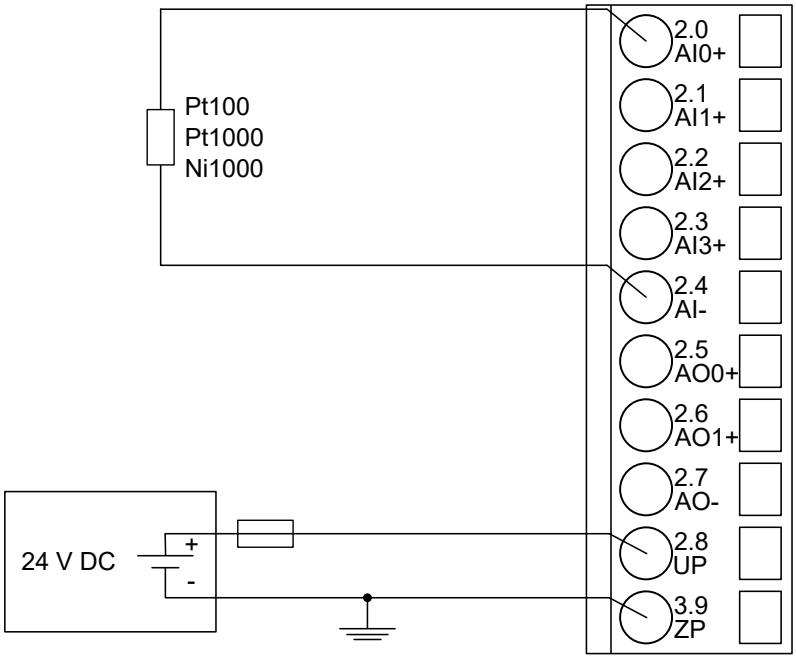


Fig. 933: Connection of resistance thermometers in 2-wire configuration to the analog inputs

Pt100	2-wire configuration, 1 channel used
Pt1000	2-wire configuration, 1 channel used
Ni1000	2-wire configuration, 1 channel used

For the measuring ranges that can be configured, please refer to sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

The module CI581-CN performs a linearization of the resistance characteristic.

To avoid error messages, configure unused analog input channels as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI581-CN provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

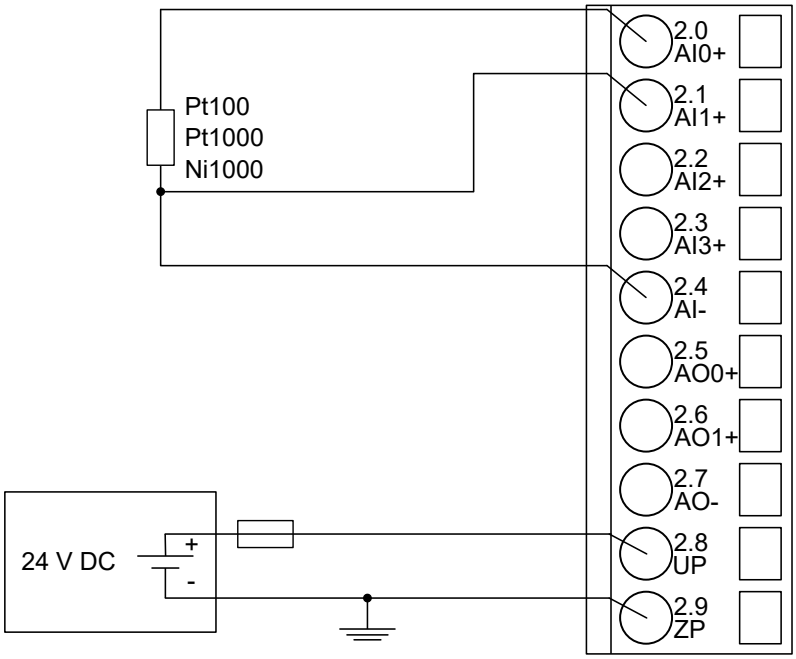


Fig. 934: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	3-wire configuration, 2 channels used
Pt1000	3-wire configuration, 2 channels used
Ni1000	3-wire configuration, 2 channels used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

The module CI581-CN performs a linearization of the resistance characteristic.

To avoid error messages, configure unused analog input channels as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

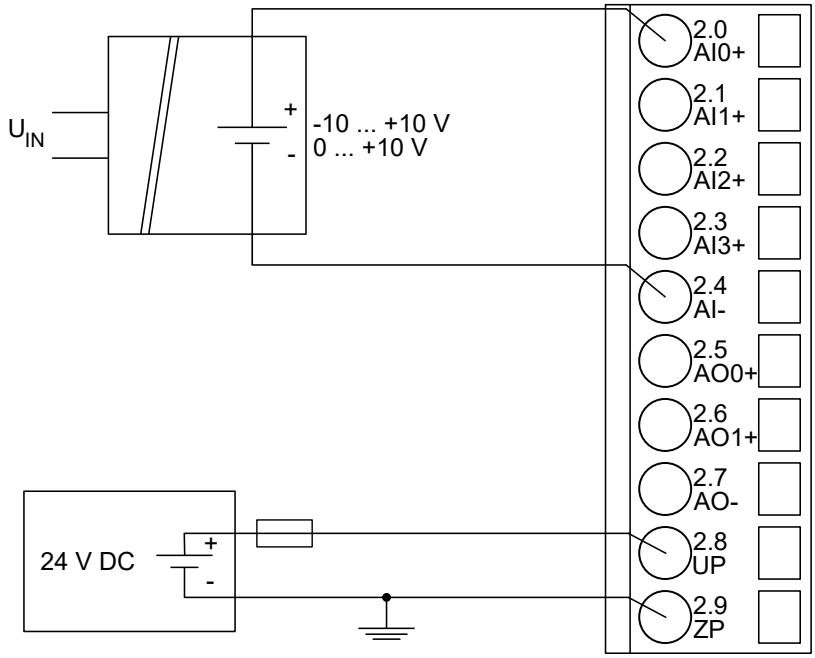


Fig. 935: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

To avoid error messages, configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

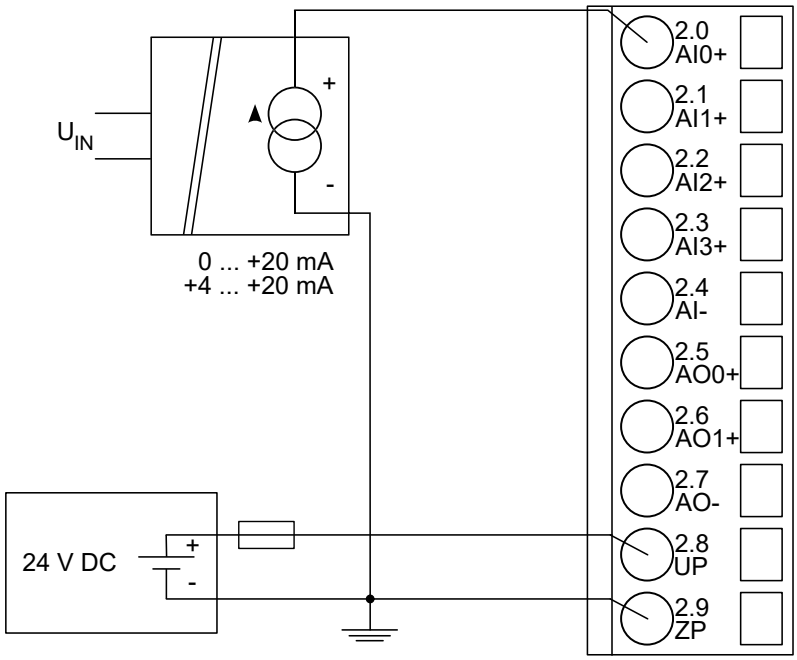


Fig. 936: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

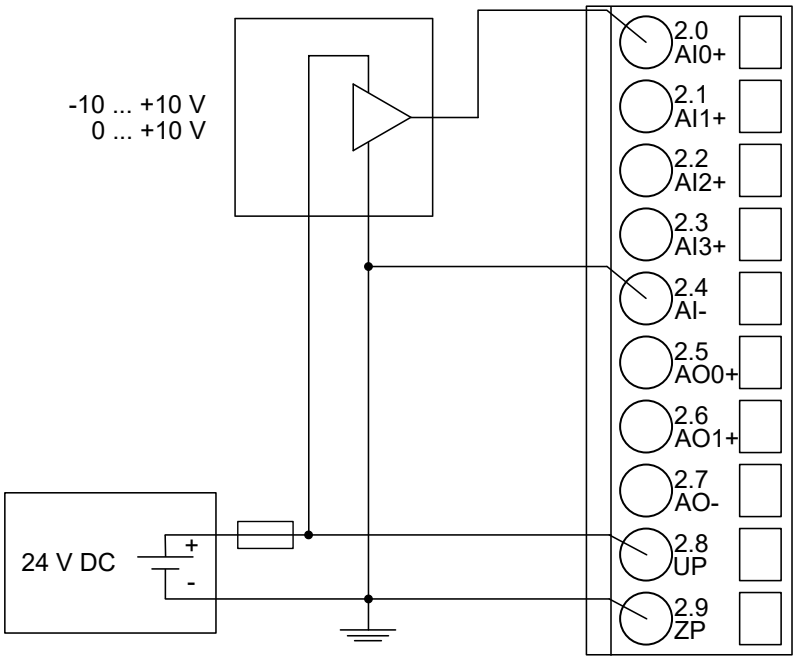


Fig. 937: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

For the measuring ranges that can be configured, plese refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

To avoid error messages, configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

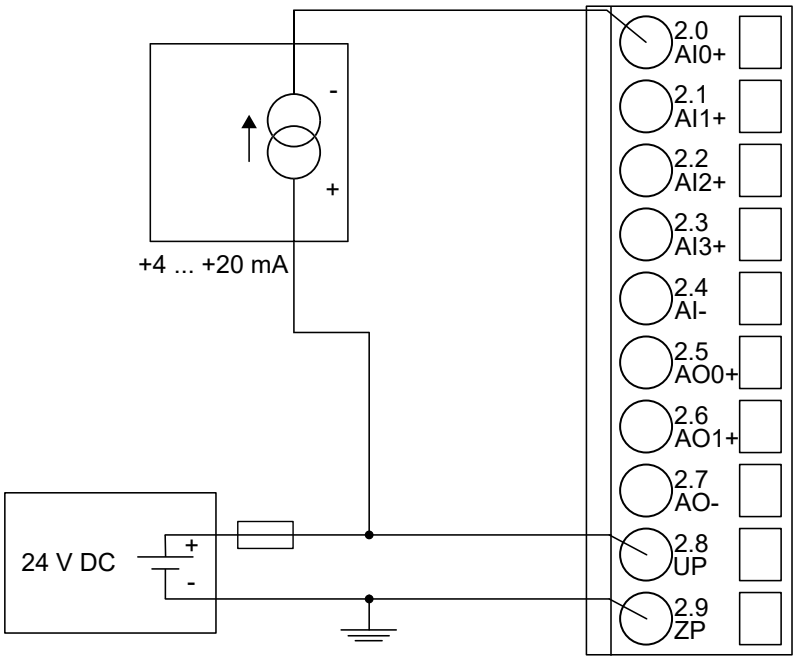


Fig. 938: Connection of passive-type analog sensors (current) to the analog inputs

Current	4...20 mA	1 channel used

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Only use sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt Zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful if analog sensors which are remotely non-isolated (e.g. the negative terminal is remotely grounded) are used.

Using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

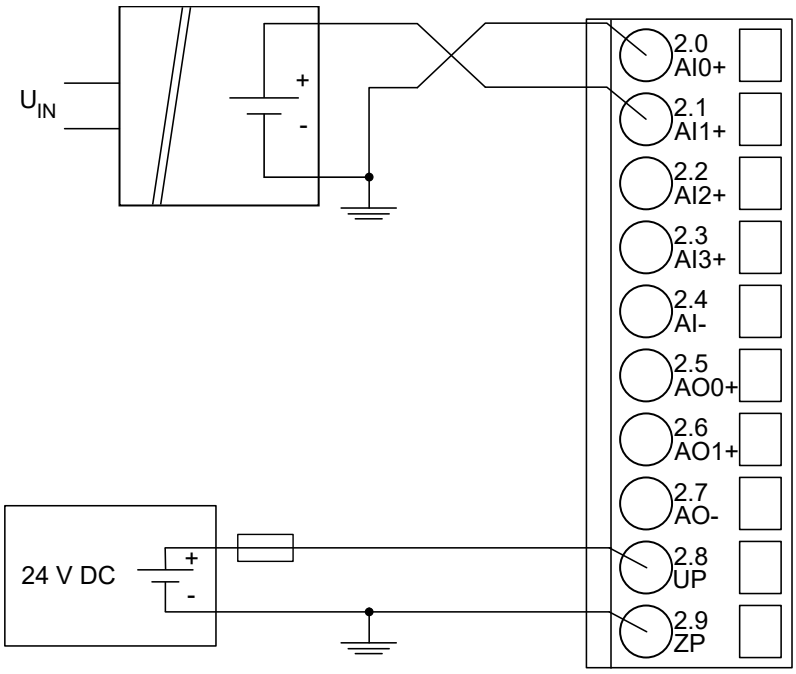


Fig. 939: Connection of active-type analog sensors (voltage) to differential analog inputs

Voltage	0...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↳ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↳ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

To avoid error messages, configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

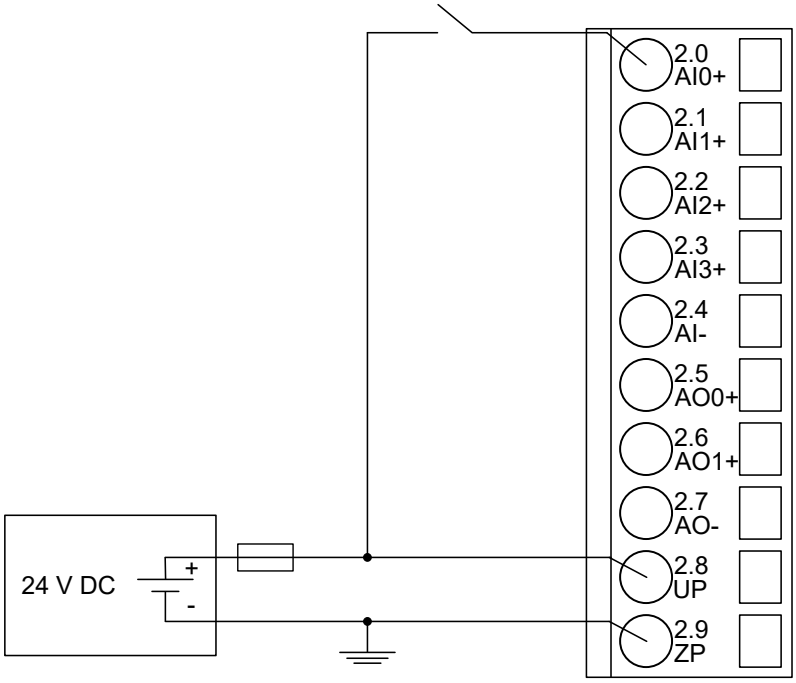


Fig. 940: Use of analog inputs as digital inputs

Digital input	24 V	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges ↬ *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization ↬ *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

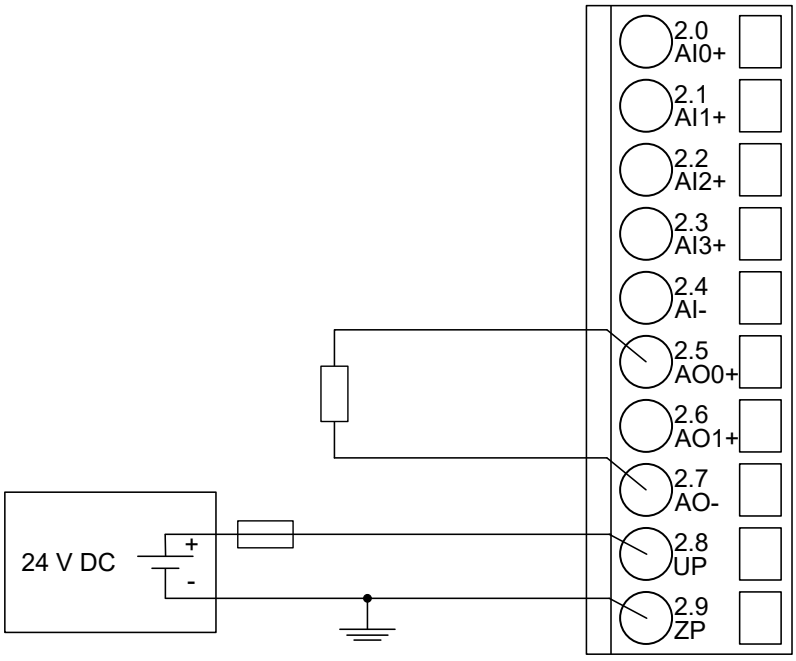


Fig. 941: Connection of analog output loads (voltage)

Voltage	-10 V...+10 V	Load ± 10 mA max.	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

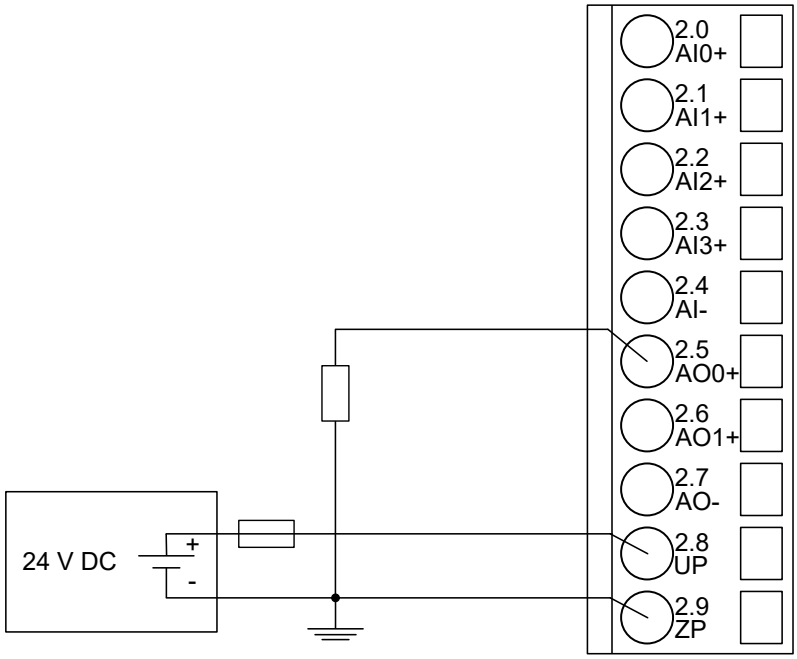


Fig. 942: Connection of analog output loads (current)

Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

For the measuring ranges that can be configured, please refer to the sections Measuring Ranges *Chapter 1.6.2.8.2.2.10 “Measuring ranges” on page 4716* and Parameterization *Chapter 1.6.2.8.2.2.7 “Parameterization” on page 4706*.

Unused analog outputs can be left open-circuited.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

A detailed description concerning addressing can be found in the documentation of ABB Control Builder Plus Software.

The CANopen communication interface module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The range of permitted CANopen slave addresses is 1 to 127. Setting a higher address (> 128) does not lead to an error response, but results in a special mode (DS401). In this special mode, the device creates the node address by subtracting the value 128 from the address switch's value.

I/O configuration

The CI582-CN CANopen bus configuration is handled by CANopen master with the exception of the slave node ID (via rotary switches) and the transmision rate (automatic detection).

The digital I/O channels and the fast counter are configured via software.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C84	WORD	0x1C84
Parameter length	Internal	54	BYTE	54
Error LED / Failsafe function (table error LED / Failsafe function ↳ *Further information on page 4706*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved	0	0	ARRAY of 24 BYTES	
Check supply (UP and UP3)	On	0	BYTE	
	Off	1		1
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission

[2]) For a description of the counter operating modes, please refer to the fast counter section ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

Table 504: Settings "Error LED / Failsafe function"

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode on *)
*) The parameters Behaviour analog outputs at communication error and Behaviour digital outputs at communication error are only evaluated if the failsafe function is enabled.	

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behavior analog outputs at communication error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0
*) The parameter Behavior analog outputs at communication error is only analyzed if the failsafe mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Operation modes of analog inputs	Operation modes of analog inputs	BYTE	0
Input 0, Check channel	Settings channel monitoring	Settings channel monitoring	BYTE	0
:	:	:	:	:
:	:	:	:	:

Name	Value	Internal value	Internal value, type	Default
Input 3, Channel configuration	Operation modes of analog inputs	Operation modes of analog inputs	BYTE	0
Input 3, Check channel	Settings channel monitoring	Settings channel monitoring	BYTE	0

Table 505: Channel configuration - Operating modes of the analog inputs

Internal Value	Operating Modes (individually configurable)
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 506: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Operation modes of analog outputs	Operation modes of analog outputs	BYTE	0
Output 0, Check channel	Channel monitoring	Channel monitoring	BYTE	0
Output 0, Substitute value	Substitute value	Substitute value	WORD	0
Output 1, Channel configuration	Operation modes of analog outputs	Operation modes of analog outputs	BYTE	0
Output 1, Check channel	Channel monitoring	Channel monitoring	BYTE	0
Output 1, Substitute value	Substitute value	Substitute value	WORD	0

Table 507: Channel configuration - Operating modes of the analog outputs

Internal value	Operating Modes (individually configurable)
0 (default)	Not used
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 508: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 509: Substitute value

Intended Behavior of Output Channel when the Control System Stops	Required Setting of the Module Parameter "Behavior of Outputs in Case of a Communication Error"	Required Setting of the Channel Parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration

Intended Behavior of Output Channel when the Control System Stops	Required Setting of the Module Parameter "Behavior of Outputs in Case of a Communication Error"	Required Setting of the Channel Parameter "Substitute value"
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behavior digital outputs at communcation error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x00
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	Off 0x00

[1]) The parameter Behavior digital outputs at communcation error is only analyzed if the failsafe mode is ON.

[2]) The state "externally voltage detected" appears if the output of a channel DC0..DC7 is to be switched on while an external voltage is connected *Chapter 1.6.2.8.2.2.3 “Connections” on page 4689*. In this case, the start-up is disabled as long as the external voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".

Diagnosis

Structure of the Diagnosis Block via CANOM_NODE_DIAG. *Chapter 1.5.4.7.1.8 “CANOM_NODE_DIAG” on page 933*

Byte Number	Description	Possible Values
1	Diagnosis byte, slot number	31 = CI581-CN (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])		Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module		Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low		Check process supply voltage

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Voltage feedback on deactivated digital output [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0..1	4	Analog value overflow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500, the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = position of the communication module;14 = I/O bus; 31 = module itself The identifier is not contained in the CI541-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = module itself; 1..10 = decentralized communication interface module
[3])	With "Module" the following allocation applies: 31 = module itself Channel error: module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears if external voltages at one or more terminals DO0..DO7 cause other digital outputs to be fed by that voltage (voltage feedback, description in 'Connections' ↯ *Chapter 1.6.2.8.2.2.3 “Connections” on page 4689*). All outputs of the digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0..DO7 has overrun the process supply voltage UP3 (description in 'Connections' ↯ *Chapter 1.6.2.8.2.2.3 “Connections” on page 4689*). Diagnosis message appears for the whole module.
[6])	This message appears if the output of a channel DO0..DO7 is to be switched on while an external voltage is connected. In this case, start-up is disabled while the external voltage is connected. Otherwise, this could produce reverse voltage flowing from this output to other digital outputs. This diagnosis message appears for each channel.
[7])	Short circuit: After a short circuit has been detected, the output is deactivated for 100ms seconds. Subsequently, a new start-up will be executed. This diagnosis message appears for each channel.

State LEDs

The state LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, CN-RUN, CN-ERR, S-ERR and I/O bus) show the operation states of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O controller	Start-up / preparing communication
	Yellow	---	---	---

LED	Color	OFF	ON	Flashing
CN-RUN	Green	---	Device configured, CANopen bus in OPERATIONAL state and cyclic data exchange running	Flashing: CANopen bus in PRE-OPERATIONAL state and slave is being configured Single flash: CANopen bus in STOPPED state. Flickering: Autodetect is active
CN-ERR	Red	No system error	CANopen Bus is OFF	Flashing: Configuration error Single flash: error counter overflow due to too many error frames Double flash: A node-guard or a heartbeat event occurred Flickering: Autodetect is active
S-ERR	Red	No error	Internal error	--
I/O bus	Green	No decentralized I/O modules connected or communication error	Decentralized I/O modules connected and operational	---

States of the 27 process LEDs:

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--

LED	Color	OFF	ON	Flashing
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
			800 : 701	0320 : 02BD
Normal range	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 3.0 to 3.7
Reference potential for all inputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC
Signal 0	-3 V...+5 V
Undefined signal	> +5 V...< +15 V
Signal 1	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V
Ripple with signal 1	Within +15 V...+30 V
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA

Parameter	Value
Input voltage +15 V	> 2 mA
Input voltage +30 V	< 8 mA
Max. cable length	
Shielded	1000 m
Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DO0 to DO7	Terminals 4.0 to 4.7
Reference potential for all outputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
Rated value per channel	500 mA at UP3 = 24 V
Max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA
Fuse for UP3	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

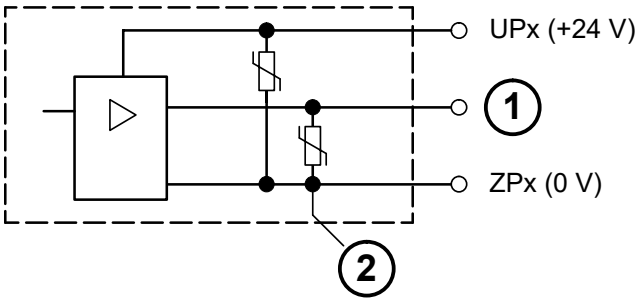


Fig. 943: Digital input/output (circuit diagram)

1	Digital output
2	Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 2.0 to2.3
Reference potential for AI0+ to AI3+		Terminal 2.4 (AI-) for voltage and RTD measurement Terminal 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10...+10 V
Galvanic isolation		Against CANopen Bus
Configurability		0...10 V, -10...+10 V, 0/4...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C

Parameter	Value
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current ↳ *Chapter 1.6.2.8.2.2.10.1 “Input ranges voltage, current and digital input” on page 4716* and Digital Input and IInput range resistance temperature detector ↳ *Chapter 1.6.2.8.2.2.10.2 “Input ranges resistance temperature detector” on page 4716*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 2.0 to 2.3
Reference potential for the inputs		Terminals 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 VDC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V...+15 V
	Signal 1	+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 2.7 (AO-) for voltage output Terminal 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current

Parameter		Value
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10...+10 V, 0...20 mA, 4...20 mA (each output can be configured individually)
Output resistance (load), as current output		0...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		See *Chapter 1.6.2.8.2.2.10.3 “Output ranges voltage and current” on page 4717*
Unused outputs		Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI0), 3.1 (DI1)
Used outputs	Terminal 4.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	Fast Counter *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
Operating modes	Operating modes *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

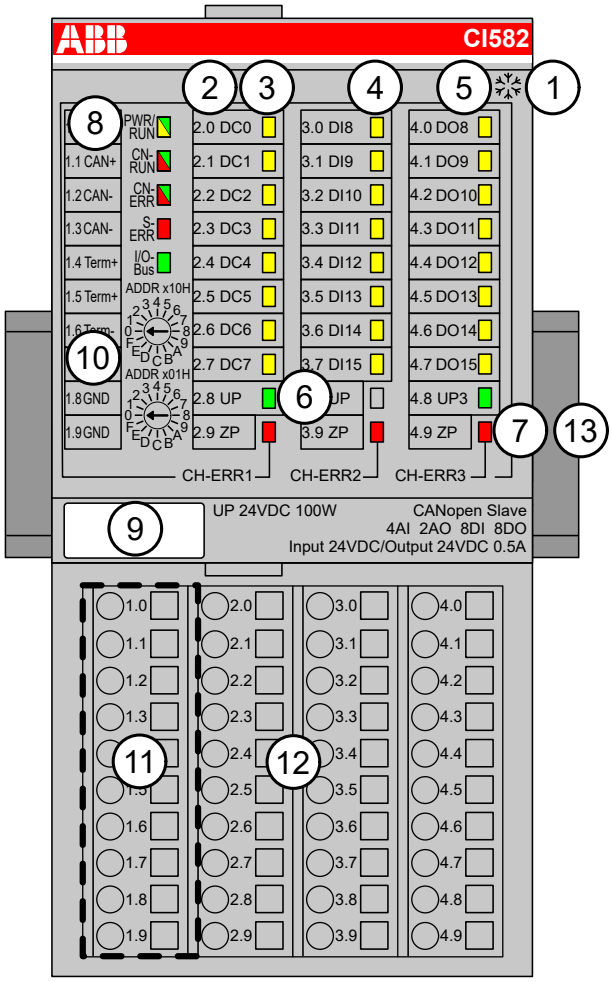
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 228 100 R0001	CI581-CN, CANopen communication interface module with 8 DI, 8 DO, 4 AI and 2 AO	Active
1SAP 428 100 R0001	CI581-CN-XC, CANopen communication interface module with 8 DI, 8 DO, 4 AI and 2 AO, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CI582-CN

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for use in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, CN-RUN, CN-ERR, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the CANopen node ID

11 10 terminals to connect the CANopen bus signals
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The CANopen communication interface module CI582-CN is used as decentralized I/O module in CANopen networks. Depending on the terminal unit used, the network connection is performed either via a female 9-pin D-sub connector or via 10 terminals (screw or spring terminals) which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the CANopen network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	CAN
Protocol	CANopen
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the CANopen Node ID for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Transmission rates	10 / 20 / 50 / 125 / 250 / 500 / 800 kbit/s 1 Mbit/s Auto transmission rate detection is supported
Bus connection	Depending on used terminal unit TU510: 9-pin D-sub connector TU518: 10-pin terminal block
Processor	Hilscher NETX 100
Expandability	Max. 10 S500 I/O modules
State display	Module state: PWR/RUN, CN-RUN, CN-ERR, E-ERR, I/O bus
Adjusting elements	2 rotary switches for generation of the node address

Parameter		Value
Ambient temperature		System data AC500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* System data AC500 XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*
Current consumption		UP: 0.2 A UP3: 0.06 A + 0.5 A max. per output
Weight (without terminal unit)		Ca. 125 g
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	CANopen interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the CANopen Node ID identifier		With 2 rotary switches at the front side of the module
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V
Required terminal unit		TU509, TU510, TU517 or TU518 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

CI582-CN: Input/ Output characteristics

Parameter	Value
Inputs and outputs	8 digital inputs (24 V DC) 8 digital transistor outputs (24 V DC, 0.5 A max.) 8 configurable digital inputs/outputs (24 V DC, 0.5 A max.)

Connections

The CANopen communication interface module is plugged on the I/O terminal units TU517 ↳ *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* or TU518 ↳ *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* and accordingly TU509 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* or TU510 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099*. Properly position the module and press until it locks in place.

The connection of the I/O channels is established using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8, 3.8, 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

Terminals 2.8 and 3.8: process supply voltage UP = +24 V DC

Terminal 4.8: process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: process supply voltage ZP = 0 V

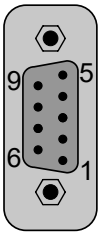
For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Possibilities of connection

Mounting on terminal units TU509 or TU510

The assignment of the 9-pin female D-sub for the CANopen signals



1	---	Reserved
2	CAN-	Inverted signal of the CAN bus
3	CAN_GND	Ground potential of the CAN bus
4	---	Reserved
5	---	Reserved
6	---	Reserved
7	CAN+	Non-inverted signal of the CAN bus

	8	---	Reserved
	9	---	Reserved
	Shield	Cable shield	Functional earth

Bus terminating resistors

The ends of the data lines have to be terminated with a 120 Ω bus terminating resistor. The bus terminating resistor is usually installed directly at the bus connector.

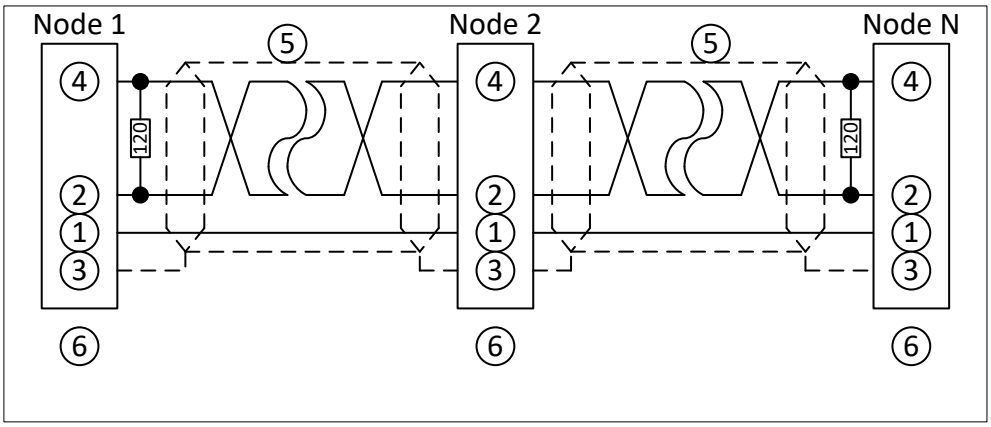


Fig. 944: CANopen interface, bus terminating resistors connected to the line ends

1	CAN_GND
2	CAN_L
3	Shield
4	CAN_H
5	Data line, shielded twisted pair
6	COMBICON connection, CANopen interface

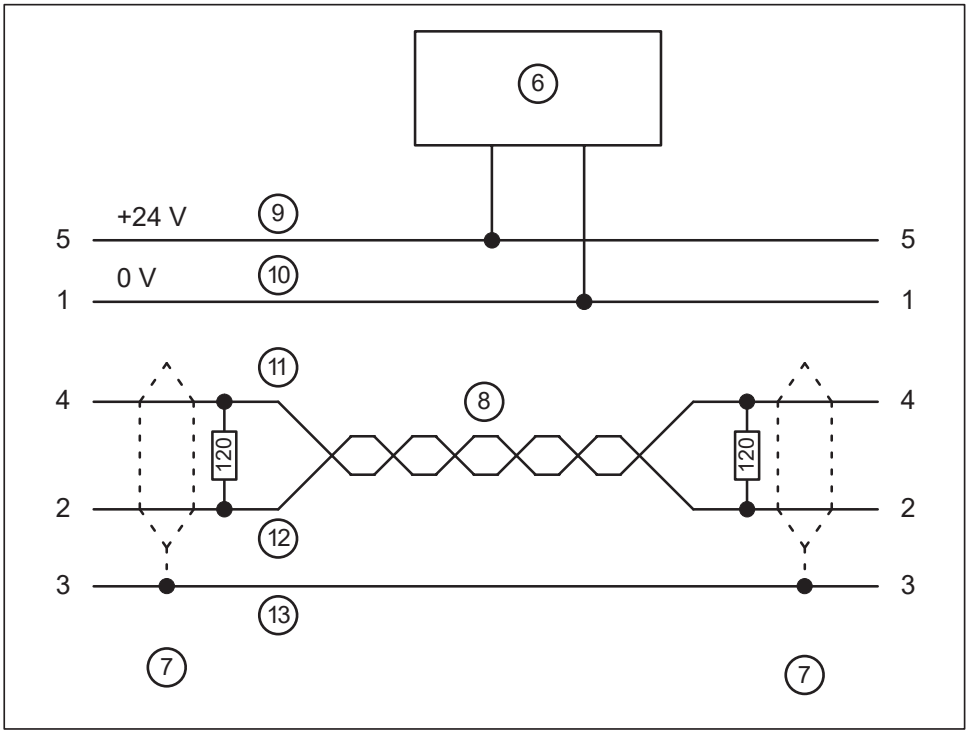


Fig. 945: DeviceNet interface, bus terminating resistors connected to the line ends

6	DeviceNet power supply
7	COMBICON connection, DeviceNet interface
8	Data lines, twisted pair cables
9	red
10	black
11	white
12	blue
13	bare

The grounding of the shield should take place at the switchgear. Please refer to *Chapter 1.6.3.6.1 “System data AC500” on page 5313.*

Mounting on terminal units TU517 or TU518

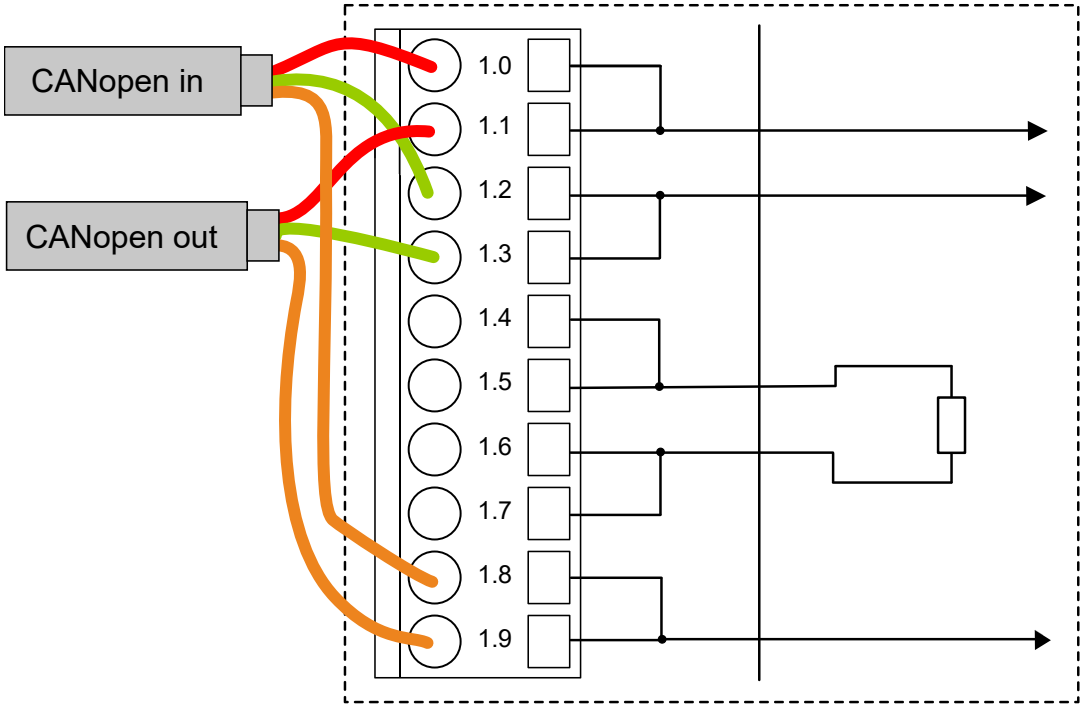
Table 510: Assignment of the terminals

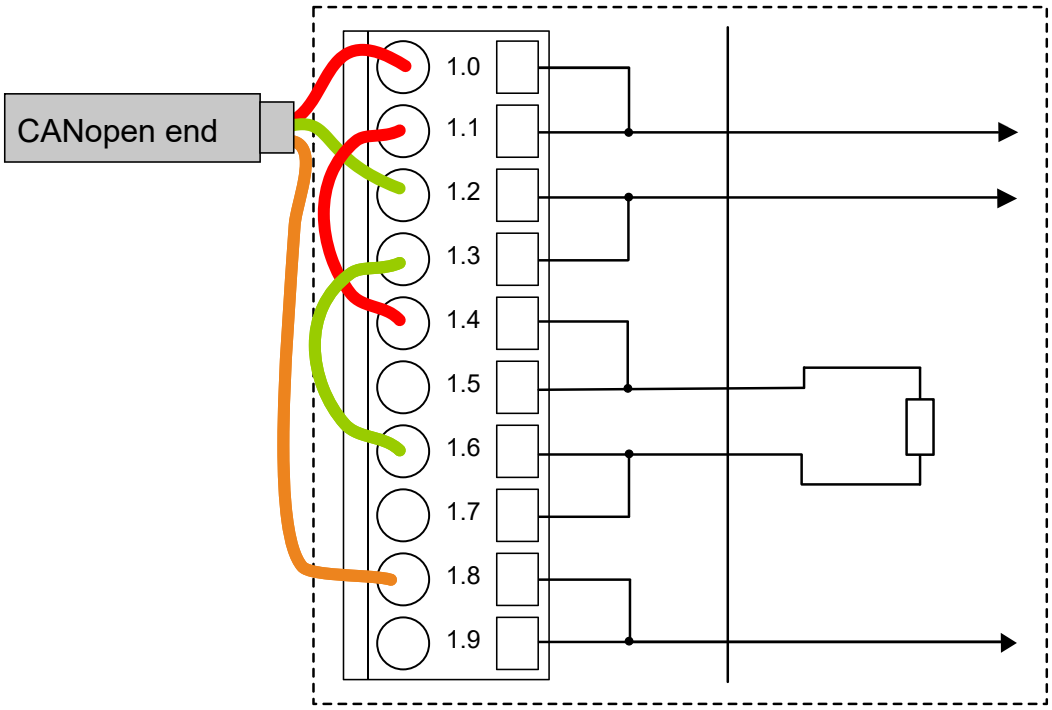
Terminal	Signal	Description
1.0	CAN+	Non-inverted signal of the CAN bus
1.1	CAN+	Non-inverted signal of the CAN bus
1.2	CAN-	Inverted signal of the CAN bus
1.3	CAN-	Inverted signal of the CAN bus
1.4	Term+	CAN bus termination for CAN+ (for bus termination, Term+ must be connected with CAN+)
1.5	Term+	CAN bus termination for CAN+ (connecting alternative for terminal 1.4)

Terminal	Signal	Description
1.6	Term-	CAN bus termination for CAN- (for bus termination, Term- must be connected with CAN-)
1.7	Term-	CAN bus termination for CAN- (connecting alternative for terminal 1.6)
1.8	CAN-GND	Ground potential of the CAN bus
1.9	CAN-GND	Ground potential of the CAN bus

At the line ends of a bus segment, terminating resistors must be connected. If TU517 or TU518 is used, the bus terminating resistors can be enabled by connecting the terminals Term+ and Term- to the data lines CAN+ and CAN- (no external terminating resistors are required, see figure below).

The following figures show the different connection options for the CANopen communication interface module:





In the case of TU517/TU518, the terminating resistors are not located inside the TU but inside the communication interface module CI581-CN. Hence, when removing the device from the TU, the bus terminating resistors are no longer connected to the bus. The bus itself will not be disconnected if a device is removed.

The grounding of the shield should take place at the switchgear cabinet. Please refer to the AC500 System-Data Chapter 1.6.3.6.1 “System data AC500” on page 5313.

Table 511: Assignment of the other terminals

Terminal	Signal	Description
2.0	DC0	Signal of the configurable digital input/output DC0
2.1	DC1	Signal of the configurable digital input/output DC1
2.2	DC2	Signal of the configurable digital input/output DC2
2.3	DC3	Signal of the configurable digital input/output DC3
2.4	DC4	Signal of the configurable digital input/output DC4
2.5	DC5	Signal of the configurable digital input/output DC5
2.6	DC6	Signal of the configurable digital input/output DC6
2.7	DC7	Signal of the configurable digital input/output DC7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI8	Signal of the digital input DI8
3.1	DI9	Signal of the digital input DI9
3.2	DI10	Signal of the digital input DI10
3.3	DI11	Signal of the digital input DI11

Terminal	Signal	Description
3.4	DI12	Signal of the digital input DI12
3.5	DI13	Signal of the digital input DI13
3.6	DI14	Signal of the digital input DI14
3.7	DI15	Signal of the digital input DI15
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO8	Signal of the digital output DO8
4.1	DO9	Signal of the digital output DO9
4.2	DO10	Signal of the digital output DO10
4.3	DO11	Signal of the digital output DO11
4.4	DO12	Signal of the digital output DO12
4.5	DO13	Signal of the digital output DO13
4.6	DO14	Signal of the digital output DO14
4.7	DO15	Signal of the digital output DO15
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

Connection of CANopen communication interface module CI582-CN:

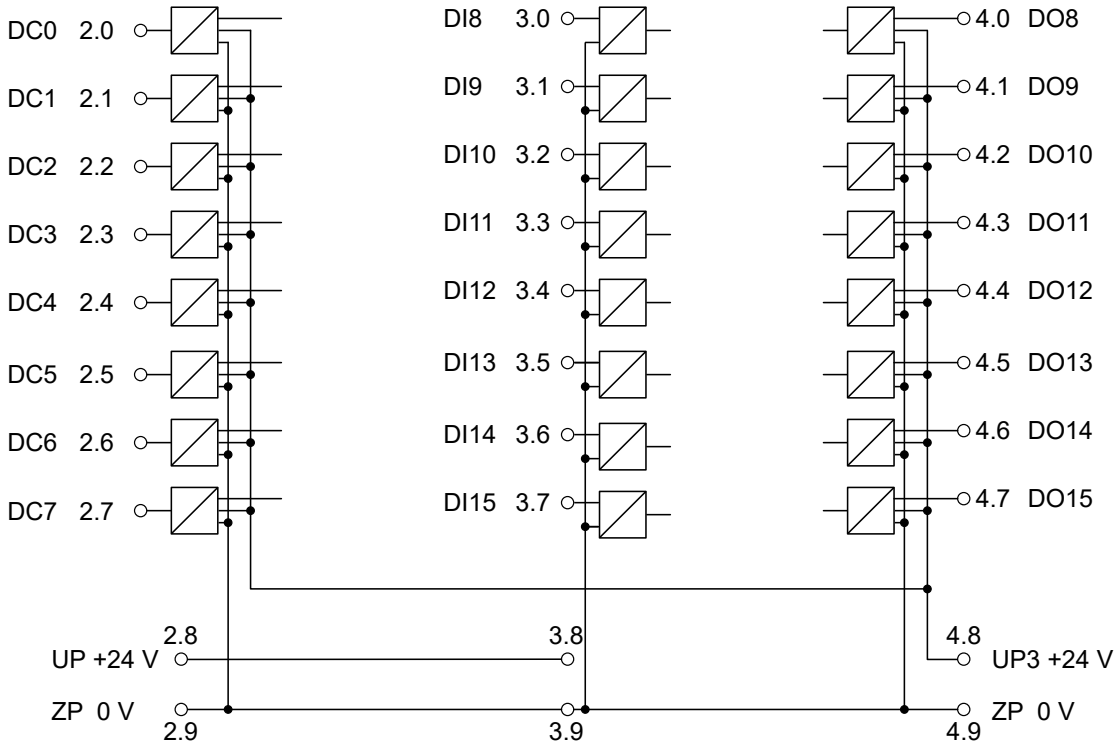


Fig. 946: Connection of the communication interface module CI582-CN

For a description of the meaning of the LEDs, please refer to the section for the state LEDs
↳ *Chapter 1.6.2.8.2.3.9 “State LEDs” on page 4740*.

Bus length

The maximum possible bus length of a CAN network depends on bit rate (transmission rate) and cable type. The sum of all bus segments must not exceed the maximum bus length

Bit Rate (speed)	Bus Length
1 Mbit/s	40 m
800 kbit/s	50 m
500 kbit/s	100 m
250 kbit/s	250 m
125 kbit/s	500 m
62.5 kbit/s	1000 m
20 kbit/s	2500 m
10 kbit/s	5000 m

Connection of the digital inputs

The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.

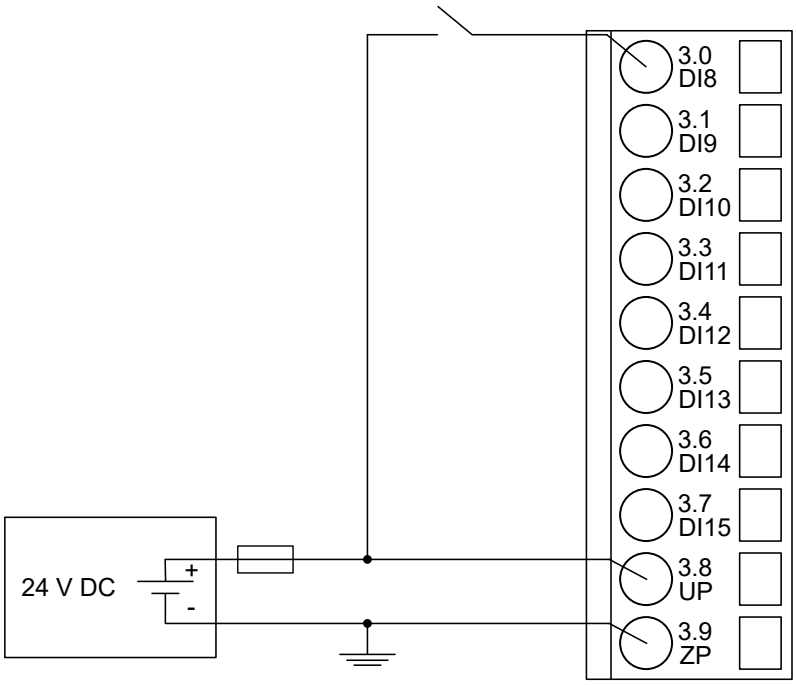


Fig. 947: Connection of the digital inputs to the module CI582-CN

Connection of the digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

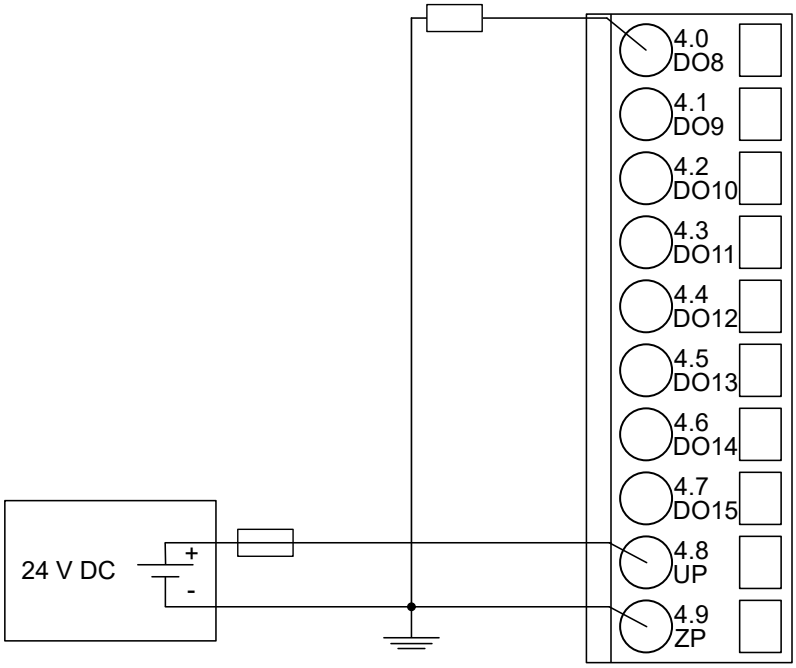


Fig. 948: Connection of configurable digital inputs/outputs to the module CI582-CN

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

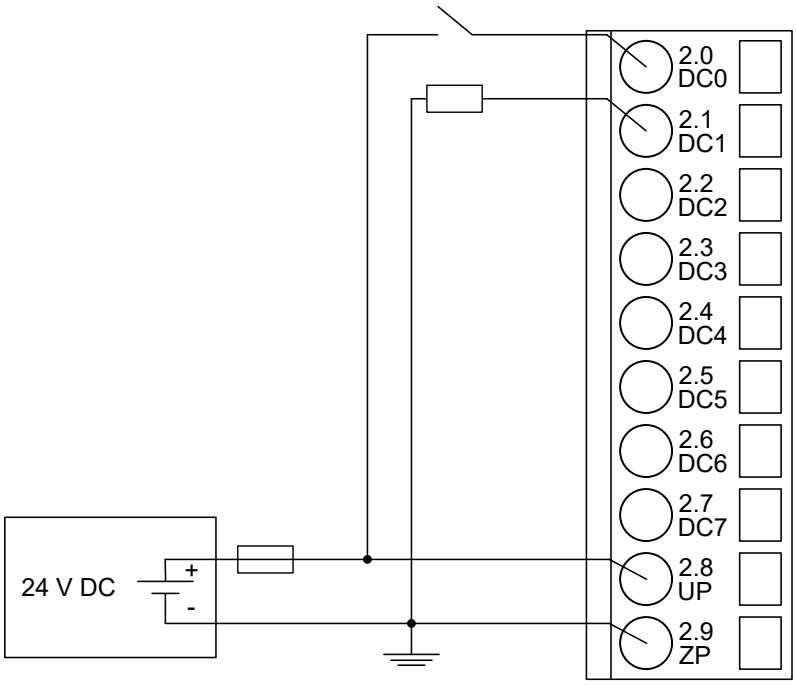


Fig. 949: Connection of configurable digital inputs/outputs to the module CI582-CN

Internal data exchange

Parameter	Value
Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

A detailed description concerning addressing can be found in the documentation of ABB Control Builder Plus Software.

The CANopen communication interface module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The range of permitted CANopen slave addresses is 1 to 127. Setting a higher address (> 128) does not lead to an error response, but results in a special mode (DS401). In this special mode, the device creates the node address by subtracting the value 128 from the address switch's value.

I/O configuration

The CI582-CN CANopen bus configuration is handled by CANopen master with the exception of the slave node ID (via rotary switches) and the transmision rate (automatic detection).

The digital I/O channels and the fast counter are configured via software.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C89	WORD	0x1C89
Parameter length	Internal	38	BYTE	38
Error LED / failsafe function table error LED / failsafe function ↳ *Table 512 “Error LED / Failsafe function” on page 4735*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved	0	0	ARRAY of 24 BYTES	
Check supply	On	0	BYTE	
	Off	1		1
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.

[2]) For a description of the counter operating modes, please refer to the 'Fast Counter' section ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

Table 512: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, failsafe mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, failsafe mode on *)
*) The parameter Behavior DO at comm. error is only analyzed if the failsafe mode is ON.	

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behavior DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behavior DO at comm. error is applied to DC and DO channels and only analyzed if the failsafe mode is ON.
[2])	The state "externally voltage detected" appears if the output of a channel DC0..DC7 is to be switched on while an external voltage is connected. In this case, start-up is disabled while the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears if external voltage at digital outputs DC0..DC7 and DO0..DO7 has exceeded the process supply voltage UP3 (see 'Connections' ↯ *Chapter 1.6.2.8.2.3.3 "Connections" on page 4726*). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters to "OFF". This parameter should only be disabled in exceptional cases as voltage overflow may produce reverse voltage.

Diagnosis

Structure of the diagnosis block via CANOM_NODE_DIAG. ↯ *Chapter 1.5.4.7.1.8 "CANOM_NODE_DIAG" on page 933*

Byte Number	Description	Possible Values
1	Diagnosis byte, slot number	31 = CI582-CN (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to Bit 5, coded error description
5	Diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The module performs reactivation automatically. Thus, an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500 display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization
4	-	31	31	31	45	Process voltage UP3 too low		Check process voltage
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])		Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module		Replace I/O module

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**CANopen diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check termi-nals/ check process supply voltage
Channel error digital							
4	-	31	2	8...15	46	Externally voltage detected at digital output DO0..DO7 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0..DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0..DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO0..DO7[7])	Check terminals

Remarks:

[1])	In AC500, the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = position of the communication module;14 = I/O bus; 31 = module itself The identifier is not contained in the CI542-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = module itself, 1..10 = expansion module

[3])	With "Module" the following allocation applies depending on the master: Module error: 31 = module itself Channel error: module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears if external voltages at one or more terminals DC0..DC7 or DO0..DO7 cause other digital outputs to be supplied by that voltage (voltage feedback, see 'Connections' ↬ *Chapter 1.6.2.8.2.3.3 “Connections” on page 4726*). All outputs of the digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0..DC7 and DO0..DO7 has exceeded the process supply voltage UP3 (see 'Connections' ↬ *Chapter 1.6.2.8.2.3.3 “Connections” on page 4726*). A diagnosis message appears for the whole module.
[6])	This message appears if the output of a channel DC0..DC7 or DO0..DO7 should be switched on while an external voltage is connected. In this case the start-up is disabled while the external voltage is connected. Otherwise, this could produce reverse voltage flowing from this output to other digital outputs. This diagnosis message appears for each channel.
[7])	Short circuit: After a short circuit has been detected, the output is deactivated for 100ms. Subsequently, a new start-up will be executed. This diagnosis message appears for each channel.

State LEDs

The LEDs are located at the front of the module. There are 2 different groups:

- The 5 system LEDs (PWR, CN-RUN, CN-ERR, S-ERR and I/O bus) show the operation states of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O controller	Start-up / preparing communication
	Yellow	---	---	---
CN-RUN	Green	---	Device configured, CANopen bus in OPERATIONAL state and cyclic data exchange running	Flashing: CANopen bus in PRE-OPERATIONAL state and slave is being configured Single flash: CANopen bus in STOPPED state. Flickering: Autodetect is active

LED	Color	OFF	ON	Flashing
CN-ERR	Red	No system error	CANopen Bus is OFF	Flashing: Configuration error Single flash: error counter overflow due to too many error frames Double flash: A node-guard or a heartbeat event occurred Flickering: Autodetect is active
S-ERR	Red	No error	Internal error	--
I/O bus	Green	No decentralized I/O modules connected or communication error	Decentralized I/O modules connected and operational	---

States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/output is OFF	Input/output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 3.0 to 3.7
Reference potential for all inputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC
Signal 0	-3 V...+5 V
Undefined signal	> +5 V...< +15 V
Signal 1	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V
Ripple with signal 1	Within +15 V...+30 V
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA
Input voltage +15 V	> 2 mA
Input voltage +30 V	< 8 mA
Max. cable length	
Shielded	1000 m
Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DO0 to DO7	Terminals 4.0 to 4.7
Reference potential for all outputs	Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	
Rated value per channel	500 mA at UP3 = 24 V
Max. value (all channels together)	4 A
Leakage current with signal 0	< 0.5 mA

Parameter	Value
Fuse for UP3	10 A fast
Demagnetization with inductive DC load	Via internal varistors (see figure below this table)
Output switching frequency	
With resistive load	On request
With inductive loads	Max. 0.5 Hz
With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof	Yes
Overload message (I > 0.7 A)	Yes, after ca. 100 ms
Output current limitation	Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals	Yes (software-controlled supervision)
Max. cable length	
Shielded	1000 m
Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

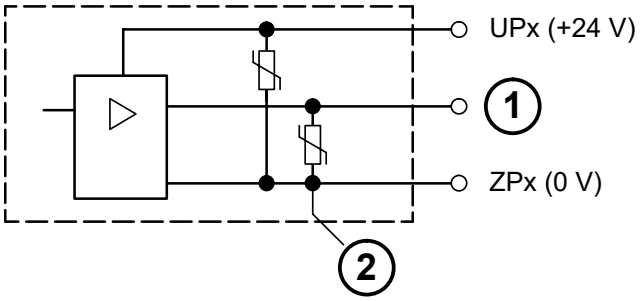


Fig. 950: Digital input/output (circuit diagram)

1	Digital output
2	Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter	Value
Number of channels per module	8 inputs/outputs (with transistors)
Distribution of the channels into groups	1 group for 8 channels
If the channels are used as inputs	
Channels DC0...DC07	Terminals 2.0...2.7
If the channels are used as outputs	
Channels DC0...DC07	Terminals 2.0...2.7

Parameter	Value
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation	From the CANopen network

Technical data of the digital inputs/outputs if used as inputs

Please refer to the Technical Data of the Digital Inputs *Chapter 1.6.2.8.2.3.10 “Technical data” on page 4741*. Deviation:

Terminals of the channels DC0 to DC7: Terminals 2.0 to 2.7

Due to the direct connection to the output, the demagnetizing varistor is also effective at the input. This is why the difference between UPx and the input signal must not exceed the clamp voltage of the varistor. The varistor limits the clamp voltage to approx. 36 V. Consequently, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Please refer to the Technical Data of the Digital Outputs *Chapter 1.6.2.8.2.3.10 “Technical data” on page 4741*. Deviation:

Terminals of the channels DC0 to DC7: Terminals 2.0 to 2.7

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

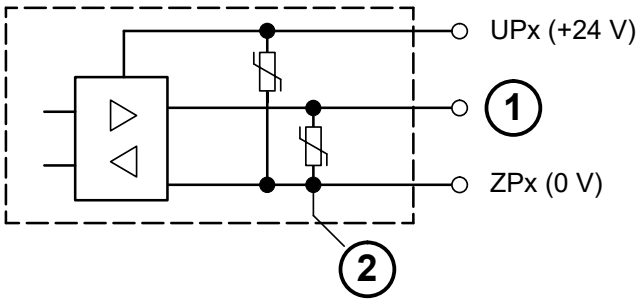


Fig. 951: Digital input/output (circuit diagram)

1	Digital input/output
2	For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI8), 3.1 (DI9)
Used outputs	Terminal 4.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

Parameter	Value
Detailed description	Fast Counter ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*
Operating modes	Operating modes ↳ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

Ordering data

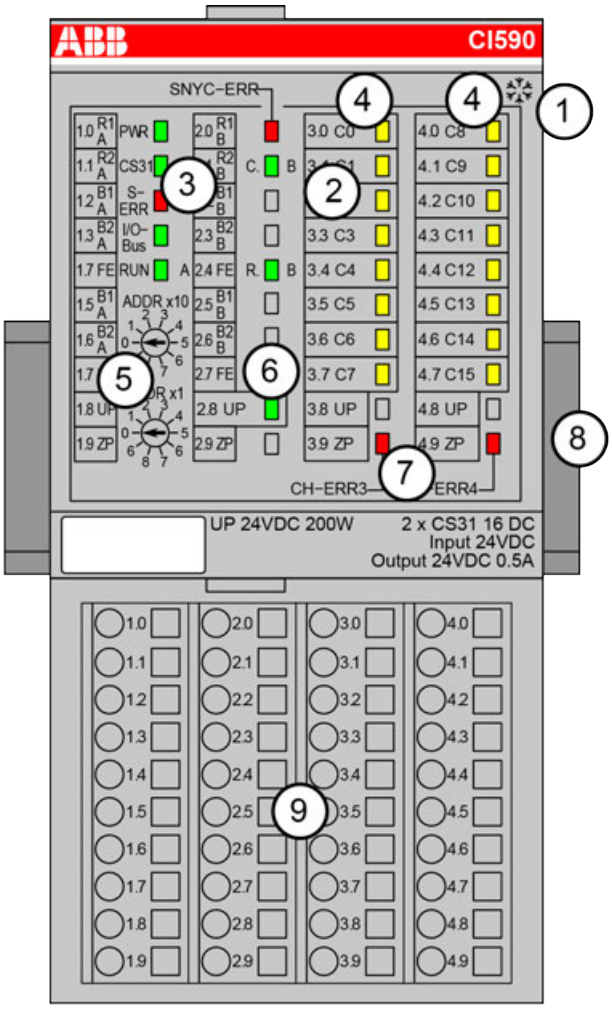
Part no.	Description	Product life cycle phase *)
1SAP 228 200 R0001	CI582-CN, CANopen communication interface module with 8 DI, 8 DO and 8 DC	Active
1SAP 428 200 R0001	CI582-CN-XC, CANopen communication interface module with 8 DI, 8 DO and 8 DC, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8.3 CS31

CI590-CS31-HA

- 16 configurable digital inputs/outputs 24 V DC
- CS31 bus connection
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 5 system LEDs
4 16 yellow LEDs to display the signal states of the configurable digital inputs/outputs C0 to C15
5 2 rotary switches to set the module's address (00d to 99d)
6 1 green LED to display the process voltage UP
7 2 red LEDs to display errors
8 DIN rail
9 Terminal unit
❄ Sign for XC version

Intended purpose

The High Availability CS31 bus module CI590-CS31-HA is used as a decentralized I/O module on CS31 field buses. The CI590-CS31-HA contains two RS-485 interfaces for connecting the module to two separate CS31 buses to have redundancy/backup or high availability. In addition, the CI590-CS31-HA provides 16 I/O channels with 16 configurable digital inputs/outputs (C0...C15) in one group. This group can be used as follows:

- 24 V DC input
- 24 V DC transistor output, 0.5 A (max.), short-circuit and overload protected
- re-readable output (combined input/output) with identical technical data of the digital inputs and outputs

The inputs and outputs are group-wise galvanically isolated from the CS31 buses and from other modules. Each CS31 bus is galvanically isolated from other terminals.

Functionality

Parameter	Value
Interface bus A	RS-485, CS31 protocol, galvanically isolated from other electronic.
Interface bus B	RS-485, CS31 protocol, galvanically isolated from other electronic.
Address switches	Two rotary switches for setting the CS31 bus address (00d to 99d).
I/O bus	I/O bus to connect S500 I/O modules (max. 7).
Digital inputs/outputs	16 configurable digital inputs/outputs in one group: 24 V DC, 0.5 A (max.), short-circuit and overload protected.
High-speed counter	Integrated, with many configurable operating modes.
LED displays	For system states, signal states, errors and power supply.
External power supply	Via UP and ZP terminal (process voltage: 24 V DC).
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU552-CS31 *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*

Connections

The CS31-HA communication interface module CI590-CS31-HA is plugged on CS31 terminal unit TU551-CS31 or TU552-CS31. Hereby, it clicks in with two mechanical locks. The terminal unit is mounted on a DIN rail or with two screws plus the additional accessory for wall mounting (TA526).

Mounting, disassembling and connection for the terminal units and the I/O modules are described in detail in the S500 system data chapters.

The connection is carried out by using the 40 terminals of the terminal unit TU551-CS31/TU552-CS31. It is possible to replace the CI590-CS31-HA without loosening the wiring.

Assignment of the terminals:

Terminal	Signal	Description
1.0	R1A	Integrated terminating resistors for CS31 bus A, terminal 1
1.1	R2A	Integrated terminating resistors for CS31 bus A, terminal 2
1.2	B1A	CS31 bus A, bus line 1
1.3	B2A	CS31 bus A, bus line 2
1.4	FE	Functional earth
1.5	B1A	CS31 bus A, bus line 1
1.6	B2A	CS31 bus A, bus line 2

Terminal	Signal	Description
1.7	FE	Functional earth
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	R1B	Integrated terminating resistors for CS31 bus B, terminal 1
2.1	R2B	Integrated terminating resistors for CS31 bus B, terminal 2
2.2	B1B	CS31 bus B, bus line 1
2.3	B2B	CS31 bus B, bus line 2
2.4	FE	Functional earth
2.5	B1B	CS31 bus B, bus line 1
2.6	B2B	CS31 bus B, bus line 2
2.7	FE	Functional earth
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	C0	Signal of the configurable digital input/output C0
3.1	C1	Signal of the configurable digital input/output C1
3.2	C2	Signal of the configurable digital input/output C2
3.3	C3	Signal of the configurable digital input/output C3
3.4	C4	Signal of the configurable digital input/output C4
3.5	C5	Signal of the configurable digital input/output C5
3.6	C6	Signal of the configurable digital input/output C6
3.7	C7	Signal of the configurable digital input/output C7
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	C8	Signal of the configurable digital input/output C8
4.1	C9	Signal of the configurable digital input/output C9
4.2	C10	Signal of the configurable digital input/output C10
4.3	C11	Signal of the configurable digital input/output C11
4.4	C12	Signal of the configurable digital input/output C12
4.5	C13	Signal of the configurable digital input/output C13
4.6	C14	Signal of the configurable digital input/output C14
4.7	C15	Signal of the configurable digital input/output C15
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

CAUTION!

Risk of damaging the PLC modules!

The PLC modules must not be removed if the plant is powered on. Make sure that all voltage sources (supply and process voltage) are switched off before removing or replacing a module.

CAUTION!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overvoltages and short circuits. Make sure that all voltage sources (supply and process voltage) are switched off before starting system operation.

The module provides several diagnostic functions (see chapter *Chapter 1.6.2.8.3.1.10 “Diagnosis” on page 4756*).

The following figure demonstrates connection of the configurable digital inputs/outputs. The digital input/output C0 is connected as an output and the digital input/output C1 is connected as an input. Connect the digital inputs/outputs C2...C15 in the same way.

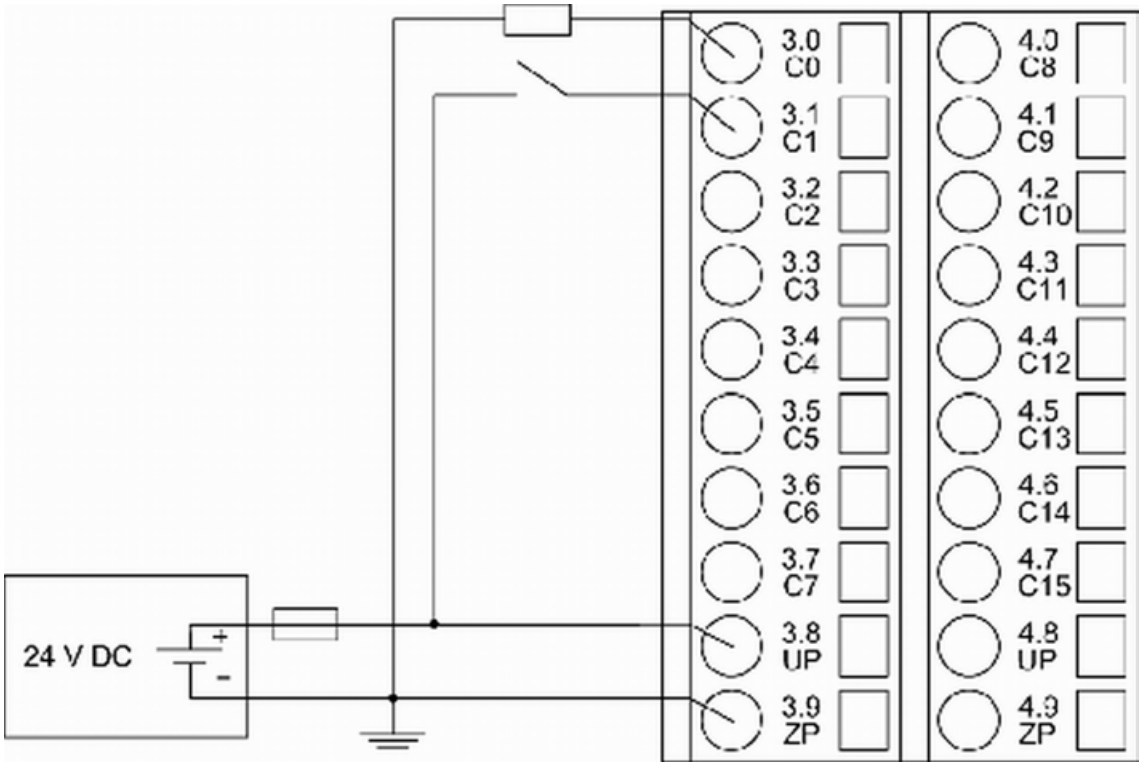


Fig. 952: CI590-02

CAUTION!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of CI590-CS31-HA. Connect a 470 Ω / 1 W resistor in series configurable inputs/outputs C8/C9 if using them as fast counter inputs to safely avoid any influences.

The meaning of the LEDs is described in the chapter *Chapter 1.6.2.8.3.1.11 “State LEDs” on page 4758*.

CS31 bus connections

CS31 bus is connected with terminals 1.0 to 1.7 and 2.0 to 2.7 through the terminal unit. The end-of-line resistor can also be activated by using external wire jumpers.

The following figure describe the different possibilities of connecting CS31 buses to the CI590-CS31-HA:

Option 1

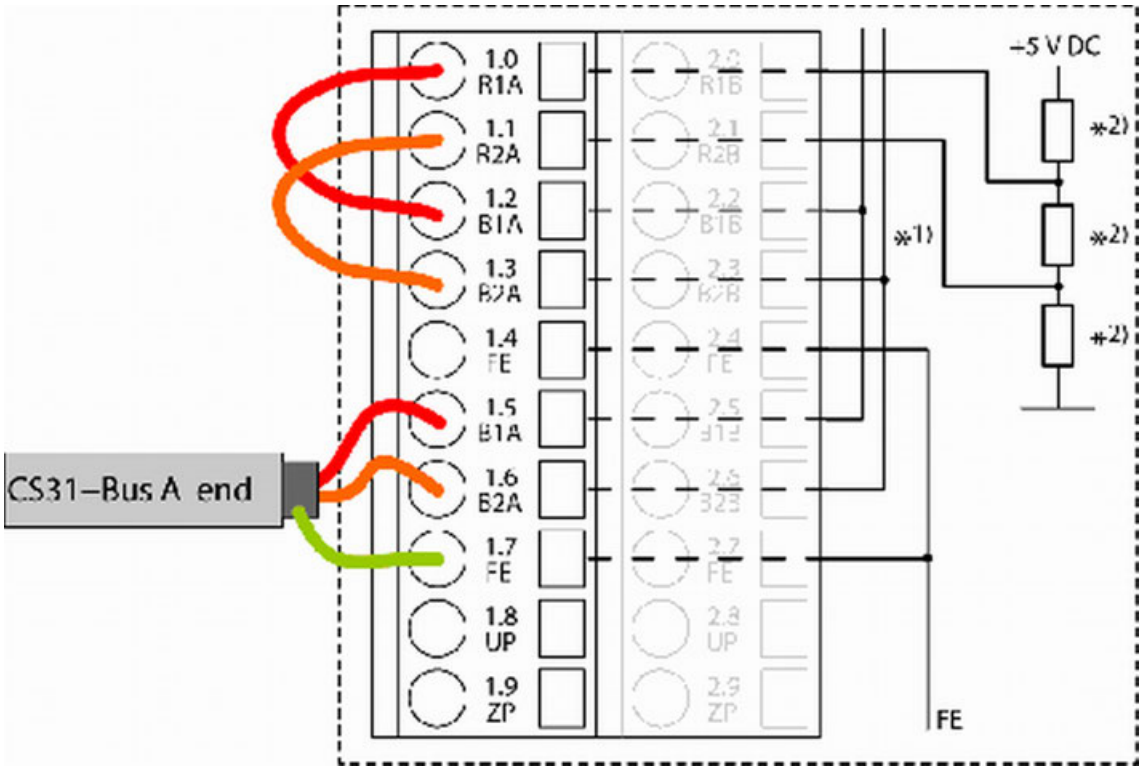


Fig. 953: Connection of CS31 bus A with CI590-CS31-HA located at the bus end

[1]) Connection between the bus lines is located inside the terminal unit.
[2]) Terminating resistors are located in the terminal unit TU551-CS31/TU552-CS31.

Option 2

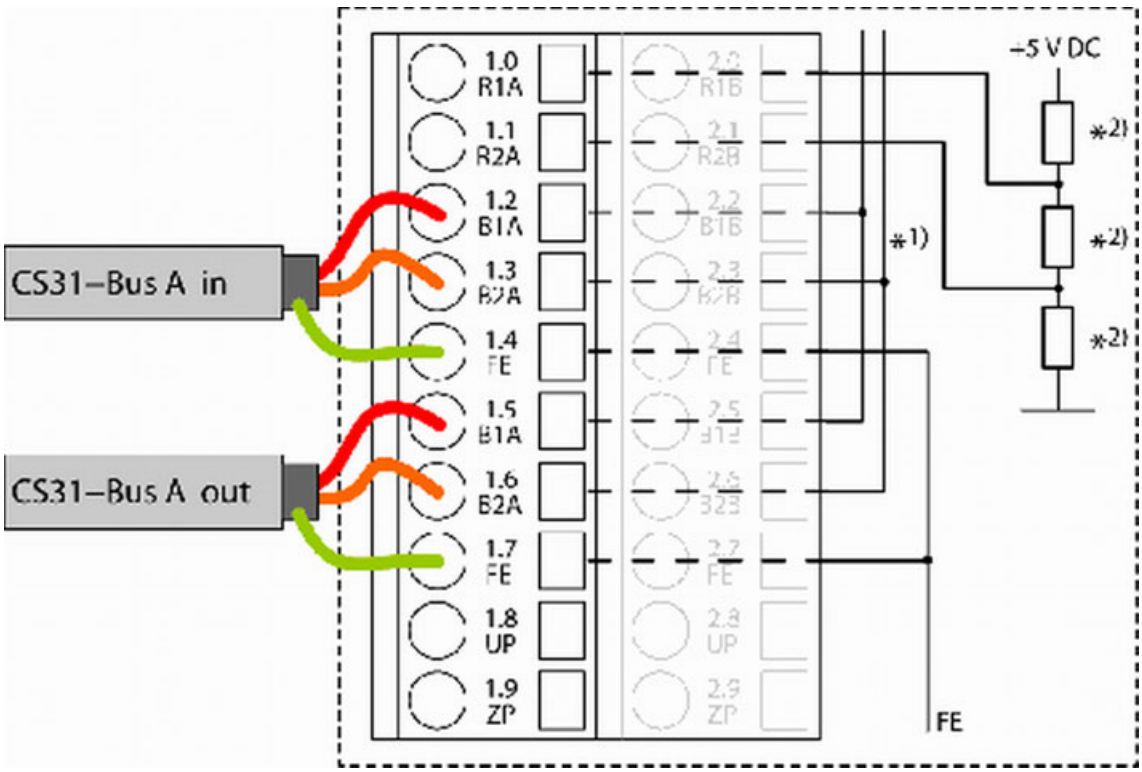


Fig. 954: Connection of CS31 bus A with CI590-CS31-HA located in the middle of the bus

[1]) Connection between the bus lines is located inside the terminal unit.
[2]) Terminating resistors are located in the terminal unit TU551-CS31/TU552-CS31.

Option 3

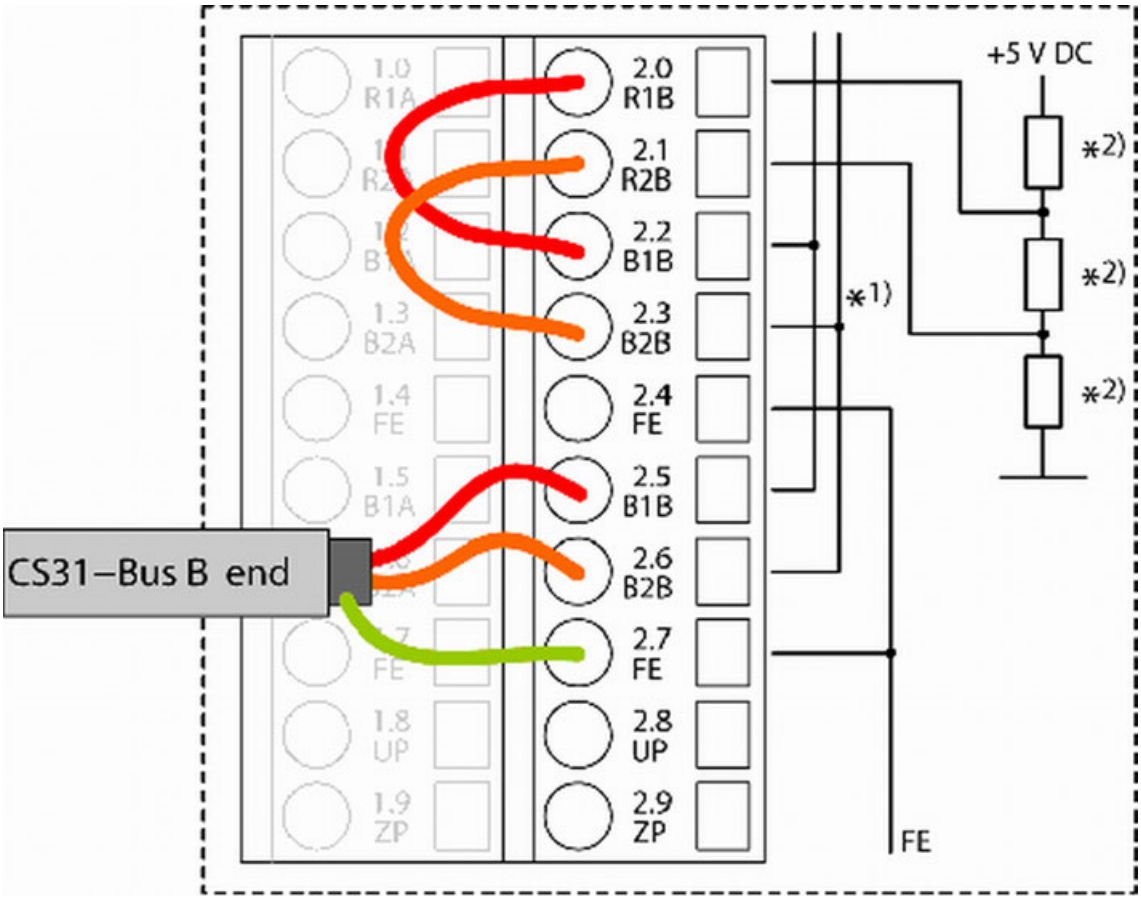


Fig. 955: Connection of CS31 bus B with CI590-CS31-HA located at the bus end

[1]) Connection between the bus lines is located inside the CI590-CS31-HA module.
[2]) Terminating resistors are located in the CI590-CS31-HA module.

Option 4

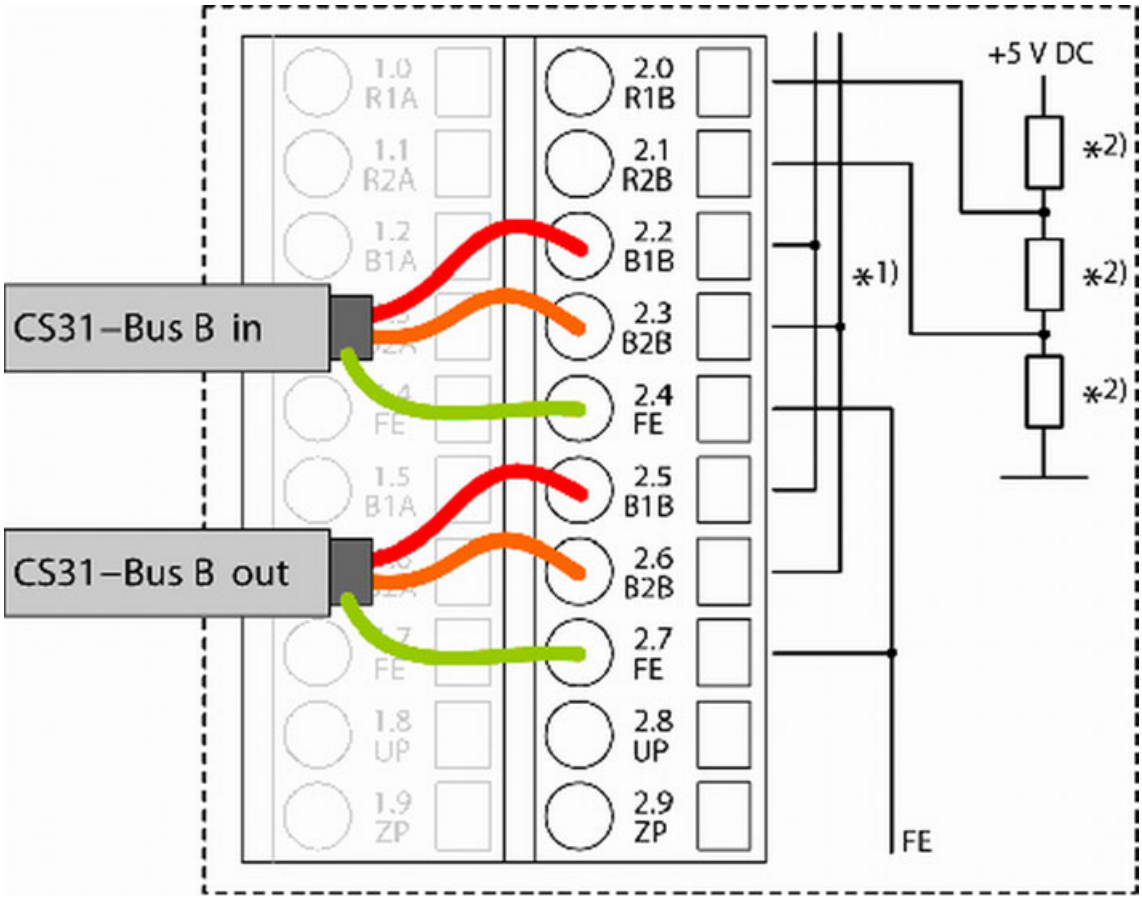


Fig. 956: Connection of CS31 bus B with CI590-CS31-HA located in the middle of the bus

[1]) Connection between the bus lines is located inside the CI590-CS31-HA module.
[2]) Terminating resistors are located in the CI590-CS31-HA module.

Details on CS31 wiring is described seperately *Chapter 1.6.3.6.4.8 “CS31 bus” on page 5347*.

Internal data exchange

Parameter	Without fast counter	With fast counter (only with AC500)
Digital inputs (bytes)	2 + expansion modules	5 + expansion modules
Digital outputs (bytes)	2 + expansion modules	5 + expansion modules
Counter input data (words)	0	4 (+4 AI)
Counter output data (words)	0	8 (+8 AO)

Addressing

An address must be set at every module so that the field bus communication module can access the specific inputs and outputs.

Only one address is used to identify the module on bus A and bus B.

CI590-CS31-HA address must be set based on the “number of CS31 modules“ calculated by Automation Builder.

The address (00d to 99d) is set with two rotary switches on the front panel of the module.

CS31 bus module reads the position of the address switches only during initialization after power on, i.e. changes of the settings during operation remain ineffective.

CI590-CS31-HA limitations

The following peculiarities concerning the CS31 bus in the AC500 must be observed when addressing S500 I/O devices at the CS31 bus:

- One CS31 software module can occupy a maximum of 15 bytes of inputs and 15 bytes of outputs in the digital area. This corresponds to 15 x 8 = 120 digital inputs and 120 outputs.
- One CS31 software module can allocate a maximum of eight words of inputs and eight words of outputs in the analog area.
- A maximum of 31 of these CS31 software modules are allowed for connection to the CS31 bus.
- If a device contains more than 15 bytes or eight words of inputs or outputs, it occupies two or more of the 31 CS31 software modules.
- The CI590-CS31 can internally manage two CS31 software modules in the digital area and five CS31 software modules in the analog area. This corresponds to a maximum of:
  – 240 digital inputs (2 x 15 bytes) and
  – 240 digital outputs (2 x 15 bytes) and
  – 40 analog inputs (5 x 8 words) and
  – 40 analog outputs (5 x 8 words).
- Address setting is done at the CI590-CS31 using two rotary switches at the module's front plate.
- To enable the fast counter of the CI590-CS31 the hardware address (HW_ADR) has to be set to the module address + 70. With activated fast counter, the module addresses 0...28 (hardware address setting 70...98) are allowed.
  Then, the CI590-CS31 registers contain two CS31 software modules using the module address (hardware address 70), once in the digital area and once in the analog area.
- CS31 software module 1 in digital area:
  -> registers using the module address.
  CS31 software module 2 in digital area:
  -> registers using module address+7 and bit "Channel ≥ 7" set.
  CS31 software module 1 in analog area:
  -> registers using the module address.
  CS31 software module 2 in analog area:
  -> registers using module address and bit "Channel ≥ 7" set.
  CS31 software module 3 in analog area:
  -> registers using the module address+1.
  CS31 software module 4 in analog area:
  -> registers using module address+1 and bit "Channel ≥ 7" set.

- The CI590-CS31 can manage a maximum of 255 parameters. This does not cause any restrictions in all configurations with the currently available S500 I/O devices.
- The next free address for a CI590-CS31 is derived from the highest address occupied in the digital area or the analog area of the previous CI590-CS31.
- When connecting several S500 expansion modules to a CI590-CS31 via the I/O Bus, their inputs and outputs follow the CI590-CS31s inputs and outputs without gap. Such a cluster can occupy up to six CS31 software modules.
- A maximum of seven S500 expansion modules (extensions) can be connected to a CI590-CS31.

I/O configuration

The CI590-CS31-HA does not store configuration data itself. The 16 configurable digital inputs/outputs are defined as inputs or outputs by the user program, i.e. each of the configurable channels can be used as input or output (or re-readable output) by interrogation or allocation with the user program.

Parametrization

Arrangement of parameter data is performed by your master configuration software Automation Builder.

CAUTION!

Risk of configuration errors!

Contradictory parameter settings may cause configuration errors of the CI590-CS31-HA and attached I/O modules. Please make sure, the fast counter mode is not set to value 0 if the module is included with fast counter in PLC configuration.

The parameter data directly influences module functionality.

For non-standard applications, it is necessary to adapt the parameters to your system configuration.

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Module address	1	2740 [1])	BYTE	2740 0 x 0AB4	0	61
Ignore module	No Yes	0 1	BYTE	No (0 x 00)	-	-
Parameter length	Internal	8 7 [2])	BYTE	8 7 [2])	0	255
Check supply	Off On	0 1	BYTE	On 0 x 01	-	-

Name	Value	Internal Value	Internal Value, Type	Default	Min.	Max.
Error LED / Failsafe Function	On Off by E4 Off by E3 On + Fail-safe Off by E4 + Failsafe Off by E3 + Failsafe	-	-	On	-	-
Stop behavior	Switch over Stop Both stop/ failsafe	0 1 2	BYTE	0	-	-
Output compare	No check Binary Analog ± 256 Analog ± 512 Binary + Analog 256 Binary + Analog 512	0 1 2 3 4 5	BYTE	0	-	-
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	8 ms 0 x 02	-	-
Fast counter	0 : 10 [3])	0 : 10	BYTE	Mode 0 0 x 00	-	-
Detection short-circuit at outputs	Off On	0 1	BYTE	On 0 x 01	-	-
Behavior outputs at communi-cation fault	Off Last value Substitute value	0 1 2	BYTE	Off 0 x 00	-	-
Substitute value	0...65535	0...0xffff	WORD	0	-	-

[1]) with CS31 and addresses less than 70 and FBP, the value is increased by 1.

[2]) with CS31 and addresses less than 70, without the parameter "Fast Counter".

[3]) Counter operating modes, see description of the fast counter.

Diagnosis

Structure of CI590-CS31-HA diagnosis block

If a CI590-CS31-HA module is connected via a CS31 bus, then the field bus master receives diagnosis information by an extended diagnosis block. The following table specifies the structure of this information. In case of an error the user can get this information by the diagnosis system, see *Chapter 1.6.2.8.3.1.10.2 “Diagnosis table CI590-CS31-HA” on page 4757*.

Byte Number	Description	Possible values
1	Data length (header included)	18
2	Diagnosis byte	0 = Communication with CI590-CS31-HA OK 1 = Communication with CI590-CS31-HA failed
3	CI590-CS31-HA diagnosis byte, module number	0 = CI590-CS31-HA (e.g. error at the integrated 16 DC) 1 = 1st attached S500 I/O module 2 = 2nd attached S500 I/O module ... 7 = 7th attached S500 I/O module
4	CI590-CS31-HA diagnosis byte, slot	According to the I/O bus specification passed on by modules to the fieldbus master
5	CI590-CS31-HA diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
6	CI590-CS31-HA diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description passed on by modules to the fieldbus master
7	CI590-CS31-HA diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error Bit 5: 1 = diag reset Bit 2 to bit 4: reserved Bit 1: 1 = explicit acknowledgement Bit 0: 1 = static error passed on by modules to the fieldbus master Value = 0: static message for other systems, which do not have a coming/leaving evaluation
8ff	reserved	

Diagnosis table CI590-CS31-HA

In case of overload or short circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**FBP diagnosis block**	
Class	**Inter-face**	**Devic e**	**Module**	**Chann el**	**Error identi-fier**	**Error message**	**Remedy**
	1)	**2)**	**3)**	**4)**			
Module Error							
3	11	ADR	31	31	3	Timeout in the I/O module	Replace I/O module
3	11	ADR	31	31	19	Checksum error in the I/O module	
3	11	ADR	31	31	36	Internal data exchange failure	
3	11	ADR	31	31	40	Different hard-/firm-ware versions in the module	
3	11	ADR	31	31	43	Internal error in the module	
3	11	ADR	31	31	9	Overflow diagnosis buffer	Restart
3	11	ADR	31	31	26	Parameter error	Check master
3	11	ADR	31	31	11	Process voltage too low	Check process voltage
3	11	ADR	1...7	31	17	No communication to the I/O module	Replace I/O module
3	11	ADR	31	31 31	28	Configurations from PLC A of PLC B are different	Check PLC CS31 module configuration
3	11	ADR ADR	31	31	36	Wait Com (Only 1 bus or 1 CPU is active/operational)	Check second CPU or other bus connection
4	11	ADR	31	31	45	Process voltage ON/OFF	Process voltage ON
4	11	ADR	31/ 1...7	31	34	Wait ready (No reply during initiali-zation of the I/O module)	Replace I/O module
4	11	ADR	31/ 1...7	31	32	Wrong I/O module in the slot	Replace I/O module or check configuration

E1...E4	d1	d2	d3	d4	Identifier 000...063	AC500 display	<– Display in
Class	Comp	Dev	Mod	Ch	Err	PS501 PLC browser	
Byte 6 Bit 6...7	-	Byte 3	Byte 4	Byte 5	Byte 6 Bit 0...5	FBP diagnosis block	
Class	Interface	Device	Module	Channel	Error identifier	Error message	Remedy
	1)	2)	3)	4)			
4	11	ADR	31	31	54	CPU conflict ● Both CPUs are in STOP mode ● HA cycle time too small ● Mismatch in comparison of analog values	● Check CPU status ● Check HA cycle ↳ *Chapter 1.5.5.1.2.3.4 "Task configuration" on page 1999* ● Check wiring between the analog modules and the CPU
Channel Error CI590-CS31-HA							
4	11	ADR	31/ 1...7	8...23	47	Short circuit at a digital output	Check connection

Remarks:

1)	In AC500 the following interface identifier applies: 11 = COM1 (protocol CS31 bus only possible with COM1)
2)	With "Device" and CS31 bus master, the hardware address of the CI590-CS31-HA (0...69) is output.
3)	With "Module" the following allocation applies: 31 = module itself, 1...7 = Expansion 1...7
4)	In case of module errors, with channel "31 = Module itself" is output.

State LEDs

Table 513: States of the LEDs:

LED	Status	Color	LED = OFF	LED = ON	LED Flashes
PWR	System voltage	Green	System firmware is not running	System firmware is running	--
CS31 A	CS31 communication	Green	No communication at CS31 bus A	Communication at CS31 bus A OK	10 Hz: Not bit lifetime management

LED	Status	Color	LED = OFF	LED = ON	LED Flashes
C. B	CS31 communication	Green	No communication at CS31 bus B	Communication at CS31 bus B OK	10 Hz: Not bit lifetime management
S-ERR	Sum Error	Red	--	Internal error detected	2 Hz: Diagnostic event happened
I/O-Bus	Communication via the I/O bus	Green	No I/O bus communication	Expansion modules connected	2 Hz: Error I/O bus
RUN A	CPU active	Green	CPU A is not primary	CPU A is primary	RUN B LED off: CI590-CS31-HA primary self selection. No primary order from both PLC. PLC A has been selected as primary. RUN B LED on: 2 primary orders. PLC B is primary.
R. B	CPU active	Green	CPU B is not primary	CPU B is primary	RUN A LED off: CI590-CS31-HA primary self selection. No primary order from both PLC. PLC B has been selected as primary. RUN A LED on: 2 primary orders. PLC A is primary.
SYNC-ERR	Outputs from CPU A and CPU B	Red	--	Configuration conflict detected	10 Hz: Not parameterized 2 Hz: Switch-over has occured
C0...C15	Digital inputs/ outputs	Yellow	Input/output = OFF	Input/output = ON (the input voltage is even displayed if the supply voltage is OFF)	--

LED	Status	Color	LED = OFF	LED = ON	LED Flashes
UP	Process supply voltage and initialization	Green	Process voltage is missing	Process voltage OK and initialization completed	Module was not initialized correctly
CH-ERR3		Red	No error	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short-circuit at an output)
CH-ERR4		Red	No error	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short-circuit at an output)
CH-ERR *)	Module error	Red	No error or process voltage is missing	Internal error	--
*) All LEDs CH-ERR2 to CH-ERR4 light up together					

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Rated supply voltage of the module		24 V DC (UP/ZP)
Current consumption of the module (UP)		50 mA
Process voltage UP:		
	Rated value	24 V DC (for inputs and outputs)
	Max. electric charge for the supply terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse at UP	10 A fast
	Galvanic isolation	CS31 bus A interface from the rest of the module CS31 bus B interface from the rest of the module
	Inrush current from UP (at power-up)	0.040 A^2s

Parameter		Value
	Current consumption from UP at normal operation / with outputs	0.1 A + max. 0.008 A per input + max. 0.5 A per output
	Connections	Terminals 1.8 - 4.8 for +24 V (UP) and 1.9 - 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W (outputs unloaded)
Number of configurable digital inputs/outputs		16
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Address setting		
Diagnosis, refer to *Chapter 1.6.2.8.3.1.10 "Diagnosis" on page 4756*		With two rotary switches on the front panel
Operating and error displays		27 LEDs altogether
Weight (without terminal unit)		Approx. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Configurable digital inputs/outputs

Each of the configurable digital inputs/outputs is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter	Value
Number of channels per module	16 inputs/outputs (with transistors)
Distribution of the channels into groups	1 group of 16 channels
Connection of the channels C0 to C7	Terminals 3.0 to 3.7
Connection of the channels C8 to C15	Terminals 4.0 to 4.7
Indication of the input/output signals	1 yellow LED per channel, the LED is ON if the input/output signal is high (signal 1)
Galvanic isolation	Yes, between the I/O channels and the rest of the module

Digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		16 digital inputs
Reference potential for all inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Input current per channel:		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
Signal 0		-3 V...+5 V *)
Undefined signal		> +5 V...<+15 V
Signal 1		+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Max. cable length:		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V if UPx = 24 V and from -6 V to +30 V if UPx = 30 V.

Digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 16 transistor outputs
Reference potential for all outputs		Terminals 1.9, 2.9, 3.9 and 4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current:		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	10 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast

Parameter		Value
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency:		
	With resistive loads	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after approx. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length:		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization if inductive loads are switched off.

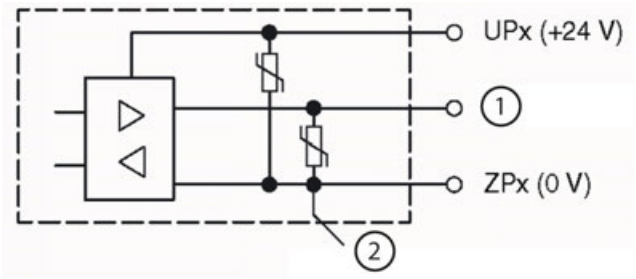


Fig. 957: Digital input/output (circuit diagram)

Technical data of the fast counter

Parameter	Value
Used inputs	C8 / C9
Used outputs	C10
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

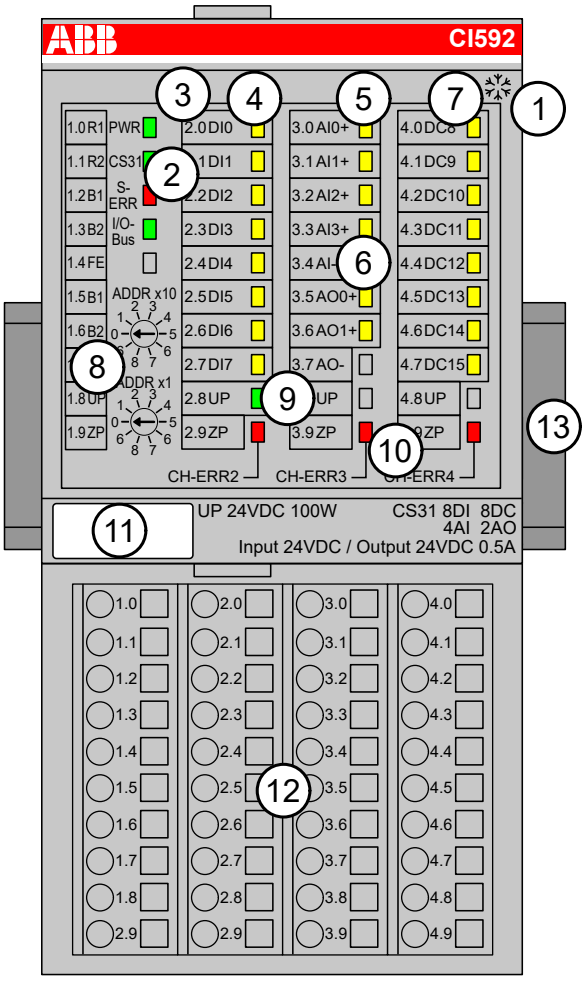
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 221 100 R0001	CI590-CS31-HA, CS31 redundant communication interface module, 16 DC	Active
1SAP 421 100 R0001	CI590-CS31-HA-XC, CS31 redundant communication interface module, 16 DC, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CI592-CS31 - Digital and analog inputs and outputs

- 8 digital inputs 24 V DC
- 8 configurable digital inputs/outputs 24 V DC
- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- CS31 bus connection
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 4 system LEDs
3 Allocation between terminal number and signal name
4 8 yellow LEDs to display the signal states of the digital inputs DI0 to DI7
5 4 yellow LEDs to display the signal states of the analog inputs AI0 to AI3
6 2 yellow LEDs to display the signal states of the analog outputs AO0 to AO1
7 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs DC8 to DC15
8 2 rotary switches to set the module's address (00d to 99d)

9 1 green LED to display the process voltage UP
10 3 red LEDs to display errors
11 Label
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The CS31 Bus Module is used as a decentralized I/O module on CS31 field buses. The bus connection is performed on a RS-485 serial interface, which allows the connection of this module to all existing CS31 buses. In addition, the CS31 Bus Module provides 22 I/O channels with the following properties:

- 8 digital inputs, 24 V DC
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- 4 analog inputs, voltage, current and RTD, resolution 12 bits plus sign
- 2 analog outputs, voltage and current, resolution 12 bits plus sign

The configuration is performed by software.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Interface	RS-485, CS31 protocol
Address switches	For setting the module's address (00d to 99d)
Digital inputs	8 (24 V DC; delay time configurable via software)
Configurable digital inputs/outputs	8 (24 V DC, 0.5 A max.)
Analog inputs	4 (configurable via software), resolution 12 bits plus sign, voltage, current and RTD input
Analog outputs	2 (configurable via software), resolution 12 bits plus sign, voltage and current output
Fast counter	Integrated, many configurable operating modes
LED displays	For system displays, signal statuses, errors and power supply
External supply voltage	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU551-CS31 or TU552-CS31 *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*

Connections

The CS31 communication interface module CI592-CS31 is plugged on the CS31 terminal unit TU551-CS31 or TU552-CS31 ↳ *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*. Hereby, it clicks in with two mechanical locks. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection is carried out by using the 40 terminals of the terminal unit TU551-CS31/TU552-CS31. It is possible to replace the CI592-CS31 without loosening the wiring.

The assignment of the terminals:

Terminal	Signal	Description
1.0	R1	Integrated terminating resistors for CS31-Bus, Terminal 1
1.1	R2	Integrated terminating resistors for CS31-Bus, Terminal 2
1.2	B1	CS31-Bus, bus line 1
1.3	B2	CS31-Bus, bus line 2
1.4	FE	Functional earth
1.5	B1	CS31-Bus, bus line 1
1.6	B2	CS31-Bus, bus line 2
1.7	FE	Functional earth
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI0	Signal of the digital input DI0
2.1	DI1	Signal of the digital input DI1
2.2	DI2	Signal of the digital input DI2
2.3	DI3	Signal of the digital input DI3
2.4	DI4	Signal of the digital input DI4
2.5	DI5	Signal of the digital input DI5
2.6	DI6	Signal of the digital input DI6
2.7	DI7	Signal of the digital input DI7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	AI0+	Positive pole of analog input signal 0
3.1	AI1+	Positive pole of analog input signal 1
3.2	AI2+	Positive pole of analog input signal 2
3.3	AI3+	Positive pole of analog input signal 3
3.4	AI-	Negative pole of analog input signals 0 to 3
3.5	AO0+	Positive pole of analog output signal 0
3.6	AO1+	Positive pole of analog output signal 1

Terminal	Signal	Description
3.7	AO-	Negative pole of analog output signals 0 and 1
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	C8	Signal of the configurable digital input/output C8
4.1	C9	Signal of the configurable digital input/output C9
4.2	C10	Signal of the configurable digital input/output C10
4.3	C11	Signal of the configurable digital input/output C11
4.4	C12	Signal of the configurable digital input/output C12
4.5	C13	Signal of the configurable digital input/output C13
4.6	C14	Signal of the configurable digital input/output C14
4.7	C15	Signal of the configurable digital input/output C15
4.8	UP	Process voltage UP (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

NOTICE!

Risk of imprecise and faulty measurements!

Analog signals may be distorted seriously by external electromagnetic influences.

Use shielded wires when wiring analog signal sources. The cable shield must be grounded at both ends of the cable. Provide a potential equalisation of a low resistance to avoid high potential differences between different parts of the plant.

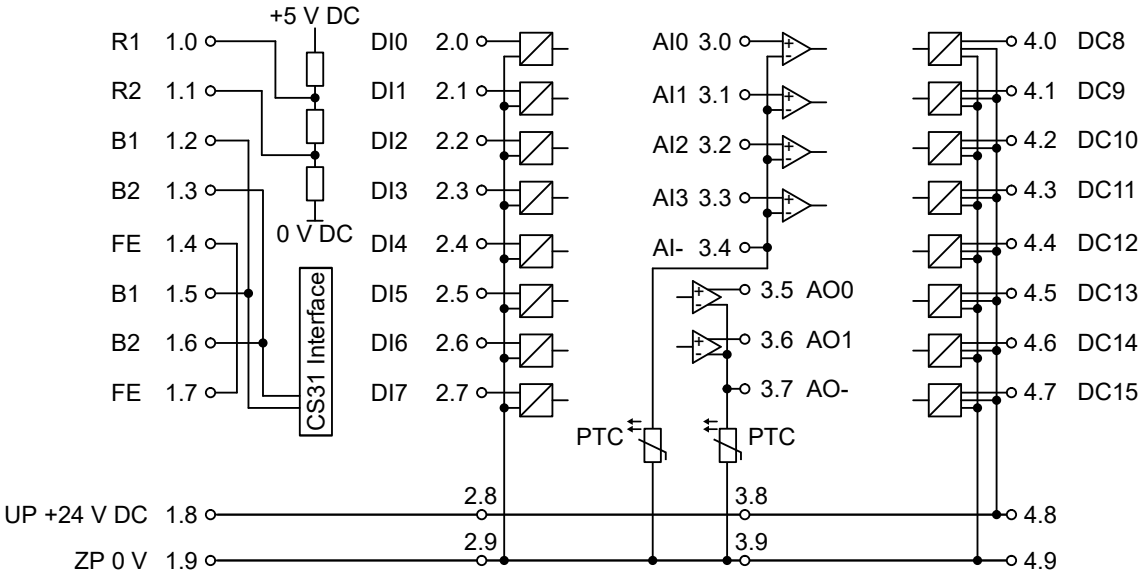


Fig. 958: Terminal assignment of the CS31 bus module CI592-CS31

The module provides several diagnosis functions *Chapter 1.6.2.8.3.2.9 “Diagnosis” on page 4785*.

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

The meaning of the LEDs is described in the section Status LEDs *Chapter 1.6.2.8.3.2.10 “State LEDs” on page 4788*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.

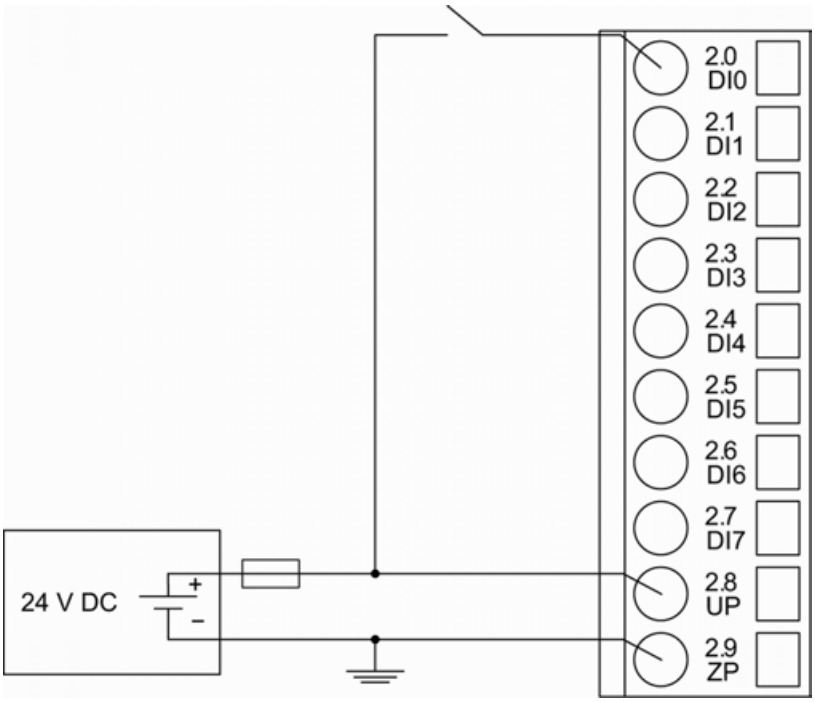


Fig. 959: Connection of the digital inputs

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC8 and DC9. DC8 is connected as an input and DC9 is connected as an output. Proceed with the configurable digital inputs/outputs DC10 to DC15 in the same way.

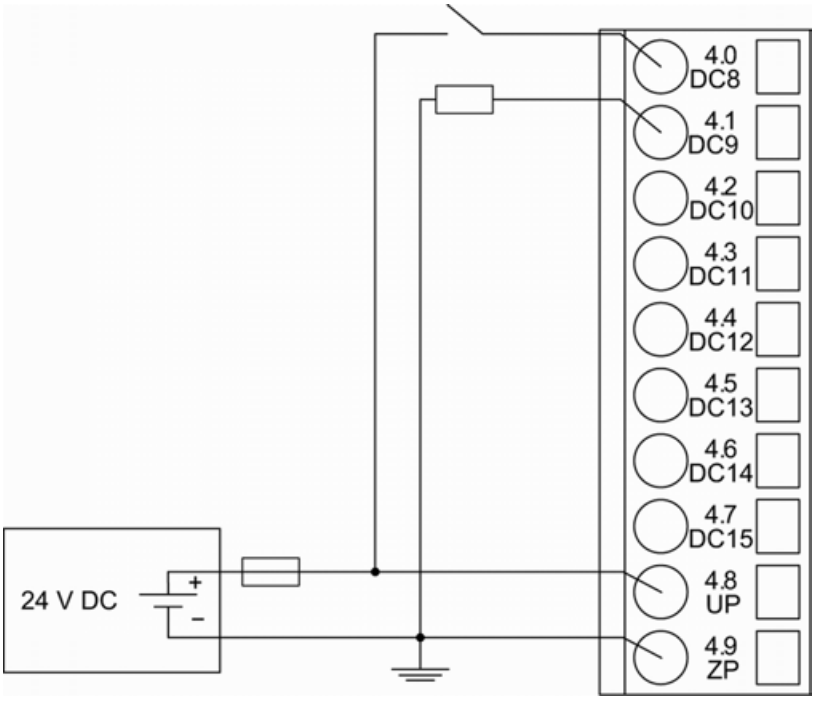


Fig. 960: Connection of configurable digital inputs/outputs

CAUTION!

Risk of influences to the connected sensors!

Some sensors may be influenced by the deactivated module outputs of CI592-CS31.

If using inputs as fast counter inputs, connect a 470 Ω / 1 W resistor in series to configurable inputs/outputs DC8/DC9.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow to build the necessary voltage drop for the evaluation. For this, the module CI592-CS31 provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

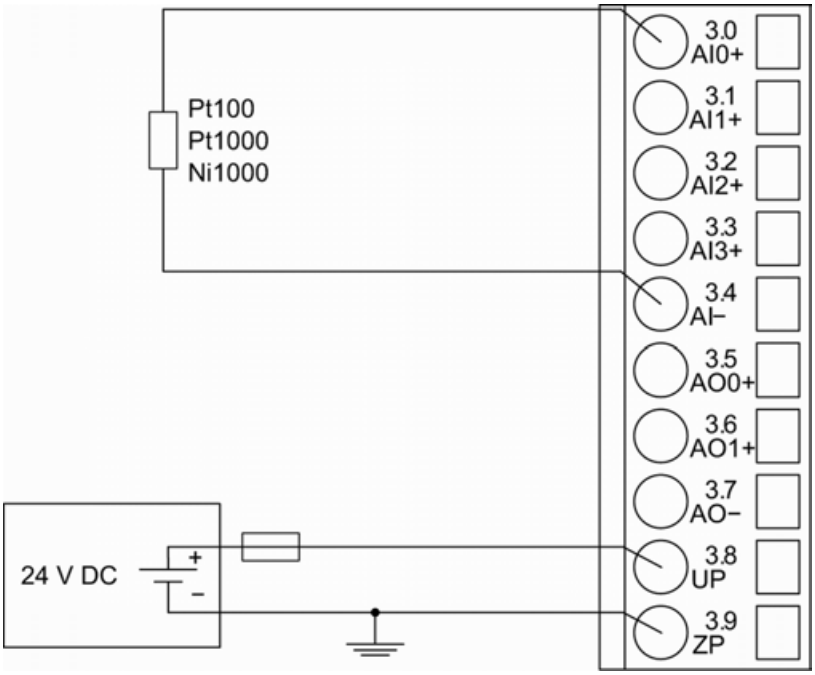


Fig. 961: Connection of resistance thermometers in 2-wire configuration to the analog inputs

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

The module CI592-CS31 performs a linearization of the resistance characteristic.

Configure unused analog input channels as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow to build the necessary voltage drop for the evaluation. For this, the module CI592-CS31 provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

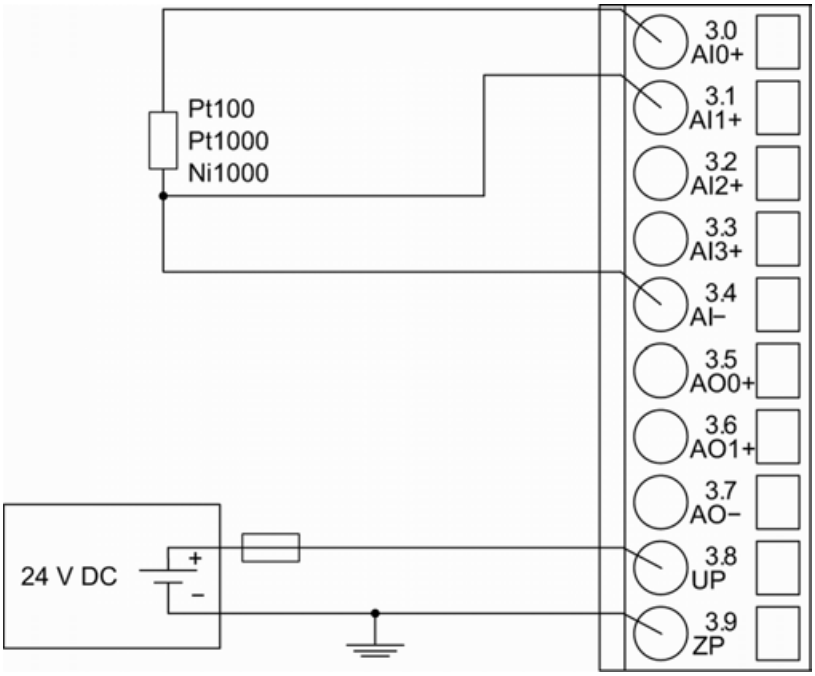


Fig. 962: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	3-wire configuration, 2 channels used
Pt1000	3-wire configuration, 2 channels used
Ni1000	3-wire configuration, 2 channels used

The measuring ranges are described in the section Measuring Ranges ↬ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↬ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

The module CI592-CS31 performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

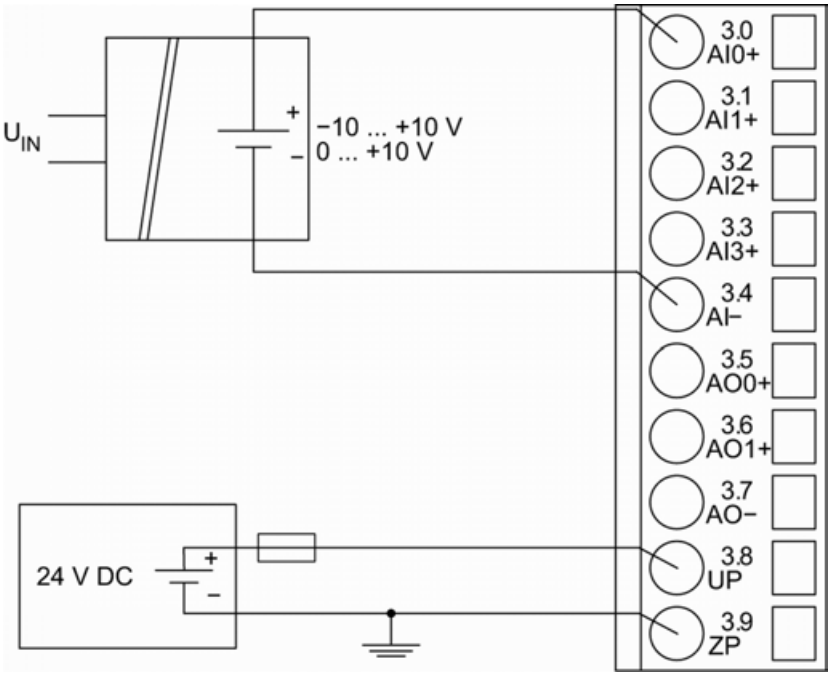


Fig. 963: Connection of active-type analog sensors (voltage) with galvanically power supply to the analog inputs

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

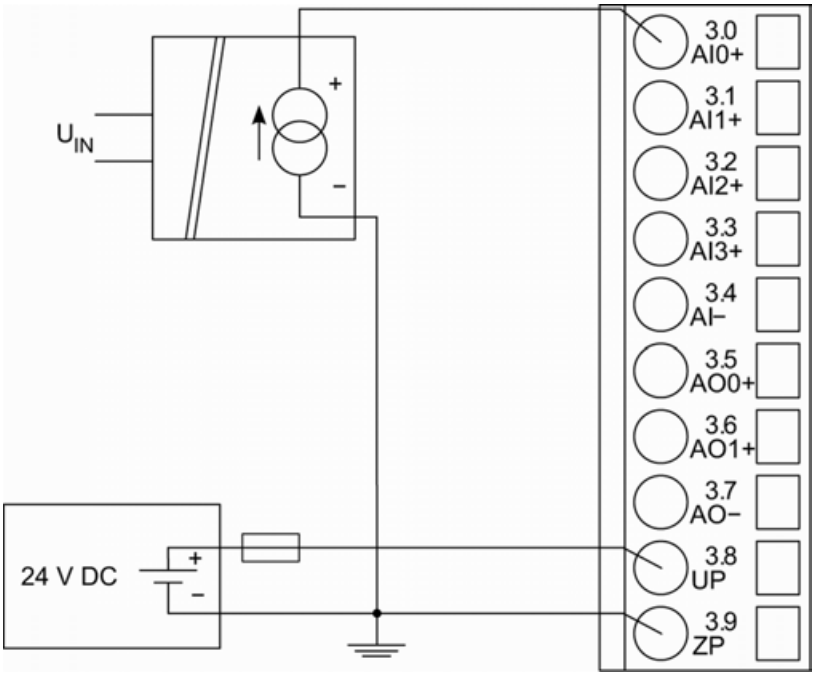


Fig. 964: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

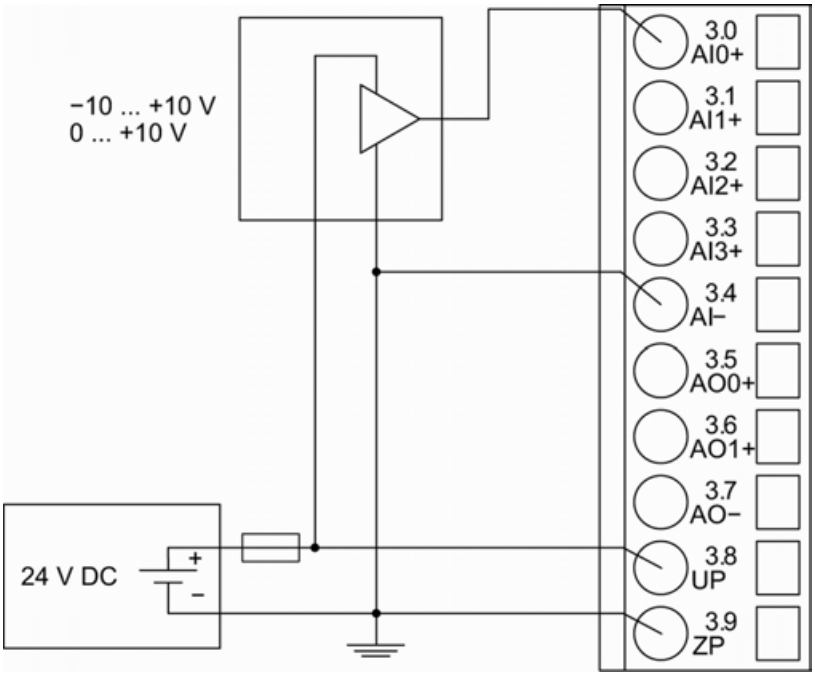


Fig. 965: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

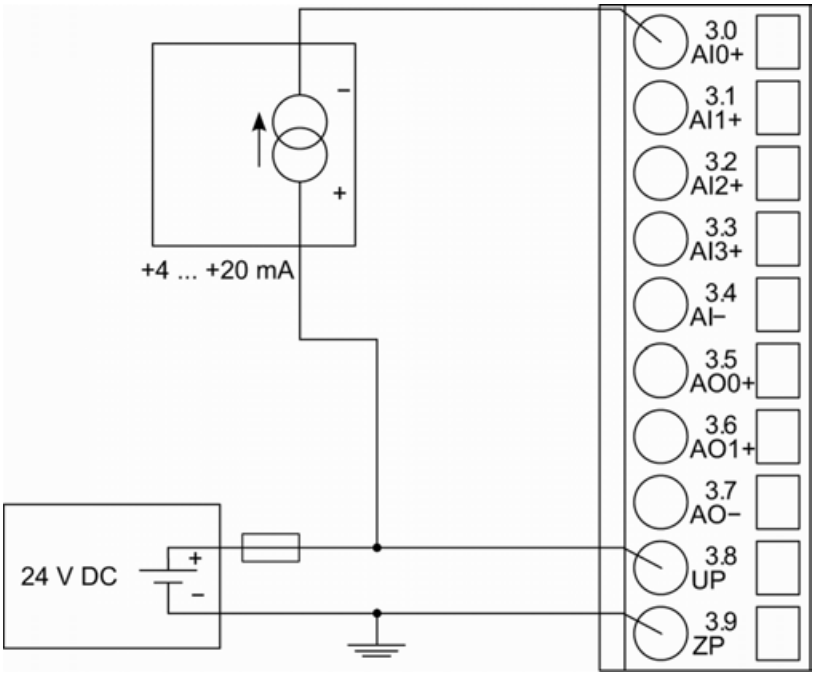


Fig. 966: Connection of passive-type analog sensors (current) to the analog inputs

Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 "Parameterization" on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 "Measuring ranges" on page 4789*:

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to I+ and I-.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

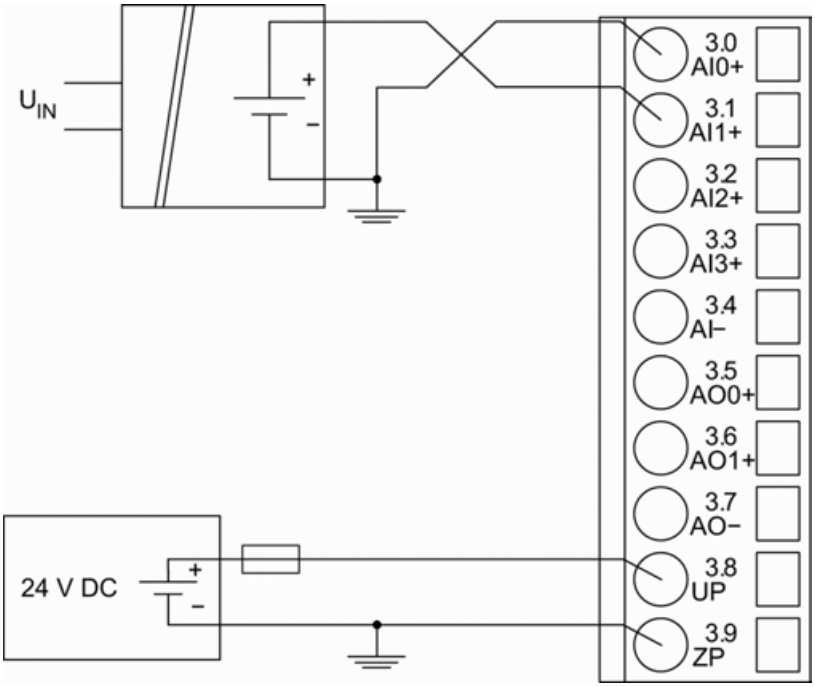


Fig. 967: Connection of active-type analog sensors (voltage) to differential analog inputs

Voltage	0...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

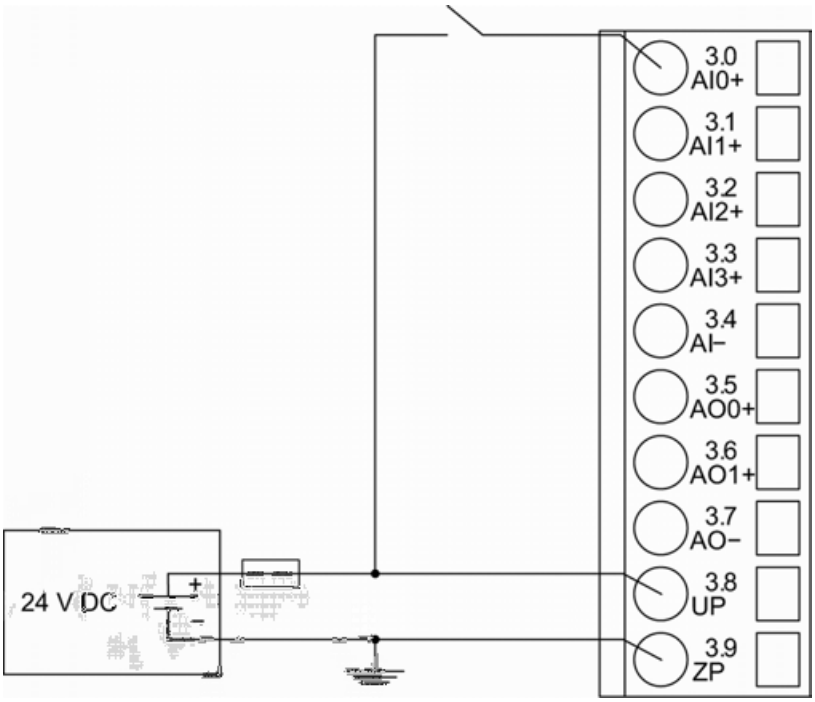


Fig. 968: Use of analog inputs as digital inputs

Digital input	24 V	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 “Parameterization” on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 “Measuring ranges” on page 4789*:

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

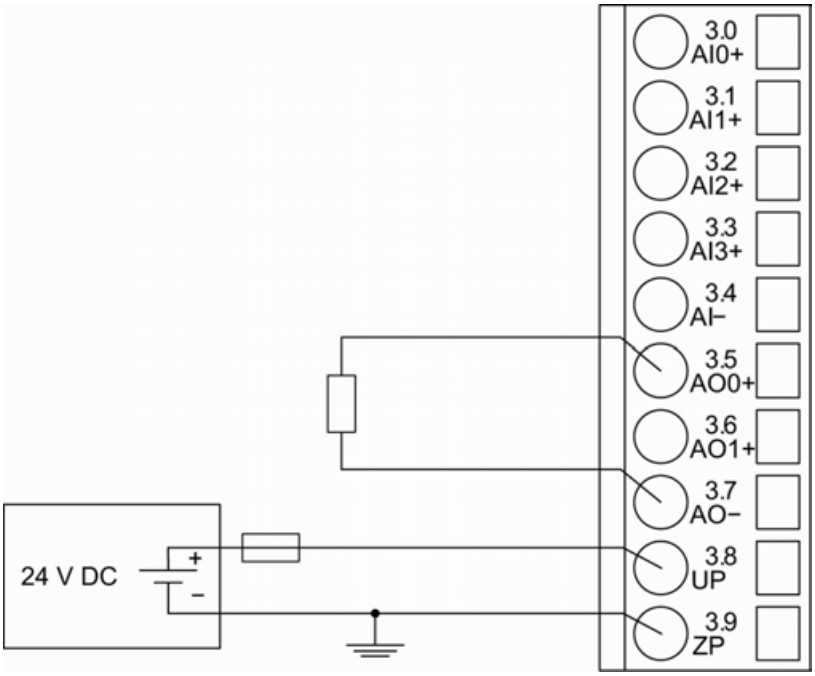


Fig. 969: Connection of analog output loads (voltage)

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 "Parameterization" on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 "Measuring ranges" on page 4789*:

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

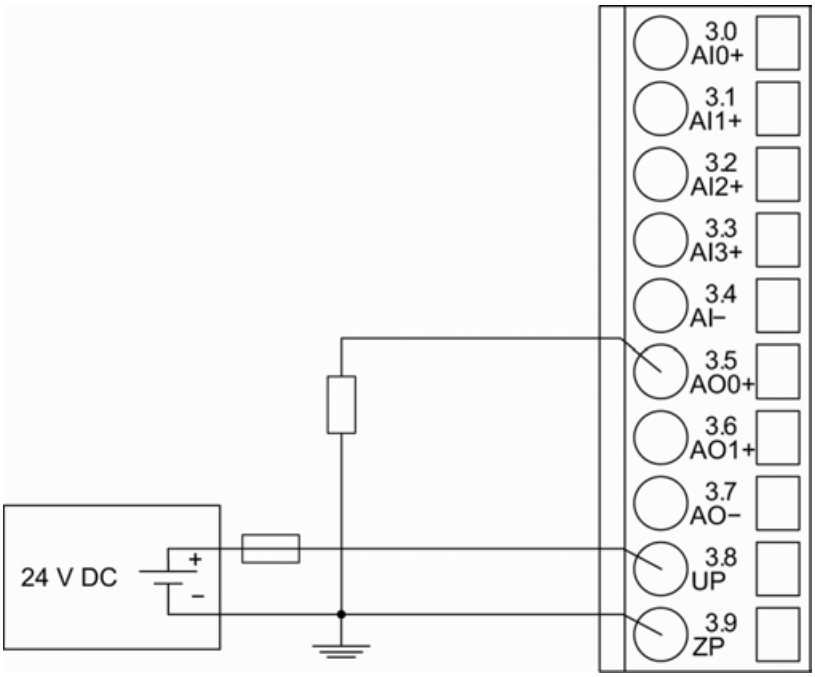


Fig. 970: Connection of analog output loads (current)

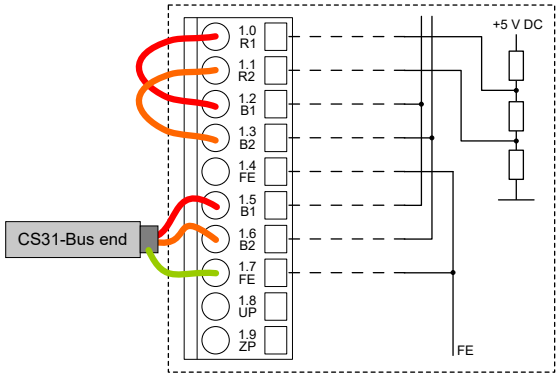
Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.3.2.8 "Parameterization" on page 4780* ↳ *Chapter 1.6.2.8.3.2.11 "Measuring ranges" on page 4789*:

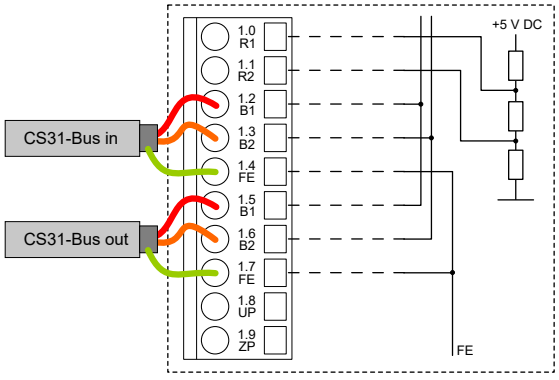
Unused analog outputs can be left open-circuited.

CS31 bus connections

The following figures show the different possibilities of connecting the CS31 buses to the CI592-CS31:



Connection of CS31-Bus - the CI592-CS31 is located at the bus end



Connection of CS31-Bus - the CI592-CS31 is located at the middle of the bus

Details on CS31 wiring is described seperately *Chapter 1.6.3.6.4.8 “CS31 bus” on page 5347*.

Internal data exchange

	without the fast counter	**with the fast counter (only with AC500)**
Digital inputs (bytes)	2 + communication interface modules	4 + communication interface modules
Digital outputs (bytes)	1 + communication interface modules	3 + communication interface modules
Analog inputs (words)	4 + communication interface modules	4 + communication interface modules
Analog outputs (words)	2 + communication interface modules	2 + communication interface modules
Counter input data (words)	0	4
Counter output data (words)	0	8

I/O configuration

The CI592-CS31 module does not store configuration data itself. The configurable channels are defined as inputs or outputs by the user program, i.e. each of the configurable channels can be used as input or output (or re-readable output) by interrogation or allocation by the user program.

Addressing

An address must be set at every module so that the field bus communication module can access the specific inputs and outputs.

A detailed description concerning "addressing" can be found in the chapters "Addressing" of the CPUs and Communication Modules.

The address (00d to 99d) is set with two rotary switches on the front panel of the module.

The CS31 Bus Module reads the position of the address switches only during the initialization after power ON, i.e. changes of the setting during operation remain ineffective.

Parameterization

Parameters of the module - if used with fast counter

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	2725	WORD	2725
Parameter length	Internal	22	BYTE	22
Error LED / Fail-safe function [2])	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + fail-safe	17		
	Off by E3 + fail-safe	19		
Check supply	off	0	BYTE	
	on	1		1

If the communication interface module is configured as a fast counter module and '0 - no Counter' in Automation Builder is selected the channel ERR LEDs stays on and the module does not start up. The address was adjusted with '71'.

Only the '0- no Counter' mode does not operate. If any other counter is selected e.g. '1-1 Up counter' the module starts up and can be utilized.

Parameters of the module - if used without fast counter

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	2726	WORD	2726
Parameter length	Internal	23	BYTE	23
Error LED / Fail-safe function [2])	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + fail-safe	17		
	Off by E3 + fail-safe	19		

Name	Value	Internal value	Internal value, type	Default
Check supply	Off	0	BYTE	
	On	1		1

Remarks:

[1]) With a faulty Module ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission

[2]) Error LED/Failsafe function:

Setting	Description
On	Error-LED lights up at errors of all error classes, Failsafe mode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe mode off
On +Failsafe	Error-LED lights up at errors of all error classes, Failsafe mode on *)
Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

*) The parameters behaviourAOatCommunicationFault and behaviourDOatCommunicationFault are only analyzed if the Failsafe mode is ON.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Fast counter	No counter 1 Up counter 1 Up counter with release input 2 UpDown counters 2 UpDown (2. On falling edges) 1 Updown dynamic set/ rising edge 1 Updown dynamic set/ falling edge 1 UpDown directional discriminator Reserved 1 UpDown directional discriminator x2 1 UpDown directional discriminator x4	0 1 2 3 4 5 6 7 8 9 10	BYTE	0
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error *)	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000

*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behaviour AO at comm. error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0

*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	see table [1])	see table [1])	BYTE	0
Input 0, Check channel	see table [2])	see table [2])	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	see table [1])	see table [1])	BYTE	0
Input 3, Check channel	see table [2])	see table [2])	BYTE	0

Table 514: Channel configuration [1])

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)

Internal value	Operating modes of the analog inputs, individually configurable
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 515: Channel monitoring [2])

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	see table [3])	see table [3])	BYTE	0
Output 0, Check channel	see table [4])	see table [4])	BYTE	0
Output 0, Substitute value	see table [5])	see table [5])	WORD	0
Output 1, Channel configuration	see table [3])	see table [3])	BYTE	0
Output 1, Check channel	see table [4])	see table [4])	BYTE	0
Output 1, Substitute value	see table [5])	see table [5])	WORD	0

Table 516: Channel configuration [3])

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 517: Channel monitoring [4])

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 518: Substitute value [5])

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6...7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0...5**	**PNIO diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	**[2])**	**[3])**	**[4])**				
Module errors CI592-CS31								
3	11	ADR	31	31	19	Checksum error in the I/O module		Replace I/O module
3	11	ADR	31	31	3	Timeout in the I/O module		

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
3	11	ADR	31	31	40	Different hard-/firm-ware versions in the module	
3	11	ADR	31	31	43	Internal error in the module	
3	11	ADR	31	31	36	Internal data exchange failure	
3	11	ADR	31	31	9	Overflow diagnosis buffer	Restart
3	11	ADR	31	31	26	Parameter error	Check master
3	11	ADR	31	31	11	Process voltage UP too low	Check process supply voltage
3	11	ADR	31/1...7	31	17	No communication with I/O module	Replace I/O module
3	11	ADR	1...7	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration
4	11	ADR	31	31	45	Process voltage UP OFF	Turn process voltage ON
4	11	ADR	1...7	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	11	ADR	31/1...7	31	34	No response during initialization of the I/O module	Replace I/O module
Channel error digital CI592-CS31							
4	11	ADR	31/1...7	14...21 [5])	47	Short circuit at dig-ital output	Check terminals

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
Channel error analog CI592-CS31							
4	11	ADR	31/1...7	8...11 [6])	48	Analog value over-flow or broken wire at an analog input	Check value or check terminals
4	11	ADR	31/1...7	8...11 [6])	7	Analog value underflow at an analog input	Check value
4	11	ADR	31/1...7	8...11 [6])	47	Short-circuit at an analog input	Check terminals
4	11	ADR	31/1...7	12...13 [7])	4	Analog value over-flow at an analog output	Check output value
4	11	ADR	31/1...7	12...13 [7])	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: 14 = I/O bus, 11 = COM1 (e.g. CS31 bus), 12 = COM2. The FBP diagnosis block does not contain this identifier.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1...7 = expansion module 1...7, ADR = Hardware address (e.g. of the DC551)
[3])	With "Module" the following allocation applies: 31 = Module itself; 1...7 = expansion 1...7
[4])	In case of module errors, with channel "31 = Module itself" is output.
[5])	Ch = 14...21 indicates the digital inputs/outputs DC8...DC15
[6])	Ch = 8...11 indicates the analog inputs AI0...AI3
[7])	Ch = 12...13 indicates the analog outputs AO0...AO1

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 4 system LEDs (PWR, CS31, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 26 process LEDs (UP, inputs, outputs, CH-ERR2 to CH-ERR4) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 519: State of the 4 system-LEDs:

LED	State	Color	OFF	ON	Flashing
PWR/RUN	System voltage	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
CS31	CS31 communication	Green	No communication at the CS31 bus module	Communication at the CS31 bus OK	Diagnosis mode
S-ERR	Sum Error	Red	No error	Internal error	--
I/O-Bus	Communication via the I/O bus	Green	No communication interface module connected or communication error	Communication interface module connected and operational	---

Table 520: State of the 27 process LEDs:

LED	State	Color	OFF	ON	Flashing
DI0 to DI7	Digital input	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DC8 to DC15	Digital input/ output	Yellow	Input/output is OFF	Input/output is ON (the input voltage is even displayed if the supply voltage is OFF)	--
AI0 to AI3	Analog input	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--

LED	State	Color	OFF	ON	Flashing
AO0 to AO1	Analog output	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
UP	Process supply voltage 24 V DC via terminal	Green	Process supply voltage is missing	Process supply voltage OK	--
CH-ERR2	Channel Error, error messages in groups (digital inputs/outputs combined into the groups 1, 2, 3, 4)	Red	No error or process supply voltage is missing	Severe error within the corresponding group	Severe error within the corresponding group (e.g. short-circuit at an output)
CH-ERR3		Red			
CH-ERR4		Red			
CH-ERR *)	Module Error	Red	--	Internal error	--
*) All of the LEDs CH-ERR2 to CH-ERR4 light up together					

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input range resistor

Range	**Pt100 / Pt1000** **-50...400 °C**	**Ni1000** **-50...150 °C**	**Digital value**	
			Decimal	**Hex.**
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
			800 : 701	0320 : 02BD
Normal range	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	**-10...+10 V**	**0...20 mA**	**4...20 mA**	**Digital value**	
				Decimal	**Hex.**
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
	-0.0004 V	0 mA	3.9994 mA	-1	FFFF
	:	:	0 mA	-6912	E500
	-10.0000 V	0 mA	0 mA	-27648	9400
Measured value too low	-10.0004 V	0 mA	0 mA	-27649	93FF
	:	:	:	:	:
	-11.7589 V	0 mA	0 mA	-32512	8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the module

Parameter		Value
Process supply voltage UP:		
	Rated value	24 V DC
	Protection against reverse voltage	Yes
	Rated protection fuse at UP	10 A fast
Current consumption		
	From UP	0.07 A + max. 0.5 A per output
	From 24 V DC power supply at the terminals UP/L+ and ZP/M of the CPU/ communication interface module (depending on system architecture)	5 mA
	Inrush current from UP (power-up)	0.040 A^2s
Interface		RS-485
Protocol		CS31
Galvanic isolation		Yes, CS31 bus from the rest of the module
Max. power dissipation within the module		6 W (outputs unloaded)

Parameter	Value
Rotary switch	2 rotary switches on the front panel for setting the module's address
Operating and error displays	30 LEDs (totally)
Weight (without terminal unit)	Approx. 125 g

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable digital I/O channels can be defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter	Value
Number of channels per module	8 inputs/outputs (with transistors)
Distribution of the channels into groups	1 group for 8 channels
If the channels are used as inputs	
Channels DC8...DC15	Terminals 4.0...4.7
If the channels are used as outputs	
Channels DC8...DC15	Terminals 4.0...4.7
Indication of the input/output signals	1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation	Yes, per module

Technical data of the digital inputs/outputs if used as inputs

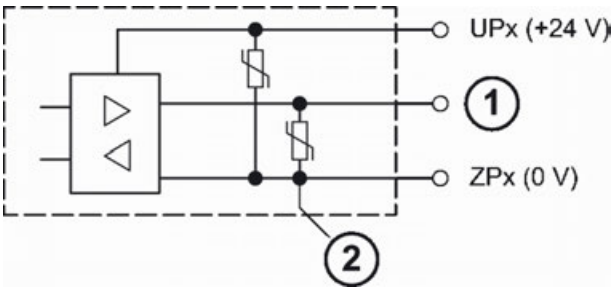
Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DC8 to DC15	Terminals 4.0 to 4.7
Reference potential for all inputs	Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC
0-Signal	-3 V...+5 V *)
Undefined Signal	> +5 V...< +15 V
1-Signal	+15 V...+30 V
Ripple with signal 0	Within -3 V...+5 V *)
Ripple with signal 1	Within +15 V...+30 V
Input current per channel	
Input voltage +24 V	Typ. 5 mA
Input voltage +5 V	> 1 mA
Input voltage +15 V	> 2 mA
Input voltage +30 V	< 8 mA
Max. cable length	
Shielded	1000 m
Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC8 to DC15		Terminals 4.0 to 4.7
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminals 1.8, 2.8, 3.8 and 4.8 (positive pole of the supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
Fuse for UP		10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	DC8 / DC9
Used outputs	DC10
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for AI0+ to AI3+		Terminal 3.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current and Digital Input ↳ *Chapter 1.6.2.8.3.2.11.1 “Input ranges voltage, current and digital input” on page 4789* and Input Range Resistor ↳ *Chapter 1.6.2.8.3.2.11.2 “Input range resistor” on page 4790*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 3.0 to 3.3
Reference potential for the inputs		Terminals 1.9, 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 3.5 and 3.6
Reference potential for AO0+ to AO1+		Terminal 3.7 (AO-) for voltage output Terminals 1.9, 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules

Parameter	Value
Configurability	-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output	0 Ω...500 Ω
Output loadability, as voltage output	±10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Table Output Ranges Voltage and Current *Chapter 1.6.2.8.3.2.11.3 “Output ranges voltage and current” on page 4790*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

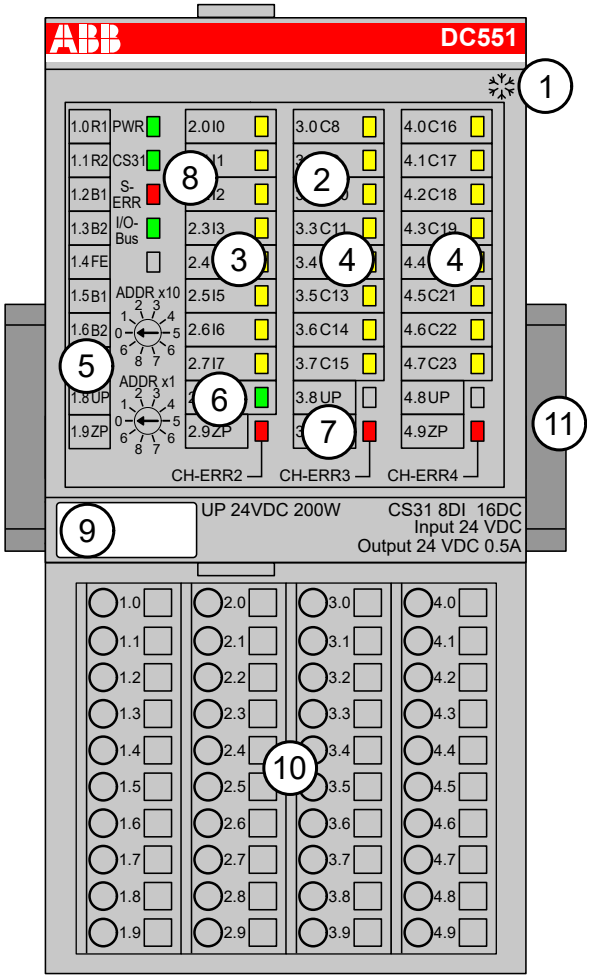
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 221 200 R0001	CI592-CS31, CS31 communication interface module with 8 DI, 8 DC, 4 AI, 2 AO	Active
1SAP 421 200 R0001	CI592-CS31-XC, CS31 communication interface module with 8 DI, 8 DC, 4 AI, 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

DC551-CS31 - Digital inputs and outputs

- 8 digital inputs 24 V DC, 16 configurable digital inputs/outputs
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital inputs I0 to I7
4 16 yellow LEDs to display the signal states of the digital inputs/outputs C8 to C23
5 2 rotary switches to set the module's address (00d to 99d)
6 1 green LED to display the process voltage UP
7 3 red LEDs to display errors
8 4 system LEDs
9 Label
10 Terminal unit
11 DIN rail
❄ Sign for XC version

Intended purpose

> *The CS31 communication interface module DC551-CS31 can only be used together with the AC500 CPUs and dedicated PS501 control builder.*

The CS31 communication interface module is used as a decentralized I/O module on CS31 field buses. The bus connection is performed on a RS-485 serial interface, which allows the connection of this module to all existing CS31 buses. In addition, the CS31 communication interface module provides 24 I/O channels with the following properties:

- 8 digital inputs 24 V DC in one group (2.0...2.7)
- 16 digital inputs/outputs in one group (3.0...4.7), of which each can be used
- as an input,

- as a transistor output with short circuit and overload protection, 0.5 A rated current or
- as a re-readable output (combined input/output) with the technical data of the digital inputs and outputs.

The inputs and output are galvanically isolated from the other electronic circuitry of the module.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Interface	RS-485, CS31 protocol
Supply of the module's electronic circuitry	From UP and ZP (power supply)
Supply of the electronic circuitry of the I/O modules attached	Through the bus interface (I/O bus)
Address switches	For setting the CS31 field bus address (0 to 99)
Digital inputs	8 (24 V DC)
Digital inputs/outputs	16 (24 V DC)
Fast Counter	Integrated, many configurable operating modes
LED displays	For system displays, signal statuses, errors and power supply
External supply voltage	Via the terminals ZP and UP (process voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU551-CS31 or TU552-CS31 ↳ *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*

Connections

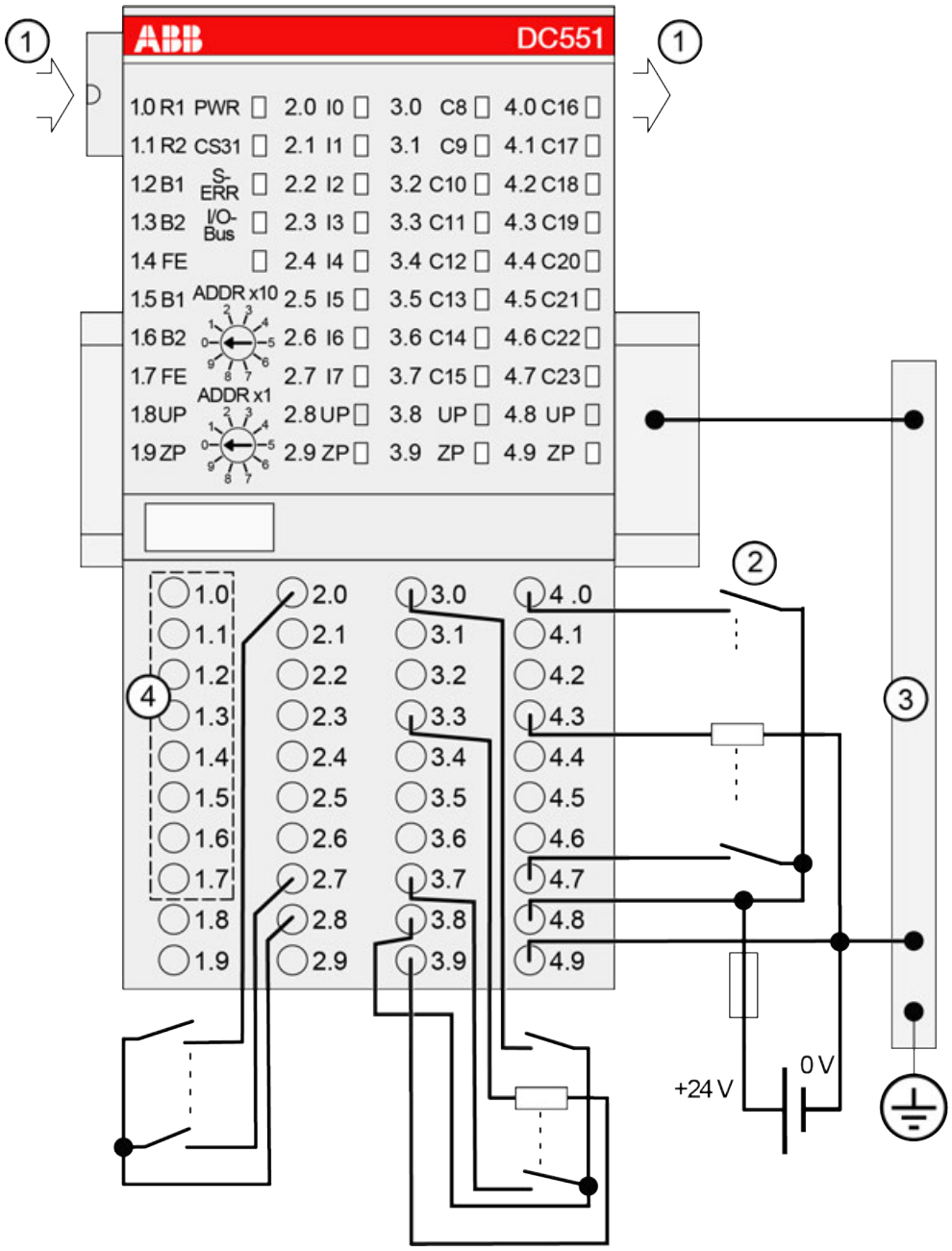
The CS31 communication interface module is plugged on the CS31 terminal unit TU551 or TU552 ↳ *Chapter 1.6.2.5.7 “TU551-CS31 and TU552-CS31 for CS31 communication interface modules” on page 4121*. Hereby, it clicks in with two mechanical locks. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 40 terminals of the CS31 terminal unit. It is possible, to replace CS31 bus modules and I/O modules without loosening the wiring.

The terminals 1.8 to 4.8 and 1.9 to 4.9 are electrically interconnected within the terminal unit and always have the same assignment, irrespective of the inserted module:

- Terminals 1.8 to 4.8: process voltage UP = +24 V DC
- Terminals 1.9 to 4.9: process voltage ZP = 0 V

The assignment of the other terminals depends on the inserted CS31 bus module.



1 I/O bus
2 4.0 - 4.7: Connected with UP (switch) -> Input;
Connected with ZP (load) -> Output
3 Switchgear cabinet earth
4 1.0 - 1.7: *Chapter 1.6.2.8.3.3.4 “CS31 bus connections” on page 4801*

Assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	RS-485	CS31 bus interface
2.0 to 2.7	I0 to I7	8 digital inputs
3.0 to 4.7	C8 to C23	16 digital inputs/outputs

CAUTION!
The process supply voltage must be included in the grounding concept (e. g. grounding of the negative pole).

The supply voltage 24 V DC for the module's electronic circuitry comes from the ZP/UP terminals.

The module provides several diagnosis functions *Chapter 1.6.2.8.3.3.11 “Diagnosis” on page 4807*).

WARNING!
Removal/Insertion under power
The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

CAUTION!
Risk of influences to the connected sensors!
Some sensors may be influenced by the deactivated module outputs of DC551-CS31.

Connect a 470 Ω / 1 W resistor in series to inputs C16/C17 if using them as fast counter inputs to safely avoid any influences.

CS31 bus connections

The CS31 bus is connected through the terminal unit with the terminals 1.0 to 1.7. The end-of-line resistor can also be activated by using external wire jumpers.

The following figure shows a CS31 communication interface module at the end of the CS31 bus (end-of-line resistor activated).

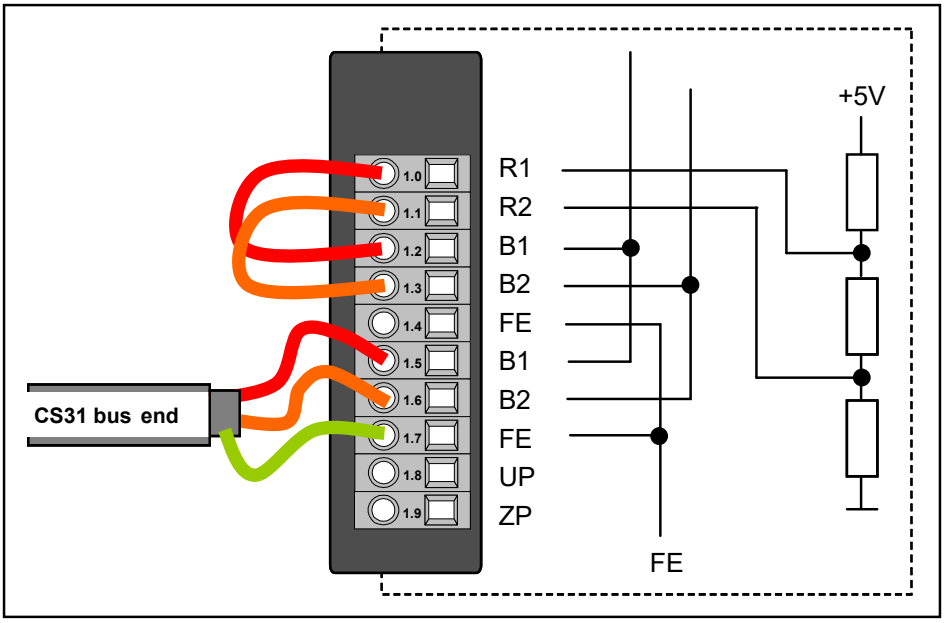


Fig. 971: CS31 bus module at the end of the CS31 Bus

The following figure shows a CS31 communciation interface module in the middle of a CS31 bus (end-of-line resistor not activated).

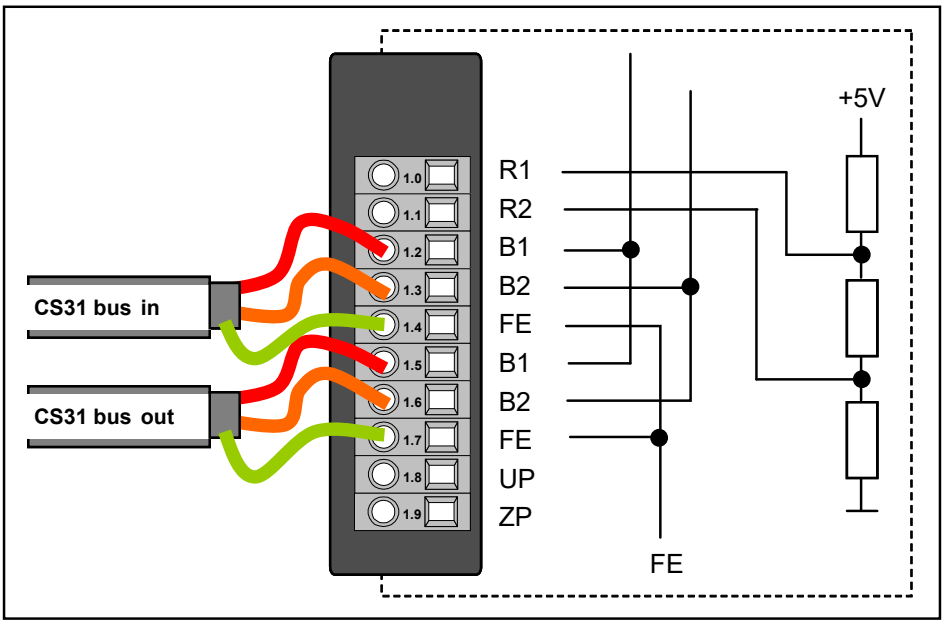


Fig. 972: CS31 communication interface module in the middle of the CS31 bus

Details on CS31 wiring is described seperately *Chapter 1.6.3.6.4.8 “CS31 bus” on page 5347*.

Internal data exchange

	without the fast counter	**with the fast counter (only with AC500)**
Digital inputs (bytes)	3 + expansion modules (see above)	5 + expansion modules (see above)
Digital outputs (bytes)	2 + expansion modules (see above)	4 + expansion modules (see above)

	without the fast counter	**with the fast counter (only with AC500)**
Counter input data (words)	0	5 (16 DI + 4 AI)
Counter output data (words)	0	9 (16 DO + 8 AO)

Addressing

An address must be set at every module so that the field bus communication module can access the specific inputs and outputs.

The address (00 to 99) is set with two rotary switches on the front panel of the module.

CS31 communication interface module reads the position of the address switches only during the initialization after power ON, i.e. changes of the setting during operation remain ineffective.

DC551-CS31 limitations

Digital I/O

DC551-CS31 is able to manage up to 240 digital I/O channels. It uses 2 digital bus addresses in this case.

The physical address to identify the I/O is	address n (switch address) for the 1st module (120 I/O)
	address n + 7 + bit 8/15 = 1 for the 2nd module

To be compatible with old CPU and EC500 using this physical address, to address I/O in user program: Use only 6 I/O modules with 32 DI.

Analog I/O

Analog limitation to 40 AI/AO with 4 bus addresses used.

Case of DC551-CS31 with fast counter

An additional bus address is used for "double word" values of the fast counter.

The maximum configuration is shown in the following table.

DC551-CS31 8DI + 16 DC + counter	16 AI	16 AI	DC532	DC532	DC532	DC532	DC532

The following configuration uses 7 bus addresses (the fast counter needs 16 DI + 16 DO + 4 AI + 8 AO):

2 bus addresses for digital I/O (24 + 16 + 5x32)DI + (16 + 16 + 5x16)DO = 200 DI (>120) + 112 DO

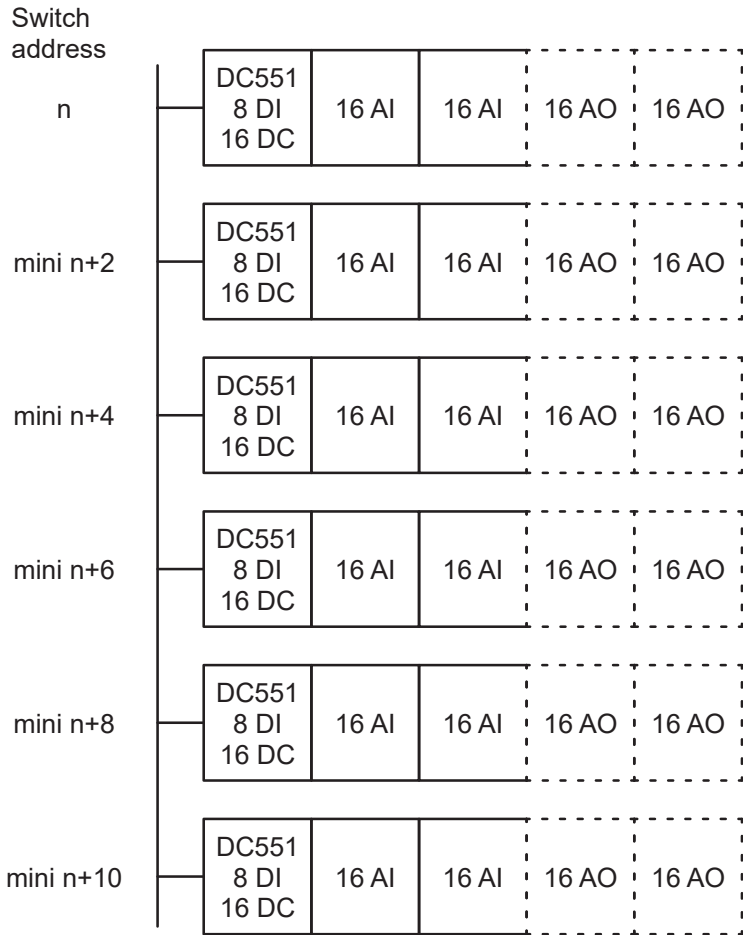
5 bus addresses for analog I/O (4 + 2x16)AI + 8 AO = 36 AI + 8 AO

If the communication interface module is configured as a fast counter module and '0 - no Counter' in Automation Builder is selected the channel ERR LEDs stays on and the module does not start up. The address was adjusted with '71'.

Only the '0- no Counter' mode does not operate. If any other counter is selected e.g. '1-1 Up counter' the module starts up and can be utilized.

Small overview of the addressing possibilities

Configuration example with 32 analog inputs with or without 32 analog outputs (fast counter not used) = 5 bus addresses by the communication interface module



If there are fewer analog outputs than analog inputs, no additional address is necessary. Change the type from "analog in" to "analog I/O".

- 30 bus addresses used, 1 bus address free
- 192 analog inputs (+ 192 analog outputs)
- 48DI / 96DC (144 DI / 96 DO for CS31 and user program)
- Switch address incremented to avoid control overlap.

In CPU table module switch address n will be seen as (idem for AC500 or old CPU):

- Address n, type digital I/O, 8 DI/16 DC
- Address n, type analog I or I/O, 8 AI (+ 8 AO)
- Address n + bit 8/15=1, type analog I or I/O, 8 AI (+ 8 AO)
- Address n+1, type analog I or I/O, 8 AI (+ 8 AO)
- Address n+1 + bit 8/15=1, type analog I or I/O, 8 AI (+ 8 AO)

I/O configuration

The DC551-CS31 module does not store configuration data itself. The 16 configurable channels are defined as inputs or outputs by the user program, i.e. each of the configurable channels can be used as input or output (or re-readable output) by interrogation or allocation by the user program.

Parameterization

No.	Name	Value	Internal value	Internal value, type	Default	Min.	Max.
1	Module ID	Internal	2715 [1])	Word	2715 0x0a9b	0	65535
2	Ignore module	No Yes	0 1	Byte	No 0x00		
14	Parameter length	Internal	8 (7 [4])	Byte	8 (7 [4])	0	255
16	Check supply	Off On	0 1	Byte	On 0x01		
17	Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	Byte	8 ms 0x02		
18	Fast counter	0 : 10 [3])	0 : 10	Byte	Mode 0 0x00		
Nr.+1	Detection short-circuit at outputs	Off On	0 1	Byte	On 0x01		
Nr.+1	Behaviour outputs at communication errors	Off Last value Substitute value	0 1 2	Byte	Off 0x00		
Nr.+1	Substitute value outputs Bit 15 = Output 15 Bit 0 = Output 0	0...65535	0...0xffff	Word	0		

[1]) With CS31 and addresses less than 70, the value is increased by 1

[3]) Counter operating modes ↬ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*, description of the fast counter ↬ *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

[4]) With CS31 and addresses less than 70, without the parameter Fast Counter

Structure of the diagnosis block of the DC551-CS31

If a DC551-CS31 module is connected via a CS31 bus, then the field bus master receives diagnosis information by an extended diagnosis block. The following table shows the structure of this diagnosis block:

Byte number	Description	Possible values
1	Data length (header included)	18
2	Diagnosis byte	0 = Communication with DC551-CS31 OK 1 = Communication with DC551-CS31 failed
3	DC551-CS31 diagnosis byte, module number	0 = DC551 (e.g. error at the integrated 8DI/16DC) 1 = 1st attached S500 I/O module ... 7 = 7th attached S500 I/O module
4	DC551-CS31 diagnosis byte, slot	According to the I/O bus specification passed on by modules to the fieldbus master
5	DC551-CS31 diagnosis byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
6	DC551-CS31 diagnosis byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description passed on by modules to the fieldbus master
7	DC551-CS31 diagnosis byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error Bit 5: 1 = Diag reset Bit 2 to bit 4: reserved Bit 1: 1 = explicit acknowledgement Bit 0: 1 = static error Passed on by modules to the fieldbus master Value = 0: static message for other systems, which do not have a coming/ leaving evaluation
8ff	Reserved	

Diagnosis

In case of overload or short-circuit, the outputs switch off automatically and try to switch on again cyclically. Therefore an acknowledgement of the outputs is not necessary. The LED error message, however, is stored.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**FBP diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	11	ADR	31	31	19	Checksum error in the I/O module		Replace I/O module
3	11	ADR	31	31	3	Timeout in the I/O module		
3	11	ADR	31	31	40	Different hard-/firm-ware versions in the module		
3	11	ADR	31	31	43	Internal error in the module		
3	11	ADR	31	31	36	Internal data exchange failure		
3	11	ADR	31	31	9	Overflow diagnosis buffer		New start
3	11	ADR	31	31	26	Parameter error		Check master
3	11	ADR	31	31	11	Process voltage too low		Check process voltage
3	11	ADR	1...7	31	17	No communication to the I/O module		Replace I/O module
4	11	ADR	31	31	45	Process voltage ON/OFF		Process voltage ON
4	11	ADR	31/1..7	31	34	No reply at initial-ization of the I/O module		Replace I/O module

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**FBP diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	11	ADR	31/1.7	31	32	Wrong I/O module in the slot	Replace I/O module or check configu-ration
Channel error DC551-CS31							
4	11	ADR	31/1..7	8..23	47	Short-circuit at a digital output	Check connec-tion

Remarks:

[1])	In AC500 the following interface identifier applies: 11 = COM1 (protocol CS31 bus only possible with COM1)
[2])	With "Device" and CS31 bus master, the hardware address of the DC551-CS31 (0...69) is output.
[3])	With "Module" the following allocation applies: 31 = Module itself, 1...7 = Expansion 1...7
[4])	In case of module errors, with channel "31 = Module itself" is output.

Status LEDs

The LEDs are on the front panels of the modules. There are two different groups:

- The 4 system LEDs (PWR, S-ERR, CS31 and I/O-Bus) show the operating status of the module and indicate possible errors.
- The 28 process LEDs (UP, inputs, outputs, CH-ERR2 to CH-ERR4) display the supply voltage and signal statuses of the inputs and outputs and indicate possible errors.

All of the S500 modules have LEDs to display operating statuses and errors.

LED	Status	Color	LED = OFF	LED = ON	LED flashes
PWR	System voltage	Green	Missing internal system voltage or field bus supply is missing	Internal system voltage is OK	--
CS31	CS31 communication	Green	No communication at the CS31 bus module	Communication at the CS31 bus OK	Diagnosis mode
S-ERR	Sum Error	Red	No error or system voltage is missing	Internal error (storing can be parameterized)	--
I/O-Bus	Communication via the I/O bus	Green	No I/O modules connected or data error	I/O modules connected	Error I/O bus
Reserved	Not defined	-	-	-	-
I0...I7	Digital inputs	Yellow	Input = OFF	Input = ON (the input voltage is even displayed if the supply voltage is OFF)	-
C8...C23	Digital inputs/ outputs	Yellow	Input/output = OFF	Input/output = ON (the input voltage is even displayed if the supply voltage is OFF)	-
UP	Process supply voltage and initialization	Green	Process voltage is missing	Process voltage OK	--
CH-ERR2	Channel Error, error messages in groups (digital inputs/outputs combined into the groups 2 to 4)	Red	No error	Severe error within the corresponding group	Error on one channel of the corresponding group (e.g. short-circuit at an output)
CH-ERR3		Red			
CH-ERR4		Red			
CH-ERR *)	Module Error	Red	No error or process voltage is missing	Internal error	--
*) All LEDs CH-ERR2 to CH-ERR4 light up together					

The status of the LEDs concerning the CS31 communication interface module in connection with the I/O modules is described in detail in the S500 system data.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Rated supply voltage of the module		24 V DC (UP/ZP)
Current consumption of the module (UP)		15 mA
Process voltage UP		
	Rated value	24 V DC (for inputs and outputs)
	Max. electric charge for the supply terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse at UP	10 A fast
	Galvanic isolation	CS31 bus interface from the rest of the module
	Inrush current from UP (at power-up)	0.040 A^2s
	Current consumption from UP at normal operation / with outputs	0.1 A + max. 0.008 A per input + max. 0.5 A per output
	Connections	Terminals 1.8 - 4.8 for +24 V (UP) and 1.9 - 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W (outputs unloaded)
Number of digital inputs		8
Number of configurable digital inputs/outputs		16
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Address setting		With 2 rotary switches on the front panel
Diagnosis		Diagnosis and Displays *Chapter 1.6.2.8.3.3.11 “Diagnosis” on page 4807*
Operating and error displays		32 LEDs altogether
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40°C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels I0 to I7		2.0 to 2.7
Terminals of the channels C8 to C23		3.0 to 4.7
Reference potential for all inputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Galvanic isolation		From the CS31 bus
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1-> 0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

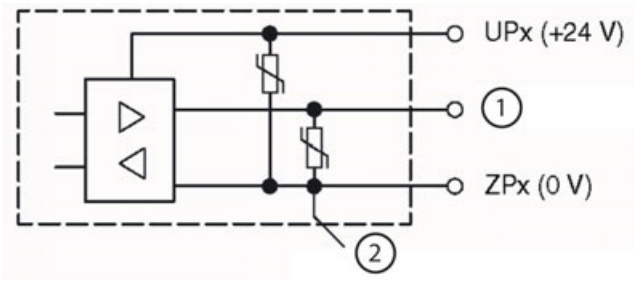
Parameter		Value
Number of channels per module		16 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group of 16 channels
If the channels are used as inputs		
	Channels I8...I23	Terminals 3.0...4.7
If the channels are used as outputs		
	Channels Q8...Q23	Terminals 3.0...4.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the CS31 bus

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		Max. 16 transistor outputs
Reference potential for all outputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Common power supply voltage		For all outputs: terminals 1.8...4.8 (positive pole of the process supply voltage, signal name UP)
Output voltage for signal 1		UP (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value, per channel	500 mA at UP = 24 V
	Maximum value (all channels together)	10 A
Leakage current with signal 0		< 0.5 mA
Rated protection fuse on UP		10 A fast
Demagnetization when inductive loads are switched off		With varistors integrated in the module (see figure below)
Switching frequency		
	With resistive loads	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	Max. 11 Hz with max. 5 W
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes
Max. cable length		

Parameter		Value
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital input/output
2 For demagnitization when inductive loads are switched off

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		Max. 16 digital inputs
Reference potential for all inputs		Terminals 1.9...4.9 (negative pole of the process supply voltage, signal name ZP)
Input current, per channel		Technical Data of the Digital Inputs
Input type acc. to EN 61131-2		Type 1
Input delay (0->1 or 1->0)		Typ. 8 ms, configurable from 0.1 to 32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		within -3 V...+5 V *)
Ripple with signal 1		within +15 V...+30 V
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the fast counter

Parameter	Value
Used inputs	C16 / C17
Used outputs	C18
Counting frequency	Max. 50 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 220 500 R0001	DC551-CS31, CS31 communication interface module, 8 DI and 16 DC	Active
1SAP 420 500 R0001	DC551-CS31-XC, CS31 communication interface module, 8 DI and 16 DC, XC version	Active

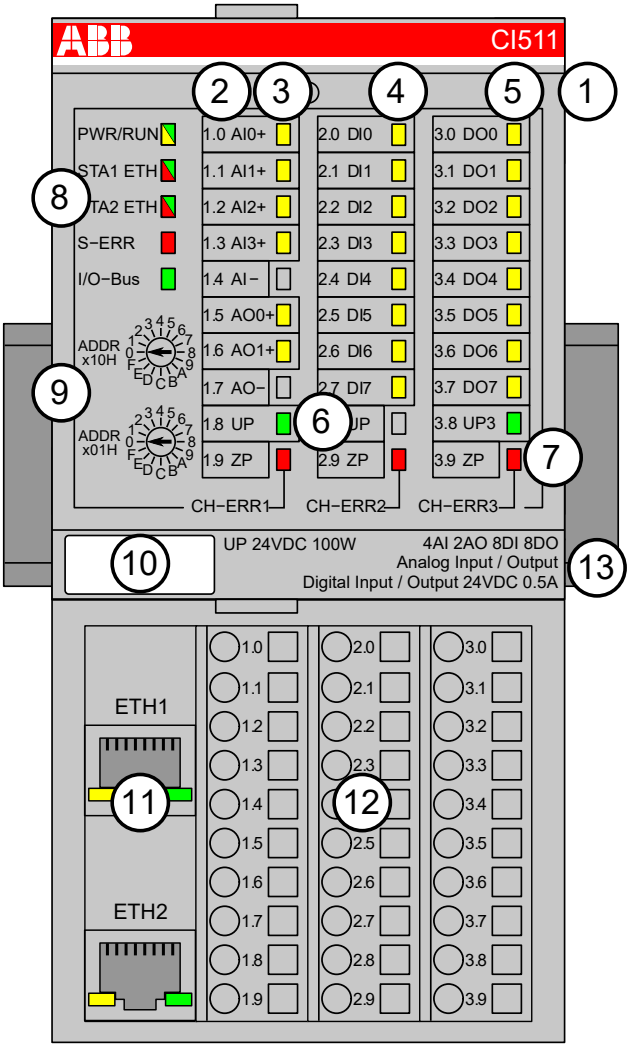
**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8.4 EtherCAT

CI511-ETHCAT

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- Cam switch functionality (see also Extended Cam Switch Library)
- Extended Cam switch functionality *) (see also Extended Cam Switch Library)
- Module-wise galvanically isolated - Expandability with up to 10 S500 I/O Modules *)

*) Applicable for device index C0 and above.



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, NET, DC, S-ERR, I/O-Bus
9 2 rotary switches (reserved for future extensions)
10 Label
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail

Intended purpose

The EtherCAT communication interface module CI511-ETHCAT is used as decentralized I/O module in EtherCAT networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)

- 8 digital outputs 24 V DC in 1 group (3.0...3.7)
- Cam switch functionality

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	EtherCAT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	Not used; reserved for future extensions
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 ↬ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet communication interface module CI511-ETHCAT is plugged on the I/O terminal unit TU507-ETH or TU508-ETH. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526).

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

The assignment of the other terminals:

Terminal	Signal	Description
1.0 to 1.3	AI0 to AI3	Positive pole of the 4 analog inputs
1.4	AI-	Negative pole of the analog inputs
1.5 to 1.6	AO0 to AO1	Positive pole of the 2 analog outputs
1.7	AO-	Negative pole of the analog outputs
2.0 to 2.7	DI0 to DI7	8 digital inputs
3.0 to 3.7	DO0 to DO7	8 digital outputs

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

CAUTION!

There is no galvanic isolation between the analog circuitry and ZP/UP. Therefore, the analog sensors must be galvanically isolated in order to avoid loops via the ground potential or the supply voltage.

CAUTION!

Because of their common reference potential, analog current inputs cannot be circuited in series, neither within the module nor with channels of other modules.

For the open-circuit detection (cut wire), each channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Analog signals are always laid in shielded cables. The cable shields are grounded at both ends of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

For simple applications (low disturbances, no high requirement on precision), the shielding can also be omitted.

The following figures show the connection of the Ethernet communication interface module CI511-ETHCAT.

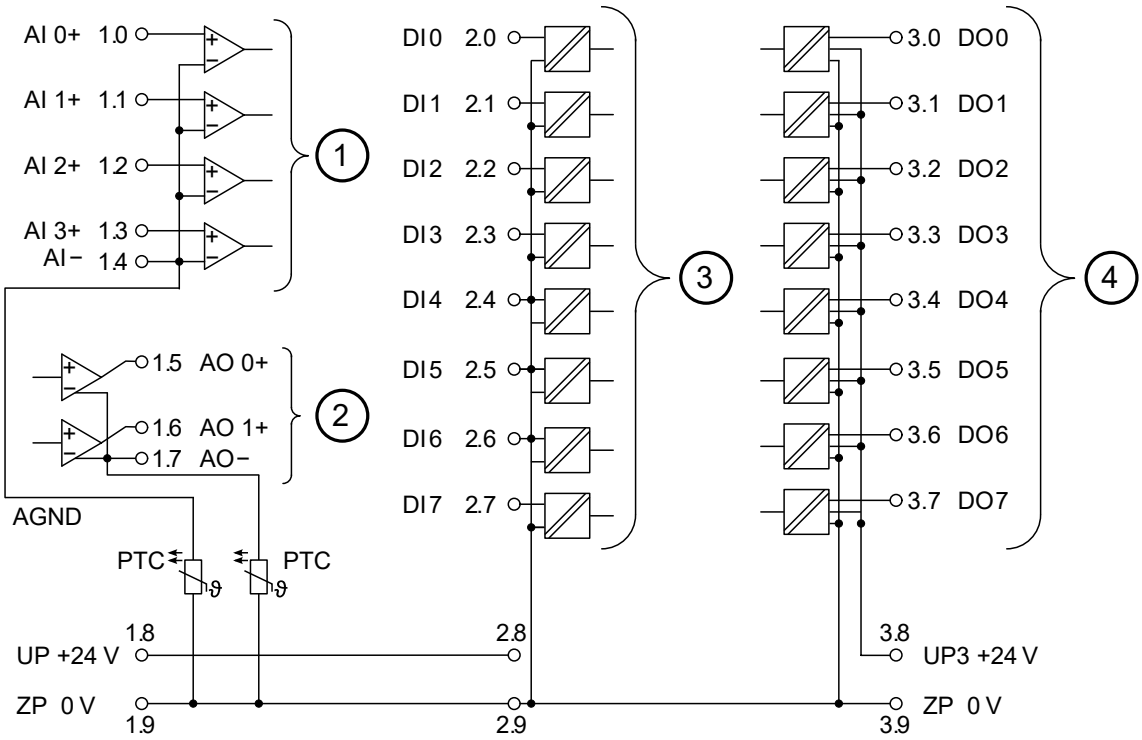


Fig. 973: Connection of the communication interface module CI511-ETHCAT

1 4 analog inputs, configurable for 0...10 V, -10...+10 V, 0/4...20 mA, Pt100/Pt1000, Ni1000 and digital signals
2 2 analog outputs, configurable for -10...+10 V, 0/4...20 mA
3 8 digital inputs 24 V DC
4 8 digital outputs 24 V DC, 0.5 A max.

In case of voltage feedback, 2 cases are distinguished:

1. The outputs are already active

The output group will be switched off. A diagnosis message will appear. After 5 seconds, the module tries automatic reactivation.

2. The outputs are not active

Only the output with voltage feedback will not be set to active. A diagnosis message will appear.

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

CAUTION!

The process supply voltage must be included within the grounding concept of the plant (e. g. grounding of the negative pole).

The module provide several diagnosis functions *Chapter 1.6.2.8.4.1.8 “Diagnosis” on page 4835*.

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

The function of the LEDs is described in the section State LEDs *Chapter 1.6.2.8.4.1.8 “Diagnosis” on page 4835*.

Connection of resistance thermometers in 2-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI511-ETHCAT provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration.

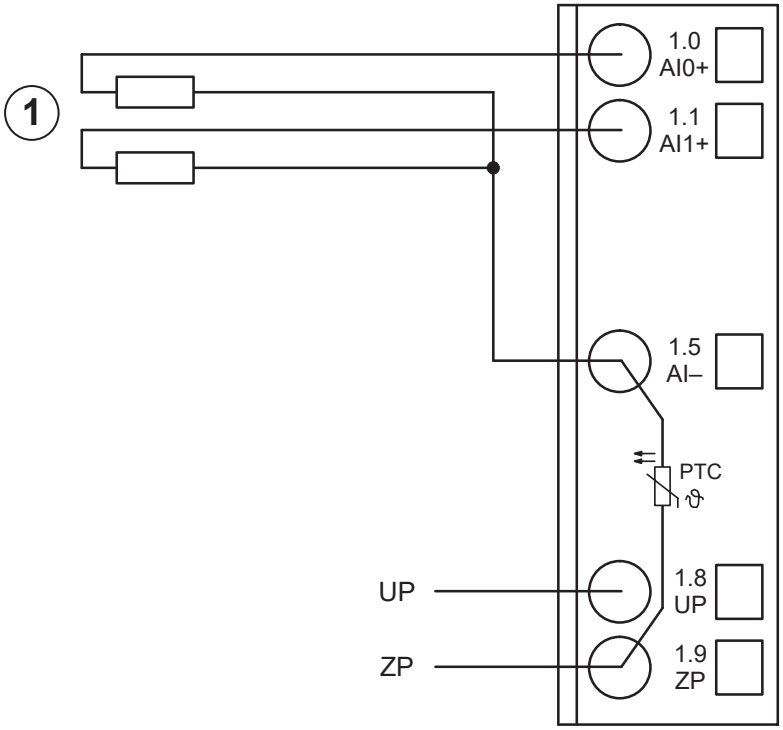


Fig. 974: Connection of resistance thermometers in 2-wire configuration

1 Pt100 (2-wire), Pt1000 (2-wire), Ni1000 (2-wire); 1 analog sensor requires 1 channel

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

The module CI511-ETHCAT performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of resistance thermometers in 3-wire configuration

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI511-ETHCAT provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration.

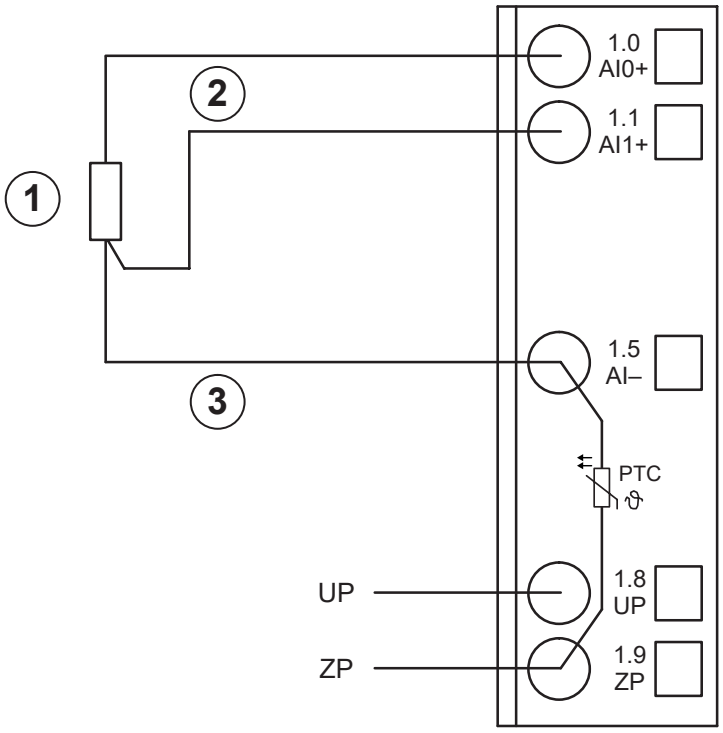


Fig. 975: Connection of resistance thermometers in 3-wire configuration

1 Pt100 (3-wire), Pt1000 (3-wire), Ni1000 (3-wire); 1 analog sensor requires 2 channels
2 Twisted pair within the cable
3 Return line: The return line is only needed once if measuring points are adjacent to each other. This saves wiring costs.

With 3-wire configuration, two adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary, to have all the involved conductors in the same cable. All the conductors must have the same cross section.

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

The module CI511-ETHCAT performs a linearization of the resistance characteristic.

In order to avoid error messages from unused analog input channels, it is useful to configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply

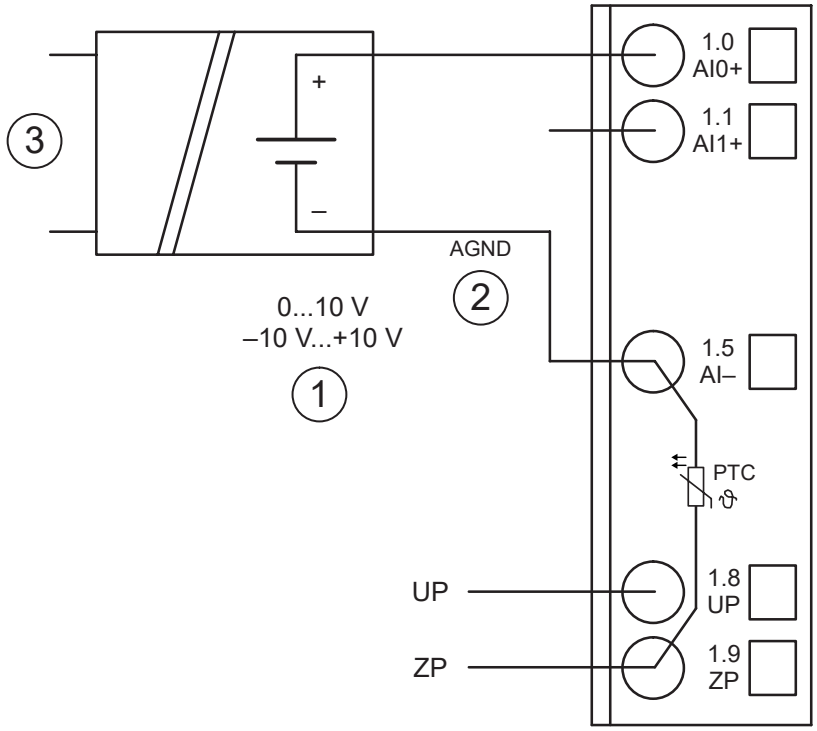


Fig. 976: Connection of active-type analog sensors (voltage) with galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 By connecting to AI-, the galvanically isolated voltage source of the sensor is referred to ZP
3 Galvanically isolated power supply for the analog sensor

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply.

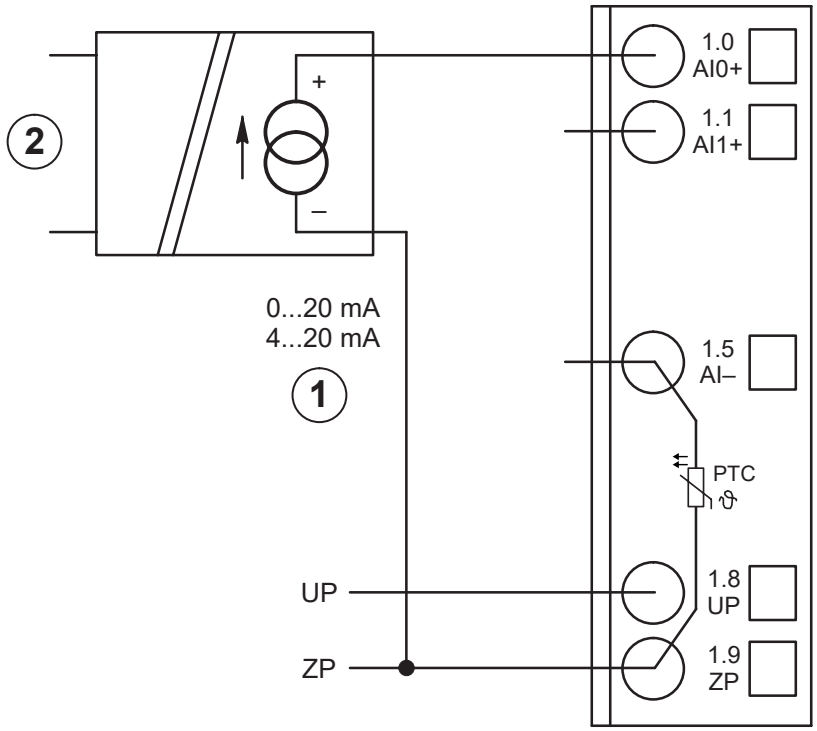


Fig. 977: Connection of active-type analog sensors (current) with galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 Galvanically isolated power supply for the analog sensor

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply

The following figure shows the connection of active-type sensors (voltage) with no galvanically isolated power supply.

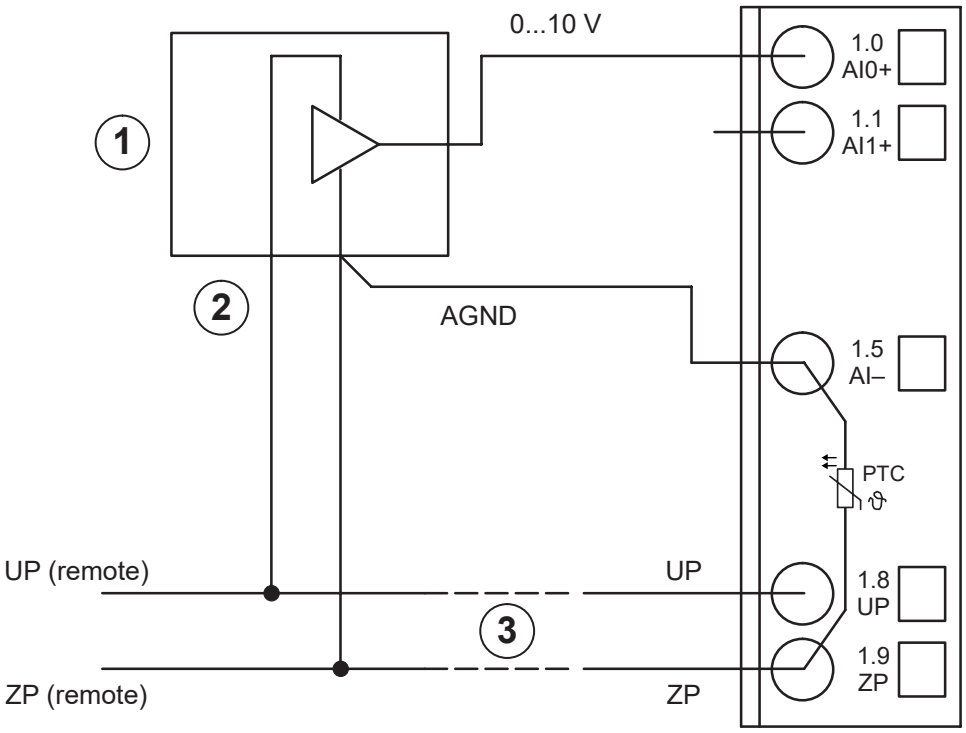


Fig. 978: Connection of active-type sensors (voltage) with no galvanically isolated power supply

1 1 analog sensor requires 1 channel
2 Power supply not galvanically isolated
3 The connection between the negative pole of the sensor and ZP has to be performed
4 Long cable

NOTICE!

Risk of faulty measurements!

The negative pole/ground potential at the sensors must not have too large a potential difference with respect to ZP (max. ± 1 V within the full signal range).

Make sure that the potential difference never exceeds ± 1 V.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V *)	1 channel used

*) if the sensor can provide this signal range

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Connection of passive-type analog sensors (Current)

The following figure shows the connection of passive-type analog sensors (current).

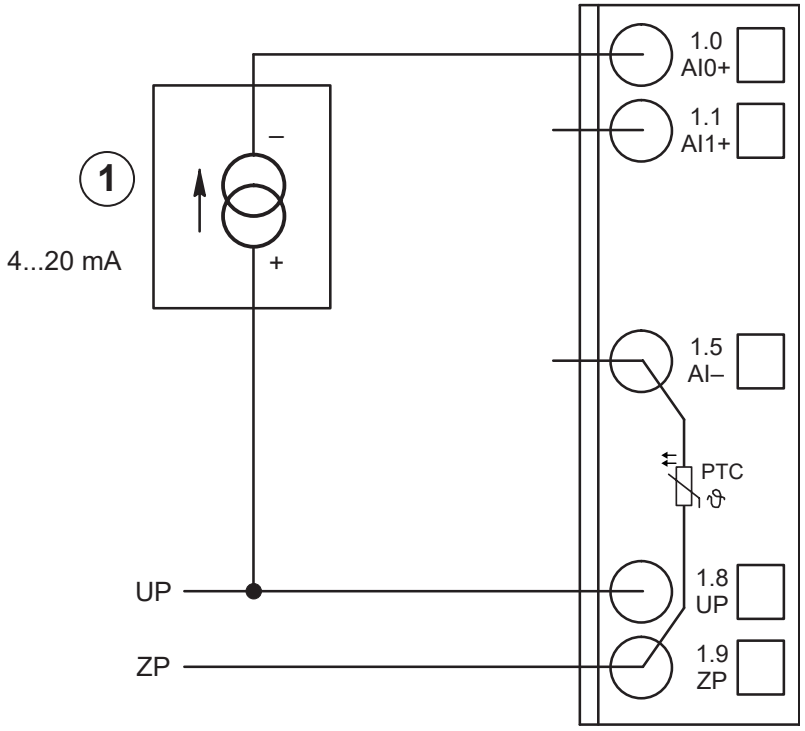


Fig. 979: Connection of passive-type analog sensors (current)

1 1 analog sensor requires 1 channel

Current	4...20 mA	1 channel used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

CAUTION!
If, during initialization, an analog current sensor supplies more than 25 mA for more than 1 second into an analog input, this input is switched off by the module (input protection). In such cases, it is recommended, to protect the analog input by a 10-volt zener diode (in parallel to I+ and I-). But, in general, it is a better solution to prefer sensors with fast initialization or without current peaks higher than 25 mA.

Unused input channels can be left open-circuited, because they are of low resistance.

Connection of active-type analog sensors (Voltage) to differential inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

Important: The ground potential at the sensors must not have a too big potential difference with respect to ZP (max. ±1 V within the full signal range). Otherwise problems can occur concerning the common-mode input voltages of the involved analog inputs

The following figure shows the connection of active-type analog sensors (voltage) to differential inputs.

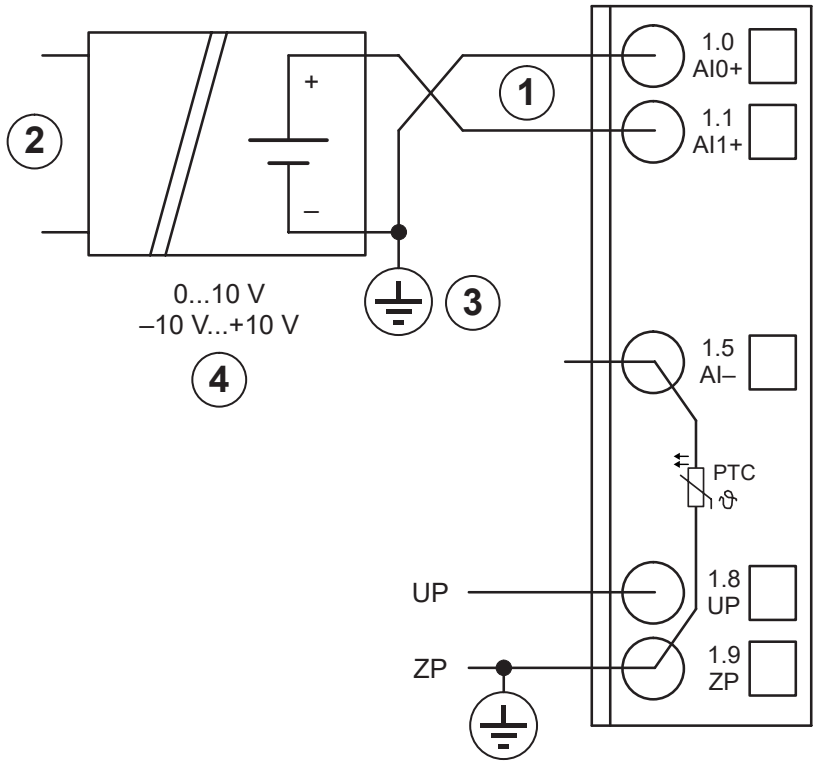


Fig. 980: Connection of active-type analog sensors (voltage) to differential inputs

1 1 analog sensor requires 2 channels
2 Galvanically isolated power supply for the analog sensor
3 Grounding at the sensor
4 0 V...10 V / -10 V...+10 V connected to differential inputs

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

The measuring ranges are described in the section Measuring Ranges *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

In order to avoid error messages or long processing times, it is useful to configure unused analog input channels as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital input. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the use of analog inputs as digital inputs.

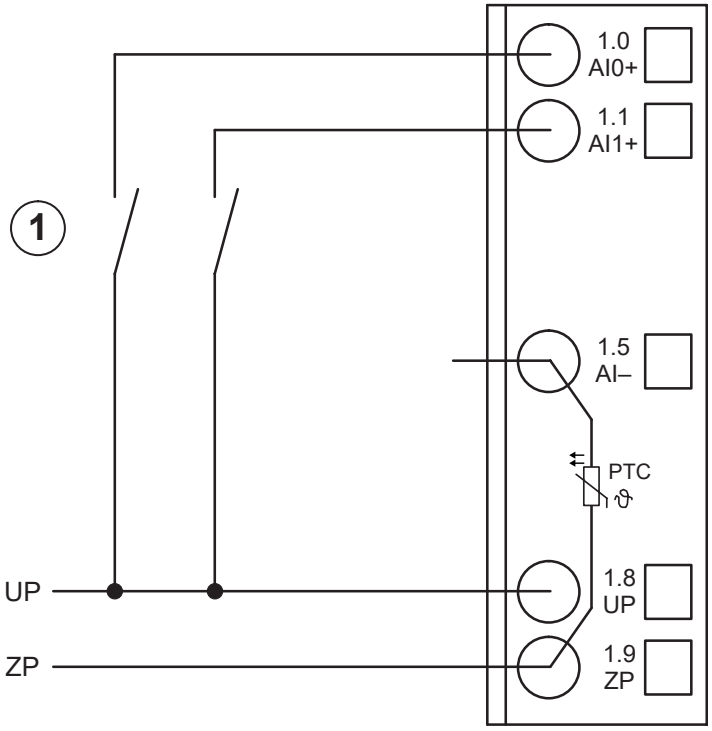


Fig. 981: Use of analog inputs as digital inputs

1 1 digital signal requires 1 channel

Digital input	24 V	1 channel used

The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* ↳ *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

Connection of analog output loads (Voltage, current)

The following figure shows the connection of analog output loads (voltage, current).

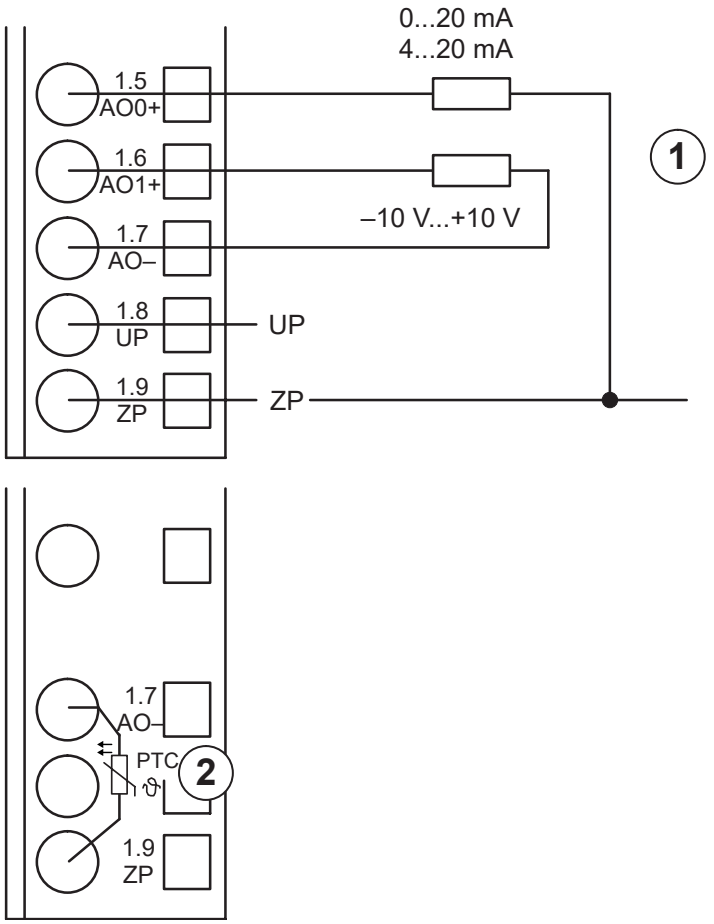


Fig. 982: Connection of analog output loads (voltage, current)

1 1 analog load requires 1 channel

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used
Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

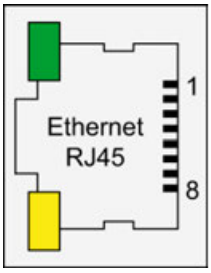
The measuring ranges are described in the section Measuring Ranges ↳ *Chapter 1.6.2.8.4.1.7 “Parameterization” on page 4829* ↳ *Chapter 1.6.2.8.4.1.10 “Measuring ranges” on page 4838*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment. The pin assignment is used for the EtherCAT master (communication module CM5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1
Analog inputs (words)	4
Analog outputs (words)	2

Addressing

The Ethernet bus module CI511-ETHCAT does not consider the position of the rotary switches at the front side of the module. The function of the rotary switches is reserved for future expansions.

I/O configuration

In order to be able to use the CI51X-ETHCAT with device index C0 or above properly, please download the corresponding device description (.xml-)files from http://www.abb.com/plc and install them to the device repository of your Automation Builder. This will allow you to use up to 10 Expandable S500 I/O modules as well as the Extended Cam Switch Library with your CI51X-ETHCAT device.

The CI511-ETHCAT does not store configuration data itself.

The analog I/O channels are configured via software.

Parameterization

Module parameter

Name	Value	Internal value	Internal value, type	Default
Module ID	Internal	48155	WORD	48155
Parameter length	Internal	28	BYTE	28
Error LED / Fail-safe function [1])	On Off by E4 Off by E3 On + failsafe Off by E4 + failsafe Off by E3 + failsafe	0 1 3 16 17 19	BYTE	0
Check Supply	Off On	0 1	BYTE	1

Table 521: Error LED / Failsafe function [1])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafemode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafemode off
Off by E3	Error LED lights up at errors of error classes E1 and E2 auf, Failsafemode off
On + failsafe	Error LED lights up at errors of all error classes, Failsafemode on *)
Off by E4 + failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafemode on *)
Off by E3 + failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafemode on *)

*) The parameters behaviourAOatCommunicationFault and behaviourDOatCommunicationFault are only analyzed if the Failsafe-mode is ON.

Group parameters of the cam switch

Name	Value	Internal value	Internal value, type	Default
numOfUsed-Cams [1])	0 ... 32 128...160	0 ... 32 218...160	WORD	0
resolution [2])	0 ... 2 -1	0 ... 2 -1	DWORD	36000
zeroShift [3])	0 ... 2 -1	0 ... 2 -1	DWORD	0
EncoderBitReso-lution [4])	8 ... 32	8 ... 32	WORD	18
Reserve	-	-	WORD	-

[1]) The parameter numOfUsedCams defines the interrupt cycle time (Therefore, it takes effect to the accuracy of the track) and the behavior of the module if the DC information is lost.

Parameter setting for numOfUsed-Cams	Number of cams used	Interrupt cycle time	Behavior if DC information is lost
0	0	50 µs	Module changes to "safe-operational" state; the outputs are activated trough the user program
1...8	1...8	80 µs	
9...16	9...16	100 µs	
17...32	17...32	200 µs	
128	0	50 µs	Module keeps in "operational" state; the outputs are activated trough the user program
129...136	1...8	80 µs	Module keeps in "operational" state; the cam switch outputs are activated according to an interpolated timing information
137...144	9...16	100 µs	
145...170	17...32	200 µs	

[2]) The parameter resolution defines the angle resolution of the track. The value gives the number of increments related to 360°; e. g. the value 36,000 corresponds to an angle resolution of 0.01°.

[3]) The parameter zeroShift defines the zero shift. With it the encoder can be adjusted to the mounting position. The value of zeroShift is set in encoder-increments. It is not assigned to the parameter resolution of the cam switch.

[4]) The parameter EncoderBitResolution defines the resolution of the used encoder (in bits), e. g. with the default setting 18 bits the encoder has 196,608 divisions.

Channel parameters for the cam switch (max. 32x)

Name	Value	Internal value	Internal value, type	Default
camToTrack0 *)	Digital Output 0 ... 7, none	0 ... 7, FF	BYTE	FF
:	:	:	:	:
camToTrack31	Digital Output 0 .. .7, none	0 ... 7, FF	BYTE	FF

*) The value of the parameter camToTrack# defines which DO (digital output) is assigned to the track. camToTrack0 = 3 for example means that track 0 is assigned to the digital output 3. If the value FFh is set to a track, no digital output is assigned to it.

Name	Value	Referred FB from extended Cam Switch Library [2])	Internal value	Internal value, type	Default
cam-Type[0] [1]) ...	Common	MCX_CamSwitchSimple_c	0	BYTE	0
	Pulsed	MCX_CamSwitchSimple_dc			
	Timed	MCX_PulseSwitch_dc	1		
	Comfort	MCX_CamSwitchTimed_dc	2		
	Cam shift	MCX_CamSwitchCom-fort_dc	3		
	Binary shift		4		
	Multiturn cam	MCX_CamShift_dc	5		
	Time timed	MCX_BinaryShift_dc	6		
	Reference	MCX_CamSwitchMulti_dc	7		
	Multiturn timed	MCX_SwitchTimeTimed_dc	8		
		MCX_BinaryReference_dc	9		
		MCX_CamSwitchMulti-Timed_dc			

[1]) camType additionally to camToTrack identifies the type of each cam switch and enables the use of a specific function block from the Extended Cam Switch Library.

[2]) camType parameters and the Extended Camswitch Library ↳ *Chapter 1.5.4.6 “Extended camswitch library” on page 862* are only available for CI511-ETHCAT and CI512-ETHCAT with device index C0 and above.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard	0	BYTE	0
Behaviour AO at comm. error *)	Off	0	BYTE	0
	Last value	1		
	Last value 5 s	6		
	Last value 10 s	11		
	Substitute value	2		
	Substitute value 5 s	7		
	Substitute value 10 s	12		

*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, channel configuration	see [1])	see [1])	BYTE	0
Input 0, check channel	see [2])	see [2])	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, channel configuration	see [1])	see [1])	BYTE	0
Input 3, channel configuration	see [2])	see [2])	BYTE	0

Channel configuration [1])

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)

Internal value	Operating modes of the analog inputs, individually configurable
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
	*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

Table 522: Channel monitoring [2])

Internal Value	Check channel
0	Plausib(ility), cut wire, short circuit
3	not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, channel configuration	see [3])	see [3])	BYTE	0
Output 0, check channel	see [4])	see [4])	BYTE	0
Output 0, substitute value	see [5])	see [5])	WORD	0
Output 1, channel configuration	see [3])	see [3])	BYTE	0
Output 1, check channel	see [4])	see [4])	BYTE	0
Output 1, substitute value	see [5])	see [5])	WORD	0

Table 523: Channel configuration [3])

Internal value	Operating modes of the analog outputs, individually configurable
0	Not used (default)
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 524: Channel monitoring [4])

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 525: Substitute value [5])

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s	Last value 5 s	0
Last value for 10 s	Last value 10 s	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s	Substitute value 5 s	Depending on configuration
Substitute value for 10 s	Substitute value 10 s	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.01 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.01 ms 0x00
Detect short circuits at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error *)	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute 5 sec Substitute 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x0000

*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])	[4])				
Module error								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	20	Slave-to-Slave mal-function		Check configu-ration
3	-	31	31	31	41	Distributed Clock malfunction		Check configu-ration
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage UP
4	-	31	31	31	45	Process voltage UP3 too low		Check process voltage
4	-	31	31	31	34	No response during initialization of the I/O module		Replace I/O module

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])	[4])			
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])	Check terminals
Channel error digital							
4	-	31	2	0..7	46	Voltage feedback on deactivated digital output [5])	Check terminals
4	-	31	2	0..7	47	Short circuit at digital output	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0..1	48	Analog value overflow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI511-ETHCAT diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself or ADR = Hardware address (e. g. of the DC551)

[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself (Module error) or Module type (1=AI, 2=DO, 3=AO; channel error)
[4])	Diagnosis message appears for the whole output group and not per channel. The message occurs if the output channel is already active.
[5])	Diagnosis message appears per channel. The message occurs if the output channel is not active.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, NET, DC, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 526: States of the 5 system LEDs

LED	Color	Off	On	Flashing	1x Flash	2x Flash
PWR/RUN	Green	Error in the internal supply voltage or process voltage missing	Internal supply voltage OK	Module is not configured	--	--
	Yellow	--	--	--	--	--
NET	Green	Init	Operational	Pre-operational	Safe-operational	--
	Red	No error	PDI Watchdog Timeout	Invalid Configuration	Unsolicited State Change	Application time out
DC *)	Green	Distributed Clock not active	Distributed Clock active	--	--	--
	Red	--	--	--	--	--
S-ERR	Red	No error	Internal error	--	--	--
I/O-Bus	Green	No communication interface modules connected or communication error	---	---	--	--
ETH1	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--

LED	Color	Off	On	Flashing	1x Flash	2x Flash
ETH2	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--

*) The state of this LED is only significant if the cam switch functionality is enabled

Table 527: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Overflow	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
		160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
			800 : 701	0320 : 02BD
Normal range	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0	0000
	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C

Range	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
			Decimal	Hex.
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 ↬ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↬ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Bus connection	2 x RJ45
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)

Parameter	Value
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability (S500 I/O modules)	Up to 10 S500 I/O modules (Index C0 and above), not available (Index below C0)
Indicators	5 LEDs for state indication
Adjusting elements	2 rotary switches (used for future topology extensions)
Quantity of input/output data	CI512-ETHCAT: 10 bytes input and 14 bytes output CI511-ETHCAT: 18 bytes input and 18 bytes output
Limit of data for input and output	144 byte
Acyclic services	SDO (1500 bytes max.) Emergency ECAT_SLV_DIAG ↳ *Chapter 1.5.4.14.1.5 “ECAT_SLV_DIAG” on page 1308*
Protective functions (according to CODESYS)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation to network

Technical data of the module

Parameter		Value
Process supply voltage UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8

Parameter	Value
Number of analog inputs	4
Number of analog outputs	2
Reference potential for all digital inputs and outputs	Negative pole of the supply voltage, signal name ZP
Diagnosis	See Diagnosis and Displays *Chapter 1.6.2.8.4.1.8 “Diagnosis” on page 4835*
Operation and error displays	32 LEDs (totally)
Weight (without terminal unit)	ca. 125 g
Mounting position	Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V

Parameter		Value
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

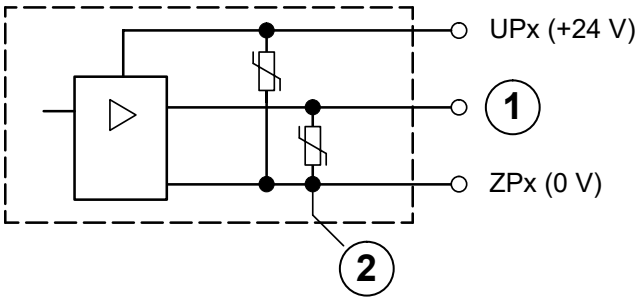


Fig. 983: Digital input/output (circuit diagram)

1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminals 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Galvanic isolation		Against Ethernet network
Configurability		0 V...10 V, -10 V...+10 V, 0/4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current and Digital Input ↳ *Chapter 1.6.2.8.4.1.10.1 “Input ranges voltage, current and digital input” on page 4838* and Input range resistance temperature detector ↳ *Chapter 1.6.2.8.4.1.10.2 “Input ranges resistance temperature detector” on page 4839*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage outputTerminals 1.9, 2.9 and 3.9 (ZP) for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against Ethernet network
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)

Parameter	Value
Output resistance (load), as current output	0 ... 500 Ω
Output loadability, as voltage output	± 10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Table Output Ranges Voltage and Current ↳ *Chapter 1.6.2.8.4.1.10.3 “Output ranges voltage and current” on page 4840*
Unused outputs	Are configured as unused (default value) and can be left open-circuited

Ordering data

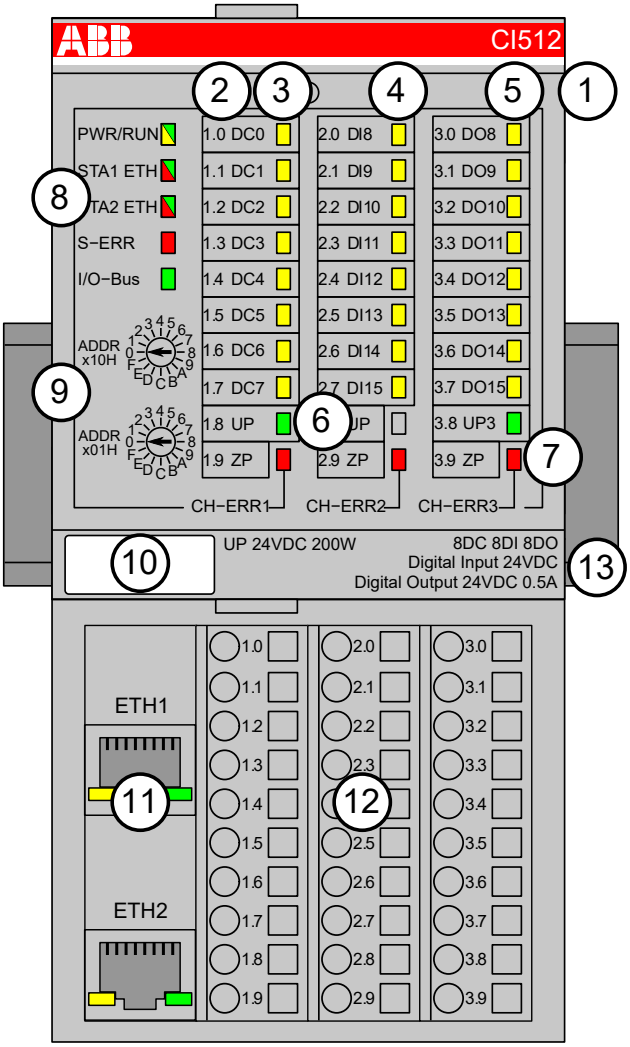
Part no.	Description	Product life cycle phase *)
1SAP 220 900 R0001	CI511-ETHCAT, EtherCAT communication interface module, 8 DI, 8 DO, 4 AI and 2 AO	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI512-ETHCAT

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Cam switch functionality (see also Extended Cam Switch Library)
- Extended Cam switch functionality *)
  (see also Extended Cam Switch Library)
- Module-wise galvanically isolated
- Expandability with up to 10 S500 I/O modules *)

*) Applicable for device index C0 and above.



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 System LEDs: PWR/RUN, NET, DC, S-ERR, I/O-Bus
9 2 rotary switches (reserved for future extensions)
10 Label
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail

Intended purpose

The EtherCAT communication interface module CI512-ETHCAT is used as decentralized I/O module in EtherCAT networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)
- Cam switch functionality

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the configurable digital inputs/outputs is performed by software.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	EtherCAT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	Not used; reserved for future extensions
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet communication interface module CI512-ETHCAT is plugged on the I/O terminal unit TU507-ETH or TU508-ETH. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

The assignment of the other terminals:

Terminals	Signal	Description
1.0 to 1.7	DC0 to DC7	8 digital inputs/outputs (configurable via software)
2.0 to 2.7	DI0 to DI7	8 digital inputs (delay time configurable via software)
3.0 to 3.7	DO0 to DO7	8 digital outputs

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figures show the connection of the Ethernet communication interface module CI512-ETHCAT.

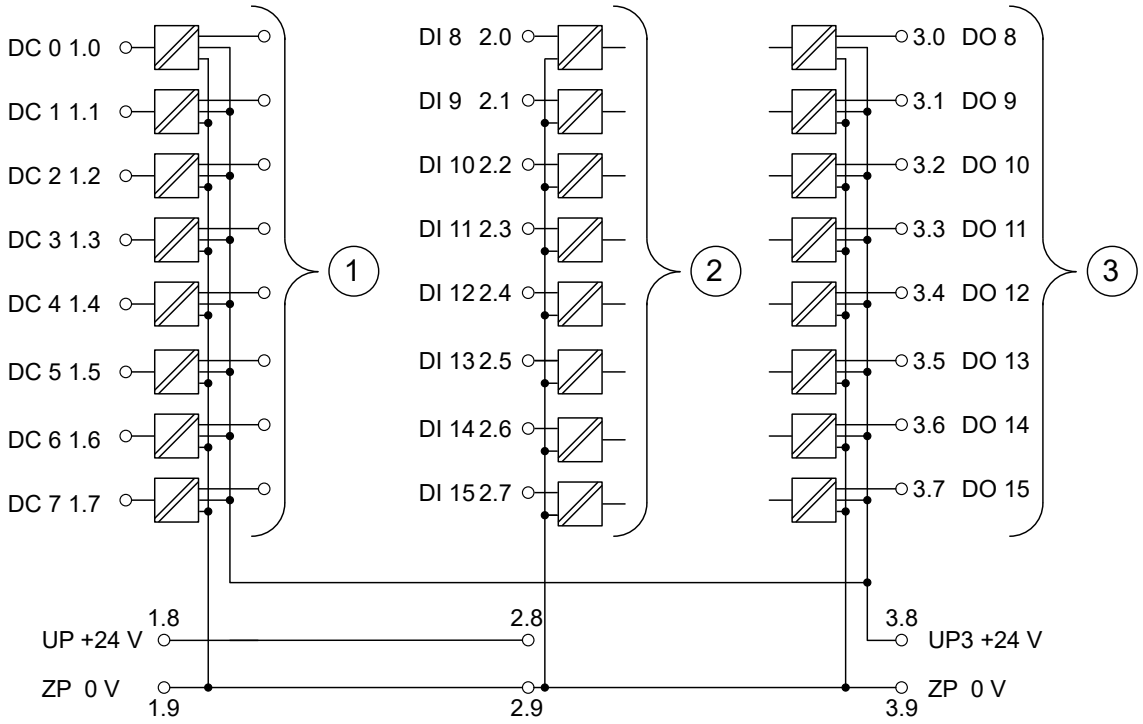


Fig. 984: Connection of the communication interface module CI512-ETHCAT

1 8 digital configurable inputs/outputs 24 V DC
2 8 digital inputs 24 V DC
3 8 digital outputs 24 V DC

In case of voltage feedback, 2 cases are distinguished:

1. The outputs are already active

The output group will be switched off. A diagnosis message will appear. After 5 seconds, the module tries automatic reactivation.

2. The outputs are not active

Only the output with voltage feedback will not be set to active. A diagnosis message will appear.

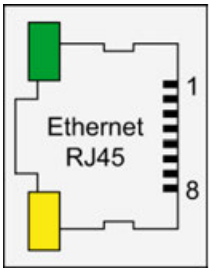
CAUTION!
The process supply voltage must be included within the grounding concept of the plant (e. g. grounding of the negative pole).

The module provides several diagnosis functions *Chapter 1.6.2.8.4.2.9 “Diagnosis” on page 4855*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment. The pin assignment is used for the EtherCAT master (communication module CM5xy-ETHCAT) as well.

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

The EtherCAT network differentiates between input-connectors (IN) and output-connectors (OUT):

At the EtherCAT slaves (communication interface modules), the ETH1-connector is IN and the ETH2-connector is OUT.

At the EtherCAT master (communication module), the ETHCAT1 connector has to be used. The ETHCAT2 connector is reserved for future extensions.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	1
Digital outputs (bytes)	1
Configurable digital inputs/outputs (bytes)	1 + 1

Addressing

The Ethernet communication interface module CI512-ETHCAT does not consider the position of the rotary switches at the front side of the module. The function of the rotary switches is reserved for future expansions.

I/O configuration

In order to be able to use the CI51X-ETHCAT with device index C0 or above properly, please download the corresponding device description (.xml-)files from http://www.abb.com/plc and install them to the device repository of your Automation Builder. This will allow you to use up to 10 Expandable S500 I/O modules as well as the Extended Cam Switch Library with your CI51X-ETHCAT device.

The CI512-ETHCAT does not store configuration data itself.

The analog I/O channels are configured via software.

Parameterization

Module parameter

Name	Value	Internal value	Internal value, type	Default
Module ID	Internal	49435	WORD	49435
Parameter length	Internal	10	BYTE	10
Error LED / Fail-safe function [1])	On Off by E4 Off by E3 On + failsafe Off by E4 + failsafe Off by E3 + failsafe	0 1 3 16 17 19	BYTE	0
Check Supply	Off On	0 1	BYTE	1

Table 528: Error LED / Failsafe function [1])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafe mode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode off
Off by E3	Error LED lights up at errors of error classes E1 and E2 auf, Failsafe mode off
On + failsafe	Error LED lights up at errors of all error classes, Failsafe mode on *)
Off by E4 + failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
Off by E3 + failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe mode on *)

*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.

Group parameters of the cam switch

Name	Value	Internal value	Internal value, type	Default
numOfUsed-Cams [1])	0 ... 32 128...160	0 ... 32 218...160	WORD	0
resolution [2])	0 ... 2 -1	0 ... 2 -1	DWORD	36000
zeroShift [3])	0 ... 2 -1	0 ... 2 -1	DWORD	0
EncoderBitReso-lution [4])	8 ... 32	8 ... 32	WORD	18
Reserve	-	-	WORD	-

Remarks:

[1]) The parameter numOfUsedCams defines the interrupt cycle time (Therefore, it takes effect to the accuracy of the track) and the behavior of the module if the DC information is lost.

Parameter setting for numOfUsed-Cams	Number of cams used	Interrupt cycle time	Behavior if DC information is lost
0	0	50 µs	Module changes to "safe-operational" state; the outputs are activated trough the user program
1...8	1...8	80 µs	
9...16	9...16	100 µs	
17...32	17...32	200 µs	
128	0	50 µs	Module keeps in "operational" state; the outputs are activated trough the user program
129...136	1...8	80 µs	Module keeps in "operational" state; the cam switch outputs are activated according to an interpolated timing information
137...144	9...16	100 µs	
145...170	17...32	200 µs	

[2]) The parameter resolution defines the angle resolution of the track. The value gives the number of increments related to 360°; e. g. the value 36,000 corresponds to an angle resolution of 0.01°.

[3]) The parameter zeroShift defines the zero shift. With it the encoder can be adjusted to the mounting position. The value of zeroShift is set in encoder-increments. It is not assigned to the parameter resolution of the cam switch.

[4]) The parameter EncoderBitResolution defines the resolution of the used encoder (in bits), e. g. with the default setting 18 bits the encoder has 196,608 divisions.

Channel parameters for the cam switch (max. 32x)

Name	Value	Internal value	Internal value, type	Default
camToTrack0 [1])	Digital Output 0 ... 15, none	0 ... 15, FF	BYTE	FF
:	:	:	:	:
camToTrack31	Digital Output 0 ... 15, none	0 ... 15, FF	BYTE	FF

[1]) The value of the parameter camToTrack# defines which DO (digital output) is assigned to the track. camToTrack0 = 3 for example means that track 0 is assigned to the digital output 3. If the value FFh is set to a track, no digital output is assigned to it.

Name	Value	Referred FB from extended Cam Switch Library [2])	Internal value	Internal value, type	Default
cam-Type[0] [1]) ...	Common	MCX_CamSwitchSimple_c	0	BYTE	0
	Pulsed	MCX_CamSwitchSimple_dc			
	Timed	MCX_PulseSwitch_dc	1		
	Comfort	MCX_CamSwitchTimed_dc	2		
	Cam shift	MCX_CamSwitchComfort_dc	3		
	Binary shift	MCX_CamShift_dc	4		
	Multiturn cam	MCX_BinaryShift_dc	5		
	Time timed	MCX_CamSwitchMulti_dc	6		
	Reference	MCX_SwitchTimeTimed_dc	7		
	Multiturn timed	MCX_BinaryReference_dc	8		
		MCX_CamSwitchMulti-Timed_dc	9		

[1]) camType additionally to camToTrack identifies the type of each cam switch and enables the use of a specific function block from the Extended Cam Switch Library.

[2]) camType parameters and the Extended Camswitch Library ↳ *Chapter 1.5.4.6 “Extended camswitch library” on page 862* are only available for CI511-ETHCAT and CI512-ETHCAT with device index C0 and above.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.01 ms	0	BYTE	0.01 ms
	1 ms	1		0x00
	8 ms	2		
	32 ms	3		
Detect short circuit at outputs	Off	0	BYTE	On
	On	1		0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error *)	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute values DO	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
*) The parameter behaviourDOatCommunicationFault is only analyzed if the Failsafe-mode is ON.				

Diagnosis

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000..063**	**AC500-Display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diagnosis block**		
Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module error								
3	-	31	31	31	43	Internal error in the module		Replace I/O module
3	-	31	31	31	20	Slave-to-Slave malfunction		Check configuration
3	-	31	31	31	41	Distributed Clock malfunction		Check configuration
3	-	31	31	31	26	Parameter error		Check master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identifier 000..063**	**AC500-Display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 Bit 6..7	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6 Bit 0..5**	**ETHCAT Diagnosis block**	

Class	**Interface**	**Device**	**Module**	**Channel**	**Error identifier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	46	Voltage feedback on activated digital outputs [4])	Check terminals
Channel error digital							
4	-	31	2	0..15	46	Voltage feedback on deactivated digital output [5])	Check terminals
4	-	31	4	0..7	47	Short circuit at digital output	Check terminals
4	-	31	2	8..15	47	Short circuit at digital output	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module; 14 = I/O bus; 31 = Module itself The identifier is not contained in the CI512-ETHCAT diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself or ADR = Hardware address (e. g. of the DC551)
[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself (Module error) or Module type (1=AI, 2=DO, 3=AO; channel error)
[4])	Diagnosis message appears for the whole output group and not per channel. The message occurs if the output channel is already active.
[5]	Diagnosis message appears per channel. The message occurs if the output channel is not active.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, NET, DC, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 529: States of the 5 system LEDs

LED	Color	Off	On	Flashing	1x flash	2x flash
PWR/RUN	Green	Error in the internal supply voltage or process voltage missing	Internal supply voltage OK	Module is not config-ured	--	--
	Yellow	--	--	--	--	--
NET	Green	Init	Operational	Pre-opera-tional	Safe-opera-tional	--
	Red	No error	PDI Watchdog Timeout	Invalid Con-figuration	Unsolicited State Change	Application time out
DC *)	Green	Distributed Clock not active	Distributed Clock active	--	--	--
	Red	--	--	--	--	--
S-ERR	Red	No error	Internal error	--	--	--
I/O-Bus	Green	No commu-nication interface modules connected or commu-nication error	---	---	--	--
ETH1	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--
ETH2	Green	No EtherCAT connection	Link OK No data transfer	Link OK Data transfer OK	--	--
	Yellow	--	--	--	--	--
*) The state of this LED is only significant if the camswitch functionality is enabled						

Table 530: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 ↳ *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Parameter	Value
Bus connection	2 x RJ45
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability (S500 I/O modules)	Up to 10 S500 I/O modules (Index C0 and above), not available (Index below C0)
Indicators	5 LEDs for state indication
Adjusting elements	2 rotary switches (used for future topology extensions)
Quantity of input/output data	CI512-ETHCAT: 10 bytes input and 14 bytes output CI511-ETHCAT: 18 bytes input and 18 bytes output
Limit of data for input and output	144 byte

Parameter	Value
Acyclic services	SDO (1500 bytes max.) Emergency ECAT_SLV_DIAG *Chapter 1.5.4.14.1.5 “ECAT_SLV_DIAG” on page 1308*
Protective functions (according to CODESYS)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation to network

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Diagnosis		See Diagnosis and Displays *Chapter 1.6.2.8.4.2.9 “Diagnosis” on page 4855*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

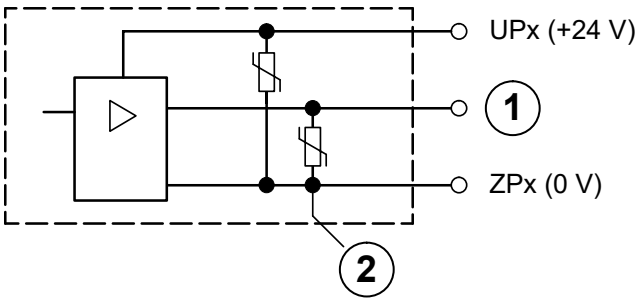


Fig. 985: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Figure:

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC07	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC07	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V *)
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

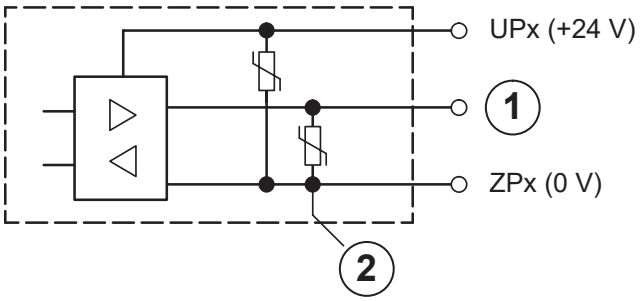


Fig. 986: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Ordering data

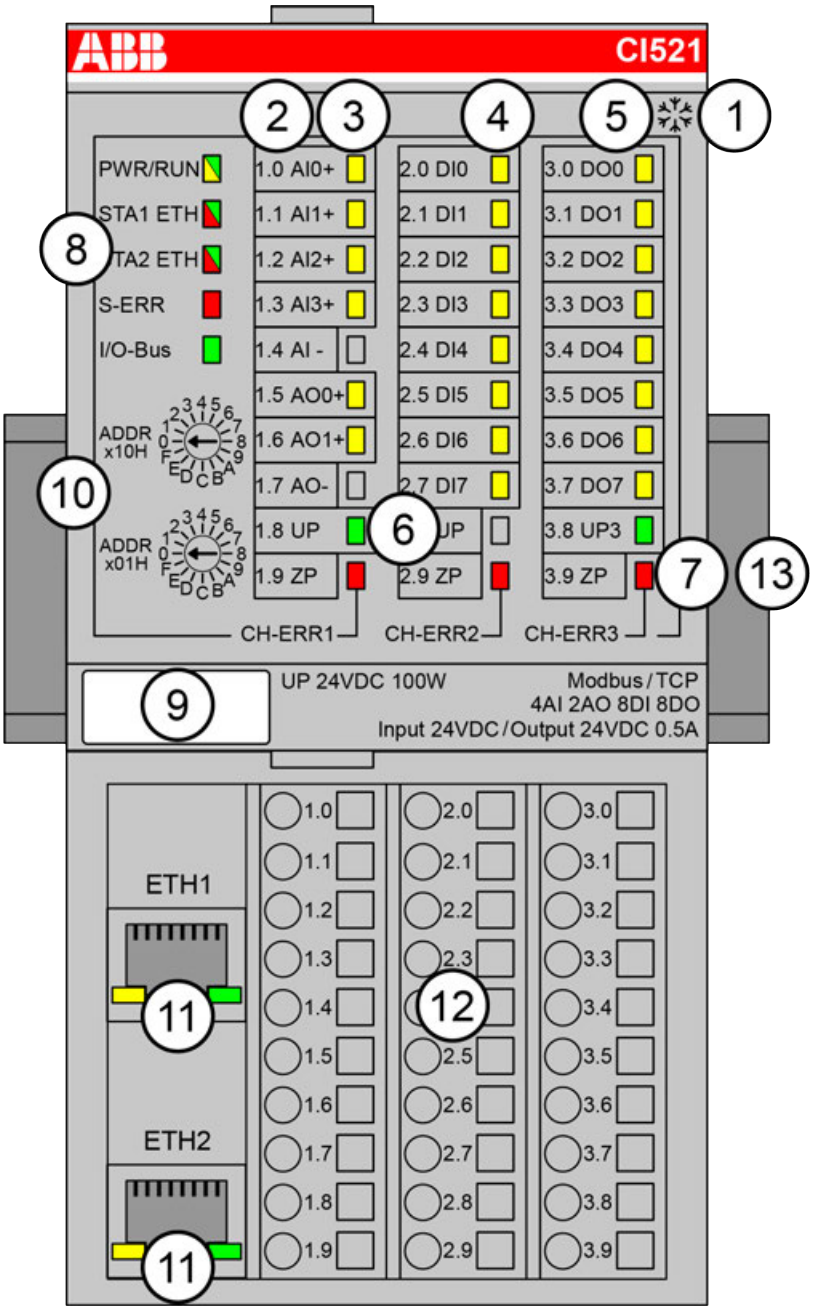
Part no.	Description	Product life cycle phase *)
1SAP 221 000 R0001	CI512-ETHCAT, EtherCAT communication interface module, 8 DI, 8 DO and 8 DC	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8.5 Modbus

CI521-MODTCP

- 4 analog inputs (resolution 12 bits plus sign)
- 2 analog outputs (resolution 12 bits plus sign)
- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the IP address
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The Modbus TCP communication interface module CI521-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	for setting the last BYTE of the IP (00h to FFh)
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Required terminal unit	TU507 or TU508 ↬ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet communication interface module CI521-MODTCP is plugged on the I/O terminal unit TU507-ETH or TU508-ETH ↬ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↬ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Conditions for undisturbed operating with older I/O expansion modules

All I/O expansion modules that are attached to the CI52x-MODTCP must be powered up together with the CI52x-MODTCP if the firmware version of these I/O expansion modules is V1.9 or lower.

The firmware version is related to the index. The index is printed on the module type label on the right side.

Modules as of index listed in the following table can be powered up independently.

S500 I/O module type	First index with firmware version above 1.9
AI523	D0
AI523-XC	D0
AI531	A3
AI531-XC	A0
AO523	D0
AO523-XC	D0
AX521	D0
AX521-XC	D0
AX522	D0
AX522-XC	D0
CD522	A2
CD522-XC	A0
DA501	A2
DA501-XC	A0
DA502	A1
DA502-XC	A1
DC522	D0
DC522-XC	D0
DC523	D0

S500 I/O module type	First index with firmware version above 1.9
DC523-XC	D0
DC532	D0
DC532-XC	D0
DI524	D0
DI524-XC	D0
DO524	A2
DO524-XC	A2
DX522	D0
DX522-XC	D0
DX531	D0
AC522	D0
PD501	D0

Do not connect any voltages externally to digital outputs!

Reason: Externally voltages at an output or several outputs may cause that other outputs are supplied through that voltage instead of voltage UP3 (reverse voltage). This ist not intended usage.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0..DO7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	AI0+	Positive pole of analog input signal 0
1.1	AI1+	Positive pole of analog input signal 1
1.2	AI2+	Positive pole of analog input signal 2
1.3	AI3+	Positive pole of analog input signal 3
1.4	AI-	Negative pole of analog input signals 0 to 3
1.5	AO0+	Positive pole of analog output signal 0
1.6	AO1+	Positive pole of analog output signal 1
1.7	AI-	Negative pole of analog output signals 0 and 1
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI0	Signal of the digital input DI0
2.1	DI1	Signal of the digital input DI1
2.2	DI2	Signal of the digital input DI2
2.3	DI3	Signal of the digital input DI3
2.4	DI4	Signal of the digital input DI4

Terminal	Signal	Description
2.5	DI5	Signal of the digital input DI5
2.6	DI6	Signal of the digital input DI6
2.7	DI7	Signal of the digital input DI7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO0	Signal of the digital output DO0
3.1	DO1	Signal of the digital output DO1
3.2	DO2	Signal of the digital output DO2
3.3	DO3	Signal of the digital output DO3
3.4	DO4	Signal of the digital output DO4
3.5	DO5	Signal of the digital output DO5
3.6	DO6	Signal of the digital output DO6
3.7	DO7	Signal of the digital output DO7
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figures show the connection of the Ethernet communication interface module CI521-MODTCP.

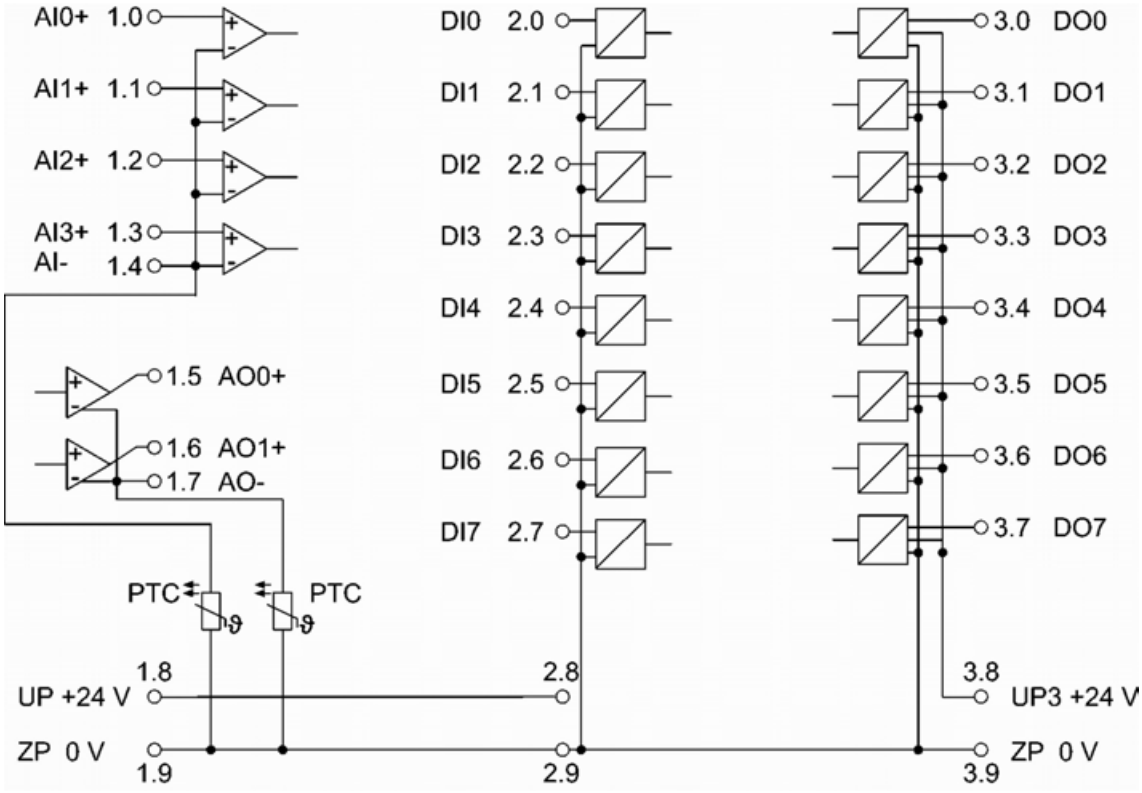


Fig. 987: Connection of the communication interface module CI521-MODTCP

Further information is provided in the System Technology chapter *Chapter 1.6.4.3.1 “Modbus communication interface module” on page 5651*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.

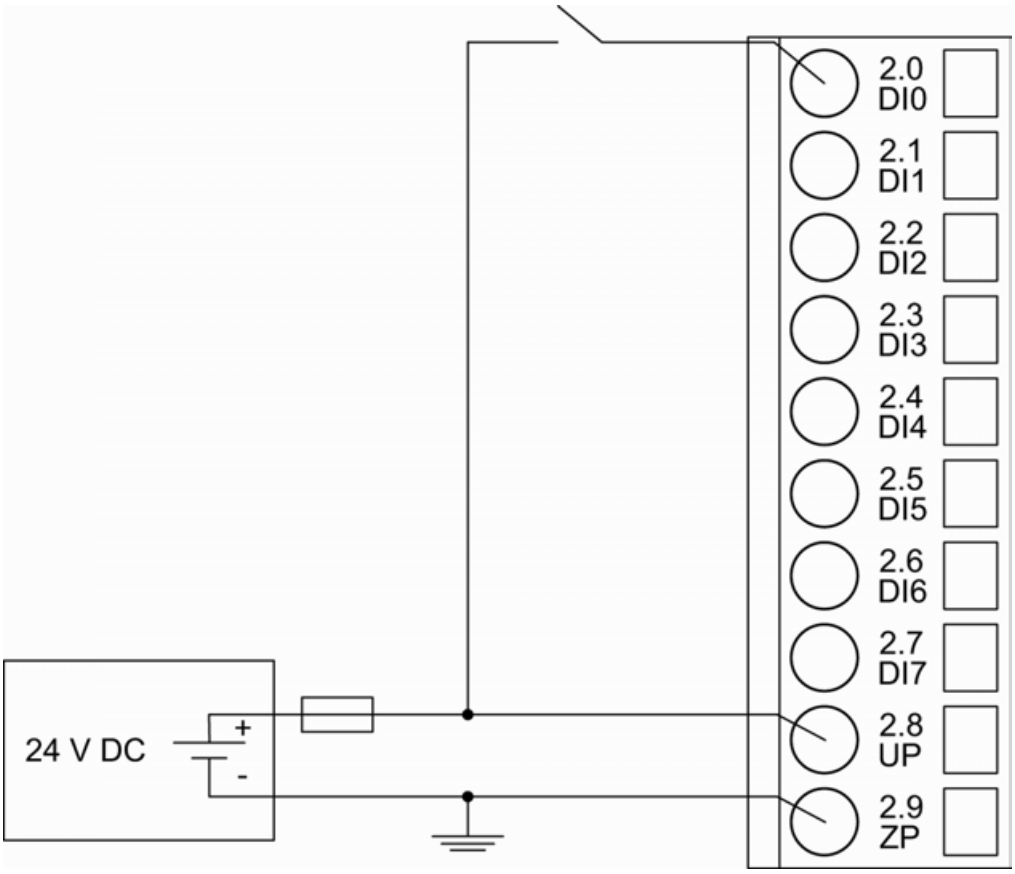


Fig. 988: Connection of the digital inputs to the module CI521-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.5.1.8.2 “State LEDs” on page 4895*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

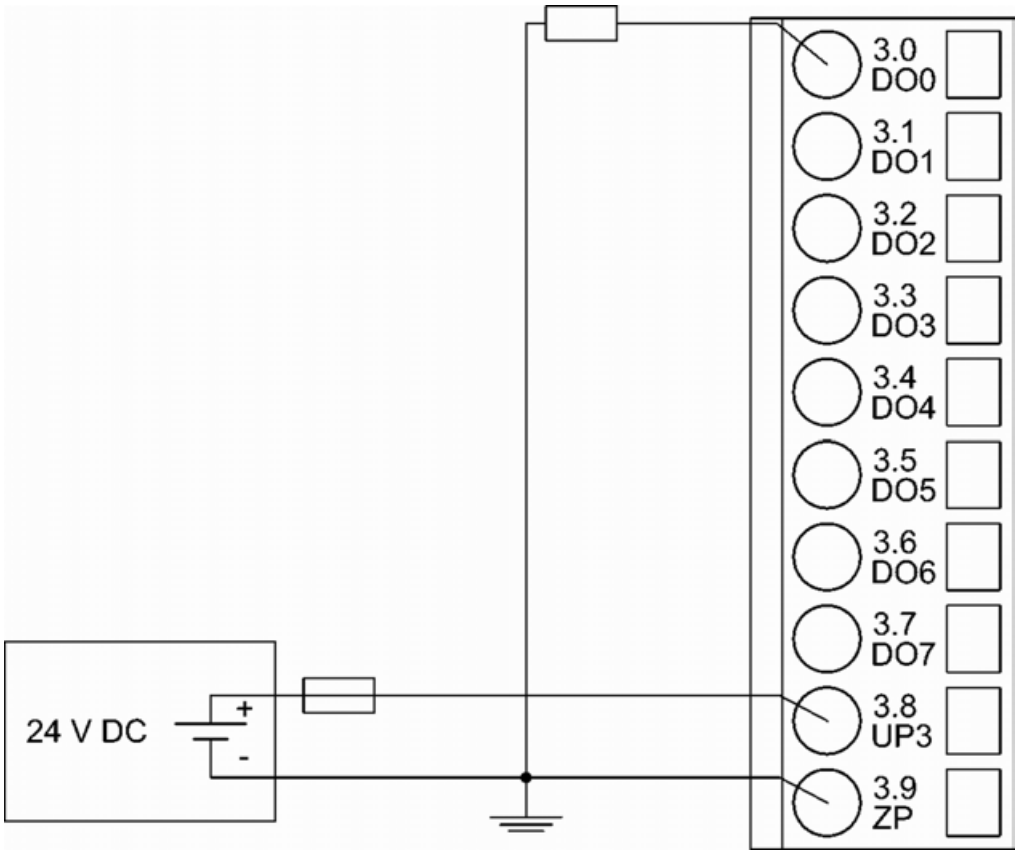


Fig. 989: Connection of configurable digital inputs/outputs to the module CI521-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.5.1.8.2 “State LEDs” on page 4895*.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI521-MODTCP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

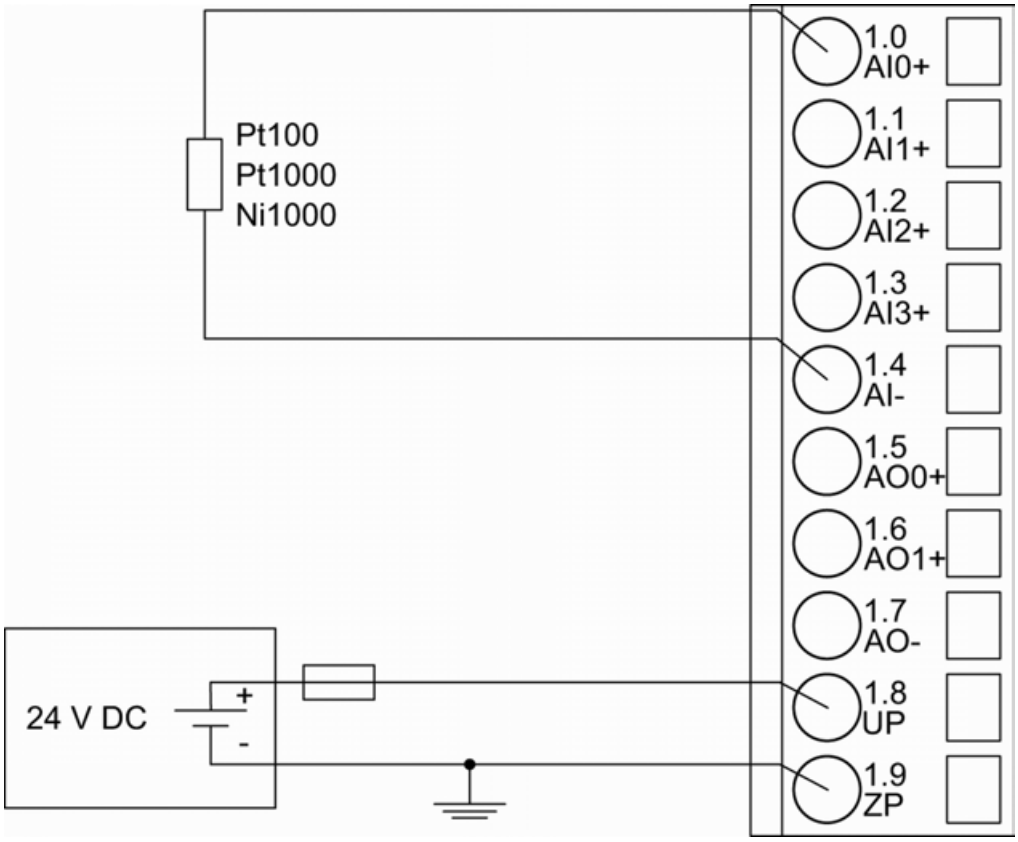


Fig. 990: Connection of resistance thermometers in 2-wire configuration to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Pt100	-50 °C...+70 °C	2-wire configuration, 1 channel used
Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

The module CI521-MODTCP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI521-MODTCP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.

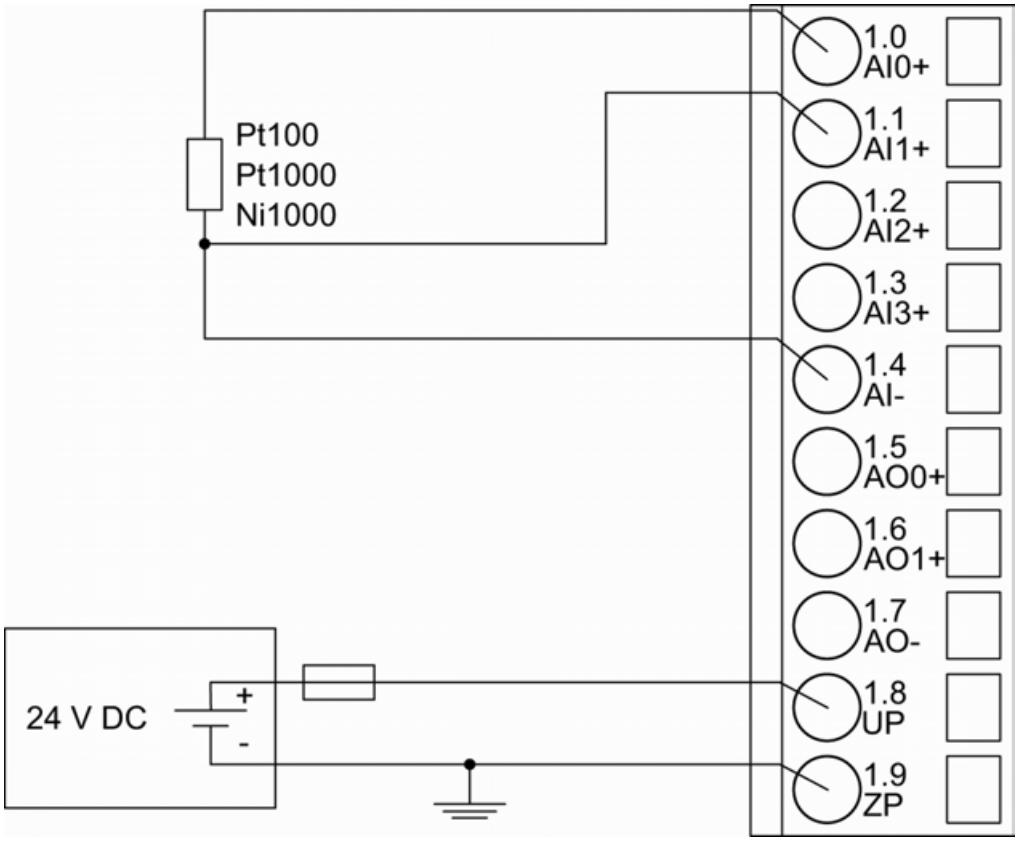


Fig. 991: Connection of resistance thermometers in 3-wire configuration to the analog inputs

With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Pt100	-50 °C...+70 °C	3-wire configuration, 2 channels used
Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

The module CI521-MODTCP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

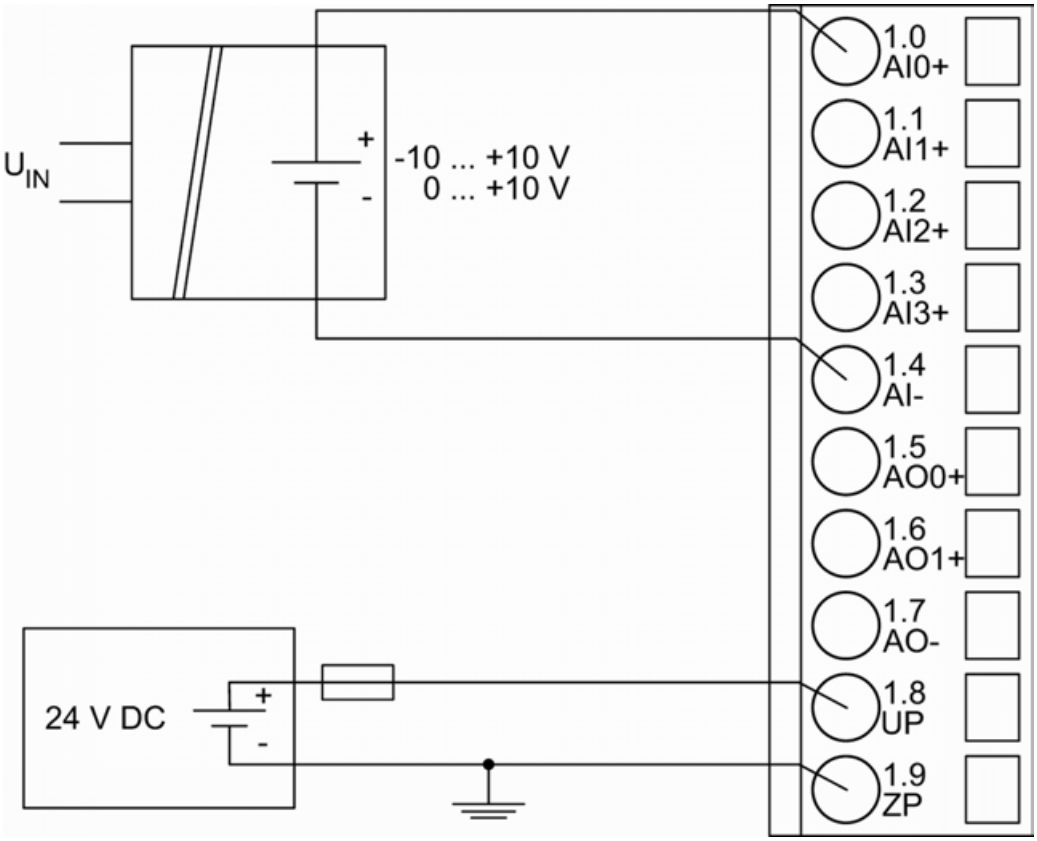


Fig. 992: Connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

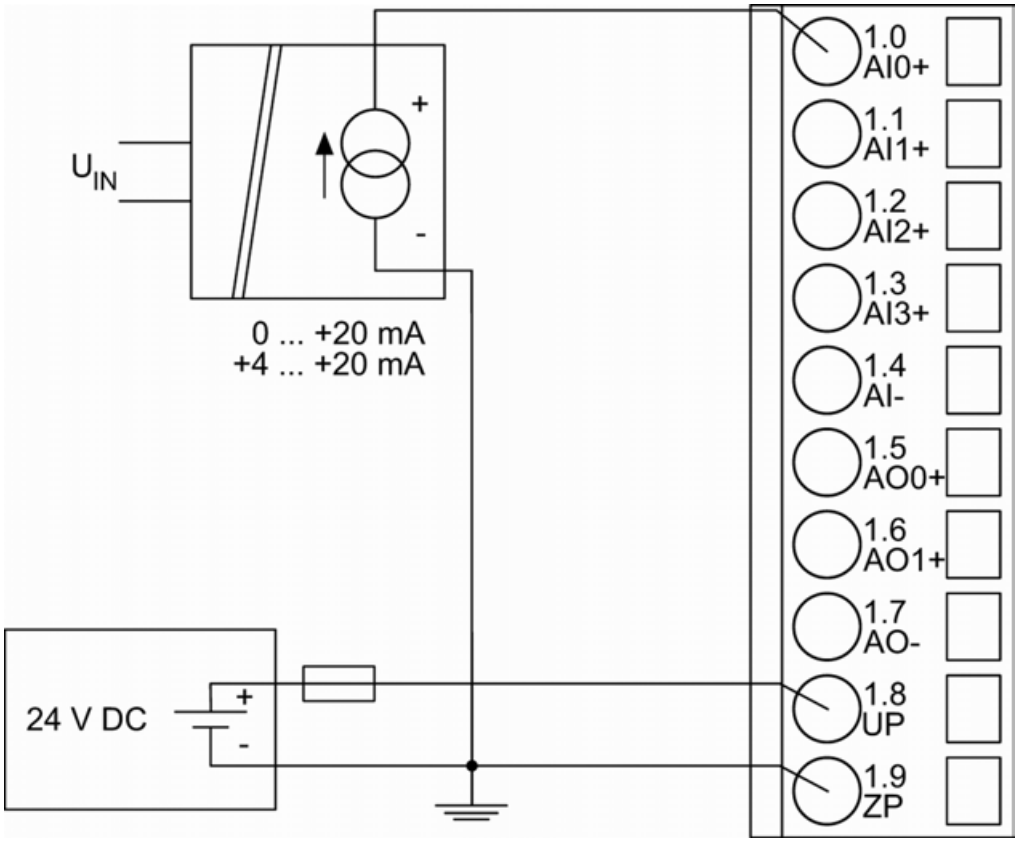


Fig. 993: Connection of active-type analog sensors (current) with galvanically isolated power supply to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Current	0...20 mA	1 channel used
Current	4...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

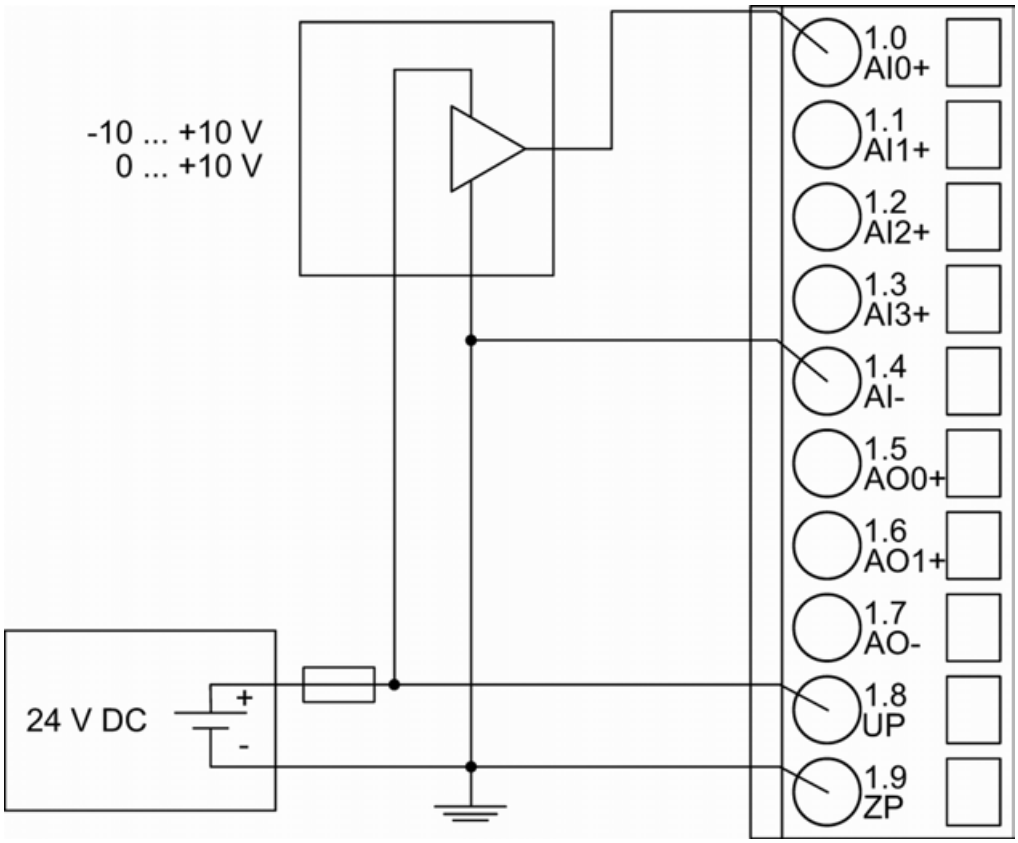


Fig. 994: Connection of active-type sensors (voltage) with no galvanically isolated power supply to the analog inputs

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ± 1 V).

Make sure that the potential difference never exceeds ± 1 V (also not with long cable lengths).

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*.

Voltage	0...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

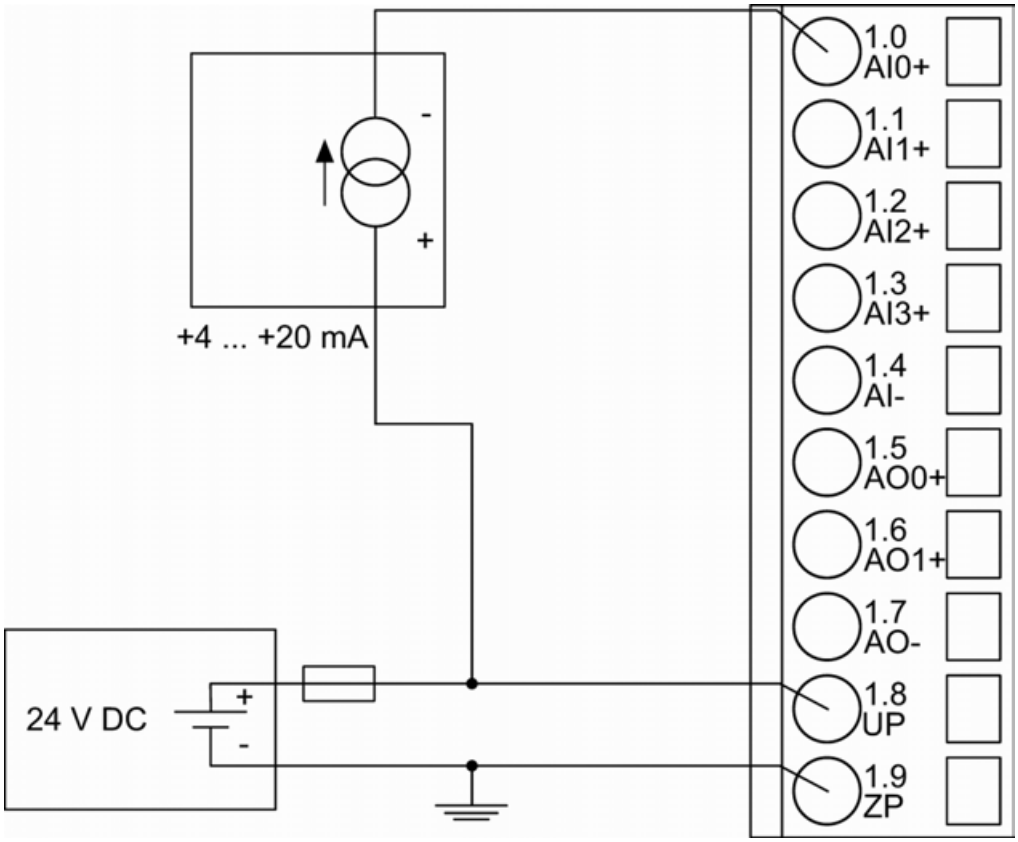


Fig. 995: Connection of passive-type analog sensors (current) to the analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Current	4...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to AIx+ and ZP.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.

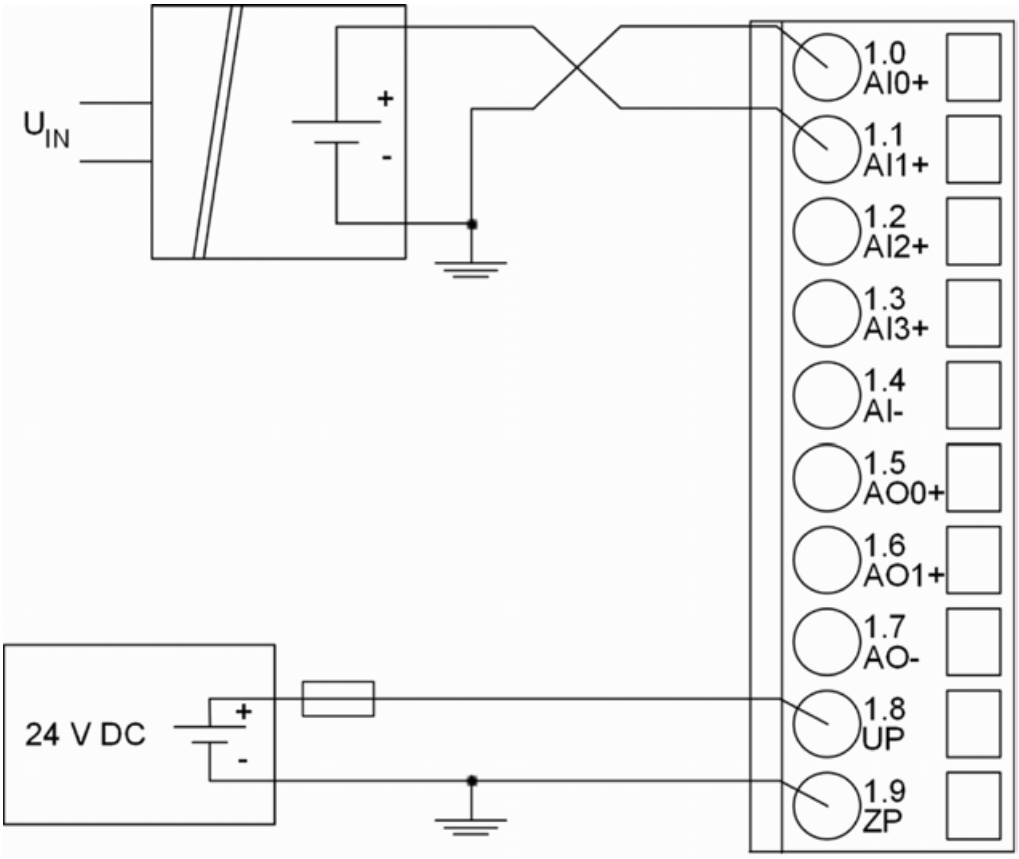


Fig. 996: Connection of active-type analog sensors (voltage) to differential analog inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Voltage	0...10 V	With differential inputs, 2 channels used
Voltage	-10 V...+10 V	With differential inputs, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs ↳ *Chapter 1.6.2.8.5.1.10.5 “Technical data of the analog inputs if used as digital inputs” on page 4902*. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

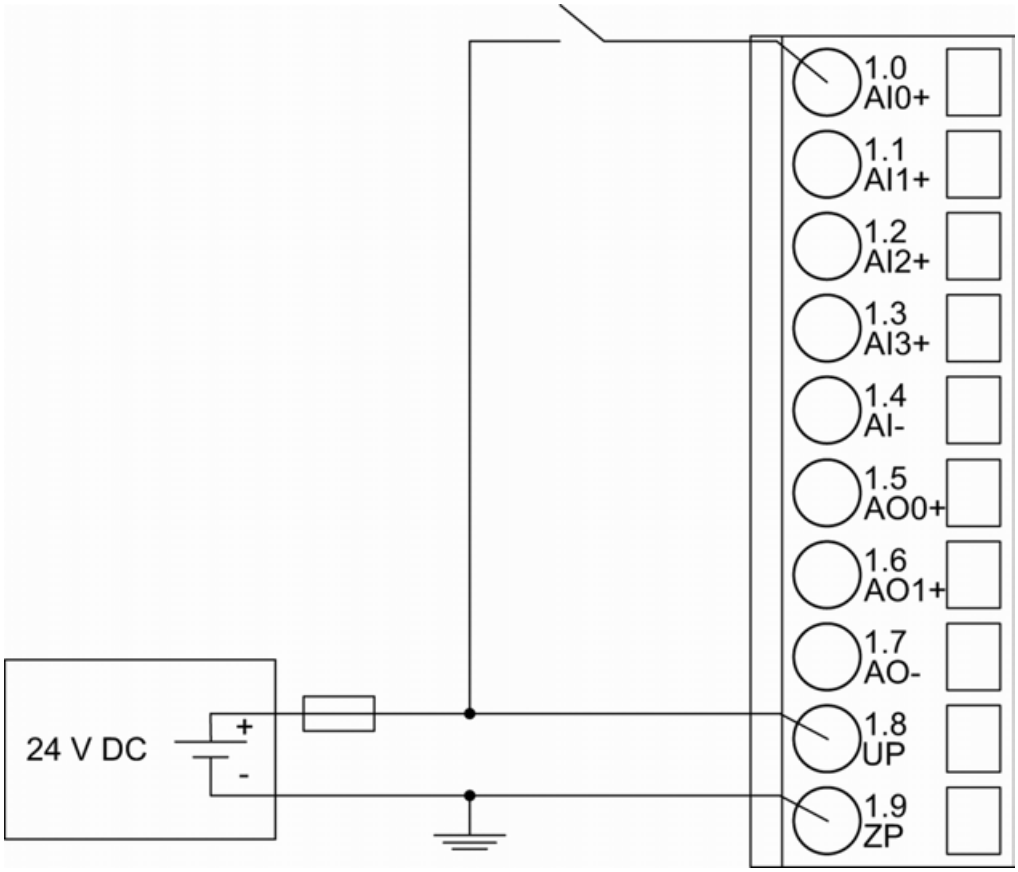


Fig. 997: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896* :

Digital input	24 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

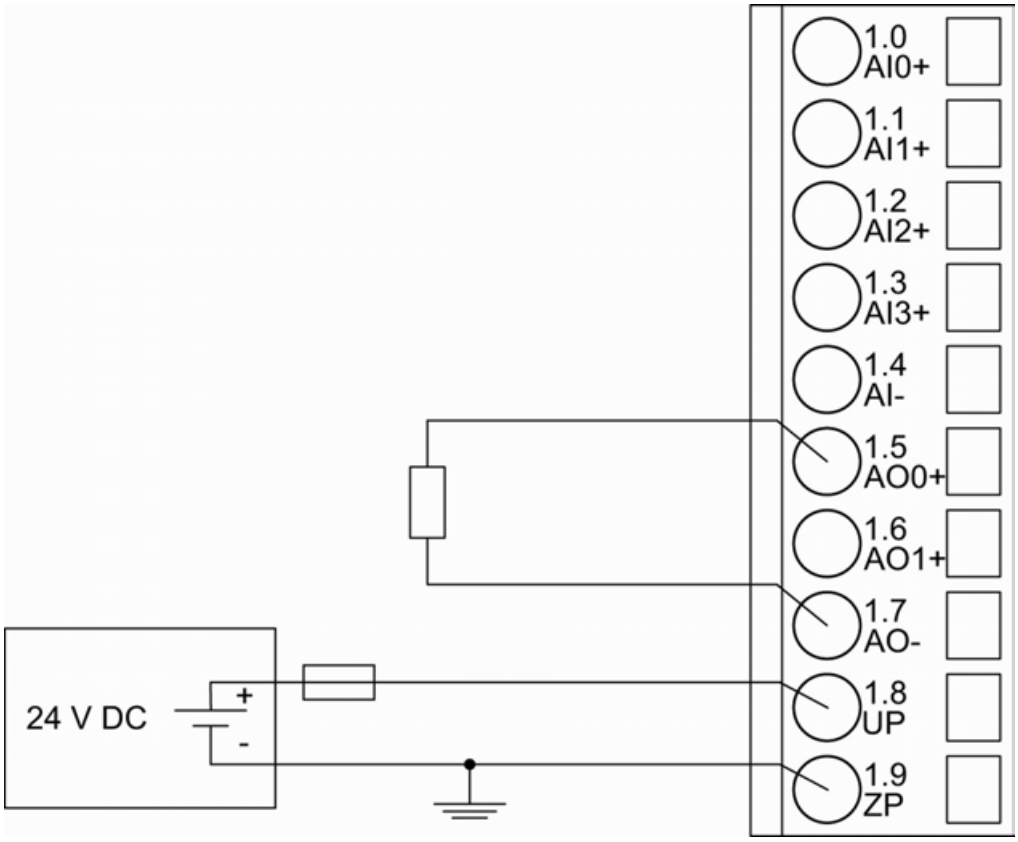


Fig. 998: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

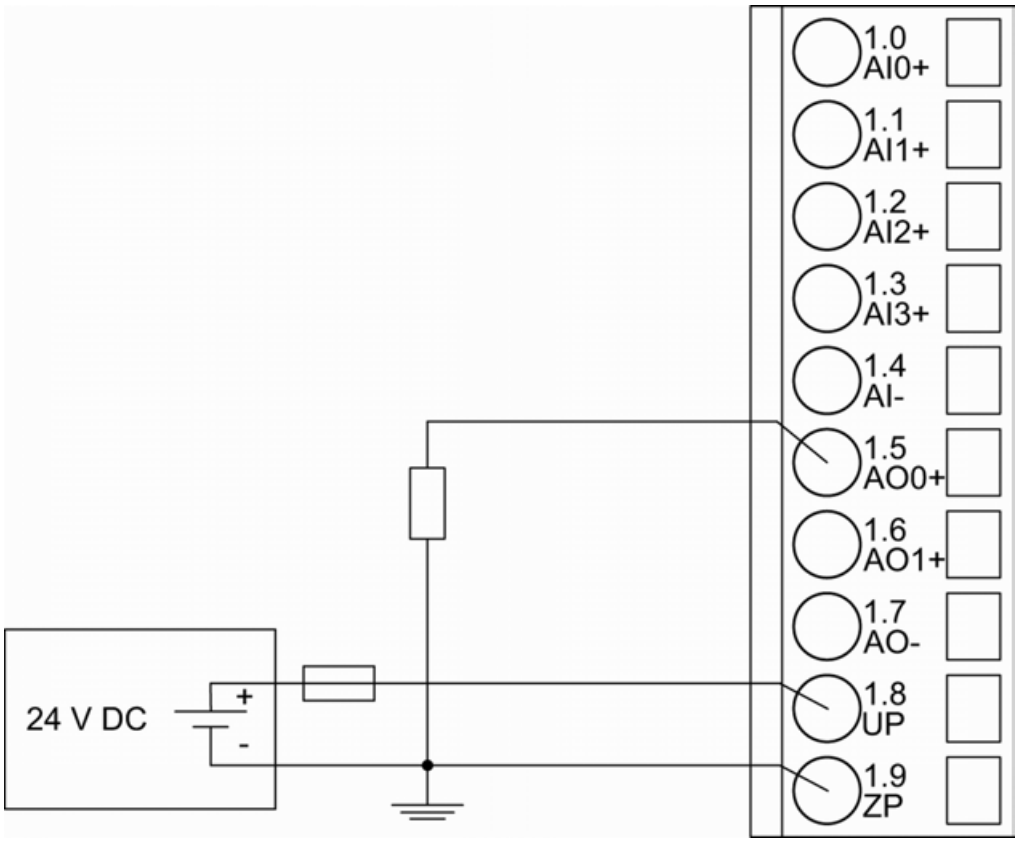


Fig. 999: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884* and ↳ *Chapter 1.6.2.8.5.1.9 “Measuring ranges” on page 4896*:

Current	0...20 mA	Load 0...500 Ω	1 channel used
Current	4...20 mA	Load 0...500 Ω	1 channel used

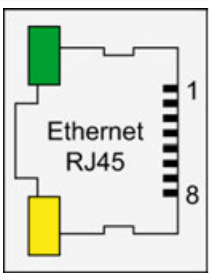
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected

Interface	PIN	Signal	Description
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

The IP address of the CI521-MODTCP Module can be set with the "ABB IP Configuration Tool". *Chapter 1.6.5.2.2.2.2 “Configuration of the IP settings with the IP configuration tool” on page 5816*

If the last byte of the IP is set to 0, the address switch will be used instead.

Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings. This is a backup so the module can always get a valid IP address and can be configured by the “ABB IP Configuration Tool”.

Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

The factory setting for the IP is 192.168.0.x (last byte is address switch).

I/O configuration

The CI521-MODTCP stores configuration parameters (IP address configuration, module parameters).

The analog/digital I/O channels are configured via software.

Details about configuration are described in Parameterization ↳ *Chapter 1.6.2.8.5.1.7 “Parameterization” on page 4884*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7400	WORD	7000
Ignore Module	Internal	0	BYTE	0
Parameter length	Internal	63	BYTE	63
Error LED / Failsafe function see table Error LED / Failsafe function ↳ *Table 531 “Error LED / Failsafe function” on page 4885*	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0

Name	Value	Internal value	Internal value, type	Default
Timeout for Bus supervision	No supervision 10 ms timeout 20 ms timeout	0 1 2	BYTE	No supervision
IO Mapping Structure [3])	Fixed Mapping Dynamic Mapping	0 1	BYTE	0
Reserved	Internal	0	ARRAY[0..2] OF BYTE	0,0,0
Check supply	off on	0 1	BYTE	1
Fast counter	0 : 10 [3])	0 : 10	BYTE	0

[1]) With a faulty ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission.

[2]) Counter operating modes, see description of the *Chapter 1.6.4.1.10 “Fast counters” on page 5498*.

[3]) Fixed Mapping means each module has its own Modbus registers for data transfer independent of the IO bus constellation. For details see *Chapter 1.6.4.3.1.2 “Modbus TCP registers” on page 5652*.

Dynamic mapping means the structure of the IO Date is dependent on the I/O bus constellation. Each I/O bus expansion module starts directly after the module before on the next Word adress.

[4]) If none of the parameters is set all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules.

If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device, all other masters / clients still have read access to the CI52x-MODTCP device.

Table 531: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameters Behaviour AO at comm. error and Behaviour DO at comm. error are only analyzed if the Failsafe-mode is ON.	

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behaviour AO at comm. error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0
*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 532 “Channel configuration” on page 4887*	Table Operating modes of the analog inputs ↳ *Table 532 “Channel configuration” on page 4887*	BYTE	0
Input 0, Check channel	Table Channel montoring ↳ *Table 533 “Channel monitoring” on page 4887*	Table Channel montoring ↳ *Table 533 “Channel monitoring” on page 4887*	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	Table Operating modes of the analog inputs ↳ *Table 532 “Channel configuration” on page 4887*	Table Operating modes of the analog inputs ↳ *Table 532 “Channel configuration” on page 4887*	BYTE	0
Input 3, Check channel	Table Channel montoring ↳ *Table 533 “Channel monitoring” on page 4887*	Table Channel montoring ↳ *Table 533 “Channel monitoring” on page 4887*	BYTE	0

Table 532: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0...20 mA
4	4...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50...+400 °C
9	3-wire Pt100 -50...+400 °C *)
10	0...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50...+70 °C
15	3-wire Pt100 -50...+70 °C *)
16	2-wire Pt1000 -50...+400 °C
17	3-wire Pt1000 -50...+400 °C *)
18	2-wire Ni1000 -50...+150 °C
19	3-wire Ni1000 -50...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 533: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Table Operating modes of the analog outputs *Table 534 “Channel configuration” on page 4888*	Table Operating modes of the analog outputs *Table 534 “Channel configuration” on page 4888*	BYTE	0
Output 0, Check channel	Table Channel monitoring *Table 535 “Channel monitoring” on page 4888*	Table Channel monitoring *Table 535 “Channel monitoring” on page 4888*	BYTE	0

Name	Value	Internal value	Internal value, type	Default
Output 0, Substitute value	Table Substitute value ↳ *Table 536 “Substitute value” on page 4888*	Table Substitute value ↳ *Table 536 “Substitute value” on page 4888*	WORD	0
Output 1, Channel configuration	Table Operating modes of the analog outputs ↳ *Table 534 “Channel configuration” on page 4888*	Table Operating modes of the analog outputs ↳ *Table 534 “Channel configuration” on page 4888*	BYTE	0
Output 1, Check channel	Table Channel monitoring ↳ *Table 535 “Channel monitoring” on page 4888*	Table Channel monitoring ↳ *Table 535 “Channel monitoring” on page 4888*	BYTE	0
Output 1, Substitute value	Table Substitute value ↳ *Table 536 “Substitute value” on page 4888*	Table Substitute value ↳ *Table 536 “Substitute value” on page 4888*	WORD	0

Table 534: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0...20 mA
130	4...20 mA

Table 535: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 536: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 255	00h ... FFh	BYTE	0 0x0000
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	On 0x01

[1]) The parameters Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.

[2]) The state "externally voltage detected" appears, if the output of a channel DC0..DC7 should be switched on while an externally voltage is connected ↳ *Chapter 1.6.2.8.5.1.3 “Connections” on page 4866*. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".

Diagnosis and state LEDs

Structure of the diagnosis block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI521-MODTCP (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O Module ... 10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

For diagnosis firmware version ≥ 3.2.6 is required.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check Master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	No process voltage UP		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket		Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug projected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])	Restart, if error persists replace terminal unit
4	-	31	31	31	46	Voltage feedback on activated digital outputs DO0...DO7 on UP3 [4])	Check terminals

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	No process voltage UP3	Check process supply voltage
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check termi-nals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Externally voltage detected at digital output DO0...DO7 [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at dig-ital output [7])	Check terminals
Channel error analog							
4	-	31	1	0..3	48	Analog value over-flow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0..3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0..3	47	Short circuit at an analog input	Check terminals

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	31	3	0..1	4	Analog value over-flow at an analog output	Check output value
4	-	31	3	0..1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI521-MODTCP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself; 1..10 = Expansion module
[3])	With "Module" the following allocation applies: 31 = Module itself Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DO0...DO7 cause that other digital outputs are supplied through that voltage ↳ *Chapter 1.6.2.8.5.1.3 “Connections” on page 4866*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0...DO7 has overrun the process supply voltage UP3 ↳ *Chapter 1.6.2.8.5.1.3 “Connections” on page 4866*. Diag-nosis message appears for the whole module.
[6])	This message appears, if the output of a channel DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100ms. Then a new start up will be executed. This diagnosis message appears per channel.

[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 537: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with IO Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	Device configured, acyclic data exchange running
	Red	---	Communication error (timeout) appeared	IP address error
STA2 ETH (System LED "SF")	Green	Device has valid parameters	Device is running parameterization sequenze	Device has no parameters
	Red	---	---	Device has invalid parameters
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 538: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
Normal range or measured value too low	-0.0004 -1.7593	-0.0004 : : : -10,0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Underflow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high		450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
	80.0 °C : 70.1 °C			800 : 701	0320 : 02BD
Normal range	70.0 °C : 0.1 °C	400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0.0 °C	0.0 °C	0.0 °C	0	0000
Normal range	-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	-0.1 °C : -50,0 °C	-1 : -500	FFFF : FE0C
Measured value too low	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-501 : -600	FE0B : FDA8
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	0 V	0 mA	0 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0,0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	0 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output

Parameter		Value
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Ethernet		10/100 base-TX, internal switch, 2 x RJ45 socket
Setting of the IP address		With ABB IP config tool and 2 rotary switches at the front side of the module
Diagnose		See Diagnosis and Displays *Chapter 1.6.2.8.5.1.8 “Diagnosis and state LEDs” on page 4890*
Operation and error displays		32 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 2.0 to 2.7
Reference potential for all inputs	Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals	1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)	Type 1
Input delay (0->1 or 1->0)	Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage	24 V DC

Parameter		Value
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload

Parameter		Value
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

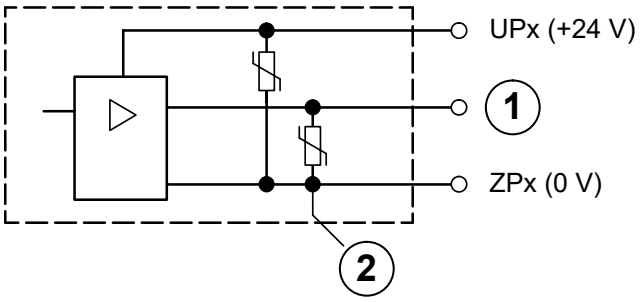


Fig. 1000: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 ... 10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 ... +10 V
Galvanic isolation		Against Ethernet network
Configurability		0...10 V, -10...+10 V, 0/4...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s

Parameter	Value
Resolution	Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Tables Input ranges voltage, current and digital input ↳ *Chapter 1.6.2.8.5.1.9.1 “Input ranges voltage, current and digital input” on page 4896* Input range resistance temperature detector ↳ *Chapter 1.6.2.8.5.1.9.2 “Input ranges resistance temperature detector” on page 4897*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter	Value
Number of channels per module	2
Distribution of channels into groups	1 group for 2 channels
Connection of the channels AO0+...AO1+	Terminals 1.5...1.6

Parameter		Value
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage outputTerminal 1.9, 2.9 and 3.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10...+10 V, 0...20 mA, 4...20 mA (each output can be configured individually)
Output resistance (load), as current output		0...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		Table Output ranges voltage and current ↳ *Chapter 1.6.2.8.5.1.9.3 “Output ranges voltage and current” on page 4898*
Unused outputs		Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI0), 2.1 (DI1)
Used outputs	Terminal 3.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	See ↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*
Operating modes	See ↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

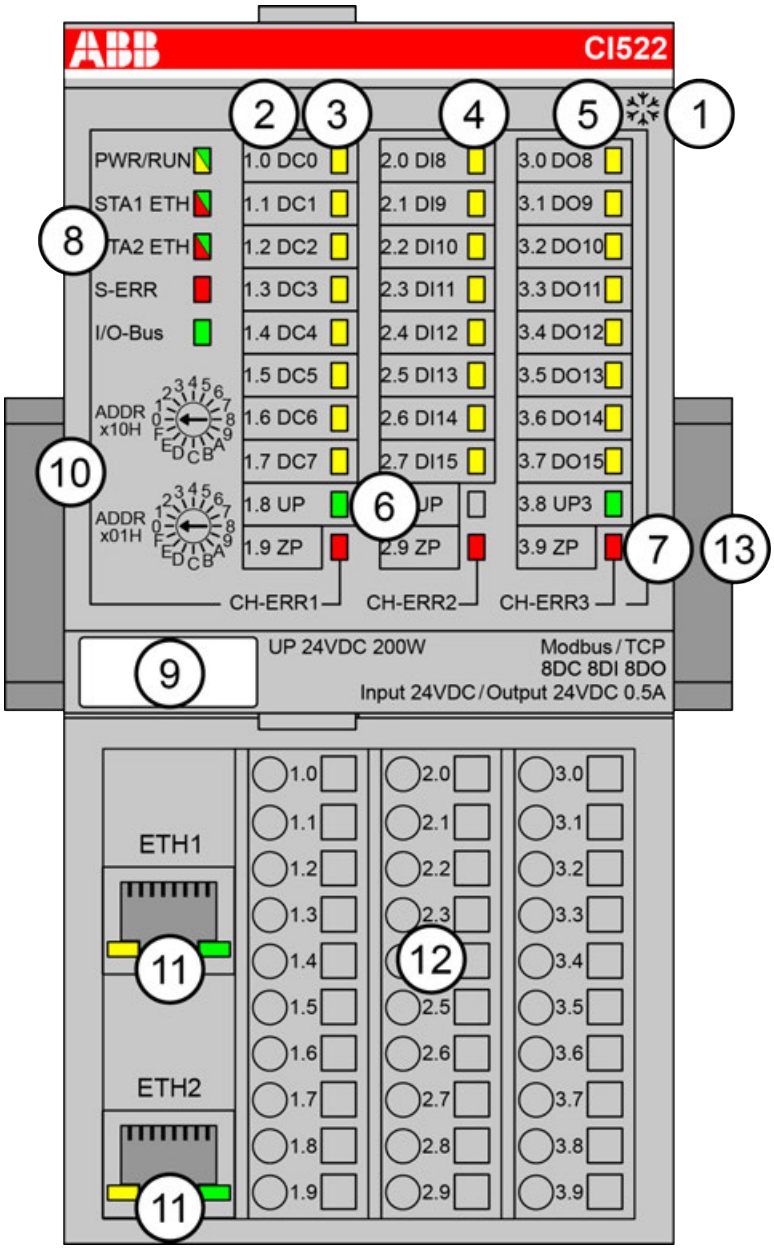
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 222 100 R0001	CI521-MODTCP, Modbus TCP communication interface module, 4 AI, 2 AO, 8 DI and 8 DO	Active
1SAP 422 100 R0001	CI521-MODTCP-XC, Modbus TCP communication interface module, 4 AI, 2 AO, 8 DI and 8 DO, XC version	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CI522-MODTCP

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the IP address
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

Modbus TCP communication interface module CI522-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the configurable digital inputs/outputs is performed by software.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Interface	Ethernet
Protocol	Modbus TCP
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	for setting the last BYTE of the IP ADDRESS (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Required terminal unit	TU507 or TU508 ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet bus module CI522-MODTCP is plugged on the I/O terminal unit TU507-ETH ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095* or TU508-ETH ↳ *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Conditions for undisturbed operating with older I/O expansion modules

All I/O expansion modules that are attached to the CI52x-MODTCP must be powered up together with the CI52x-MODTCP if the firmware version of these I/O expansion modules is V1.9 or lower.

The firmware version is related to the index. The index is printed on the module type label on the right side.

Modules as of index listed in the following table can be powered up independently.

S500 I/O module type	First index with firmware version above 1.9
AI523	D0
AI523-XC	D0
AI531	A3
AI531-XC	A0
AO523	D0
AO523-XC	D0
AX521	D0
AX521-XC	D0
AX522	D0
AX522-XC	D0
CD522	A2
CD522-XC	A0
DA501	A2
DA501-XC	A0
DA502	A1
DA502-XC	A1
DC522	D0
DC522-XC	D0
DC523	D0
DC523-XC	D0
DC532	D0
DC532-XC	D0
DI524	D0

S500 I/O module type	First index with firmware version above 1.9
DI524-XC	D0
DO524	A2
DO524-XC	A2
DX522	D0
DX522-XC	D0
DX531	D0
AC522	D0
PD501	D0

Do not connect any voltages externally to digital outputs!

This ist not intended usage.

Reason: Externally voltages at one or more terminals DC0...DC7 or DO8...DO15 may cause that other digital outputs are supplied through that voltage instead of voltage UP3 (reverse voltage).

This is also possible, if DC channels are used as inputs. For this, the source for the input signals should be the impressed UP3 of the device.

This limitation does not apply for the input channels DI0..DI7.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO8...DO15 and DC0...DC7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	DC0	Signal of the configurable digital input/output DC0
1.1	DC1	Signal of the configurable digital input/output DC1
1.2	DC2	Signal of the configurable digital input/output DC2
1.3	DC3	Signal of the configurable digital input/output DC3
1.4	DC4	Signal of the configurable digital input/output DC4
1.5	DC5	Signal of the configurable digital input/output DC5
1.6	DC6	Signal of the configurable digital input/output DC6
1.7	DC7	Signal of the configurable digital input/output DC7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)

Terminal	Signal	Description
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO8	Signal of the digital output DO8
3.1	DO9	Signal of the digital output DO9
3.2	DO10	Signal of the digital output DO10
3.3	DO11	Signal of the digital output DO11
3.4	DO12	Signal of the digital output DO12
3.5	DO13	Signal of the digital output DO13
3.6	DO14	Signal of the digital output DO14
3.7	DO15	Signal of the digital output DO15
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figure shows the connection of the Ethernet bus module CI522-MODTCP.

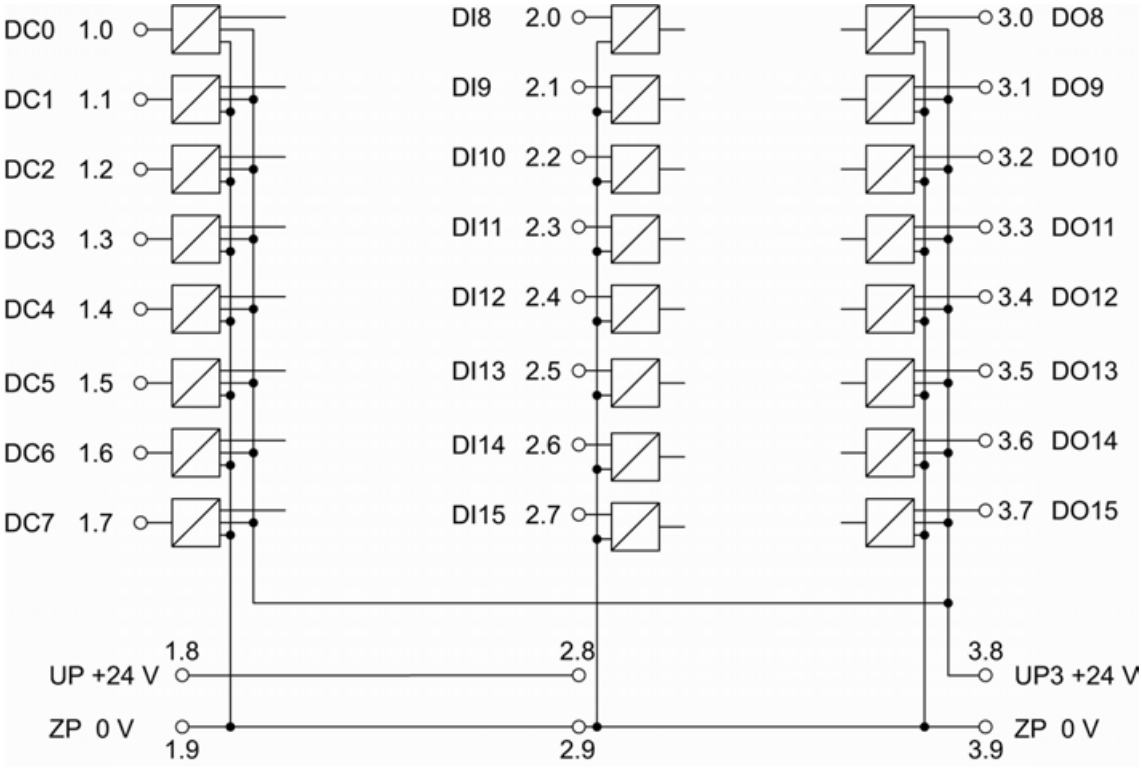


Fig. 1001: Connection of the communication interface module CI522-MODTCP

Further information is provided in the System Technology chapter *Chapter 1.6.4.3.1 “Modbus communication interface module” on page 5651*.

Connection of the digital inputs

The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.

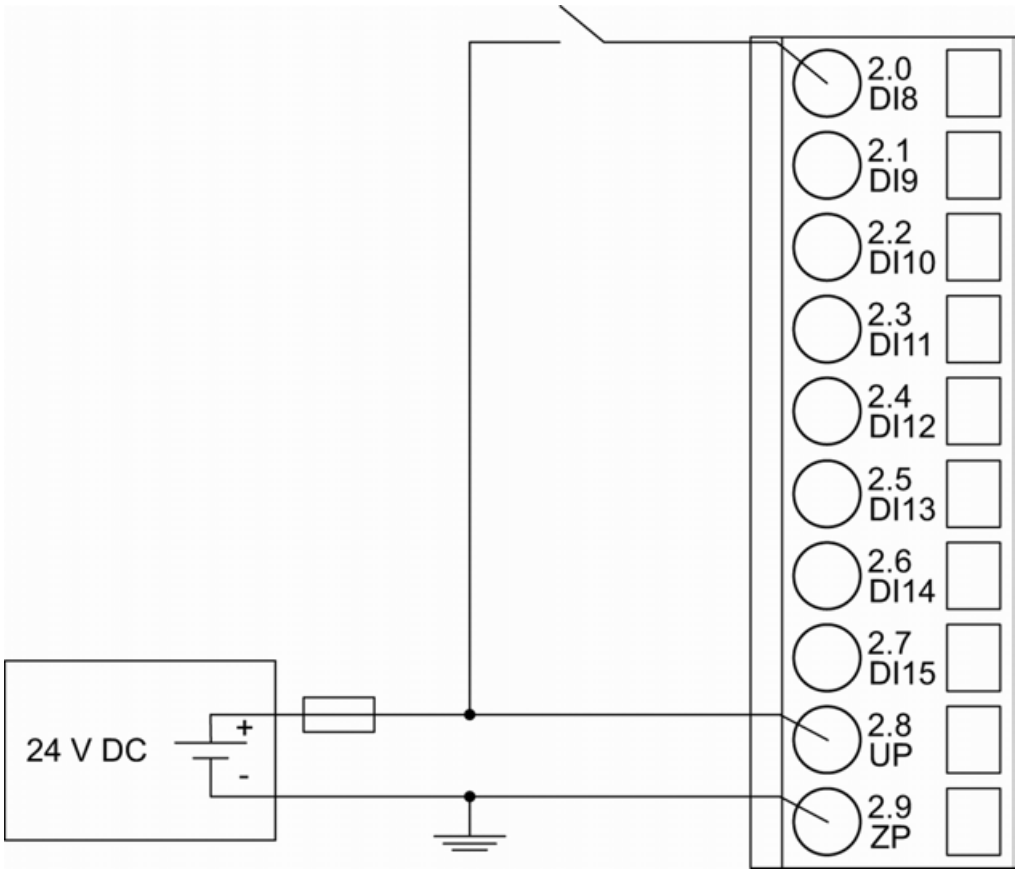
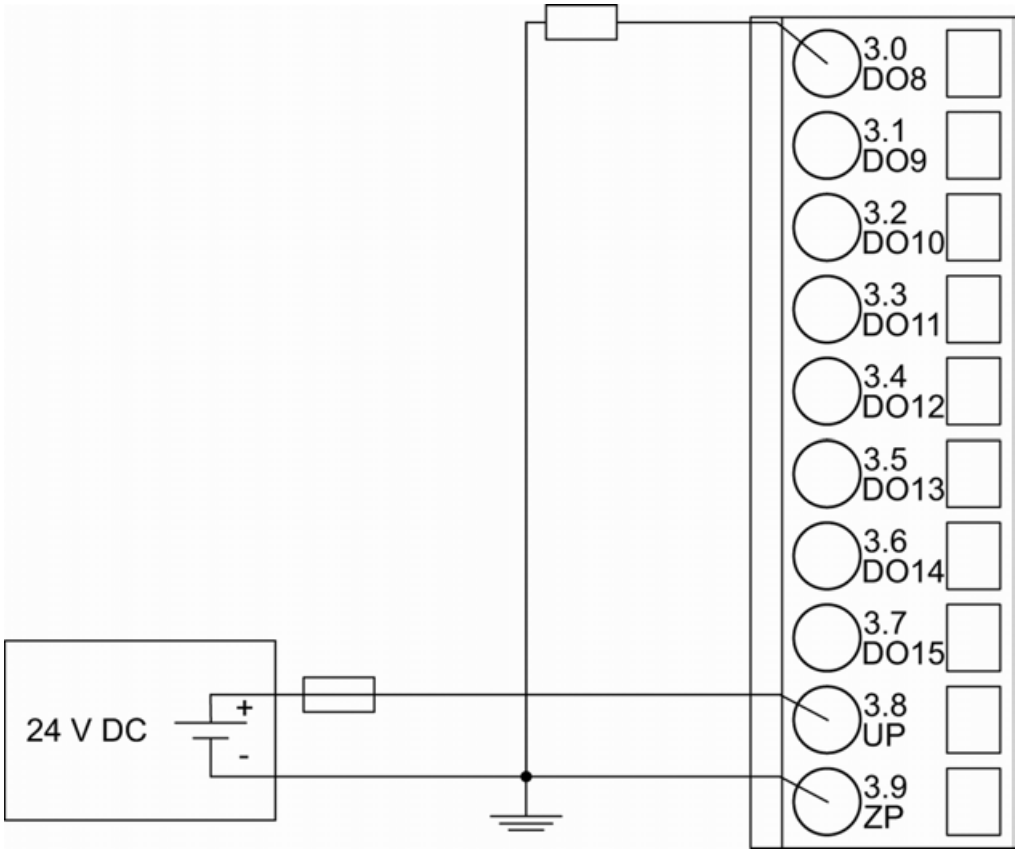


Fig. 1002: Connection of the digital inputs to the module CI522-MODTCP

The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.5.2.8.1 “State LEDs” on page 4922*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

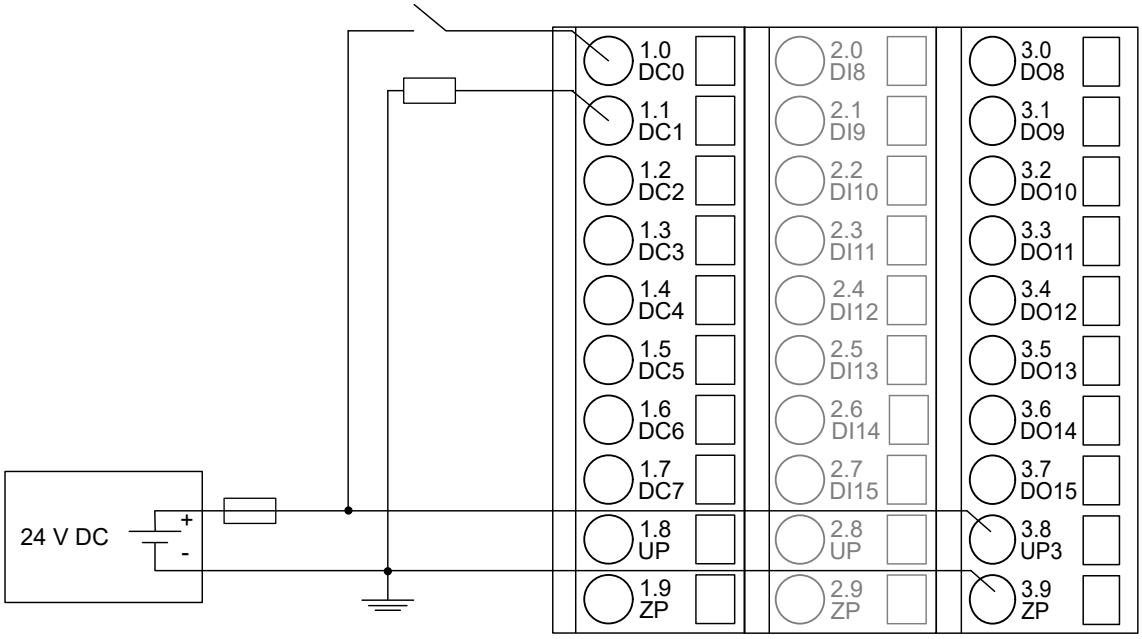


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.5.2.8.1 “State LEDs” on page 4922*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

CAUTION!
If a DC channel is used as input, the source for the input signals should be the impressed UP3 of the device *Chapter 1.6.2.8.5.2.3 “Connections” on page 4906*.

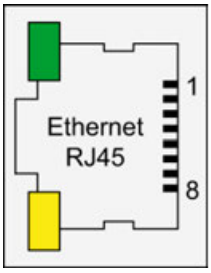


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.5.2.8.1 “State LEDs” on page 4922*.

Assignment of the Ethernet ports

The terminal unit for the Communication Interface Module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

Internal data exchange

Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

The IP address of the CI5221-MODTCP Module can be set with the “ABB IP Configuration Tool”. *Chapter 1.6.5.2.2.2.2 “Configuration of the IP settings with the IP configuration tool” on page 5816*.

If the last byte of the IP is set to 0, the address switch will be used instead.

Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings. This is a backup so the module can always get a valid IP address and can be configured by the “ABB IP Configuration Tool”.

Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

The factory setting for the IP is 192.168.0.x (last byte is address switch).

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI522-MODTCP stores configuration parameters (IP address configuration, module parameters).

The digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.2.8.5.2.7 “Parameterization” on page 4914*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7405	WORD	7405
Ignore Module	Internal	0	BYTE	0
Parameter length	Internal	47	BYTE	47
Error LED / Failsafe function (Table Error LED / Failsafe function *Table 539 “Table Error LED / Failsafe function” on page 4916*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		

Name	Value	Internal value	Internal value, type	Default
	Off by E3 + fail-safe	19		
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Master IP for Write restriction [4])	No master IP Master IP	0,0,0,0 W,X,y,z	ARRAY[0..3] OF BYTE	0,0,0,0
Timeout for Bus supervision	No supervision 10 ms timeout 20 ms timeout	0 1 2	BYTE	No supervision
IO Mapping Structure [3])	Fixed Mapping Dynamic Mapping	0 1	BYTE	0
Reserved	Internal	0	ARRAY[0..2] OF BYTE	0,0,0
Check supply	off on	0 1	BYTE	1
Fast counter	0 : 10 [2])	0 : 10	BYTE	0

Remarks:

[1])	With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.
[2])	Counter operating modes *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*

[3])	Fixed Mapping means each module has its own Modbus registers for data transfer independent of the I/O bus constellation description. For details see *Chapter 1.6.4.3.1.2 “Modbus TCP registers” on page 5652*. Dynamic mapping means the structure of the IO Date is dependent on the I/O bus constellation. Each I/O bus expansion module starts directly after the module before on the next Word adress.
[4])	If none of the parameters is set all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules. If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device, all other masters / clients still have read access to the CI52x-MODTCP device.

Table 539: Table Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.	

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0 ... 65535	0000h ... FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behaviour DO at comm. error is apply to DC and DO channels and only analyzed if the Failsafe-mode is ON.
[2])	The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears, if externally voltage at digital outputs DC0...DC7 and accordingly DO8...DO15 has exceeded the process supply voltage UP3 ↳ *Chapter 1.6.2.8.5.2.3 “Connections” on page 4906* (see description in section). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters on "OFF". This parameter should only be disabled in exceptional cases for voltage overflow may produce reverse voltage.

Diagnosis

Structure of the Diagnosis Block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI502-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O Module ... 10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error
6	Reserved	0

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

For diagnosis firmware version ≥ 3.2.6 is required.

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
Module errors							
3	-	31	31	31	19	Checksum error in the I/O module	Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module	
3	-	31	31	31	40	Different hard-/firm-ware versions in the module	
3	-	31	31	31	43	Internal error in the module	
3	-	31	31	31	36	Internal data exchange failure	
3	-	31	31	31	9	Overflow diagnosis buffer	Restart
3	-	31	31	31	26	Parameter error	Check Master
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6..7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap terminal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug projected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])	Restart, if error persists replace terminal unit
4	1...6	255	2	0	45	The connected Communication Module has no connection to the network	Check cabeling

E1..E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier 000..063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 Bit 6..7	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4 Bit 0..5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	46	Reverse voltage from digital out-puts DO8...DO15 to UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow at outputs (above UP3 level) [5])	Check termi-nals/ check process supply voltage
Channel error digital							
4	-	31	2	8..15	46	Externally voltage detected at digital output DO8...DO15 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0...DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0...DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO8...DO15[7])	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI502-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1..10 = Expansion module
[3])	With "Module" the following allocation applies dependent of the master: Module error: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DC0...DC7 oder DO8...DO15 cause that other digital outputs are supplied through that voltage (voltage feedback, see description in 'Connections' ↳ *Chapter 1.6.2.8.5.2.3 "Connections" on page 4906*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0...DC7 and accordingly DO8...DO15 has exceeded the process supply voltage UP3 ↳ *Chapter 1.6.2.8.5.2.3 "Connections" on page 4906*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DC0...DC7 or DO8...DO15 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 2000ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 540: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---

LED	Color	OFF	ON	Flashing
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	Device configured, acyclic data exchange running
	Red	---	Communication error (timeout) appeared	IP address error
STA2 ETH (System LED "SF")	Green	Device has valid parameters	Device is running parameterization sequenze	Device has no parameters
	Red	---	---	Device has invalid parameters
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 541: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Ethernet		10/100 base-TX, internal switch, 2 x RJ45 socket
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnosis		See Diagnosis and Displays *Chapter 1.6.2.8.5.2.8 “Diagnosis” on page 4917*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40°C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI8 to DI15		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO8 to DO15		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

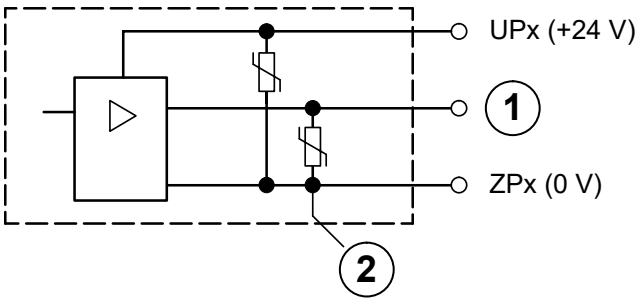


Fig. 1003: Digital input/output (circuit diagram)

1 Digital Output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC7	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC7	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V *)
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V *)
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0,8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.

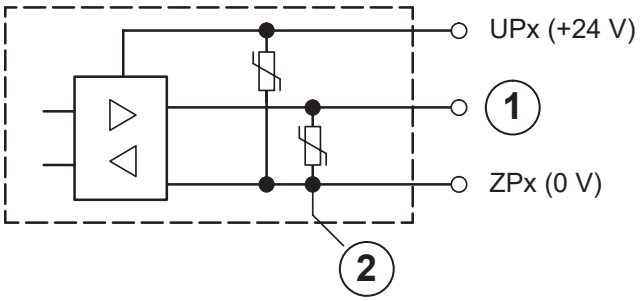


Fig. 1004: Digital input/output (circuit diagram)

1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI8),Terminal 2.1 (DI9)
Used outputs	Terminal 3.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1- 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz
Detailed description	See *Chapter 1.6.4.1.10 “Fast counters” on page 5498*
Operating modes	See *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

Ordering data

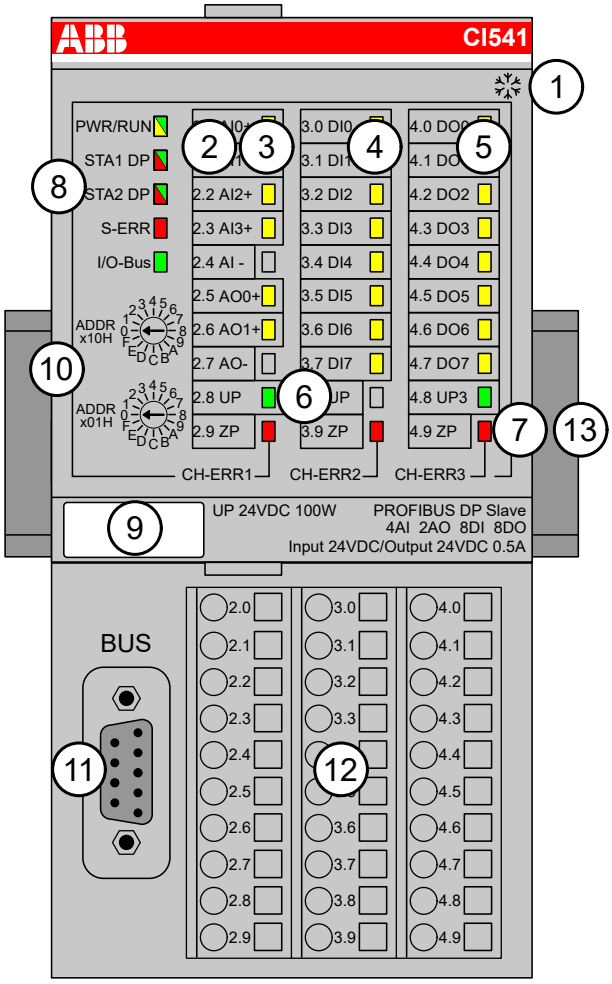
Ordering No.	Scope of delivery	Product life cycle phase *)
1SAP 222 200 R0001	CI522-MODTCP, Modbus TCP communication interface module, 8 DC, 8 DI and 8 DO	Active
1SAP 422 200 R0001	CI522-MODTCP-XC, Modbus TCP communication interface module, 8 DC, 8 DI and 8 DO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8.6 PROFIBUS

CI541-DP

- 4 configurable analog inputs (2-wire/single-ended) or 2 configurable analog inputs (3-wire/ differential)
  Resolution 12 bits plus sign
- 2 analog outputs
  Resolution 12 bits plus sign
- 8 digital inputs 24 V DC in 1 group
- 8 digital outputs 24 V DC in 1 group, 0.5 A max.
- Fast counter
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 DP, STA2 DP, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the PROFIBUS ID
11 9-pin D-SUB connector to connect the PROFIBUS DP signals

12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFIBUS DP communication interface module is used as decentralized I/O module in PROFIBUS DP networks. Depending on the used terminal unit the network connection is performed either via 9-pole female D-sub or via 10 terminals (screw-type or spring terminals) which are integrated in the terminal unit. The communication interface module contains 22 I/O channels.

The inputs/outputs are galvanically isolated from the PROFIBUS DP network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

- 4 configurable analog inputs (2-wire/single-ended) or 2 configurable analog inputs (3-wire/differential)
  Resolution 12 bits plus sign
- 2 analog outputs
  Resolution 12 bits plus sign
- 8 digital inputs 24 V DC in 1 group
- 8 digital outputs 24 V DC in 1 group, 0.5 A max.
- Fast counter
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available

Functionality

Parameter	Value
Interface	PROFIBUS
Protocol	PROFIBUS DP (DP-V0 and DP-V1)
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the PROFIBUS ID for configuration purposes (00h to FFh)
Fast counter	Integrated, configurable operating modes
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU509, TU510, TU517 or TU518 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* ↳ *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*

Connections

> *For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The PROFIBUS DP communication interface module CI541-DP is plugged on the I/O terminal units TU509 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* or TU510 ↳ *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* and accordingly TU517 ↳ *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* or TU518 ↳ *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8 and 3.8 as well as 2.9, 3.9 and 4.9 are interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 2.8 and 3.8: Process supply voltage UP = +24 V DC

Terminal 4.8: Process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: Process supply voltage ZP = 0 V

> *With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.*

> *Do not connect any voltages externally to digital outputs!*
>
> *Reason: Externally voltages at an output or several outputs may cause that other outputs are supplied through that voltage instead of voltage UP3 (reverse voltage). This ist not intended usage.*

CAUTION!

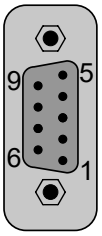
Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0..DO7.

Possibilities of connection

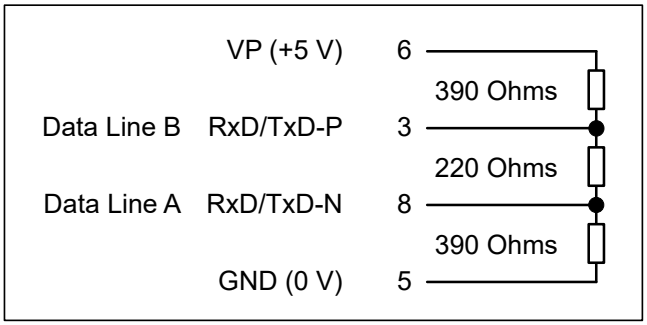
Connection on terminal units TU509 or TU510

The assignment of the 9-pole female D-sub for the PROFIBUS signals:

	1	---	Reserved
	2	---	Reserved
	3	B	Data line B (receive and send line, positive)
	4	---	Reserved
	5	DGND	Reference potential for data transmissions and +5 V
	6	VP (5 V)	+5 V (Power supply voltage for terminating resistors)
	7	---	Reserved
	8	A	Data line A (receive and send line, negative)
	9	---	Reserved
	Shield	Shield	Shield, functional earth

Bus termination

The line ends of the bus segment must be equipped with bus terminating resistors. Normally, these resistors are integrated in the interface connectors.



The grounding of the shield should take place at the switchgear cabinet, see System Data AC500 ↪ Chapter 1.6.3.6.1 “System data AC500” on page 5313.

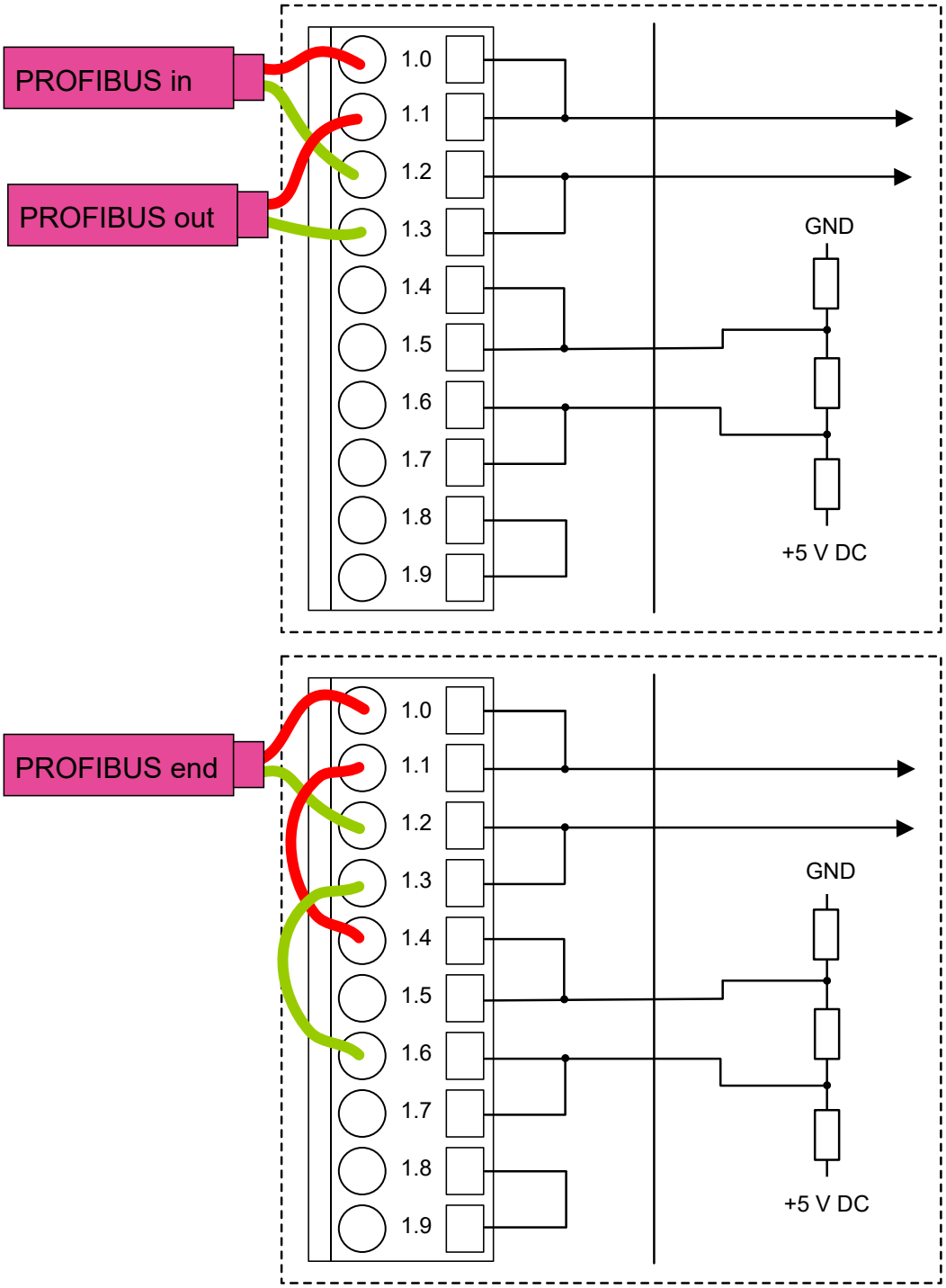
Mounting on terminal units TU517 or TU518

The assignment of the terminals 1.0 - 1.9:

Terminal	Signal	Description
1.0	B	Data line B (receive and send line, positive)
1.1	B	Data line B (receive and send line, positive)
1.2	A	Data line A (receive and send line, negative)
1.3	A	Data line A (receive and send line, negative)
1.4	TermB	Bus termination data line B
1.5	TermB	Bus termination data line B
1.6	TermA	Bus termination data line A

Terminal	Signal	Description
1.7	TermA	Bus termination data line A
1.8	DGND	Reference potential for data transmission
1.9	DGND	Reference potential for data transmission

At the line ends of a bus segment, terminating resistors must be connected. If using TU517/ TU518, the bus terminating resistors can be enabled by connecting the terminals TermA and TermB to the data lines A and B (no external terminating resistors are required, see figure below).



If using TU517/TU518, note that the terminating resistors are not located inside the TU, but inside the communication interface module CI541-DP. I. e. when removing the device from the TU, the bus terminating resistors are not connected to the bus any more. The bus itself will not be disconnected if a device is removed.

If using TU517/TU518 the max. permitted transmission rate is limited to 1.5 MBaud.

The grounding of the shield should take place at the switchgear cabinet, see System Data AC500 ↬ Chapter 1.6.3.6.1 “System data AC500” on page 5313.

Technical data bus cable

Parameter	Value
Type	Twisted pair (shielded)
Characteristic impedance	135...165 Ω
Cable capacitance	< 30 pF/m
Conductor diameter of the cores	≥ 0.64 mm
Conductor cross section of the cores	≥ 0.34 mm²
Cable resistance per core	≤ 55 Ω/km
Loop resistance (resistance of two cores)	≤ 110 Ω/km

Cable length

The maximum possible cable length of a PROFIBUS subnet within a segment depends on the transmission rate (baud rate).

Transmission rate	Maximum cable length
9.6 kBaud to 93.75 kBaud	1200 m
187.5 kBaud	1000 m
500 kBaud	400 m
1.5 MBaud	200 m
3 MBaud to 12 MBaud	100 m

The assignment of the other terminals:

Terminal	Signal	Description
2.0	AI0+	Positive pole of analog input signal 0
2.1	AI1+	Positive pole of analog input signal 1
2.2	AI2+	Positive pole of analog input signal 2
2.3	AI3+	Positive pole of analog input signal 3

Terminal	Signal	Description
2.4	AI-	Negative pole of analog input signals 0 to 3
2.5	AO0+	Positive pole of analog output signal 0
2.6	AO1+	Positive pole of analog output signal 1
2.7	AI-	Negative pole of analog output signals 0 and 1
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI0	Signal of the digital input DI0
3.1	DI1	Signal of the digital input DI1
3.2	DI2	Signal of the digital input DI2
3.3	DI3	Signal of the digital input DI3
3.4	DI4	Signal of the digital input DI4
3.5	DI5	Signal of the digital input DI5
3.6	DI6	Signal of the digital input DI6
3.7	DI7	Signal of the digital input DI7
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO0	Signal of the digital output DO0
4.1	DO1	Signal of the digital output DO1
4.2	DO2	Signal of the digital output DO2
4.3	DO3	Signal of the digital output DO3
4.4	DO4	Signal of the digital output DO4
4.5	DO5	Signal of the digital output DO5
4.6	DO6	Signal of the digital output DO6
4.7	DO7	Signal of the digital output DO7
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

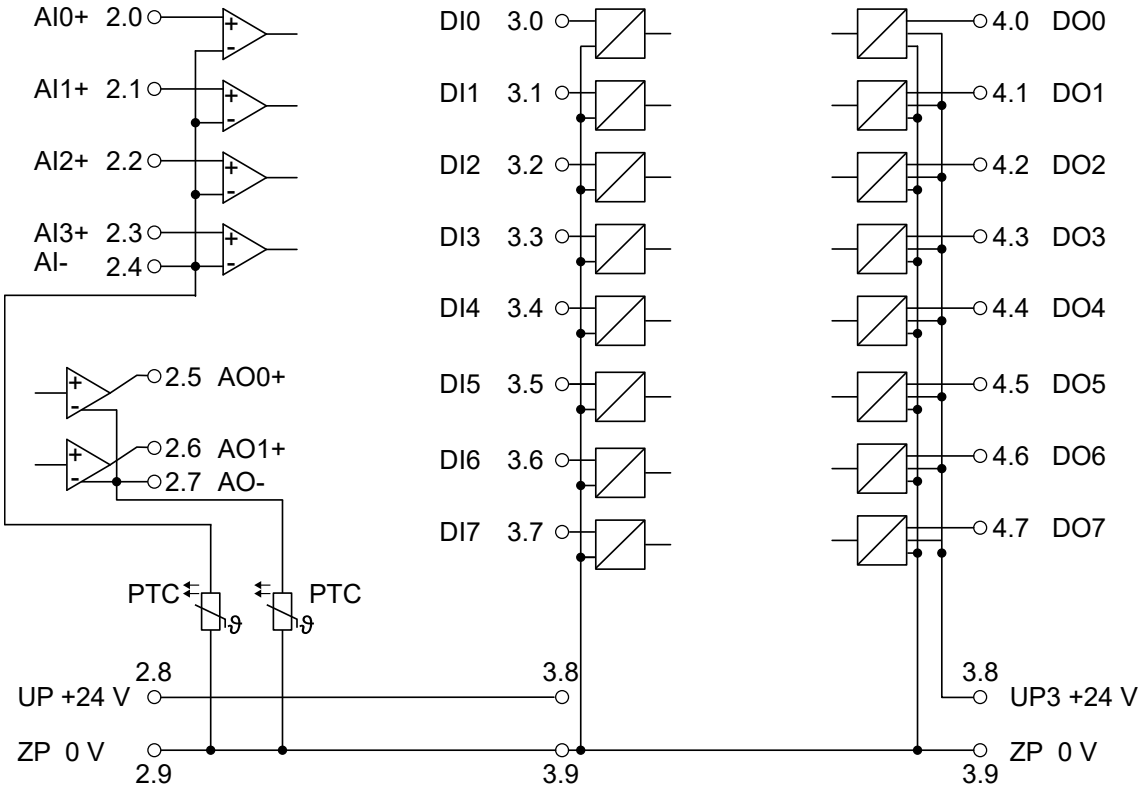
– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Analog signals are always laid in shielded cables. The cable shields are grounded at both ends of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

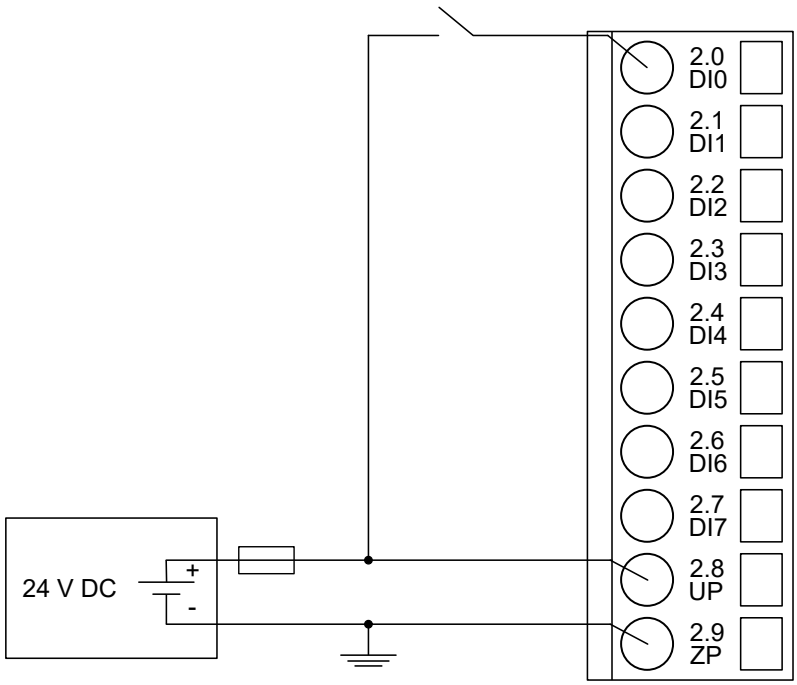
For simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

The following figures show the connection of the PROFIBUS DP communication interface module CI541-DP.



Connection of the digital inputs

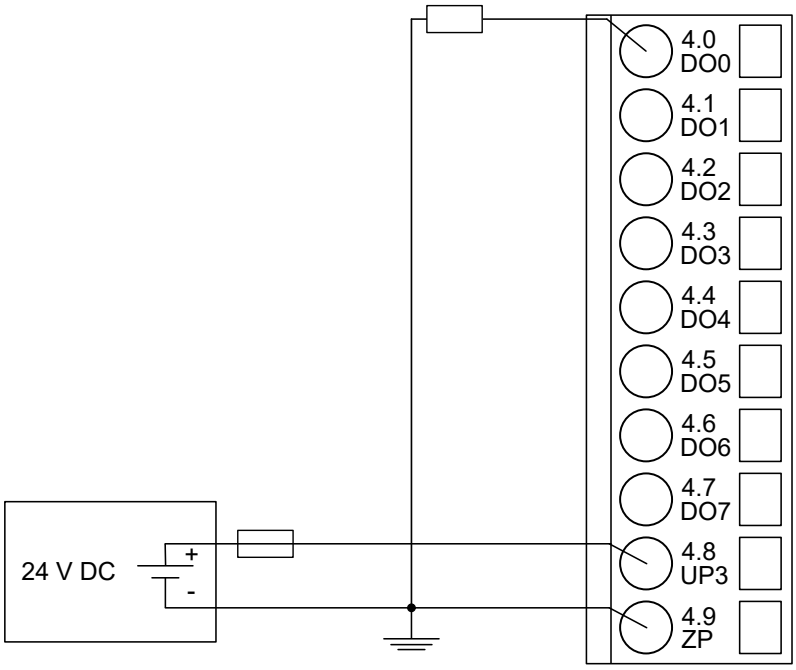
The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

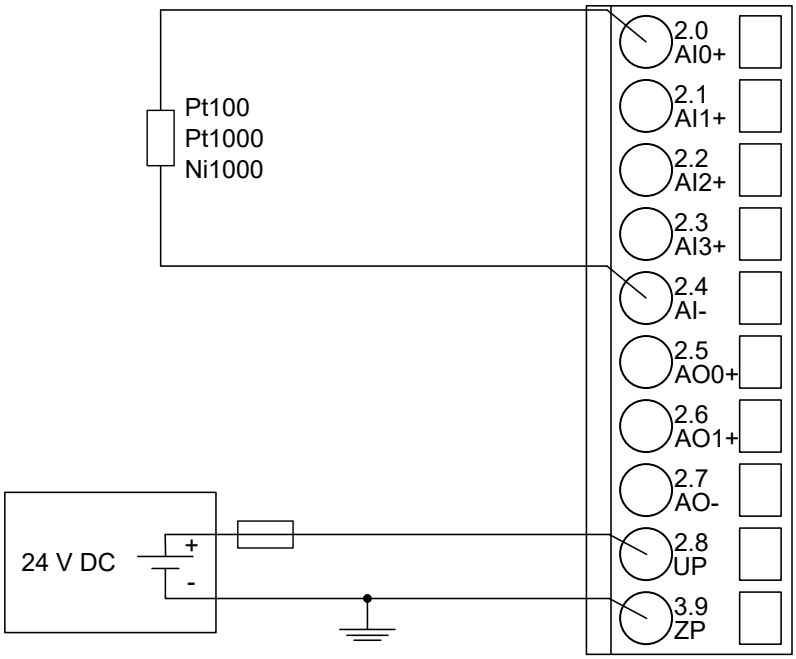


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI541-DP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

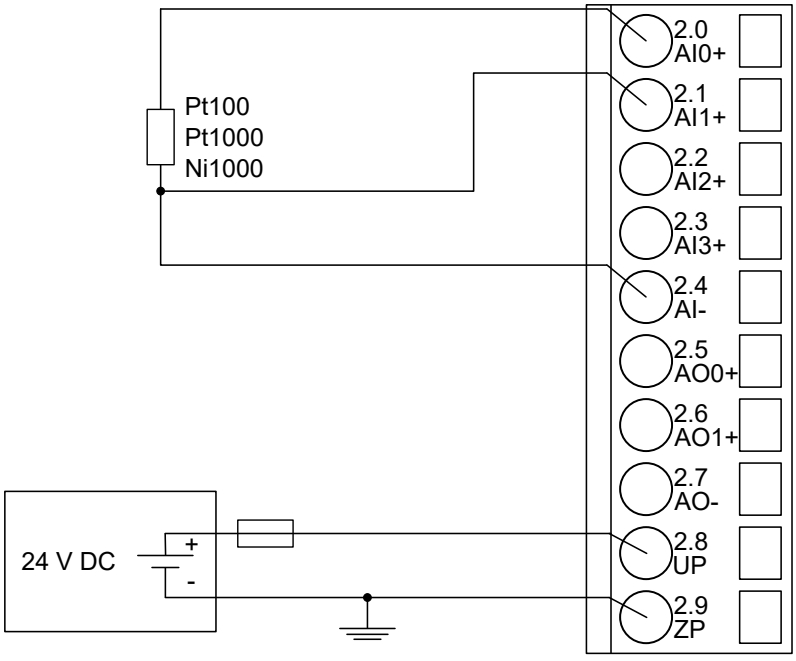
The module CI541-DP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI541-DP provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.



With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

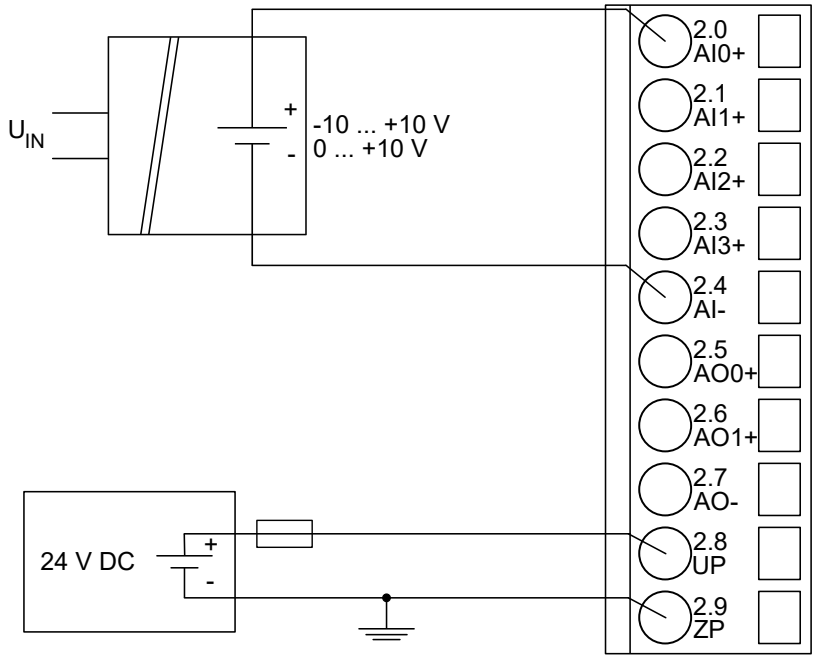
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

The module CI541-DP performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

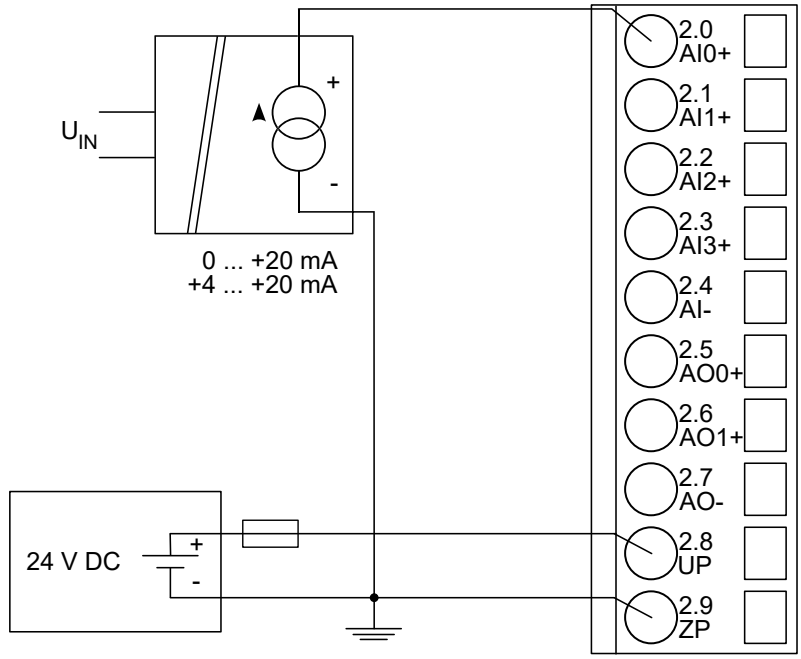
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

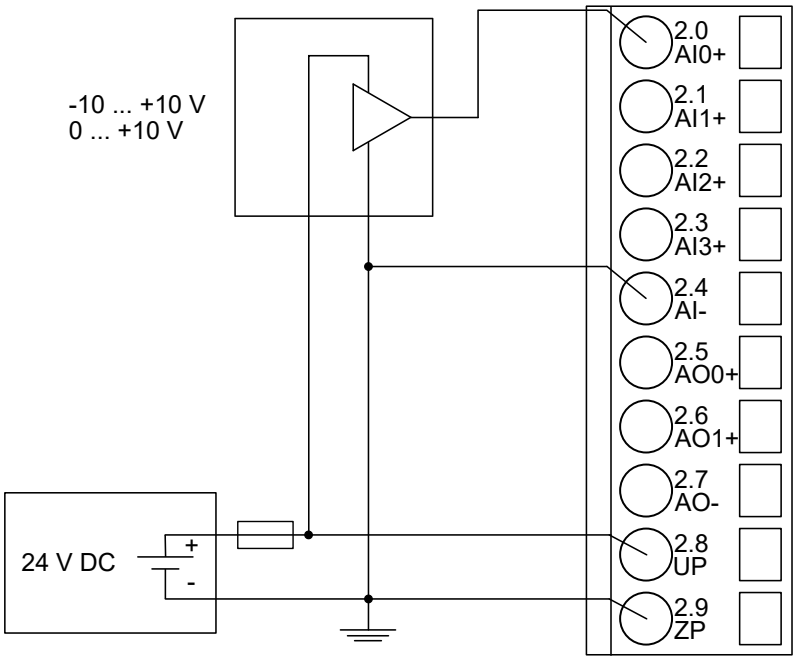
The function of the LEDs is described under *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ± 1 V).

Make sure that the potential difference never exceeds ± 1 V (also not with long cable lengths).

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

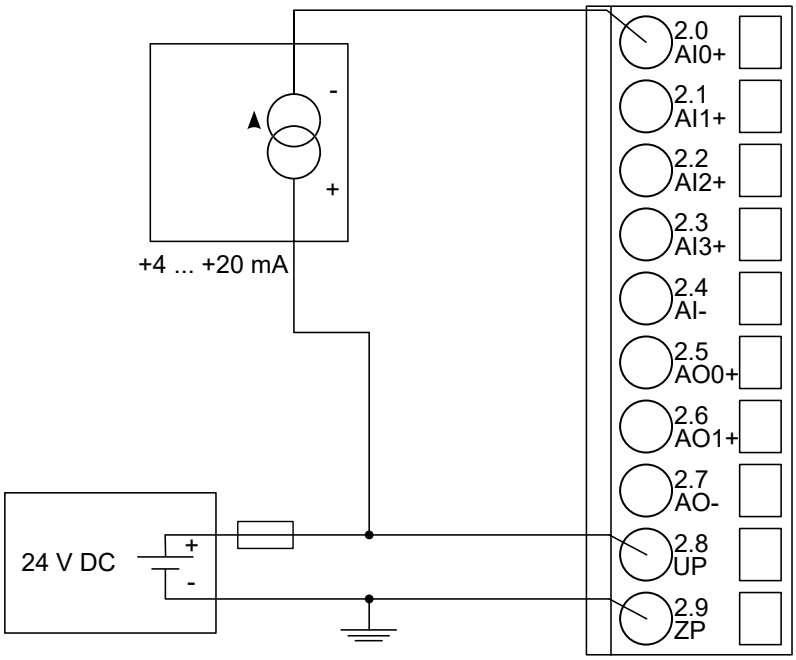
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* :

Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to AIx+ and ZP.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4 mA...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

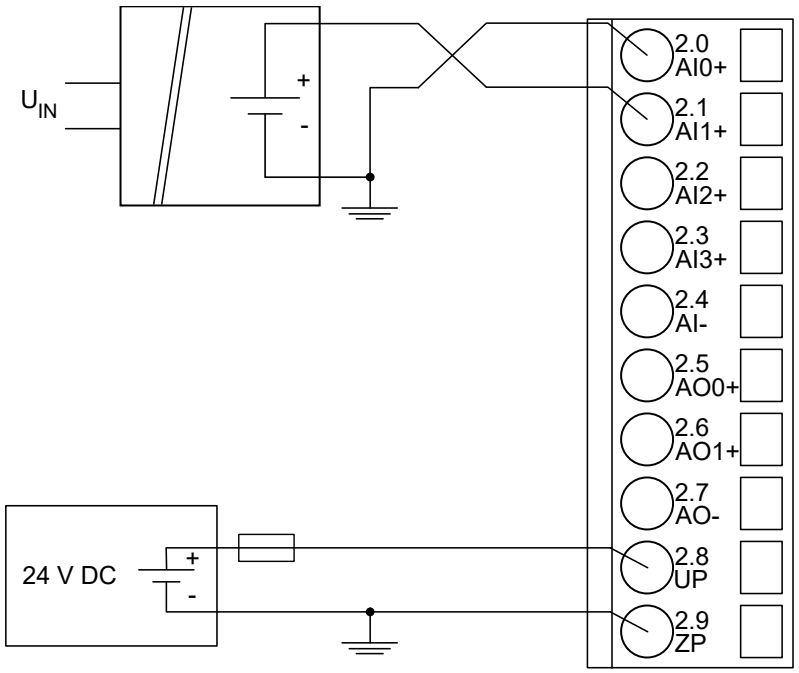
CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Voltage	0 V...10 V	with differential inputs, 2 channels used
Voltage	-10 V...+10 V	with differential inputs, 2 channels used

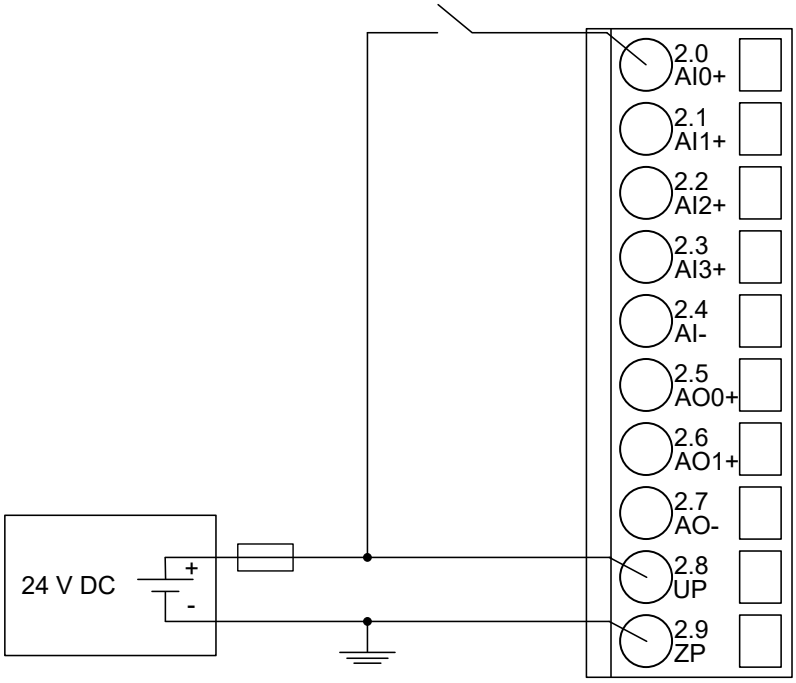
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



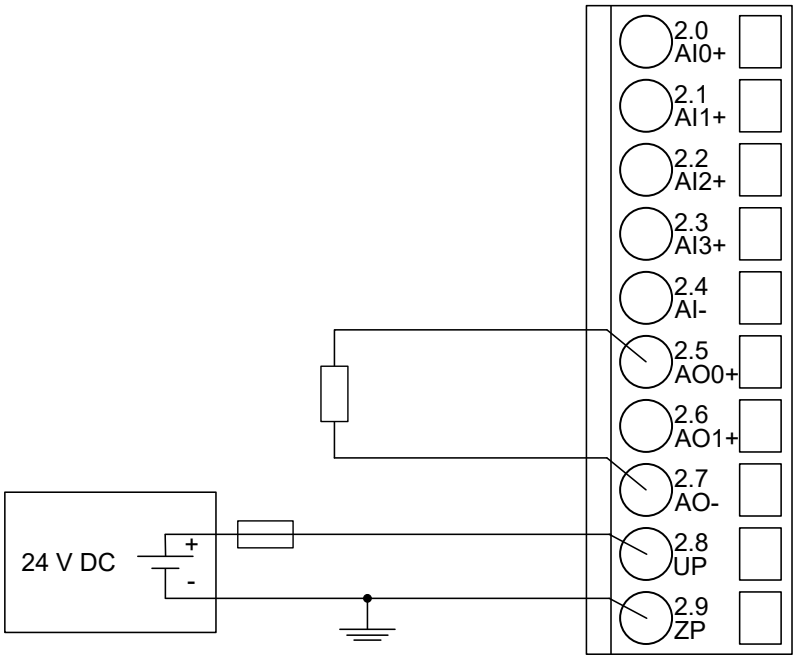
The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Digital input	24 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

Connection of analog output loads (Voltage)

The following figure shows the connection of analog output loads (voltage) to the analog output AO0. Proceed with the analog output AO1 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

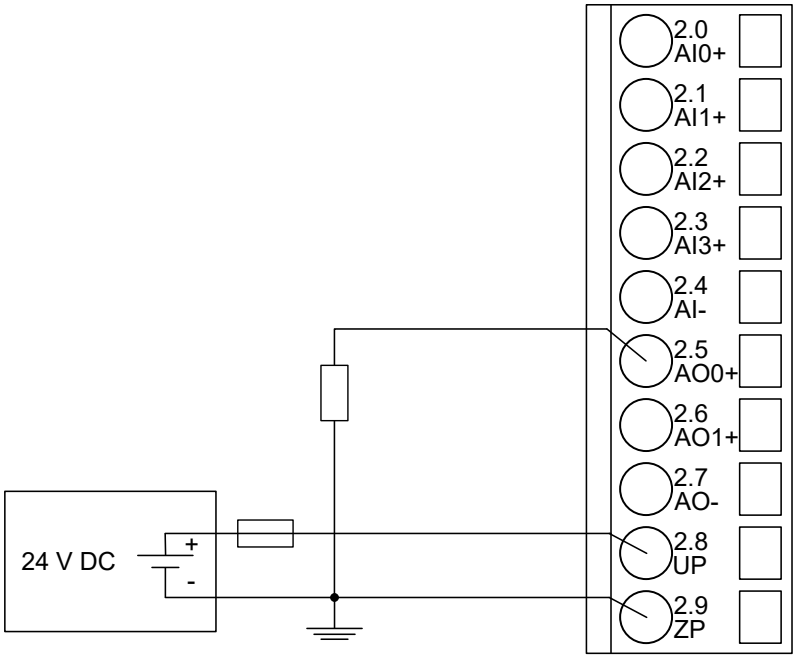
Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays *Chapter 1.6.2.8.6.1.9 “State LEDs” on page 4958*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of analog output loads (current) to the analog output AO0. Proceed with the analog output AO1 in the same way.



The following measuring ranges can be configured *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948* *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*:

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays *Chapter 1.6.2.8.6.1.8 “Diagnosis” on page 4953*.

Unused analog outputs can be left open-circuited.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI541-DP PROFIBUS DP Bus configuration is handled by PROFIBUS DP master with the exception of the slave bus ID (via rotary switches) and the transmission rate (automatic detection).

The analog/digital I/O channels and the fast counter are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.2.8.6.1.7 “Parameterization” on page 4948*.

Parameterization

Parameters of the module

Table 542: Parameters of the module:

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C20	WORD	0x1C20
Parameter length	Internal	47	BYTE	47
Reserved (1 byte)	0	0	BYTE	0
Error LED / Failsafe function (see *Table 543 “Settings "Error LED / Failsafe function"” on page 4949*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved (20 bytes)	0	0	BYTE	0
Check supply (UP and UP3)	On	0	BYTE	
	Off	1		1
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the Modules reports a "parameter error" and does not perform cyclic process data transmission

[2]) Counter operating modes, see description of the fast counter *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

Table 543: Settings "Error LED / Failsafe function"

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe mode on *)
*) The parameters Behaviour analog outputs at communication error and Behaviour digital outputs at communication error are only evaluated if failsafe function is enabled.	

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard Reserved	0 255	BYTE	0
Behaviour analog outputs at communication error *)	Off Last value Last value 5 s Last value 10 s Substitute value Substitute value 5 s Substitute value 10 s	0 1 6 11 2 7 12	BYTE	0
*) The parameter Behaviour analog outputs at communication error is only analyzed if the Failsafe mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Operation modes of analog inputs ↳ *Table 544 “Operation modes of analog inputs:” on page 4950*	Operation modes of analog inputs ↳ *Table 544 “Operation modes of analog inputs:” on page 4950*	BYTE	0
Input 0, Check channel	Settings channel monitoring ↳ *Further information on page 4951*	Settings channel monitoring ↳ *Further information on page 4951*	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	Operation modes of analog inputs ↳ *Table 544 “Operation modes of analog inputs:” on page 4950*	Operation modes of analog inputs ↳ *Table 544 “Operation modes of analog inputs:” on page 4950*	BYTE	0
Input 3, Check channel	Settings channel monitoring ↳ *Further information on page 4951*	Settings channel monitoring ↳ *Further information on page 4951*	BYTE	0

Channel configuration

Table 544: Operation modes of analog inputs:

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

19	3-wire Ni1000 -50 °C...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Channel monitoring

Table 545: Table settings channel monitoring:

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Operation modes of analog outputs ↳ *Table 546 “Table operation modes of analog outputs:” on page 4952*	Operation modes of analog outputs ↳ *Table 546 “Table operation modes of analog outputs:” on page 4952*	BYTE	0
Output 0, Check channel	Channel monitoring ↳ *Table 547 “Table channel monitoring:” on page 4952*	Channel monitoring ↳ *Table 547 “Table channel monitoring:” on page 4952*	BYTE	0
Output 0, Substitute value	Substitute value ↳ *Table 548 “Table substitute value:” on page 4952*	Substitute value ↳ *Table 548 “Table substitute value:” on page 4952*	WORD	0
Output 1, Channel configuration	Operation modes of analog outputs ↳ *Table 546 “Table operation modes of analog outputs:” on page 4952*	Operation modes of analog outputs ↳ *Table 546 “Table operation modes of analog outputs:” on page 4952*	BYTE	0
Output 1, Check channel	Channel monitoring ↳ *Table 547 “Table channel monitoring:” on page 4952*	Channel monitoring ↳ *Table 547 “Table channel monitoring:” on page 4952*	BYTE	0
Output 1, Substitute value	Substitute value ↳ *Table 548 “Table substitute value:” on page 4952*	Substitute value ↳ *Table 548 “Table substitute value:” on page 4952*	WORD	0

Channel configuration

Table 546: Table operation modes of analog outputs:

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Channel monitoring

Table 547: Table channel monitoring:

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Substitute value

Table 548: Table substitute value:

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour digital outputs at communcation error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...255	00h...FFh	BYTE	0 0x00
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	Off 0x00
[1]) The parameters Behaviour digital outputs at communcation error is only analyzed if the Failsafe-mode is ON. [2]) The state "externally voltage detected" appears, if the output of a channel DC0..DC7 should be switched on while an externally voltage is connected *Chapter 1.6.2.8.6.1.3 “Connections” on page 4932*. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".				

Diagnosis

Structure of the Diagnosis Block via DPM_SLV_DIAG function block. *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*.

Byte Number	Description	Possible Values
1	Data length (header included)	7
2	PROFIBUS DP V1 coding: Vendor specific	129
3	Diagnosis Byte, slot number	31 = CI541-DP (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O Module ... 10 = 10th connected S500 I/O Module
4	Diagnosis Byte, module number	According to the I/O Bus specification passed on by modules to the fieldbus master
5	Diagnosis Byte, channel	According to the I/O Bus specification passed on by modules to the fieldbus master

Byte Number	Description	Possible Values
6	Diagnosis Byte, error code	According to the I/O Bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
7	Diagnosis Byte, flags	According to the I/O Bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<− Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O device	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])	Restart, if error persists replace terminal unit
4	-	31	31	31	46	Reverse voltage from digital outputs DO0...DO7 to UP3 [4])	Check connection
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Externally voltage detected on digital output DO0...DO7 [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0...3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0...3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0...3	47	Short-circuit at an analog input	Check terminals
4	-	31	3	0...1	4	Analog value overflow at an analog output	Check output value
4	-	31	3	0...1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI541-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself; 1...10 = Expansion module
[3])	With "Module" the following allocation applies: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DO0...DO7 cause that other digital outputs are supplied through that voltage (voltage feedback, see description in section 'Connection' *Chapter 1.6.2.8.6.1.3 "Connections" on page 4932*). All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0...DO7 has overrun the process supply voltage UP3 (see description in section 'Connection' *Chapter 1.6.2.8.6.1.3 "Connections" on page 4932*). Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1-DP, STA2-DP, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 549: States of the 5 system LEDs:

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1-DP	Green	---	PROFIBUS running	Invalid device parameters

LED	Color	OFF	ON	Flashing
STA2-DP	Red	No error	Bus timeout	No communication to master
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No communication interface modules connected or communication error	Communication interface modules connected and operational	---

Table 550: States of the 27 process LEDs:

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 toDO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Overflow	>11.7589	>11.7589	>23.5178	>22.8142	
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006	
Normal range	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input
Normal range or measured value too low	0.0000	0.0000	0	4	Off
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0	
Measured value too low		-10.0004 : -11.7589			
Underflow	< -1.7593	<-11.7589	<0.0000	<0.0000	

Range	Digital value	
	Decimal	Hex.
Overflow	32767	7FFF
Measured value too high	32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	27648 : 1	6C00 : 0001
	0	0000
	-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low	-27649 : -32512	93FF : 8100
Underflow	-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C
Measured value too high		450.0 °C : 400.1 °C	

Range	**Pt100 / Pt1000** **-50...70 °C**	**Pt100 / Pt1000** **-50...400 °C**	**Ni1000** **-50...150 °C**
			160.0 °C : 150.1 °C
Normal range		400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C
		0.0 °C	0.0 °C
		-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C
Measured value too low		-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C

Range	**Digital value**	
	Decimal	**Hex.**
Overflow	32767	7FFF
Measured value too high	4500 : 4001	1194 : 0FA1
	1600 : 1501	0640 : 05DD
	800 : 701	0320 : 02BD
Normal range	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
	0	0000
	-1 : -500	FFFF : FE0C

Range	Digital value	
	Decimal	**Hex.**
Measured value too low	-501 : -600	FE0B : FDA8
Underflow	-32768	8000

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA
Overflow	>11.7589 V	>23.5178 mA	>22.8142 mA
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA
	0.0000 V	0.0000 mA	4.0000 mA
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA
Underflow	0 V	0 mA	0 mA

Range	Digital value	
	Decimal	**Hex.**
Overflow	> 32511	> 7EFF
Measured value too high	32511 : 27649	7EFF : 6C01
Normal range	27648 : 1	6C00 : 0001
	0	0000
	-1 -6912 -27648	FFFF E500 9400

Range	Digital value	
	Decimal	**Hex.**
Measured value too low	-27649 : -32512	93FF : 8100
Underflow	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	PROFIBUS interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Configurable digital inputs/outputs		8
Number of digital inputs		8
Number of digital outputs		8
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the PROFIBUS DP identifier		With 2 rotary switches at the front side of the module
Diagnose		See Diagnosis *Chapter 1.6.2.8.6.1.8 “Diagnosis” on page 4953*

Parameter	Value
Operation and error displays	32 LEDs (totally)
Weight (without terminal unit)	Ca. 125 g
Mounting position	Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 3.0 to 3.7
Reference potential for all inputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA

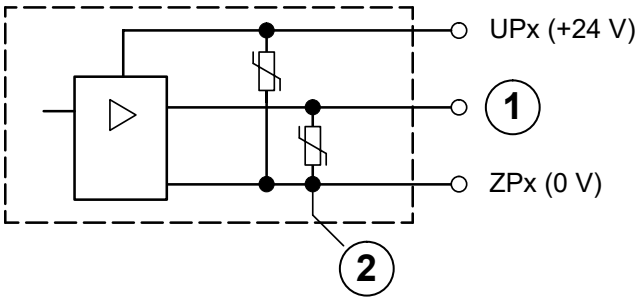
Parameter		Value
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 4.0 to 4.7
Reference potential for all outputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The module provides several diagnosis functions ↳ *Chapter 1.6.2.8.6.1.8 “Diagnosis” on page 4953*.

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 2.0 to 2.3
Reference potential for AI0+ to AI3+		Terminal 2.4 (AI-) for voltage and RTD measurement Terminal 2.9, 3.9 and 4.9 for current measurement
Input type		
	Unipolar	Voltage 0 V...10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V...+10 V
Galvanic isolation		Against PROFIBUS
Configurability		0 V...10 V, -10 V...+10 V, 0/4 mA...20 mA, Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0...10 V: 12 bits Range -10...+10 V: 12 bits + sign Range 0...20 mA: 12 bits Range 4...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Tables Input Ranges Voltage, Current and Digital Input and Input range resistance temperature detector ↳ *Chapter 1.6.2.8.6.1.10 “Measuring ranges” on page 4959*
Unused inputs	Are configured as "unused" (default value)
Overvoltage protection	Yes

Technical data of the analog inputs if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 2.0 to 2.3
Reference potential for the inputs		Terminals 2.9, 3.9 and 4.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +15 V
	Signal 1	+15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 2.5 ... 2.6
Reference potential for AO0+ to AO1+		Terminal 2.7 (AO-) for voltage output Terminal 2.9, 3.9 and 4.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against PROFIBUS
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output		0...500 Ω

Parameter	Value
Output loadability, as voltage output	±10 mA max.
Indication of the output signals	1 LED per channel (brightness depends on the value of the analog signal)
Resolution	12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)	Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range	Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code	Table Output Ranges Voltage and Current ↳ *Chapter 1.6.2.8.6.1.10.3 “Output ranges voltage and current” on page 4962*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI0), 3.1 (DI1)
Used outputs	Terminal 4.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

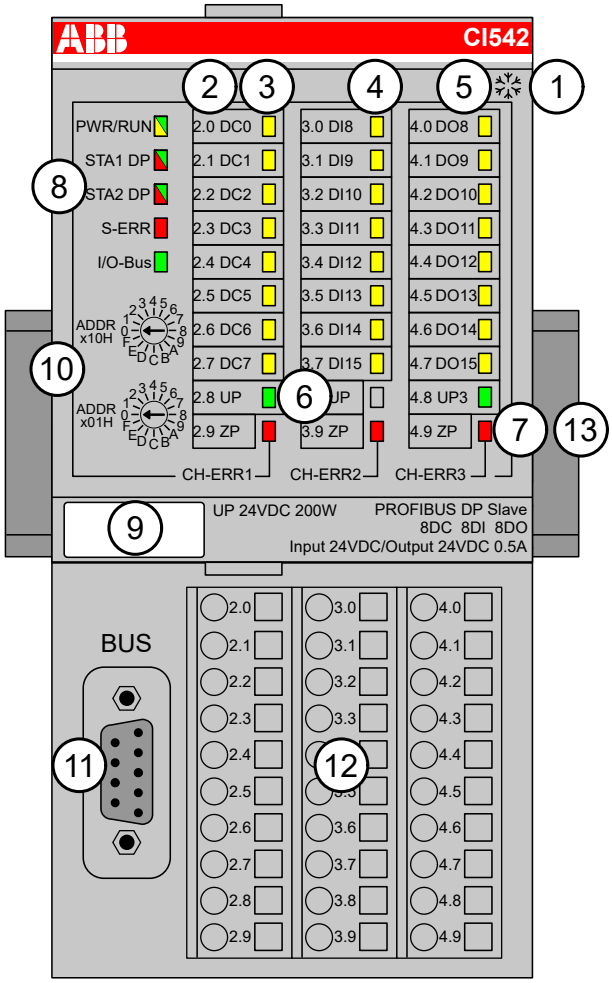
Ordering data

Ordering No.	Scope of delivery	Product life cycle phase *)
1SAP 224 100 R0001	CI541-DP, PROFIBUS DP communication interface module, 8 DI, 8 DO, 4 AI and 2 AO	Active
1SAP 424 100 R0001	CI541-DP-XC, PROFIBUS DP communication interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI542-DP

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the configurable digital inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 DP, STA2 DP, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the PROFIBUS ID
11 9-pin D-SUB connector to connect the PROFIBUS DP signals
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFIBUS DP communication interface module is used as decentralized I/O module in PROFIBUS networks. Depending on the used terminal unit the network connection is performed either via 9-pole female D-sub or via 10 terminals (screw-type or spring terminals) which are integrated in the terminal unit.

The inputs/outputs are galvanically isolated from the PROFIBUS network. There is no potential separation between the channels. The configuration of the configurable digital inputs/outputs is performed by software.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	PROFIBUS
Protocol	PROFIBUS DP (DP-V0 and DP-V1)
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the PROFIBUS ID for configuration purposes (00h to FFh)
Fast counter	Integrated, configurable operating modes
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU509, TU510, TU517 or TU518 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*

Connections

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly chapter *Chapter 1.6.3.5 “AC500-eCo” on page 5233.*

The PROFIBUS DP communication interface module CI542-DP is plugged on the I/O terminal units TU509 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* or TU510 *Chapter 1.6.2.5.2 “TU509 and TU510 for communication interface modules” on page 4099* and accordingly TU517 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109* or TU518 *Chapter 1.6.2.5.4 “TU517 and TU518 for communication interface modules” on page 4109*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

The terminals 2.8 and 3.8 as well as 2.9, 3.9 and 4.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 2.8 and 3.8: Process supply voltage UP = +24 V DC

Terminal 4.8: Process supply voltage UP3 = +24 V DC

Terminals 2.9, 3.9 and 4.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to digital outputs!

This ist not intended usage.

Reason: Externally voltages at one or more terminals DC0...DC7 or DO0...DO7 may cause that other digital outputs are supplied through that voltage instead of voltage UP3 (reverse voltage).

This is also possible, if DC channels are used as inputs. For this, the source for the input signals should be the impressed UP3 of the device.

This limitation does not apply for the input channels DI0...DI7.

CAUTION!

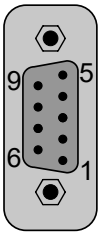
Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0...DO7 and DC0...DC7.

Possibilities of connection

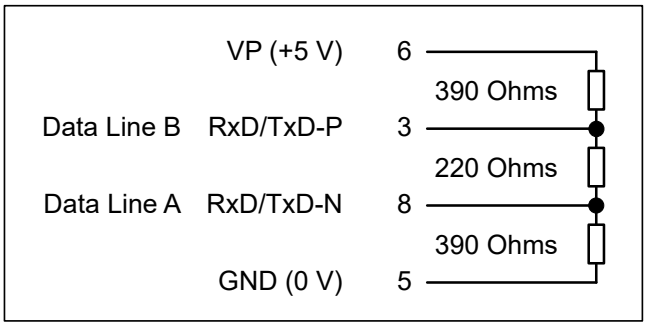
Mounting on terminal units TU509 or TU510:

The assignment of the 9-pole female D-sub for the PROFIBUS DP signals.

Serial Interface	Pin	Signal	Description
	1	---	Reserved
	2	---	Reserved
	3	B	PROFIBUS DP signal B
	4	---	Reserved
	5	DGND	Ground for 5 V power supply
	6	VP (5 V)	5 V power supply
	7	---	Reserved
	8	A	PROFIBUS DP signal A
	9	---	Reserved
	Shield	Cable shield	Functional earth

Bus termination

The line ends of the bus segment must be equipped with bus terminating resistors. Normally, these resistors are integrated in the interface connectors.



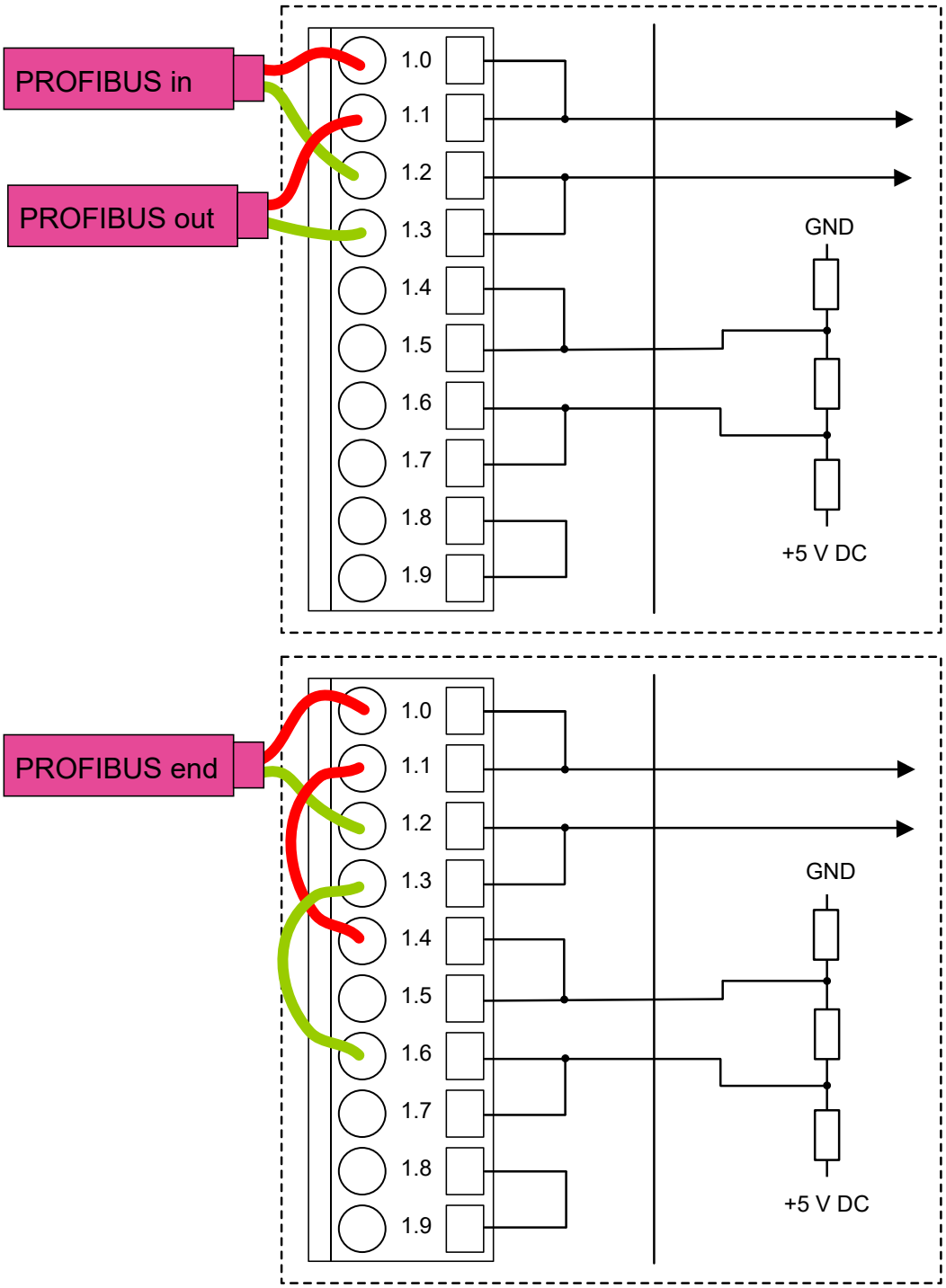
The grounding of the shield should take place at the switchgear cabinet, see System-Data AC500 ↬ Chapter 1.6.3.6.1 “System data AC500” on page 5313.

Mounting on terminal units TU517 or TU518:

The assignment of the terminals 1.0 - 1.9:

Terminal	Signal	Description
1.0	B	Data line B (receive and send line, positive)
1.1	B	Data line B (receive and send line, positive)
1.2	A	Data line A (receive and send line, negative)
1.3	A	Data line A (receive and send line, negative)
1.4	TermB	Bus termination data line B
1.5	TermB	Bus termination data line B
1.6	TermA	Bus termination data line A
1.7	TermA	Bus termination data line A
1.8	DGND	Reference potential for data transmission
1.9	DGND	Reference potential for data transmission

At the line ends of a bus segment, terminating resistors must be connected. If using TU517/ TU518, the bus terminating resistors can be enabled by connecting the terminals TermA and TermB to the data lines A and B (no external terminating resistors are required, see figure below).



If using TU517/TU518, note that the terminating resistors are not located inside the TU, but inside the communication interface module CI541-DP. I. e. when removing the device from the TU, the bus terminating resistors are not connected to the bus any more. The bus itself will not be disconnected if a device is removed.

If using TU517/TU518 the max. permitted transmission rate is limited to 1.5 MBaud.

Technical data bus cable

Parameter	Value
Type	Twisted pair (shielded)
Characteristic impedance	135 Ω...165 Ω
Cable capacitance	< 30 pF/m
Conductor diameter of the cores	≥ 0.64 mm
Conductor cross section of the cores	≥ 0.34 mm^2
Cable resistance per core	≤ 55 Ω/km
Loop resistance (resistance of two cores)	≤ 110 Ω/km

Cable length

The maximum possible cable length of a PROFIBUS subnet within a segment depends on the transmission rate (baud rate).

Transmission rate	Maximum cable length
9.6 kBaud to 93.75 kBaud	1200 m
187.5 kBaud	1000 m
500 kBaud	400 m
1.5 MBaud	200 m
3 MBaud to 12 MBaud	100 m

The assignment of the other terminals:

Terminal	Signal	Description
2.0	DC0	Signal of the configurable digital input/output DC0
2.1	DC1	Signal of the configurable digital input/output DC1
2.2	DC2	Signal of the configurable digital input/output DC2
2.3	DC3	Signal of the configurable digital input/output DC3
2.4	DC4	Signal of the configurable digital input/output DC4
2.5	DC5	Signal of the configurable digital input/output DC5
2.6	DC6	Signal of the configurable digital input/output DC6
2.7	DC7	Signal of the configurable digital input/output DC7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DI8	Signal of the digital input DI8
3.1	DI9	Signal of the digital input DI9
3.2	DI10	Signal of the digital input DI10
3.3	DI11	Signal of the digital input DI11
3.4	DI12	Signal of the digital input DI12
3.5	DI13	Signal of the digital input DI13
3.6	DI14	Signal of the digital input DI14

Terminal	Signal	Description
3.7	DI15	Signal of the digital input DI15
3.8	UP	Process voltage UP (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)
4.0	DO8	Signal of the digital output DO8
4.1	DO9	Signal of the digital output DO9
4.2	DO10	Signal of the digital output DO10
4.3	DO11	Signal of the digital output DO11
4.4	DO12	Signal of the digital output DO12
4.5	DO13	Signal of the digital output DO13
4.6	DO14	Signal of the digital output DO14
4.7	DO15	Signal of the digital output DO15
4.8	UP3	Process voltage UP3 (24 V DC)
4.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

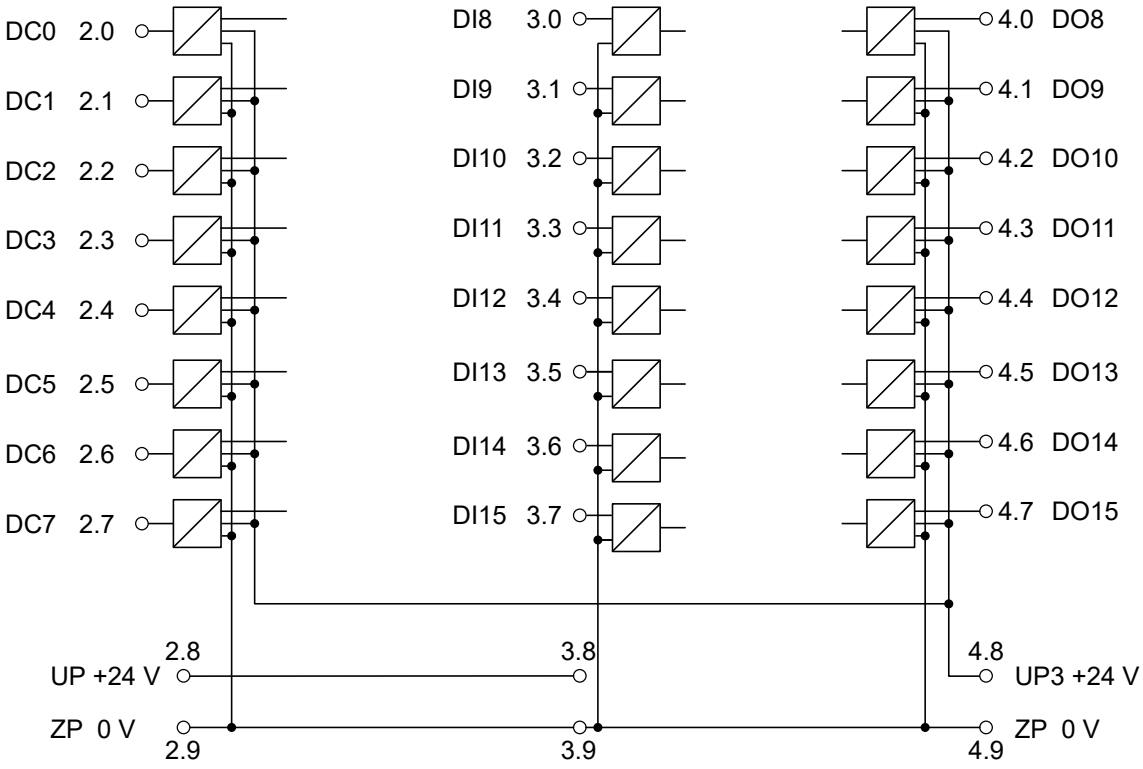
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

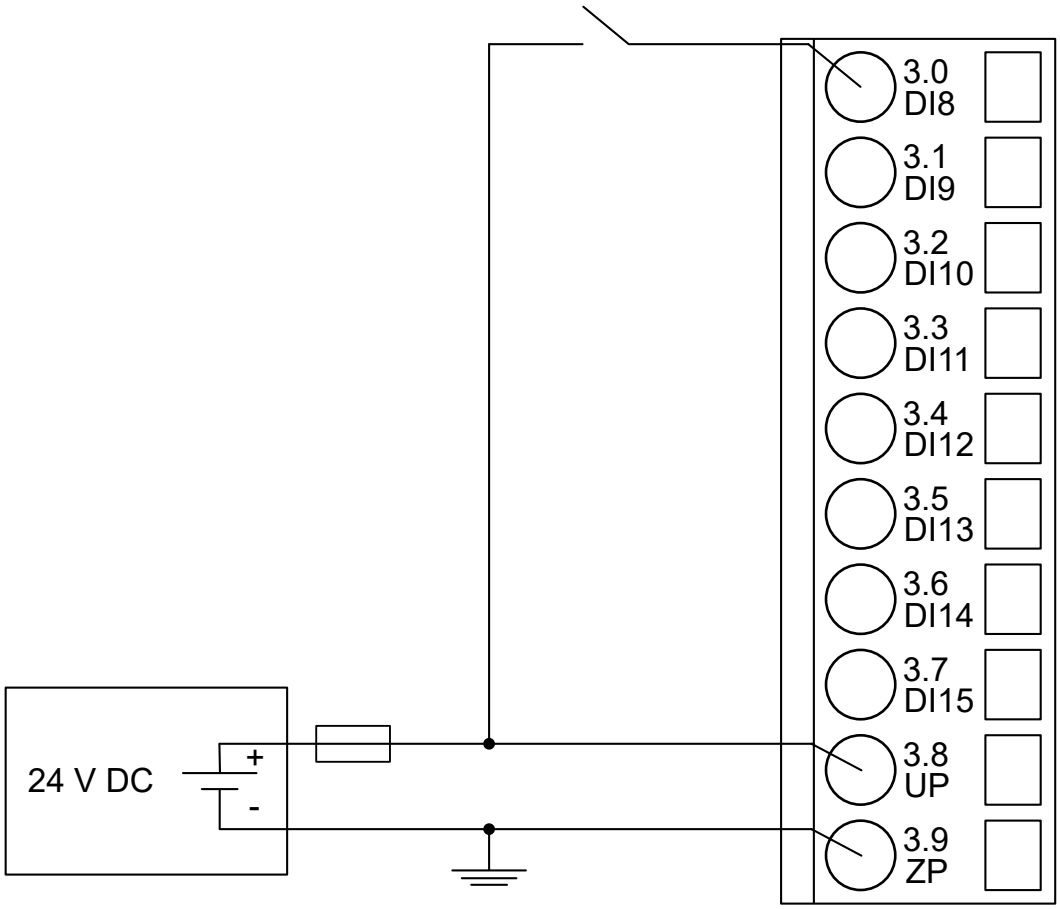
– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

The following figures show the connection of the PROFIBUS DP communication interface module CI542-DP.



Connection of the digital inputs

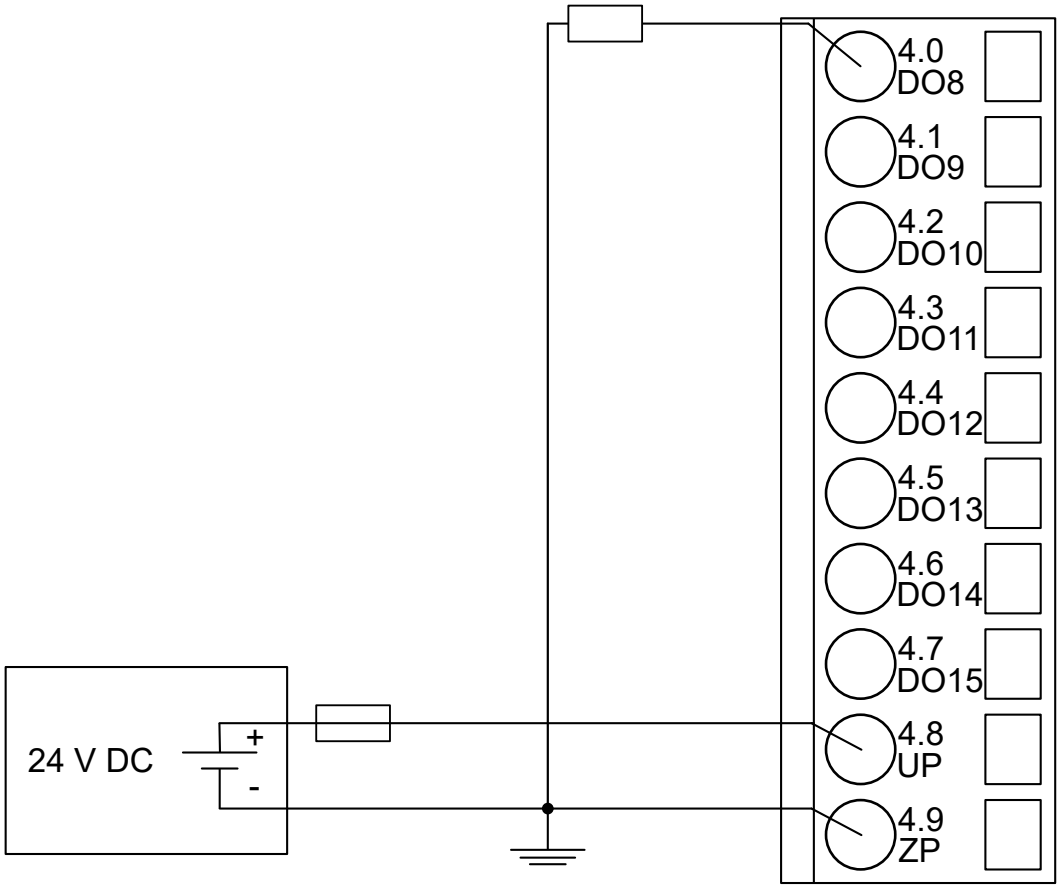
The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.6.2.9 “State LEDs” on page 4986*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

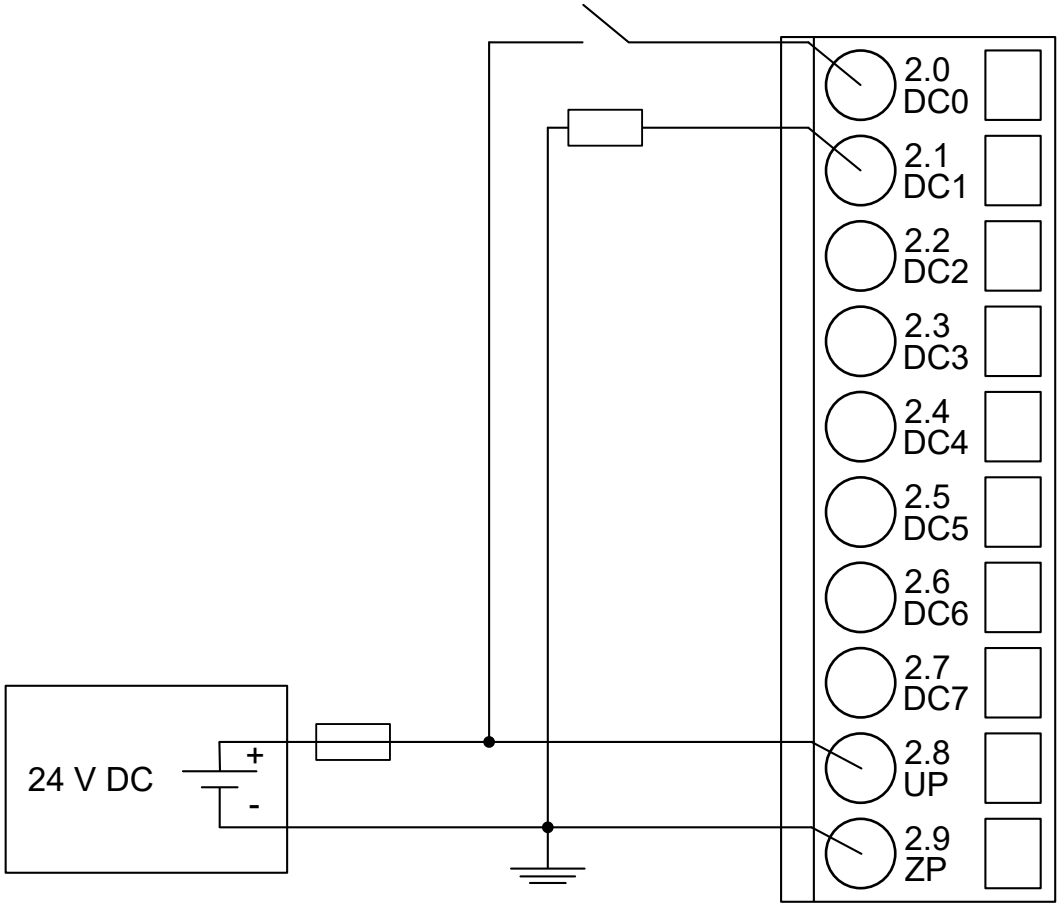


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.6.2.9 “State LEDs” on page 4986*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

CAUTION!
If a DC channel is used as input, the source for the input signals should be the impressed UP3 of the device *Chapter 1.6.2.8.6.2.3 “Connections” on page 4970*.



The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.6.2.9 “State LEDs” on page 4986*.

Internal data exchange

Parameter	Value
Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI542-DP PROFIBUS DP bus configuration is handled by PROFIBUS DP master with the exception of the slave bus ID (via rotary switches) and the transmission rate (automatic detection).

The digital I/O channels and the fast counter are configured via software.

Details about configuration are described in Parameterization.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	0x1C25	WORD	0x1C25
Parameter length	Internal	31	BYTE	31
Reserved (1 byte)	0	0	BYTE	0
Error LED / Failsafe function *Table 551 “Settings "Error LED / Failsafe function"” on page 4979* (see table)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	2		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	18		
Reserved (20 bytes)	0	0	BYTE	0
Check supply	On	0	BYTE	
	Off	1		1
Fast counter	0	0	BYTE	0
	:	:		
	10 [2])	10		

[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.

[2]) Counter operating modes, see 'Fast Counter' *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

Table 551: Settings "Error LED / Failsafe function"

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe mode on *)

Setting	Description
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe mode on *)
*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe mode is ON.	

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...65535	0000h...FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behaviour DO at comm. error is apply to DC and DO channels and only analyzed if the Failsafe-mode is ON.
[2])	The state "externally voltage detected" appears, if the output of a channel DC0..DC7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears, if externally voltage at digital outputs DC0..DC7 and accordingly DO0..DO7 has exceeded the process supply voltage UP3 ↬ *Chapter 1.6.2.8.6.2.3 “Connections” on page 4970*. The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters on "OFF". This parameter should only be disabled in exceptional cases for voltage overflow may produce reverse voltage.

Diagnosis

Structure of the Diagnosis Block via DPM_SLV_DIAG function block ↬ *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*.

Byte Number	Description	Possible Values
1	Data length (header included)	7
2	PROFIBUS DP V1 coding: Vendor specific	129
3	Diagnosis Byte, slot number	31 = CI542-DP (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
4	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
5	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
6	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
7	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
Module errors							
3	-	31	31	31	19	Checksum error in the I/O module	Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module	
3	-	31	31	31	40	Different hard-/firm-ware versions in the module	
3	-	31	31	31	43	Internal error in the module	
3	-	31	31	31	36	Internal data exchange failure	
3	-	31	31	31	9	Overflow diagnosis buffer	Restart
3	-	31	31	31	26	Parameter error	Check master
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	1...10	31	6	42	No communication with hot swap terminal unit [9])	Restart, if error persists replace terminal unit
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage
4	-	31	31	31	46	Reverse voltage from digital outputs DO0..DO7 to UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow at outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	8...15	46	Externally voltage detected at digital output DO0..DO7 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0..DC7 [6])	Check terminals

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 6 **Bit 6...7**	**-**	**Byte 3**	**Byte 4**	**Byte 5**	**Byte 6** **Bit 0...5**	**PROFIBUS DP diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	31	4	0...7	47	Short circuit at digital output DC0..DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO0..DO7[7])	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0 ... 4 or 10 = Position of the Communication Module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI542-DP diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1..10 = expansion module
[3])	With "Module" the following allocation applies dependent of the master: Module error: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DC0..DC7 oder DO0..DO7 cause that other digital outputs are supplied through that voltage. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0..DC7 and accordingly DO0..DO7 has exceeded the process supply voltage UP3 ↬ *Chapter 1.6.2.8.6.2.3 “Connections” on page 4970*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DC0..DC7 or DO0..DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100ms. Then a new start up will be executed. This diagnosis message appears per channel.

[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 DP, STA2 DP, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 552: States of the 5 system LEDs:

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1-DP	Green	---	PROFIBUS running	Invalid device parameters
STA2-DP	Red	No error	Bus timeout	No communication to master
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No communication interface modules connected or communication error	Communication interface module connected and operational	---

Table 553: States of the 29 process LEDs:

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--

LED	Color	OFF	ON	Flashing
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	PROFIBUS interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 2.8 and 3.8 for +24 V (UP) Terminal 4.8 for +24 V (UP3) Terminals 2.9, 3.9 and 4.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the PROFIBUS DP identifier		With 2 rotary switches at the front side of the module
Diagnose		See Diagnosis *Chapter 1.6.2.8.6.2.8 “Diagnosis” on page 4981*

Parameter	Value
Operation and error displays	34 LEDs (totally)
Weight (without terminal unit)	Ca. 125 g
Mounting position	Horizontal Or vertical with derating (output load reduced to 50 % at 40 °C per group)
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads
No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Technical data of the digital inputs

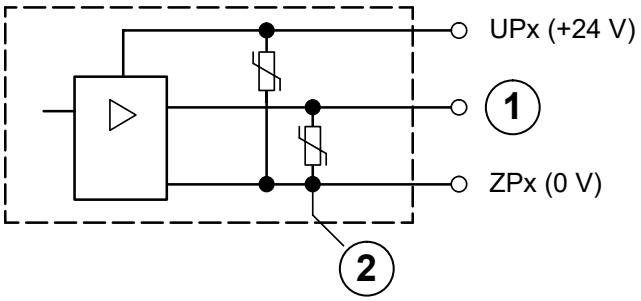
Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 3.0 to 3.7
Reference potential for all inputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA

Parameter		Value
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 4.0 to 4.7
Reference potential for all outputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC07	Terminals 2.0...2.7
If the channels are used as outputs		
	Channels DC0...DC07	Terminals 2.0...2.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the PROFIBUS network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V

Parameter		Value
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

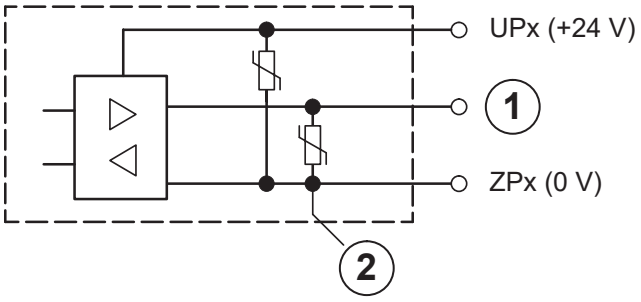
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 2.0 to 2.7
Reference potential for all outputs		Terminals 2.9 ... 4.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 4.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)

Parameter		Value
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 3.0 (DI0),Terminal 3.1 (DI1)
Used outputs	Terminal 4.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1- 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

Ordering Data

Part no.	Description	Product life cycle phase *)
1SAP 224 200 R0001	CI542-DP, PROFIBUS DP communication interface module, 8 DI, 8 DO and 8 DC	Active
1SAP 424 200 R0001	CI542-DP-XC, PROFIBUS DP communication interface module, 8 DI, 8 DO and 8 DC, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.8.7 PROFINET

Comparison of the CI5xx-PNIO modules

The PROFINET IO devices combine the advantages of decentralized I/O modules with the reaction time of AC500 mounted central I/O modules. The devices for PROFINET provide the extension -PNIO in the device name.

The communication module CM579-PNIO acts as I/O controller in a PROFINET network. It is connected to the processor module via an internal communication bus. Depending on the terminal base, several communication modules can be used for one processor module.

The communication interface modules CI5xx-PNIO act as I/O devices in a PROFINET network.

Additionally the communication module CM589-PNIO(-4) can be used to setup a AC500 PLC to act as I/O module in a PROFINET network.

The difference of the CI5xx-PNIO devices can be found in their input and output characteristics ↳ *Chapter 1.6.2.8.7.1.1.1 “Characteristics of CI50x-PNIO” on page 4993*.

The characteristics for CM589-PNIO(-4) can be found in the device description for CM589-PNIO ↳ *Further information on page 4093*.

PROFINET IO devices CI50x-PNIO

Characteristics of CI50x-PNIO

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms

Parameter	Value
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Input/Output characteristics of CI501-PNIO

The PROFINET communication interface module CI501-PNIO is used as decentralized I/O module in PROFINET networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit. The communication interface module contains 22 I/O channels with the following properties:

- 4 analog inputs (1.0...1.3), configurable as:
  - -10 ... +10 V
  - 0 ... +10 V
  - -10 ... +10 V (differential voltage)
  - 0 ... 20 mA
  - 4 ... 20 mA
  - Pt100 , Pt1000, Ni1000 (for each 2-wire and 3-wire)
  - 24 V digital input function
- 2 analog outputs (1.5...1.6), configurable as:
  - -10 ... +10 V
  - 0 ... 20 mA
  - 4 ... 20 mA
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital transistor outputs 24 V DC (0.5 A max.) in 1 group (3.0...3.7)
- Resolution of the analog channels: 12 bits

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Input/Output characteristics of CI502-PNIO

- 8 digital inputs 24 V DC
- 8 digital transistor outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available

Technical data of the serial interfaces of CI504-PNIO

Parameter	Value
Number of serial interfaces	3
Connectors for serial interfaces	X11 for COM1 X12 for COM2 X13 for COM3
Supported physical layers	RS-232 RS-422 RS-485
Supported protocols	ASCII
Transmission rate	Configurable from 300 bit/s to 115.200 bit/s

Technical data of the serial interfaces of CI506-PNIO

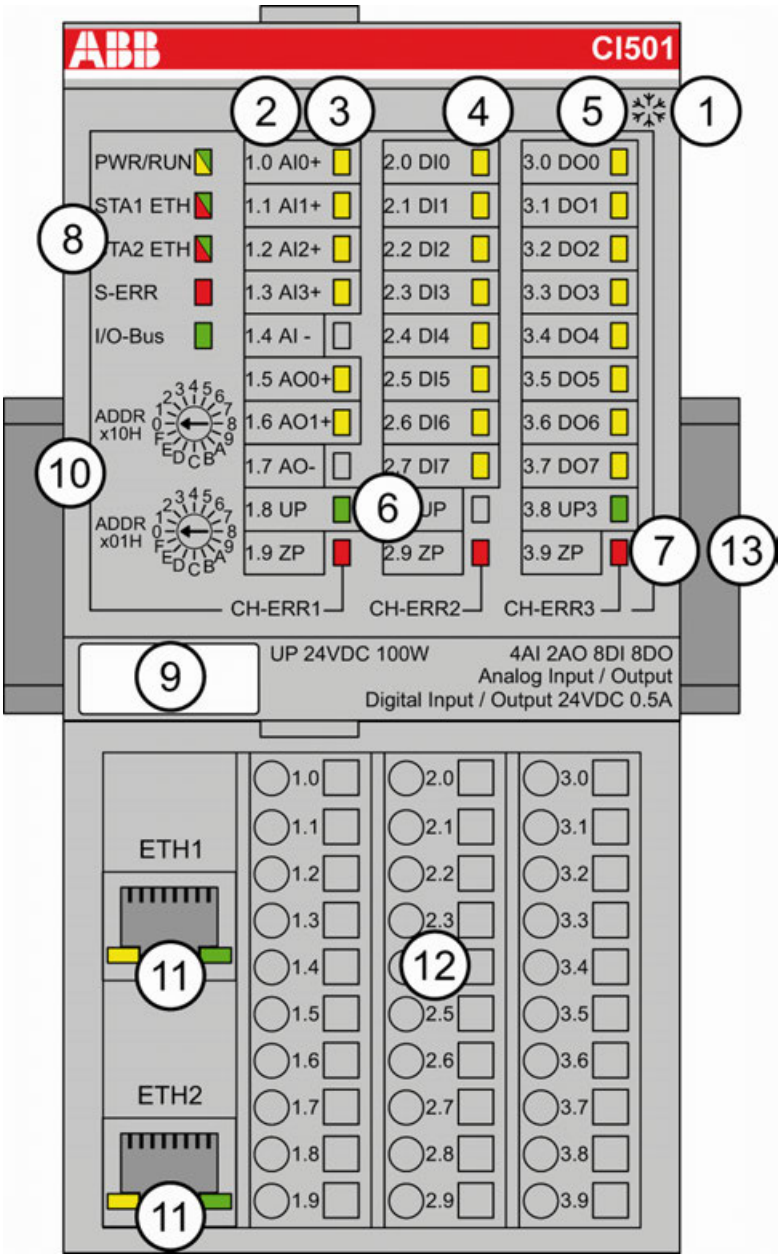
Parameter	Value
Number of serial interfaces	2
Connectors for serial interfaces	X11 for COM1 X12 for COM2
Supported physical layers	RS-232 RS-422 RS-485
Supported protocols	ASCII
Transmission rate	Configurable from 300 bit/s to 115.200 bit/s

Technical data of the CANopen interfaces (CI506-PNIO)

Parameter	Value
Number of CANopen interfaces	1
Connector for CANopen Interface	X13
Transmission rate	Up to 1 Mbit/s

CI501-PNIO

- 4 analog inputs, 2 analog outputs, 8 digital inputs, 8 digital outputs
- Resolution 12 bits plus sign
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 6 yellow LEDs to display the signal states of the analog inputs/outputs (AI0 - AI3, AO0 - AO1)
4 8 yellow LEDs to display the signal states of the digital inputs (DI0 - DI7)
5 8 yellow LEDs to display the signal states of the digital outputs (DO0 - DO7)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the I/O device identifier
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface modules CI501-PNIO and CI502-PNIO are used as communication interface modules in PROFINET networks. The network connection is performed by Ethernet cables which are inserted in the RJ45 connectors in the terminal unit. An Ethernet switch in the communication interface module allows daisy chaining of the network.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

The communication interface module contains 22 I/O channels with the following properties:

- 4 configurable analog inputs (2-wire / single-ended) or 2 configurable analog inputs (3-wire / differential) (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC, 0.5 A max. in 1 group (3.0...3.7)

The inputs/outputs are galvanically isolated from the PROFINET network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Parameter	Value
Interface	Ethernet
Protocol	PROFINET IO RT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the I/O device identifier for configuration purposes (00h to FFh)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507 or TU508 *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet communication interface module CI501-PNIO is plugged on the I/O terminal unit TU507-ETH or TU508-ETH *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↬ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to digital outputs!

Reason: External voltages at an output or several outputs may cause that other outputs are supplied through that voltage instead of voltage UP3 (reverse voltage). This is unintended usage.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is connected at the outputs DO0...DO7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	AI0+	Positive pole of analog input signal 0
1.1	AI1+	Positive pole of analog input signal 1
1.2	AI2+	Positive pole of analog input signal 2
1.3	AI3+	Positive pole of analog input signal 3
1.4	AI-	Negative pole of analog input signals 0 to 3
1.5	AO0+	Positive pole of analog output signal 0
1.6	AO1+	Positive pole of analog output signal 1
1.7	AI-	Negative pole of analog output signals 0 and 1
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)
2.0	DI0	Signal of the digital input DI0
2.1	DI1	Signal of the digital input DI1
2.2	DI2	Signal of the digital input DI2
2.3	DI3	Signal of the digital input DI3
2.4	DI4	Signal of the digital input DI4
2.5	DI5	Signal of the digital input DI5

Terminal	Signal	Description
2.6	DI6	Signal of the digital input DI6
2.7	DI7	Signal of the digital input DI7
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO0	Signal of the digital output DO0
3.1	DO1	Signal of the digital output DO1
3.2	DO2	Signal of the digital output DO2
3.3	DO3	Signal of the digital output DO3
3.4	DO4	Signal of the digital output DO4
3.5	DO5	Signal of the digital output DO5
3.6	DO6	Signal of the digital output DO6
3.7	DO7	Signal of the digital output DO7
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

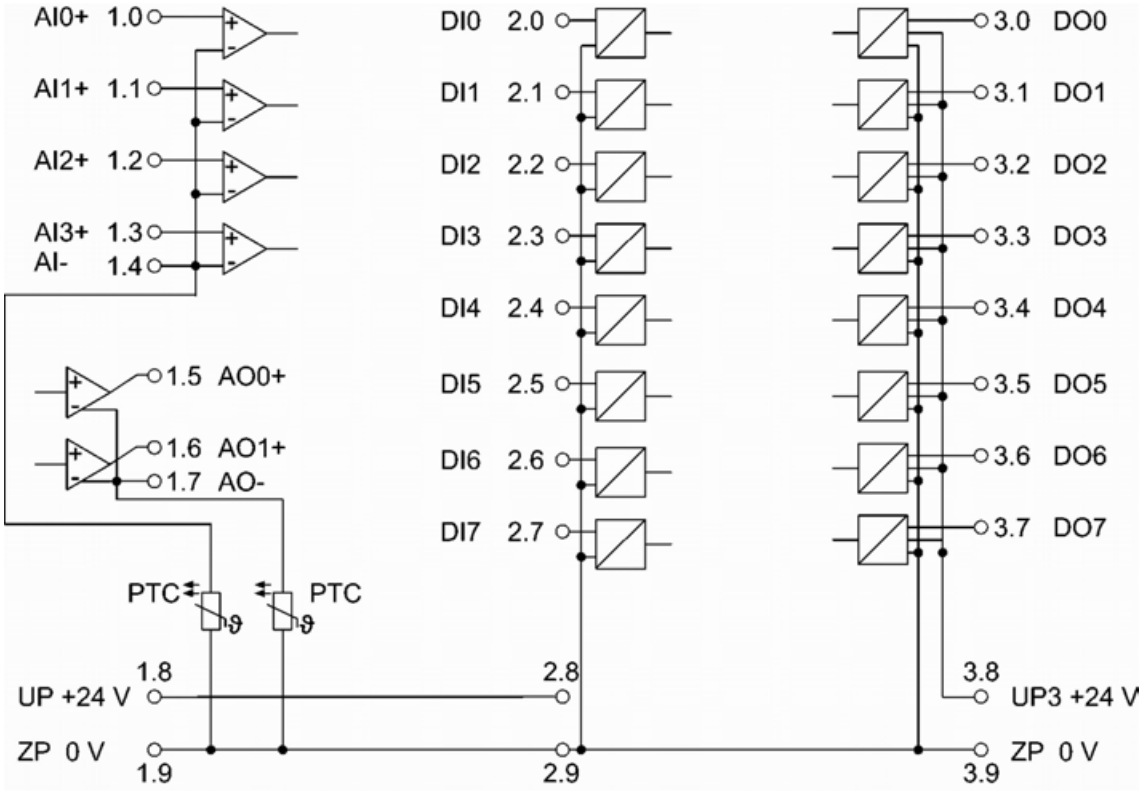
– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

For the open-circuit detection (cut wire), each analog input channel is pulled up to "plus" by a high-resistance resistor. If nothing is connected, the maximum voltage will be read in then.

Generally, analog signals must be laid in shielded cables. The cable shields must be grounded at both sides of the cables. In order to avoid unacceptable potential differences between different parts of the installation, low resistance equipotential bonding conductors must be laid.

Only for simple applications (low electromagnetic disturbances, no high requirement on precision), the shielding can also be omitted.

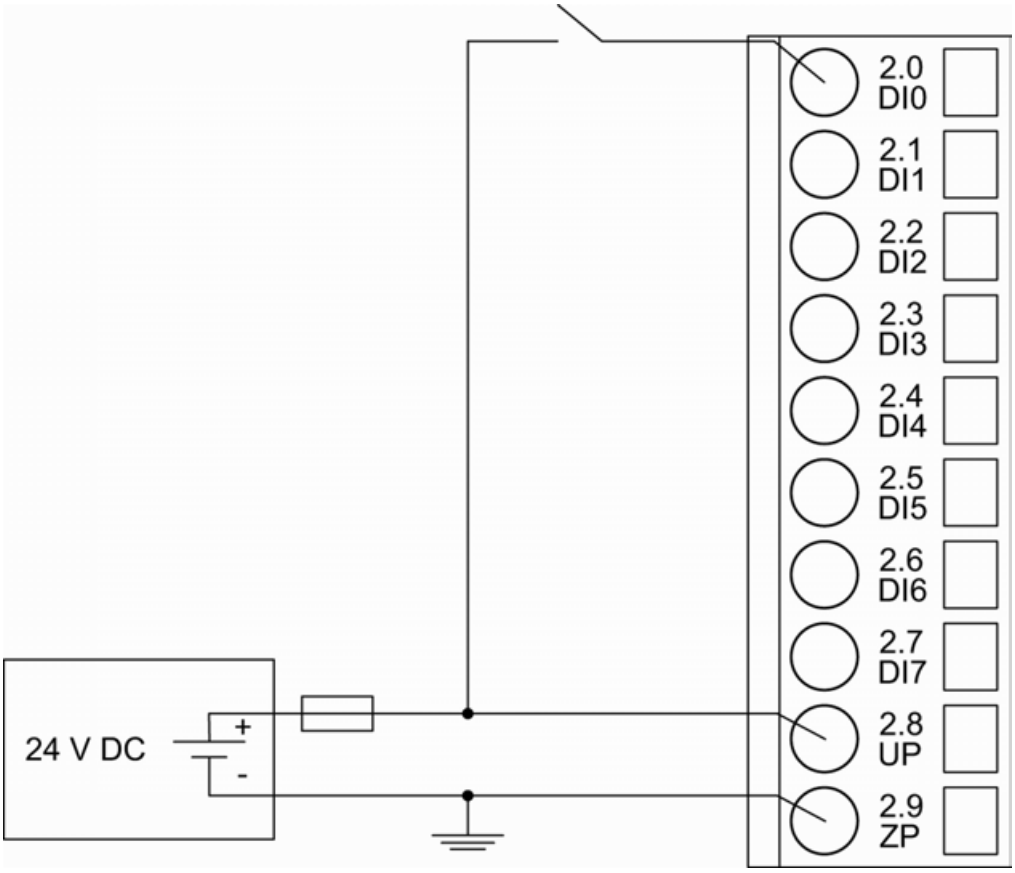
The following figures show the connection of the Ethernet bus module CI501-PNIO.



Further information is provided in the System Technology chapter *Chapter 1.6.4.3.3 “PROFINET communication interface module” on page 5681*.

Connection of the digital inputs

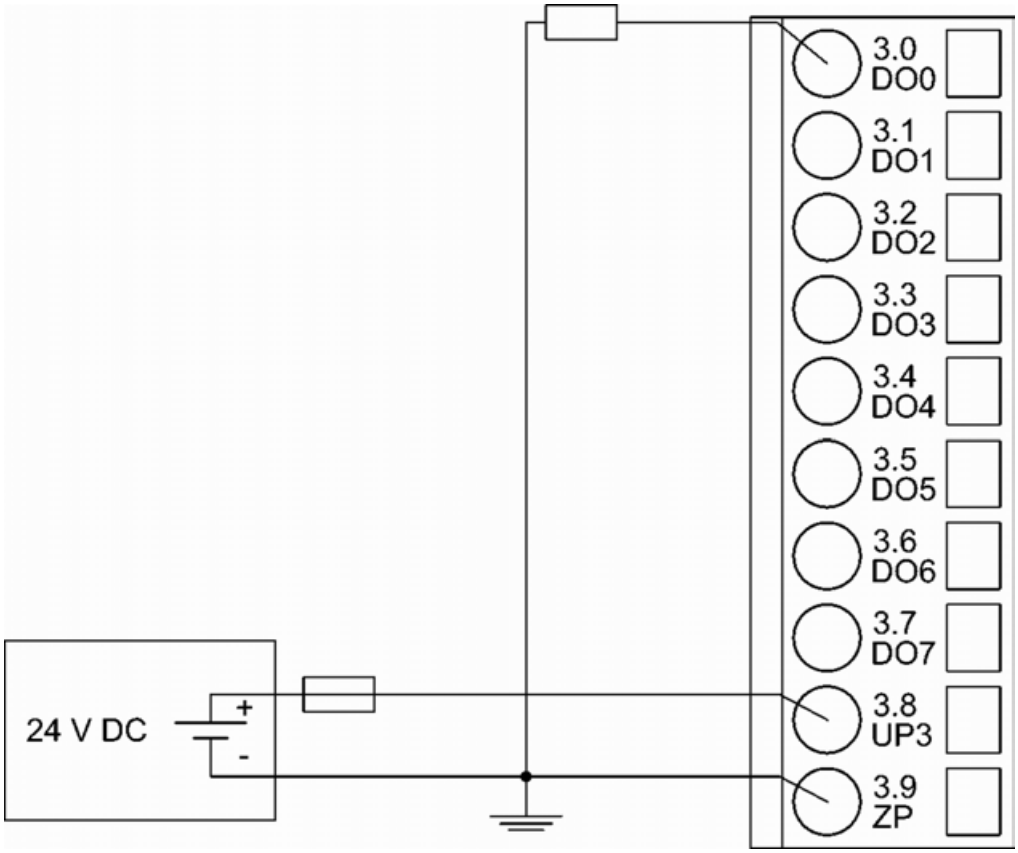
The following figure shows the connection of the digital input DI0. Proceed with the digital inputs DI1 to DI7 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.7.2.8.2 “State LEDs” on page 5025*.

Connection of the digital outputs

The following figure shows the connection of the digital output DO0. Proceed with the digital outputs DO1 - DO7 in the same way.

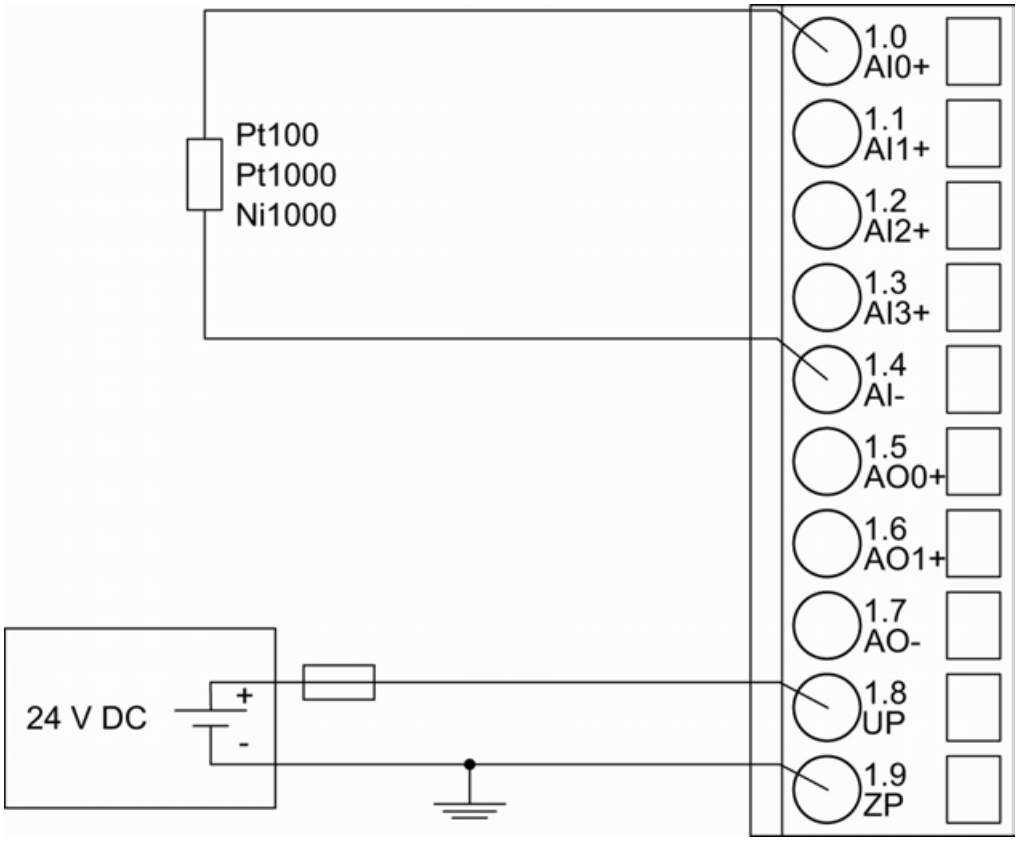


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.7.2.8.2 “State LEDs” on page 5025*.

Connection of resistance thermometers in 2-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI501-PNIO provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 2-wire configuration to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*:

Pt100	-50 °C...+400 °C	2-wire configuration, 1 channel used
Pt1000	-50 °C...+400 °C	2-wire configuration, 1 channel used
Ni1000	-50 °C...+150 °C	2-wire configuration, 1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

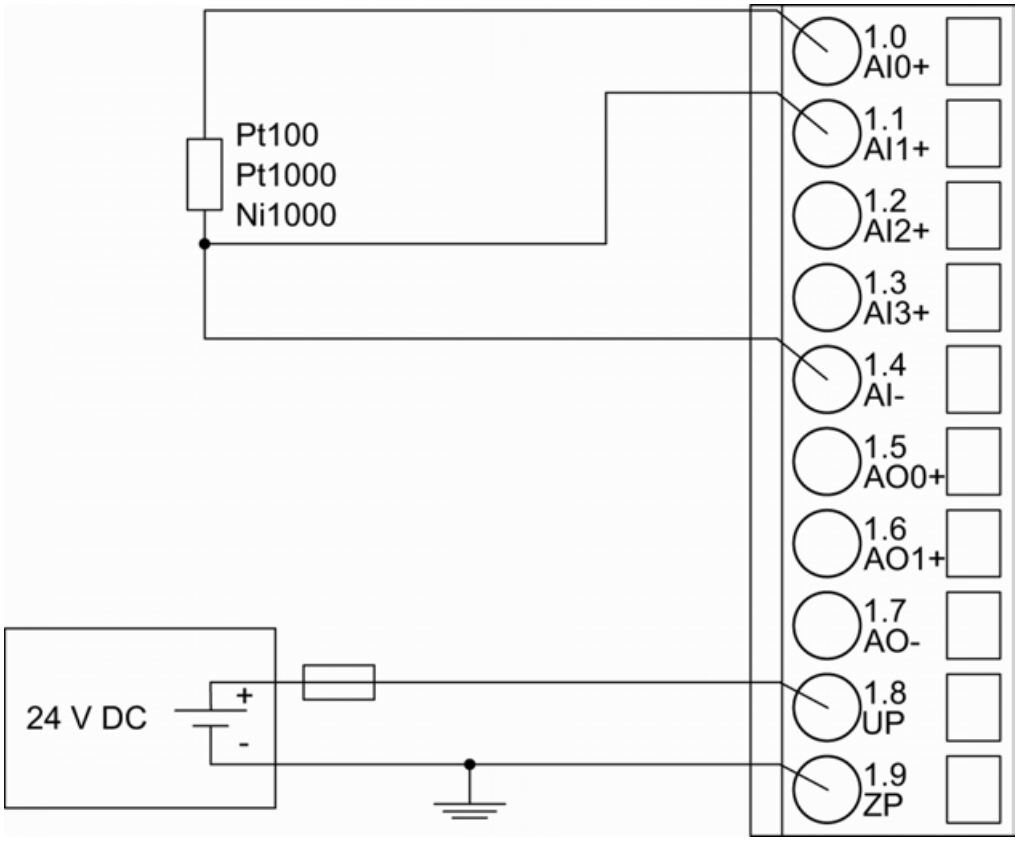
The module CI501-PNIO performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of resistance thermometers in 3-wire configuration to the analog inputs

When resistance thermometers (Pt100, Pt1000, Ni1000) are used, a constant current must flow through them to build the necessary voltage drop for the evaluation. For this, the module CI501-PNIO provides a constant current source which is multiplexed over the max. 4 analog input channels.

The following figure shows the connection of resistance thermometers in 3-wire configuration to the analog inputs AI0 and AI1. Proceed with the analog inputs AI2 and AI3 in the same way.



With 3-wire configuration, 2 adjacent analog channels belong together (e. g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1).

The constant current of one channel flows through the resistance thermometer. The constant current of the other channel flows through one of the cores. The module calculates the measured value from the two voltage drops and stores it under the input with the higher channel number (e. g. I1).

In order to keep measuring errors as small as possible, it is necessary to have all the involved conductors in the same cable. All the conductors must have the same cross section.

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*:

Pt100	-50 °C...+70 °C	3-wire configuration, 2 channels used
Pt100	-50 °C...+400 °C	3-wire configuration, 2 channels used
Pt1000	-50 °C...+400 °C	3-wire configuration, 2 channels used
Ni1000	-50 °C...+150 °C	3-wire configuration, 2 channels used

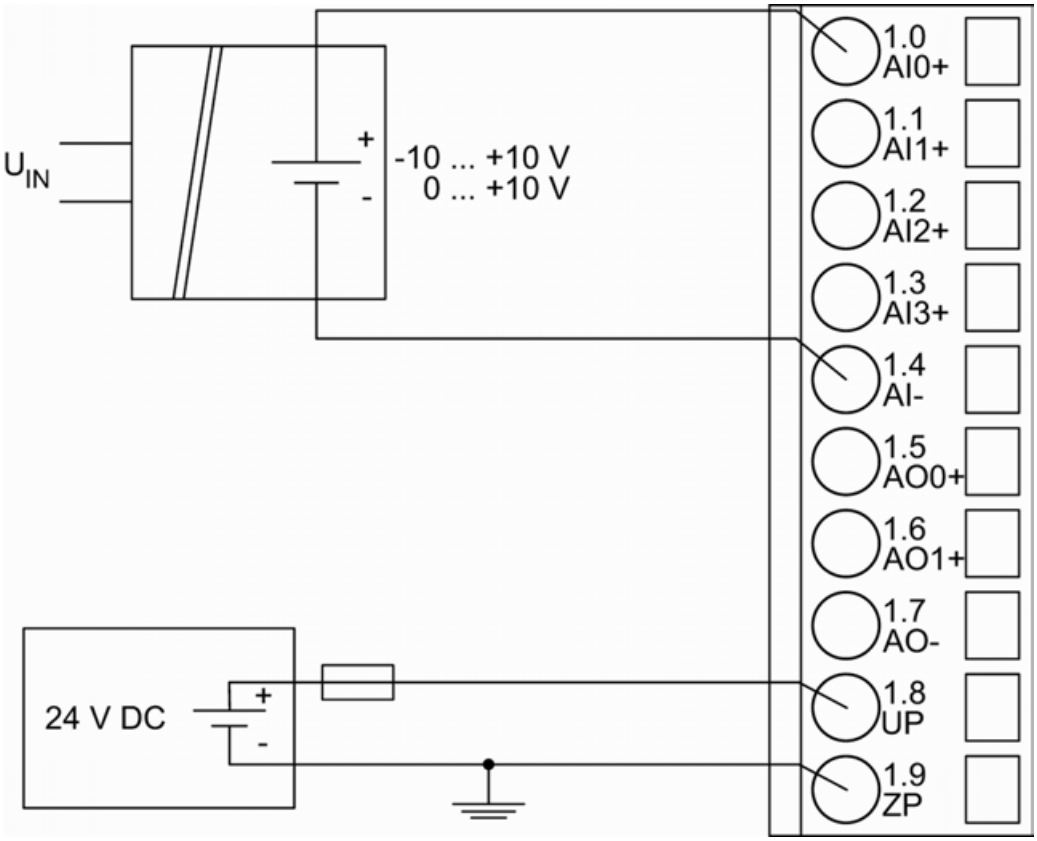
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

The module CI501-PNIO performs a linearization of the resistance characteristic.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Voltage) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*:

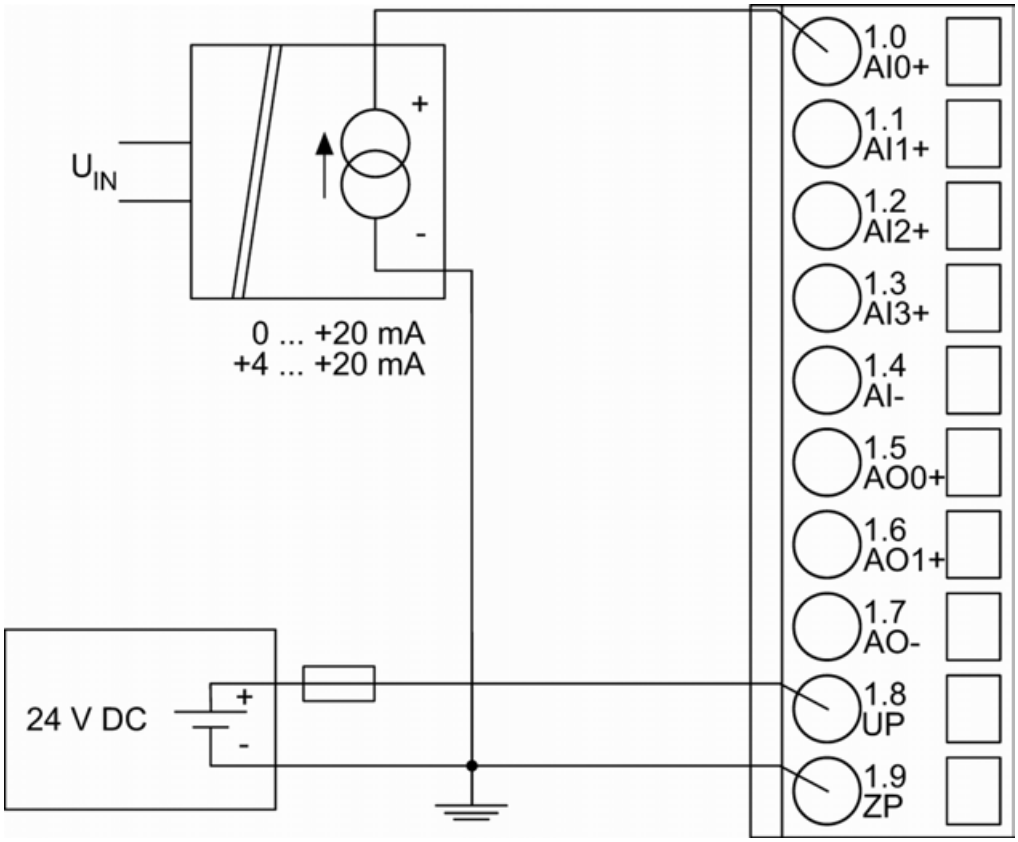
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of active-type analog sensors (Current) with galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (current) with galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 "Parameterization" on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 "Input ranges voltage, current and digital input" on page 5027*:

Current	0 mA...20 mA	1 channel used
Current	4 mA...20 mA	1 channel used

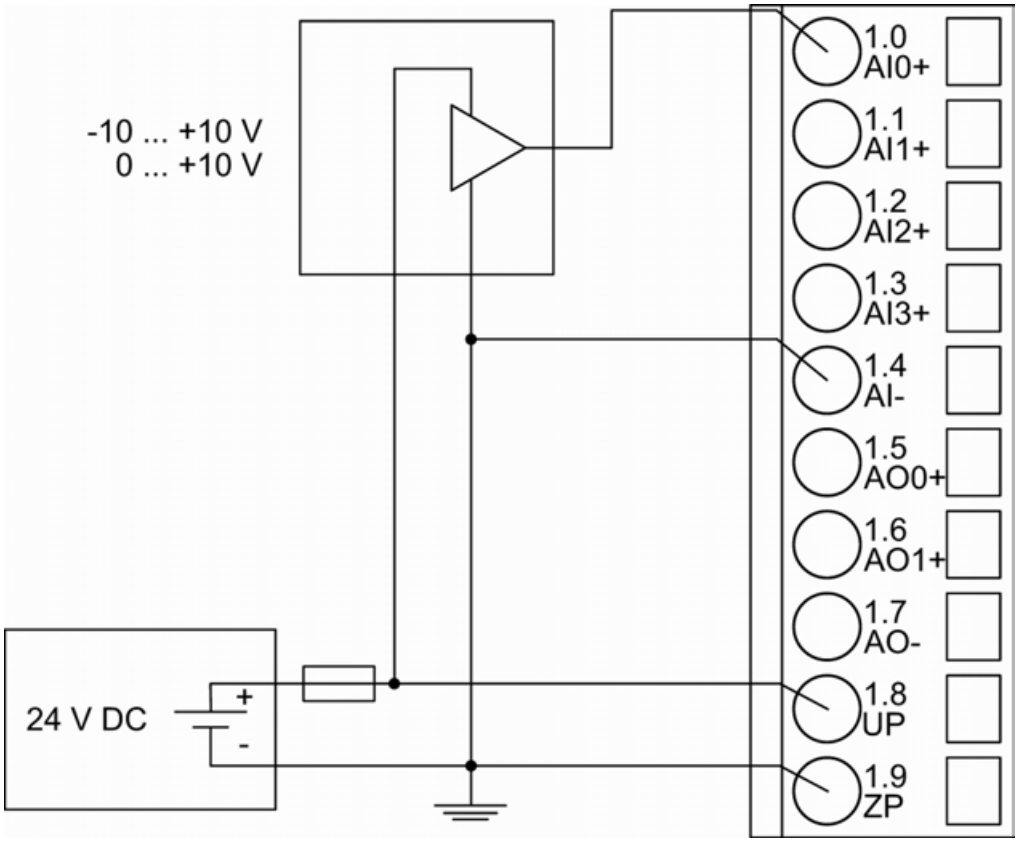
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 "Diagnosis and state LEDs" on page 5020*.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4 mA...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) with no galvanically isolated power supply to the analog inputs

The following figure shows the connection of active-type analog sensors (voltage) with no galvanically isolated power supply to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V (also not with long cable lengths).

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014*:

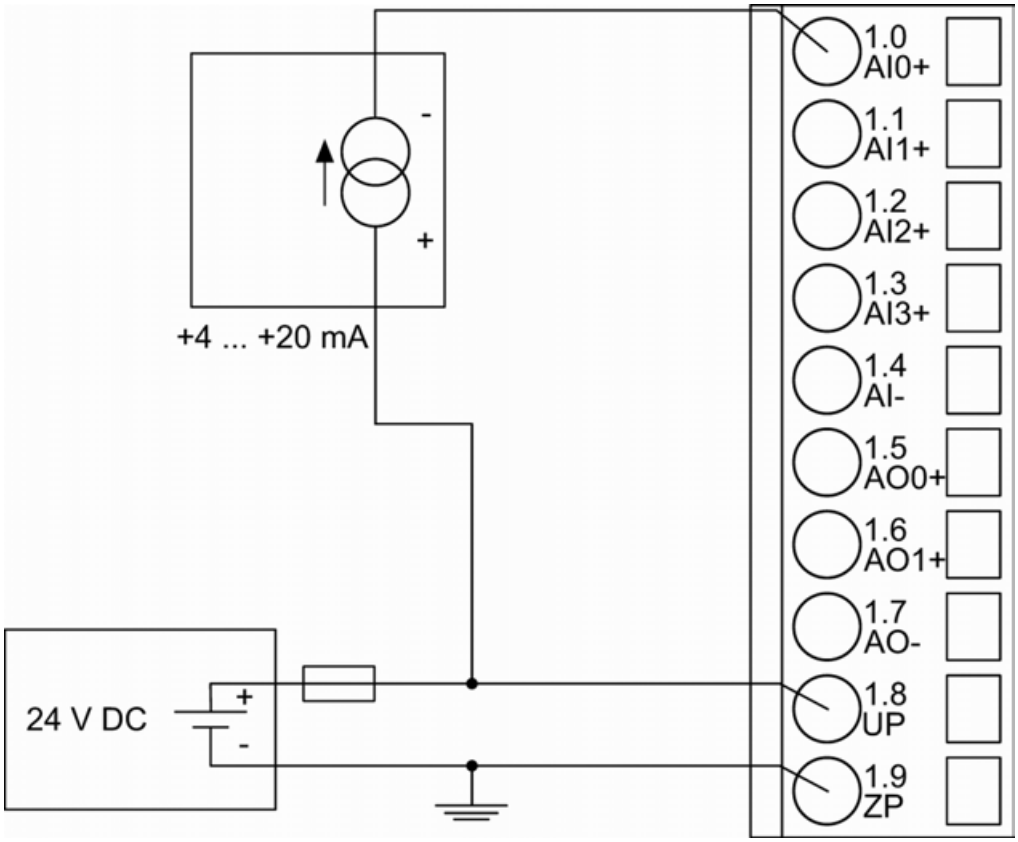
Voltage	0 V...10 V	1 channel used
Voltage	-10 V...+10 V	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

To avoid error messages from unused analog input channels, configure them as "unused".

Connection of passive-type analog sensors (Current) to the analog inputs

The following figure shows the connection of passive-type analog sensors (current) to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*:

Current	4 mA...20 mA	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

CAUTION!

Risk of overloading the analog input!

If an analog current sensor supplies more than 25 mA for more than 1 second during initialization, this input is switched off by the module (input protection).

Use only sensors with fast initialization or without current peaks higher than 25 mA. If not possible, connect a 10-volt zener diode in parallel to AIx+ and ZP.

Unused input channels can be left open-circuited, because they are of low resistance.

To avoid error messages through unused analog input channels in measuring range 4 mA...20 mA, these channels should be configured as "Not used".

Connection of active-type analog sensors (Voltage) to differential analog inputs

Differential inputs are very useful, if analog sensors are used which are remotely non-isolated (e.g. the minus terminal is remotely grounded).

The evaluation using differential inputs helps to considerably increase the measuring accuracy and to avoid ground loops.

With differential input configurations, two adjacent analog channels belong together (e.g. the channels 0 and 1). In this case, both channels are configured according to the desired operating mode. The lower address must be the even address (channel 0), the next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).

The analog value is calculated by subtraction of the input value with the higher address from the input value of the lower address.

The converted analog value is available at the odd channel (higher address).

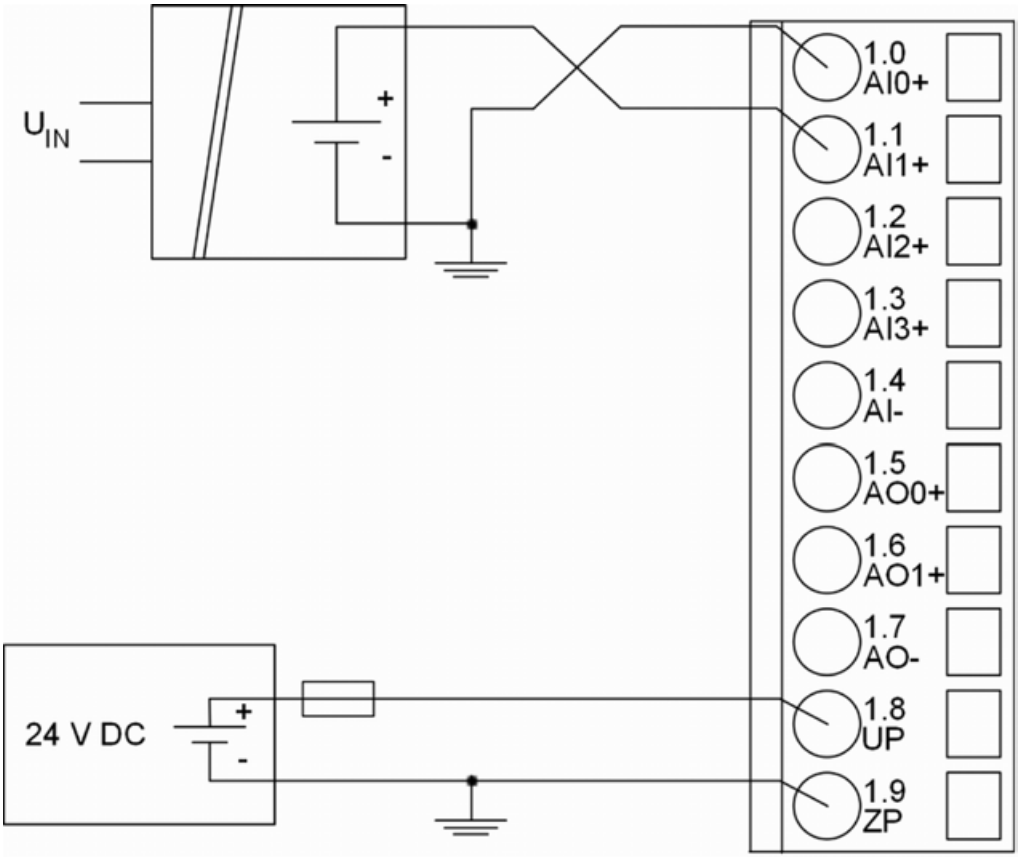
CAUTION!

Risk of faulty measurements!

The negative pole at the sensors must not have too big a potential difference with respect to ZP (max. ±1 V).

Make sure that the potential difference never exceeds ±1 V.

The following figure shows the connection of active-type analog sensors (voltage) to differential analog inputs AI0 and AI1. Proceed with AI2 and AI3 in the same way.



The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*:

Voltage	0 V...10 V	With differential inputs, 2 channels used
Voltage	-10 V...+10 V	With differential inputs, 2 channels used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

To avoid error messages from unused analog input channels, configure them as "unused".

Use of analog inputs as digital inputs

Several (or all) analog inputs can be configured as digital inputs. The inputs are not galvanically isolated against the other analog channels.

The following figure shows the connection of digital sensors to the analog input AI0. Proceed with the analog inputs AI1 to AI3 in the same way.

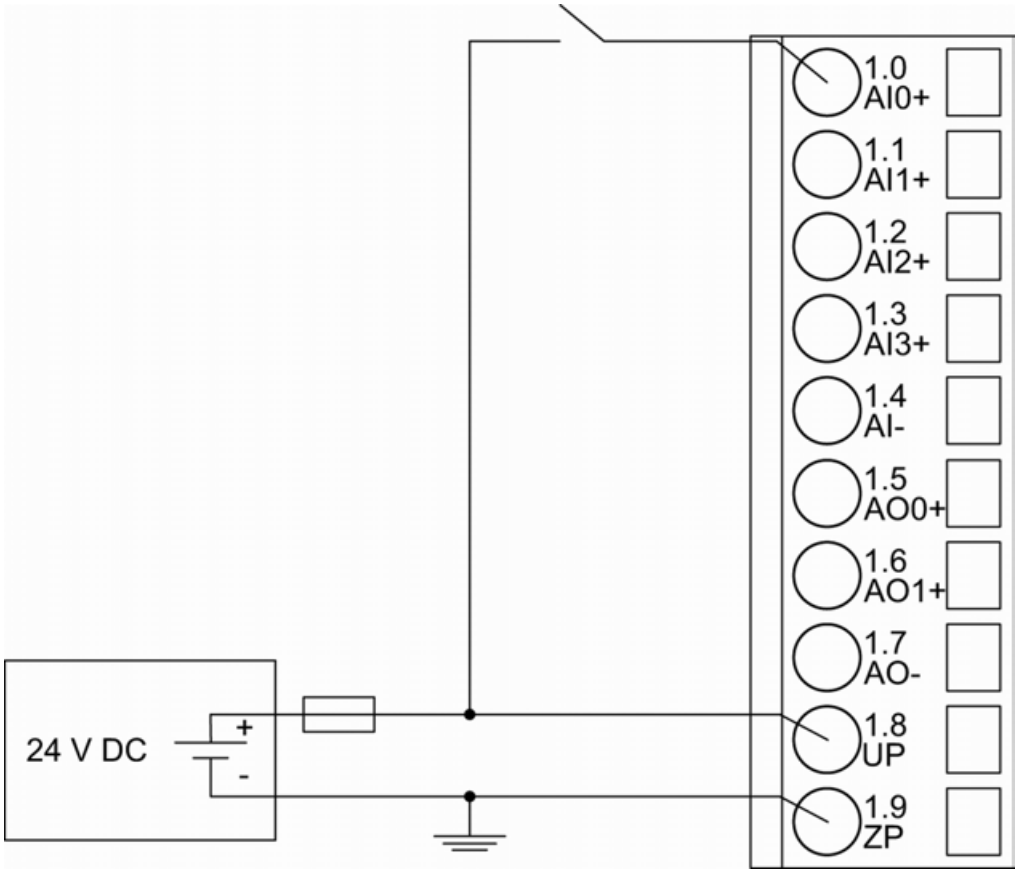


Fig. 1005: Use of analog inputs as digital inputs

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027* :

Digital input	24 V	1 channel used
Effect of incorrect input terminal connection		Wrong or no signal detected, no damage up to 35 V

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

Connection of analog output loads (Voltage)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

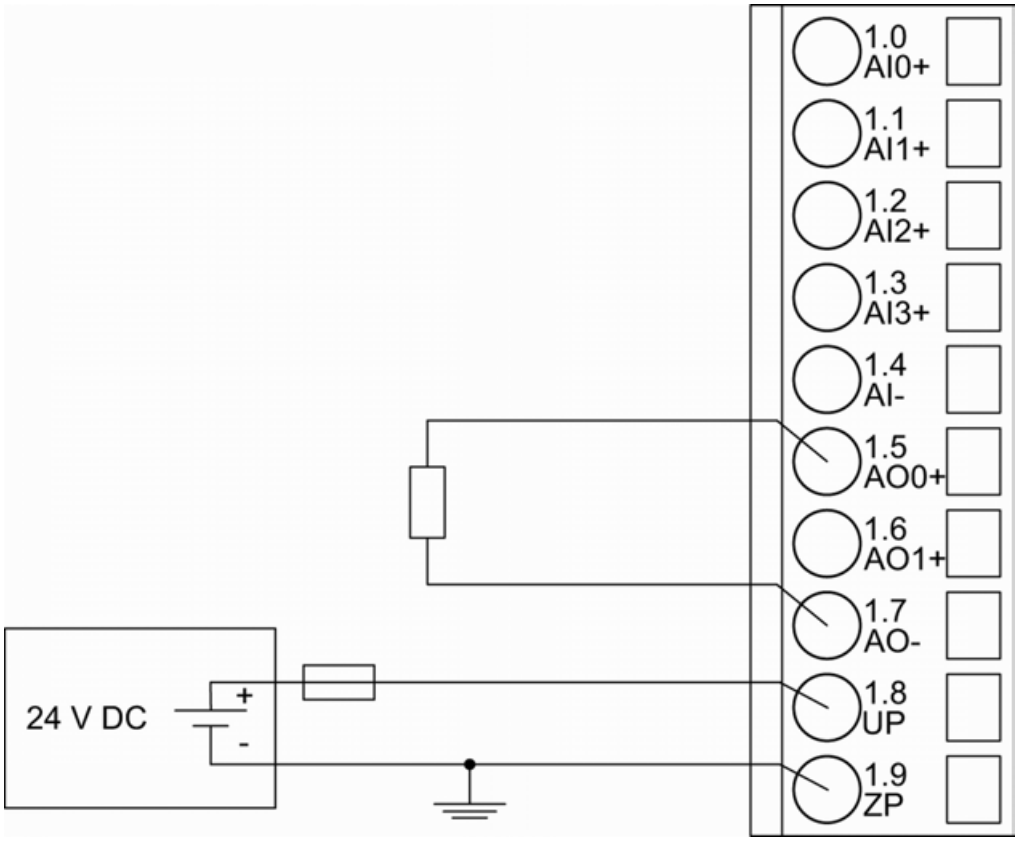


Fig. 1006: Connection of analog output loads (voltage)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*

Voltage	-10 V...+10 V	Load ±10 mA max.	1 channel used

The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*.

Unused analog outputs can be left open-circuited.

Connection of analog output loads (Current)

The following figure shows the connection of output loads to the analog output AO0. Proceed with the analog output AO1 in the same way.

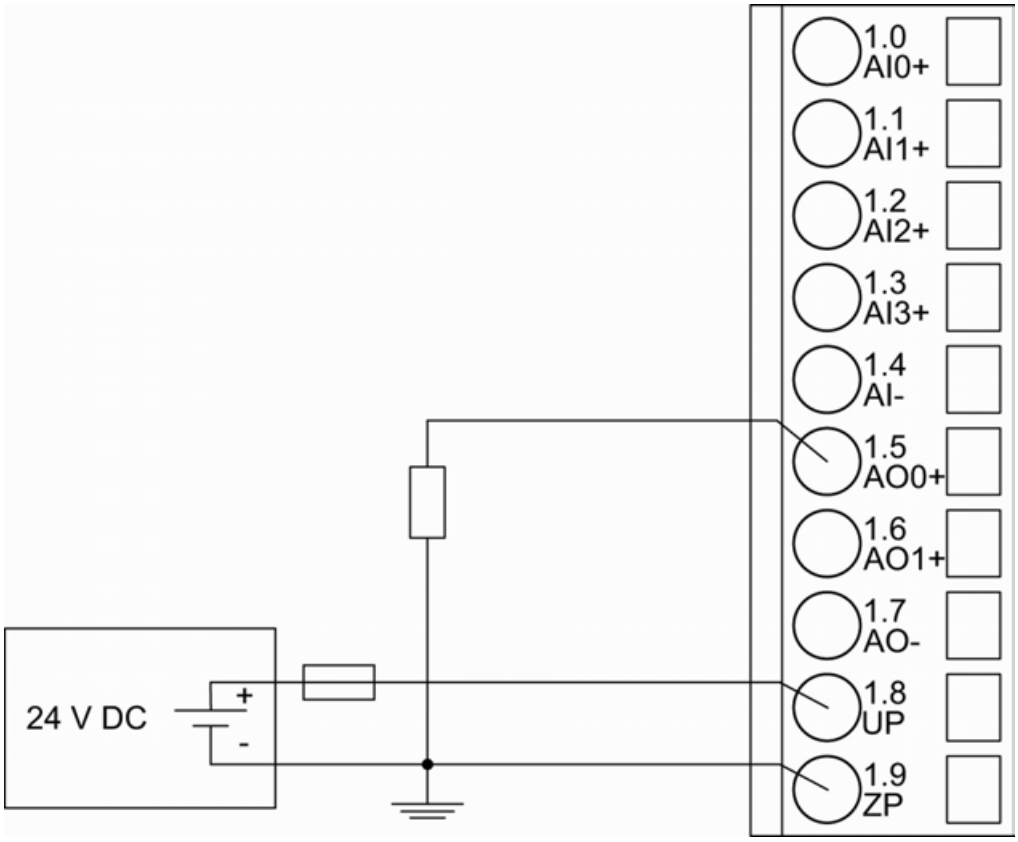


Fig. 1007: Connection of analog output loads (current)

The following measuring ranges can be configured ↳ *Chapter 1.6.2.8.7.2.7 "Parameterization" on page 5014* ↳ *Chapter 1.6.2.8.7.2.9.1 "Input ranges voltage, current and digital input" on page 5027*:

Current	0 mA...20 mA	Load 0 Ω...500 Ω	1 channel used
Current	4 mA...20 mA	Load 0 Ω...500 Ω	1 channel used

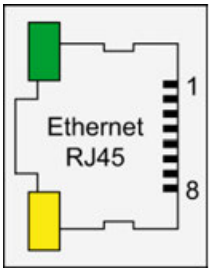
The function of the LEDs is described under Diagnosis and displays / Displays ↳ *Chapter 1.6.2.8.7.2.8 "Diagnosis and state LEDs" on page 5020*.

Unused analog outputs can be left open-circuited.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected

Interface	PIN	Signal	Description
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet ↬ *Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	3
Digital outputs (bytes)	3
Analog inputs (words)	4
Analog outputs (words)	2
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI501-PNIO stores some PROFINET configuration parameters (I/O device identifier, I/O device type and IP address configuration). No more configuration data is stored.

The analog/digital I/O channels are configured via software.

Details about configuration are described in Parameterization ↬ *Chapter 1.6.2.8.7.2.7 “Parameterization” on page 5014*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7000	WORD	7000
Parameter length	Internal	25	BYTE	25
Error LED / Failsafe function see table Error LED / Failsafe function *Table 554 "Error LED / Failsafe function" on page 5015*	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Process cycle time [2])	1 ms process cycle time	1	BYTE	1 ms
	2 ms process cycle time	2		
	3 ms process cycle time	3		
	4 ms process cycle time	4		
	5 ms process cycle time	5		
	6 ms process cycle time	6		
	7 ms process cycle time	7		
	8 ms process cycle time	8		
	9 ms process cycle time	9		
	10 ms process cycle time	10		
	11 ms process cycle time	11		
	12 ms process cycle time	12		
	13 ms process cycle time	13		
	14 ms process cycle time	14		
	15 ms process cycle time	15		
	16 ms process cycle time	16		
Check supply	off	0	BYTE	1
	on	1		

Name	Value	Internal value	Internal value, type	Default
Input delay	8 ms	8 ms	BYTE	8 ms
Fast counter	0 : 10 [3])	0 : 10	BYTE	0
Detect short circuit at outputs	On	1	BYTE	On
Behavior digital outputs at comm. error	Off	0	BYTE	Off
Substitute value digital outputs	0	0..255	BYTE	0
Overvoltage behavior on output	Off	0	BYTE	Off
Behavior analog outputs atcomm. error	Off	0	BYTE	Off
I/O-Bus reset	Off	0	BYTE	Off
	On	1	BYTE	Off

Remarks:

[1])	With a faulty ID, the modules reports a "parameter error" and does not perform cyclic process data transmission.
[2])	As for device index C0 the parameter is no longer evaluated.
[3])	Counter operating modes, see description of the Fast counter *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*.

Table 554: Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On +Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameters Behaviour AO at comm. error and Behaviour DO at comm. error are only analyzed if the Failsafe-mode is ON.	

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *"On"* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *"On"* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *"0"* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *"Off"* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *"Off"* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *"0"* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Group parameters for the analog part

Name	Value	Internal value	Internal value, type	Default
Analog data format	Standard	0	BYTE	0
	Reserved	255		
Behaviour AO at comm. error *)	Off	0	BYTE	0
	Last value	1		
	Last value 5 s	6		
	Last value 10 s	11		
	Substitute value	2		
	Substitute value 5 s	7		
	Substitute value 10 s	12		
*) The parameter Behaviour AO at comm. error is only analyzed if the Failsafe-mode is ON.				

Channel parameters for the analog inputs (4x)

Name	Value	Internal value	Internal value, type	Default
Input 0, Channel configuration	Table Operating modes of the analog inputs *Table 555 “Channel configuration” on page 5017*	Table Operating modes of the analog inputs *Table 555 “Channel configuration” on page 5017*	BYTE	0
Input 0, Check channel	Table Channel montoring *Table 556 “Channel monitoring” on page 5018*	Table Channel montoring *Table 556 “Channel monitoring” on page 5018*	BYTE	0
:	:	:	:	:
:	:	:	:	:
Input 3, Channel configuration	Table Operating modes of the analog inputs *Table 555 “Channel configuration” on page 5017*	Table Operating modes of the analog inputs *Table 555 “Channel configuration” on page 5017*	BYTE	0
Input 3, Check channel	Table Channel montoring *Table 556 “Channel monitoring” on page 5018*	Table Channel montoring *Table 556 “Channel monitoring” on page 5018*	BYTE	0

Table 555: Channel configuration

Internal value	Operating modes of the analog inputs, individually configurable
0 (default)	Not used
1	0 V...10 V
2	Digital input
3	0 mA...20 mA
4	4 mA...20 mA
5	-10 V...+10 V
8	2-wire Pt100 -50 °C...+400 °C
9	3-wire Pt100 -50 °C...+400 °C *)
10	0 V...10 V (voltage diff.) *)
11	-10 V...+10 V (voltage diff.) *)
14	2-wire Pt100 -50 °C...+70 °C
15	3-wire Pt100 -50 °C...+70 °C *)
16	2-wire Pt1000 -50 °C...+400 °C
17	3-wire Pt1000 -50 °C...+400 °C *)
18	2-wire Ni1000 -50 °C...+150 °C

Internal value	Operating modes of the analog inputs, individually configurable
19	3-wire Ni1000 -50 °C...+150 °C *)
*) In the operating modes with 3-wire configuration or with differential inputs, two adjacent analog inputs belong together (e.g. the channels 0 and 1). In these cases, both channels are configured in the desired operating mode. The lower address must be the even address (channel 0). The next higher address must be the odd address (channel 1). The converted analog value is available at the higher address (channel 1).	

Table 556: Channel monitoring

Internal Value	Check Channel
0 (default)	Plausib(ility), cut wire, short circuit
3	Not used

Channel parameters for the analog outputs (2x)

Name	Value	Internal value	Internal value, type	Default
Output 0, Channel configuration	Table Operating modes of the analog outputs *Further information on page 5019*	Table Operating modes of the analog outputs *Further information on page 5019*	BYTE	0
Output 0, Check channel	Table Channel monitoring *Table 558 “Channel monitoring” on page 5019*	Table Channel monitoring *Table 558 “Channel monitoring” on page 5019*	BYTE	0
Output 0, Substitute value	Table Substitute value *Table 559 “Substitute value” on page 5019*	Table Substitute value *Table 559 “Substitute value” on page 5019*	WORD	0
Output 1, Channel configuration	Table Operating modes of the analog outputs *Further information on page 5019*	Table Operating modes of the analog outputs *Further information on page 5019*	BYTE	0
Output 1, Check channel	Table Channel monitoring *Table 558 “Channel monitoring” on page 5019*	Table Channel monitoring *Table 558 “Channel monitoring” on page 5019*	BYTE	0
Output 1, Substitute value	Table Substitute value *Table 559 “Substitute value” on page 5019*	Table Substitute value *Table 559 “Substitute value” on page 5019*	WORD	0

Table 557: Channel configuration

Internal value	Operating modes of the analog outputs, individually configurable
0 (default)	Not used
128	-10 V...+10 V
129	0 mA...20 mA
130	4 mA...20 mA

Table 558: Channel monitoring

Internal value	Check channel
0	Plausib(ility), cut wire, short circuit
3	None

Table 559: Substitute value

Intended behavior of output channel when the control system stops	Required setting of the module parameter "Behaviour of outputs in case of a communication error"	Required setting of the channel parameter "Substitute value"
Output OFF	Off	0
Last value infinite	Last value	0
Last value for 5 s and then turn off	Last value 5 sec	0
Last value for 10 s and then turn off	Last value 10 sec	0
Substitute value infinite	Substitute value	Depending on configuration
Substitute value for 5 s and then turn off	Substitute value 5 sec	Depending on configuration
Substitute value for 10 s and then turn off	Substitute value 10 sec	Depending on configuration

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01

Name	Value	Internal value	Internal value, type	Default
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...255	00h...FFh	BYTE	0 0x0000
Detect voltage overflow at outputs [2])	Off On	0 1	BYTE	On 0x01
[1]) The parameters Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON. [2]) The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected *Chapter 1.6.2.8.7.2.3 "Connections" on page 4997*. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".				

Diagnosis and state LEDs

Structure of the diagnosis block via PNIO_DEV_ALARM function block

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI501-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master

Byte Number	Description	Possible Values
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
3	-	31	31	31	11	Process voltage UP too low	Check process supply voltage
3	-	31	31	31	45	No process voltage UP	Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O module	Replace I/O module
3	-	1...10	31	31	32	Wrong I/O module type on socket	Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function	Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])	Plug I/O module, replace I/O module
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap ter-minal unit [9])	Restart, if error persists replace terminal unit
4	-	31	31	31	46	Voltage feedback on activated digital outputs DO0...DO7 on UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	1...6	255	2	0	45	The connected Communication Module has no con-nection to the net-work	Check cabeling
4	-	31	31	31	45	No process voltage UP3	Check process supply voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	10	Voltage overflow on outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	0...7	46	Externally voltage detected at digital output DO0...DO7 [6])	Check terminals
4	-	31	2	0...7	47	Short circuit at digital output [7])	Check terminals
Channel error analog							
4	-	31	1	0...3	48	Analog value overflow or broken wire at an analog input	Check value or check terminals
4	-	31	1	0...3	7	Analog value underflow at an analog input	Check value
4	-	31	1	0...3	47	Short circuit at an analog input	Check terminals
4	-	31	3	0...1	4	Analog value overflow at an analog output	Check output value
4	-	31	3	0...1	7	Analog value underflow at an analog output	Check output value

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the communication module;14 = I/O bus; 31 = Module itself The identifier is not contained in the CI501-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself; 1...10 = Expansion module
[3])	With "Module" the following allocation applies: 31 = Module itself Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DO0...DO7 cause that other digital outputs are supplied through that voltage ↳ *Chapter 1.6.2.8.7.2.3 "Connections" on page 4997*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage on digital outputs DO0...DO7 has overrun the process supply voltage UP3 ↳ *Chapter 1.6.2.8.7.2.3 "Connections" on page 4997*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 100 ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 27 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 560: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	---

LED	Color	OFF	ON	Flashing
	Red	---	---	Device is not configured
STA2 ETH (System LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 561: States of the 27 process LEDs

LED	Color	OFF	ON	Flashing
AI0 to AI3	Yellow	Input is OFF	Input is ON (brightness depends on the value of the analog signal)	--
AO0 to AO1	Yellow	Output is OFF	Output is ON (brightness depends on the value of the analog signal)	--
DI0 to DI7	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO0 to DO7	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Measuring ranges

Input ranges voltage, current and digital input

Range	0...10 V	-10...+10 V	0...20 mA	4...20 mA	Digital input	Digital value	
						Decimal	Hex.
Overflow	>11.7589	>11.7589	>23.5178	>22.8142		32767	7FFF
Measured value too high	11.7589 : 10.0004	11.7589 : 10.0004	23.5178 : 20.0007	22.8142 : 20.0006		32511 : 27649	7EFF : 6C01
Normal range Normal range or measured value too low	10.0000 : 0.0004	10.0000 : 0.0004	20.0000 : 0.0007	20.0000 : 4.0006	: : On	27648 : 1	6C00 : 0001
	0.0000	0.0000	0	4	Off	0	0000
	-0.0004 -1.7593	-0.0004 : : : -10.0000		3.9994 : 0		-1 -4864 -6912 : -27648	FFFF ED00 E500 : 9400
Measured value too low		-10.0004 : -11.7589				-27649 : -32512	93FF : 8100
Under-flow	<0.0000	<-11.7589	<0.0000	<0.0000		-32768	8000

The represented resolution corresponds to 16 bits.

Input ranges resistance temperature detector

Range	Pt100 / Pt1000 -50...+70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
Overflow	> 80.0 °C	> 450.0 °C	> 160.0 °C	32767	7FFF
Measured value too high	80.0 °C	450.0 °C : 400.1 °C		4500 : 4001	1194 : 0FA1
			160.0 °C : 150.1 °C	1600 : 1501	0640 : 05DD
Normal range		400.0 °C : : : 0.1 °C	150.0 °C : : 0.1 °C	800 : 701	0320 : 02BD

Range	Pt100 / Pt1000 -50...+70 °C	Pt100 / Pt1000 -50...400 °C	Ni1000 -50...150 °C	Digital value	
				Decimal	Hex.
		0.0 °C	0.0 °C	4000 1500 700 : 1	0FA0 05DC 02BC : 0001
		-0.1 °C : -50.0 °C	-0.1 °C : -50.0 °C	0	0000
Measured value too low	< -60.0 °C	-50.1 °C : -60.0 °C	-50.1 °C : -60.0 °C	-1 : -500	FFFF : FE0C
Underflow	< -60.0 °C	< -60.0 °C	< -60.0 °C	-501 : -600	FE0B : FDA8

Output ranges voltage and current

Range	-10...+10 V	0...20 mA	4...20 mA	Digital value	
				Decimal	Hex.
Overflow	> 11.7589 V	> 23.5178 mA	> 22.8142 mA	> 32511	> 7EFF
Measured value too high	11.7589 V : 10.0004 V	23.5178 mA : 20.0007 mA	22.8142 mA : 20.0006 mA	32511 : 27649	7EFF : 6C01
Normal range	10.0000 V : 0.0004 V	20.0000 mA : 0.0007 mA	20.0000 mA : 4.0006 mA	27648 : 1	6C00 : 0001
	0.0000 V	0.0000 mA	4.0000 mA	0	0000
	-0.0004 V : -10.0000 V	0 mA : 0 mA	3.9994 mA 0 mA 0 mA	-1 -6912 -27648	FFFF E500 9400
Measured value too low	-10.0004 V : -11.7589 V	0 mA : 0 mA	0 mA : 0 mA	-27649 : -32512	93FF : 8100
Underflow	< -11.7589 V	0 mA	0 mA	< -32512	< 8100

The represented resolution corresponds to 16 bits.

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.2 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of analog inputs		4
Number of analog outputs		2
Input data length		2 bytes
Output data length		2 bytes
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnose		See Diagnosis and Displays *Chapter 1.6.2.8.7.2.8 “Diagnosis and state LEDs” on page 5020*
Operation and error displays		32 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)

Parameter	Value
Extended ambient temperature (XC version)	>60 °C on request
Cooling	The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms

Parameter	Value
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the digital inputs

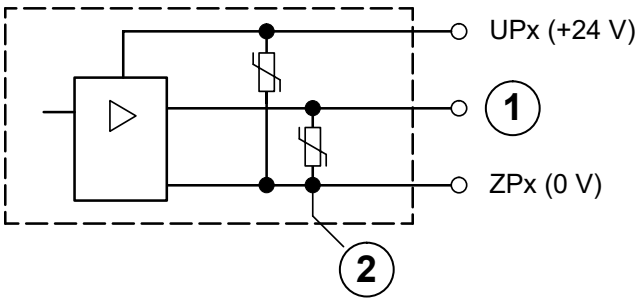
Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DI0 to DI7		Terminals 2.0 to 2.7
Reference potential for all inputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	0-Signal	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	1-Signal	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels

Parameter		Value
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the analog inputs

Parameter		Value
Number of channels per module		4
Distribution of channels into groups		1 group with 4 channels
Connection if channels AI0+ to AI3+		Terminals 1.0 to1.3
Reference potential for AI0+ to AI3+		Terminal 1.4 (AI-) for voltage and RTD measurement Terminal 1.9, 2.9 and 3.9 for current measurement
Input type		
	Unipolar	Voltage 0 V... 10 V, current or Pt100/Pt1000/ Ni1000
	Bipolar	Voltage -10 V... +10 V
Galvanic isolation		Against Ethernet network
Configurability		0 V...10 V, -10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA Pt100/1000, Ni1000 (each input can be configured individually)
Channel input resistance		Voltage: > 100 kΩ Current: ca. 330 Ω
Time constant of the input filter		Voltage: 100 µs Current: 100 µs
Indication of the input signals		1 LED per channel (brightness depends on the value of the analog signal)
Conversion cycle		1 ms (for 4 inputs + 2 outputs); with RTDs Pt/ Ni... 1 s
Resolution		Range 0 V...10 V: 12 bits Range -10 V...+10 V: 12 bits + sign Range 0 mA...20 mA: 12 bits Range 4 mA...20 mA: 12 bits Range RTD (Pt100, PT1000, Ni1000): 0.1 °C
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %
Relationship between input signal and hex code		Tables Input ranges voltage, current and digital input and Input range resistance temperature detector ↳ *Chapter 1.6.2.8.7.2.9.1 “Input ranges voltage, current and digital input” on page 5027*
Unused inputs		Are configured as "unused" (default value)
Overvoltage protection		Yes

Technical data of the analog inputs, if used as digital inputs

Parameter		Value
Number of channels per module		Max. 4
Distribution of channels into groups		1 group of 4 channels
Connections of the channels AI0+ to AI3+		Terminals 1.0 to 1.3
Reference potential for the inputs		Terminals 1.9, 2.9 and 3.9 (ZP)
Indication of the input signals		1 LED per channel
Input signal voltage		24 V DC
	Signal 0	-30 V...+5 V
	Undefined signal	+5 V ... +13 V
	Signal 1	+13 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 7 mA
	Input voltage +5 V	Typ. 1.4 mA
	Input voltage +15 V	Typ. 3.7 mA
	Input voltage +30 V	< 9 mA
Input resistance		Ca. 3.5 kΩ

Technical data of the analog outputs

Parameter		Value
Number of channels per module		2
Distribution of channels into groups		1 group for 2 channels
Connection of the channels AO0+...AO1+		Terminals 1.5...1.6
Reference potential for AO0+ to AO1+		Terminal 1.7 (AO-) for voltage output terminal 1.9, 2.9 and 3.9 for current output
Output type		
	Unipolar	Current
	Bipolar	Voltage
Galvanic isolation		Against internal supply and other modules
Configurability		-10 V...+10 V, 0 mA...20 mA, 4 mA...20 mA (each output can be configured individually)
Output resistance (load), as current output		0 Ω...500 Ω
Output loadability, as voltage output		±10 mA max.
Indication of the output signals		1 LED per channel (brightness depends on the value of the analog signal)
Resolution		12 bits (+ sign)
Settling time for full range change (resistive load, output signal within specified tolerance)		Typ. 5 ms
Conversion error of the analog values caused by non-linearity, adjustment error at factory and resolution within the normal range		Typ. 0.5 %, max. 1 %

Parameter	Value
Relationship between input signal and hex code	Table Output ranges voltage and current ↳ *Chapter 1.6.2.8.7.2.9.3 “Output ranges voltage and current” on page 5028*
Unused outputs	Are configured as "unused" (default value) and can be left open-circuited

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI0), 2.1 (DI1)
Used outputs	Terminal 3.0 (DO0)
Counting frequency	Depending on operation mode: Mode 1 - 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

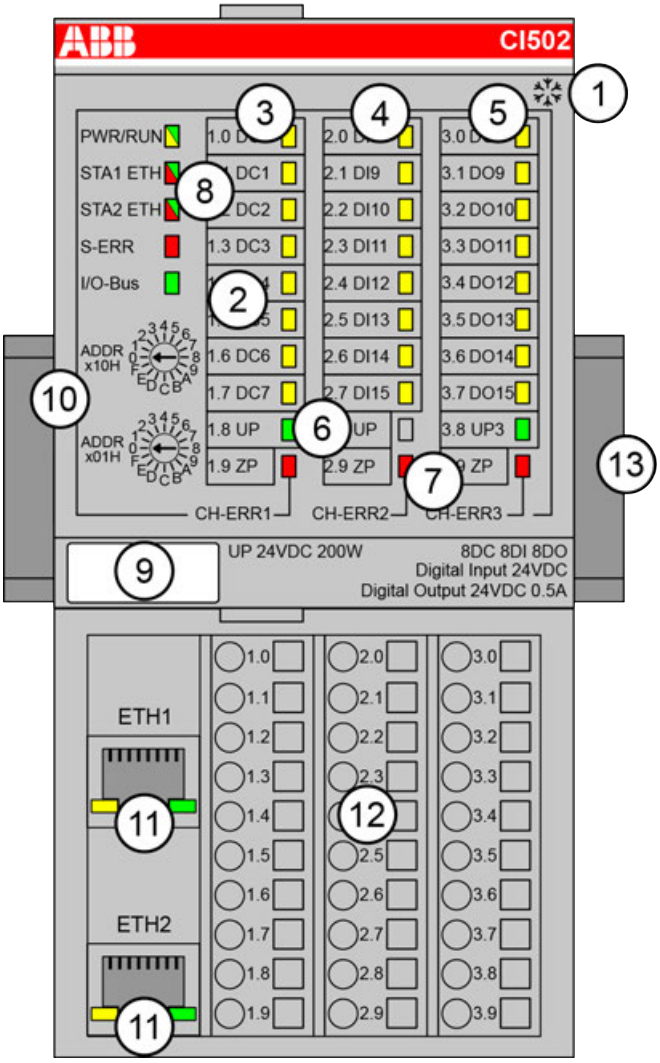
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 220 600 R0001	CI501-PNIO (V3), PROFINET communication interface module, 8 DI, 8 DO, 4 AI and 2 AO	Active
1SAP 420 600 R0001	CI501-PNIO-XC (V3), PROFINET communication interface module, 8 DI, 8 DO, 4 AI and 2 AO, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI502-PNIO

- 8 digital inputs 24 V DC
- 8 digital outputs 24 V DC, 0.5 A max.
- 8 configurable digital inputs/outputs 24 V DC, 0.5 A max.
- Module-wise galvanically isolated
- Fast counter
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 Allocation between terminal number and signal name
3 8 yellow LEDs to display the signal states of the digital configurable inputs/outputs (DC0 - DC7)
4 8 yellow LEDs to display the signal states of the digital inputs (DI8 - DI15)
5 8 yellow LEDs to display the signal states of the digital outputs (DO8 - DO15)
6 2 green LEDs to display the process supply voltage UP and UP3
7 3 red LEDs to display errors (CH-ERR1, CH-ERR2, CH-ERR3)
8 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
9 Label
10 2 rotary switches for setting the I/O device identifier
11 Ethernet interfaces (ETH1, ETH2) on the terminal unit
12 Terminal unit
13 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface module CI502-PNIO is used as communication interface module in PROFINET networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

For usage in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

The CI502 communication interface module contains 24 I/O channels with the following properties:

- 8 digital configurable inputs/outputs
- 8 digital inputs: 24 V DC
- 8 digital outputs: 24 V DC, 0.5 A max.

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels. The configuration of the analog inputs/outputs is performed by software.

Parameter	Value
Interface	Ethernet
Protocol	PROFINET IO RT
Power supply	From the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the IO device identifier for configuration purposes (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU507-ETH or TU508-ETH *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*

Connections

The Ethernet communication interface module CI502-PNIO is plugged on the I/O terminal unit TU507-ETH *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095* or TU508-ETH *Chapter 1.6.2.5.1 “TU507-ETH and TU508-ETH for Ethernet communication interface modules” on page 4095*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the I/O channels is carried out using the 30 terminals of the I/O terminal unit. I/O modules can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The terminals 1.8 and 2.8 as well as 1.9, 2.9 and 3.9 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Terminals 1.8 and 2.8: Process supply voltage UP = +24 V DC

Terminal 3.8: Process supply voltage UP3 = +24 V DC

Terminals 1.9, 2.9 and 3.9: Process supply voltage ZP = 0 V.

The assignment of the other terminals:

With a separate UP3 power supply, the digital outputs can be switched off externally. This way, an emergency-off functionality can be realized.

Do not connect any voltages externally to digital outputs!

This ist not intended usage.

Reason: Externally voltages at one or more terminals DC0..DC7 or DO0..DO7 may cause that other digital outputs are supplied through that voltage instead of voltage UP3 (reverse voltage).

This is also possible, if DC channels are used as inputs. For this, the source for the input signals should be the impressed UP3 of the device.

This limitation does not apply for the input channels DI0..DI7.

CAUTION!

Risk of malfunction by unintended usage!

If the function cut-off of the digital outputs is to be used by deactivation of the supply voltage UP3, be sure that no external voltage is conncted at the outputs DO0...DO7 and DC0...DC7.

The assignment of the other terminals:

Terminal	Signal	Description
1.0	DC0	Signal of the configurable digital input/output DC0
1.1	DC1	Signal of the configurable digital input/output DC1
1.2	DC2	Signal of the configurable digital input/output DC2
1.3	DC3	Signal of the configurable digital input/output DC3
1.4	DC4	Signal of the configurable digital input/output DC4
1.5	DC5	Signal of the configurable digital input/output DC5
1.6	DC6	Signal of the configurable digital input/output DC6
1.7	DC7	Signal of the configurable digital input/output DC7
1.8	UP	Process voltage UP (24 V DC)
1.9	ZP	Process voltage ZP (0 V DC)

Terminal	Signal	Description
2.0	DI8	Signal of the digital input DI8
2.1	DI9	Signal of the digital input DI9
2.2	DI10	Signal of the digital input DI10
2.3	DI11	Signal of the digital input DI11
2.4	DI12	Signal of the digital input DI12
2.5	DI13	Signal of the digital input DI13
2.6	DI14	Signal of the digital input DI14
2.7	DI15	Signal of the digital input DI15
2.8	UP	Process voltage UP (24 V DC)
2.9	ZP	Process voltage ZP (0 V DC)
3.0	DO8	Signal of the digital output DO8
3.1	DO9	Signal of the digital output DO9
3.2	DO10	Signal of the digital output DO10
3.3	DO11	Signal of the digital output DO11
3.4	DO12	Signal of the digital output DO12
3.5	DO13	Signal of the digital output DO13
3.6	DO14	Signal of the digital output DO14
3.7	DO15	Signal of the digital output DO15
3.8	UP3	Process voltage UP3 (24 V DC)
3.9	ZP	Process voltage ZP (0 V DC)

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

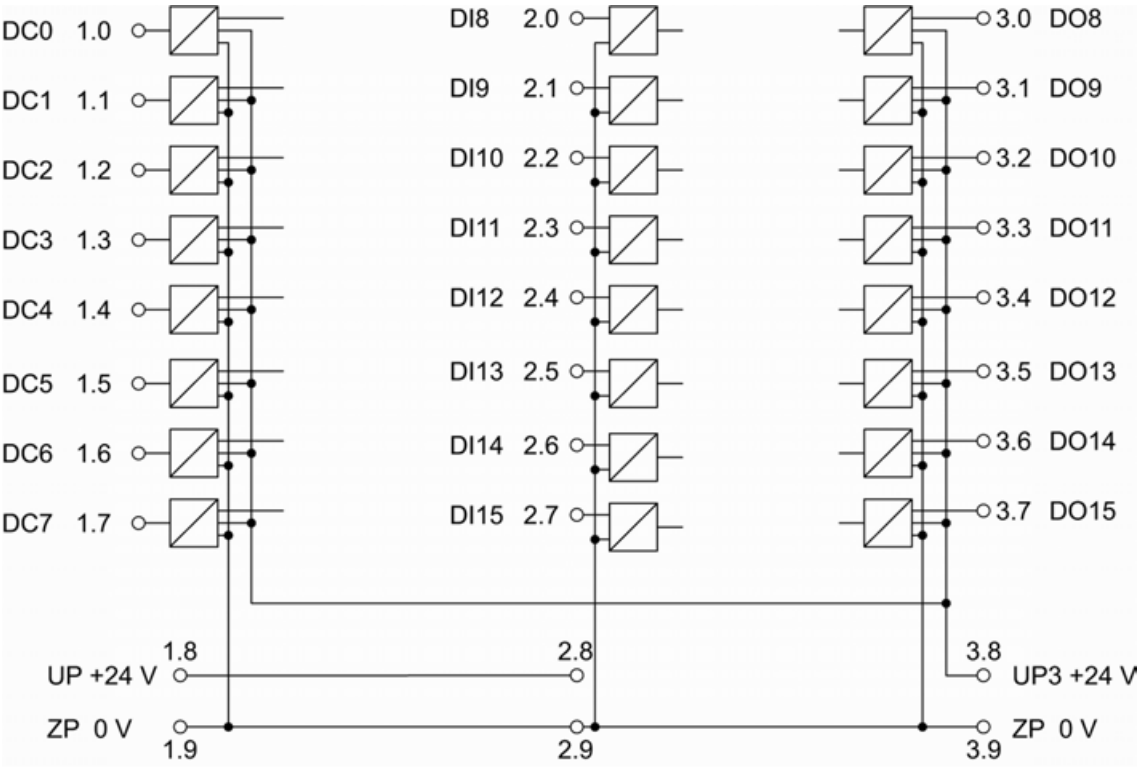
NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

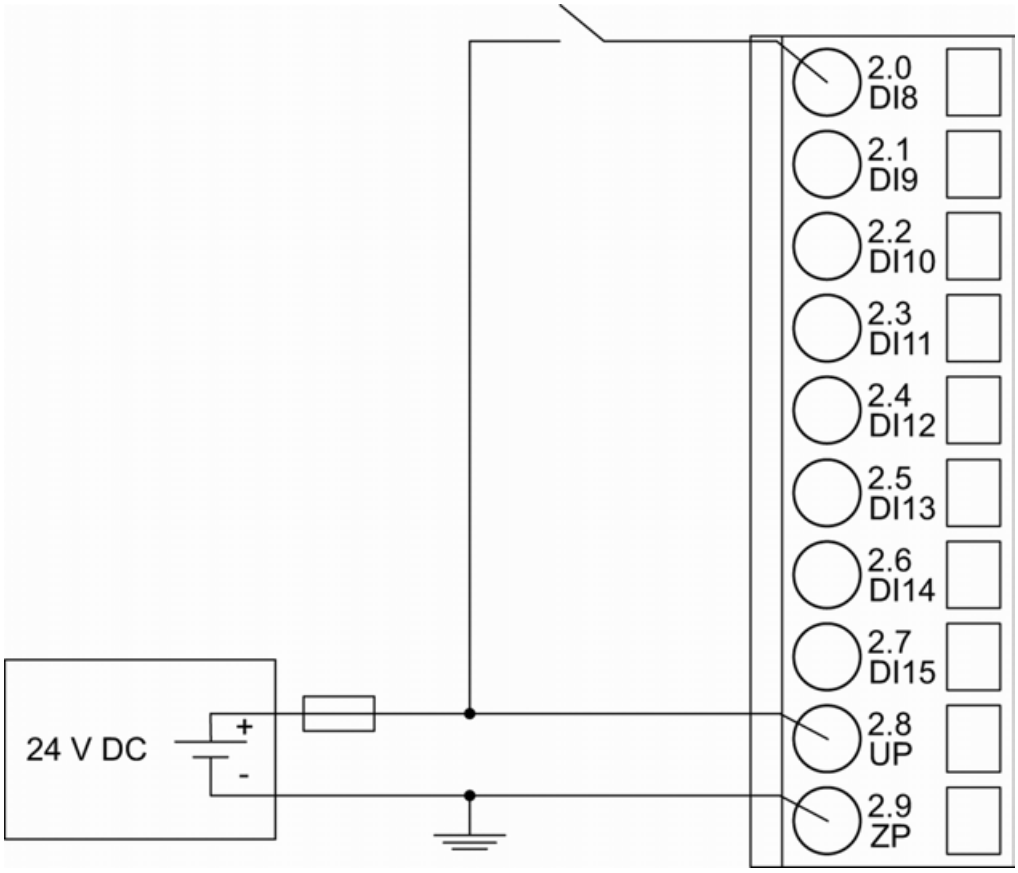
The following figure shows the connection of the Ethernet communication interface module CI502-PNIO.



Further information is provided in the System Technology chapter PROFINET *Chapter 1.6.4.3.3 “PROFINET communication interface module” on page 5681*.

Connection of the Digital inputs

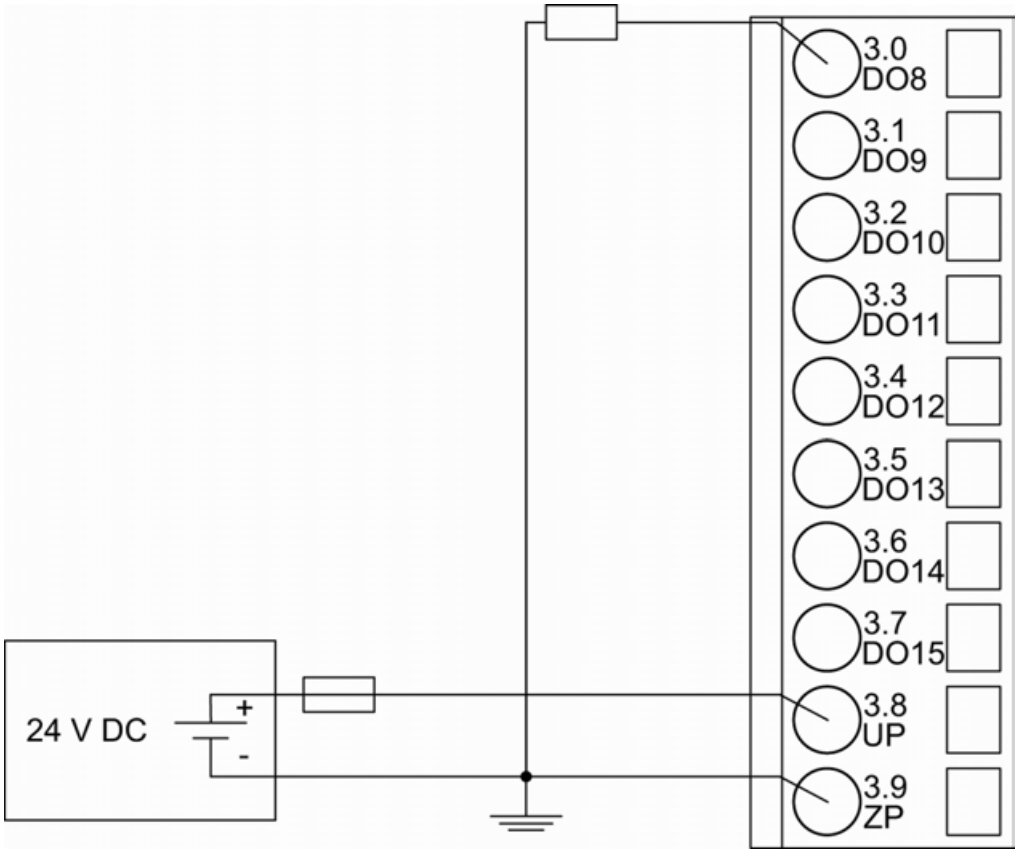
The following figure shows the connection of the digital input DI8. Proceed with the digital inputs DI9 to DI15 in the same way.



The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.7.3.8.1 “State LEDs” on page 5052*.

Connection of the Digital outputs

The following figure shows the connection of the digital output DO8. Proceed with the digital outputs DO9 - DO15 in the same way.

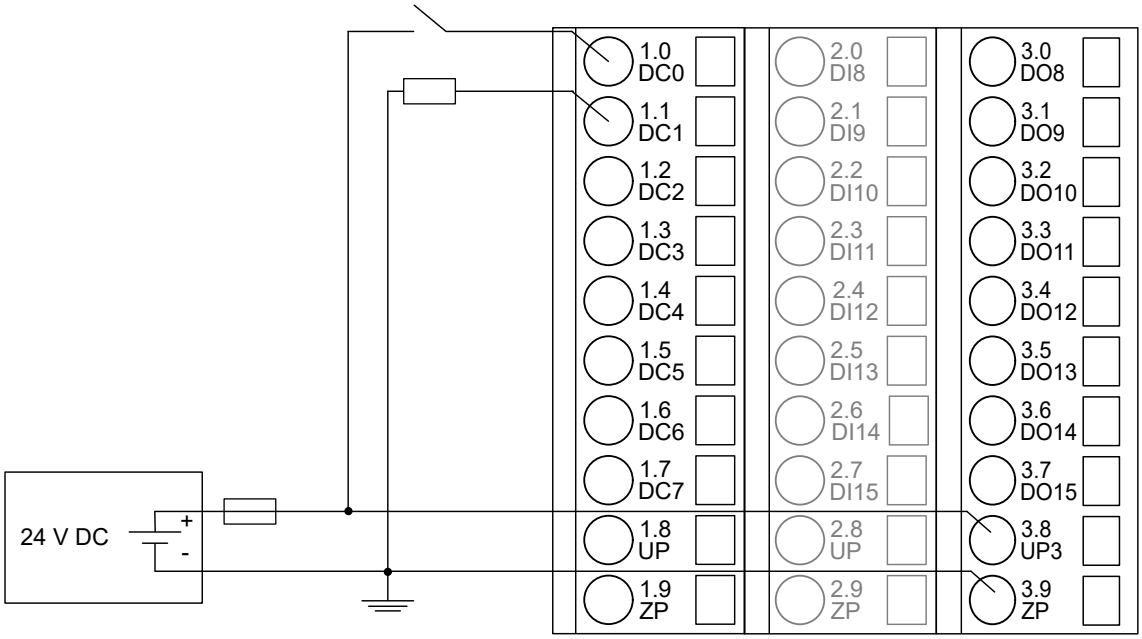


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.7.3.8.1 “State LEDs” on page 5052*.

Connection of the configurable digital inputs/outputs

The following figure shows the connection of the configurable digital input/output DC0 and DC1. DC0 is connected as an input and DC1 is connected as an output. Proceed with the configurable digital inputs/outputs DC2 to DC7 in the same way.

CAUTION!
If a DC channel is used as input, the source for the input signals should be the impressed UP3 of the device *Chapter 1.6.2.8.7.3.3 "Connections" on page 5037*.

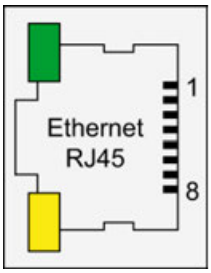


The meaning of the LEDs is described in Displays *Chapter 1.6.2.8.7.3.8.1 "State LEDs" on page 5052*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet *Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.*

Internal data exchange

Parameter	Value
Digital inputs (bytes)	5
Digital outputs (bytes)	5
Counter input data (words)	4
Counter output data (words)	8

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI502-PNIO stores some PROFINET configuration parameters (I/O device identifier, I/O device type and IP address configuration). No more configuration data is stored.

The digital I/O channels are configured via software.

Details about configuration are described in Parameterization *Chapter 1.6.2.8.7.3.7 “Parameterization” on page 5044*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7005	WORD	7005
Parameter length	Internal	8	BYTE	8

Name	Value	Internal value	Internal value, type	Default
Error LED / Failsafe function (Table Error LED / Failsafe function *Further information on page 5044*)	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
Process cycle time	1 ms process cycle time	1	BYTE	1 ms
	2 ms process cycle time	2		
	3 ms process cycle time	3		
	4 ms process cycle time	4		
	5 ms process cycle time	5		
	6 ms process cycle time	6		
	7 ms process cycle time	7		
	8 ms process cycle time	8		
	9 ms process cycle time	9		
	10 ms process cycle time	10		
	11 ms process cycle time	11		
	12 ms process cycle time	12		
	13 ms process cycle time	13		
	14 ms process cycle time	14		
	15 ms process cycle time	15		
	16 ms process cycle time	16		
Check supply	Off On	0 1	BYTE	1
Fast counter	0 : 10 [2])	0 : 10	BYTE	0
I/O-Bus reset	Off	0	BYTE	Off

Name	Value	Internal value	Internal value, type	Default
	On	1	BYTE	Off
[1]) With a faulty ID, the module reports a "parameter error" and does not perform cyclic process data transmission.				
[2]) Counter operating modes *Chapter 1.6.2.6.1.2.10 “Fast counter” on page 4351*				

Table 562: Table Error LED / Failsafe function

Setting	Description
On	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode off
On + Failsafe	Error LED (S-ERR) lights up at errors of all error classes, Failsafe-mode on *)
Off by E4 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1, E2 and E3, Failsafe-mode on *)
Off by E3 + Failsafe	Error LED (S-ERR) lights up at errors of error classes E1 and E2, Failsafe-mode on *)
*) The parameter Behaviour DO at comm. error is only analyzed if the Failsafe-mode is ON.	

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *“On”* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *“On”* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *“0”* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *“Off”* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *“Off”* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *“0”* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Group parameters for the digital part

Name	Value	Internal value	Internal value, type	Default
Input delay	0.1 ms 1 ms 8 ms 32 ms	0 1 2 3	BYTE	0.1 ms 0x00
Detect short circuit at outputs	Off On	0 1	BYTE	On 0x01
Behaviour DO at comm. error [1])	Off Last value Last value 5 sec Last value 10 sec Substitute value Substitute value 5 sec Substitute value 10 sec	0 1 6 11 2 7 12	BYTE	Off 0x00
Substitute value at output	0...65535	0000h...FFFFh	WORD	0 0x0000
Preventive voltage feedback monitoring for DC0..DC7 [2])	Off On	0 1	BYTE	Off 0x00
Detect voltage overflow at outputs [3])	Off On	0 1	BYTE	Off 0x00

Remarks:

[1])	The parameter Behaviour DO at comm. error is apply to DC and DO channels and only analyzed if the Failsafe-mode is ON.
[2])	The state "externally voltage detected" appears, if the output of a channel DC0...DC7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. The monitoring of this state and the resulting diagnosis message can be disabled by setting the parameters to "OFF".
[3])	The error state "voltage overflow at outputs" appears, if externally voltage at digital outputs DC0...DC7 and accordingly DO0...DO7 has exceeded the process supply voltage UP3 ↬ *Chapter 1.6.2.8.7.3.3 “Connections” on page 5037* (see description in section). The according diagnosis message "Voltage overflow on outputs " can be disabled by setting the parameters on "OFF". This parameter should only be disabled in exceptional cases for voltage overflow may produce reverse voltage.

Diagnosis

Structure of the Diagnosis Block via PNIO_DEV_ALARM function block ↬ *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*.

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI502-PNIO (e. g. error at integrated 8 DI / 8 DO) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

In cases of short circuit or overload, the digital outputs are turned off. The modules performs reactivation automatically. Thus an acknowledgement of the errors is not necessary. The error message is stored via the LED.

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
Module errors								
3	-	31	31	31	19	Checksum error in the I/O module		Replace I/O module
3	-	31	31	31	3	Timeout in the I/O module		

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
3	-	31	31	31	40	Different hard-/firm-ware versions in the module		
3	-	31	31	31	43	Internal error in the module		
3	-	31	31	31	36	Internal data exchange failure		
3	-	31	31	31	9	Overflow diagnosis buffer		Restart
3	-	31	31	31	26	Parameter error		Check master
3	-	31	31	31	11	Process voltage UP too low		Check process supply voltage
3	-	31	31	31	45	Process voltage UP gone		Check process supply voltage
3	-	31/1...10	31	31	17	No communication with I/O device		Replace I/O module
3	-	1...10	31	31	32	Wrong I/O device type on socket		Replace I/O module / Check configu-ration
4	-	1...10	31	31	31	At least one module does not support failsafe function		Check modules and parame-terization
4	-	1...10	31	5	8	I/O module removed from hot swap terminal unit or defective module on hot swap ter-minal unit [9])		Plug I/O module, replace I/O module

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	1...10	31	5	28	Wrong I/O module plugged on hot swap terminal unit [9])	Remove wrong I/O module and plug pro-jected I/O module
4	-	1...10	31	5	42	No communication with I/O module on hot swap terminal unit [9])	Replace I/O module
4	-	1...10	31	5	54	I/O module does not support hot swap [8]) [9])	Power off system and replace I/O module
4	-	1...10	31	6	8	Hot swap terminal unit configured but not found	Replace terminal unit by hot swap terminal unit
4	-	1...10	31	6	42	No communication with hot swap ter-minal unit [9])	Restart, if error persists replace terminal unit
4	1...6	255	2	0	45	The connected Communication Module has no con-nection to the net-work	Check cabeling
4	-	31	31	31	45	Process voltage UP3 too low	Check process voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500-Display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC Browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error-** **Identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
4	-	31	31	31	46	Reverse voltage from digital outputs DO0..DO7 to UP3 [4])	Check terminals
4	-	31/1...10	31	31	34	No response during initialization of the I/O module	Replace I/O module
4	-	31	31	31	11	Process voltage UP3 too low	Check process supply voltage
4	-	31	31	31	45	Process voltage UP3 gone	Check process supply voltage
4	-	31	31	31	10	Voltage overflow at outputs (above UP3 level) [5])	Check terminals/ check process supply voltage
Channel error digital							
4	-	31	2	8...15	46	Externally voltage detected at digital output DO0..DO7 [6])	Check terminals
4	-	31	4	0...7	46	Externally voltage detected at digital output DC0..DC7 [6])	Check terminals
4	-	31	4	0...7	47	Short circuit at digital output DC0..DC7[7])	Check terminals
4	-	31	2	8...15	47	Short circuit at digital output DO0..DO7[7])	Check terminals

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the Communication Module;14 = I/O-Bus; 31 = Module itself The identifier is not contained in the CI502-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself, 1..10 = Expansion module
[3])	With "Module" the following allocation applies dependent of the master: Module error: 31 = Module itself Channel error: Module type (1 = AI, 2 = DO, 3 = AO)
[4])	This message appears, if externally voltages at one or more terminals DC0...DC7 oder DO0...DO7 cause that other digital outputs are supplied through that voltage (voltage feedback, see description in 'Connections' *Chapter 1.6.2.8.7.3.3 "Connections" on page 5037*. All outputs of the apply digital output groups will be turned off for 5 seconds. The diagnosis message appears for the whole output group.
[5])	The voltage at digital outputs DC0...DC7 and accordingly DO0...DO7 has exceeded the process supply voltage UP3 *Chapter 1.6.2.8.7.3.3 "Connections" on page 5037*. Diagnosis message appears for the whole module.
[6])	This message appears, if the output of a channel DC0...DC7 or DO0...DO7 should be switched on while an externally voltage is connected. In this case the start up is disabled, as long as the externally voltage is connected. Otherwise this could produce reverse voltage from this output to other digital outputs. This diagnosis message appears per channel.
[7])	Short circuit: After a detected short circuit, the output is deactivated for 2000 ms. Then a new start up will be executed. This diagnosis message appears per channel.
[8])	In case of an I/O module doesn't support hot swapping, do not perform any hot swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.
[9])	Diagnosis for hot swap available as of version index F0.

State LEDs

The LEDs are located at the front of module. There are 2 different groups:

- The 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- The 29 process LEDs (UP, UP3, inputs, outputs, CH-ERR1 to CH-ERR3) show the process supply voltage and the states of the inputs and outputs and display possible errors.

Table 563: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with IO Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System-LED "BF")	Green	---	Device configured, cyclic data exchange running	---

LED	Color	OFF	ON	Flashing
	Red	---	---	Device is not configured
STA2 ETH (System LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No expansion modules connected or communication error	Expansion modules connected and operational	---
ETH1	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 564: States of the 29 process LEDs

LED	Color	OFF	ON	Flashing
DC0 to DC7	Yellow	Input/Output is OFF	Input/Output is ON	--
DI8 to DI15	Yellow	Input is OFF	Input is ON (the input voltage is even displayed if the supply voltage is OFF)	--
DO8 to DO15	Yellow	Output is OFF	Output is ON	--
UP	Green	Process supply voltage missing	Process supply voltage OK and initialization finished	--
UP3	Green	Process supply voltage missing	Process supply voltage OK	--
CH-ERR1 to CH-ERR3	Red	No error or process supply voltage missing	Internal error	Error on one channel of the corresponding group

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP/UP3		
	Rated value	24 V DC (for inputs and outputs)
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP/UP3	10 A fast
	Galvanic isolation	Ethernet interface against the rest of the module
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Current consumption via UP3	0.06 A + 0.5 A max. per output
	Connections	Terminals 1.8 and 2.8 for +24 V (UP) Terminal 3.8 for +24 V (UP3) Terminals 1.9, 2.9 and 3.9 for 0 V (ZP)
Max. power dissipation within the module		6 W
Number of digital inputs		8
Number of digital outputs		8
Number of configurable digital inputs/outputs		8
Input data length		12 bytes
Output data length		20 bytes
Reference potential for all digital inputs and outputs		Negative pole of the supply voltage, signal name ZP
Setting of the I/O device identifier		With 2 rotary switches at the front side of the module
Diagnosis		See Diagnosis and Displays *Chapter 1.6.2.8.7.3.8 “Diagnosis” on page 5047*
Operation and error displays		34 LEDs (totally)
Weight (without terminal unit)		Ca. 125 g
Mounting position		Horizontal or vertical with derating (output load reduced to 50 % at 40 °C per group)
Extended ambient temperature (XC version)		> 60 °C on request
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the digital inputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DI0 to DI7	Terminals 2.0 to 2.7

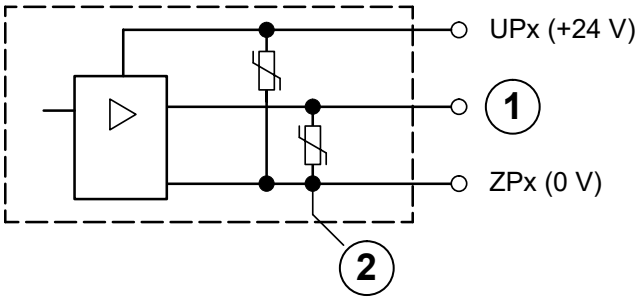
Parameter		Value
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

Technical data of the digital outputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DO0 to DO7		Terminals 3.0 to 3.7
Reference potential for all outputs		Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage		For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1		UP3 (-0.8 V)
Output delay (0->1 or 1->0)		On request
Output current		
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request

Parameter		Value
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload-proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital output
2 Varistors for demagnetization when inductive loads are turned off

Technical data of the configurable digital inputs/outputs

Each of the configurable I/O channels is defined as input or output by the user program. This is done by interrogating or allocating the corresponding channel.

Parameter		Value
Number of channels per module		8 inputs/outputs (with transistors)
Distribution of the channels into groups		1 group for 8 channels
If the channels are used as inputs		
	Channels DC0...DC07	Terminals 1.0...1.7
If the channels are used as outputs		
	Channels DC0...DC07	Terminals 1.0...1.7
Indication of the input/output signals		1 yellow LED per channel, the LED is ON when the input/output signal is high (signal 1)
Galvanic isolation		From the Ethernet network

Technical data of the digital inputs/outputs if used as inputs

Parameter		Value
Number of channels per module		8
Distribution of the channels into groups		1 group of 8 channels
Terminals of the channels DC0 to DC7		Terminals 1.0 to 1.7
Reference potential for all inputs		Terminals 1.9...3.9 (Negative pole of the supply voltage, signal name ZP)
Indication of the input signals		1 yellow LED per channel, the LED is ON when the input signal is high (signal 1)
Input type (according EN 61131-2)		Type 1
Input delay (0->1 or 1->0)		Typ. 0.1 ms, configurable from 0.1...32 ms
Input signal voltage		24 V DC
	Signal 0	-3 V...+5 V
	Undefined Signal	> +5 V...< +15 V
	Signal 1	+15 V...+30 V
Ripple with signal 0		Within -3 V...+5 V
Ripple with signal 1		Within +15 V...+30 V
Input current per channel		
	Input voltage +24 V	Typ. 5 mA
	Input voltage +5 V	> 1 mA
	Input voltage +15 V	> 2 mA
	Input voltage +30 V	< 8 mA
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

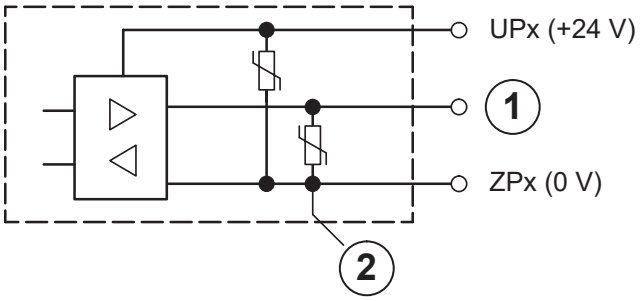
*) Due to the direct connection to the output, the demagnetizing varistor is also effective at the input (see figure) above. This is why the difference between UPx and the input signal may not exceed the clamp voltage of the varistor. The varistor limits the voltage to approx. 36 V. Following this, the input voltage must range from -12 V to +30 V when UPx = 24 V and from -6 V to +30 V when UPx = 30 V.

Technical data of the digital inputs/outputs if used as outputs

Parameter	Value
Number of channels per module	8
Distribution of the channels into groups	1 group of 8 channels
Terminals of the channels DC0 to DC7	Terminals 1.0 to 1.7
Reference potential for all outputs	Terminals 1.9...3.9 (negative pole of the supply voltage, signal name ZP)
Common power supply voltage	For all outputs terminal 3.8 (positive pole of the supply voltage, signal name UP3)
Output voltage for signal 1	UP3 (-0.8 V)
Output delay (0->1 or 1->0)	On request
Output current	

Parameter		Value
	Rated value per channel	500 mA at UP3 = 24 V
	Max. value (all channels together)	4 A
Leakage current with signal 0		< 0.5 mA
	Fuse for UP3	10 A fast
Demagnetization with inductive DC load		Via internal varistors (see figure below this table)
Output switching frequency		
	With resistive load	On request
	With inductive loads	Max. 0.5 Hz
	With lamp loads	11 Hz max. at 5 W max.
Short-circuit-proof / overload proof		Yes
Overload message (I > 0.7 A)		Yes, after ca. 100 ms
Output current limitation		Yes, automatic reactivation after short circuit/overload
Resistance to feedback against 24 V signals		Yes (software-controlled supervision)
Max. cable length		
	Shielded	1000 m
	Unshielded	600 m

The following drawing shows the circuitry of a digital input/output with the varistors for demagnetization when inductive loads are switched off.



1 Digital input/output
2 For demagnetization when inductive loads are turned off

Technical data of the fast counter

Parameter	Value
Used inputs	Terminal 2.0 (DI8),Terminal 2.1 (DI9)
Used outputs	Terminal 3.0 (DO8)
Counting frequency	Depending on operation mode: Mode 1- 6: max. 200 kHz Mode 7: max. 50 kHz Mode 9: max. 35 kHz Mode 10: max. 20 kHz

↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*

↳ *Chapter 1.6.4.4.2.2 “Operating modes” on page 5716*

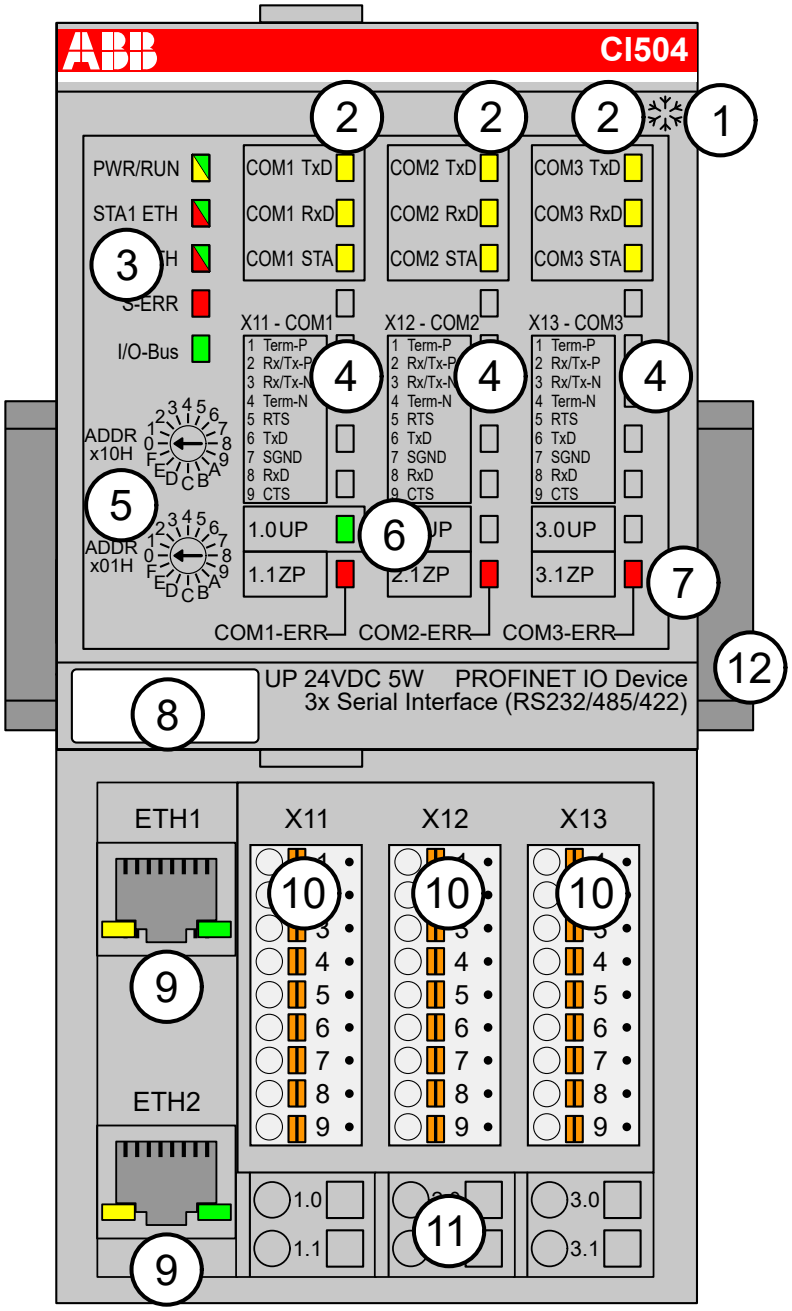
Ordering data

Active	Active	Product life cycle phase *)
1SAP 220 700 R0001	CI502-PNIO (V3), PROFINET communication interface module, 8 DI, 8 DO and 8 DC	Active
1SAP 420 700 R0001	CI502-PNIO-XC (V3), PROFINET communication interface module, 8 DI, 8 DO and 8 DC, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI504-PNIO

- 3 serial UART interfaces (RS-232, RS-422 or RS-485)
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 3 x 3 yellow LEDs to display the signal states of the serial interfaces COM1, COM2 and COM3
3 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
4 Allocation between terminal number and signal name of the serial interfaces
5 2 rotary switches for setting the I/O device identifier
6 1 green LED to display the process voltage UP
7 3 red LEDs to display errors (COM1-ERR, COM2-ERR, COM3-ERR) of the serial interfaces
8 Label
9 Ethernet Interfaces (ETH1, ETH2) on the terminal unit
10 3 removable connectors to connect the interfaces
11 6 spring terminals for power supply voltage (UP)
12 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface module CI504-PNIO provides 3 onboard serial interfaces. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

The bus interfaces are galvanically isolated from the Ethernet network.

For usage in extreme ambient conditions (e. g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Interface	Ethernet
Protocol	PROFINET IO RT
Serial Interfaces	3 Serial UART interfaces RS-232, RS-422 and RS-485 available as physical layer
Serial protocol	ASCII
I/O bus interface	For up to 10 AC500 I/O Modules
Rotary switches	For setting the I/O device identifier for configuration purposes (00h to FFh)
LED displays	For system displays, field bus indication, errors and power supply
Power supply	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU520 *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*

Connections

The PROFINET communication interface module CI504-PNIO is plugged on the terminal unit TU520-ETH *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the power supply voltage is carried out using the 6 terminals and the 3 removable connectors of the terminal unit. The CI504-PNIO can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter *Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.*

The terminals 1.0, 2.0 and 3.0 as well as 1.1, 2.1 and 3.1 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Table 565: Assignment of the terminals

Terminal	Signal	Description
1.0	UP	Process voltage UP (+24 V DC)
1.1	ZP	Process voltage ZP (0 V DC)
2.0	UP	Process voltage UP (+24 V DC)
2.1	ZP	Process voltage ZP (0 V DC)
3.0	UP	Process voltage UP (+24 V DC)
3.1	ZP	Process voltage ZP (0 V DC)

Table 566: Assignment of the terminals of removable connectors X11, X12 and X13 (Serial interfaces)

Terminal	Signal	Description	
1	Term-P	RS-485	Internal line terminating resistor for non-inverted signal (Rx/Tx-P)
		RS-422	Non-inverted receive signal terminal (RxD+)
2	Rx/Tx-P	RS-485	Non-inverted I/O signal terminal for each channel
		RS-422	Non-inverted transmit signal terminal (TxD+)
3	Rx/Tx-N	RS-485	Inverted I/O signal terminal for each channel
		RS-422	Inverted transmit signal terminal (TxD-)
4	Term-N	RS-485	Internal line-terminating resistor for inverted signal (Rx/Tx-N) terminal
		RS-422	Inverted receive signal terminal (RxD-)
5	RTS	RS-232	Request To Send signal terminal for each channel
6	TxD	RS-232	Transmit signal terminal for each channel
7	SGND	RS-232	Signal ground for each channel
8	RxD	RS-232	Receive signal terminal for each channel
9	CTS	RS-232	Clear To Send signal terminal for each channel

The connection of SGND (ground) is optional for RS-485/RS-422.

For RS-422, no external line-terminating resistors have to connected. They are already connected inside the module.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

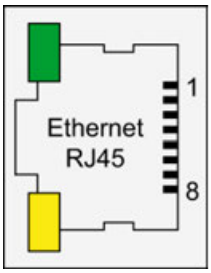
The module provide several diagnosis functions *Chapter 1.6.2.8.7.4.7 “Diagnosis” on page 5069*.

Further information is provided in the System Technology chapter PROFINET *Chapter 1.6.4.2.3 “PROFINET communication modules” on page 5534*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet
Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7010	WORD	7010
Parameter length	Internal	33	BYTE	33
Error LED / Fail-safe function see table [2])	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + fail-safe	17		
	Off by E3 + fail-safe	19		
I/O-Bus reset	Off	0	BYTE	Off
	On	1	BYTE	Off

Remarks:

[1]) With a faulty module ID, the module reports a "parameter error" and does not perform cyclic process data transmission

Table 567: Error LED / Failsafe function [2])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe-mode off

Setting	Description
Off by E3	Error LED lights up at errors of error classes E1 and E2, Fail-safe-mode off
On + Failsafe	Error LED lights up at errors of all error classes, Failsafe-mode on
Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe-mode on
Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Fail-safe-mode on

All values are validated during the parameterization of the CI504-PNIO according to the appended expansion modules. In the case of error, a diagnosis message "parameter errors" is generated and the cyclic process data transfer is terminated.

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *"On"* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *"On"* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *"0"* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *"Off"* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *"Off"* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *"0"* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Parameters of the 3 serial channels

Name	Value	Internal value	Internal value, type	Default
Behavior for serial channel communication during PROFINET communication fault	Stop communication and reset FIFO	0	BYTE	0
	Continue serial communication	1		
Number of frames/data blocks in reception FIFO	1...40	1...40	BYTE	1

Name	Value	Internal value	Internal value, type	Default
Number of frames/Data blocks in transmission FIFO	1...40	1...40	BYTE	1
Behavior during reception FIFO overflow	Discard new received frames	1	BYTE	2
	Overwrite oldest frame in FIFO	2		
	Discard new received frames and send PROFINET alarm	3		
	Overwrite oldest frame in FIFO and send PROFINET alarm	4		
Physical layer	RS-232	1	BYTE	1
	RS-485	2		
	RS-422	3		
RTS control	None	0	BYTE	1
	Telegram	1		
	RTS/CTS (DTE <-> DTE)	2		
	RTS/CTS (DTE -> DCE)	3		
	RTS/CTS (DCE <- DTE)	4		
TLS (RTS leading cycle)	0...850 ms	0...850	WORD	0
CDLY (RTS trailing cycle)	0...850 ms	0...850	WORD	0
Character timeout	0/32 bits	0/32	WORD	0
Telegram ending selection	None	0	BYTE	None
	String (check reception)	1		
	Telegram length	2		
	Character timeout	4		
Telegram ending character	0...255	0...255	BYTE	0
Telegram ending value	0...65535	0...65535	WORD	0
Checksum	None	0	BYTE	0
	CRC8	1		
	CRC16	2		
	LRC	3		
	ADD	4		

Name	Value	Internal value	Internal value, type	Default
	CS31	5		
	CRC8-FBP	6		
	XOR	7		
	CRC16 (Intel)	8		
Handshake mode	None	0	BYTE	0
	XON/XOFF	2		
Transmission rate	Channel inactive	0	DWORD	19200
	300 bit/s	300		
	1200 bit/s	1200		
	4800 bit/s	4800		
	9600 bit/s	9600		
	14400 bit/s	14400		
	19200 bit/s	19200		
	38400 bit/s	38400		
	38400 bit/s	57600		
	57600 bit/s	57600		
	115200 bit/s	115200		
Parity	No parity	0	BYTE	No parity
	Odd parity	1		
	Even parity	2		
Data bits	5 bits	0	BYTE	8
	6 bits	1		
	7 bits	2		
	8 bits	3		
Stop bits	1 bit	0	BYTE	1
	2 bits	1		

Configuration with Automation Builder

The physical layers are selectable as submodules in PROFINET configuration (parameter Physical Layer not visible and fixed with the correct value). Certain parameters are not visible if a certain physical layer is selected. This concept of parameterization provides a better usability than configuring via GSDML (see below).

Configuration via GSDML (use by 3rd party PROFINET configuration tool)

All parameters are visible independent of the configured physical layer (via parameter “Physical Layer”). The user must take precautions for each parameter since certain parameter values are invalid for certain physical layers. Nevertheless, the CI5xx-PNIO module performs a parameter check depending on the configured physical layer and generates a diagnosis message (parameter error) in the case of error.

General precautions

- If parameter telegram ending selection is set to value Character Timeout, the value in the parameter Character Timeout must be set to 0. The parameter End Value must be set to 32 (equivalent to 32-bits character timeout). Only 32-bits character timeout is supported.
- Checksum is only supported if a telegram ending selection is active.
- Please refer to AC500 serial channel documentation for additional precautions.

Precautions for RS-485/RS-422

DTE/DCE is not supported. The parameter RTS Control must be set to value Telegram or to None.

Diagnosis

Structure of the Diagnosis Block via PNIO_DEV_ALARM function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI504-PNIO (e. g. error at integrated Serial Interface) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
Module error							
3	-	31	31	31	43	Internal error in the module	Replace module
3	-	31	31	31	9	Overflow diagnosis buffer	New start
3	-	31	31	31	26	Parameter error	Check master
3	-	31	31	31	11	Process voltage too low	Check process voltage
3	-	31	31	31	45	Process voltage gone	Check process voltage
3	-	1...10	31	31	17	No communication with I/O module	Replace I/O module
4	-	1...10	31	31	31	At least 1 I/O Module does not support failsafe mode	Check I/O modules and parame-terization
4	-	1...10	31	31	32	Wrong I/O Module type on socket	Replace I/O module Check configu-ration
4	-	1...10	31	31	34	No response during initialization of the I/O Module	Replace I/O module
Serial Channel error							

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	

Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	**[2])**	**[3])**				
4	-	31	31	1...3	12	Reception SW FIFO overrun	Check modules and parame-terization
4	-	31	31	1...3	26	Parameter error	Check modules and parame-terization

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the Communication Module; 14 = I/O bus; 31 = Module itself The identifier is not contained in the CI504-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: 31 = Module itself
[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself or 1...10 expansion module

State LEDs

The LEDs are located at the front of module. There are 4 different groups:

- 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- 4 Ethernet state LEDs located at the terminal unit TU520-ETH
- 12 state LEDs for the serial interfaces
- 1 LED to display the presence of the process supply voltage UP

Table 568: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System LED "BF")	Green	---	Device configured, cyclic data exchange running	---
	Red	---	---	Device is not configured
STA2 ETH (System LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No communication interface module connected or communication error	communication interface module connected and operational	---

Table 569: States of the 4 Ethernet state LEDs

LED	Color	OFF	ON	Flashing
ETH1-Link	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
ETH1-Rx Tx	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2-Link	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
Eth2-Rx Tx	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 570: States of the 12 state LEDs (4 per channel) of the serial interfaces

LED	Color	OFF	ON	Flashing
COMx TxD	Yellow	No data transmission over serial network	--	Channel is transmitting data via the serial interface (flashing rate depending on the telegram transmission frequency)
COMx RxD	Yellow	No data reception from serial network	--	Channel is receiving data from the serial interface (flashing rate depending on the telegram reception frequency)
COMx STA	Yellow	RS-232: RTS signal not active RS-485: Channel is in reception mode RS-422:Channel is not enabled	RS-232: RTS signal is active RS-485: Channel is transmitting RS-422: Channel is enabled (able to receive and transmit)	--
COMx-ERR	Red	Channel enabled, no error OR Channel deactivated	Channel boot up	Channel error (receive buffer overflow)

Table 571: State of the power supply LED

LED	Color	OFF	ON	Flashing
UP	Green	No process voltage available	Process voltage available	--

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP		
	Rated value	24 V DC
	Max. load for the terminals	10 A
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Connections	Terminals 1.0, 2.0 and 3.0 for +24 V (UP) Terminals 1.1, 2.1 and 3.1 for 0 V (ZP)
Input data length		0...36 bytes
Output data length		0...36 bytes
Max. power dissipation within the module		5 W
Setting of the I/O module identifier		With 2 rotary switches at the front side of the module
Operation and error displays		18 LEDs (total)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Galvanic isolation	Ethernet interface against the rest of the module, each serial port against each other and the rest of the module
Diagnosis	See Diagnosis *Chapter 1.6.2.8.7.4.7 “Diagnosis” on page 5069*

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated

Parameter	Value
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the serial interfaces

Parameter	Value
Number of serial interfaces	3
Connectors for serial interfaces	X11 for COM1 X12 for COM2 X13 for COM3
Supported physical layers	RS-232 RS-422 RS-485
Supported protocols	ASCII
Transmission rate	Configurable from 300 bit/s to 115.200 bit/s

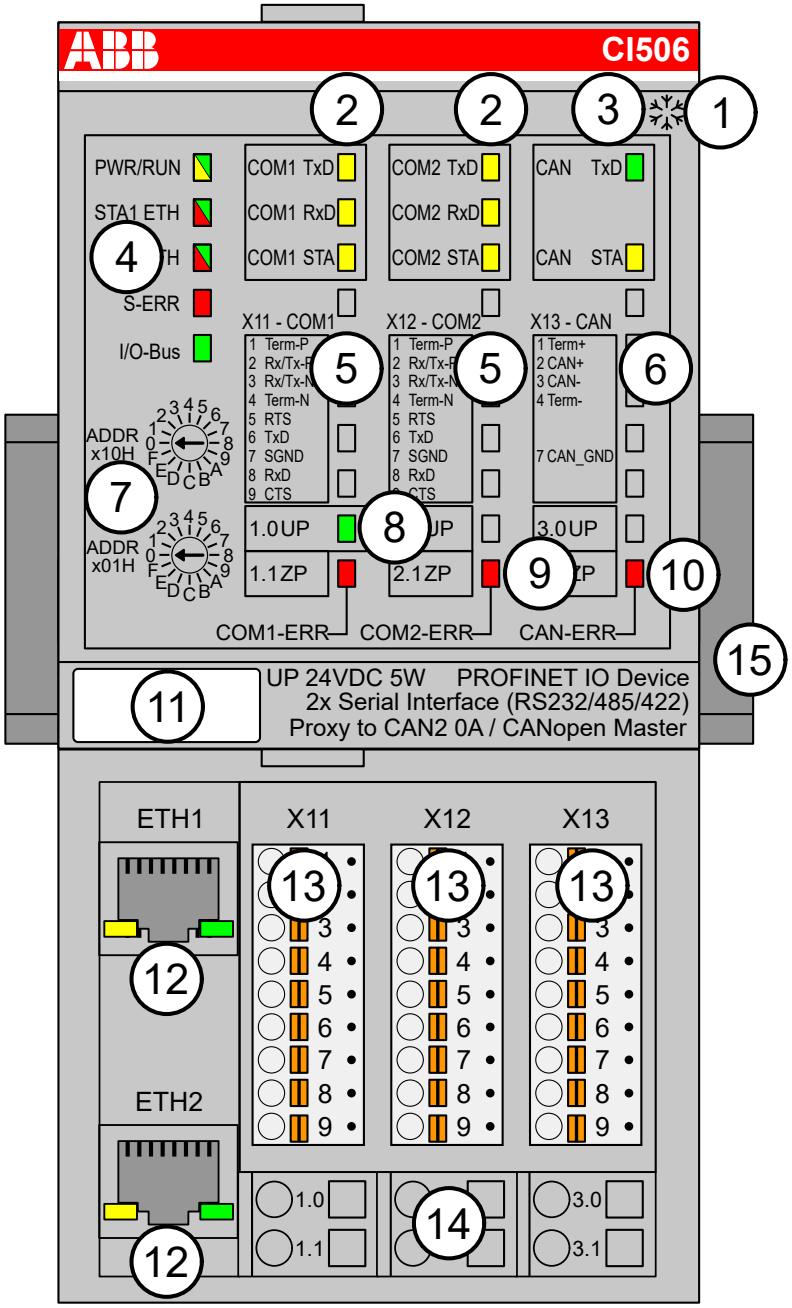
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 221 300 R0001	CI504-PNIO, PROFINET communication interface module with 3 serial interfaces	Active
1SAP 421 300 R0001	CI504-PNIO-XC, PROFINET communication interface module with 3 serial interfaces, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CI506-PNIO

- 2 serial UART interfaces (RS-232, RS-422 or RS-485)
- 1 CANopen master interface
- Module-wise galvanically isolated
- XC version for usage in extreme ambient conditions available



1 I/O bus
2 2 x 3 yellow LEDs to display the signal states of the serial interfaces COM1 and COM2
3 1 green and 1 yellow LEDs to display the signal states of the CANopen interface
4 5 system LEDs: PWR/RUN, STA1 ETH, STA2 ETH, S-ERR, I/O-Bus
5 Allocation between terminal number and signal name of the serial interfaces
6 Allocation between terminal number and signal name of the CANopen interface
7 2 rotary switches for setting the I/O device identifier
8 1 green LED to display the process voltage UP
9 2 red LEDs to display errors (COM1-ERR, COM2-ERR) of the serial interfaces
10 1 red LED to display errors (CAN-ERR) of the CANopen interface
11 Label
12 Ethernet Interfaces (ETH1, ETH2) on the terminal unit
13 3 removable connectors to connect the subordinated interfaces
14 6 spring terminals for power supply voltage (UP)
15 DIN rail
❄ Sign for XC version

Intended purpose

The PROFINET communication interface module CI506-PNIO provides 2 onboard serial interfaces and 1 CANopen master interface. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

The bus interfaces are galvanically isolated from the Ethernet network.

For usage in extreme ambient conditions (e. g. wider temperature and humidity range), a special XC version of the device is available.

Functionality

Parameter	Value
Primary interface	Ethernet
Protocol (1st interface)	PROFINET IO RT
Secondary interface	CAN
Protocol (2nd interface)	CANopen
CANopen master	Transmission rate up to 1 Mbit/s Support for up to 126 CANopen slaves
Serial Interfaces	2 Serial UART interfaces RS-232, RS-422 and RS-485 available as physical layer
Serial protocol	ASCII
I/O bus interface	For up to 10 AC500 I/O modules
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the I/O device identifier for configuration purposes (00h to FFh)
LED displays	For system displays, field bus indication, errors and power supply
Power supply	Via terminals UP and ZP (process supply voltage 24 V DC)
Effect of incorrect input terminal connection	Wrong or no signal detected, no damage up to 35 V
Required terminal unit	TU520 ↳ *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*

Connections

The Ethernet Bus Module CI506-PNIO is plugged on the terminal unit TU520-ETH ↳ *Chapter 1.6.2.5.5 “TU520-ETH for PROFINET communication interface modules” on page 4112*. Properly seat the module and press until it locks in place. The terminal unit is mounted on a DIN rail or with 2 screws plus the additional accessory for wall mounting (TA526 ↳ *Chapter 1.6.2.9.2.6 “TA526 - Wall mounting accessory” on page 5180*).

The connection of the power supply voltage is carried out using the 6 terminals and the 3 removable connectors of the terminal unit. The CI506-PNIO can be replaced without re-wiring the terminal units.

For a detailed description of the mounting, disassembly and connection of the module, please refer to the System Assembly, Construction and Connection chapter ↳ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313.

The terminals 1.0, 2.0 and 3.0 as well as 1.1, 2.1 and 3.1 are electrically interconnected within the terminal unit and have always the same assignment, independent of the inserted module:

Table 572: Assignment of the terminals

Terminal	Signal	Description
1.0	UP	Process voltage UP (+24 V DC)
1.1	ZP	Process voltage ZP (0 V DC)
2.0	UP	Process voltage UP (+24 V DC)
2.1	ZP	Process voltage ZP (0 V DC)
3.0	UP	Process voltage UP (+24 V DC)
3.1	ZP	Process voltage ZP (0 V DC)

Table 573: Assignment of the terminals of removable connectors X11 and X12 (Serial interfaces)

Terminal	Signal	Description	
1	Term-P	RS-485	Internal line terminating resistor for non-inverted signal (Rx/Tx-P)
		RS-422	Non-inverted receive signal terminal (RxD+)
2	Rx/Tx-P	RS-485	Non-inverted I/O signal terminal for each channel
		RS-422	Non-inverted transmit signal terminal (TxD+)
3	Rx/Tx-N	RS-485	Inverted I/O signal terminal for each channel
		RS-422	Inverted transmit signal terminal (TxD-)
4	Term-N	RS-485	Internal line-terminating resistor for inverted signal (Rx/Tx-N) terminal
		RS-422	Inverted receive signal terminal (RxD-)
5	RTS	RS-232	Request To Send signal terminal for each channel
6	TxD	RS-232	Transmit signal terminal for each channel
7	SGND	RS-232	Signal ground for each channel
8	RxD	RS-232	Receive signal terminal for each channel
9	CTS	RS-232	Clear To Send signal terminal for each channel

The connection of SGND (ground) is optional for RS-485/RS-422.

For RS-422, no external line-terminating resistors have to connected. They are already connected inside the module.

Table 574: Assignment of the terminals of removable connector X13 (CANopen interface)

Terminal	Signal	Description
1	TERM+	Internal line-terminating resistor for CAN bus. Bridging to CAN HIGH terminal if bus termination is required
2	CAN+	Non-inverted CAN data terminal
3	CAN-	Inverted CAN data terminal
4	TERM-	Internal line-terminating resistor for CAN bus. Bridging to CAN LOW terminal if bus termination is required
5	Not used	Not used
6	Not used	Not used
7	CAN_GND	CAN ground terminal
8	Not used	Not used
9	Not used	Not used

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

NOTICE!

Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

– Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
– Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

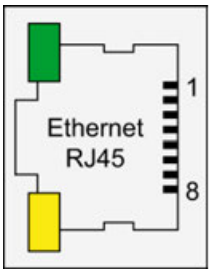
The module provide several diagnosis functions ↳ *Chapter 1.6.2.8.7.5.8 “Diagnosis” on page 5087*.

Further information is provided in the System Technology chapter PROFINET *Chapter 1.6.4.2.3 “PROFINET communication modules” on page 5534*.

Assignment of the Ethernet ports

The terminal unit for the communication interface module provides two Ethernet interfaces with the following pin assignment:

Pin assignment

Interface	PIN	Signal	Description
	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	RxD-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

In corrosive environment, please protect unused connectors using the TA535 accessory.

Not supplied with this device.

For further information regarding wiring and cable types see chapter Ethernet Chapter 1.6.3.6.4.10 “Ethernet connection details” on page 5353.

Addressing

The module reads the position of the rotary switches only during power-up, i. e. changes of the switch position during operation will have no effect until the next module initialization.

I/O configuration

The CI506-PNIO stores some PROFINET configuration parameters:

- Slave station name
- Slave station type
- IP address configuration
- MAC address
- Production data

No more configuration data is stored. The serial interfaces and the CANopen interface is configured via software. For details, refer to Parameterization ↳ *Chapter 1.6.2.8.7.5.7 "Parameterization" on page 5082*.

Parameterization

Parameters of the module

Name	Value	Internal value	Internal value, type	Default
Module ID [1])	Internal	7015	WORD	7015
Parameter length	Internal	33	BYTE	33
Error LED / Failsafe function see table [2])	On	0	BYTE	0
	Off by E4	1		
	Off by E3	3		
	On + failsafe	16		
	Off by E4 + failsafe	17		
	Off by E3 + failsafe	19		
I/O-Bus reset	Off	0	BYTE	Off
	On	1	BYTE	Off

Remarks:

[1]) With a faulty module ID, the module reports a "parameter error" and does not perform cyclic process data transmission

Table 575: Error LED / Failsafe function [2])

Setting	Description
On	Error LED lights up at errors of all error classes, Failsafe-mode off
Off by E4	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe-mode off
Off by E3	Error LED lights up at errors of error classes E1 and E2, Failsafe-mode off
On + Failsafe	Error LED lights up at errors of all error classes, Failsafe-mode on
Off by E4 + Failsafe	Error LED lights up at errors of error classes E1, E2 and E3, Failsafe-mode on
Off by E3 + Failsafe	Error LED lights up at errors of error classes E1 and E2, Failsafe-mode on

All values are validated during the parameterization of the CI506-PNIO according to the appended communication interface modules. In the case of error, a diagnosis message "parameter error" is generated and the cyclic process data transfer is terminated.

IO-BUS reset after PROFINET reconnection

IO-BUS reset after PROFINET reconnection controls the behavior of PROFINET CI modules in relation to connected I/O modules (both safety and non-safety I/O modules).

- IO-BUS reset after PROFINET reconnection = *"On"* resets and, thus, re-parameterizes all attached I/O modules. All internal I/O modules states are reset, including the related diagnosis information.

  Note that if the parameter is set to *"On"* then:
  - The bumpless re-start of non-safety I/O modules will not be supported. It means, for example, that non-safety output channels will go from fail-safe values to *"0"* values during the re-connection and re-parameterization time and after that go to new output values.
  - Safety I/O modules will be re-parameterized and re-started as newly started modules, which may not require their PROFIsafe reintegration, depending on safety CPU state, in the safety application.
- IO-BUS reset after PROFINET reconnection = *"Off"* will not reset all attached I/O modules. It will re-parameterize I/O modules only if parameter change is detected during the reconnection. All internal I/O modules states are not reset, including the related diagnosis information.

  Note that if the parameter is set to *"Off"* then:
  - The bumpless re-start of non-safety I/O modules is supported (if no parameters are changed). It means, for example, that non-safety output channels will not go from fail-safe values to *"0"* values during the re-connection and re-parameterization time, but directly from fail-safe values to new output values.
  - Safety I/O modules will not be re-parameterized (if no parameters are changed). Thus, they may continue their operation, which may require their PROFIsafe reintegration in the safety application on the safety CPU, e.g., if PROFIsafe watchdog time for this safety I/O module has expired. Any reintegration of such safety I/O modules will be not only application specific but also PROFIsafe specific and depend on the safety I/O handling in the safety application.

Parameters of the 2 serial channels

Name	Value	Internal value	Internal value, type	Default
Behavior for serial channel communication during PROFINET communication fault	Stop communication and reset FIFO	0	BYTE	0
	Continue serial communication	1		
Number of frames/data blocks in reception FIFO	1...40	1...40	BYTE	1
Number of frames/Data blocks in transmission FIFO	1...40	1...40	BYTE	1
Behavior during reception FIFO overflow	Discard new received frames	1	BYTE	2
	Overwrite oldest frame in FIFO	2		
	Discard new received frames and send PROFINET alarm	3		
	Overwrite oldest frame in FIFO and send PROFINET alarm	4		
Physical layer	RS-232	1	BYTE	1
	RS-485	2		
	RS-422	3		

Name	Value	Internal value	Internal value, type	Default
RTS control	None	0	BYTE	1
	Telegram	1		
	RTS/CTS (DTE <-> DTE)	2		
	RTS/CTS (DTE -> DCE)	3		
	RTS/CTS (DCE <- DTE)	4		
TLS (RTS leading cycle)	0...850 ms	0...850	WORD	0
CDLY (RTS trailing cycle)	0...850 ms	0...850	WORD	0
Character timeout	0/32 bits	0/32	WORD	0
Telegram ending selection	None	0	BYTE	None
	String (check reception)	1		
	Telegram length	2		
	Character timeout	4		
Telegram ending character	0 - 255	0 - 255	BYTE	0
Telegram ending value	0 - 65535	0 - 65535	WORD	0
Checksum	None	0	BYTE	0
	CRC8	1		
	CRC16	2		
	LRC	3		
	ADD	4		
	CS31	5		
	CRC8-FBP	6		
	XOR	7		
	CRC16 (Intel)	8		
Handshake mode	None	0	BYTE	0
	XON/XOFF	2		
Transmission rate	Channel inactive	0	DWORD	19200
	300 bit/s	300		
	1200 bit/s	1200		
	4800 bit/s	4800		
	9600 bit/s	9600		
	14400 bit/s	14400		
	19200 bit/s	19200		
	38400 bit/s	38400		
	38400 bit/s	57600		
	57600 bit/s	57600		
	115200 bit/s	115200		
Parity	No parity	0	BYTE	No parity

Name	Value	Internal value	Internal value, type	Default
	Odd parity	1		
	Even parity	2		
Data bits	5 bits	0	BYTE	8
	6 bits	1		
	7 bits	2		
	8 bits	3		
Stop bits	1 bit	0	BYTE	1
	2 bits	1		

Configuration with Automation Builder

The physical layers are selectable as submodules in PROFINET configuration (parameter Physical Layer not visible and fixed with the correct value). Certain parameters are not visible if a certain physical layer is selected. This concept of parameterization provides a better usability than configuring via GSDML (see below).

Configuration via GSDML (use by 3rd party PROFINET configuration tool)

All parameters are visible independent of the configured physical layer (via parameter "Physical Layer"). The user must take precautions for each parameter since certain parameter values are invalid for certain physical layers. Nevertheless, the CI5xx-PNIO module performs a parameter check depending on the configured physical layer and generates a diagnosis message (parameter error) in the case of error.

General precautions

- If parameter telegram ending selection is set to value Character Timeout, the value in the parameter Character Timeout must be set to 0. The parameter End Value must be set to 32 (equivalent to 32-bits character timeout). Only 32-bits character timeout is supported.
- Checksum is only supported if a telegram ending selection is active.
- Please refer to AC500 serial channel documentation for additional precautions.

Precautions for RS-485/RS-422

DTE/DCE is not supported. The parameter RTS Control must be set to value Telegram or to None.

Parameters of the CANopen master

Name	Value	Internal value	Internal value, type	Default
CANopen master transmission rate	1000 kbit/s	0	DWORD	0
	800 kbit/s	1		
	500 kbit/s	2		

Name	Value	Internal value	Internal value, type	Default
	250 kbit/s	3		
	125 kbit/s	4		
	100 kbit/s	5		
	50 kbit/s	6		
	20 kbit/s	7		
	10 kbit/s	8		
CANopen master SYNC object ID *)	0x01 to 0x7FFF	1 - 32767	DWORD	0x80
CANopen master SYNC cycle time *)	SYNC OFF	0	DWORD	0
	1 ms to 65535 ms	1 - 65535		
CANopen master heartbeat producer time *)	Heartbeat producer OFF	0	DWORD	10
	1 ms to 65535 ms	1 - 65535		
*) Parameter becomes irrelevant if the CANopen master function is not selected.				

The CANopen master functionality can only be activated when using ControlBuilderPlus/Automation Builder.

CAN2A / CAN2B parameters

Name	Value	Internal value	Internal value, type	Default
CAN transmission rate	1000 kbit/s	0	DWORD	0
	800 kbit/s	1		
	500 kbit/s	2		
	250 kbit/s	3		
	125 kbit/s	4		
	100 kbit/s	5		
	50 kbit/s	6		
	20 kbit/s	7		
	10 kbit/s	8		

Configuration via GSDML (use by 3rd party PROFINET configuration tool)
The parameter CAN transmission rate must be set twice for each CAN2A and CAN2B interfaces, and they must be set with identical values.

Buffer parameters (to be configured for each used buffer)

Name	Value	Internal value	Internal value, type	Default
Identifier	0..2047 (CAN2A)	0..2047 (CAN2A)	WORD (CAN2A)	0
	0..536870911 (CAN2B)	0..536870911 (CAN2B)	DWORD (CAN2B)	
Receive buffer size (size in numbers of telegrams)	1...32	1...32	BYTE	1
Behaviour on receive buffer overflow *)	Overwrite	0	BYTE	0
	Discard	1		
	Overwrite and send diagnostics (PROFINET alarm)	3		
	Discard and send diagnostics (PROFINET alarm)	4		
*) The following table describes the values in detail.				

Setting	Description
Overwrite	The oldest buffer entry which is stored in the buffer is overwritten with the new incoming telegram.
Discard	The new incoming telegram is discarded.
Overwrite and send diagnostics (PROFINET alarm)	The oldest buffer entry which is stored in the buffer is overwritten with the new incoming telegram. Additionally, a PROFINET alarm (diagnostic) will be sent to inform the user of the overflow occurrence.
Discard and send diagnostics (PROFINET alarm)	The new incoming telegram is discarded. Additionally a PROFINET alarm (diagnostic) will be sent to inform the user of the overflow occurrence.

Up to 64 buffers are allowed to be configured for each CAN2A and CAN2B type, each buffer containing the parameters described above.

Diagnosis

Structure of the Diagnosis Block via PNIO_DEV_ALARM function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI506-PNIO (e. g. error at integrated serial interface) 1 = 1st connected S500 I/O module ... 10 = 10th connected S500 I/O module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<– Display in**
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**	
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**	
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**	**Remedy**
	[1])	[2])	[3])				
Module error							
3	-	31	31	31	43	Internal error in the module	Replace module
3	-	31	31	31	9	Overflow diagnosis buffer	New start
3	-	31	31	31	26	Parameter error	Check master
3	-	31	31	31	11	Process voltage too low	Check process voltage

E1...E4	**d1**	**d2**	**d3**	**d4**	**Identi-fier** **000...063**	**AC500 display**	**<− Display in**	
Class	**Comp**	**Dev**	**Mod**	**Ch**	**Err**	**PS501 PLC browser**		
Byte 4 **Bit 6...7**	**-**	**Byte 1**	**Byte 2**	**Byte 3**	**Byte 4** **Bit 0...5**	**PNIO diag-nosis block**		
Class	**Inter-face**	**Device**	**Module**	**Channel**	**Error identi-fier**	**Error message**		**Remedy**
	[1])	[2])	[3])					
3	-	1..10	31	31	17	No communication with I/O Module		Replace I/O module
4	-	1..10	31	31	31	At least 1 I/O Module does not support failsafe mode		Check I/O modules and parame-terization
4	-	1..10	31	31	32	Wrong I/O Module type on socket		Replace I/O module Check configu-ration
4	-	1..10	31	31	34	No response during initialization of the I/O Module		Replace I/O Module
Serial Channel error								
4	-	31	31	1...2	12	Reception SW FIFO overrun		Check modules and parame-terization
4	-	31	31	1...2	26	Parameter error		Check modules and parame-terization
CANopen Channel error [4])								
4	-	31	31	12...75	12	Reception SW FIFO (CAN2.0A) overrun (Buffer number 1...64) [5])		Check modules and parame-terization
4	-	31	31	112...175	12	Reception SW FIFO (CAN2.0B) overrun (Buffer number 1...64) [5])		Check modules and parame-terization

Remarks:

[1])	In AC500 the following interface identifier applies: "-" = Diagnosis via bus-specific function blocks; 0...4 or 10 = Position of the Communication Module; 14 = I/O bus; 31 = Module itself The identifier is not contained in the CI506-PNIO diagnosis block.
[2])	With "Device" the following allocation applies: ADR = Hardware address (e.g. of the CI506-PNIO)
[3])	With "Module" the following allocation applies dependent of the master: 31 = Module itself
[4])	All CANopen master and slave diagnostics are not available as PROFINET alarms; instead they can be read via PROFINET acyclic service. In AC500 PLC these are available in form of function blocks.
[5])	CAN2A Buffers 1...64 are mapped to the channel values 12...75, so the correlation value 11 has to be subtracted from the channel value to get the correct buffer number. CAN2B Buffers 1...64 are mapped to the channel values 112...175, so the correlation value 111 has to be subtracted from the channel value to get the correct buffer number

State LEDs

The LEDs are located at the front of module. There are 4 different groups:

- 5 system LEDs (PWR, STA1 ETH, STA2 ETH, S-ERR and I/O-Bus) show the operation state of the module and display possible errors.
- 4 Ethernet state LEDs located at the terminal unit TU520-ETH
- 11 state LEDs for the serial interfaces an the CANopen Interface
- 1 LED to display the presence of the process supply voltage UP

Table 576: States of the 5 system LEDs

LED	Color	OFF	ON	Flashing
PWR/RUN	Green	Process supply voltage missing	Internal supply voltage OK, module ready for communication with I/O Controller	Start-up / preparing communication
	Yellow	---	---	---
STA1 ETH (System-LED "BF")	Green	---	Device configured, cyclic data exchange running	---
	Red	---	---	Device is not configured
STA2 ETH (System-LED "SF")	Green	---	---	Got identification request from I/O controller
	Red	No system error	System error (collective error)	---

LED	Color	OFF	ON	Flashing
S-ERR	Red	No error	Internal error	--
I/O-Bus	Green	No communication interface modules connected or communication error	Communication interface module connected and operational	---

Table 577: States of the 4 Ethernet state LEDs

LED	Color	OFF	ON	Flashing
ETH1-Link	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
ETH1-Rx Tx	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams
ETH2-Link	Green	No connection at Ethernet interface	Connected to Ethernet interface	---
Eth2-Rx Tx	Yellow	---	Device is transmitting telegrams	Device is transmitting telegrams

Table 578: States of the 8 state LEDs (4 per channel) of the serial interfaces

LED	Color	OFF	ON	Flashing
COMx TxD	Yellow	No data transmission over serial network	--	Channel is transmitting data via the serial interface (flashing rate depending on the telegram transmission frequency)
COMx RxD	Yellow	No data reception from serial network	--	Channel is receiving data from the serial interface (flashing rate depending on the telegram reception frequency)
COMx STA	Yellow	RS-232: RTS signal not active RS-485: Channel is in reception mode RS-422:Channel is not enabled	RS-232: RTS signal is active RS-485: Channel is transmitting RS-422: Channel is enabled (able to receive and transmit)	--
COMx-ERR	Red	Channel enabled, no error or Channel deactivated	Channel boot up	Channel error (receive buffer overflow)

Table 579: States of the 3 state LEDs of the CANopen interfaces

LED	Color	OFF	ON	Flashing
CAN-RUN	Yellow	--	Device configured, CANopen Bus in OPERATIONAL state and cyclic data exchange running	Flashing cyclically: CANopen Bus in Pre-operational state and slave is being configured Single flash: CANopen Bus in Stopped state.
CAN-STA	Yellow	No data transmission	Channel is transmitting data	--
CAN-ERR	Red	No error	CANopen bus is OFF	Flashing cyclically: Configuration error Single flash: Error counter overflow due to too many error frames Double flash: A Node-Guard or a Heartbeat event occurred

Table 580: State of the power supply LED

LED	Color	OFF	ON	Flashing
UP	Green	No process voltage available	Process voltage available	--

Technical data

The system data of AC500 and S500 *Chapter 1.6.3.6.1 “System data AC500” on page 5313* are applicable to the standard version.

The system data of AC500-XC *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389* are applicable to the XC version.

Only additional details are therefore documented below.

The technical data are also applicable to the XC version.

Technical data of the module

Parameter		Value
Process supply voltages UP		
	Rated value	24 V DC
	Max. load for the terminals	10 A

Parameter		Value
	Protection against reversed voltage	Yes
	Rated protection fuse on UP	10 A fast
	Inrush current from UP (at power up)	On request
	Current consumption via UP (normal operation)	0.15 A
	Connections	Terminals 1.0, 2.0 and 3.0 for +24 V (UP) Terminals 1.1, 2.1 and 3.1 for 0 V (ZP)
Input data length		0...36 bytes
Output data length		0...36 bytes
Max. power dissipation within the module		5 W
Setting of the I/O module identifier		With 2 rotary switches at the front side of the module
Operation and error displays		18 LEDs (total)
Weight (without terminal unit)		ca. 125 g
Mounting position		Horizontal or vertical
Cooling		The natural convection cooling must not be hindered by cable ducts or other parts in the switchgear cabinet.

Galvanic isolation	Ethernet interface against the rest of the module, each serial and CAN port against each other and the rest of the module
Diagnosis	See Diagnosis *Chapter 1.6.2.8.7.5.8 “Diagnosis” on page 5087*

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Multiple overloads

No effects of multiple overloads on isolated multi-channel modules occur, as every channel is protected individually by an internal smart high-side switch.

Parameter	Value
Bus connection	2 x RJ45
Switch	Integrated
Technology	Hilscher NETX 100
Transfer rate	10/100 Mbit/s (full-duplex)
Transfer method	According to Ethernet II, IEEE 802.3
Ethernet	100 base-TX, internal switch, 2x RJ45 socket

Parameter	Value
Expandability	Max. 10 S500 I/O modules
Adjusting elements	2 rotary switches for generation of an explicit name
Supported protocols	RTC - real time cyclic protocol, class 1 *) RTA - real time acyclic protocol DCP - discovery and configuration protocol CL-RPC - connectionless remote procedure Call LLDP - link layer discovery protocol MRP - MRP Client
Acyclic services	PNIO read / write sequence (max. 1024 bytes per telegram) Process-Alarm service
Supported alarm types	Process Alarm, Diagnostic Alarm, Return of SubModule, Plug Alarm, Pull Alarm
Min. bus cycle	1 ms
Conformance class	CC A
Protective functions (according to IEC 61131-3)	Protected against: ● short circuit ● reverse supply ● overvoltage ● reverse polarity Galvanic isolation from the rest of the module

*) Priorization with the aid of VLAN-ID including priority level

Technical data of the serial interfaces

Parameter	Value
Number of serial interfaces	2
Connectors for serial interfaces	X11 for COM1 X12 for COM2
Supported physical layers	RS-232 RS-422 RS-485
Supported protocols	ASCII
Transmission rate	Configurable from 300 bit/s to 115.200 bit/s

Technical data of the CANopen interface

Parameter	Value
Number of CANopen interfaces	1
Connector for CANopen Interface	X13
Transmission rate	Up to 1 Mbit/s

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 221 500 R0001	CI506-PNIO, PROFINET communication interface module with 2 serial interfaces and 1 CANopen master interface	Active
1SAP 421 500 R0001	CI506-PNIO-XC, PROFINET communication interface module with 2 serial interfaces and 1 CANopen master interface, XC version	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.9 Accessories

1.6.2.9.1 AC500-eCo

CR2032 - Battery for real-time clock

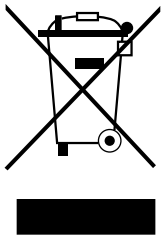
Intended purpose

A standard lithium battery (type CR2032) is used to backup the real-time clock (RTC) in the adapters TA561-RTC *Chapter 1.6.2.9.1.6 “TA561-RTC - Real-time clock adapter” on page 5113* and TA562-RS-RTC *Chapter 1.6.2.9.1.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5125* during power failures.

The CPU monitors the discharge degree of the battery. An diagnoses message is output before the battery condition becomes critical (about 2 weeks before). After the diagnosis message has appeared, the battery should be replaced as soon as possible.

Handling instruction

- The handling instructions of the battery manufacturer must be observed.
- The Material Safety Data Sheet (MSDS) of the battery manufacturer must be observed.
- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Recycle exhausted batteries meeting the environmental standards.



Transport

Transport of lithium batteries or equipment with installed lithium batteries:

- The transport and handling instructions of the battery producer must be observed.
- The transport regulations for transport of lithium batteries must be observed e.g. for transport by road or air.
- The forwarder must be informed if batteries are contained in the shipment.

Connections

Assembling and connection of the battery is described in chapters of TA561-RTC ↬ *Chapter 1.6.2.9.1.6 “TA561-RTC - Real-time clock adapter” on page 5113* and TA562-RS-RTC ↬ *Chapter 1.6.2.9.1.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5125*.

Battery lifetime

The battery lifetime is the time the battery can operate the RTC while the CPU is not powered. The typical lifetime is 300 days (at 25 °C).

As long as the CPU is powered, the battery will only be discharged by its own leakage current.

Technical data

The battery must meet die following technical data:

Parameter	Value
Battery designation	CR2032
Description	Manganese dioxide button cell, primary cell, not rechargeable
Nominal voltage	3 V DC
Capacity	230 mAh (measured with 5.6 kΩ load at 20 °C, discharging down to 2.0 V)
Typical lifetime (at 25 °C, CPU not powered)	300 days
Temperature range	≥ 0 °C ...+70 °C
Diameter	20 mm
Height	3.2 mm

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

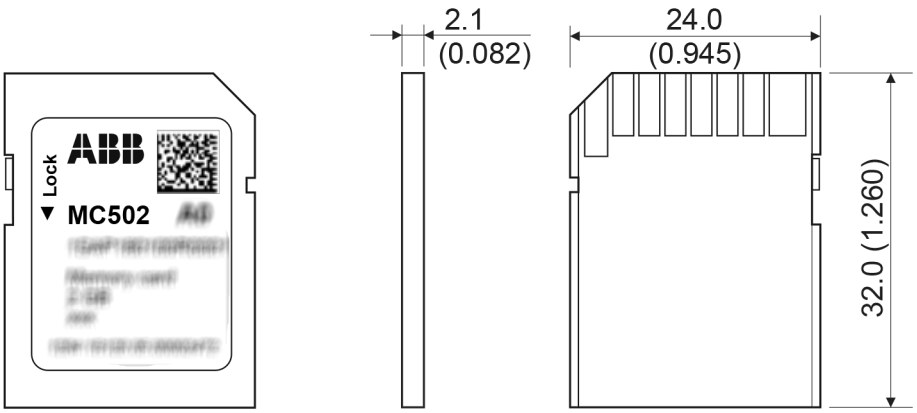
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

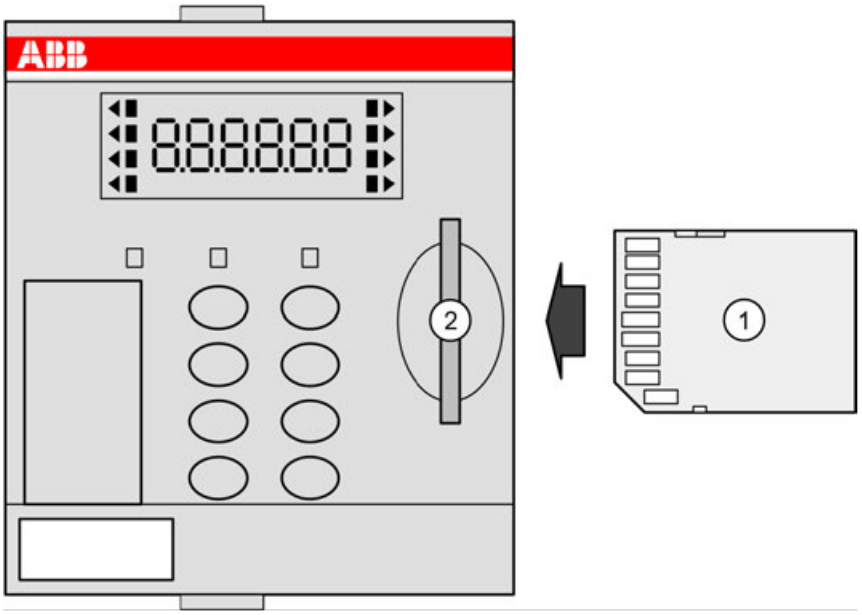


Fig. 1008: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

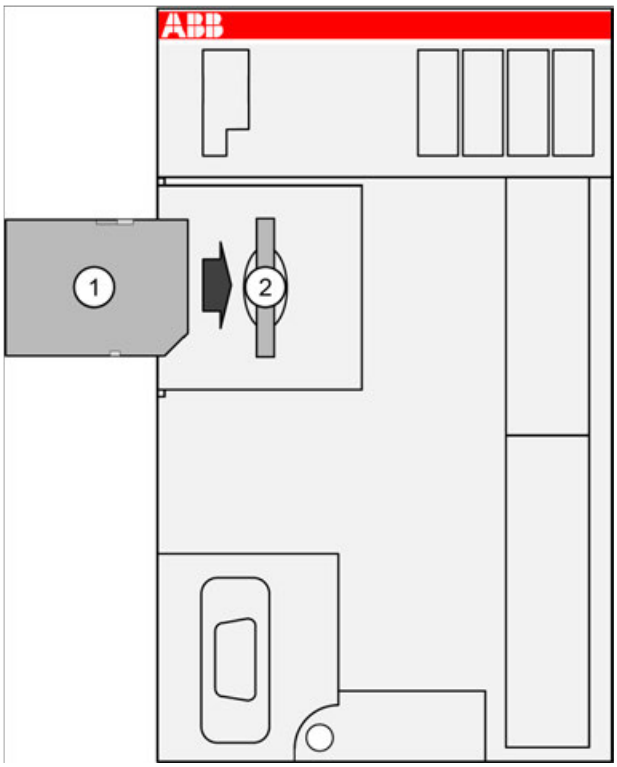


Fig. 1009: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

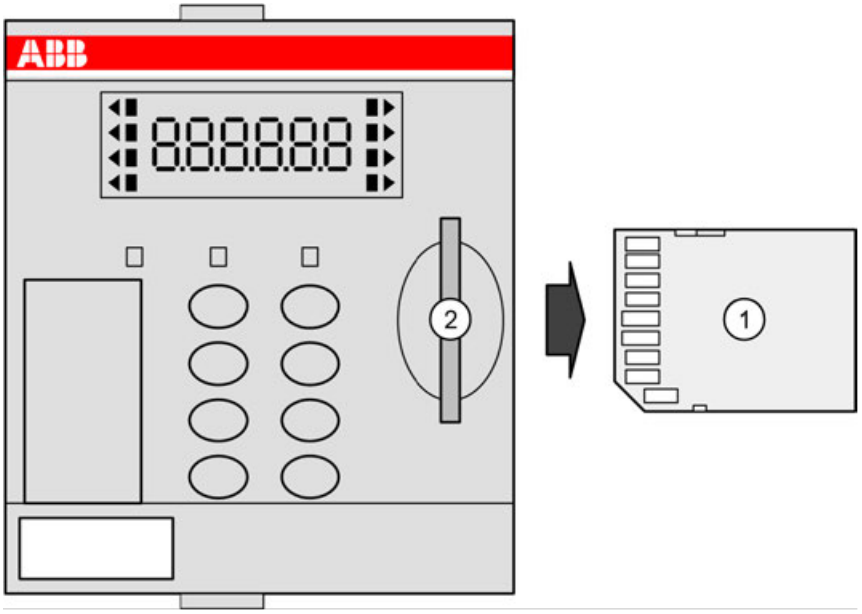


Fig. 1010: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

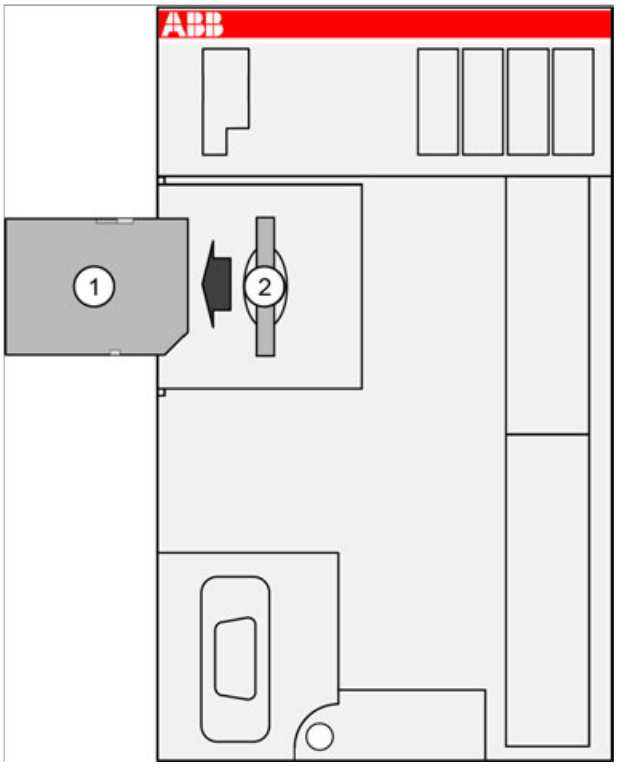


Fig. 1011: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter	Value
Memory capacity	2 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

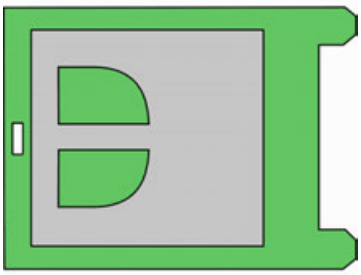
Further information on using the memory card in AC500 PLCs is provided in the chapter
↳ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC503 - Memory card adapter



Intended purpose

The MC503 memory card adapter is used for expanding processor modules PM55x-xP or PM56x-xP with a memory card slot.

A memory card MC502 or a micro memory card MC5102 with micro memory card adapter is not included in the scope of delivery and must be ordered separately.

The memory card can be used for:

- saving process data,
- saving user programs,
- upgrading the firmware.

Insert the memory card adapter

1. Make sure, that the power supply of the processor module is turned off.

> **WARNING!**
> **Risk of electric shock!**
> With an opened option board slot cover, energized parts of the processor module could be touched.
> – Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
> – Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

2. Remove the option board slot cover of the processor module totally by pushing it to the left side.
3. Plug the memory card adapter to the left expansion slot of the processor module. Make sure that the 2 noses of the expansion module fit to the holes of the processor module printed circuit board.
4. Remove the bar located in the middle of the option board slot cover for memory card slot.
5. Refit the option board slot cover.
6. To insert the memory card, see MC502 ↳ *Chapter 1.6.2.9.1.2 “MC502 - Memory card” on page 5096* or MC5102 ↳ *Chapter 1.6.2.9.1.4 “MC5102 - Micro memory card with micro memory card adapter” on page 5103*.

Remove the memory card adapter

1. Make sure that the power supply of the processor module is turned off.

> **WARNING!**
> **Risk of electric shock!**
> With an opened option board slot cover, energized parts of the processor module could be touched.
> – Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
> – Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

2. Remove the option board slot cover of the processor module totally by pushing it to the left side.
3. Remove the memory card adapter out of the processor module by lifting it up with a screwdriver.
4. Refit the option board slot cover. The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo V2 processor modules). ↳ *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*

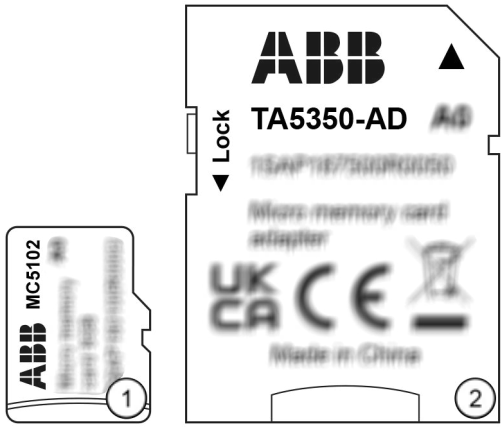
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R0100	MC503, memory card adapter for PM55x-xP or PM56x-xP	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

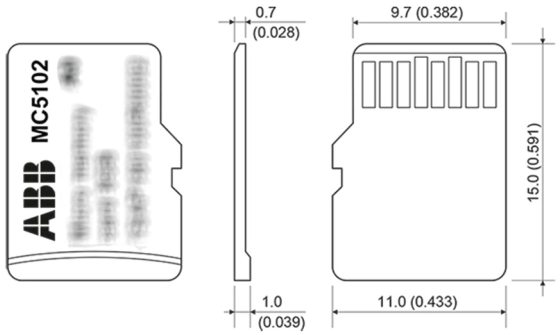
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

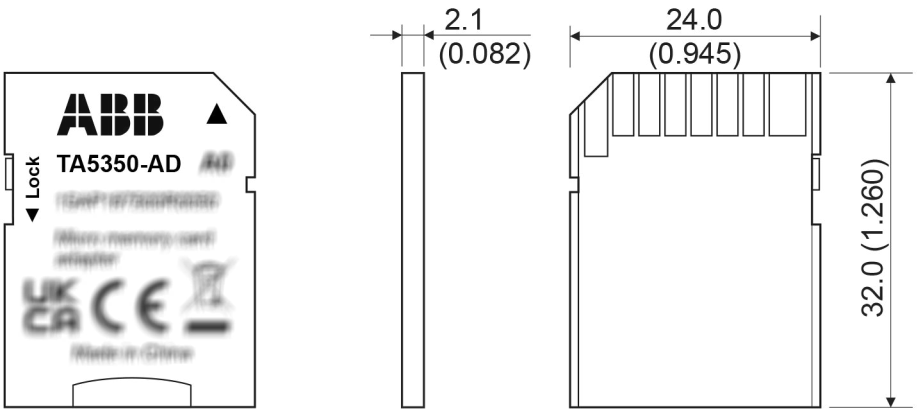
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V2 and AC500-eCo V2

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

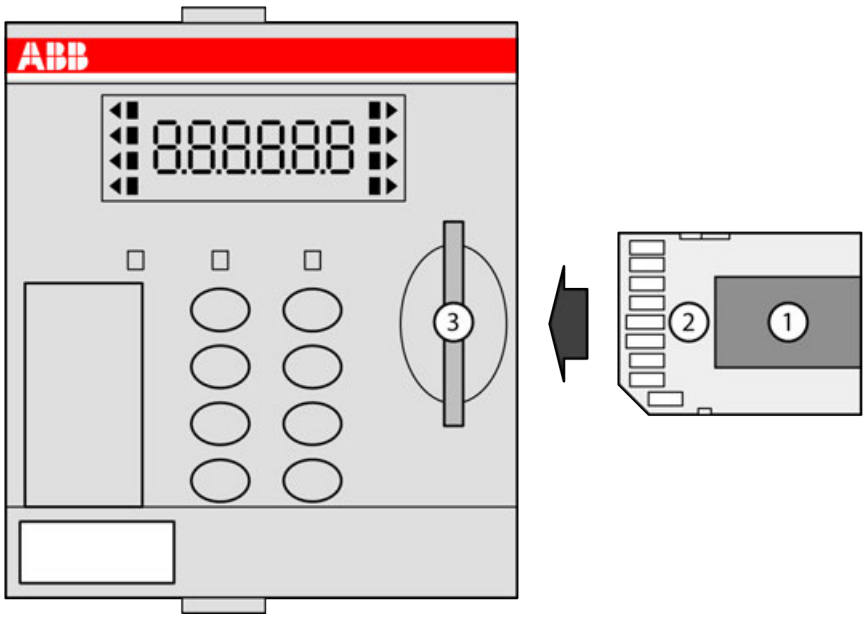


Fig. 1012: Insert micro memory card into PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

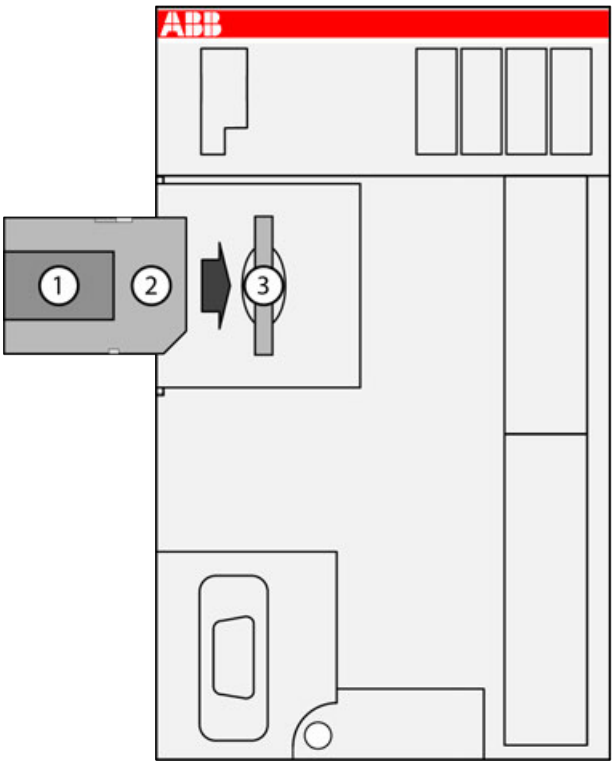


Fig. 1013: Insert micro memory card into PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Remove the micro memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

Remove the micro memory card with micro memory card adapter only when the RUN LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

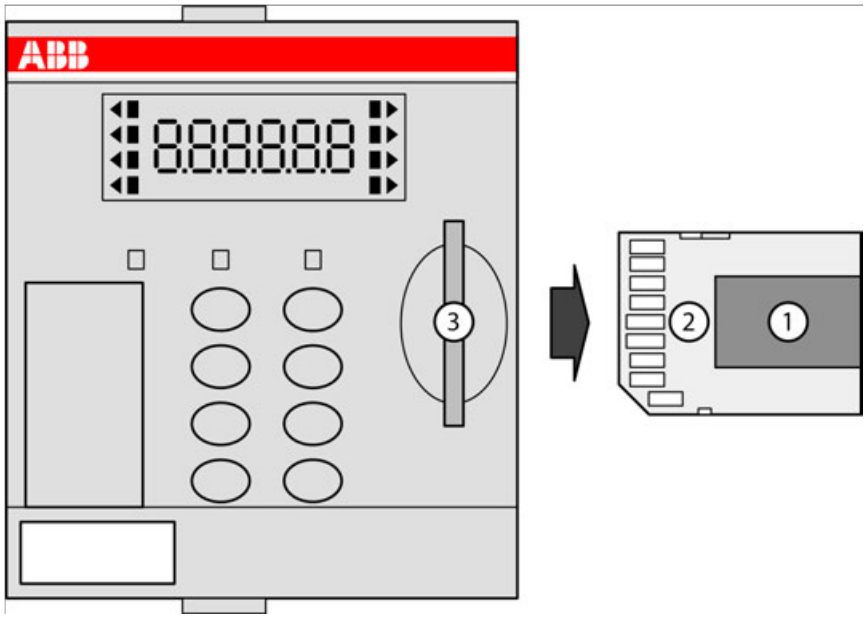


Fig. 1014: Remove micro memory card from PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

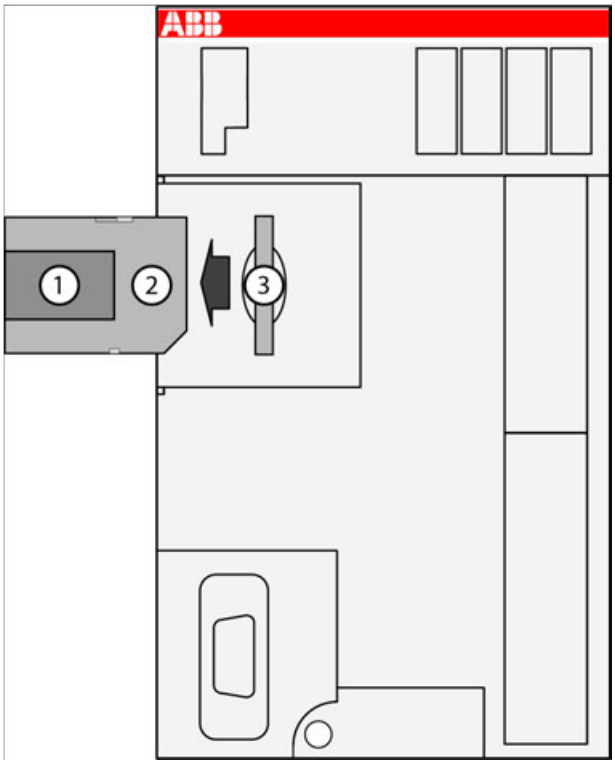


Fig. 1015: Remove micro memory card from PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		8 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

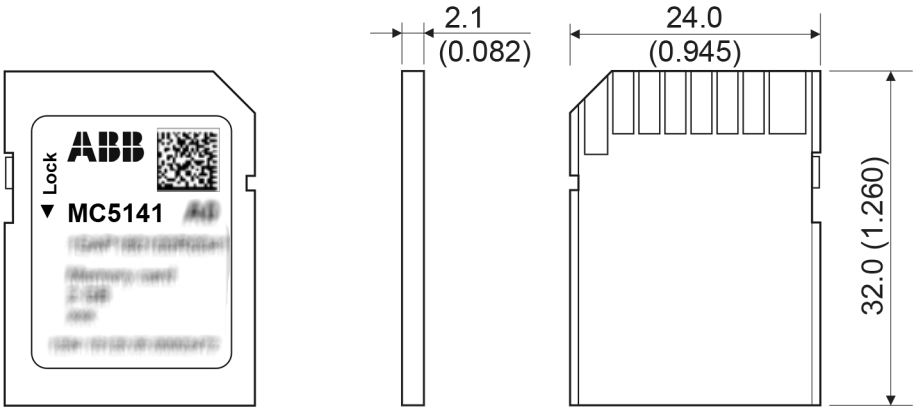
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

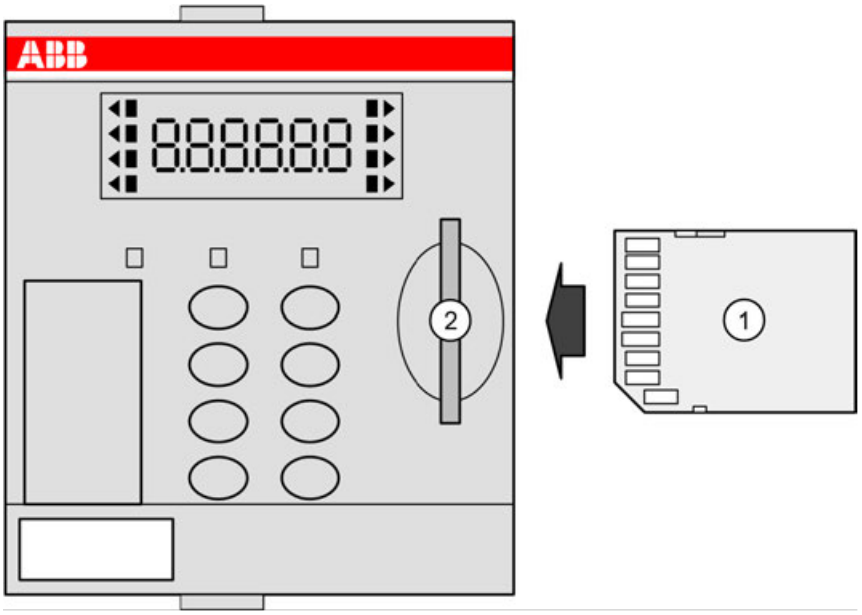


Fig. 1016: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

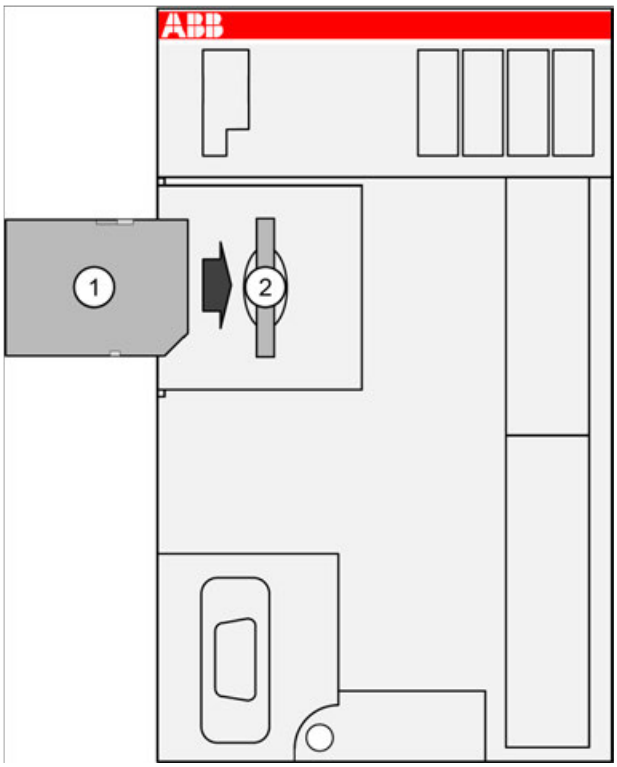


Fig. 1017: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

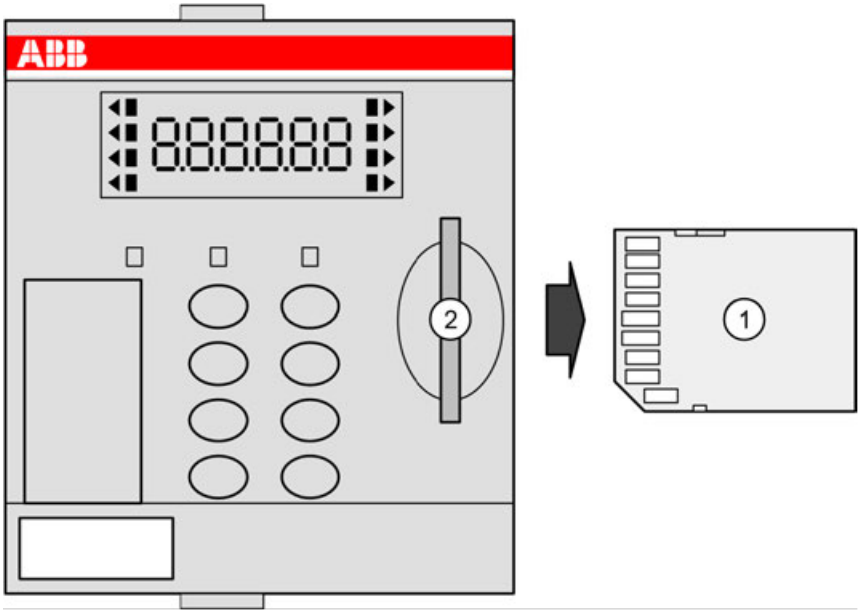


Fig. 1018: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

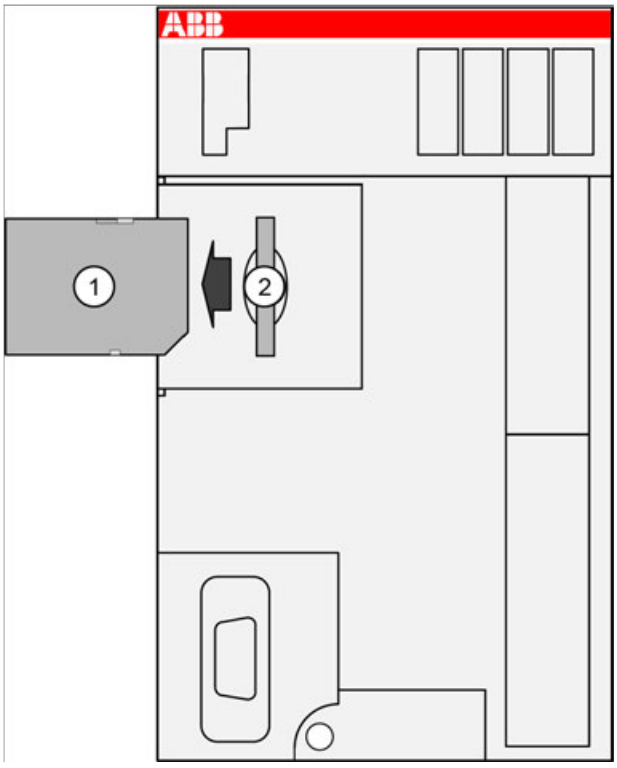


Fig. 1019: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter	Value
Memory capacity	2 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

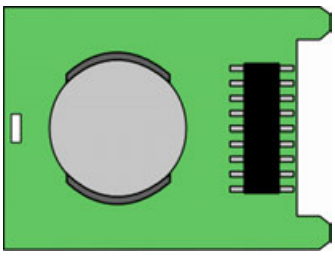
Further information on using the memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA561-RTC - Real-time clock adapter



Intended purpose

The TA561-RTC real-time clock adapter is used for equipping AC500-eCo processor modules with a real-time clock.

The real-time clock can be buffered via an optional standard lithium battery (CR2032) during power supply failures (see lithium battery for real-time clock of AC500-eCo processor modules *Chapter 1.6.2.9.1.1 “CR2032 - Battery for real-time clock” on page 5095*).

Insertion and replacement of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*).

Replacement of the battery

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

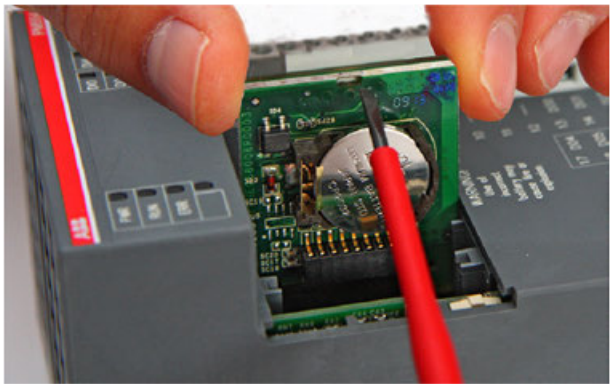
NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

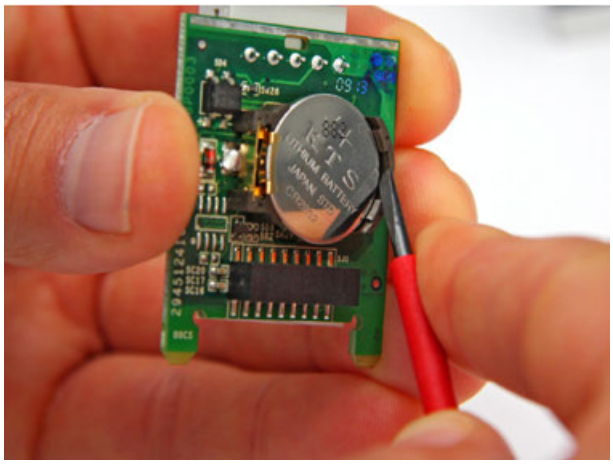
3. Remove the option board from the CPU by lifting it up with a screwdriver.

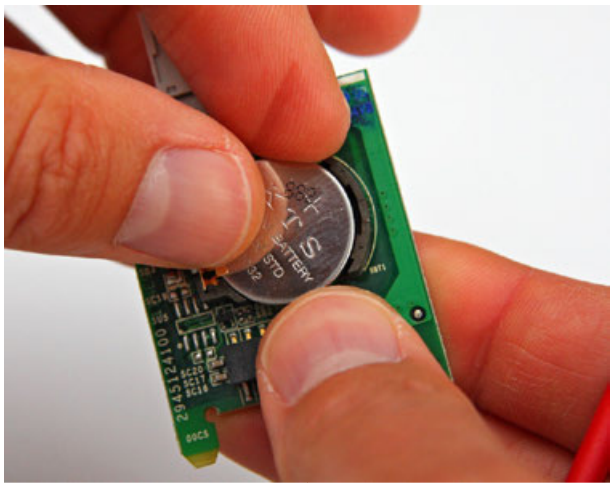


⇨

Remove memory card (if installed) / terminal block (COM2).

4. Remove the battery.





⇨

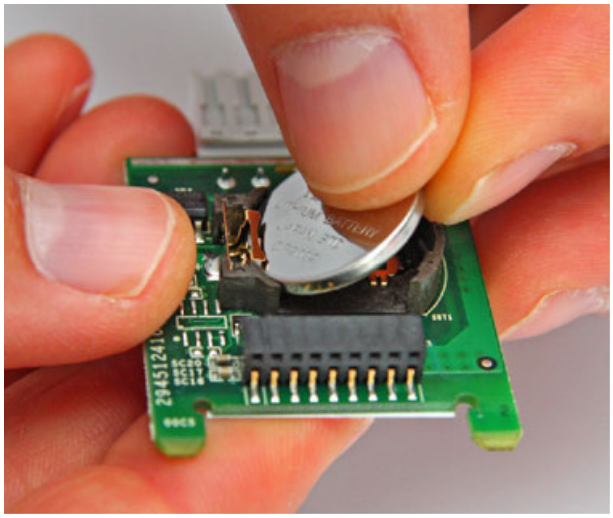
ATTENTION!

Lithium batteries must not be recharged, not be disassembled and not be disposed of in fire.

Exhausted batteries must be recycled to respect the environment.

Dispose of battery properly according to disposal procedures for lithium batteries.

5. Insert replacement battery.



⇨

ATTENTION!

*A standard battery CR2032 can be used for **TA561-RTC** and **TA562-RS-RTC**.*

*Nominal voltage: **3 V DC**.*

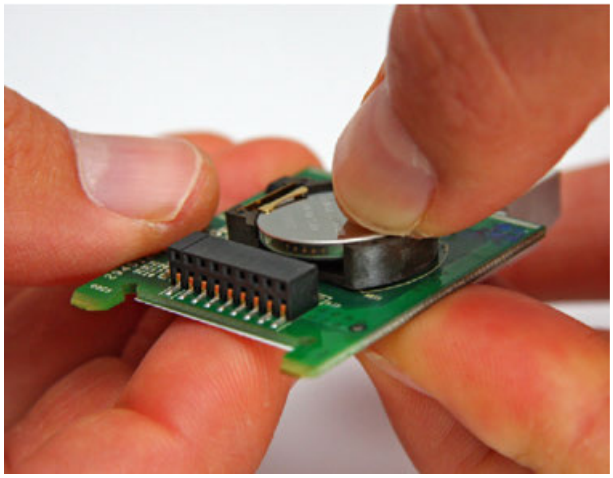
*Required capacity: **230 mAh**.*

*Required temperature range for discharge: **0 °C...+70 °C**.*

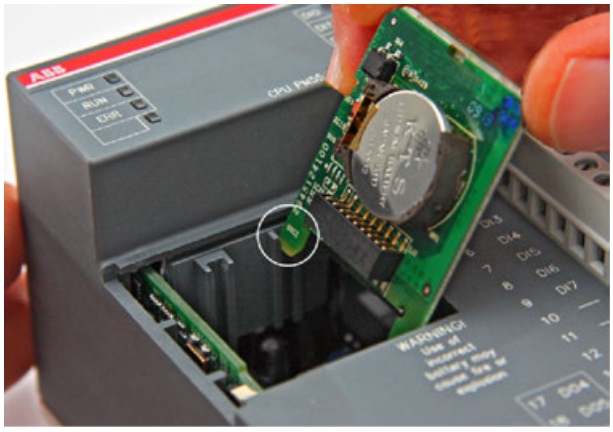
After replacement of the battery, the real-time clock (RTC) date and time must be set again by the user.

Don't use a battery older than 3 years for replacement (e.g. battery kept too long in stock).

Batteries must be stored in a dry place.



6. Insert option board into the CPU.



⇨ Insert the adapter TA56x-RTC into the slot on the right of the CPU.

> *Make sure that the 2 noses of the extension module fit to the holes of the CPU PCB.*
>
> *See white circle in figure above.*

7. Refit the option board slot cover of the CPU.

⇨

> *Remember to re-insert a memory card first if it has been removed previously.*

8. Only now the CPU can be connected to power.

> *Set the time of the real-time clock.*

Technical data

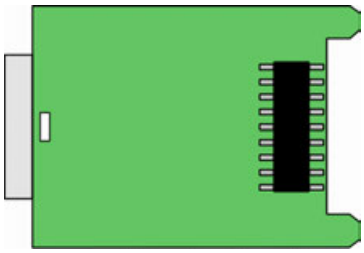
Parameter	Value
RTC accuracy (at 25 °C)	Typ. ±2 s / 24 h

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 181 400 R0001	TA561-RTC, real-time clock adapter for PM55x-xP and PM56x-xP	Active
1TNE 968 901 R3200	TA561-RTC, real-time clock adapter for PM55x-xP and PM56x-xP, lithium battery included (available in China only)	Limited

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA562-RS - Serial RS-485 adapter



Intended purpose

The serial RS-485 adapter is used for equipping AC500-eCo processor modules with a second serial interface COM2. The COM2 interface can be used for:

- online access
- free protocol communication
- Modbus RTU, client and server

> **CAUTION!**
> The serial RS-485 Interface is not galvanically isolated.

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*).

Removal of the option board

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

3. Remove the option board.

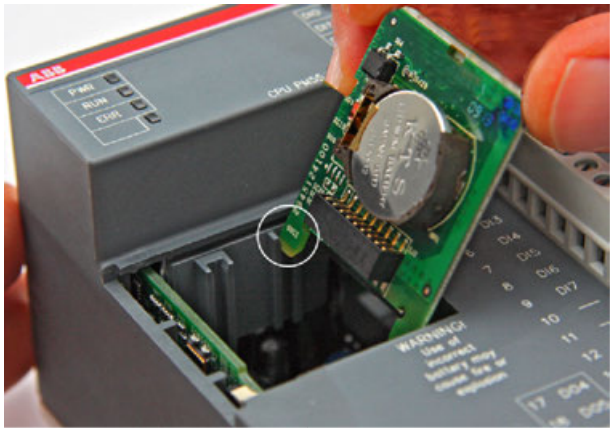
⇨

Remove memory card (if installed) / terminal block (COM2).

Remove the option board from the CPU by lifting it up with a screwdriver.

Insertion of the option board

1. Insert option board into the CPU.



⇨

Make sure that the 2 noses of the expansion module fit to the holes of the CPU PCB.

See white circle in figure above.

2. Refit the option board slot cover of the CPU.

⇨

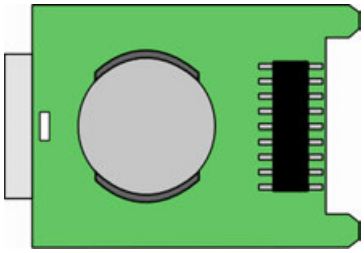
Remember to re-insert a memory card first if it has been removed previously.

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R4300	TA562-RS, serial RS-485 adapter for PM55x/PM56x	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA562-RS-RTC - Serial RS-485 adapter with real-time clock



Intended purpose

The TA562-RS-RTC serial RS-485 adapter with real-time clock is used for equipping AC500-eCo processor modules with a real-time clock and a second serial RS-485 interface COM2.

The real-time clock can be buffered via an optional standard lithium battery (CR2032) during power supply failures (see lithium battery for real-time clock of AC500-eCo processor modules ↳ *Chapter 1.6.2.9.1.1 “CR2032 - Battery for real-time clock” on page 5095*).

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

- Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
- Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules ↳ *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*).

Replacement of the battery

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

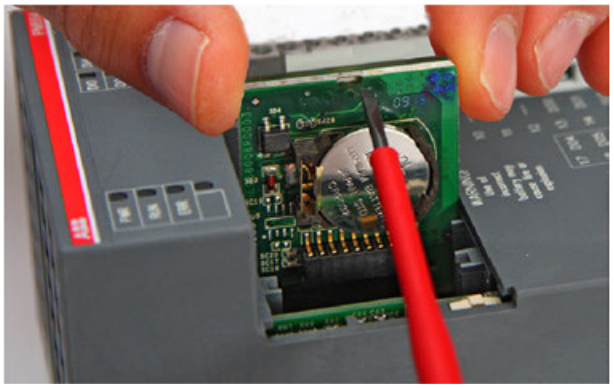
NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

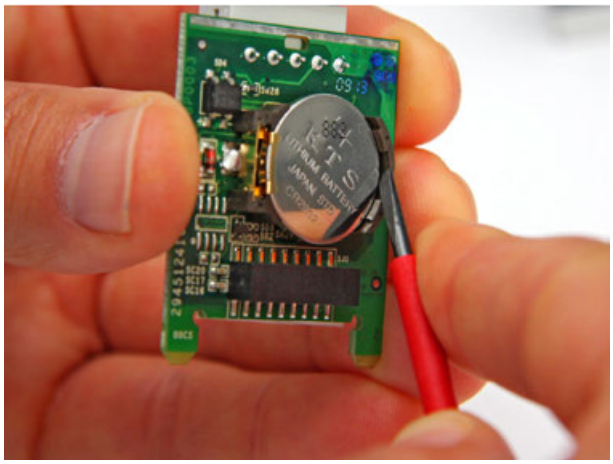
3. Remove the option board from the CPU by lifting it up with a screwdriver.

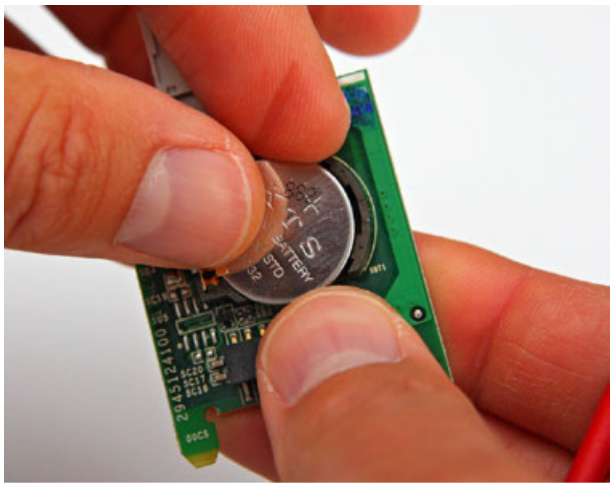


⇨

Remove memory card (if installed) / terminal block (COM2).

4. Remove the battery.





⇨

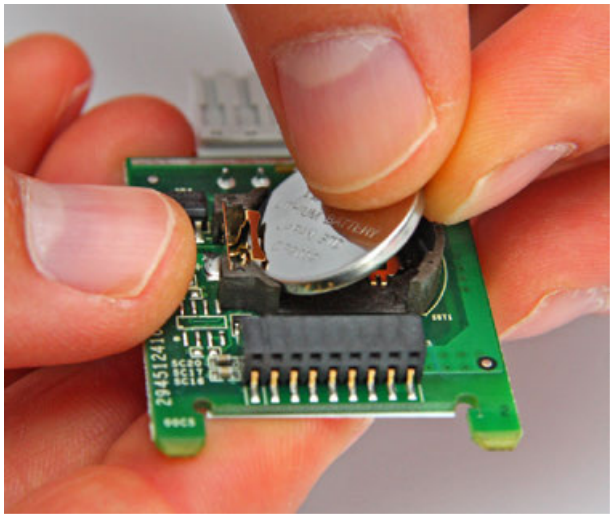
ATTENTION!

Lithium batteries must not be recharged, not be disassembled and not be disposed of in fire.

Exhausted batteries must be recycled to respect the environment.

Dispose of battery properly according to disposal procedures for lithium batteries.

5. Insert replacement battery.



⇨

ATTENTION!

*A standard battery CR2032 can be used for **TA561-RTC** and **TA562-RS-RTC**.*

*Nominal voltage: **3 V DC**.*

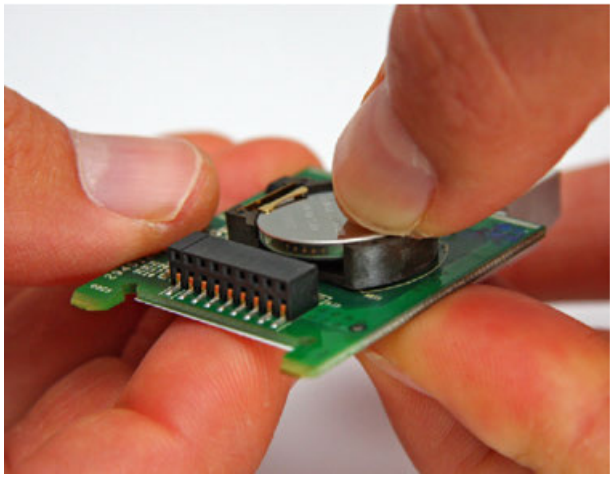
*Required capacity: **230 mAh**.*

*Required temperature range for discharge: **0 °C...+70 °C**.*

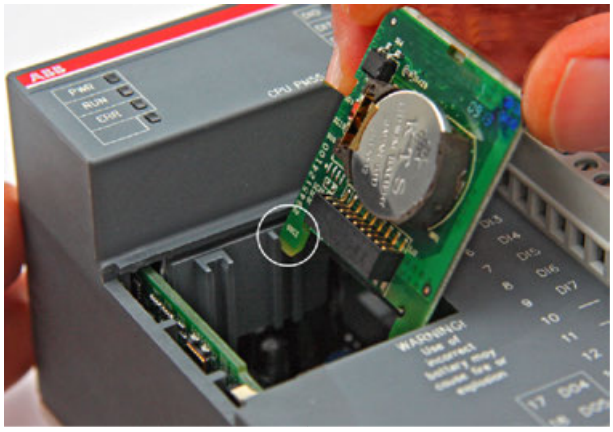
After replacement of the battery, the real-time clock (RTC) date and time must be set again by the user.

Don't use a battery older than 3 years for replacement (e.g. battery kept too long in stock).

Batteries must be stored in a dry place.



6. Insert option board into the CPU.



⇨ Insert the adapter TA56x-RTC into the slot on the right of the CPU.

> *Make sure that the 2 noses of the extension module fit to the holes of the CPU PCB.*
>
> *See white circle in figure above.*

7. Refit the option board slot cover of the CPU.

⇨

> *Remember to re-insert a memory card first if it has been removed previously.*

8. Only now the CPU can be connected to power.

> *Set the time of the real-time clock.*

Technical data

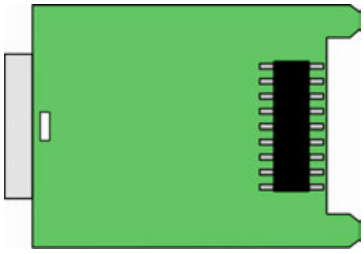
Parameter	Value
RTC accuracy (at 25 °C)	Typ. ±2 s / 24 h

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 181 500 R0001	TA562-RS-RTC, serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP	Active
1TNE 968 901 R5210	TA562-RS-RTC, serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP, lithium battery included (available in China only)	Limited

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA569-RS-ISO - Serial RS-485 isolated adapter



Intended purpose

The TA569-RS-ISO serial RS-485 isolated adapter is used for equipping AC500-eCo processor modules with a second serial interface COM2. The COM2 interface can be used for:

- online access
- free protocol communication
- Modbus RTU, client and server

The serial interface is isolated.

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.2.9.1.10 “TA570 - Spare part set” on page 5136*).

Removal of the option board

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

3. Remove the option board.

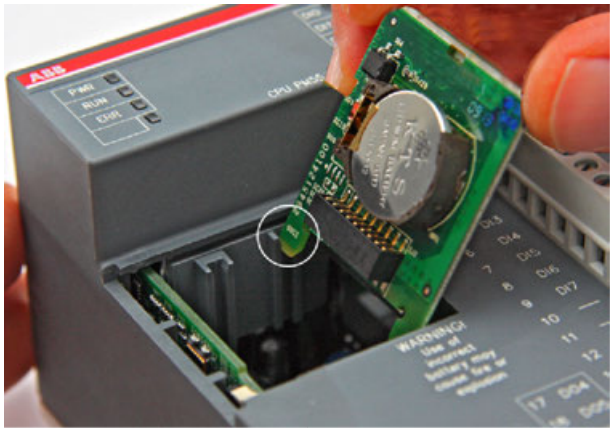
⇨

Remove memory card (if installed) / terminal block (COM2).

Remove the option board from the CPU by lifting it up with a screwdriver.

Insertion of the option board

1. Insert option board into the CPU.



⇨

> *Make sure that the 2 noses of the expansion module fit to the holes of the CPU PCB.*
>
> *See white circle in figure above.*

2. Refit the option board slot cover of the CPU.

⇨

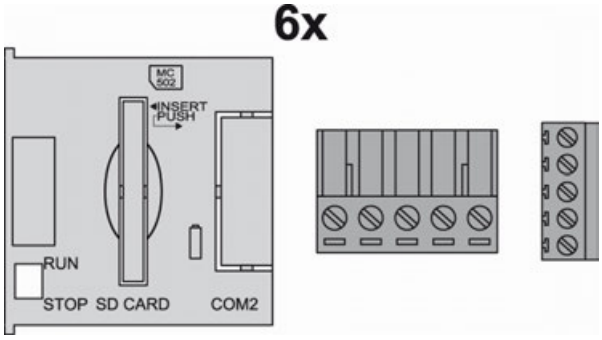
> *Remember to re-insert a memory card first if it has been removed previously.*

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 186 400 R0001	TA569-RS-ISO, serial RS-485 isolated adapter for PM55x/ PM56x	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA570 - Spare part set



Intended purpose

The TA570 spare part set is used to replace lost or damaged parts of AC500-eCo processor modules. It contains the following parts:

- Option board slot cover
- Terminal block for power supply
- Terminal block for serial RS-485 adapter

Every spare is included 6x inside TA570.

Technical data

Table 581: Option board slot cover

Parameter	Value
Weight	5 g
Dimensions	40 mm x 40 mm x 3 mm

Table 582: Terminal block for power supply

Parameter		Value
Type		Screw clamp plug, wire connection from front
Usage		For AC500-eCo processor modules
Conductor cross section		
	Solid	0.2 mm^2...2.5 mm^2
	Flexible (with wire-end ferrule only)	0.2 mm^2...2.5 mm^2
Stripped conductor end		7 mm...8 mm
Fastening torque		0.5 Nm
Degree of protection		IP20

Parameter	Value
Dimensions	25.4 mm x 17.4 mm x 15.1 mm
Weight	5 g

Table 583: Terminal block for serial RS-485 adapter

Parameter		Value
Type		Screw clamp plug, wire connection from side
Usage for		*Chapter 1.6.2.9.1.7 “TA562-RS - Serial RS-485 adapter ” on page 5120* *Chapter 1.6.2.9.1.9 “TA569-RS-ISO - Serial RS-485 isolated adapter” on page 5131* *Chapter 1.6.2.9.1.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5125*
Conductor cross section		
	Solid	0.14 mm^2...1.5 mm^2
	Flexible (with wire-end ferrule only)	0.14 mm^2...1.5 mm^2
Stripped conductor end		7 mm
Fastening torque		0.4 Nm
Degree of protection		IP20
Dimensions		19.05 mm x 8.7 mm x 19.1 mm
Weight		5 g

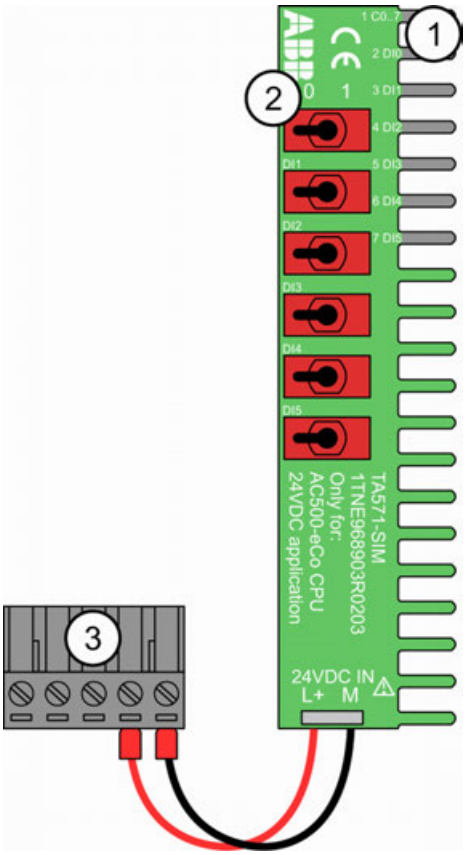
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3203	TA570, spare part set for AC500-eCo processor modules, 3x6 pieces	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA571-SIM - Input simulator

- Input Simulator for 6 digital inputs 24 V DC
- For usage with AC500-eCo processor modules

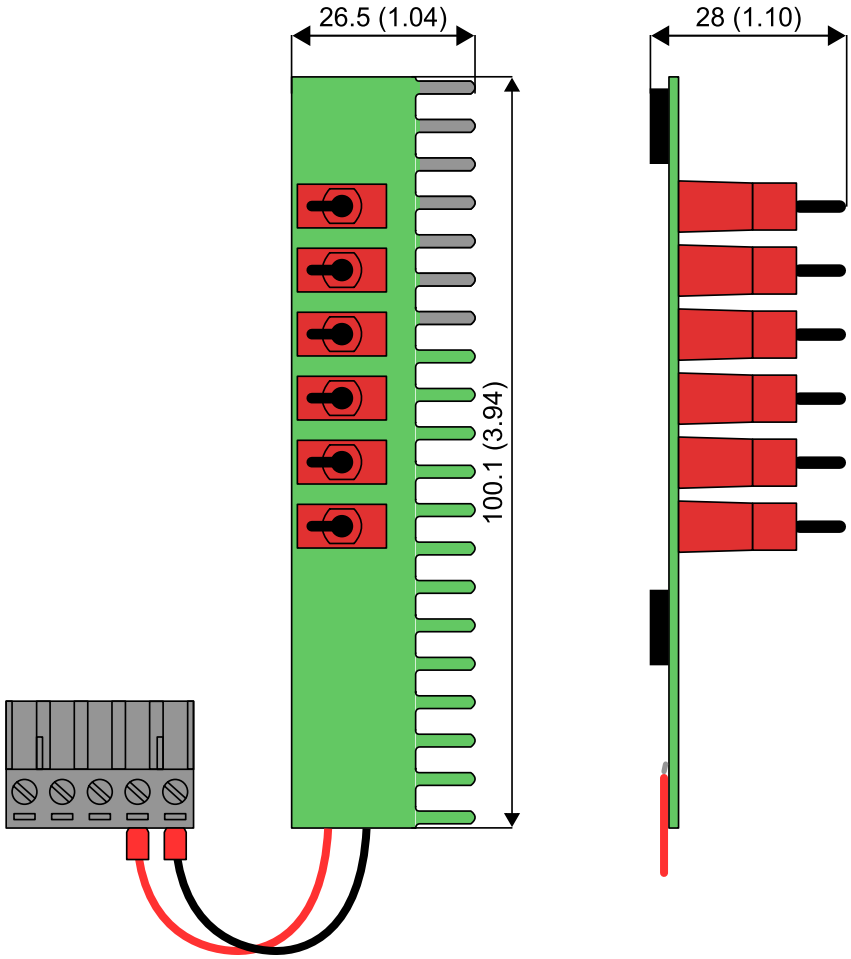


1 Contacts to connect to clamps of onboard I/Os
2 6 switches for the digital inputs DI0 ... DI5 (0 means opened switch, 1 means closed switch)
3 Terminal block for power supply connector of processor module PM55x/PM56x

Intended purpose

The input simulator TA571-SIM is used for test and training purposes with AC500-eCo processor modules PM55x and PM56x. It can simulate 6 digital 24 V DC input signals to the digital inputs DI0...DI5 of onboard I/Os.

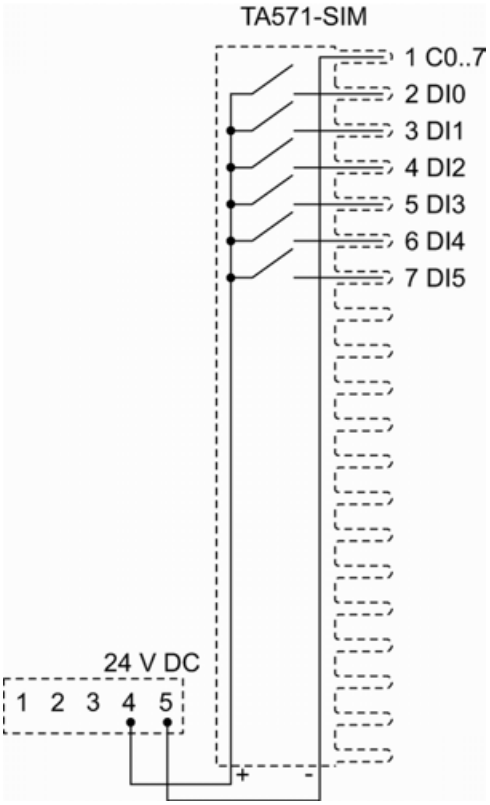
Dimensions



The dimensions are in mm and in brackets in inch.

Electrical diagram

The diagram below shows the connection of the input simulator.



Mounting

Insertion of the input simulator

1. Make sure that the power supply of the processor module is turned off.

CAUTION!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overvoltages and short circuits. Make sure that all voltage sources (supply and process voltage) are switched off before you begin with operations on the system. Never connect any voltages > 24 V DC to pins 4/5 of the terminal block of input simulator TA571-SIM.

CAUTION!

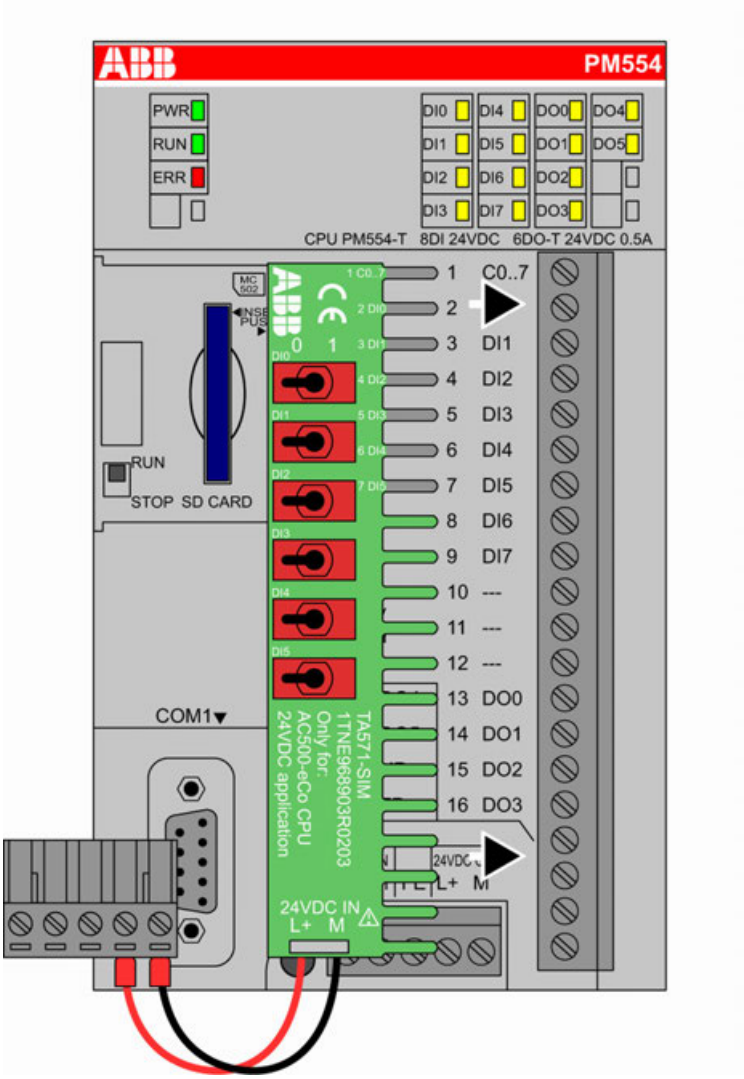
Risk of damaging the input simulator or PLC modules!

The input simulator must only be used with AC500-eCo processor modules PM55x and PM56x. Never use the input simulator with other devices.

The input simulator must only be used for test and training purposes. Never use it within productive plants.

2. Remove the terminal block for power supply from the processor module by a flat-blade screwdriver.
3. Make sure that all clamps of the onboard I/Os are totally open.
4. Use a flat-blade screwdriver to unplug the terminal block for power supply of the processor module.

5. Insert the input simulator as shown in the figure.



6. Tighten all screws of the onboard I/O clamps (max. torque 1.2 Nm).
7. Plug in the terminal block for power supply of the TA571-SIM to the connector of the processor module.
8. Connect the CPU power supply wires (24 V DC or 100-240 V AC).

Usage

With input simulator TA571-SIM, the digital 24 V DC inputs DI0...DI5 of can be turned OFF and ON separately:

- If the lever of the switch is on the right side, the input is ON.
- If the lever of the switch is on the left side, the input is OFF.

Removal

Removal of the input simulator

1. Make sure that the power supply of the processor module is turned off.

CAUTION!

Risk of damaging the PLC modules!

The PLC modules can be damaged by overvoltages and short circuits. Make sure that all voltage sources (supply and process voltage) are switched off before you begin with operations on the system.

2. Disconnect the processor module power supply wires (24 V DC or 100-240 V AC) from the terminal block for power supply.
3. Unplug the terminal block for power supply with a flat-blade screwdriver of the power connector.
4. Loosen all screws of the onboard I/Os.
5. Remove the input simulator by pulling it to the left side.

Technical data

Table 584: Technical data of the module

Parameter		Value
Process Supply Voltage		
	Connections	Terminal 4 (L+) for +24 V DC and terminal 5 (M) for 0 V DC
	Rated value	24 V DC
	Max. ripple	5 %
	Protection against reversed voltage	Yes
Galvanic isolation		Yes, per module
Isolated Groups		1 (6 channels per group)
Weight		On request
Mounting position		Horizontal or vertical

Table 585: Technical data of the inputs

Parameter	Value
Number of channels per module	6 digital input channels (+24 V DC)
Distribution of the channels into groups	1 (6 channels per group)
Connections of channels DI0 to DI5	Terminals 2...7
Reference potential for the channels DI0 to DI5	Terminal 1 (negative pole of the process supply voltage, signal name C0...7)
Input current per active channel (at input voltage +24 V DC) The current is given through the used processor module.	Typ. 5 mA
Inrush current per active channel The current is given through the used processor module.	Typ. 5 mA

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 903 R0203	TA571-SIM, input simulator for PM55x and PM56x	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TK504 - COM2 USB programming cable

- PC-side: USB connector type A
- AC500-side: 5-pole terminal block
- Length 3 m

Intended purpose

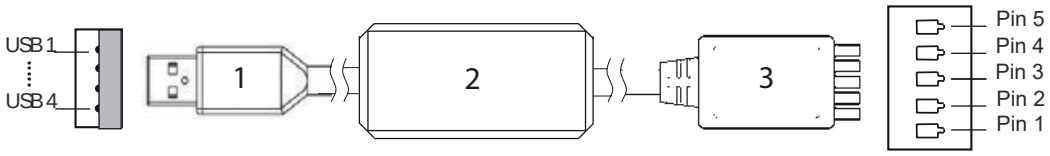
TK504 programming cable connects the USB interface of a PC with the serial interface of processor module PM55x and PM56x. It is used for programming purposes.

CAUTION!
Risk of communication faults!
The mechanical connection of TK504 may get lost due to mechanical vibration.

Use TK504 only for programming and debugging. A permanent usage is not foreseen.

With AC500/AC500-eCo processor modules, only the ABB programming cables TK50x can be used. Other cables may cause communication faults and must not be used.

Connections



1 USB connector type A (PC side)
2 USB/RS-485 converter
3 Terminal block, 5-pin, (RS-485, PLC side)

Table 586: TK504 programming cable wiring USB pin

USB pin	Signal	Description
USB 1	VBUS	USB power
USB 2	-D	USB data negative
USB 3	+D	USB data positive
USB 4	GND	Ground

Table 587: TK504 programming cable wiring Terminal block, 5-pin

Pin	Signal	Description
Pin 1	-	Not connected
Pin 2	RXD/TXD-P	Receive/Transmit positive
Pin 3	RXD/TXD-N	Receive/Transmit negative
Pin 4	-	Not connected
Pin 5	FE	Functional earth

1. Install the device driver for the programming cable (see).

> *Once you have installed the device driver of the cable in your Windows system, make sure that you always use the same USB port on your computer.*
>
> *Otherwise, Windows will ask you to install the driver a second time if you connect the cable to a different USB port of your computer.*

2. Connect the 5-pole terminal block of the TK504 to the processor module *Chapter 1.6.3.5.4.3 “Serial interface COM2” on page 5254*.
3. Plug the USB connector to an USB interface at your PC.

Technical data

Parameter	Value
Connector at the PC (USB interface)	USB connector type A
Connector at the processor module	Single conductors
Length	3 m
Cable type	Programming cable
Weight	0.4 kg

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R2100	TK504, COM2 USB programming cable -> single conductors, length 3 m	Active

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Installation of cable driver

OPERATION INSTRUCTION

PROGRAMMING CABLE TK503 / TK504

USB DRIVER INSTALLATION

Contents

1 Introduction and Basics

1.1 Intended Use

The TK503/TK504 programming cable can be used to operate and to configure the PLC via a PC or laptop. For this, CODESYS software, driver and utility programs must be installed and a TK503 or TK504 programming cable must be connected.

NOTICE!

The TK503/TK504 programming cable cannot be used for AC500 V3 Processor Modules.

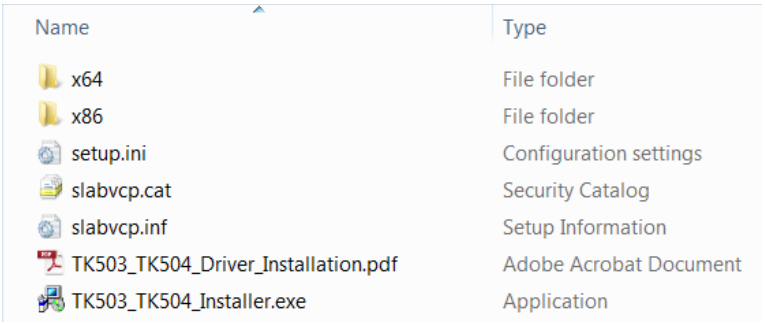
1.2 PC System Requirements

- Platform: Microsoft Windows Vista, Windows 7, Windows 10
- CD-ROM drive
- USB port available for connecting the TK503/TK504 programming cable

NOTICE!

Microsoft, Windows and the Windows logo are trademarks of Microsoft Corporation in the USA and/or other countries. All other product and company names are trademarks of their respective owners.

1.3 Content of the Installation Package



2 Installation

2.1 Installation Steps

Before you can use the TK503/TK504 programming cable, the appropriate USB driver must be installed on your PC or laptop.

The driver for the TK503/TK504 programming cable is installed in two steps:

– Pre-installation of the driver on your PC using the program *TK503_TK504_Installer.exe*.

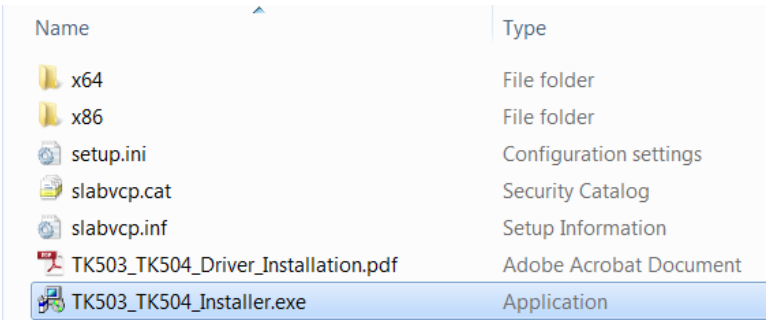
– Installation of the new hardware in Windows after the TK503 programming cable or TK504 programming cable is plugged in for the first time.

NOTICE!

Before you connect the TK503/TK504 programming cable with the PC, install the USB driver first.

2.2 Pre-Installation Routine

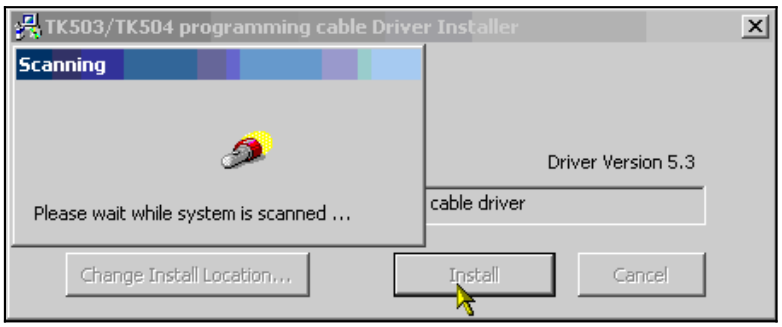
1. Uninstall all existing versions of the driver software.
2. Start the pre-installation of the driver by calling *TK503_TK504_Installer.exe.*



NOTICE!

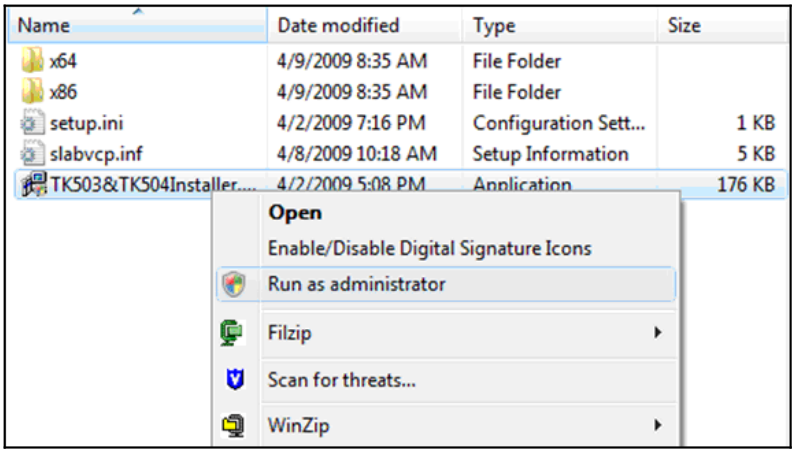
You must have administrator rights to run the installation.

3. Define the installation directory and click **Install**.



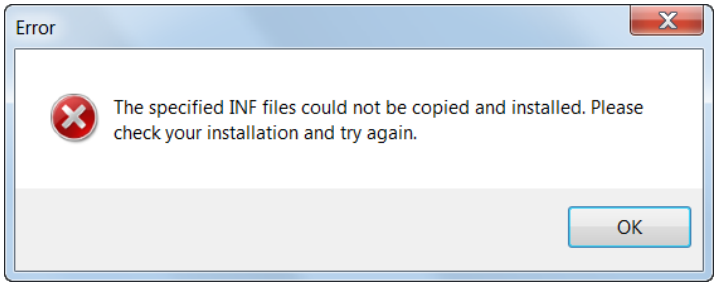
Windows Vista users only:

Start the *TK503&TK504Installer.exe* with the **Run as administrator** option, even if you have administrator rights. Acknowledge the following dialog with **Allow**.

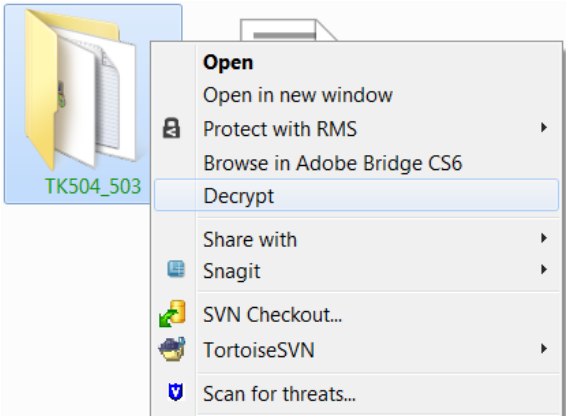


Windows 7 users only:

Windows will display an error message after clicking **Install**.



On this condition, decrypt the installation folder:



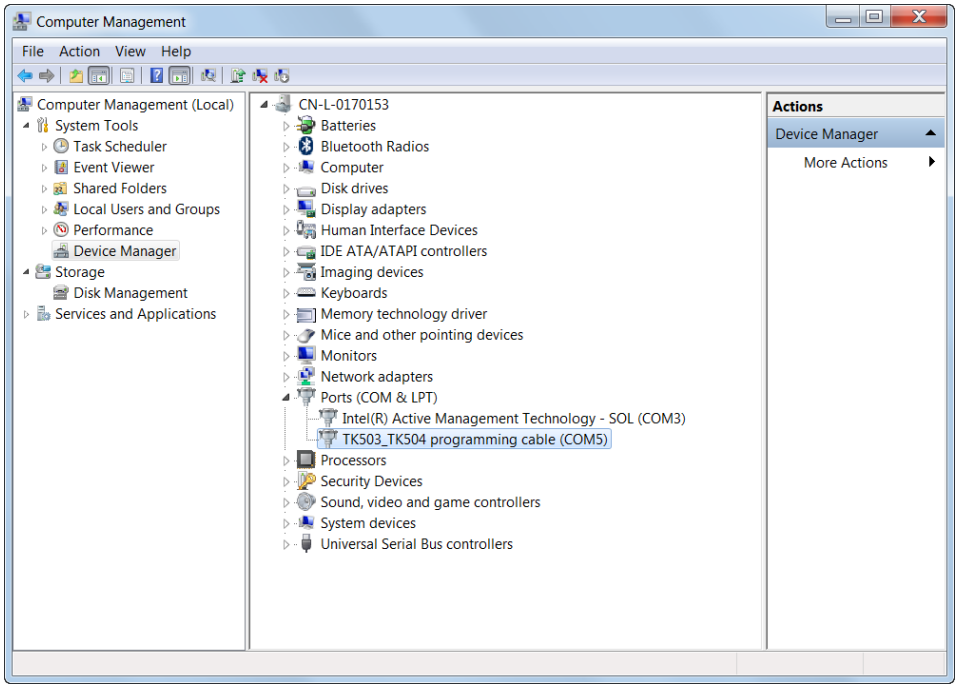
Then, start *TK503_TK504_Installer.exe* with the **Run as administrator** option again.

3 Communication

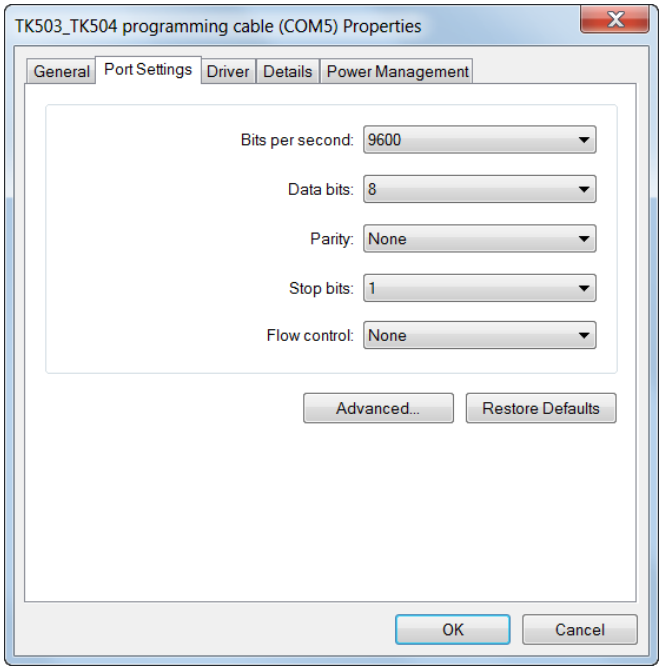
3.1 Virtual Communication Port Configuration

If the TK503/TK504 programming cable is plugged in a USB interface, Windows creates a virtual communication port (COM port).

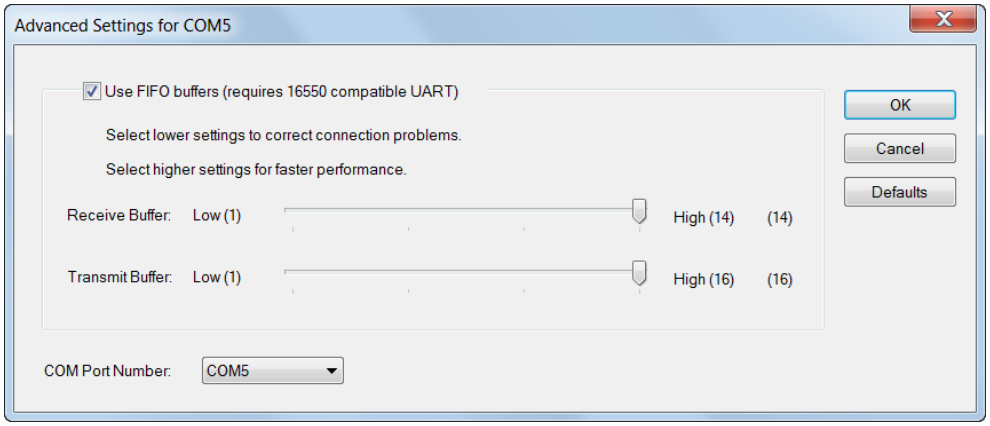
All communication ports can be viewed in the Windows Control Panel under Device Manager.



4. In the Ports settings click **Properties** to set the baud rate.



5. Set the COM port number under **Advanced** (up to COM32).

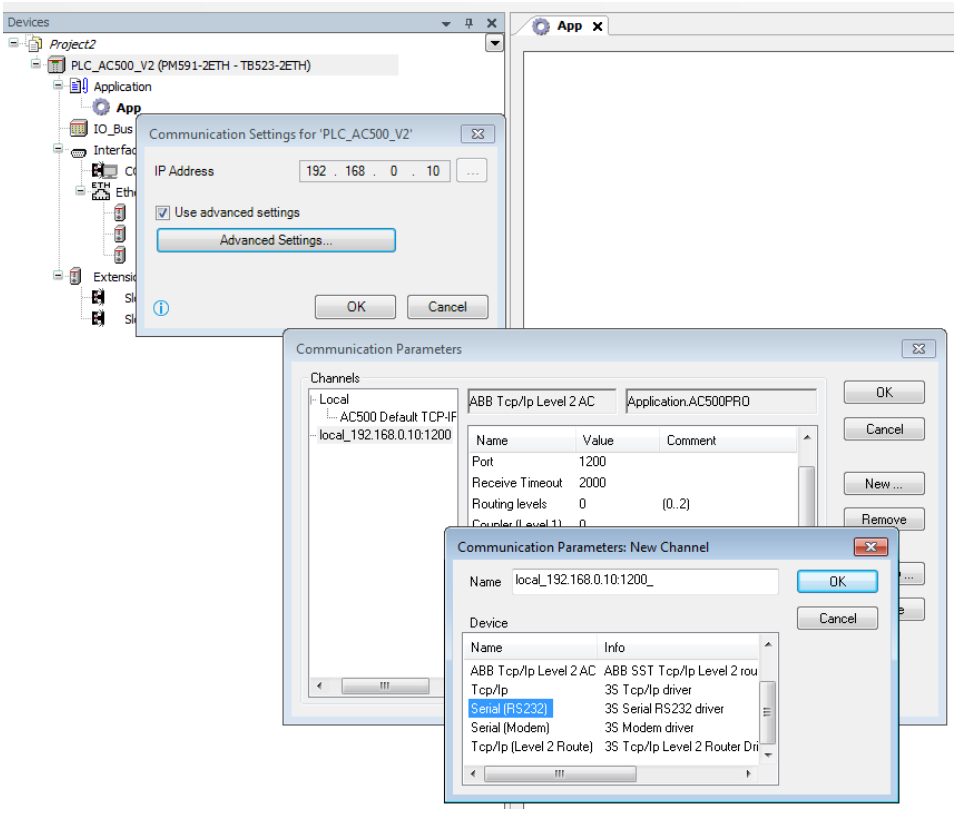


NOTICE!

When configuring the communication connection in CODESYS, the baud rate can also be set separately for each COM connection.

4 Automation Builder Communication

1. Install TK503/TK504 programming cable driver.
2. Connect the TK503 or TK504 programming cable to aPC or laptop. Windows detects the new hardware – complete the installation.
3. Start Automation Builder and open the project.
4. Right-click the PLC root node and select **Communication Parameters.**
5. Select the new virtual COM port.

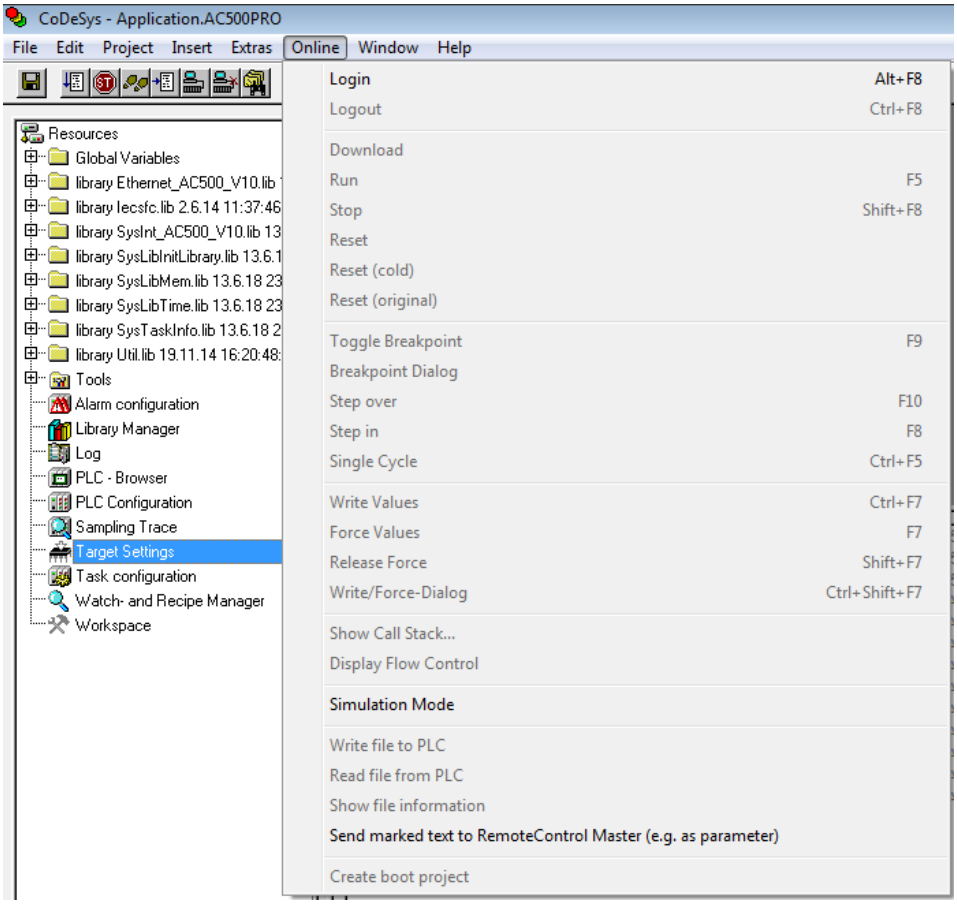


NOTICE!

The port number must be the same as the port number in the Device manager – Port – TK503/TK504 programming cable (COMx). Otherwise the communication cannot be built up.

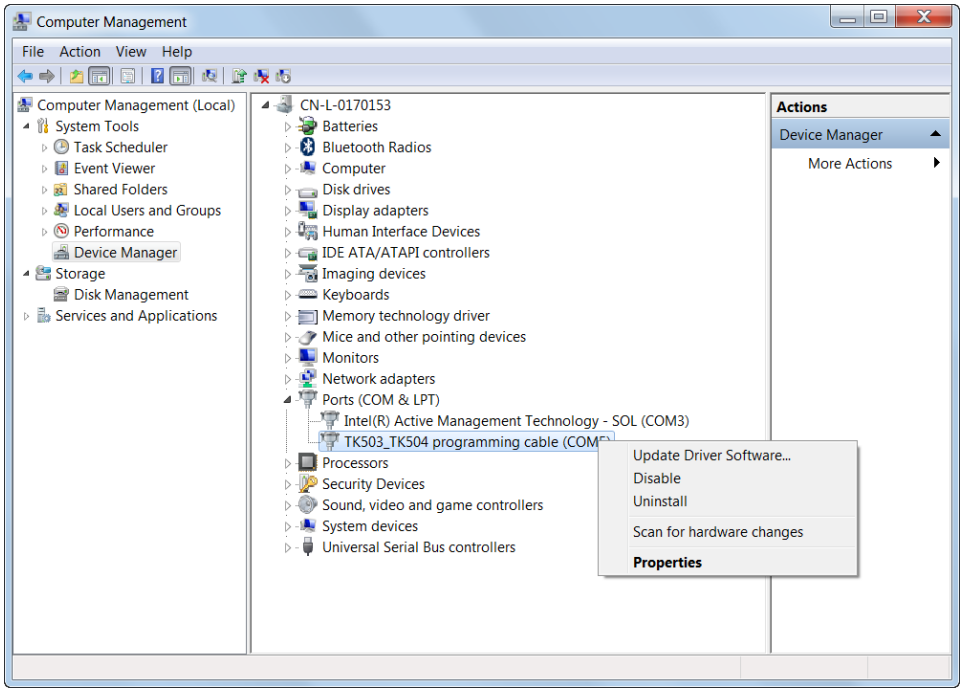
The number of COM ports depends on the availability on your computer.
The baud rate can be selected between 19200 and 115200 bps.

6. In CODESYS, create the communication between Automation Builder and the PLC.


CoDeSys - Application.AC500PRO
File Edit Project Insert Extras Online Window Help
Resources
Global Variables
library Ethernet_AC500_V10.lib
library Iecsfc.lib 2.6.14 11:37:46
library SysInt_AC500_V10.lib 13
library SysLibInitLibrary.lib 13.6.1
library SysLibMem.lib 13.6.18 23
library SysLibTime.lib 13.6.18 23
library SysTaskInfo.lib 13.6.18 2
library Util.lib 19.11.14 16:20:48
Tools
Alarm configuration
Library Manager
Log
PLC - Browser
PLC Configuration
Sampling Trace
Target Settings
Task configuration
Watch- and Recipe Manager
Workspace
Login Alt+F8
Logout Ctrl+F8
Download
Run F5
Stop Shift+F8
Reset
Reset (cold)
Reset (original)
Toggle Breakpoint F9
Breakpoint Dialog
Step over F10
Step in F8
Single Cycle Ctrl+F5
Write Values Ctrl+F7
Force Values F7
Release Force Shift+F7
Write/Force-Dialog Ctrl+Shift+F7
Show Call Stack...
Display Flow Control
Simulation Mode
Write file to PLC
Read file from PLC
Show file information
Send marked text to RemoteControl Master (e.g. as parameter)
Create boot project

5 Uninstallation / Update

1. In the Windows Control Panel open the Device Manger.
2. Right-click on the entry**TK503/TK504 programming cable** and select **Uninstall** or **Update Driver**.

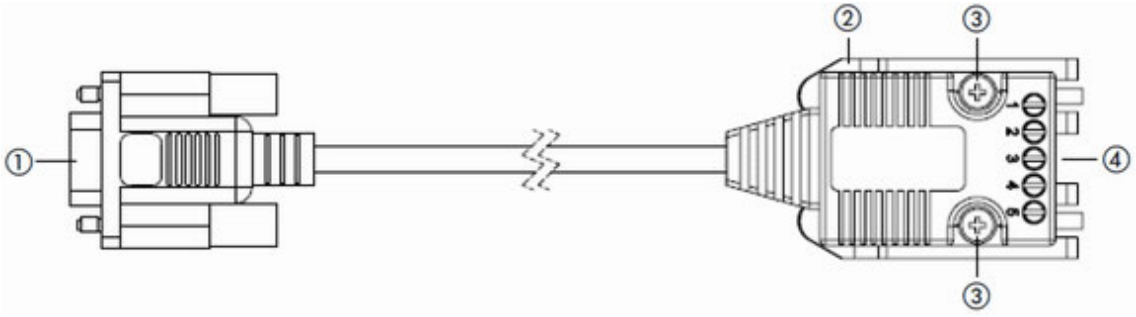


Document Number: 3ADR010872, 1, en US

ABB AG
Eppelheimer Straße 82
69123 Heidelberg, Germany
Phone: +49 62 21 701 1444
Fax : +49 62 21 701 1382
E-Mail: plc.support@de.abb.com
www.abb.com/plc

TK506 - RS-485 isolator for COM1

- Isolated side: 5-pin terminal
- AC500-eCo-side: D-sub 9-pin, male
- Length 0.6 m



1 D-sub 9-pin, male, RS-485
2 DIN rail mounting spring
3 Holes for mounting with 2x M4 screws
4 5-pin terminal, screw-type, RS-485

Intended purpose

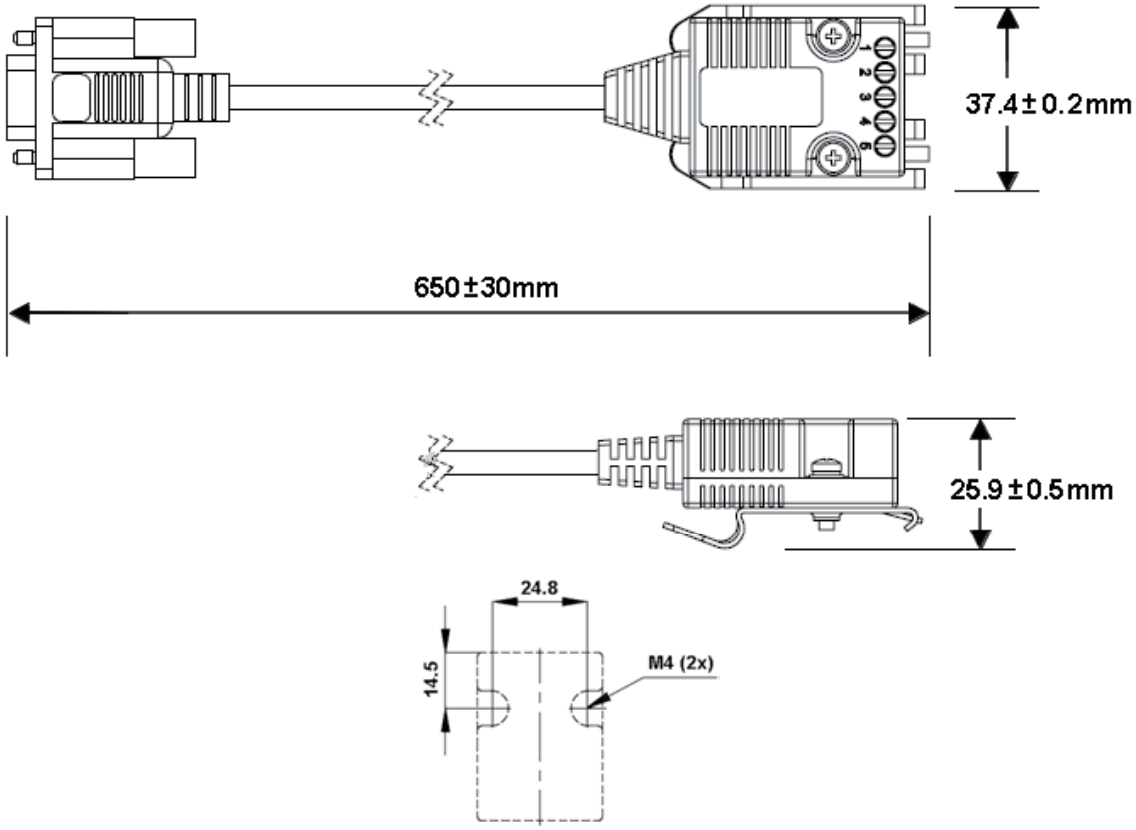
The TK506 RS-485 isolator for COM1 of processor modules PM55x and PM56x allows longer cable length for serial communication. The product can be used for the communication protocols Modbus RTU or CS31 bus.

The TK506 RS-485 isolator supports the processor modules PM55x and PM56x with the following ordering numbers and version indices:

– *1TNE968900Rxxxx with version index ≥ A3 (see figure below)*
– *1SAP12xx00Rxxxx independent of the version index* ↳ *Table 269 “Processor modules for AC500-eCo” on page 3818*

The isolator provides galvanic isolation of the RS-485 communication signals. It is supplied via the 3.3 V output of the COM1 interface of the processor module. The isolator automatically detects and follows serial data flow direction changes. It is adapted to communication speeds up to 187.5 kBaud.

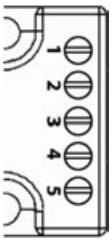
Dimensions



The dimensions are in mm and in brackets in inch.

Connections

Connection: Interface

	PIN	Signal	Description
	1	Terminator P	Terminator positive
	2	RxD/TxD-P	Receive/transmit positive
	3	RxD/TxD-N	Receive/transmit negative
	4	Terminator N	Terminator negative
	5	FE	Functional earth (internally connected to DIN rail spring)

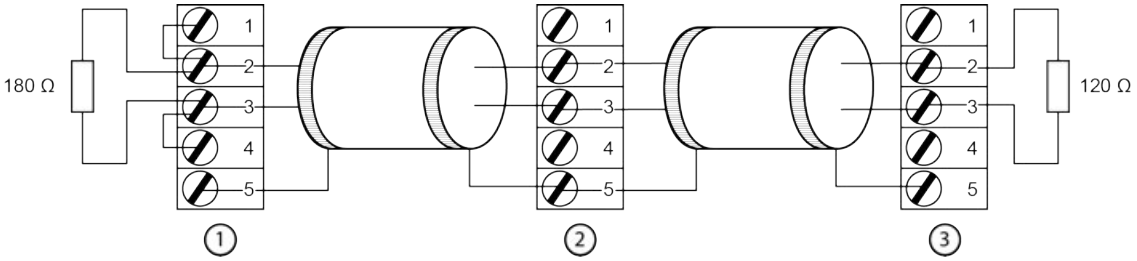
RS-485 communication requires an electrical termination of the communication line. The following is necessary:

- 2 suitable resistors at both line ends (to avoid signal reflections)
- a pull-up resistor at RxD/TxD-P and a pull-down resistor at RxD/TxD-N. These 2 resistors care for a defined high level on the bus, while there is no data exchange.

In every RS-485 network 1 pull-up and pull-down resistors must be activated. It is recommended to activate the pull-up and the pull-down resistors at the bus master. These 2 resistors are integrated inside the TK506 RS-485 isolator. They can be activated by connecting the terminals 1-2 and 3-4 of the terminal block with cable bridges.

Master at the bus line end

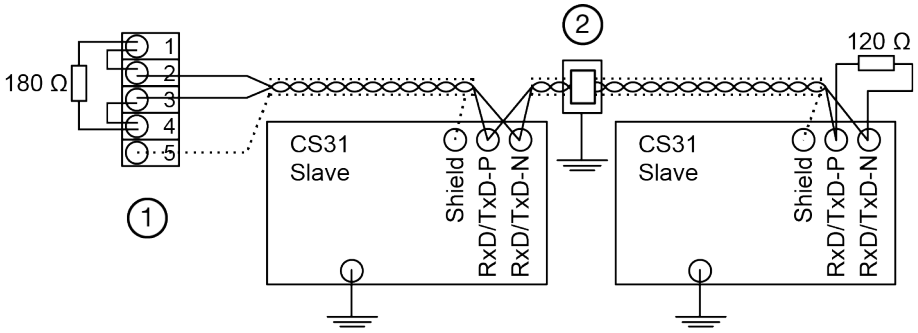
The following figure shows a RS-485 bus with the master at the end of bus line.



1 Master at the end of bus line, pull-up and pull-down resistors are activated, bus termination with 180 Ω resistor
2 Slave within the bus line
3 Slave at the end of bus line, bus termination with 120 Ω resistor

Connection: CS31 protocol

The following figure shows an CS31 bus with the master at the end of bus line.



1 Master at the end of bus line, pull-up and pull-down activated, bus termination with 180 Ω resistor
2 Direct grounding clip or steel plate

NOTICE!
Risk of EMC disturbances!
Unshielded cables may cause EMC disturbances.
Always use shielded cables and connect the shield at every device.

Technical data

Parameter		Value
Physical link		RS-485
Galvanic isolation		Yes
Usage / Supported protocols		Modbus (Master and Slave) CS31 (Master only)
Supported transmission rates [baud]		
	Modbus	9.6 k, 14.4 k, 19.2 k, 38.4 k and 187.5 k
	CS31 bus	187.5 k
Connector at the communication line		5-pin screw terminal block
Connector at PM554 or PM564		D-sub 9, male

Parameter		Value
Cable type and specification		Twist rate minimum 10 per meter, with common shield Capacitance between the cores: < 55 nF/km Characteristic impedance: 120 Ω
	Recommended cable cross section	Conductor cross section 0.5 mm² Resistance per core: < 40 Ω/km
	Thinnest cable cross section	Conductor cross section 0.22 mm² Resistance per core: < 100 Ω/km
Max. cable length for Modbus		
	at 19.2 kBaud	500 m with cable cross section 0.5 mm² or 400 m with cable cross section 0.22 mm²
Max. cable length for CS31 bus		500 m with cable cross section 0.5 mm² or 400 m with cable cross section 0.22 mm²
Specification for external terminating resistor		120 Ω, 1 %, ≥ 0.25 W or 180 Ω, 1 %, ≥ 0.25 W
Length		0.6 m
Weight		80 g
Isolation voltage		500 V DC (type test)
Surge voltage (common mode)		1000 V (type test)

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 186 100 R0001	TK506, RS-485 isolator for COM1, D-sub 5 terminal	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.9.2 AC500 (standard)

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

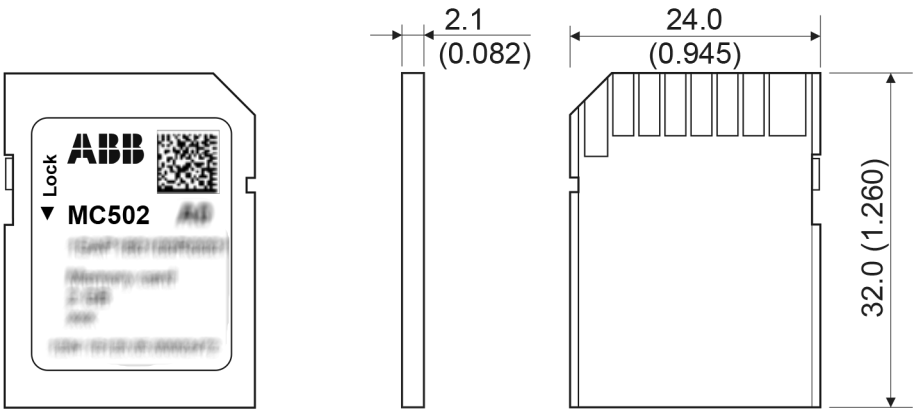
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

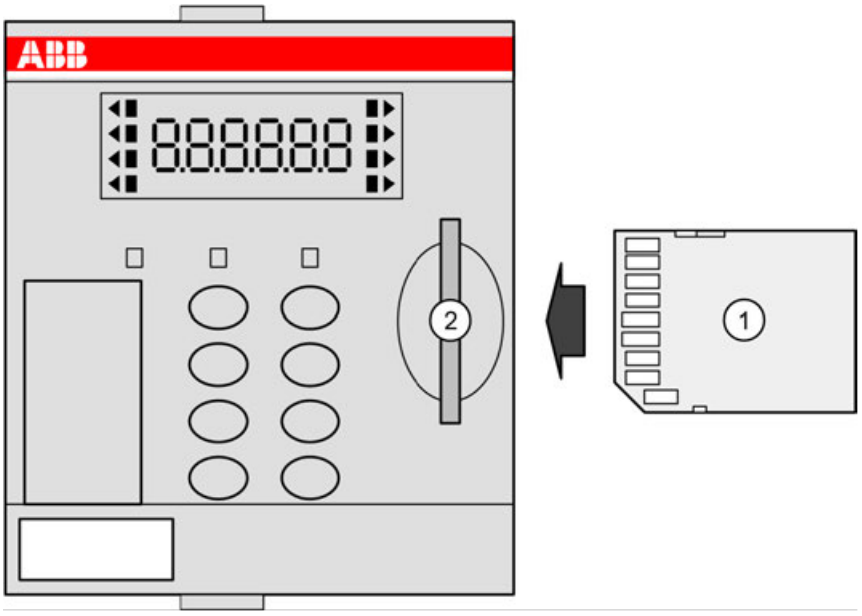


Fig. 1020: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

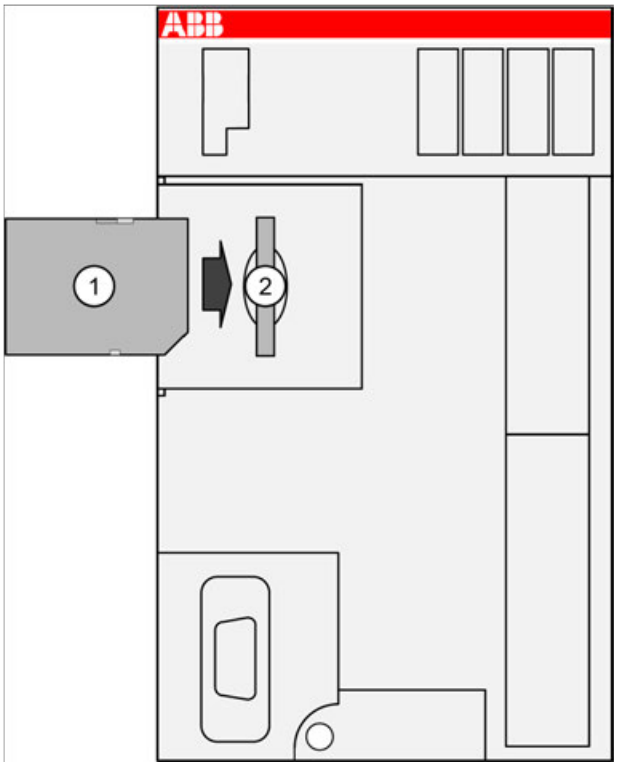


Fig. 1021: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

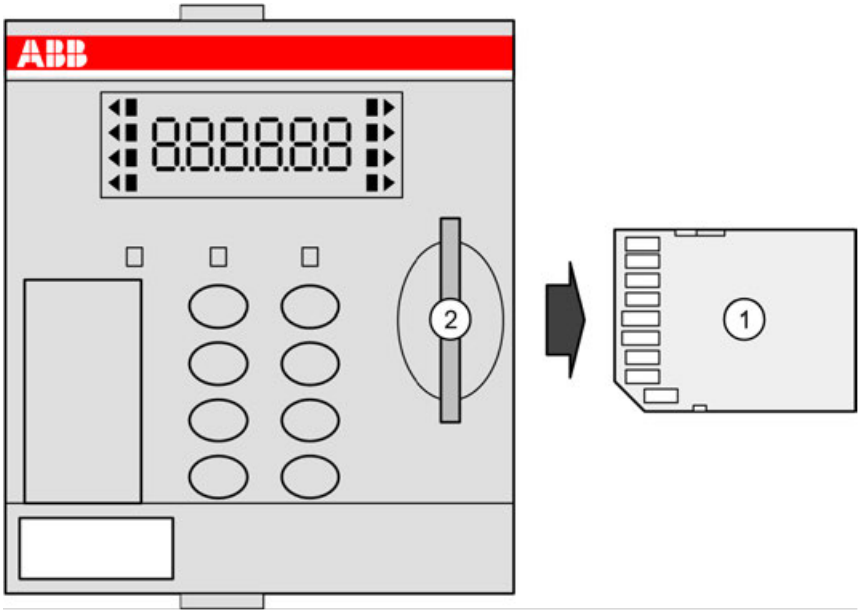


Fig. 1022: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

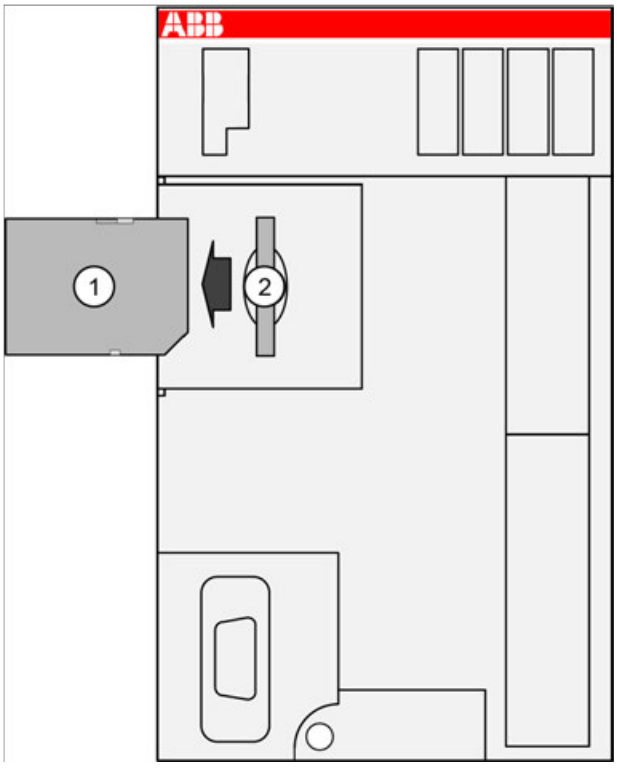


Fig. 1023: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter	Value
Memory capacity	2 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
↳ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

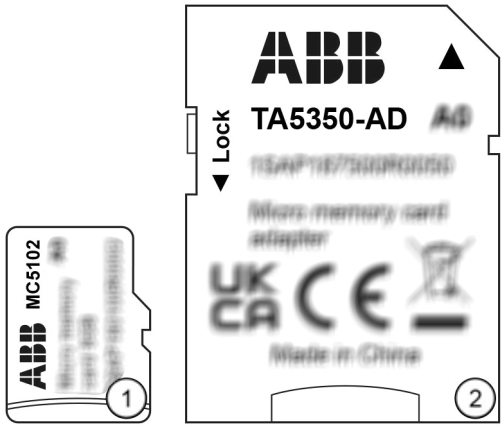
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

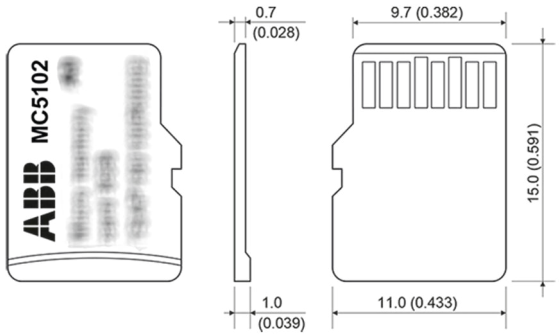
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

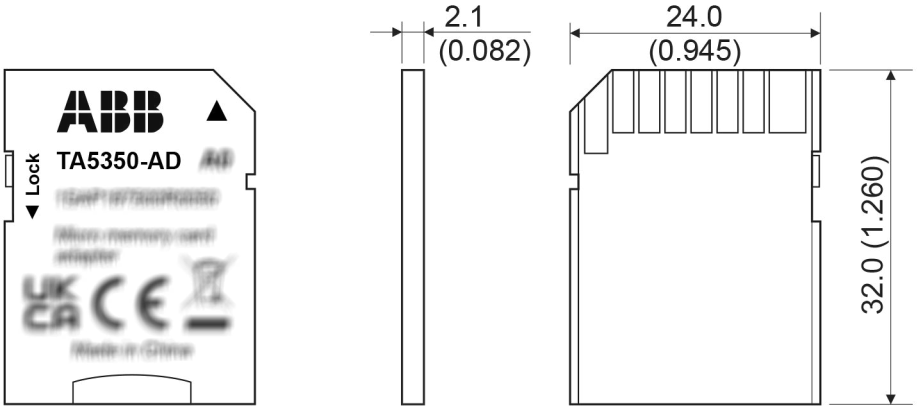
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V2 and AC500-eCo V2

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

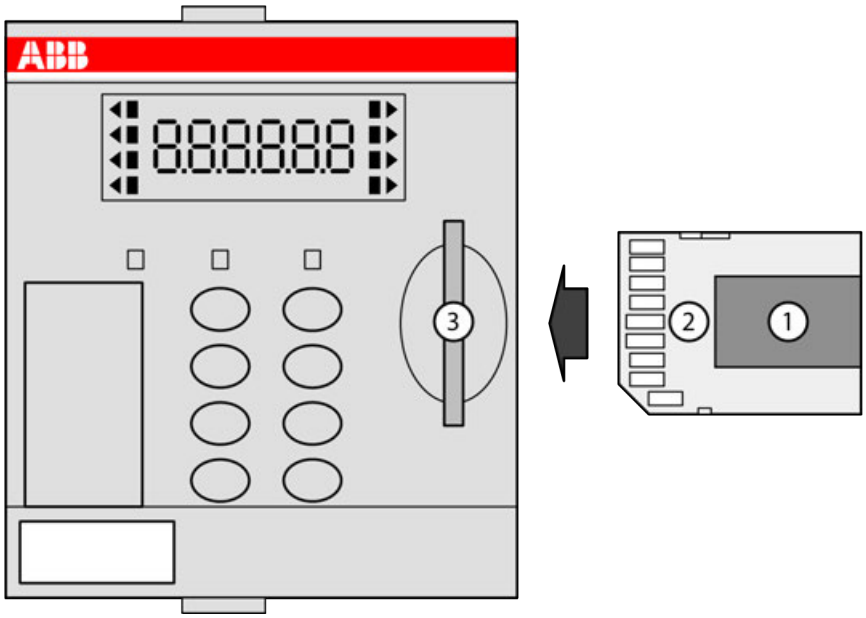


Fig. 1024: Insert micro memory card into PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

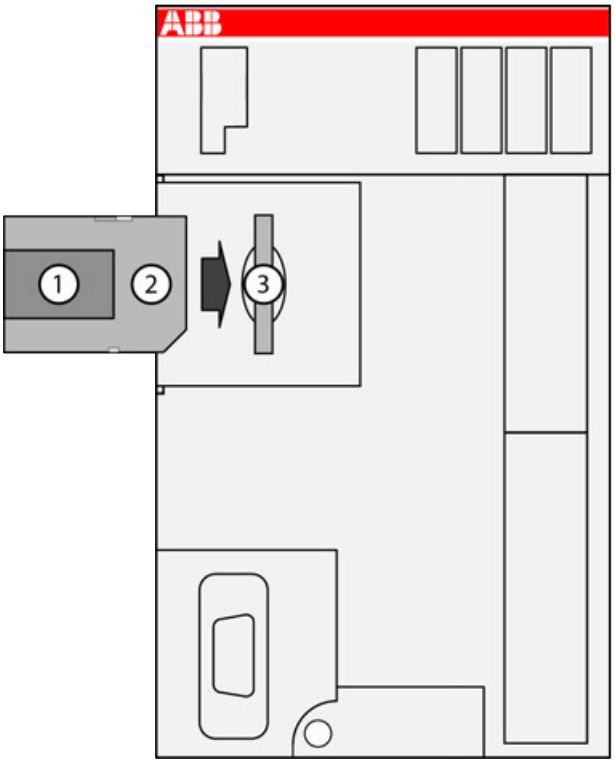


Fig. 1025: Insert micro memory card into PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Remove the micro memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

Remove the micro memory card with micro memory card adapter only when the RUN LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

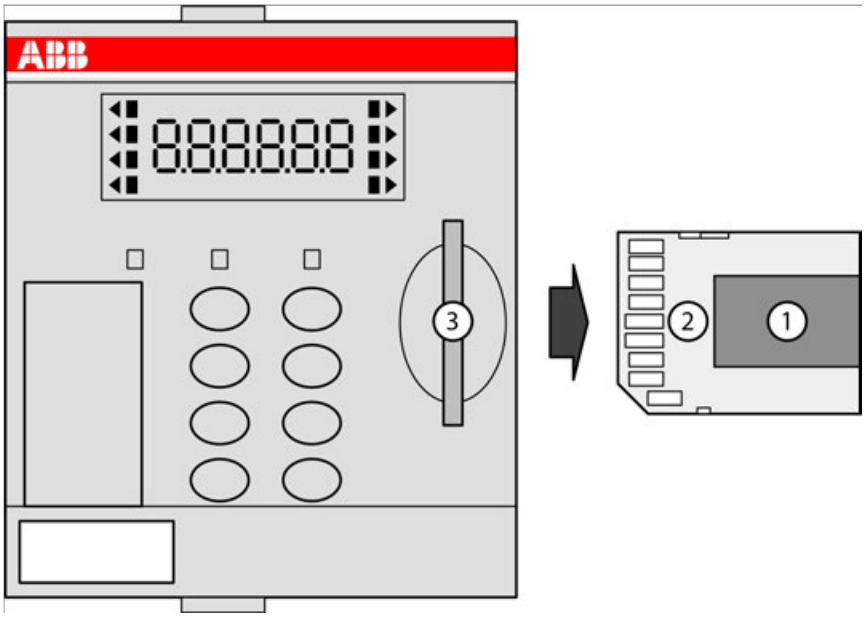


Fig. 1026: Remove micro memory card from PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

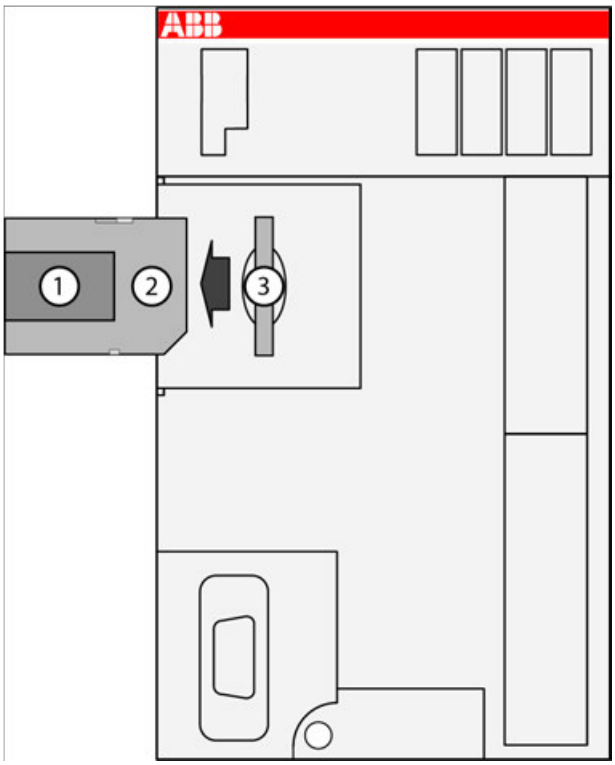


Fig. 1027: Remove micro memory card from PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		8 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter ↳ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3]	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1]	x [1] [2]	x [1]	x	x [2]	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

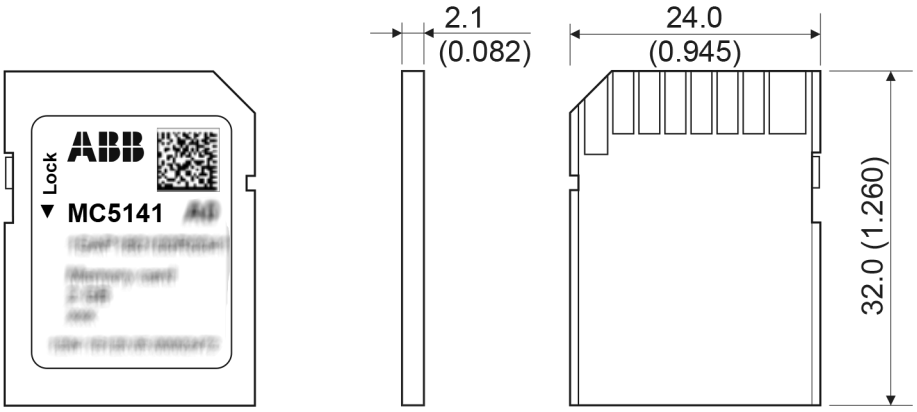
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.2.9.1.3 “MC503 - Memory card adapter” on page 5101*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

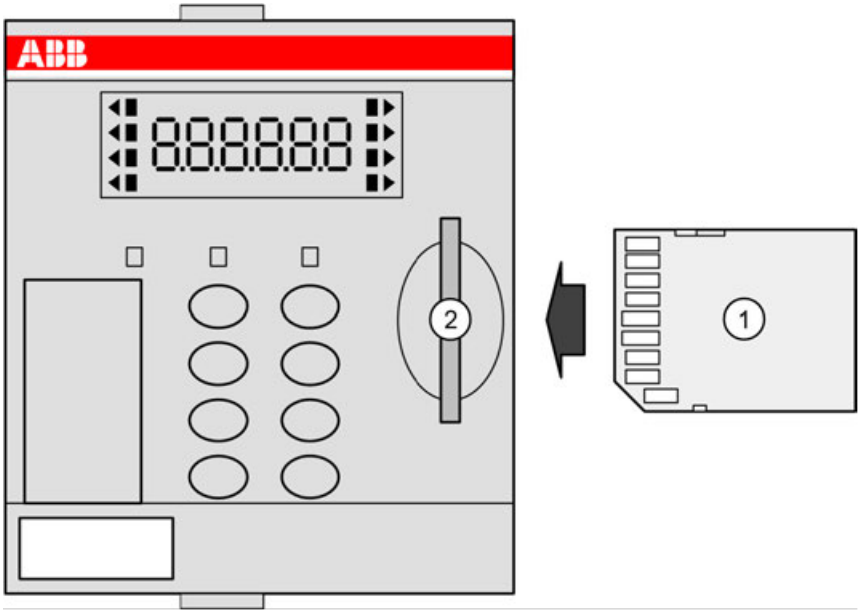


Fig. 1028: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

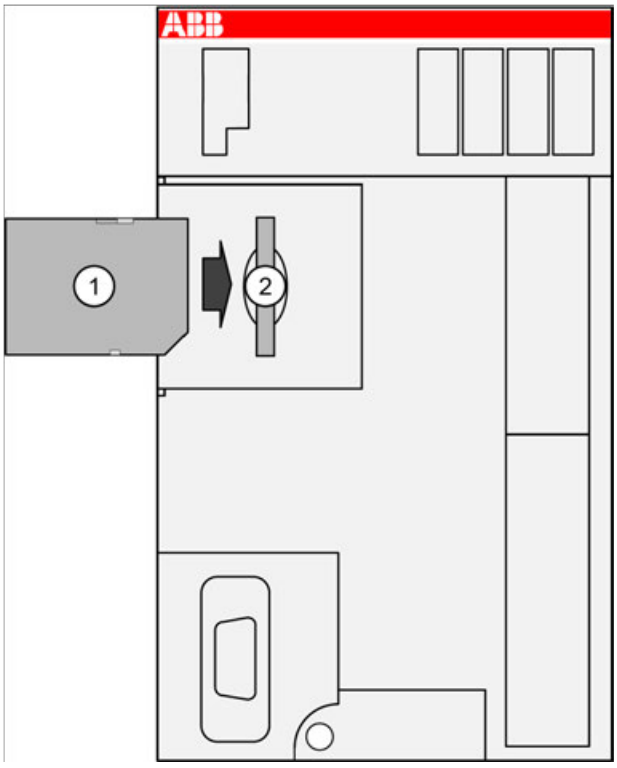


Fig. 1029: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

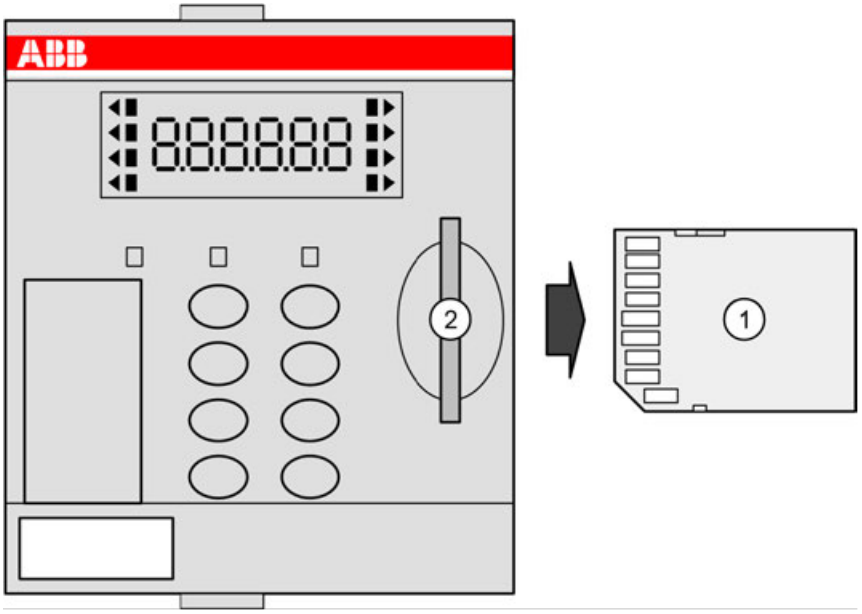


Fig. 1030: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

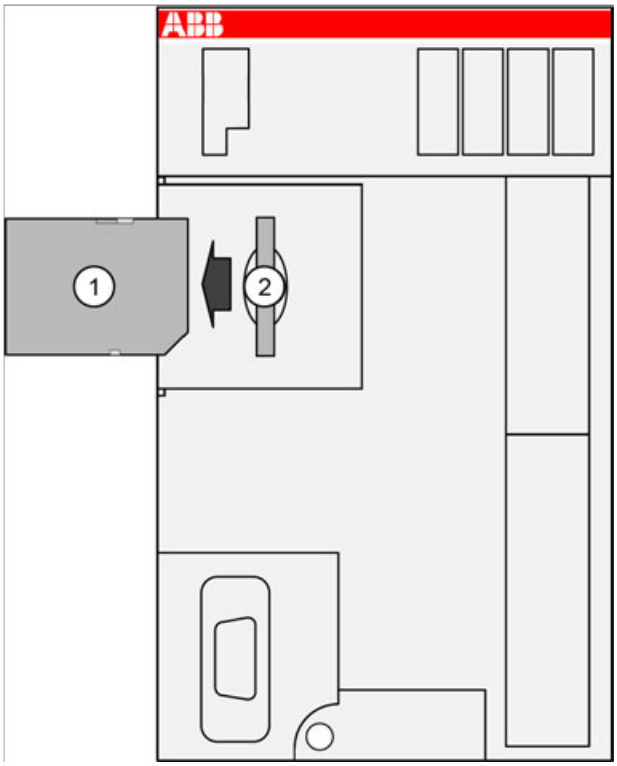


Fig. 1031: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
↬ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

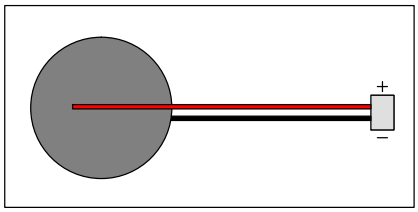
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA521 - Battery

- Manganese dioxide lithium battery, 3 V, 560 mAh
- Non-rechargeable



Purpose

The TA521 battery is the only applicable battery for the AC500 processor modules *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*. It cannot be recharged.

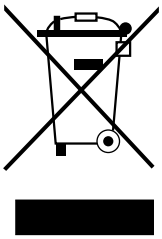
The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.4.1.4.2 “AC500 battery” on page 5419*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



Battery lifetime

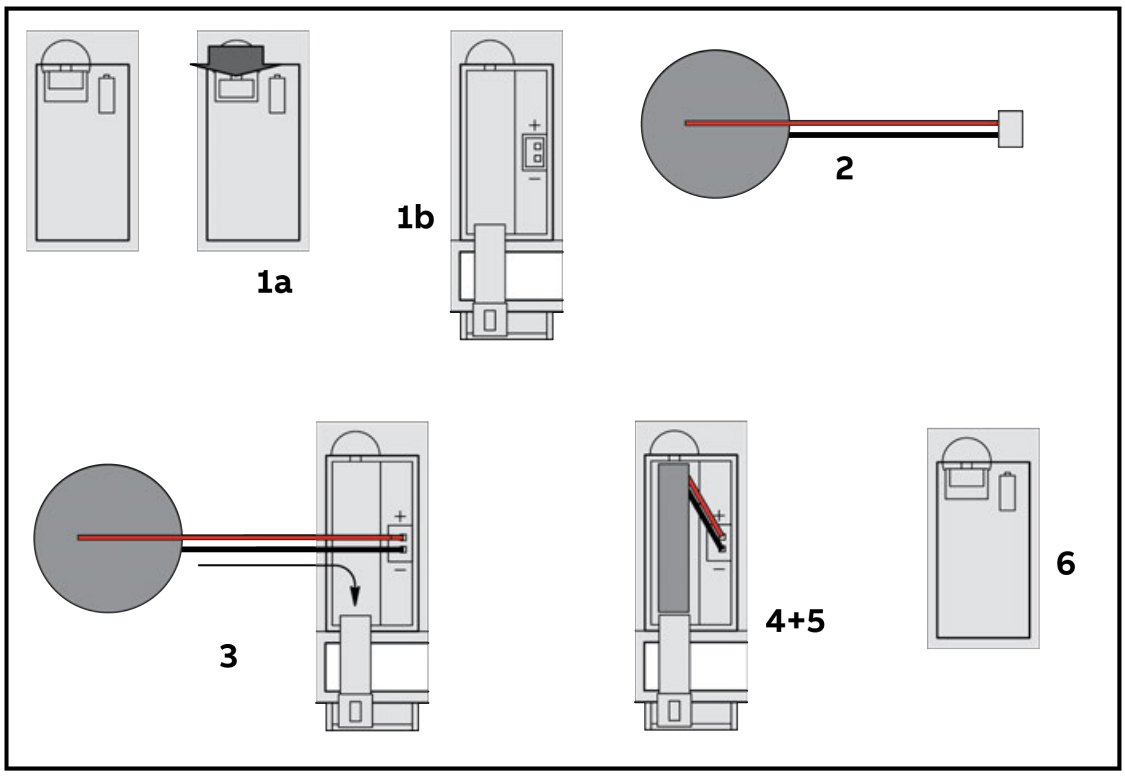
The battery lifetime is the time, the battery can store data while the processor module is not powered. As long as the processor module is powered, the battery will only be discharged by its own leakage current.

To avoid a short battery discharge, the battery should always be inserted or replaced while the process module is under power, then the battery is correctly recognized and will not shortly discharged.

Insertion

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

WARNING!
Risk of fire or explosion!
Use of incorrect Battery may cause fire or explosion.



1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front face of the processor module and cannot be removed.
2. Remove the TA521 battery from its package and hold it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.
3. Insert the battery connector into the small connector port of the compartment. The connector is keyed to find the correct polarity (red = positive pole = above).
4. Insert first the cable and then the battery into the compartment, push it until it reaches the bottom of the compartment.
5. Arrange the cable in order not to inhibit the door to close.
6. Pull-up the door and press until the locking mechanism snaps.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front view of the processor module and cannot be removed.
2. Remove the old TA521 battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

3. Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Technical data

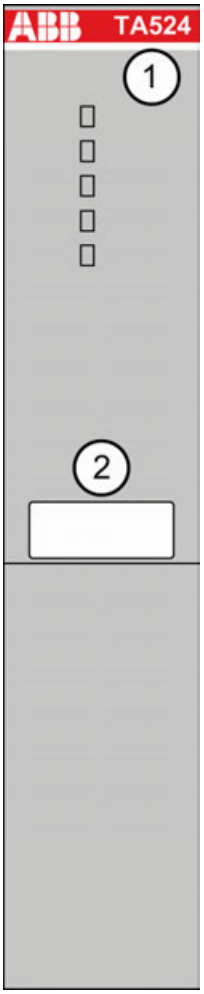
Parameter	Value
Nominal voltage	3 V
Nominal capacity	560 mAh
Temperature range (index below C0)	Operating: 0 °C...+60 °C Storage: -20 °C...+60 °C Transport: -20 °C...+60 °C
Temperature range (index C0 and above)	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	2 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug.
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug, black = negative pole,
Weight	7 g
Dimensions	Diameter of the button cell: 24.5 mm Thickness of the button cell: 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 300 R0001	TA521, lithium battery	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA524 - Dummy communication module



1 Type
2 Label

Purpose

TA524 is used to cover an unused communication module slot of a terminal base ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*. It protects the terminal base from dust and inadvertent touch.

Handling instructions

TA524 is mounted in the same way as a common communication module ↳ *Chapter 1.6.3.6.3.6 “Mounting/Demounting the communication modules” on page 5335*.

Technical data

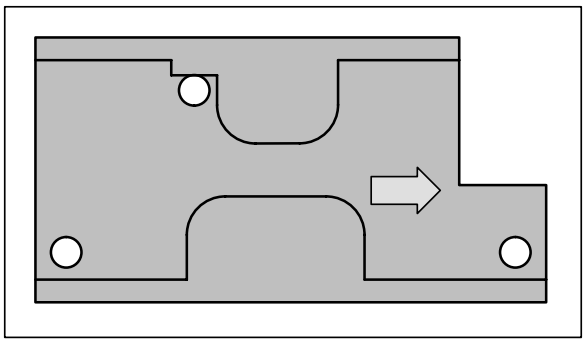
Parameter	Value
Weight	50 g
Dimensions	135 mm x 28 mm x 62 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 600 R0001	TA524, dummy communication module	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* ↬ *"Mounting with screws" on page 5328* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* ↬ *"Mounting with screws" on page 5326*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA541 - Battery

- Manganese dioxide lithium battery, 3 V
- Non-rechargeable

Purpose

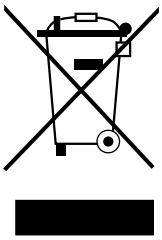
The TA541 lithium battery is the only applicable battery for PM595 *Chapter 1.6.2.3.2.2 “PM595-4ETH” on page 3863*. It is used to save RAM content of the processor module (PM595-4ETH-F only) and to back-up the real-time clock (all PM595 variants). It cannot be recharged.

The processor modules are supplied without a lithium battery. It therefore must be ordered separately. The lithium battery is used to save RAM contents of AC500 processor modules and back-up the real-time clock. Although the processor modules can work without a battery, its use is still recommended in order to avoid process data being lost.

The CPU monitors the discharge degree of the battery. A warning is output, before the battery condition becomes critical (about 2 weeks before). After the warning message has appeared, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



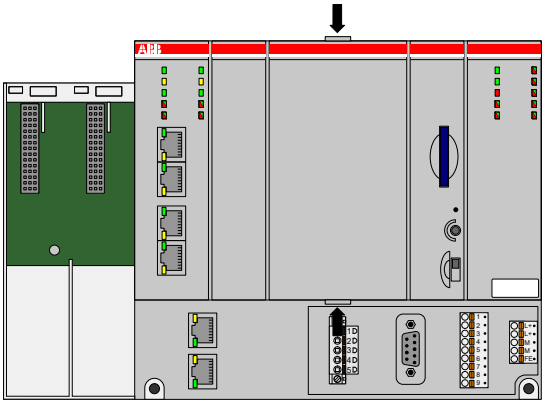
Battery lifetime

The battery lifetime is the time the battery can store data while the CPU is not powered. As long as the CPU is powered, the battery will only be discharged by its own leakage current.

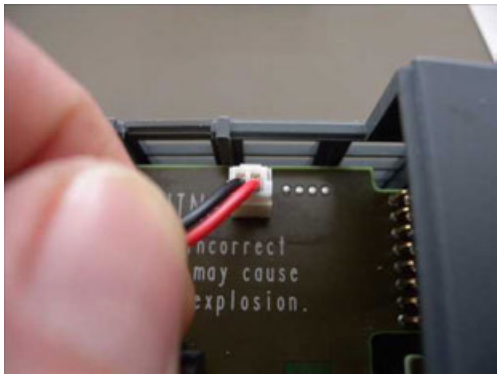
Insertion

The TA541 lithium battery is the only applicable battery for processor modules PM595.

1. Remove the front cover / display by pressing the marked areas with your fingers and pull it to the front.

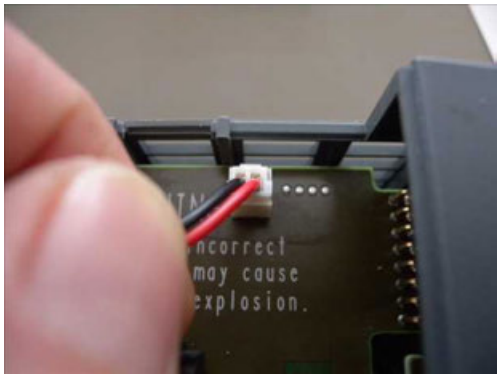


2. Remove the old battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket.



3. Remove the battery from its package and hold it by the small cable.

4. Insert the battery connector into the connector port of the PCB. The connector is keyed to find the correct polarity (red = positive pole = right side).



5. Insert the battery into the battery compartment on the left side as shown in the figure.

6. Re-assemble the front cover / display by pressing it straight from the front until it snaps in.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

For PM595-4ETH-F only: battery replacement should be done with the system under power. Without battery and power supply there is no data buffering possible.

For PM595-4ETH-M-XC only: battery only back-ups the real-time clock.

1. Remove the front cover / display by pressing the marked areas and pull it to the front.
2. Remove the old battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

   Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Technical data

Parameter	Value
Nominal voltage	3 V
Nominal capacity	1800 mAh

Parameter	Value
Temperature range	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	1 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug Black = negative pole
Weight	17 g
Dimensions	Diameter of the battery: ca. 18 mm Height of the battery: ca. 35 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 700 R0001	TA541, lithium battery	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

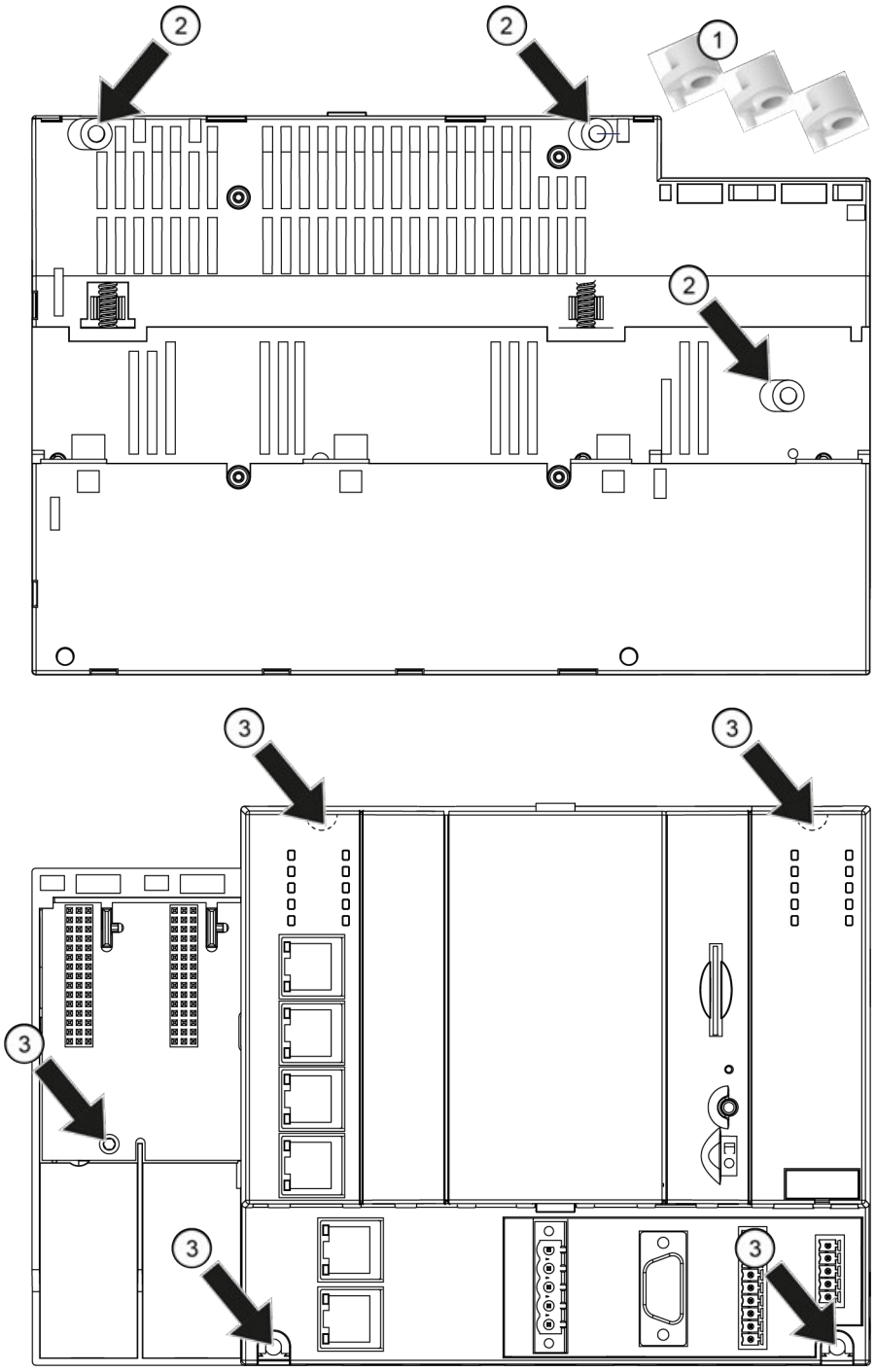
TA543 - Screw mounting accessory

Intended purpose

The TA543 screw mounting accessory is used for mounting the processor module PM595 without DIN rail.

Handling instruction

3 TA543 must be snapped on the backside of PM595 ↳ *Chapter 1.6.3.6.3.3 “Mounting/ Demounting the processor module PM595” on page 5330*.



1 3 parts of screw mounting accessory TA543
2 3 slots for screw mounting accessory TA543
3 5 holes for screw mounting

Technical data

Parameter	Value
Weight	5 g
Dimensions	12 mm x 8.5 mm x 10 mm

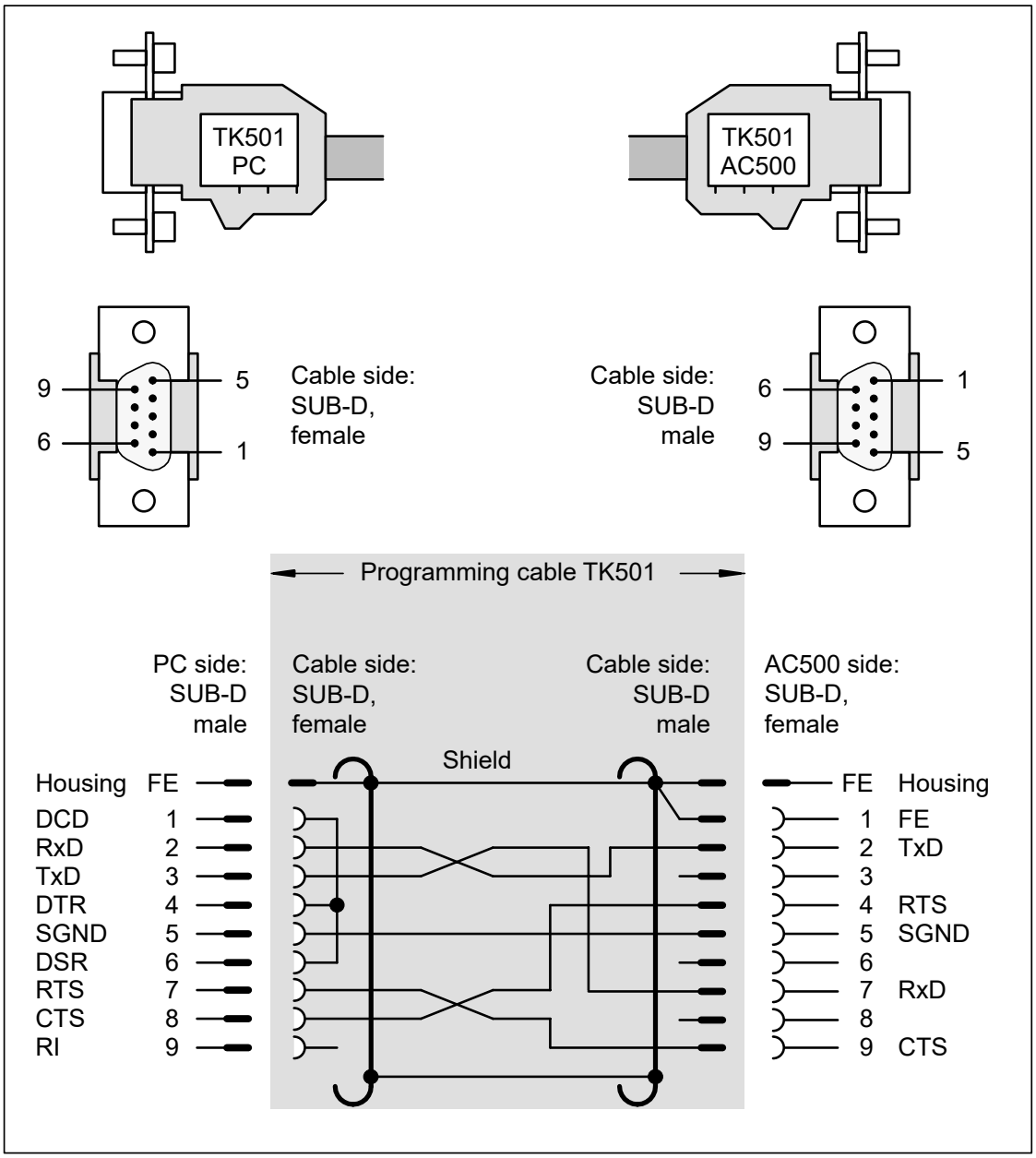
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 800 R0001	TA543, screw mounting accessory for PM595	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TK501 - Programming cable

- Cable on PC side: D-sub, 9-pole, female, RS-232, for COM interface
- Cable on AC500 side: D-sub, 9-pole, male, RS-232, for COM2 interface
- Cable length: 5 m


TK501
PC
TK501
AC500
9
6
5
1
Cable side:
SUB-D,
female
Cable side:
SUB-D
male
6
9
1
5
Programming cable TK501
PC side:
SUB-D
male
Cable side:
SUB-D,
female
Cable side:
SUB-D
male
AC500 side:
SUB-D,
female
Shield
Housing FE
DCD 1
RxD 2
TxD 3
DTR 4
SGND 5
DSR 6
RTS 7
CTS 8
RI 9
FE Housing
1 FE
2 TxD
3
4 RTS
5 SGND
6
7 RxD
8
9 CTS

CTS Clear To Send
DCD Data carrier detect
DTR Data terminal ready
DSR Data set ready
FE Functional earth
RI Ring indicator
RTS Request To Send
RxD Receive data
SGND Signal ground
TxD Transmit data

Purpose

The TK501 cable connects a 9-pole serial COM interface of a PC with the serial COM2 interface of PM57x, PM58x and PM59x. It is used for programming purposes.

With AC500/AC500-eCo processor modules, only the ABB programming cables TK50x can be used. Other cables may cause communication faults and must not be used.

Connections

The 2 plugs are put on the 2 COM interfaces and tightened there.

Technical data

Parameter	Value
Connector at the PC (COM interface)	D-sub, 9-pole, female
Connector at the processor module (COM2)	D-sub, 9-pole, male
Cable length	5 m
Cable type	LiYCY 5 x 0.14 mm^2, shielded
Weight	220 g

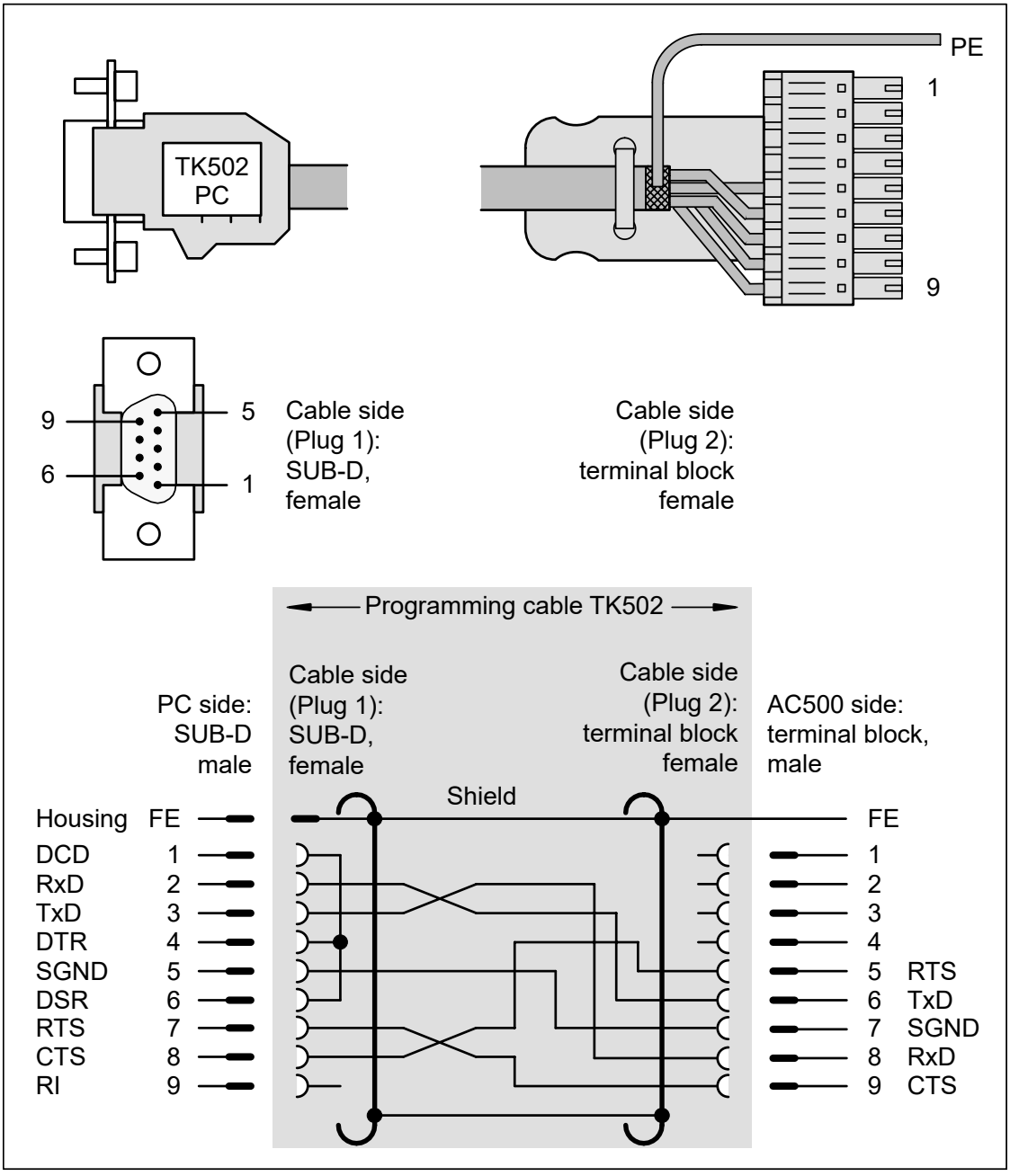
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 200 R0001	TK501, programming cable D-sub / D-sub, length: 5 m	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TK502 - Programming cable

- Cable on PC side: D-sub, 9-pole, female, RS-232, for COM interface
- Cable on AC500 side: terminal block, 9-pole, female, RS-232, for COM1 interface
- Cable length: 5 m


TK502
PC
PE
1
9
9
5
6
1
Cable side
(Plug 1):
SUB-D,
female
Cable side
(Plug 2):
terminal block
female
Programming cable TK502
Cable side
(Plug 1):
SUB-D,
female
Cable side
(Plug 2):
terminal block
female
PC side:
SUB-D
male
AC500 side:
terminal block,
male
Shield
Housing FE
DCD 1
RxD 2
TxD 3
DTR 4
SGND 5
DSR 6
RTS 7
CTS 8
RI 9
FE
1
2
3
4
5 RTS
6 TxD
7 SGND
8 RxD
9 CTS

CTS Clear To Send
DCD Data carrier detect
DTR Data terminal ready
DSR Data set ready
FE Functional earth
RI Ring indicator
RTS Request To Send
RxD Receive data
SGND Signal ground
TxD Transmit data

Purpose

The TK502 cable connects a 9-pole serial COM interface of a PC with the serial COM1 interface of PM57x, PM58x and PM59x. It is used for programming purposes.

With AC500/AC500-eCo processor modules, only the ABB programming cables TK50x can be used. Other cables may cause communication faults and must not be used.

Connections

The 2 plugs are put on the two COM interfaces and the plug at the PC side is tightened then.

Technical data

Parameter	Value
Connector at the PC (COM interface)	D-sub, 9-pole, female
Connector at the AC500 CPU (COM1)	terminal block, 9-pole, female
Cable length	5 m
Cable type	LiYCY 5 x 0.14 mm^2, shielded
Weight	220 g

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 200 R0101	TK502, programming cable terminal block / D-sub, length: 5 m	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TK503 - COM1 USB programming cable

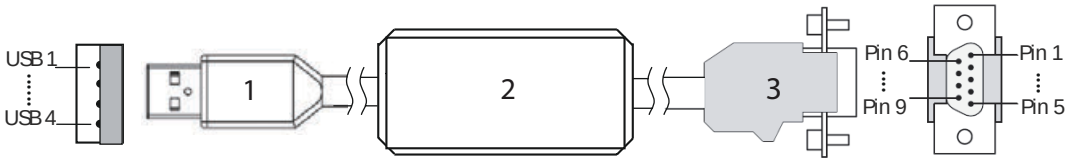
- PC-side: USB connector type A
- AC500-side: D-sub, 9-pin, male
- Length 3 m

Intended purpose

TK503 programming cable connects the USB interface of a PC with the serial interface of a processor module. It is used for programming purposes. TK503 can be used with all AC500 processor modules.

> *With AC500/AC500-eCo processor modules, only the ABB programming cables TK50x can be used. Other cables may cause communication faults and must not be used.*

Connections



1 USB connector type A (PC side)
2 USB/RS-485 converter
3 D-sub, 9-pin, (RS-485, PLC side)

Table 588: TK503 programming cable wiring USB pin

USB pin	Signal	Description
USB 1	VBUS	USB power
USB 2	-D	USB data negative
USB 3	+D	USB data positive
USB 4	GND	Ground

Table 589: TK503 programming cable wiring D-sub, 9-pin

Pin	Signal	Description
Pin 1	FE	Functional earth
Pin 2	-	Not connected
Pin 3	RXD/TXD-P	Receive/Transmit positive
Pin 4	-	Not connected
Pin 5	SGND	Signal ground
Pin 6	3.3 V/5 V	Power supply

Pin	Signal	Description
Pin 7	-	Not connected
Pin 8	RXD/TXD-N	Receive/Transmit negative
Pin 9	-	Not connected

1. Install the device driver for the programming cable (see *“Installation of cable driver” on page 5192*).

> *Once you have installed the device driver of the cable in your Windows system, make sure that you always use the same USB port on your computer.*
>
> *Otherwise, Windows will ask you to install the driver a second time if you connect the cable to a different USB port of your computer.*

2. Plug the 9-pin D-sub male connector to the connector at the processor module and tighten it there.
3. Plug the USB connector to an USB interface at your PC.

Technical data

Parameter	Value
Connector at the PC (USB interface)	USB connector type A
Connector at the processor module	D-sub, 9-pin, male
Length	3 m
Cable type	Programming cable
Weight	0.4 kg

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R1100	TK503, COM1 USB programming cable -> D-sub (RS-485), length 3 m	Active

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

Installation of cable driver

OPERATION INSTRUCTION

PROGRAMMING CABLE TK503 / TK504

USB DRIVER INSTALLATION

Contents

1 Introduction and Basics

1.1 Intended Use

The TK503/TK504 programming cable can be used to operate and to configure the PLC via a PC or laptop. For this, CODESYS software, driver and utility programs must be installed and a TK503 or TK504 programming cable must be connected.

NOTICE!

The TK503/TK504 programming cable cannot be used for AC500 V3 Processor Modules.

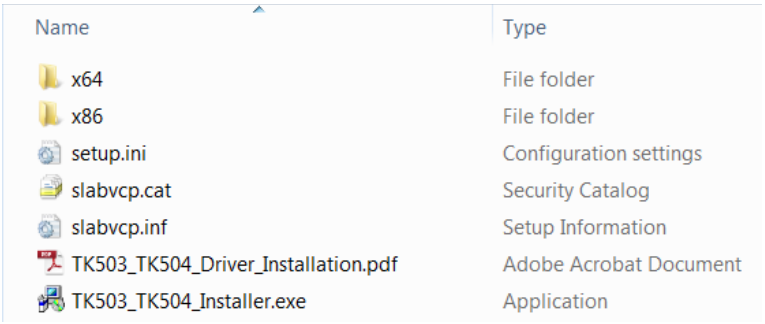
1.2 PC System Requirements

- Platform: Microsoft Windows Vista, Windows 7, Windows 10
- CD-ROM drive
- USB port available for connecting the TK503/TK504 programming cable

NOTICE!

Microsoft, Windows and the Windows logo are trademarks of Microsoft Corporation in the USA and/or other countries. All other product and company names are trademarks of their respective owners.

1.3 Content of the Installation Package



2 Installation

2.1 Installation Steps

Before you can use the TK503/TK504 programming cable, the appropriate USB driver must be installed on your PC or laptop.

The driver for the TK503/TK504 programming cable is installed in two steps:

– Pre-installation of the driver on your PC using the program *TK503_TK504_Installer.exe*.

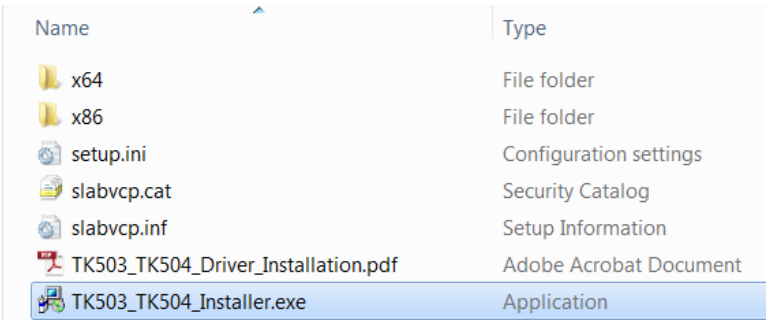
– Installation of the new hardware in Windows after the TK503 programming cable or TK504 programming cable is plugged in for the first time.

NOTICE!

Before you connect the TK503/TK504 programming cable with the PC, install the USB driver first.

2.2 Pre-Installation Routine

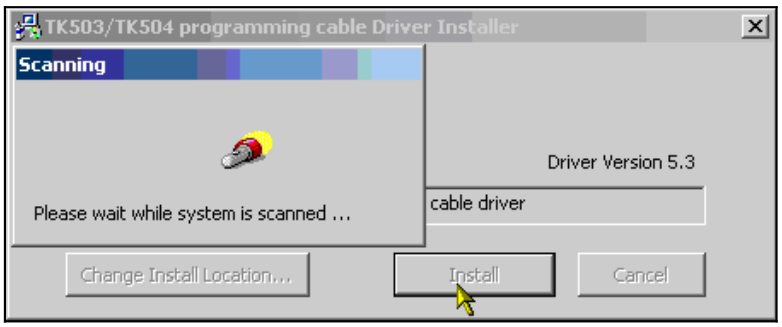
1. Uninstall all existing versions of the driver software.
2. Start the pre-installation of the driver by calling *TK503_TK504_Installer.exe.*



NOTICE!

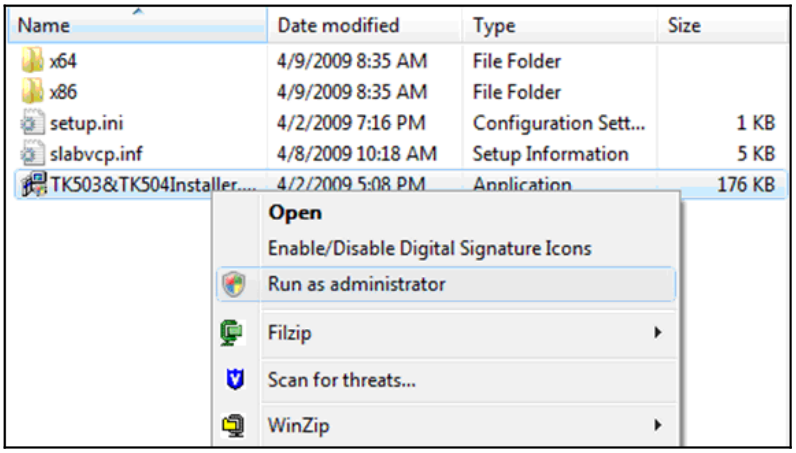
You must have administrator rights to run the installation.

3. Define the installation directory and click **Install**.



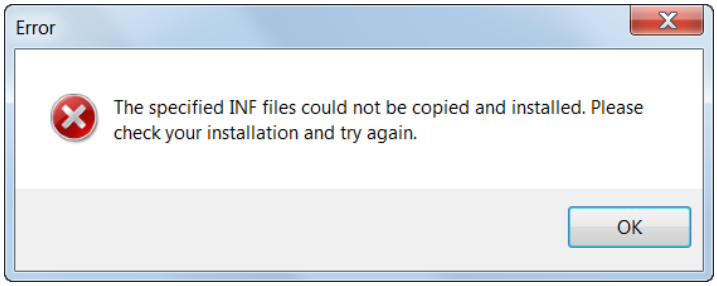
Windows Vista users only:

Start the *TK503&TK504Installer.exe* with the **Run as administrator** option, even if you have administrator rights. Acknowledge the following dialog with **Allow**.

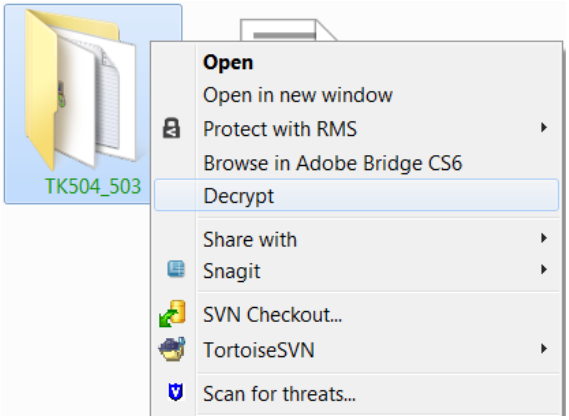


Windows 7 users only:

Windows will display an error message after clicking **Install**.



On this condition, decrypt the installation folder:



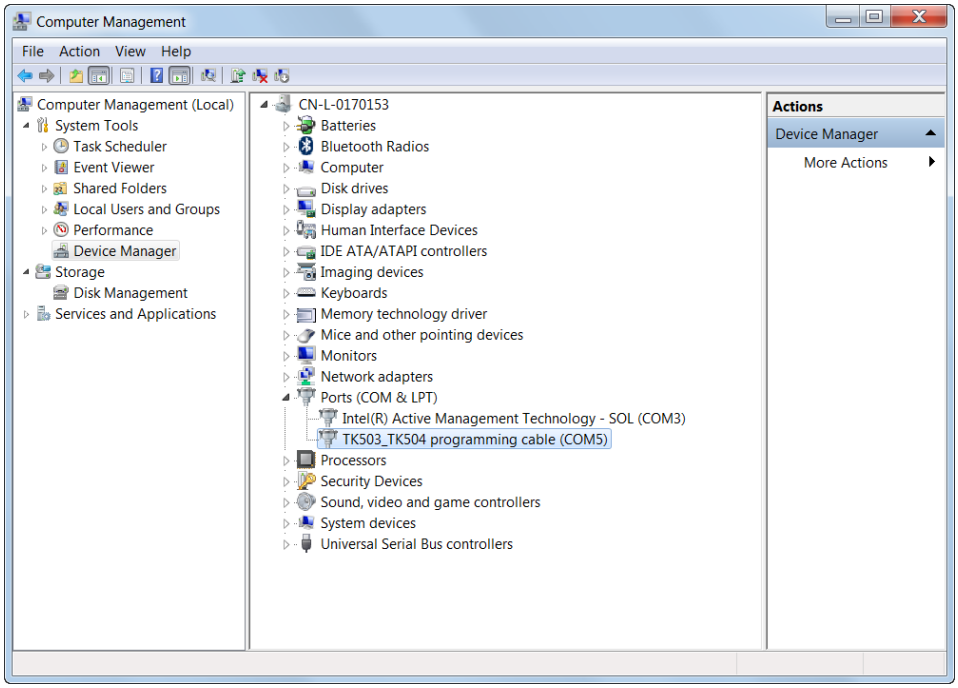
Then, start *TK503_TK504_Installer.exe* with the **Run as administrator** option again.

3 Communication

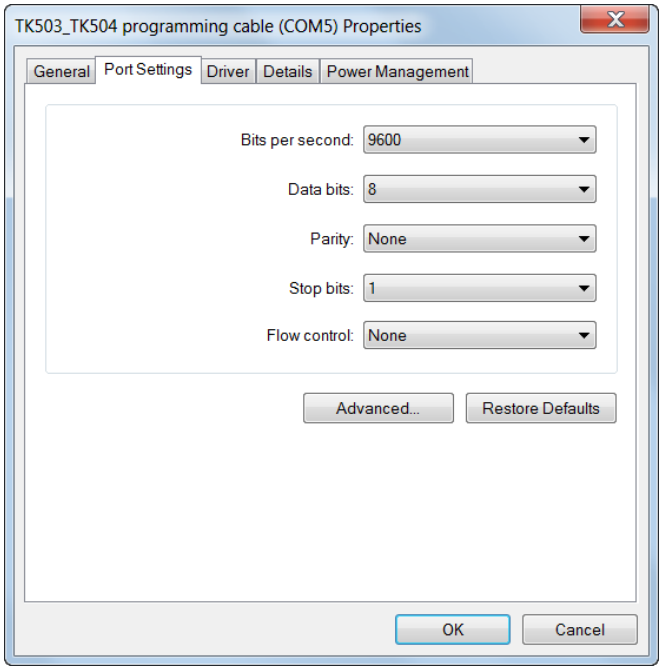
3.1 Virtual Communication Port Configuration

If the TK503/TK504 programming cable is plugged in a USB interface, Windows creates a virtual communication port (COM port).

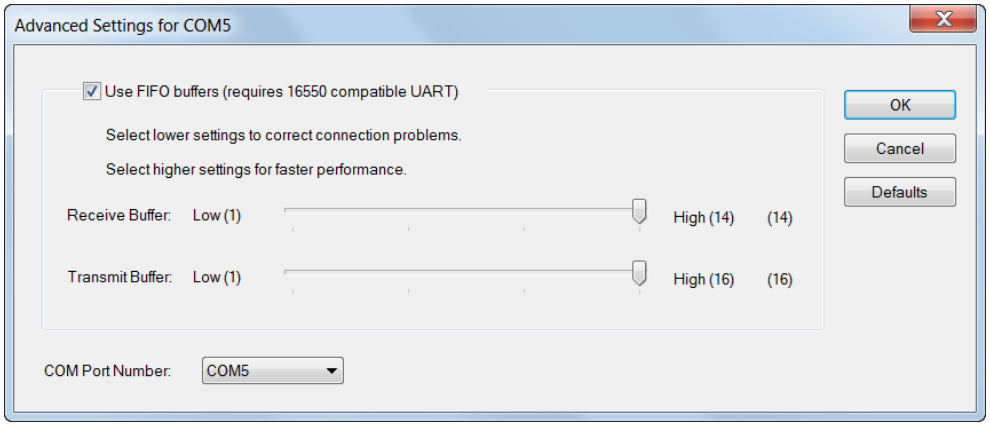
All communication ports can be viewed in the Windows Control Panel under Device Manager.



4. In the Ports settings click **Properties** to set the baud rate.



5. Set the COM port number under **Advanced** (up to COM32).

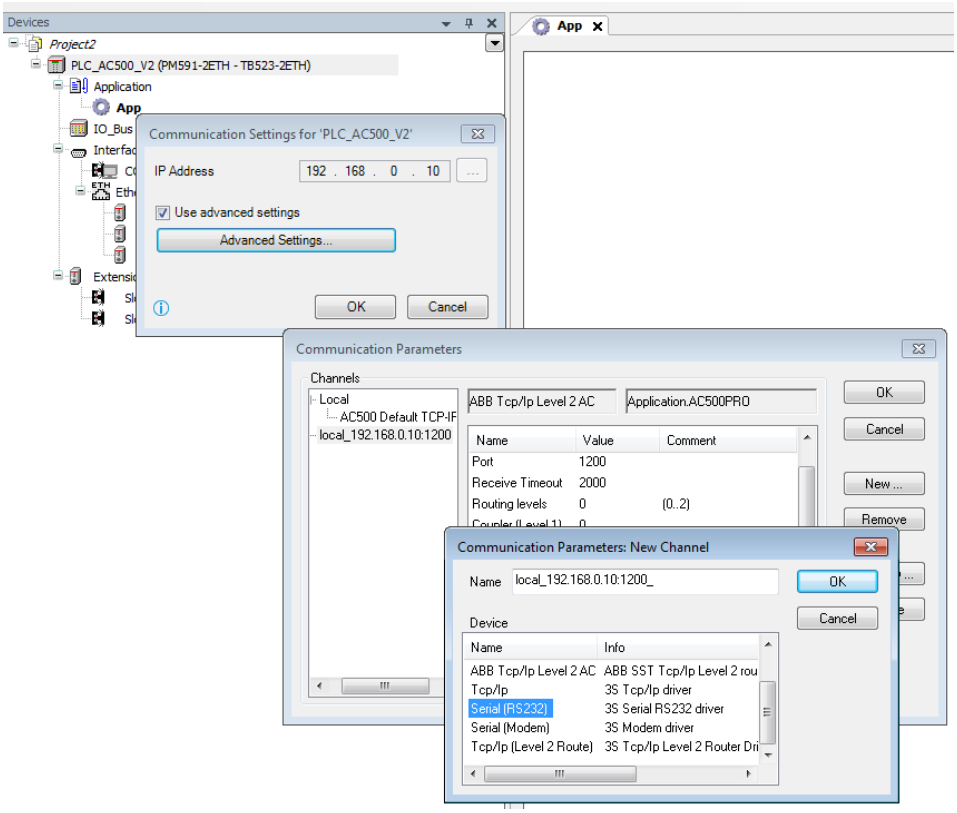


NOTICE!

When configuring the communication connection in CODESYS, the baud rate can also be set separately for each COM connection.

4 Automation Builder Communication

1. Install TK503/TK504 programming cable driver.
2. Connect the TK503 or TK504 programming cable to aPC or laptop. Windows detects the new hardware – complete the installation.
3. Start Automation Builder and open the project.
4. Right-click the PLC root node and select **Communication Parameters.**
5. Select the new virtual COM port.

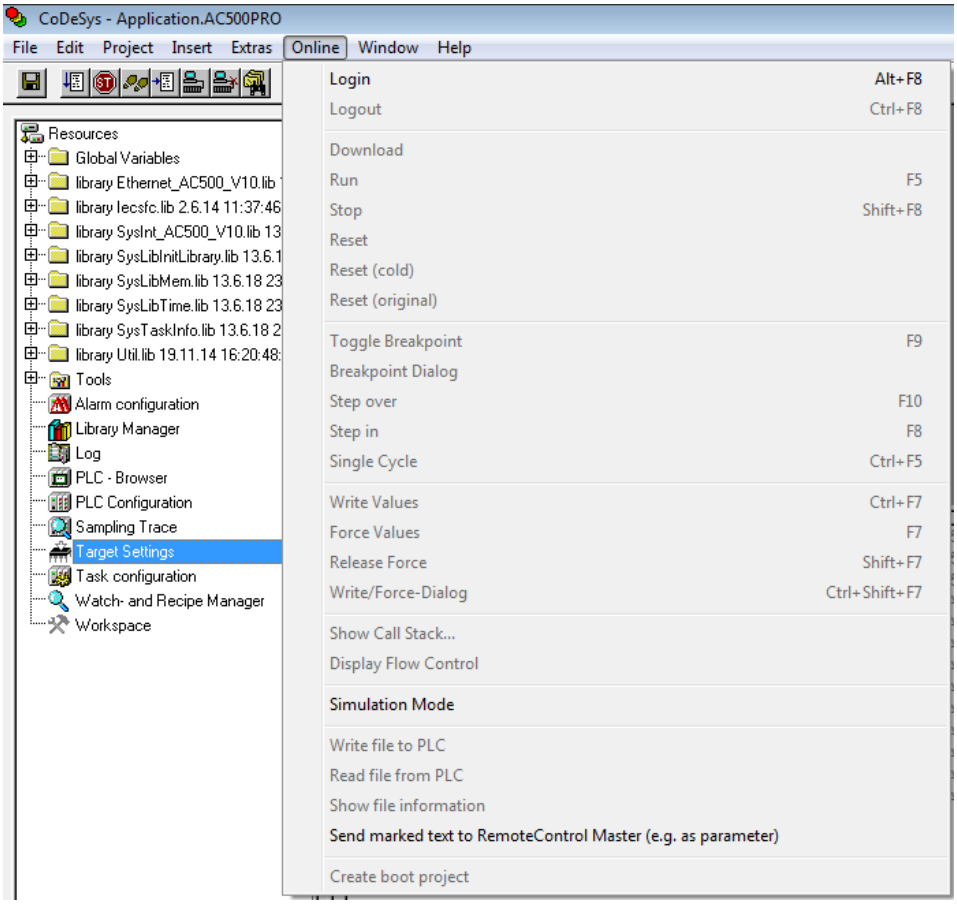


NOTICE!

The port number must be the same as the port number in the Device manager – Port – TK503/TK504 programming cable (COMx). Otherwise the communication cannot be built up.

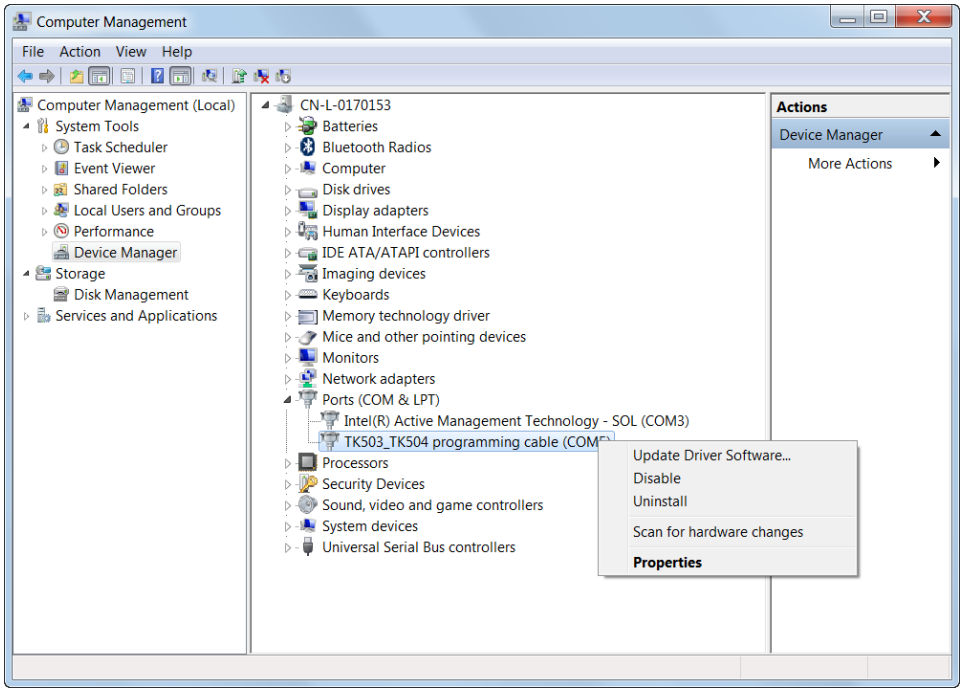
The number of COM ports depends on the availability on your computer.
The baud rate can be selected between 19200 and 115200 bps.

6. In CODESYS, create the communication between Automation Builder and the PLC.


CoDeSys - Application.AC500PRO
File Edit Project Insert Extras Online Window Help
Resources
Global Variables
library Ethernet_AC500_V10.lib
library Iecsfc.lib 2.6.14 11:37:46
library SysInt_AC500_V10.lib 13
library SysLibInitLibrary.lib 13.6.1
library SysLibMem.lib 13.6.18 23
library SysLibTime.lib 13.6.18 23
library SysTaskInfo.lib 13.6.18 2
library Util.lib 19.11.14 16:20:48
Tools
Alarm configuration
Library Manager
Log
PLC - Browser
PLC Configuration
Sampling Trace
Target Settings
Task configuration
Watch- and Recipe Manager
Workspace
Login Alt+F8
Logout Ctrl+F8
Download
Run F5
Stop Shift+F8
Reset
Reset (cold)
Reset (original)
Toggle Breakpoint F9
Breakpoint Dialog
Step over F10
Step in F8
Single Cycle Ctrl+F5
Write Values Ctrl+F7
Force Values F7
Release Force Shift+F7
Write/Force-Dialog Ctrl+Shift+F7
Show Call Stack...
Display Flow Control
Simulation Mode
Write file to PLC
Read file from PLC
Show file information
Send marked text to RemoteControl Master (e.g. as parameter)
Create boot project

5 Uninstallation / Update

1. In the Windows Control Panel open the Device Manger.
2. Right-click on the entry**TK503/TK504 programming cable** and select **Uninstall** or **Update Driver**.



Document Number: 3ADR010872, 1, en US

ABB AG
Eppelheimer Straße 82
69123 Heidelberg, Germany
Phone: +49 62 21 701 1444
Fax : +49 62 21 701 1382
E-Mail: plc.support@de.abb.com
www.abb.com/plc

1.6.2.9.3 S500-eCo

TA563-TA565 - Terminal blocks

These terminal blocks must only be used with AC500-eCo I/O modules and AC500-eCo processor modules.

Intended purpose

The TA563-TA565 terminal blocks are used to connect process signals and process voltages to AC500-eCo I/O modules and AC500-eCo processor modules (with -P extension inside their type designator only). 3 different kind of terminal blocks are available:

- Screw terminals with cable insertion on the side
- Screw terminals with cable insertion on the front
- Spring terminals with cable insertion on the front

Of each kind, 2 sizes are available:

- Terminals with 9 pins
- Terminals with 11 pins.

There are 2 compatible variants of each kind and size.

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

Technical data

Table 590: Screw-type terminals (TA563/TA564)

Parameter		Value
Type		Front terminal or side terminal (depending on model)
Conductor cross section		
	Solid	0.5 mm^2 to 2.5 mm^2
	Flexible	0.5 mm^2 to 2.5 mm^2
Stripped conductor end		
	TA563	8 mm
	TA564	10 mm
Width of the screwdriver		3.5 mm
Fastening torque		0.4 Nm - 0.5 Nm
Degree of protection		IP 20 (if all terminal screws are tightened)
Conductor cross section flexible, with ferrule with/without plastic sleeve		Min. 0.25 mm^2 Max. 1.5 mm^2

Table 591: Spring terminals (TA565)

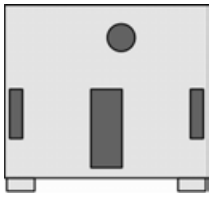
Parameter		Value
Type		Front terminal
Conductor cross section		
	Solid	0.5 mm^2 to 2.5 mm^2
	Flexible	0.5 mm^2 to 2.5 mm^2
Stripped conductor end		10 mm
Degree of protection		IP 20
Conductor cross section flexible, with ferrule with/without plastic sleeve		Min. 0.25 mm^2 Max. 1.5 mm^2

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3101	Terminal Block TA563-9, 9-pole, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal Block TA563-11, 11-pole, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3103	Terminal Block TA564-9, 9-pole, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal Block TA564-11, 11-pole, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal Block TA565-9, 9-pole, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal Block TA565-11, 11-pole, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA566 - Wall mounting accessory



Intended purpose

The TA566 wall mounting accessory is used for mounting S500-eCo I/O modules and AC500-eCo processor modules without DIN rail.

Handling instruction

The TA566 is snapped into the back side of the device's housing ↳ *Chapter 1.6.3.5.3.2 “Mounting and demounting of S500-eCo I/O modules” on page 5245*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	29 mm x 28 mm x 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3107	TA566, wall mounting accessory, 100 pieces	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

1.6.2.9.4 S500

CP-E - Economic range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

- Wide-range input voltage
- Mounting on DIN rail
- High efficiency of up to 90 %
- Low power dissipation and low heating
- Wide ambient temperature range from -40 °C...+70 °C
- No-load-proof, overload-proof, continuous short-circuit-proof
- Power factor correction (depending on the type)
- Approved in accordance with all relevant international standards

Table 592: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR427030R0000	CP-E 24/0.75	100-240 V AC or 120-370 V DC	24 V DC, 0.75 A	-	22.5
1SVR427031R0000	CP-E 24/1.25	100-240 V AC or 90-375 V DC	24 V DC, 1.25 A	-	40.5
1SVR427032R0000	CP-E 24/2.5	100-240 V AC or 90-375 V DC	24 V DC, 2.5 A	-	40.5
1SVR427034R0000	CP-E 24/5.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 5 A	-	63.2
1SVR427035R0000	CP-E 24/10.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 10 A	-	83
1SVR427036R0000	CP-E 24/20.0	115-230 V AC or 120-370 V DC	24 V DC, 20 A	-	175

CP-C.1 - High performance range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

The CP-C.1 power supplies are ABB's high performance and most advanced range. With excellent efficiency, high reliability and innovative functionality it is prepared for the most demanding industrial applications. These power supplies have a 50 % integrated power reserve and operate at an efficiency of up to 94 %. They are equipped with overheat protection and active power factor correction. Combinded with a broad AC and DC input range and extensive worldwide approvals the CP-C.1 power supplies are the preferred choice for professional DC applications.

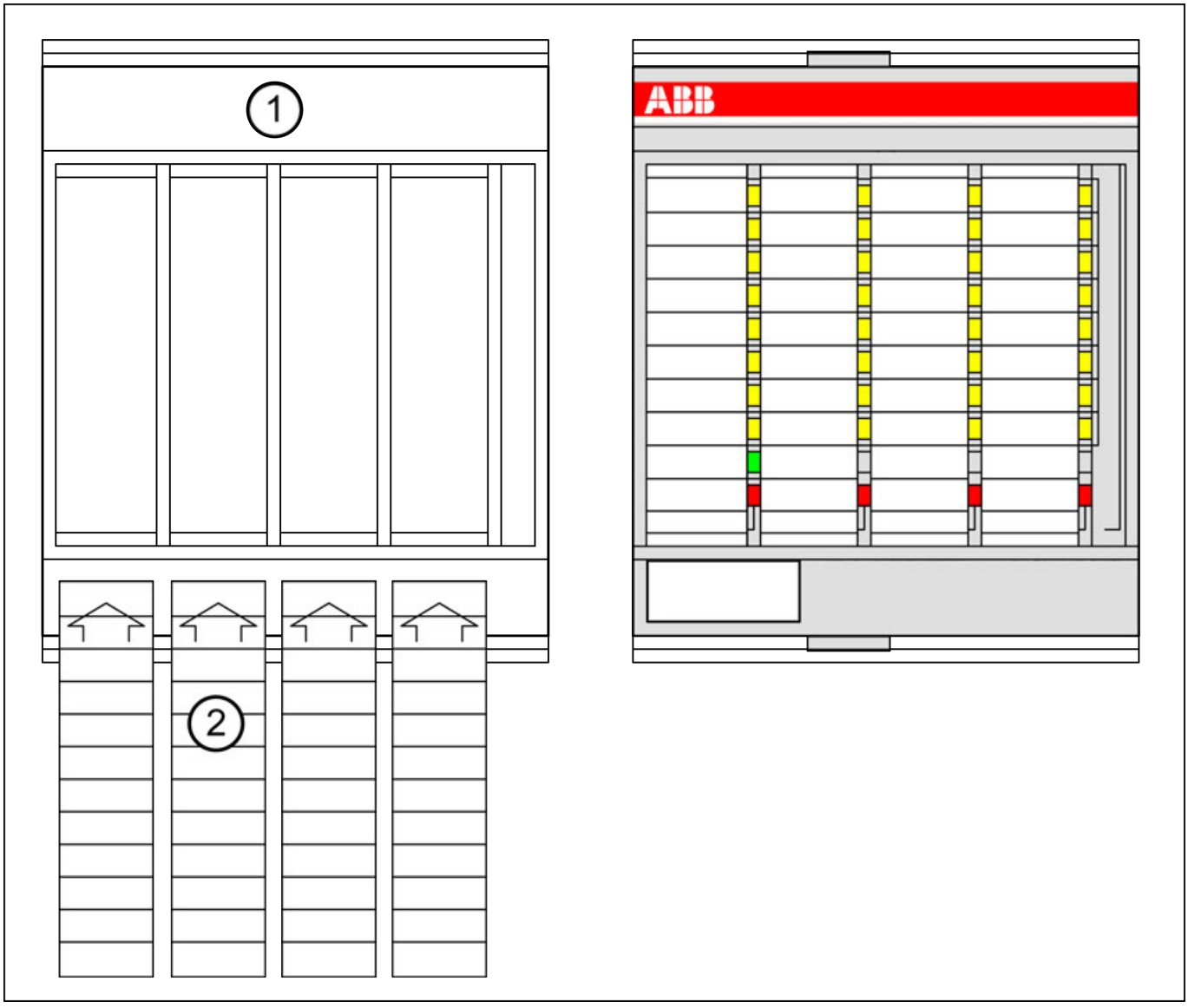
- Typical efficiency of up to 94 %
- Power reserve design delivers up to 150 % of the nominal output current
- Signaling outputs for DC OK and power reserve mode
- High power density leads to very compact and small devices
- No-load-proof, overload-proof, continuous short-circuit-proof
- Active power factor correction (PFC)

Table 593: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR360563R1001	CP-C.1 24/5.0	110-240 V AC or 90-300 V DC	24 V DC, 5 A	+50 %	40
1SVR360663R1001	CP-C.1 24/10.0	110-240 V AC or 90-300 V DC	24 V DC, 10 A	+50 %	60
1SVR360763R1001	CP-C.1 24/20.0	110-240 V AC or 90-300 V DC	24 V DC, 20 A	+30 %	82

TA523 - Pluggable label mounting

For labelling the channels of S500 I/O modules.



1 Pluggable label mounting TA523
2 Plastic labels to be inserted into the holder

Purpose

The pluggable label mounting is used to hold 4 plastic labels, on which the meaning of the I/O channels of I/O modules can be written down. The holder is transparent so that after snapping it onto the module the LEDs shine through.

Handling instructions

The plastic labels can be printed out from TA563.doc *http://new.abb.com/products/ABB1SAP180500R0001*.

Technical data

Parameter	Value
Use	For labelling channels of I/O modules
Mounting	Snap-on to the module
Weight	20 g
Dimensions	82 mm x 67 mm x 13 mm

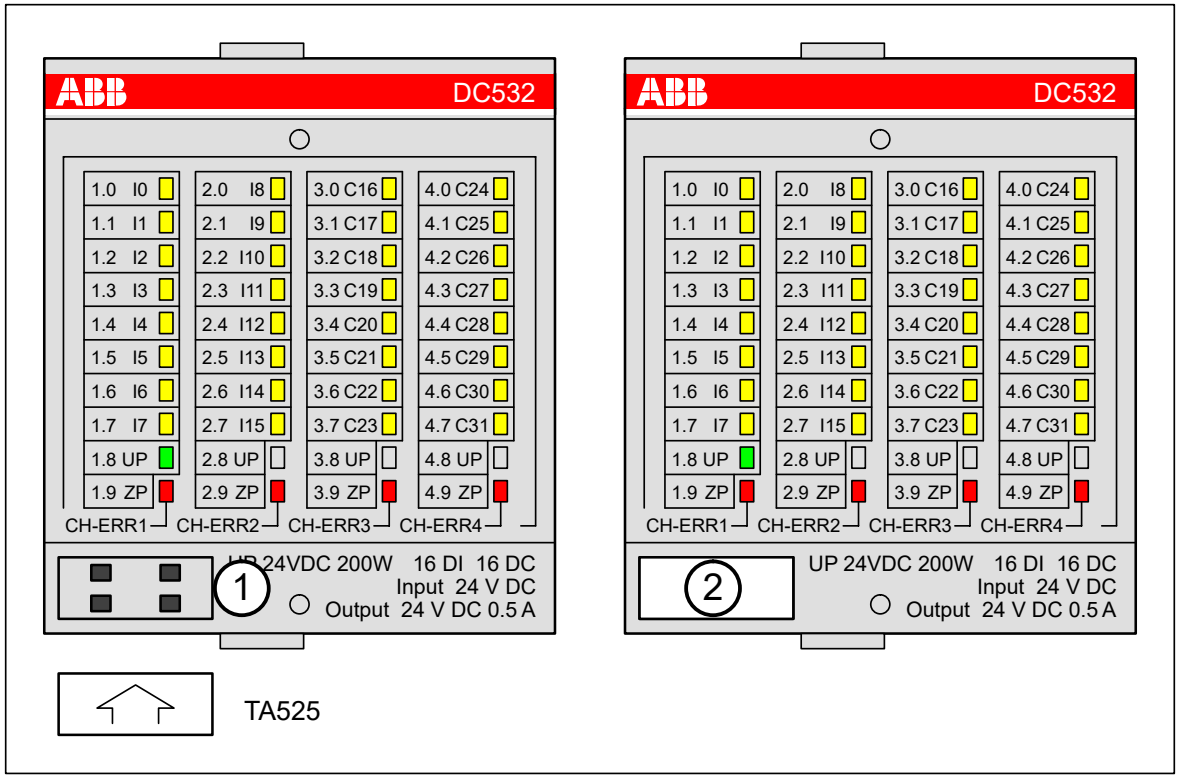
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 500 R0001	TA523, pluggable label mounting (10 pieces)	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA525 - Plastic labels

Accessory to label AC500 and S500 modules.



1 Module without plastic label TA525
2 Module with plastic label TA525

Purpose

The plastic labels are suitable for labelling AC500 and S500 modules (CPUs, communication modules and I/O modules). The small plastic parts can be written on with a standard waterproof pen.

Handling instructions

The plastic labels are inserted under a slight pressure. For disassembly, a small screwdriver is inserted at the lower edge of the module.

Technical data

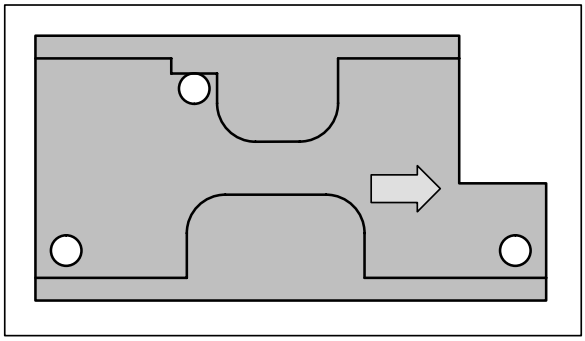
Parameter	Value
Use	For labelling AC500 and S500 modules
Mounting	Insertion under a slight pressure
Disassembly	With a small screwdriver
Scope of delivery	10 pieces
Weight	1 g per piece
Dimensions	8 mm x 20 mm x 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 700 R0001	TA525, Set of 10 white plastic labels	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* ↳ *"Mounting with screws" on page 5328* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* ↳ *"Mounting with screws" on page 5326*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA535 - Protective caps for XC devices

Purpose

Accessory to cover unused connectors of XC devices in salt mist environments.

One TA535 package includes different cap types for the following connectors:

- RJ45 connectors
- 9-pole D-sub connector
- FieldBusPlug connector

Protection should be done for all unused slots of -XC devices.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 300 R0001	TA535, Protective Caps for XC devices	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

1.6.3 System assembly, construction and connection

1.6.3.1 Introduction

This chapter provides information on assembly, construction and connection of control systems of the product family AC500.

The AC500 product family consists of the sub-families:

- AC500 (standard): standard PLC that offers a wide range of performance levels and scalability.
- AC500-eCo: cost-effective PLC that offers total inter-operability with the core AC500 range.
- AC500-S: PLC for special safety requirements in all functional safety applications.

AC500 (standard) and AC500-S provide devices with -XC extension as a product variant. Those devices operate mainly identical to the appropriate AC500 product family, however, can be operated under extreme conditions ↳ *Chapter 1.6.3.7.1 “System data AC500-XC” on page 5389*.

AC500 product family is characterized by functional modularity, i.e. the devices of all AC500 sub-families can be combined flexible.

As assembly, construction and connection for the devices of the AC500 product family is similar, information that is valid for all sub-families is provided within an overall section. Details that are only valid for a specific AC500 sub-family are described in separate sections.

As assembly, construction and connection for the devices of the AC500 product family is similar, information that is valid for all sub-families is provided within an overall section ↳ *Chapter 1.6.3.4 “Overall information (valid for complete AC500 product family)” on page 5218*. Details that are only valid for a specific AC500 sub-family are described in separate sections.

Consider the safety instructions

In the description, special attention must be paid to designs using galvanic isolation, grounding and EMC measures for the reasons stated. Consider the safety instructions for AC500 product family ↳ *Chapter 1.6.3.3 “Safety instructions” on page 5214.*

1.6.3.2 Regulations

Appropriate system setup

The following regulations have to be taken into due consideration:

- DIN VDE 0100: "Regulations for the Setting up of Power Installations"
- DIN VDE 0110 Part 1 and Part 2: "The Rating of Creepage Distances and Clearances"
- DIN VDE 0160 and DIN VDE 0660 Part 500: "The Equipment of Power Installations with Electrical Components"

To ensure project success and proper installation of all systems, customers must be familiar and proficient with the following standards and must comply with their directives:

- DIN VDE 0113 Part 1 & Part 200: "Working & Process Machinery"
- DIN VDE 0106 Part 100: "Close proximity to dangerous voltages"
- DIN VDE 0160, DIN VDE 0110 Part 1: "Protection against direct contact"

The user has to guarantee that the devices and the components are mounted following these regulations. For operating the machines and installations, other national and international relevant regulations, concerning prevention of accidents and using technical working means, also have to be met.

AC500 devices are designed according to IEC 1131 Part 2 under overvoltage category II per DIN VDE 0110 Part 2.

For direct connection of AC Category III overvoltages provide protection measures for overvoltage category II according to IEC-Report 664/1980 and DIN VDE 0110 Part 1.

Equivalent standards:

- DIN VDE 0110 Part 1 ↔ IEC 664
- DIN VDE 0113 Part 1 ↔ EN 60204 Part 1
- DIN VDE 0660 Part 500 ↔ EN 60439-1 ↔ IEC 439-1

All rights reserved to change design, size, weight, etc.

Qualified personnel

Both the control system AC500 and other components in the vicinity are operated with dangerous contact voltages. Touching parts, which are under such voltages, can cause grave damage to health.

In order to avoid such risks and the occurrence of material damage, persons involved with the assembly, starting up and servicing must possess pertinent knowledge of the following:

- Automation technology sector
- Dealing with dangerous voltages
- Using standards and regulations, in particular VDE, accident prevention regulations and regulations concerning special ambient conditions (e.g. areas potentially endangered by explosive materials, heavy pollution or corrosive influences).

1.6.3.3 Safety instructions

The examples and diagrams in this manual are included solely for illustrative purposes. Because of the many variants and requirements associated with any particular installation, ABB cannot assume responsibility or liability for actual use based on the examples and diagrams.

No patent liability is assumed by ABB with respect to use of information, circuits, equipment or software described in this manual. No liability is assumed for the direct or indirect consequences of the improper use, improper application or inadequate maintenance of these devices. In no event will ABB be responsible or liable for indirect or consequential damages resulting from the use or application of this equipment.

PLC specific safety notices

The product family AC500 control system is designed according to EN 61131-2 IEC 61131-2 standards. Data, different from IEC 61131, are caused by the higher requirements of Maritime Services. Other differences are described in the technical data description of the devices.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

NOTICE!

PLC damage due to operation conditions

Protect the devices from dampness, dirt and damage during transport, storage and operation!

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

Cleaning instruction

Do not use cleaning agent for cleaning the device.

Use a damp cloth instead.

Connection plans and user software must be created so that all technical safety aspects, legal regulations and standards are observed. In practice, possible shortcircuits and breakages must not be able to lead to dangerous situations. The extent of resulting errors must be kept to a minimum.

Do not operate devices outside of the specified, technical data!

Trouble-free functioning cannot be guaranteed outside of the specified data.

NOTICE!

PLC damage due to missing grounding

- Ensure to earth the devices.
- The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

In the description for the devices (operating manual or AC500 system description), reference is made at several points to grounding, galvanic isolation and EMC measures. One of the EMC measures consists of discharging interference voltages into the grounding via Y-type capacitors. Capacitor discharge currents must basically be able to flow off to the grounding (in this respect, see also VBG 4 and the relevant VDE regulations).

CAUTION!

Do not obstruct the ventilation for cooling!

The ventilation slots on the upper and lower side of the devices must not be covered.

CAUTION!

Run signal and power wiring separately!

Signal and supply lines (power cables) must be laid out so that no malfunctions due to capacitive and inductive interference can occur (EMC).

WARNING!

Labels on or inside the device alert people that dangerous voltage may be present or that surfaces may have dangerous temperatures.

WARNING!

Splaying of strands can cause hazards!

During wiring of terminals with stranded conductors, splaying of strands shall be avoided.

– Ferrules can be used to prevent splaying.

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

– connect or disconnect any signal or terminal block
– remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

Information on batteries

CAUTION!

Use only ABB approved lithium battery modules!

At the end of the battery's lifetime, always replace it only with a genuine battery module.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Environment considerations

Recycle exhausted batteries. Dispose batteries in an environmentally conscious manner, in accordance to local-authority regulations.

Environment and enclosure information

This equipment is intended for use in a Pollution Degree 2 industrial environment, in overvoltage Category II applications (as defined in IEC publication 60664-1), at altitudes up to 2.000 meters without derating.

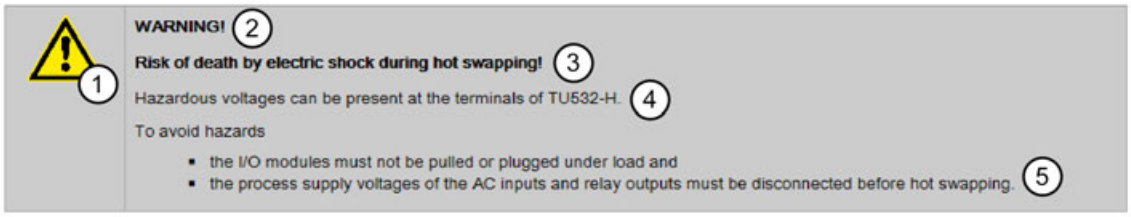
This equipment is considered Group 1, Class A industrial equipment according to IEC/CISPR Publication 11. Without appropriate precautions, there may be potential difficulties ensuring electromagnetic compatibility in other environments due to conducted as well as radiated disturbance.

This equipment is supplied as "open type" equipment. It must be mounted within an enclosure that is suitably designed for those specific environmental conditions that will be present and appropriately designed to prevent personal injury resulting from accessibility to live parts. The interior of the enclosure must be accessible only by the use of a tool. Subsequent sections of this publication may contain additional information regarding specific enclosure type ratings that are required to comply with certain product safety certifications.

Refer to NEMA Standards publication 250 and IEC publication 60529, as applicable, for explanations of the degrees of protection provided by different types of enclosure. Also see the appropriate sections in this manual.

1.6.3.3.1 Safety notice

Throughout the documentation we use the following types of safety and information notices according to ANSI Z535 make you aware of safety considerations or advice on AC500 products usage.



1 **Safety alert symbol** indicates the danger.
2 **Signal word** classifies the danger.
3 **Type and source of the risk** are mentioned.
4 **Possible consequences** of the risk are described.
5 **Measures to avoid these consequences** (enumerations).

Signal words

DANGER!
DANGER indicates a hazardous situation which, if not avoided, will result in death or serious injury.

Ensure to take measures to prevent the described impending danger.

WARNING!
WARNING indicates a hazardous situation which, if not avoided, could result in death or serious injury.

Ensure to take measures to prevent the described dangerous situation.

CAUTION!
CAUTION indicates a hazardous situation which, if not avoided, could result in minor or moderate injury.

Ensure to take measures to prevent the described dangerous situation.

NOTICE!
NOTICE is used to address practices not related to physical injury but might lead to property damage for example damage of the product.

Ensure to take measures to prevent the described dangerous situation.

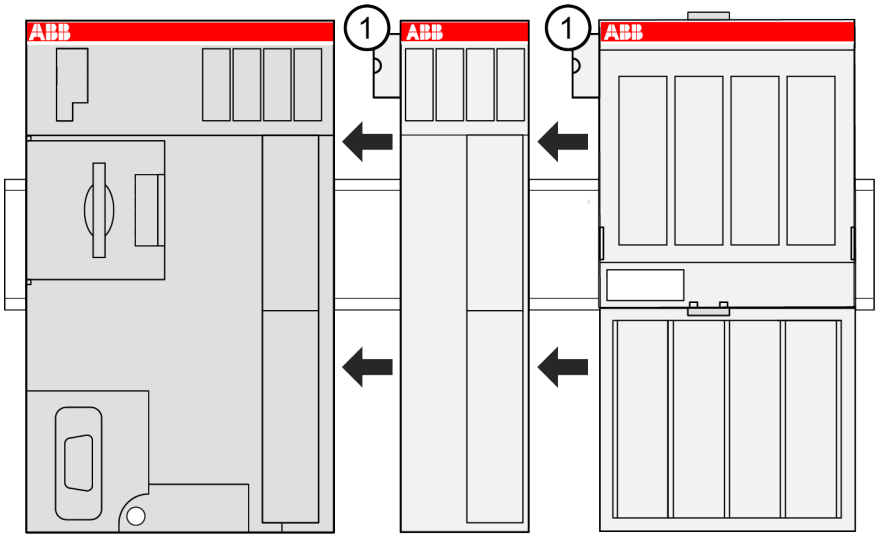
NOTE provides additional information on the product, e.g. advices for configuration or best practice scenarios.

1.6.3.4 Overall information (valid for complete AC500 product family)

1.6.3.4.1 Serial I/O bus

The synchronized serial I/O bus is the I/O data bus for the I/O modules connected with the processor modules or communication interface modules. Through this bus, I/O and diagnosis data are transferred.

Up to 10 I/O terminal units (for 1 I/O module each) can be added to one terminal base or to one AC500-eCo processor module. The I/O terminal units and the AC500-eCo I/O modules, have a bus input at the left side and a bus output at the right side. Thus the length of the I/O bus increases with the number of attached I/O modules.



1 I/O bus connection

The connection of the I/O bus is performed automatically by telescoping the modules on the DIN rail. The I/O bus provides the following signals:

- Supply voltage of 3.3 V DC for feeding the electronic interface components
- 3 data lines for the synchronized serial data exchange
- several control signals

NOTICE!

The I/O bus is not designed for plugging and unplugging modules while in operation. If a module is plugged or replaced while the bus is in operation, the following consequences are possible

- reset of the station or of the CPU
- system lockup
- damage of the module

WARNING!

Removal/Insertion under power

The devices are not designed for removal or insertion under power. Because of unforeseeable consequences, it is not allowed to plug or unplug devices with the power being ON.

Make sure that all voltage sources (supply and process voltage) are switched off before you

- connect or disconnect any signal or terminal block
- remove, mount or replace a module.

Disconnecting any powered devices while energized in a hazardous location could result in an electric arc, which could create a flammable ignition resulting in fire or explosion.

Make sure that power is removed and that the area has been thoroughly checked to ensure that flammable materials are not present prior to proceeding.

The devices must not be opened when in operation. The same applies to the network interfaces.

With its fast data transmission, the I/O bus obtains very low reaction times. Depending on the device and on the version of firmware and Automation Builder, the following numbers of I/O devices can be connected to the I/O bus.

Device	Version Automation Builder	Version firmware	Max. number of I/O devices
AC500-eCo PM55x and PM56x (-ETH variants only)	As of V2.0.0	As of V2.0.0	7
AC500-eCo PM55x and PM56x	As of V2.1.0	As of V2.0.6	10
CS31 bus modules DC551-CS31 and CI592-CS31-HA	All	All	7
CANopen bus modules CI581-CN and CI582-CN	As of V2.1.0	All	10
PROFIBUS bus modules CI541-DP and CI542-DP	As of V2.1.0	all	10
PROFINET bus modules CI504-PNIO and CI506-PNIO	As of V2.1.0	all	10
EtherCAT communication interface module CI511-ETHCAT and CI512-ETHCAT	As of V1.1	As of V2.0.x	10

Table 594: General data

Parameter	Value
Supply voltage, signal level	3.3 V DC ± 10 %
Max. supply current	On request
Type of the data interface	Synchronized serial data exchange
Bus data transmission speed	1.8 Mb/s
Minimum bus cycle time	500 µs [1])
Galvanic isolation	I/O bus is galvanic connected to CPU and communication interface logic ciruits. Galvanic isolation of I/O bus is I/O module specific. See each module specification for details.
Protection against electrostatic discharge (ESD)	TB5xx, TB56xx: with protection diodes, no ESD discharge allowed on the port.
Max. bus length	1 m
[1]) Minimum bus cycle time: This value is valid for all module combinations (from 1 to 10 I/O modules)	

Table 595: Wiring (bus connection)

Parameter	Value
Bus connection	Left-side and right-side connection from module to module via a 10-pole HE plug (male at the left side, female at the right side)
Mechanical connection	Established by the terminal units
Max. bus length	1 m

1.6.3.4.2 Mechanical encoding

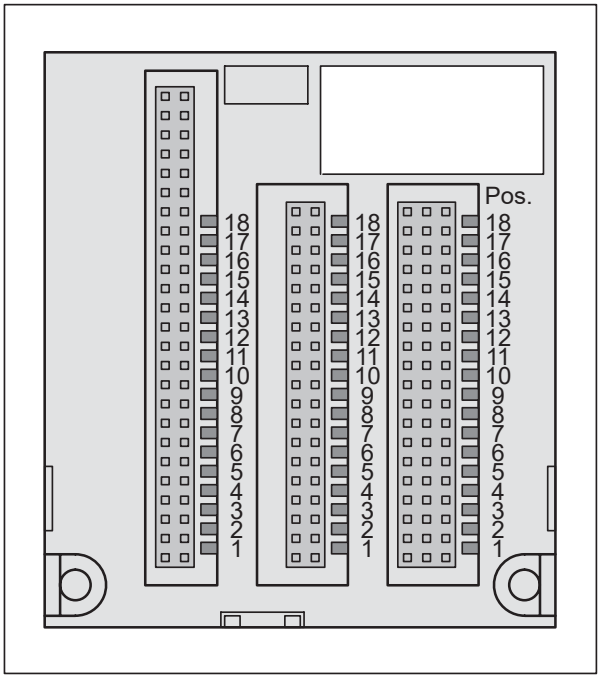


Fig. 1032: Possible positions for mechanical encoding (1 to 18)

NOTICE!
Terminal units and terminal bases have a mechanical coding which prevents modules (from) being inserted into the wrong places for cases that might result in dangerous parasitic voltages or if modules could be destroyed.

The coding either makes it impossible to insert the module to the wrong place or blocks its electrical function (outputs are not activated).

The following figures show the possible encodings.

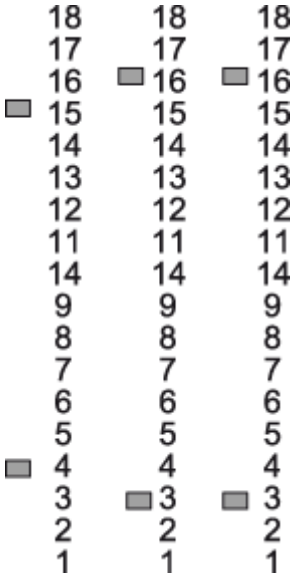


Fig. 1033: Encoding for processor modules with Ethernet interface

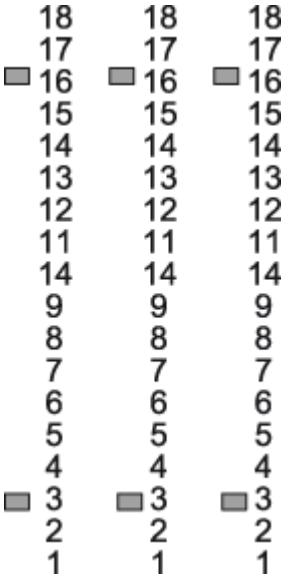


Fig. 1034: Encoding for processor modules with ARCNET interface

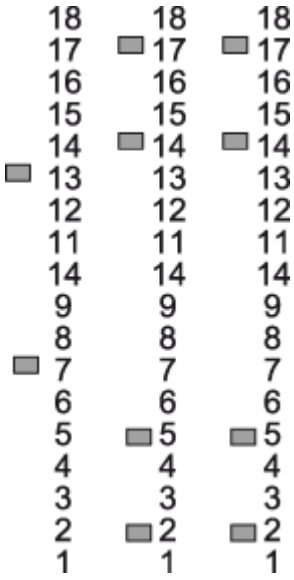


Fig. 1035: Encoding for real-time Ethernet modules

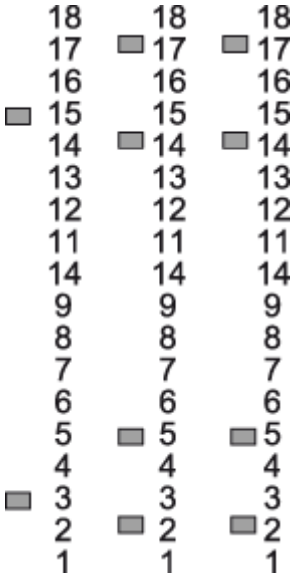


Fig. 1036: Encoding for communication interface modules

Fig. 1037: Encoding for I/O modules (24 VDC)

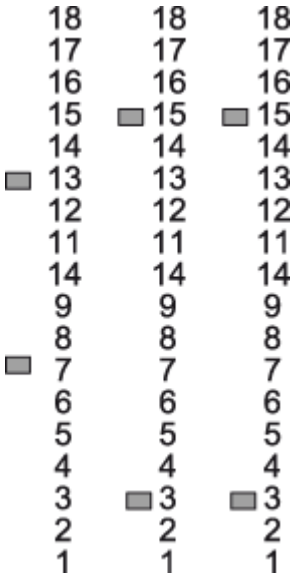


Fig. 1038: Encoding for communication interface modules with PROFINET interface

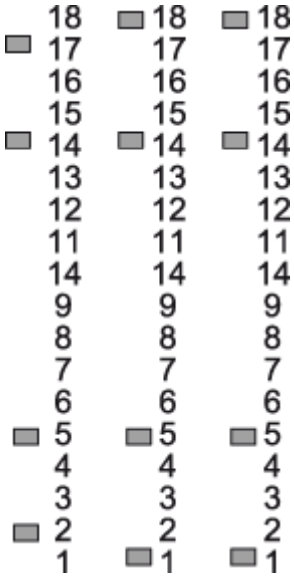


Fig. 1039: Encoding for I/O modules (120 VAC / 230 VAC)

Fig. 1040: Encoding for positioning modules

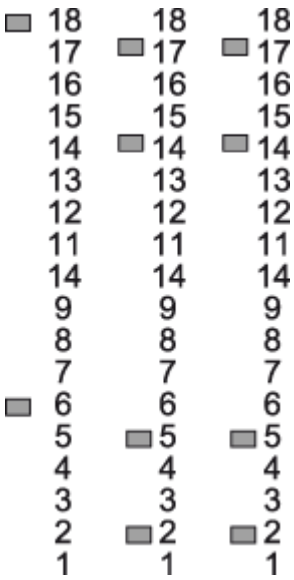


Fig. 1041: Encoding for CS31 fieldbus modules

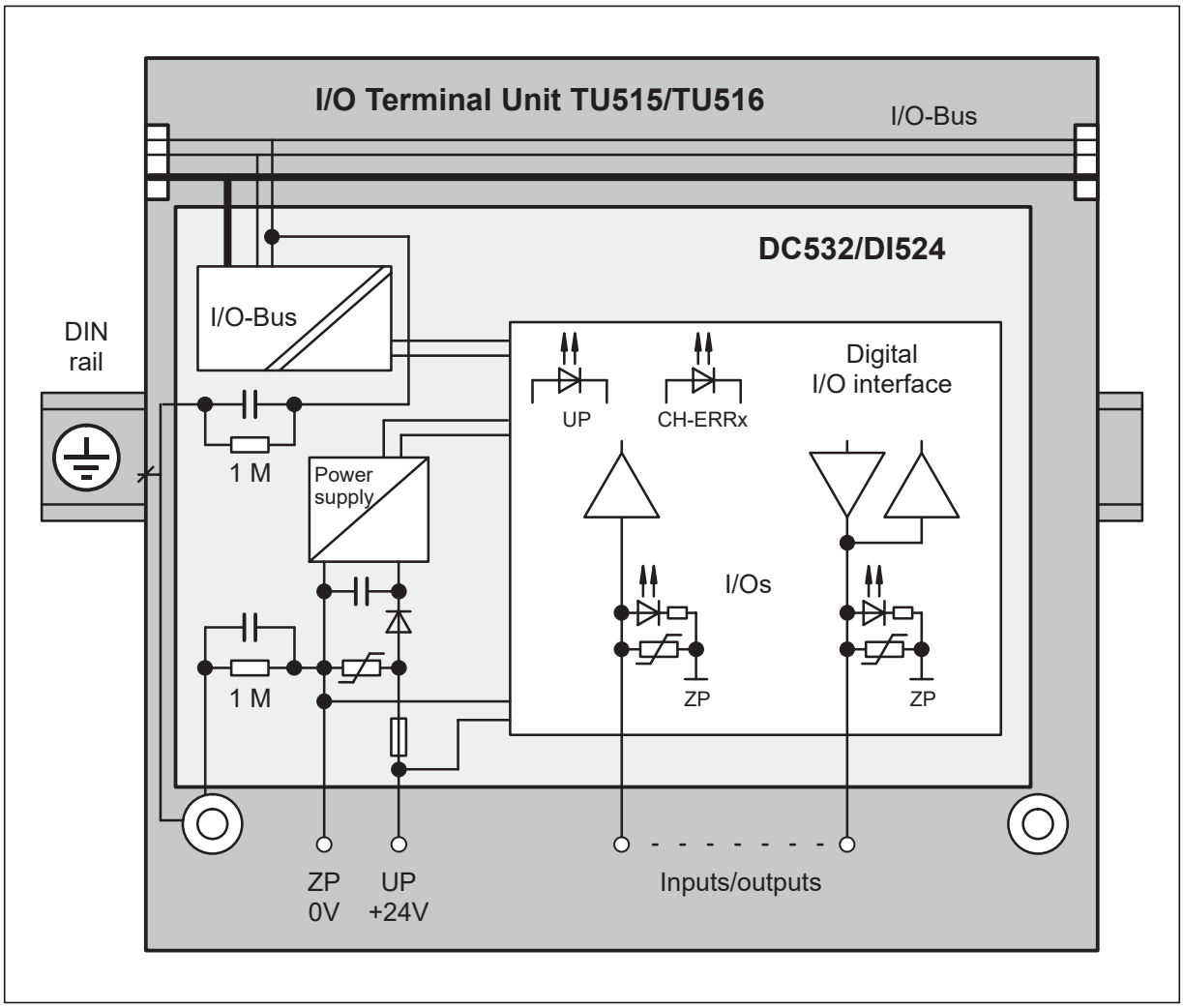
1.6.3.4.3 Earthing concept (Block diagrams)

NOTICE!

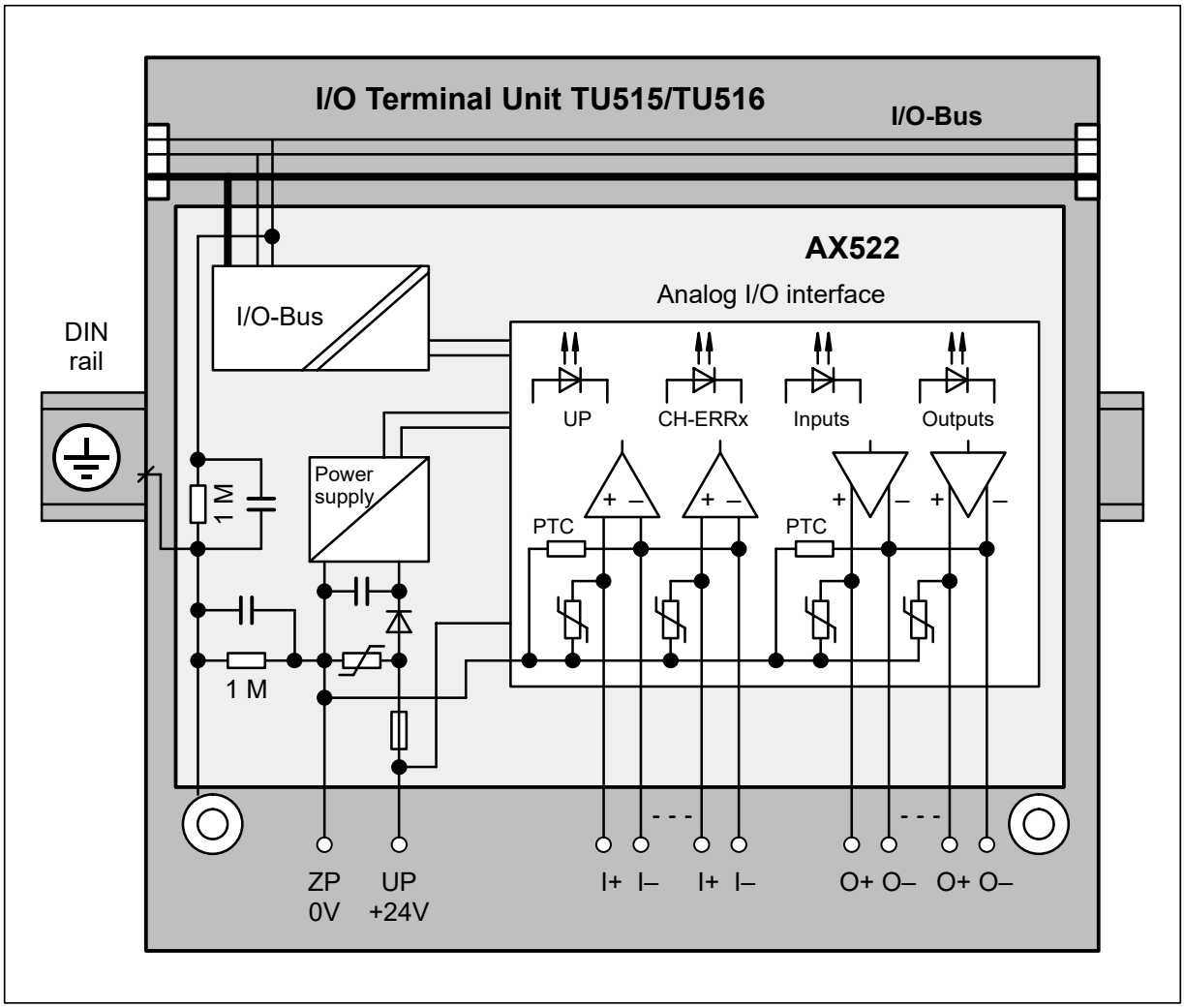
PLC damage due to missing grounding

- Ensure to earth the devices.
- The grounding (switch cabinet grounding, PE) is supplied both by the mains connection (or 24 V supply voltage) and via DIN rail. The DIN rail must be connected to the ground before the device is subjected to any power. The grounding may be removed only if it is certain that no more power is being supplied to the control system.

Block diagram: Digital I/O modules



Block diagram: Analog I/O modules



1.6.3.4.4 EMC-conforming assembly and construction

General principles

General considerations

Electric and electronical devices have to work correctly on site. This is also valid when electro-magnetic influences affect them in defined and/or expected strength. The devices themselves must not emit electro-magnetic noises.

Advant Controller components have a very high noise immunity.

When the wiring and grounding instructions are met, an error-free operation is given.

High electro-magnetic noises of nearby mounted applications must be taken in consideration during the planning phase.

An EMC compatible earthing concept will also guarantee an error-free operation here.

There are three important principles to be especially considered:

- *Keep all connections as short as possible (in particular the grounding conductors)*
- *Use large conductor cross sections (in particular for the grounding conductors)*
- *Create low-impedance, i.e. good and large-sized contacts (in particular for the grounding conductors)*

Pay attention to the following:

- *Use vibration-resistant connections*
- *Clean metallic contact areas*
- *Use solid plug and screw-type connections*
- *Use earth cable shields with clips on a well-grounded metallic surface*
- *Do not use aluminium parts*
- *Do not use sheath wires*
- *Do not use toothed lock washers under screw connections*

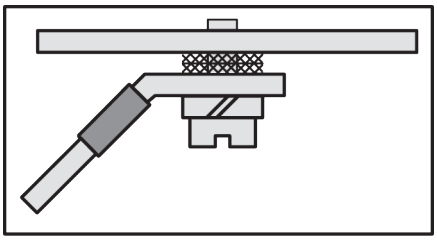


Fig. 1042: Assembly: wrong

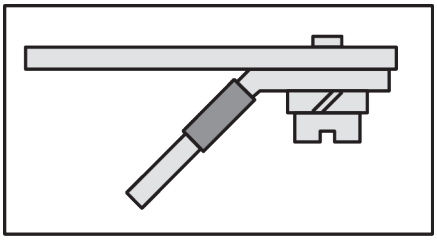


Fig. 1043: Assembly: correct

Make a connection between the DIN rails and PE (Protective Earth). For this, use an grounding wire with a minimum conductor cross section of 10 mm^2.

The wire is connected to the DIN rail with an M6 screw.

A large-area contact of the DIN rail with the metallic mounting plate improves the EMC behavior significantly, as the disturbances can be discharged more effective.

Cable routing

- Route cables meeting the standards.
- Sort the cables into cable groups:
  - Power current cables
  - Power supply cables
  - Signal cables
  - Data cables

- Rout signal cables and data cables separately from the power cables.
  – Separate cable ducts or cable bundles.
  – The distance should be 20 cm or greater.
- Lay signal and data cables close to earthed surfaces.

Cable shields

- Use only shielded data cables. The shield should be grounded at both ends.
  A cable shield only grounded at one end can only protect from capacitively coupled interference and low-frequency disturbances (50 Hz hum).
- Avoid parasitic currents flowing through the cable shields.
  This can be done by installing current-carrying equipotential bondings.
- Use only cables with braided shields.
  Foil shields are not robust enough, cannot be contacted well and have poor HF properties.
- Use only metallic or [metal]-plated plugs for shielded data cables.
- Use only shielded cables for analog signals.
  For small signals ground the shield only at one end.
- Ground the cable shield directly with a clip when entering the switchgear cabinet.
  Do not cut the shield until the cable reaches the module connected.

The connection between the PE bar and the shield bar must have a low impedance.

Switchgear cabinet

According to DNV GL mounting in a seperate metall cabinet is required for:

- *FM502-CMS*
- *FM502-CMS-XC*
- *TF501-CMS*
- *TF501-CMS-XC*
- *TF521-CMS*
- *TF521-CMS-XC*
- *SM560-S-FD-1*
- *CI521-MODTCP*
- *CI522-MODTCP*
- *CM589-PNIO*

Connections The connections between the switchgear cabinet, the mounting plates, the PE bar and the shield bar must have a low impedance.

Grounding Ground the switchgear cabinet doors with short and highly flexible conductors.

Illumination Only use filament lamps (bulbs) or fluorescent tubes with interference suppression.

For supplying the PC Use the mains socket which is located inside the switchgear cabinet.

↳ *Chapter 1.6.3.5.2.1 “Switchgear cabinet assembly” on page 5238*

Reference potential

- Provide a uniform reference potential in the entire installation and ground all electrical appliances if possible.
- Route your grounding conductors in a star configuration so that no ground loops can occur.

Equipotential bonding

The Installation of equipotential bondings are necessary if there are present or expected potential differences between parts of your application.

- *The impedance of equipotential bonding must be equal or lower than 10 % of the shield impedance of the shielded signal cables between the same points.*
- *The conductor cross section of a equipotential bonding must be 16 mm² to withstand the maximum possible compensating current.*
- *Equipotential bondings and shielded signal cables should be laid close to each other.*
- *Equipotential bondings must be connected to PE with low impedance.*

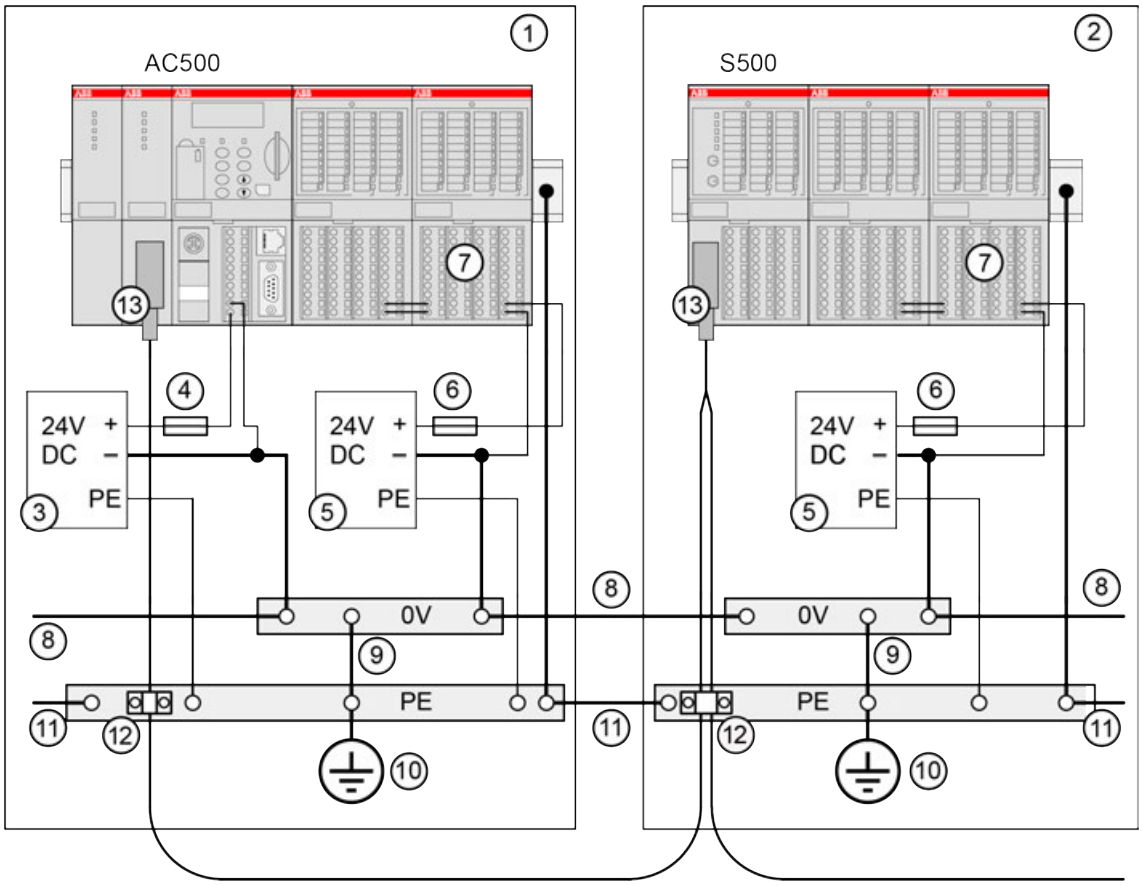


Fig. 1044: AC500, equipotential bonding

1 Cabinet 1
2 Cabinet 2
3 Power supply for the CPU
4 Fuse for the CPU power
5 Power supply for the I/Os
6 Fuse for the I/O power
7 For fuses for the contacts of the relay outputs
8 0V rail
9 Grounding of the 0V rail
10 Cabinet grounding
11 Equipotential bonding between the cabinets min. 16 mm^2
12 Cable shields grounding
13 Fieldbus connection (e.g. Ethernet)

1.6.3.4.5 Power consumption of an entire station

The power consumption of a complete station consists of the sum of all individual consumptions.

- Consumers over terminals L+ and M on the AC500 terminal base/AC500-eCo CPU:
  - CPU itself
  - I/O modules attached on the I/O bus
  - Communication modules attached (AC500 terminal base)
- Consumers over the process supply voltage terminals ZP and UP of the AC500 terminal units / the L+/M or UP/ZP terminals of the AC500-eCo I/O modules:
  - Digital I/O modules
  - Analog I/O modules

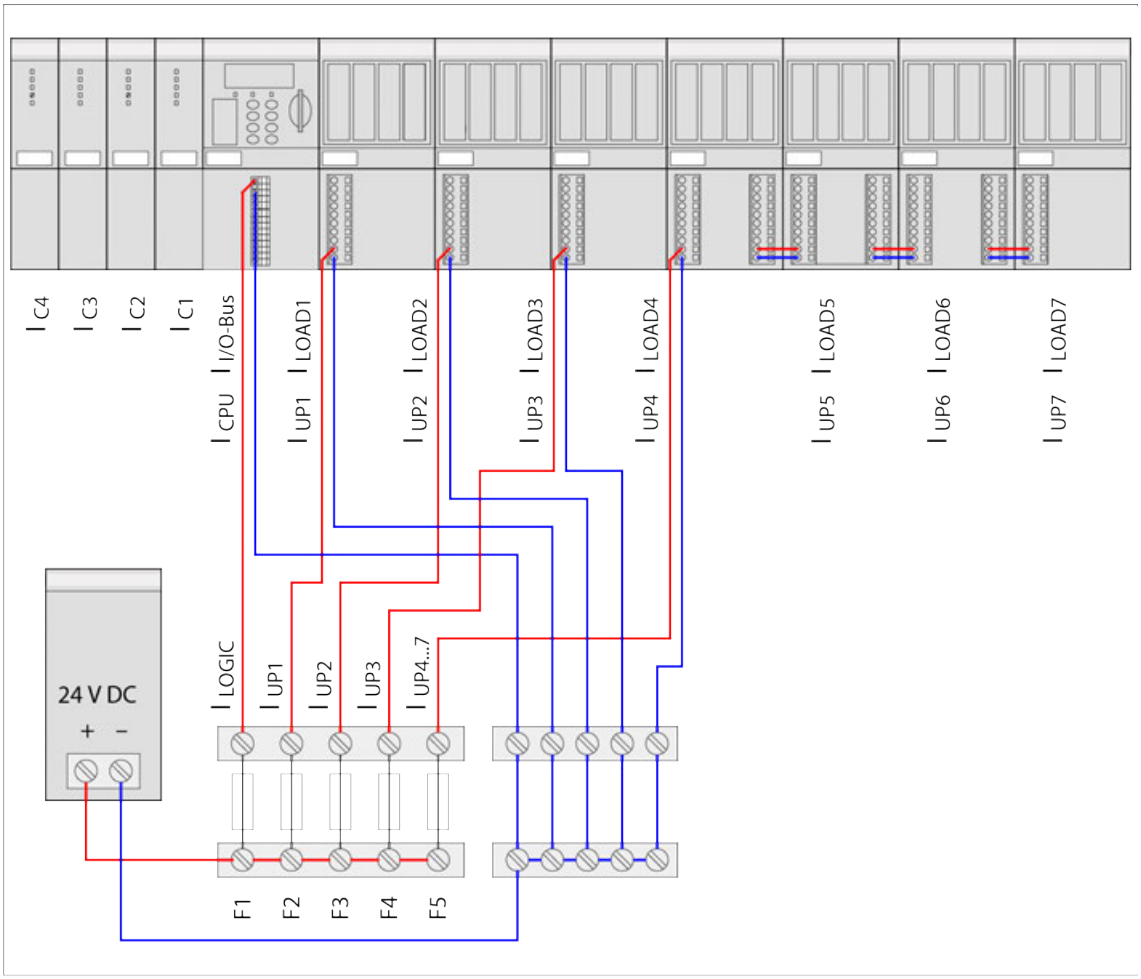
The two supply voltages can be provided by the same power supply unit. The CPU and the I/O modules should, however, be fused separately. Of course also separate power supplies are possible.

Calculation of the total current consumption

Example

In the example, the AC500 control system consists of the following devices:

- AC500 CPU with Ethernet interface
- 4 communication modules
- 7 I/O modules (digital and analog)
- As well as the required terminal bases and terminal units



Because of the high total current consumption of the digital I/O modules (from UP = 24 V DC), the supply is divided up into several electric circuits fused separately.

The maximum permitted total current over the supply terminals of the I/O terminal units is 8 A.

The total current can be calculated as follows:

$I_{Total} = I_{LOGIC} + I_{UP}$

with the assumptions

$I_{LOGIC} = I_{CPU} + I_{I/O\ bus} + I_{C1} + I_{C2} + I_{C3} + I_{C4}$ (CPU + communication modules + I/O bus)

$I_{I/O\ bus}$ = Number of expansion modules × Current consumption through the I/O bus per module

and

$I_{UP} = I_{UP1} + I_{LOAD1} + I_{UP2} + I_{LOAD2} + I_{UP3} + I_{LOAD3} + I_{UP4} + I_{LOAD4} + I_{UP5} + I_{LOAD5} + I_{UP6} + I_{LOAD6} + I_{UP7} + I_{LOAD7}$

If one assumes that all outputs are switched on and are operated with their maximum permitted load currents (under compliance with the maximum permitted currents at the supply terminals), then the following values are the result for an example shown above:

	I_{CPU} *)	I_{Cx} *)	$I_{I/O\ bus}$ *)	I_{UPx} *)	I_{LOADx} *)
CPU / communication module part					
CPU	0.110 A	-	-	-	-
C1	-	0.050 A	-	-	-
C2	-	0.085 A	-	-	-
C3	-	0.050 A	-	-	-
C4	-	0.050 A	-	-	-
I/O module part					
Analog1	-	-	0.002 A	0.150 A	-
Analog2	-	-	0.002 A	0.150 A	0.160 A
Analog3	-	-	0.002 A	0.100 A	0.080 A
Analog4	-	-	0.002 A	0.100 A	0.080 A
Digital1	-	-	0.002 A	0.050 A	8.000 A
Digital2	-	-	0.002 A	0.050 A	8.000 A
Digital3	-	-	0.002 A	0.050 A	8.000 A
Σ columns	0.110 A	0.235 A	0.014 A	0.650 A	24.320 A
	$\Sigma\ I_{LOGIC} \approx 0.4$ A			$\Sigma\ I_{UP} \approx 25$ A	
	$I_{Total} \approx 25.4$ A				
*) All values in this column are exemplary values					

Dimensioning of the fuses

To be able to select the fuses for the station correctly, both the current consumption and the inrush currents (melting integral for the series-connected fuse) must be taken into consideration.

Fuse	**for**	**Σ of the melting integrals in A^2s**	**$I_{Logic\ A}$**	**$I_{UPx\ A}$**	**Recommended fuse**	
					Type	**Value**
F1	CPU logic	1.000	≈ 0.4	-	Quick	10 A
F2	Module Digital1	0.005	-	8.050	Quick	10 A
F3	Module Digital2	0.008	-	8.050	Quick	10 A
F4	Module Digital3	0.007	-	8.050	Quick	10 A
F5	Modules Analog1 + Analog2 + Analog3 + Analog4	0.130	-	0.820	Quick	10 A

1.6.3.4.6 Decommissioning

Secure decommissioning of a functional CPU

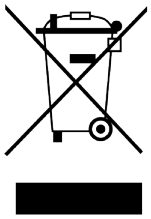
1. Delete applications.
2. Delete applications from memory card, if available.
3. If available, remove memory card and battery from CPU.
4. Demount and dispose the hardware modules ↳ *Chapter 1.6.3.5.3 “Mounting and demounting” on page 5241* ↳ *Chapter 1.6.3.6.3 “Mounting and demounting” on page 5325* ↳ *Chapter 1.6.3.4.7 “Recycling” on page 5233*.

Secure decommissioning of a not functional CPU

▷ If you can not access the data stored in the CPU, e.g., because the CPU is not functional any more, then physically destroy the device.

⇨ This ensures that the credentials that are stored in the device, can not be misused.

1.6.3.4.7 Recycling



Disposal and recycling information

This symbol on the product (and on its packaging) is in accordance with the European Union's Waste Electrical and Electronic Equipment (WEEE) Directive.

The symbol indicates that this product must be recycled/disposed of separately from other household waste.

It is the end user’s responsibility to dispose of this product by taking it to a designated WEEE collection facility for the proper collection and recycling of the waste equipment.

The separate collection and recycling of waste equipment will help to conserve natural resources and protect human health and the environment.

For more information about recycling, please contact your local environmental office, an electrical/electronic waste disposal company or the store where you purchased the product.

1.6.3.5 AC500-eCo

1.6.3.5.1 System data AC500-eCo

Environmental conditions

Table 596: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
24 V AC		
	Voltage	24 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
100 V AC		
	Voltage	100 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
230 VAC		
	Voltage	230 V (-15 %, +10 %)

Parameter		Value
	Frequency	50/60 Hz (-6 %, +4 %)
100...240 V AC wide-range supply		
	Voltage	100 V...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply, according to EN 61131-2		
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2
	AC supply	Interruption < 0.5 periods, time between 2 interruptions > 1 s

NOTICE!
Exceeding the maximum power supply voltage (> 30 V DC) for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

Parameter		Value
Temperature		
	Operating	0 °C...+60 °C (horizontal mounting of modules) 0 °C...+40 °C (vertical mounting of modules and output load reduced to 50 % per group)
	Storage	-40 °C...+70 °C
	Transport	-40 °C...+70 °C
Humidity		Max. 95 %, without condensation
Air pressure		
	Operating	> 800 hPa / < 2000 m
	Storage	> 660 hPa / < 3500 m

Creepage distances and clearances

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Insulation test voltages, routine test

According to EN 61131-2

Parameter	Value	
200 V...240 V circuits against other circuitry	2500 V	1.2/50 µs
100 V...127 V circuits against other circuitry	1500 V	1.2/50 µs
100 V...240 V circuits against other circuitry	2500 V	1.2/50 µs

Parameter	Value	
24 V circuits (supply, 24 V inputs/outputs, analog inputs/ outputs), if they are galvanically isolated against other circuitry	500 V	1.2/50 µs
COM interfaces, galvanically isolated	500 V	1.2/50 µs
COM interfaces, electrically not isolated	Not applicable	Not applicable
FBP interface	500 V	1.2/50 µs
Ethernet	500 V	1.2/50 µs
ARCNET	500 V	1.2/50 µs
200 V... 240 V circuits against other circuitry	1350 V	AC 2 s
100 V circuits against other circuitry	820 V	AC 2 s
100 V...240 V circuits against other circuitry	1350 V	AC 2 s
24 V circuits (supply, 24 V inputs/outputs, analog inputs/ outputs), if they are galvanically isolated against other circuitry	350 V	AC 2 s
COM interfaces, galvanically isolated	350 V	AC 2 s
COM interfaces, electrically not isolated	Not applicable	Not applicable
FBP interface	350 V	AC 2 s
Ethernet	350 V	AC 2 s
ARCNET	350 V	AC 2 s

Power supply units

For the supply of the modules, power supply units according to SELV or PELV specifications must be used.

Safety Extra Low Voltage (SELV) and Protective Extra Low Voltage (PELV)

To ensure electrical safety of AC500/AC500-eCo extra low voltage circuits, 24 V DC supply, communication interfaces, I/O circuits, and all connected devices must be powered from sources meeting requirements of SELV, PELV, class 2, limited voltage or limited power according to applicable standards.

WARNING!

Improper installation can lead to death by touching hazardous voltages!

To avoid personal injury, safe separation, double or reinforced insulation and separation of the primary and secondary circuit must be observed and implemented during installation.

– Only use power converters for safety extra-low voltages (SELV) with safe galvanic separation of the primary and secondary circuit.
– Safe separation means that the primary circuit of mains transformers must be separated from the secondary circuit by double or reinforced insulation. The protective extra-low voltage (PELV) offers protection against electric shock.

Electromagnetic compatibility

Electromagnetic Compatibility		
Device suitable for:		
	Industrial applications	Yes
	Domestic applications	No
Immunity against electrostatic discharge (ESD):		According to IEC 61000-4-2, zone B, criterion B
	Electrostatic voltage in case of air discharge	8 kV
	Electrostatic voltage in case of contact discharge	4 kV, in a closed switchgear cabinet 6 kV [1])
	ESD with communication connectors	In order to prevent operating malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.
Immunity against the influence of radiated (CW radiated):		According to IEC 61000-4-3, zone B, criterion A
	Test field strength	10 V/m
Immunity against transient interference voltages (burst):		According to IEC 61000-4-4, zone B, criterion B
	Supply voltage units (DC)	2 kV
	Supply voltage units (AC)	2 kV
	Digital inputs/outputs (24 V DC / 24 VAC)	1 kV
	Digital inputs/outputs (100 V AC...240 V AC)	2 kV
	Analog inputs/outputs	1 kV
	Serial RS-485 interfaces (COM)	1 kV
	Ethernet	1 kV
	I/O supply, DC-out	1 kV
Immunity against the influence of line-conducted interferences (CW conducted):		According to IEC 61000-4-6, zone B, criterion A
	Test voltage	10 V

Electromagnetic Compatibility		
High energy surges		According to IEC 61000-4-5, zone B, criterion B
	Power supply AC	2 kV CM / 1 kV DM [2])
	Power supply DC	1 kV CM / 0.5 kV DM [2])
	DC I/O supply, add. DC-supply-out	1 kV CM / 0.5 kV DM [2])
	Communication lines, shielded	1 kV CM [2])
	AC I/O unshielded [3])	2 kV CM / 1 kV DM [2])
	I/O analog, I/O DC unshielded [3])	1 kV CM / 0.5 kV DM [2])
Radiation (radio disturbance)		According to IEC 55011, group 1, class A

[1]) High requirement for shipping classes are achieved with additional specific measures (see specific documentation).

[2]) CM = Common Mode, DM = Differential Mode

[3]) When DC I/O inputs are used with AC voltage, external filters limiting high energy surges to 1 kV CM / 0.5 DM are required to meet requirements according IEC 61131-2.

Mechanical data

Parameter	**Value**
Mounting	Horizontal
Degree of protection	IP 20 (if all terminal screws are tightened)
Housing	Classification V-2 according to UL 94
Vibration resistance acc. to EN 61131-2	all three axes (DIN rail mounting) 5 Hz...8.4 Hz, continuous 3.5 mm 8.4 Hz...150 Hz, continuous 1 g
Shock test	All three axes 15 g, 11 ms, half-sinusoidal
Mounting of the modules:	
DIN rail according to DIN EN 50022	35 mm, depth 7.5 mm or 15 mm
Mounting with screws	Screws with a diameter of 4 mm
Fastening torque	1.2 Nm

Approvals and certifications

Information on approvals and certificates can be found in the corresponding chapter of the *Main catalog, PLC Automation*.

1.6.3.5.2 Mechanical dimensions

Switchgear cabinet assembly

Information on EMC-conforming assembly and construction is provided within the overall functions section ↳ Chapter 1.6.3.4.4 “EMC-conforming assembly and construction” on page 5226.

PLC enclosure

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

To protect PLCs against:

- unauthorized access,
- dusting and pollution,
- moisture and wetness and
- mechanical damage,

switchgear cabinet IP54 for common dry factory floor environment is suitable.

Maintain spacing from:

- enclosure walls
- wireways
- adjacent equipment

Allow a minimum of 20 mm clearance on all sides. This provides ventilation and galvanic isolation.

It is recommended to mount the modules on an grounded mounting plate, or an grounded DIN rail, independent of the mounting location.

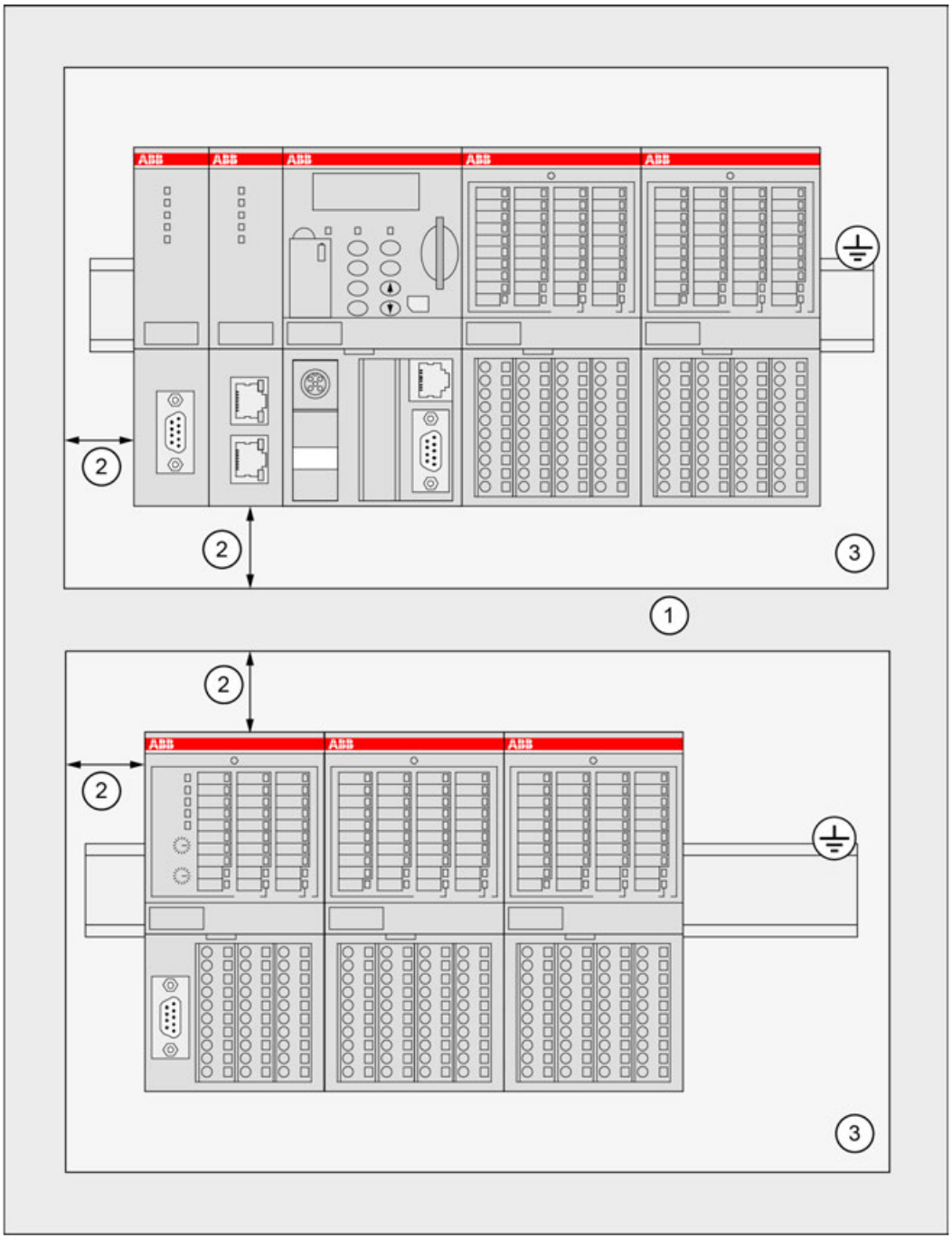


Fig. 1045: Installation of AC500/S500 modules in a switchgear cabinet

1 Cable duct
2 Distance from cable duct ≥20 mm
3 Mounting plate, grounded

NOTICE!

Horizontal mounting is highly recommended.

Vertical mounting is possible, however, derating consideration should be made to avoid problems with poor air circulation and overheating (see ↳ *Chapter 1.6.3.6.1.1 “Environmental conditions” on page 5313*).

When vertically mounted, always place an end-stop terminal block (e.g. type BADL, P/N: 1SNA399903R0200) on the bottom and on the top of the modules to properly secure the modules.

With high vibration applications and horizontal mounting, we also recommend to place end-stop terminals at the right and left side of the device to properly secure the modules, e.g. type BADL, P/N: 1SNA399903R0200.

Mechanical dimensions AC500-eCo

All mechanical dimensions are given in millimeters and inches. The value in brackets is the inch-value.

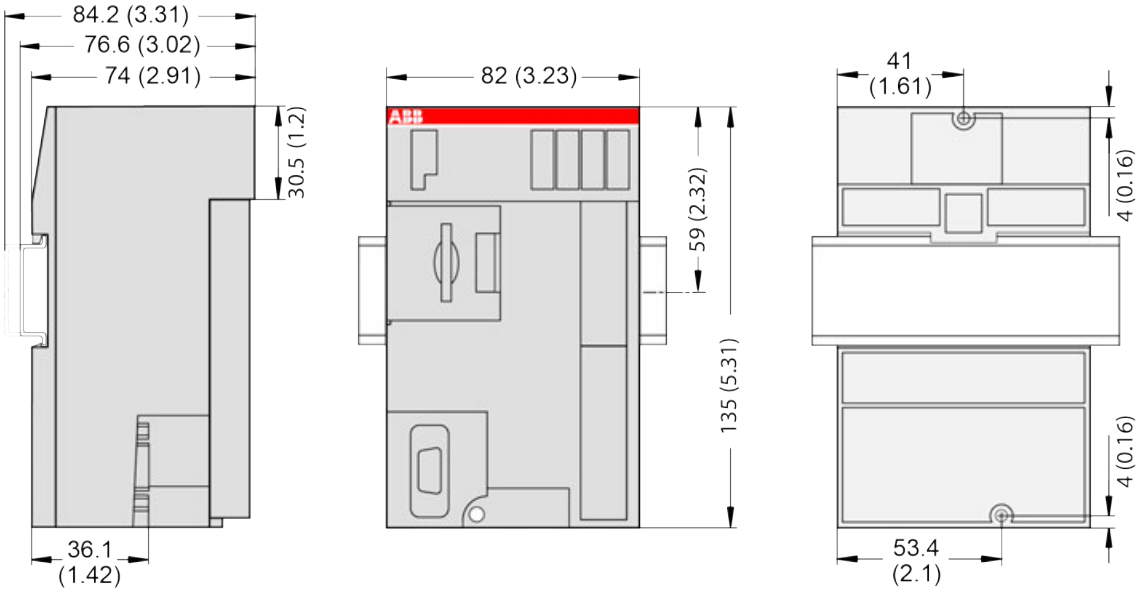


Fig. 1046: Side, front and back view

Mechanical dimensions S500-eCo

All mechanical dimensions are given in millimeters and inches. The value in brackets is the inch-value.

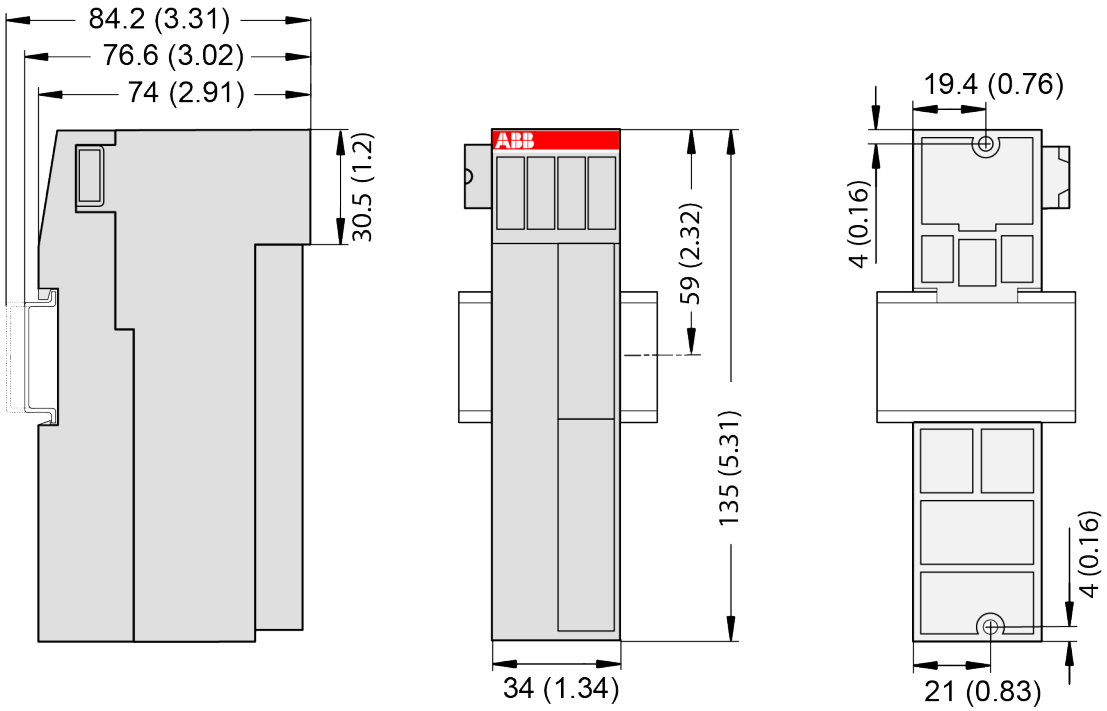


Fig. 1047: Side, front and back view

1.6.3.5.3 Mounting and demounting

The control system is designed to be mounted to a well-grounded mounting surface such as a metal panel. Additional grounding connections from the mounting tabs or DIN rail (if used), are not required unless the mounting surface cannot be grounded.

During panel or DIN rail mounting of all devices, be sure that all debris (metal chips, wire strands, etc.) is kept from falling into the controller. Debris that falls into the controller could cause damage while the controller is energized.

All devices are grounded through the DIN rail to chassis ground. Use zinc plated yellow-chromate stell DIN rail to assure proper grounding. The use of other DIN rail materials (e.g. aluminium, plastic, etc.) that can corrode, oxidize, or are poor conductors, can result in improper or intermittent grounding.

Mounting and demounting of the AC500-eCo CPUs

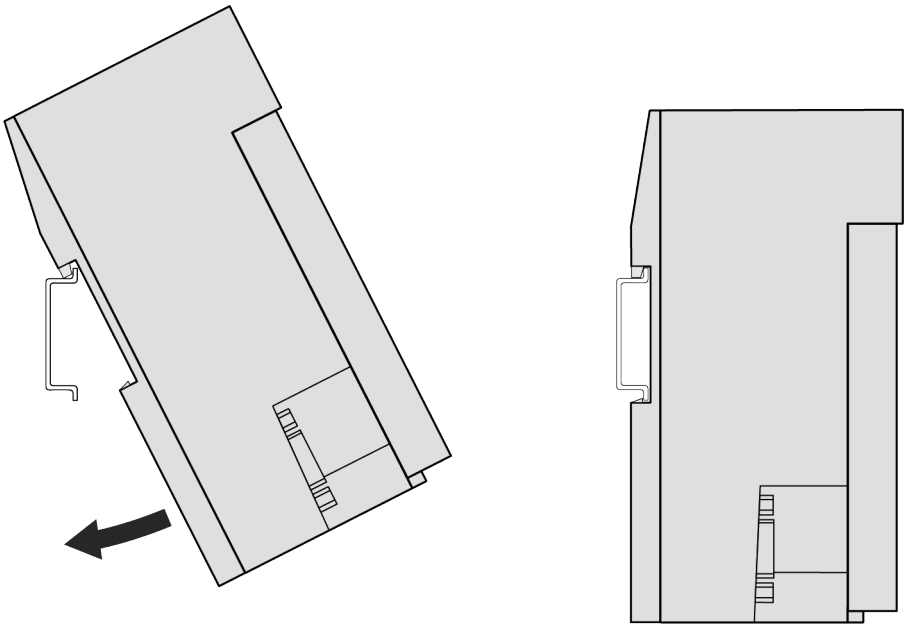
Mounting a processor module on a DIN rail

NOTICE!

Risk of function faults!

The processor module is grounded via DIN rail.

The DIN rail must be included into the earthing conception of the plant.

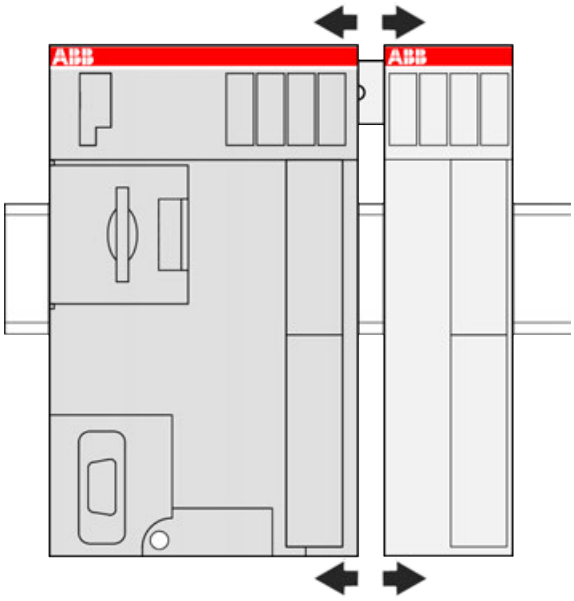


Mount the processor module at the top of the DIN rail, then snap it in below.

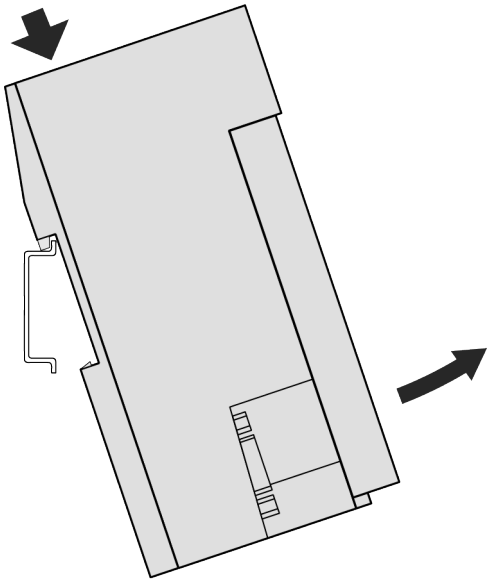
> *See Hardware description of PM55x-xP and PM56x-xP ↪ Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804 for connection.*

Demounting a processor module mounted on a DIN rail

1. Remove I/O modules if connected.



2. While pressing down processor module pull it away from DIN rail.

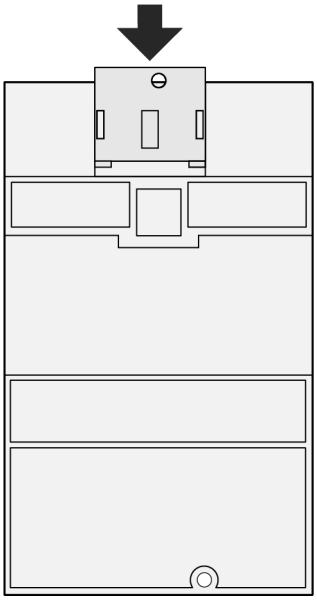


Mounting a processor module on a metal plate

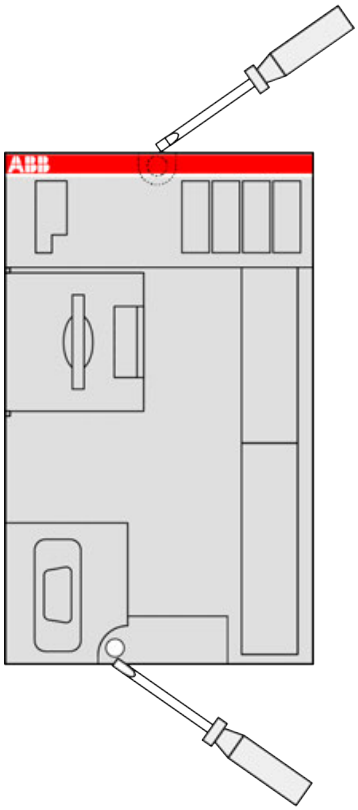
NOTICE!
Risk of function faults!
Missing electrical contact by isolating screws or washers!
Use metal screws on the metal plate.
The metal plate must be included into the earthing concept of the plant.
Do NOT use insulating washers!

One TA566 Wall Mounting Accessory ↪ Chapter 1.6.2.9.3.2 “ TA566 - Wall mounting accessory” on page 5205 is needed per processor module.

1. Snap in the TA566 at the back side of the processor module.



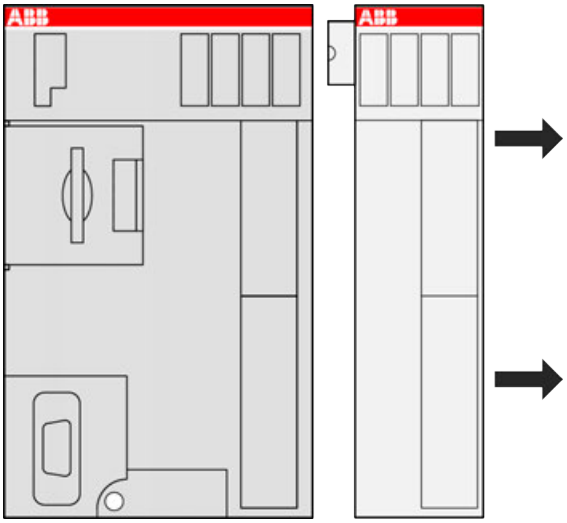
2. Fasten the processor module with two screws (max. diameter: 4 mm) to the metal plate.



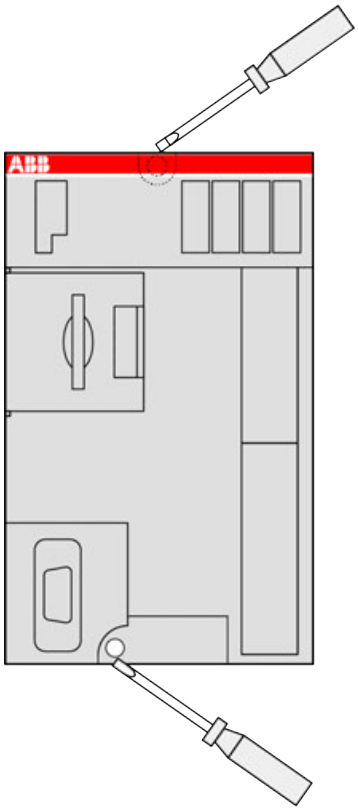
See Hardware description of PM55x-xP and PM56x-xP ↪ Chapter 1.6.2.3.1.1 “PM55x-xP and PM56x-xP” on page 3804 for connection.

Demounting a processor module mounted on a metal plate

1. Remove I/O modules if connected.



2. Remove the 2 screws.



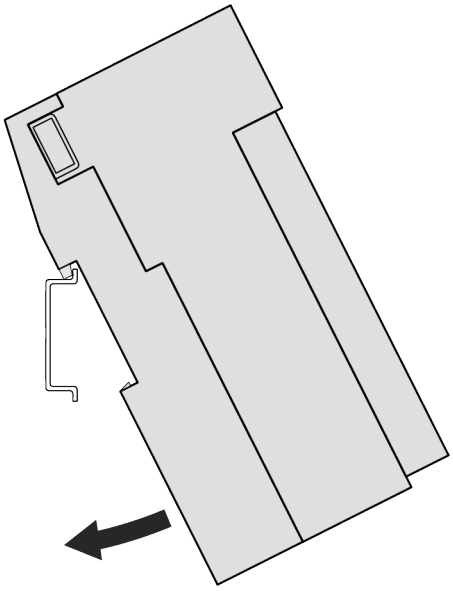
Mounting and demounting of S500-eCo I/O modules

S500-eCo I/O modules can be mounted either on a DIN rail or with screws on a metal plate.

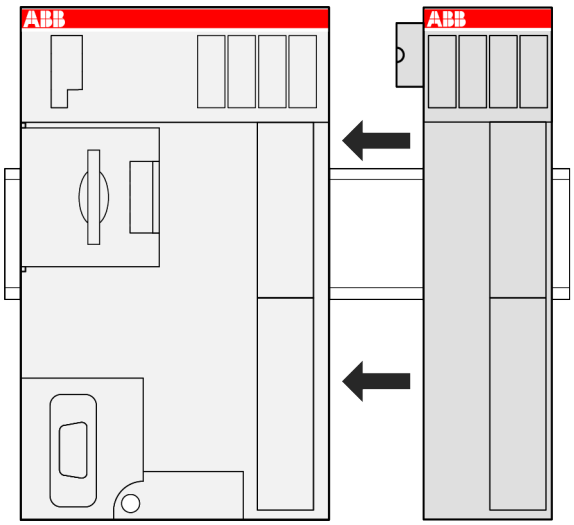
Mounting I/O modules on a DIN rail

NOTICE!
Risk of function faults!
The S500-eCo I/O modules are grounded via the DIN rail.
The DIN rail must be included into the earthing concept of the plant.
Use only metal screws.

1. Mount I/O module at the top of the DIN rail, then snap it in below.

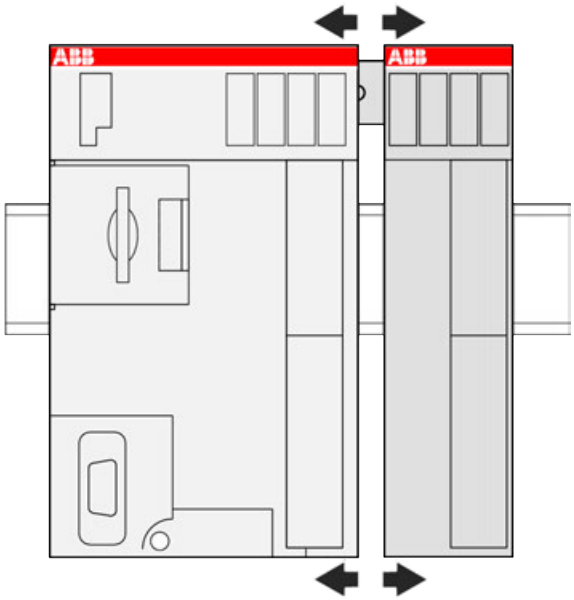


2. Attach I/O module by hand to an other module. The serial I/O bus is connected automatically.

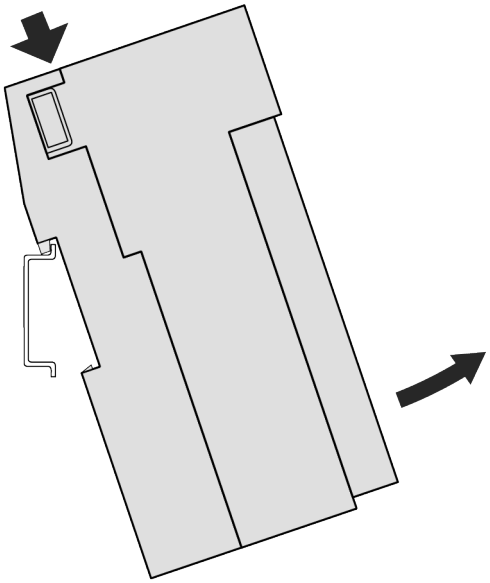


Demounting I/O modules mounted on a DIN rail

1. Remove I/O module by hand if connected.



2. While pressing down I/O module pull it away from DIN rail.



Mounting I/O modules on a metal plate

NOTICE!
Risk of function faults!
Missing electrical contact by isolating screws or washers!

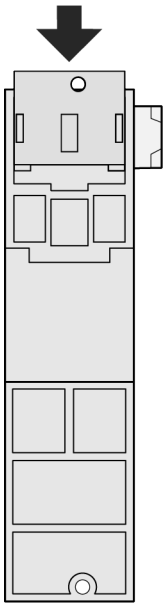
Use metal screws on the metal plate.

The metal plate must be included into the earthing concept of the plant.

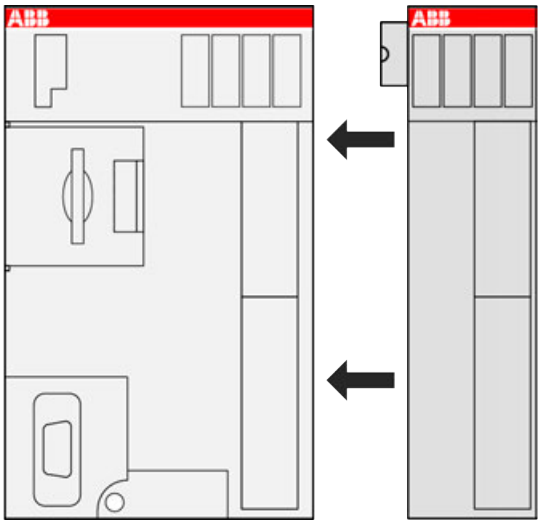
Do NOT use insulating washers!

> *One TA566 wall mounting accessory ↳ Chapter 1.6.2.9.3.2 “ TA566 - Wall mounting accessory” on page 5205 is needed per S500-eCo I/O module.*

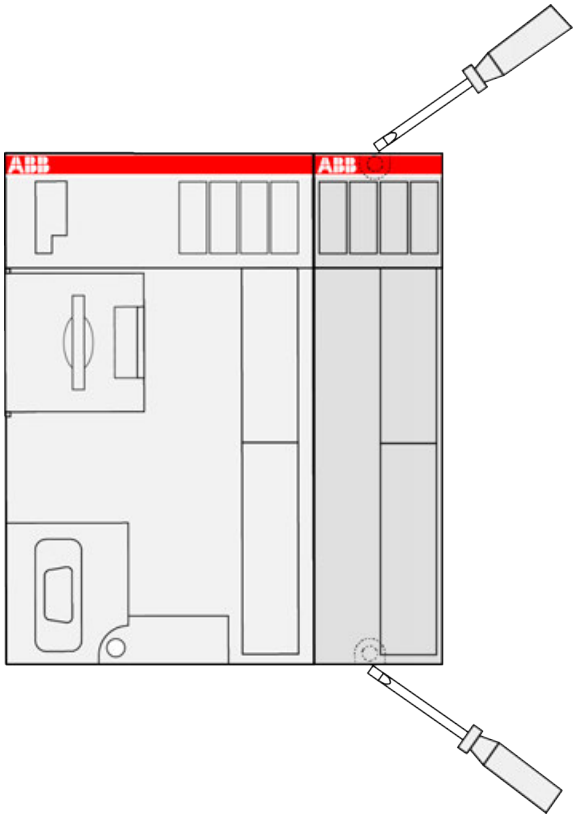
1. Snap in the TA566 at the back side of the I/O module.



2. Attach the I/O module by hand to an other module. The serial I/O bus is connected automatically.

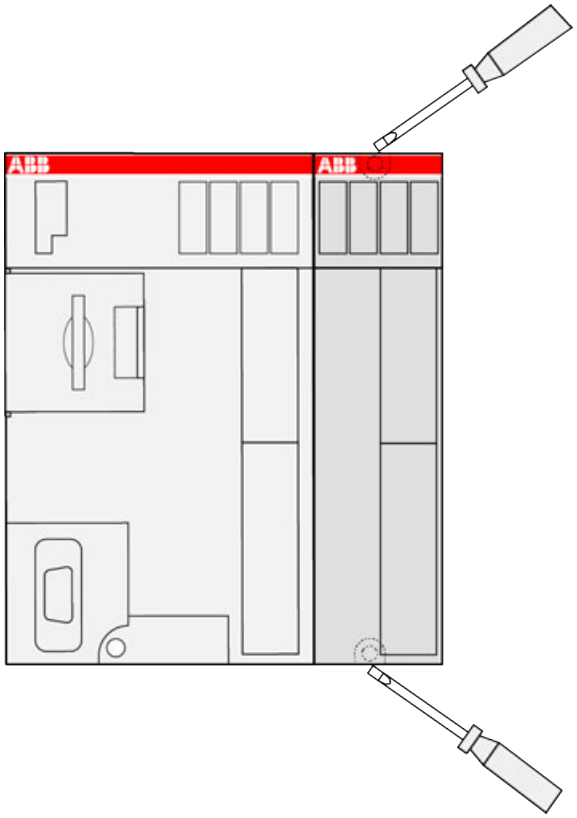


3. Fasten the I/O module with two screws (max. diameter: 4 mm) to the metal plate.

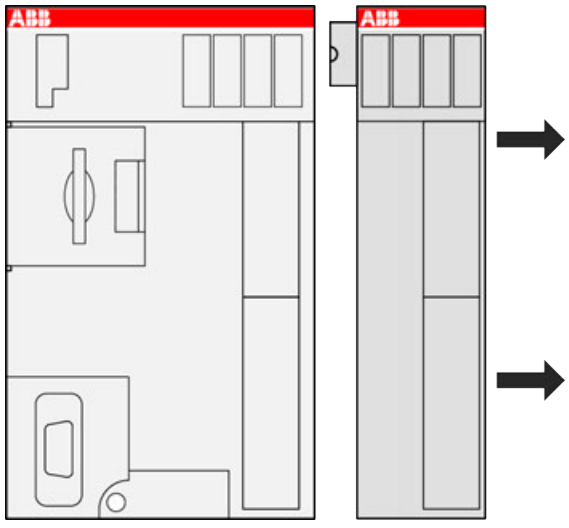


Demounting I/O modules mounted on a metal plate

1. Remove the 2 screws.



2. Remove the I/O module from the connected module by hand.



Mounting/Demounting the accessories

Additional components such as batteries, cables, etc. are required for commissioning the PLC system. Information on assembly, replacement or basic use of the orderable components can be found in the description of the respective accessory.

Chapter 1.6.3.6.5 “Handling of accessories” on page 5359

Hardware details can be found in the device specifications of the accessory.

↳ *Chapter 1.6.2.9 “Accessories” on page 5095*

1.6.3.5.4 Connection and wiring

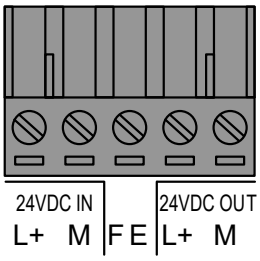
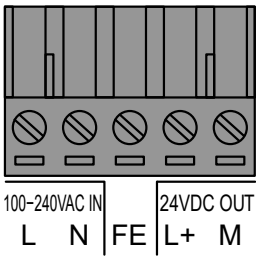
For detailed information such as technical data of your mounted devices (AC500 product family) refer to the hardware device specification of the appropriate device.

NOTICE!
Attention:
All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

Power supply

Power supply

Depending on the variant, the processor modules can be connected to the following supply voltages:

	
24 V DC	100 - 240 V AC

The connection is established via a removable 5-pin terminal block. As the terminal block is also available as a spare part (inside TA570 Spare Part Set for AC500-eCo processor modules).

The 24 V DC variant contains 2 L+ and M terminals. The L+ terminal on the left side is the input and the right side is the output. The M terminals are internally interconnected. The supply can be easily looped through to the onboard digital inputs.

CAUTION!
Risk of damaging the processor module and the connected modules!
Voltages > 35 V DC (DC variants only) or > 288 V AC (AC variants only) might damage the processor module and the connected modules.

Make sure that the supply voltage never exceeds 35 V DC / 288 V AC.

CAUTION!
Risk of damaging the processor module!
Excess currents at 24 V DC output (24 V DC processor module variant) will damage the processor module.

Use an appropriate fuse ↳ *Chapter 1.6.2.3.1.1.8 “Technical data” on page 3814* within 24 V DC input connection.

The 100-240 V AC variant contains an internal power supply with a wide-range input. It provides a 24 V DC output at the terminals L+ and M which can be used to supply the onboard digital inputs.

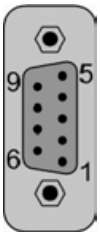
> *The voltage output at 100 V AC ... 240 V AC variants can provide 180 mA max. The output is protected against overload by a self-resetting fuse (PTC).*

Serial interface COM1

The serial interfaces COM1 and COM2 are designed according to the standard EIA RS-485. Both interfaces can be operated in RS-485 mode.

Parameter	Value
Standard of the serial interfaces	RS-485
Interface connectors	COM1: 9-pin D-sub connector (female) COM2: 5-pole connector with screw-type connection (optional)
Electrical isolation	none (with TA562) 500 V DC (with TA569-RS-ISO)
Serial interface parameters	Configurable by the software
Operating modes	Programming or data exchange
Supported protocols	Modbus or serial data exchange using special software function blocks

Table 597: Pin assignment

Serial Interface	Pin	Signal	Description
Serial Interface 	1	FE	Functional earth
	2	SGND	0 V power supply, internally connected to M terminal
	3	RxD/TxD-P	Receive/Transmit positive
	4	Reserved	Reserved, not connected
	5	SGND	0 V power supply, internally connected to M terminal
	6	+3.3 V	3.3 V power supply
	7	Reserved	Reserved, not connected
	8	RxD/TxD-N	Receive/Transmit negative
	9	Reserved	Reserved, not connected
	Shield	Cable shield	Functional earth

The serial non-isolated interface COM1 is connected to a 9-pole D-sub connector. It is configurable for RS-485 and can be used for:

- online access with Automation Builder (via RS-485 programming cable e. g. TK504 ↳ *Chapter 1.6.2.9.1.12 “TK504 - COM2 USB programming cable” on page 5143*),
- as Modbus RTU, client and server
- for ASCII serial protocols
- a CS31 bus (RS-485), as master only.

The serial RS-485 interface is not galvanically isolation.

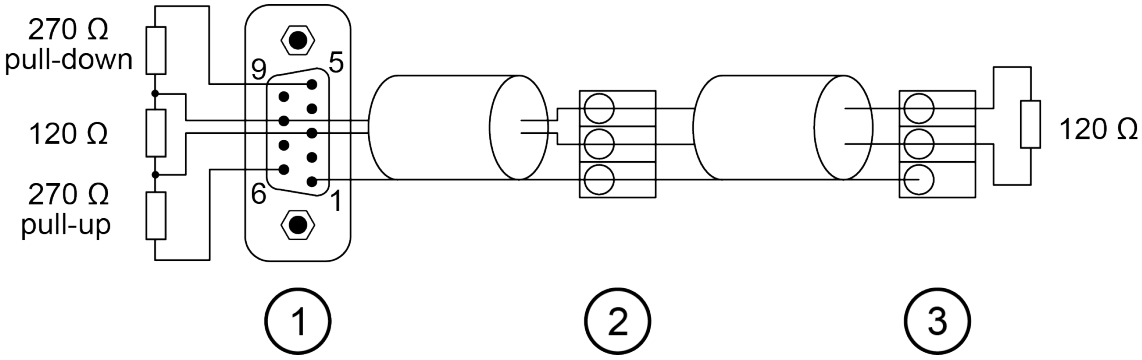
If the RS-485 bus is used, each interconnected bus line (each bus segment) must be electrically terminated. The following is necessary:

- 2 resistors of 120 Ω each at both line ends (to avoid signal reflections)
- In addition, a pull-up resistor at RxD/TxD-P and a pull-down resistor at RxD/TxD-N. These 2 resistors care for a defined high level on the bus, while there is no data exchange.

The pull-up, pull-down and termination resistors are not included inside the processor module and must be connected externally.

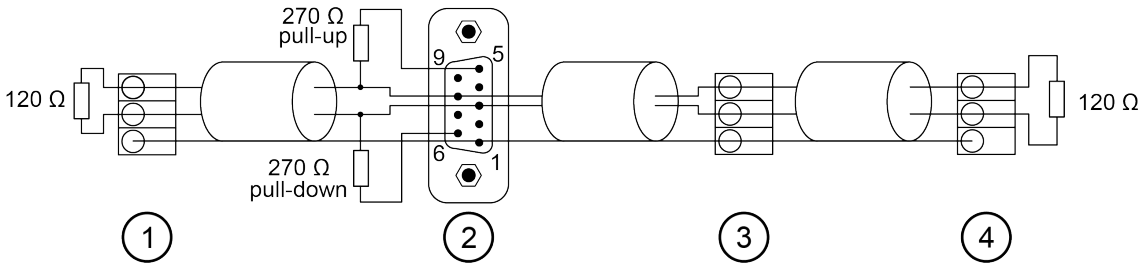
It is useful to add both the pull-up and the pull-down resistors, which only are necessary once on every bus line, at the bus master.

The following figure shows an RS-485 bus with the bus master at one line end.



1 Master at the bus line end, pull-up and pull-down activated, bus termination with 120 Ω resistors
2 Slave within the bus line
3 Slave at the bus line end, bus termination with 120 Ω resistors

If the master is located within the bus line, it does not need a terminating resistor. The pull-up and the pull-down resistors, however, are necessary:



1 Slave at the bus line end, bus termination with 120 Ω resistors
2 Master within the bus line, pull-up and pull-down activated
3 Slave within the bus line
4 Slave at the bus line end, bus termination with 120 Ω resistors

NOTICE!
Risk of EMC disturbances!
Unshielded cables may cause EMC disturbances.
Always use shielded cables and connect the shield at every device.

NOTICE!
Risk of malfunctions!
The pull-up/pull-down resistors must be used only one time within a bus line.
Use the pull-up/pull-down resistors only at 1 master.

The cable shields must be grounded. See CS31 bus *Chapter 1.6.3.5.4.4 “CS31 bus” on page 5257*.

Serial interface COM2

The serial interfaces COM1 and COM2 are designed according to the standard EIA RS-485. Both interfaces can be operated in RS-485 mode.

Parameter	Value
Standard of the serial interfaces	RS-485
Interface connectors	COM1: 9-pin D-sub connector (female) COM2: 5-pole connector with screw-type connection (optional)
Electrical isolation	none (with TA562) 500 V DC (with TA569-RS-ISO)
Serial interface parameters	Configurable by the software
Operating modes	Programming or data exchange
Supported protocols	Modbus or serial data exchange using special software function blocks

The serial interface COM2 is connected via a 5-pin terminal block and can be used for

- online access
- free protocol communication
- Modbus RTU, client and server

The serial RS-485 interface is not galvanically isolated using TA562-RS or TA562-RS-RTC.
Using TA569-RS-ISO the serial RS-485 interface has galvanic isolation.

It is not intended to use COM2 to establish a CS31 bus.

If the RS-485 bus is used, each interconnected bus line (each bus segment) must be electrically terminated. The following is necessary:

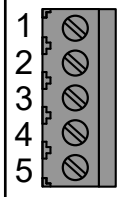
- 2 suitable resistors at both line ends (to avoid signal reflections)
- A pull-up resistor at RxD/TxD-P and a pull-down resistor at RxD/TxD-N. These 2 resistors care for a defined high level on the bus, while there is no data exchange.

The pull-up and the pull-down resistors are included inside the processor module's serial RS-485 adapter. The terminating resistor is not included inside the processor module and must be connected externally.

It is useful to activate both the pull-up and the pull-down resistors, which are only necessary once on every bus line, at the bus master. For this reason, these 2 resistors are already integrated within the COM2 interface of the processor module. They can be activated by connecting the terminals 1-2 and 3-4 of COM2.

For equipping AC500-eCo processor modules with a real-time clock and a second serial RS-485 interface COM2, use TA562-RS-RTC serial RS-485 and real-time clock adapter ↳ Chapter 1.6.3.5.5.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5296.

Table 598: Pin assignment

Serial Interface	Pin	Description
	1	Terminator P
	2	TxD/RxD-P
	3	TxD/RxD-N
	4	Terminator N
	5	Functional earth

NOTICE!
Risk of EMC disturbances!
Unshielded cables may cause EMC disturbances.

Always use shielded cables and connect the shield at every device.

NOTICE!
Risk of malfunctions!
The pull-up/pull-down resistors must be used only one time within a bus line.

Use the pull-up/pull-down resistors only at 1 master.

The ground potential of the interface COM2 is internally connected to the M terminal of the CPU power supply connector (not for TA569-RS-ISO).

The cable shields must be grounded. See CS31 bus ↳ *Chapter 1.6.3.5.4.4 “CS31 bus” on page 5257*.

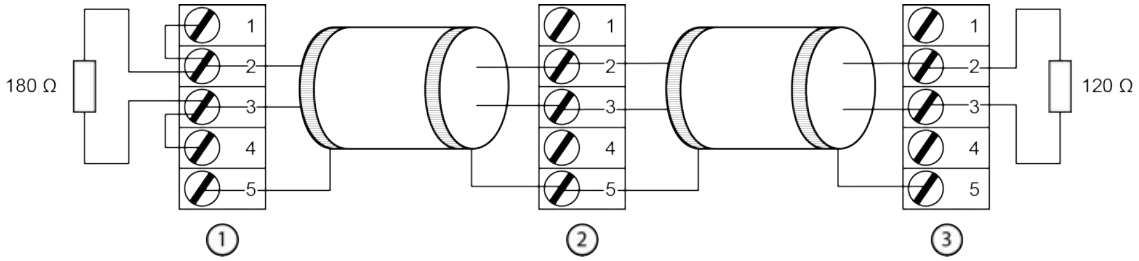
COM2 as master of RS-485 communication system

When COM2 is configured as a master in serial communication application, internal pull-up/pull-down resistors have to be activated to comply minimum 200 mV input voltage on A/B line during idle state.

COM2 as master at the bus line end

It is recommended to apply COM2 at the line end if RS-485 master is configured.

When COM2 is applied to the bus line end as a master it needs a 180 Ω terminator and pull-up/pull-downdown resistors wiring to comply with signal integrity and impedance matching. Terminator wiring and pull-up/pull-down resistors activating can be as:



1 COM2 as master at the end of bus line, pull-up and pull-down resistors are activated, bus termination with 180 Ω resistor
2 Slave within the bus line
3 Slave at the end of bus line, bus termination with 120 Ω resistor

AC500-eCo as master must always be located at the end of the bus line.

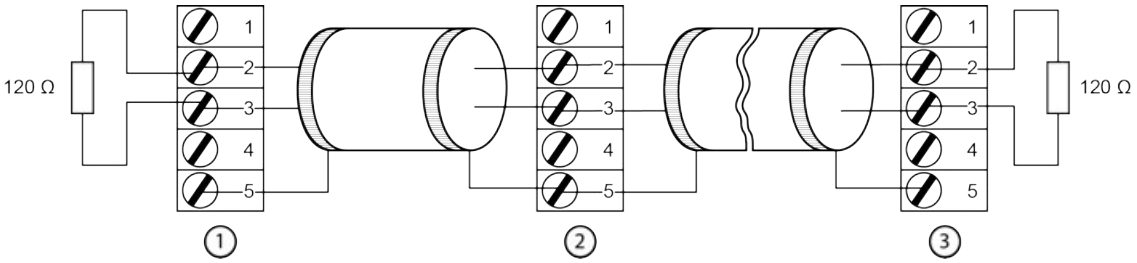
COM2 as slave of RS-485 communication system

When COM2 is configured as a slave in serial communication application, pull-up/pull-down resistors must be inactivated. Terminator wiring complies with the node position.

It does not matter wherever the master is located when COM2 is configured as slave in the line.

COM2 as slave at the bus line end

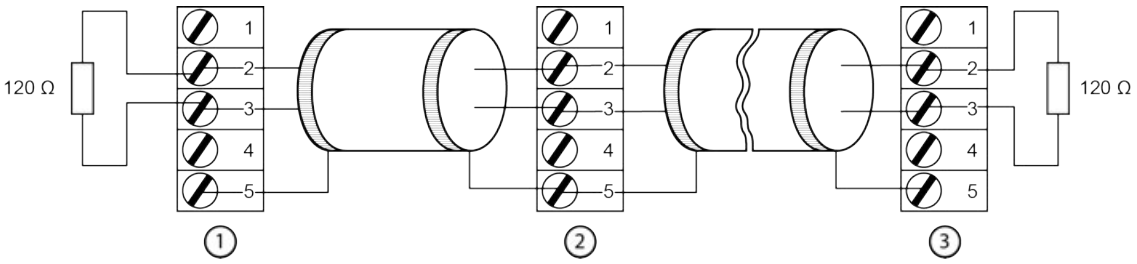
A 120 Ω 1/2 W resistor is a typical terminator to match the impedance of most of cable applied when COM2 is located at the end of bus line.



1 COM2 as slave at the end of bus line, bus termination with 120 Ω resistor, but the pull-up and pull-down termination must be inactivated
2 Slave within the bus line
3 Slave at the end of bus line, bus termination with 120 Ω resistor

COM2 as slave located within the bus line

If COM2 is configured as a slave node within the bus line, it does not need a terminator. Pull-up and pull-down resistors are not required by a slave node.



1 Slave at the end of bus line, bus termination with 120 Ω resistor
2 COM2 as slave within the bus line, pull-up and pull-down termination must be inactivated
3 Slave at the end of bus line, bus termination with 120 Ω resistor

CS31 bus

Connection

The AC500-eCo processor module can be used as a CS31 bus master. They cannot be used as a CS31 bus slave. The connection is performed via the serial interface COM1 used as a CS31 bus (see chapter Serial Interface COM1 ↯ *Chapter 1.6.2.3.1.1.3 “Connections” on page 3806*). Connection of the bus signals: pin 3 and pin 8.

Wiring

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm² (24 AWG)
Recommendation	0.5 mm² corresponds to 0.8 mm
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends
Remarks	Shielded cables with PVC core insulation and a core diameter of 0.8 mm can be used up to a length of ca. 50 m. In this case, the bus terminating resistor is ca. 100 Ω.

Wiring remarks Shielded cables with PVC core insulation and a core diameter of 0.8 mm can be used up to a length of ca. 50 m. In this case, the bus terminating resistor is ca. 100 Ω.

Bus topology A CS31 bus always contains only one bus master (CPU or Communication Module) which controls all actions on the bus. Up to 31 slaves can be connected to the bus, e. g. remote modules or slave-configured CPUs. Besides the wiring instructions shown below, the wiring and grounding instructions provided with the descriptions of the modules are valid additionally.

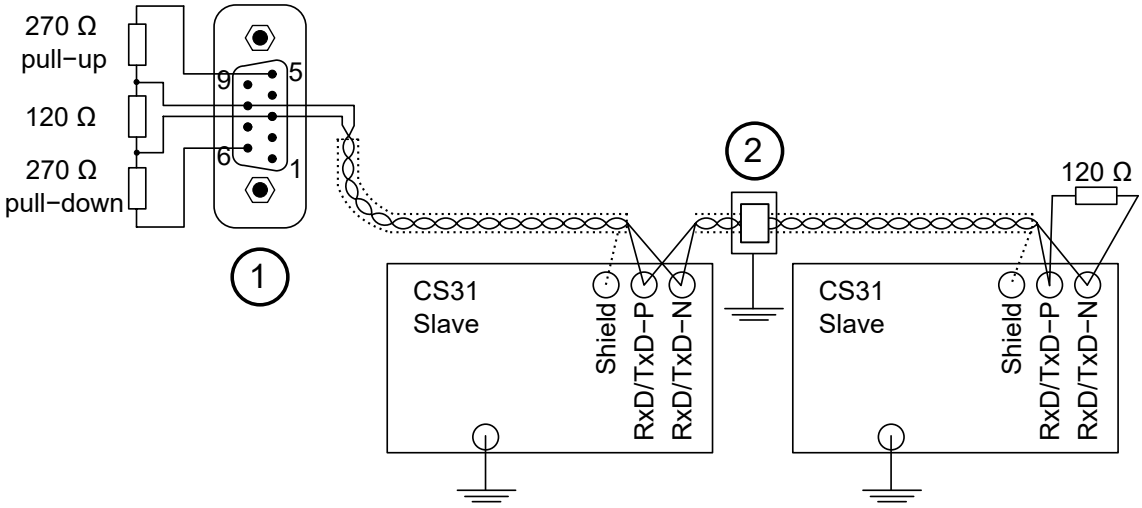


Fig. 1048: Bus topology for a CS31 bus at COM1 (Master is at the end of the bus line)

1 Master at the bus line end, pull-up and pull-down activated, bus termination with 120 Ω resistors
2 Direct grounding with clip or steel plate

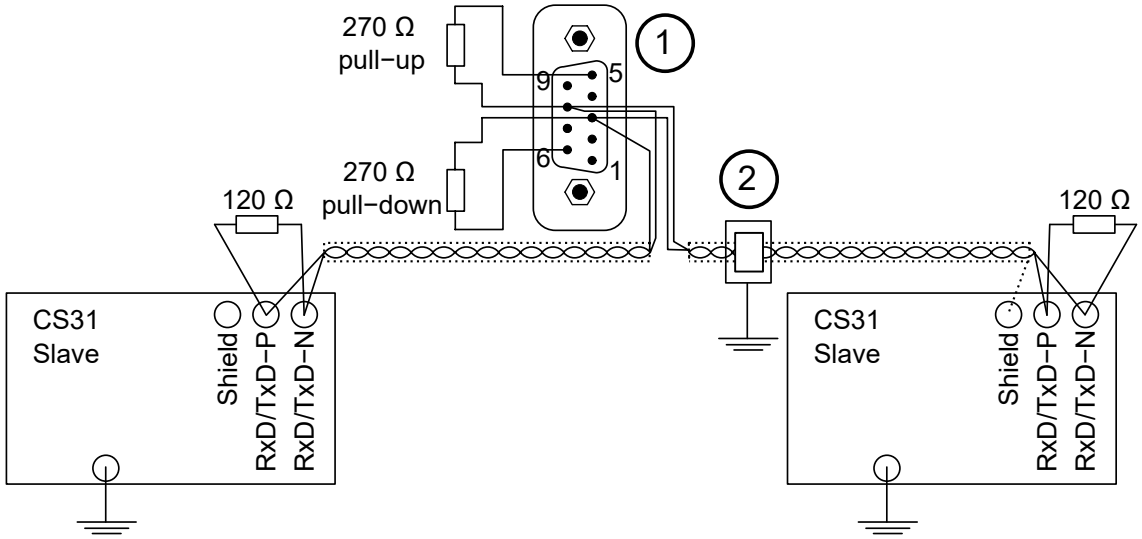


Fig. 1049: Bus topology for a CS31 bus at COM1 (Master is within the bus line)

1 Master within the bus line, pull-up and pull-down activated
2 Direct grounding with clip or steel plate

NOTICE!
Risk of malfunctions!
Spur lines are not allowed within the CS31 bus.
Loop the bus line from module to module.

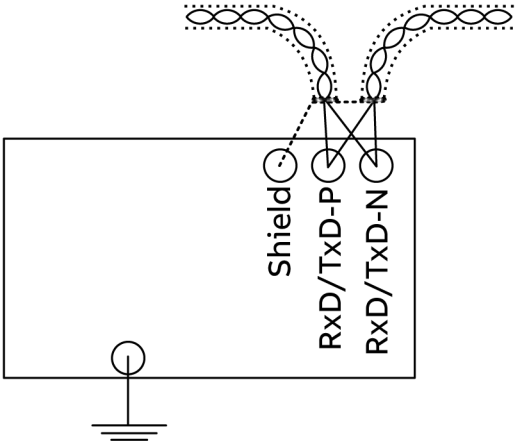


Fig. 1050: Correct

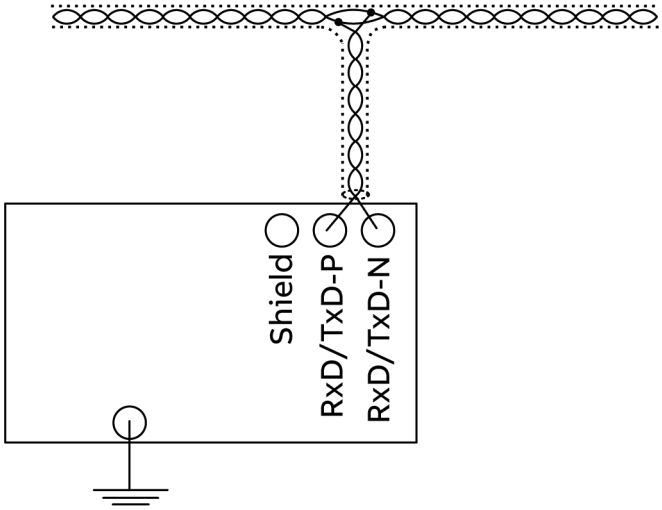


Fig. 1051: Wrong

Grounding In order to avoid disturbance, the cable shields must be grounded directly.

Case A Multiple switchgear cabinets: If it can be guaranteed that no potential differences can occur between the switchgear cabinets by means of current-carrying metal connections (grounding bars, steel constructions etc.), the direct grounding is chosen.

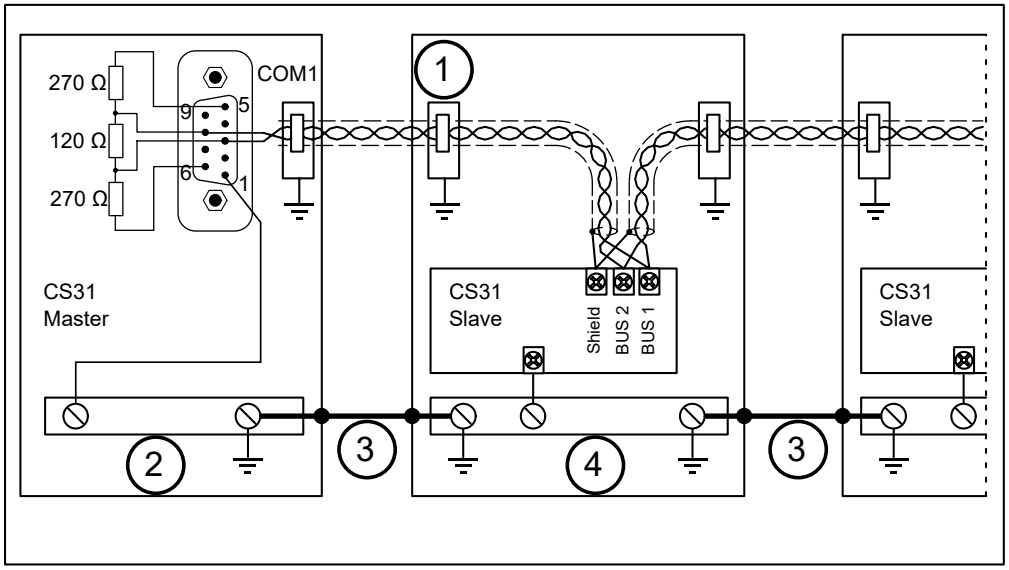


Fig. 1052: Direct grounding

1 Direct grounding with clip or steel plate
2 Ground of Cabinet 1
3 Current-carrying connection
4 Ground of Cabinet 2

Case B Multiple switchgear cabinets: If potential differences can occur between the switchgear cabinets, the capacitive grounding method is chose0n in order to avoid circulating currents on the cable shields.

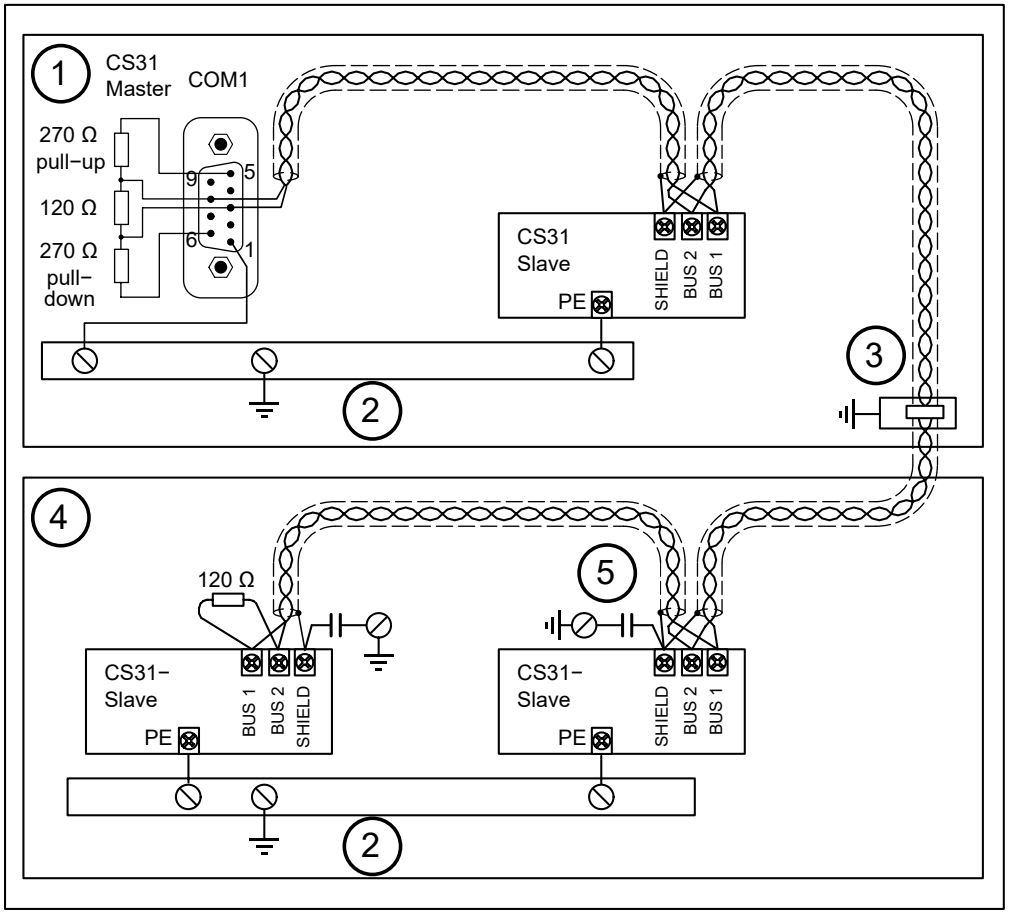


Fig. 1053: Earthing concept with several switchgear cabinets: direct grounding of cable shields when cables enter the first switchgear cabinet (containing the master), and capacitive grounding at the modules

1 Cabinet 1
2 Cabinet grounding
3 Direct grounding with clip or steel plate
4 Cabinet 2
5 Capacitive grounding with 0.1 μF X-type capacitor directly on the cabinet steel plate

Everywhere is valid: The total length of the grounding connections between the shield of the Terminal Base and the grounding bar must be as short as possible (max. 25 cm). The conductor cross section must be at least 2.5 mm².

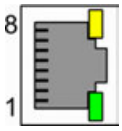
VDE 0160 requires, that the shield must be grounded directly at least once per system.

Ethernet

Ethernet is also used for PROFINET, EtherCAT and Modbus TCP connection.

Ethernet interface

The Ethernet interface is carried out via a RJ45 jack. The pin assignment of the Ethernet interface:

Interface	Pin	Description	
	1	Tx+	Transmit Data +
	2	Tx-	Transmit Data -
	3	Rx+	Receive data +
	4	NC	Not connected
	5	NC	Not connected
	6	Rx-	Receive data -
	7	NC	Not connected
	8	NC	Not connected
	Shield	Cable shield	Functional earth

The supported protocols and used Ethernet ports can be found in a separate chapter ↳ *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

Communication via Modbus TCP/IP is described in detail in a separate chapter ↳ *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

Wiring

Cable length restrictions

For the maximum possible cable lengths within an Ethernet network, various factors have to be taken into account. Twisted pair cables (TP cables) are used as transmission medium for 10 Mbit/s Ethernet (10Base-T) as well as for 100 Mbit/s (Fast) Ethernet (100Base-TX). For a transmission rate of 10 Mbit/s, cables of at least category 3 (IEA/TIA 568-A-5 Cat3) or class C (according to European standards) are allowed. For fast Ethernet with a transmission rate of 100 Mbit/s, cables of category 5 (Cat5) or class D or higher have to be used. The maximum length of a segment, which is the maximum distance between two network components, is restricted to 100 m due to the electric properties of the cable.

Furthermore, the length restriction for one collision domain has to be observed. A collision domain is the area within a network which can be affected by a possibly occurring collision (i.e. the area the collision can propagate over). This, however, only applies if the components operate in half-duplex mode since the CSMA/CD access method is only used in this mode. If the components operate in full-duplex mode, no collisions can occur. Reliable operation of the collision detection method is important, which means that it has to be able to detect possible collisions even for the smallest possible frame size of 64 bytes (512 bits). But this is only guaranteed if the first bit of the frame arrives at the most distant subscriber within the collision domain before the last bit has left the transmitting station. Furthermore, the collision must be able to propagate to both directions at the same time. Therefore, the maximum distance between two ends must not be longer than the distance corresponding to the half signal propagation time of 512 bits. Thus, the resulting maximum possible length of the collision domain is 2000 m for a transmission rate of 10 Mbit/s and 200 m for 100 Mbit/s. In addition, the bit delay times caused by the passed network components also have to be considered.

The following table shows the specified properties of the respective cable types per 100 m.

Table 599: Specified cable properties:

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Attenuation [dB / 100m]	10.7	23.2
NEXT [dB / 100m]	23	24
ACR [dB / 100m]	N/A	4
Return loss [dB / 100m]	18	10
Wave impedance [Ohms]	100	100

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Category	3 or higher	5
Class	C or higher	D or higher

TP cable

The TP cable has eight wires arranged in four pairs of twisted wires. Different color codes exist for the coding of the wires, the coding according to EIA/TIA 568, version 1, being the one most commonly used. In this code, the individual pairs are coded with blue, orange, green and brown color. One wire of a pair is unicolored and the corresponding second wire is striped, the respective color alternating with white. For shielded cables, a distinction is made between cables that have one single shield around all pairs of wires and cables that have an additional individual shield for each pair of wires. The following table shows the different color coding systems for TP cables:

Table 600: Color coding of TP cables:

Pairs	EIA/TIA 568 Version 1		EIA/TIA 568 Version 2		DIN 47100		IEC 189.2	
Pair 1	white/ blue	blue	green	red	white	brown	white	blue
Pair 2	white/ orange	orange	black	yellow	green	yellow	white	orange
Pair 3	white/ green	green	blue	orange	grey	pink	white	green
Pair 4	white/ brown	brown	brown	slate	blue	red	white	brown

Two general variants are distinguished for the pin assignment of the normally used RJ45 connectors: EIA/TIA 568 version A and version B. The wiring according to EIA/TIA 568 version B is the one most commonly used.

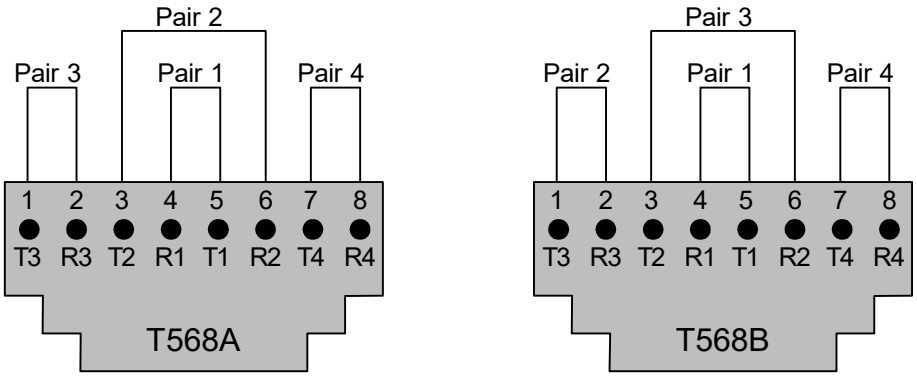


Fig. 1054: Pin assignment of RJ45 sockets

Cable types

Straight-through cable

For networks with more than two subscribers, hubs or switches have to be used additionally for distribution. These active devices already have the crossover functionality implemented which allows a direct connection of the terminal devices using straight-through cables.

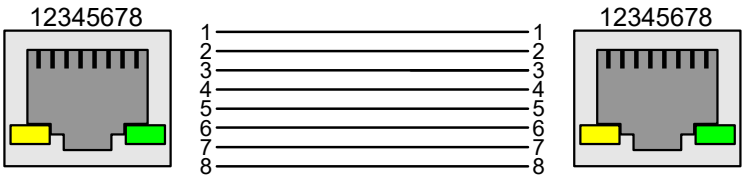


Fig. 1055: Wiring of a straight-through cable

CAUTION!
Risk of communication faults!
When using inappropriate cables, malfunctions in communication may occur.

Only use network cables of the categories 5 (Cat 5, Cat 5e, Cat 6 or Cat 7) or higher within PROFINET networks.

Modbus RTU connection details

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Available serial interfaces can work as Modbus interfaces simultaneously.

The Modbus client operating mode of an interface is set with the function block COM_MOD_MAST.

↳ *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*

Technical data

The Modbus operating mode and the interface parameters are set in the ↳ *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*.

Table 601: Description of the Modbus protocol

Parameter	Value
Supported standard	PM55x and PM56x: EIA RS-485 PM57x, PM58x and PM59x: EIA RS-232 / RS-485
Number of connection points	1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol	Modbus
Operating mode	Client/server
Address	Server only
Data transmission control	CRC16
Data transmission speed	From 300 bits/s to 187,500 bits/s

Parameter			Value
Encoding			1 start bit 8 data bits 1 parity bit, (optional) even, odd, mark or space 1 or 2 stop bits
Max. cable length for RS-485 on COM1 / COM2 for AC500 CPU			1.200 m at 19.200 baud
Max. cable length for RS-485 on COM1 / COM2 for AC500-eCo CPU			
	COM1:		
		Non-isolated:	Max. 50 m (with shielded cable)
		Isolated with TK506:	Max. 500 m at 19.200 baud (with shielded cable *)
	COM2:		
		Non-isolated with TA562:	Max. 50 m (with shielded cable)
		Isolated with TA569:	Max. 500 m at 19.200 baud (with shielded cable *)

*) 500 m cable type STP-120 Ω/AWG-20

If a processor module provides more than one serial interface, both interfaces (COM1/COM2) can be operated simultaneously as Modbus interfaces and can operate as Modbus server as well as Modbus client.

Bus topology

Point-to-point with RS-232 or bus topology with RS-485. Modbus is a master-slave protocol.

For further information on Modbus see chapter ↳ *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

1.6.3.5.5 Handling of accessories

This section only describes accessories that are frequently used for system assembly, connection and construction. A description of all additional accessories that can be used to supplement AC500 system can be found in the Hardware PLC device description.

CR2032 - Battery for real-time clock

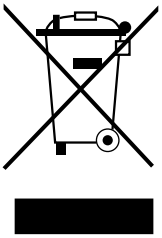
Intended purpose

A standard lithium battery (type CR2032) is used to backup the real-time clock (RTC) in the adapters TA561-RTC ↳ *Chapter 1.6.3.5.5.6 “TA561-RTC - Real-time clock adapter” on page 5284* and TA562-RS-RTC ↳ *Chapter 1.6.3.5.5.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5296* during power failures.

The CPU monitors the discharge degree of the battery. An diagnoses message is output before the battery condition becomes critical (about 2 weeks before). After the diagnosis message has appeared, the battery should be replaced as soon as possible.

Handling instruction

- The handling instructions of the battery manufacturer must be observed.
- The Material Safety Data Sheet (MSDS) of the battery manufacturer must be observed.
- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Recycle exhausted batteries meeting the environmental standards.



Transport

Transport of lithium batteries or equipment with installed lithium batteries:

- The transport and handling instructions of the battery producer must be observed.
- The transport regulations for transport of lithium batteries must be observed e.g. for transport by road or air.
- The forwarder must be informed if batteries are contained in the shipment.

Connections

Assembling and connection of the battery is described in chapters of TA561-RTC ↬ *Chapter 1.6.3.5.5.6 “TA561-RTC - Real-time clock adapter” on page 5284* and TA562-RS-RTC ↬ *Chapter 1.6.3.5.5.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5296*.

Battery lifetime

The battery lifetime is the time the battery can operate the RTC while the CPU is not powered. The typical lifetime is 300 days (at 25 °C).

As long as the CPU is powered, the battery will only be discharged by its own leakage current.

Technical data

The battery must meet die following technical data:

Parameter	Value
Battery designation	CR2032
Description	Manganese dioxide button cell, primary cell, not rechargeable
Nominal voltage	3 V DC
Capacity	230 mAh (measured with 5.6 kΩ load at 20 °C, discharging down to 2.0 V)
Typical lifetime (at 25 °C, CPU not powered)	300 days
Temperature range	≥ 0 °C ...+70 °C
Diameter	20 mm
Height	3.2 mm

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

> *The memory card has a write protect switch.*
>
> *In the position "LOCK", the memory card can only be read.*

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

> *The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.*

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

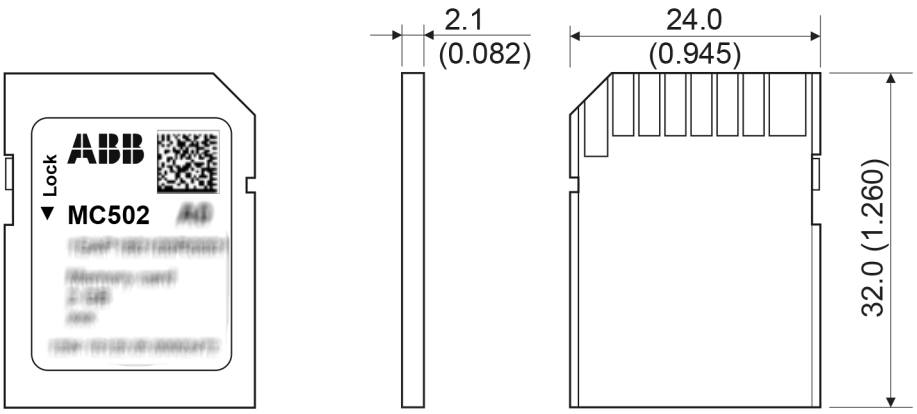
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

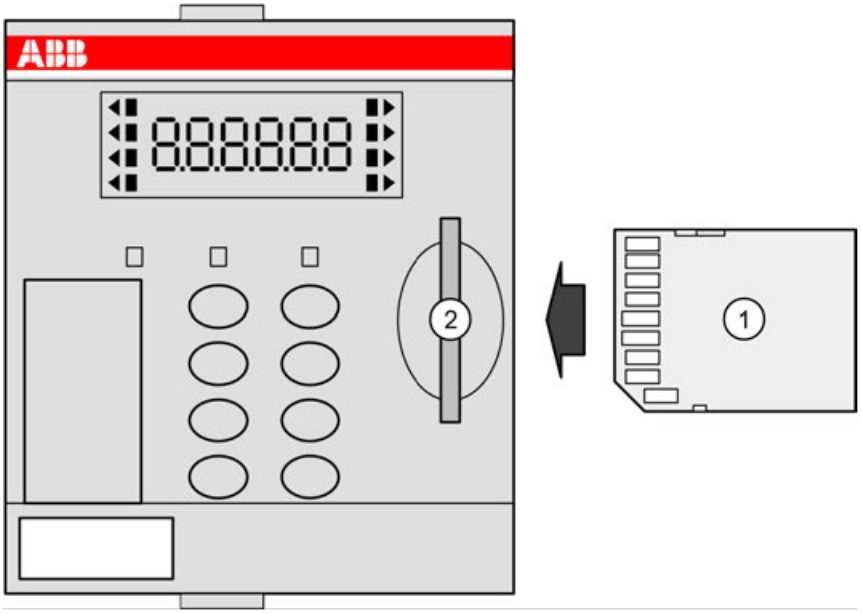


Fig. 1056: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

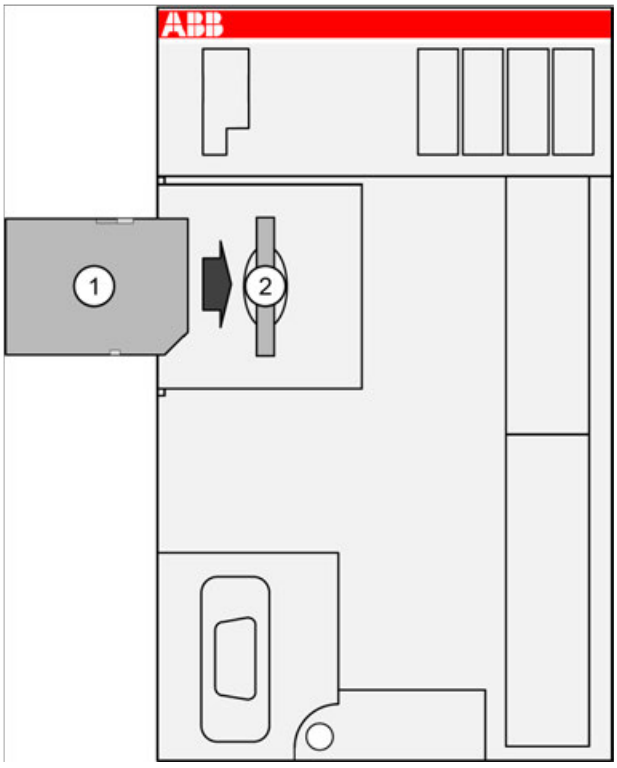


Fig. 1057: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

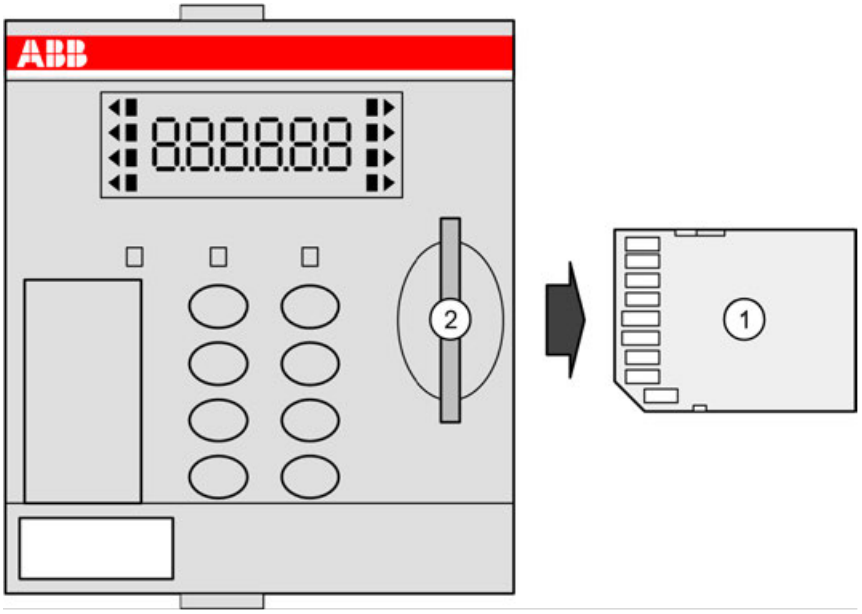


Fig. 1058: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

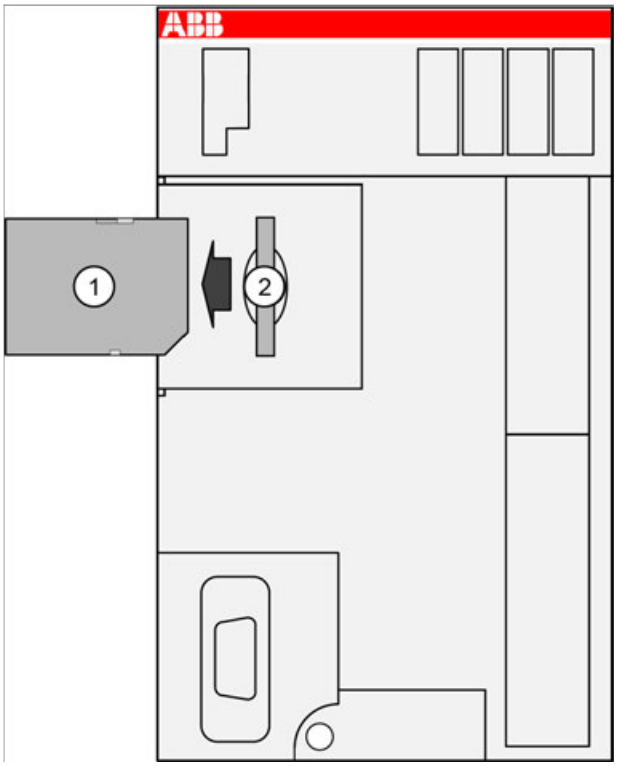


Fig. 1059: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

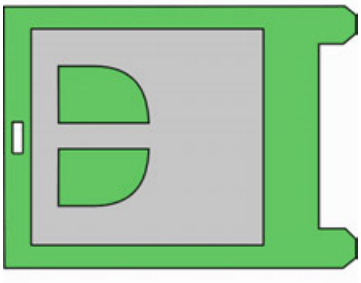
Further information on using the memory card in AC500 PLCs is provided in the chapter ↬ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC503 - Memory card adapter



Intended purpose

The MC503 memory card adapter is used for expanding processor modules PM55x-xP or PM56x-xP with a memory card slot.

A memory card MC502 or a micro memory card MC5102 with micro memory card adapter is not included in the scope of delivery and must be ordered separately.

The memory card can be used for:

- saving process data,
- saving user programs,
- upgrading the firmware.

Insert the memory card adapter

1. Make sure, that the power supply of the processor module is turned off.

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

2. Remove the option board slot cover of the processor module totally by pushing it to the left side.
3. Plug the memory card adapter to the left expansion slot of the processor module. Make sure that the 2 noses of the expansion module fit to the holes of the processor module printed circuit board.
4. Remove the bar located in the middle of the option board slot cover for memory card slot.
5. Refit the option board slot cover.

6. To insert the memory card, see MC502 ↳ *Chapter 1.6.3.5.5.2 “MC502 - Memory card” on page 5267* or MC5102 ↳ *Chapter 1.6.3.5.5.4 “MC5102 - Micro memory card with micro memory card adapter” on page 5273*.

Remove the memory card adapter

1. Make sure that the power supply of the processor module is turned off.

WARNING!
Risk of electric shock!
With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

2. Remove the option board slot cover of the processor module totally by pushing it to the left side.
3. Remove the memory card adapter out of the processor module by lifting it up with a screwdriver.
4. Refit the option board slot cover. The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo V2 processor modules). ↳ *Chapter 1.6.3.5.5.12 “TA570 - Spare part set” on page 5309*

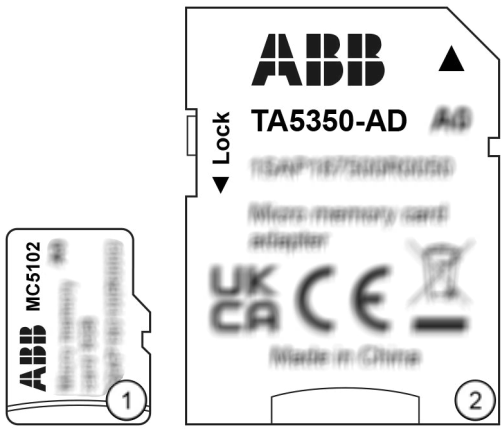
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R0100	MC503, memory card adapter for PM55x-xP or PM56x-xP	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	**AC500 V2**	**AC500-XC V2**	**AC500-eCo V2 [3])**	**AC500 V3**	**AC500-XC V3**	**AC500-eCo V3**
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

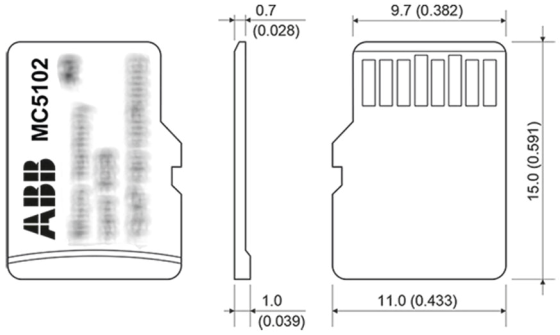
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

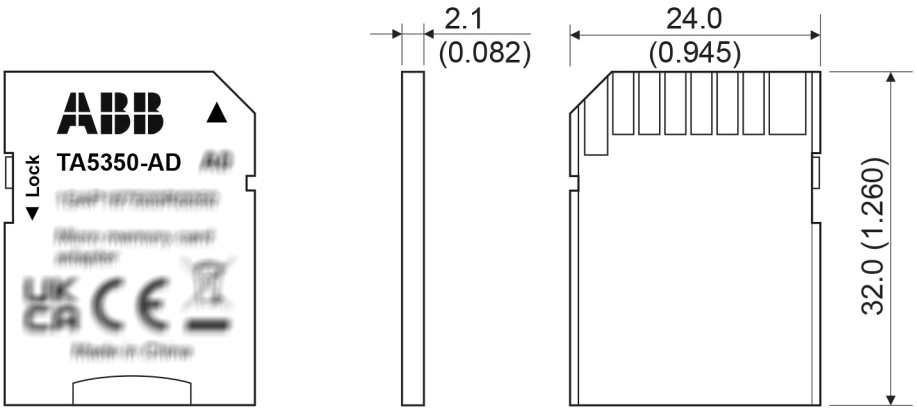
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V2 and AC500-eCo V2

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

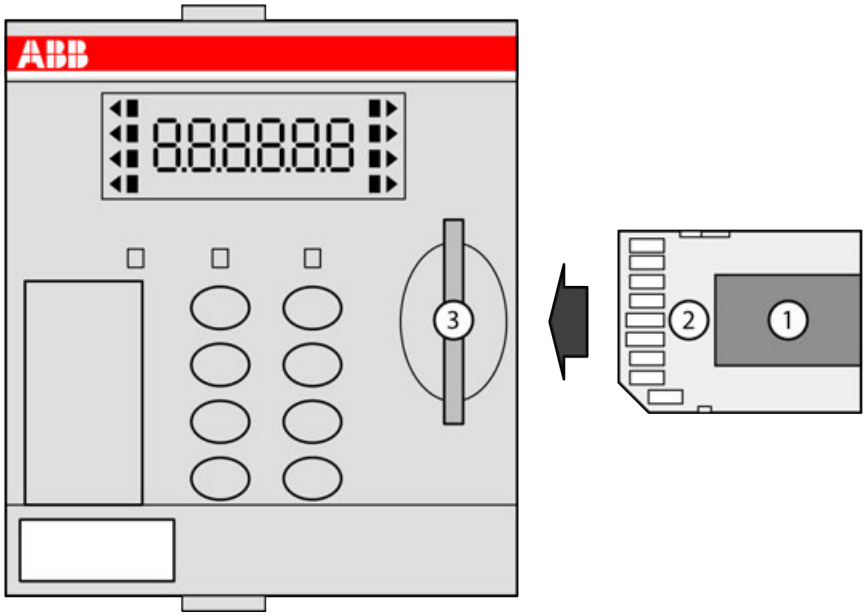


Fig. 1060: Insert micro memory card into PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

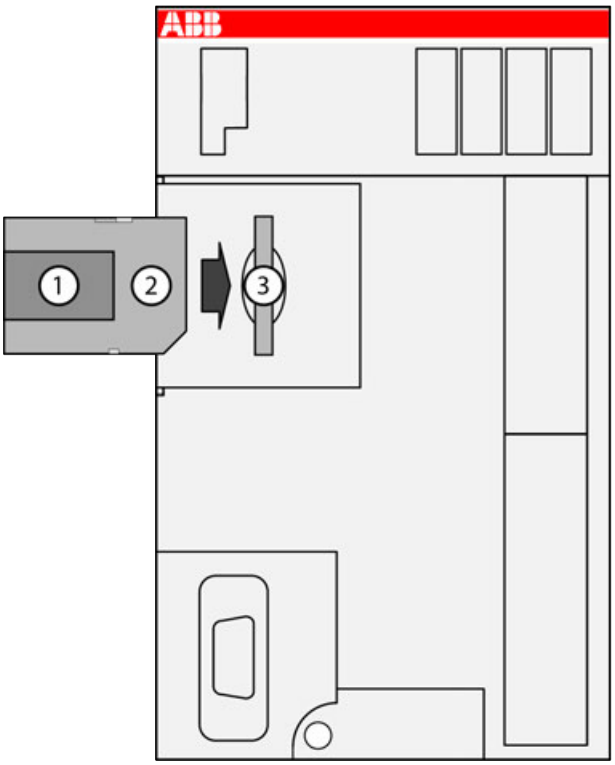


Fig. 1061: Insert micro memory card into PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Remove the micro memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

Remove the micro memory card with micro memory card adapter only when the RUN LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

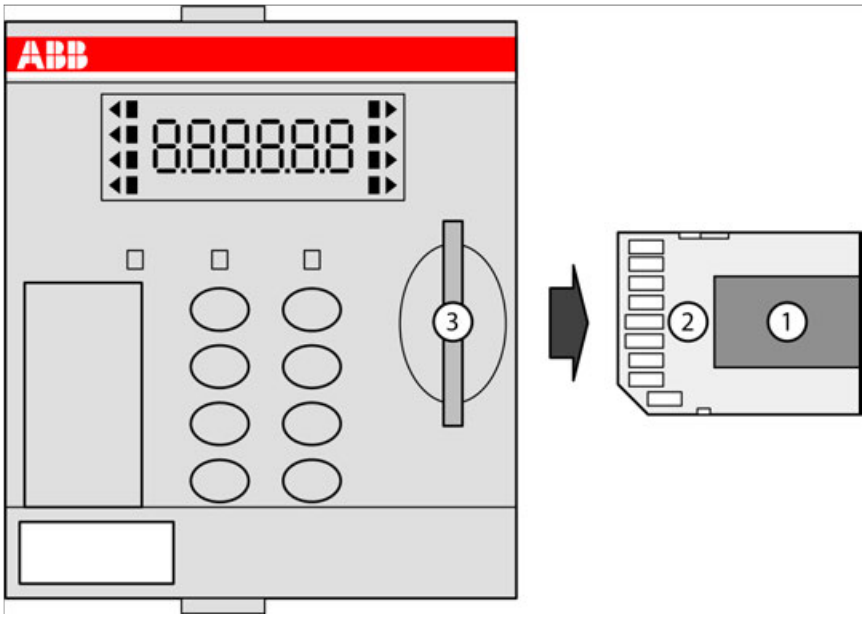


Fig. 1062: Remove micro memory card from PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

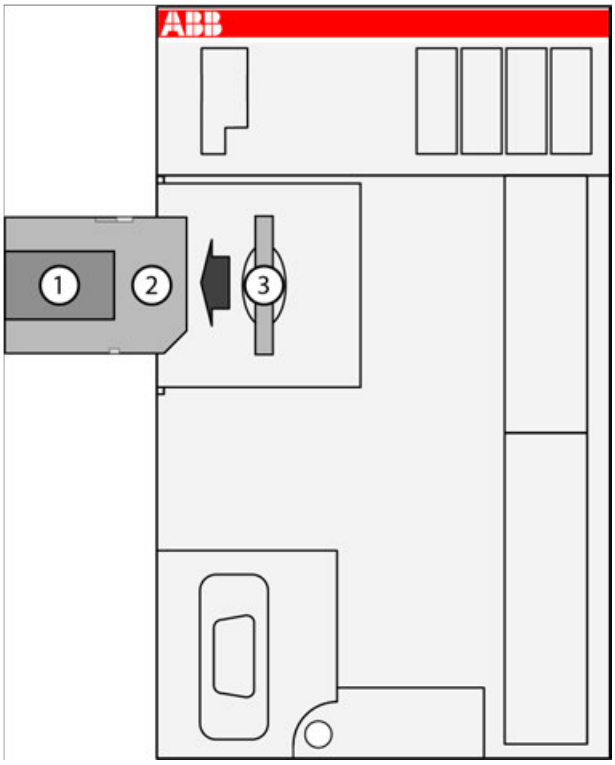


Fig. 1063: Remove micro memory card from PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		8 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	**AC500 V2**	**AC500-XC V2**	**AC500-eCo V2 [3])**	**AC500 V3**	**AC500-XC V3**	**AC500-eCo V3**
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

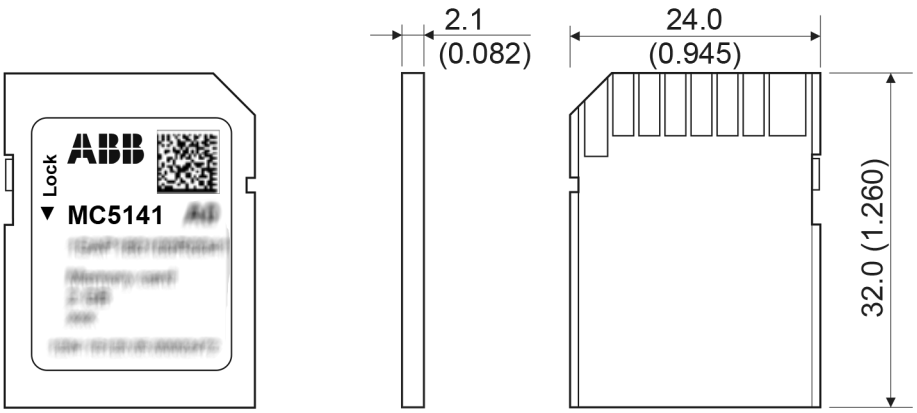
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

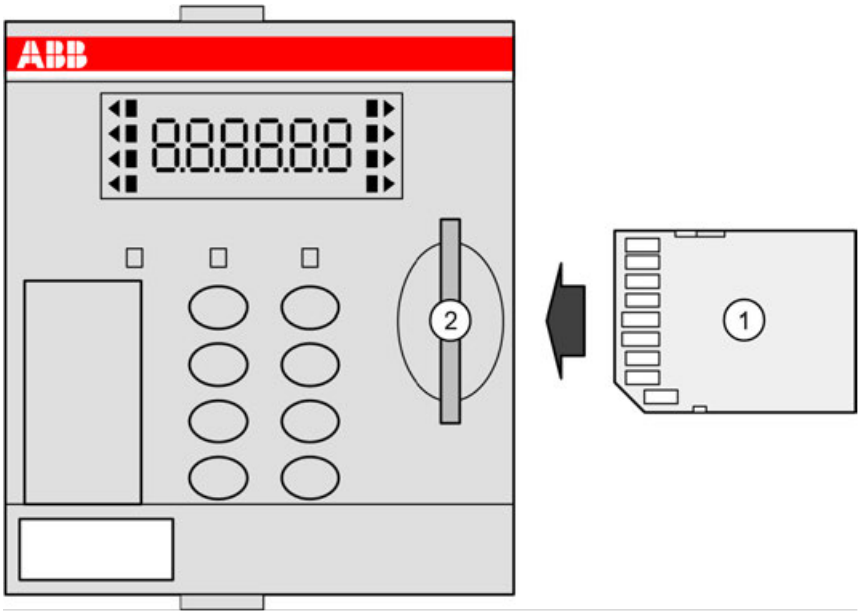


Fig. 1064: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

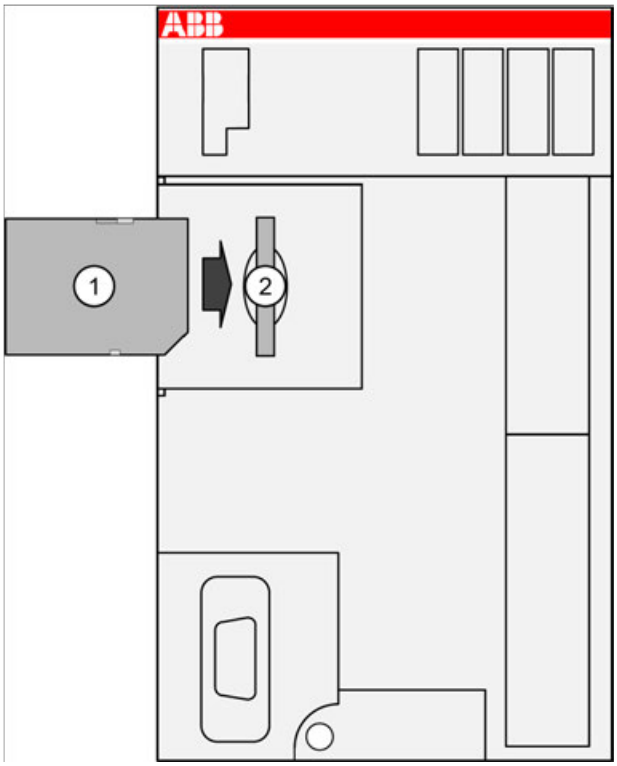


Fig. 1065: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

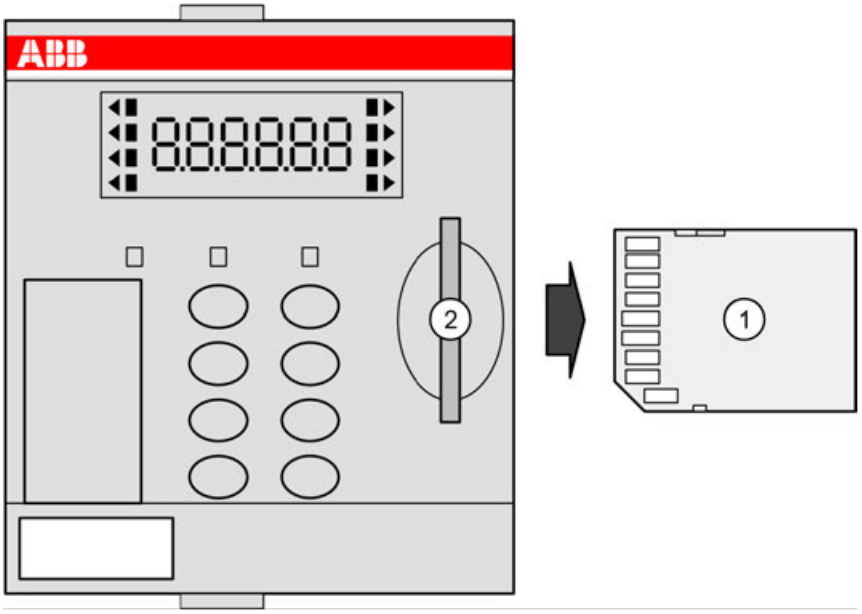


Fig. 1066: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

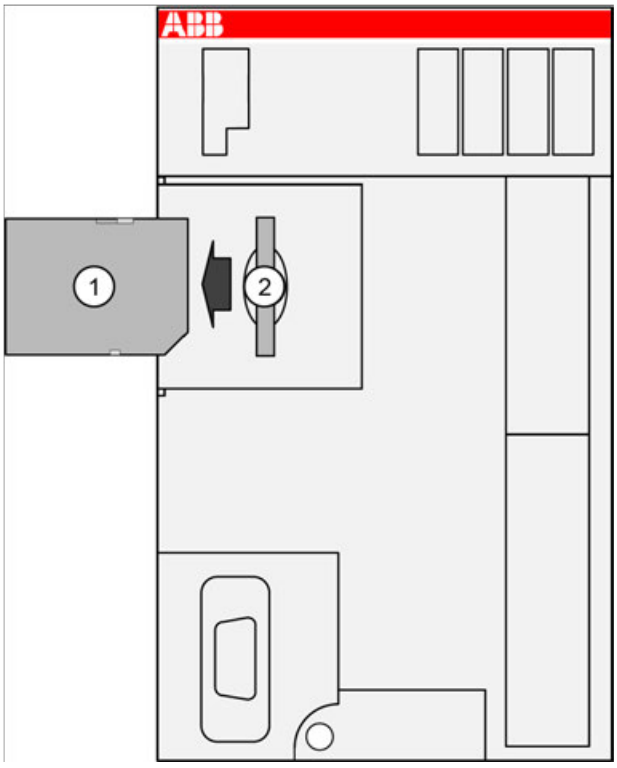


Fig. 1067: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter	Value
Memory capacity	2 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

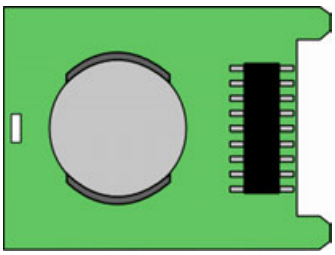
Further information on using the memory card in AC500 PLCs is provided in the chapter *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA561-RTC - Real-time clock adapter



Intended purpose

The TA561-RTC real-time clock adapter is used for equipping AC500-eCo processor modules with a real-time clock.

The real-time clock can be buffered via an optional standard lithium battery (CR2032) during power supply failures (see lithium battery for real-time clock of AC500-eCo processor modules *Chapter 1.6.3.5.5.1 “CR2032 - Battery for real-time clock” on page 5265*).

Insertion and replacement of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.3.5.5.12 “TA570 - Spare part set” on page 5309*).

Replacement of the battery

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

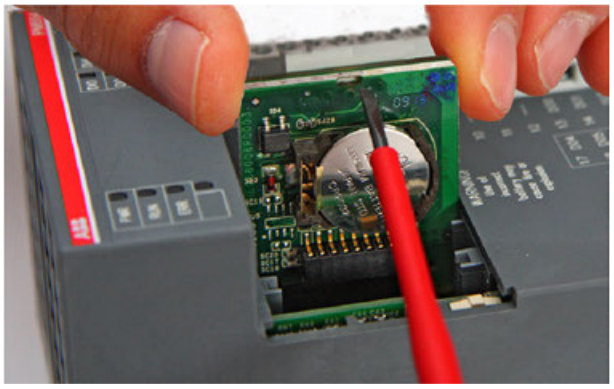
NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

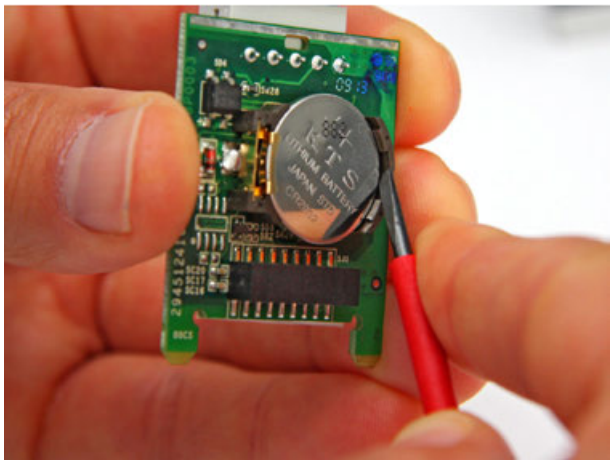
3. Remove the option board from the CPU by lifting it up with a screwdriver.

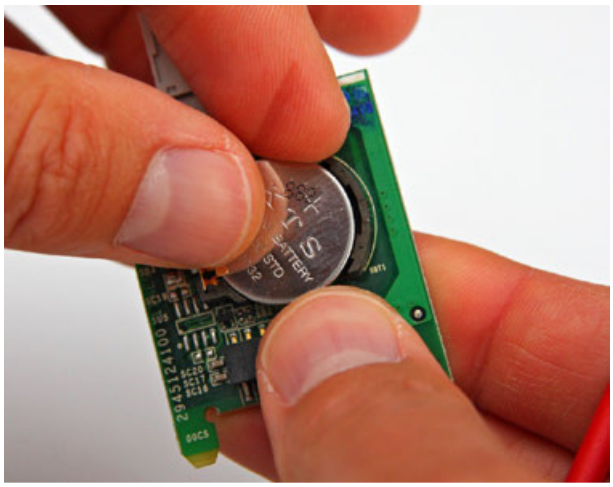


⇨

Remove memory card (if installed) / terminal block (COM2).

4. Remove the battery.





⇨

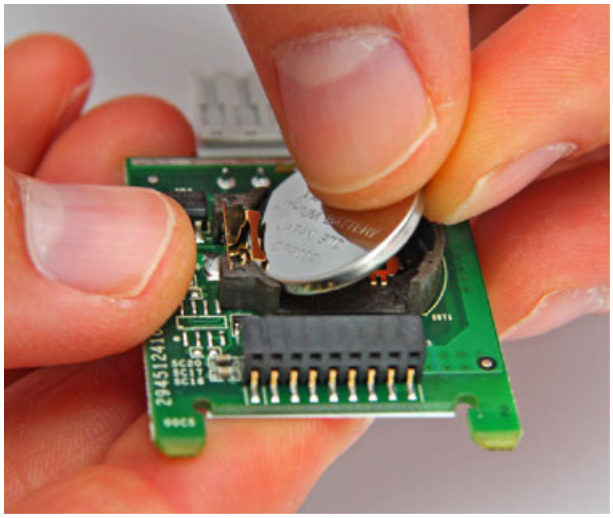
ATTENTION!

Lithium batteries must not be recharged, not be disassembled and not be disposed of in fire.

Exhausted batteries must be recycled to respect the environment.

Dispose of battery properly according to disposal procedures for lithium batteries.

5. Insert replacement battery.



⇨

ATTENTION!

*A standard battery CR2032 can be used for **TA561-RTC** and **TA562-RS-RTC**.*

*Nominal voltage: **3 V DC**.*

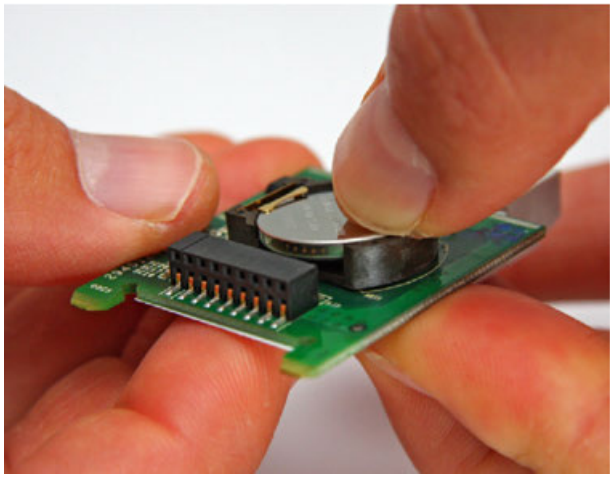
*Required capacity: **230 mAh**.*

*Required temperature range for discharge: **0 °C...+70 °C**.*

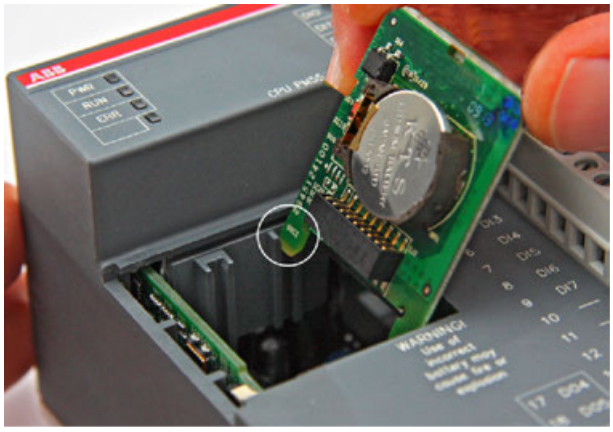
After replacement of the battery, the real-time clock (RTC) date and time must be set again by the user.

Don't use a battery older than 3 years for replacement (e.g. battery kept too long in stock).

Batteries must be stored in a dry place.



6. Insert option board into the CPU.



⇨ Insert the adapter TA56x-RTC into the slot on the right of the CPU.

> *Make sure that the 2 noses of the extension module fit to the holes of the CPU PCB.*
>
> *See white circle in figure above.*

7. Refit the option board slot cover of the CPU.

⇨

> *Remember to re-insert a memory card first if it has been removed previously.*

8. Only now the CPU can be connected to power.

> *Set the time of the real-time clock.*

Technical data

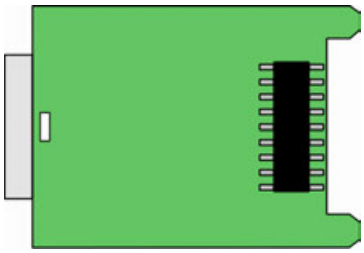
Parameter	Value
RTC accuracy (at 25 °C)	Typ. ±2 s / 24 h

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 181 400 R0001	TA561-RTC, real-time clock adapter for PM55x-xP and PM56x-xP	Active
1TNE 968 901 R3200	TA561-RTC, real-time clock adapter for PM55x-xP and PM56x-xP, lithium battery included (available in China only)	Limited

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA562-RS - Serial RS-485 adapter



Intended purpose

The serial RS-485 adapter is used for equipping AC500-eCo processor modules with a second serial interface COM2. The COM2 interface can be used for:

- online access
- free protocol communication
- Modbus RTU, client and server

> **CAUTION!**
> The serial RS-485 Interface is not galvanically isolated.

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.3.5.5.12 “TA570 - Spare part set” on page 5309*).

Removal of the option board

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

3. Remove the option board.

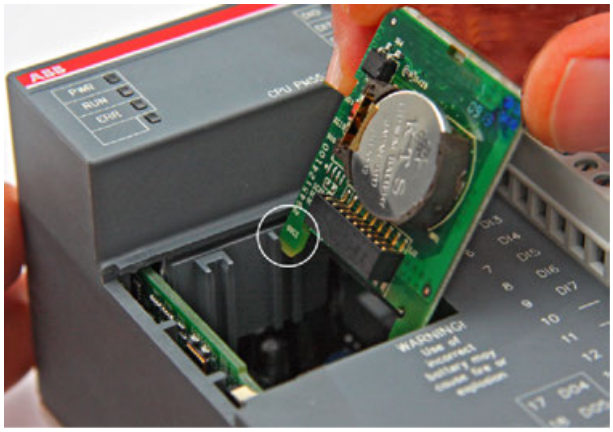
⇨

Remove memory card (if installed) / terminal block (COM2).

Remove the option board from the CPU by lifting it up with a screwdriver.

Insertion of the option board

1. Insert option board into the CPU.



⇨

Make sure that the 2 noses of the expansion module fit to the holes of the CPU PCB.

See white circle in figure above.

2. Refit the option board slot cover of the CPU.

⇨

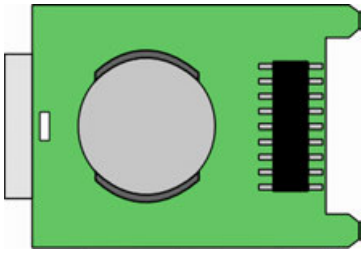
Remember to re-insert a memory card first if it has been removed previously.

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R4300	TA562-RS, serial RS-485 adapter for PM55x/PM56x	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA562-RS-RTC - Serial RS-485 adapter with real-time clock



Intended purpose

The TA562-RS-RTC serial RS-485 adapter with real-time clock is used for equipping AC500-eCo processor modules with a real-time clock and a second serial RS-485 interface COM2.

The real-time clock can be buffered via an optional standard lithium battery (CR2032) during power supply failures (see lithium battery for real-time clock of AC500-eCo processor modules ↳ *Chapter 1.6.3.5.5.1 “CR2032 - Battery for real-time clock” on page 5265*).

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

- Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
- Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules ↳ *Chapter 1.6.3.5.5.12 “TA570 - Spare part set” on page 5309*).

Replacement of the battery

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

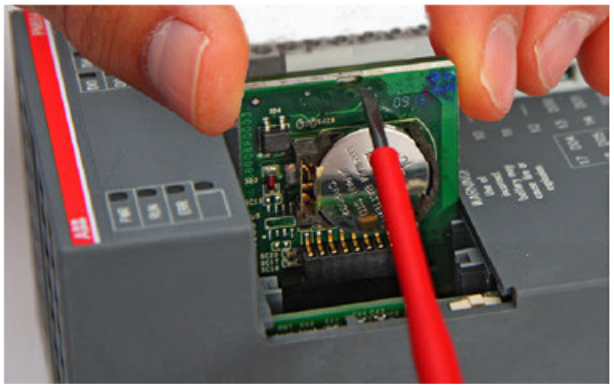
NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

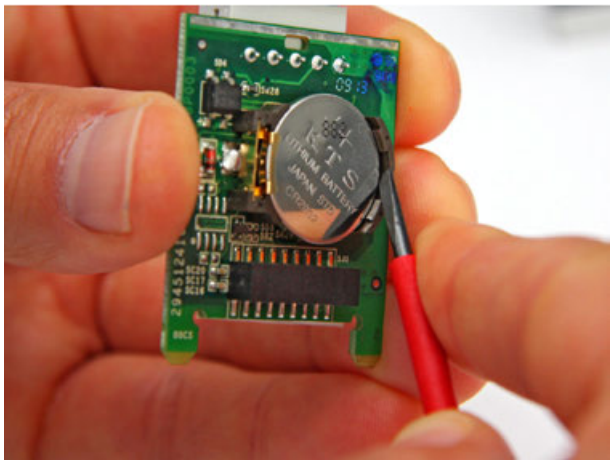
3. Remove the option board from the CPU by lifting it up with a screwdriver.

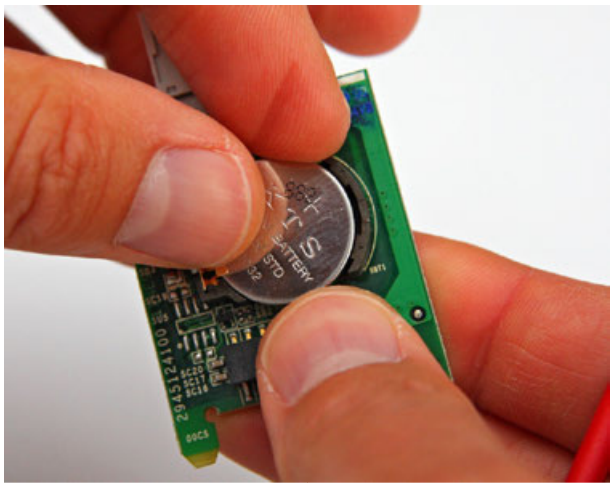


⇨

Remove memory card (if installed) / terminal block (COM2).

4. Remove the battery.





⇨

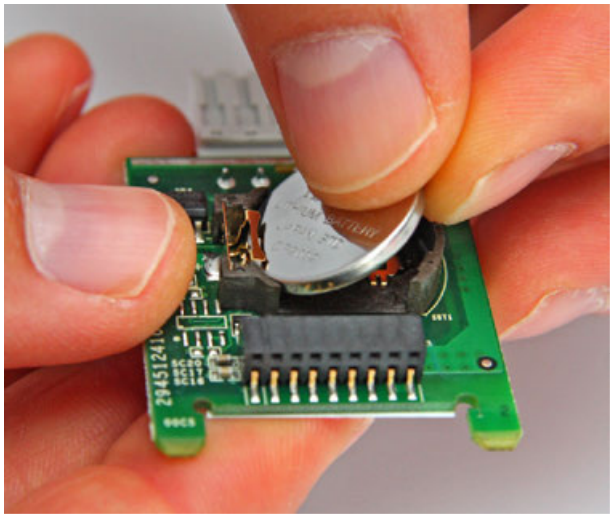
ATTENTION!

Lithium batteries must not be recharged, not be disassembled and not be disposed of in fire.

Exhausted batteries must be recycled to respect the environment.

Dispose of battery properly according to disposal procedures for lithium batteries.

5. Insert replacement battery.



⇨

ATTENTION!

*A standard battery CR2032 can be used for **TA561-RTC** and **TA562-RS-RTC**.*

*Nominal voltage: **3 V DC**.*

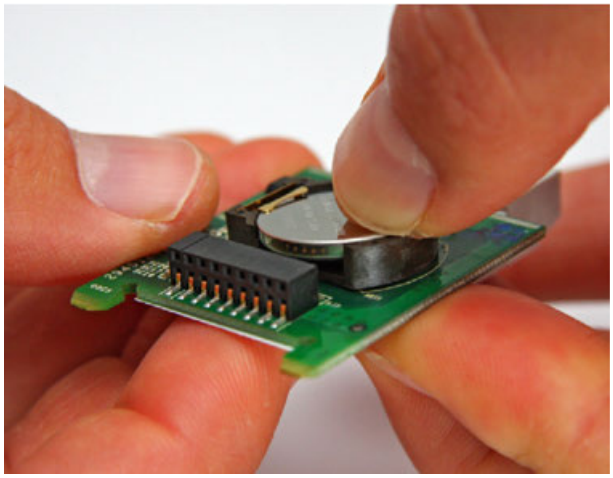
*Required capacity: **230 mAh**.*

*Required temperature range for discharge: **0 °C...+70 °C**.*

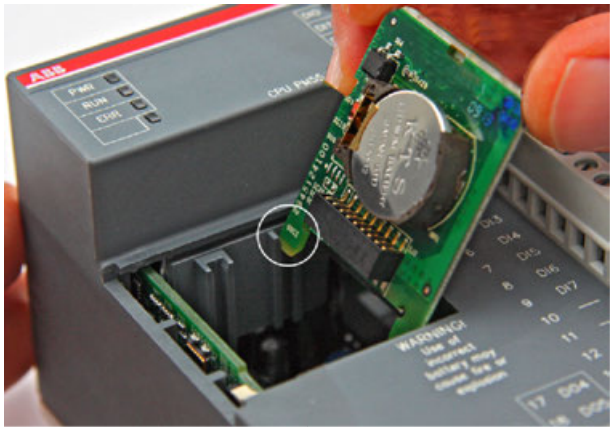
After replacement of the battery, the real-time clock (RTC) date and time must be set again by the user.

Don't use a battery older than 3 years for replacement (e.g. battery kept too long in stock).

Batteries must be stored in a dry place.



6. Insert option board into the CPU.



⇨ Insert the adapter TA56x-RTC into the slot on the right of the CPU.

> *Make sure that the 2 noses of the extension module fit to the holes of the CPU PCB.*
>
> *See white circle in figure above.*

7. Refit the option board slot cover of the CPU.

⇨

> *Remember to re-insert a memory card first if it has been removed previously.*

8. Only now the CPU can be connected to power.

> *Set the time of the real-time clock.*

Technical data

Parameter	Value
RTC accuracy (at 25 °C)	Typ. ±2 s / 24 h

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 181 500 R0001	TA562-RS-RTC, serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP	Active
1TNE 968 901 R5210	TA562-RS-RTC, serial RS-485 adapter with real-time clock for PM55x-xP and PM56x-xP, lithium battery included (available in China only)	Limited

> **) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA563-TA565 - Terminal blocks

> *These terminal blocks must only be used with AC500-eCo I/O modules and AC500-eCo processor modules.*

Intended purpose

The TA563-TA565 terminal blocks are used to connect process signals and process voltages to AC500-eCo I/O modules and AC500-eCo processor modules (with -P extension inside their type designator only). 3 different kind of terminal blocks are available:

- Screw terminals with cable insertion on the side
- Screw terminals with cable insertion on the front
- Spring terminals with cable insertion on the front

Of each kind, 2 sizes are available:

- Terminals with 9 pins
- Terminals with 11 pins.

There are 2 compatible variants of each kind and size.

WARNING!

For screw terminals only: Danger of death by electric shock!

The IP 20 protection degree is only provided if all terminal screws are tightened.

Tighten all screws of unused load terminals of relay outputs if voltages > 24 V are connected to the relay group.

Technical data

Table 602: Screw-type terminals (TA563/TA564)

Parameter		Value
Type		Front terminal or side terminal (depending on model)
Conductor cross section		
	Solid	0.5 mm^2 to 2.5 mm^2
	Flexible	0.5 mm^2 to 2.5 mm^2
Stripped conductor end		
	TA563	8 mm
	TA564	10 mm
Width of the screwdriver		3.5 mm
Fastening torque		0.4 Nm - 0.5 Nm
Degree of protection		IP 20 (if all terminal screws are tightened)
Conductor cross section flexible, with ferrule with/without plastic sleeve		Min. 0.25 mm^2 Max. 1.5 mm^2

Table 603: Spring terminals (TA565)

Parameter		Value
Type		Front terminal
Conductor cross section		
	Solid	0.5 mm^2 to 2.5 mm^2
	Flexible	0.5 mm^2 to 2.5 mm^2
Stripped conductor end		10 mm
Degree of protection		IP 20
Conductor cross section flexible, with ferrule with/without plastic sleeve		Min. 0.25 mm^2 Max. 1.5 mm^2

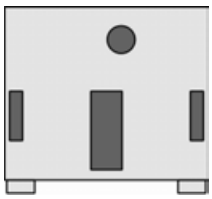
Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3101	Terminal Block TA563-9, 9-pole, screw front, cable side, 6 pieces per unit	Active
1TNE 968 901 R3102	Terminal Block TA563-11, 11-pole, screw front, cable side, 6 pieces per unit	Active

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3103	Terminal Block TA564-9, 9-pole, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3104	Terminal Block TA564-11, 11-pole, screw front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3105	Terminal Block TA565-9, 9-pole, spring front, cable front, 6 pieces per unit	Active
1TNE 968 901 R3106	Terminal Block TA565-11, 11-pole, spring front, cable front, 6 pieces per unit	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA566 - Wall mounting accessory



Intended purpose

The TA566 wall mounting accessory is used for mounting S500-eCo I/O modules and AC500-eCo processor modules without DIN rail.

Handling instruction

The TA566 is snapped into the back side of the device's housing *Chapter 1.6.3.5.3.2 “Mounting and demounting of S500-eCo I/O modules” on page 5245*.

Technical data

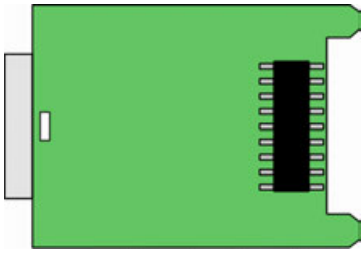
Parameter	Value
Weight	5 g
Dimensions	29 mm x 28 mm x 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3107	TA566, wall mounting accessory, 100 pieces	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA569-RS-ISO - Serial RS-485 isolated adapter



Intended purpose

The TA569-RS-ISO serial RS-485 isolated adapter is used for equipping AC500-eCo processor modules with a second serial interface COM2. The COM2 interface can be used for:

- online access
- free protocol communication
- Modbus RTU, client and server

The serial interface is isolated.

Insertion/ Removal of the adapter

WARNING!

Risk of electric shock!

With an opened option board slot cover, energized parts of the processor module could be touched.

– Always turn off and disconnect the power supply for the processor module before you open the option board slot cover.
– Make sure that the option board slot cover is closed before reconnecting the processor module to the power supply.

The option board slot cover is available as a spare part (see TA570 spare part set for AC500-eCo processor modules *Chapter 1.6.3.5.5.12 “TA570 - Spare part set” on page 5309*).

Removal of the option board

WARNING!

Risk of death by electric shock!

Hazardous voltages can be present at the terminals of the module.

Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.

1. Switch off power supply of the system and verify that the CPU is powerless.

⇨ LEDs (PWR, RUN, ERR) must be off.

2. Remove the option board slot cover.

⇨ Remove the option board slot cover of the CPU totally by pushing it to the outer side.

NOTICE!

Avoidance of electrostatic charging

PLC devices and equipment are sensitive to electrostatic discharge, which can cause internal damage and affect normal operation. Observe the following rules when handling the system:

- Touch a grounded object to discharge potential static.
- Wear an approved grounding wrist strap.
- Do not touch connectors or pins on component boards.
- Do not touch circuit components inside the equipment.
- If available, use a static-safe workstation.
- When not in use, store the equipment in appropriate static-safe packaging.

3. Remove the option board.

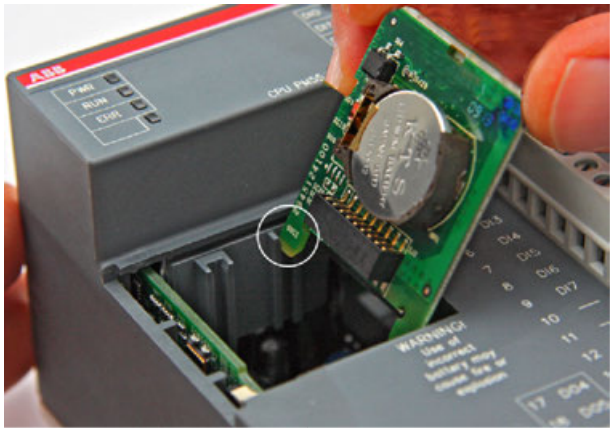
⇨

Remove memory card (if installed) / terminal block (COM2).

Remove the option board from the CPU by lifting it up with a screwdriver.

Insertion of the option board

1. Insert option board into the CPU.



⇨

Make sure that the 2 noses of the expansion module fit to the holes of the CPU PCB.

See white circle in figure above.

2. Refit the option board slot cover of the CPU.

⇨

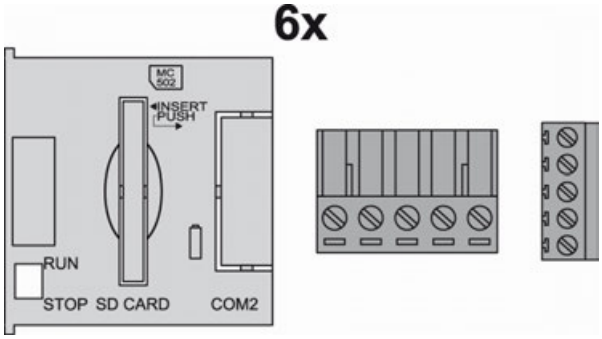
Remember to re-insert a memory card first if it has been removed previously.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 186 400 R0001	TA569-RS-ISO, serial RS-485 isolated adapter for PM55x/ PM56x	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA570 - Spare part set



Intended purpose

The TA570 spare part set is used to replace lost or damaged parts of AC500-eCo processor modules. It contains the following parts:

- Option board slot cover
- Terminal block for power supply
- Terminal block for serial RS-485 adapter

Every spare is included 6x inside TA570.

Technical data

Table 604: Option board slot cover

Parameter	Value
Weight	5 g
Dimensions	40 mm x 40 mm x 3 mm

Table 605: Terminal block for power supply

Parameter		Value
Type		Screw clamp plug, wire connection from front
Usage		For AC500-eCo processor modules
Conductor cross section		
	Solid	0.2 mm^2...2.5 mm^2
	Flexible (with wire-end ferrule only)	0.2 mm^2...2.5 mm^2
Stripped conductor end		7 mm...8 mm
Fastening torque		0.5 Nm
Degree of protection		IP20

Parameter	Value
Dimensions	25.4 mm x 17.4 mm x 15.1 mm
Weight	5 g

Table 606: Terminal block for serial RS-485 adapter

Parameter		Value
Type		Screw clamp plug, wire connection from side
Usage for		*Chapter 1.6.3.5.5.7 “TA562-RS - Serial RS-485 adapter ” on page 5291* *Chapter 1.6.3.5.5.11 “TA569-RS-ISO - Serial RS-485 isolated adapter” on page 5305* *Chapter 1.6.3.5.5.8 “TA562-RS-RTC - Serial RS-485 adapter with real-time clock” on page 5296*
Conductor cross section		
	Solid	0.14 mm^2...1.5 mm^2
	Flexible (with wire-end ferrule only)	0.14 mm^2...1.5 mm^2
Stripped conductor end		7 mm
Fastening torque		0.4 Nm
Degree of protection		IP20
Dimensions		19.05 mm x 8.7 mm x 19.1 mm
Weight		5 g

Ordering data

Part no.	Description	Product life cycle phase *)
1TNE 968 901 R3203	TA570, spare part set for AC500-eCo processor modules, 3x6 pieces	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

CP-E - Economic range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

- Wide-range input voltage
- Mounting on DIN rail
- High efficiency of up to 90 %
- Low power dissipation and low heating
- Wide ambient temperature range from -40 °C...+70 °C
- No-load-proof, overload-proof, continuous short-circuit-proof
- Power factor correction (depending on the type)
- Approved in accordance with all relevant international standards

Table 607: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR427030R0000	CP-E 24/0.75	100-240 V AC or 120-370 V DC	24 V DC, 0.75 A	-	22.5
1SVR427031R0000	CP-E 24/1.25	100-240 V AC or 90-375 V DC	24 V DC, 1.25 A	-	40.5

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR427032R0000	CP-E 24/2.5	100-240 V AC or 90-375 V DC	24 V DC, 2.5 A	-	40.5
1SVR427034R0000	CP-E 24/5.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 5 A	-	63.2
1SVR427035R0000	CP-E 24/10.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 10 A	-	83
1SVR427036R0000	CP-E 24/20.0	115-230 V AC or 120-370 V DC	24 V DC, 20 A	-	175

CP-C.1 - High performance range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

The CP-C.1 power supplies are ABB's high performance and most advanced range. With excellent efficiency, high reliability and innovative functionality it is prepared for the most demanding industrial applications. These power supplies have a 50 % integrated power reserve and operate at an efficiency of up to 94 %. They are equipped with overheat protection and active power factor correction. Combinded with a broad AC and DC input range and extensive worldwide approvals the CP-C.1 power supplies are the preferred choice for professional DC applications.

- Typical efficiency of up to 94 %
- Power reserve design delivers up to 150 % of the nominal output current
- Signaling outputs for DC OK and power reserve mode
- High power density leads to very compact and small devices
- No-load-proof, overload-proof, continuous short-circuit-proof
- Active power factor correction (PFC)

Table 608: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR360563R1001	CP-C.1 24/5.0	110-240 V AC or 90-300 V DC	24 V DC, 5 A	+50 %	40
1SVR360663R1001	CP-C.1 24/10.0	110-240 V AC or 90-300 V DC	24 V DC, 10 A	+50 %	60
1SVR360763R1001	CP-C.1 24/20.0	110-240 V AC or 90-300 V DC	24 V DC, 20 A	+30 %	82

1.6.3.6 AC500 (Standard)

1.6.3.6.1 System data AC500

Environmental conditions

Table 609: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
120 V AC		
	Voltage	120 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
230 V AC		
	Voltage	230 V AC (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
120 V AC...240 V AC wide-range supply		
	Voltage	120 V...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply, according to EN 61131-2		

Parameter		Value
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2
	AC supply	Interruption < 0.5 periods, time between 2 interruptions > 1 s

NOTICE!
Exceeding the maximum power supply voltage for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

NOTICE!
Improper voltage level or frequency range which cause damage of AC inputs:
– AC voltage above 264 V
– Frenquency below 47 Hz or above 62.4 Hz

NOTICE!
Improper connection leads cause overtemperature on terminals.
PLC modules may be destroyed by using wrong cable type, wire size and cable temperature classification.

Parameter		Value
Temperature		
	Operating	0 °C...+60 °C: Horizontal mounting of modules. 0 °C...+40 °C: Vertical mounting of modules. Output load reduced to 50 % per group.
	Storage	-40 °C...+70 °C
	Transport	-40 °C...+70 °C
Humidity		Max. 95 %, without condensation
Air pressure		
	Operating	> 800 hPa / < 2000 m
	Storage	> 660 hPa / < 3500 m
Ingress protection		IP20

Creepage distances and clearances

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Insulation test voltages, routine test

According to EN 61131-2

Parameter	Value	
230 V circuits against other circuitry	2500 V	1.2/50 µs
120 V circuits against other circuitry	1500 V	1.2/50 µs
120 V...240 V circuits against other circuitry	2500 V	1.2/50 µs
24 V circuits (supply, 24 V inputs/outputs, analog inputs/outputs), if they are galvanically isolated against other circuitry	500 V	1.2/50 µs
COM interfaces, galvanically isolated	500 V	1.2/50 µs
COM interfaces, electrically not isolated	Not applicable	Not applicable
FBP interface	500 V	1.2/50 µs
Ethernet	500 V	1.2/50 µs
ARCNET	500 V	1.2/50 µs
230 V circuits against other circuitry	1350 V	AC 2 s
120 V circuits against other circuitry	820 V	AC 2 s
120 V...240 V circuits against other circuitry	1350 V	AC 2 s
24 V circuits (supply, 24 V inputs/outputs, analog inputs/outputs), if they are galvanically isolated against other circuitry	350 V	AC 2 s
COM interfaces, galvanically isolated	350 V	AC 2 s
COM interfaces, electrically not isolated	Not applicable	Not applicable
FBP interface	350 V	AC 2 s
Ethernet	350 V	AC 2 s
ARCNET	350 V	AC 2 s

Power supply units

For the supply of the modules, power supply units according to SELV or PELV specifications must be used.

Safety Extra Low Voltage (SELV) and Protective Extra Low Voltage (PELV)

To ensure electrical safety of AC500/AC500-eCo extra low voltage circuits, 24 V DC supply, communication interfaces, I/O circuits, and all connected devices must be powered from sources meeting requirements of SELV, PELV, class 2, limited voltage or limited power according to applicable standards.

WARNING!

Improper installation can lead to death by touching hazardous voltages!

To avoid personal injury, safe separation, double or reinforced insulation and separation of the primary and secondary circuit must be observed and implemented during installation.

- Only use power converters for safety extra-low voltages (SELV) with safe galvanic separation of the primary and secondary circuit.
- Safe separation means that the primary circuit of mains transformers must be separated from the secondary circuit by double or reinforced insulation. The protective extra-low voltage (PELV) offers protection against electric shock.

Electromagnetic compatibility

Table 610: Range of use

Parameter	Value
Industrial applications	Yes
Domestic applications	No

Table 611: Immunity against electrostatic discharge (ESD), according to IEC 61000-4-2, zone B, criterion B

Parameter	Value
Electrostatic voltage in case of air discharge	8 kV
Electrostatic voltage in case of contact discharge	4 kV, in a closed switchgear cabinet 6 kV [1])
ESD with communication connectors	In order to prevent operating malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.
ESD with connectors of terminal bases	The connectors between the Terminal Bases and processor modules or Communication Modules must not be touched during operation. The same is valid for the I/O bus with all modules involved.

[1]) High requirement for shipping classes are achieved with additional specific measures (see specific documentation).

Table 612: Immunity against the influence of radiated (CW radiated), according to IEC 61000-4-3, zone B, criterion A

Parameter	Value
Test field strength	10 V/m

Table 613: Immunity against fast transient interference voltages (burst), according to IEC 61000-4-4, zone B, criterion B

Parameter	Value
Supply voltage units (DC)	2 kV
Supply voltage units (AC)	2 kV
Digital inputs/outputs (24 V DC)	1 kV
Digital inputs/outputs (120 V AC...240 V AC)	2 kV
Analog inputs/outputs	1 kV
CS31 bus	1 kV
Serial RS-485 interfaces (COM)	1 kV
Serial RS-232 interfaces (COM, not for PM55x and PM56x)	1 kV
ARCNET	1 kV
FBP	1 kV
Ethernet	1 kV
I/O supply (DC-out)	1 kV

Table 614: Immunity against the influence of line-conducted interferences (CW conducted), according to IEC 61000-4-6, zone B, criterion A

Parameter		Value
Test voltage		3V zone B, 10 V is also met.
High energy surges		According to IEC 61000-4-5, zone B, criterion B
	Power supply DC	1 kV CM / 0.5 kV DM [2])
	DC I/O supply	0.5 kV CM / 0.5 kV DM [2])
	Communication Lines, shielded	1 kV CM [2])
	AC I/O unshielded [3])	2 kV CM / 1 kV DM [2])
	I/O analog, I/O DC unshielded [3])	1 kV CM / 0.5 kV DM [2])
Radiation (radio disturbance)		According to IEC 55011, group 1, class A

[2]) CM = Common Mode, DM = Differential Mode

[3]) When DC I/O inputs are used with AC voltage, external filters limiting high energy surges to 1 kV CM / 0.5 DM are required to meet requirements according IEC 61131-2.

Mechanical data

Parameter	Value
Mounting	Horizontal
Degree of protection	IP 20

Parameter	Value
Housing	Classification V-2 according to UL 94
Vibration resistance acc. to EN 61131-2	all three axes 2 Hz...8.4 Hz, continuous 3.5 mm 8.4 Hz...150 Hz, continuous 1 g (higher values on request)
Shock test	All three axes 15 g, 11 ms, half-sinusoidal
Mounting of the modules:	
DIN rail according to DIN EN 50022	35 mm, depth 7.5 mm or 15 mm
Mounting with screws	Screws with a diameter of 4 mm
Fastening torque	1.2 Nm

Approvals and certifications

Information on approvals and certificates can be found in the corresponding chapter of the *Main catalog, PLC Automation*.

1.6.3.6.2 Mechanical dimensions

Switchgear cabinet assembly

Information on EMC-conforming assembly and construction is provided within the overall functions section ↳ Chapter 1.6.3.4.4 “EMC-conforming assembly and construction” on page 5226.

PLC enclosure

NOTICE!

PLC damage due to wrong enclosures

Due to their construction (degree of protection IP 20 according to EN 60529) and their connection technology, the devices are suitable only for operation in enclosed switchgear cabinets.

To protect PLCs against:

- unauthorized access,
- dusting and pollution,
- moisture and wetness and
- mechanical damage,

switchgear cabinet IP54 for common dry factory floor environment is suitable.

Maintain spacing from:

- enclosure walls
- wireways
- adjacent equipment

Allow a minimum of 20 mm clearance on all sides. This provides ventilation and galvanic isolation.

It is recommended to mount the modules on an grounded mounting plate, or an grounded DIN rail, independent of the mounting location.

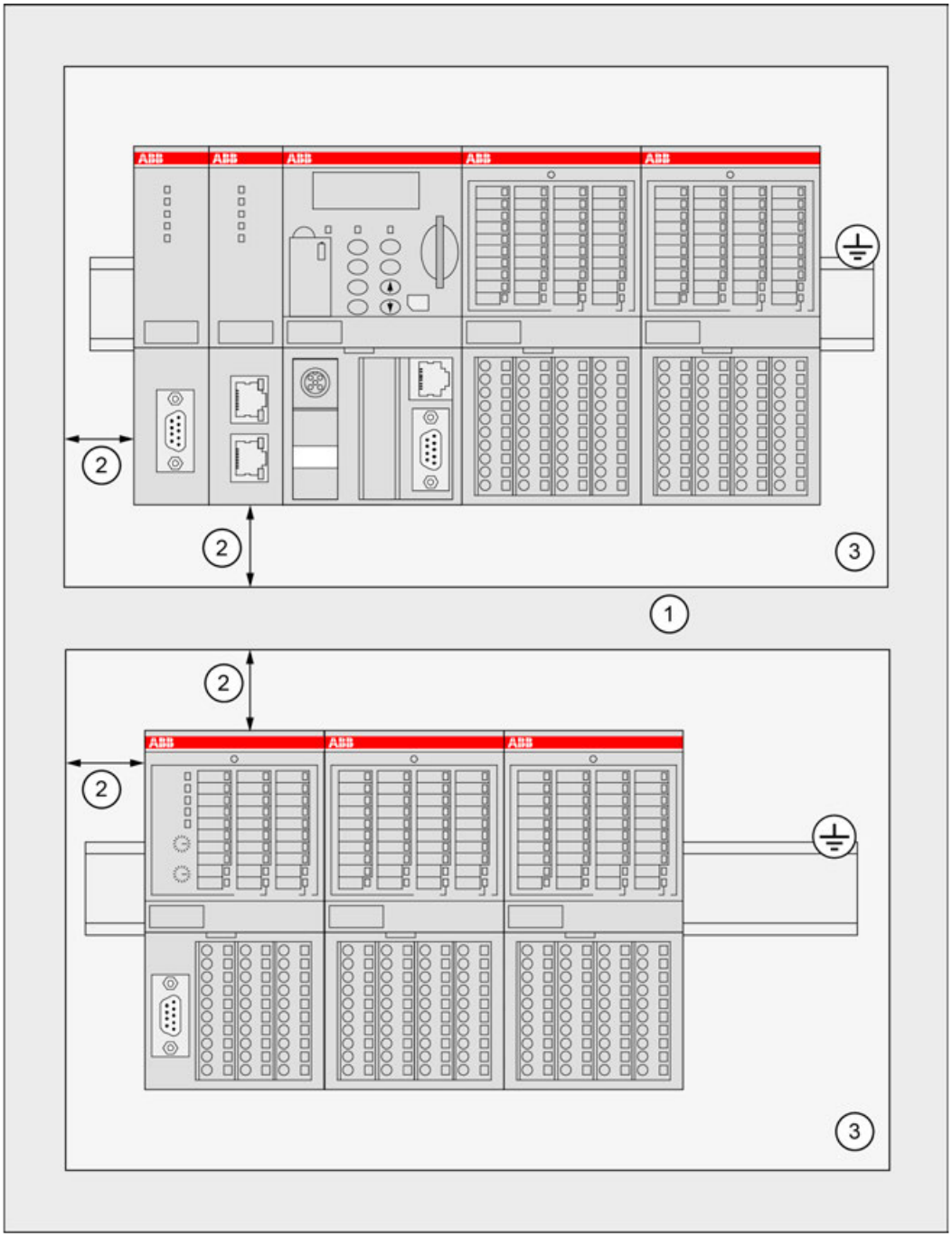


Fig. 1068: Installation of AC500/S500 modules in a switchgear cabinet

1 Cable duct
2 Distance from cable duct ≥20 mm
3 Mounting plate, grounded

NOTICE!

Horizontal mounting is highly recommended.

Vertical mounting is possible, however, derating consideration should be made to avoid problems with poor air circulation and overheating (see *Chapter 1.6.3.6.1.1 “Environmental conditions” on page 5313*).

When vertically mounted, always place an end-stop terminal block (e.g. type BADL, P/N: 1SNA399903R0200) on the bottom and on the top of the modules to properly secure the modules.

With high vibration applications and horizontal mounting, we also recommend to place end-stop terminals at the right and left side of the device to properly secure the modules, e.g. type BADL, P/N: 1SNA399903R0200.

Mechanical dimensions AC500

Dimensions: terminal bases

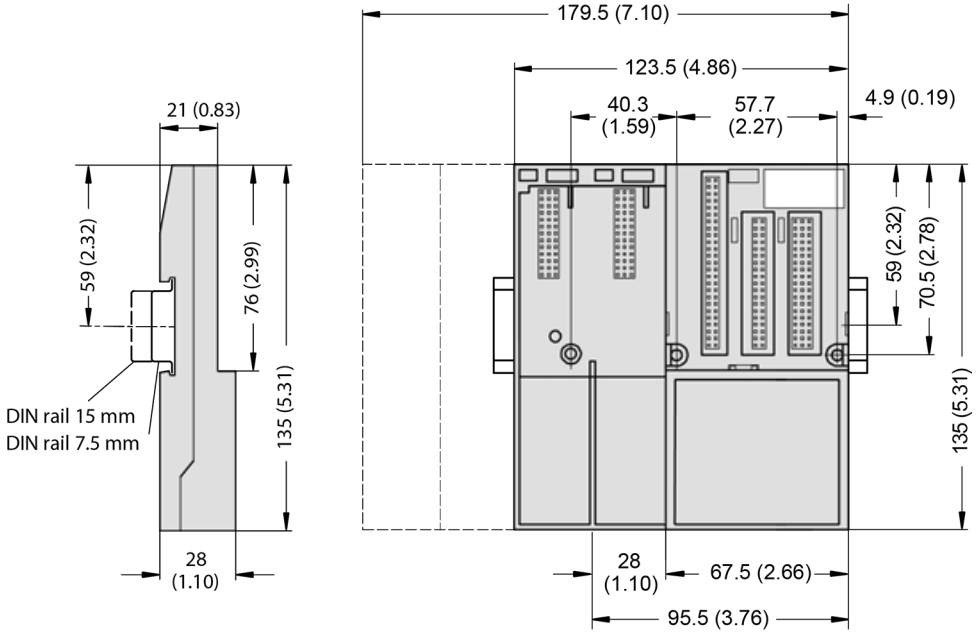


Fig. 1069: Terminal bases, side view and front view

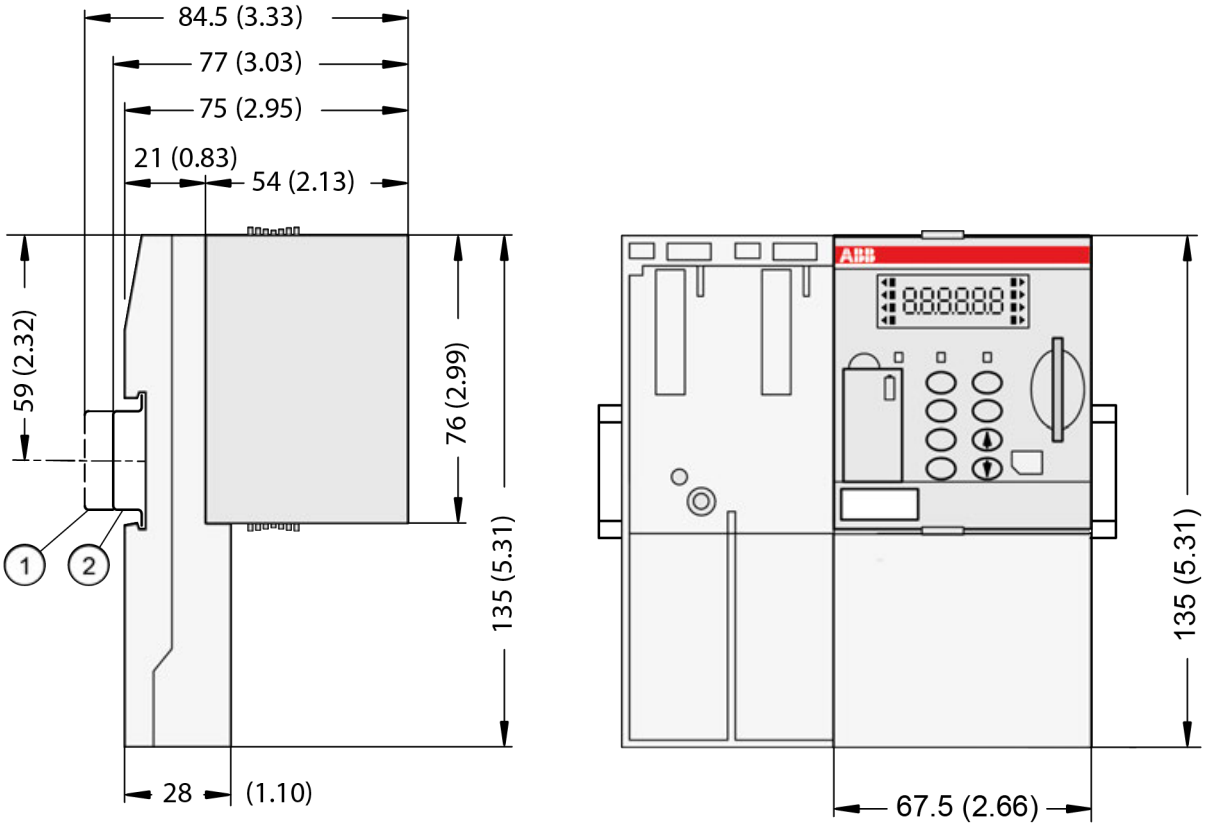


Fig. 1070: Terminal bases with processor modules, side view and front view

Dimensions:
function module
terminal bases

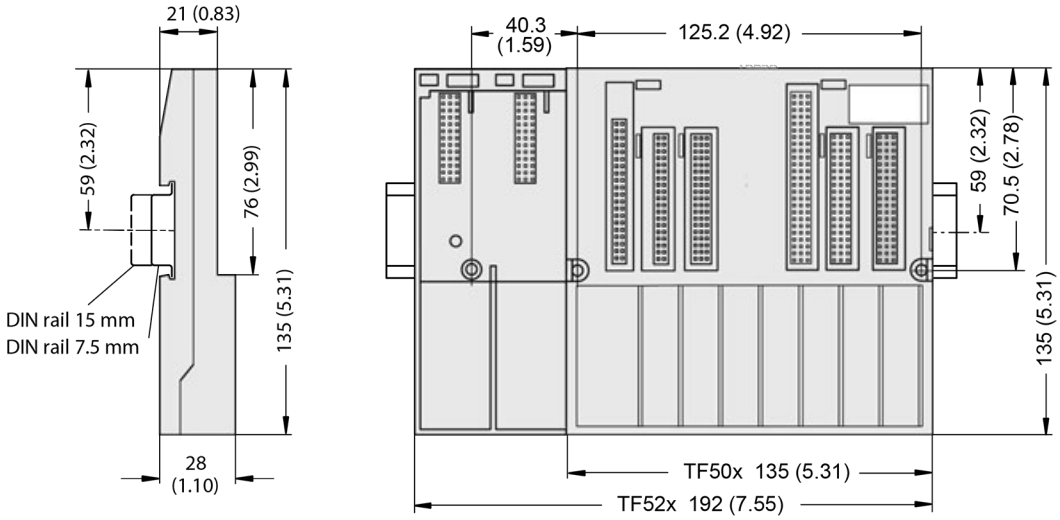


Fig. 1071: Function module terminal bases, side view and front view

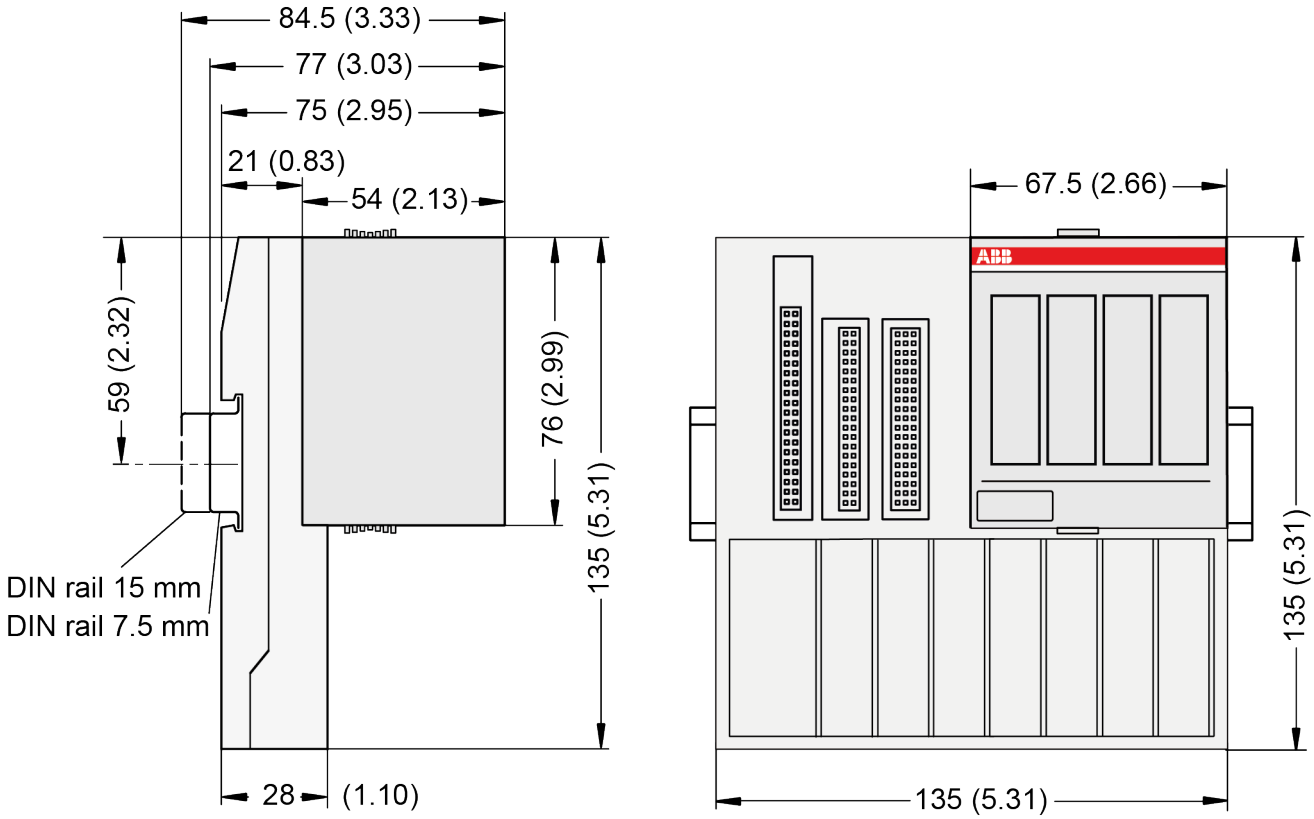


Fig. 1072: Function module terminal bases with function modules for CMS, side view and front view

Dimensions:
PM595

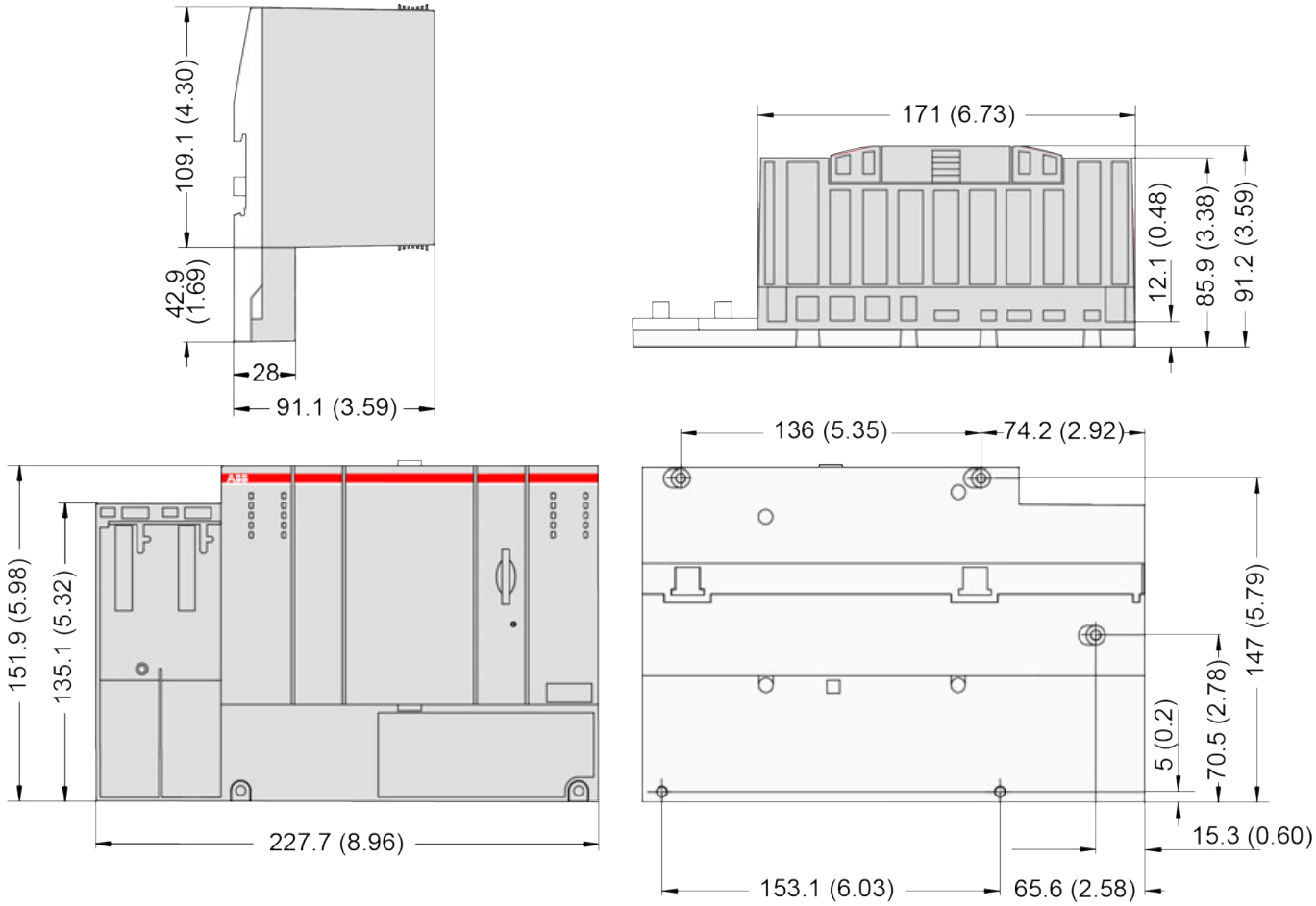


Fig. 1073: Processor podule PM595, side view, top view, front view, back view

Mechanical dimensions S500

Dimensions:
Terminal units

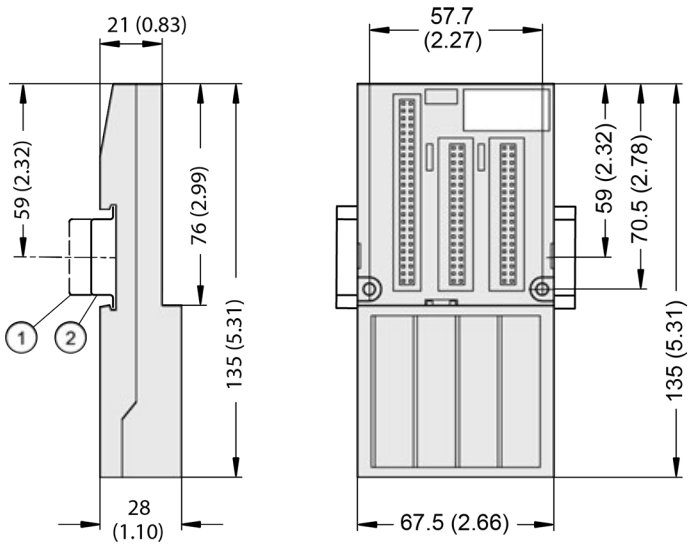


Fig. 1074: Terminal units, side view and front view

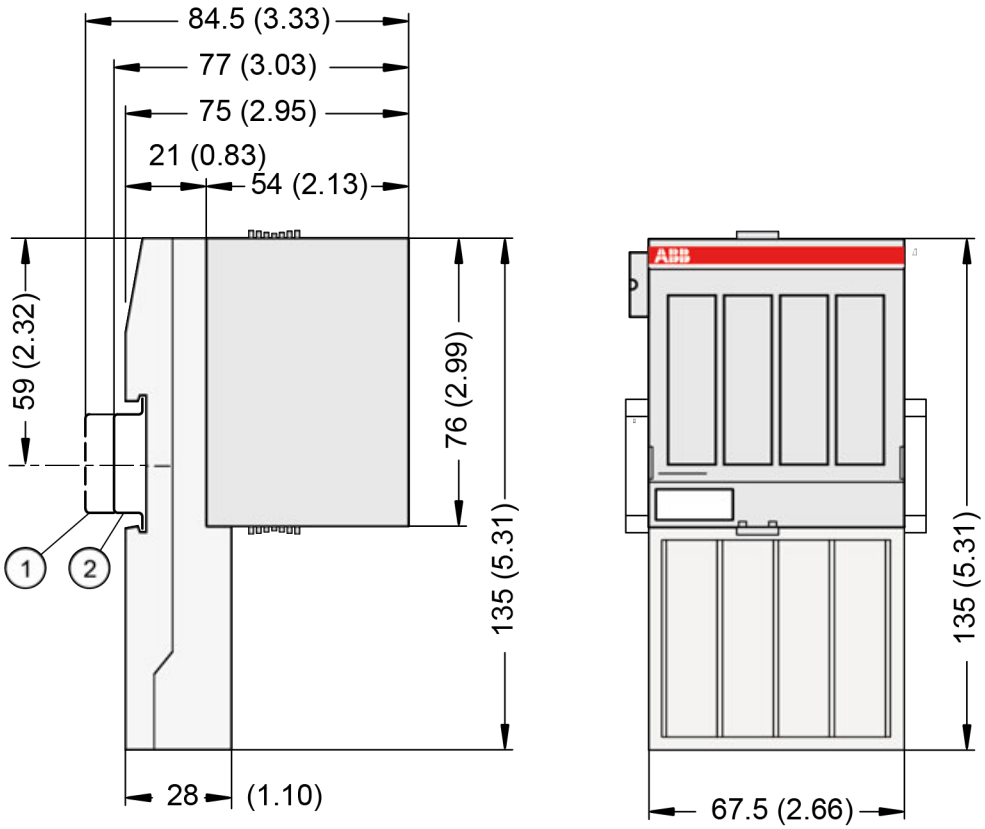


Fig. 1075: Terminal units and S500 modules, side view and front view

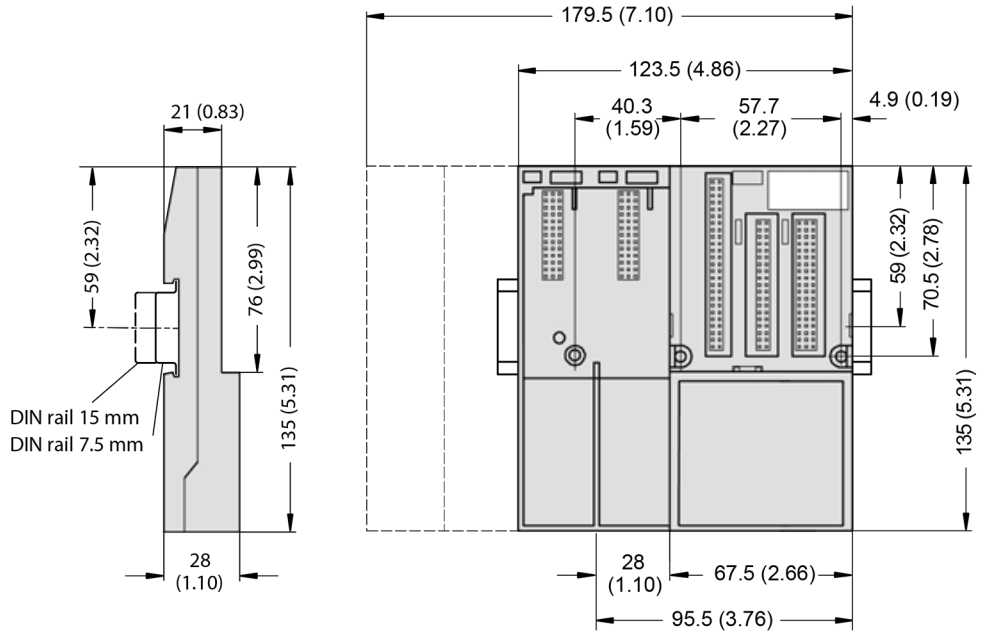


Fig. 1076: Terminal base (for comparison)

All dimensions are in mm (in.). Hole spacing tolerance: ±0.4 mm (0.016 in.)

Dimensions:
FM502-CMS

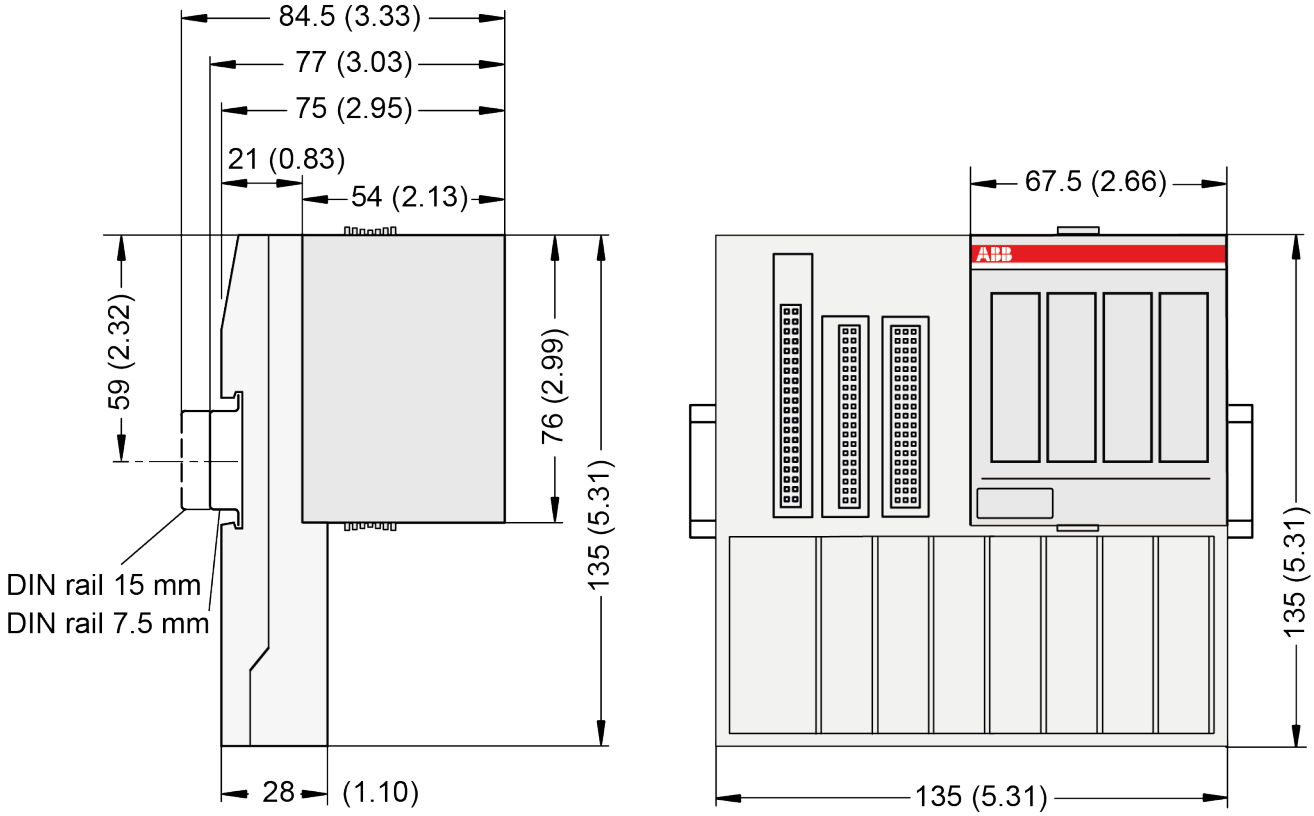


Fig. 1077: Function module terminal bases and function modules for CMS, side view and front view

1.6.3.6.3 Mounting and demounting

The control system is designed to be mounted to a well-grounded mounting surface such as a metal panel. Additional grounding connections from the mounting tabs or DIN rail (if used), are not required unless the mounting surface cannot be grounded.

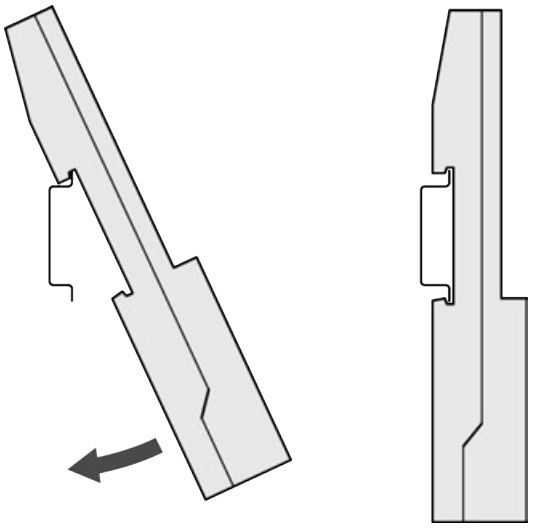
During panel or DIN rail mounting of all devices, be sure that all debris (metal chips, wire strands, etc.) is kept from falling into the controller. Debris that falls into the controller could cause damage while the controller is energized.

All devices are grounded through the DIN rail to chassis ground. Use zinc plated yellow-chromate stell DIN rail to assure proper grounding. The use of other DIN rail materials (e.g. aluminium, plastic, etc.) that can corrode, oxidize, or are poor conductors, can result in improper or intermittent grounding.

Mounting/Demounting terminal bases and function module terminal bases

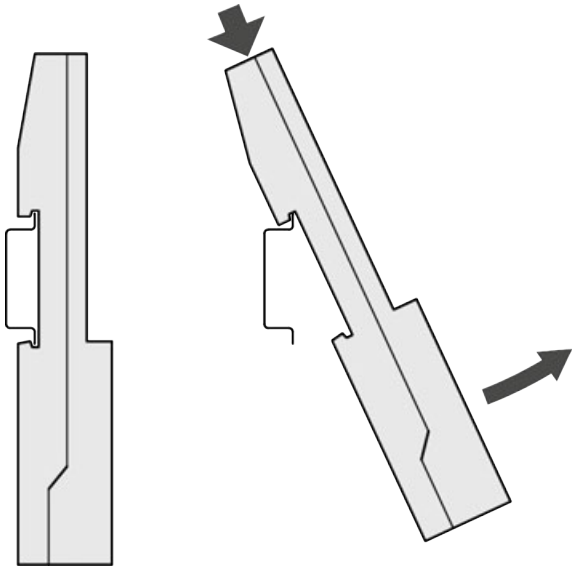
Demounting on DIN rail

1. Mount DIN rail 7.5 mm or 15 mm.
2. Mount the terminal base/function module terminal base:



⇨ The terminal base is put on the DIN rail above and then snapped-in below.

3. The demounting is carried out in a reversed order.



Mounting with screws

If the Terminal Base should be mounted with screws, wall mounting accessories TA526 ↳ *Chapter 1.6.3.6.5.5 “TA526 - Wall mounting accessory” on page 5378* must be inserted at the rear side first. These plastic parts prevent bending of the terminal base while screwing on. TB51x needs one TA526, TB52x and TB54x need two TA526.

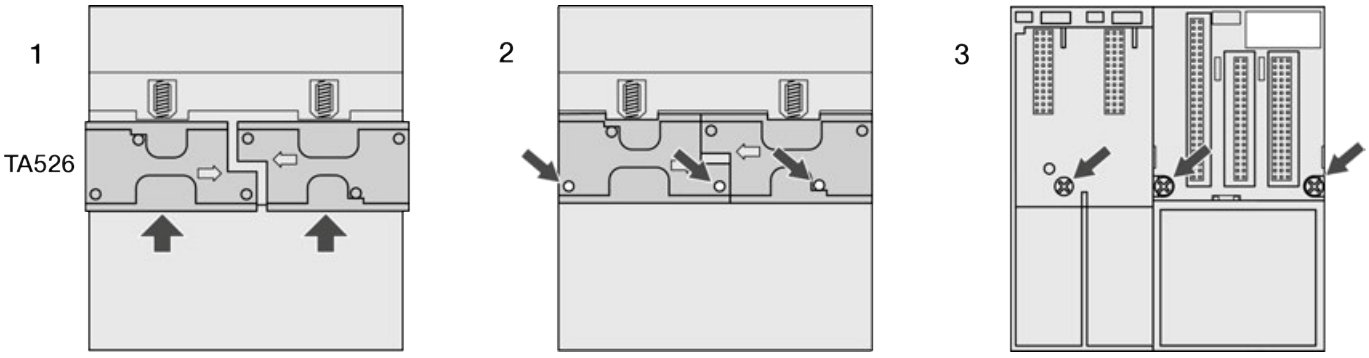


Fig. 1078: Terminal bases, Fastening with screws

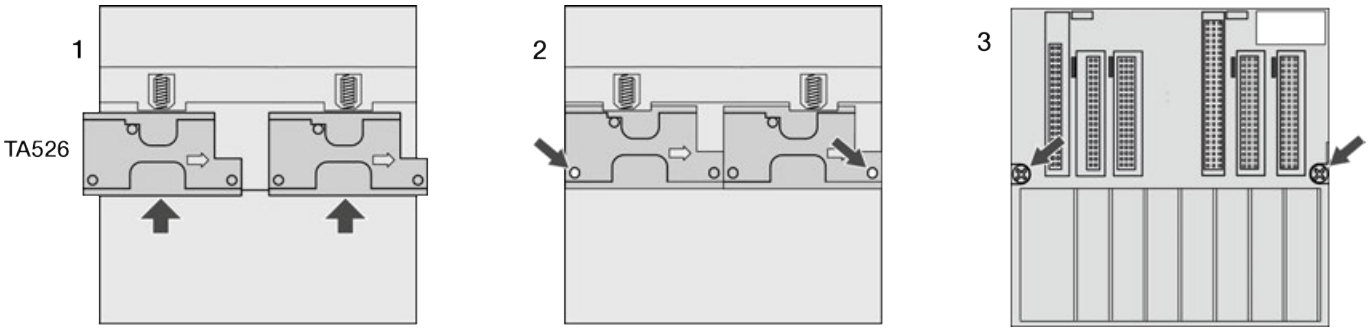


Fig. 1079: Function module terminal bases, Fastening with screws

> *By wall mounting, the terminal base is grounded through the screws. It is necessary that*
>
> – *the screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated)*
> – *the mounting plate is grounded*
> – *the screws have a good electrical contact to the mounting plate*

Practical tip

The following procedure allows you to use the mounted modules as a template for drilling holes in the panel. Due to module mounting hole tolerance, it is important to follow these procedures:

1. On a clean work surface, mount no more than 3 modules (e.g. one terminal base and two terminal units).
2. Using the mounted modules as a template, carefully mark the center of all module-mounting holes on the panel.
3. Return the mounted modules to the clean work surface, including any previously mounted modules.
4. Drill and tap the mounting holes for the screws (M4 or #8 recommended).
5. Place the modules back on the panel and check for proper hole alignment.
6. Attach the modules to the panel using the mounting screws.

> *If mounting more modules, mount only the last one of this group and put the others aside. This reduces remounting time during drilling and tapping of the next group.*

7. Repeat the steps for all remaining modules.

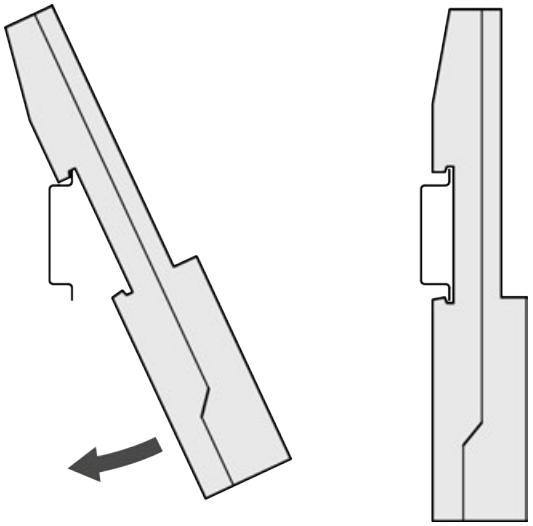
Mounting/Demounting the terminal unit

Mounting on DIN rail

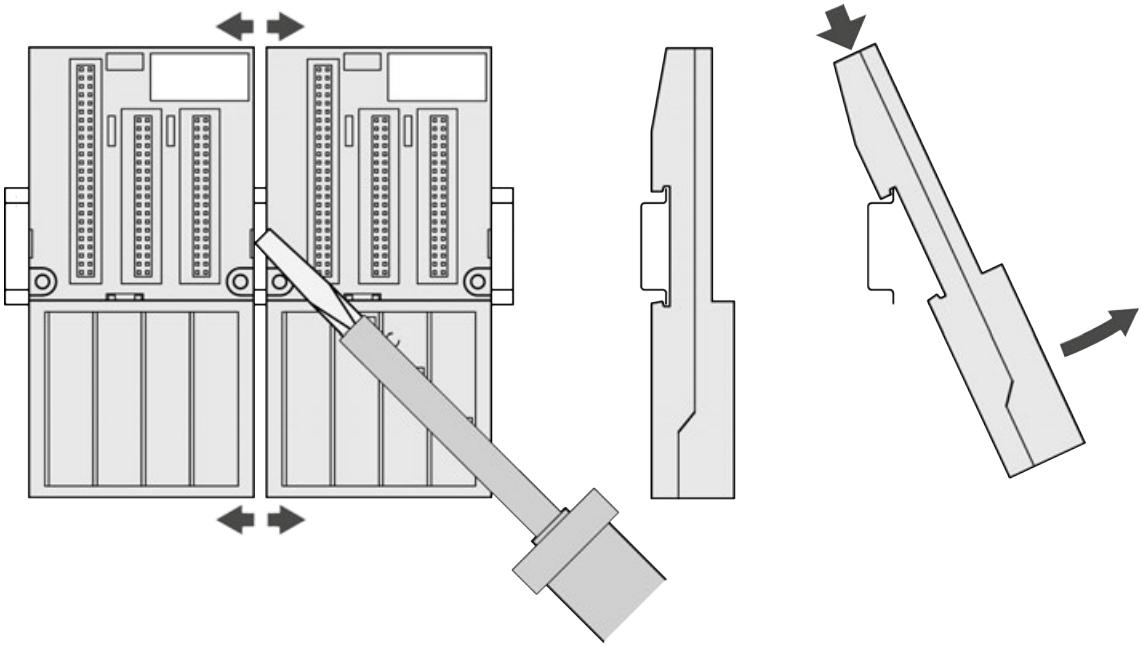
1. Mount DIN rail 7.5 mm or 15 mm.
2. Mount the terminal unit.

   The terminal unit is snapped into the DIN rail in the same way as the Terminal Base. Once secured to the DIN rail, slide the terminal unit to the left until it fully locks into place creating a solid mechanical and connection.

> *When attaching the devices, make sure the bus connectors are securely locked together to ensure proper connection. Max. 10 terminal units can be attached.*



3. Demounting: A screwdriver is inserted in the indicated place to separate the terminal units.



Mounting with screws

If the terminal unit should be mounted with screws, wall mounting accessories TA526 ↳ *Chapter 1.6.3.6.5.5 “TA526 - Wall mounting accessory” on page 5378* must be inserted at the rear side first. These plastic parts prevent bending of the Terminal Base while screwing on.

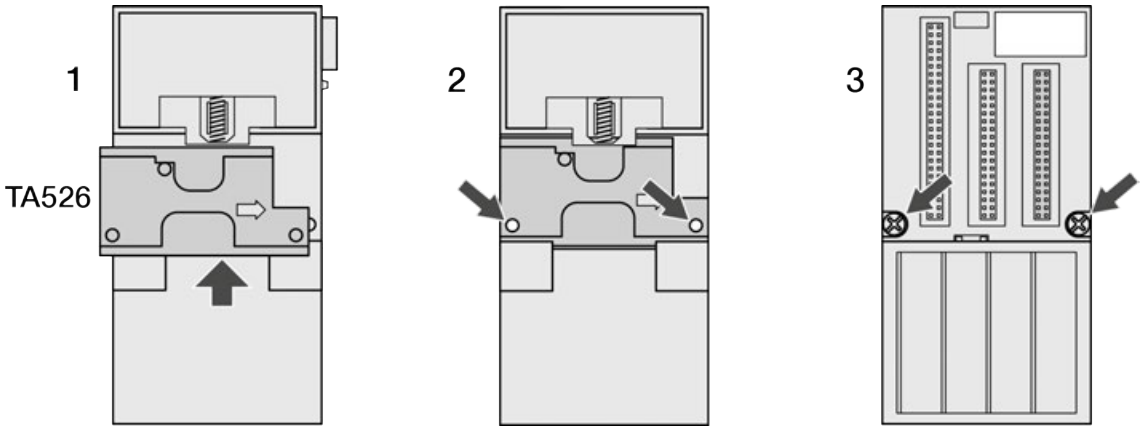


Fig. 1080: Fastening with screws

By wall mounting, the terminal unit is grounded through the screws. It is necessary that

- *the screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated)*
- *the mounting plate is grounded*
- *the screws have a good electrical contact to the mounting plate*

Practical tip

The following procedure allows you to use the mounted modules as a template for drilling holes in the panel. Due to module mounting hole tolerance, it is important to follow these procedures:

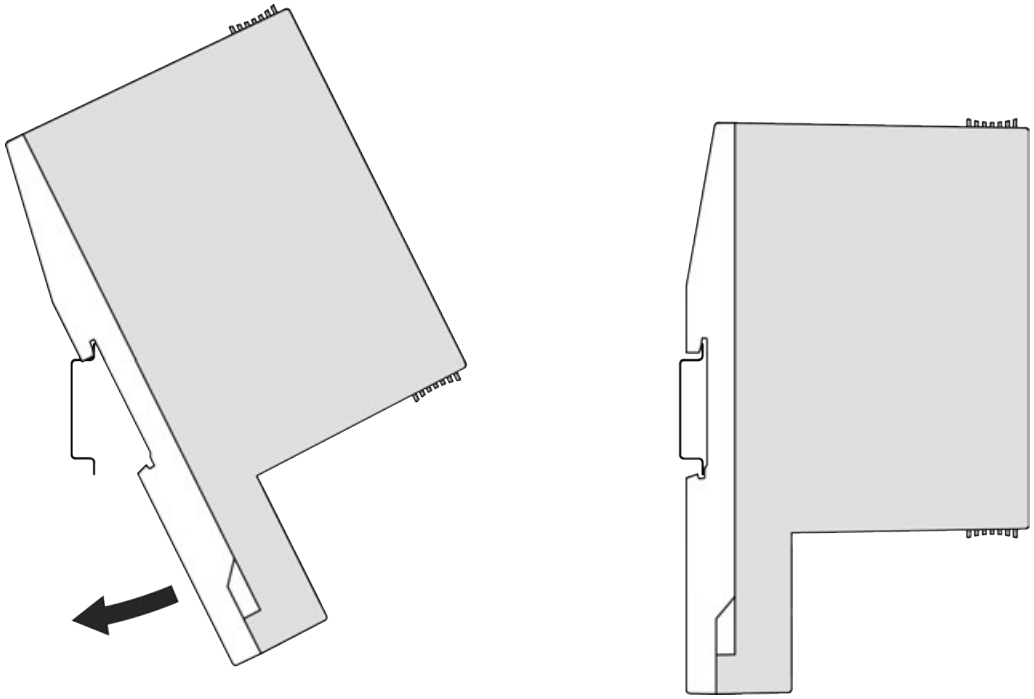
1. On a clean work surface, mount no more than 3 modules (e.g. one terminal base and two terminal units).
2. Using the mounted modules as a template, carefully mark the center of all module-mounting holes on the panel.
3. Return the mounted modules to the clean work surface, including any previously mounted modules.
4. Drill and tap the mounting holes for the screws (M4 or #8 recommended).
5. Place the modules back on the panel and check for proper hole alignment.
6. Attach the modules to the panel using the mounting screws.

   If mounting more modules, mount only the last one of this group and put the others aside. This reduces remounting time during drilling and tapping of the next group.

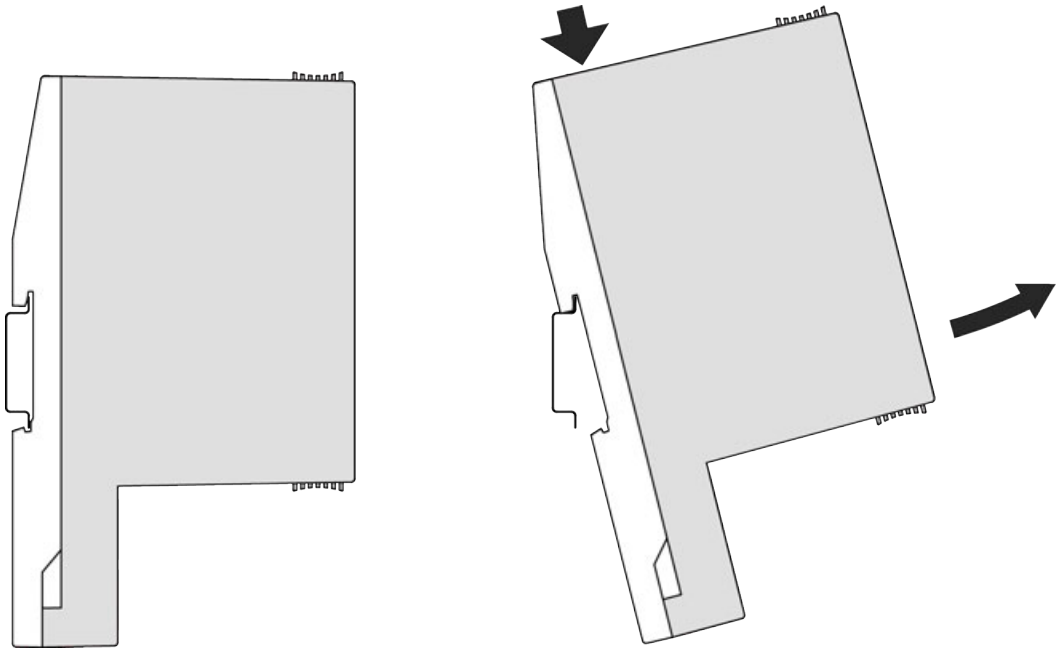
7. Repeat the steps for all remaining modules.

Mounting/Demounting the processor module PM595

Mounting on DIN rail



▷ Put the processor module on the DIN rail above and then snapped-in below. The demounting is carried out in a reversed order.



1. Pull down the processor module.
2. Remove it.

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↳ *Chapter 1.6.3.6.5.7 “TA524 - Dummy communication module” on page 5383*.
– I/O bus connectors must not be touched during operation.

NOTICE!

Only use TA543 accessory when the PLC is to be screw mounted. With DIN rail mounting the PLC could not be removed from the rail without the risk of damaging the housing.

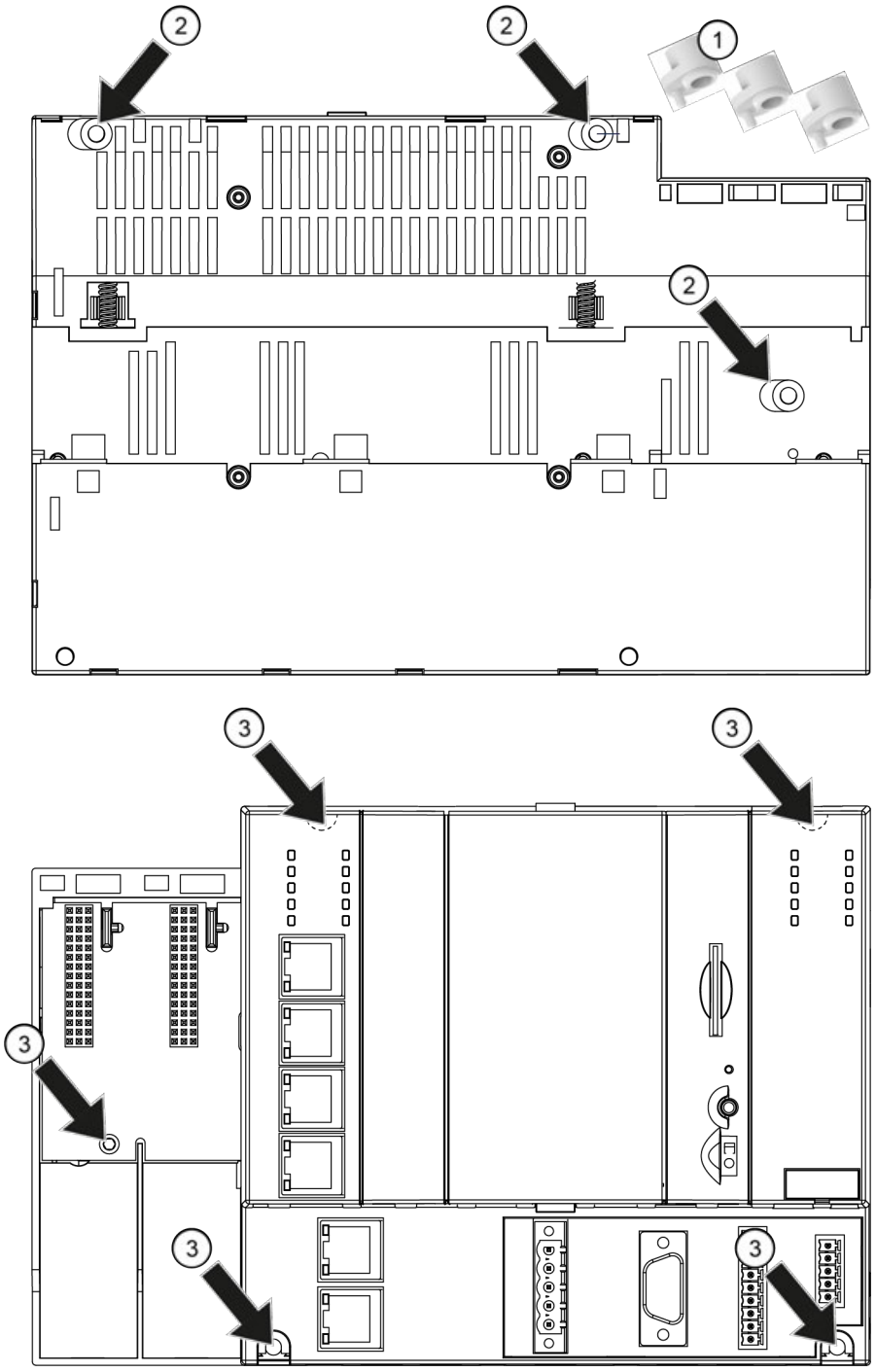
Mounting with screws

NOTICE!

Use screw mounting accessory to avoid damage!

For screw mounting, the use of the TA543 screw mounting accessory (1SAP182800R0001) is mandatory to prevent bending and damage to the module.

A dimension drawing for the position of screw's holes can be found in mechanical dimensions AC500 ↳ *Chapter 1.6.3.6.2.2 “Mechanical dimensions AC500” on page 5320.*



1 3 parts of screw mounting accessory TA543
2 3 slots for screw mounting accessory TA543
3 5 holes for screw mounting

1. Insert 3 parts of screw mounting accessory TA543 into the slots on the backside of the processor module PM595.

⇨

NOTICE!

Use screw mounting accessory to avoid damage!

For screw mounting, the use of the TA543 screw mounting accessory (1SAP182800R0001) is mandatory to prevent bending and damage to the module.

2. Fasten the processor module PM595 with 5 screws (M4, max 1.2 Nm) from the front side.

⇨

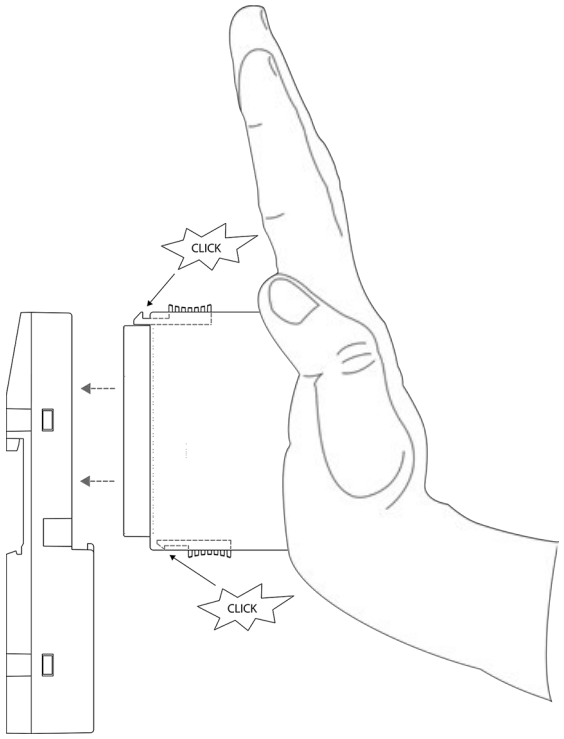
By screw mounting, the processor module PM595 is grounded through the screws. It is necessary that

- *the screws have a conductive surface (e.g. steel zinc-plated or brass nickel-plated)*
- *the mounting plate is grounded*
- *the screws have a good electrical contact to the mounting plate*

Thread lock washer is highly recommended to prevent the screw from loosening after long time use.

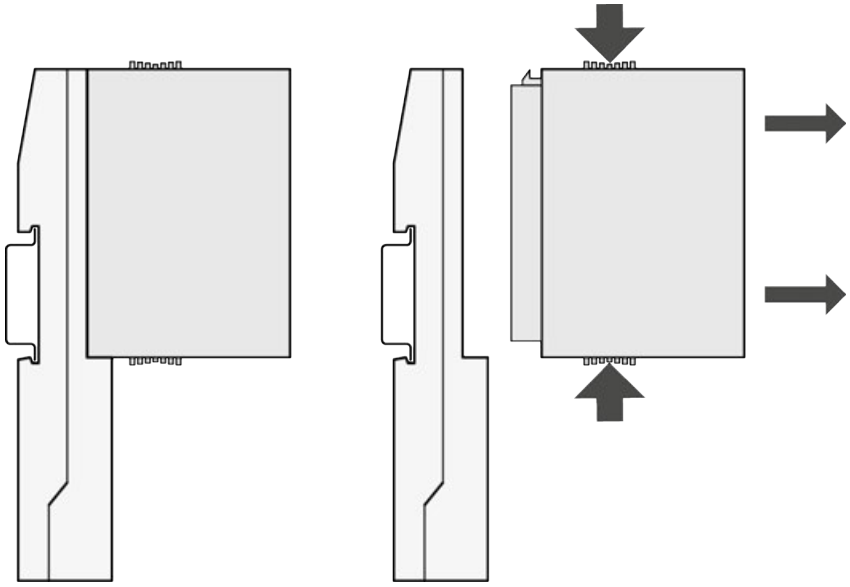
Mounting processor modules PM57x, PM58x, PM59x and PM56xx

1. After mounting the Terminal Base on the DIN rail, mount the processor module.



2. Press the processor module into the Terminal Base until it locks in place.

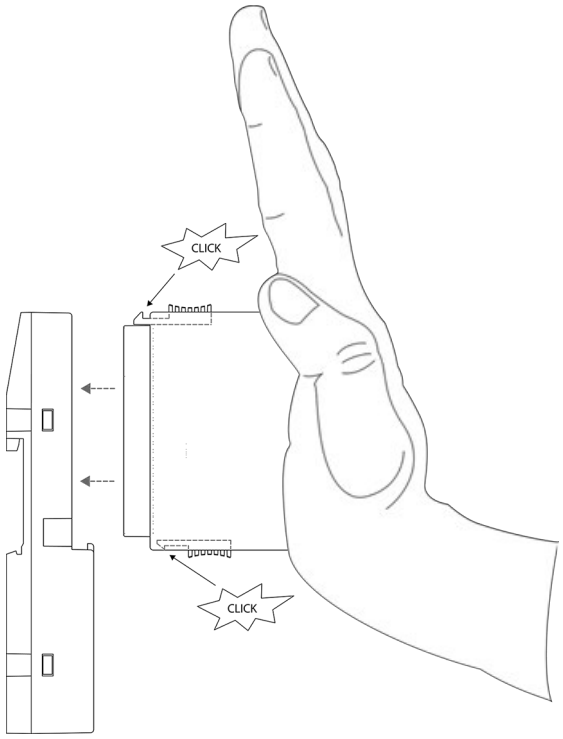
3. The demounting is carried out in a reversed order. Press above and below, then remove the processor module.



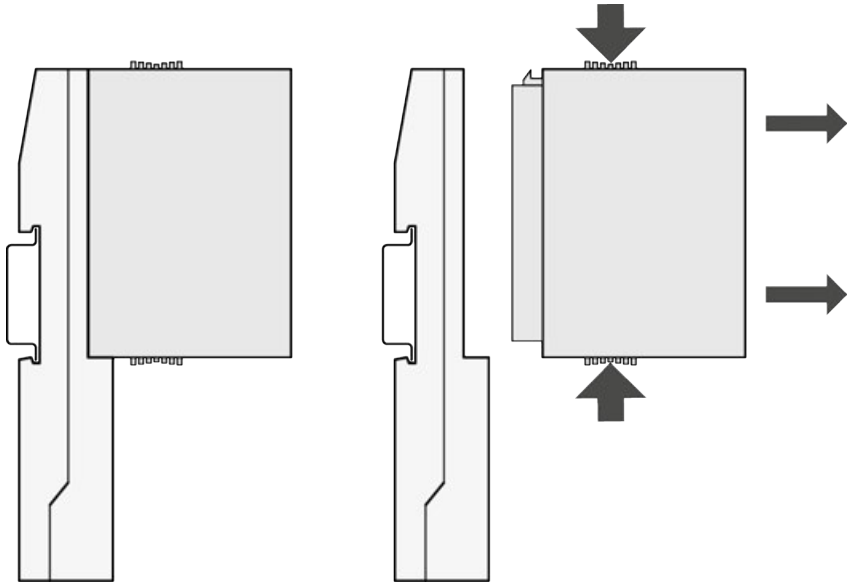
Mounting/Demounting the I/O modules

After mounting the terminal unit, mount the I/O modules.

1. Press the I/O module into the terminal unit until it locks in place.



2. The demounting is carried out in a reversed order.
   Press above and below, then remove the module.



Mounting/Demounting the communication modules

Communication modules are mounted on the left side of the processor module on the same terminal base. The connection is established automatically when mounting the communication module.

NOTICE!

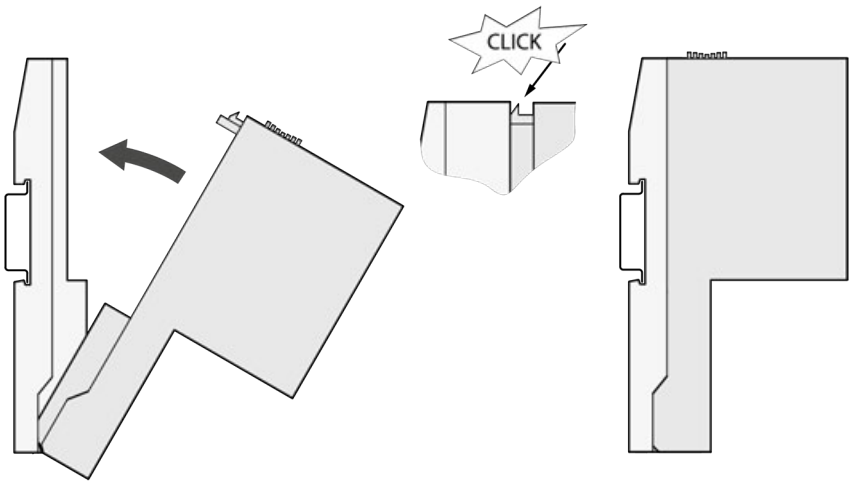
Risk of damaging the PLC modules!

Overvoltages and short circuits might damage the PLC modules.

- Make sure that all voltage sources (supply voltage and process supply voltage) are switched off before you begin with operations on the system.
- Never connect any voltages or signals to reserved terminals (marked with ---). Reserved terminals may carry internal voltages.

After mounting the terminal base, mount the communication modules.

1. First insert the bottom nose of the communication module into the dedicated holes of the terminal base. Then, rotate the communication module on the dedicated terminal base slot until it is locked in place.



⇨

NOTICE!

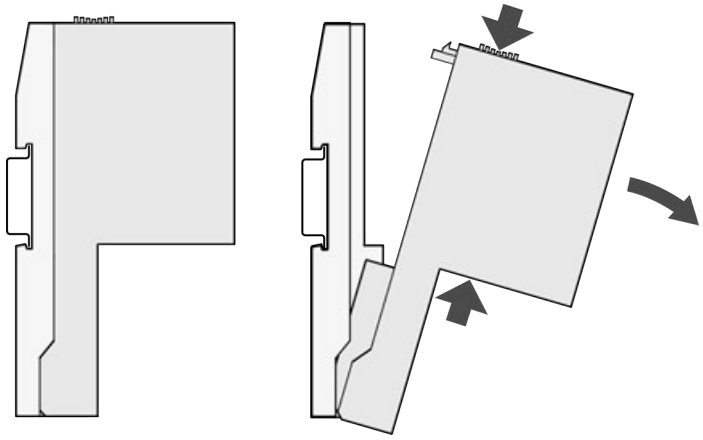
Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↬ *Chapter 1.6.3.6.5.7 “TA524 - Dummy communication module” on page 5383*.
– I/O bus connectors must not be touched during operation.

2. The demounting is carried out in a reversed order.

   Press above and below, then rotate the communication module and remove it.



Mounting/Demounting the accessories

Additional components such as batteries, cables, etc. are required for commissioning the PLC system. Information on assembly, replacement or basic use of the orderable components can be found in the description of the respective accessory.

↬ *Chapter 1.6.3.6.5 “Handling of accessories” on page 5359*

Hardware details can be found in the device specifications of the accessory.

↬ *Chapter 1.6.2.9 “Accessories” on page 5095*

1.6.3.6.4 Connection and wiring

For detailed information such as technical data of your mounted devices (AC500 product family) refer to the hardware device description of the appropriate device.

NOTICE!

Attention:

The devices should be installed by experts who are trained in wiring electronic devices. In case of bad wiring, the following problems could occur:

- On the terminal base, the terminals L+ and M are doubled. If the power supply is badly connected, a short circuit could happen and lead to a destruction of the power supply or its fuse. If no suitable fuse exists, the terminal base itself might be destroyed.
- The terminal bases and all electronic modules and terminal units are protected against reverse polarity.
- All necessary measures should be carried out to avoid damages to modules and wiring. Notice the wiring plans and connection examples.

NOTICE!

Attention:

All I/O channels (digital and analog) are protected against reverse polarity, reverse supply, short circuit and continuous overvoltage up to 30 V DC.

NOTICE!

Attention:

Due to possible loss of communcation, the communication cables should be fixed with cable duct or bracket or clamp during application.

Power supply

AC500 system power supply

As soon as the power supply of the processor module (CPU) is higher than the minimum Process and supply voltage (see ↳ *Chapter 1.6.3.6.1.1 “Environmental conditions” on page 5313*), the power supply detection is activated and the processor module is started. Power supply of processor module and I/O modules should be powered on the same time, otherwise the processor module will not switch to run after startup.

When during operation the power supply is going down lower than the minimum Process and supply voltage (see ↳ *Chapter 1.6.3.6.1.1 “Environmental conditions” on page 5313*) for more than 10 ms, the processor module is switched to safety mode (display shows “AC500”). A restart of the processor module only occurs by switching the power supply off and on again.

If an I/O module is disconnected during normal operation from power supply while processor module is still powered, the processor module will continue its normal operation on all other powered peripherals (I/O modules, communication modules and communication interfaces), but freezes the input image. After recovery of I/O Module power supply it will continue normal operation and inputs and outputs were updated.

Logic Controller Supply: AC500 logic controller power supply is provided through terminals L+ / M.

Process Power Supply: S500 process power supply is provided through terminals UP / ZP.

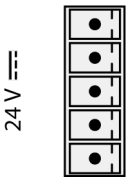
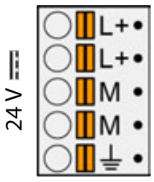
Logic Controller Supply is galvanic isolated from Process Power Supply.

As system power supply for AC500/S500, the ABB CP power supply series can be used.

Power supply for processor modules

The supply voltage of 24 V DC is connected to a removable 5-pin terminal block. L+/M exist twice. It is therefore possible to feed e.g. external sensors (up to 8 A max. with 1.5 mm^2 conductor) via these terminals.

Pin assignment

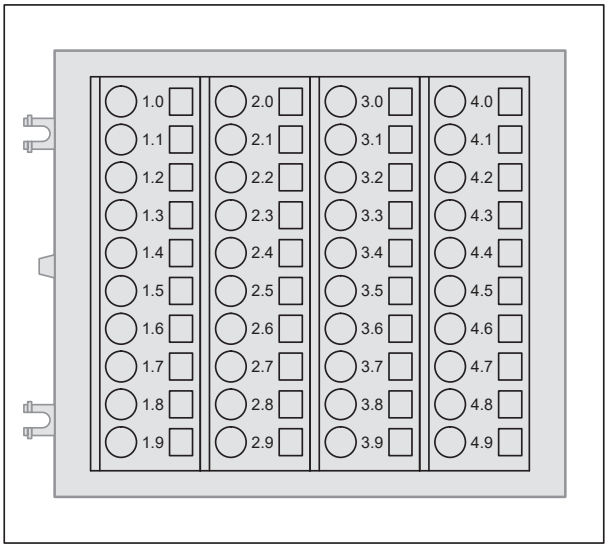
Pin Assignment		Label	Function	Description
 Terminal block removed	 Terminal block inserted	L+	+24 V DC	Positive pin of the power supply voltage
		L+	+24 V DC	Positive pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		M	0 V	Negative pin of the power supply voltage
		⏚	FE	Functional earth

Terminals for power supply and the COM1 interface

Terminal type: Spring terminal

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 1.5 mm^2
1	Flexible	0.08 mm^2 to 1.5 mm^2
1 with wire-end ferrule (without plastic sleeve)	Flexible	0.25 mm^2 to 1.5 mm^2
1 with wire-end ferrule (with plastic sleeve)	Flexible	0.25 mm^2 to 0.5 mm^2
1 (TWIN wire end ferrule)	Flexible	0.5 mm^2

Terminals at the terminal unit



Terminal type: Screw-type terminal

Front terminal, conductor connection vertically with respect to the printed circuit board.

Parameter	Value
Type	Front terminal
Degree of protection	IP 20
Stripped conductor end	9 mm, min. 8 mm
Fastening torque	0.6 Nm
Needed tool	Slotted screwdriver
Dimensions	Blade diameter 3.5 mm

Terminal units with product index < C0 e. g. 1SAP 212 200 R0001 B0

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule	Flexible	0.25 mm^2 to 1.5 mm^2
2	Solid	Not intended
2	Flexible	Not intended
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.25 mm^2 or 2 x 0.5 mm^2 or 2 x 0.75 mm^2, with square cross-section of the wire-end ferrule also 2 x 1.0 mm^2

Terminal units with product index ≥ C0 e. g. 1SAP 212 200 R0001 C0

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule without plastic sleeve	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule with plastic sleeve	Flexible	0.14 mm^2 to 1.5 mm^2
2	Solid	0.08 mm^2 to 1.5 mm^2
2	Flexible	0.08 mm^2 to 1.5 mm^2
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.5 mm^2 to 2 x 1.0 mm^2
2 with separate wire-end ferrule without plastic sleeve	Flexible	0.08 mm^2 to 0.75 mm^2

Terminal type: Spring terminal

Front terminal, conductor connection vertically with respect to the printed circuit board.

Parameter	Value
Type	Front terminal
Degree of protection	IP 20
Stripped conductor end	9 mm, min. 8 mm
Needed tool	Slotted screwdriver
Dimensions	2.5 x 0.4 to 3.5 x 0.5 mm, screwdriver must be at least 15 mm free of insulation at the tip

Number of cores per terminal	Conductor type	Cross section
1	Solid	0.08 mm^2 to 2.5 mm^2
1	Flexible	0.08 mm^2 to 2.5 mm^2
1 with wire-end ferrule	Flexible	0.25 mm^2 to 1.5 mm^2
2	Solid	Not intended
2	Flexible	Not intended
2 with TWIN wire end ferrule (length 10 mm) with plastic sleeve	Flexible	2 x 0.25 mm^2 or 2 x 0.5 mm^2 or 2 x 0.75 mm^2, with square cross-section of the wire-end ferrule also 2 x 1.0 mm^2

Connection of wires at the spring terminals

Connection

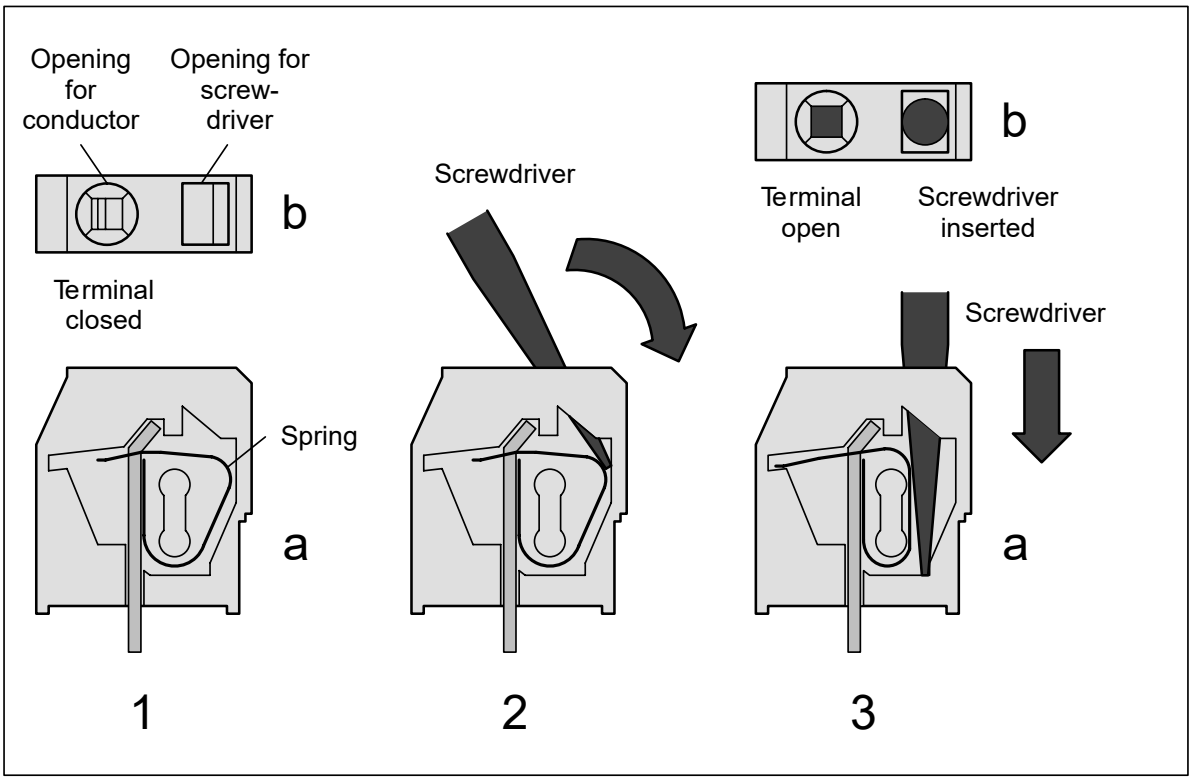


Fig. 1081: Connect the wire to the spring terminal (steps 1 to 3)

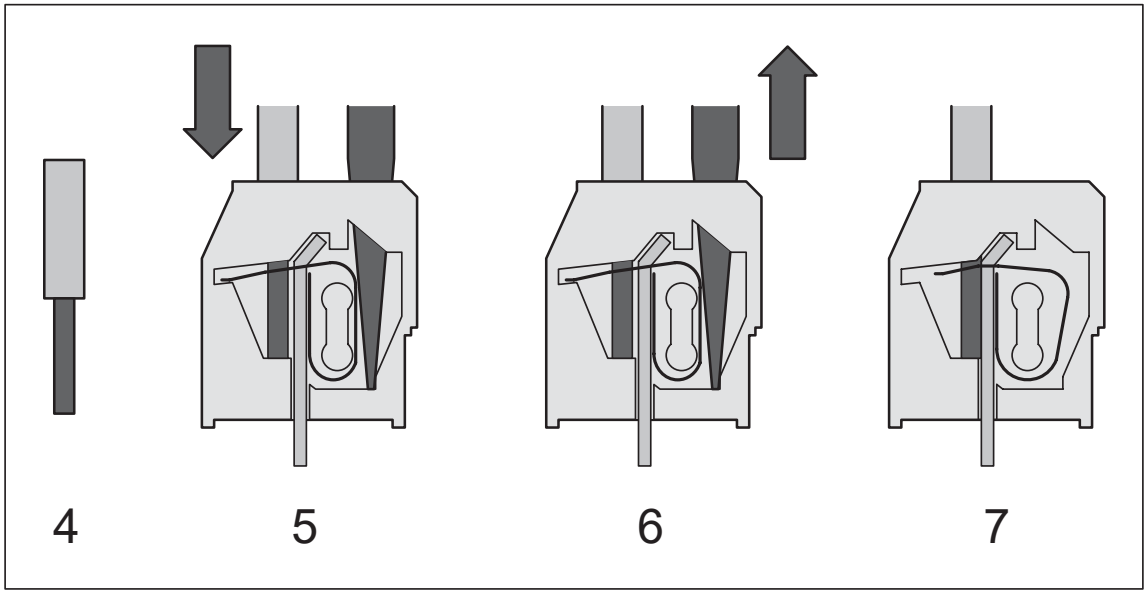


Fig. 1082: Connect the wire to the spring terminal (steps 4 to 7)

1. Side view (open terminal drawn for illustration)
2. The top view shows the openings for wire and screwdriver
3. Insert screwdriver (2.5 x 0.4 to 3.5 x 0.5 mm) at an angle, screwdriver must be at least 15 mm free of insulation at the tip
4. While erecting the screwdriver, insert it until the stop (requires a little strength)
5. Screwdriver inserted - terminal open
6. Strip the wire for 7 mm (and put on wire-end ferrule)
7. Insert wire into the open terminal
8. Done

Disconnection

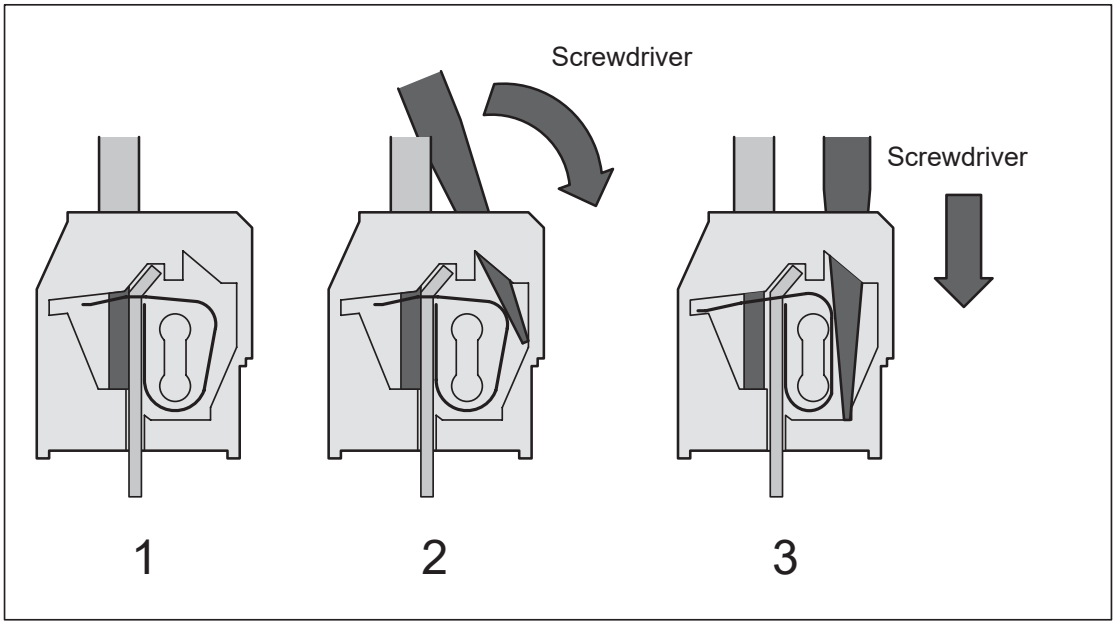


Fig. 1083: Disconnect wire from the spring terminal (steps 1 to 3)

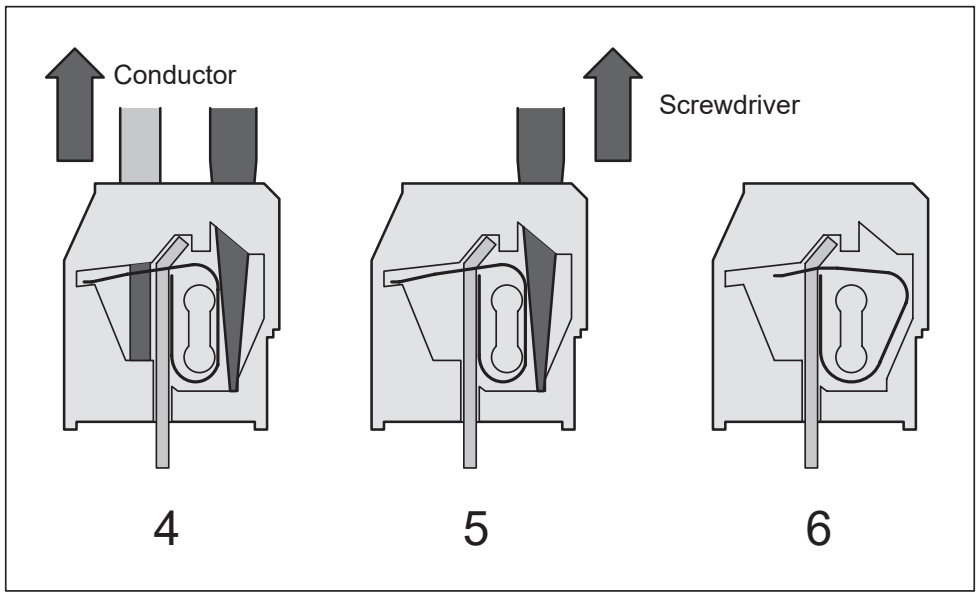


Fig. 1084: Disconnect wire from the spring terminal (steps 4 to 6)

1. Terminal with wire connected
2. Insert screwdriver (2.5 x 0.4 to 3.5 x 0.5 mm) at an angle, screwdriver must be at least 15 mm free of insulation at the tip
3. While erecting the screwdriver, insert it until the stop (requires a little strength) - terminal is now open
4. Remove wire from the open terminal
5. Done

Terminals for CANopen/DeviceNet communication modules

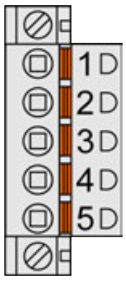


Fig. 1085: Combicon, 5-pole, female, removable plug with spring terminals

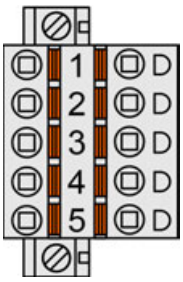


Fig. 1086: Combicon, 5-pole, female, removable plug with spring terminals

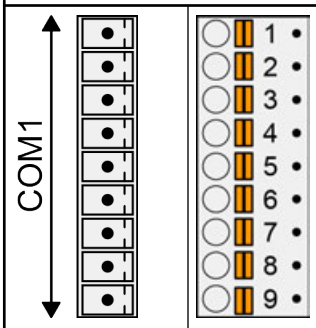
Terminal type: Spring terminal

Number of cores per terminal	Conductor type	Cross section	Stripped conductor end
1	solid	0.2 mm^2 to 2.5 mm^2	10 mm
1	flexible	0.2 mm^2 to 2.5 mm^2	10 mm
1 with wire-end ferrule (without plastic sleeve)	flexible	0.25 mm^2 to 2.5 mm^2	10 mm
1 with wire-end ferrule (with plastic sleeve)	flexible	0.25 mm^2 to 2.5 mm^2	10 mm

Serial interface COM1 of the terminal bases

The serial interface COM1 is connected via a removable 9-pin terminal block. It is configurable for RS-232 or RS-485 and can be used for:

- Online access (not valid for PM56xx),
- A free protocol,
- Modbus RTU, client and server,
- CS31 bus, as master only (not valid for PM56xx) *Chapter 1.6.3.6.4.8 “CS31 bus” on page 5347*.



Terminal block removed

Terminal block inserted

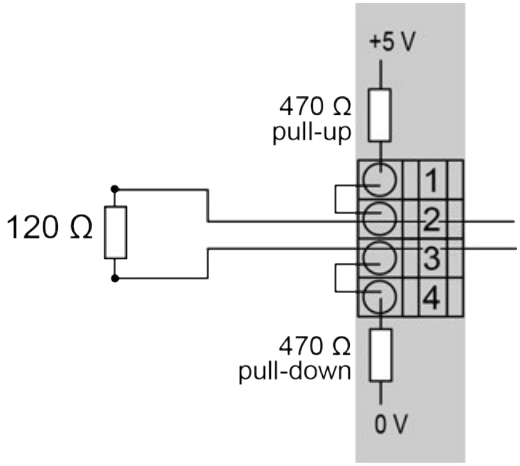
Pin	Signal	Interface	Description
1	Terminator P	RS-485	Terminator P
2	RxD/TxD-P	RS-485	Receive/Transmit, positive
3	RxD/TxD-N	RS-485	Receive/Transmit, negative
4	Terminator N	RS-485	Terminator N
5	RTS	RS-232	Request to send (output)
6	TxD	RS-232	Transmit data (output)
7	SGND	Signal Ground	
8	RxD	RS-232	Receive data (input)
9	CTS	RS-232	Clear to send (input)

RS-485 bus

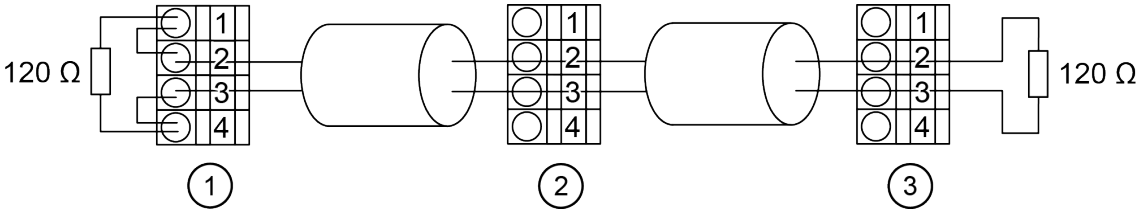
If the RS-485 bus is used, each interconnected bus line (each bus segment) must be electrically terminated. The following is necessary:

- 2 resistors of 120 Ω each at both line ends (to avoid signal reflections)
- Pull-up resistor at RxD/TxD-P and a pull-down resistor at RxD/TxD-N. These 2 resistors care for a defined high level on the bus, while there is no data exchange.

It is useful, to activate both the pull-up and the pull-down resistors, which only are necessary once on every bus line, at the bus master. For this reason, these two resistors are already integrated within the COM1 interface of the AC500 terminal bases. They can be activated by connecting the terminals 1-2 and 3-4 of COM1.

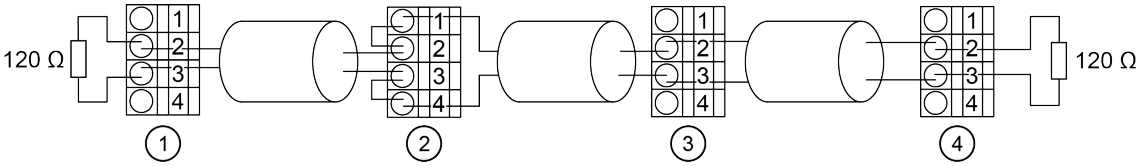


The following drawing shows an RS-485 bus with the bus master at the line end.



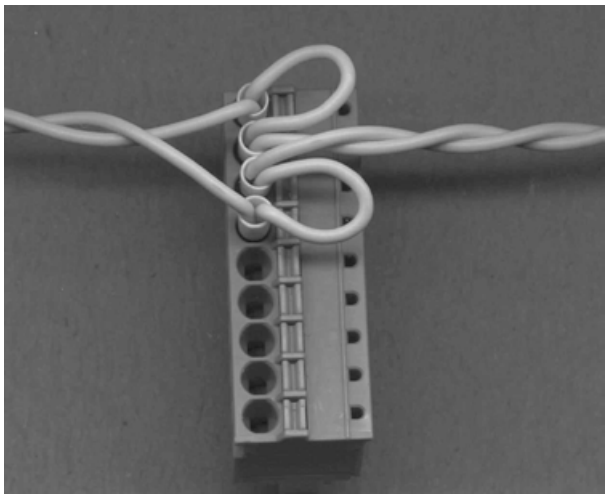
1 Master at the bus line end, pull-up and pull-down activated, bus termination with 120 Ω resistors
2 Slave within the bus line
3 Slave at the bus line end, bus termination with 120 Ω resistors

If the master is located within the bus line, it does not need a terminating resistor. The pull-up and the pull-down resistors, however, must be activated (see the following drawing).



1 Slave at the bus line end, bus termination with 120 Ω resistors
2 Master within the bus line, pull-up and pull-down activated
3 Slave within the bus line
4 Slave at the bus line end, bus termination with 120 Ω resistors

The following photo shows a wiring example "master within the bus line", wired at the COM1 bus connector of the terminal base:



If the bus is operated with several masters, the pull-up and pull-down resistors may only be activated at one master.

The grounding of the cable shields of the bus lines are described in the CS31 bus (PM57x, PM58x and PM59x) ↳ *Chapter 1.6.3.6.4.8 “CS31 bus” on page 5347*.

Serial interface COM2 of the terminal bases

The serial interface COM2 is not available at:

– *Processor modules with type designator -2ETH (e. g. PM591-2ETH)*
– *Processor modules PM56xx*

The serial interface COM2 is connected via a 9-pole D-sub connector. It is not intended to use COM2 to establish a CS31 system bus. It is configurable for RS-232 or RS-485 and can be used for

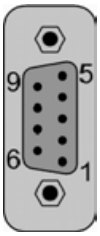
- online access
- a free protocol
- Modbus RTU, master and slave

If the RS-485 bus is used, each interconnected bus line (each bus segment) must be electrically terminated. The following is necessary:

- 2 resistors of 120 Ω each at both line ends (to avoid signal reflections)
- a pull-up resistor at RxD/TxD-P and a pull-down resistor at RxD/TxD-N. These 2 resistors care for a defined high level on the bus, while there is no data exchange.

It is useful, to activate both the pull-up and the pull-down resistors, which only are necessary once on every bus line, at the bus master.

Pin assignment

Serial Interface	Pin	Signal	Interface	Description	
	1	FE	-	Functional earth	
	2	TxD	RS-232	Transmit data	Output
	3	RxD/TxD-P	RS-485	Receive/Transmit	Positive
	4	RTS	RS-232	Request to send	Output
	5	SGND	Signal ground	0 V supply out	
	6	+5 V	-	5 V supply out	
	7	RxD	RS-232	Receive data	Input
	8	RxD/TxD-N	RS-485	Receive/Transmit	Negative
	9	CTS	RS-232	Clear to send	Input
	Shield	FE	-	Functional earth	

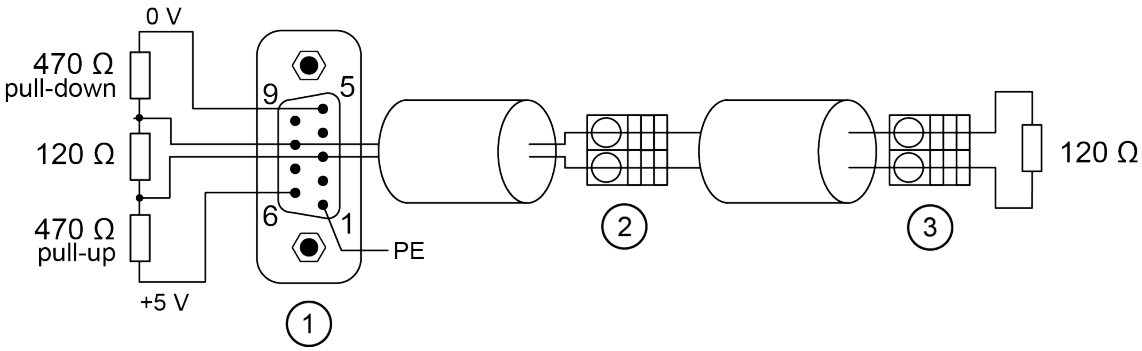
NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

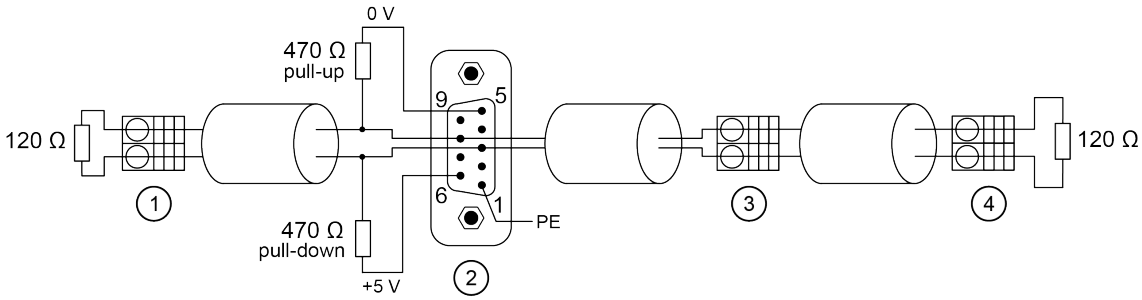
Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

The following drawing shows an RS-485 bus with the bus master at the line end.



1 Master at the bus line end, pull-up and pull-down activated, bus termination with 120 Ω resistors
2 Slave within the bus line
3 Slave at the bus line end, bus termination with 120 Ω resistors

If the master is located within the bus line, it does not need a terminating resistor. The pull-up and the pull-down resistors, however, are necessary:



1 Slave at the bus line end, bus termination with 120 Ω resistors
2 Master within the bus line, pull-up and pull-down activated
3 Slave within the bus line
4 Slave at the bus line end, bus termination with 120 Ω resistors

NOTICE!

If the bus is operated with several masters, the pull-up and pull-down resistors may only be installed at one master.

The cable shields must be earthed. See CS31 system bus *Chapter 1.6.3.5.4.4 “CS31 bus” on page 5257*.

CS31 bus

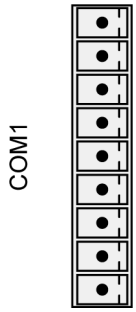
Connection of the processor module to the CS31 bus

*The PM56xx processor module does **not** support the CS31 bus.*

COM1 of the terminal base

The processor module can be used as a CS31 bus master. The connection is performed via the serial interface COM1 used as a CS31 bus.

Pin assignment (RS-485 / RS-232)



Terminal block removed

Terminal block inserted

Pin	Signal	Interface	Description
1	Terminator P	RS-485	Terminator P
2	RxD/TxD-P	RS-485	Receive/Transmit, positive
3	RxD/TxD-N	RS-485	Receive/Transmit, negative
4	Terminator N	RS-485	Terminator N
5	RTS	RS-232	Request to send (output)
6	TxD	RS-232	Transmit data (output)
7	SGND	Signal Ground	Signal Ground
8	RxD	RS-232	Receive data (input)
9	CTS	RS-232	Clear to send (input)

NOTICE!

Unused connector!

Make sure that the terminal block is always connected to the terminal base or communication module, even if you do not use the interface.

With connecting the terminals 1-2 and 3-4, a pull-up and a pull-down resistor can be activated (see chapter Serial Interface COM1 *Chapter 1.6.3.6.4.6 “Serial interface COM1 of the terminal bases” on page 5343*.

Wiring

Wiring

Bus line	
Construction	2 cores, twisted, with common shield
Conductor cross section	> 0.22 mm² (24 AWG)
Recommendation	0.5 mm² corresponds to 0.8 mm
Twisting rate	> 10 per meter (symmetrically twisted)
Core insulation	Polyethylene (PE)

Bus line	
Resistance per core	< 100 Ω/km
Characteristic impedance	ca. 120 Ω (100 Ω...150 Ω)
Capacitance between the cores	< 55 nF/km (if higher, the max. bus length must be reduced)
Terminating resistors	120 Ω ¼ W at both line ends
Remarks	Shielded cables with PVC core insulation and a core diameter of 0.8 mm can be used up to a length of ca. 50 m. In this case, the bus terminating resistor is ca. 100 Ω.

Remarks:

Cables with PVC core insulation and a core diameter of 0.8 mm can be used up to a length of ca. 250 m. In this case, the bus terminating resistor is ca. 100 Ω.

Cables with PE core insulation can be used up to a length of ca. 500 m.

Bus topology

A CS31 bus always contains only one bus master (CPU or communication module) which controls all actions on the bus. Up to 31 slaves can be connected to the bus, e.g. remote modules or slave-configured CPUs. Besides the wiring instructions shown below, the wiring and grounding instructions provided with the descriptions of the modules are valid additionally.

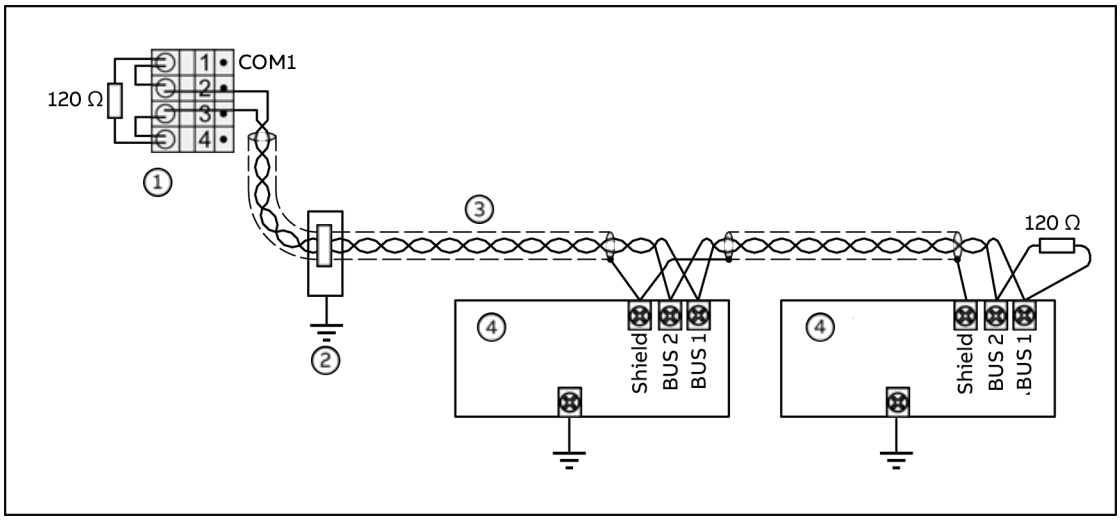


Fig. 1087: Bus topology for a CS31 bus at COM1 (bus master at one end of the bus line)

1 CS31 bus master (e.g. PM581, master at the bus line end, pull-up and pull-down activated, bus termination 120 Ω)
2 Direct earthing with clip on cabinet steel plate
3 CS31 bus
4 CS31 slave

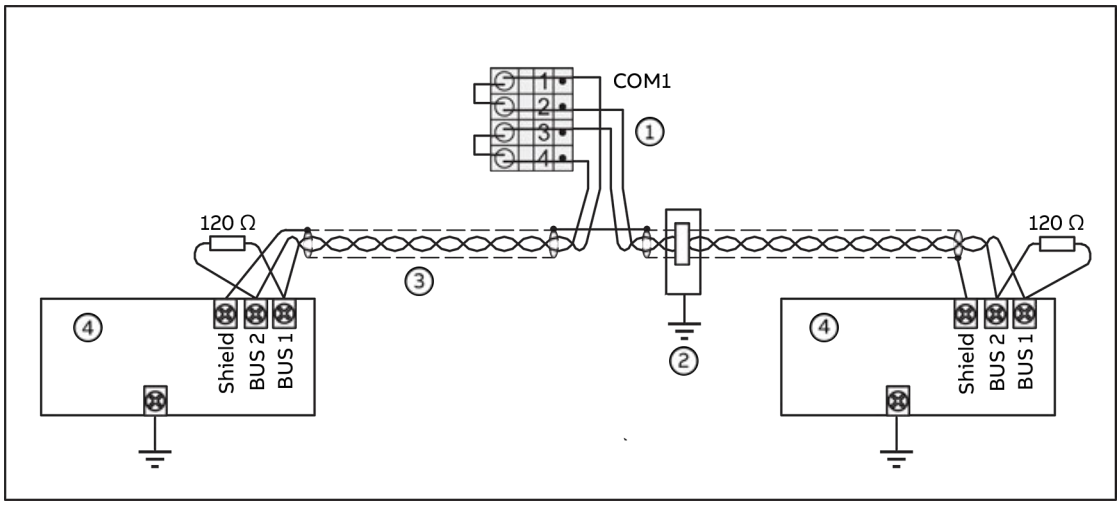


Fig. 1088: Bus topology for a CS31 bus at COM1 (bus master within the bus line)

1 CS31 bus master (e.g. PM581, master at the bus line end, pull-up and pull-down activated, bus termination 120 Ω)
2 Direct earthing with clip
3 CS31 bus
4 CS31 slave

NOTICE!
Risk of malfunctions!
Spur lines are not allowed within the CS31 bus.
Loop the bus line from module to module.

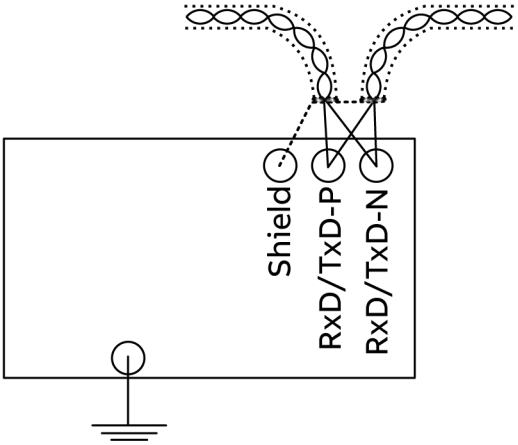


Fig. 1089: CS31 slave - Bus line: Correct

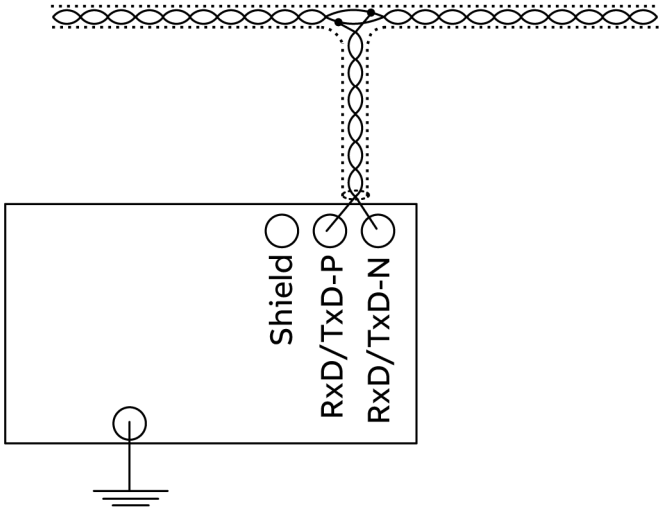


Fig. 1090: CS31 slave - Bus line: Wrong

Grounding

In order to avoid disturbance, the cable shields must be grounded directly.

Case a:

Multiple switchgear cabinets: If it can be guaranteed that no potential differences can occur between the switchgear cabinets by means of current-carrying metal connections (grounding bars, steel constructions etc.), the direct grounding is chosen.

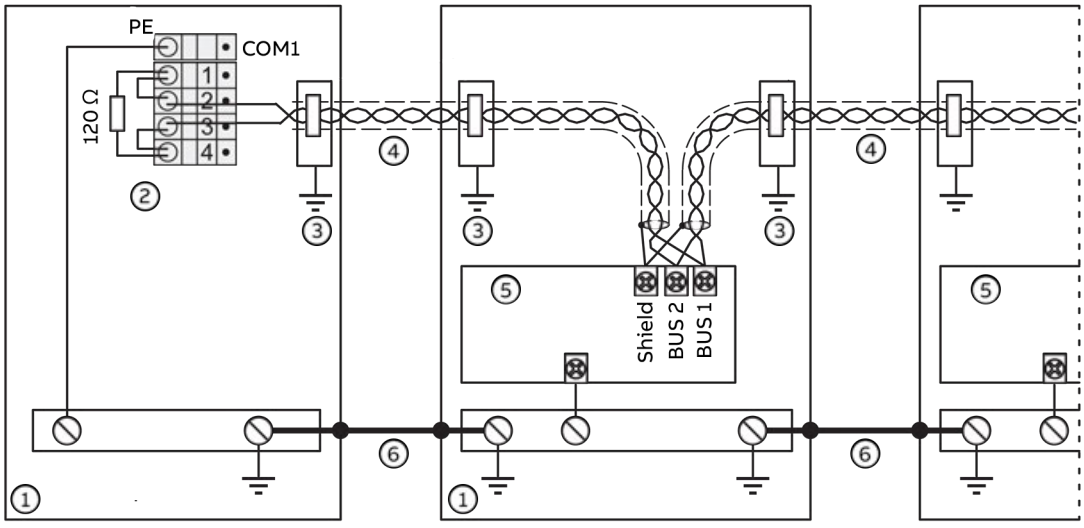


Fig. 1091: Direct grounding

1 Cabinet
2 CS31 bus master (e.g. PM581)
3 Direct grounding of shields when entering the cabinet
4 CS31 bus system
5 CS31 slave
6 Current-carrying connection

Case b:

Multiple switchgear cabinets: If potential differences can occur between the switchgear cabinets, the capacitive grounding method is chosen in order to avoid circulating currents on the cable shields.

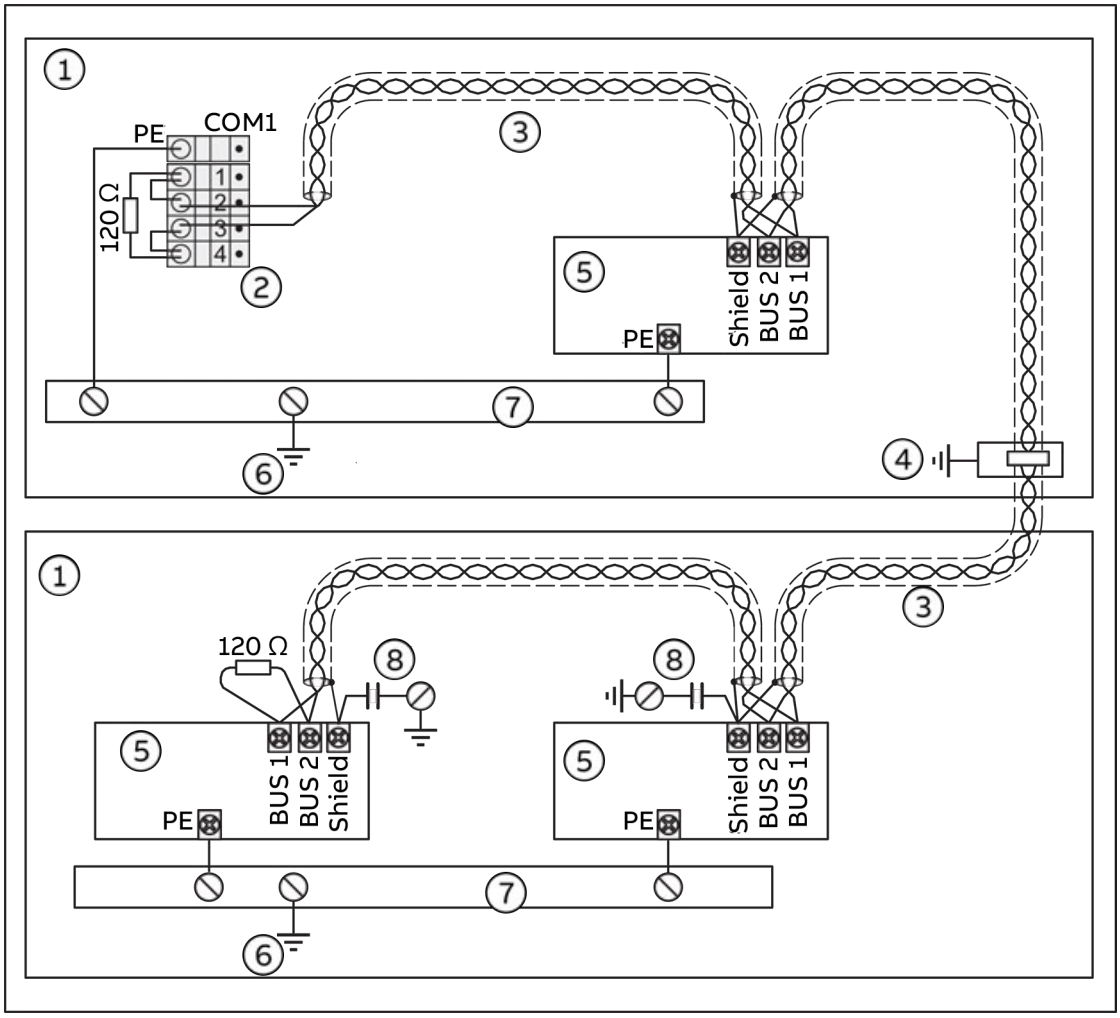


Fig. 1092: Earthing concept with several switchgear cabinets: direct grounding of cable shields when cables enter the first switchgear cabinet (containing the master), and capacitive grounding at the modules

1 Cabinet
2 CS31 bus master e.g. PM581
3 CS31 bus system
4 Direct grounding of shields when entering the cabinet
5 CS31 slave
6 Cabinet grounding
7 Grounding bar
8 Capacitive grounding 0.1 uF X-type capacitor directly on on the cabinet's steel plate

Everywhere is valid: The total length of the grounding connections between the shield of the Terminal Base and the grounding bar must be as short as possible (max. 25 cm). The conductor cross section must be at least 2.5 mm^2.

VDE 0160 requires, that the shield must be grounded directly at least once per system.

CANopen field bus

Types of bus cables

For CANopen, only bus cables with characteristics as recommended in ISO 11898 are to be used. The requirements for the bus cables depend on the length of the bus segment. Regarding this, the following recommendations are given by ISO 11898:

Length of segment [m]	Bus cable (shielded, twisted pair)			Max. transmission rate [kbit/s]
	Conductor cross section [mm²]	Line resistance [Ω/km]	Wave impedance [Ω]	
0...40	0.25...0.34 / AWG23, AWG22	70	120	1000 at 40 m
40...300	0.34...0.60 / AWG22, AWG20	< 60	120	< 500 at 100 m
300...600	0.50...0.60 / AWG20	< 40	120	< 100 at 500 m
600...1000	0.75...0.80 / AWG18	< 26	120	< 50 at 1000 m

NOTICE!

Risk of telegram and data errors!

The use of wrong cable type and quality could lead to limitations in cable length, causing telegram and data errors.

NOTICE!

Risk of damaging the terminating resistor!

A bus-line short-circuit to the 24 V DC power supply can cause damage by exceeding the power rating of the terminating resistor.

NOTICE!

Risk of telegram and data errors!

Miss- or unterminated data lines can cause reflections on the bus, leading to telegram and data errors. For maximum cable length and transmission rate, the bus must always be terminated on both ends with the characteristic impedance of the cable type.

NOTICE!

Verification of termination (Make sure the power supply on all CAN nodes is turned off)!

To verify the termination, the DC resistance between CAN_H and CAN_L can be measured. The value should be between 50 Ω and 70 Ω.

Check for correct resistor values, short circuits and correct number of terminating resistors, if the measurement is showing deviations.

Installation hint

Ensure that the termination and FE connection will not be removed when removing CAN modules from the bus.

Branches are not allowed in a CAN network. Stubs should be avoided or kept as short as possible (< 0.3 m).

When connecting the cable take care to use one dedicated twisted pair for the CAN signals (CAN_L and CAN_H) and another free wire for CAN_GND. CAN_GND must be connected as reference, to avoid common mode problems causing telegram errors.

Keep the CAN bus wiring away from electrical disturbance and close to earth potential to minimize interference.

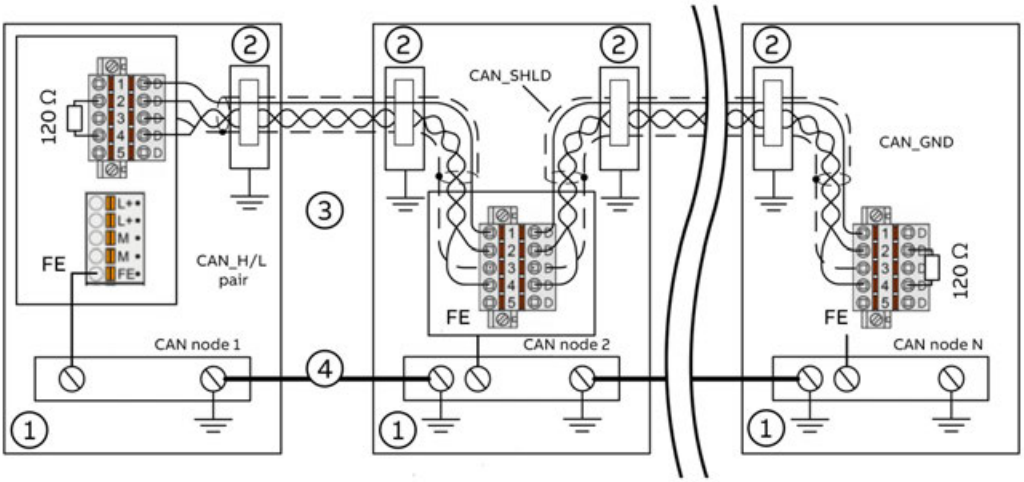


Fig. 1093: CAN bus, connection and wiring

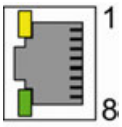
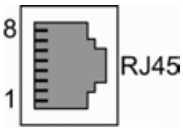
1 Cabinet
2 Direct earthing of shields when entering the cabinet
3 CAN bus segment
4 Current-carrying connection

Ethernet connection details

Ethernet is also used for PROFINET, EtherCAT and Modbus TCP connection.

Ethernet interface

Pin assignment

Interface	Pin	Signal	Description
 or 	1	TxD+	Transmit data +
	2	TxD-	Transmit data -
	3	RxD+	Receive data +
	4	NU	Not used
	5	NU	Not used
	6	RxD-	Receive data -
	7	NU	Not used
	8	NU	Not used
	Shield	Cable shield	Functional earth

See supported protocols and used Ethernet ports: *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

See communication via Modbus TCP/IP: *Chapter 1.6.4.1.9 “Communication with Modbus TCP/IP” on page 5488*.

See communication via Modbus RTU: *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

Wiring

Cable length restrictions

For the maximum possible cable lengths within an Ethernet network, various factors have to be taken into account. Twisted pair cables (TP cables) are used as transmission medium for 10 Mbit/s Ethernet (10Base-T) as well as for 100 Mbit/s (Fast) Ethernet (100Base-TX). For a transmission rate of 10 Mbit/s, cables of at least category 3 (IEA/TIA 568-A-5 Cat3) or class C (according to European standards) are allowed. For fast Ethernet with a transmission rate of 100 Mbit/s, cables of category 5 (Cat5) or class D or higher have to be used. The maximum length of a segment, which is the maximum distance between two network components, is restricted to 100 m due to the electric properties of the cable.

Furthermore, the length restriction for one collision domain has to be observed. A collision domain is the area within a network which can be affected by a possibly occurring collision (i.e. the area the collision can propagate over). This, however, only applies if the components operate in half-duplex mode since the CSMA/CD access method is only used in this mode. If the components operate in full-duplex mode, no collisions can occur. Reliable operation of the collision detection method is important, which means that it has to be able to detect possible collisions even for the smallest possible frame size of 64 bytes (512 bits). But this is only guaranteed if the first bit of the frame arrives at the most distant subscriber within the collision domain before the last bit has left the transmitting station. Furthermore, the collision must be able to propagate to both directions at the same time. Therefore, the maximum distance between two ends must not be longer than the distance corresponding to the half signal propagation time of 512 bits. Thus, the resulting maximum possible length of the collision domain is 2000 m for a transmission rate of 10 Mbit/s and 200 m for 100 Mbit/s. In addition, the bit delay times caused by the passed network components also have to be considered.

The following table shows the specified properties of the respective cable types per 100 m.

Table 615: Specified cable properties:

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Attenuation [dB / 100m]	10.7	23.2
NEXT [dB / 100m]	23	24
ACR [dB / 100m]	N/A	4

Parameter	10Base-T [10 MHz]	100Base-TX [100 MHz]
Return loss [dB / 100m]	18	10
Wave impedance [Ohms]	100	100
Category	3 or higher	5
Class	C or higher	D or higher

TP cable

The TP cable has eight wires arranged in four pairs of twisted wires. Different color codes exist for the coding of the wires, the coding according to EIA/TIA 568, version 1, being the one most commonly used. In this code, the individual pairs are coded with blue, orange, green and brown color. One wire of a pair is unicolored and the corresponding second wire is striped, the respective color alternating with white. For shielded cables, a distinction is made between cables that have one single shield around all pairs of wires and cables that have an additional individual shield for each pair of wires. The following table shows the different color coding systems for TP cables:

Table 616: Color coding of TP cables:

Pairs	EIA/TIA 568 Version 1		EIA/TIA 568 Version 2		DIN 47100		IEC 189.2	
Pair 1	white/ blue	blue	green	red	white	brown	white	blue
Pair 2	white/ orange	orange	black	yellow	green	yellow	white	orange
Pair 3	white/ green	green	blue	orange	grey	pink	white	green
Pair 4	white/ brown	brown	brown	slate	blue	red	white	brown

Two general variants are distinguished for the pin assignment of the normally used RJ45 connectors: EIA/TIA 568 version A and version B. The wiring according to EIA/TIA 568 version B is the one most commonly used.

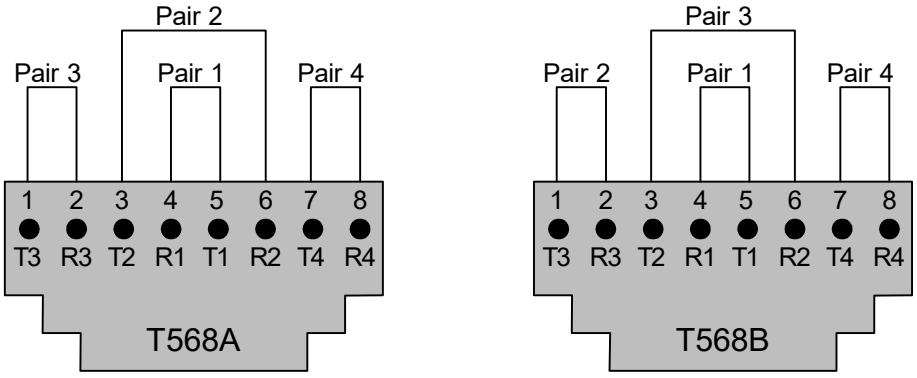


Fig. 1094: Pin assignment of RJ45 sockets

Cable types

Crossover cable

Particular use

Crossover cables are needed only for a direct Ethernet connection without crossover functionality. In particular for AC500 modules in product life cycle phase "Classic".

Crossover cables are for a direct Ethernet connection of two terminal devices as the simplest variant of a network. From transmission lines of the first station to the reception lines of the second station.

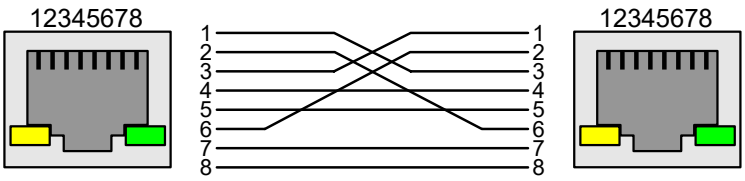


Fig. 1095: Wiring of a crossover cable

Straight-through cable

For networks with more than two subscribers, hubs or switches have to be used additionally for distribution. These active devices already have the crossover functionality implemented which allows a direct connection of the terminal devices using straight-through cables.

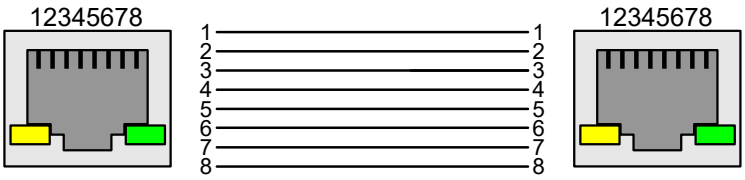


Fig. 1096: Wiring of a straight-through cable

CAUTION!

Risk of communication faults!

When using inappropriate cables, malfunctions in communication may occur.

Only use network cables of the categories 5 (Cat 5, Cat 5e, Cat 6 or Cat 7) or higher within PROFINET networks.

PROFIBUS connection details

Attachment plug for the bus cable

9-pin D-sub connector, male

Parameter	Value
Fastening torque	0.4 Nm

Assignment

Pin	Signal	Description
1	Shield	Shielding, protective ground
2	not used	-
3	RxD/TxD-P	Reception / transmission line, positive
4	CBTR-P	Control signal for repeater, positive (optional)
5	DGND	Reference potential for data lines and +5 V
6	VP	+5 V, supply voltage for bus terminating resistors
7	not used	-

Pin	Signal	Description
8	RxD/TxD-N	Reception / transmission line, negative
9	CNTR-N	Control signal for repeater, negative (optional)

Bus cable

Parameter	Value
Type	Twisted pair (shielded)
Characteristic impedance	135 Ω...165 Ω
Cable capacitance	< 30 pF/m
Conductor diameter of the cores	≥ 0.64 mm
Conductor cross section of the cores	≥ 0.34 mm²
Cable resistance per core	≤ 55 Ω/km
Loop resistance (resistance of two cores)	≤ 110 Ω/km

Cable lengths

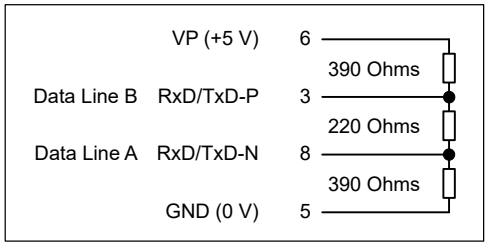
The maximum possible cable length of a PROFIBUS subnet within a segment depends on the tranmission rate (baud rate).

Transmission Rate	Maximum Cable Length
9.6 / 19.2 / 93.75 kBaud	1200 m
187.5 kBaud	1000 m
500 kBaud	400 m
1.5 MBaud	200 m
3 MBaud to 12 MBaud	100 m

Branch lines are generally permissible for transmission rates of up to 1500 kbit/s. But in fact they should be avoided for transmission rates higher than 500 kbit/s.

Bus terminating resistors

The line ends (of the bus segments) have to be terminated using bus terminating resistors according to the drawing below. The bus terminating resistors are usually placed inside the bus connector.



Repeaters

One bus segment can have up to 32 subscribers. Using repeaters a system can be expanded to up to 126 subscribers. Repeaters are also required for longer transfer lines. Please note that a repeater's load to the bus segment is the same as the load of a normal bus subscriber. The sum of normal bus subscribers and repeaters in one bus segment must not exceed 32.

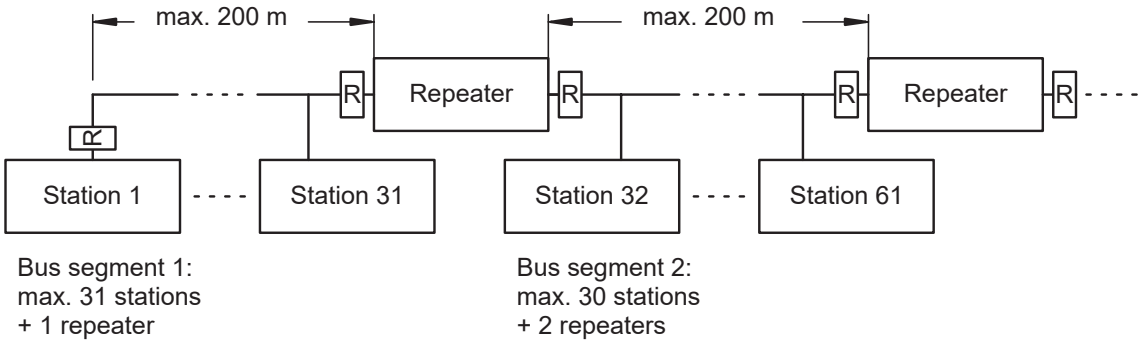


Fig. 1097: Principle example for a PROFIBUS-DP system with repeaters (1500 kbit/s baud rate)

Modbus RTU connection details

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Available serial interfaces can work as Modbus interfaces simultaneously.

The Modbus client operating mode of an interface is set with the function block COM_MOD_MAST.

Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698

Technical data

The Modbus operating mode and the interface parameters are set in the *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*.

Table 617: Description of the Modbus protocol

Parameter	Value
Supported standard	PM55x and PM56x: EIA RS-485 PM57x, PM58x and PM59x: EIA RS-232 / RS-485
Number of connection points	1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol	Modbus
Operating mode	Client/server
Address	Server only
Data transmission control	CRC16
Data transmission speed	From 300 bits/s to 187,500 bits/s
Encoding	1 start bit 8 data bits 1 parity bit, (optional) even, odd, mark or space 1 or 2 stop bits
Max. cable length for RS-485 on COM1 / COM2 for AC500 CPU	1.200 m at 19.200 baud

Parameter			Value
Max. cable length for RS-485 on COM1 / COM2 for AC500-eCo CPU			
	COM1:		
		Non-isolated:	Max. 50 m (with shielded cable)
		Isolated with TK506:	Max. 500 m at 19.200 baud (with shielded cable *)
	COM2:		
		Non-isolated with TA562:	Max. 50 m (with shielded cable)
		Isolated with TA569:	Max. 500 m at 19.200 baud (with shielded cable *)

*) 500 m cable type STP-120 Ω/AWG-20

If a processor module provides more than one serial interface, both interfaces (COM1/COM2) can be operated simultaneously as Modbus interfaces and can operate as Modbus server as well as Modbus client.

Bus topology Point-to-point with RS-232 or bus topology with RS-485. Modbus is a master-slave protocol.

For further information on Modbus see chapter *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467*.

1.6.3.6.5 Handling of accessories

This section only describes accessories that are frequently used for system assembly, connection and construction. A description of all additional accessories that can be used to supplement AC500 system can be found in the Hardware PLC device description.

MC502 - Memory card

- Solid state flash memory storage

1 MC502 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

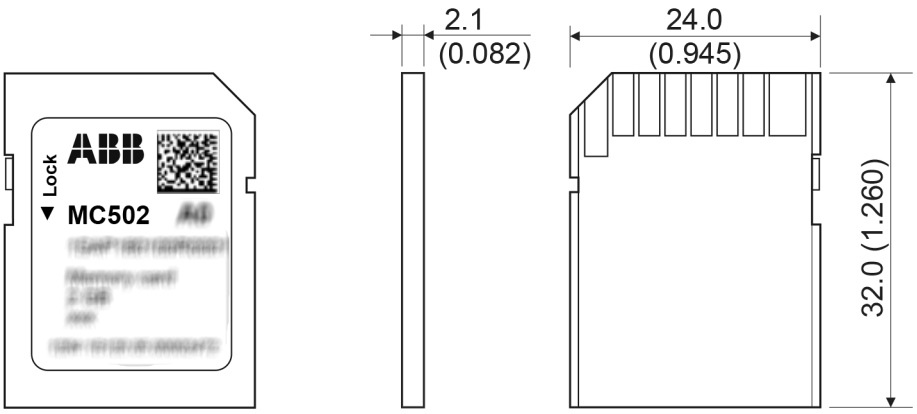
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

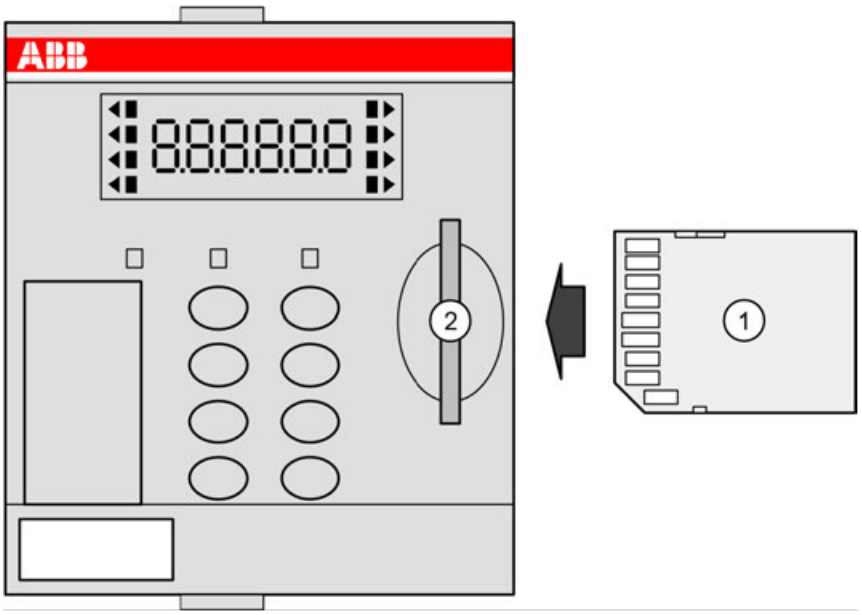


Fig. 1098: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

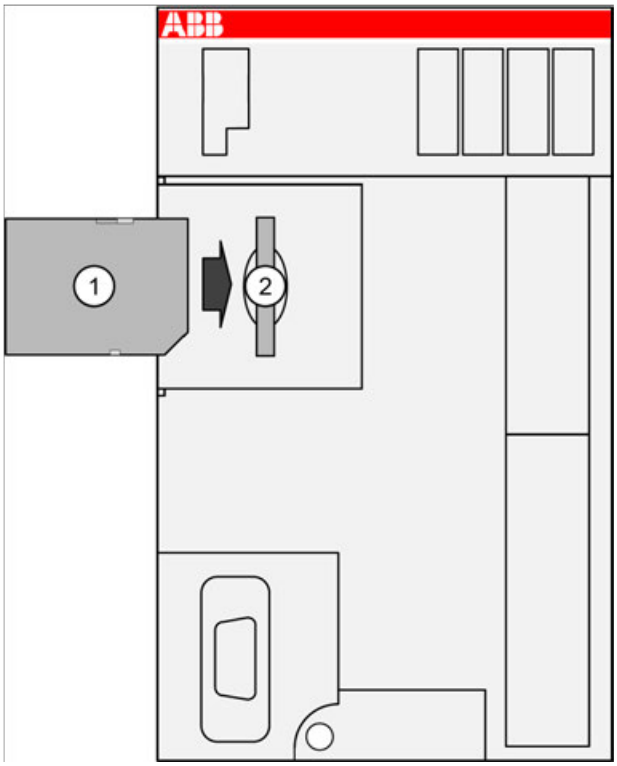


Fig. 1099: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

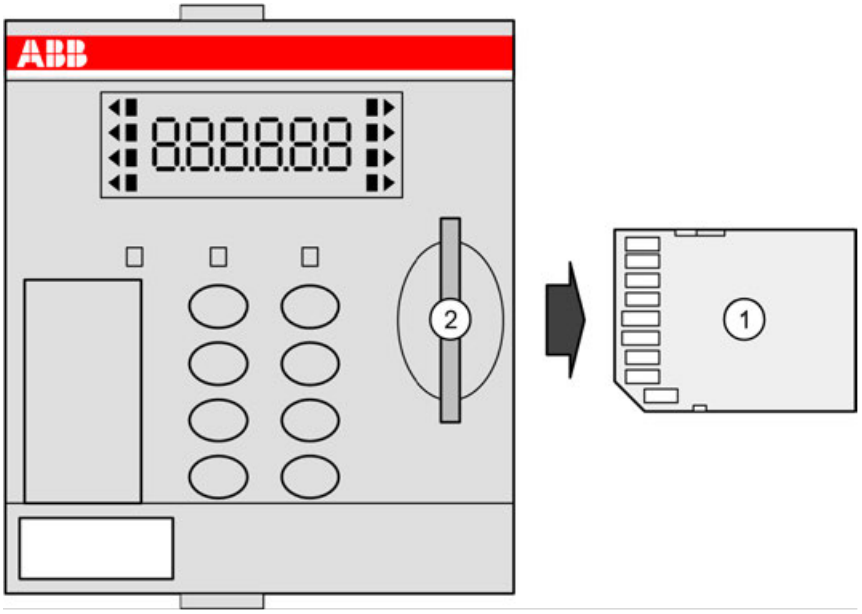


Fig. 1100: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

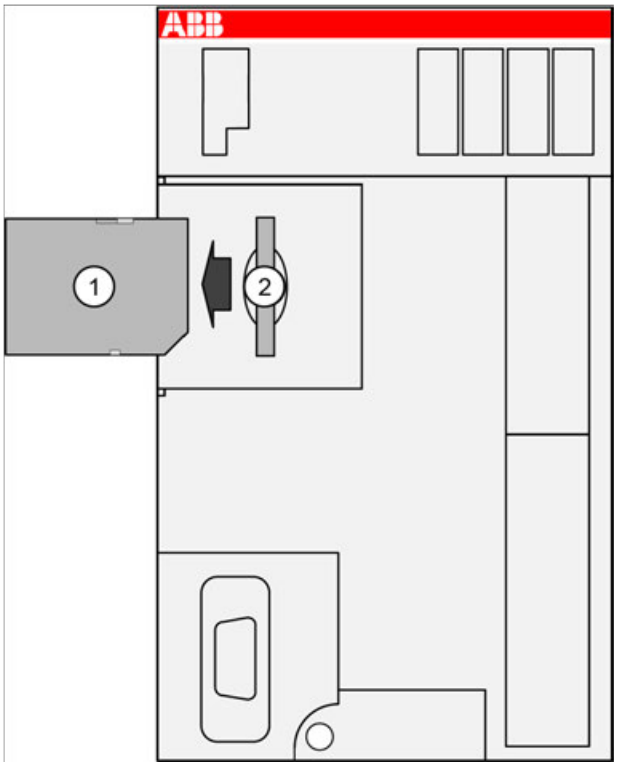


Fig. 1101: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter	Value
Memory capacity	2 GB
Total bytes written (TBW)	On request

Parameter		Value
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
↳ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

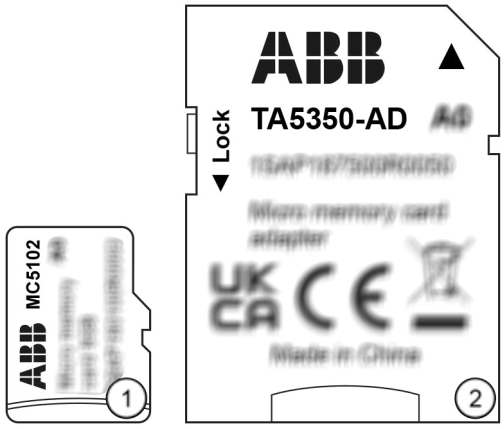
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0001	MC502, memory card	Classic

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

MC5102 - Micro memory card with micro memory card adapter

- Solid state flash memory storage



1 Micro memory card
2 TA5350-AD micro memory card adapter

The MC5102 micro memory card has no write protect switch.

The TA5350-AD micro memory card adapter has a write protect switch.

In the position "LOCK", the inserted micro memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other micro memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

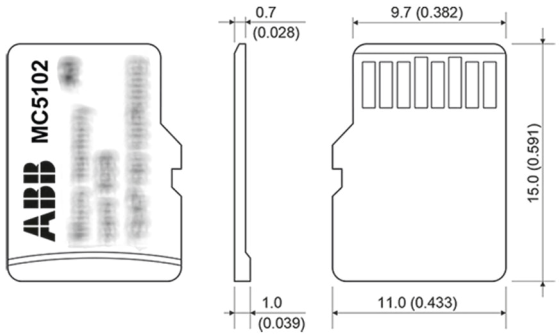
The micro memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The micro memory card can only be used temporarily in standard and XC applications.

The memory card can be read/written on a PC with a SDHC compatible memory card reader when using TA5350-AD micro memory card adapter.

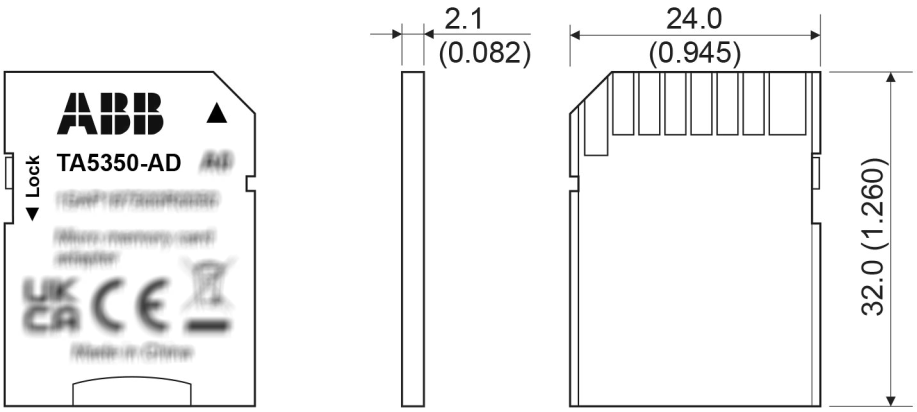
Dimensions

Micro memory card



The dimensions are in mm and in brackets in inch.

Micro memory card adapter



The dimensions are in mm and in brackets in inch.

Insert the micro memory card

AC500 V2 and AC500-eCo V2

1. Unpack the micro memory card and insert it into the supplied micro memory card adapter.
2. Insert the micro memory card adapter with integrated micro memory card into the memory card slot of the processor module until locked.

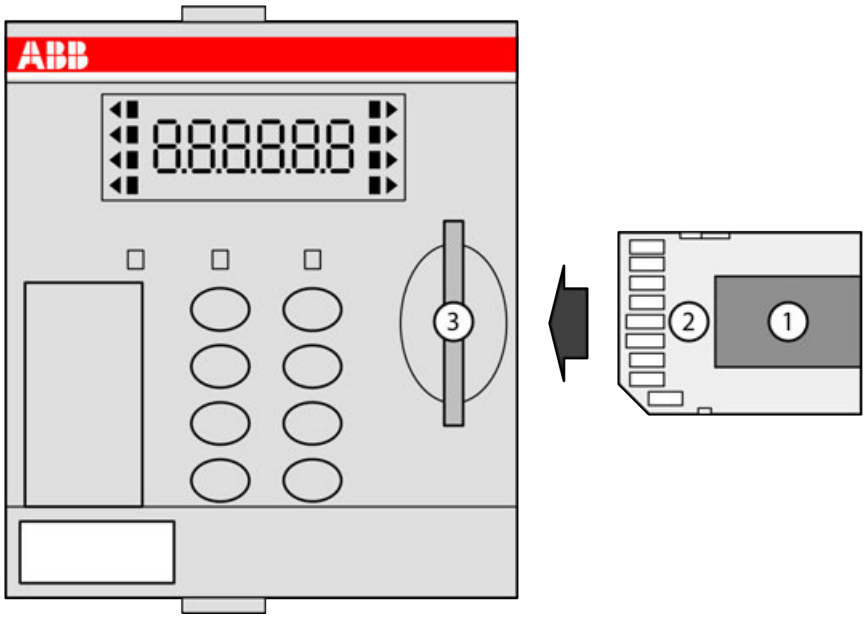


Fig. 1102: Insert micro memory card into PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

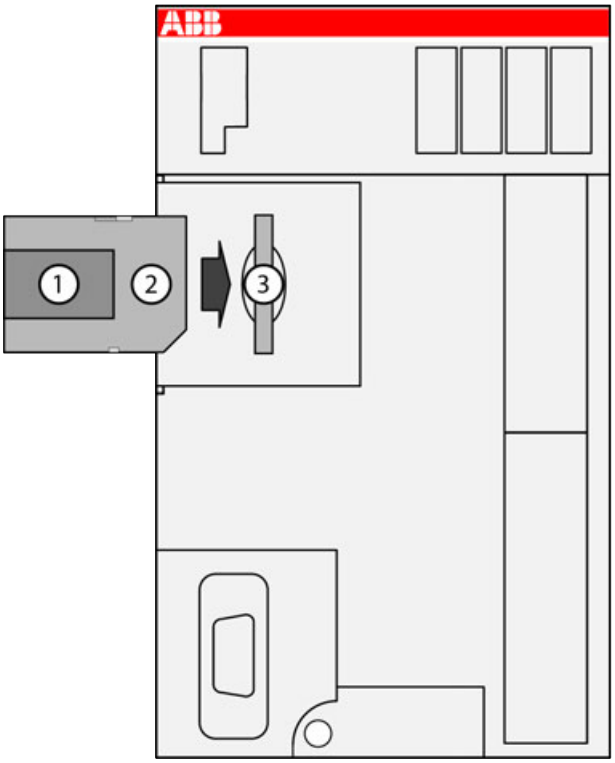


Fig. 1103: Insert micro memory card into PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Remove the micro memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the micro memory card

Do not remove the micro memory card when it is working!

Remove the micro memory card with micro memory card adapter only when the RUN LED is not blinking.

Otherwise the micro memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the micro memory card adapter with the integrated micro memory card, push on the micro memory card adapter until it moves forward.
2. By this, the micro memory card adapter is unlocked and can be removed.

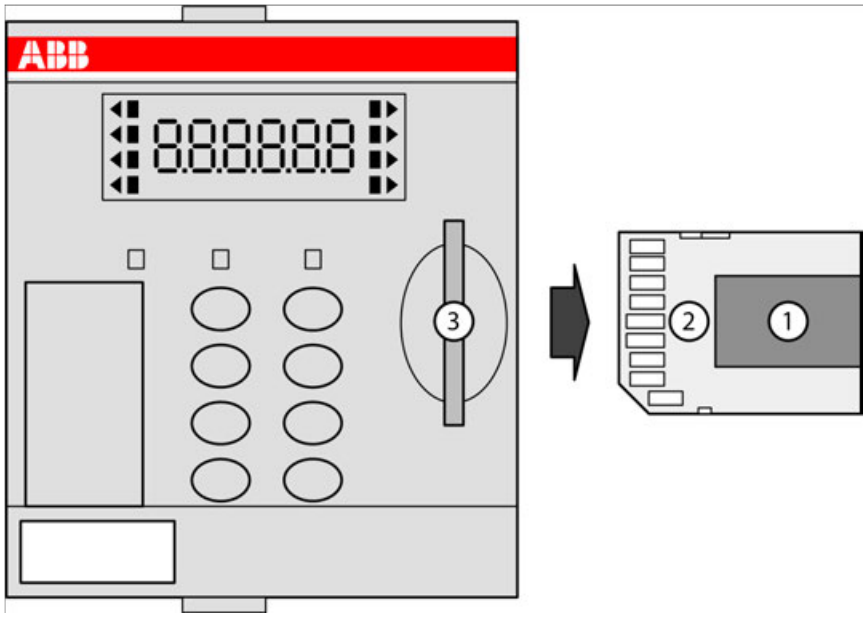


Fig. 1104: Remove micro memory card from PM57x, PM58x and PM59x

1 Micro memory card
2 Micro memory card adapter
3 Memory card slot

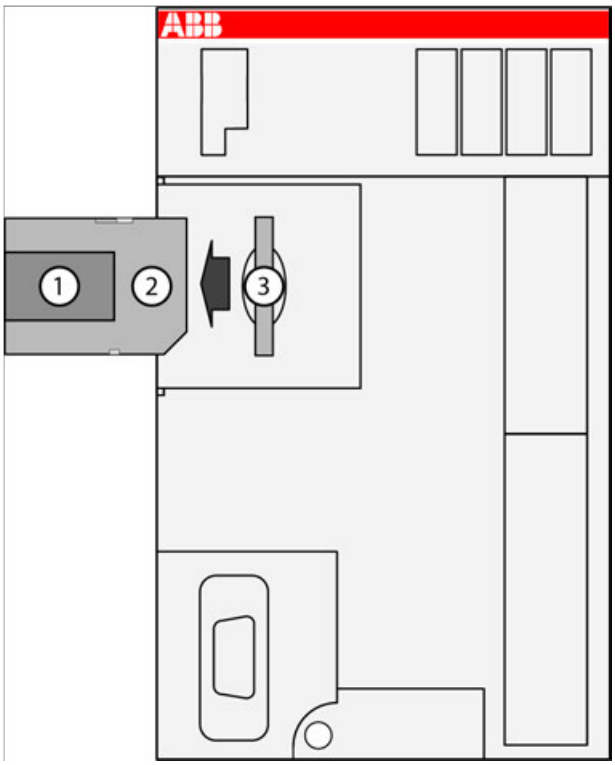


Fig. 1105: Remove micro memory card from PM55x-xP and PM56x-xP

1 Micro memory card
2 Micro memory card adapter
3 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		8 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		
	Micro memory card	No
	Micro memory card adapter	Yes
Weight		0.25 g
Dimensions		15 mm x 11 mm x 0.7 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the micro memory card in AC500 PLCs is provided in the chapter ↳ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0002	MC5102, micro memory card with TA5350-AD micro memory card adapter	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

MC5141 - Memory card

- Solid state flash memory storage

1 MC5141 memory card

The memory card has a write protect switch.

In the position "LOCK", the memory card can only be read.

Memory card type	AC500 V2	AC500-XC V2	AC500-eCo V2 [3])	AC500 V3	AC500-XC V3	AC500-eCo V3
MC502	x	x	x	x	x	-
MC5141	x	x	x	x	x	-
MC5102 **with** TA5350-AD micro memory card adapter	x [1])	x [1]) [2])	x [1])	x	x [2])	-
MC5102 **without** TA5350-AD micro memory card adapter	-	-	-	-	-	x

[1]) As of firmware 2.5.x

[2]) Temporary use of MC5102 is possible under normal environmental conditions, but MC5141 should be preferred.

[3]) A memory card can only be inserted when a MC503 memory card adapter is installed in the processor module.

The use of other memory cards is prohibited. ABB is not responsible nor liable for consequences resulting from use of unapproved memory cards.

Purpose

Processor modules can be operated with and without (micro) memory card.

Processor modules are supplied without (micro) memory card. It must be ordered separately.

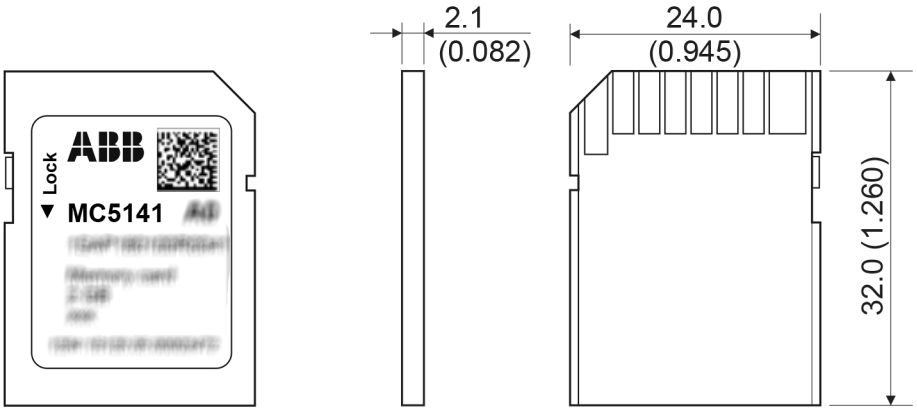
AC500-eCo V2 processor modules must be equipped with an MC503 memory card adapter if a memory card is used. *Chapter 1.6.3.5.5.3 “MC503 - Memory card adapter” on page 5272*

The memory card is used to store or backup application data and/or application programs or project source codes as well as to update the internal CPU firmware.

The memory card is intended for long-term use in standard and XC application.

The memory card can be read/written on a PC with a SDHC compatible memory card reader.

Dimensions



The dimensions are in mm and in brackets in inch.

Insert the memory card

AC500 V2 and AC500-eCo V2

1. Unpack the memory card.
2. Insert the memory card into the memory card slot of the processor module until locked.

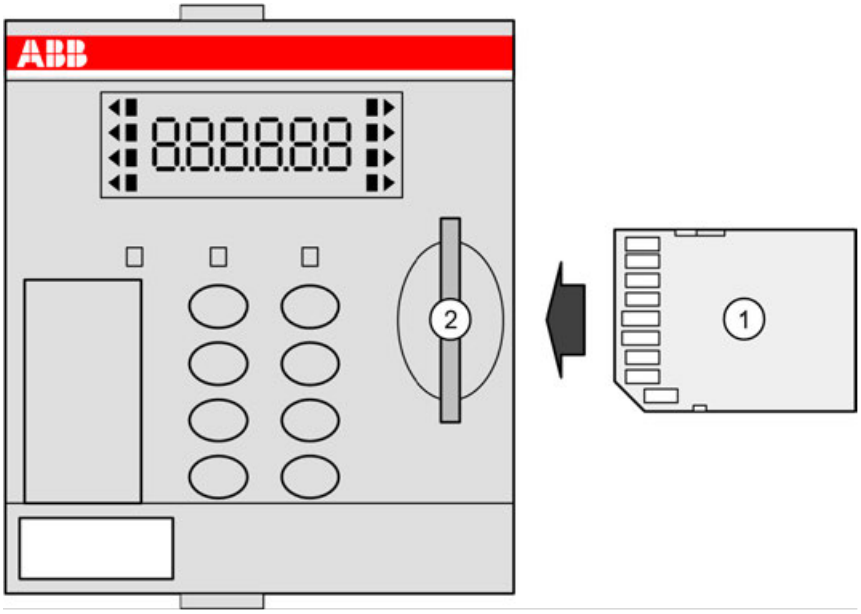


Fig. 1106: Insert memory card into PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

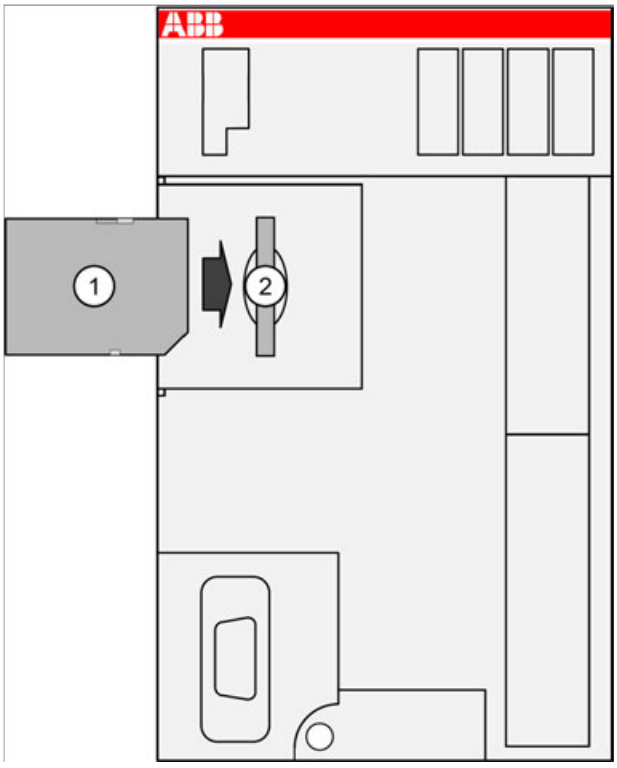


Fig. 1107: Insert memory card into PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Remove the memory card

AC500 V2 and AC500-eCo V2

NOTICE!

Removal of the memory card

Do not remove the memory card when it is working!

Remove the memory card only when the RUN LED is not blinking.

Otherwise the memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

1. To remove the memory card, push on the memory card until it moves forward.
2. By this, the memory card is unlocked and can be removed.

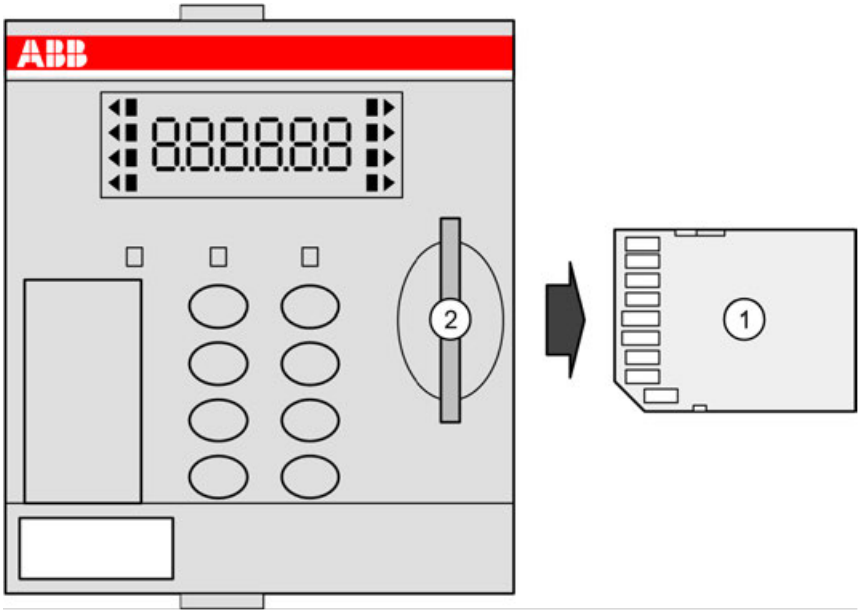


Fig. 1108: Remove memory card from PM57x, PM58x and PM59x

1 Memory card
2 Memory card slot

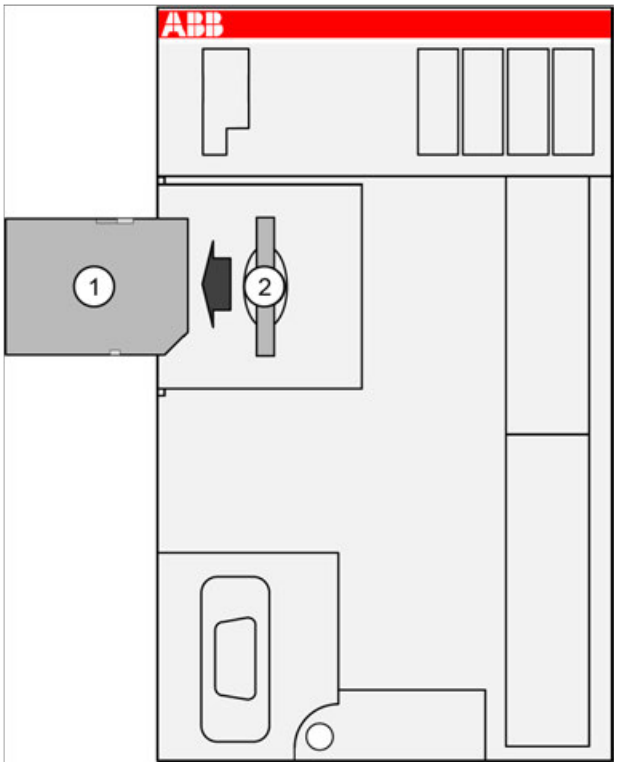


Fig. 1109: Remove memory card from PM55x-xP and PM56x-xP

1 Memory card
2 MC503 memory card adapter

Technical data

Parameter		Value
Memory capacity		2 GB
Total bytes written (TBW)		On request
Data retention		
	at beginning	10 years at 40 °C
	when number of write processes has been 90 % of lifetime of each cell	1 year at 40 °C
Write protect switch		Yes, at the edge of the memory card
Weight		2 g
Dimensions		24 mm x 32 mm x 2.1 mm

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

Further information on using the memory card in AC500 PLCs is provided in the chapter
↬ *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*.

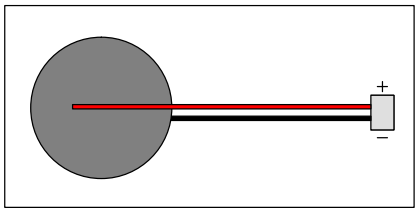
Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 100 R0041	MC5141, memory card	Active

*) *Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA521 - Battery

- Manganese dioxide lithium battery, 3 V, 560 mAh
- Non-rechargeable



Purpose

The TA521 battery is the only applicable battery for the AC500 processor modules *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*. It cannot be recharged.

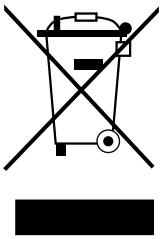
The processor modules are supplied without lithium battery. It must be ordered separately. The TA521 lithium battery is used for data (SRAM) and RTC buffering while the processor module is not powered.

See system technology - AC500 battery. *Chapter 1.6.4.1.4.2 “AC500 battery” on page 5419*

The CPU monitors the discharge degree of the battery. A warning is issued before the battery condition becomes critical (about 2 weeks before). Once the warning message appears, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



Battery lifetime

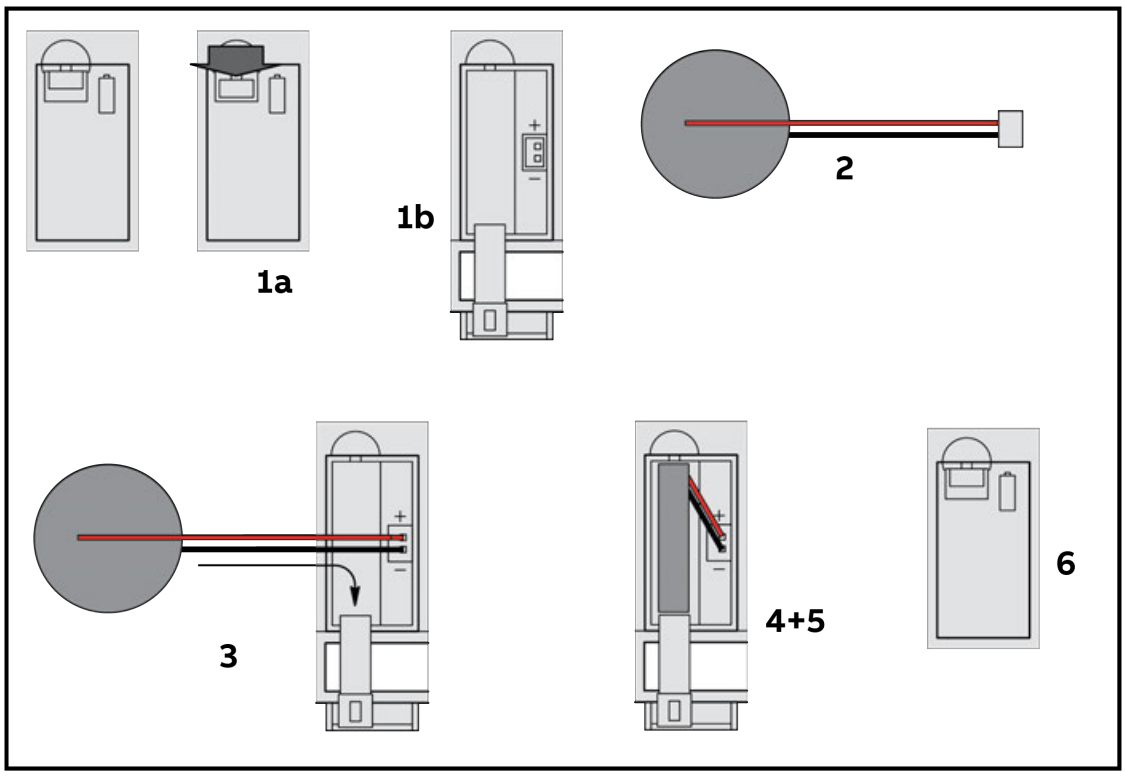
The battery lifetime is the time, the battery can store data while the processor module is not powered. As long as the processor module is powered, the battery will only be discharged by its own leakage current.

To avoid a short battery discharge, the battery should always be inserted or replaced while the process module is under power, then the battery is correctly recognized and will not shortly discharged.

Insertion

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

WARNING!
Risk of fire or explosion!
Use of incorrect Battery may cause fire or explosion.



1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front face of the processor module and cannot be removed.
2. Remove the TA521 battery from its package and hold it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.
3. Insert the battery connector into the small connector port of the compartment. The connector is keyed to find the correct polarity (red = positive pole = above).
4. Insert first the cable and then the battery into the compartment, push it until it reaches the bottom of the compartment.
5. Arrange the cable in order not to inhibit the door to close.
6. Pull-up the door and press until the locking mechanism snaps.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

To ensure propper operation and to prevent data loss, the battery insertion or replacement must be always done with the system under power. Without battery and power supply there is no data buffering possible.

1. Open the battery compartment with the small locking mechanism, press it down and slip down the door. The door is attached to the front view of the processor module and cannot be removed.
2. Remove the old TA521 battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

3. Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Technical data

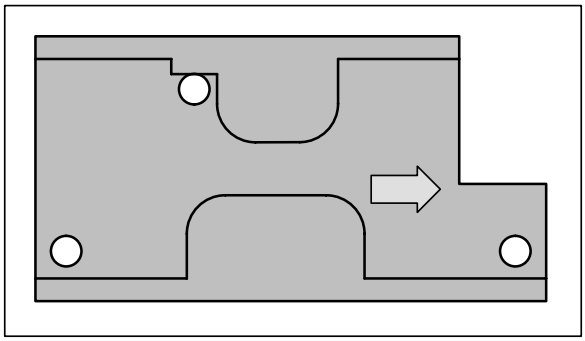
Parameter	Value
Nominal voltage	3 V
Nominal capacity	560 mAh
Temperature range (index below C0)	Operating: 0 °C...+60 °C Storage: -20 °C...+60 °C Transport: -20 °C...+60 °C
Temperature range (index C0 and above)	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	2 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug.
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug, black = negative pole,
Weight	7 g
Dimensions	Diameter of the button cell: 24.5 mm Thickness of the button cell: 5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 300 R0001	TA521, lithium battery	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA526 - Wall mounting accessory



Purpose

If a terminal base TB5xx or a terminal unit TU5xx should be mounted with screws, the wall mounting accessories TA526 must be inserted at the rear side first. This plastic parts prevent bending of terminal bases and terminal units while screwing up.

Handling instructions

Handling of the wall mounting accessory is described in detail in the section *Mounting and disassembling the terminal unit* ↳ *"Mounting with screws" on page 5328* and *Mounting/Disassembling Terminal Bases and Function Module Terminal Bases* ↳ *"Mounting with screws" on page 5326*.

Technical data

Parameter	Value
Weight	5 g
Dimensions	67 mm x 35 mm x 5,5 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 800 R0001	TA526, wall mounting accessory	Active

**) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.*

TA541 - Battery

- Manganese dioxide lithium battery, 3 V
- Non-rechargeable

Purpose

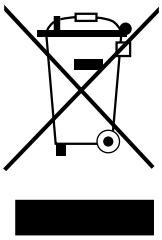
The TA541 lithium battery is the only applicable battery for PM595 ↳ *Chapter 1.6.2.3.2.2 "PM595-4ETH" on page 3863*. It is used to save RAM content of the processor module (PM595-4ETH-F only) and to back-up the real-time clock (all PM595 variants). It cannot be recharged.

The processor modules are supplied without a lithium battery. It therefore must be ordered separately. The lithium battery is used to save RAM contents of AC500 processor modules and back-up the real-time clock. Although the processor modules can work without a battery, its use is still recommended in order to avoid process data being lost.

The CPU monitors the discharge degree of the battery. A warning is output, before the battery condition becomes critical (about 2 weeks before). After the warning message has appeared, the battery should be replaced as soon as possible.

Handling instructions

- Do not short-circuit or re-charge the battery! It can cause excessive heating and explosion.
- Do not disassemble the battery!
- Do not heat up the battery and not put into fire! Risk of explosion.
- Store the battery in a dry place.
- Replace the battery with supply voltage ON in order not to risk data being lost.
- Recycle exhausted batteries meeting the environmental standards.



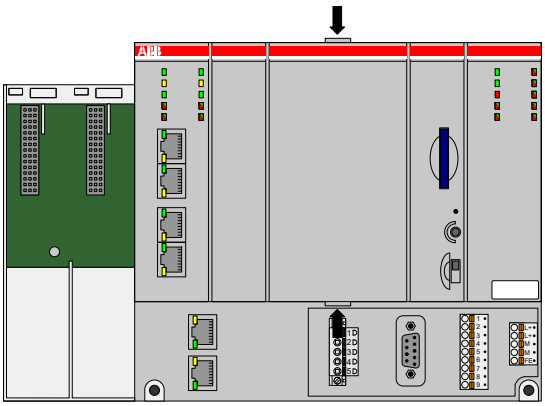
Battery lifetime

The battery lifetime is the time the battery can store data while the CPU is not powered. As long as the CPU is powered, the battery will only be discharged by its own leakage current.

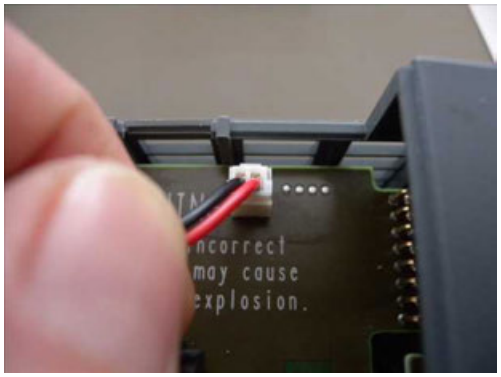
Insertion

The TA541 lithium battery is the only applicable battery for processor modules PM595.

1. Remove the front cover / display by pressing the marked areas with your fingers and pull it to the front.

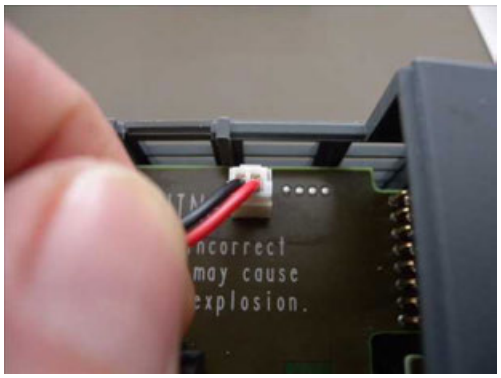


2. Remove the old battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket.



3. Remove the battery from its package and hold it by the small cable.

4. Insert the battery connector into the connector port of the PCB. The connector is keyed to find the correct polarity (red = positive pole = right side).



5. Insert the battery into the battery compartment on the left side as shown in the figure.

6. Re-assemble the front cover / display by pressing it straight from the front until it snaps in.

In order to prevent data losses or problems, the battery should be replaced after 3 years of utilisation or at least as soon as possible after receiving the "low battery warning" indication.

Do not use a battery older than 3 years for replacement, do not keep batteries too long in stock.

Replacement of the battery

For PM595-4ETH-F only: battery replacement should be done with the system under power. Without battery and power supply there is no data buffering possible.

For PM595-4ETH-M-XC only: battery only back-ups the real-time clock.

1. Remove the front cover / display by pressing the marked areas and pull it to the front.
2. Remove the old battery from the battery compartment by pulling it by the small cable. Remove then the small connector from the socket, do this best by lifting it out with a screwdriver.

   Follow the previous instructions to insert a new battery.

CAUTION!

Risk of explosion!

Do not open, re-charge or disassemble a lithium battery. Attempts to charge lithium batteries lead to overheating and possible explosions.

Protect them from heat and fire and store them in a dry place.

Never short-circuit or operate lithium batteries with the polarities reversed. The batteries are likely to overheat and explode. Avoid chance short circuiting and therefore do not store batteries in metal containers and do not place them on metallic surfaces. Escaping lithium is a health hazard.

Technical data

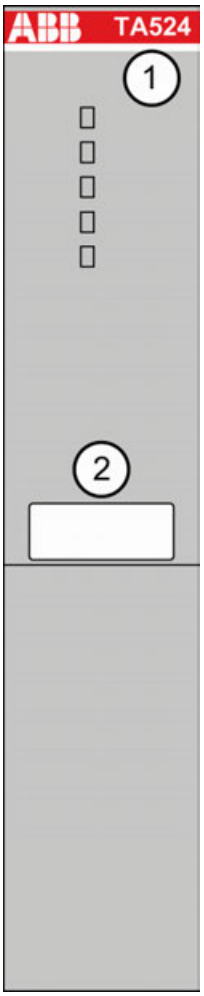
Parameter	Value
Nominal voltage	3 V
Nominal capacity	1800 mAh
Temperature range	Operating: -40 °C...+70 °C Storage: -40 °C...+85 °C Transport: -40 °C...+85 °C
Battery lifetime	Typ. 3 years at 25 °C
Self-discharge	1 % per year at 25 °C 5 % per year at 40 °C 20 % per year at 60 °C
Protection against reverse polarity	Yes, by mechanical coding of the plug
Insulation	The battery is completely insulated.
Connection	Red = positive pole = above at plug Black = negative pole
Weight	17 g
Dimensions	Diameter of the battery: ca. 18 mm Height of the battery: ca. 35 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 700 R0001	TA541, lithium battery	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

TA524 - Dummy communication module



1 Type
2 Label

Purpose

TA524 is used to cover an unused communication module slot of a terminal base *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*. It protects the terminal base from dust and inadvertent touch.

Handling instructions

TA524 is mounted in the same way as a common communication module *Chapter 1.6.3.6.3.6 “Mounting/Demounting the communication modules” on page 5335*.

Technical data

Parameter	Value
Weight	50 g
Dimensions	135 mm x 28 mm x 62 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 180 600 R0001	TA524, dummy communication module	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

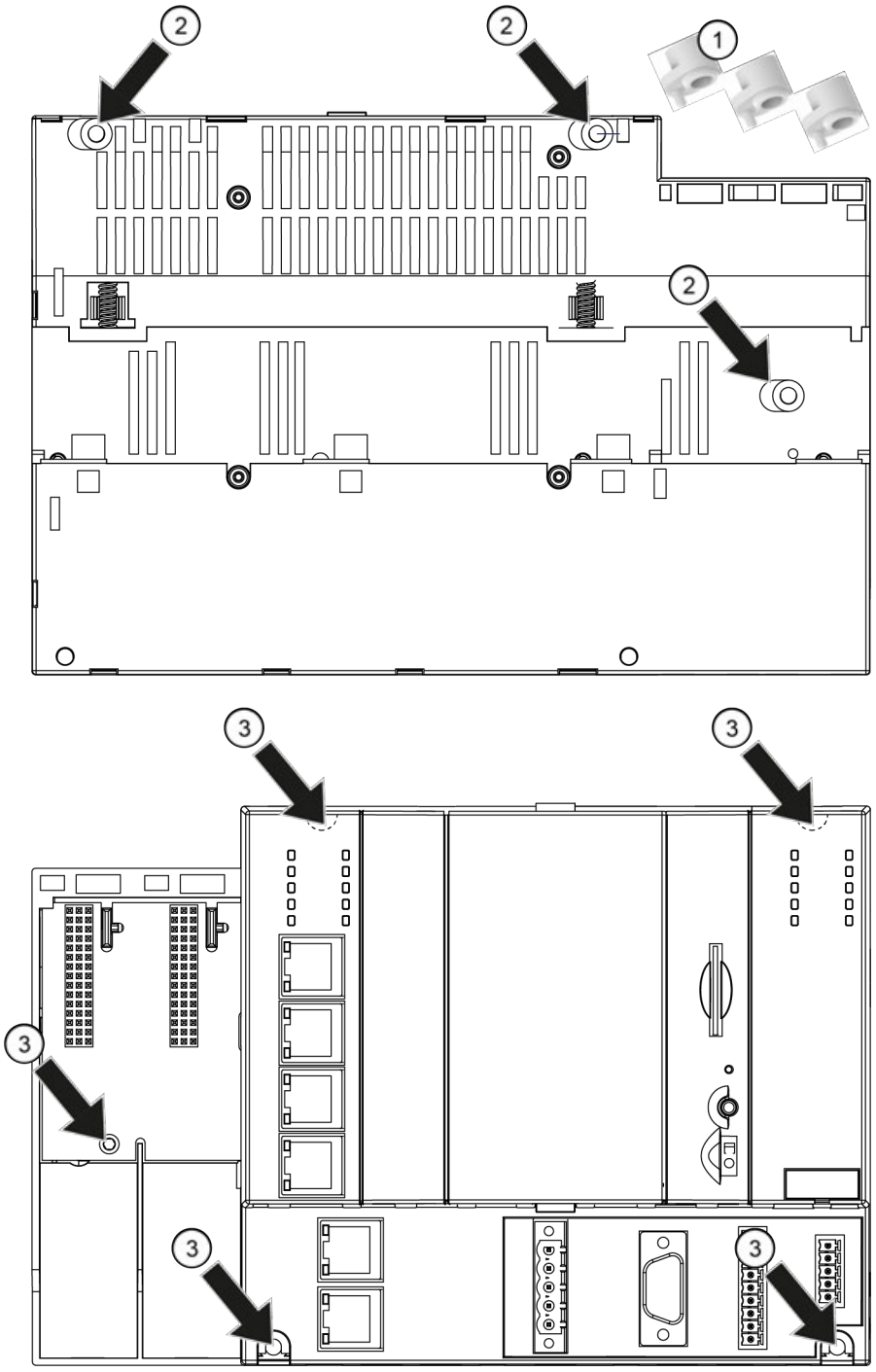
TA543 - Screw mounting accessory

Intended purpose

The TA543 screw mounting accessory is used for mounting the processor module PM595 without DIN rail.

Handling instruction

3 TA543 must be snapped on the backside of PM595 *Chapter 1.6.3.6.3.3 “Mounting/ Demounting the processor module PM595” on page 5330*.



1 3 parts of screw mounting accessory TA543
2 3 slots for screw mounting accessory TA543
3 5 holes for screw mounting

Technical data

Parameter	Value
Weight	5 g
Dimensions	12 mm x 8.5 mm x 10 mm

Ordering data

Part no.	Description	Product life cycle phase *)
1SAP 182 800 R0001	TA543, screw mounting accessory for PM595	Active

*) Modules in lifecycle Classic are available from stock but not recommended for planning and commissioning of new installations.

CP-E - Economic range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

- Wide-range input voltage
- Mounting on DIN rail
- High efficiency of up to 90 %
- Low power dissipation and low heating
- Wide ambient temperature range from -40 °C...+70 °C
- No-load-proof, overload-proof, continuous short-circuit-proof
- Power factor correction (depending on the type)
- Approved in accordance with all relevant international standards

Table 618: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR427030R0000	CP-E 24/0.75	100-240 V AC or 120-370 V DC	24 V DC, 0.75 A	-	22.5
1SVR427031R0000	CP-E 24/1.25	100-240 V AC or 90-375 V DC	24 V DC, 1.25 A	-	40.5
1SVR427032R0000	CP-E 24/2.5	100-240 V AC or 90-375 V DC	24 V DC, 2.5 A	-	40.5
1SVR427034R0000	CP-E 24/5.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 5 A	-	63.2
1SVR427035R0000	CP-E 24/10.0	115/230 V AC auto select or 210-370 V DC	24 V DC, 10 A	-	83
1SVR427036R0000	CP-E 24/20.0	115-230 V AC or 120-370 V DC	24 V DC, 20 A	-	175

CP-C.1 - High performance range

The power supplies feature series and parallel connection as well as a true redundant setup via a redundancy module.

The CP-C.1 power supplies are ABB's high performance and most advanced range. With excellent efficiency, high reliability and innovative functionality it is prepared for the most demanding industrial applications. These power supplies have a 50 % integrated power reserve and operate at an efficiency of up to 94 %. They are equipped with overheat protection and active power factor correction. Combinded with a broad AC and DC input range and extensive worldwide approvals the CP-C.1 power supplies are the preferred choice for professional DC applications.

- Typical efficiency of up to 94 %
- Power reserve design delivers up to 150 % of the nominal output current
- Signaling outputs for DC OK and power reserve mode
- High power density leads to very compact and small devices
- No-load-proof, overload-proof, continuous short-circuit-proof
- Active power factor correction (PFC)

Table 619: Ordering data

Order No.	Type	Input	Output	Overload capacity	Module width [mm]
1SVR360563R1001	CP-C.1 24/5.0	110-240 V AC or 90-300 V DC	24 V DC, 5 A	+50 %	40
1SVR360663R1001	CP-C.1 24/10.0	110-240 V AC or 90-300 V DC	24 V DC, 10 A	+50 %	60
1SVR360763R1001	CP-C.1 24/20.0	110-240 V AC or 90-300 V DC	24 V DC, 20 A	+30 %	82

1.6.3.7 AC500-XC

1.6.3.7.1 System data AC500-XC

Assembly, construction and connection of devices of the variant AC500-XC is identical to AC500 (standard) ↬ Chapter 1.6.3.6 “AC500 (Standard)” on page 5313. The following description provides information on general technical data of AC500-XC system.

Environmental conditions

Table 620: Process and supply voltages

Parameter		Value
24 V DC		
	Voltage	24 V (-15 %, +20 %)
	Protection against reverse polarity	Yes
120 V AC...240 V AC wide-range supply		
	Voltage	120...240 V (-15 %, +10 %)
	Frequency	50/60 Hz (-6 %, +4 %)
Allowed interruptions of power supply		
	DC supply	Interruption < 10 ms, time between 2 interruptions > 1 s, PS2

NOTICE!
Exceeding the maximum power supply voltage for process or supply voltages could lead to unrecoverable damage of the system. The system might be destroyed.

NOTICE!
For the supply of the modules, power supply units according to PELV or SELV specifications must be used.

The creepage distances and clearances meet the requirements of the overvoltage category II, pollution degree 2.

Parameter		Value
Temperature		
	Operating	-40 °C...+70 °C -40 °C...-30 °C: Proper start-up of system; technical data not guaranteed -40 °C...0 °C: Due to the LCD technology, the display might respond very slowly. -40 °C...+40 °C: Vertical mounting of modules possible, output load limited to 50 % per group +60 °C...+70 °C with the following deratings: ● System is limited to max. 2 communication modules per terminal base ● Applications certified for cULus up to +60 °C ● Digital inputs: maximum number of simultaneously switched on input channels limited to 75 % per group (e.g. 8 channels => 6 channels) ● Digital outputs: output current maximum value (all channels together) limited to 75 % per group (e.g. 8 A => 6 A) ● Analog outputs only if configured as voltage output: maximum total output current per group is limited to 75 % (e.g. 40 mA => 30 mA) ● Analog outputs only if configured as current output: maximum number of simultaneously used output channels limited to 75 % per group (e.g. 4 channels => 3 channels)
	Storage / Transport	-40 °C...+85 °C
Humidity		Operating / Storage: 100 % r. H. with condensation
Air pressure		Operating: -1000 m....4000 m (1080 hPa...620 hPa) > 2000 m (< 795 hPa): ● max. operating temperature must be reduced by 10 K (e.g. 70 °C to 60°C) ● I/O module relay contacts must be operated with 24 V nominal only
Immunity to corrosive gases		Operating: Yes, according to: ISA S71.04.1985 Harsh group A, G3/GX IEC 60721-3-3 3C2 / 3C3
Immunity to salt mist		Operating: Yes, horizontal mounting only, according to IEC 60068-2-52 severity level: 1

NOTICE!

Risk of corrosion!

Unused connectors and slots may corrode if XC devices are used in salt-mist environments.

Protect unused connectors and slots with TA535 protective caps for XC devices. *Chapter 1.6.2.9.4.6 “TA535 - Protective caps for XC devices” on page 5212*

Table 621: Electromagnetic compatibility

Parameter		Value
Device suitable for:		
	Industrial applications	Yes
	Domestic applications	No
Radiated emission (radio disturbances)		Yes, according to: CISPR 16-2-3
Conducted emission (radio disturbances)		Yes, according to: CISPR 16-2-1, CISPR 16-1-2
Electrostatic discharge (ESD)		Yes, according to: IEC 61000-4-2, zone B, criterion B
Fast transient interference voltages (burst)		Yes, according to: IEC 61000-4-4, zone B, criterion B
High energy transient interference voltages (surge)		Yes, according to: IEC 61000-4-5, zone B, criterion B
Influence of radiated disturbances		Yes, according to: IEC 61000-4-3, zone B, criterion A
Influence of line-conducted interferences		Yes, according to: IEC 61000-4-6, zone B, criterion A
Influence of power frequency magnetic fields		Yes, according to: IEC 61000-4-8, zone B, criterion A

In order to prevent malfunctions, it is recommended, that the operating personnel discharge themselves prior to touching communication connectors or perform other suitable measures to reduce effects of electrostatic discharges.

NOTICE!

Risk of malfunctions!

Unused slots for communication modules are not protected against accidental physical contact.

– Unused slots for communication modules must be covered with dummy communication modules to achieve IP20 rating ↳ *Chapter 1.6.3.6.5.7 “TA524 - Dummy communication module” on page 5383*.
– I/O bus connectors must not be touched during operation.

Mechanical data

Parameter	Value
Wiring method	Spring terminals
Degree of protection	IP 20
Vibration resistance	Yes, according to: IEC 61131-2 IEC 60068-2-6 IEC 60068-2-64
Shock resistance	Yes, according to: IEC 60068-2-27
Assembly position	Horizontal Vertical (no application in salt mist environment)
Assembly on DIN rail	
DIN rail type	According to IEC 60715 35 mm, depth 7.5 mm or 15 mm
Assembly with screws	
Screw diameter	4 mm
Fastening torque	1.2 Nm

Environmental tests

Parameter	Value
Storage	IEC 60068-2-1 Test Ab: cold withstand test -40 °C / 16 h IEC 60068-2-2 Test Bb: dry heat withstand test +85 °C / 16 h
Humidity	IEC 60068-2-30 Test Db: Cyclic (12 h / 12 h) damp-heat test 55 °C, 93 % r. H. / 25 °C, 95 % r. H., 6 cycles IEC 60068-2-78, stationary humidity test: 40 °C, 93 % r. H., 240 h
Insulation Test	IEC 61131-2

Parameter	Value
Vibration resistance	IEC 61131-2 / IEC 60068-26: 5 Hz...500 Hz, 2 g (with memory card inserted) IEC 60068-2-64: 5 Hz...500 Hz, 4 g rms
Shock resistance	IEC 60068-2-27: all 3 axes 15 g, 11 ms, half-sinusoidal

Table 622: EMC immunity

Parameter	Value
Electrostatic discharge (ESD)	Electrostatic voltage in case of air discharge: 8 kV Electrostatic voltage in case of contact discharge: 6 kV
Fast transient interference voltages (burst)	Supply voltage units (DC): 4 kV Digital inputs/outputs (24 V DC): 2 kV Analog inputs/outputs: 2 kV Communication lines shielded: 2 kV I/O supply (DC-out): 2 kV
High energy transient interference voltages (surge)	Supply voltage units (DC): 1 kV CM *) / 0.5 kV DM *) Digital inputs/outputs (24 V DC): 1 kV CM *) / 0.5 kV DM *) Digital inputs/outputs (AC): 4 kV Analog inputs/outputs: 1 kV CM *) / 0.5 kV DM *) Communication lines shielded: 1 kV CM)* I/O supply (DC-out): 0,5 kV CM *) / 0.5 kV DM *)
Influence of radiated disturbances	Test field strength: 10 V/m
Influence of line-conducted interferences	Test voltage: 10 V
Power frequency magnetic fields	30 A/m 50 Hz 30 A/m 60 Hz

*) CM = Common Mode, * DM = Differential Mode

1.6.3.8 AC500-S

The AC500-S safety user manual must be read and understood before using safety configuration and programming tools of Automation Builder / PS501 Control Builder Plus. Only qualified personnel shall be allowed to work with AC500-S safety PLCs.

In order to have always the latest version and due to a different lifecycle compared to Automation Builder help, the *AC500-S safety user manual* is only available on our website.

The AC500-S safety PLC includes the following safety-relevant hardware components.

- SM560-S / SM560-S-FD-1 / SM560-S-FD-4
- DI581-S
- DX581-S
- AI581-S
- TU582-S

1.6.4 System technology for AC500 V2 products

This chapter provides advanced information on the system technology of AC500 control systems from a general perspective. It provides information to link the details from the hardware descriptions (provided in the device specifications section) with detailed information on configuring/programming a corresponding library (provided in the individual library sections).

Configuration of a specific device with Automation Builder is described in the PLC configuration section.

1.6.4.1 System technology of CPU and overall system

1.6.4.1.1 Inputs, outputs and flags for AC500 V2 products

All operands supported by CODESYS are described in the CODESYS documentation
↳ *Chapter 1.4.1.7 “Operands” on page 435*.

In the following details on the "address" operands (%I for inputs, %Q for outputs, %M area and %R area for variables with address) and the data backup and initialization.

All addressable operands can be accessed bitwise (X), bytewise (B), wordwise (W) and double-wordwise (D) in Automation Builder. The Motorola byteorder is used for operand access.

Declaration of addressable operands

The declaration of the operands in the addressable flag area is done as follows:

Symbol AT address : Type [:= initialization value]; (* comment *)

[.] optional

The inputs and outputs are declared using the PLC configuration.

Interfaces for inputs and outputs

PM57x, PM58x and PM59x (AC500)

The following input and output interfaces are available for the AC500 CPUs PM57x, PM58x and PM59x:

No.	Type	Designation	Number of Inputs and Outputs
1	I/O bus	Interface for I/O modules	Max. 10 modules with a maximum of 32 channels (IX, QX, IW, QW) per module
2	COM1	CS31 bus master	Max. 31 modules with a maximum of 32 channels per module, address 0-61
3	COM2	Reserved	RS-232 / RS-485
4	FBP	FieldBusPlug - Slave	Max. 8 modules with 16 IW + 16 QW + 16 IB + 16 QB with modular FBP, depending on fieldbus
5	Onboard I/Os	Reserved	Max. 96 bytes Inputs and Outputs
6	Line 0	Internal communication module	4 kB %I0.xx / %Q0.xx each
7	Line 1	Communication module 1	4 kB %I1.xx / %Q1.xx each
8	Line 2	Communication module 2	4 kB %I2.xx / %Q2.xx each
9	Line 3	Communication module 3	4 kB %I3.xx / %Q3.xx each
10	Line 4	Communication module 4	4 kB %I4.xx / %Q4.xx each
11	Line 5	Internal communication module ETH3 of PM595-4ETH	4 kB %I5.xx / %Q5.xx each
12	Line 6	Internal communication module ETH4 of PM595-4ETH	4 kB %I6.xx / %Q6.xx each

PM55x and PM56x (AC500-eCo)

The following inputs and outputs interfaces are available for the AC500-eCo CPUs PM55x and PM56x:

No.	Type	Designation	Number of Inputs and Outputs
1	I/O bus	Interface for I/O modules	Max. 10 I/O modules with a maximum of 32 channels (IX, QX, IW, QW) per module
2	COM1	CS31 bus master	Max. 31 modules with a maximum of 32 channels per module, address 0-61
3	COM2	Reserved	RS-485
4	Onboard I/O	Onboard I/O	Per module: Option 1: 8DI+6DO Option 2: 8DI+6DO+2AI+1AO

Address scheme for inputs and outputs

- The Communication Module I/Os are addressed as follows (two-stage process):
  %I(Q)B**Communication ModuleNumber.ByteCommunication Module**
- No Communication Module numbers are assigned to I/Os that are connected to the CPU. These I/Os are configured with the PLC configuration in Automation Builder.
- I/Os connected to the basic unit are assigned to the following address areas:

I/O Bus:	%IB0 ..	%IB999	and %QB0 ..	%QB999
COM1:	%IB1000 ..	%IB1999	and %QB1000 ..	%QB1999
COM2 :	%IB2000 ..	%IB2999	and %QB2000 ..	%QB2999
FBP slave:	%IB3000 ..	%IB3999	and %QB3000 ..	%QB3999
Onboard I/O	%IB4000 ..	%IB4095	and %QB4000 ..	%QB4095

- Addressing of the digital channels is done byte-oriented.
- Motorola byteorder is used to access the inputs and outputs.

Example for addressing in BOOL / BYTE / WORD / DWORD

Address	**Addr**	**Addr + 1**	**Addr +2**	**Addr +3**
	16#xxxx x000	16#xxxx x001	16#xxxx x002	16#xxxx x003
BYTE	**%IB0**	**%IB1**	**%IB2**	**%IB3**
BOOL	7 ... 0	7 ... 0	7 ... 0	7 ... 0
	%IX0.7 ... %IX0.0	%IX1.7 ... %IX1.0	%IX2.7 ... %IX2.0	%IX3.7 ... %IX3.0
WORD	**%IW0**		**%IW1**	
	15 ... 8	7 ... 0	15 ... 8	7 ... 0
DWORD	**%ID0**			
	31 ... 24	23 ... 16	15 ... 8	7 ... 0

Examples:

%IX0.0	:= TRUE		
	%IB0 := 1	:= 16#01	
	%IW0 := 256	:= 16#0100	(Bit 8 = TRUE)
	%ID0 := 16777216	:= 16#01000000	(Bit 24 = TRUE)

%IX3.0	:= TRUE		
	%IB3 := 1	:= 16#01	
	%IW1 := 1	:= 16#0001	
	%ID0 := 1	:= 16#00000001	

Addressing of inputs and outputs

PM57x, PM58x and PM59x (AC500)

No.	Device	Input / Output	Interface		Range	Addresses
0 ... 5	CPU I/Os and	Inputs (4 kB)	CPU	I/O Bus	0000..0999	%IB0 ... %IB4095
				COM1	1000..1999	%IW0 ... %IW2047
				COM2	2000..2999	%ID0 ... %ID1023
				FBP	3000..3999	%IX0.0 ... %IX4095.7
				Onboard IO	4000..4095	
		Outputs (4 kB)		I/O Bus	0000..0999	%QB0 ... %QB4095
				COM1	1000..1999	%QW0 ... %QW2047
				COM2	2000..2999	%QD0 ... %QD1023
				FBP	3000..3999	%QX0.0 ... %QX4095.7
				Onboard IO	4000..4095	
	Internal communi-cation module	Inputs (4 kB)	Line 0		0.0000 ... 0.4095	%IB0.0 ... %IB0.4095
						%IW0.0 ... %IW0.2047
						%ID0.0 ... %ID0.1023
						%IX0.0.0 ... %IX0.4095.7
		Outputs (4 kB)				%QB0.0 ... %QB0.4095
						%QW0.0 ... %QW0.2047
						%QD0.0 ... %QD0.1023
						%QX0.0.0 ... %QX0.4095.7
6	Commu-nication Module 1	Inputs (4kB)	Line 1		1.0000 ... 1.4095	%IB1.0 ... %IB1.4095
						%IW1.0 ... %IW1.2047
						%ID1.0 ... %ID1.1023

No.	Device	Input / Output	Interface	Range	Addresses
					%IX1.0.0 ...%IX1.4095.7
		Outputs (4kB)			%QB1.0 ... %QB1.4095
					%QW1.0 ... %QW1.2047
					%QD1.0 ... %QD1.1023
					%QX1.0.0 ... %QX1.4095.7
...					
12	Communication Module 6	Inputs (4kB)	Line 6	6.0000 ... 6.4095	%IB4.0 ... %IB4.4095
					%IW6.0 ... %IW6.2047
					%ID6.0 ... %ID6.1023
					%IX6.0.0 ... %IX6.4095.7
		Outputs (4kB)			%QB6.0 ... %QB6.4095
					%QW6.0 ... %QW6.2047
					%QD6.0 ... %QD6.1023
					%QX6.0.0 ... %QX6.4095.7

PM55x and PM56x (AC500-eCo)

No.	Device	Input / Output	Interface		Range	Addresses
0 ... 5	CPU I/Os	Inputs (4kB)	CPU	I/O Bus	0000..0999	%IB0 ... %IB4095
				COM1	1000..1999	%IW0 ... %IW2047
				COM2	2000..2999	%ID0 ... %ID1023
				not used	3000..3999	%IX0.0 ... %IX4095.7
				Onboard I/O	4000..4095	
		Outputs (4kB)		I/O Bus	0000..0999	%QB0 ... %QB4095
				COM1	1000..1999	%QW0 ... %QW2047
				COM2	2000..2999	%QD0 ... %QD1023

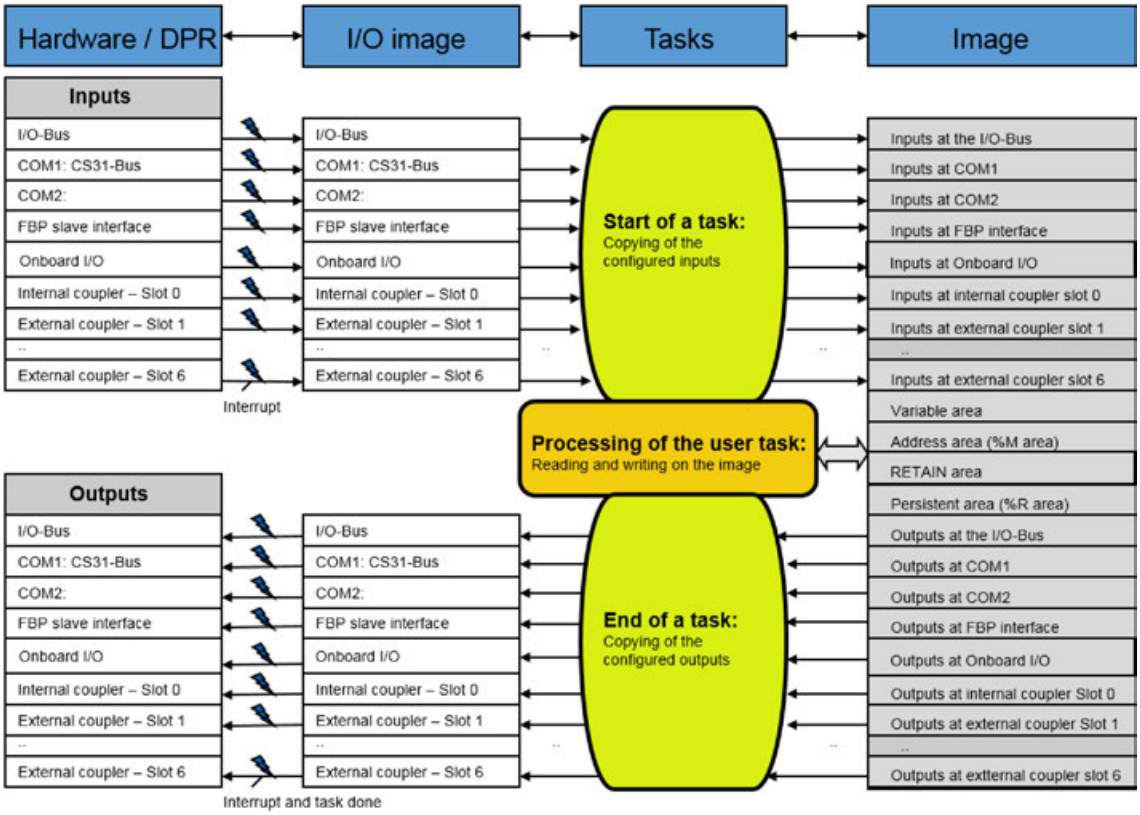
No.	Device	Input / Output	Interface		Range	Addresses
				not used	3000..3999	%QX0.0 ... %QX4095.7
				Onboard I/O	4000..4095	

Processing of inputs and outputs in the multitasking system

CAUTION!
Multitasking
All configured inputs are updated in Image with the values of IO image at the start of an IEC task. All configured outputs are updated in IO image with the values of Image at the end of an IEC task (see following figure).

If, for example, task 1 has the higher priority and input %IX0.0 is used in task 1 and task 2, the value can change during the cycle of task 2 as it is updated every time task 1 is started. This is not relevant for programs with only one task.

The following figure shows how the inputs and outputs are processed in the multitasking system.



Generation of the input data image

Inputs at I/O bus

1. After all I/O modules have been processed at the I/O bus, a corresponding interrupt is generated in the processor. The inputs are copied to the input data image during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied.

Inputs at CS31 bus

2. After the CS31 driver has processed all I/O modules, a corresponding interrupt is generated in the processor. The inputs are copied to the input data image during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied.

Inputs of onboard I/O

3. After Onboard I/Os of AC500 PM55x and PM56x CPUs have been processed, a corresponding interrupt is generated in the processor. The inputs are copied to the input data image during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied.

Inputs of communication modules line 0 to 6

4. Once a Communication Module has received new data, a corresponding interrupt is generated in the processor. The inputs are copied from the DPR to the input data image of the processor during the Interrupt Service Routine (ISR). If the outputs were updated by a task, the outputs in the output data image are copied to the DPR.

   Precondition for this is a valid Communication Module configuration.

Starting a task When starting a task, all configured inputs are copied from the input data image to the image.

Processing a task All tasks access the image, i.e., inputs are read from the image and outputs are written to the image. In ONLINE mode, the inputs/outputs of the image are displayed.

Termination by a task At the end of the task processing, all configured outputs are copied from the image to the output data image.

Writing the outputs

1. Outputs at the I/O Bus: With the next interrupt of the I/O bus driver, the outputs of the output data image will be written.
2. Outputs at the CS31 bus: With the next interrupt of the CS31 processor, the outputs of the output data image will be written.
3. Outputs of Onboard I/O of AC500 PM55x and PM56x CPUs: With the next interrupt of the I/O Bus driver, the outputs of the output data image will be written.
4. Outputs of the Communication Module Line 0 to 6: With the next interrupt of the Communication Module, the outputs of the output data image will be written to the DPR.

I/O update task In order to update the inputs/outputs not used in the task, all inputs/outputs of the image are updated by a lower priority task (I/O update task). This task is only processed if no other user task runs.

Addressable flag area (%M area)

Allocation The addressable flag area for the AC500 is divided into several segments with a size of 64 kB per segment. A maximum of 8 segments can be addressed. The availability of the segments or partial segments depends on the CPU. The size of the %M area can be found in the technical data of the *Chapter 1.6.2.3 “Processor modules” on page 3803*.

Segment	Operands	Size, cumulative [kB]	PM55x/ PM56x	PM57x	PM573	PM58x	PM59x
0	%MB0.0...%MB0.65535	64	2 kB	4 kB	128 kB	+	+
1	%MB1.0...%MB1.65535	128	-	-	-	+	+
2	%MB2.0...%MB2.65535	192	-	-	-	-	+
3	%MB3.0...%MB3.65535	256	-	-	-	-	+
4	%MB4.0...%MB4.65535	320	-	-	-	-	+
5	%MB5.0...%MB5.65535	284	-	-	-	-	+
6	%MB6.0...%MB6.65535	448	-	-	-	-	+
7	%MB7.0...%MB7.65535	512	-	-	-	-	+

The %M area in PM55x and PM65x cannot be retained, even if declared as VAR_RETAIN. Use %R area instead of %M area if retained variables are needed.

Access to %M area

The Modbus TCP and Modbus RTU protocols are implemented in the AC500. With the help of the Modbus protocols, the segments 0 and 1 of the addressable flag area can be accessed. The chapter Modbus *Chapter 1.6.4.1.8 “Communication with Modbus RTU” on page 5467* in this documentation contains a detailed description of the Modbus protocols and the corresponding addressing.

For the AC500 CPUs PM55x and PM56x, 2kB = %MB0.0 .. %MB0.2047 (i.e., not a complete segment) are available for the addressable flag area. Thus, not all Modbus addresses can be accessed.

Access to operands

The operands in the %M area can be accessed bitwise, bytewise, wordwise and double-word-wise.

Byte SINT / BYTE	Byte SINT / BYTE	Word INT / WORD	Word INT / WORD
Segment 0			
%MB0.0	%MX0.0.0 ... %MX0.0.7	%MW0.0	%MD0.0
%MB0.1	%MX0.1.0 ... %MX0.1.7		
%MB0.2	%MX0.2.0 ... %MX0.2.7	%MW0.1	
%MB0.3	%MX0.3.0 ... %MX0.3.7		
...	...	...	...
%MB0.65532	%MX0.65532.0 ... %MX0.65532.7	%MW0.32766	%MD0.16383

Byte SINT / BYTE	Byte SINT / BYTE	Word INT / WORD	Word INT / WORD
%MB0.65533	%MX0.65533.0 ... %MX0.65533.7		
%MB0.65534	%MX0.65534.0 ... %MX0.65534.7	%MW0.32767	
%MB0.65535	%MX0.65535.0 ... %MX0.65535.7		
Segment 1			
%MB1.0	%MX1.0.0 ... %MX1.0.7	%MW1.0	%MD1.0
%MB1.1	%MX1.1.0 ... %MX1.1.7		
%MB1.2	%MX1.2.0 ... %MX1.2.7	%MW1.1	
%MB1.3	%MX1.3.0 ... %MX1.3.7		
...	...	...	...
%MB1.65532	%MX1.65532.0 ... %MX1.65532.7	%MW1.32766	%MD1.16383
%MB1.65533	%MX1.65533.0 ... %MX1.65533.7		
%MB1.65534	%MX1.65534.0 ... %MX1.65534.7	%MW1.32767	
%MB1.65535	%MX1.65535.0 ... %MX1.65535.7		
Segment 2			
%MB2.0	%MX2.0.0 ... %MX2.0.7	%MW2.0	%MD2.0
%MB2.1	%MX2.1.0 ... %MX2.1.7		
%MB2.2	%MX2.2.0 ... %MX2.2.7	%MW2.1	
%MB2.3	%MX2.3.0 ... %MX2.3.7		
...	...	...	...
%MB2.65532	%MX2.65532.0 ... %MX2.65532.7	%MW2.32766	%MD2.16383
%MB2.65533	%MX2.65533.0 ... %MX2.65533.7		
%MB2.65534	%MX2.65534.0 ... %MX2.65534.7	%MW2.32767	
%MB2.65535	%MX2.65535.0 ... %MX2.65535.7		
...	...	...	...
Segment 7			
%MB7.0	%MX7.0.0 ... %MX7.0.7	%MW7.0	%MD7.0
%MB7.1	%MX7.1.0 ... %MX7.1.7		
%MB7.2	%MX7.2.0 ... %MX7.2.7	%MW7.1	
%MB7.3	%MX7.3.0 ... %MX7.3.7		
...	...	...	...
%MB7.65532	%MX7.65532.0 ... %MX7.65532.7	%MW7.32766	%MD7.16383

Byte SINT / BYTE	Byte SINT / BYTE	Word INT / WORD	Word INT / WORD
%MB7.65533	%MX7.65533.0 ... %MX7.65533.7		
%MB7.65534	%MX7.65534.0 ... %MX7.65534.7	%MW7.32767	
%MB7.65535	%MX7.65535.0 ... %MX7.65535.7		

Absolute addresses of operands

ADR

For particular blocks or in case of accessing operands via pointers, the absolute address of an operand must be determined. To do this, Automation Builder provides the address operator ADR *Chapter 1.4.1.6.7.1 “ADR” on page 421*.

The following description describes only the peculiarities of bit operands.

The addresses provided by the address operator can be used as inputs for blocks that require absolute addresses (such as xxx_MOD_MAST, COM_SND). If these blocks shall be applied to internal variables, it must be guaranteed that the variables are set to successive addresses. This is achieved by declaring ARRAYs and STRINGs.

The address operator ADR provides the address of an operand in one double word DWORD (i.e., 32 bits). The address operator returns the address of the first byte of a variable (byte address). For the user-definable variables, variables of the type BOOL are stored as byte.

BITADR

For inputs, outputs and variables of the addressable flag area (%M area) or addressable PERSISTENT area (%R area), operands of the type BOOL occupy one bit. The address of this type of variables cannot be determined with the operator ADR.

When processing the statement: dwAddress := ADR(%MX0.0.0); the following error message appears:

Error 4031:

PLC_PRG(xx): ADR is not allowed for bits! Use BITADR instead.

BITADR returns the bit offset within the area %I, %Q or %M as DWORD.

The following table shows the position of the operands within the memory (considering %MD0.0 and %MD0.1 as example). Here you get information about which addresses the operator ADR returns and which offsets BITADR returns.

The addresses shown are example addresses and thus can have other values.

Position of operands within memory and values of operators ADR and BITADR:

Byte SINT / BYTE	Word INT / WORD	Double word DINT / DWORD	Bit (byte-oriented) BOOL	ADR	BITADR
%MB0.0	%MW0.0	%MD0.0	%MX0.0.0	16#08000000	8
			%MX0.0.1		9
			%MX0.0.2		10
			%MX0.0.3		11
			%MX0.0.4		12

Byte SINT / BYTE	Word INT / WORD	Double word DINT / DWORD	Bit (byte-oriented) BOOL	ADR	BITADR
			%MX0.0.5		13
			%MX0.0.6		14
			%MX0.0.7		15
%MB0.1			%MX0.1.0	16#08000001	0
			%MX0.1.1		1
			%MX0.1.2		2
			%MX0.1.3		3
			%MX0.1.4		4
			%MX0.1.5		5
			%MX0.1.6		6
			%MX0.1.7		7
%MB0.2	%MW0.1		%MX0.2.0	16#08000002	24
			%MX0.2.1		25
			%MX0.2.2		26
			%MX0.2.3		27
			%MX0.2.4		28
			%MX0.2.5		29
			%MX0.2.6		30
			%MX0.2.7		31
%MB0.3			%MX0.3.0	16#08000003	16
			%MX0.3.1		17
			%MX0.3.2		18
			%MX0.3.3		19
			%MX0.3.4		20
			%MX0.3.5		21
			%MX0.3.6		22
			%MX0.3.7		23
%MB0.4	%MW0.2	%MD0.1	%MX0.4.0	16#08000004	40
			...		...
			%MX0.4.7		47
%MB0.5			%MX0.5.0	16#08000005	32
			...		...
			%MX0.5.7		39
%MB0.6	%MW0.3		%MX0.6.0	16#08000006	56
			...		...
			%MX0.6.7		63
%MB0.7			%MX0.7.0	16#08000007	48
			...		...
			%MX0.7.7		55

Addressable PERSISTENT area (%R area)

Special features The addressable PERSISTENT area or %R area has the following peculiarities:

- Variables declared in the %R area are always located at the same position in the PLC's operand memory, because they have addresses assigned (like the variables in the %M area).
- Variables in the %R area are declared as follows:

```
VAR (caution: no RETAIN or PERSISTENT option),
   Symbol AT %RTypeSegment.Offset : TYPE;      (* Comment *), or also
   Symbol AT %RTypeSegment.Offset : ARRAY[start..end] OF TYPE  (*
Comment *)
END_VAR
```

where:	`Symbol`	symbolic name of the variable
	`Type`	X=BOOL (Bit), B=BYTE, W=WORD, D=DWORD
	`Segment`	0..15 (availability depends on CPU type)
	`Offset`	0..65535 (availability depends on CPU type)
	`TYPE`	BOOL, BYTE, WORD, DWORD or defined type (such as structure)
	`start`	Index of the first ARRAY element
	`end`	Index of the last ARRAY element

- For each segment in the %R area, an area can be set in the PLC configuration, which is buffered in case the **battery is installed and fully charged**. In this case, the variables behave like variables declared as VAR RETAIN PERSISTENT, i.e., they keep their values even after
  - Online changes (like all other variables),
  - Power OFF/ON (like VAR RETAIN), and
  - download (like VAR PERSISTENT).
- In contrast to the variables declared as VAR PERSISTENT, these variables have the great advantage that no program code is required for dumping the variables during a download.
- The buffered part of the %R area can be written to the memory card and read from the card (see *“Saving the buffered data of the %R area” on page 5406*).

Segmentation The addressable PERSISTENT area in the AC500 is divided into several segments with a size of 64 kB per segment. A maximum of 8 segments can be addressed. The availability of the segments or partial segments depends on the CPU:

	Operands	**eCo(-ETH)** **PM55x** **PM56x**	**PM572**	**PM573-ETH** **PM582** **PM583-ETH**	**PM590-ETH** **PM591-ETH/2ETH** **PM592-ETH**	**PM595-4ETH**
%R retain & PERISISTENT		**1 kB**	**4 kB**	**128 kB**	**512 kB**	**1024 kB**
Segment 0	%RB0.0...%RB0.65535	+	+	+	+	+
Segment 1	%RB1.0...%RB1.65535			+	+	+
Segment 2	%RB2.0...%RB2.65535				+	+

	Operands	eCo(-ETH) PM55x PM56x	PM572	PM573-ETH PM582 PM583-ETH	PM590-ETH PM591-ETH/2ETH PM592-ETH	PM595-4ETH
%R retain & PERISISTENT		**1 kB**	**4 kB**	**128 kB**	**512 kB**	**1024 kB**
Segment 3	%RB3.0...%RB3.65535				+	+
Segment 4	%RB4.0...%RB4.65535				+	+
Segment 5	%RB5.0...%RB5.65535				+	+
Segment 6	%RB6.0...%RB6.65535				+	+
Segment 7	%RB7.0...%RB7.65535				+	+
Segment 8	%RB8.0...%RB8.65535					+
Segment 9	%RB9.0...%RB9.65535					+
Segment 10	%RB10.0...%RB10.65535					+
Segment 11	%RB11.0...%RB11.65535					+
Segment 12	%RB12.0...%RB12.65535					+
Segment 13	%RB13.0...%RB13.65535					+
Segment 14	%RB14.0...%RB14.65535					+
Segment 15	%RB15.0...%RB15.65535					+

Saving the buffered data of the %R area

The buffered part of the %R area can be saved on the memory card and read from the card. This can be necessary, if, for example, the controller has to be replaced.

Saving data

1. Copy the data from the %R area and write it to the CPU's RAM disk as file
2. Save the file to the memory card.

Reading data from the memory card

1. Load the file from the memory card to the CPU's RAM disk.
2. Copy the data from the RAM disk to the %R area.

CAUTION!
Data mismatch can occur!

The variables structure / layout has to be identical to the old one and should not be changed!

Saving and reading the data can be done using function blocks in the user program or with the PLC Browser contained in the Automation Builder. The function blocks are contained in the *Chapter 1.5.4.19 “Internal system library” on page 1500*.

Function	PLC Browser command	Function block
Copy from %R area to RAM disk	persistent save	PERSISTENT_SAVE
Save file to SD Card	persistent export	PERSISTENT_EXPORT
Read file from SD Card to RAM disk	persistent import	PERSISTENT_IMPORT
Copy data from RAM disk to %R area	persistent restore	PERSISTENT_RESTORE
Delete buffered data of the PERSISTENT area	persistent clear	PERSISTENT_CLEAR

CAUTION!

Cycle consistency for data

If cycle consistency is required for the data, this has to be implemented in the user program. That means, that the data may not be changed during copying to/from the %R area from/to the RAM disk. If saving is done using the PLC Browser, this can be easily carried out by stopping the user program.

CAUTION!

Cycle time for copying the PERSISTENT area

Copying the PERSISTENT area takes some milliseconds (see the following table). Thus, an according cycle time has to be set in the task configuration. Please note the remarks on the task configuration!

Action	Time in ms			
	CPU PM55x CPU PM56x	CPU PM57x	CPU PM58x	CPU PM59x
Restoring 1 kB (1024 bytes)				
PERSISTENT_CLEAR	<1	<1	<1	<1
PERSISTENT_SAVE	2	2	2	2
PERSISTENT_EXPORT	900	1000	1000	500
PERSISTENT_IMPORT	100	500	1000	500
PERSISTENT_RESTORE	3	2	<1	1
Restoring 4 kB (4096 bytes)				
PERSISTENT_CLEAR	not possible	<1	<1	<1
PERSISTENT_SAVE	not possible	2	3	2
PERSISTENT_EXPORT	not possible	1000	1000	500
PERSISTENT_IMPORT	not possible	500	1000	500
PERSISTENT_RESTORE	not possible	3	3	2
Restoring 64 kB (65536 bytes)				
PERSISTENT_CLEAR	not possible	not possible	8	2
PERSISTENT_SAVE	not possible	not possible	11	6
PERSISTENT_EXPORT	not possible	not possible	2500	1000

Action	Time in ms			
	CPU PM55x **CPU PM56x**	**CPU PM57x**	**CPU PM58x**	**CPU PM59x**
PERSISTENT_IMPORT	not possible	not possible	2000	500
PERSIS-TENT_RESTORE	not possible	not possible	12	5
Restoring max. PERSISTENT area				
		4 kB	**128 kB**	**512 kB**
PERSISTENT_CLEAR	<1	<1	17	22
PERSISTENT_SAVE	2	2	22	35
PERSISTENT_EXPORT	900	1000	4000	8000
PERSISTENT_IMPORT	100	500	3000	4000
PERSIS-TENT_RESTORE	3	3	22	31

Access to operands in the addressable PERSISTENT area (%R Area):

The operands in the %R area can be accessed bitwise, bytewise, wordwise and double-word-wise.

Byte SINT / BYTE	Bit (byte-oriented) BOOL	Word INT / WORD	Double word DINT / DWORD
Segment 0			
%RB0.0	%RX0.0.0 ... %RX0.0.7	%RW0.0	%RD0.0
%RB0.1	%RX0.1.0 ... %RX0.1.7		
%RB0.2	%RX0.2.0 ... %RX0.2.7	%RW0.1	
%RB0.3	%RX0.3.0 ... %RX0.3.7		
%RB0.65532	%RX0.65532.0...%RX0.65532.7	%RW0.32766	%RD0.16383
%RB0.65533	%RX0.65533.0...%RX0.65533.7		
%RB0.65534	%RX0.65534.0...%RX0.65534.7	%RW0.32767	
%RB0.65535	%RX0.65535.0...%RX0.65535.7		
Segment 1			
%RB1.0	%RX1.0.0...%RX1.0.7	%RW1.0	%RD1.0
%RB1.1	%RX1.1.0...%RX1.1.7		
%RB1.2	%RX1.2.0...%RX1.2.7	%RW1.1	
%RB1.3	%RX1.3.0...%RX1.3.7		
%RB1.65532	%RX1.65532.0...%RX1.65532.7	%RW1.32766	%RD1.16383
%RB1.65533	%RX1.65533.0...%RX1.65533.7		

Byte SINT / BYTE	Bit (byte-oriented) BOOL	Word INT / WORD	Double word DINT / DWORD
%RB1.65534	%RX1.65534.0...%RX1.65534.7	%RW1.32767	
%RB1.65535	%RX1.65535.0...%RX1.65535.7		
Segment 2			
%RB2.0	%RX2.0.0...%RX2.0.7	%RW2.0	%RD2.0
%RB2.1	%RX2.1.0...%RX2.1.7		
%RB2.2	%RX2.2.0...%RX2.2.7	%RW2.1	
%RB2.3	%RX2.3.0...%RX2.3.7		
%RB2.65532	%RX2.65532.0...%RX2.65532.7	%RW2.32766	%RD2.16383
%RB2.65533	%RX2.65533.0...%RX2.65533.7		
%RB2.65534	%RX2.65534.0...%RX2.65534.7	%RW2.32767	
%RB2.65535	%RX2.65535.0...%RX2.65535.7		
Segment 7			
%RB7.0	%RX7.0.0...%RX7.0.7	%RW7.0	%RD7.0
%RB7.1	%RX7.1.0...%RX7.1.7		
%RB7.2	%RX7.2.0...%RX7.2.7	%RW7.1	
%RB7.3	%RX7.3.0...%RX7.3.7		
%RB7.65532	%RX7.65532.0...%RX7.65532.7	%RW7.32766	%RD7.16383
%RB7.65533	%RX7.65533.0...%RX7.65533.7		
%RB7.65534	%RX7.65534.0...%RX7.65534.7	%RW7.32767	
%RB7.65535	%RX7.65535.0...%RX7.65535.7		
Segment 15			
%RB15.0	%RX15.0.0...%RX15.0.7	%RW15.0	%RD15.0
%RB15.1	%RX15.1.0...%RX15.1.7		
%RB15.2	%RX15.2.0...%RX15.2.7	%RW15.1	
%RB15.3	%RX15.3.0...%RX15.3.7		
%RB15.65532	%RX7.65532.0...%RX7.65532. 7	%RW15.32766	%RD15.16383
%RB15.65533	%RX7.65533.0...%RX7.65533. 7		

Byte SINT / BYTE	Bit (byte-oriented) BOOL	Word INT / WORD	Double word DINT / DWORD
%RB15.65534	%RX7.65534.0...%RX7.65534. 7	%RW15.32767	
%RB15.65535	%RX7.65535.0...%RX7.65535. 7		

Only the first 4 kB in segment 0 are available for PM57x, i.e., %RB0.0..%RB0.4095 or %RW0.0..%RW0.2047 or %RD0.0..%RD0.1023. Only the first 1 kB in segment 0 are available for PM55x and PM56x, i.e., %RB0.0..%RB0.1023 or %RW0.0..%RW0.511 or %RD0.0..%RD0.255.

Access to the %R Area Using the Modbus Protocol:

As of PLC firmware version V2.1.3 and Control Builder Plus (CBP) 2.1.0 %M area (default) or %R area can be used for access via Modbus TCP or Modbus RTU. The selection is a parameter in the Modbus server settings.

With the help of the Modbus protocol, the segments 0 and 1 of the %R area can be accessed. Using this feature it is easy to access buffered data via Modbus TCP ↳ *Chapter 1.6.4.1 “System technology of CPU and overall system” on page 5395* or Modbus RTU ↳ *Chapter 1.6.4.1 “System technology of CPU and overall system” on page 5395*.

For the AC500 CPUs PM55x and PM56x, 1kB = %RB0.0 .. %RB0.1023 (i.e., not a complete segment) are available for the addressable flag area. Thus, not all Modbus addresses can be accessed.

System start-up / Program processing

See ↳ *Chapter 1.6.4.1.2.1 “System start-up / Program processing” on page 5412*.

Data backup and initialization

Initialization of variables, overview

The initialization of variables to 0 or to the initialization value is performed by switching power ON, by a reset or after downloading the user program.

AC500 supports all in IEC 61131-5 defined types of buffered data:

- VAR RETAIN - keep their value during online change
- VAR PERSISTENT - keep their value during online change and download
- VAR RETAIN PERSISTENT - keep their value during online change, download and power OFF/ON.

For this the variable are copied before and after download one by one. This copy process can take a long time.

So it is recommended to use VAR RETAIN PERSISTENT only, if there is a small amount (less 100) of buffered variables.

In addition to the IEC 61131-2 buffered variable types AC500 provides the so called PERSISTENT area (%R area). See detailed description ↳ *Chapter 1.6.4.1.1.6 “ Addressable PERSISTENT area (%R area)” on page 5405*.

The big advantage of the %R area is that there is no copy of variables necessary!

So its recommended to use the %R area for buffered variables.

If internal variables shall be buffered, these variables have to be marked as "VAR_RETAIN" or "VAR_RETAIN PERSISTENT". This applies to both the internal variables and the variables of the addressable flag area (%M area).

Variables of the addressable flag area (%M area) for AC500-eCo processor modules cannot be buffered.

The order of the internal RETAIN variables is only kept when using the online change command.

If the program is rebuilt, the order can change and, due to this, the buffered variables do no match. See CODESYS Remanent variables ↳ Chapter 1.4.1.3.9.7 “Remanent variables” on page 302.

Consider the description on the behavior of RETAIN variables on download.

The following table shows an overview of the initialization values of the individual variables:

Variable / Action	VAR and %MDx.x	VAR (%MDx.x) := Value	VAR_RETAIN (%MDx.x)	VAR_RETAIN (%MDx.x) := Value	VAR_PERSISTENT (%MDx.x)	VAR_PERSISTENT (%MDx.x) := Value	VAR_RETAIN PERSISTENT (%MDx.x)	VAR_RETAIN PERSISTENT (%MDx.x) := Value	%R not saved (:=Value)	%R saved (:=Value)
Download	0	Value	0	Value	unch.	unch.	unch.	unch.	0	unch.
Reset	0	Value	unch.	unch.	0	Value	unch.	unch.	0	unch.
Reset (cold)	0	Value	0	Value	0	Value	0	Value	0	unch.
Download after Reset (Origin)	0	Value	0	Value	0	Value	0	Value	0	0
Voltage OFF/ON without battery	0	Value	0	0	0	Value	0	0	0	0
Voltage ON/OFF with battery	0	Value	unch.	unch.	0	Value	unch.	unch.	0	unch.
Start/Stop	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.
Online Change	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.	unch.
Reset (Origin)	0	0	0	0	0	0	0	0	0	0

For AC500-eCo processor modules, there is a 1 kB VAR_RETAIN area and an 1 kB %R area.

Independently of an inserted battery, the memory values are stored in flash memory at power failures and are recovered when power returns.

For the %R area, the values can be configured in Automation Builder.

Declaration of buffered variables and constants

To guarantee the correct initialization or backing up of variables according to the table shown above, the following rules have to be observed when declaring variables.

Retentive internal variables

The variables have to be declared as VAR_RETAIN PERSISTENT or VAR_GLOBAL RETAIN PERSISTENT.

Example

(* Declaration in the global variable lists *)

```
VAR_GLOBAL RETAIN PERSISTENT
  byVar : BYTE;
  wVar  : WORD;
  rVar  : REAL;
END_VAR
```

(* Declaration in the program *)

```
VAR RETAIN PERSISTENT
  byVar1 : BYTE;
END_VAR
```

Buffered variables in %M area

The variables have to be declared as VAR_RETAIN PERSISTENT or VAR_GLOBAL RETAIN PERSISTENT.

See *Chapter 1.6.4.1.1.6 “ Addressable PERSISTENT area (%R area)” on page 5405*

Constants

Constants are declared as VAR_GLOBAL CONSTANT or VAR_CONSTANT.

Example

(* Declaration as global constants *)

```
VAR_GLOBAL CONSTANT
  byConst_1 : BYTE := 1;
END_VAR
```

(* Declaration in the program *)

```
VAR CONSTANT
  byConst_2 : BYTE := 2;
END_VAR
```

1.6.4.1.2 System processing

System start-up / Program processing

AC500-eCo processor modules do not have an integrated display and keyboard. All functions related to keyboard and display are not applied for those devices.

Definitions: PLC system start-up

Cold start

The AC500-eCo V3 does not use a battery for buffering the operand areas specified below, hence the “cold start” mode does not exist in this product.

- A cold start is performed by switching power OFF/ON if no battery is connected.
- All RAM memory modules are checked and erased.
- If no user program is stored in the Flash EPROM, the default values (as set on delivery) are applied to the interfaces.
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

Warm start

- A warm start is performed by switching power OFF/ON with a battery connected.
- All RAM memory modules are checked and erased except of the buffered operand areas and the RETAIN variables .
- If there is a user program stored in the Flash EPROM, it is loaded into RAM.
- The default operating modes set by the PLC configuration are applied.

RUN -> STOP

- RUN -> STOP means pressing the RUN function key on the PLC while the PLC is in run mode (AC500 PLC display "run", AC500-eCo PLC "RUN LED" is ON).
- If a user program is loaded into RAM, execution is stopped.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP", AC500-eCo "RUN LED" changes from ON to OFF.

START -> STOP

- START -> STOP means stopping the execution of the user program in the PLC's RAM using the menu item "Online/Stop" in the programming system.
- All outputs are set to FALSE or 0.
- Variables keep their current values, i.e., they are not initialized.
- The AC500 PLC display changes from "run" to "StoP".

Reset

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (exception: RETAIN variables) are set to their initialization values.
- Reset is performed using the menu item "Online/Reset" in the programming system or pressing the function key RUN for $\geq$ 5 s in STOP mode.

Reset (cold)

- Performs a START -> STOP process.
- Preparation for program restart, i.e., the variables (VAR) (also RETAIN variables) are set to their initialization values.
- Reset (cold) is performed using the menu item "Online/Reset (cold)" in the programming system.

Reset (original)

- Resets the controller to its original state (deletion of Flash, SRAM (%M, area, %R area, RETAIN, RETAIN PERSISTENT), Communication Module configurations and user program!).
- Reset (original) is performed using the menu item "Online/Reset (original)" in the programming system.

STOP -> RUN

- STOP -> RUN means short pressing the RUN function key on the PLC while the PLC is in STOP mode (AC500 PLC display "StoP", AC500-eCo "RUN LED" is ON). "RUN LED" is OFF of the toggle switch of an AC500-eCo CPU.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo "RUN LED" changes from OFF to ON.

STOP -> START

- STOP -> START means continuing the execution of the user program in the PLC's RAM using the menu item "Online/Start" in the programming system.
- If a user program is loaded into RAM, execution is continued, i.e., variables will not be set to their initialization values.
- The AC500 PLC display changes from "StoP" to "run", AC500-eCo PLC "RUN LED" changes from OFF to ON.

Download

- Download means loading the complete user program into the PLC's RAM. This process is started by selecting the menu item "Online/Download" in the programming system or after confirming a corresponding system message when switching to online mode (menu item "Online/Login").
- Execution of the user program is stopped.
- In order to store the user program to the Flash memory, the menu item "Online/Create boot project" must be called after downloading the program.
- Variables are set to their initialization values according to the initialization table.
- RETAIN variables can have wrong values as they can be allocated to other memory addresses in the new project!
- A download is forced by the following:
  - changed PLC configuration
  - changed task configuration
  - changed library management
  - changed compile-specific settings (segment sizes)
  - execution of the commands "Project/Clean all" and "Project/Rebuild All".

Online change

- After a project has changed, only these changes are compiled when pressing the key <F11> or calling the menu item "Project/Build". The changed program parts are marked with a blue arrow in the block list.
- The term Online Change means loading the changes made in the user program into the PLC's RAM using the programming system (after confirming a corresponding system message when switching to online mode, menu item "Online/Login").
- Execution of the user program is not stopped. After downloading the program changes, the program is re-organized. During re-organization, no further online change command is allowed. The storage of the user program to the Flash memory using the command "Online/Create boot project" cannot be initiated until re-organization is completed.
- Online Change is not possible after:
  - changes in the PLC configuration
  - changes in the task configuration
  - changes in the library management
  - changed compile-specific settings (segment sizes)
  - performing the commands "Project/Clean all" and "Project/Rebuild All".

Data buffering

- Data buffering, i.e., maintaining data after power ON/OFF, is only possible, if a battery is connected for AC500 CPU and the buffering will take place in FLASH with AC500-eCo V3 CPU. The following data can be buffered completely or in parts:
  - Data in the addressable flag area (%M area)
  - RETAIN variable
  - PERSISTENT variable (number is limited, no structured variables)
  - PERSISTENT area (%R area)
- In order to buffer particular data, the data must be excluded from the initialization process (see *Chapter 1.6.4.1.1.8 “Data backup and initialization” on page 5410*).

Start of the user program

The user program (UP) is started according to the following table. It is assumed that a valid user program is stored to the Flash memory.

See *Chapter 1.6.6.1.4 “Storage device details” on page 6334*.

Action	No memory card with UP installed Auto run = ON	No memory card with UP installed Auto run = OFF	Memory card with UP installed Auto run = ON	Memory card with UP installed Auto run = OFF
Voltage ON or Warm start or Cold start	UP is loaded from Flash into RAM and started from Flash.	No UP is loaded from Flash. When logging in, the message "No program available in the controller ..." is displayed.	UP is loaded from the memory card into Flash memory and RAM and then started from RAM.	UP is loaded from the memory card to the Flash memory. RAM remains empty. When logging in, the message "No program available in the controller ..." is displayed.
STOP -> RUN	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
STOP -> START	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.	UP in RAM is started.
Download [1)]	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.	The UP currently stored in the CPU's RAM is stopped. The built UP is loaded from the PC into the PLC's RAM.	The built UP is loaded from the PC into the PLC's RAM.
Online Change [2)]	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized.	Processing of the UP currently stored in the CPU's RAM is continued. The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized and processed.	The changes made to the UP are loaded from the PC into the PLC's RAM. The UP is reorganized.

Remarks:

[1)]: After the download is completed, the program is not automatically stored to the Flash memory. To perform this, create a boot project. If the UP is not stored to the Flash memory, the UP is reloaded from the Flash memory after voltage OFF/ON. Start the program either by pressing the RUN/STOP function key or using Automation Builder.

Processing times

Terms

The most important times for the use of the AC500 basic unit with or without connected remote modules are:

- The reaction time is the time between a signal transition at the input terminal and the signal response at the output terminal.
  For binary signals, the reaction time is composed of the input delay, the cycle time of the program execution and the bus transmission time if the system is expanded by remote modules.
- The cycle time determines the time intervals after which the processor restarts the execution of the user program.
  The cycle time has to be specified by the user. It should be greater than the program processing time of the user program plus the data transfer times and the related waiting times.
  The cycle time is also the time base for some time-controlled functions.
- The program processing time is the net time for processing the user program.

Program processing time

Statements	PM55x-xP PM56x-xP	PM57x	PM58x	PM59x
- Binary statements of the type:				
!M /M &M =M !NM /NM &NM =NM !M /M &M =SM !NM /NM &NM =RM Processing time for 1 instruction:	min. 0.08 µs	min. 0.06 µs	min. 0.05 µs	min. 0.002 µs
- Word statements of the type:				
!MW +MW -MW =MW !-MW -MW +MW =-MW !MW *MW :MW =MW !-MW *-MW :-MW =-MW Processing time for 1 instruction:	min. 0.1 µs	min. 0.09 µs	min. 0.06 µs	min. 0.004 µs
- Floating point:				
Processing time for 1 instruction:	min. 1.2 µs	min. 0.70 µs	min. 0.50 µs	min. 0.004 µs

Set cycle time

It is assumed that the processor always gets access in a moment with a worst-case condition.

The cycle time is stored in the task configuration and can be selected in steps of 1 ms. If the selected cycle time is too short, the processor will not be able to completely process the tasks assigned to it every cycle. This will result in a time delay.

If this lack of time becomes too large over several cycles, the processor aborts the program execution and outputs an error (E2).

For some function blocks, such as the PID controller, the error-free execution depends on an exact timing sequence. Make sure that there is a larger time reserve.

To check the correct cycle time, perform the following steps:

- Load the user program into the basic unit.
- Check the capacity utilization with "Online/PLC Browser/cpuload".
- Change the cycle time until the capacity utilization is below 80 %.

When setting the cycle time, consider the following values:

- Time for reading and copying the input signals from the I/O driver to the I/O image.
- Time for copying the input signals of the user task from the I/O image to the image memory.
- Program processing time.
- Time for copying the output signals of the user task from the image memory to the I/O image.
- Time for copying the output signals from the I/O image to the I/O driver and applying the I/Os to the I/O module.
- Receiving/sending interrupts from communication module telegrams within the cycle time.
- Receiving/sending interrupts from the serial interface within the cycle time.
- Task changes.
- Run time of the watchdog task.

Task configuration

The task model processes different kinds of tasks. Handling and configuration of the task processing depending on its priority is described in detail in the CODESYS task configuration section.

- *Chapter 1.4.1.4.8.1 “Overview” on page 390*

As of firmware V2.4

For a new project a task with the following properties is created:

- Type = cyclic
- Priority = 10
- Cycle time = t # 10 ms
- Program call = PLC_PRG
- Watchdog = t # 100 ms
- Sensitivity = 5

We recommend to create a task according to your needs.

Up to firmware V2.4

Specify a task in your project according to your needs.

All 32 priorities can be selected for the user tasks, where 0 is the highest priority and 31 the lowest. The default priority is 10.

Priorities lower than 10 are reserved for high-priority processes with a very short program execution time. The priorities 10 to 31 are intended for "normal" user tasks or tasks with a long program execution time.

NOTICE!
Using, for example, a priority lower than 10 for a task with a long program execution time can cause device errors, e.g. the CS31 bus.

System events

NOTICE!
System events belong to the group of online tasks. Floating point calculations are not allowed in online tasks, for they are not rechecked via software like IEC tasks.

Setting standard configuration

If the target setting configuration is changed, standard configuration can be restored:

1. Open CODESYS.
2. In the *"Resources"* tab, double-click *"PLC Configuration"*.
3. Select *"Menu Extras ➔ Standard Configuration"*.

1.6.4.1.3 User Management

With the help of the integrated user management, user groups with different access rights and authorizations can be defined. Configuration and handling of the user managment in Automation Builder and a AC500 V2 is decribed in an *application note*.

1.6.4.1.4 Real-time clock and battery

Notes on real-time clock

The real-time clock is an optional function for AC500-eCo V3 Basic processor modules (e.g. PM5012-x-ETH) and requires a TA5131-RTC. All other AC500-eCo V3 processor modules have an integrated real-time clock.

The real-time clock operates as a PC clock. It saves date and time to a DWORD in DT format (DATE AND TIME FORMAT), i.e., in seconds passed since the start time: 1 January 1970 at 00:00.

For AC500-eCo V3, Basic CPU with TA5131-RTC buffers the real-time clock for 7 days, and Standard/Pro CPU buffers the integrated real-time clock for 20 days. When the CPU is not powered over the buffering time, the real-time clock data will be cleared.

If a battery is connected and full, the real-time clock continues to run even if the control voltage is switched off.

If no battery is inserted or the battery is empty, the real-time clocks starts with the value 0 (=1970-01-01, 00:00:00).

When switching on the control voltage, the system clock of the operating system is set to the value of the real-time clock.

Real-time clock

Real-time clock with PLC browser

The PLC browser/PLC shell commands `date` and `time` are used to set the real-time clock.

The commands `date` <ENTER> or `time` <ENTER> display the current date and time of the real-time clock.

The command: `date yyyy-mm-dd`<ENTER> (year-month-day) sets the date.

The command: `time hh-mm-ss`<ENTER> (hours-minutes-seconds) sets the time.

Example:

The real-time clock should be set to 22 February 2005, 16:50.

1. Enter the date:

   `date 2005-02-22<ENTER>`

   ⇨ Display: `date 2005-02-22 Clock set to 2005-02-22 08:01:07`

   The time remains unchanged.

2. Enter the time:

   `time 16:50<ENTER>`

   ⇨ Display: `time 16:50 Clock set to 2005-02-22 16:50:00`

Real-time clock with user program

The following function blocks located in the folder "Realtime clock" of the system library *SysExt_AC500_Vxx.lib* can be used to set and display the real-time clock (RTC) with help of the user program:

Function block	Function
↬ *Chapter 1.5.4.16.1.2 “CLOCK” on page 1341*	Sets and displays the real-time clock with values for year, month, day, hours, minutes and seconds. Also the day of week is indicated (Mo=1, Tue=2, Wed=3, Thu=4, Fr=5, Sa=6, Su=0). Note: The week of day cannot be set. It is given by the real-time clock. The input DAY_SET is ignored.
↬ *Chapter 1.5.4.16.1.3 “CLOCK_DT” on page 1345*	Sets and displays the real-time clock in DT format, for example DT#2005-02-17-17:15:00.

The function blocks CLOCK and CLOCK_DT are described in the documentation for the system library *SysExt_AC500_Vxx.lib*.

AC500 battery

The AC500 battery buffers the following data in case of "control voltage off":

- Retentive variables in SRAM (VAR_RETAIN..END_VAR) ↬ *Chapter 1.6.4.1.1 “Inputs, outputs and flags for AC500 V2 products” on page 5395*
- Persistent data in %R area
- Date and time of the real-time clock

- *For AC500-eCo processor modules the battery is not required for retentive/persistent variables. These variables are stored in the flash memory when the processor module is shutting down.*
- *For AC500 (Standard) processor module PM595-4ETH-M, no battery is needed to buffer retentive/persistent variables as the MRAM will keep its content during power-off. Only the real-time clock is buffered by the battery.*

Further information:

- *Chapter 1.6.5.2.3.3 “Parameters of the processor module” on page 5839*
- *Chapter 1.7.3 “Diagnosis messages” on page 6429*

To prevent data loss when using the AC500 battery, the battery status should be periodically monitored by the user or by the user program.

If no battery is inserted or if the battery is empty, a warning (E4) is generated and the LED "ERR" lights up.

If no battery is required for the application (and thus no battery is inserted), a warning is generated and the error LED lights up each time the controller is switched on. To avoid this battery error indication, the parameter "Check Battery" is available under "CPU parameters" in the PLC configuration. The default setting of this parameter is "On", i.e., battery check is performed. If this parameter is set to "Off", the battery check is still performed and a corresponding error message is still generated each time the control voltage is switched on, but the system automatically quits this error and therefore the error LED does not light up (provided no further error exists).

Battery status

The battery status can be monitored either with the help of a user program on the PLC or in Automation Builder.

In the PLC browser of Automation Builder the command "batt" can be used . *Chapter 1.6.5.4.3 “AC500-specific PLC browser commands” on page 6222* The following is output:

0	Battery empty
20	Remaining battery charge below 20 %
100	Battery charge OK

In the user program, the battery status can be checked with the function *Chapter 1.5.4.16.1.1 “BATT” on page 1340* which is available in the folder "Battery" of the system library *SysExt_AC500_Vxx.lib*. The following is output:

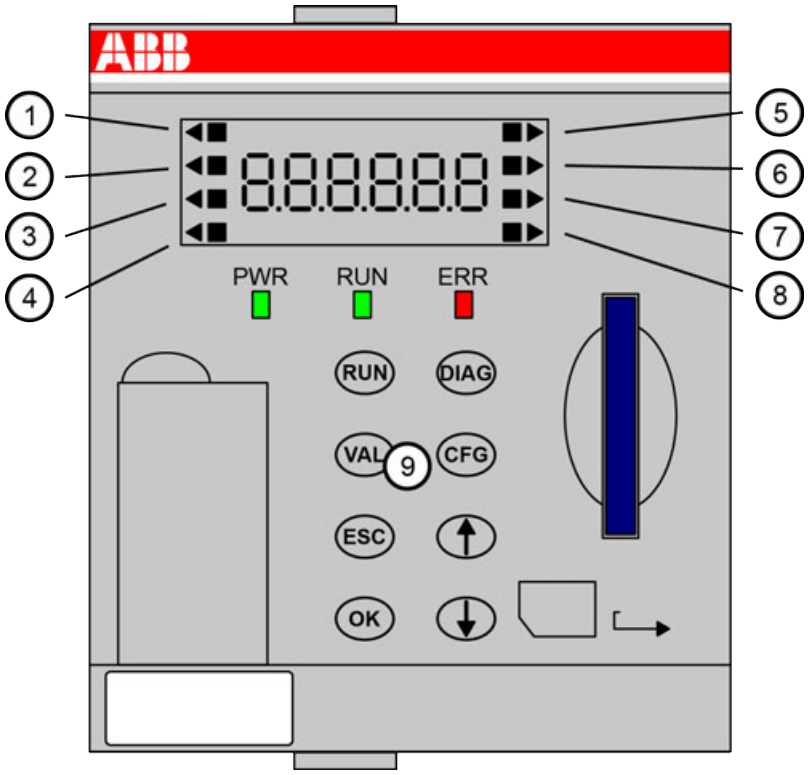
0	Battery empty
20	Remaining battery charge below 20 %, battery must be replaced
100	Battery charge OK

On the device, the battery status can be checked with the function keys of a processor module. *Chapter 1.6.4.1.5 “LEDs, display and function keys on the front panel” on page 5422*

Chapter 1.6.4.1.5.4.4 “Reading out values” on page 5440

1.6.4.1.5 LEDs, display and function keys on the front panel

Overview



The display of a processor module is equipped with a background-lighted 7-segment display. This display consists of 6 digits for plain text or error codes.

- error numbers and information on the error
- current settings of the processor module

Further, the display can be used for simple configurations such as address modifications.

Display indicators

- A black square (■) denotes the state/working activity of the corresponding object on the left/right side of the display. The black square flashes according to the device's activity, e.g. during data exchange on ETH1, COM1, etc.
- A black triangle (▶) points to the selected item/interface on the left/right side of the display to be configured or read. Further, it acts as a cursor for the count up/count down function keys.

> *A black triangle (◀) at the BATT item indicates a missing or uncharged battery.*

The indicators point to the following items on the left side of the display:

No.	On the left Side	Description
1	<empty item>	Cannot be used.
2	SYS (system)	Refers to the system status.
3	BATT (battery)	Refers to the battery status.
4	I/O bus	Refers to I/O bus connection.

The indicators point to the following items on the right side of the display:

No.	On the right side	Description
5	<empty item>	Cannot be used.
	ETH (Ethernet)	Refers to Ethernet interface or can be left empty.
	ETH1	Refers to the first Ethernet interface or can be left empty.
	ARCNET	Refers to ARCNET connection or can be left empty.
6	FBP (FieldBusPlug)	Refers to FBP (FieldBusPlug) or can be left empty.
	ETH2	Refers to the second Ethernet interface or can be left empty.
7	COM1	Refers to COM1 interface or can be left empty.
8	<empty item>	Cannot be used.
	COM2	Refers to COM2 interface or can be left empty.

9	Function keys on front panel

The following table describes the different displays of processor modules:

Processor module	Display variant	Description
PM5xx		Display of a processor module with FBP support.
PM5xx-ETH		Display of a processor module with FBP and Ethernet support.
PM5xx-ARC		Display of a processor module with ARCNET support.
PM5xx-2ETH		Display of a processor module with support for 2 Ethernet interfaces.

FieldBusPlug

All 127 FBP addresses can be set with Automation Builder. On the display only up to 99 FBP addresses can be set.

NOTICE!

Setting the FBP slave address by LEDs, display and function keys

Though it is possible to set the FBP slave address by using the LEDs, display and function keys, this is not recommended. Direct configuration on the device replaces address configurations defined via Automation Builder.

If the FBP address (set by Automation Builder or on the device) is different from the address assigned by the master device for the same station, the station cannot be accessed. The complete fieldbus cannot work properly or is completely down!

A modified address defined on the device is valid not until power OFF/ON of the processor module!

AC500 processor module equipped with FBP is always a slave device on the bus. To act as a master, the processor module should be equipped with a master communication module (e.g. CM592-DP for PROFIBUS DP).

Startup procedure of the PLC

The startup procedure depends on the selected PLC mode.

PLC Mode	Display	Startup Behavior
00		Boot project is loaded and PLC application is set to "RUN".
01		No boot project is loaded.
02		Boot project is loaded and the PLC application is set to "RUN" depending on CPU parameter "autostart".

State	Display	Description
0		Display on system start (power on).
1		PLC is in boot mode (see *Further information on page 5423*).
2		PLC is in initialization mode (see *Further information on page 5423*).
3		PLC is in STOP mode (see *Further information on page 5423*). **Same as status Stop in Automation Builder.**
4		PLC is in RUN mode (see *Further information on page 5423*). **Switch into RUN mode is only possible if a valid boot project is available in the flash memory.**

Description of LEDs

The LEDs below the display indicate the status of the processor module:

LED	State	Color	LED = ON	LED = OFF	LED flashes
Power LED (PWR)	Denotes the power supply state of the processor module	Green	Voltage is present (24 V DC)	Voltage is missing	-
Run LED (RUN)	Denotes the activity state of the processor module	Green	Processor module is in RUN mode	Processor module is in STOP mode	If the Run LED (RUN) flashes fast (4 Hz), the processor module is reading/writing the memory card. Together with a flashing Error LED (ERR), the processor module is writing the internal Flash. If the Run LED flashes slowly (1 Hz), a firmware update from the memory card is finished without errors.
Error LED (ERR)	Denotes an error	Red	An error has occurred. After pressing the DIAG function key, the error type and the error code will be displayed. The error codes can be shown by means of the DIAG and OK function keys.	No errors or only warnings have occurred.	If the Error LED flashes fast (4 Hz) together with a flashing Run LED, the firmware is updated and a Flash is written. If the Error LED flashes slowly (1 Hz) a firmware update from the memory card is finished with errors.

A running processor module is indicated with the state RUN on the display, a deactivated processor module is indicated with the state STOP. In both cases the display's backlight is off.

Description of the function keys

Overview

The processor module can be operated manually using the function keys on the front panel:

Function Key	Description	Description
	Run	Toggles between RUN and STOP mode. **Switching into RUN mode is only possible if an error free project has been created and downloaded with Automation Builder**.
	Value	Shows different state values of the processor module.

Function Key	Description	Description
ESC	Escape	Quits the current menu, submenu or function without saving.
OK	OK / Acknowledgement	Acknowledges the current value or selects a menu/ submenu. Changes that have been sent to the processor module successfully are confirmed with *donE* on the display.
DIAG	Diagnostic	Allows evaluation of error messages in detail.
CFG	Configuration	Allows navigation through addresses and system settings.
↑	Count up	Press the function key repeatedly in order to increase the value each time by 1. Keep the function key pressed in order to count up fast.
↓	Count down	Press the function key repeatedly in order to decrease the value each time by 1. Keep the function key pressed in order to count down fast.

Backlight is switched on for about 20 seconds by pressing any function key.

Start and stop PLC

Function key *RUN*

State	Description Menu level 0	Result on pressing one of the function keys		
		RUN	OK	ESC
0		Short click: State 1 is displayed. Long click (>5 sec): State 2 is displayed.	No action	No action
1	 PLC only in state RUN if a correct project is in RAM of PLC	State 0 is displayed. STOP - same as *Online stop* in Automation Builder (halt, no init of variables)		
2		***RUN*** LED=ON	Perform RESET same as *Online reset* in Automation Builder (stop and init variables) State 0 is displayed.	No RESET State 0 is displayed.

Configuration

Configuration: Version ≥ 2.4

The following tables describe how to configure a processor module manually. Processor module can be configured in RUN mode or STOP mode, hence this device state is called RUN/STOP mode in the following. By pressing the ***CFG*** function key repeatedly, you simply navigate through the statuses of the device.

However, to configure the processor module, use the ***OK***, ***ESC*** and ***CFG*** function keys alternately.

1. The processor module is in RUN/STOP mode. By pressing the ***CFG*** function key once, the IP configuration for Ethernet connection is displayed (IPETH1).
2. By pressing the ***CFG*** function key again, address configuration for Ethernet connection is displayed (Adr000).
3. By pressing the ***ESC*** function key, menu is aborted. You revert to the RUN/STOP mode.
4. By pressing the ***OK*** function key, submenu for Ethernet interface is opened. Either a static IP address can be set or a DHCP address is defined automatically.

Example: static IP address

The following example describes how to set a static IP address for Ethernet interface with a Processor Module with firmware Version ≥ 2.4.

Table 623: Function key CFG IPETH1 or IPETH2; DHCP not active

State	Description	Result on pressing one of the function keys		
		CFG	ESC	OK
1.1		State 4.2 is displayed.	Return into RUN/ STOP mode.	State 1.2 is displayed.
1.2	 IP Configuration (address, subnet mask, gateway)	State 1.3 is displayed.	Aborts the menu unchanged. Return to State 1.1	State 3.2 is displayed.
1.3	 Reset to production data (default settings)	State 1.4 is displayed.	Aborts the menu unchanged. Return to State 1.1	Activate RESET to default by pressing ***OK*** twice. Shows DONE, your settings are saved. Return into RUN/STOP mode.
1.4	 Activate DHCP Sets a DHCP address.	State 1.2 is displayed.	Aborts the menu unchanged. Return to State 1.1	Activate DHCP to default by pressing ***OK*** twice Shows DONE, your settings are saved. Return into RUN/STOP mode.

Table 624: Function key CFG IPETH1 or IPETH2; DHCP active

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
2.1		State 4.2 is displayed.	Aborts the menu unchanged. Return to State 0.	State 2.2 is displayed.
2.2	 DHCP active	State 2.3 is displayed.	Aborts the menu unchanged. Return to State 2.1.	--
2.3	 IP Configuration (address, subnet mask, gateway)	State 2.4 is displayed.	Aborts the menu unchanged. Return to State 2.1.	State 3.2 is displayed.
2.4	 Reset to production data (default settings)	--	Aborts the menu unchanged. Return to State 2.1.	Activate RESET to default by pressing ***OK*** twice Shows DONE, your settings are saved. Return into RUN/STOP mode.

Function key *CFG* submenu STATIC

State	Description - Submenu 3	Result on pressing one of the function keys		
		CFG	ESC	OK
3.1	 IP Configuration (address, subnet mask, gateway)	State 2.4 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	State 3.2 is displayed.
3.2	 IP address A1-A4 ★ Number is blinking if value has changed and is not yet sent to CPU	State 3.3 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

State	Description - Submenu 3	Result on pressing one of the function keys		
		CFG	ESC	OK
3.3	 Subnet mask N1-N4 ★ Number is blinking if value has changed and is not yet sent to CPU	Press ***CFG*** from n2 to n4 State 3.4 is displayed.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2)	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.
3.4	 Gateway G1-G4 ★ Number is blinking if value has changed and is not yet sent to CPU	Press ***CFG*** from g2 to g4 State 3.2 is displayed again.	Aborts the menu unchanged. Return to State 1.1 (sub menu IPETH1 or IPETH2) Aborts the menu unchanged. Return to State 1.	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

Function key *CFG* sub menu ADR

State	Description - Submenu 4	Result on pressing one of the function keys		
		CFG	ESC	OK
4.1		State 4.2 is displayed.	Aborts the menu unchanged. Return to State 1	*DHCP not active*: State 1.2 is displayed *DHCP active*: State 2.2 is displayed

State	Description - Submenu 4	Result on pressing one of the function keys		
		CFG	ESC	OK
4.2	 Change the values with the ***Count up/Count down*** function keys starting with current value. ★ Number is blinking if value has changed and is not yet sent to CPU	Next interface is displayed.	Aborts the menu unchanged. Return to State 4.1	Sends changed values to CPU and go to default menue RUN/STOP Displays: DONE New settings stored in CPU. or: FAIL Failed to write new settings to CPU.

Navigation through the display

Navigation through the display of a processor module as of version ≥ 2.4 starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the three function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - Main menu	Result on pressing one of the function keys		
		CFG	ESC	OK
0	 The processor module is in RUN/STOP mode.	State 1 is displayed.	Remains in RUN/ STOP mode.	Remains in RUN/ STOP mode.
1	 IP configuration for Ethernet interface - if connected.	State 2 is displayed.	Return into RUN/ STOP mode.	Opens the submenu for IP configuration. Refer to State 1.1 in the following table.
2	 Address configuration for Ethernet interface. Change the values with the ***Count up/Count down*** function keys. ★ Number is blinking if value has changed and is not yet sent to CPU	State 3 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 2 is displayed.

State	Description - Main menu	Result on pressing one of the function keys		
		CFG	ESC	OK
3	 Address configuration for FBP connection. Change the values with the **Count up/ Count** down function keys.	State 4 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 3 is displayed.
4	 Address configuration for COM1 interface - if connected. Change the values with the ***Count up/Count down*** function keys.	State 5 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 4 is displayed.
5	 Address configuration for COM2 interface - if connected. Change the values with the ***Count up/Count down*** function keys.	State 6 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 5 is displayed.
6	 Startup mode configuration for the processor module. Change the values with the ***Count up/Count down*** function keys.	State 7 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 1 is displayed.
7	 ID configuration for FlexConf for the processor module. Change the values with the ***Count up/Count down*** function keys.	State 8 is displayed.	Return into RUN/ STOP mode.	Your settings are saved. State 1 is displayed.
The following States are only displayed if CM597-ETH is plugged:				
8	 IP configuration for the Communication Module in slot 1 - if Ethernet interface is set.	State 9 is displayed.	Aborts the menu unchanged. Return to State 7.	Opens the submenu for slot 1 configuration. Refer to State 8.1 in the following table.

State	Description - Main menu	Result on pressing one of the function keys		
		CFG	ESC	OK
9	 IP configuration for the Communication Module in slot 1 - if Ethernet interface is set.	State 10 is displayed.	Aborts the menu unchanged. Return to State 7.	Opens the submenu for slot 2 configuration. Refer to State 8.1 in the following table.
10	 IP configuration for the Communication Module in slot 1 - if Ethernet interface is set.	State 11 is displayed.	Aborts the menu unchanged. Return to State 7.	Opens the submenu for slot 3 configuration. Refer to State 8.1 in the following table.
11	 IP configuration for the Communication Module in slot 1 - if Ethernet interface is set.	State 8 is displayed.	Aborts the menu unchanged. Return to State 7.	Opens the submenu for slot 4 configuration. Refer to State 8.1 in the following table.

States for CM597-ETH

The following states are only displayed if CM597-ETH is plugged

State	Description - Submenu 1	Result on pressing one of the function keys		
		CFG	ESC	OK
8.1	 Submenu for slot 1 configuration opens. Sets a DHCP address.	State 8.2 is displayed.	Aborts the menu unchanged. Return to State 1.	Opens the submenu 2. Refer to State 1.2.1 in the following table.
8.2	 Configuration of a static IP address.	Opens the submenu 2. Refer to State 1.2.1 in the following table.	Aborts the menu unchanged. Return to State 1.	No functionality.

***CFG* Submenu 2 *STATIC* IP-settings**

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
1.2.1	 Submenu opens. Configuration of address A1-A4. Change the values with the ***Count up/Count down*** function keys.	State 1.2.2 is displayed.	Aborts the menu unchanged. Return to State 1.	No functionality.
1.2.2	 Configuration of subnet mask N1-N4. Change the values with the ***Count up/Count down*** function keys.	State 1.2.3 is displayed.	Aborts the menu unchanged. Return to State 1.	Your settings are saved. State 1.1 is displayed.
1.2.3	 Configuration of gateway G1-G4. Change the values with the ***Count up/Count down*** function keys.	State 1.2.1 is displayed.	Aborts the menu unchanged. Return to State 1.	Your settings are saved. State 1.1 is displayed.
The following States are only displayed if CM597-ETH is plugged:				
8.1.1	 Submenu opens. Configuration of address A1-A4. Change the values with the ***Count up/Count down*** function keys.	State 8.1.2 is displayed.	Aborts the menu unchanged. Return to State 8.	No functionality.
8.1.2	 Configuration of subnet mask N1-N4. Change the values with the ***Count up/Count down*** function keys.	State 8.1.3 is displayed.	Aborts the menu unchanged. Return to State 8.	No functionality.
8.1.3	 Configuration of gateway G1-G4. Change the values with the ***Count up/Count down*** function keys.	State 8.1.1 is displayed.	Aborts the menu unchanged. Return to State 8.	No functionality.

Configurations: Version < 2.4

The following tables describe navigation through the display of a Processor Module with processor module Firmware Version < 2.4. Navigation starts with the processor module being in RUN/STOP mode (State 0). By pressing one of the three function keys a certain action is triggered. The result of this action is described in the result columns of the tables.

State	Description - Main menu	Result on pressing one of the function keys		
		CFG	ESC	OK
0	 The processor module is in RUN/STOP mode.	State 1 is displayed.	Remains in RUN/ STOP mode.	Remains in RUN/ STOP mode.
1	 *IP-Set*: IP configuration.	State 2 is displayed.	Return to RUN/STOP mode.	Opens the sub-menu 1 for IP configuration. Refer to State 1.1 in the following table.
2	 *Adr000*: Address configuration for Ethernet interface. Change the values with the ***Count up/Count down*** function keys.	State 3 is displayed.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.
3	 Address configuration for FBP connection. Change the values with the ***Count up/ Count*** down function keys.	State 4 is displayed.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.
4	 *Adr000*: Address configuration for COM1 interface - if connected. Change the values with the ***Count up/ Count down*** function keys.	State 5 is displayed.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.
5	 *Adr000*: Address configuration for COM2 interface - if connected. Change the values with the ***Count up/ Count down*** function keys.	State 6 is displayed.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.
6	 Startup mode configuration for the processor module. Change the values with the ***Count up/Count down*** function keys.	State 7 is displayed.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.

State	Description - Main menu	Result on pressing one of the function keys		
		CFG	ESC	OK
7	 ID configuration for FlexConf (multiple hardware configurations) for the processor module.	Return to State 1.	You revert to the RUN/ STOP mode.	Return to RUN/ STOP mode.

Function key *CFG* submenue 1

State	Description - Submenu 1	Result on pressing one of the function keys		
		CFG	ESC	OK
1.1	 Configuration of slot 1 - 4.	No functionality.	Aborts the menu unchanged. Return to State 1.	State 1.2 is displayed.
1.2	 Configuration of a static IP address.	State 1.3 is displayed.	Aborts the menu unchanged. Return to State 1.	Opens the submenu 2. Refer to State 1.2.1 in the following table.
1.3	 Reset option is displayed.	State 1.4 is displayed.	Aborts the menu unchanged. Return to State 1.	Reset to Production data (*reset*).
1.4	 Configuration of a DHCP address.	State 1.2 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.

CFG Submenu 2 _STATIC_ IP-settings

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
1.2.1	 Address configuration for A1. Change the values with the ***Count up/Count down*** function keys.	State 1.2.2 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
1.2.2	 Address configuration for A2. Change the values with the ***Count up/Count down*** function keys.	State 1.2.3 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.3	 Address configuration for A3. Change the values with the ***Count up/Count down*** function keys.	State 1.2.4 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.4	 Address configuration for A4. Change the values with the ***Count up/Count down*** function keys.	State 1.2.5 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.5	 Configuration of the subnet mask for N1. Change the values with the ***Count up/Count down*** function keys.	State 1.2.6 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.6	 Configuration of the subnet mask for N2. Change the values with the ***Count up/Count down*** function keys.	State 1.2.7 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.7	 Configuration of the subnet mask for N3. Change the values with the ***Count up/Count down*** function keys.	State 1.2.8 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.

State	Description - Submenu 2	Result on pressing one of the function keys		
		CFG	ESC	OK
1.2.8	 Configuration of the subnet mask for N4. Change the values with the ***Count up/Count down*** function keys.	State 1.2.9 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.9	 Gateway configuration for G1. Change the values with the ***Count up/Count down*** function keys.	State 1.2.10 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.10	 Gateway configuration for G2. Change the values with the ***Count up/Count down*** function keys.	State 1.2.11 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.11	 Gateway configuration for G3. Change the values with the ***Count up/Count down*** function keys.	State 1.2.12 is displayed.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.
1.2.12	 Gateway configuration for G4. Change the values with the ***Count up/Count down*** function keys.	Return to State 1.2.1.	Aborts the menu unchanged. Return to State 1.	Return to RUN/ STOP mode.

FBP slave address

It is recommended to set FBP slave address by using Automation Builder software. Neverthless it is possible, however not recommended, to set this address by LED display of the processor module.

All 127 FBP addresses can be set with Automation Builder. On the display only up to 99 FBP addresses can be set.

NOTICE!

Setting the FBP slave address by LEDs, display and function keys

Though it is possible to set the FBP slave address by using the LEDs, display and function keys, this is not recommended. Direct configuration on the device replaces address configurations defined via Automation Builder.

If the FBP address (set by Automation Builder or on the device) is different from the address assigned by the master device for the same station, the station cannot be accessed. The complete fieldbus cannot work properly or is completely down!

A modified address defined on the device is valid not until power OFF/ON of the processor module!

AC500 processor module equipped with FBP is always a slave device on the bus. To act as a master, the processor module should be equipped with a master Communication Module.

The FBP must have a properly assigned slave module address. The AC500 CPU gives the FBP an address at system power-up. The address can be set with the use of the LED display and the softkeys on the front panel.

To configure the FBP address, please follow the procedure described below:

First select the item to be configured by pressing the ***CFG*** key. The CPU changes to configuration mode and a small triangle is displayed on the LCD on the first right up position of the display beside the ETH inscription and the already configured address is diplayed.	CFG	
Press one time more the ***CFG*** key to select the item FBP. The FBP address is shown and a small triangle is displayed on the LCD beside the FBP inscription.	CFG	
Press the arrow keys ***UP*** or ***DOWN*** to increase or decrease the address. The modified value blinks to indicate that it differs from the previously stored one.	↑ ↓	
Once the desired address is reached, press ***OK*** to accept and quit. Or press ESC to exit the menu without saving the changes. The CPU status is then displayed run or Stop.	OK or ESC	

The modified address will only be valid after a Power OFF / Power ON.

A processor module equipped with a FieldBusPlug is always a slave device on the bus. To act as a master, an AC500 CPU should be equipped with master Communication Modules.

The FieldBusPlug module has to be connected to the master device and the power supply has to be provided. Please use the dedicated accessories to the FBP used for the desired Fieldbus.

Reading out values

Reading out values

The following settings of the processor module can be read out by pressing the function key ***VAL*** repeatedly:

1. Displays time of the processor module (hh.mm.ss).
2. Displays date of the processor module (yy.mm.dd).
3. Displays state of battery (ub 100 = 100%, ub 020 = 20% or ub 000 = empty).
4. Displays version of display firmware (e.g. d 2.5 r (= display version 2.5 release).
5. Displays version of CPU firmware (e.g. C 2.3.3r (= CPU version 2.5 release).
6. Displays CPU type.
7. Displays default text (RUN/STOP).

Reading out diagnosis messages on the CPU

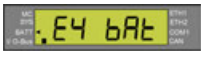
Table 625: Example: no diagnosis message in status list

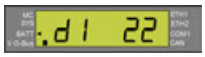
State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1			No action	No action	Return into RUN/STOP mode.	

Table 626: Example: diagnosis messages in status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	The processor module is in RUN/STOP mode.	State 1 is displayed	-	-	-	-
1	 Number of diagnosis messages; here 4		Go to first/ next diagnosis message in status list (e.g., state 2)	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Return into RUN/STOP mode.
2	 Diagnosis message example: *Error battery empty or missing* Toggling between state 2 and 3	Selects displayed diagnosis message and shows details ↪ *Table 627 “Example: error battery empty or missing” on page 5441*	Go to first/ next diagnosis message in status list	Go to last/ previous diagnosis message in status list	Return into RUN/STOP mode.	Acknowledge and return into RUN/STOP mode.
3	 Error ID example Toggling between state 2 and 3					

Table 627: Example: error battery empty or missing

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
0	 E4 = error severity 4 bAt = subdevice battery Toggling between state 0 and 1	State 2 is displayed	State 2 is displayed	State 6 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list

State	Display	Result on pressing one of the function keys				
		DIAG	↓	↑	ESC	OK
1	 Error ID example Toggling between state 0 and 1					
2	 Error number 8 Battery is missing or empty		State 3 is displayed	State 0 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Displays state 0 Return to diagnosis status list
3	 Detail 1 Subdevice 22: battery		State 4 is displayed	State 2 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
4	 Detail 2 Error type 0: device		State 5 is displayed	State 3 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
5	 Detail 3 Error type number 0: device itself		State 6 is displayed	State 4 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list
6	 Detail 4 Additional information 0: none		State 1 is displayed	State 5 is displayed	State 0 is displayed Return to diagnosis status list	State 0 is displayed Return to diagnosis status list

1.6.4.1.6 Onboard technologies

Ethernet

Ethernet protocols and ports for AC500 V2 products

Firmware and control builder plus until V2.3.x Supported Ethernet Protocols in AC500 in CPU Firmware and Control Builder Plus until V2.3.x:

Description	CPU							
	PM55x-/ PM56x-ETH	PM572	PM573-ETH	PM582	PM583-/ PM590-ETH	PM591-ETH	PM592-ETH	up from CPU firmware revision
Onboard Ethernet	x		x		x	x	x	V2.0.2
DHCP	x		x		x	x	x	V2.0.2
BOOTP								V2.0.2
ABB netConfig	x		x		x	x	x	V2.0.2
Hilscher IPconfig								V2.0.2
Online access with driver 3S TCP/IP on port 1201	x		x		x	x	x	V2.0.2
Online access with driver ABB TCP/IP Level 2 AC on port 1200	x		x		x	x	x	V2.0.2
Online access with driver 3S TCP/IP Level 2 Route on port 1201	x		x		x	x	x	V2.1.3
Modbus TCP Slave	x		x		x	x	x	V2.0.2
Modbus TCP Master with POU ↳ *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202*	x		x		x	x	x	V2.0.2
UDP with AC31 header	x		x		x	x	x	V2.0.2
UDP no AC31 header (standard UDP)	x		x		x	x	x	V2.0.2
TCP/IP out of user program with library SysLibSockets.lib	x		x		x	x	x	V2.0.2
Web server on PLC with web visualization and JAVA applet	x		x		x	x	x	V2.0.6
Web server on PLC with support of JAVA script (no applet)	x		x		x	x	x	V2.1.3
SNTP (Simple Network Time Protocol) client and server			x		x	x	x	V2.0.2
SMTP client (Simple Mail Transfer Protocol) send email out of user program with POU ↳ *Chapter 1.5.4.13.1.8 “ETH_SMTP_EMAIL_SEND” on page 1215*	x [1)]		x		x	x	x	V2.1.3

Description	CPU							up from CPU firmware revision
	PM55x-/ PM56x-ETH	PM572	PM573-ETH	PM582	PM583-/ PM590-ETH	PM591-ETH	PM592-ETH	
IEC60870-5-104 control station			[2])		x	x	x	V2.0.2
IEC60870-5-104 substation			x		x	x	x	V2.0.2
FTP server	x		x		x	x	x	V2.1.3
PING out of user program with POU ↳ *Chapter 1.5.4.13.1.2 "ETH_ICMP_PING" on page 1197*	x		x		x	x	x	V2.1.3
DNS - supplies the IP address of a host by its name with POU ↳ *Chapter 1.5.4.13.1.1 "ETH_DNS_RESOLVE" on page 1194*	x		x		x	x	x	V2.1.3

Remarks:

[1]): SMTP is not available for PM5x5-xx-ETH modules.

[2]): Not recommended. The control station claims nearly all resources of the processor module.

Firmware as of V2.4.x and Automation Builder

Supported Ethernet Protocols in AC500 in CPU Firmware as of V2.4.x and Automation Builder

Description	CPU														CM	up from CPU firmware revision
	PM554-ETH	PM556-ETH	PM56x-ETH	PM572	PM573-ETH	PM582	PM583-ETH	PM585-ETH	PM590-ETH	PM590-ARC	PM591-ETH	PM591-2ETH	PM592-ETH	PM595-4ETH	CM597-ETH	
Onboard Ethernet	x		x		x		x		x		x					V2.0.2
Onboard Ethernet													x			V2.1.3
Onboard Ethernet		x														V2.3.1
Onboard Ethernet												x		x		V2.4.0

Description	CPU														CM	up from CPU firmware revision
	PM554-ETH	PM556-ETH	PM56x-ETH	PM572	PM573-ETH	PM582	PM583-ETH	PM585-ETH	PM590-ETH	PM590-ARC	PM591-ETH	PM591-2ETH	PM592-ETH	PM595-4ETH	CM597-ETH	
Onboard Ethernet								x								V2.5.0
External Ethernet communication module CM597-ETH				x	x	x	x		x	x	x	x	x	x	x	V2.4.0

Table 628: Ethernet protocols

Description	CPU								CM	up from CPU firmware revision
	PM55x-/ PM56x-ETH	PM573-ETH	PM583-ETH	PM590-ETH	PM591-ETH	PM591-2ETH	PM592-ETH	PM595-4ETH	CM597-ETH	
DHCP	x	x	x	x	x	x	x	x	x	V2.0.2
BOOTP									x	V2.0.2
ABB netConfig	x	x	x	x	x	x	x	x	x	V2.0.2
Hilscher IPconfig									x	V2.0.2
Online access with driver 3S Tcp/Ip on port 1201	x	x	x	x	x	x	x	x	x	V2.0.2
Online access with driver ABB Tcp/Ip L2 AC on port 1200	x	x	x	x	x	x	x	x	x	V2.0.2
Online access with driver 3S Tcp/Ip L2 Route on port 1201	x	x	x	x	x	x	x	x	x	V2.1.3
Modbus TCP Slave	x	x	x	x	x	x	x	x	x	V2.0.2
Modbus TCP Master with POU *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202*	x	x	x	x	x	x	x	x	x	V2.0.2
UDP with AC31 header	x	x	x	x	x	x	x	x	x	V2.0.2
UDP no AC31 header (standard UDP)	x	x	x	x	x	x	x	x	x	V2.0.2
TCP/IP out of user program with library SysLibSockets.lib	x	x	x	x	x	x	x	x		V2.0.2

Description	CPU								CM	up from CPU firmware revision
	PM55x-/ PM56x-ETH	PM573-ETH	PM583-ETH	PM590-ETH	PM591-ETH	PM591-2ETH	PM592-ETH	PM595-4ETH	CM597-ETH	
Web server on PLC with IEC 61131-3 web visualization and JAVA applet	x	x	x	x	x	x	x	x		V2.0.6
Web server on PLC with support of JAVA script (no applet)	x	x	x	x	x	x	x	x		V2.1.3
SNTP (Simple Network Time Protocol) client and server		x	x	x	x	x	x	x		V2.0.2
SNTP (Simple Network Time Protocol) client	x									V2.4.0
SMTP client (Simple Mail Transfer Protocol) send email out of user program with POU *Chapter 1.5.4.13.1.8 “ETH_SMTP_EMAIL_SEND” on page 1215*	x	x	x	x	x	x	x	x		V2.1.3
IEC60870-5-104 control station			x	x	x	x	x	x		V2.0.2
IEC60870-5-104 substation		x	x	x	x	x	x	x		V2.0.2
FTP server	x	x	x	x	x	x	x	x		V2.1.3
PING out of user program with POU *Chapter 1.5.4.13.1.2 “ETH_ICMP_PING” on page 1197*	x	x	x	x	x	x	x	x	x	V2.1.3
DNS - supplies the IP address of a host by its name with POU *Chapter 1.5.4.13.1.1 “ETH_DNS_RESOLVE” on page 1194*	x	x	x	x	x	x	x	x		V2.1.3
IEC60870-5-104 2nd connection		x	x	x	x	x	x	x		V2.4.0

Description	CPU								CM	up from CPU firmware revision
	PM55x-/ PM56x-ETH	PM573-ETH	PM583-ETH	PM590-ETH	PM591-ETH	PM591-2ETH	PM592-ETH	PM595-4ETH	CM597-ETH	
IEC60870-5-104 2nd port						x		x		V2.4.0
Connection to MySQL data base with library MySQL_AC500_V22.lib	x	x	x	x		x		x		V2.2.0

Overview of protocols, sockets and ports

Protocol	Port	Sockets
DHCP	67	1 socket during startup
BOOTP	67	
ABB netConfig	24567	1 permanent socket if configured in Automation Builder
Hilscher IPconfig	25383	1 permanent socket if configured in Automation Builder for CM597-ETH
Online access with driver 3S Tcp/Ip	1201	1 socket per connection + 1 listen
Online access with driver ABB Tcp/Ip L2 AC	1200	1 socket per connection + 1 listen
Online access with driver 3S Tcp/Ip L2 Route	1201	1 socket per connection + 1 listen
Modbus TCP Server	502	1 socket per server connection, number of server connections is configurable in Automation Builder
Modbus TCP Master with POU *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202*	random	1 socket per instance of ETH_MOD_MAST
UDP with AC31 header	0 ... 65535	1 socket if enabled in Automation Builder
UDP no AC31 header (standard UDP)	0 ... 65535	1 socket per connection
TCP/IP out of user program with library SysLibSockets.lib	0 ... 65535	1 socket per connection
Web server on PLC with web visualization and JAVA applet	80	2 system sockets + 2 per connection
Web server on PLC with support of JAVA script (no applet)	80	2 system sockets + 2 per connection
SNTP (Simple Network Time Protocol)	123	1 permanent socket if configured in Automation Builder
SMTP client (Simple Mail Transfer Protocol) send email out of user program with POU *Chapter 1.5.4.13.1.8 “ETH_SMTP_EMAIL_SEND” on page 1215*	25	1 per connection if POU is enabled

Protocol	Port	Sockets
IEC60870-5-104 control station	random	1 per connection
IEC60870-5-104 substation	2404	1 per connection
FTP server	21	1 per session, max. 4 allowed
PING out of user program with POU ↳ *Chapter 1.5.4.13.1.2 “ETH_ICMP_PING” on page 1197*	none	1 per POU if POU is enabled
DNS - supplies the IP address of a host by its name with POU ↳ *Chapter 1.5.4.13.1.1 “ETH_DNS_RESOLVE” on page 1194*	53	1 per POU if POU is enabled

Numbers and usage of Ethernet sockets

Overview

Device	Number of sockets	Hereof system sockets	Hereof user sockets
PM55x-ETH	16	3	13
PM56x-ETH	16	3	13
PM573-ETH	16	3	13
PM583-ETH	25	3	22
PM585-ETH	32	3	29
PM590-ETH	32	3	29
PM591-ETH	32	3	29
PM592-ETH	32	3	29
PM556-ETH	16	3	13
PM591-2ETH	64	4	60
PM595-4ETH-F	64	4	60
PM595-4ETH-M	64	4	60
CM597-ETH	18	3	15

Online access - System sockets

3 sockets are required for Online Access:

- 1x driver 3S Tcp/Ip or 3S Tcp/Ip L2 Route on port 1201
- 1x driver ABB Tcp/Ip AC on port 1200
- 1x listen on port 1200/1201 if PLC has established online connection

↳ *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*

Online access can be established from:

- CODESYS V2.3.9.x (Online -> Login)
- OPC Server
- Control Builder Plus up from V2.1.0/Automation Builder
- Panel CP600 series

Each established connection needs 1 socket.

In addition, 1 socket on port 1200 and 1 on port 1201 is listening.

Example

- Startup PLC
  => 2 sockets:
  1 socket on port 1200 and 1 on port 1201 listen
- Login from CODESYS V2.3.9.x via driver ABB Tcp/Ip L2 AC on port 1200
  => 3 sockets:
  2 sockets on port 1200 (1x Online, 1x Listen) and 1 socket on port 1201 listen
- Additional login out of PS501 Control Builder Plus with the same driver:
  3 sockets on port 1200 (2x Online, 1x Listen) and 1 socket on port 1201 listen
  => 4 sockets:
- Additional connect CP600 via driver 3S Tcp/Ip L2 Route => 5 sockets:
  3 sockets on port 1200 (2x Online, 1x Listen) and
  2 sockets on port 1201 (1x Panel, 1x Listen)

Maximum numbers of sockets per protocol

Table 629: Maximum numbers of sockets per protocol up to firmware V2.3.x

No.	Description	CPU							up from CPU firmware revision
		PM55x-ETH	PM56x-ETH	PM573-ETH	PM583-ETH	PM590-ETH	PM591-ETH	PM592-ETH	
8	Modbus TCP Slave	12	12	12	12	12	12	12	V2.0.2
11	UDP no AC31 header	12	12	12	12	12	12	12	V2.0.2
13	Web server on PLC with web visualization and JAVA applet	7	7	9	18	25	25	25	V2.0.6
14	Web server on PLC with support of JAVA script (no applet)	7	7	9	18	25	25	25	V2.1.3
19	FTP server	2	2	2	2	4	4	4	V2.1.3

Maximum numbers of sockets per protocol as of firmware V2.4.x

No.	Description	CPU										CM	up from CPU firmware revision
		PM55x-ETH	PM56x-ETH	PM573-ETH	PM583-ETH	PM585-ETH	PM590-ETH	PM591-ETH	PM591-2 ETH	PM592-ETH	PM595-4 ETH	CM597-ETH	
8	Modbus TCP Slave	12	12	12	12	12	12	12	24	12	24	12	V2.0.2
11	UDP no AC31 header	12	12	12	12	12	12	12	24	12	24	12	V2.0.2

No.	Description	CPU										CM	up from CPU firmware revision
		PM5 5x-ETH	PM5 6x-ETH	PM5 73-ETH	PM5 83-ETH	PM58 5-ETH	PM5 90-ETH	PM5 91-ETH	PM5 91-2 ETH	PM5 92-ETH	PM5 95-4 ETH	CM5 97-ETH	
13	Web server on PLC with web visualization and JAVA applet	7	7	9	18	25	25	25	50	25	50	0	V2.0.6
14	Web server on PLC with support of JAVA script (no applet)	7	7	9	18	25	25	25	50	25	50	0	V2.1.3
19	FTP server	2	2	2	2	4	4	4	8	4	8	0	V2.1.3

The total number of sockets used by following protocols (over all Interfaces) must not exceed 32:

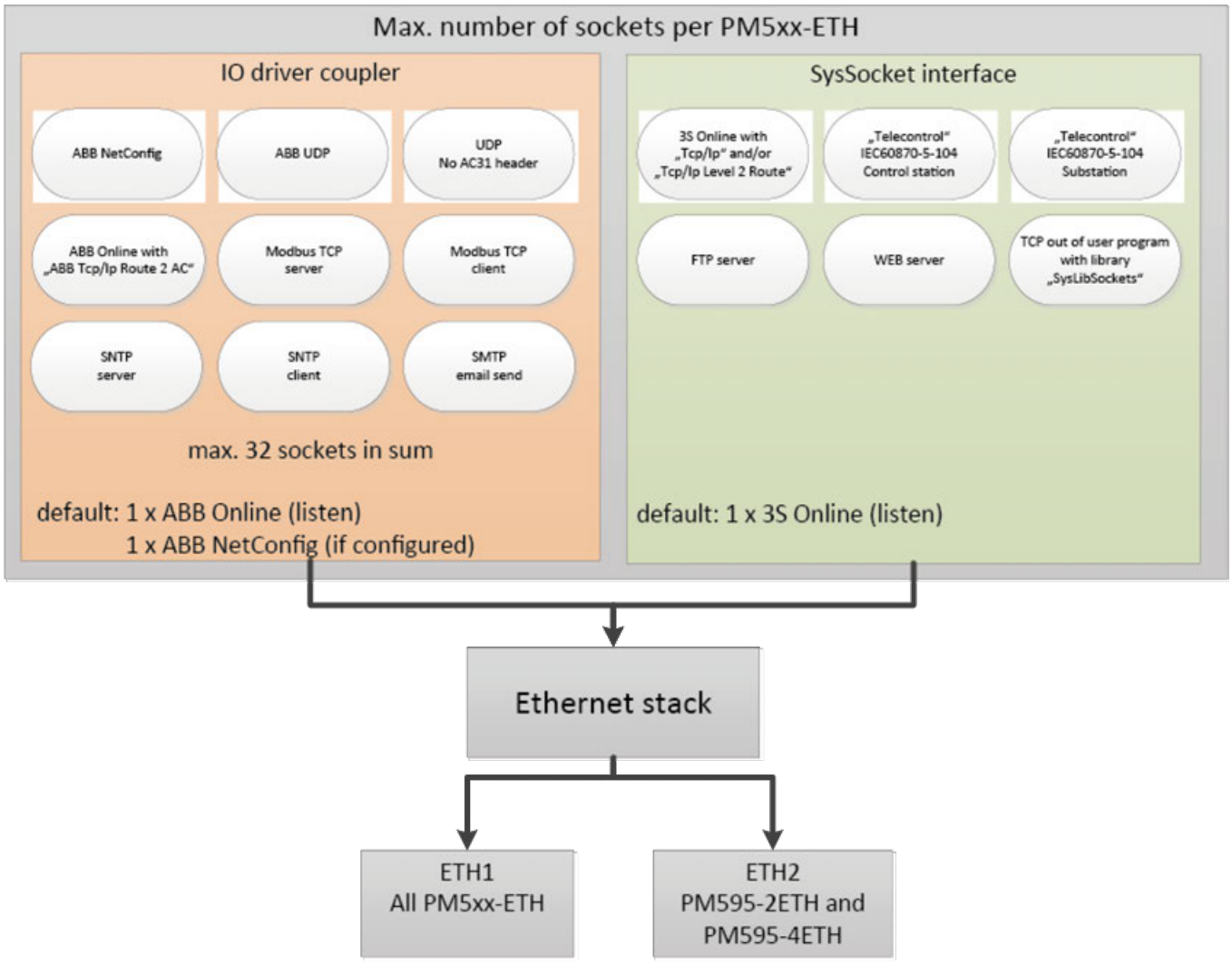
- *UDP data exchange*
- *UDP no AC31 Header*
- *ABB netConfig*
- *Online with ABB TCP/IP Level 2AC (listen, connected, routing)*
- *SNTP (onboard only)*
- *SMTP (onboard only)*
- *Modbus Client*
- *Modbus TCP server*

For CPUs with multiple Onboard Ethernet interface (e.g. PM591-2ETH, PM595-4ETH) the maximum number of sockets per protocol is the sum for all Ethernet interfaces.

Example: Max number of Modbus TCP server for PM591-2ETH is 24. If 1st Ethernet interface ETH1 uses 11 Modbus server connections, the 2nd Ethernet interface can use as maximum 13 Modbus server connections.

The usage of the sockets can be shown in the PLC browser or by using the command: `coupler settings`

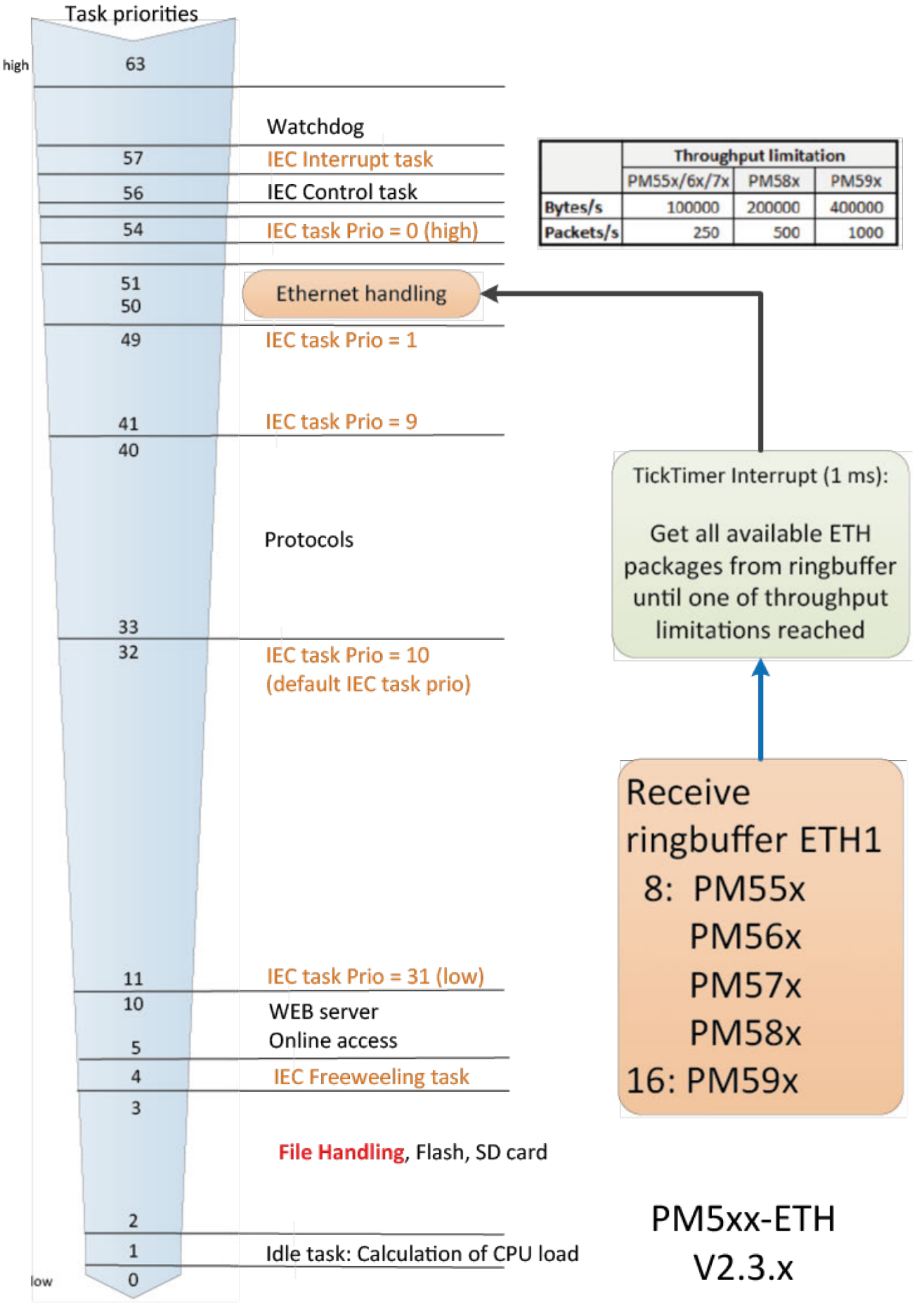
Handling of Ethernet protocols in AC500 CPUs firmware as of V2.4.0



The usage of the sockets can be shown in the PLC browser or by using the command: `coupler settings`

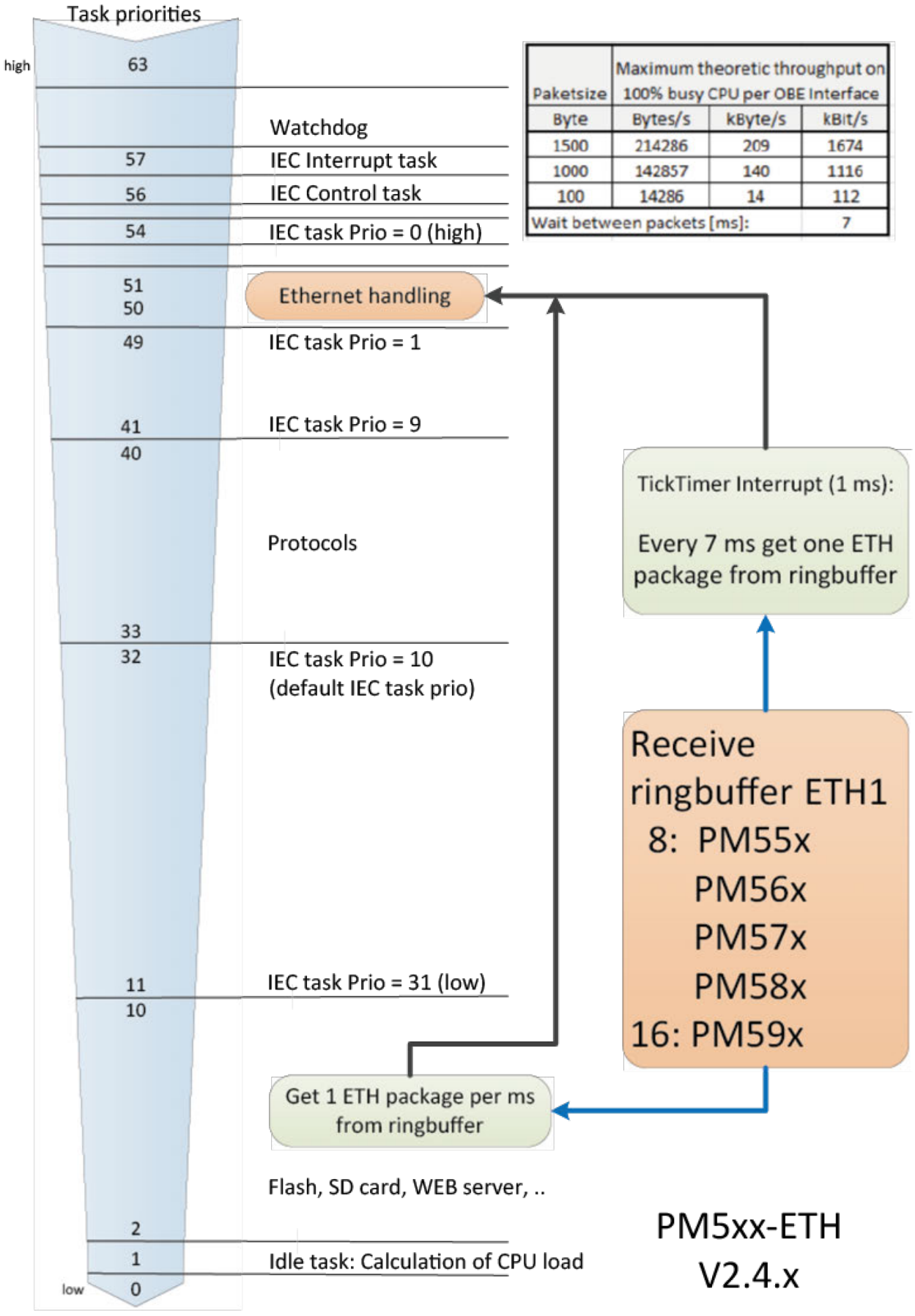
Onboard Ethernet handling in CPU firmware

Until firmware V2.3.x


Task priorities
high
63
Watchdog
57
IEC Interrupt task
56
IEC Control task
54
IEC task Prio = 0 (high)
51
50
Ethernet handling
49
IEC task Prio = 1
41
IEC task Prio = 9
40
Protocols
33
32
IEC task Prio = 10
(default IEC task prio)
11
IEC task Prio = 31 (low)
10
WEB server
5
Online access
4
IEC Freeweeling task
3
File Handling, Flash, SD card
2
1
Idle task: Calculation of CPU load
0
low
Throughput limitation
PM55x/6x/7x
PM58x
PM59x
Bytes/s
100000
200000
400000
Packets/s
250
500
1000
TickTimer Interrupt (1 ms):
Get all available ETH packages from ringbuffer until one of throughput limitations reached
Receive ringbuffer ETH1
8: PM55x
PM56x
PM57x
PM58x
16: PM59x
PM5xx-ETH
V2.3.x

The Ethernet communication has direct impact on the IEC tasks jitter.

As of firmware V2.4.0 for PM5xx-ETH

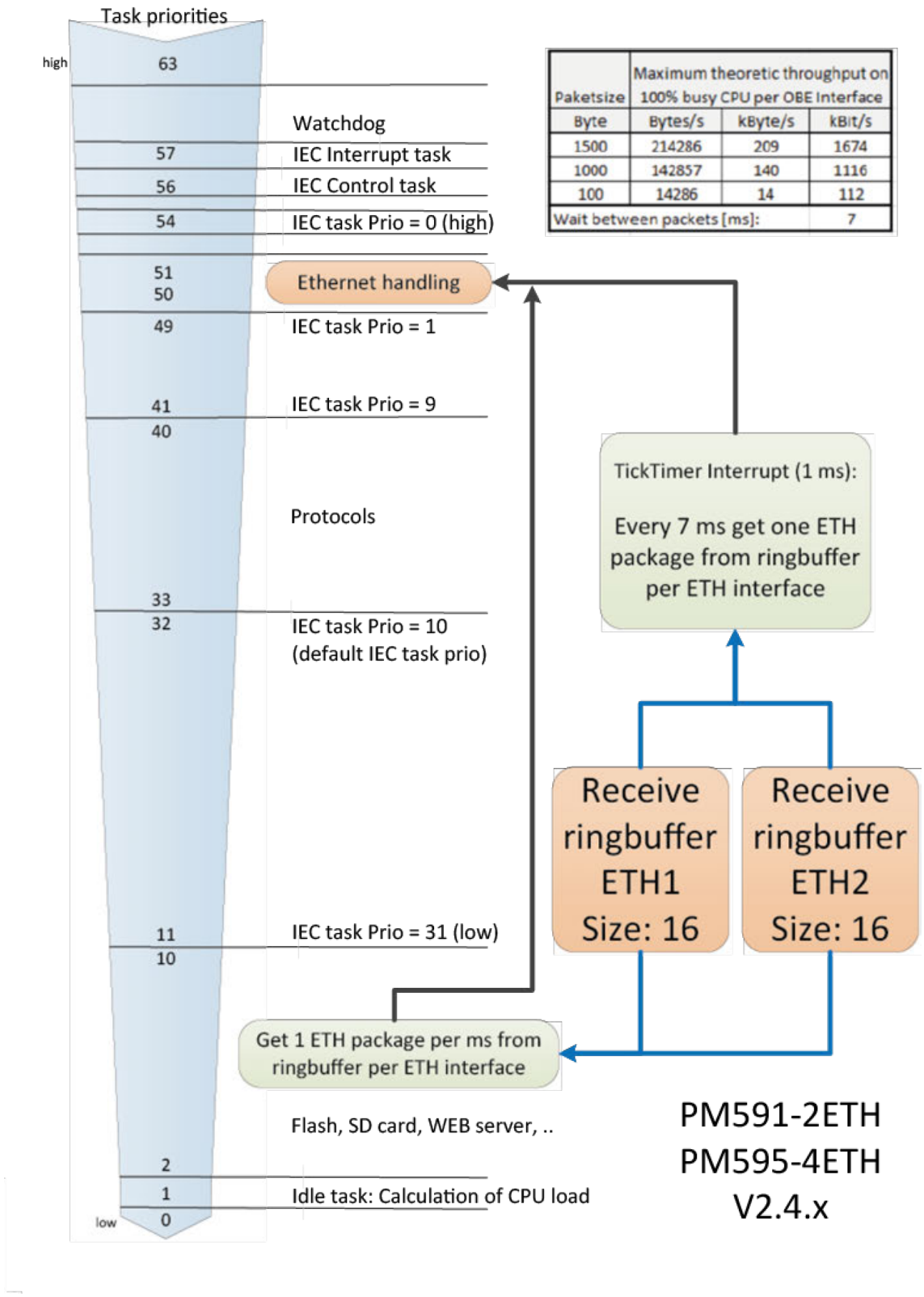


Paketsize	Maximum theoretic throughput on 100% busy CPU per OBE Interface		
Byte	Bytes/s	kByte/s	kBit/s
1500	214286	209	1674
1000	142857	140	1116
100	14286	14	112
Wait between packets [ms]:			7

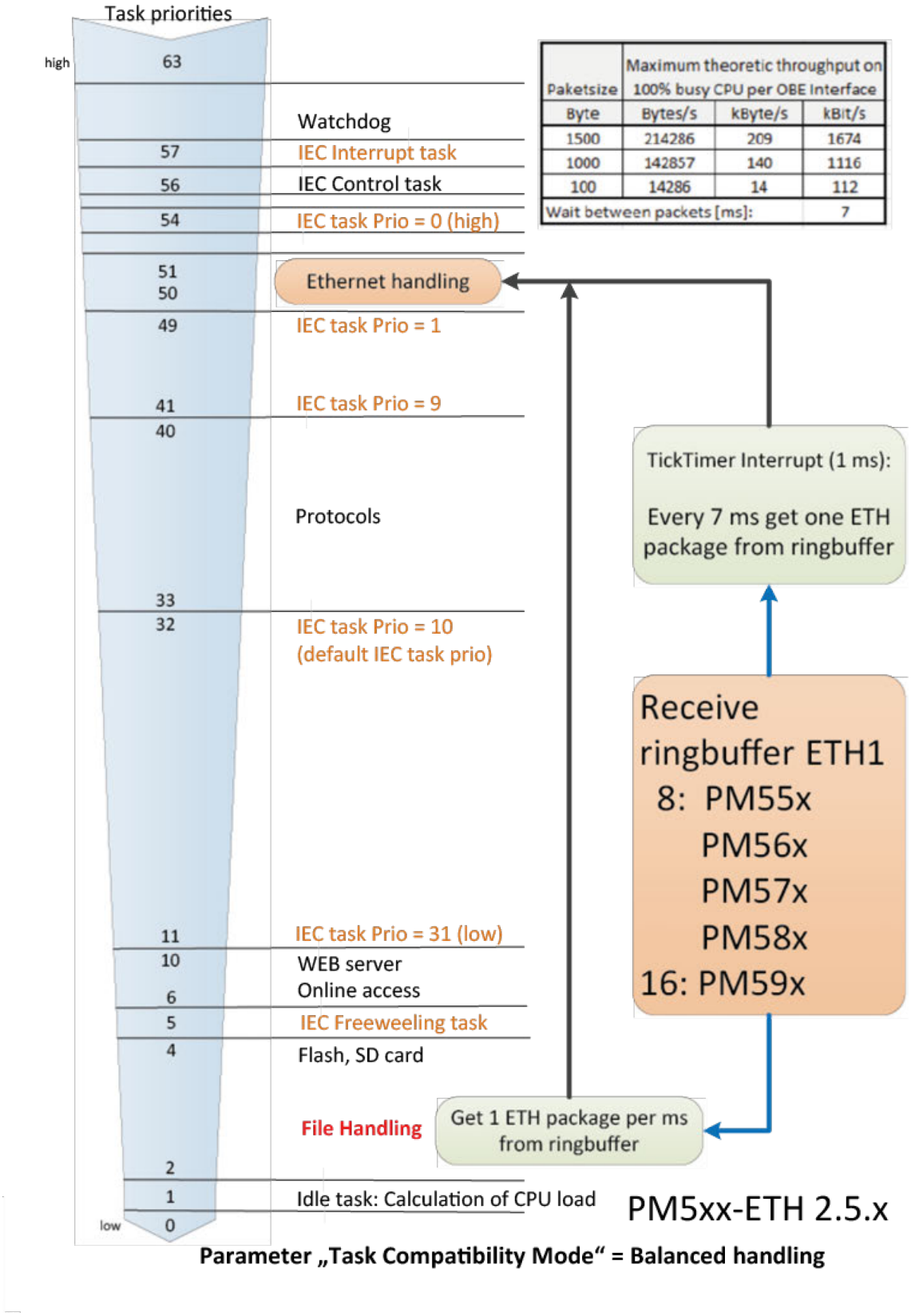
Minimal impact on task jitter of IEC tasks.

Ethernet throughput depends on CPU load of PM5xx-ETH.

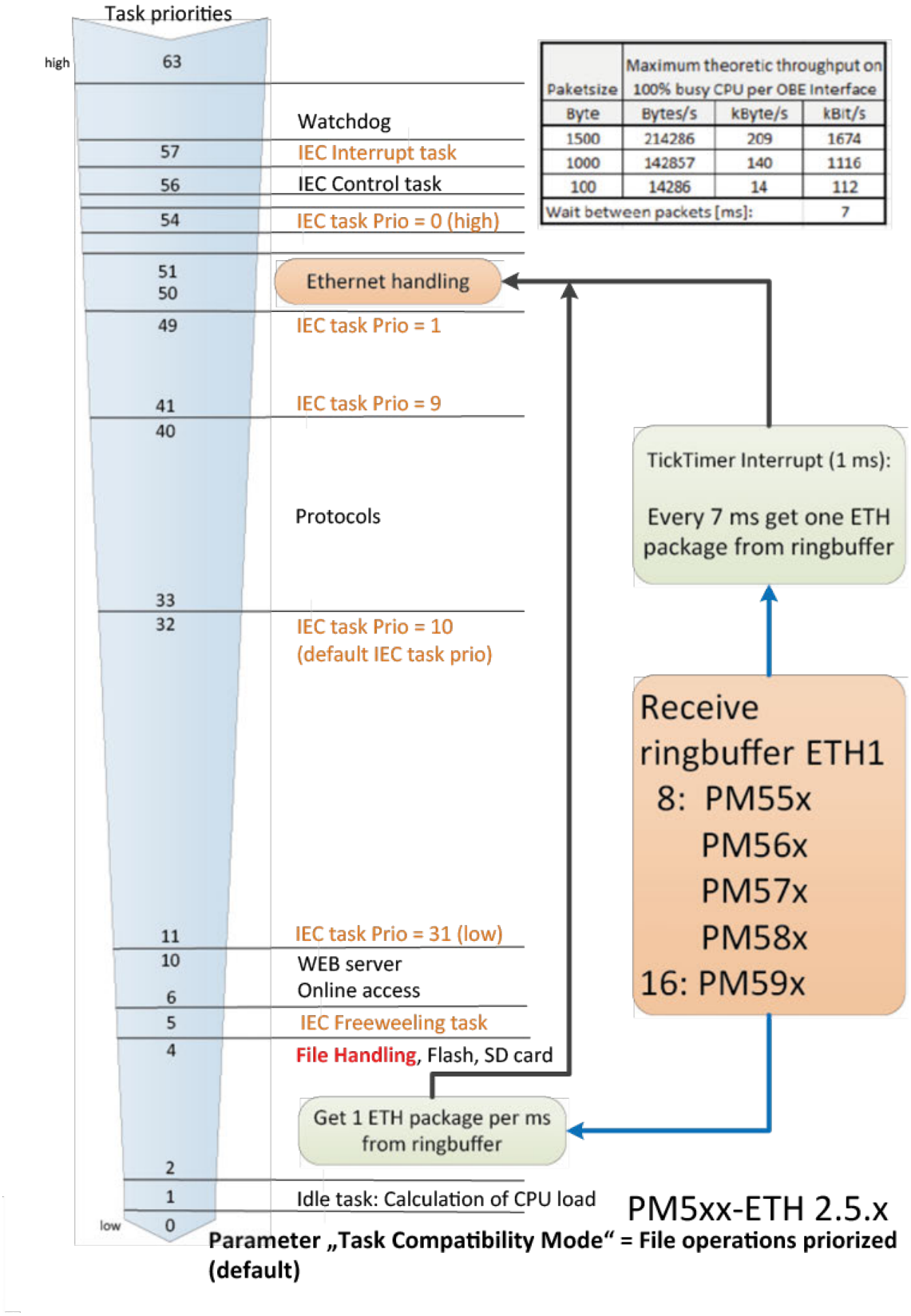
As of firmware V2.4.0 for PM5xx-2ETH and PM5xx-4ETH



As of firmware V2.5.0 for PM5xx-ETH



As of firmware V2.5.0 for PM5xx-2ETH and PM5xx-4ETH



Paketsize	Maximum theoretic throughput on 100% busy CPU per OBE Interface		
Byte	Bytes/s	kByte/s	kBit/s
1500	214286	209	1674
1000	142857	140	1116
100	14286	14	112
Wait between packets [ms]:			7

SNTP client and server

All AC500 processor modules with onboard Ethernet support can be configured as Simple Network Time Protocol (SNTP) servers and clients.

The SNTP protocol is specified in *RFC 4330*.

Since the packet format is the same between NTP and SNTP, an AC500 SNTP client can also use any (full) NTP server to receive time information.

SNTP client

The AC500 can receive time information from NTP server(s) in the network and adjust its local clock accordingly. The configuration of the SNTP protocol (see *Chapter 1.6.5.3.6.2.1 “SNTP client configuration” on page 6183*) allows to specify two IP addresses for Servers as well as the interval of requests to the server.

When the main server cannot be reached for 5 consecutive times, an Error class 4 is raised and the backup server will be used. If the backup server also cannot be reached for 5 consecutive times, an Error class 3 is raised.

According to the RFC specification, the SNTP client will wait a random amount of time between 1 and 5 minutes before performing the first time request after starting up the user program (PLC in RUN).

To avoid this behavior, the configuration option *Wait for sync before RUN* can be set, so that the user program is not allowed to run before the time was synchronized at least once.

The PLC will try to sync for a configurable amount of time (*Time to wait for sync*) before either allowing the user program to switch to RUN or raising an error class 4 and keeping the PLC is STOP.

The clock in the AC500 is prone to drift just like any clock based on an oscillating quartz.

Any adjustments of the clock will be attempted by speeding up or slowing down the internal clock.

But there may be situations where the difference becomes too large to adjust the clock this way and the clock needs to be set to the new time without gradual adjustments. This is called a time jump and may not be wanted in some applications. To mitigate this, another set of configuration options exist: *Allow time jumps* and *allowed threshold for time jumps*”.

These parameters can be used to either completely disallow jumps or keep them in an acceptable range. By default time jumps are allowed with a maximum of 60 seconds difference. Any jumps higher than the configured threshold will not be allowed and raise an error class 4.

The first synchronization is always allowed to jump any value because the default date in the PLC is 01.01.1970 00:00 and thus needs to jump several years to reach current date.

SNTP server

The AC500 can function as an SNTP Server to other clients (such as other AC500 PLCs).

For PLCs with multiple Interfaces, the server can be configured to run on either one or both interfaces (ETH1, ETH2).

The configuration of the SNTP protocol (see *Chapter 1.6.5.3.6.2.2 “SNTP server configuration” on page 6185*) allows to restrict the IP address range which will be served by this SNTP server. By default, any request will be served with a reply. To restrict the requests to a specific subnet, the CIDR notation is used.

Example

- Restrict access to the SNTP server to the subnet from 192.168.0.1 to 192.168.0.254 by CIDR notation 192.168.0.0/24.
- Restrict access to the SNTP server to the subnet from 10.0.0.0 to 10.255.255.255 by notation 10.0.0.0/8

The SNTP server will use the local clock to serve its time.

Sync the local clock to a more precise time source:

- *Being SNTP client and receiving the time from a more reliable NTP server.*
- *By syncing the local clock to another time source like a GPS or DCF77 clock.*

ARCNET

- ARCNET (Attached Resource Computer Network) is a system for data transmission in local networks.
- The ARCNET protocol is based on the Token Passing principle.
- By passing an identifier (token) from station to station it is guaranteed, that only one station can start a data transmission (transmission without collision).
- The order of sequence, in which the stations are accessed, is automatically adapted by the existing conditions in the network, i.e. that the network is reconfigured automatically each time a station is added to the network or switched off.

ARCNET bus topology

The networking possibilities of linear ARCNET

- The Linear ARCNET connects the individual stations directly to each other, i.e. without using any distribution units.
- Each station is connected to the network by using a T connector.
- Both cable ends must be terminated by terminating resistors.
- A maximum of 8 stations can be connected to one Linear ARCNET.
- The maximum cable length of the network is 300 m.
- An additional segment can be connected at the end of the wired segment via an Active Hub (active distribution unit), see next but one drawing.

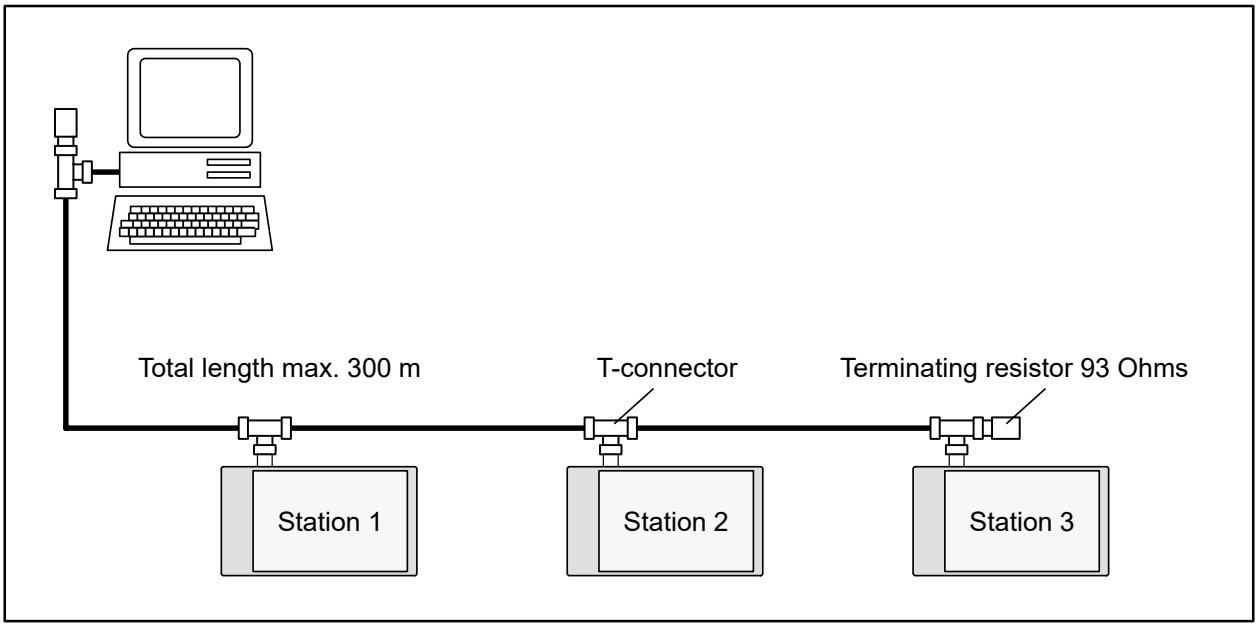


Fig. 1110: Linear ARCNET

Linear ARCNET, expanded by active distribution units (Active hubs)

Active Hubs amplify the arriving signals. So they stabilize the network configuration and allow especially for high distances. The Active Hub decouples the station connectors from each other. Therefore, the entire network does not fail when one of the connections fails.

The maximum length of the network is 6 km.

A maximum of 255 stations can be used.

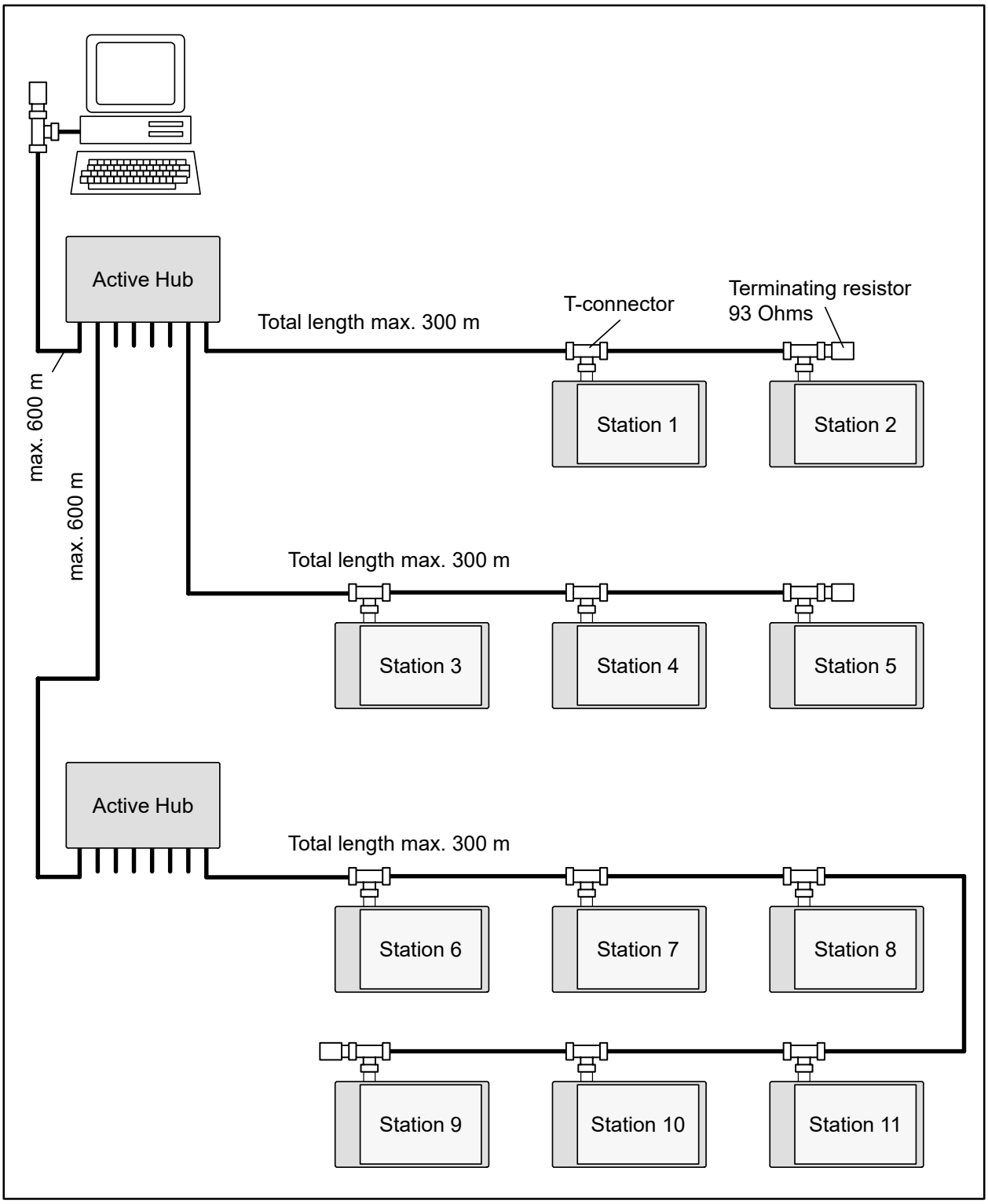


Fig. 1111: Linear ARCNET, expanded by active distribution units (Active Hubs)

1.6.4.1.7 Hot swap

Preconditions for using hot swap

Hot swap

WARNING!

Risk of explosion or fire in hazardous environments during hot swapping!

Hot swap must not be performed in flammable environments to avoid life-threatening injury and property damage resulting from fire or explosion.

WARNING!

Electric shock due to negligent behavior during hot swapping!

To avoid electric shock

- make sure the following conditions apply:
  - Digital outputs are not under load.
  - Input/output voltages above safety extra low voltage/ protective extra low voltage (SELV/PELV) are switched off.
  - Modules are fully interlocked with the terminal unit with both snap-fits engaged before switching on loads or input/output voltage.
- Never touch exposed contacts (dangerous voltages).
- Stay away from electrical contacts to avoid arc discharge.
- Do not operate a mechanical installation improperly.

NOTICE!

Risk of damage to I/O modules!

Hot swapping is only allowed for I/O modules.

Processor modules and communication interface modules must not be removed or inserted during operation.

H = **H**ot swap

Hot swap

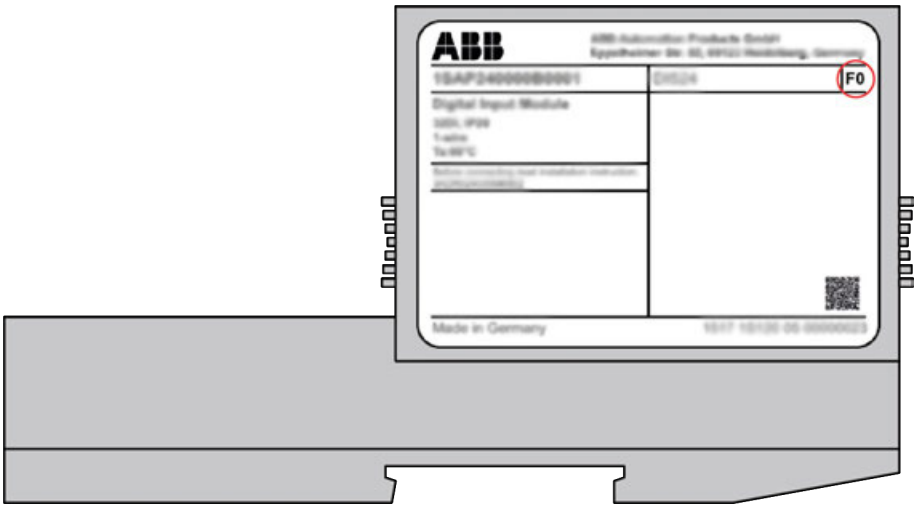
System requirements for hot swapping of I/O modules:

- *Types of terminal units that support hot swapping of I/O modules have the appendix TU5xx-**H**.*
- *I/O modules as of index F0.*

The following I/O bus masters support hot swapping of attached I/O modules:

- *Communication interface modules CI5xx as of index F0.*
- *Processor module PM585-ETH with firmware version as of V2.8.1.*

Hot swap is not supported by AC500-eCo V3 CPU!



The index of the module is in the right corner of the label.

NOTICE!

Risk of damage to I/O modules!

Modules with index below F0 can be damaged when inserted or removed from the terminal unit in a powered system.

NOTICE!

Risk of damage to I/O modules!

Do not perform hot swapping if any I/O module with firmware version lower than 3.0.14 is part of the I/O configuration.

For min. required device index see table below.

Device	Min. required device index for I/O module as of FW Version 3.0.14
AC522(-XC)	F0
AI523 (-XC)	D2
AI531	D4
AI531-XC	D2
AI561	B2
AI562	B2
AI563	B3
AO523 (-XC)	D2
AO561	B2
AX521 (-XC)	D2
AX522 (-XC)	D2
AX561	B2
CD522 (-XC)	D1

Device	Min. required device index for I/O module as of FW Version 3.0.14
DA501 (-XC)	D2
DA502 (-XC)	F0
DC522 (-XC)	D2
DC523 (-XC)	D2
DC532 (-XC)	D2
DC561	B2
DC562	A2
DI524 (-XC)	D2
DI561	B2
DI562	B2
DI571	B2
DI572	A1
DO524 (-XC)	A3
DO526	A2
DO526-XC	A0
DO561	B2
DO562	A2
DO571	B3
DO572	B2
DO573	A1
DX522 (-XC)	D2
DX531	D2
DX561	B2
DX571	B3
FM562	A1

Compatibility of hot swap

	Central I/O on V2 CPU
I/O module on TU5xx-H connected to I/O bus master	AC5000 V2 CPU types: PM585-ETH, PM59x-ETH, PM591-2ETH or PM595-4ETH
Required version of I/O bus master	Firmware as of V2.8.1
Fieldbus master when used as remote I/O with AC500 V2	-
When used as remote I/O on third party controller (PLC or DCS)	-

Hot swap behavior

The following table describes the behavior in case of I/O attached to the AC500 CPU with firmware supporting hot swap on the I/O bus.

Hot Swap Behavior	Central I/O on V2 CPU
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H	System and I/O modules attached to the CPU are starting (except missing or damaged module when mounted on hot swap terminal unit). As soon as the correct and operational I/O module is plugged on the terminal unit, the module is configured and ready to start. Precondition: ● "Run On Config Fault" must be configured ● "Max Wait Run" = default (3000)
Start-up behavior with wrong I/O module type on any terminal unit	System and I/O modules are not starting
Diagnosis of presence of hot swap terminal unit	Diagnosis using PLC browser command "io-bus desc" in Automation Builder V2. The PLC browser then provides an overview of the modules on the I/O bus including the position of hot swap terminal units in the I/O bus.
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	I/O bus master generates ● Diagnosis in case that a not hot-swap-pable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configu-ration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	I/O bus master generates diagnosis in AC500 format. The diagnosis is available in the diagnosis system.
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO
Diagnosis after plugging the I/O module on the hot swap terminal unit	Diagnosis message "diagnosis gone" is generated

1.6.4.1.8 Communication with Modbus RTU

Protocol description

The Modbus RTU protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

The Modbus operating mode of a serial interface is set in the PLC configuration. See *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*

Modbus client

In this operating mode, the telegram traffic with the server(s) is handled via the function block *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698*.

This function block sends Modbus request telegrams to the server(s) via the set interface and receives Modbus response telegrams from the server(s) via this interface.

The Modbus blocks transferred by the server contain the following information:

- Modbus address of the interrogated server (1 byte)
- Function code that defines the request of the client (1 byte)
- Data to be exchanged (n bytes)
- CRC16 control code (2 bytes)

Modbus server

In this operating mode, no function block is required for Modbus communication. Sending and receiving Modbus telegrams is performed automatically.

The AC500 CPUs process the following Modbus operation codes:

Function code		Description
DEC	HEX	
01 or 02	01 or 02	Read n bits
03 or 04	03 or 04	Read n words
05	05	Write one bit (encoded in one word)
06	06	Write one word
07	07	Fast reading the status byte of the CPU
15	0F	Write n bits (encoded in one byte)
16	10	Write n words
22	16	Mask write
23	17	Read/write multiple words in one telegram

The following restrictions apply to the length of the data to be sent:

Function code		Max. length
DEC	HEX	
01 or 02	01 or 02	2000 bits
03 or 04	03 or 04	125 words / 62 double words
05	05	1 bit

Function code		Max. length
DEC	**HEX**	
06	06	1 word
07	07	8 bits
15	0F	1968 bits
16	10	123 words / 61 double words
22	16	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 120 words / 60 double words

Technical data

The Modbus operating mode and the interface parameters are set in the *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*.

Table 630: Description of the Modbus protocol

Parameter			Value
Supported standard			PM55x and PM56x: EIA RS-485 PM57x, PM58x and PM59x: EIA RS-232 / RS-485
Number of connection points			1 client Max. 1 server with RS-232 interface Max. 31 servers with RS-485
Protocol			Modbus
Operating mode			Client/server
Address			Server only
Data transmission control			CRC16
Data transmission speed			From 300 bits/s to 187,500 bits/s
Encoding			1 start bit 8 data bits 1 parity bit, (optional) even, odd, mark or space 1 or 2 stop bits
Max. cable length for RS-485 on COM1 / COM2 for AC500 CPU			1.200 m at 19.200 baud
Max. cable length for RS-485 on COM1 / COM2 for AC500-eCo CPU			
	COM1:		
		Non-isolated:	Max. 50 m (with shielded cable)
		Isolated with TK506:	Max. 500 m at 19.200 baud (with shielded cable *)
	COM2:		

Parameter			Value
		Non-isolated with TA562:	Max. 50 m (with shielded cable)
		Isolated with TA569:	Max. 500 m at 19.200 baud (with shielded cable *)

*) 500 m cable type STP-120 Ω/AWG-20

If a processor module provides more than one serial interface, both interfaces (COM1/COM2) can be operated simultaneously as Modbus interfaces and can operate as Modbus server as well as Modbus client.

Modbus addresses for AC500 CPUs

Modbus address table

A range of 128 kB is allowed for the access via Modbus, i.e., the segments line 0 and line 1 of the addressable flag area (%M area) can be accessed. Thus, the complete address range 0000hex up to FFFFhex is available for Modbus.

The availability of the segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs and in the target system settings.

Inputs and outputs cannot be directly accessed using Modbus.

Address assignment (bit accesses)

The address assignment for bit accesses is done according to the following table:

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
Line 0					
0000	0	%MB0.0	%MX0.0.0	%MW0.0	%MD0.0
0001	1		%MX0.0.1		
0002	2		%MX0.0.2		
0003	3		%MX0.0.3		
0004	4		%MX0.0.4		
0005	5		%MX0.0.5		
0006	6		%MX0.0.6		
0007	7		%MX0.0.7		
0008	8	%MB0.1	%MX0.1.0		
0009	9		%MX0.1.1		
000A	10		%MX0.1.2		
000B	11		%MX0.1.3		
000C	12		%MX0.1.4		
000D	13		%MX0.1.5		
000E	14		%MX0.1.6		
000F	15		%MX0.1.7		
0010	16	%MB0.2	%MX0.2.0	%MW0.1	
0011	17		%MX0.2.1		
0012	18		%MX0.2.2		

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0013	19		%MX0.2.3		
0014	20		%MX0.2.4		
0015	21		%MX0.2.5		
0016	22		%MX0.2.6		
0017	23		%MX0.2.7		
0018	24	%MB0.3	%MX0.3.0		
0019	25		%MX0.3.1		
001A	26		%MX0.3.2		
001B	27		%MX0.3.3		
001C	28		%MX0.3.4		
001D	29		%MX0.3.5		
001E	30		%MX0.3.6		
001F	31		%MX0.3.7		
0020	32	%MB0.4	%MX0.4.0	%MW0.2	%MD0.1
0021	33		%MX0.4.1		
0022	34		%MX0.4.2		
...	...	...	...	...	...
0FFF	4095	%MB0.511	%MX0.511.7	%MW0.255	%MD0.127
1000	4096	%MB0.512	%MX0.512.0	%MW0.256	%MD0.128
...	...	...	...	...	...
7FFF	32767	%MB0.4095	%MX0.4095.7	%MW0.2047	%MD0.1023
8000	32768	%MB0.4096	%MX0.4096.0	%MW0.2048	%MD0.1024
...	...	...	...	...	...
FFFF	65535	%MB0.8191	%MX0.8191.7	%MW0.4095	%MD0.2047

Calculation of the bit variable from the hexa-decimal address:

Formula:			
	Bit variable (BOOL) := %MX0.BYTE.BIT		
where:	DEC	Decimal address	
	BYTE	DEC / 8	
	BIT	DEC mod 8	(Modulo division)

Examples:

- Address hexadecimal = 16#2002
  DEC := 8194
  BYTE := 8194 / 8 := 1024
  BIT := 8194 mod 8 := 2
  Bit variable: %MX0.1024.2
- Address hexadecimal = 16#3016
  DEC := 12310
  BYTE := 12310 / 8 := 1538,75 -> 1538
  BIT := 12310 mod 8 := 6
  Bit variable: %MX0.1538.6
- Address hexadecimal = 16#55AA
  DEC := 21930
  BYTE := 21930 / 8 := 2741,25 -> 2741
  BIT := 21930 mod 8 := 2
  Bit variable: %MX0.2741.2

Calculation of the hexadecimal address from the bit variable:

Examples:

- Bit variable := %MX0.515.4
  DEC := 515 * 8 + 4 := 4124
  Address hex := 16#101C
- Bit variable := %MX0.3.3
  DEC := 3 * 8 + 3 := 27
  Address hex := 16#001B
- Bit variable := %MX0.6666.2
  DEC := 6666 * 8 + 2 := 53330
  Address hex := 16#D052

Peculiarities for accessing Modbus addresses

Peculiarities for bit access:

- As you can see in the address table, a WORD in the %M area is assigned to each Modbus address 0000hex .. FFFFhex
- Bit addresses 0000hex .. FFFFhex are contained in the word range %MW0.0 .. %MW0.4095

Areas protected from read/write access by Modbus client

As described in the PLC configuration, one write-protected and one read-protected area can be defined for each segment line 0 and line 1. If you try to write to a write-protected area or to read from a read-protected area, an exception response is generated.

Segment exceedance for line 0 and line 1:

A write- or read-protected area that lies in both segments, line 0 and line 1, cannot be accessed with a write/read operation. In case of a segment exceedance, an exception response is generated.

Example

Read 10 words beginning at address := 7FFEhex

This includes the addresses: 7FFEhex...8007hex with the operands %MW0.32766...%MW1.7. Because line 0 is exceeded in this case, an exception response is generated.

Due to this, two requests have to be generated here:

1. Read 2 words beginning at address := 7FFEhex.

2. Read 8 words beginning at address := 8000hex.

Comparison between AC500 and AC31/S90 Modbus addresses

The following table shows the addresses for AC500 controllers and its predecessor AC31/S90:

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
0000 ... 0FFF	01, 02	%IX0.0 ... %IX255.15	01, 02, 05, 07, 0F	%MX0.0.0 ... %MX0.511.7
0000		%IX0.0		%MX0.0.0
0001		%IX0.1		%MX0.0.1
0002		%IX0.2		%MX0.0.2
...		...		...
0010		%IX1.0		%MX0.2.0
...		...		...
0FFF		%IX255.15		%MX0.511.7
1000 ... 1FFF	01, 02, 05, 0F	%QX0.0 ... %QX255.15	01, 02, 05, 07, 0F	%MX0.512.0 ... %MX0.1023.7
1000		%QX0.0		%MX0.512.0
1001		%QX0.1		%MX0.512.1
1002		%QX0.2		%MX0.512.2
...		...		...
1010		%QX1.0		%MX0.514.0
...		...		...
1FFF		%QX255.15		%MX0.1023.7
2000 ... 2FFF	01, 02, 05, 07, 0F	%MX0.0 ... %MX255.15	01, 02, 05, 07, 0F	%MX0.1024.0 ... %MX0.1535.7
2000		%MX0.0		%MX0.1024.0
2001		%MX0.1		%MX0.1024.1
2002		%MX0.2		%MX0.1024.2
...		...		...
2010		%MX1.0		%MX0.1026.0
...		...		...
2FFF		%MX255.15		%MX0.1535.7
3000 ... 3FFF	01, 02, 05, 07, 0F	%MX5000.0 ... %MX5255.15	01, 02, 05, 07, 0F	%MX0.1536.0 ... %MX0.2047.7

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
3000		%MX5000.0		%MX0.1536.0
3001		%MX5000.1		%MX0.1536.1
3002		%MX5000.2		%MX0.1536.2
...		...		...
3010		%MX5001.0		%MX0.1538.0
...		...		...
3FFF		%MX5255.15		%MX0.2047.7
4000 ... FFFF		No access	01, 02, 05, 07, 0F	%MX0.2048.0 ... %MX0.8191.7
Word accesses				
0000 ... 0CFF	03, 04	%IW1000.0 ... %IW1207.15	03, 04, 06, 10	%MW0.0 ... %MW0.3327
0D00 ... 0FFF	03, 04	No access	03, 04, 06, 10	%MW0.3328 ... %MW0.4095
1000 ... 1CFF	03, 04, 06, 10	%QW1000.0 ... %QW1207.15	03, 04, 06, 10	%MW0.4096 ... %MW0.7423
1D00 ... 1FFF		No access	03, 04, 06, 10	%MW0.7424 ... %MW0.8191
2000 ... 2FFF	03, 04, 06, 10	%MW1000.0 ... %MW1255.15	03, 04, 06, 10	%MW0.8192 ... %MW0.12287
3000 ... 359F	03, 04, 06, 10	%MW3000.0 ... %MW3089.15	03, 04, 06, 10	%MW0.12288 ... %MW0.13727
35A0 ... 3FFF		No access	03, 04, 06, 10	%MW0.13728 ... %MW0.16383
4000 ... 47FF		%MW2000.0.0 ... %MW2063.15.1 No access	03, 04, 06, 10	%MW0.16384 ... %MW18431
4800 ... 4FFF		No access	03, 04, 06, 10	%MW0.18432 ... %MW0.20479
5000 ... 517F		%MW4000.0.0 ... %MW4023.15.1 No access	03, 04, 06, 10	%MW0.20480 ... %MW0.21247
5180 ... FFFF		No access	03, 04, 06, 10	%MW0.21248 ... %MW1.32767
Double word accesses				
0000 ... 3FFF		No access	03, 04, 06, 10	%MD0.0 ... %MD0.8191
4000 ... 47FF	03, 04, 06, 10	%MD2000.0 ... %MD2063.15	03, 04, 06, 10	%MD0.8192 ... %MD0.9215
4800 ... 4FFF		No access	03, 04, 06, 10	%MD0.9216 ... %MD0.10239

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
5000 ... 537F	03, 04, 06, 10	%MD4000.0 ... %MD4023.15	03, 04, 06, 10	%MD0.1240 ... %MD0.10815
5480 ... FFFF		No access	03, 04, 06, 10	%MD0.10816 ... %MD1.16383

Local data of the Modbus client

The address of the area from which data are to be read or to which data are to be written is specified in the function block *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698* at input "Data", via the ADR operator.

For the AC500, the following areas can be accessed using the ADR operator:

- Inputs area (%I area)
- Outputs area (%Q area)
- Area of non-buffered variables (VAR .. END_VAR or VAR_GLOBAL END_VAR)
- Addressable flag area (also protected areas for %M area)
- Area of buffered variables (VAR RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR)

Modbus telegrams

The send and receive of telegrams shown in this section are not visible in the PLC. However, the complete telegrams can be made visible using a serial data analyzer connected to the connection line between server and client, if required.

The amount of user data depends on the capabilities of the server and the client.

For the following examples, it is assumed that one AC500 Modbus module is used as client and another one is used as server. There may be different properties if modules of other manufacturers are used.

FCT 1 or 2: Read n bits

Table 631: Client request

Server address	Function code	Server operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

Table 632: Server response

Server address	Function code	Number of Bytes	...Data...	CRC	
				High	Low

Example

Table 633: Example:

Modbus interface of the client	COM1
Client reads from	Server 1
Data	%MX0.1026.4 = FALSE %MX0.1026.5 = TRUE %MX0.1026.6 = FALSE
Source address at server	%MX0.1026.4 : 2014HEX = 8212DEC
Target address at client	abReadBool: ARRAY[0..2] OF BOOL
The values of the flags %MX0.1026.4..%MX0.1026.6 on the server are written to the ARRAY abReadBool on the client.	

Table 634: Modbus request of the client

Server address	**Function code**	**Server operand address**		**Number of bits**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	01HEX	20HEX	14HEX	00HEX	03HEX	37HEX	CFHEX

Table 635: Modbus response of the server

Server address	**Function code**	**Number of bytes**	**Data**	**CRC**	
				High	**Low**
01HEX	01HEX	01HEX	02HEX	D0HEX	49HEX

Table 636: Parameterization of the COM_MOD_MAST function block inputs NB = Number of bits

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**NB**	**DATA**
FALSE -> TRUE	1	1	1	Application- specific	8212	3	ADR (abReadBool[0])

FCT 3 or 4: Read n words

Table 637: Client request

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Table 638: Server response

Server address	Function code	Number of Bytes	Data		CRC	
			High	Low	High	Low

Example

Table 639: Example

Modbus interface of the client	COM1
Client reads from	Server 1
Data	%MW0.8196 = 4; %MW0.8197 = 5; %MW0.8198 = 6
Source address at server	%MW0.8196 : 2004HEX = 8196DEC
Target address at client	awReadWord : ARRAY[0..2] OF WORD;
The values of the flag words %MW0.8196..%MW0.8198 on the server are written to the ARRAY awReadWord on the client.	

Table 640: Modbus request of the client

Server address	**Function code**	**Server operand address**		**Number of words**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	03HEX	20HEX	04HEX	00HEX	03HEX	4FHEX	CAHEX

Table 641: Modbus response of the server

Server address	**Function code**	**Number of bytes**	**Data**	**Data**	**Data**	**CRC**	
			High / Low	**High / Low**	**High / Low**	**High**	**Low**
01HEX	03HEX	06HEX	00HEX / 04HEX	00HEX / 05HEX	00HEX / 06HEX	40HEX	B6HEX

Table 642: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of words**	**DATA**
FALSE -> TRUE	1	1	3	Application- specific	8196	3	ADR (awRead-Word[0])

FCT 3 or 4: Read n double words

The function code "read double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer) Same tables as *Chapter 1.6.4.1.8.5.2 “FCT 3 or 4: Read n words” on page 5475*.

Example

Table 643: Example:

Modbus interface of the client	COM1
Client reads from	Server 1

Data	%MD0.8193 = 32DEC = 00000020HEX %MD0.8194 = 80000DEC = 00013880HEX
Source address at server	%MD0.8193: 4002HEX = 16386DEC
Target address at client	adwReadDWord : ARRAY[0..1] OF DWORD
The values of the flag double words %MD0.8193..%MD0.8194 on the server are written to the ARRAY adwReadDWord on the client.	

Table 644: Modbus request of the client

Server address	**Function code**	**Server operand address**		**Number of words**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	03HEX	40HEX	02HEX	00HEX	04HEX	F0HEX	09HEX

Table 645: Modbus response of the server

Server address	**Func-tion code**	**Number of bytes**	**Data**	**Data**	**Data**	**Data**	**CRC**	
			High / Low	**High / Low**	**High / Low**	**High / Low**	**High**	**Low**
01HEX	03HEX	08HEX	00HEX / 00HEX	00HEX / 20HEX	00HEX / 01HEX	38HEX / 80HEX	57HEX	B0HEX

Table 646: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of words**	**DATA**
FALSE -> TRUE	1	1	3	Applica-tion- spe-cific	16386	4	ADR (adwRead DWord[0])

FCT 5: Write 1 bit

For the function code "write 1 bit", the value of the bit to be written is encoded in one word.

BIT = TRUE -> Data word = FF 00 HEX

BIT = FALSE -> Data word = 00 00 HEX

Table 647: Client request

Function code	Server operand address		Number of words		CRC	
	High	Low	High	Low	High	Low

Table 648: Server response

Function code	Server operand address		Data		CRC	
	High	Low	High	Low	High	Low

Example

Table 649: Example:

Modbus interface of the client	COM1
Client writes to	Server 1
Data	bBit := TRUE
Source address at client	bBit : BOOL
Target address at server	%MX0.1026.7 : 2017HEX = 8215DEC
The value of the BOOL variable bBit on the server is written to %MX0.1026.7 on the server.	

Table 650: Modbus request of the client

Server address	**Function code**	**Server operand address**		**Data**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	05HEX	20HEX	17HEX	FFHEX	00HEX	37HEX	FEHEX

Table 651: Modbus response of the server (mirrored)

Server address	**Function code**	**Server operand address**		**Data**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	05HEX	20HEX	17HEX	FFHEX	00HEX	37HEX	FEHEX

Table 652: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of bits**	**DATA**
FALSE -> TRUE	1	1	5	Applica-tion- spe-cific	8215	1	ADR (bBit)

FCT 6: Write 1 word

Table 653: Server request

Server address	Function code	Server operand address		Data		CRC	
		High	Low	High	Low	High	Low

Table 654: Server response

Server address	Function code	Server operand address		Data		CRC	
		High	Low	High	Low	High	Low

Example

Table 655: Example:

Modbus interface of the client	COM1
Client writes to	Server 1
Data	wData := 7
Source address at server	wData : WORD
Target address at client	%MW0.8199 : 2007HEX = 8199DEC
The value of the WORD variable bBit on the client is written to %MW0.8199 on the server.	

Table 656: Modbus request of the client

Server address	**Function code**	**Server operand address**		**Data**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	06HEX	20HEX	07HEX	00HEX	07HEX	72HEX	09HEX

Table 657: Modbus response of the server (mirrored)

Server address	**Function code**	**Server operand address**		**Data**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**
01HEX	06HEX	20HEX	07HEX	00HEX	07HEX	72HEX	09HEX

Table 658: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of words**	**DATA**
FALSE -> TRUE	1	1	6	Applica-tion- spe-cific	8215	1	ADR (wData)

FCT 7: Fast reading the status byte of the CPU

Server address	Function code	CRC	
		High	Low

Example

Table 659: Example:

Modbus interface of the client	COM1
Client writes to	Server 1

Table 660: Modbus request of the client

Server address	**Function code**	**CRC**					
		High	**Low**				
01HEX	07HEX	41HEX	E2HEX				

Table 661: Modbus response of the server

Server address	Function code	Data byte	CRC			
			High	Low		
01HEX	07HEX	00HEX	xxHEX	xxHEX		

Table 662: Parameterization of the COM_MOD_MAST function block inputs

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	Number of bits	DATA
FALSE -> TRUE	1	1	7	Applica-tion- spe-cific	0	0	ADR (BoolVar)

FCT 15: Write n bits

Table 663: Client request

Server operand address		Number of bits		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 664: Server response

Server address	Function code	Server operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low

Example

Table 665: Example:

Modbus interface of the client	COM1
Client writes to	Server 1
Data	abWriteBool[0] := TRUE abWriteBool[1] := FALSE abWriteBool[2] := TRUE
Source address at client	abWriteBool : ARRAY[0..2] OF BOOL
Target address at server	%MX0.1026.1 : 2011HEX = 8209DEC
The values of the BOOL variables abWriteBool[0]..abWriteBool[2] on the client are written to %MX0.1026.1..%MX0.1026.3 on the server.	

Table 666: Modbus request of the client

Server address	Func-tion code	Server operand address		Number of bits		Numbe r of bytes	Data	CRC	
		High	Low	High	Low			High	Low
01HEX	0FHEX	20HEX	11HEX	00HEX	03HEX	01HEX	05HEX	B4HEX	37HEX

Table 667: Modbus response of the server

Server address	Function code	Server operand address		Number of bits		CRC	
		High	Low	High	Low	High	Low
01HEX	0FHEX	20HEX	11HEX	00HEX	03HEX	4EHEX	0FHEX

Table 668: Parameterization of the COM_MOD_MAST function block inputs

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	Number of bits	DATA
FALSE -> TRUE	1	1	15	Application- specific	8209	3	ADR (abWrite-Bool[0])

FCT 16: Write n words

Table 669: Client request

Server operand address		Number of words		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 670: Server response

Function code	Server operand address		Number of words		CRC	
	High	Low	High	Low	High	Low

Example

Table 671: Example:

Modbus interface of the client	COM1
Client writes to	Server 1
Data	awWriteWord[0] := 1 awWriteWord[1] := 2 awWriteWord[2] := 3
Source address at server	awWriteWord : ARRAY[0..2] OF WORD
Target address at client	%MW0.8193 : 2001HEX = 8193DEC
The values of the WORD variables awWriteWord[0]..awWriteWord[2] on the client are written to %MW0.8193..%MW0.8195 on the server.	

Table 672: Modbus request of the client

Server address	Function code	Server operand address	Number of words	Number of bytes	Data	Data	Data	CRC
		High / Low	High / Low		High / Low	High / Low	High / Low	High / Low
01HEX	10HEX	20HEX / 01HEX	00HEX / 03HEX	06HEX	00HEX / 01HEX	00HEX / 02HEX	00HEX / 03HEX	C0HEX / 84HEX

Table 673: Modbus response of the server

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	10HEX	20HEX	01HEX	00HEX	03HEX	DAHEX	08HEX

Table 674: Parameterization of the COM_MOD_MAST function block inputs

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	Number of words	DATA
FALSE -> TRUE	1	1	16	Application- specific	8193	3	ADR (awWrite-Word[0])

FCT 16: Write n double words

The function code "write double word" is not defined in the Modbus RTU standard. This is why the double word is composed of a low word and a high word (depending on the manufacturer).

Table 675: Client request

Server operand address		Number of words		Number of bytes	...Data...	CRC	
High	Low	High	Low			High	Low

Table 676: Server response

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low

Example

Table 677: Example:

Modbus interface of the client	COM1
Client writes to	Server 1
Data	adwWriteDWord[0] := 18DEC = 00000012HEX; adwWriteDWord[1] := 65561DEC = 00010019HEX;
Source address at client	adwWriteDWord : ARRAY[0..1] OF DWORD;
Target address at server	%MD0.8192 : 4000HEX = 16384DEC
The values of the Double WORD variables adwWriteDWord[0].. adwWriteDWord[1] on the client are written to %MD0.8192..%MD0.8193 on the server.	

Table 678: Modbus request of the client

Server address	Function code	Server operand address	Number of words	Number of bytes	Data	Data	Data	Data	CRC
		High / Low	High / Low		High / Low	High / Low	High / Low	High / Low	High / Low
01HEX	10HEX	40HEX / 00HEX	00HEX / 04HEX	08HEX	00HEX / 00HEX	00HEX / 12HEX	00HEX / 01HEX	00HEX / 19HEX	60HEX / B3HEX

Table 679: Modbus response of the server

Server address	Function code	Server operand address		Number of words		CRC	
		High	Low	High	Low	High	Low
01HEX	10HEX	40HEX	00HEX	00HEX	04HEX	DAHEX	0AHEX

Table 680: Parameterization of the COM_MOD_MAST function block inputs

EN	COM	SLAVE	FCT	TIMEOUT	ADDR	Number of words = 2 x Number of double words	DATA
FALSE -> TRUE	1	1	16	Application- specific	16384	4	ADR (adwWriteD-Word[0])

FCT 22: Mask write register

Table 681: Client request

Server address	Function code	Server operand address		AND Mask		OR Mask		CRC	
		High	Low	High	Low	High	Low	High	Low

Table 682: Server response

Server address	Function code	Server operand address		AND Mask		OR Mask		CRC	
		High	Low	High	Low	High	Low	High	Low

Example

Table 683: Example

Modbus interface of the client	COM1
Client writes to	Server 1

Data	sMask.wAND_Mask := 16#00F2 sMask.wOR_Mask := 16#0025
Source address at the client	sMask : COM_MOD_FCT22_TYPE
Target address at server	%MW0.8193 : 2001HEX = 8193DEC
The values of the WORD variables sMask.wAND_Mask and sMask.wAND_Mask on the client are applied as masks on %MW0.8193..%MW0.8195 on the server.	

Table 684: Modbus request of the client

Server address	**Function code**	**Server operand address**	**AND Mask**	**OR Mask**	**CRC**
		High / Low	**High / Low**	**High / Low**	**High / Low**
01HEX	16HEX	20HEX / 01HEX	00HEX / F2HEX	00HEX / 25HEX	

Table 685: Modbus response of the server

Server address	**Function code**	**Server operand address**		**AND Mask**		**OR Mask**		**CRC**	
		High	**Low**	**High**	**Low**	**High**	**Low**	**High**	**Low**
01HEX	16HEX	20HEX	01HEX	00HEX	F2HEX	00HEX	25HEX	DAHEX	08HEX

Table 686: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of words**	**DATA**
FALSE -> TRUE	1	1	22	Applica-tionspe-cific	8193	1	ADR (sMask)

FCT 23: Read/Write n words

Table 687: Client request

Server address	Func-tion code	Operand addr. read		Number of words read		Operand addr. write		Number of words write		Number of bytes write	...Data...	CRC	
		High	Low	High	Low	High	Low	High	Low			High	Low

Table 688: Server response

Server address	Function code	Number of bytes read	...Data...	CRC	
				High	Low

Example

Table 689: Example

Modbus interface of the client	COM1
Client writes to	Server 1
Data	sData : COM_MOD_FCT23_TYPE sData.pByDataWrite := ADR(awWriteWord) sData.pByDataRead := ADR(awReadWord) sData.wDataAddressRead := 4193 sData.wNumDataUnitsRead := 5 awWriteWord[0] := 1 awWriteWord[1] := 2 awWriteWord[2] := 3
Source address at client	awWriteWord : ARRAY[0..2] OF WORD
Target address at client	awReadWord : ARRAY[0..4] OF WORD
Target address at server	%MW0.8193 : 2001HEX = 8193DEC
Source address at server	%MW0.4193 : 1001HEX = 4193DEC
The values of the WORD variables awWriteWord[0]..awWriteWord[2] on the client are written to %MW0.8193..%MW0.8195 on the server, the values read from server's . %MW0.4193..%MW0.4195 are written to the client's variables awReadWord[0]..awRead-Word[4]	

Table 690: Parameterization of the COM_MOD_MAST function block inputs

EN	**COM**	**SLAVE**	**FCT**	**TIMEOUT**	**ADDR**	**Number of words**	**DATA**
FALSE -> TRUE	1	1	17	Applica-tionspe-cific	8193	3	ADR (sData)

Exception response by server

In operating mode Modbus client, the AC500 does only send requests, if the parameters at the MOD_MAST inputs are logically correct.

Nevertheless, it can happen that a server cannot process the request of the client or that the server cannot interpret the request due to transmission errors or in case it's capabilities are exceeded in any way. In those cases, the server returns an exception response to the client. In order to identify this response as an exception response, the function code returned by the server is a logical OR interconnection of the function code received from the client and the value 80HEX.

Table 691: Server response

Server address	OR 80HEX	Error code	CRC	
			High	Low

Possible error codes of the client

Code	Description
01DEC	ILLEGAL FUNCTION The server does not support the function requested by the client
02DEC	ILLEGAL DATA ADDRESS Invalid operand address in the server or operand area exceeded
03DEC	ILLEGAL DATA VALUE At least one value is outside the permitted range of values
04DEC	SERVER DEVICE FAILURE An unrecoverable error occurred while the server was attempting to perform the requested action
05DEC	ACKNOWLEDGE Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed
06DEC	SERVER DEVICE BUSY Specialized use in conjunction with programming commands. The server is engaged in processing a long–duration program command. The client should retransmit the message later when the server is free.
07DEC	NEGATIVE ACKNOWLEDGE Specialized use in conjunction with programming commands. The server cannot perform the programming functions. Client should request diagnostic or error information from server.
08DEC	MEMORY PARITY ERROR Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
10DEC	GATEWAY PATH UNAVAILABLE Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
11DEC	GATEWAY TARGET DEVICE FAILED TO RESPOND Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Example

Table 692: Example:

Modbus request of the client:			
	Function code:	01	Read n bits
	Server operand address:	4000HEX = 16384DEC	Area for read access disabled in server

Modbus response of the server:			
	Function code:	81HEX	
	Error code:	03	

Function block COM_MOD_MAST

This function block is only required in the operating mode Modbus client. It handles the communication (transmission of telegrams to the servers and receipt of telegrams from the servers). The function block can be used for the local serial interfaces of the controller. A separate instance of the function block has to be used for each interface.

↳ *Chapter 1.5.4.22.1.1 “COM_MOD_MAST” on page 1698* is contained in the library Modbus_AC500_V1x.lib.

1.6.4.1.9 Communication with Modbus TCP/IP

Protocol description

The Modbus TCP protocol is implemented in the AC500 processor modules.

Modbus is a master-slave (client-server) protocol. The client sends a request to the server(s) and receives the response(s).

Each Ethernet interface can work as Modbus client and server interface in parallel if required.

The Modbus operating mode of an Ethernet interface is set in ↳ *Chapter 1.6.5.3.3.1 “Modbus on TCP/IP protocol” on page 6173*.

Modbus client

In the operating mode client, the telegram traffic with the server(s) is handled via the function block ↳ *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202* or ↳ *Chapter 1.5.4.13.1.19 “ETHx_MOD_MAST” on page 1250*. These function blocks send Modbus request telegrams to the server(s) via the set interface and receive Modbus response telegrams from the server(s) via this interface.

The Modbus function blocks transferred by the client contain the following information:

- Transaction identifier for synchronization between messages of server and client (2 byte)
- Protocol identifier (0 for Modbus/TCP) (2 byte)
- Length field (Number of bytes in frame) (2 byte)
- Unit identifier (1 byte)
- Function code that defines the request of the client (1 byte)
- Data to be exchanged (n bytes)

Modbus server

In this operating mode, no function block is required for Modbus communication. Sending and receiving Modbus telegrams is performed automatically.

The AC500 CPUs process the following Modbus operation codes:

Function code		**Description**
DEC	**HEX**	
01 or 02	01 or 02	Read n bits
03 or 04	03 or 04	Read n words
05	05	Write one bit (encoded in one word)
06	06	Write one word
07	07	Fast reading the status byte of the CPU
15	0F	Write n bits (encoded in one byte)
16	10	Write n words
22	16	Mask write
23	17	Read/write multiple words in one telegram

The following restrictions apply to the length of the data to be sent:

Function code		Max. length
DEC	HEX	
01 or 02	01 or 02	2000 bits
03 or 04	03 or 04	125 words / 62 double words
05	05	1 bit
06	06	1 word
07	07	8 bits
15	0F	1968 bits
16	10	123 words / 61 double words [1])
22	16	Write: 1 word
23	17	Read: 125 words / 62 double words Write: 120 words / 60 double words

Technical data

Configuration of Modbus on TCP/IP is described in the chapter *Chapter 1.6.5.3.3.1 “Modbus on TCP/IP protocol” on page 6173*.

Modbus addresses for AC500 CPUs

Modbus address table

A range of 128 kB is allowed for the access via Modbus, i.e., the segments line 0 and line 1 of the addressable flag area (%M area) can be accessed. Thus, the complete address range 0000hex up to FFFFhex is available for Modbus.

The availability of the segments depends on the CPU. The size of the %M area can be found in the technical data of the CPUs and in the target system settings.

Inputs and outputs cannot be directly accessed using Modbus.

The address assignment for word and double word accesses is done according to the following table:

Table 693: Modbus addresses (word accesses)

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
Line 0					
0000	0	%MB0.0	%MX0.0.0 ... %MX0.0.7	%MW0.0	%MD0.0
		%MB0.1	%MX0.1.0 ... %MX0.1.7		
0001	1	%MB0.2	%MX0.2.0 ... %MX0.2.7	%MW0.1	
		%MB0.3	%MX0.3.0 ... %MX0.3.7		

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0002	2	%MB0.4	%MX0.4.0 ... %MX0.4.7	%MW0.2	%MD0.1
		%MB0.5	%MX0.5.0 ... %MX0.5.7		
0003	3	%MB0.6	%MX0.6.0 ... %MX0.6.7	%MW0.3	
		%MB0.7	%MX0.7.0 ... %MX0.7.7		
...					
7FFE	32766	%MB0.65532	%MX0.65532.0 ... %MX0.65532.7	%MW0.32766	%MD0.16383
		%MB0.65533	%MX0.65533.0 ... %MX0.65533.7		
7FFF	32767	%MB0.65534	%MX0.65534.0 ... %MX0.65534.7	%MW0.32767	
		%MB0.65535	%MX0.65535.0 ... %MX0.65535.7		
Line 1					
8000	32768	%MB1.0	%MX1.0.0 ... %MX1.0.7	%MW1.0	%MD1.0
		%MB1.1	%MX1.1.0 ... %MX1.1.7		
8001	32769	%MB1.2	%MX1.2.0 ... %MX1.2.7	%MW1.1	
		%MB1.3	%MX1.3.0 ... %MX1.3.7		
8002	32770	%MB1.4	%MX1.4.0 ... %MX1.4.7	%MW1.2	%MD1.1
		%MB1.5	%MX1.5.0 ... %MX1.5.7		
8003	32771	%MB1.6	%MX1.6.0 ... %MX1.6.7	%MW1.3	
		%MB1.7	%MX1.7.0 ... %MX1.7.7		
...					
FFFE	65534	%MB1.65532	%MX1.65532.0 ... %MX1.65532.7	%MW1.32766	%MD1.16383

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
		%MB1.65533	%MX1.65533.0 ... %MX1.65533.7		
FFFF	65535	%MB1.65534	%MX1.65534.0 ... %MX1.65534.7	%MW1.32767	
		%MB1.65535	%MX1.65535.0 ... %MX1.65535.7		

Address assignment (bit accesses)

The address assignment for bit accesses is done according to the following table:

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
Line 0					
0000	0	%MB0.0	%MX0.0.0	%MW0.0	%MD0.0
0001	1		%MX0.0.1		
0002	2		%MX0.0.2		
0003	3		%MX0.0.3		
0004	4		%MX0.0.4		
0005	5		%MX0.0.5		
0006	6		%MX0.0.6		
0007	7		%MX0.0.7		
0008	8	%MB0.1	%MX0.1.0		
0009	9		%MX0.1.1		
000A	10		%MX0.1.2		
000B	11		%MX0.1.3		
000C	12		%MX0.1.4		
000D	13		%MX0.1.5		
000E	14		%MX0.1.6		
000F	15		%MX0.1.7		
0010	16	%MB0.2	%MX0.2.0	%MW0.1	
0011	17		%MX0.2.1		
0012	18		%MX0.2.2		
0013	19		%MX0.2.3		
0014	20		%MX0.2.4		
0015	21		%MX0.2.5		
0016	22		%MX0.2.6		

Modbus address		Byte	Bit (byte-oriented)	Word	Double word
HEX	DEC	BYTE	BOOL	WORD	DWORD
0017	23		%MX0.2.7		
0018	24	%MB0.3	%MX0.3.0		
0019	25		%MX0.3.1		
001A	26		%MX0.3.2		
001B	27		%MX0.3.3		
001C	28		%MX0.3.4		
001D	29		%MX0.3.5		
001E	30		%MX0.3.6		
001F	31		%MX0.3.7		
0020	32	%MB0.4	%MX0.4.0	%MW0.2	%MD0.1
0021	33		%MX0.4.1		
0022	34		%MX0.4.2		
...	...	...	...	...	...
0FFF	4095	%MB0.511	%MX0.511.7	%MW0.255	%MD0.127
1000	4096	%MB0.512	%MX0.512.0	%MW0.256	%MD0.128
...	...	...	...	...	...
7FFF	32767	%MB0.4095	%MX0.4095.7	%MW0.2047	%MD0.1023
8000	32768	%MB0.4096	%MX0.4096.0	%MW0.2048	%MD0.1024
...	...	...	...	...	...
FFFF	65535	%MB0.8191	%MX0.8191.7	%MW0.4095	%MD0.2047

Calculation of the bit variable from the hexadecimal address:

Formula:			
	Bit variable (BOOL) := %MX0.BYTE.BIT		
where:	DEC	Decimal address	
	BYTE	DEC / 8	
	BIT	DEC mod 8	(Modulo division)

Examples:

- Address hexadecimal = 16#2002
  DEC := 8194
  BYTE := 8194 / 8 := 1024
  BIT := 8194 mod 8 := 2
  Bit variable: %MX0.1024.2
- Address hexadecimal = 16#3016
  DEC := 12310
  BYTE := 12310 / 8 := 1538,75 -> 1538
  BIT := 12310 mod 8 := 6
  Bit variable: %MX0.1538.6
- Address hexadecimal = 16#55AA
  DEC := 21930
  BYTE := 21930 / 8 := 2741,25 -> 2741
  BIT := 21930 mod 8 := 2
  Bit variable: %MX0.2741.2

Calculation of the hexadecimal address from the bit variable:

Examples:

- Bit variable := %MX0.515.4
  DEC := 515 * 8 + 4 := 4124
  Address hex := 16#101C
- Bit variable := %MX0.3.3
  DEC := 3 * 8 + 3 := 27
  Address hex := 16#001B
- Bit variable := %MX0.6666.2
  DEC := 6666 * 8 + 2 := 53330
  Address hex := 16#D052

Peculiarities for accessing Modbus addresses

Peculiarities for bit access:

- As you can see in the address table, a WORD in the %M area is assigned to each Modbus address 0000hex .. FFFFhex
- Bit addresses 0000hex .. FFFFhex are contained in the word range %MW0.0 .. %MW0.4095

Areas protected from read/write access by Modbus client

As described in the PLC configuration, one write-protected and one read-protected area can be defined for each segment line 0 and line 1. If you try to write to a write-protected area or to read from a read-protected area, an exception response is generated.

Segment exceedance for line 0 and line 1:

A write- or read-protected area that lies in both segments, line 0 and line 1, cannot be accessed with a write/read operation. In case of a segment exceedance, an exception response is generated.

Example

Read 10 words beginning at address := 7FFEhex

This includes the addresses: 7FFEhex...8007hex with the operands %MW0.32766...%MW1.7. Because line 0 is exceeded in this case, an exception response is generated.

Due to this, two requests have to be generated here:

1. Read 2 words beginning at address := 7FFEhex.

2. Read 8 words beginning at address := 8000hex.

Comparison between AC500 and AC31/S90 Modbus addresses

The following table shows the addresses for AC500 controllers and its predecessor AC31/S90:

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
0000 ... 0FFF	01, 02	%IX0.0 ... %IX255.15	01, 02, 05, 07, 0F	%MX0.0.0 ... %MX0.511.7
0000		%IX0.0		%MX0.0.0
0001		%IX0.1		%MX0.0.1
0002		%IX0.2		%MX0.0.2
...		...		...
0010		%IX1.0		%MX0.2.0
...		...		...
0FFF		%IX255.15		%MX0.511.7
1000 ... 1FFF	01, 02, 05, 0F	%QX0.0 ... %QX255.15	01, 02, 05, 07, 0F	%MX0.512.0 ... %MX0.1023.7
1000		%QX0.0		%MX0.512.0
1001		%QX0.1		%MX0.512.1
1002		%QX0.2		%MX0.512.2
...		...		...
1010		%QX1.0		%MX0.514.0
...		...		...
1FFF		%QX255.15		%MX0.1023.7
2000 ... 2FFF	01, 02, 05, 07, 0F	%MX0.0 ... %MX255.15	01, 02, 05, 07, 0F	%MX0.1024.0 ... %MX0.1535.7
2000		%MX0.0		%MX0.1024.0
2001		%MX0.1		%MX0.1024.1
2002		%MX0.2		%MX0.1024.2
...		...		...
2010		%MX1.0		%MX0.1026.0
...		...		...
2FFF		%MX255.15		%MX0.1535.7
3000 ... 3FFF	01, 02, 05, 07, 0F	%MX5000.0 ... %MX5255.15	01, 02, 05, 07, 0F	%MX0.1536.0 ... %MX0.2047.7

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
3000		%MX5000.0		%MX0.1536.0
3001		%MX5000.1		%MX0.1536.1
3002		%MX5000.2		%MX0.1536.2
...		...		...
3010		%MX5001.0		%MX0.1538.0
...		...		...
3FFF		%MX5255.15		%MX0.2047.7
4000 ... FFFF		No access	01, 02, 05, 07, 0F	%MX0.2048.0 ... %MX0.8191.7
Word accesses				
0000 ... 0CFF	03, 04	%IW1000.0 ... %IW1207.15	03, 04, 06, 10	%MW0.0 ... %MW0.3327
0D00 ... 0FFF	03, 04	No access	03, 04, 06, 10	%MW0.3328 ... %MW0.4095
1000 ... 1CFF	03, 04, 06, 10	%QW1000.0 ... %QW1207.15	03, 04, 06, 10	%MW0.4096 ... %MW0.7423
1D00 ... 1FFF		No access	03, 04, 06, 10	%MW0.7424 ... %MW0.8191
2000 ... 2FFF	03, 04, 06, 10	%MW1000.0 ... %MW1255.15	03, 04, 06, 10	%MW0.8192 ... %MW0.12287
3000 ... 359F	03, 04, 06, 10	%MW3000.0 ... %MW3089.15	03, 04, 06, 10	%MW0.12288 ... %MW0.13727
35A0 ... 3FFF		No access	03, 04, 06, 10	%MW0.13728 ... %MW0.16383
4000 ... 47FF		%MW2000.0.0 ... %MW2063.15.1 No access	03, 04, 06, 10	%MW0.16384 ... %MW18431
4800 ... 4FFF		No access	03, 04, 06, 10	%MW0.18432 ... %MW0.20479
5000 ... 517F		%MW4000.0.0 ... %MW4023.15.1 No access	03, 04, 06, 10	%MW0.20480 ... %MW0.21247
5180 ... FFFF		No access	03, 04, 06, 10	%MW0.21248 ... %MW1.32767
Double word accesses				
0000 ... 3FFF		No access	03, 04, 06, 10	%MD0.0 ... %MD0.8191
4000 ... 47FF	03, 04, 06, 10	%MD2000.0 ... %MD2063.15	03, 04, 06, 10	%MD0.8192 ... %MD0.9215
4800 ... 4FFF		No access	03, 04, 06, 10	%MD0.9216 ... %MD0.10239

Address HEX	FCT HEX	AC1131 operand	FCT HEX	AC500 operand
Bit accesses				
5000 ... 537F	03, 04, 06, 10	%MD4000.0 ... %MD4023.15	03, 04, 06, 10	%MD0.1240 ... %MD0.10815
5480 ... FFFF		No access	03, 04, 06, 10	%MD0.10816 ... %MD1.16383

Local data of the Modbus client

The address of the area from which data are to be read or to which data are to be written is specified in the function block *Chapter 1.5.4.13.1.4 “ETH_MOD_MAST” on page 1202* or *Chapter 1.5.4.13.1.19 “ETHx_MOD_MAST” on page 1250* at input "Data", via the ADR operator.

For the AC500, the following areas can be accessed using the ADR operator:

- Inputs area (%I area)
- Outputs area (%Q area)
- Area of non-buffered variables (VAR .. END_VAR or VAR_GLOBAL END_VAR)
- Addressable flag area (also protected areas for %M area)
- Area of buffered variables (VAR RETAIN .. END_VAR or VAR_GLOBAL RETAIN .. END_VAR)

Modbus telegrams

For a detailed description of the Modbus TCP telegrams and their elements please see the corresponding specifications on public websites.

Exception response by server

In operating mode Modbus client, the AC500 does only send requests, if the parameters at the MODMAST inputs are logically correct. Nevertheless, it can happen that a server cannot process the request of the client or that the server cannot interpret the request due to transmission errors or in case it’s capabilities are exceeded in any way. In those cases, the server returns an exception response to the client. In order to identify this response as an exception response, the function code returned by the server is a logical OR interconnection of the function code received from the client and the value 80HEX.

General telegram description

Table 694: Server response

Error code	CRC	
	High	Low

Possible error codes of the client

Code	Description
01DEC	ILLEGAL FUNCTION The server does not support the function requested by the client
02DEC	ILLEGAL DATA ADDRESS Invalid operand address in the server or operand area exceeded
03DEC	ILLEGAL DATA VALUE At least one value is outside the permitted range of values
04DEC	SERVER DEVICE FAILURE An unrecoverable error occurred while the server was attempting to perform the requested action
05DEC	ACKNOWLEDGE Specialized use in conjunction with programming commands. The server has accepted the request and is processing it, but a long duration of time will be required to do so. This response is returned to prevent a timeout error from occurring in the client. The client can next issue a Poll Program Complete message to determine if processing is completed
06DEC	SERVER DEVICE BUSY Specialized use in conjunction with programming commands. The server is engaged in processing a long–duration program command. The client should retransmit the message later when the server is free.
07DEC	NEGATIVE ACKNOWLEDGE Specialized use in conjunction with programming commands. The server cannot perform the programming functions. Client should request diagnostic or error information from server.
08DEC	MEMORY PARITY ERROR Specialized use in conjunction with function codes 20 and 21 and reference type 6, to indicate that the extended file area failed to pass a consistency check. The server attempted to read record file, but detected a parity error in the memory. The client can retry the request, but service may be required on the server device.
09DEC	UNDEFINED Actually not defined by Modbus specification but might be used by particular servers.
10DEC	GATEWAY PATH UNAVAILABLE Specialized use in conjunction with gateways, indicates that the gateway was unable to allocate an internal communication path from the input port to the output port for processing the request. Usually means that the gateway is misconfigured or overloaded.
11DEC	GATEWAY TARGET DEVICE FAILED TO RESPOND Specialized use in conjunction with gateways, indicates that no response was obtained from the target device. Usually means that the device is not present on the network.

Example

Table 695: Example:

Modbus request of the client:			
	Function code:	01	Read n bits
	Server operand address:	4000HEX = 16384DEC	Area for read access disabled in server

Modbus response of the server:			
	Function code:	81HEX	
	Error code:	03	

Function blocks ETH_MOD_MAST and ETHx_MOD_MAST

These function blocks are only required for the operating mode Modbus client. They handle the communication (transmission of telegrams to the servers and receipt of telegrams from the servers). The function blocks can be used for any Ethernet interface of the controller itself or it's communication modules.

↳ *Chapter 1.5.4.13.1.4 "ETH_MOD_MAST" on page 1202* and ↳ *Chapter 1.5.4.13.1.19 "ETHx_MOD_MAST" on page 1250* are contained in the library Ethernet_AC500_V1x.lib.

1.6.4.1.10 Fast counters

Fast counters in AC500 devices

For AC500 devices the function "fast counter" is available in S500 I/O modules as of firmware version V1.3.

For AC500-eCo devices the function "fast counter" is available in onboard I/Os of PM55x and PM56x.

Integrated fast counters are only available for digital I/O modules.

The digital I/O modules on the I/O bus contain two fast counters each.

If the counter is used, it needs up to 2 digital inputs and one digital output.

If the fast counter is deactivated, the inputs and outputs reserved for the counter can be used for other tasks.

See ↳ *Chapter 1.6.5.2.9.8 "Fast counter" on page 6063*.

A fast counter is available in the following constellations:

- In digital I/O modules, connected to an AC500 processor module.
- In AC500-eCo V2 processor modules PM55x and PM56x with onboard I/Os.
- In CS31 and CANopen communication interface modules.
- In Modbus, PROFIBUS and PROFINET communication interface modules and in the connected digital I/O modules.
- In digital I/O modules, connected to an EtherCAT communication interface module.

Fast counter integrated in S500 modules

The following table shows the S500 modules which contain a fast counter and which of the digital inputs and outputs are reserved for the counter.

Module	Assigned inputs [1])		Assigned output	Remarks
	Channel A	Channel B	Channel C [2]) or (CF)	
DA501	DC16	DC17	DC18	The counter function is not available if the modules are mounted on the communication interface modules CI581-CN or CI582-CN
DA502	DC16	DC17	DC18 - in mode 1 and mode 2 DO0 - in mode 101 and mode 102 [3])	
DC522	C8	C9	C10	
DC523	C16	C17	C18	
DC532	C24	C25	C26	
DI524	I24	I25	No hardware output available	
DX522	I0	I1	The counter does not activate any relay output	
DC551-CS31	C16	C17	C18	The counter function is activated by setting the correct address on the module [4])
CI501-PNIO, CI541-DP, CI581-CN, CI521-MODTCP	DI0	DI1	DO0	
CI502-PNIO, CI542-DP, CI582-CN; CI522-MODTCP	DI8	DI9	DO8	
CI590-CS31-HA	C8	C9	C10	The counter function is activated by setting the correct address on the module [4])
CI592-CS31	DC8	DC9	DC10	The counter function is activated by setting the correct address on the module [4])

[1]) The two hardware inputs (channels A and B) are also and always available within the normal process image, irrespective of the operating mode of the counter.

[2]) The hardware output channel C is activated by the fast counter only in the operating modes 1 and 2.

[3]) Especially for module DA502: The counter operating mode 101 is the same as mode 1, but the assigned output is DO0 instead of DC18. Also the counter operating mode 102 is the same as mode 2, but the assigned output is DO0 instead of DC18.

[4]) The counter function of the CS31 communucation interface module can only be activated if a bus address greater than 70 is set on the module by means of the address rotary switches. In this case, the effective bus address equals the set address minus 70, and the counter is ready for operation. Example: A set bus address of 83 means that the effective bus address = (83 - 70) = 13 and that the integrated fast counter can be used. In this case the parameter “Fast counter operating mode” may not be 0!

The counter function is performed within the communication interface module and, accordingly, in the digital I/O module(s). It works independently of the user program and is therefore able to respond quickly to external signals. A simultaneous counter operation of several digital I/O modules is possible.

Each module counter can be configured for one out of 10 possible modes. The desired operating mode is selected in the PLC configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. While integrating a module containing a fast counter in the PLC configuration, the necessary operands are created and reserved immediately. Thus, a counter implementation carried out later on does not cause an address shift.

Features independent of the fast counter operating mode

- The pulses at the fast counters' inputs or the evaluated signals of the traces A and B in case of incremental position sensors are counted.
- The maximum counting frequency of digital I/O modules, operated at an AC500-CPU or CS31 communucation interface module, is 50 kHz. In certain operating modes, the maximum counting frequency is lower (see table *Chapter 1.6.5.2.9.8.1.2 “Operating modes” on page 6066*).
- The counting frequencies of the communication interface modules of PROFINET, PROFIBUS and CANopen are max. 200 kHz (in modes 1 to 6), max. 50 kHz (in mode 7), max. 35 kHz (in mode 9), and max. 20 kHz (in mode 10).
- If the modules DA501, DC522, DC523, DC532, DC551-CS31, CI592-CS31 and CI590-CS31-HA are used, each counting input must be circuited externally in series with a resistor of 470 Ω / 1 W, in order to safely avoid influences from the deactivated module outputs to the connected sensors.
- The positive signal edges are counted, if not noted differently.
- By setting the operating mode 0, the counting function is switched off. In this case, the reserved inputs and outputs can be used for other tasks. Simultaneous use of these terminals for the fast counter and other signals must be avoided.
- The fast counter's actual value is provided as a double word (32 bits).
- The fast counter can count upwards in all operating modes. It counts beginning at the start value (set value) up to the end value (max. from 0 to 4,294,967,295 or hexadecimal from 00 00 00 00 to FF FF FF FF. After reaching 4,294,967,295, the counter jumps with the next pulse to 0. When the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only when the fast counter is set again (set value), CF is reset to FALSE.

Further information

- Operating modes of the fast counter: *Chapter 1.6.5.2.9.8.1.2 “Operating modes” on page 6066*
- Configuration of the fast counter: *Chapter 1.6.5.2.9.8 “Fast counter” on page 6063*
- Operation with the library *Counter_AC500_V<>.lib*: *Chapter 1.5.4.9 “Counter library” on page 1037*

Fast counter in AC500-eCo (Onboard I/O in PM55x and PM56x)

For AC500 devices the function "fast counter" is available in S500 I/O modules as of firmware version V1.3.

For AC500-eCo devices the function "fast counter" is available in onboard I/Os of PM55x and PM56x.

In Processor Modules with onboard I/Os (AC500-eCo) fast counter functionality can be activated within the onboard I/O configuration. A Processor Module with onboard I/Os contains up to 2 fast counters (channel 0 and 1) according to the operating mode .

The fast counter can work in normal counter mode and A/B track counter mode. The normal counter detects the rising edge of the counter input. It will increase or decrease the count value at every rising edge. The A/B track counter is used to count the signal from a motion sensor. The counter can count with single phase, double phases or quad phases. In the following the behavior of the A/B track counter is described.

The fast counter cannot be used together with interrupt inputs at the same time.

Configuring the fast counter

The configuration of the fast counter is described in the chapter *Chapter 1.6.5.2.9.8 “Fast counter” on page 6063*.

Counting modes of the fast counter

The counting modes of the fast counters is described in the chapter *Chapter 1.6.5.2.9.8 “Fast counter” on page 6063*.

Operating the fast counter with the user program

The following function blocks can be used to operate the fast counter with help of user program:

Group: *Chapter 1.5.4.9 “Counter library” on page 1037*	
Chapter 1.5.4.9.1.2 “CNT_IO” on page 1043 *Chapter 1.5.4.9.1.3 “CNT_IO_EXT” on page 1050*	Fast counter of digital S500 I/O devices

Group: *Chapter 1.5.4.25 “Onboard IO library” on page 1733*	
Chapter 1.5.4.25.1.1 “ONB_IO_CNT” on page 1734	Handle fast counter on onboard I/Os

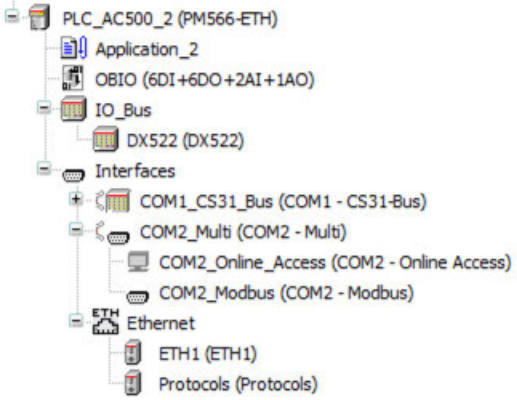
1.6.4.1.11 Special function blocks and programs

Function block COM_SET_PROT

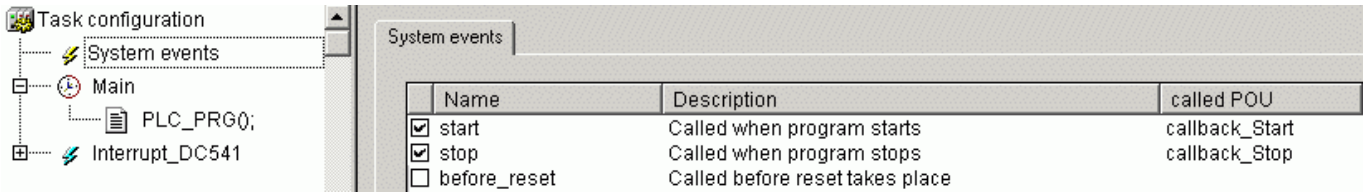
Function block *Chapter 1.5.4.19.3.1 “COM_SET_PROT” on page 1578* from SysInt_AC500_V10.lib can be used for different functions. In the following example COM2 is switched between "Online access" and Modbus (master):

1. In the device tree, append the *“COM2 - Multi”* protocol to the *“COM2”* node (*[Add object]*).
2. Append the *“COM2 - Online Access”* protocol to the *“COM2 - Multi”* node (Protocol with index 0).

3. Append the *"COM2 - Modbus"* protocol to the *"COM2 - Multi"* node (Protocol with index 1).



4. Set the system events START and STOP in the task configuration:



5. Call of block COM_SET_PROT in system events:

-

```
FUNCTION callback_Start: DWORD
VAR_INPUT
    dwEvent : DINT;
    dwFilter: DINT;
    dwOwner : DINT;
END_VAR
COM_SET_PROT(EN := FALSE);                      (* for edge
creation *)
COM_SET_PROT(EN := TRUE, COM := 2, IDX := 1);   (* switch to
Modbus *)

FUNCTION callback_Stop : BOOL
VAR_INPUT
    dwEvent : DINT;
    dwFilter: DINT;
    dwOwner : DINT;
END_VAR
COM_SET_PROT(EN := FALSE);                      (* for edge
creation *)
COM_SET_PROT(EN := TRUE, COM := 2, IDX := 0);   (* switch to
Online access *)
```

Sending/Receiving data with SysLibCom protocol

The following example shows how data is sent/received with the protocol "SysLibCom".

-> Telegrams of 32 bytes length are to be received and sent.

1. Declaration part of the program PROGRAM proSysLibCom_Test:

-

```
VAR

  strComSettings     : COMSETTINGS;             (* Structure of COM settings *)
  dwHandle           : DWORD;
  byStep             : BYTE;                    (* Step chain *)
  dwRead             : DWORD;                   (* Number of characters received
*)
  dwWritten          : DWORD;                   (* Number of characters sent *)
  bEnSend            : BOOL;                    (* Enable sending *)
  byCom              : BYTE := COM2;            (* COM number *)
  dwBaudrate         : DWORD := 19200;          (* Baudrate *)
  wLenRec            : WORD := 32;              (* Number of characters to be
received *)
  wLenTele           : WORD := 32;              (* Telegram length, here 32
characters for example *)
  wLenSend           : WORD := 32;              (* Number of characters to be
sent, for example 32 characters *)
  dwTimeoutSend      : DWORD := 0;              (* Timeout in [ms] for sending *)
  dwTimeoutRec       : DWORD := 0;              (* Timeout in [ms] for receiving *)
  abyRecBuffer       : ARRAY[0..271] OF BYTE;   (* Receive buffer, 272 bytes min.!
*)
  abyTeleBuffer      : ARRAY[0..543] OF BYTE;   (* Telegram buffer, 2 x receive
buffer min. *)
  aby SendBuffer     : ARRAY[0..271] OF BYTE;   (* Send buffer, telegram length
max.! *)
  strDataRec         : StrucReceiveData;        (* Structure of receive telegram
*)
  strDataSend        : StrucSendData            (* Structure of send telegram *)

END_VAR
```

2. Code part of the program

   -> Processing of a step chain containing the following steps:

   -

```
CASE byStep OF

0:  (* Step 0: Open the interface COMx -> SysComOpen -> get handle *)
    strComSettings.Port :=
byCom;                                                          (* COM_Number *)
    dwHandle :=
SysComOpen(strComSettings.Port);                                          (* Open
COM interface -> get handle *)
    byStep := SEL( dwHandle <> INVALID_HANDLE, 250,
1);                                  (* handle ok -> Step 1, otherwise error step
250 *)

1:  (* Step 1: Transfer of COMx interface parameters *)
    strComSettings.dwBaudRate :=
dwBaudrate;                                               (* Set baudrate *)
    (* Enter at this point the number of stop bits and parity, if necessary *)

    (* set COM settings -> if OK, run step 2, in case of an error step 250 *)
    byStep := SEL( SysComSetSettings(dwHandle, ADR(strComSettings)), 250, 2);

2:  (* Step 2: Initialization completed successfully -> now receiving and/or
sending *)

    (* Receive data: read all data received since last run, but wLenRec max.! *)
    dwRead := SysComRead (dwHandle, ADR(abyRead),
                          WORD_TO_DWORD(wLenRec),
dwTimeoutRec);                        (* READ DATA *)
    IF (dwRead > 0) THEN (* Number of characters received; in bytes *)
                         (* here, ignore for example all characters until valid
telegram start detected *)

                         (* Number of receiving cycles for the telegram *)
                         dwNumReadPerTele[byCom] := dwNumReadPerTele[byCom] + 1;
                         (* Copy data to buffer *)
                         SysMemCpy      (dwDest :=
ADR(abySumDataRead[dwSum]DataRead]),
                                         dwSrc := ADR(abyRead[0]),
                                         dwCount := dwRead );
                         (* Sum of read data of a telegram *)
                         dwSumDataRead := dwSumDataRead + dwRead;
                         IF dwSumDataRead >= wLenTele
THEN                                    (* Telegram complete ? *)
                                         dwRecCount := dwRecCount
+1;                       (* Number of telegrams received *)
                                         (* Copy received telegram to structure
strDataRec *)
                                         SysMemCpy( dwDest := ADR(strDataRec,
                                                    dw Src :=
ADR(abySumDataRead[0]),
                                                    dwCount := wLenTele );
                                         dwNumReadPerTele :=
0;                              (* init for following telegram *)
                                         dwSumDataRead := 0;

                                        (* here the evaluation of data starts !!!
*)


END_IF;                                                                (* Telegram
complete *)

END_IF;
    (* Data received *)

   (* Send data *)
   (* Enabling the sending of data can be done, for example, cyclical or by
program control *)
   IF bEnSend
THEN                                                                          (*
Enable sending *)
                                    (* Copy data to be sent to send buffer *)
                                    SysMemCpy (dwDest := ADR(abyDataSend[0]),
                                               dwSrc := ADR(strDataSend),
                                           dwCount := wLenSend);
```

```
                                        (* Send data *)
                                        dwWritten := SysComWrite( dwHandle,
                                                        ADR(abyDataSend[0]),

WORD_TO_DWORD(wLenSend),

dwTimeoutSend);               SEND DATA !!! *)
                                        IF (dwWritten <> wLenSend THEN
                                                      byStep :=
250;                           (* Error when sending *)

END_IF;                                               (* dwWritten *)
                                        bEnSend :=
FALSE;                                     (* Deactivate enable sending *)

END_IF;                                                          (* bEnSend *)

250:  (* Step 250: Error step -> Close COMx and start with step 0 *)
      bResult  :=
SysComClose(dwHandle);                                                    (* Close
COM interface *)
      dwHandle := 0;
        byStep := 0;

END_CASE;
         (* End of step chain *)
```

1.6.4.2 System technology of the AC500 communication modules

1.6.4.2.1 Ethernet communication modules

Ethernet

Frame formats

One fundamental part of the Ethernet specification is the arrangement of the data transfer format. When transferring data via Ethernet, the actual user data are preceded by a so-called preamble (which is among other things used to synchronize the receiver stations) as well as the hardware source and target address and a type length field. A checksum follows after the user data. All information mentioned above together constitute an Ethernet frame. During the development of the Ethernet, different types of frames arose.

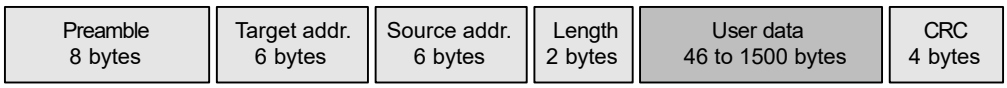


Fig. 1112: Structure of an Ethernet 802.3 frame

It has to be observed that the transferred user data do not inevitably contain only useful information. When transmitting data using a protocol above Ethernet (refer to 1.2.5), each protocol layer passed prior to the actual transmission supplements the original user data by its specific frame or header, so that the maximum number of actual user data is smaller, depending on the used protocols.

MAC address

Each Ethernet terminal device that has the MAC layer functions implemented (refer to 1.2.5 Ethernet and TCP/IP) has a world-wide unique hardware and MAC address. In this 6 bytes address, the two most significant bits of the first byte have specific functions. The most significant bit is also called the I/G bit (Individual/Group bit) and indicates whether it is an individual world-wide unique address (unicast address, I/G bit = 0) or a group address (I/G bit = 1). The second most significant bit is called the G/L bit and indicates whether it is a globally or a locally administered MAC address. A GAA (Globally Administered Address, G/L bit = 0) is an address which is fixed programmed by the device manufacturer and has to be unique all over the world. An LAA (Locally Administered Address, G/L bit = 1) can be a MAC address which has been changed afterwards for the use within a network. For this, it has to be observed that a MAC address has to be unique within a network.

The first 3 bytes of a MAC address are the manufacturer-related address part. Using this value, the manufacturer of an Ethernet chip can be determined. Each manufacturer of Ethernet components has one or several pre-defined address ranges assigned he can use for his products. 3COM, for example, uses among others the MAC address range 02-60-8C-xx-xx-xx.

For Ethernet, the MAC address is represented in a canonical form. This representation starts with the least significant bit (LSB) and ends with the most significant bit (MSB) of a byte. The following figure shows a global unicast address of the manufacturer 3COM.

Canonical representation 02-60-8C-00-00-01

Binary representation 01000000-00000110-00110001-00000000-00000000-10000000

Bus access methods

Ethernet uses the CSMA/CD access method (Carrier Sense Multiple Access / Collision Detection). With this method, the station that wants to transmit data, first "listens" to the carrier whether data are currently being transmitted by another station (carrier sense). If the carrier is busy, the station later tries to access the carrier again. If the carrier is idle, the station starts the transmission.

With this method, particularly in greater networks, it can happen that several stations try to transmit at the same time (multiple access). As a result, they "listen" to the carrier, detect that the carrier is free and correspondingly start the transmission. This can cause collisions between the different data packets. This is why each station verifies whether a collision occurred during transmission (collision detection). If this is the case, the station aborts the transmission and then tries to send its data again after a wait time which is determined by a random generator.

Collisions within an Ethernet network do not cause loss of data, but they reduce the available bandwidth of the network. In practice, for a network with 30 stations on the bus, a net bandwidth of approx. 40 % is assumed. This means that a bandwidth of only approx. 4 Mbit/s is available in a network with a theoretical bandwidth of 10 Mbit/s, for instance. This has to be considered when planning an Ethernet network. The number of collisions can be reduced to a minimum if the network is carefully planned and if only suitable network components are used (refer to 1.4 Cabling and 1.5.4 Media converters).

Half duplex and full duplex

If communication is only possible in one direction (transmission or reception), this is called half duplex mode. However, the separate transmit and receive lines of today's twisted pair cabling for Ethernet networks also allow full duplex operation. In full duplex mode, the stations can simultaneously exchange data in both directions independent from each other. Due to this, the CSMA/CD method is not necessary in full duplex mode. Networks with more than two stations working in full duplex mode can only be implemented using switches because these switches establish peer-to-peer connections between the individual stations (refer to 1.4 Cabling).

Auto negotiation

Today, Ethernet uses transmission rates of 10, 100 or 1000 Mbit/s in half duplex or in full duplex mode. However, not all devices support all possible settings. This particularly makes the optimum network configuration more difficult for networks using twisted pair cables of the same kind and components which can be used with 10 Mbit/s or 100 Mbit/s in half duplex or in full duplex mode as desired. Imperfect configurations can lead to link errors or at least to performance losses because the maximum possible transmission rate is not used.

Due to this, the auto negotiation functionality (in the past also called Nway) has been established with the introduction of Fast Ethernet. With this functionality, the stations agree on the highest possible transmission rate and, if possible, full duplex operation. Then, all subscribers on the network configure themselves optimally.

However, problems could arise if one component in one segment is configured manually, i.e. if it has been set to a fixed transmission rate and mode and the auto negotiation function has been switched off. In this case, a device operating in auto negotiation mode informs the manually configured device about its possible settings but does not receive any response.

Ethernet and TCP/IP

Like nearly all standards in the field of data transmission, Ethernet also follows the ISO/OSI layer model. Based on this reference model, the principle course of a transmission is described. Each of the 7 parts (layers) has a particular function and makes it available for the next higher layer.

The Ethernet standard IEEE-802.3 defines the function of the two lowest layers. These layers consist of the following components and the Logical Link Control (LLC) which is described in the IEEE standard 802.2.

- Media Access Control Protocol (MAC)
- Physical Layer Signalling (PLS)
- Attachment Unit Interface (AUI)
- Medium Dependent Interface (MDI)
- Physical Medium Attachment (PMA)

Since the ISO/OSI model did not yet exist when the development of TCP/IP protocols started, these protocols are based on the DoD architecture. The DoD model cannot be clearly transferred to the ISO/OSI model. The following figure shows a comparison of Ethernet in the ISO/OSI model and the TCP/IP protocols adapted to that model. This shall explain that Ethernet does not necessarily mean TCP/IP (and vice versa). To be precise, TCP/IP is only based on Ethernet and can also be used in other data networks (e.g. for satellite links). In return, TCP/IP is not the only protocol used in Ethernet networks. Actually, TCP/IP is only one of numerous protocols which are used side by side.

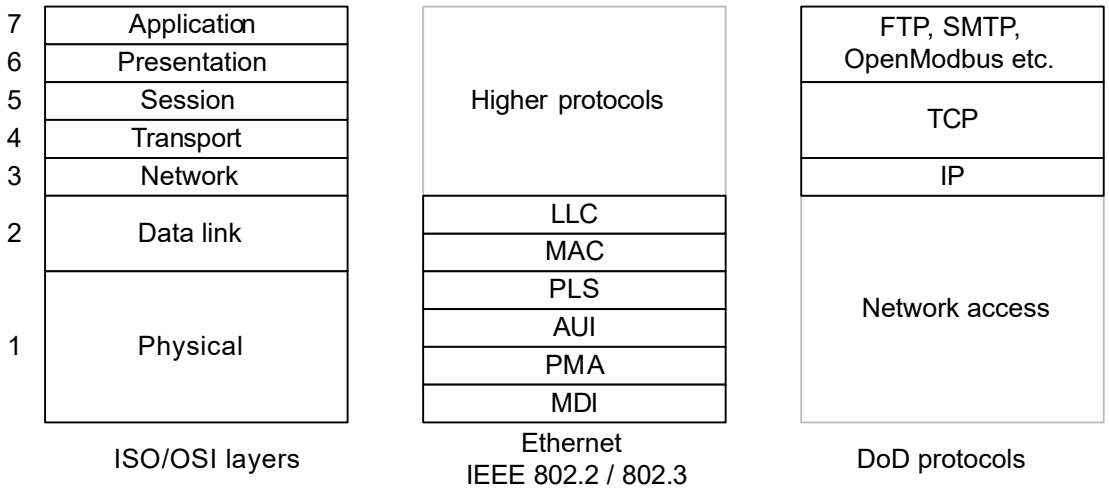


Fig. 1113: Ethernet in the ISO/OSI model

Supported protocols

IP - Internet protocol (RFC 791)

- Freely configurable IP address and network mask
- Configurable IP address of the standard gateway
- IP datagram size: 1500 bytes max.
- Route cache size: 32 entries
- Route timeout: 900 seconds
- Number of IP multicast groups: 64 for reception, unlimited for transmission

IP - Internet protocol (RFC 791)

TCP - Transmission control protocol (RFC 793, RFC 896)

Amount of user data for TCP telegrams: 1460 bytes max.

UDP - User datagram protocol (RFC 768)

Amount of user data for UDP telegrams: 1472 bytes max.

BOOTP - Bootstrap protocol (RFC 951, RFC 1542, RFC 2132)

DHCP - Dynamic host configuration protocol (RFC 2131, RFC 2132)

OpenModbus

- Client and/or server mode (several times)
- Up to 8 simultaneous client or server connections
- Supported function codes: 1, 2, 3, 4, 5, 6, 7, 15, 16. As of FW 2.2: in addition 22, 23

- Maximum amount of data per telegram: 100 coils (words) or 255 registers (bits)
- Configurable connection monitoring functions

NetIdent

- Devices can be identified and accessed via the network (even unconfigured devices)
- Unique identification and localization via rotary switch on the devices

ARP - Address resolution protocol (RFC 826)

- ARP Cache size: 64 entries
- ARP Timeout: 600 seconds

ICMP - Internet control message protocol (RFC 792)

IGMPv2 - Internet group management protocol, version 2 (RFC 2236)

Sockets

- Number of sockets: 16
- Socket options can be set individually

Restrictions

- IP fragmentation is not supported
- TCP Urgent Data is not supported
- TCP port 0 is not supported
- TCP port 502 is reserved for OpenModbus
- TCP port 1200 is reserved for gateway access
- UDP port 67 is reserved for BOOTP and DHCP
- UDP port 25383 is reserved for NetIdent protocol
- UDP port 32768 is reserved for UDP blocks

Designing and planning a network

Introduction

To obtain optimum performance within a network, it is absolutely essential to plan the network beforehand. This applies to both the initial installation as well as its expansion. Rashly installed networks can not only cause poor network performance, they even can lead to a loss of data since restrictions given by the standard are possibly not kept. At first glance, designing a network causes additional costs, but it will later reduce maintenance expenditures during operation.

The following sections shall explain some principle methods for determining a suitable network structure and give some hints how to find out the network utilization and performance.

Concepts for structuring a network

Designing and planning a network

To obtain optimum performance within a network, it is absolutely essential to plan the network beforehand. This applies to both the initial installation as well as its expansion. Rashly installed networks can not only cause poor network performance, they even can lead to a loss of data since restrictions given by the standard are possibly not kept. At first glance, designing a network causes additional costs, but it will later reduce maintenance expenditures during operation.

The following sections shall explain some principle methods for determining a suitable network structure and give some hints how to find out the network utilization and performance.

Models

Regarding the network technology the following three general models are distinguished:

- Hierarchy model
- Redundant model
- Safe model

The selection of the suitable model as a basis for planning a network depends on the specific requirements of the installation. Office networks are typically built up based on the hierarchy model since the individual clients do not very often exchange data with each other but only periodically contact the server. Installation-internal networks which do not have any connection to the company network often only consist of automation devices and do not have a server. The connected controllers transmit data in short intervals directly to each other. Furthermore, the operational safety of installation-internal networks has a higher importance since data transmission malfunctions can result in incorrect behavior of the installation or even in production stops. In such cases it is more suitable to choose the redundant model or a safe model.

In the end, all three models shown above are based on the use of switching hubs (switches). Whereas in the past simple hubs were increasingly used to set up a network wherever permitted by the requirements, today almost exclusively switches are used. Using switches, historic Ethernet rules such as the length restrictions of a collision domain no longer have to be observed. This considerably simplifies the network design. Even though the use of switches could make us believe that networks can be expanded to infinite size, it has to be considered that each switch involved in a data transfer causes a delay. Therefore, the IEEE-802.1d bridging standard recommends to limit the number of switches to be passed between two terminal devices to a maximum of seven switches.

Hierarchy model

The hierarchy model intends the subdivision of the network into several levels and a graduation of the data rate between the individual levels. For this purpose, normally at least two grades are used e.g. by connecting the server with a data rate of 100 Mbit/s to the network and the clients with 10 Mbit/s. The advantage of this design is that the server has 10 times the bandwidth of the clients available which enables the server to provide sufficient bandwidth and response time for several clients. Despite the fact that 10 times the bandwidth does not mean that 10 clients can simultaneously access the server, the data transmitted to or from the clients do only need one tenth of the time. In total, this reduces the response time for each individual client.

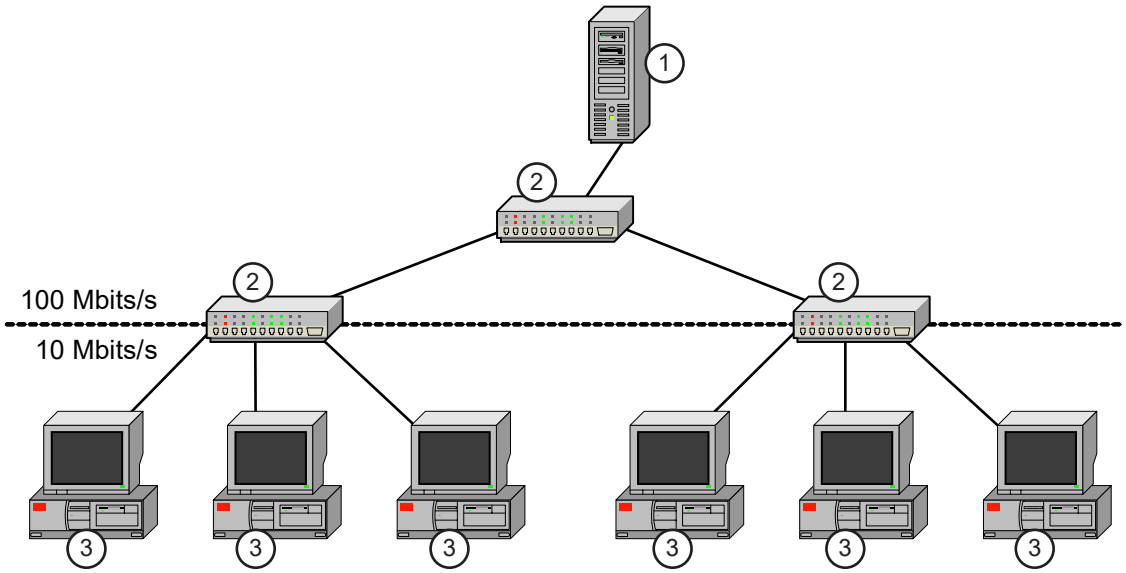


Fig. 1114: Hierarchy model

1	Server
2	Switch
3	Client

When dimensioning the individual levels the utilization of the particular level has to be considered. Devices connected to each other via hubs can only be operated in half-duplex mode. Consequently they have to share the commonly used network (shared media). If the utilization of such a shared media is higher than 40 % over a longer period of time, a switch should be used instead of a hub in order to subdivide the collision domain and thus remove load from it. The utilization threshold within such a switched media is 80 %. If this value is exceeded, the utilization should be reduced by selecting a smaller grouping.

Redundant model

The meshed Ethernet structure is a typical example for a redundant network model. To obtain fault tolerance, several connections have to be established between switches or nodes. This way, data exchange can be performed using another (redundant) connection if one connection fails.

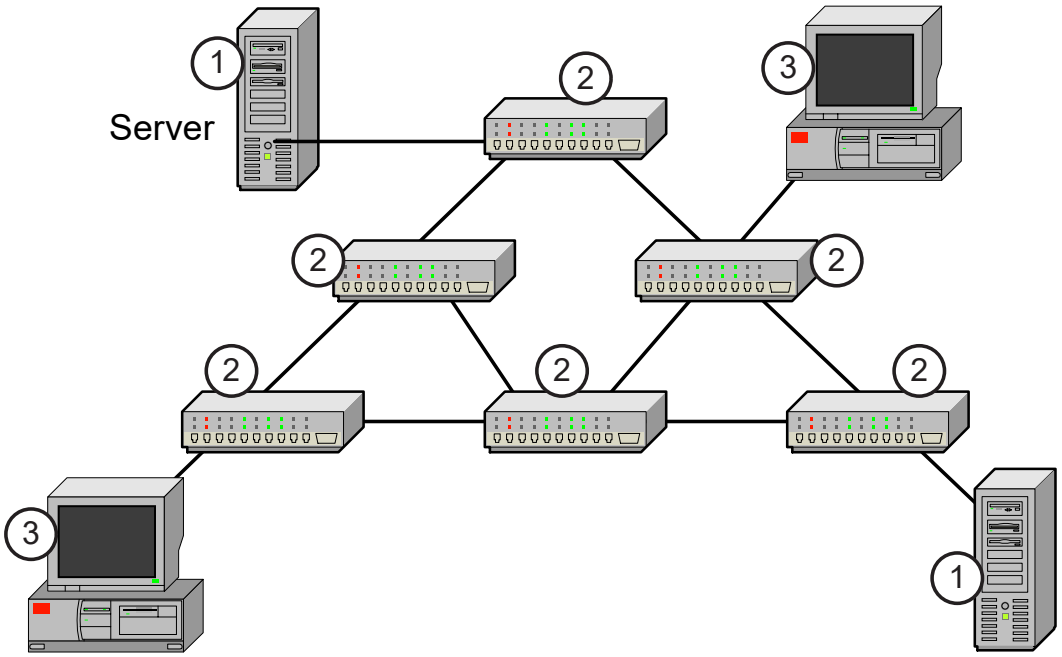


Fig. 1115: Redundant model

1	Server
2	Switch
3	Client

However, this meshed constellation leads to loops which would make well-ordered data exchange impossible. The loops would cause the broadcast or multicast data packets to endless stray in the network. In order to suppress such loops, the spanning tree mechanism ↳ *Chapter 1.6.4.2.1.3.3 “Network components” on page 5518* is used which always activates only one unique connection and deactivates all other possible connections. On the occurrence of a fault (e.g. caused by an interruption of the network line) the redundant connection is re-activated and then maintains communication between the switches. However, switching of the connection is not without interruption. The time needed for switching depends on the size and structure of the network.

The use of link aggregation which is often also called "trunking" likewise provides increased transmission reliability. Link aggregation actually means the parallel connection of several data lines. This way the bandwidths of the individual data lines are bundled in order to increase the total bandwidth. Furthermore, the parallel connection establishes a redundant connection. If one data line fails, the data can still be transmitted via the remaining lines even though only with reduced bandwidth.

Safe models

To obtain a certain grade of safety for the transmitted data against unauthorized access or to optimize the network utilization, it is suitable to design so-called Virtual Bridged Local Area Networks (VLANs). In a VLAN the data flow is grouped. The simplest variant of a VLAN is obtained by a port-related grouping which means that particular ports of a switch are assigned to a VLAN and data exchange is then only performed within this VLAN. A VLAN can be considered as a group of terminal stations which communicate like in a usual LAN although they can be located in different physical segments. In the end, establishing VLANs leads to a limitation of the broadcast domains. As a result, all subscribers of a VLAN only receive data packages which have been sent by subscribers of the same VLAN. Independent of their physical location, all subscribers of a VLAN are logically put together to one broadcast domain. The limitation of the broadcast domains relieves load from the network and provides safety since only the members of the VLAN are able to receive the data packets.

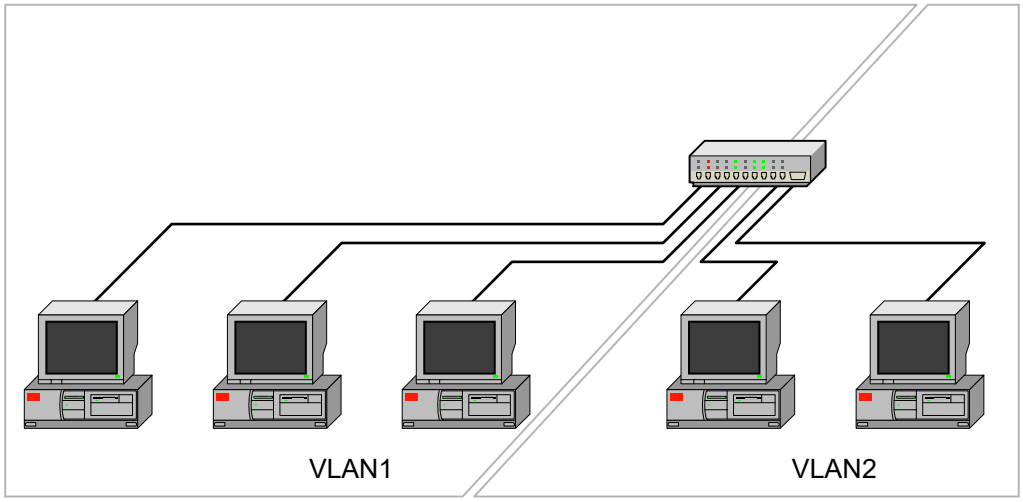


Fig. 1116: Safe models

In order to enable a terminal device connected to a switch to exchange data beyond the borders of the VLAN, the port of the switch has to be assigned to several VLANs. Apart from the simple variant of the port-based VLAN, it is also possible to establish VLANs by evaluating additional information contained in the Ethernet frames.

Utilization and performance

In the description of the network models it has already been mentioned that the existing hubs should be replaced by switches in order to subdivide the collision domain and thus remove load, if the utilization of a shared media is higher than 40 % over a longer period of time. If the utilization within such a switched media is permanently above 80 %, it is recommended to further relieve load by performing smaller grouping.

However, a network should basically not be dimensioned for the burst utilization. During normal operation usually many smaller data packets are transmitted rather than large data streams. This means that the network load regarding the bandwidth is not as high. Nevertheless, if any bottle-necks occur, the simplest method to eliminate them is to increase the data rate (e.g. from 10 Mbit/s to 100 Mbit/s). In existing networks, however, this is not always possible without problems since the cable infrastructure is possibly not suitable for the higher data rate and the expenditure for a new cabling is possibly not defensible. The only solution in such cases is a segmentation of the network which results in a reduction of the number of devices within the network or collision domain and thus provides more bandwidth for the remaining devices.

A segmentation of a network can be obtained with routers, bridges or switches. However, segmentation is only meaningful if the 80/20 rule is considered and observed. The 80/20 rule says that 80 % of the data traffic have to take place within the segment and only 20 % of the data traffic are forwarded to another segment. This is why a previous analysis of the network traffic is required to enable meaningful grouping. In this analysis it has to be determined which station is communicating with which other stations in the network and which amount of data is flowing for this communication. For shared media the network should be divided in a way that stations producing roughly the same load should be grouped in one collision domain, if it is not possible to make a division based on the communication paths. This way it is guaranteed that stations with lower data traffic are able to meet the typical requirements regarding short response times. Stations with permanently high data traffic generally cause a drastic increase of the response times.

Best performance increase can be obtained by using switches and connecting each single station directly to the switches. This way each station has its own connection to a switch and thus can use the full bandwidth of a port in full-duplex mode. This subdivision and the provision of the dedicated connections is called micro-segmentation. For micro-segmentation the 80/20 rule does no longer apply. It has only to be guaranteed that a switch is able to provide sufficient internal bandwidth.

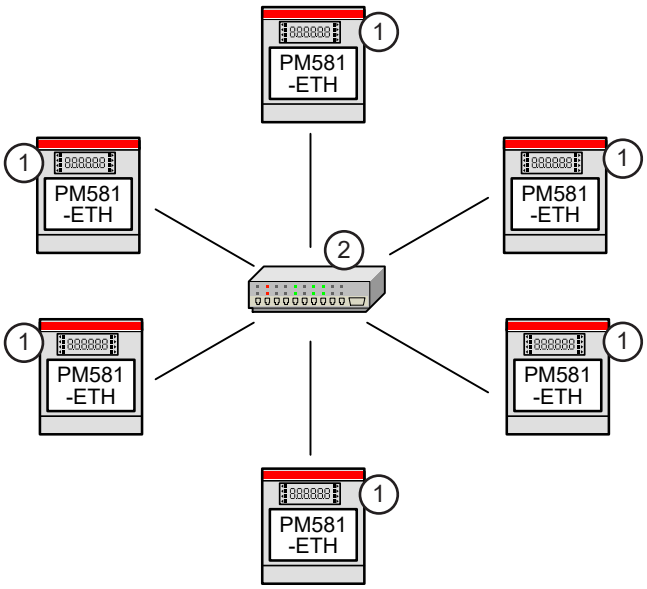


Fig. 1117: Direct connection of all stations to switches

1 AC500 Processor Module with Ethernet
2 Switch

In order to plan a network with optimum performance, we have to think about the question what a network is able to achieve at all. Taking the standards as a basis it can be determined how many data per time can be transmitted via a network theoretically. The smallest Ethernet frame size is 64 bytes long and contains 46 bytes of user data, the maximum frame size is 1518 bytes at 1500 bytes of user data, each plus 64 bits for the preamble and 96 bits for the inter-frame gap. This results in a minimum length of 672 bits (64 x 8 + 64 + 96) and a maximum length of 12304 bits (1518 x 8 + 64 + 96). The transmission of one bit takes 10 ns for fast Ethernet (100 Mbit/s) and 100 ns for Ethernet (10 Mbit/s). Using these values we can calculate how many data packets of the smallest and the maximum length can be transmitted per second theoretically (see tables). The calculation of the corresponding amount of user data which can be transmitted (without taking into account the additional overheads of the higher protocols) now shows the considerably higher protocol overhead caused by the small data packets.

Table 696: Data rate at 10 Mbit/s:

10 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	100	67 200	14 880	46	684 480
max. frame	12 304	100	1 230 400	813	1 500	1 219 500

Table 697: Data rate at 100 Mbit/s:

100 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	10	6 720	148 800	46	6 844 800
max. frame	12 304	10	123 400	8 127	1 500	12 195 000

The corresponding net bandwidth can be calculated from the ratio of the amount of user data per second to the available network bandwidth. The net bandwidth is independent of the transmission rate and calculated in the following table taking a transmission rate of 100 Mbit/s as an example.

Table 698: Net bandwidth at 100 Mbit/s:

100 Mbit/s	User data [bits/s]	Network bandwidth [bits/s]	Net bandwidth [%]
min. frame	54 758 400	100 000 000	54.7
max. frame	97 524 000	100 000 000	97.5

These calculations point out that the percentage of the network performance is considerably higher for the transmission of larger frames. The efficiency of the data transmission which is independent of the transmission rate is shown in the following table using some selected frame sizes as an example. However, the values given in the table only consider the protocol overhead of the MAC and the network layer. The user data are reduced accordingly by the additional overhead of the corresponding higher layers.

Table 699: Efficiency of data transmission:

User data [bits]	Frame size [bits]	Overhead [%]	Efficiency [%]
1500	1518	1.2	98.8
982	1000	1.8	98.2
494	512	3.6	96.4
46	64	39.1	60.9

A calculation of the typical transmitted frame sizes may be still possible for small closed networks inside an installation with only automation devices connected. But, for instance, if PCs are additionally connected to the network (even if they are connected only temporarily) the frame sizes can vary considerably. This makes it impossible to perform an exact calculation of the bandwidth or to make a precise statement regarding the performance. However, the following index values could be determined with the help of various studies about network performance.

- For low utilization of 0 to 50 % of the available bandwidth, short response times can be expected. The stations are able to send frames with a typical delay of smaller than 1 ms.
- For medium utilization between 50 and 80 %, the response times can possibly increase to values between 10 and 100 ms.
- For high utilization over 80 %, high response time and wide distribution can be expected. The sending of frames can possibly take up to 10 seconds.

This is why the following principles should be observed when designing an Ethernet network.

- Mixed operation of stations which have to transmit high data volumes and stations which have to operate with short response times (real time) should be avoided. Due to the wide distribution, short response times cannot be guaranteed within such combinations.
- As few as possible stations should be positioned inside of one collision domain. For this purpose, collision domains should be subdivided using switching hubs.

Utilization and performance

In the description of the network models it has already been mentioned that the existing hubs should be replaced by switches in order to subdivide the collision domain and thus remove load, if the utilization of a shared media is higher than 40 % over a longer period of time. If the utilization within such a switched media is permanently above 80 %, it is recommended to further relieve load by performing smaller grouping.

However, a network should basically not be dimensioned for the burst utilization. During normal operation usually many smaller data packets are transmitted rather than large data streams. This means that the network load regarding the bandwidth is not as high. Nevertheless, if any bottle-necks occur, the simplest method to eliminate them is to increase the data rate (e.g. from 10 Mbit/s to 100 Mbit/s). In existing networks, however, this is not always possible without problems since the cable infrastructure is possibly not suitable for the higher data rate and the expenditure for a new cabling is possibly not defensible. The only solution in such cases is a segmentation of the network which results in a reduction of the number of devices within the network or collision domain and thus provides more bandwidth for the remaining devices.

A segmentation of a network can be obtained with routers, bridges or switches. However, segmentation is only meaningful if the 80/20 rule is considered and observed. The 80/20 rule says that 80 % of the data traffic have to take place within the segment and only 20 % of the data traffic are forwarded to another segment. This is why a previous analysis of the network traffic is required to enable meaningful grouping. In this analysis it has to be determined which station is communicating with which other stations in the network and which amount of data is flowing for this communication. For shared media the network should be divided in a way that stations producing roughly the same load should be grouped in one collision domain, if it is not possible to make a division based on the communication paths. This way it is guaranteed that stations with lower data traffic are able to meet the typical requirements regarding short response times. Stations with permanently high data traffic generally cause a drastic increase of the response times.

Best performance increase can be obtained by using switches and connecting each single station directly to the switches. This way each station has its own connection to a switch and thus can use the full bandwidth of a port in full-duplex mode. This subdivision and the provision of the dedicated connections is called micro-segmentation. For micro-segmentation the 80/20 rule does no longer apply. It has only to be guaranteed that a switch is able to provide sufficient internal bandwidth.

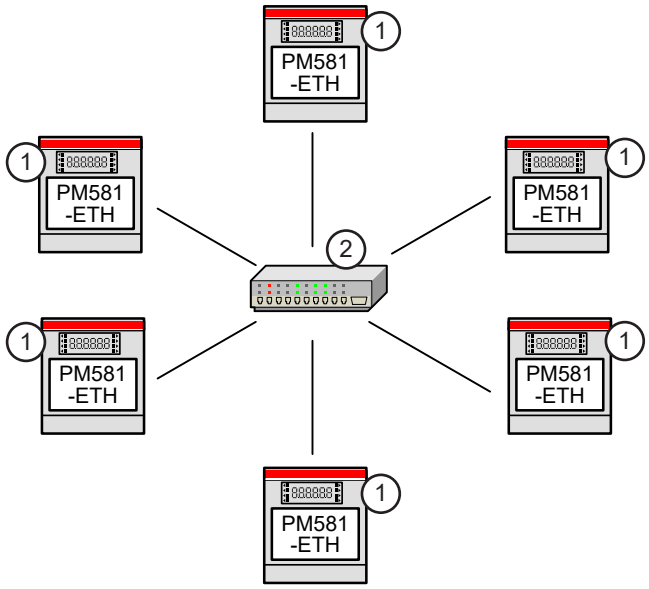


Fig. 1118: Direct connection of all stations to switches

1 AC500 Processor Module with Ethernet
2 Switch

In order to plan a network with optimum performance, we have to think about the question what a network is able to achieve at all. Taking the standards as a basis it can be determined how many data per time can be transmitted via a network theoretically. The smallest Ethernet frame size is 64 bytes long and contains 46 bytes of user data, the maximum frame size is 1518 bytes at 1500 bytes of user data, each plus 64 bits for the preamble and 96 bits for the inter-frame gap. This results in a minimum length of 672 bits (64 x 8 + 64 + 96) and a maximum length of 12304 bits (1518 x 8 + 64 + 96). The transmission of one bit takes 10 ns for fast Ethernet (100 Mbit/s) and 100 ns for Ethernet (10 Mbit/s). Using these values we can calculate how many data packets of the smallest and the maximum length can be transmitted per second theoretically (see tables). The calculation of the corresponding amount of user data which can be transmitted (without taking into account the additional overheads of the higher protocols) now shows the considerably higher protocol overhead caused by the small data packets.

Table 700: Data rate at 10 Mbit/s:

10 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	100	67 200	14 880	46	684 480
max. frame	12 304	100	1 230 400	813	1 500	1 219 500

Table 701: Data rate at 100 Mbit/s:

100 Mbit/s	Length [bits]	Time/bit [ns]	Time/frame [ns]	Frames [ns]	User data/ frame [1/s]	User data [bytes/s]
min. frame	672	10	6 720	148 800	46	6 844 800
max. frame	12 304	10	123 400	8 127	1 500	12 195 000

The corresponding net bandwidth can be calculated from the ratio of the amount of user data per second to the available network bandwidth. The net bandwidth is independent of the transmission rate and calculated in the following table taking a transmission rate of 100 Mbit/s as an example.

Table 702: Net bandwidth at 100 Mbit/s:

100 Mbit/s	User data [bits/s]	Network bandwidth [bits/s]	Net bandwidth [%]
min. frame	54 758 400	100 000 000	54.7
max. frame	97 524 000	100 000 000	97.5

These calculations point out that the percentage of the network performance is considerably higher for the transmission of larger frames. The efficiency of the data transmission which is independent of the transmission rate is shown in the following table using some selected frame sizes as an example. However, the values given in the table only consider the protocol overhead of the MAC and the network layer. The user data are reduced accordingly by the additional overhead of the corresponding higher layers.

Table 703: Efficiency of data transmission:

User data [bits]	Frame size [bits]	Overhead [%]	Efficiency [%]
1500	1518	1.2	98.8
982	1000	1.8	98.2
494	512	3.6	96.4
46	64	39.1	60.9

A calculation of the typical transmitted frame sizes may be still possible for small closed networks inside an installation with only automation devices connected. But, for instance, if PCs are additionally connected to the network (even if they are connected only temporarily) the frame sizes can vary considerably. This makes it impossible to perform an exact calculation of the bandwidth or to make a precise statement regarding the performance. However, the following index values could be determined with the help of various studies about network performance.

- For low utilization of 0 to 50 % of the available bandwidth, short response times can be expected. The stations are able to send frames with a typical delay of smaller than 1 ms.
- For medium utilization between 50 and 80 %, the response times can possibly increase to values between 10 and 100 ms.
- For high utilization over 80 %, high response time and wide distribution can be expected. The sending of frames can possibly take up to 10 seconds.

This is why the following principles should be observed when designing an Ethernet network.

- Mixed operation of stations which have to transmit high data volumes and stations which have to operate with short response times (real time) should be avoided. Due to the wide distribution, short response times cannot be guaranteed within such combinations.
- As few as possible stations should be positioned inside of one collision domain. For this purpose, collision domains should be subdivided using switching hubs.

Network components

The topology of an Ethernet network is like a star or tree structure. Up to two stations can be connected to each segment where active distribution devices like hubs or switches are also considered as a station. The following figure shows an example of a simple Ethernet network.

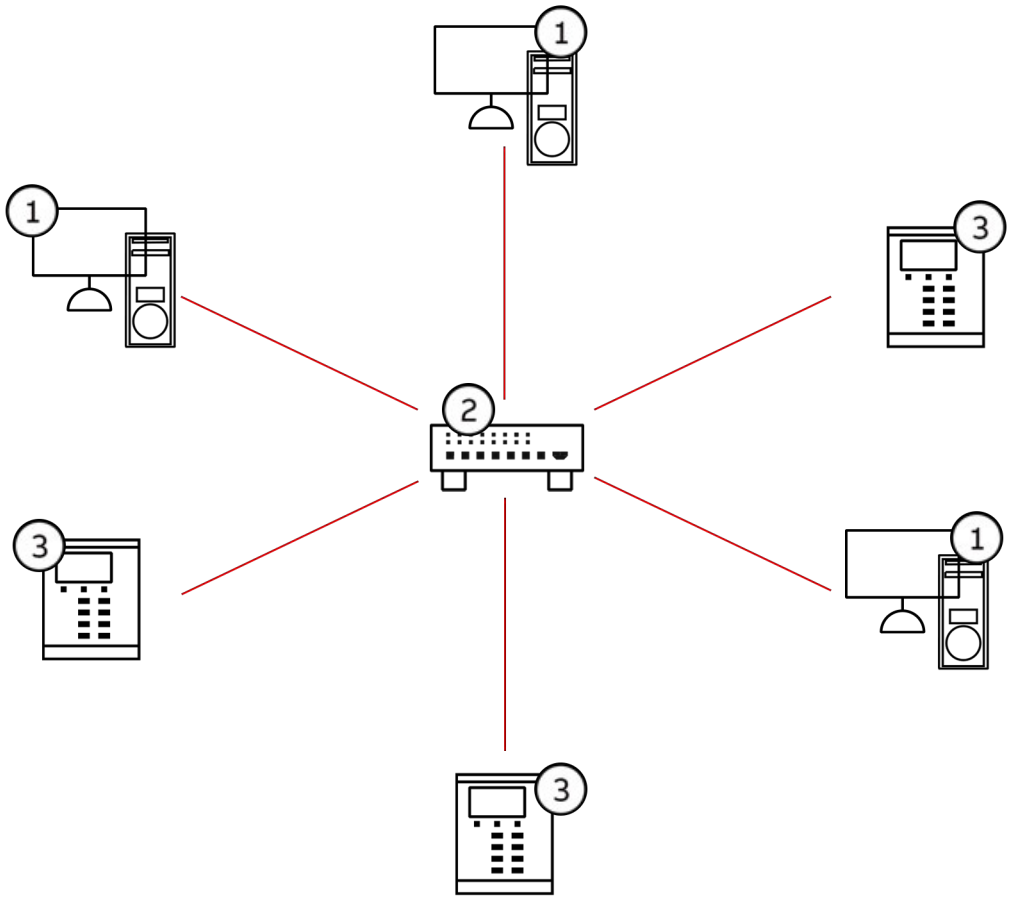


Fig. 1119: Example of a simple network

1 PC
2 Hub/Switch
3 AC500 processor module with Ethernet

The following sections introduce the different types of components required for a network.

Terminal devices

Terminal devices are devices that are able to send and receive data via Ethernet, e.g. controllers with an Ethernet Communication Module or PCs with an integrated network adapter. With this, one of the essential functions of a network adapter is to transfer all data packets to the PC itself instantly and without any loss. Occurring defiles or even errors can cause data packets to be lost. Such losses of data have to be got under control by higher protocols (e.g. TCP/IP) which results in considerable performance reductions. The direct implementation of higher protocols on the network adapter can increase the performance and save the resources of the host system (e.g. controller).

Repeaters and hubs

At the dawning of the Ethernet, the repeaters had only two network connections and were used to connect two segments to each other in order to extend the segment length. Later, repeaters with more than two network connections were available. Those star distributors are called hubs. They are able to connect several segments. Apart from the number of network connections the functionality of hubs and repeaters is identical. This is why we only use the term "hub" in the following descriptions.

Hubs are operating on the lowest layer of the ISO/OSI model and are therefore independent of the protocols used on Ethernet. The network connections of hubs are exclusively operated in half duplex mode. Due to this, collision domains can freely propagate beyond the hubs. A hub can only support one transmission rate for all connections. Therefore it is not possible to connect segments with different transmission rates via a simple hub. For this purpose a dual-speed hub has to be used. The fundamental functions of hubs are as follows:

- Restoration of the signal magnitude
- Regeneration of the signal timing
- Propagation of a detected collision
- Expansion of short fragments
- Creation of a new preamble
- Isolation of a faulty segment

When transmitted over the medium (e.g. a twisted pair cable) the data signal is attenuated. The task of a hub is to amplify an incoming signal in order to make the full signal magnitude available at the outputs again. Furthermore, a distortion of the binary signal's on-off ratio (jitter) can occur during data transmission. When transmitted via a hub, the hub is able to restore the correct on-off ratio of the signal which avoids propagation of the signal jitter beyond the segment.

However, one of the most important tasks of a hub is to propagate occurring collisions within the entire collision domain so that the collision can be detected by all connected stations. If it detects a collision on one of its connections, the hub sends a so-called jam signal over all connections. If a hub receives a data fragment which, by its principle, could only be created by a collision, it first brings the fragment to a length of 96 bits and then forwards it via the ports. This shall guarantee that the data fragment can be received by all stations independent of their distance to the hub and removed from the network. The detected data fragments are removed by the terminal devices by not forwarding them to the higher layers.

By means of the data packet preamble the beginning of a data packet is detected so that the recipient can synchronize to the incoming data stream. However, during the data transmission it can occur that the first bits of a preamble are lost. The task of the hub is to restore a possibly incomplete preamble before forwarding it.

If collisions occur within one segment in large numbers in a short period of time or if e.g. a short circuit on a data line causes failures, the hub switches off the faulty segment to avoid interference to the entire collision domain.

10 Mbit/s hubs

The 10Base-T connections of a 10 Mbit/s hub are implemented as MDI-X ports and therefore already crossed internally. The advantage is that the terminal devices can be directly connected using 1:1 twisted pair cables and no crossover cables are required. Some hubs additionally have a so-called uplink port which can be used to connect another hub. In order to also enable the use of a 1:1 cable for this port, it is implemented as a normal non-crossed MDI port. In many cases this port can also be switched between MDI and MDI-X or is implemented as a double port with two connections in parallel (1 x MDI, 1 x MDI-X). In this case, it has to be observed that these parallel ports may only be used alternatively and not at the same time.

Hubs are normally equipped with several LEDs for status indication. So, for example a Link LED indicates the correct connection between the terminal device and port at the hub. This way, incorrect cabling can be quickly detected. Further LEDs indicate for example the data traffic over a port or the collisions.

The maximum permitted number of 10 Mbit/s hubs within one collision domain is limited to 4. This restriction is due to two reasons. One reason is that the bit period time delay, which is inevitably increased by each hub, must not exceed 576 bit periods. The second reason is that the so-called interframe gap (IFG) must not be shorter than at least 47 bit periods. The interframe gap describes the time interval between two data packets and shall allow the receiving stations to recover from the incoming data stream. However, the regeneration of an incomplete preamble performed by the hub reduces the time between the data packets due to the completion of possibly missing bits.

One possibility to get round the restriction to four hubs is the use of stackable hubs. These hubs are connected to each other via a special interface instead of using the uplink port and therefore constitute one logic unit. As a result, they appear as one single big hub to the external.

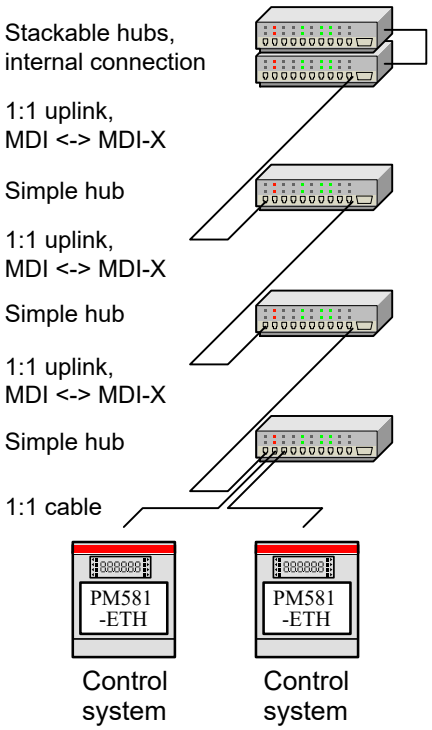


Fig. 1120: Stackable hubs

100 Mbit/s hubs

The principle operation of 100 Mbit/s hubs is like the 10 Mbit/s hubs. However, the hubs for 100 Mbit/s Ethernet have to be additionally distinguished to class I and class II hubs.

Class I hubs (or class I repeaters) are able to connect two segments with different transmission media. For this purpose the complete data stream has to be decoded on the receiving side and encoded again on the transmission side according to the transmission medium. This conversion process leads to higher delay times. Due to this, only one class I hub is permitted within one collision domain.

In contrast, class II hubs support only one transmission medium. No conversion of the data stream is required. This leads to shorter delay times compared with class I hubs. This is why for two segments with a maximum length of 100 m each, up to two class II hubs which are again connected to each other via a 5 m long segment can be used within one collision domain.

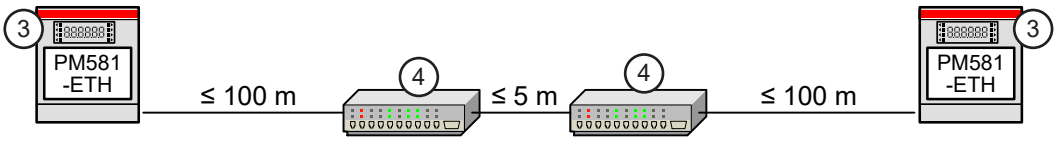


Fig. 1121: Use of a class II hub

3 AC500 Processor Module with Ethernet
4 Class II hub

10/100 Mbit/s dual-speed hubs

In contrast to the simple hubs, dual-speed hubs are able to support two transmission rates and thus enable to connect two Ethernet networks with different data rates to each other. Dual-speed hubs are internally structured like two separate hubs or paths (one for each data rate). By means of the auto negotiation function the transmission rate of the connected station is determined and automatically switched to the corresponding path. Each internal path is a separate hub. For the temporary storage of the data packets, the paths are connected to each other via an internal switch. Dual-speed hubs likewise operate in half duplex mode. However, the internal switch provides a clear separation of the 10 Mbit/s and the 100 Mbit/s side so that unlike the simple hubs a collision domain cannot reach beyond the borders of the corresponding side of the dual speed hub.

Bridges, switches and switching hubs

Basically the terms bridge, switch and switching hub designate the same. In the early beginning of the Ethernet, the term bridge was formed by the fact that a bridge had only two network connections. Later, so-called multiport bridges with several connections came up which were also called switches or switching hubs. This is why we use the common term "switch" in the following descriptions for all the components mentioned above.

The use of a switch is another variant of connecting network segments to each other. The decisive difference between a hub and a switch is that a switch is operating on the second layer of the ISO/OSI model, the MAC layer.

The following sections describe the functionality of such a layer 2 switch. For reasons of completeness it has to be mentioned that switches operating on higher and therefore protocol-specific layers also exist.

Using a switch, load separation between networks can be implemented which leads to an increased performance due to the reduced load of the individual segments. In contrast to a hub, a switch does not operate transparently (i.e. it doesn't forward all data packets via all ports) but decides on the basis of the MAC target address whether and via which port an incoming data packets has to be forwarded. The data packet is only forwarded if the target station is located in another segment or if the target address of the data packet contains a multicast or broadcast address.

As already mentioned, the decisive advantage of a switch is the logical separation of networks. Therefore, a switch represents a border for a collision domain. Aside from the performance improvement, the use of a switch allows a network to be extended beyond the usual borders.

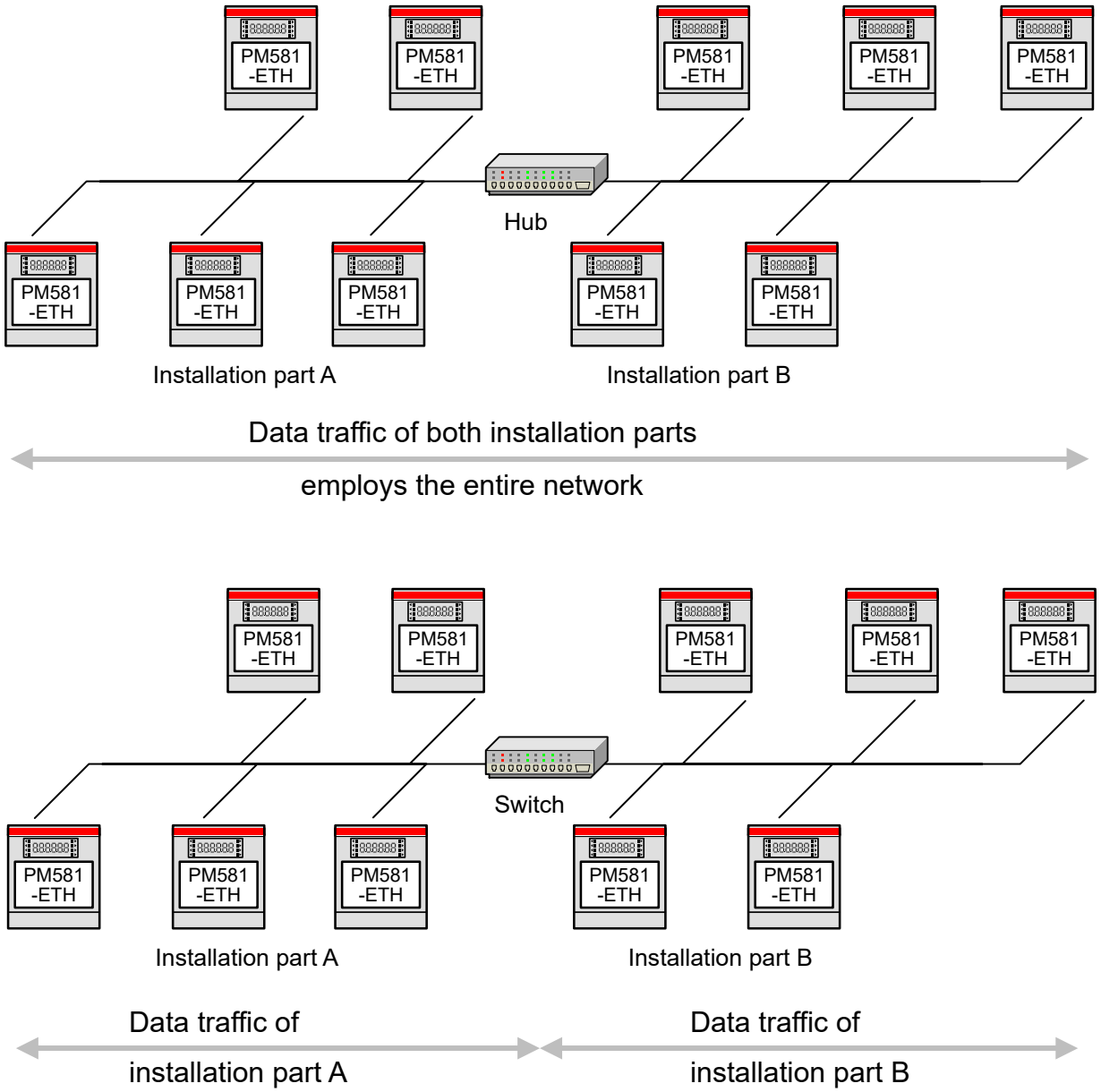


Fig. 1122: Use of hubs and switches

To enable crosswise traffic between the segments, a switch has to be able to temporarily store the incoming data packets until they can be transmitted on the forwarding segment. The decision about forwarding of data packets is done using address tables. These address tables are generated by the switch itself during a self-learning process. During this process, the switch remembers the source addresses (MAC addresses) of incoming data packets of a port. If it later receives further data packets, the switch compares their target addresses with the entries in the address tables of the ports and, in case of a match, forwards the respective package via the corresponding port. Here, the following cases have to be distinguished:

- If the source station and the target station are located within the same segment, the data packet is not forwarded.
- If the station of the target address is located in another segment than the source station, the data packet is forwarded to the target segment.
- Data packets containing a multicast or a broadcast address as the target address are forwarded via all ports.
- A data packet with a target address which is not contained in the address tables is forwarded via all ports (Frame Flooding).

The latter case normally only occurs during the first time after starting a switch since the address is usually entered after some time when exchanging a data packet.

In order to limit the size of the address tables, addresses which are not used over a longer period of time are additionally removed from the tables. This also avoids incorrect forwarding as it would appear e.g. when a station is moved within the network.

To enable the building of a redundant network structure (as it is often found in more complex networks) using switches, the so-called spanning tree method has been introduced. With this method, the switches exchange configuration messages among themselves. This way the optimum route for forwarding data packets is determined and the creation of endless loops is avoided. The exchange of messages is performed cyclic. As a result a connection breakdown is detected and forwarding is automatically changed to another route.

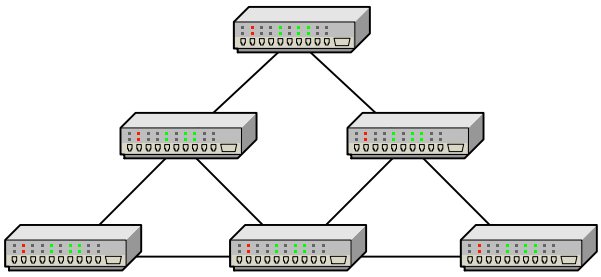


Fig. 1123: Redundant network structure using only switches

Using a switch instead of a hub increases the bandwidth of the individual segments and therefore leads to an increased performance. Building a network consistently with switches furthermore enables full duplex operation and thus simultaneous data traffic in both directions since switches are able to establish dedicated peer-to-peer connections between the individual ports. The use of the access method CSMA/CD is not required since collisions can no longer occur. Depending on the network structure, this can further increase the performance drastically. For full duplex connections furthermore no length restrictions of the collision domain have to be observed.

Media converters

Media converters provide the possibility of connecting components to each other via different media. The most frequently occurring case for this is the conversion between twisted pair (TP) and fibre optic cabling.

When using media converters it has to be observed that a connected port operating in half duplex mode is no collision domain border. This is often not considered when using optical fibres to bridge a larger distance. The fibre optical port of a media converter furthermore does not support the auto negotiation function. Due to this, if an Ethernet component is directly connected to the fibre optic side of a media converter, the transmission mode has to be set fixed according to the component connected on the twisted pair side. If a connection between two twisted pair components is established using two media converters, it is absolutely required that both twisted pair components are operating with the same transmission mode. If necessary, manual setting has to be performed.

Routers

Routers connect networks with identical protocols or addressing mechanisms. The main task of a router is to perform the routing for the transmission of data packets from the sender to the recipient. Routers are able to effectively reduce the data traffic between individual networks by using different algorithms. The dynamic routing leads to a load reduction for the entire network. If the router has several alternative routes to the target station available, it will always choose the optimum way depending on the current load on the network and the expected costs.

In contrast to the switches which forward the packets on the basis of layer 2 (e.g. Ethernet), the routers operate on layer 3 (e.g. IP). While the switches forward the packets on the basis of the MAC addresses, the routers evaluate the contained IP addresses. For this purpose, when receiving a data packet a router first has to remove the outer telegram frame in order to be able to interpret the addresses of the inner protocol and then it has to re-assemble the data packet again before forwarding it. This results in higher latency periods (time of stay)

of the data within the router itself. The investigation of a data packet necessary for routing makes clear that a router has to be able to process all network protocols to be routed over this router. Due to the increasing spread of heterogeneous networks, today often routers are used which are able to support several network protocols (e.g. IP, IPX, DECnet, AppleTalk) instead of special IP routers. Such routers which are able to process several network protocols are called multi-protocol routers. Some routers additionally have a bridge functionality (bridge routers, Brouters) which enables them to also forward the data packets of protocols a router cannot interpret or which do not support the routing function (e.g. NetBios).

Gateways

A gateway is a computer which is able to couple completely different networks. Gateways are operating on a layer above layer 3 of the ISO/OSI model. They are used to convert different protocols to each other. For the connected subnetworks, a gateway is a directly addressable computer (node) with the following tasks:

- Address and format transformation
- Conversions
- Flow control
- Necessary adaptations of transmission rates for the transition to the other subnetworks.

Gateways can furthermore be used to implement safety functions on the application layer (firewalls). For example, gateways are used for the coupling of PCs located in local area networks (LAN) to public long distance communications (wide area networks, WAN).

Programming access via Ethernet

Programming via Ethernet on TCP/IP in Automation Builder is described in the configuration chapter *Chapter 1.6.5.2.2.2.3 “Configuration of communication via Ethernet (TCP/IP)” on page 5829*.

Modbus on TCP/IP

Usage and configuration of Modbus on TCP/IP in Automation Builder is described in the configuration chapter *Chapter 1.6.5.3.3.1 “Modbus on TCP/IP protocol” on page 6173*.

Fast data communication via UDP/IP

Usage and configuration of UDP/IP in Automation Builder is described in the configuration chapter *Chapter 1.6.5.3.7.1 “Contents of the UDP protocol configuration” on page 6185*.

1.6.4.2.2 PROFIBUS DP communication modules

Introduction

This chapter gives a description of the PROFIBUS functionality in the AC500 system . The chapter contains of a general PROFIBUS overview, the specific handling in the AC500 system and describes the available ABB AC500 devices for PROFIBUS.

Topics and subtopics:

General PROFIBUS Description *Chapter 1.6.4.2.2.2 “PROFIBUS overview” on page 5525*

AC500 devices

- Communication Modules CM582-DP PROFIBUS DP Slave and CM592-DP-PROFIBUS DP Master (*Chapter 1.6.2.4.8 “PROFIBUS” on page 4075*).
- Communication Interface Modules (S500) CI541-DP, CI542-DP (*Chapter 1.6.2.8.6 “PROFIBUS” on page 4930*).

Configuration in Automation Builder

- *Chapter 1.6.5.2.6.2.1.2 “Configuration of a PROFIBUS DP master” on page 5883*
- *Chapter 1.6.5.2.6.2.2.1 “Configuration of PROFIBUS DP slave” on page 5894*
- Configuration of PROFIBUS-Devices
  - Configuration of CI541-DP and CI542-DP
  - Configuration of the *Chapter 1.6.5.2.6.2.1.4 “Configuration of the PROFIBUS DP slaves connected via FBP” on page 5889*
  - GSD based device (see *Chapter 1.6.5.2.6.2.1.3 “Configuration of 3rd party PROFIBUS DP slaves” on page 5887*).
- Online Diagnosis
  - CM592-DP PROFIBUS DP Communication Module Statistic Views
  - Fieldbus Commissioning (see *Chapter 1.7.2.7.1 “Fieldbus commissioning” on page 6392*)

IEC library

(see *Chapter 1.5.4.26.1 “Function blocks” on page 1750*)

- DPM_CTRL
- DPM_READ_INPUT
- DPM_READ_OUTPUT
- DPM_SET_PRM
- DPM_SLV_DIAG
- DPM_STAT
- DPM_SYS_DIAG
- DPV1_MSAC1_READ
- DPV1_MSAC1_WRITE

PROFIBUS overview

PROFIBUS basics

PROFIBUS DP is intended for fast data exchange in the field area. Here, central control units (e.g. PLC/PC) communicate with decentralized field devices like I/O, drives and valves via a fast serial connection. The data exchange with the decentralized modules is mainly performed cyclically.

The communication functions, required for data exchange, are defined by the PROFIBUS DP basic functions in accordance to IEC 61158/61784.

For parameterization, diagnosis and alarm handling during the running cyclic data exchange also non-cyclic communication functions are necessary for intelligent field devices.

ISO/OSI model

The definition of PROFIBUS is based on the experience concerning data transmission collected during long years.

One base is the ISO/OSI model (Open Systems Interconnection Reference Model). It is an open layer model with 7 layers for the communication in information processing systems. The model describes uniformed procedures and rules for the exchange of data.

Fieldbus systems normally use only three of the 7 layers:

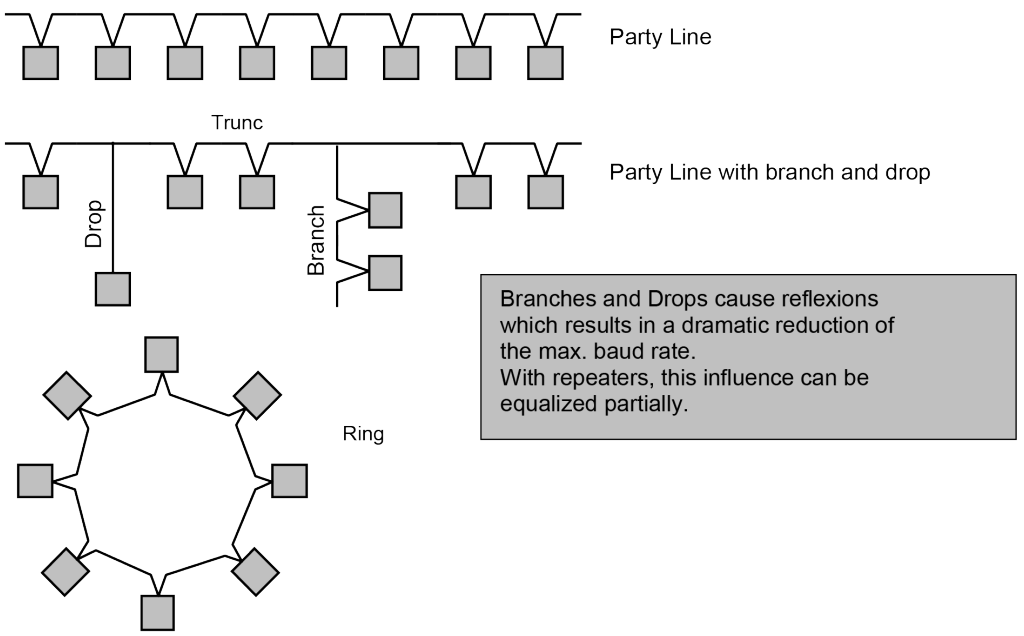
ISO/OSI	Transmitting CPU		Receiving CPU		
Layer 7	Application layer		Application layer	=	Interface to the application program (CPU) with application oriented commands (read, write)
...	...		...		
Layer 2	Data-link layer		Data-link layer	=	Access control (to the line), telegram (start, length,..), data security (e.g. CRC=Cyclic Redundancy Code)
Layer 1	Physical layer		Physical layer	=	Definition of the medium (Twinax, optical fiber, ..), coding ("1"=-4V), transmission speed (transmission rate)..
Transmission medium (physical)					

As a result of the ISO/OSI layer model, each layer can be defined separately and (nearly) independent of the other layers.

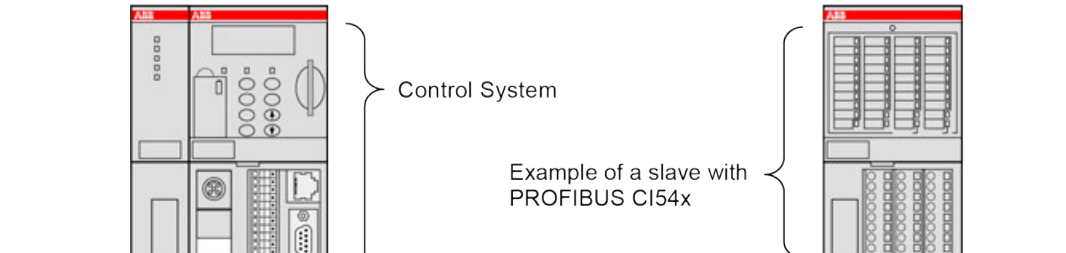
Indeed, it is possible and common to use conventional cables, but also optical fibers as physical layer for the PROFIBUS DP or have a mixture of both in a single bus configuration.

For the application layer, there are also different versions possible, e.g. PROFIBUS DP-V0, PROFIBUS DP-V1 but also others that are not regarded here.

Typical field bus topologies



Overview of transferred data



	Group	Type / example *	Direction	Comment
Cyclic data exchange	Monitored signals (inputs)	DI = digital input	read	
		AI = analog input	read	
	Commands (outputs)	DO=digital output	write	
		AO=analog output	write	
other basic data transfer	Diagnosis	Faults and warnings	read	DP/V0 and DP/V1
	Configuration, Identification	Qty. of DI, DO,...+ product code..	read / write	
	Bus specific data	baud rate, time-out..	read / write	
	Parameters**	Control function… trip class …	write	
Acyclic data exchange	Single parameters	Control function… trip class …	read / write	DP/V1 only

* The quantities of bytes/words are defined by the connected device.

** Parameters are transferred during power-up.

PROFIBUS DP-V0 <---> PROFIBUS DP-V1

Commands and monitoring signals

The transfer of commands and monitoring signals is the essential task of the field bus and the connected units.

They control and inform mainly about the process, e.g. control digital/analog outputs or read digital/analog inputs. They are the same for DP-V0 and DP-V1

Command and monitoring telegrams represent the cyclic data transfer.

Diagnosis

The diagnosis telegram provides detailed information if there is any problem, particularly in the process. A short circuit on a digital output is an example. Diagnosis data are automatically read by the PROFIBUS DP master if it gets a general fault info within a monitoring telegram.

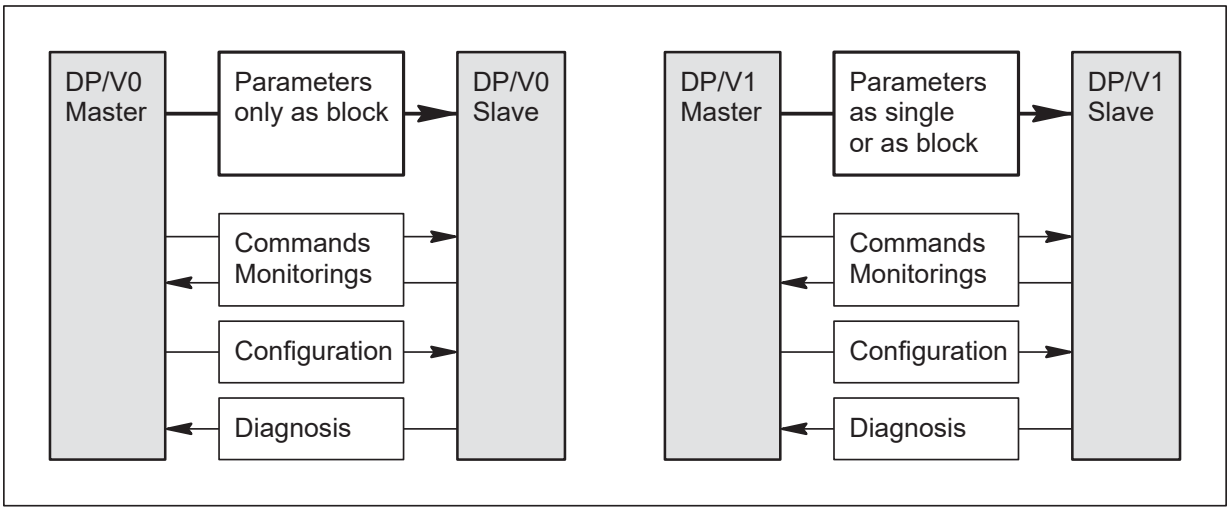
Parameters

Parameters are necessary to adapt the device to the process.

E.g., for the Digital S500 I/O modules the parameter "Input Delay" can be configured to filter contact switch bouncing

Parameters can also include service-oriented data such as "Operation hours".

The main difference between the PROFIBUS DP versions DP-V0 and DP-V1 is:



DP-V0 only allows to write the complete parameter set in one block.

The bus master sends the parameter block to the slave during power-up of the slave/device. Some control systems also allow to send the parameter block during normal operation.

DP-V1 offers reading and writing single parameters.

The possibility to read single parameters is an important advantage: E.g. to read single parameters of a drive (configuration parameters). In AC500 the function block DPV1_MSCAC1_READ can be used.

In all cases only the bus master can start the data exchange on the PROFIBUS DP bus.

PROFIBUS DP master class 1, PROFIBUS DP master class 2

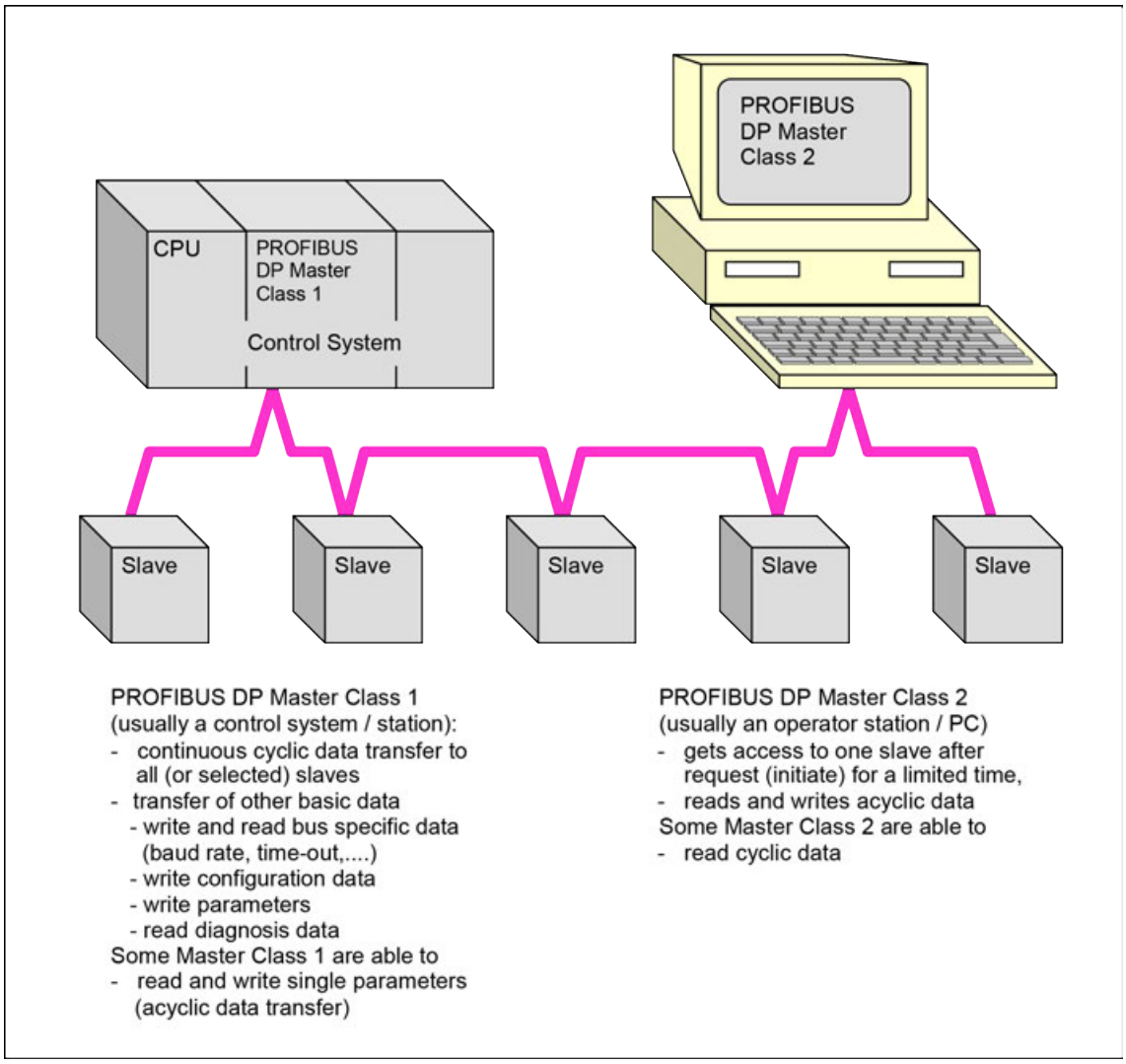


Fig. 1124: PROFIBUS DP master, class 1 and class 2

Features

Transmission technique

- RS-485, potential separated.
- Twisted pair cable or optical fibre as a medium for the bus.
- Up to 32 stations (master and slave modules) without repeaters and up to 126 stations on one bus with repeaters.
- Integrated repeater controller.

Communication

- Up to 244 bytes of input data and 244 bytes of output data per slave.
- Cyclic user data transfer between DP master and DP slave.
- Acyclic data transfer from master to master.
- Slave configuration check.
- Efficient diagnosis functions, 3 graduated diagnosis messaging levels.
- Synchronization of inputs and/or outputs via control commands.

Protection functions

- Message transfer with Hamming distance HD = 4.
- Errors during data transfer are detected by the CRC check and cause a repetition of the telegram.
- Access protection for inputs and outputs of the slaves.

- Incorrect parameter settings are avoided since bus stations with faulty parameters are not included in the user data operation.
- A failure of a bus station is registered in the master and indicated via a common diagnosis.

Standardization: PROFIBUS DP IEC61158 / DIN EN series

Terms, definitions and abbreviations

PROFIBUS DP

PROFIBUS DP	Process Fieldbus - Decentral Periphery
DPM1	DP master (class 1), normal bus master
DPM2	DP master (class 2), commissioning device
DPS	DP slave, I/O module
GSD	Modules master data
DPV1	Guideline for functional expansions of PROFIBUS DP
PNO	PROFIBUS Nutzer Organisation (PROFIBUS user organization)

Designing and planning a network

The PROFIBUS Communication Module is connected to the bus via the 9-pole SUB-D socket. For EMC suppression and protection against dangerous contact voltages, the shield of the bus line has to be connected to protective earth outside the housing.

Single master system

The single master system is the simplest version of a PROFIBUS network. It consists of a class 1 DP master and one or more DP slaves. Up to 31 DP slaves can be connected to the bus without using a repeater. If the number of bus segments is increased by means of repeaters, up to 126 DP slaves can be handled. The line ends of the bus segments have to be terminated using bus terminating resistors.

The DP master of class 1 is able to:

1. Parameterize DP slaves (e.g. timing supervision bus interchange). "bus interchange" means e.g. data and control Exchange on the Profibus.
2. Configure DP slaves (e.g. type / number of channels).
3. Read input and output data of the DP slaves.
4. Write output data of the DP slaves.
5. Read diagnosis data of the DP slaves.
6. Send control commands to the DP slaves (e.g. freezing input signals).

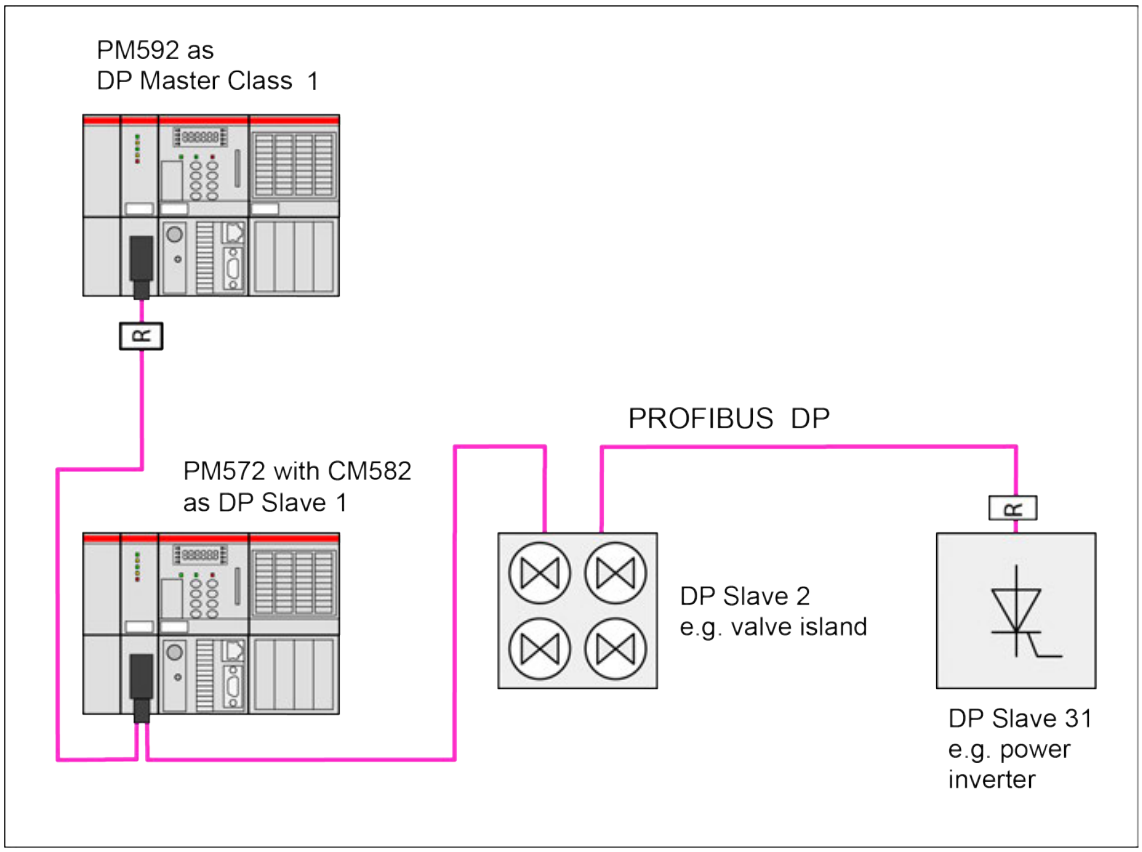


Fig. 1125: Single master system example

Multi master system

A PROFIBUS network containing several DP masters is called a multi-master system. Up to 32 stations (DP masters and DP slaves) can be operated on one bus segment. Using repeaters the system can be expanded to up to 126 stations. In a multi-master system no data exchange between the DP masters is performed. The entire system is divided into logical subsystems inside of which one DP master communicates with the assigned DP slaves. Each DP slave can be assigned to only one DP master. The master has unlimited access to its assigned slaves while all other masters on the bus can only read the input and output data of these slaves.

All DP masters of class 1 (normal bus master, here: AC500) and class 2 (commissioning device, typically a PC) can read the input and output data of all slaves.

Additionally the DP masters of class 1 and class 2 have the following access possibilities to their assigned DP slaves. They are able to:

- Parameterize DP slaves (e.g. timing supervision, bus interchange).
- Configure DP slaves (e.g. type / number of channels).
- Write output data of the DP slaves.
- Read diagnosis data of the DP slaves.
- Send control commands to the DP slaves (e.g. freezing input signals).

A DP master of class 2 is additionally able to:

- Read and write configuration data of the class 1 DP masters.
- Read configuration data of the DP slaves.
- Read diagnosis data of the class 1 DP masters.
- Read out the diagnosis data of the DP slaves assigned to the respective DP master.

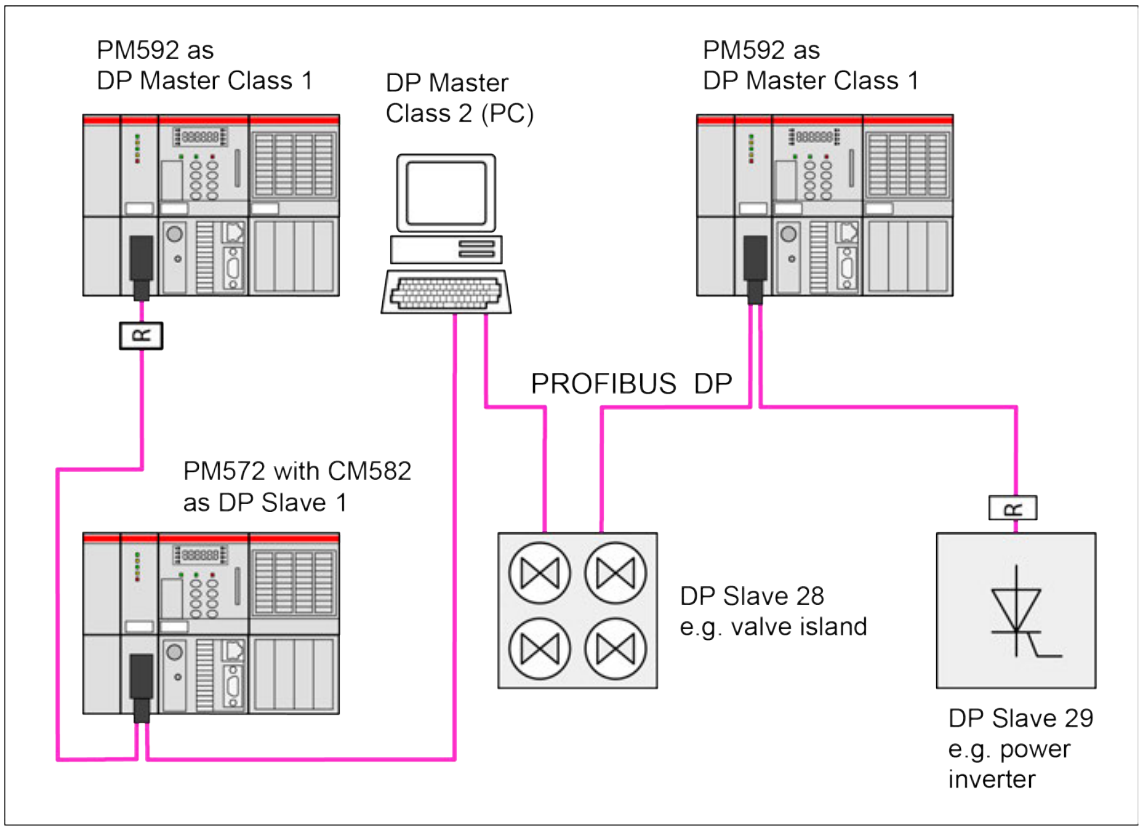


Fig. 1126: Multi master system example

PROFIBUS implementation

System start-up behavior

Initial Operation

The PROFIBUS DP protocol is automatically handled by the Communication Module and the operating system of the controller. The Communication Module is only active on the bus if it was correctly initialized before and if the user program is running. No connection elements are required for the cyclic exchange of process data via PROFIBUS DP. Special PROFIBUS DP functions can be realized using the function blocks of the corresponding PROFIBUS library.

Communication via PROFIBUS is established by the communication module when starting the user program and starts with the initialization of the configured slaves. After its successful initialization, the slave is added to the cyclic process data exchange. The "RDY" LED lights up steadily after at least one slave was successfully taken into operation. If the user program is stopped, the communication module shuts down the PROFIBUS system in a controlled manner.

The DP master operation mode is completely integrated to the operating system of the controller. The transmit or receive data of the slaves can be directly accessed in the corresponding operand areas. Access can be performed either via operands or symbolically. No function blocks are required.

The function block library contains various blocks which can be used e.g. to poll status information of the communication module or to execute specific acyclic PROFIBUS DP functions. If necessary, these blocks can be inserted additionally.

For further information on the configuration of PROFIBUS master/ PROFIBUS slave devices, see *Chapter 1.6.5.2.6.2 “PROFIBUS” on page 5883*.

Diagnosis

Error diagnosis

PROFIBUS DP communication errors are generally indicated by the red "ERR" LED of the Communication Module. Malfunctions of the PROFIBUS driver or the Communication Module itself are additionally indicated via the E error flags and the corresponding LEDs of the CPU.

Furthermore, the PROFIBUS library provides different function blocks that allow a detailed error diagnosis. Amongst other things, the following information can be polled:

- The condition of the Communication Module itself
- A detailed PROFIBUS diagnosis of an individual slave or
- A system diagnosis overview.

Function blocks

The function blocks listed in the table below are contained in the PROFIBUS DP Library
↳ *Chapter 1.5.4.26 “PROFIBUS DP library” on page 1750*.

Group	Function block	Function
General		
	PROFI_INFO	Reading of Communication Module information
Status / Diagnosis		
	↳ *Chapter 1.5.4.26.1.6 “DPM_STAT” on page 1775*	Reading the Communication Module status
	↳ *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*	Reading the detailed PROFIBUS diagnosis of a slave
	↳ *Chapter 1.5.4.26.1.7 “DPM_SYS_DIAG” on page 1781*	Reading the system diagnosis
Parameter		
	↳ *Chapter 1.5.4.26.1.4 “ DPM_SET_PRM” on page 1762*	Sending user parameters to a DP slave
Controller		
	↳ *Chapter 1.5.4.26.1.1 “DPM_CTRL” on page 1750*	Sending control commands to slaves
Acyclic reading		
	↳ *Chapter 1.5.4.26.1.2 “DPM_READ_INPUT” on page 1756*	Reading input data of slaves which are not assigned to the master
	↳ *Chapter 1.5.4.26.1.3 “DPM_READ_OUTPUT” on page 1759*	Reading output data of slaves which are not assigned to the master

1.6.4.2.3 PROFINET communication modules

PROFINET overview

PROFINET is an open standard for the realization of Industrial Ethernet based automation applications. Time-critical and non-time-critical applications could be realized inside the same Network in parallel. Additionally, the network is further on usable for standard office TCP/IP communication.

PROFINET is a cyclic communication. Only for parameterization, diagnosis and alarm-messages of intelligent field devices, acyclic messages will be used.

> *AC500 PROFINET RT is PROFINET IO based on PROFINETRT.*

Standardization: PROFINET

- IEC 61131-3: PLC Standardization
- IEC 61158: PROFINET Standardization
- IEC 11801: Wire and connection elements for Ethernet

Device certification

To ensure that PROFINET devices particularly of different vendors can communicate with each other without problems the PNO brought up a certification procedure. In this procedure the conformance in hardware as well as in software matters are tested and documented by an accredited test laboratory.

Fig. 1127: PROFINET logo

The certification of the AC500 PROFINET devices took place according to conformance class B and was executed by the accredited laboratory.

Furthermore the AC500 PROFINET devices have CE declaration.

Terms, definitions and abbreviations

Table 704: PROFINET

CAT 5	Ethernet cable for 100 MHz data size
CAT 5e	Ethernet cable for 100 MHz data size for Gigabit-Ethernet
CAT 6	Ethernet cable for 250 MHz data size
CAT 7	Ethernet cable for 600 MHz data size
CAT 8	Ethernet cable for 1 GBit data size
GSDML	Device data sheet in XML
I/O controller	Conform to a master in a PROFINET network

I/O module	Conform to a slave in a PROFINET network
I/O supervisor	PC-based engineering tool for initial operating or diagnosis
IRT	Isochronous real time
NRT	Non-Real-Time
PNO	PROFIBUS Nutzerorganisation e. V.
PROFINET	Process fieldbus network
PROFINET CBA	PROFINET Component Based Automation for complex assembly modules
PROFINET IO	PROFINET for decentralized field bus devices, motion control, etc.
RT	Real-Time
VLAN-Tag	Virtual Local Area Network Tag, advanced telegram for Ethernet telegram priority

PROFINET designations

PROFINET CBA / PROFINET IO

PROFINET contains 2 different solutions:

- PROFINET IO
- PROFINET CBA (Component Based Automation)

PROFINET IO is used for communication with decentral periphery like IOs, drives, etc.

PROFINET CBA is a communication solution for autonomously acting partial units of machines or plants.

PROFINET IO and PROFINET CBA are using different kind of communications variations of PROFINET:

- PROFINET NRT (non-real-time)
- PROFINET RT (real-time)
- PROFINET IRT (isochronous real time)

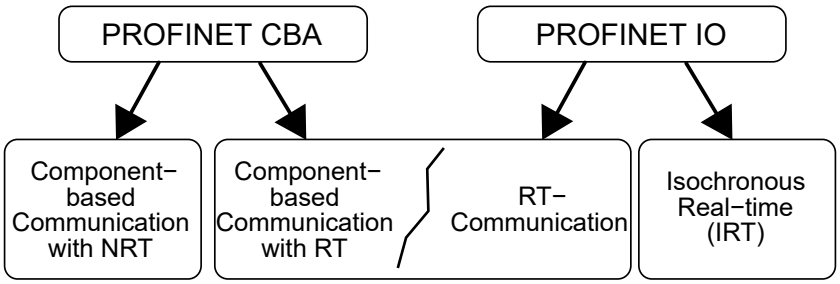


Fig. 1128: PROFINET CBA, PROFINET IO

PROFINET NRT PROFINET NRT is used by PROFINET CBA and PROFINET IO for the exchange of non-critical data by using standard TCP/IP and UDP/IP transfer mechanism according to IEE 802.3.

PROFINET RT PROFINET IO on RT is the real-time communication for time-critical process data between PLC and decentral periphery like I/O-devices, drives etc.

To reach the real-time behavior, PROFINET RT does not use the addressing features and control mechanisms of TCP and UDP. PROFINET RT is using a separate real-time channel for transmission with a reduced overhead. If used for communication between different networks, the "RT over UDP" mode is available.

PROFINET IO based on RT can be used in separate and existing networks. The PROFINET communication runs in parallel to TCP/IP and TCP/UDP communication.

PROFINET IRT PROFINET IRT is used for extremely time-critical applications (e. g. motion control). It uses hardware-aided real-time communication. To use PROFINET IRT, special hardware with cycle synchronization is necessary.

Real-time behavior means that a process has to be executed in a predefined time. The duration of this predefined time interval does not matter; it is only necessary that the process is done during this time. Due to this predictability of this process it is possible (with aid of additional synchronization methods) to cause chronologically adapted processes.

PROFINET nomenclature

I/O controller The I/O controller is the master in the PROFINET system. It coordinates the start of the bus communication and the parameterization of the I/O modules. The I/O controller gets the process data and the diagnostic alerts of the I/O modules and forwards them to the control system (for example PLC).

I/O module The I/O modules are the decentralized field devices (for example Input/Output devices or drives) in the PROFINET system.

The I/O modules get parameterized by the I/O controller (or I/O supervisor), and exchange process data with the I/O controller and send upcoming diagnostic alerts to the I/O controller.

I/O supervisor The I/O supervisor is the engineering tool to access the I/O modules temporarily for commissioning. This functionality can also be integrated into the I/O controller.

Transfer mechanism of PROFINET

The PROFINET IO communication is not a typical master-slave-communication. In fact there is a parameter in the controller that sets the communication basic cycle ("SendClockFactor"). For each I/O module it is possible to set a reduction rate ("ReductionRatio"). Due to this reduction rate the cycle time between I/O controller and each I/O module can be set separately in dependence of the performance requests *Chapter 1.6.4.2.3.4 “System performance” on page 5547*.

The I/O controller sends the output data to each planned I/O module (following the device specific parameterized cycle time). Each I/O module sends its input data with the same cycle time to the I/O controller.

Communication channels PROFINET IO is building up on standard Ethernet and uses two different communication channels:

For non-time-critical processes PROFINET IO uses standard Ethernet communication over UDP/IP (NRT communication). These non-time-critical processes are:

- "Start up" of the bus communication
- Allocation of the IP-addresses to the PROFINET IO devices (DHCP)
- Parameterization of the PROFINET IO devices

For time-critical data exchange PROFINET IO uses the RT communication channel. The time-critical processes are:

- Reading and writing of process data of the PROFINET IO devices
- Alarm messages of a certain PROFINET IO device
- Allocation of names and addresses to the PROFINET IO devices

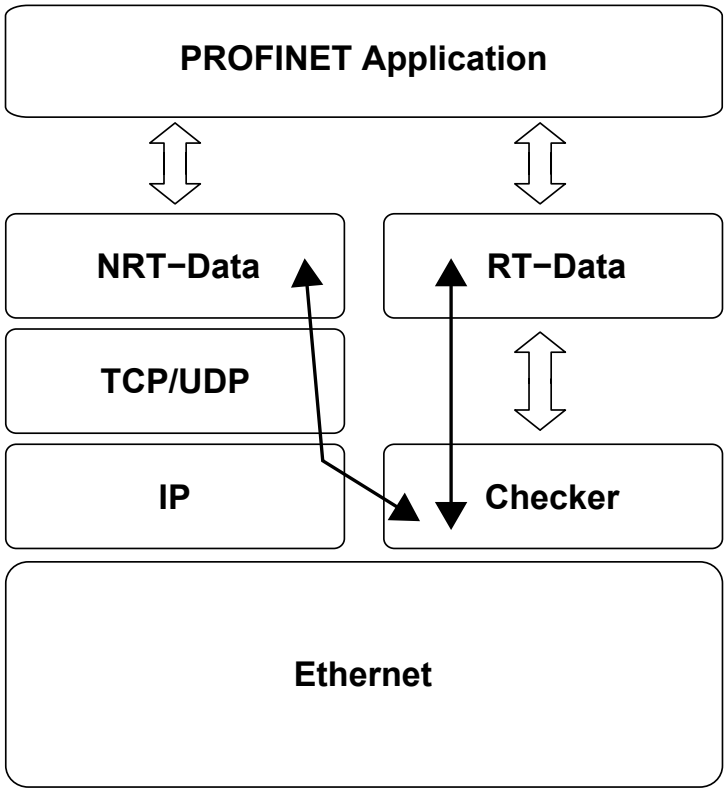


Fig. 1129: PROFINET NRT and RT

Frame structure The PROFINET RT frames consist of MAC header, process data and a checksum. The frames have the following structure:

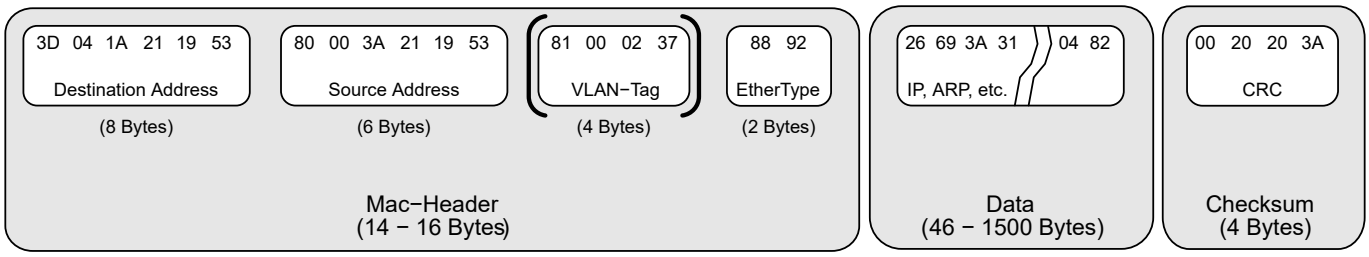


Fig. 1130: PROFINET RT telegram

The individual parts of the frame have the following sense:

- Destination address: MAC address of the receiver
- Source address: MAC address of the sender
- VLAN-Tag: prioritizing of the data exchange
- EtherType: Identifier of the following data (PROFINET RT uses 0x8892)
- Data: process data
- Checksum: checksum of the transferred data

The PROFINET NRT frames also contain IP/UDP, RPC NDR. In this case the VLAN tag is not used.

Prioritization of the data exchange with VLAN

For privileged forwarding of the PROFINET RT frames the real-time communication uses the VLAN tag according to IEEE802.1Q. So it is possible to define 7 different priorities. With the VLAN tag it is ensured that the PROFINET RT frames have the highest priority. Thereby "real-time" communication is made possible if PROFINET RT is used together with other network protocols (for example TCP/IP, basic network communication, internet).

The PROFINET RT communication is realizable with every commercial Ethernet controller that supports VLAN.

The VLAN tag has the following structure:

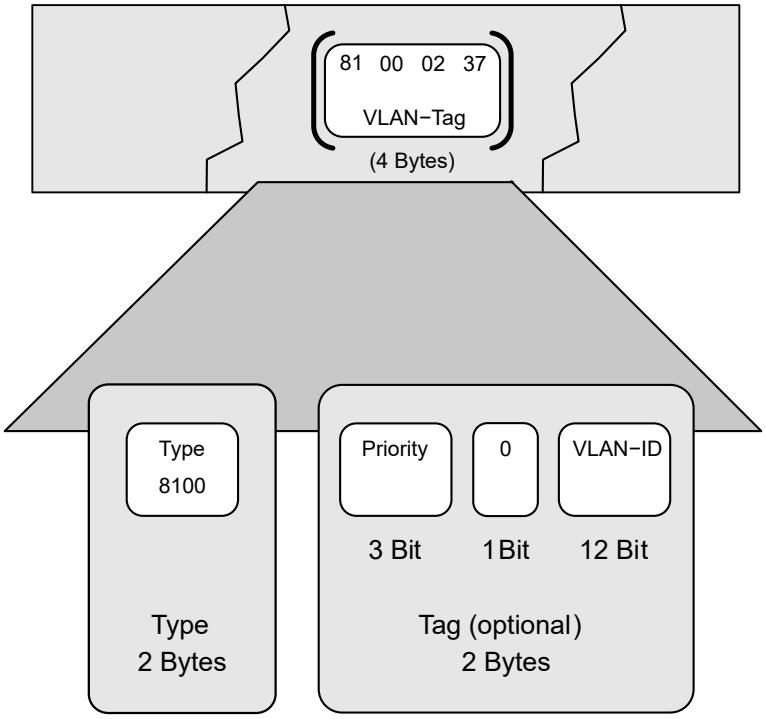


Fig. 1131: Contents of the VLAN-tag

The individual parts have the following sense:

- Type: Identifies the following data as a VLAN information
- Priority: Defines the priority of the telegram
- VLAN_ID: ID of the associated VLAN group

CAUTION!
Risk of communication faults!
PROFINET IO does not work in networks if Ethernet components do not support VLAN.

Make sure that all Ethernet components support the VLAN-tag.

Additional notices for the conformance classes

The MRP (Media Redundancy Protocol) is necessary for ring topology. It has to be integrated in the I/O controller as well as in the I/O modules and ensures the communication at a bus disconnect.

The I&M functions (Identification & Maintenance) are necessary for reading out the vendor and system specific information of an I/O module. The contents of these functions are defined in a specific PNO specification (Profile Guidelines Part 1: I&M fuctions, Rev. 1.1.1, order no. 3.502).

> *AC500 PROFINET RT is certified according to IEC 61158 and is conform to the conformance class CC A.*

Further information on PROFINET can be found on the website of the "PNO" following this link: *http://www.profibus.com*.

General station description file

The device specific features of a PROFINET IO device are documented in an electronic data sheet, the General Station Description File "GSDML". This XML-based GSD-file describes uniquely and completely the characteristics of a device in a vendor-independent format. For the CI50x-PNIO Devices these characteristics are:

- Description of the inputs and outputs of the I/O modules
- Parameters that are necessary for correct operation of the I/O modules
- Diagnostics and its meanings that can be delivered from an I/O module
- Description of the S500 expansion modules that can be used on the I/O bus

Due to the predefined file format the configuration of a PROFINET system is simplified. The GSDML files are normally supported by the vendors.

The PROFIBUS user organization (PNO) provides the GSDML files of several PROFINET devices in its product overview.

The address of the PNO is: *http://www.profibus.com*.

PROFINET conformance classes

PROFINET as an open standard that defines three conformance classes with raising functionality to provide multivendor interoperability and functionality compatibility.

Within a third party certification *Chapter 1.6.4.2.3.2 “PROFINET modules” on page 5540* is it ensured that a device provides the functionality of a certain conformance class. The conformance classes are divided in the categories CC A , CC B and CC C (see following figure).

The PROFINET certificate attests the norm compliance in a PROFINET network according to IEC 61158.

	- Switches with IEEE802-conformity with bus-, phase- and real-time clock synchronization - IRT communication - TCP/IP/RT and IRT redundancy	CC C

	- Switches with MRP support (conform with IEEE 802) - Providing of data for device-to-device-communication - Application of SNMP - Comfortable device-replacement without usage of an engineering tool - Client-functionality with MRP redundancy		CC B	
	- Conduction-bound and wireless data transfer - Cyclic RT communication - Acyclic TCP/IP communication - Alarms/diagnosis - Definition of transfer cycle - Automatic address resolution - Identification & maintenance functionality - Basic mechanism for detection of adjacent devices - Prioritization of data transfer according to IEEE standard - Transfer media: 100 Base TX/FX	CC A		
PROFINET conformance classes				

PROFINET modules

Communication modules and communication interface modules

The Communication Module ↬ *Chapter 1.6.2.4.9.1 “CM579-PNIO - PROFINET IO RT controller” on page 4084*acts as I/O controller in a PROFINET network. It is connected to the processor module via an internal communication bus.

The Communication Interface Modules CI50x-PNIO act as I/O modules for PROFINET network. The difference of those devices can be found in their input and output characteristics ↬ *Chapter 1.6.2.8.7.1 “Comparison of the CI5xx-PNIO modules” on page 4993*.

The communication modules ↬ *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089* enables an AC500 PLC to act as I/O module in a PROFINET network.

Device model of AC500 PROFINET IO devices

PROFINET standard defines modules and submodules to give structure to I/O modules data. These items are used in hierarchical order wherein a module may include one or more submodules. The input and output data of an I/O module are located inside these submodules. The modules and submodules can be identified via ident-numbers (module ident-number and submodule ident-number) and can be assigned to slots and subslots. Basically 32767 slot indexes and also 32767 subslot indexes are available to design the device structure.

PROFINET standard defines the following submodule types which represent the Device Access Point (DAP) to provide standard device functionality. In AC500 PROFINET IO devices the protocol stack defines to assign these special submodules at module slot 0.

Submodule type	Assigned subslot
DIM	1
Interface	32768
Port 1	32769
Port 2	32770

Automation Builder configuration assigns DIM, interface, port 1 and port 2 to desired slot/subslots. These modules are inserted automatically in hidden style so they are not visible to the user. It is only required to assign manually the modules/submodules needed for providing I/O data.

Module types provided by AC500 PROFINET IO devices support one single submodule only. This single submodule is inserted automatically in hidden style so it is not visible to the user. The available module types depend on the device type. See *Chapter 1.6.2.8.7.1 “Comparison of the CI5xx-PNIO modules” on page 4993* for CI50x module types and see *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089* for CM589-PNIO(-4) module types.

PROFINET standard defines the property API (Application Process Identifier) to define standardized behavior to I/O modules. In AC500 PROFINET IO devices support API 0 only. Automation Builder defines corresponding API setting automatically.

Allocation of the device name

General information

There are 2 possibilities for the allocation of the device name of the modules CI50x-PNIO and CM589-PNIO(-4):

- Allocation of the device name via DCP (Engineering Tool needed)
- Allocation of the device name via address switches (without Engineering Tool)

For the start-up of PROFINET, the address information "MAC address" and a unique "device name" is sufficient. The allocation of the IP address is performed via the I/O controller automatically during start-up of the bus communication.

CAUTION!
Malfunctions due to wrong device name settings!
Each device name can only be used once in a network to be explicit.

Make sure that each device has a unique device name.

A maximum of 256 PROFINET IO devices can be used within the same network. The ABB CM579-PNIO PROFINET IO Controller can handle up to 128 PROFINET IO devices.

Allocation of the device name via DCP

The allocation of the device name via DCP is standard for PROFINET networks. For this possibility of allocation, it is absolutely necessary to set both address switches to "00".

A device name set via DCP will be also present after a restart of the device (is stored permanently).

If the address switches are not set to "00", the device name via DCP is also stored permanently. But after a restart, the stored device name is not used.

Allocation of the device name via address switches

The AC500 PROFINET IO RT Devices (CI50x-PNIO and CM589-PNIO(-4)) are equipped with 2 rotary switches to set an explicit name to the I/O modules before commissioning. No engineering tool is needed.

The device gets its name (including the fixed part of the device name) directly from the setting of the switches (01h...FFh). This name can be used directly within the device configuration.

This name is for example:

ci501-pn-xx / cm589-pn-xx

ci501-pn- / cm589-pn- is the fixed part of the device name and "xx" represents the position of the rotary switch (0..255d or 0..FFh).

Designing and planning a network

With PROFINET IO it is also possible to include wireless parts using WLAN into the network. The only restriction is that the wireless component must support VLAN.

The reaction time will increase if using WLAN, because the wireless/WLAN transfer rate is slower than wired networking.

CAUTION!

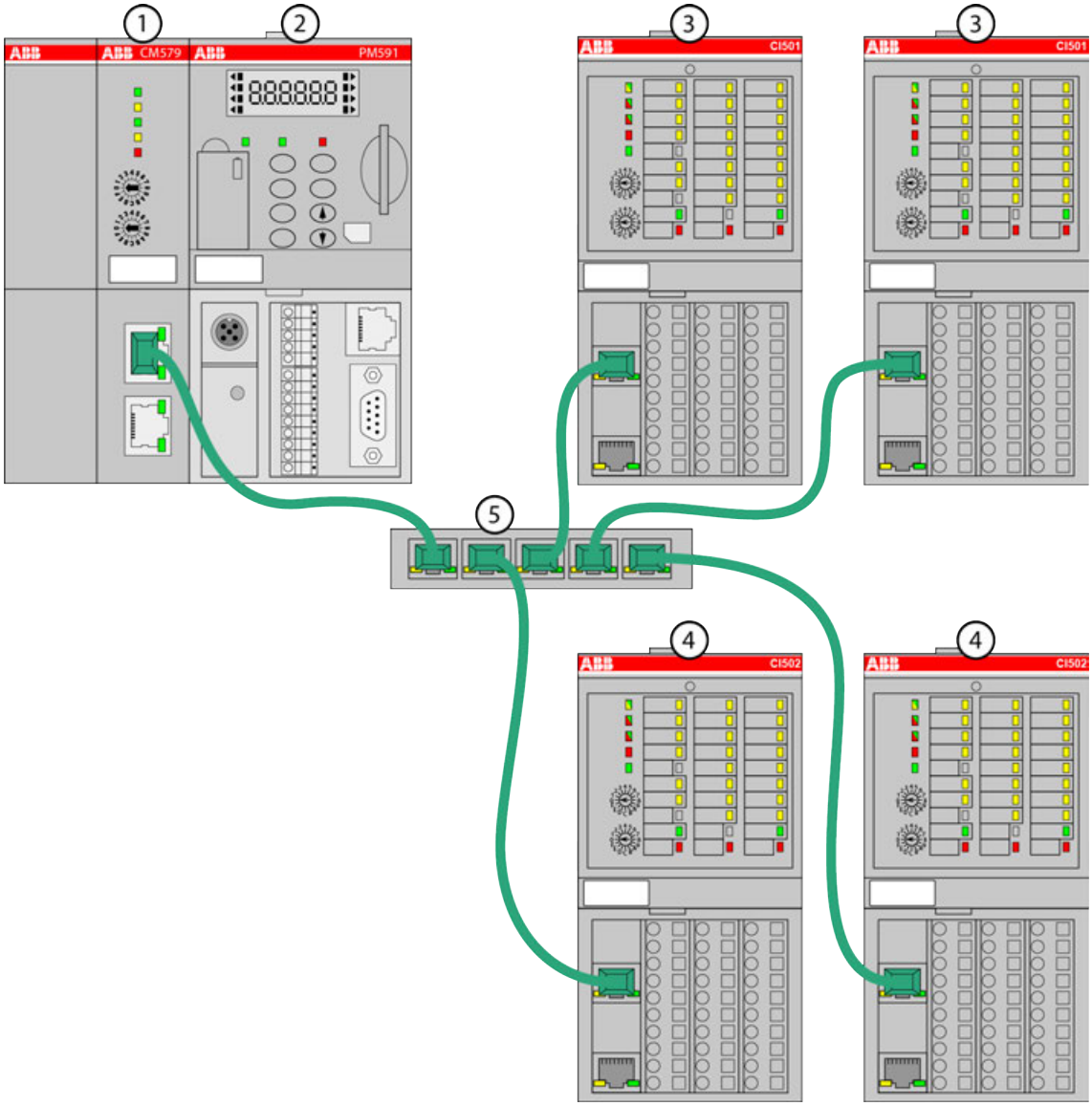
Risk of communication faults!

Ethernet components without VLAN-tag-support will not work properly in PROFINET IO networks. Make sure that all Ethernet-devices support the VLAN-tag.

Because of PROFINET IO requirements on data throughput, switches with support of 100 MBit/s and full duplex are mandatory.

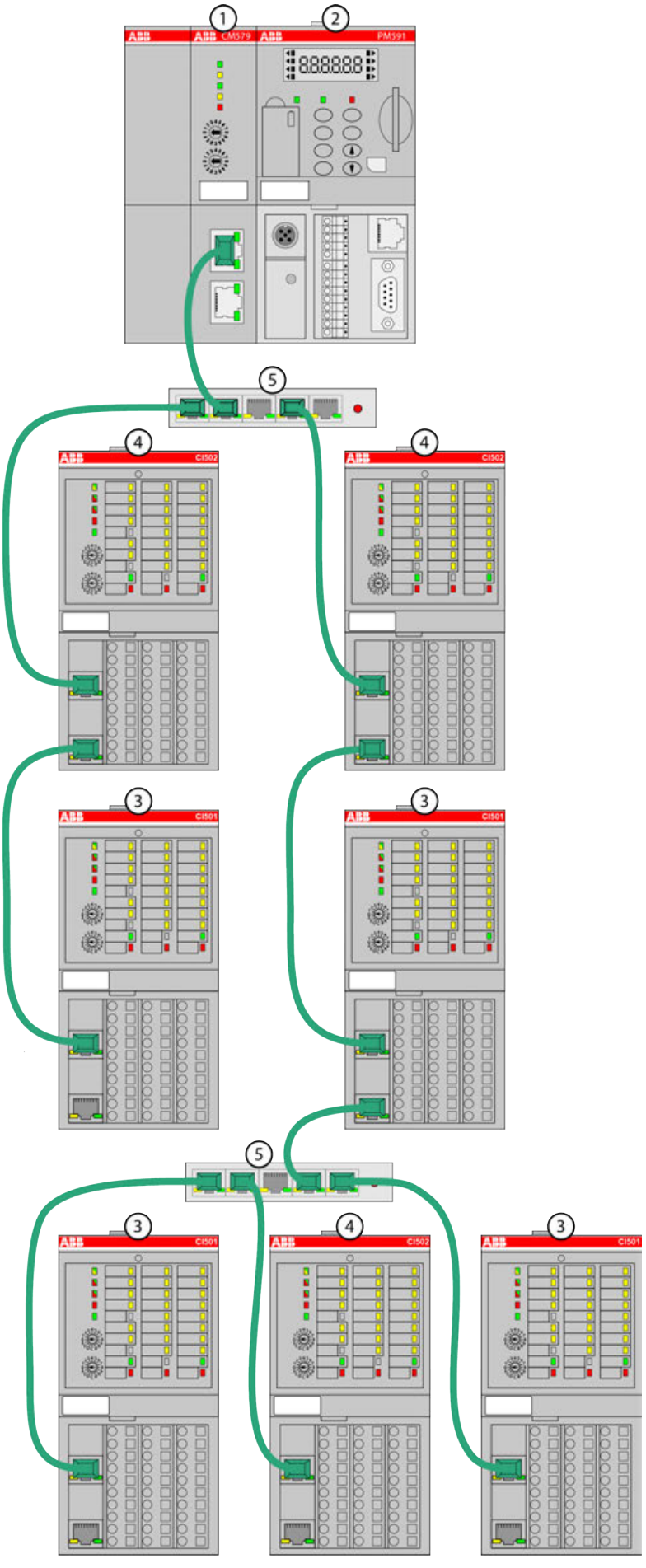
Topologies

Star



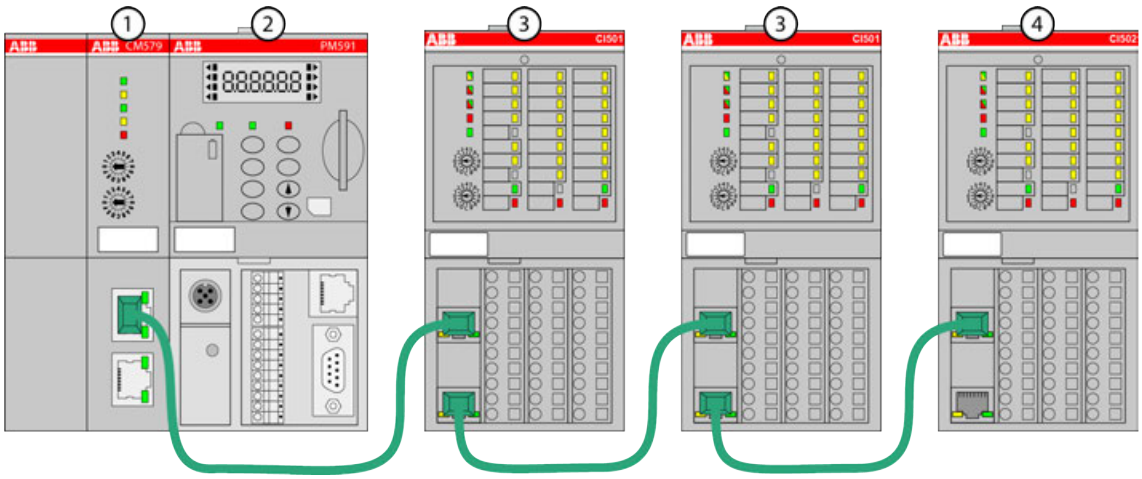
1 CM579
2 PM591
3 CI501
4 CI502
5 Switch

Tree


1
2
CM579
PM591
5
4
CI502
4
CI502
3
CI501
3
CI501
5
3
CI501
4
CI502
3
CI501

1 CM579
2 PM591
3 CI501
4 CI502
5 Switch

Bus (also called line)

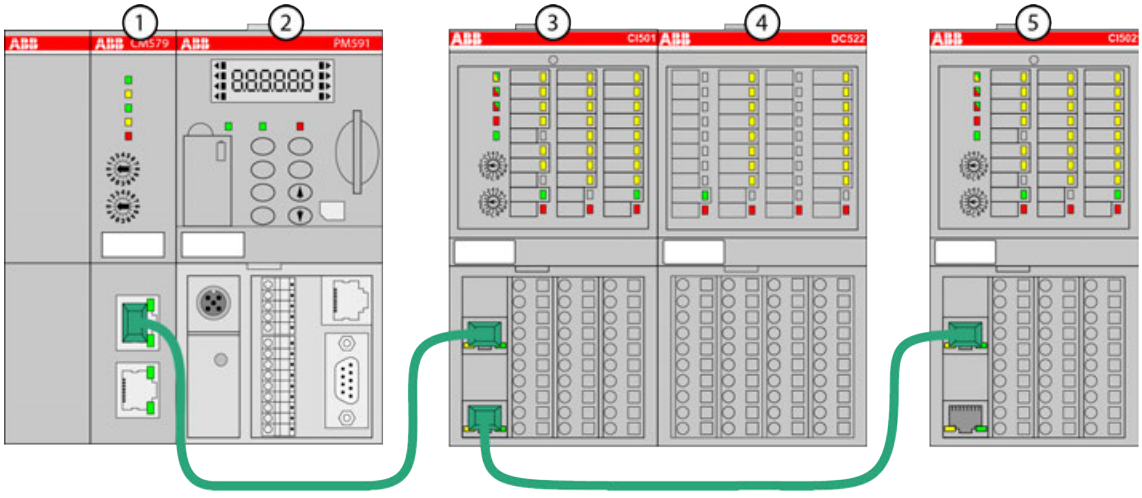


1 CM579
2 PM591
3 CI501
4 CI502

A ring structure in AC500 PROFINET is only allowed with the usage of special 3rd party switches supporting ring structure.

According to requirements, topologies can be combined and mixed.

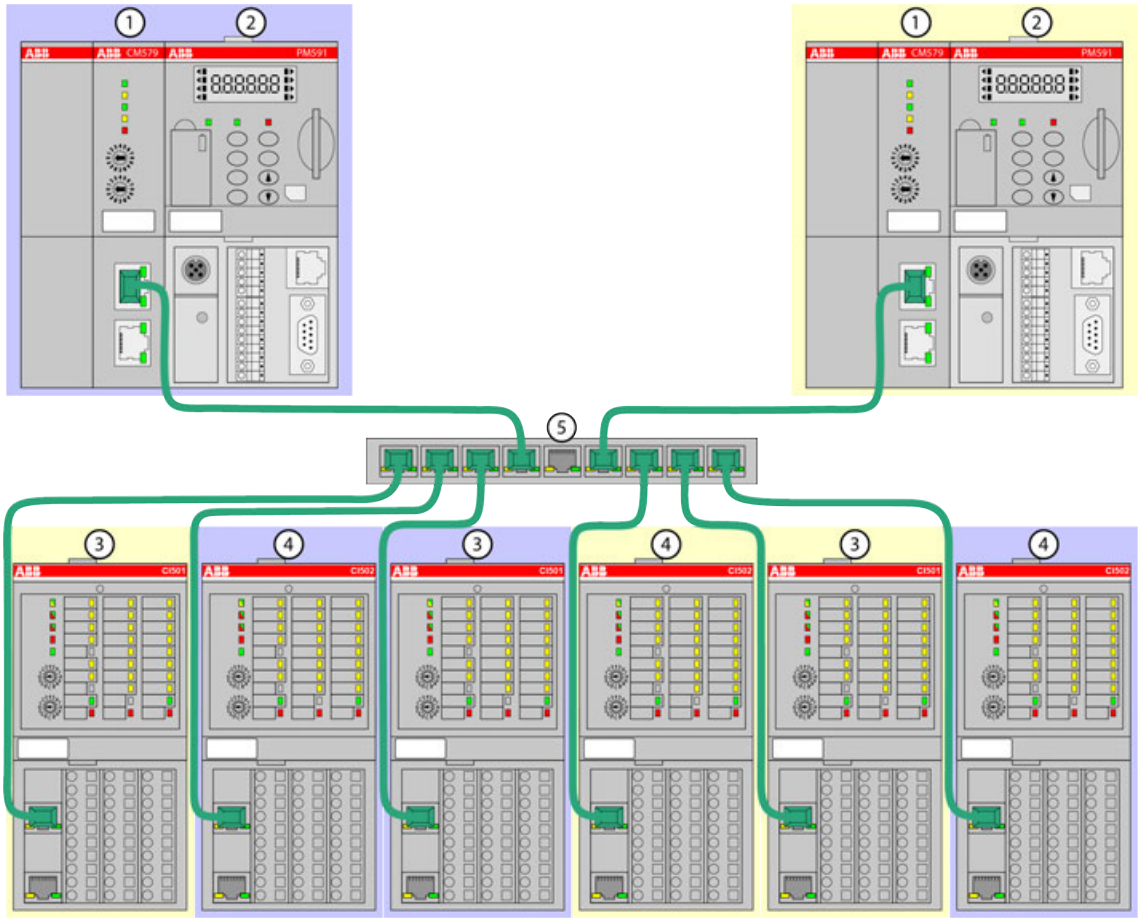
An easy controller-device-system consists of a controller CM579-PNIO and a certain quantity of CI50x-PNIO (if necessary expanded with S500 expansion modules). A typical constellation is described in the part "PROFINET configuration example".



1 CM579
2 PM591
3 CI501
4 DC522
5 CI502

Parallel operation of different PROFINET IO systems in one network

With PROFINET IO it is possible to create multiple controller-device-systems in one network. They are built from functional independent subsystems, each consisting of one controller and a part of the I/O modules in the network. The used topologies and layout of the I/O modules do not matter.



1 CM579
2 PM591
3 CI501
4 CI502
5 Switch

System performance

Bus communication

The PROFINET communication is realized using 100 MBit/s transfer rate. Real-time frames contain in addition to the process data (36 ... 1440 bytes) also an overhead of 44 bytes (gap, preamble and FCS included). The result is a frame length of 80 - 1484 bytes, that means a frame time of 6.4 µs up to 116.4 µs.

The modules CI50x-PNIO are the I/O modules for PROFINET RT. The difference of those devices can be found in their input and output characteristics ↬ *Chapter 1.6.2.8.7.1 “Comparison of the CI5xx-PNIO modules” on page 4993*.

- The PROFINET IO RT devices CI501-PNIO, CI502-PNIO and CI504-PNIO have a process data length of 80 bytes up to 213 bytes, depending on the usage of I/O modules. That results in a frame time of 6.4 µs up to 17.0 µs for 1 I/O module.
- The PROFINET IO RT device CI506-PNIO has a process data length of 80 bytes up to 1024 bytes, depending on the usage of I/O modules. That results in a frame time of 6.4 µs up to 81.7 µs for 1 I/O module.

To get the best performance for the PROFINET RT channel a few factors have to be taken into account. These parameters can be set in the configuration tool and define the frequency and distribution of the RT frames.

SendClockFactor

The SendClockFactor multiplied with a basic time of 31.25 µs defines the cyclic basic clock rate of the whole PROFINET system, i.e. the smallest cycle time for the exchange of process data. The typical value for the SendClockFactor is 32, that means the basic bus cycle is 1 ms. For 100 MBit/s it is recommended that the SendClockFactor is between 5 and 128.

Table 705: Examples for the SendClockFactor:

SendClockFactor	PROFINET basic cycle time	Annotation
5	156 µs	Fastest theoretical value for the basic cycle time at 100 MBit/s
32	1 ms	Fastest basic cycle time for the PROFINET IO controllers
128	4 ms	Slowest theoretical value for the basic cycle time at 100 MBit/s

The possible values for SendClockFactor and ReductionRatio (the product of both is the cycle time) of a CI506-PNIO device are dependent on the connected CANopen Slaves (number of PDOs. The dependency is shown in the following table.

For a CI506-PNIO for example:

PROFINET Cycle Time	Maximum number of CANopen PDOs *)
1 ms	8
2 ms	32
4 ms	128
16 ms	244

*) Remarks:

- 50 % RxPDO, 50 % TxPDO
- 8 byte process data for each PDO
- CANopen cycle time min. 4 ms or about 80 % CAN bus load (e.g. 20 ms for 128 PDOs)

ReductionRatio

This value can be set individually for each I/O module and defines its ReductionRatio to the basic cycle time of the PROFINET communication. That means if the basic cycle time is 1 ms and the ReductionRatio is set to 32, the process data will be exchanged every 32 ms between the I/O module and the communication module.

The ReductionRatio provides the user the possibility to adapt the network times of each slave individually and optimally to the system restrictions. With this it is possible to decrease the bus load for getting lower reaction times for more time critical data of other I/O modules.

Table 706: Examples for the ReductionRatio:

PROFINET basic cycle time	ReductionRatio of I/O module	Update time of the process data of an I/O module
1 ms	1	1 ms
1 ms	2	2 ms
1 ms	4	4 ms

Allowed values for the ReductionRatio are 1, 2, 4, 8, 16, 32, 128, 256 and 512.

For configuration of the ReductionRatio, attention should be paid to:

- the basic bus cycle time should not be overloaded, i.e. that not more RT frames can be transferred than would fit in the fastest bus cycle time.
- a time reserve for non RT frames (NRT) should be hold out (typically about 30 % of the projected basic bus cycle time)

Help on projecting

For getting an overview over the bus load, information is displayed in this section, with this information it is easier to configure a PROFINET IO system optimally.

The PROFINET IO frame has a minimal length of 80 bytes (44 bytes overhead and 36 bytes minimum user data). That means an extension of this frame size takes only place if the sum of the process data (respectively divided in input and output data) is greater than 36 Byte.

In the following figure this is pointed up again: The first 2 examples (CI501-PNIO alone and CI502-PNIO with 5 I/O modules DC532) will fit into the minimum frame size. In the third example, only two I/O modules AX522 are used on a CI501-PNIO but that suffices to get above the minimum frame size.

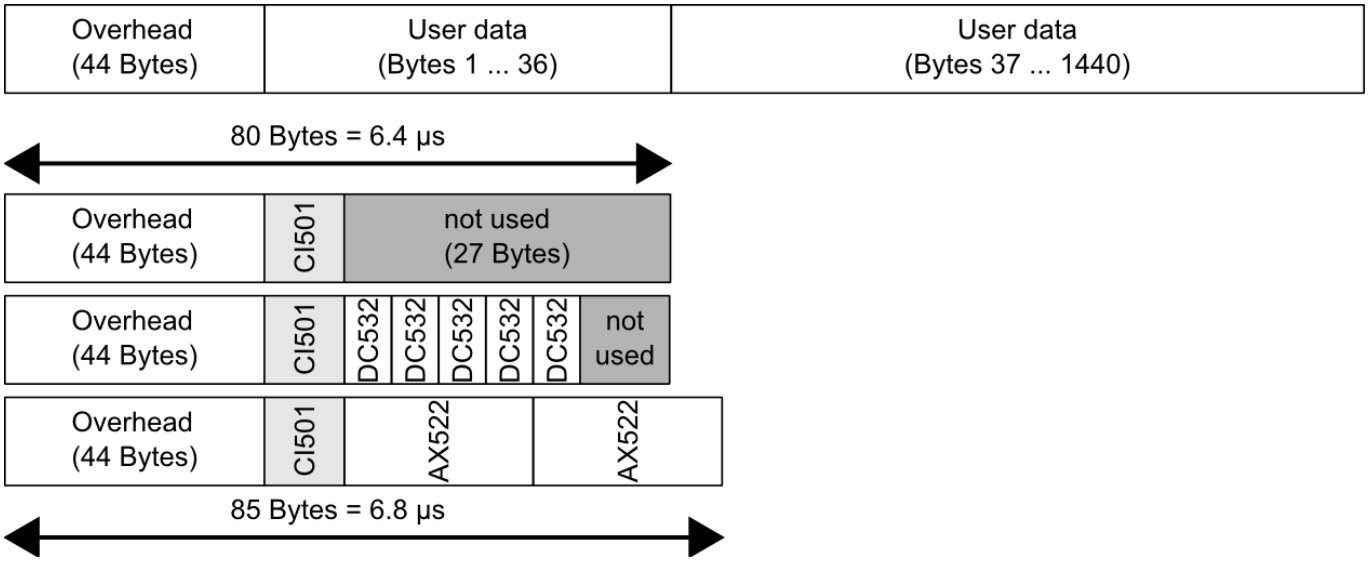


Fig. 1132: Effect of the process data length on the frame length

For better overview, the data length of the CRC, which follows the user data, is included into the overhead.

The following tables show the length of the process data of all available S500 PROFINET IO devices and of all S500 modules. With these tables and the aid of the following formula, it is possible to calculate the frame time for a modular I/O module (separately for input and output data).

Module type	Input data length [bytes]	Output data length [bytes]
CD522	24	32
CI501-PNIO	9	5
CI502-PNIO	2	2
CI504-PNIO	0 ... 36	0 ... 36
CI506-PNIO	0 ... 36	0 ... 36

Table 707: Length of the process data of the S500 I/O modules:

Module type	Input data length [bytes]	Output data length [bytes]
DA501	12	8
DC522	2	2
DC523	3	3
DC532	4	2
DC561	2	2
DI524	4	0
DI561	1	0
DI562	2	0
DI571	1	0
DO561	0	1
DO571	0	1
DO572	0	1
DX522	1	1
DX531	1	1
DX561	1	1
DX571	1	1
AI523	32	0
AI531	32	1
AI561	8	0
AI562	4	0
AI563	8	0
AO523	0	32
AO561	0	4
AX521	8	8
AX522	16	16
AX561	8	4

t_{input} = Max(6.4 µs, (SUM_{input} + 44) * 0.08 µs) / ReductionRatio

t_{output} = Max(6.4 µs, (SUM_{output} + 44) * 0.08 µs) / ReductionRatio

$t_{sum} = t_{input} + t_{output}$

For CI506-PNIO the I/O data size of connected CANopen slaves must also be considered.

With these values, an estimation of the expected bus load is possible. Afterwards, all frame times of input and output must be added. The result is the average time the complete communication needs. In the following table this is done for an example application for getting a sense of the expected times. The sum of the frame times of the different I/O modules is here 130.3 µs, that means with a basic cycle time of 1 ms the bus load is only 13.26 % and that means there is enough reserve for NRT data transfer for other network applications.

Device	DAP	EXP1	EXP2	EXP3	EXP4	EXP5	EXP6	EXP7	Send Clock Factor [ms]	Reduction Ratio	Time [µs]
1	CI501-PNIO	DC532	AX521						1	1	12.8
2	CI501-PNIO	DC532	AX521						1	2	6.4
3	CI501-PNIO	DC532	AX521						1	4	3.2
4	CI501-PNIO	DC532	AX521						1	8	1.6
5	CI501-PNIO	DC532	DC532	DC532	DC532				1	1	12.8
6	CI501-PNIO	DC532	DC532	DC532	DC532				1	2	6.4
7	CI501-PNIO	DC532	DC532	DC532	DC532				1	4	3.2
8	CI501-PNIO	DC532	DC532	DC532	DC532				1	8	1.6
9	CI501-PNIO	DC532	DC532	DC532	DC532	AX522	AX522	AX522	1	1	17.8
10	CI501-PNIO	DC532	DC532	DC532	DC532	AX522	AX522	AX522	1	2	8.9
11	CI501-PNIO	DC532	DC532	DC532	DC532	AX522	AX522	AX522	1	4	4.4
12	CI501-PNIO	DC532	DC532	DC532	DC532	AX522	AX522	AX522	1	8	2.2
13	CI501-PNIO	AX522	AX522	AX522	AX522	AX522	AX522	AX522	1	1	26.1
14	CI501-PNIO	AX522	AX522	AX522	AX522	AX522	AX522	AX522	1	2	13.0
15	CI501-PNIO	AX522	AX522	AX522	AX522	AX522	AX522	AX522	1	4	6.6

Device	DAP	EXP1	EXP2	EXP3	EXP4	EXP5	EXP6	EXP7	Send Clock Factor [ms]	Reduction Ratio	Time [µs]
16	CI501-PNIO	AX522	AX522	AX522	AX522	AX522	AX522	AX522	1	8	3.3
										TOTAL	130.3

Configuration examples and their delivery time

The following examples give an estimated overview of the PROFINET communication usage rate.

Example 1 (typical)

A typical entire module consists of a CI501-PNIO with one DC532 and one AX521. This constellation contains 21 bytes input data and 15 bytes output data. The RT telegram is sent in both directions with a time of 6.4 µs.

Bus cycle time = 1 ms

ReductionRatio (RR) = 1

RT transmission time = 1 ms -30 % = 700 µs

Number of I/O devices = 700 µs / (6.4 µs + 6.4 µs) = 55

You can operate with up to 55 I/O devices in one network and you still have enough reserves for the non-real-time channel.

The reserve of the real-time and non-real-time channel is decreased by using more than 55 I/O devices. (With 78 I/O devices, there are no reserves left.)

Example 2 (with 7 I/O modules)

An I/O module with a larger data range is a CI501-PNIO with seven AI523. This constellation contains 233 bytes input data and 5 bytes output data. In input direction a RT telegram is sent with about 24 µs.

Bus cycle time = 1 ms

ReductionRatio (RR) = 1

RT transmission time = 1 ms - 30 % = 700 µs

Number of I/O modules = 700 µs / (24 µs + 6.4 µs) = 24

You can operate with up to 24 I/O modules with the same configuration.

If more than 32 I/O modules are operating, the data exchange must be decreased by changing the ReductionRatio.

Ethernet TCP/IP Simultaneous operation of PROFINET and Ethernet TCP/IP in one network is possible. The parallel usage of non-real-time protocols (office applications, TCP/IP) has no influence to the real-time channel. On the other hand, a high usage rate of the real-time channel delays the non-real-time communication.

Simultaneous usage of PROFINET and EtherCAT

The simultaneous operation of PROFINET and EtherCAT in one network is not possible.

Simultaneous programming via network

The PROFINET Communication Module has no programming interface to the AC500 PLC. The PROFINET network is usable for programming purposes by connecting a CM597-ETH Communication Module or a PM5x1 Ethernet port to the PROFINET network.

Delivery time of CI501-PNIO and CI502-PNIO

The PROFINET Communication Module CM579-PNIO supports a communication with up to 128 I/O modules. The aimed delivery times raise by a greater upgrade of the system.

The following tables show the different delivery times in dependence of a CM579-PNIO Communication Module and the number of CI50x-PNIO modules.

The basic PROFINET cycle is 1 ms (SendClockFactor = 32). The I/O modules ReductionRatio value is 1 and the process update cycle is 1 ms.

Quantity of CI50x-PNIO	Terminal-to-terminal response time of digital IO [ms]	Comment
2	7.1	The information regards the "Worst Case"
16	7.1	
32	13.1	
Terminal-to-terminal response time of digital I/O depending on the quantity of I/O modules in the PROFINET network		

Quantity of CI50x-PNIO	Terminal-to-terminal response time of analog I/O [ms]	Comment
2	14.0	The information regards the "Worst Case"
16	14.0	
32	18.0	
Terminal-to-terminal response time of analog I/O depending on the quantity of I/O modules in the PROFINET network		

PROFINET implementation

System start-up behavior

Initial operation

The PROFINET protocol is handled automatically by the PROFINET Communication Module and the PLC operation system. When the Communication Module is initialized in the proper way and the user application is running, the Communication Module and the bus become active.

No function blocks are needed for the cyclic process data exchange. The access to the send and receive data to the according operands range can be performed in the direct way. The access takes place either via operands or symbolic variables. Special PROFINET functions are realised by function blocks of the PROFINET Library.

IP assignment

After switching on the power supply of the PLC the user application is loaded and the Communication Module is configured. Then the IP addresses of the IO devices will be assigned by means of DCP. For the identification of the IO devices the Profinet IO device name is used. Which has to be previously assigned using the DCP protocol. For ABB devices additionally the default name and the rotary switch can be used for name assignment.

Initialization

When the user application changes into the run mode, the configured I/O modules are initialized. At this time, the I/O modules get their configuration (and the configuration of possibly connected I/O modules) by the I/O controller.

Then the configured devices are compared with the available I/O modules and I/O modules of the real assembly. If the result of the compare is conformed, the I/O modules get their configuration. Otherwise the available devices get their configuration and the failure is displayed with the error LEDs of the communication interface module. The error can also be displayed by using the diagnosis function blocks.

The Communication Module and the I/O modules change into the cyclic process data exchange when the configuration transfer is completed.

If the configuration is not successful or the cyclic process data exchange between I/O controller and I/O module is broken (e. g. removing of the plug), both participant

- close their communication
- change their status into the initial condition
- try to build up a new connection.

This procedure has no influence on devices where the configuration was successful.

Because of that a replacement of a faulty I/O module can be done without restarting the PLC. But you have to consider that the new device must have the same position of the switches for setting the device identifier like the replaced one and switch off the power supply of the device you want to replace.

Diagnosis

In addition to the user data transportation PROFINET RT provides a wide range of commissioning and diagnostic functions. The upcoming diagnosis events from the I/O modules are centralized in the I/O controller and can be displayed using several function blocks. Due to this the localization of errors is made very simple *Chapter 1.6.4.2.3.6.4 “Diagnosis views” on page 5556*.

PROFINET function block library

The function block library PROFINET_AC500_V13.lib contains different function blocks to get I/O controller and connected I/O modules information about the status of communication and error states. These function blocks can be embedded additionally, especially for the initial operation.

A detailed function block description can be found in the PROFINET library.

PROFINET status and diagnosis via function blocks

The status messages of the PROFINET bus are requested by the function blocks PNIO_STATE and PNIO_SYS_DIAG, which give following information:

Function block name	Function
Chapter 1.5.4.27.1.9 “PNIO_STATE” on page 1826	Common information about the PROFINET bus state
Chapter 1.5.4.27.1.10 “PNIO_SYS_DIAG” on page 1829	Detailed information about the PROFINET bus state

PROFINET IO device error diagnosis via function blocks

Error and status messages describing I/O devices are queried by PNIO_DEV_ALARM.

Function block name	Function
Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794	Representation of diagnosis messages from connected I/O modules

Identification and maintenance functions

PROFINET standard defines "Identification & Maintenance" data (I&M) to have access to device specific information like serial number, order number etc. This I&M data is described as I&M0 and I&M1 – I&M5 data. I&M0 and I&M1 – I&M4 data should be provided by all types of I/O modules; I&M5 is optional.

Access to this data should be provided via PROFINET acyclic read/write services at specific slot/subslot indexes.

The following shows how to access I&M data in AC500:

	Read / Write	Accessible at Slot / Subslot	Supported by
I&M0	Read only	Any configured slot/subslot	CI50x + CM589-PNIO + CM589-PNIO-4
I&M1-4	Read	Any configured slot/subslot Responds with data stored at DAP submodule interface Serves as representative for all configured slots/subslots	CM589-PNIO + CM589-PNIO-4
I&M1-4	Write	DAP submodule Interface Representative for all configured slots/subslots	CM589-PNIO + CM589-PNIO-4
I&M5	Not supported	-	-

I&M data provides the following details:

I&M0	VendorID (WORD) OrderID (STRING:20) SerialNr (STRING:16) HW-Revision (WORD) SW-Revision (1 CHAR plus 3 x BYTE, e. g. V.1.2.3) RevisionCounter (WORD) ProfileID (WORD) IM_ProfileSpecType (WORD) IM_Version (2 x BYTE) IM_supported (WORD)
I&M1	Tag function (STRING:32) Tag location (STRING:22)
I&M2	Installation date (STRING:16)

I&M3	Descriptor (STRING:54)
I&M4	Signature (STRING:54)

AC500 PROFINET function block Library provides function block *PNIO_IM0* to read I&M 0 data for a certain I/O module.

See for detailed function block description *Chapter 1.5.4.27.1.6 “PNIO_IM0” on page 1813*.

The I/O modules CI501-PNIO and CI502-PNIO support I&M0 function.

The structure and the meaning of these information is specified in the PNO "Profile Guidelines Part 1: Identification & Maintenance Functions" (version 1.1.1, order number 3.502).

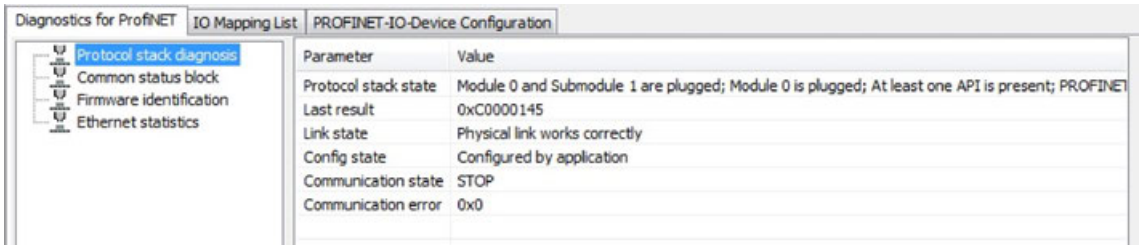
Online diagnosis

The online diagnosis is only available by using special function blocks.

Diagnosis views

PROFINET IO devices provide diagnosis views which consist of the following parts:

- *Basic diagnosis views* which are common for all netX based communication modules:
  - Common status block view
  - Firmware info view
- *Protocol specific views* which provide specific PROFINET IO information for the PROFINET IO device protocol.
  - Protocol stack diagnosis
  - Ethernet statistics



Protocol stack diagnosis view

The Protocol stack diagnosis shows PROFINET IO device protocol stack related diagnosis information.

Parameter	Description
Protocol stack state	Current state of the protocol stack. This value is bit coded and is able to carry several state information in combination. Following details may be combined to a value: ● Device Information set ● PROFINET stack started ● At least one API present ● Module 0 plugged ● Module 0 and sub-module 1 plugged ● Network communication allowed ● Network communication enabled ● Configuration locked ● Fatal error occurred ● PROFINET diagnosis exists ● PROFINET maintenance required ● PROFINET maintenance demanded ● FiberOptic maintenance demanded port 0 ● FiberOptic maintenance required port 0 ● FiberOptic maintenance demanded port 1 ● FiberOptic maintenance required port 1
Last result	Last error code that has occurred in past.
Link state	Current state of the communication link. Values: ● No information available ● Physical link works correctly ● Low speed of physical link ● No physical link present
Config state	Current state of configuration of the protocol stack. Values: ● Not configured ● Configured with DBM Files ● Error during configuration with DBM Files ● Configured by application ● Configuration by application is running ● Error during configuration by application ● Configured with warm-start parameters ● Configuration with warm-start parameters is running ● Error during configuration with warm-start parameters
Communication state	Current state of communication of the protocol stack. Values: ● Unknown ● Offline ● Stop ● Idle ● Operate
Communication error	Currently active error code. If the cause of an error is resolved the value is set 0 (OK) and can be checked in the entry *Last result*.

Firmware info view

Parameter	Description
System channel version	Version number of the system channel implementation
System channel firmware name	Firmware name of the system channel implementation
System channel firmware build date	Build date of the system channel implementation
Protocol channel version	Version number of the protocol channel implementation
Protocol channel firmware name	Firmware name of the protocol channel implementation
Protocol channel firmware build date	Build date of the protocol channel implementation

Ethernet statistics view

For analyzing this view, basic Ethernet knowledge is required.

Ethernet statistics provides information about transmit and receive traffic. The load and the quality of communication can be observed. Problems due to too much network load or internal resource problems will be shown here in terms of error counters.

Parameter	Description
Frames transmitted successfully	Number of frames transmitted successfully
Frames with single collision	Number of frames with single collision detected
Frames with multiple collisions	Number of frames with multiple collisions detected
Late collisions	Number of frames with collision detected later than 512 bit times
Link down during transmission	Number of frames transmitted while link state went down
UTX FIFO underflow	Number of UTX FIFO underflows occurred while transmitting frames
Fatal transmission errors	Number of TPU error codes detected while transmitting frames
Frames received successfully	Number of frames received successfully
Sequence check errors	Number of frames received with valid length but not passing the FCS check
Alignment errors	Number of frames received with not valid length and not passing the FCS check
Frame too long errors	Number of frames received with maximum permitted frame size exceeded
Valid frames with length between 42 and 63 bytes	Number of frames received with length between 42 to 63 bytes and valid checksum

Parameter	Description
Collision fragments	Number of frames received with length smaller than 64 bytes and invalid checksum
Dropped frames due to low resources	Number of frames received while no resources available to handle the frames
Dropped frames due to URX FIFO overflow	Number of UTX FIFO overflows occurred while receiving frames
Fatal receive errors	Number of RPU error codes detected while receiving frames

1.6.4.2.4 EtherCAT communication module

EtherCAT overview

Features

EtherCAT is an open standard for the realization of Industrial Ethernet based automation applications and is highly suitable for time critical applications. Its protocol minimizes overhead and increases the share of user data. Further, it provides mechanism to synchronize inputs and outputs. For general information on EtherCAT see *https://www.ethercat.org*.

EtherCAT is a cyclic communication. Only for parameterization, diagnosis and alarms-messages of intelligent field devices, non-cyclic messages will be used.

Standardization: EtherCAT

- IEC 61131-3: PLC Standardization
- IEC 61158: EtherCAT Standardization
- IEC 11801: Wire and connection elements for Ethernet

EtherCAT is registered trademark and patented technology, licensed by Beckhoff Automation GmbH, Germany

Terms, definitions, abbreviations

EtherCAT

CAT 5	Ethernet cable for 100 MHz data size
CAT 5e	Ethernet cable for 100 MHz data size for Gigabit-Ethernet
CAT 6	Ethernet cable for 250 MHz data size
CAT 7	Ethernet cable for 600 MHz data size
CAT 8	Ethernet cable for 1 GBit data size
CoE	CAN over EtherCAT
DC	Distributed Clock
DDF	Device Description File in XML format
ETG	EtherCAT Technology Group
EtherCAT	Ethernet for Control Automation Technology
FMMU	Fieldbus Memory Management Unit
NRT	Non Real-Time

RT	Real-Time
WKC	Working Counter

Transfer mechanism EtherCAT

Seen from Ethernet view, a EtherCAT bus is a single Ethernet participant. This participant receives and sends Ethernet telegrams. Inside this Ethernet participant there are many EtherCAT slaves, which process data on the fly.

The telegrams are only delayed for a few micro seconds by each slave. Each telegram is processed by all the slaves until it arrives at the last slaves in the line. The processed telegram is sent back from the last slave to the first slave (see figure), which relays it back to the master.

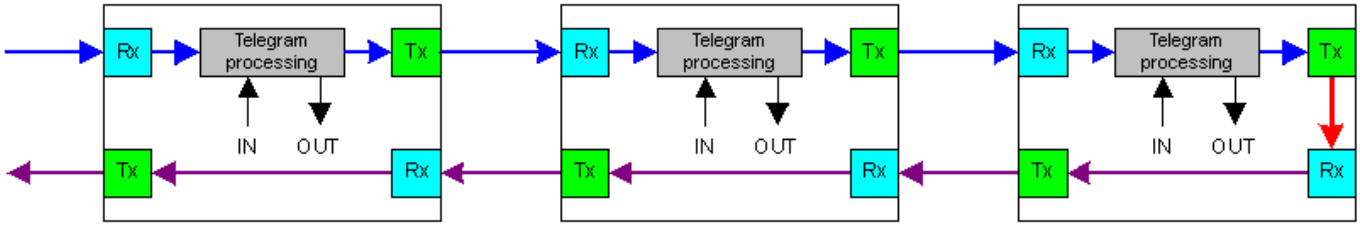


Fig. 1133: EtherCAT telegram routing

Telegram processing

The telegrams are processed on the fly. During forwarding the telegram to the next device, the slaves interpret the EtherCAT commands and read its input data. Output data for the master or for another slave is also inserted in the telegram.

The processing of telegrams is carried out by the hardware. Thus, the processing speed of an eventually connected micro processor does not influence the processing.

Several EtherCAT Process Data Units (PDU) can be embedded in one EtherCAT telegram. Each of them can address one or more slaves.

An EtherCAT PDU consists of

- Header
- Data Area
- "Working Counter".

The Working Counter is incremented by each slave which is addressed by this PDU and which has successfully carried out the command embedded in the header of the PDU. The master-communication-module compares the received Working Counter with the expected value (No. of slaves).

There are two types of EtherCAT telegrams: with and without UDP/IP. Telegrams without UDP/IP can only be used in an Ethernet subnet. Telegrams with UDP/IP allow IP routing and perform communication over router.

AC500 EtherCAT does not support UDP/IP at present.

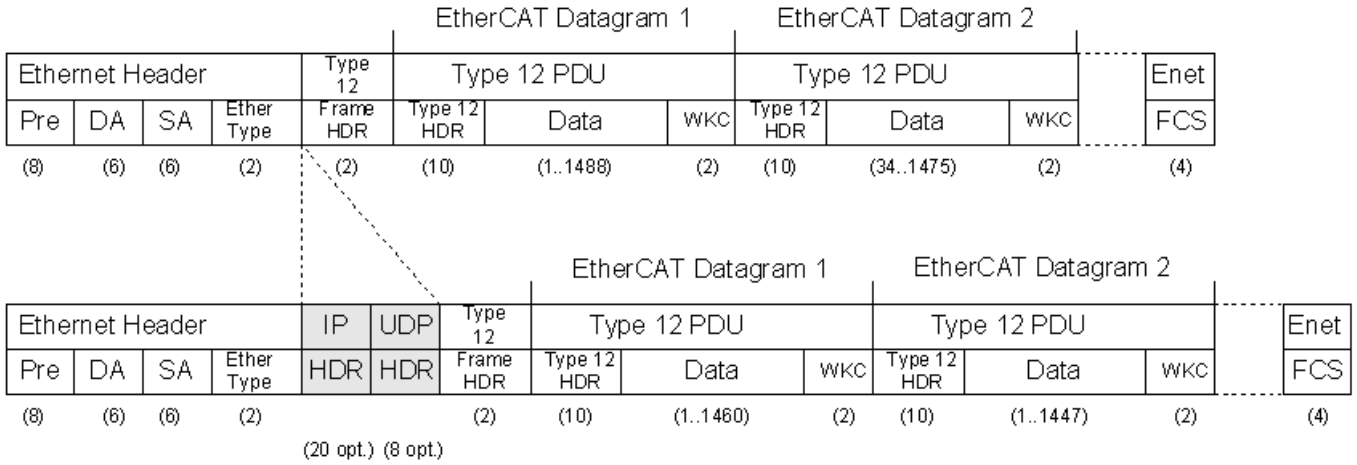


Fig. 1134: Telegram structure

Telegram structure

The individual parts of the frame have the following meaning:

Pre (Preamble)	For synchronization with the signal frequency
DA (Destination Address)	MAC address of the receiver
SA (Source Address)	MAC address of the sender
EtherType	Identifier of the following data (EtherCAT telegram uses 0x88A4)
IP HDR	Optional IP header
UDP HDR	Optional UDP header
Frame HDR	Contains protocol type and frame length
Type 12 PDU	EtherCAT Process Data Unit
Type 12 Header	contains among others: - CMD: command - IDX: index - Address - Length of the data
Data	Process data
WKC	Working Counter
FCS	Frame Checksum

Reduction of overhead

The overhead is reduced through two mechanisms:

- All slaves in an EtherCAT subnet can be addressed with a single telegram
- A telegram can consist of several EtherCAT Process Data Units which contain different commands and data. Each command can perform read/write operations on several slaves.

Addressing modes

Different addressing modes are supported for the slaves. Each telegram contains the segment addressing and a device addressing or a logical addressing.

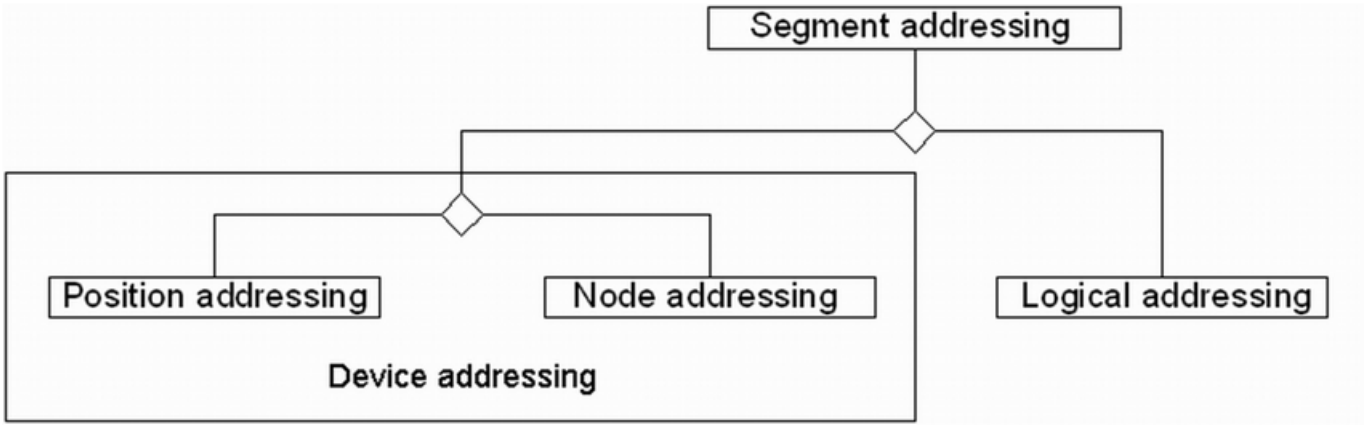


Fig. 1135: Addressing modes

The following figure shows how different addressing modes are integrated in the telegram structure:

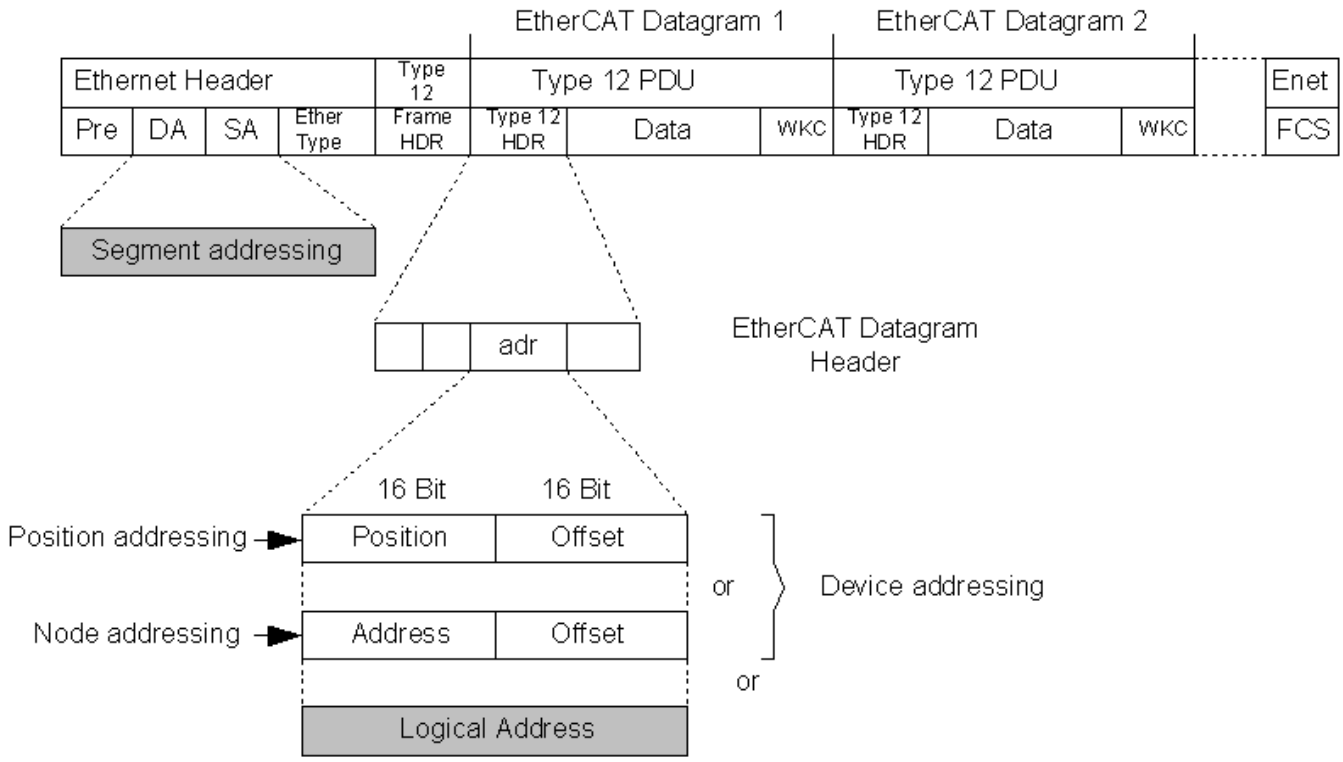


Fig. 1136: The different addressing modes in the telegram structure

Device addressing

In this addressing mode a single slave is addressed with the associated PDU. This addressing mode is usually used for transferring parameters.

There are two sorts of device addressing: position addressing and node addressing.

Position addressing

With this mode, the slaves are identified using their position in the communication network. Normally position addressing is used during start-up. After communication with the slaves is established, the master can give the slave a new node ID according to the projected value.

Node addressing

After receiving the node ID from the master, the slaves can be addressed by using the node addressing mode.

This ensures that, even if the segment topology is changed or devices are added or removed, the slave devices can still be addressed via the same configured address.

Logical addressing

With logical addressing, slaves are not addressed individually. Instead, a section of the segment-wide logical address space (data) is allocated to each salve. Any number of slaves may use separate, the same or overlapping sections.

The Fieldbus Memory Management Units (FMMU) on the slaves handle the assignment of slave local memory address to logical addresses. With this it is possible to address several slaves with a single PDU.

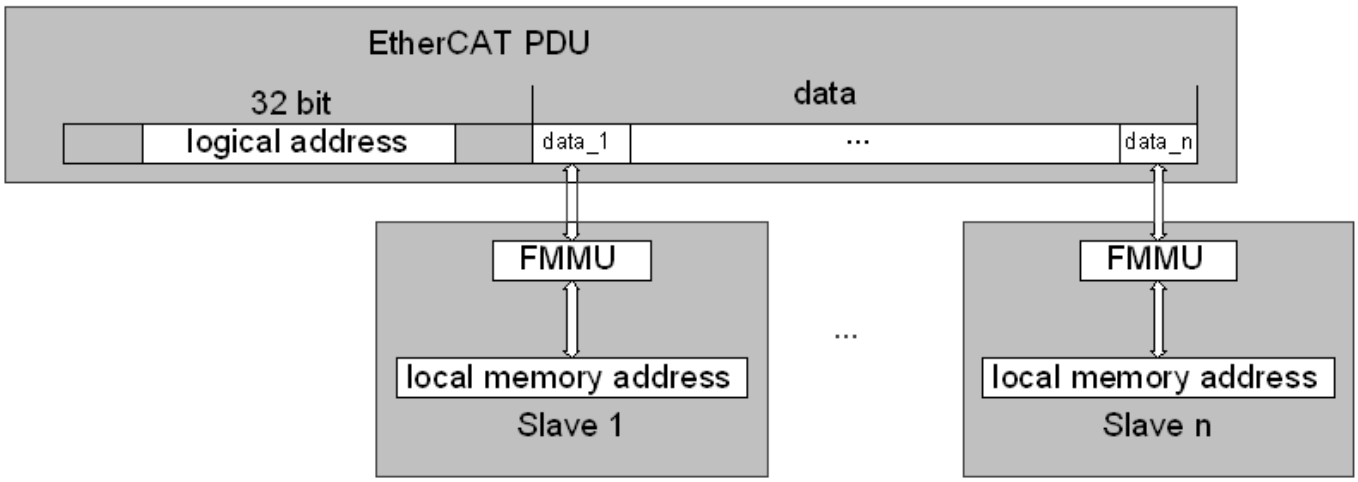


Fig. 1137: Examples of a simple logical address mapping

Distributed clock

Precisely synchronization between the slaves is always required if distributed slaves are used to carry out actions at the same time. One of those applications for example is the control of several servo axes, which shall carry out coordinated movements.

Another application is the AC500 camswitch functionality. In this application it is important to calculate the angle velocity and the position of the main axis precisely.

EtherCAT uses synchronization according to IEEE 1588. This method provides synchronization which is very resistant to communication disturbances.

This synchronization mechanism eliminates:

- different start-up times
- delay times between the slave with the master clock and all other slaves
- drift of the local clocks.

CANopen over EtherCAT (CoE)

The CANopen protocol is preferably used for transferring configuration parameters to the slaves. It can also used for the transmission of process data. Instead of a standard CANopen protocol, which can only transfer 9 bytes of user data, CoE can transfer a maximum of 1478 bytes. Therefor, the CoE protocol can be used to transfer process data with variable length.

Device description file (DDF)

The features of an EtherCAT slave are described in an electronic data sheet, the Device Description File (DDF). This XML-based document gives the characteristics of the device unambiguously and completely with a vendor-independence format.

The characteristics of the CI51x-ETHCAT devices are:

- Description of each input/output group of the device
- Parameters, which are required for the correct functionality of the device

The configuration of an EtherCAT system is highly simplified with the DDF, which is normally provided by the vendor.

Device names and allocation of addresses

The Communication Module ↳ *Chapter 1.6.2.4.6 "EtherCAT" on page 4066* acts as a master device in an EtherCAT master-slave arrangement. It is connected to the Processor Module via an internal communication bus.

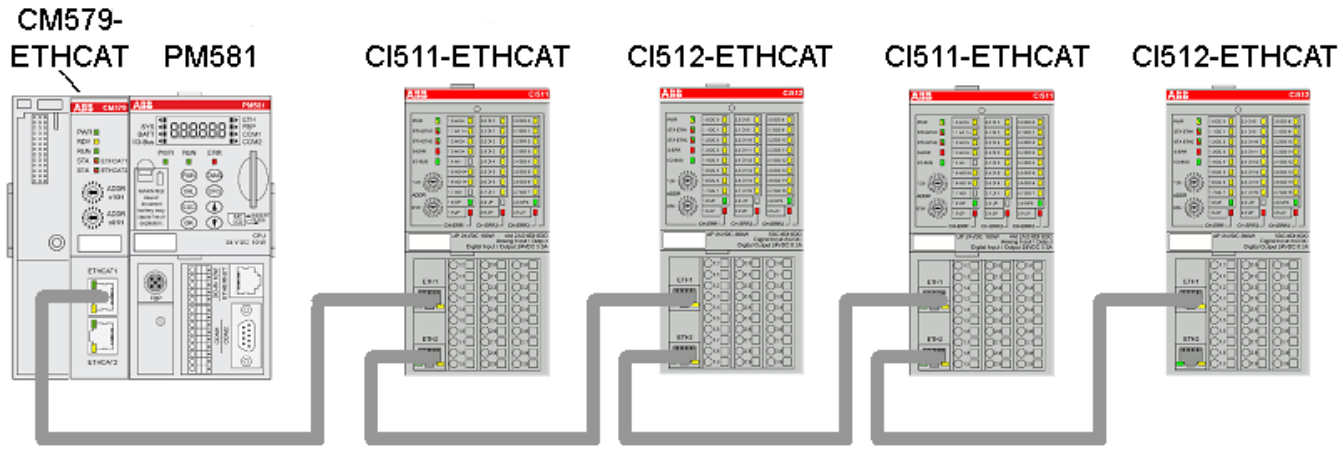
The Communication Interface Modules ↳ *Chapter 1.6.2.8.4 "EtherCAT" on page 4814* act as slave devices in an EtherCAT master-slave arrangement. The slave devices are identified through their position in the bus segment. Address switches are not used. Thus it is possible to give the devices a specific name with a configuration tool before system activation.

The default settings of the EtherCAT slaves need not to be changed before operation.

> ***Risk of communication faults!***
> *The device names are used to generate global variables for the PLC program. Each device names can only be used once within a network.*
>
> *Make sure that all device names are unique.*

Designing and planning a network

AC500 EtherCAT supports only the bus (line) structure.



System performance

Bus communication

The EtherCAT communication is realised using 100Mbit. There is one telegram for all EtherCAT slaves in a segment, so overhead is substantially reduced.

The terminal-to-terminal response time depends on the size of the total process image of all slaves and the CPU load of PLC.

Terminal-to-terminal response time of the CI511-ETHCAT/ CI512-ETHCAT

The terminal-to-terminal response time for digital I/Os of CI51x-ETHCAT is 2 bus cycles (e.g. 2 ms with bus cycle time set to 1 ms) with the CPU PM591, EtherCAT distributed clock activated and synchronized task type (triggered by external event). The terminal-to-terminal response time for analog I/Os is about 13 ms for bus cycle time of 1 ms.

50 % CI511-ETHCAT and 50 % CI512-ETHCAT means e. g. that in case of 16 devices, 8 CI511-ETHCAT and 8 CI512-ETHCAT are involved.

Distributed control cams with EtherCAT

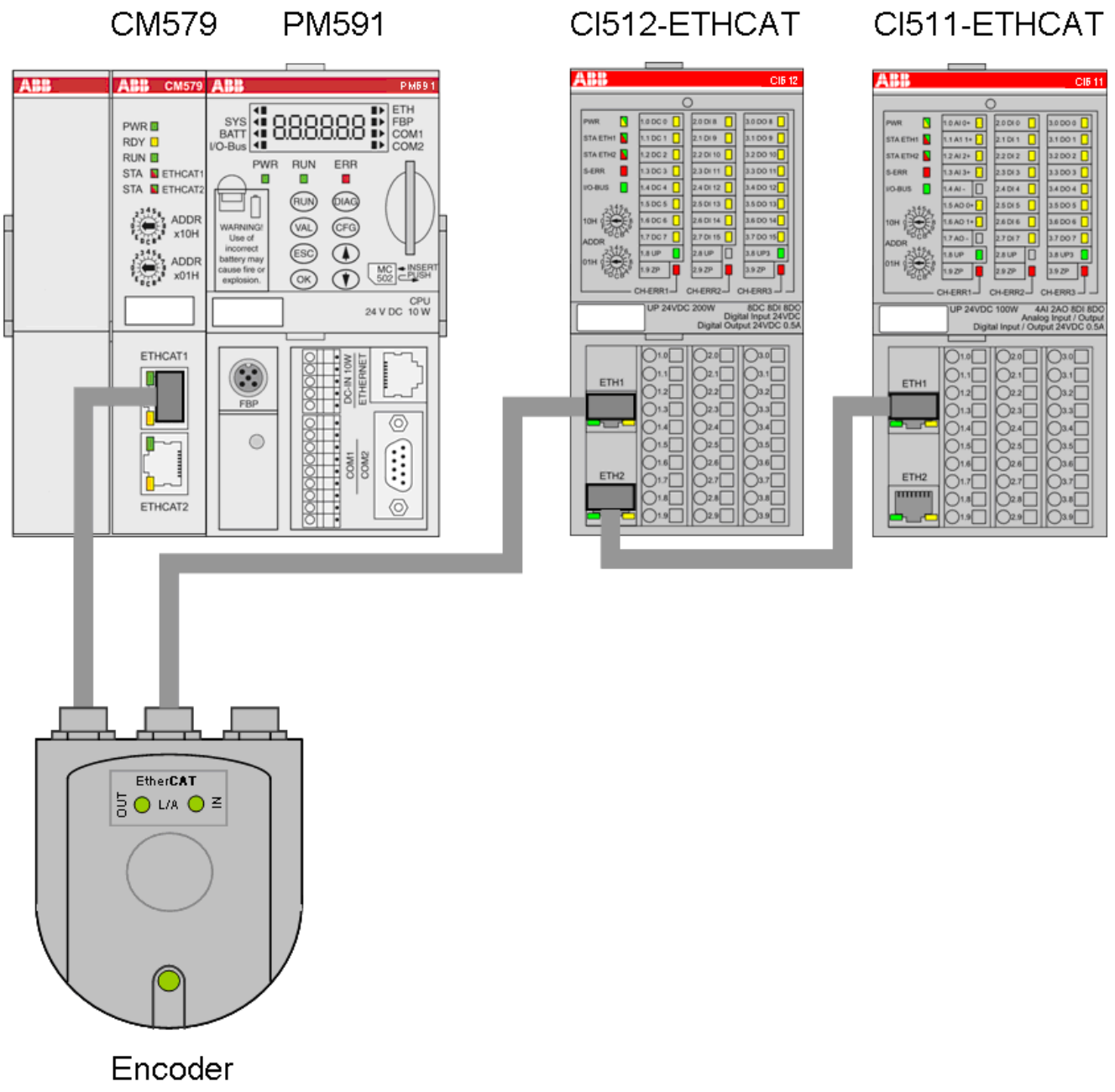
Concept

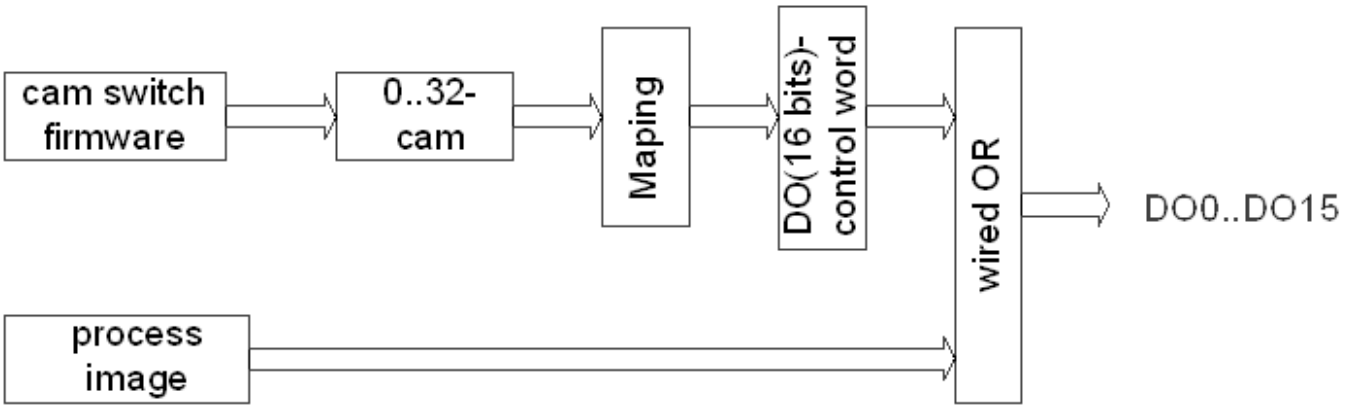
The devices CI511-ETHCAT and CI512-ETHCAT can be used as distributed electrical cams with communication over EtherCAT.

The encoder, which has to be the first slave of the line structure, generates the reference axis.

Each device has up to 8 (CI511-ETHCAT) respectively 16 (CI512-ETHCAT) digital outputs and up to 32 cams. The cams can be mapped to the digital outputs according to the user specifications.

The devices can also process dynamic cams with compensation time for each cam.





The 32 cams of each device can be assigned to any digital outputs, so several switching actions for each digital output and each turn are possible. Each cam consists of one switch-on and one switch-off-action. If a digital output is not assigned to a cam, it can be used as a standard output and controlled by the PLC.

The PLC can further more set any digital output at any time to overwrite the cam state. It can also clear the enable bit of the cams. In this case, the digital output will follow the value in the process image.

This conception allows a flexible utilization of the available digital outputs as cam output or standard digital outputs.

The mapping between the cams and the digital outputs is a part of the configuration and can not be changed online.

Cam accuracy

The cam in the CI51x-ETHCAT uses the positions with time stamp, which comes from the encoder with each telegram, to calculate its switching time. It uses the internal synchronized clock to eliminate

- dead times,
- jitter of EtherCAT telegrams,
- cycle time of the encoder.

The accuracy of the CI51x-ETHCAT cams implementation depends only on the number of the projected cams (parameter NumOfUsedCam) of each device.

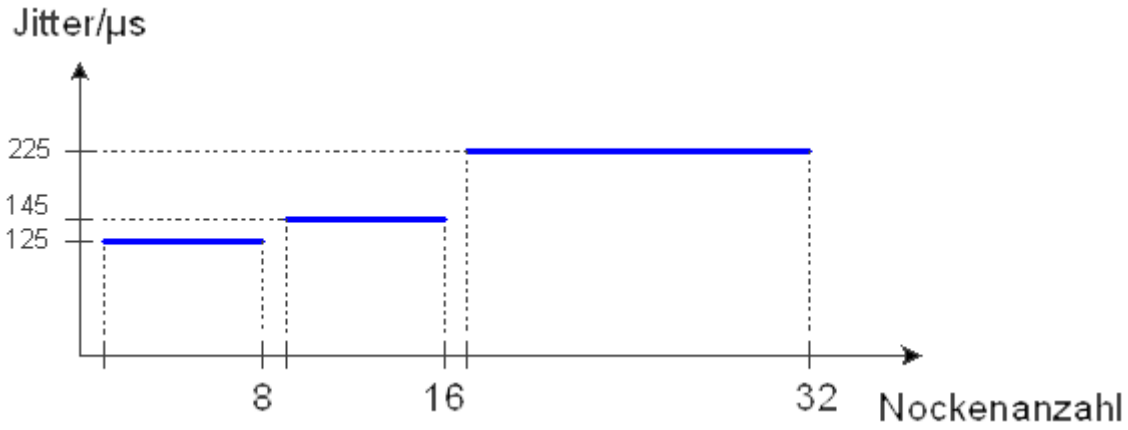


Fig. 1138: Cam accuracy (worst case)

Particular features

In order to reduce the data amount of the process data image, some parameters for the cams, e. g. FirstOnPosition, LastOnPosition and Compensation, are transferred to the EtherCAT slave using the multiplexing method. Parameters of only one cam out of 32 cams are embedded in each EtherCAT telegram. The multiplexing mechanism is handled completely by the function blocks. The user program is released from this task.

The enable-bits of the cams are treated differently: They are transferred to the slaves with each EtherCAT telegram, so the cam can be enabled or disabled very quickly.

Encoder

To facilitate the cam switch functionality, an encoder must be used. The encoder gives the time stamped position of the referent axis to the CI51x-ETHCAT. The encoder must be the first EtherCAT slave in the line structure.

ABB hardware-components work properly with the 18 bit encoder GBMMS or GBMMW of Baumer-IVO. Different bit resolutions of the encoder can be used, but a high resolution will reduce jitter of the cams. The specified data in Cam Accuracy is applied for encoder with 18 bits resolution.

Further details of the IVO encoder can be found at *http://www.ivo.de*.

Encoder wiring

ABB Components use Ethernet cables with RJ45 connectors, but the encoder uses a M12 connector. A suitable connection between the ETHCAT Communication Module (CM5xy-ETHCAT) and encoder respectively between encoder and ETHCAT Communication Interface Modules (CI5xy-ETHCAT) is required. The following connector can be used:

Table 708: Connection of EM 12 S OCTOPUS:

	Name	Ethernet cable	EM 12 S OCTOPUS
	TxD+	White/green	Yellow pin 1
	TxD-	Green	Orange pin 3
	RxD+	White/orange	White pin 2
	RxD-	Orange	Blue pin 4

Software components

In addition to the standard EtherCAT_AC500 library, there are two function blocks in the MC_EtherCAT_AC500 library: ↳ *Chapter 1.5.4.5.1.1 “MC_CamSwitch_DC” on page 853* and ↳ *Chapter 1.5.4.5.1.2 “PS_DigitalPLS” on page 857*.

The two function blocks allow a simply control of the cams.

Detail information about configuration and programming application with cam can be found in "EtherCAT configuration examples".

EtherCAT implementation

System start-up behavior

Initial operation

The EtherCAT protocol is handled by the EtherCAT Communication Module and the PLC operating system automatically. When the Communication Module is initiated in the proper way and the user application is running, the Communication Module and the bus become active.

No function blocks are necessary for exchanging process data via EtherCAT. Furthermore, it is possible to access the send data and the receive data in the according operands range directly. The access takes place with either operand or symbolic variables. Special EtherCAT functions can be realized by using the function blocks of the EtherCAT library.

To include an AC500 EtherCAT Communication Module in the PLC system, create an EtherCAT configuration and write it into the Communication Module. A detailed documentation can be found in the document "Example: EtherCAT Configuration".

System start-up

When the user application changes into the run mode, the communication module tries to establish communication with the configured slaves.

The communication module will not change into run mode if:

- the number of the configured slaves is not equal to the number of connected slaves.
- the type of the configured slaves is not equal to the type of connected slaves.

In this case, the LED STA2 ETHCAT (master communication module) switches on.

After communication to the slaves is established, the distributed clocks will be synchronized. This process can last up to 1 minute. During the synchronizing process, the LED S-ERR (slave) switches on. After a successful synchronization of the distributed clocks, the LED S-ERR switches off and the LED PWR, STA1 ETHCAT and STA2 ETHCAT of the slave are on.

For details see the device description of ↳ *Chapter 1.6.2.4.6.1 “CM579-ETHCAT - EtherCAT master” on page 4066*, ↳ *Chapter 1.6.2.8.4.1 “CI511-ETHCAT” on page 4814* and ↳ *Chapter 1.6.2.8.4.2 “CI512-ETHCAT” on page 4846*.

If a communication interface module will be disconnected from the EtherCAT network during operation, it will not continue with the communication after reconnecting. A restart of the whole system must be performed.

EtherCAT sync

Overview

The EtherCAT master CM579-ETHCAT supports 2 modes for synchronized operation. These modes can be used for synchronization between PLC task and IO image.

The master has the same functionality as in unsynchronized mode, but with every new bus cycle it additionally creates an interrupt. With this interrupt the PLC task will be triggered.

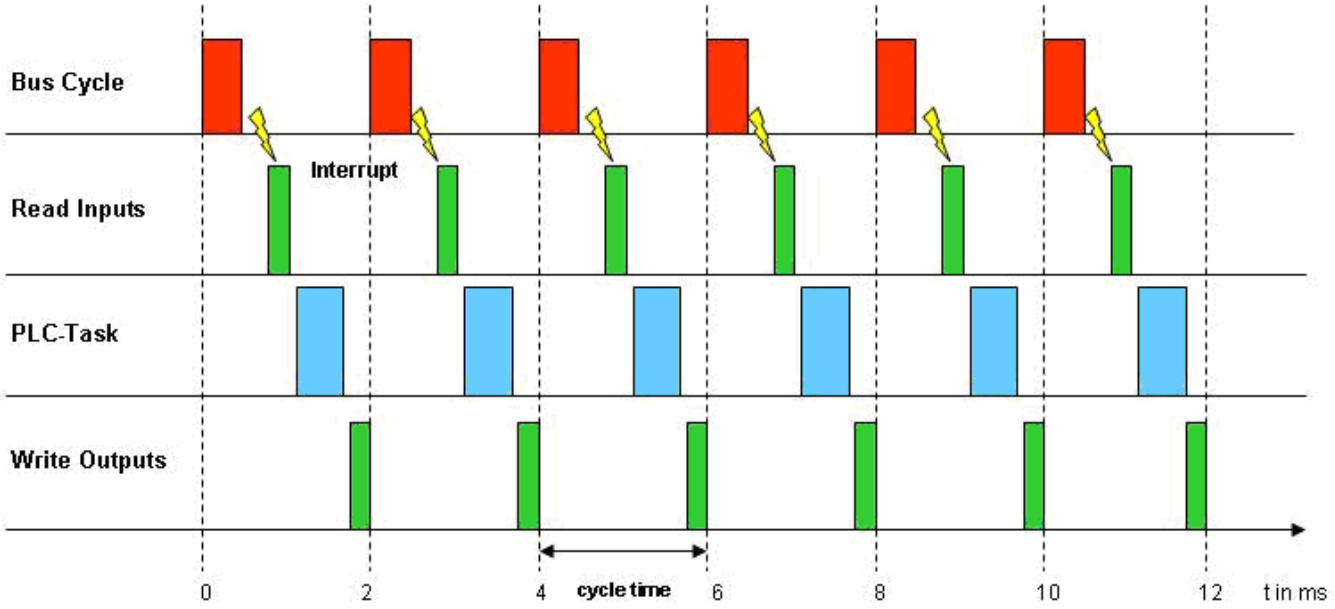
System requirements

The functionality described here has the following system requirements.

PLC	Firmware version 2.1.3
CM579-ETHCAT	Firmware version 2.4.11
Control Builder Plus	2.1.0
Automation Builder	1.0

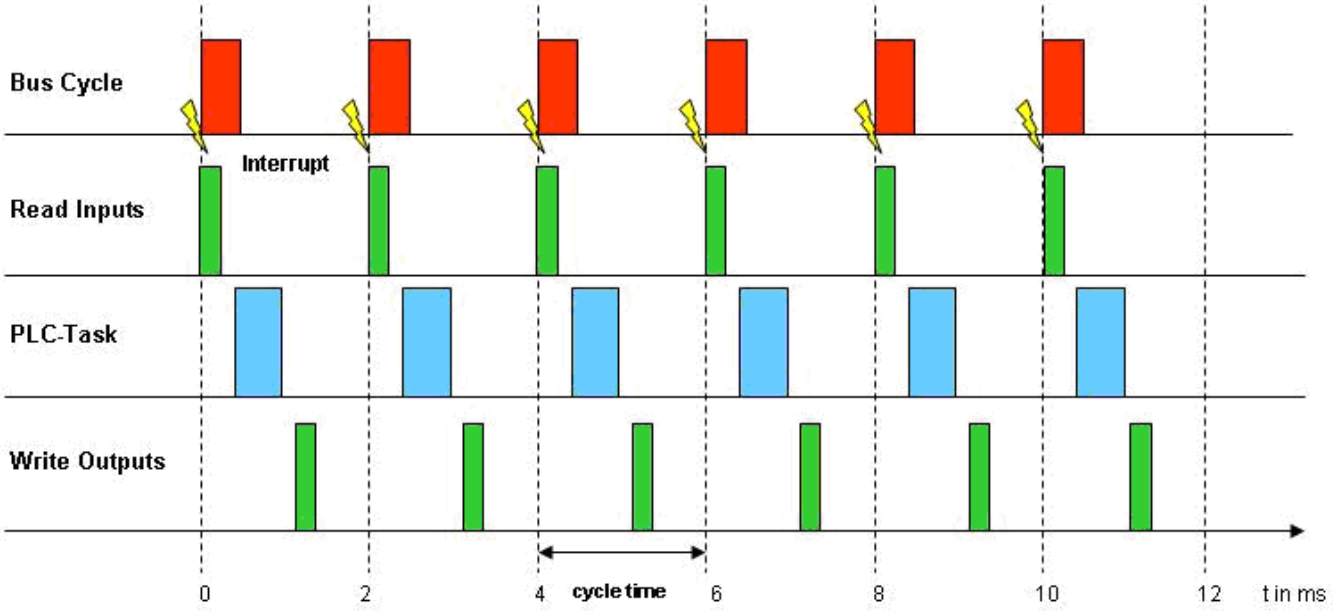
Sync mode 1

In this mode, the interrupt for task synchronization is generated at the end of the current bus cycle. If the PLC is triggered by this interrupt, it reads the input values at first. Then it starts the corresponding task. And at the end of the task, it writes the output values back to EtherCAT master. If the PLC task is not finished within a bus cycle a diagnosis message will be created.



Sync mode 2

The difference between sync mode 1 and sync mode 2 is the instant of time when the interrupt is generated. In sync mode 2 it will be generated at the beginning of a bus cycle. The PLC task uses therefore the input values of previous bus cycle. If the PLC task is not finished within a bus cycle a diagnosis message will be created.



Diagnosis

2 error counters are available for diagnosis purposes. It can be read and evaluated within a PLC program using function block ↳ *Chapter 1.5.4.14.1.7 “ECAT_SYNC” on page 1315* from EtherCAT Library. More detailed information can be found in the corresponding documentation of this library ↳ *Chapter 1.5.4.14 “EtherCAT library” on page 1295*.

Diagnosis

In addition to the user data transportation, EtherCAT provides a wide range of diagnostic functions, which can be useful for system activating. The upcoming diagnostic events from the EtherCAT slaves are centralized in the master and can be displayed by using several function blocks. Localization of failures is greatly simplified with this function.

EtherCAT function block library

The function block library EtherCAT_AC500_V13.lib (since firmware version 1.3.0) ↳ *Chapter 1.5.4.14 “EtherCAT library” on page 1295* contains different function blocks to get information about the status of communication and error states of the EtherCAT Communication Module and connected EtherCAT slaves. These function blocks can be embedded additionally, especially for the initial operation.

EtherCAT status and diagnosis via function blocks

The EtherCAT status messages can be requested by the following function blocks:

Function block name	Function
↳ *Chapter 1.5.4.14.1.6 “ECAT_STATE” on page 1311*	Common information about the status of the EtherCAT bus
↳ *Chapter 1.5.4.14.1.1 “ECAT_BUS_DIAG” on page 1296*	Information about the status of all bus participants
↳ *Chapter 1.5.4.14.1.4 “ECAT_GET_DCLK_DEVI” on page 1305*	Information about the distributed clock of every slave
Status function blocks in the library EtherCAT_AC500_V13.lib	

A detailed function block description is in the document The EtherCAT Library.

EtherCAT slave diagnosis via function blocks

Status and error messages, which describe the slave state, are requested via the ECAT_SLV_DIAG function block:

Function block name	Function
Chapter 1.5.4.14.1.5 “ECAT_SLV_DIAG” on page 1308	Representation of diagnosis messages of connected slaves
Slave diagnosis function blocks in the library EtherCAT_AC500_V13.lib	

Device revision

Hardware and firmware revision can be queried using the *Chapter 1.5.4.15.1.6 “ECAT_SOE_READ” on page 1333* function block with index (idx) 2006h and subindex (subidx) 1:

- HW-Revision (WORD) is stored in the first 2 bytes,
- SW-Revision (1 CHAR plus 3 x BYTE, e. g. V.1.2.3)

Online diagnosis

The online diagnosis in Automation Builder software is only available by using special function blocks *“EtherCAT slave diagnosis via function blocks” on page 5571*.

1.6.4.2.5 CANopen communication modules

CANopen overview

Features

Fundamental properties and fields of application

CANopen is a standardized layer 7 protocol used for decentralized industrial automation systems based on the Controller Area Network (CAN) and the CAN Application Layer (CAL). CANopen is based on a communication profile containing the determination of basic communication mechanisms and their descriptions, such as the mechanisms used for exchanging process data in real-time or for sending alarm telegrams. This common communication profile is the basis for the various CANopen device profiles. The device profiles describe the specific functionality and/or the parameters of a device class. Such device profiles are available for the most important device classes used in industrial automation, such as digital and analog I/O modules, sensors, drives, control units, regulators, programmable controllers or encoders. Further device profiles are projected.

The central element of the CANopen standard is the device functionality description in an object directory (OD). The object directory is divided into one general area containing information about the device (e.g. device identification, manufacturer's name, etc.) as well as communication parameters, and the device-specific area describing the particular functionality of the device. These properties of a CANopen module are documented in the form of a standardized "electronic data sheet" (EDS file).

A CANopen network can consist of a maximum of 128 modules, one NMT master and up to 127 NMT slaves. In contrast to the typical master-slave systems (e.g. PROFIBUS systems), the meanings of the terms master and slave are different for CANopen. In operational mode, all modules are independently able to send messages via the bus. Moreover, the master is able to change the operating mode of the slaves. The CANopen master is normally implemented by a PLC or a PC. The bus addresses of the CANopen slaves can be set in the range between 1 and 127. The device address results in a number of identifiers occupied by this module.

CANopen supports transmission rates of 10 kbit/s, 20 kbit/s, 50 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s, 800 kbit/s and 1 Mbit/s. Each CANopen device has at least to support a transmission rate of 20 kbit/s. Other transmission rates are optional.

Standardization: CANopen

BOSCH CAN specification - version 2.0, part A and part B

ISO 11898

CiA DS 201 V1.1 - CAN Application Layer

CiA DS 301 V3.0 - CAL based Communication Profile for Industrial Systems

CiA DS 301 V4.02 - CANopen Application Layer and Communication Profile

CiA DS 401 V2.1 - CANopen Device Profile Generic I/O modules

CiA DS 402 V2.0 - CANopen Device Profile Driver and Motion Control

CiA DS 406 V3.0 - CANopen Device Profile Encoder

Terms, definitions and abbreviations

CANopen

CAL	CAN Application Layer
CAN	Controller Area Network
CiA	CAN in Automation e.V.
DLC	Data Length Code
EDS	Electronic Data Sheet
ISO	International Standardization Organization
NMT	Network Management
OD	Object Directory
PDO	Process Data Object
RTR	Remote Transmission Request
SDO	Service Data Object

CANopen

- Operating mode CANopen-Master
- Process image with a maximum of 57344 I/O points
- Supports min. boot-up, emergency messages and life guarding
- Supported PDO modes: Event-controlled, synchronous, cyclic and remote PDO transmission
- Integrated device profiles: CiA DS-401, CiA DS-402 and CiA DS-406

CAN (additional functionality, not necessary for pure CANopen operation)

- Support of 11 bit identifiers according to CAN 2.0 A and 29 bit identifiers according to CAN 2.0 B
- Transmission and reception of any CAN telegrams via function blocks in the user program

Transmission technique

- ISO 11898, potential separated
- Transfer rates of 20 kbit/s, 125 kbit/s, 250 kbit/s, 500 kbit/s and 1 Mbit/s
- Bus length up to 1000 m at 20 kbit/s and up to 40 m at 1 Mbit/s
- One bus can have up to 128 subscribers (master + 127 slaves)
- 5-pin COMBICON connector for bus

Communication

- Message-oriented bus access, CSMA/CA
- Predefined master-slave connections
- 8 bytes of non-fragmented user data, for fragmentation any size is possible
- Synchronization of inputs and/or outputs via synchronous PDOs

Protection functions

- Message transfer with Hamming distance HD = 6
- CAN fault recognition mechanisms via 15 bit CRC, frame check, acknowledge, bit monitoring and bit stuffing

- Incorrect parameter settings are avoided because bus subscribers with faulty parameters are not included in the user data operation.
- Adjustable behavior on subscriber failure. System continues normal operation and the error is indicated at the master or the entire system is stopped.
- Response monitoring of the subscribers (node guarding)

Communication mechanisms

CANopen distinguishes two basic mechanisms for data transmission: The fast exchange of short process data via Process Data Objects (PDOs) and the access to entries of the Object Directory using Service Data Objects (SDOs). Service Data Objects are primarily used to transmit parameters during device configuration. The transmission of Process Data Objects is normally performed event oriented, cyclic or on request as broadcast objects.

Service data objects

Service Data Objects (SDOs) are used to modify Object Directory entries as well as for status requests. Transmission of SDOs is performed as a confirmed data transfer with two CAN objects in the form of a peer-to-peer connection between two network nodes. The corresponding Object Directory entry is addressed by specifying the index and the sub-index of the entry. It is possible to transmit messages of unlimited length. If necessary, the data are segmented into several CAN messages.

Process data objects

For the transmission of process data, the Process Data Object (PDO) mechanism is available. A PDO is transmitted unconfirmed because, in the end, the CAN link layer ensures the error-free transmission. According to the CAN specification, a maximum of 8 data bytes can be transmitted within one PDO. In conjunction with a synchronization message, the transmission as well as the take over of PDOs can be synchronized over the entire network (synchronous PDOs). The assignment of application objects to a PDO can be set using a structural description (PDO mapping) that is stored in the object directory. Thus, an adaptation according to the requirements of the individual applications is possible. The transmission of process data can be performed by various methods.

Event

The PDO transmission is controlled by an internal event, e.g. by a changing level of a digital input or by an expiring device-internal timer.

Request

In this case, another bus subscriber is requesting the process data by sending a remote transmission request (RTR) message.

Synchronous

In case of synchronous transmission, synchronization telegrams are sent by a bus subscriber. These telegrams are received by a PDO producer which in turn transmits the process data.

Network management

Within a CANopen network, only one NMT master exists (NMT = Network management). All other modules are NMT slaves. The NMT master completely controls all modules and is able to change their states. The following states are distinguished:

Initialization

After switching-on, a node is first in the initialization state. During this phase, the device application and the device communication are initialized. Furthermore, a so-called boot-up message is transmitted by the node to signalize its basic readiness for operation. After this phase is finished, the node automatically changes to the pre-operational state.

Pre-operational In this state, communication with the node is possible via Service Data Objects (SDOs). The node is not able to perform PDO communication and does not send any emergency messages.

Operational In this state, SDO and PDO communication is possible. The access to the Object Directory via the SDO protocol depends on the device implementation and may be restricted

Stopped In the stopped state, a node is completely disconnected from the network. Neither SDO communication nor PDO communication is possible. A state change of the node can only be initiated by a corresponding network command (e.g. Start Node).

Node guarding and heartbeat

Testing the functionality of a CAN node is particularly required if the node does not continuously send messages (cyclic PDOs). Two mechanisms can alternatively be used to monitor the CANopen nodes. When the node guarding protocol is used, the NMT master sends messages to the available CANopen slaves which have to respond to these messages within a certain time period. Therefore, the NMT master is able to detect if a node fails. Furthermore, the heartbeat protocol can be used with CANopen. In this case, each node automatically sends a periodic message. This message can be monitored by each other subscriber in the network.

Object directory

The object directory describes the entire functionality of a CANopen device. It is organized as a table. The object directory does not only contain the standardized data types and objects of the CANopen communication profile and the device profiles. If necessary, it also contains manufacturer-specific objects and data types. The entries are addressed by means of a 16 bit index (table row, 65536 entries max.) and an 8 bit sub-index (table column, 256 entries max.). Thus, objects belonging together can be easily grouped. The following table shows the structure of this CANopen object directory:

Index		**Object**
dec	**hex**	
0	0000	not used
1...31	0001...001F	Static data types
32...63	0020...003F	Complex data types
64...95	0040...005F	Manufacturer-specific data types
96...607	0060...025F	Device profile specific data types
608...1023	0260...03FF	Reserved
1024...4095	0400...0FFF	Reserved
4096...8191	1000...1FFF	Communication profile (DS-301)
8192...24575	2000...5FFF	Manufacturer-specific parameters
24576...40959	6000...9FFF	Parameters of the standardized device profiles
40960...45055	A000...AFFF	Standardized network variable area

Index		Object
dec	hex	
45056…49151	B000…BFFF	Standardized system variable area
49152...65535	CA000...FFFF	Reserved

Several data types are defined for the objects themselves. If required, other structures (e.g. ARRAY, STRUCT) can be created from these standard types.

Identifiers

CANopen always uses identifiers with a length of 11 bits (standard frames). The number of available and allowed identifiers given by this is divided into several ranges by the pre-defined connection set. This structure is designed in a way that a maximum of 128 modules (1 NMT master and up to 127 slaves) can exist in a CANopen network. The list of identifiers is composed of some fix identifiers (e.g. network management identifier 0) and various functional groups where each existing node, that supports the corresponding function, is assigned to one unique identifier (e.g. Receive PDO 1 of node 3 = 512 + node number = 515). Using the pre-defined connection set therefore avoids double assignment of identifiers.

Identifier	Function	Calculation
0	Network management (NMT)	-
1...127	not used	-
128	Synchronization (SYNC)	-
129...255	Emergency message	128 + node ID
256	Timestamp message	-
257...384	not used	-
385...511	Transmit PDO 1	384 + node ID
512	not used	-
513...639	Receive PDO 1	512 + node ID
640	not used	-
641...767	Transmit PDO 2	640 + node ID
768	not used	-
769...895	Receive PDO 2	768 + node ID
896	not used	-
897...1023	Transmit PDO 3	896 + node ID
1024	not used	-
1025...1151	Receive PDO 3	1024 + node ID
1152	not used	-
1153...1279	Transmit PDO 4	1152 + node ID
1280	not used	-
1281...1407	Receive PDO 4	1280 + node ID
1408	not used	-
1409...1535	Transmit SDO	1408 + node ID
1536	not used	-

Identifier	Function	Calculation
1537...1663	Receive SDO	1536 + node ID
1664...1792	not used	-
1793...1919	NMT error (node guarding, heartbeat, boot-up)	1792 + node ID
1920...2014	not used	-
2015...2031	NMT, LMT, DBT	-

PDO mapping

As already explained, all 8 data bytes of a CAN message are available for the transmission of process data. As there is no additional protocol information, the data format has to be agreed between the sending (producer) and the receiving party (consumer). This is done by the so-called PDO mapping.

If a fixed mapping is used, the process data are arranged in a pre-defined order within the PDO message. This arrangement is predetermined by the device manufacturer and cannot be changed. If variable mapping is used, the process data can be arranged as desired within the PDO message. For this purpose, the address, consisting of index and sub-index, as well as the size (number of bytes) of an object directory entry are entered into the mapping object.

EDS files

The characteristic properties of a CANopen module are documented in the form of an electronic data sheet (EDS file, electronic data sheet). The file completely and clearly describes the characteristics (objects) of a module type in a standardized and manufacturer independent format. Programs for configuring a CANopen network use the module type descriptions available in the EDS files. This strongly simplifies the configuration of a CANopen system. Usually the EDS files are provided by the device manufacturer.

Master-Slave-Arrangement

CANopen master CM598-CN

The communication module CM598-CN is a master device in a CANopen master-slave arrangement. It is connected to the processor module via an internal communication bus. A detailed device description of the device can be found in the device description for ↳ *Chapter 1.6.2.4.5.2 “CM598-CN - CANopen master” on page 4060*

CANopen slaves CI581-CN / CI582-CN

The communication interface modules CI581-CN / CI582-CN are I/O slave devices in a CANopen master-slave arrangement.

A detailed description of the devices can be found in the device descriptions of ↳ *Chapter 1.6.2.8.2.2 “CI581-CN” on page 4685* and ↳ *Chapter 1.6.2.8.2.3 “CI582-CN” on page 4723*.

CANopen slave CM588-CN

The communication module CM588-CN is a slave device in a CANopen master-slave arrangement. It is connected to the processor module via an internal communication bus. The CM588-CN allows communicating of multiple processor modules within a CANopen network.

A detailed description of the device can be found in the device description of ↳ *Chapter 1.6.2.4.5.1 “CM588-CN - CANopen slave” on page 4053*.

Designing and planning a network

The CANopen communication module is connected to the bus using the 5-pin COMBICON connector. For EMC suppression and protection against dangerous contact voltages, the shield of the bus cable has to be connected to protective earth outside the housing. The line ends of the bus cable have to be terminated using bus terminating resistors.

Within a CANopen network, the controller with the CANopen communication module is the NMT master. No other NMT master is allowed in this network. The NMT master completely controls all modules and their operational states. Up to 127 NMT slaves can be connected to an NMT master.

The CANopen master is able to:

- Change operational states of the slaves
- Parameterize the slaves (e.g. communication connections, time supervision, bus traffic)
- Configure slaves (e.g. type, number and channel operating mode)
- Read input data of the slaves
- Write output data of the slaves
- Read diagnosis data of the slaves
- Monitor the availability of the slaves
- Transmit control commands to synchronize the inputs or outputs of the slaves
- Read and write slave objects even during running operation

The CANopen communication module is as well able to:

- Transmit and receive CAN telegrams according to CAN 2.0 A (11 bit identifier) and CAN 2.0 B (29 bit identifier). This additional functionality is not required for pure CANopen operation.

CANopen implementation

System start-up behavior

Initial Operation

The CANopen protocol is handled by the CANopen communication module and the PLC operating system automatically.

When the communication module is initiated in the proper way and the user application is running, the communication module and the bus become active.

No function blocks are necessary for exchanging process data via CANopen. Special CAN functions can be realized by using the function blocks of the CANopen library.

The communication module starts communication via CANopen after the user application is started and then attempts to initialize the configured slaves. After a successful initialization, the communication module and the slaves exchange the process data.

The behavior of the communication module when the user application is stopped can be configured with the CPU parameter *Behavior of outputs in Stop*. See description of the CPU parameters for details.

Synchronization of an application task with the I/O update

CANopen implements the synchronization (SYNC) protocol which is used to synchronize the I/O update of the CANopen slaves. It can also be used to synchronize the I/O update with the execution of an application task.

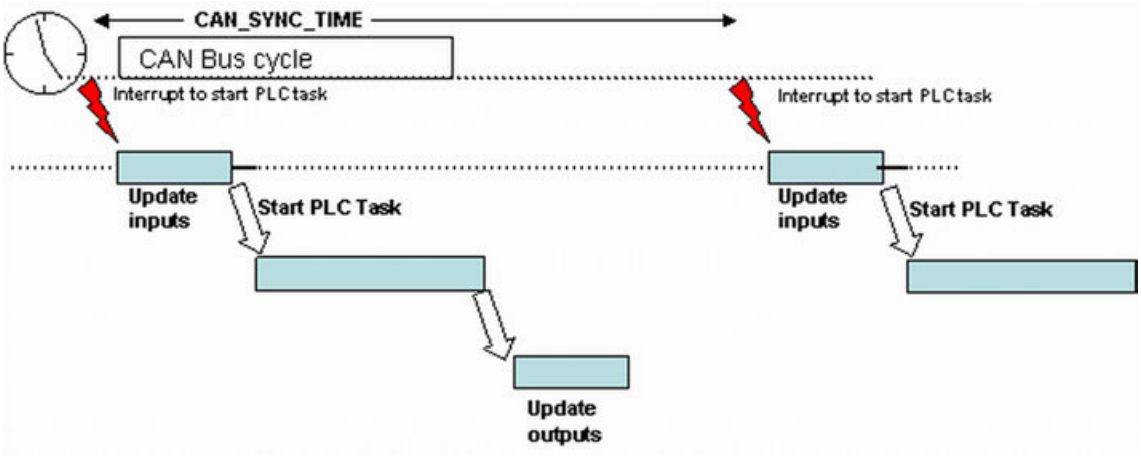
When the Sync message is sent the communication module generates an interrupt. This will trigger the execution of an associated application task.

The described functionality is supported as of the following firmware versions of PLC and communication module:

For CM598-CN

- PLC version: V2.5.3
- Communication module version: V2.0.0.2

The figure below gives a schematic overview of the synchronization between I/O update and execution of an application task:



The CAN_SYNC_TIME is the time between 2 messages of the synchronization protocol. This time has to be chosen long enough to update the I/Os and execute the application task within this time.

The execution time required by the application task can be checked online in the CODESYS task configuration.

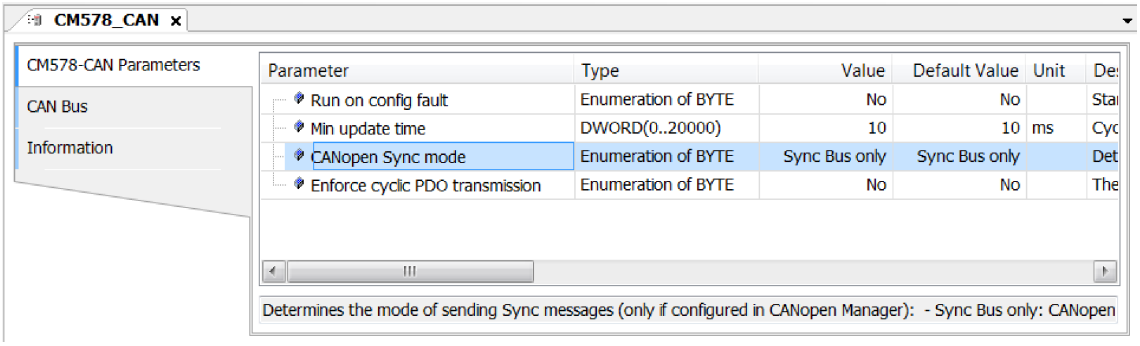
The tasks which are triggered with the CANsync interrupt have to be a higher priority than tasks which are running cyclically.

If other communication modules are attached to the same CPU, the MIN_UPDATE_TIME of those communication modules must be larger than the MIN_UPDATE_TIME of the CANsync communication module.

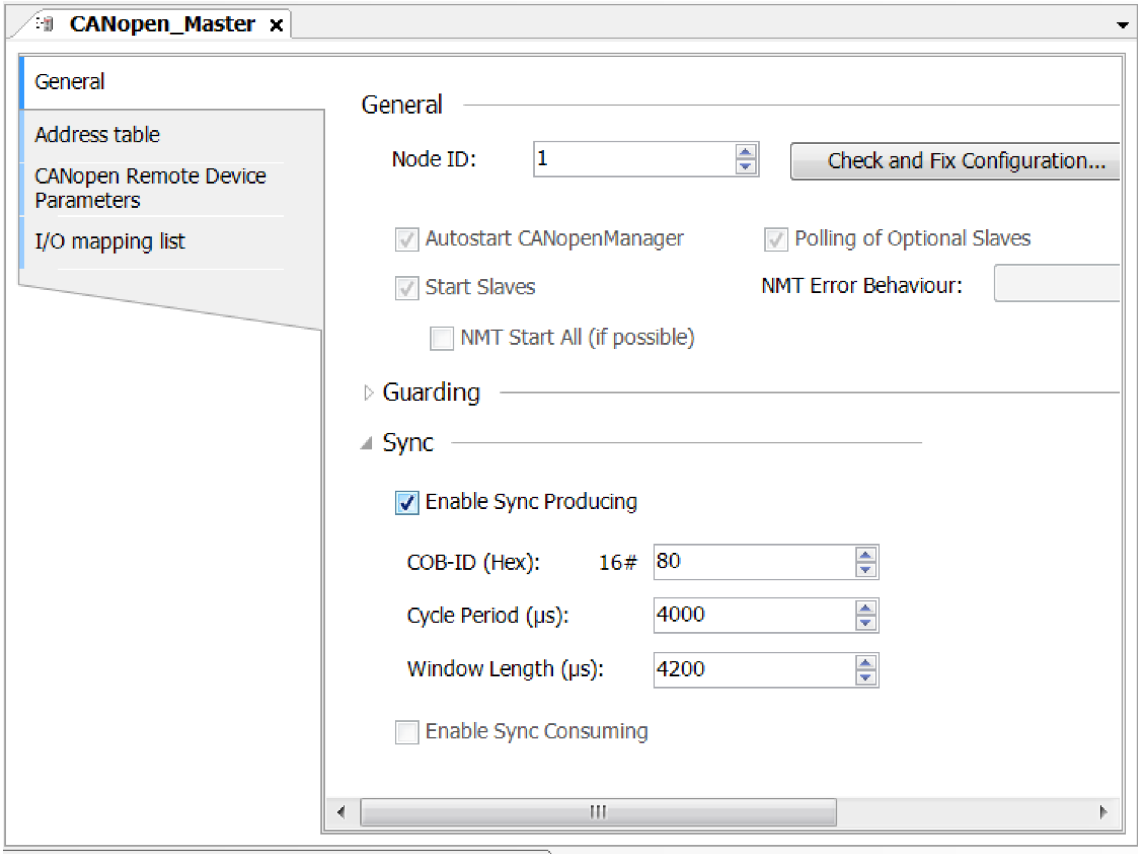
The online overview of the task configuration will display implausible values of the CANsync task. This will not influence the functionality.

Synchronization settings

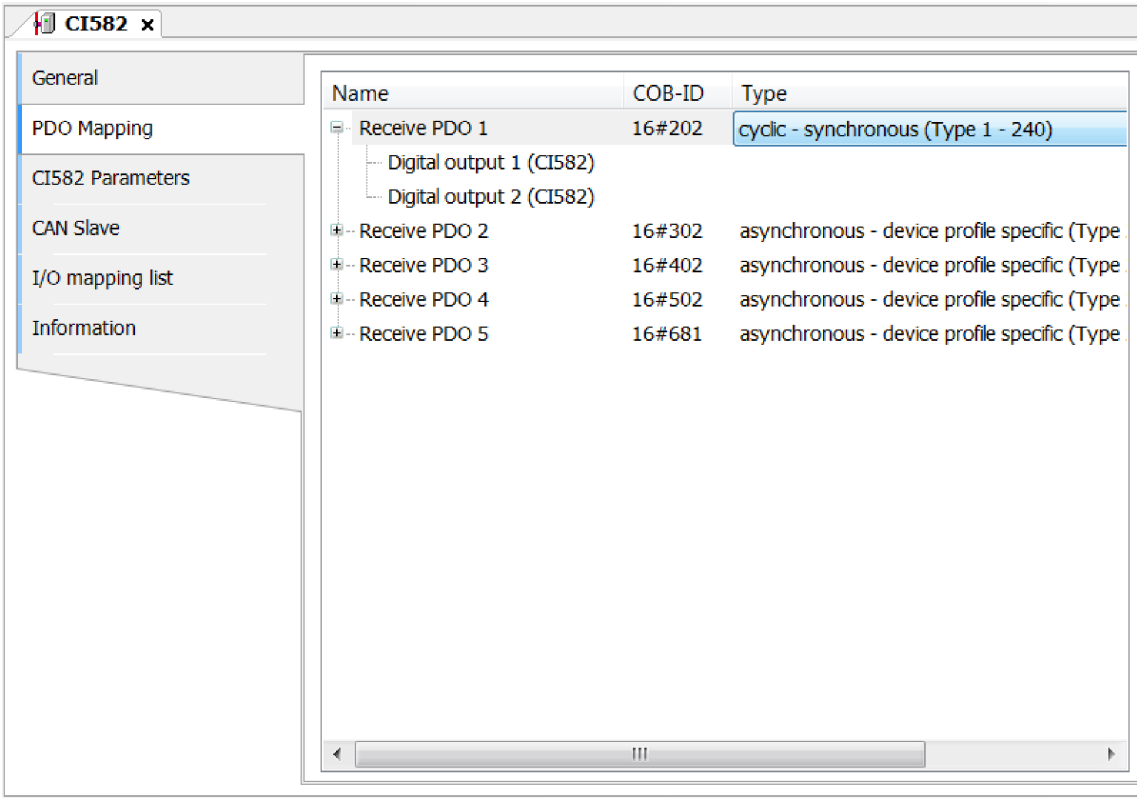
- Parameter *CANopen Sync mode* has to be set to the value *Sync Bus and Task*.



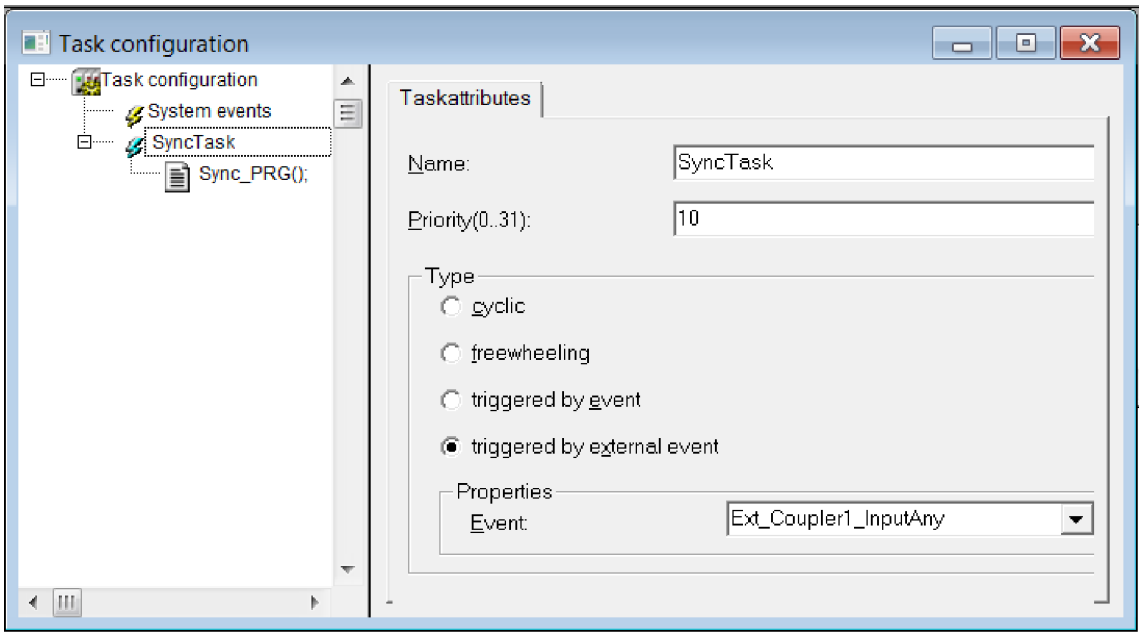
- Checkbox *Enable Sync Producing* has to be enabled in the CANopen configuration.



- At least one PDO should have a transmission mode of type *cyclic - synchronous (1-240)*.



- An application task has to be defined and linked with the associated external event.



Event	Communication module slot	Description
Int_Coupler0_ InputAny	0	Application task will be triggered by I/O update of the communication module at slot 1 - 6.
Ext Coupler <slot> InputAny	1 ... 6	Application task will be triggered by I/O update of the communication module at slot 1 - 6.

Special SDOs for CI581-CN / CI582-CN

Block wise single parameterization (Object 2700hex to 270Ahex)

In this mode the objects 2700hex to 270Ahex are used.

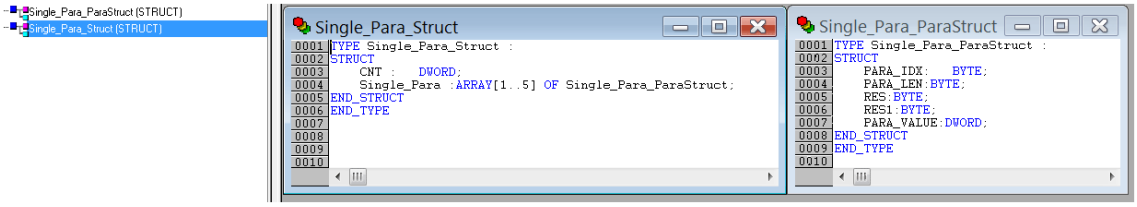
Only sub object 0 is used, the parameter index is transferred inside the data content.

Object 2700hex is for single parameters of CI581-CN / CI582-CN itself.

Objects 2701hex to 270Ahex are used for connected expansion modules in ascending order.

This mode only works if fragmented SDO transfer is available.

The transferred data must have the following structure:



The length of the SDO is dependent on the number of transferred single parameters (CNT = the number of parameters to be request) - (up to 5 parameters can be transferred with one SDO).

Very important:
The CAN uses Big Endian.
This means:
If CNT or PARA_VALUE is 2, you must assign a value: 16#02000000 to them.

The length in bytes is calculated as follows:

Length (bytes) = 4bytes + (number of single parameters * 8bytes).

So the minimum length is 12 bytes and the maximum length is 44 bytes.

If the length or any other data inside the SDO isn´t correct, the CI device will reject the SDO with abort code 0800 0000hex.

For the used parameter indexes see Parameter Indexes of CI581-CN ↳ *Chapter 1.6.4.2.5.4.3.1.1 “Parameter indexes of CI581-CN” on page 5581* and Parameter Indexes of CI582-CN ↳ *Chapter 1.6.4.2.5.4.3.1.2 “Parameter indexes of CI582-CN” on page 5582*.

Parameter indexes of CI581-CN

Parameter	Parameter index	Parameter length
Error LED Failsafe	4	1
Check supply	27	1
Analogue data format	28	1
Input Delay	29	1
Short circuit detection	31	1
Behavior binary outputs at communication fault	32	1
Substitute value binary outputs	33	1
Overvoltage monitoring	34	1
Behavior analogue outputs at communication fault	35	1
Configuration AI0	36	1
Check AI0	37	1
Configuration AI1	38	1
Check AI1	39	1
Configuration AI2	40	1
Check AI2	41	1
Configuration AI3	42	1
Check AI3	43	1
Configuration AO0	44	1
Check AO3	45	1
Substitute Value AO0	46	2
Configuration AO1	47	1
Check AO1	48	1
Substitute Value AO1	49	2

Parameter indexes of CI582-CN

Parameter	Parameter index	Parameter length
Error LED Failsafe	4	1
Check supply	27	1
Input Delay	28	1
Short circuit detection	30	1
Behavior binary outputs at communication fault	31	1
Substitute value binary outputs	32	2
Voltage Feedback monitoring	33	1
Overvoltage monitoring	34	1

Block wise read of single parameterization (Object 2710hex to 271Ahex)

For reading of single parameters a special sequence is needed:

- Writing of SDO to the corresponding object with data structure defined in Parameter Indexes of CI581-CN.
  - This writing requests the parameter indexes that should be read.
- Reading of SDO of the corresponding object to get the read parameters.
  - The delivered data uses the same structure as defined in Parameter Indexes of CI582-CN.
  - If the reading of the data isn´t finished the device will reject the read service with abort code 08000021hex
  - If no writing has been send before reading the device will reject the read service with abort code 08000000hex

Eeprom download (Object 3100hex)

This functionality is only used during production. All requests to this object will be rejected.

Factory test mode (Object 5010hex)

This functionality is only used during production. All requests to this object will be rejected.

CI581-CN / CI582-CN in DS401 mode

Activation of DS401 mode in CI581-CN / CI582-CN

The DS401 mode inside the CI581-CN / CI582-CN can be activated using the hexadecimal address switches.

The DS401 mode is activated if the addresses switch position is greater than 80hex.

The node ID in the CANopen network is then calculated as follows:

Node ID = address switch 80hex.

PDO mapping in DS401 mode

The PDOs 1...4 are mapped as described in the DS401 specification. For additional PDOs (up to 20 PDOs per direction possible) there is no fixed mapping.

RxPDO mapping in DS401 mode

As defined by DS401 the first RxPDO only contains binary outputs. The first bytes contain the onboard binary outputs of the CI581-CN / CI582- CN devices. If binary I/O expansion modules are connected to the CI581-CN / CI582-CN these output data will also be put into RPDO1.

The second RxPDO contains only analog outputs (as defined by DS401). In case of CI581-CN the first two analog outputs are the onboard AOs. If analog I/O expansion modules are connected to the CI581-CN / CI582-CN these output data will also be put into RPDO2.

RxPDO 3 and RxPDO 4 are reserved for additional analog outputs (as defined by DS401) of connected analog I/O expansion modules.

For RxPDO 5&20 there is no fixed mapping. This mapping is user defined.

For details see following charts:

RxPDO mapping in DS401 mode for CI581-CN

RxPDO 1	
CI581 DO0 (1 byte, 8 channels)	I/O bus binary expansion DO1..DO7 (7 bytes, 56 channels) (only if attached to CI581-CN)

RxPDO 2		
CI581 AO0 (2 bytes, 1 channel)	CI581 AO1 (2 bytes, 1 channel)	I/O bus analog expansion AO2, AO3 (4 bytes, 2 channels) (only if attached to CI581-CN)

RxPDO 3
I/O bus analog expansion AO4..AO7 (8 bytes, 4 channels) (only if attached to CI581-CN)

RxPDO 4
I/O bus analog expansion AO8&AO11 (8 bytes, 4 channels) (only if attached to CI581-CN)

RxPDO mapping in DS401 mode for CI582-CN

RxPDO 1		
CI582 DO0 (1 byte, 8 channels)	CI582 DO1 (1 byte, 8 channels)	I/O bus binary expansion DO2..DO7 (6 bytes, 48 channels) (only if attached to CI582-CN)

RxPDO 2
I/O bus analog expansion AO0..AO3 (8 bytes, 4 channels) (only if attached to CI582-CN)

RxPDO 3
I/O bus analog expansion AO4..AO7 (8 bytes, 4 channels) (only if attached to CI582-CN)

RxPDO 4
I/O bus analog expansion AO8..AO11 (8 bytes, 4 channels) (only if attached to CI582-CN)

TxPDO mapping in DS401 mode

As defined by DS401 the first TxPDO only contains binary inputs. The first bytes contain the onboard binary inputs of the CI581-CN / CI582-CN devices. If binary I/O expansion modules are connected to the CI581-CN / CI582-CN these input data will also be put into TPDO1.

The second TxPDO contains only analog inputs (as defined by DS401). In case of CI581-CN the first four analog inputs are the onboard AIs. If analog I/O expansion modules are connected to the CI581-CN / CI582-CN these inputs data will also be put into TPDO2.

TxPDO 3 and TxPDO 4 are reserved for additional analog inputs (as defined by DS401) of connected analog I/O expansion modules.

For TxPDO 5...20 there is no fixed mapping. This mapping is user defined.

For details see following charts:

TxPDO mapping in DS401 mode for CI581-CN

TxPDO 1	
CI581 DI0 (1 byte, 8 channels)	I/O bus binary expansion DI1..DI7 (7 bytes, 56 channels) (only if attached to CI581-CN)

TxPDO 2			
CI581 AI0 (2 bytes, 1 channel)	CI581 AI1 (2 bytes, 1 channel)	CI581 AI2 (2 bytes, 1 channel)	CI581 AI3 (2 bytes, 1 channel)

TxPDO 3
I/O bus analog expansion AI4..AI7 (8 bytes, 4 channels) (only if attached to CI581-CN)

TxPDO 4
I/O bus analog expansion AI8..AI11 (8 bytes, 4 channels) (only if attached to CI581-CN)

TxPDO mapping in DS401 mode for CI582-CN

TxPDO 1		
CI582 DI0 (1 byte, 8 channels)	CI582 DI1 (1 byte, 8 channels)	I/O bus binary expansion DI2..DI7 (6 bytes, 48 channels) (only if attached to CI582-CN)

TxPDO 2
I/O bus analog expansion AI0..AI3 (8 bytes, 4 channels) (only if attached to CI582-CN)

TxPDO 3
I/O bus analog expansion AI4..AI7 (8 bytes, 4 channels) (only if attached to CI582-CN)

TxPDO 4
I/O bus analog expansion AI8..AI11 (8 bytes, 4 channels) (only if attached to CI582-CN)

Parameterization of CI581-CN / CI582-CN in DS401 mode

In DS401 mode the CI581-CN / CI582-CN are parameterized during startup with 0 parameters that means all parameter values are set to 0. For description of parameters see the corresponding device description.

The parameters can be changed during run time using the single parameterization feature.

The two different single parameterization modes are described below.

In single parameterization the parameters are addressed by a parameter index which can be found in the tables below.

Block wise single parameterization (Object 2700hex to 270Ahex)

See Block Wise Single Parameterization (Object 2700hex to 270Ahex) for detail.

Single parameterization without fragmented SDOs (Object 2720hex to Object 272Ahex)

For systems without the capability to use fragmented SDOs there is a second method of single parameterization. This method only uses non-fragmented SODs.

The objects 2720hex to 272Ahex are used for this mode. The sub object here is the parameter index. The length is the parameter length. Object 2720hex is for single parameters of CI581-CN / CI582-CN itself. Objects 2721hex to 272Ahex are used for connected expansion modules in ascending order.

For the used parameter indexes see Parameter Indexes of CI581-CN and Parameter Indexes of CI582-CN.

Limitations for CI581-CN / CI582-CN in DS401 mode

- Fast counter functionality is not available in DS401 mode (except CD522).
- Single Parameterization of I/O modules is only supported for I/O modules as of firmware version 3.x.
- The following restrictions apply for connected I/O modules.

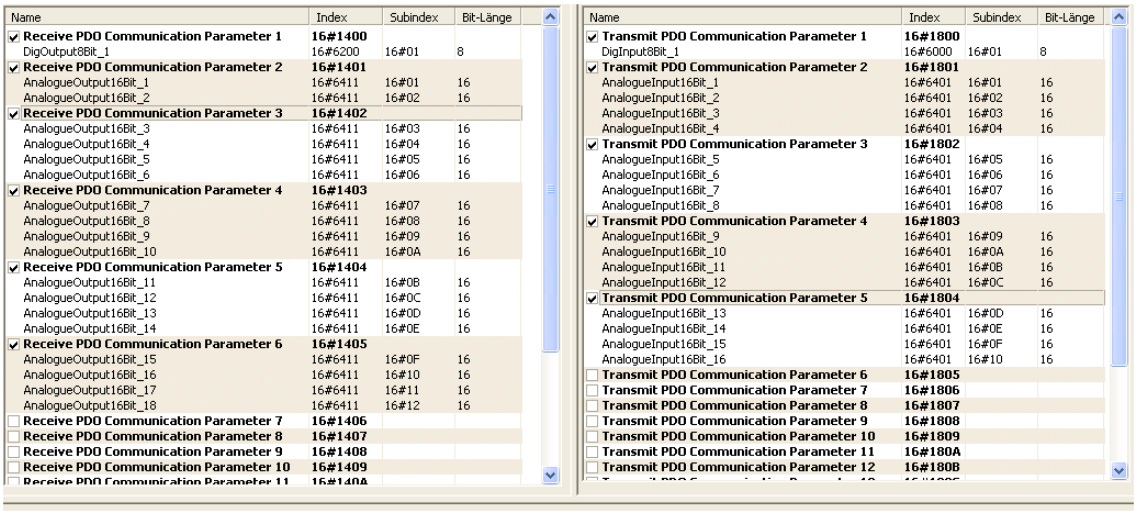
DA501

- The DA501 module only reports analog IOs even though it has also binary inputs / outputs.
- It uses the following IO data structure:
  Inputs:
  - Digital Inputs (channel 0..15) (16bits, 2 bytes)
  - Analog Input 0 (2 bytes)
  - Analog Input 1 (2 bytes)
  - Analog Input 2 (2 bytes)
  - Analog Input 3 (2 bytes)
  - Binary Inputs (channel 16..23) / reserved (16bits, 2 bytes)
    bits 0..7 binary inputs
    bits 8..15 reserved
    Outputs:
  - Analog Output 0 (2 bytes)
  - Analog Output 1 (2 bytes)
  - Reserved (2 bytes)
  - Binary Outputs (channel 0..7) / reserved (16bits, 2 bytes)
    bits 0..7 binary outputs
    bits 8..15 reserved

CD522

- The CD522 module only reports analog IOs even though it has also binary inputs / outputs.
- It uses the following IO data structure:
  Inputs
  - State S0/S1 % pulse (2 bytes)
  - State byte / inputs counter 0 (2 bytes)
  - TOUCH counter 0 value high word (2 bytes)
  - TOUCH counter 0 value low word (2 bytes)
  - 32-bit counter 0 high word (2 bytes)
  - 32-bit counter 0 low word (2 bytes)
  - 32-bit counter 1 high word (2 bytes)
  - 32-bit counter 1 low word (2 bytes)
  - Reserved (2 bytes)
  - State byte / inputs counter 1 (2 bytes)
  - TOUCH counter 1 value high word (2 bytes)
  - TOUCH counter 1 value low word (2 bytes)

  Outputs
  - PWM frequency 0 high-word (2 bytes)
  - PWM duty/cycle/pulse 0 low word (2 bytes)
  - PWM control byte C0 / reserved (2 bytes)
  - Reserved (2 bytes)
  - PWM frequency 1 high-word (2 bytes)
  - PWM duty/cycle/pulse 1 low word (2 bytes)
  - PWM control byte C1 / outputs (2 bytes)
  - Reserved (2 bytes)
  - Counter 0 settings high word (2 bytes)
  - Counter 0 settings low word (2 bytes)
  - Counter 0 control byte / counter (2 bytes)
  - Reserved (2 bytes)
  - Counter 1 settings high word (2 bytes)
  - Counter 1 settings low word (2 bytes)
  - Counter 1 control byte / counter (2 bytes)
  - Reserved (2 bytes)
- The IO data of the CD522 have to be mapped continuously in ascending order (without gaps or other IO data inbetween) beginning with a new PDO.
  For an example see the following figure:
  - AnalogueInput16Bit_5 to AnalogueInput16Bit_16 are the inputs of the CD522.
  - AnalogueOutput16Bit_3 to AnalogueOutput16Bit_18 are the outputs of the CD522.



Name	Index	Subindex	Bit-Länge
Receive PDO Communication Parameter 1	**16#1400**		
DigOutput8Bit_1	16#6200	16#01	8
Receive PDO Communication Parameter 2	**16#1401**		
AnalogueOutput16Bit_1	16#6411	16#01	16
AnalogueOutput16Bit_2	16#6411	16#02	16
Receive PDO Communication Parameter 3	**16#1402**		
AnalogueOutput16Bit_3	16#6411	16#03	16
AnalogueOutput16Bit_4	16#6411	16#04	16
AnalogueOutput16Bit_5	16#6411	16#05	16
AnalogueOutput16Bit_6	16#6411	16#06	16
Receive PDO Communication Parameter 4	**16#1403**		
AnalogueOutput16Bit_7	16#6411	16#07	16
AnalogueOutput16Bit_8	16#6411	16#08	16
AnalogueOutput16Bit_9	16#6411	16#09	16
AnalogueOutput16Bit_10	16#6411	16#0A	16
Receive PDO Communication Parameter 5	**16#1404**		
AnalogueOutput16Bit_11	16#6411	16#0B	16
AnalogueOutput16Bit_12	16#6411	16#0C	16
AnalogueOutput16Bit_13	16#6411	16#0D	16
AnalogueOutput16Bit_14	16#6411	16#0E	16
Receive PDO Communication Parameter 6	**16#1405**		
AnalogueOutput16Bit_15	16#6411	16#0F	16
AnalogueOutput16Bit_16	16#6411	16#10	16
AnalogueOutput16Bit_17	16#6411	16#11	16
AnalogueOutput16Bit_18	16#6411	16#12	16
Receive PDO Communication Parameter 7	**16#1406**		
Receive PDO Communication Parameter 8	**16#1407**		
Receive PDO Communication Parameter 9	**16#1408**		
Receive PDO Communication Parameter 10	**16#1409**		
Receive PDO Communication Parameter 11	**16#140A**		

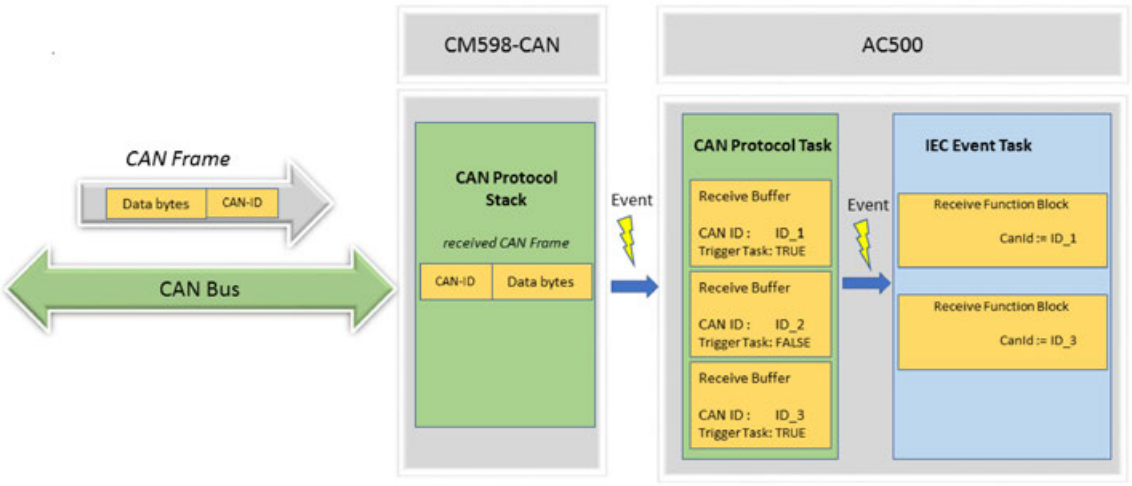
Name	Index	Subindex	Bit-Länge
Transmit PDO Communication Parameter 1	**16#1800**		
DigInput8Bit_1	16#6000	16#01	8
Transmit PDO Communication Parameter 2	**16#1801**		
AnalogueInput16Bit_1	16#6401	16#01	16
AnalogueInput16Bit_2	16#6401	16#02	16
AnalogueInput16Bit_3	16#6401	16#03	16
AnalogueInput16Bit_4	16#6401	16#04	16
Transmit PDO Communication Parameter 3	**16#1802**		
AnalogueInput16Bit_5	16#6401	16#05	16
AnalogueInput16Bit_6	16#6401	16#06	16
AnalogueInput16Bit_7	16#6401	16#07	16
AnalogueInput16Bit_8	16#6401	16#08	16
Transmit PDO Communication Parameter 4	**16#1803**		
AnalogueInput16Bit_9	16#6401	16#09	16
AnalogueInput16Bit_10	16#6401	16#0A	16
AnalogueInput16Bit_11	16#6401	16#0B	16
AnalogueInput16Bit_12	16#6401	16#0C	16
Transmit PDO Communication Parameter 5	**16#1804**		
AnalogueInput16Bit_13	16#6401	16#0D	16
AnalogueInput16Bit_14	16#6401	16#0E	16
AnalogueInput16Bit_15	16#6401	16#0F	16
AnalogueInput16Bit_16	16#6401	16#10	16
Transmit PDO Communication Parameter 6	**16#1805**		
Transmit PDO Communication Parameter 7	**16#1806**		
Transmit PDO Communication Parameter 8	**16#1807**		
Transmit PDO Communication Parameter 9	**16#1808**		
Transmit PDO Communication Parameter 10	**16#1809**		
Transmit PDO Communication Parameter 11	**16#180A**		
Transmit PDO Communication Parameter 12	**16#180B**		

Triggering of event tasks with CAN-IDs

For CM598-CAN module the execution of a PLC application task can be triggered automatically by a certain event, i.e. by incoming CAN 2.0 A or CAN 2.0 B frames. For this, the PLC application task is to be configured as external event task.

Prerequisites

- *PLC firmware version 2.5.3 and Automation Builder as of version 1.2.2.*
- *For CAN 2.0. A and CAN 2.0 B a seperate PLC application task is required.*
- *Triggering of event tasks is only supported for the communication module CM598-CAN.*



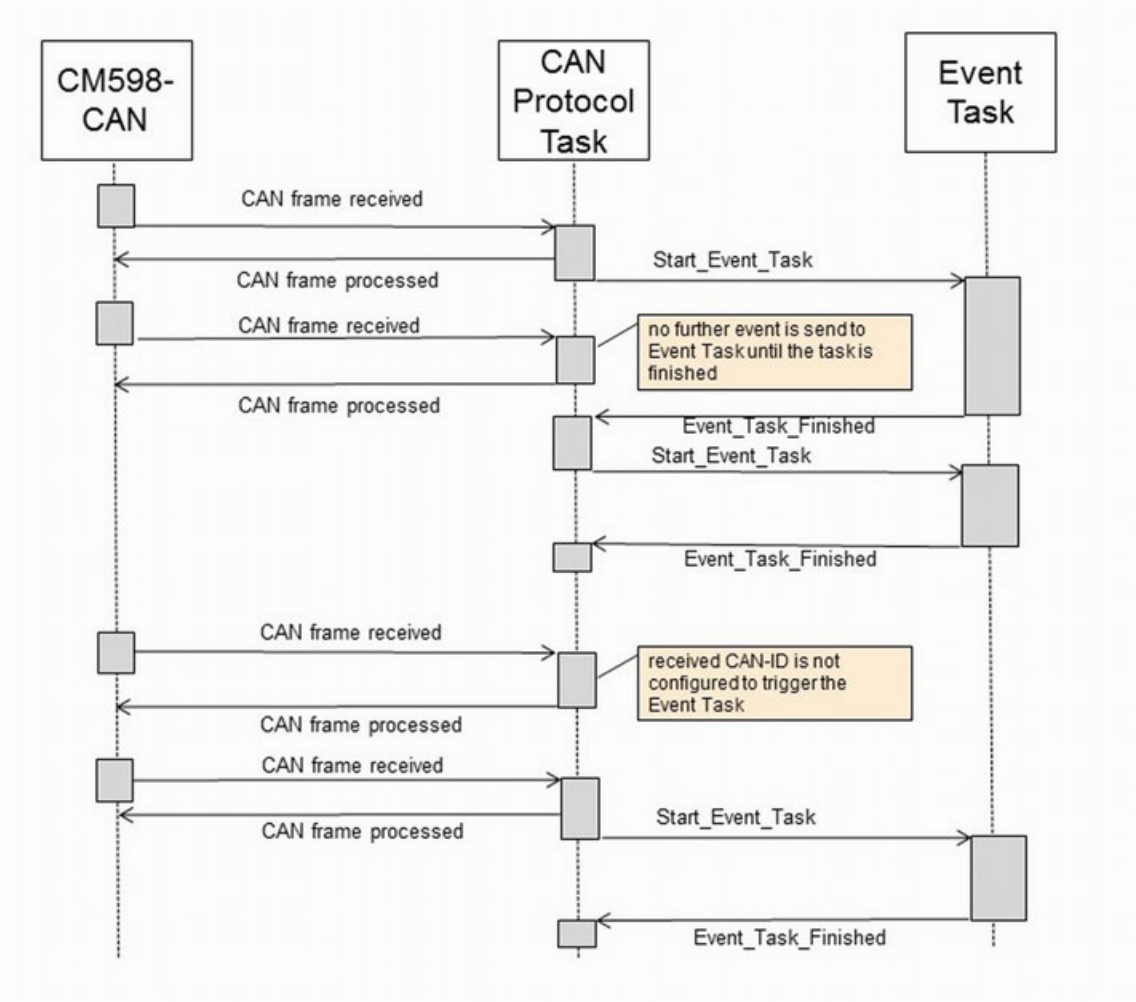
Every incoming CAN frame on a CM598-CAN module processes an event in the AC500 PLC. If the parameter "Trigger PLC Task" is set to TRUE, the CAN protocol task checks via the receive buffer configuration and the corresponding CAN-ID of the CAN frame whether a CAN frame is to be executed or not. Only those CAN-IDs that are configured in the protocol configuration will be processed. All other CAN frames will be rejected. If a CAN frame is to be processed, the CAN frame data is copied to the receive buffer and an event on the IEC event task is triggered.

The IEC event task will be executed for one cycle.

The IEC event task will be triggered continuously until all associated receive buffers have been emptied. Hence, ensure that the buffers are emptied by the task, otherwise the task will run into a loop.

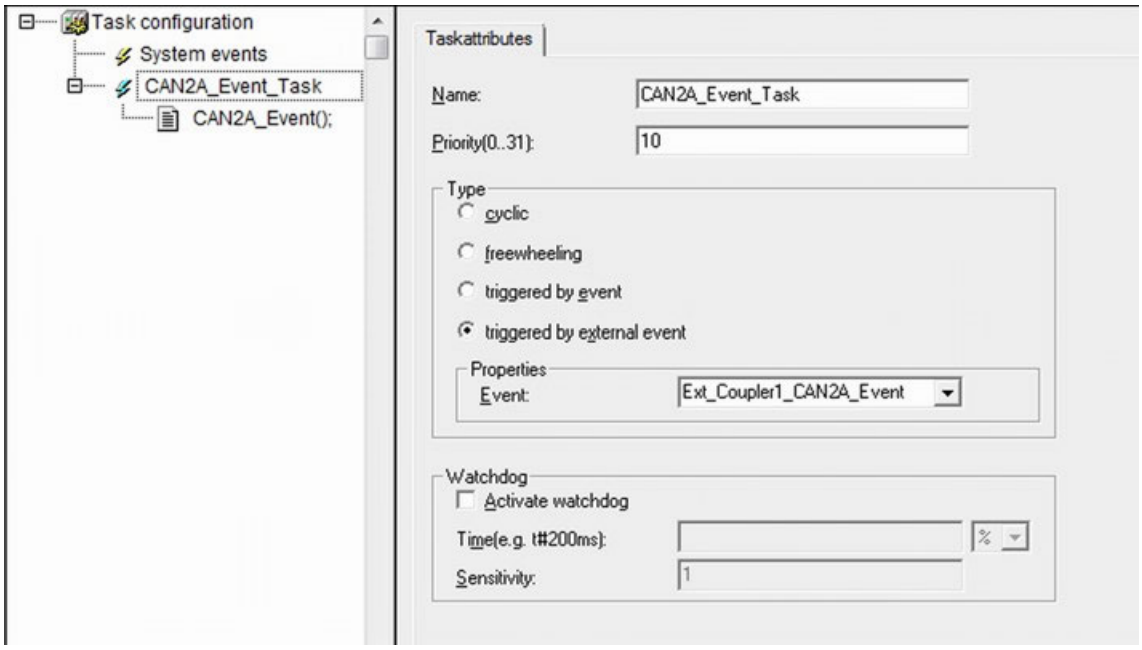
CAN frame processing

The following figure shows the sequence CAN frames processing when the triggering of event task is used.


CM598-CAN
CAN Protocol Task
Event Task
CAN frame received
CAN frame processed
Start_Event_Task
CAN frame received
no further event is send to Event Task until the task is finished
CAN frame processed
Event_Task_Finished
Start_Event_Task
Event_Task_Finished
CAN frame received
received CAN-ID is not configured to trigger the Event Task
CAN frame processed
CAN frame received
CAN frame processed
Start_Event_Task
Event_Task_Finished

- *One event task can be assigned to each of the protocols CAN 2.0 A and CAN 2.0 B.*
- *One common event for all of the selected CAN-IDs of one protocol. It must be evaluated which CAN-IDs have been received.*
- *It is possible that CAN frames are lost when necessary system resources are in use or when the CAN frames could not be processed in time due to high system load. So, the PLC application must monitor the task which consumes the events of the CAN protocol with a watchdog mechanism or something similar.*
- *During the execution of the PLC application task no further events are sent when new CAN frames are received before the task is finished. It is even possible to read newly received CAN frames in the current cycle of the PLC application task.*
- *The order of CAN frame reception could not be determined in the PLC application task.*
- *The task which should be triggered by the protocols CAN2A and CAN2B is a normal PLC application task and will be scheduled according its priority. So it is possible that events could not be processed on time due to the task is blocked by other tasks in the system.*
- *The setting of the parameter “behavior on receive buffer overflow” has an influence on the processing of CAN frames and the corresponding events. The documentation of the respective value has to be considered.*
- *CAN frames and events may be lost, if the receive buffer is full. Take care that the receive buffers are emtied in time and the PLC load and CAN bus load are not too high.*

Event task configuration



Event	Communication module slot	Description
Int_Coupler0_CAN2A_Event	0	PLC event task will be triggered when a CAN frame with a specified CAN ID was received by the CAN2A protocol running at communication module slot 0.
Int_Coupler0_CAN2B_Event	0	PLC event task will be triggered when a CAN frame with a specified CAN ID was received by the CAN2B protocol running at communication module slot 0.
Ext_Coupler<slot>_CAN2A_Event	1 ... 6	PLC event task will be triggered when a CAN frame with a specified CAN ID was received by the CAN2A protocol running at communication module slot 1 - 6.
Ext_Coupler<slot>_CAN2B_Event	1 ... 6	PLC event task will be triggered when a CAN frame with a specified CAN ID was received by the CAN2B protocol running at communication module slot 1 - 6.

Diagnosis

CANopen communication errors are indicated by the Communication Module LEDs. Malfunctions of the CANopen driver or the Communication Module itself are indicated by the corresponding error class in the PLC. Furthermore, the CANopen library provides different function blocks which allow detailed error diagnosis (refer to CANopen library ↳ *Chapter 1.5.4.7 “CANopen library” on page 912*).

Error messages

Error messages/emergency messages are used to signalize device errors. An emergency message contains a code that clearly identifies the error (specified in the communication profile DS-301 and in the individual device profiles DS-40x). The following table shows some of the available error codes. Emergency messages are automatically sent by all CANopen modules.

Error code (hex)	Description / error cause
00xx	Error on reset or no error
10xx	General error
20xx	Current error
21xx	- Error on device input side
22xx	- Error inside the device
23xx	- Error on device output side
30xx	Voltage error
31xx	- Supply voltage error
32xx	- Error inside the device
33xx	- Error on device output side
40xx	Temperature error
41xx	- Ambient temperature
42xx	- Temperature inside the device

Error code (hex)	Description / error cause
50xx	Hardware error in the device
60xx	Software error in the device
61xx	- Device-internal software
62xx	- Application software
63xx	- Data
70xx	Error in additional modules
80xx	Monitoring
81xx	Communication
90xx	External error
F0xx	Error of additional functions
FFxx	Device-specific errors

Function blocks

CANopen_AC500_V25.lib or CANopen_AC500_V11.lib

Group: CAN 2.0A	
Chapter 1.5.4.7.1.1 “CAN2A_INFO” on page 913	Reading information about CAN 2.0A communication
Chapter 1.5.4.7.1.2 “CAN2A_REC” on page 916	Reading CAN 2.0A telegrams (with 11 bit identifier) from a receive buffer
Chapter 1.5.4.7.1.3 “CAN2A_SEND” on page 919	Transmitting CAN 2.0A telegrams (with 11 bit identifier)

Group: CANopen master / NMT controller	
Chapter 1.5.4.7.1.7 “CANOM_NMT” on page 931	Controlling NMT node states via network management

Group: CANopen master / Status / Diagnosis	
Chapter 1.5.4.7.1.8 “CANOM_NODE_DIAG” on page 933	Polling diagnosis data from a slave
Chapter 1.5.4.7.1.10 “CANOM_RES_ERR” on page 941	Resetting the Communication Module's error indications
Chapter 1.5.4.7.1.14 “CANOM_STATE ” on page 952	Reading the CANopen Communication Module status
Chapter 1.5.4.7.1.15 “CANOM_SYS_DIAG” on page 957	Displaying status surveys of all slaves

Group: SDO parameters	
Chapter 1.5.4.7.1.11 “CANOM_SDO_READ” on page 944	Reading the value of a slave object
Chapter 1.5.4.7.1.12 “CANOM_SDO_WRITE” on page 947	Writing the value of a slave object

1.6.4.2.6 Serial communication module

CM574-RS - Serial communication module for AC500

Overview

The capabilities of the communication module correspond to a processor module. Further information on the processor module can be found in the corresponding device description *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*.

The communication module can be plugged into any slot for external communication modules on a AC500 terminal base. Up to four CM574-RS modules can be used in one AC500 system.

Programming can be done either directly via a serial cable between PC and serial interface of the communication module, or routed via the AC500 CPU using the serial interface or Ethernet or ARCNET.

Operation Modes

- CM574 is an additional CPU and program code to handle the serial interfaces run in CM574.
  Communication with the main PLC can be done in two ways:
  – Cyclic data exchange via Inputs/outputs *Chapter 1.6.4.2.6.2.6.2 “Cyclic data exchange CM574-RS/CS31 bus <-> AC500 CPU” on page 5597*
  – Acyclic data exchange via messages with FB’s from the user program *Chapter 1.6.4.2.6.3.1 “Function blocks for acyclic data exchange CM574-RS/AC500 CPU” on page 5607*
- CM574 work in shared mode. That mean module is a Interface extension (2 additional Serial Interfaces) and all the needed codes and configurations are running in the main PLC.

Option 1 reduces CPU Load in the main PLC, but communication between CM574 and main PLC must be done manually and especially for Modbus the use of the %M Area has to be considered.

Option 2 is easy to program and configure. Only configuration for shared mode must be download to the CM574. Configuration and code for the serial interfaces will be handled in the main PLC. Load for main PLC is higher.

User program size and operands of the CM574-RS

Values The following table shows the values set for program memory and operands in the CM574 target systems:

Parameter	Value
User program (code), see note 1	256 kB
Number of POUs	1024 kB
Number of tasks	3
Floating point processor, see note 2	no
Global and local variables: VAR or VAR GLOBAL	128 kB
Addressable flag area: VAR AT %Mx.y	128 kB
Persistent area: VAR AT %Rx.y	0 kB
Inputs %I, see note 4	4 kB
Outputs %Q, see note 4	4 kB
FLASH for user data	2 x 64 kB = 128 kB

Note 1 The user program is composed of:

- The compiled code of all POUs called in the program

- The initialization code for the variables

The configuration data are not included in the user program size.

Remark 2 All AC500 processor modules can perform floating point operations. For processor modules without floating point processor (like the CM574-RS), these operations are performed by an emulation library and are therefore slower. Emulation is faster for LREAL variables than for REAL variables. Thus, the use of LREAL variables is recommended.

Remark 3 The information shown in the message box exclusively contains the retain data of the RETAIN area, and not the variables of the addressable flag area %Mx.y. that are declared as VAR RETAIN.

The communication module CM574-RS is not equipped with a battery. For this reason, ALL operands are initialized once the control voltage is switched on.

In the communication module, data can be stored fail-safe in the flash memory using the blocks *Chapter 1.5.4.19.2.18 “FLASH_WRITE” on page 1547* and *Chapter 1.5.4.19.2.17 “FLASH_READ” on page 1544*.

Remark 4 For the inputs and the outputs, the same rules apply as for the AC500-CPUs.

The following input and output assignment applies for the CM574-RS:

CPU communication:	%IB0 ..	%IB999	and %QB0 ..	%QB999
Of this: Channel 1:	%IB0	%IB499	and %QB0 ..	%QB499
Channel 2:	%IB500 ..	%IB999	and %QB500 ..	%QB999
COM1:	%IB1000 ..	%IB1999	and %QB1000 ..	%QB1999
COM2 :	%IB2000 ..	%IB2999	and %QB2000 ..	%QB2999

Additional information can be found in the documentation "System Technology of the CPUs" in AC500 inputs, outputs and flags.

Connection and transmission media

Possibilities of connection The pin assignment of the serial interfaces COM1 and COM2 correspond to those on the terminal base for the processor modules PM57x, PM58x and PM59x *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*.

Information about connection options for the serial interfaces can be found in the system assembly chapter *Chapter 1.6.3.6.4 “Connection and wiring” on page 5337* .

Connection cables The programming cable for direct online communication between PC and CM574-COMx has the same pin assignment as the programming cable *Chapter 1.6.2.9.2.10 “TK502 - Programming cable” on page 5188* for the serial interface COM1 of PM57x, PM58x and PM59x.

Protocols of the serial interfaces of the CM574-RS

Protocol 'COMx - Online access'

This protocol can only be used, if CM574 is configured as a PLC. If configured as communication module within a PLC project, this protocol is not available.

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.2 “Setting COMx - Online access” on page 6099*.

If the transmisson rate of the interface is changed to another value, a corresponding gateway channel must be created. Example: COM1 is to be operated with 115200 baud and the protocol 'Online access'.

1. Configuration of COM1 -> Setting the transmission rate to 115200 baud.
2. Selection of the gateway channel -> e.g. COM4_19200 with transmission rate 19200 baud.
3. Login and download of user defined project (if required, create boot project).
4. Logout -> Interface is changed to 115200 baud.
5. Selection of the gateway channel -> e.g. COM4_115kB with transmission rate 115200 baud.
6. Login -> with 115200 baud.

Protocol 'COMx - ASCII'

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.3 “Setting COMx - ASCII” on page 6100*.

Protocol 'COMx - Modbus'

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*.

The address assignment as well as a detailed description of all implemented Modbus functions can be found in the chapter of Communication with Modbus RTU .

In the CM574-RS, 128 kB data are available in the addressable flag range %M. This way, the entire Modbus address range 0000hex .. FFFFhex can be covered.

Protocol 'COMx - SysLibCom'

This protocol can only be used, if CM574 is configured as a PLC. If configured as Communication Module within a PLC project, this protocol is not available.

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.6 “Setting COMx - SysLibCom” on page 6110*.

Protocol 'COMx - Multi'

This protocol can only be used, if CM574 is configured as a PLC. If configured as Communication Module within a PLC project, this protocol is not available.

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.7 “Setting COMx - Multi” on page 6114*.

Protocol 'COM1 - CS31 Bus'

See protocol description for the serial interfaces COM1 and COM2 ↳ *Chapter 1.6.5.2.11.5 “Setting COMx - CS31” on page 6110*.

There are several options for processing the inputs and outputs of the CS31 bus modules:

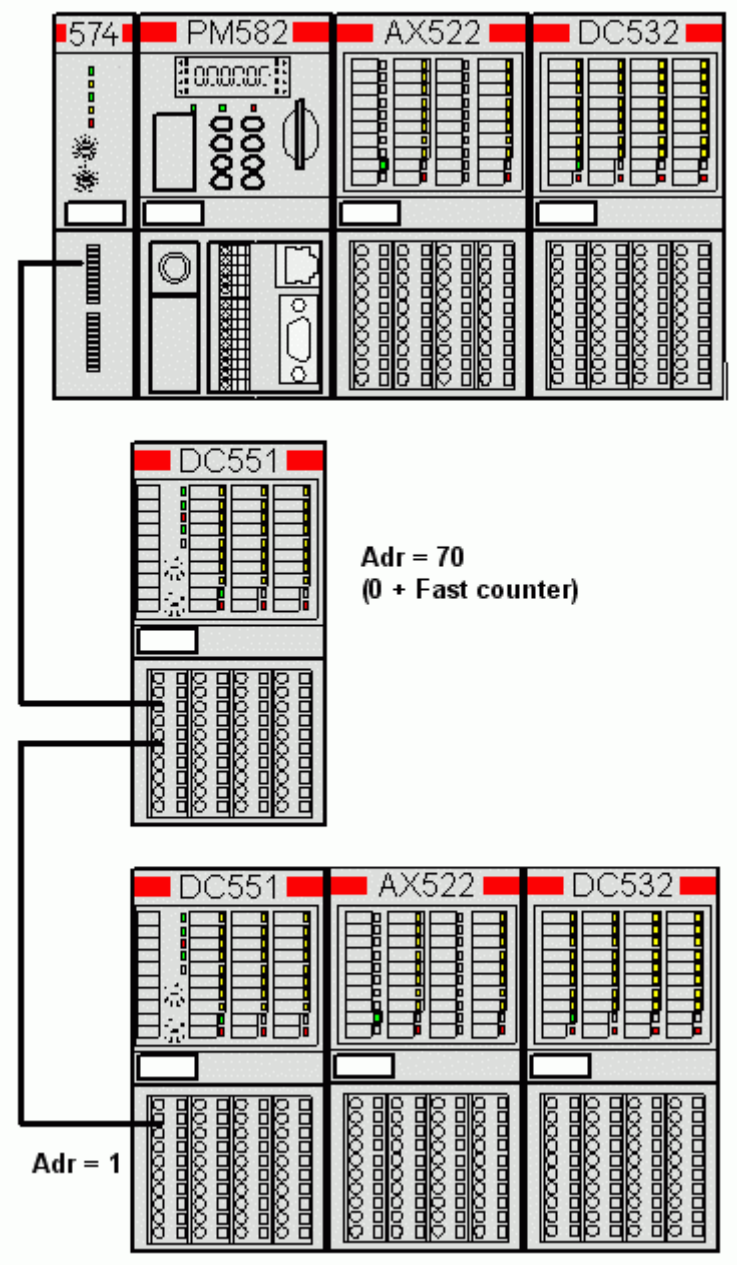
- Direct processing in the user-defined program of the CM574-RS
- Transfer to the AC500 CPU via cyclic data exchange

Transfer to the AC500 CPU via acyclic data exchange is not recommended.

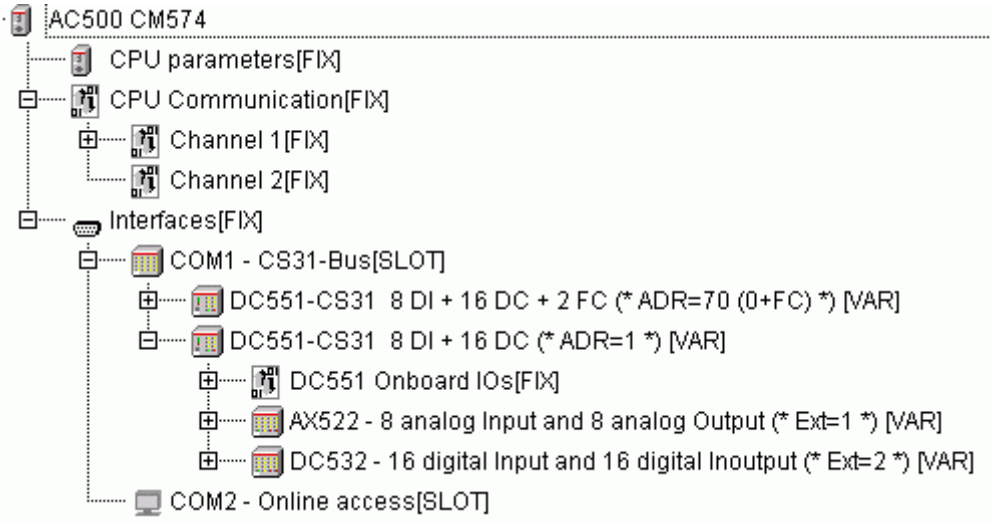
Planning example for CM574-RS/CS31 bus

Planning the cyclic data exchange of the inputs and outputs of the CS31 bus modules on the CM574-RS with a processor module is illustrated in the following example configuration:

- PM582 with CM574-RS in slot 1
- CM574-RS: COM1 CS31 bus, COM2 online access
- CS31 cluster 1: DC551 with HW address switch = 70, i.e. address 0 and fast counter enabled
- CS31 cluster 2: DC551 with HW address switch = 1 and S500 extensions 1xAX522 + 1xDC532



Accordingly, PLC configuration in the CM574-RS project is as follows:



If you open the last module (here: DC532), you can see that the highest input byte is %IB1037, and the highest output byte is %QB1041.

Thus, the configuration on the CS31 bus has

38 bytes for inputs and 42 bytes for outputs.

Cyclic data exchange CM574-RS/CS31 bus <-> AC500 CPU

In case of cyclic data exchange, the inputs and outputs of the modules on the CS31 bus are transferred by means of I/O modules via the DPRAM.

Cyclic data exchange is described in detail in Function blocks for acyclic data exchange CM574-RS/AC500 CPU ↳ *Chapter 1.6.4.2.6.3 “Acyclic data exchange CM574/AC500 CPU” on page 5607*.

Transfer of the inputs

For the transfer to the CPU, the inputs of the CS31 modules are outputs of the CM574-RS in the CPU communication. For this reason, the input modules in the CM574-RS include IEC outputs (%Q).

The following table shows how the inputs of the CS31 modules are transferred and where the respective operands are located in the CPU:

Inputs CS31 bus	**CM574 - COM1 CS31 bus**			**CM574 - DPR**			**AC500 CPU**		
	%IB	**%IW**	**%ID**	**%QB**	**%QW**	**%QD**	**%IB**	**%IW**	**%ID**
DC551_0 Input 0-7	1000	500	250	0	0	0	1.0	1.0	1.0
Not used	1001			1			1.1		
DC551_0 Input 8-15	1002	501		2	1		1.2	1.1	

Inputs CS31 bus	**CM574 - COM1** **CS31 bus**			**CM574 - DPR**			**AC500 CPU**		
	%IB	**%IW**	**%ID**	**%QB**	**%QW**	**%QD**	**%IB**	**%IW**	**%ID**
DC551_0 Input 16-23	1003			3			1.3		
				Module 4 Byte Input			Module 4 Byte Input		
DC551_0 Actual value 1 (HH)	1004	502	251	4	2	1	1.4	1.2	1.1
DC551_0 Actual value 1 (H)	1005			5			1.5		
DC551_0 Actual value 1 (L)	1006	503		6	3		1.6	1.3	
DC551_0 Actual value 1 (LL)	1007			7			1.7		
DC551_0 Actual value 2 (HH)	1008	504	252	8	4	2	1.8	1.4	1.2
DC551_0 Actual value 2 (H)	1009			9			1.9		
DC551_0 Actual value 2 (L)	1010	505		10	5		1.10	1.5	
DC551_0 Actual value 2 (LL)	1011			11			1.11		
				2 x Module 1 DWORD Input			2 x Module 1 DWORD Input		
DC551_0 Status byte 1	1012	506	253	12	6	3	1.12	1.6	1.3

Inputs CS31 bus	**CM574 - COM1** **CS31 bus**			**CM574 - DPR**			**AC500 CPU**		
	%IB	**%IW**	**%ID**	**%QB**	**%QW**	**%QD**	**%IB**	**%IW**	**%ID**
DC551_0 Status byte 2	1013			13			1.13		
				2 x 1 Byte Input			2 x 1 Byte Input		
DC551_1 Input 0-7	1014	507		14	7	3	1.14	1.7	1.3
DC551_1 Input 8-15	1015			15			1.15		
DC551_1 Input 16-23	1016	508	254	16	8	4	1.16	1.8	1.4
Not used	1017			17			1.17		
DC551_1_AX522 Anal. input 0 (H)	1018	509		18	9		1.18	1.9	
DC551_1_AX522 Anal. input 0 (L)	1019			19			1.19		
DC551_1_AX522 Anal. input 1 (H)	1020	510	255	20	10	5	1.20	1.10	1.5
DC551_1_AX522 Anal. input 1 (L)	1021			21			1.21		
DC551_1_AX522 Anal. input 2 (H)	1022	511		22	11		1.22	1.11	

Inputs CS31 bus	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	%IB	%IW	%ID	%QB	%QW	%QD	%IB	%IW	%ID
DC551_1_AX522 Anal. input 2 (L)	1023			23			1.23		
DC551_1_AX522 Anal. input 3 (H)	1024	512	256	24	12	6	1.24	1.12	1.6
DC551_1_AX522 Anal. input 3 (L)	1025			25			1.25		
DC551_1_AX522 Anal. input 4 (H)	1026	513		26	13		1.26	1.13	
DC551_1_AX522 Anal. input 4 (L)	1027			27			1.27		
DC551_1_AX522 Anal. input 5 (H)	1028	514	257	28	14	7	1.28	1.14	1.7
DC551_1_AX522 Anal. input 5 (L)	1029			29			1.29		
DC551_1_AX522 Anal. input 6 (H)	1030	515		30	15		1.30	1.15	
DC551_1_AX522 Anal. input 6 (L)	1031			31			1.31		

Inputs CS31 bus	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	%IB	%IW	%ID	%QB	%QW	%QD	%IB	%IW	%ID
DC551_1_AX522 Anal. input 7 (H)	1032	516	258	32	16	8	1.32	1.16	1.8
DC551_1_AX522 Anal. input 7 (L)	1033			33			1.33		
DC551_1_DC532 Input 0-7	1034	517		34	17		1.34	1.17	
DC551_1_DC532 Input 8-15	1035			35			1.35		
DC551_1_DC532 Input 16-23	1036	518	259	36	18	9	1.36	1.18	1.9
DC551_1_DC532 Input 24-31	1037			37			1.37		
				3 x 4 Word Input			3 x 4 Word Input		
Not used	1038	519		38	19	9	1.38	1.19	1.9
Not used	1039			39			1.39		

Transfer of the outputs

For the transfer to the CPU, the outputs of the CS31 modules are inputs of the CM574-RS in the CPU communication. For this reason, the output modules in the CM574-RS include IEC outputs (%Q).

Table 709: Transferring the outputs of the CS31 modules:

CS31 bus out-puts	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	QIB	%QW	%QD	%IB	%IW	%ID	%QB	%QW	QID
DC551_0 Out-puts 8-15	1000	500	250	0	0	0	1.0	1.0	1.0
DC551_0 Out-puts 16-23	1001			1			1.1		
Not used	1002	501		2	1		1.2	1.1	
Not used	1003			3			1.3		
				Module 4 Byte Output			Module 4 Byte Output		
DC551_0 Start value 1 (HH)	1004	502	251	4	2	1	1.4	1.2	1.1
DC551_0 Start value 1 (H)	1005			5			1.5		
DC551_0 Start value 1 (L)	1006	503		6	3		1.6	1.3	
DC551_0 Start value 1 (LL)	1007			7			1.7		
DC551_0 End value 1 (HH)	1008	504	252	8	4	2	1.8	1.4	1.2
DC551_0 End value 1 (H)	1009			9			1.9		
DC551_0 End value 1 (L)	1010	505		10	5		1.10	1.5	
DC551_0 End value 1 (LL)	1011			11			1.11		
DC551_0 Start value 2 (HH)	1012	506	253	12	6	3	1.12	1.6	1.3

CS31 bus out-puts	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	QIB	%QW	%QD	%IB	%IW	%ID	%QB	%QW	QID
DC551_0 Start value 2 (H)	1013			13			1.13		
DC551_0 Start value 2 (L)	1014	507		14	7		1.14	1.7	
DC551_0 Start value 2 (LL)	1015			15			1.15		
DC551_0 End value 2 (HH)	1016	508	254	16	8	4	1.16	1.8	1.4
DC551_0 End value 2 (H)	1017			17			1.17		
DC551_0 End value 2 (L)	1018	509		18	9		1.18	1.9	
DC551_0 End value 2 (LL)	1019			19			1.19		
				Module 4 DWORD Output			Module 4 DWORD Output		
DC551_0 Control byte 1	1020	510	255	20	10	5	1.20	1.10	1.5
DC551_0 Control byte 2	1021			21			1.21		
DC551_1 Outputs 8-15	1022	511		22	11		1.22	1.11	
DC551_1 Outputs 16-23	1023			23			1.23		
				Module 4 Byte Output			Module 4 Byte Output		
DC551_1_AX522 Anal. output 0 (H)	1024	512	256	24	12	6	1.24	1.12	1.6

CS31 bus out-puts	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	QIB	%QW	%QD	%IB	%IW	%ID	%QB	%QW	QID
DC551_1_AX522 Anal. output 0 (L)	1025			25			1.25		
DC551_1_AX522 Anal. output 1 (H)	1026	513		26	13		1.26	1.13	
DC551_1_AX522 Anal. output 1 (L)	1027			27			1.27		
DC551_1_AX522 Anal. output 2 (H)	1028	514	257	28	14	7	1.28	1.14	1.7
DC551_1_AX522 Anal. output 2 (L)	1029			29			1.29		
DC551_1_AX522 Anal. output 3 (H)	1030	515		30	15		1.30	1.15	
DC551_1_AX522 Anal. output 3 (L)	1031			31			1.31		
DC551_1_AX522 Anal. output 4 (H)	1032	516	258	32	16	8	1.32	1.16	1.8
DC551_1_AX522 Anal. output 4 (L)	1033			33			1.33		

CS31 bus out-puts	CM574 - COM1 CS31 bus			CM574 - DPR			AC500 CPU		
	QIB	%QW	%QD	%IB	%IW	%ID	%QB	%QW	QID
DC551_1_AX522 Anal. output 5 (H)	1034	517		34	17		1.34	1.17	
DC551_1_AX522 Anal. output 5 (L)	1035			35			1.35		
DC551_1_AX522 Anal. output 6 (H)	1036	518	259	36	18	9	1.36	1.18	1.9
DC551_1_AX522 Anal. output 6 (L)	1037			37			1.37		
DC551_1_AX522 Anal. output 7 (H)	1038	519		38	19		1.38	1.19	
DC551_1_AX522 Anal. output 7 (L)	1039			39			1.39		
				2 x Module 4 Word Output			2 x Module 4 Word Output		
DC551_1_DC532 Output 16-23	1040	520	260	40	20	10	1.40	1.20	1.10
DC551_1_DC532 Output 24-31	1041			41			1.41		
				Module 1 Word Output			Module 1 Word Output		
Not used	1042	521	260	42	21	10	1.42	1.21	1.10
Not used	1043			43			1.43		

In the CPU, the input modules include IEC inputs (%I) and the output modules include IEC outputs (%Q). Consequently, from the CPUs point of view, the inputs of the CS31 module are read as IEC input, and the outputs of the CS31 module are written as output.

The easiest method to transfer the inputs in the PLC program of the CM574-RS is the *Chapter 1.5.4.20.1.2 “DPRAM_IO_COPY” on page 1629* function.

For this purpose, symbolic names are assigned to each first byte on the CS31 bus and in the CPU communication area. For automatic length calculation, the last input byte is required.

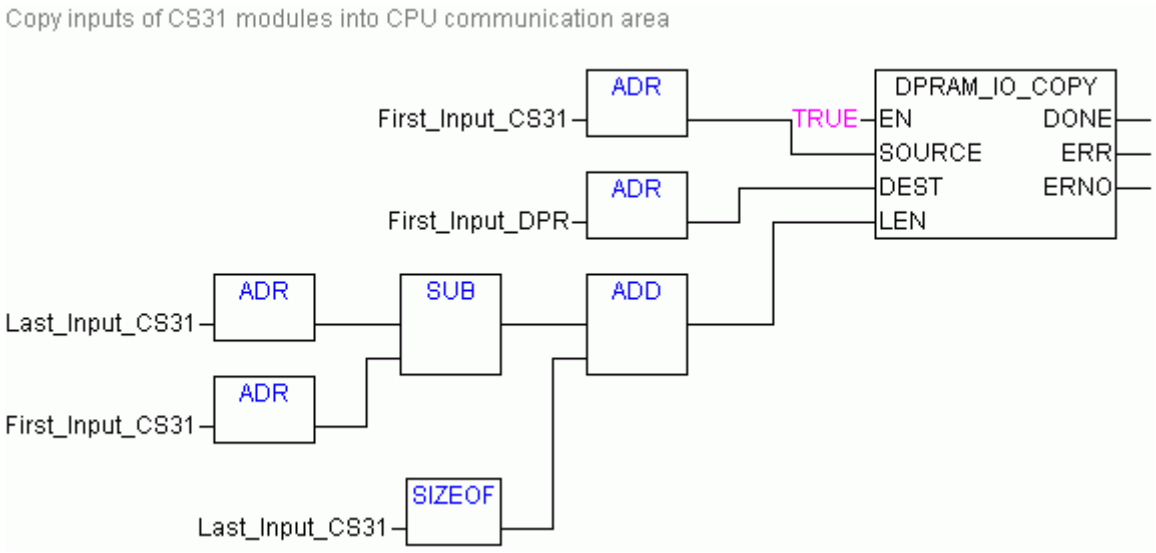
In the example, these are:

First_Input_CS31 for %IB1000,

Last_Input_CS31 for %IB1037 and

First_Input_DPR for %QB0

The PLC program (FBD) then includes the instruction:



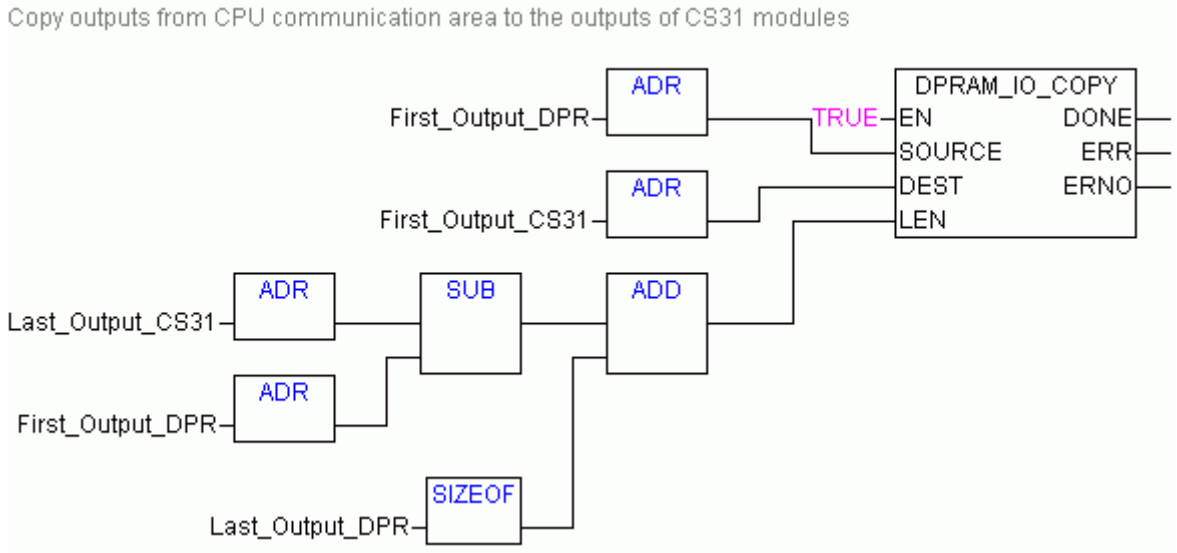
The transfer of the outputs from the CPU to the CS31 bus is performed analogously. The following symbolic names are entered into the PLC configuration:

First_Output_DPR for %IB0,

First_Output_CS31 for %QB1000 and

Last_Output_CS31 for %QB1041

The PLC program (FBD) then includes the instruction:



The entire PLC program on the CM574-RS consists of these two networks or function calls.

It is necessary to use the function block DPRAM_IO_COPY. If SysMemCpy is used instead, data consistency of the I/Os is not guaranteed.

Acyclic data exchange CM574/AC500 CPU

Function blocks for acyclic data exchange CM574-RS/AC500 CPU

Unlike the cyclic data exchange via input/output signals, the acyclic data exchange has to be programmed in the user program.

The acyclic data exchange is used mainly for transmitting commands for own protocols. For example, the entire data exchange between the AC500 CPU and a CM574-RCOM occurs via these blocks.

The following blocks are available:

Description	CM574-RS	AC500 CPU
Library	SysIntExt_AC500_V13.lib	SysInt_AC500_V10.lib
Transmission from the CM574 to the AC500 CPU or reception in the AC500 CPU from the Communication Module	*Chapter 1.5.4.20.1.5 “DPRAM_PM5XX_SEND” on page 1635*	*Chapter 1.5.4.19.2.15 “DPRAM_CM5XX_SEND” on page 1540*
Reception in the CM574 from the AC500 CPU or transmission from the AC500 CPU to the Communication Module	*Chapter 1.5.4.20.1.4 “DPRAM_PM5XX_REC” on page 1633*	*Chapter 1.5.4.19.2.14 “DPRAM_CM5XX_REC” on page 1537*

Programming example for acyclic data exchange

In the program example it is assumed that a maximum of 100 bytes is exchanged between CPU and Communication Module. The CM574-RS Communication Module is plugged into slot 1 (SLOT=1).

In principle, the program is the same in the AC500 CPU and the CM574-RS. The respective block names are to be used in the variable declaration for the CPU and the CM574.

In the example, the transmit and receive data are simply assigned to global byte ARRAYs:

```
VAR_GLOBAL
    abyRecDataCM574_1_1     : ARRAY[0..cdwMaxRec] OF BYTE;   (* Data from DPRAM *)
    abySendDataCM574_1_1    : ARRAY[0..cdwMaxSend] OF BYTE;  (* Data to DPRAM *)
END_VAR
VAR_GLOBAL CONSTANT
    cdwMaxRec               : DWORD := 99;                   (* +1 byte receive *)
    cdwMaxSend              : DWORD := 99;                   (* +1 byte send *)
END_VAR
```

The program for receiving and transmitting the data looks as follows:

```
PROGRAM proComm_CPU_CM574
VAR
                    bNewRecData         : BOOL;                 (* new data received *)
                    fbRecCM574_1        : DPRAM_CM5XX_REC;      (* CPU: receive data from first CM574 *)
                                        : DPRAM_PM5XX_REC;      (* CM574: receive data from CPU *)
                    bRecErr             : BOOL;                 (* receive error *)
                    wRecErno            : WORD;                 (* receive error number *)
                    bNewSend            : BOOL;                 (* new data from program *)
                    bNewSendData        : BOOL;                 (* new data available to send *)
                    bSendDone           : BOOL;                 (* data successfully sent *)
                    fbSendCM574_1       : DPRAM_CM5XX_SEND;     (* CPU: send data to first CM574 *)
                                        : DPRAM_PM5XX_SEND;     (* CM574: send data to CPU *)
                    bSendErr            : BOOL;                 (* send error *)
                    wSendErno           : WORD;                 (* send error number *)
END_VAR
VAR CONSTANT
                    cbySlotCM574_1      : BYTE := 1;            (* SLOT number of first CM574 *)
                    cbyChannelCM574_1_1 : BYTE := 1;            (* Channel 1 of first CM574 *)
                    cbyChannelCM574_1_2 : BYTE := 2;            (* Channel 2 of first CM574 *)
END_VAR
```

Part 1: Reception of data

```
(* receive data from CM574 --> DONE=TRUE -> new data available *)
bNewRecData := FALSE;                                           (* reset new receive data *)
fbRecCM574_1( EN := TRUE, SLOT := cbySlotCM574_1, CH := cbyChannelCM574_1_1,
                    DATA := ADR(abyRecDataCM574_1_1) );
IF fbRecCM574_1.DONE THEN                                       (* new receive *)
```

```
                IF NOT fbRecCM574_1.ERR THEN                (* with error ? *)
                                    bRecErr := FALSE;       (* receive OK *)
                                    bNewRecData :=          (* new data received
                                    TRUE;                   *)
                ELSE
                                    bRecErr := TRUE;        (* save receive error *)
                                    wRecErno :=             (* save receive error
                                    fbRecCM574_1.ERN        number *)
                                    O;
                END_IF; (* ERR *)
        END_IF; (* Rec DONE *)
```

Part 2: Processing of data

```
        IF bNewRecData THEN                                 (* new data received *)
                                                            (* process data *)
                                bNewSendData := TRUE;       (* send new data if possible *)
        END_IF; (* bNewRecData *)
```

Part 3: Transmission of data

```
        bSendDone := FALSE;                                 (* reset send
                                                            done *)
        IF fbSendCM574_1.EN OR bNewSendData THEN            (* send activ or
                                                            new cycle ? *)
                fbSendCM574_1( EN := TRUE, SLOT := cbySlotCM574_1, CH := cby-
                ChannelCM574_1_1,
                        DATA := ADR(abySendDa-
                        taCM574_1_1),
                        DATA_LEN := cdwMaxSend + 1);        (* run send POU
                                                            *)
                IF fbSendCM574_1.DONE THEN                  (* last send is
                                                            now ready *)
                        IF fbSendCM574_1.ERR THEN           (* error in last
                                                            SEND *)
                                bSendErr :=                 (* save send
                                TRUE;                       error *)
                                wSendErno :=                (* save error
                                fbSendCM574_1.              number *)
                                ERNO;
                        ELSE
                                bSendErr :=                 (* no error in last
                                FALSE;                      transmission *)
                                bSendDone :=                (* data success-
                                TRUE;                       fully transmitted
                                                            *)
                        END_IF; (* ERR *)
```

```
                                    fbSendCM574_1 (EN := FALSE);      (* call with
                                                                      EN:=FALSE for
                                                                      new edge *)
                          END_IF; (* DONE *)
 END_IF; (* EN *)
```

Special functions of the CM574-RS

Function blocks for data storage in flash memory

The library "SysInt_AC500_V10.lib" located in the directory "Data storage/Flash" contains the following blocks which are used to store data in the Flash memory:

Block	Function
Chapter 1.5.4.19.2.16 “FLASH_DEL” on page 1542	Deletes a data segment in the Flash memory
Chapter 1.5.4.19.2.17 “FLASH_READ” on page 1544	Reads a data segment from the Flash memory
Chapter 1.5.4.19.2.18 “FLASH_WRITE” on page 1547	Writes a data segment to the Flash memory

The blocks and their functionality are the same as for the AC500 CPU. These blocks are described in the library documentation of the Internal System Library *Chapter 1.5.4.19.2.18 “FLASH_WRITE” on page 1547*.

LEDs control of the CM574-RS

The library "SysIntExt_AC500_V10.lib" contains the function block LED_SET for controlling the LEDs of the CM574-RS. See description of the function block *Chapter 1.5.4.20.1.6 “LED_SET” on page 1638* for further information.

> *If the Error LED is to be activated from the user program, the parameter "Error LED" has to be set to "Off" Chapter 1.6.5.2.6.3.2 “Configuration of CM574-RS” on page 5903.*

Reading the address switch of the CM574-RS

The library "SysIntExt_AC500_V10.lib" contains the function block CPU_OWN_ADR for reading the address switch of the CM574-RS. See description of the function block *Chapter 1.5.4.20.1.1 “CPU_OWN_ADR” on page 1627* for further information.

The address set with the rotary switches is output at the ADDR output.

Programming access to the CM574-RS

Programming via the serial interface of the CM574-RS

Programming via the serial interface is performed with the same drivers and settings as programming via the serial interface COM1 or COM2 of the AC500 CPU *Chapter 1.6.5.2.6.3.2 “Configuration of CM574-RS” on page 5903*.

The pin assignment of the programming cable is the same as for COM1 of the AC500 CPU.

Programming the CM574-RS via the AC500 CPU (Routing)

For programming via the CPU, the following programming interfaces are available:

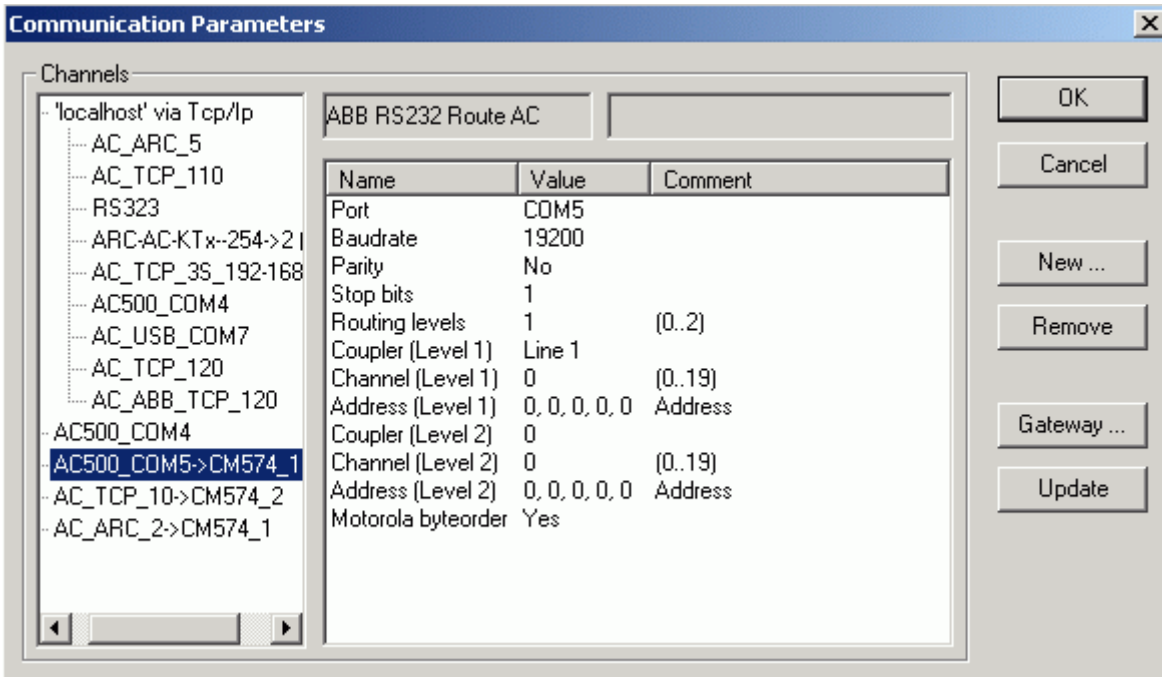
- Serial driver "ABB RS232 Route AC"
- Ethernet driver (TCP/IP) "ABB Tcp/Ip Level2 AC"
- ARCNET driver "ABB ARCNET AC"

Setting up a gateway channel is described in the section *Chapter 1.6.5.4.2 “Programming and testing” on page 6198*.

This chapter only describes the settings for the routing to the CM574-RS.

Programming via CPU with serial driver "ABB RS232 Route AC"

The following figure shows the setting of a gateway channel for COM5 of the PC with routing to the CM574-RS plugged into slot 1 (line 1).

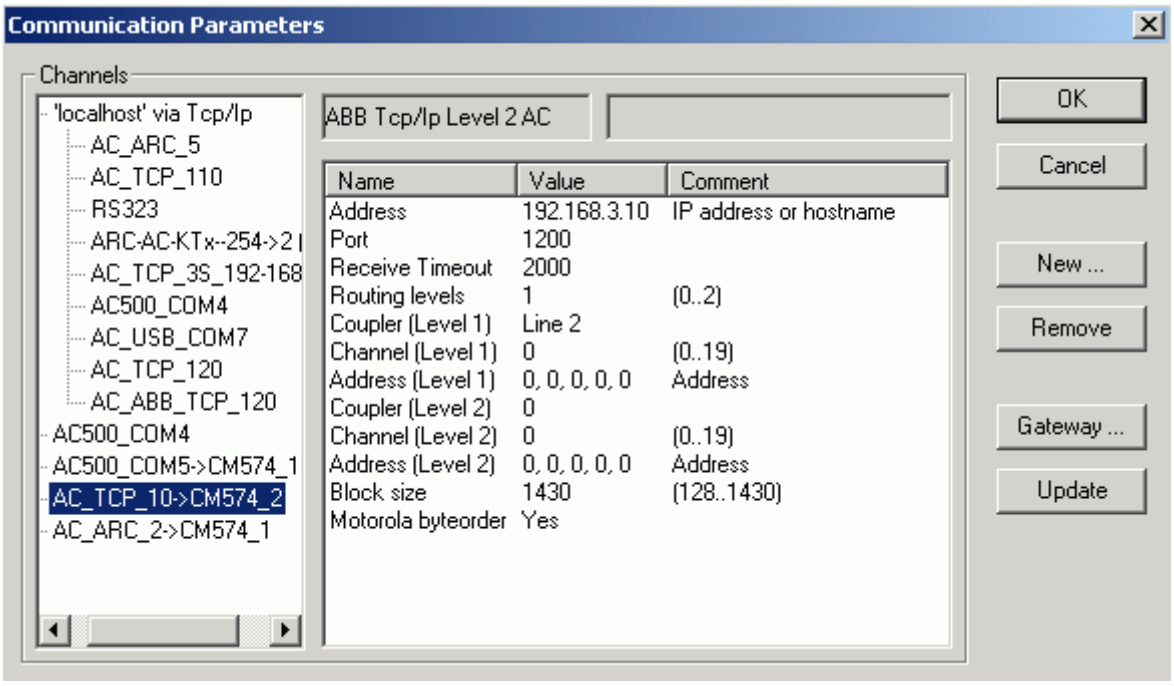


The following settings have to be specified:

Port	Serial interface (COMx) of the PC, in the example: COM5
Transmission rate	Transmission rate of COMx of the AC500 CPU
Routing levels	1
Communication Module (Level 1)	Line 1
Address (Level 1)	0,0,0,0,0
Motorola byteorder	Yes

Programming via CPU with Ethernet Driver "ABB Tcp/Ip Level2 AC"

The following figure shows the setting of a gateway channel for connection via the AC500 CPU with the IP address 192.168.3.10 with routing to the CM574-RS plugged into slot 2 (line 2).

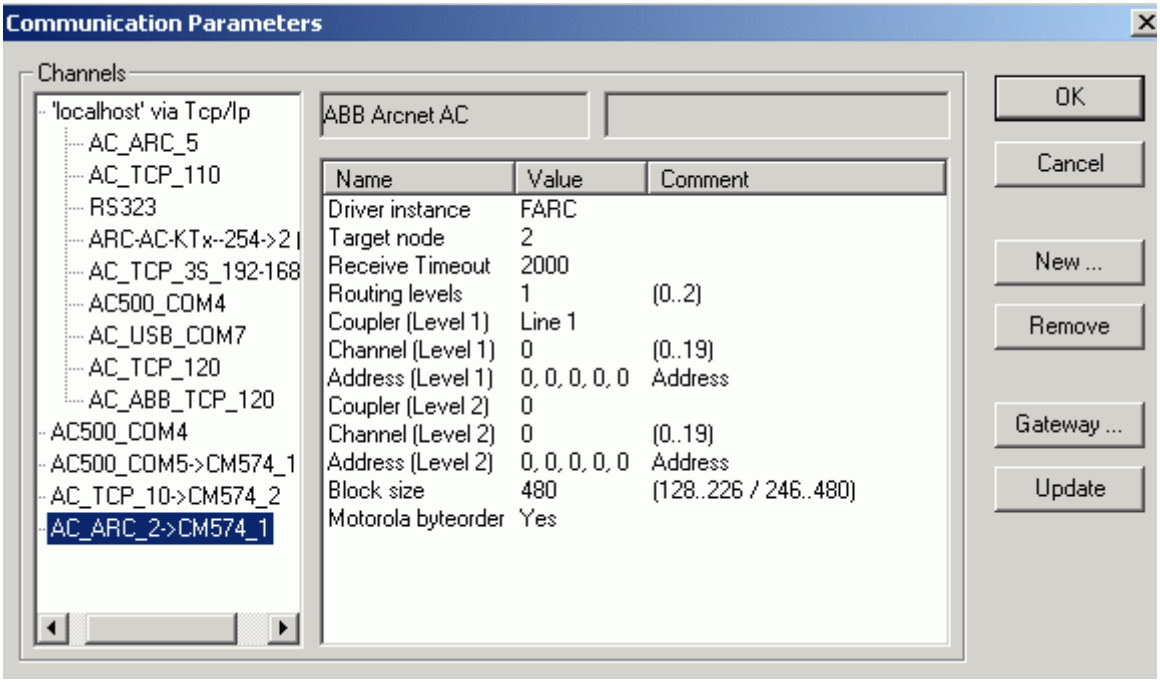


The following settings have to be specified:

Address	- IP address of the AC500 CPU, in the example: 192.168.3.10
Routing levels	- 1
Communication Module (Level 1)	- Line 2
Address (Level 1)	- 0,0,0,0,0
Motorola byteorder	- Yes

Programming via CPU with ARCNET Driver "ABB ARCNET AC"

The following figure shows the setting of a gateway channel for connection via the AC500 CPU with the ARCNET address (node) 2 with routing to the CM574-RS plugged into slot 1 (line 1).



The following settings have to be specified:

Target node	ARCNET address of the AC500 CPU, in the example: 2
Routing levels	1
Communication Module (Level 1)	Line 1
Address (Level 1)	0,0,0,0,0
Motorola byteorder	Yes

1.6.4.2.7 RCOM/RCOM+ communication module

CM574-RCOM - RCOM/RCOM+ communication module

Overview

The RCOM/RCOM+ communication module CM574-RCOM requires a processor module with firmware version V1.3.0 or above.

The capabilities of the communication module correspond to a processor module. Further information on the processor modules can be found in the corresponding device description ↳ *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*.

CM574-RCOM can be plugged into any slot for communication modules on a terminal base TB511, TB521 or TB541. Further information on the terminal bases can be found in the corresponding device description ↳ *Chapter 1.6.2.2.1 “TB51x-TB54x” on page 3786*. Up to 4 CM574-RCOM can be used in one AC500 system.

The communication module provides 2 serial interfaces which are designed according to the standards EIA RS-232 and EIA RS-485. The pin assignment of the interfaces corresponds to the pin assignment of COM1 at processor module PM57x, PM58x and PM59x.

Features

- The CM574-RCOM communication module can be configured as RCOM master or RCOM slave.
- Up to 254 RCOM slaves are possible in a network (max. 8 slaves when using Master-Piece 200 and max. 30 slaves when using dial-up operation).
- Available RCOM services are cold start, warm start, normalization, clock synchronization, writing and reading data, event polling.
- The RCOM interface for connecting the modem corresponds to EIA RS-232. Operation according to EIA RS-485 is also possible.
- An additional RS-232 interface (CONSOLE) is available for commissioning (for displaying communication history, planning phone numbers, etc.).
- Software clock (can be used by the PLC program).

RUN/STOP behavior

After initialization, the CM574-RCOM runs until it is reinitialized. The abortion of a running program can cause the interruption of RCOM connection elements. If a connection element does not respond within 2 seconds, the CM574-RCOM displays the message "command not reset by CPU" or "no CPU reaction".

If the communication module is configured as RCOM slave, the corresponding error message ("application part not ready", RCOM error number 4020h) is sent to the RCOM master.

If the communication module is configured as RCOM master, the interrupted job is repeated until the processor module resets the job. This is done at the latest with the next initialization performed by the RCOM_INIT connection element.

Single step and single cycle

Since the CM574-RCOM monitors the reaction of the CPU module in case of incoming and triggered services, it is not possible to run the RCOM connection elements in single step or single cycle mode. Thus, you should always run the communication part of the processor module program in "real time".

Communication module control

The communication module is controlled by the processor module by means of commands. The commands from the master station and the responses of the communication module are exchanged via the dual port RAM (DPRAM) of the AC500 processor module.

The user controls the communication module exclusively by means of the supplied connection elements of the RCOM library RCOM_AC500_V13.lib. The library contains connection elements for all system and data transmission services. This permits the user to define the required communication sequence very simply ↳ *Chapter 1.5.4.30 “RCOM/RCOM+ library” on page 1903*.

Function blocks for RCOM/ RCOM+

The communication between the processor module and the Communication Module CM574-RCOM is performed via function blocks of the RCOM library RCOM_AC500_V13.lib ↳ *Chapter 1.5.4.30 “RCOM/RCOM+ library” on page 1903*.

Parameters

All important parameters (transmission rate, timeout times etc.) for the communication module are defined in the PLC configuration of the processor module's PLC program. Further information can be found in the description of configuration of the communication module ↳ *Chapter 1.6.5.2.6.4.1 “Module parameters” on page 5907*.

The parameters for the individual services (e.g. slave address, data set number etc.) are preset directly on the connection element for this service.

Commissioning

A terminal can be connected to the CM574-RCOM in order to simplify commissioning. The communication module then issues messages concerning incoming commands from the processor module, received or transmitted RCOM telegrams and any occurring errors.

This function can be deactivated after commissioning. The communication processor then continues operation without a terminal.

Program examples for RCOM communication

When installing Automation Builder software, the example projects for RCOM dedicated line and RCOM dial-up line are installed.

RCOM/RCOM+ protocol

RCOM is a transmission protocol which is particularly suitable for data transmission over long distances (RCOM = Remote COMmunication).

The protocol is based on a simple V.24 interface and thus permits the use of standard data teletransmission devices (e.g. modems).

The main fields of application for RCOM are as follows:

- Networking AC500 systems to AC31 systems
- AC500 stations and AC31 stations
- Networking of AC500 stations

You can use either dedicated lines (e.g. existing cable paths or leased lines) or telephone lines with dial-up modems for communication.

Differences between RCOM and RCOM+

There are the following differences between RCOM and RCOM+ concerning data transmission:

- The "BREAK" character of RCOM is replaced by a transmission break of configurable length for RCOM+
- The 8 bit "exclusive-or check sum" (XOR) of RCOM is replaced by a 16 bit CRC16 checksum for RCOM+

Switching between RCOM and RCOM+

Switching between RCOM and RCOM+ is performed by selecting the corresponding operation mode in the PLC configuration ↬ *Chapter 1.6.5.2.6.4.1 “Module parameters” on page 5907*.

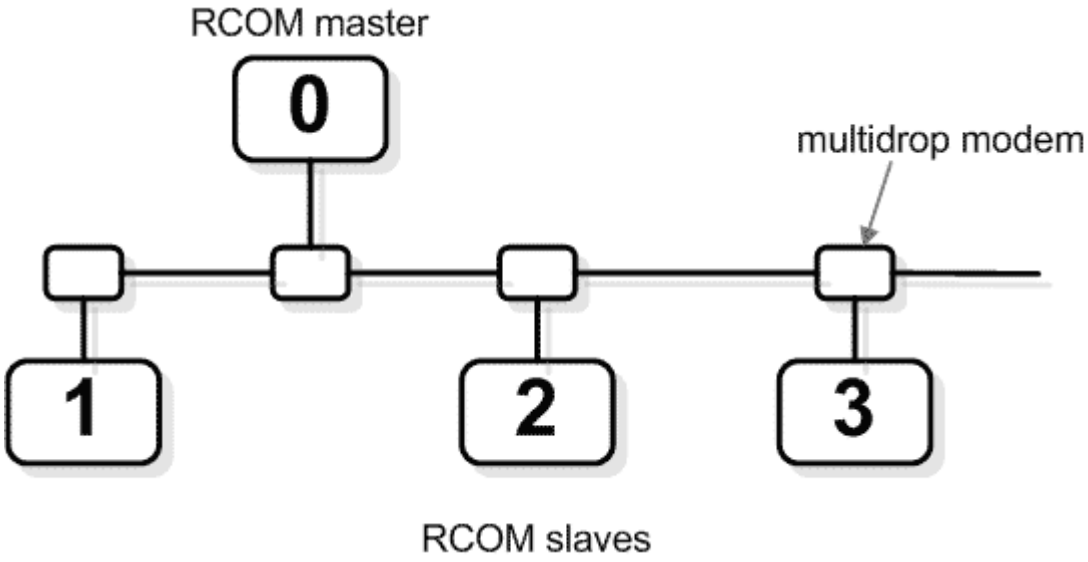
RCOM networks

A RCOM network consists of 2 or more users, e.g. control computers or substations etc. One user is always configured as RCOM master. All other users are RCOM slaves.

The users are connected by means of a transmission medium. In case of RCOM, this may be a direct line (point-to-point connection), a dedicated line with multi-drop modem (these permit coupling of several users to one line) or a dial-up connection over the public telephone network.

Each subscriber in the network has an address assigned that can be used to address (call) this specific device. This address is a number between 1 and 254 for slaves, or 0 for the master.

The figure below shows an RCOM network with multi-drop modems:



With each job, the master transfers the specific address of the addressed slave and only this slave will respond to the job.

Master-slave structure

There is a simple convention for controlling data communication on the line: One user in the network is the master. Only the master can send jobs to other users, the slaves. The slaves respond to a job telegram with a telegram that indicates whether the job has been understood and whether it has been possible to execute it.

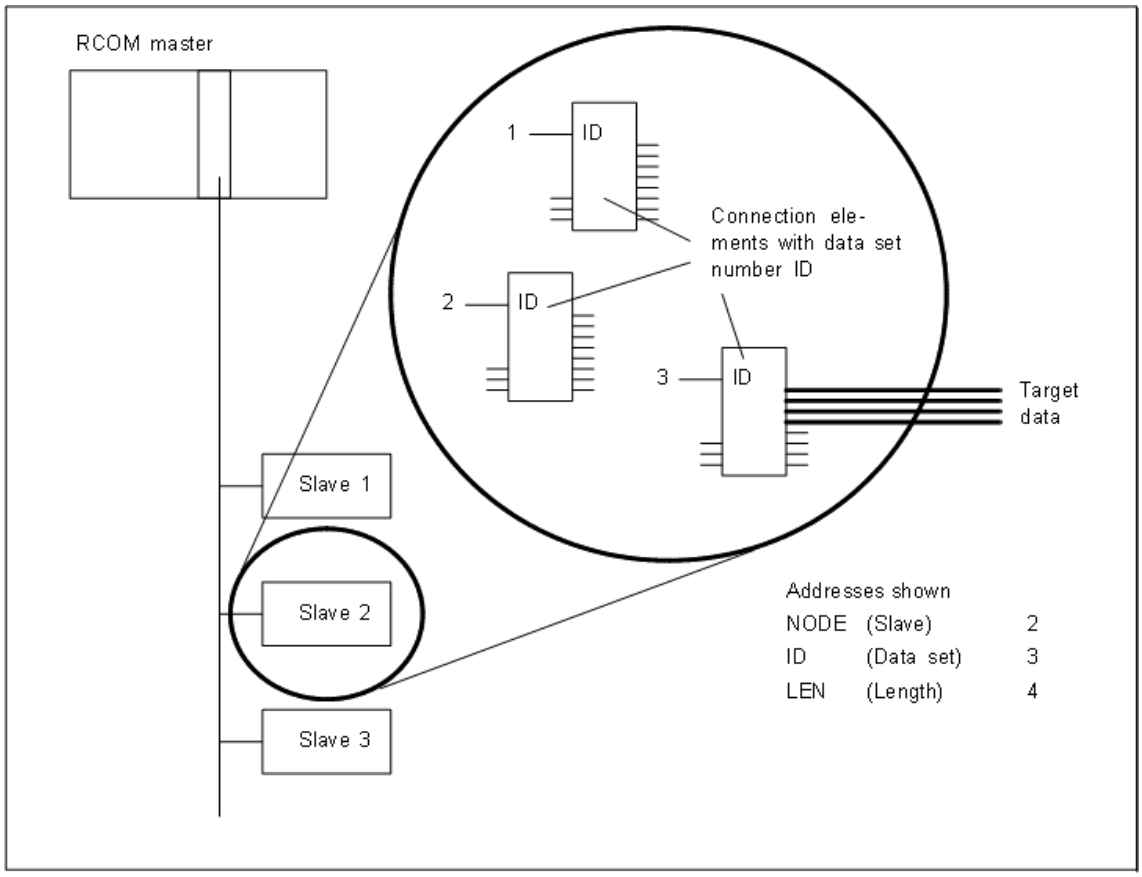
This always results in the sequence job telegram (response telegram) on the data line.

Addressing in the RCOM network

A complete address has to be specified in order to address a data set on a specific slave. This address consists of the following parts:

- Number of the slave (NODE). Up to 254 slaves may exist in an RCOM network. Number 0 is used by the master.
- Number of the data set (ID).
- Number of data words of the data set to be transmitted (LEN). Data sets do not always have to be transmitted completely. However, a minimum of two data words have to be transmitted and the number of data words has to be even-numbered. Transmission always starts with the first data word.

The figure below shows an example of a RCOM system and the addressing path:



Broadcast

Slave number FF hex (255) can be used to transmit specific services to all slaves simultaneously. Such a service is not answered by any of the slaves and is thus repeated several times by the master as a safety measure.

Broadcast is permitted only for jobs where no user data are transmitted (system services).

Job types

There are several types of jobs that can be divided into two groups:

- System services for RCOM network management, e.g. for connection set-up, clear down and reinitializing the network etc.
- Services for data transmission. After the RCOM network has been started successfully, the master triggers these jobs in order to transmit user data.

System services

The following services are provided for RCOM network management:

- Cold start: The addressed slave is reset, i.e. all protocol control characters are set to the initial state. In addition, the event queue is deleted. After a cold start, data sets cannot be transmitted again until normalization is completed.
- Warm start: During a warm start, the event queue is deleted. A warm start can be performed after a transmission error in order to resynchronize master and slave. After a warm start, data sets cannot be transmitted again until normalization is completed.
- Normalization (normalize user part): Normalization enables the transmission of data sets after a cold start or a warm start. This job has to be used to enable communication.
- Set clock (clock synchronization): The CM574-RCOM provides a clock that generates time stamps for events. This clock can be set by the master.
- All system services can be started in the master using corresponding connection elements. In the slave, the system services are handled automatically by the CM574-RCOM, i.e. nothing needs to be configured for the system services in the RCOM slaves.

All system services can be started in the master using corresponding connection elements. In the slave, the system services are handled automatically by the CM574-RCOM, i.e. nothing needs to be configured for the system services in the RCOM slaves.

Terms

Do not confuse the RCOM services cold start / warm start and the corresponding PLC commands referring to the hardware state. In this section, the terms "cold start" and "warm start" always refer to the RCOM system services and therefore only affect the protocol state.

Data transmission

Data sets

All user data in the RCOM network are transmitted in so-called data sets (DSs for short). They consist of a maximum of sixteen 16 bit data words (eight 32-bit double words in case of MasterPiece).

A maximum of one data set can be transferred in each job telegram. For transmission of large amounts of data should, several jobs have to be started.

For identification, each data set has a number ("ID") assigned, which is also transferred in the telegram.

In the PLC program, data sets can be stored in word arrays. The address of the start word must be set at input "DATA" by using the ADR operator.

For transmission, the data set number "ID", the first of 16 words (start word at "DATA") and the number of data words to be transferred ("LEN") has to be specified at the connection element used for transmitting the data set.

Three options for transmission of data sets are available:

- Write data sets to slave
- Read data sets from slave
- Event polling

Write data set

The master can write a data set to the slave by reading the user data and sending them to the slave by means of a telegram. The data are stored at the specified address of the function block, e.g. in a word array. The slave confirms the reception of data in the response telegram.

Read data set

The master reads a data set from the slave by first sending a job telegram to the slave. The slave receives this telegram, reads the data from the address specified by input "DATA" and returns the data in the response telegram. The data of the response telegram are stored at the address specified at input "DATA" of the corresponding function block.

Event polling

In many applications it is required that the slave transfer data to the master, e.g. if an important event is contained in the process.

Normally, the slave is not allowed to start communication itself (master-slave structure). This means, the slave has to wait until the master reads the required data set.

The RCOM protocol provides a simple mechanism to solve this problem: Event polling.

If the slave wants to transmit data to the master, it can initiate the transmission as follows:

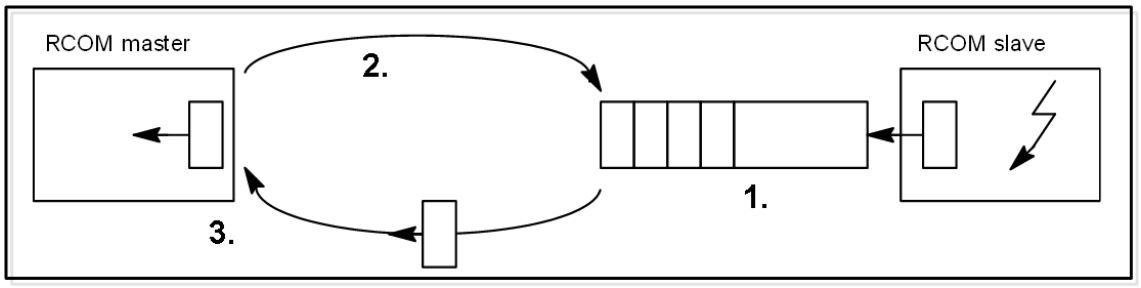
- The slave transfers the data set to a queue (event queue) on the CM574-RCOM Communication Module.
- The master cyclically polls all slaves consecutively to search for events in the queues. If an event is available, the addressed slave sends the data set in the response. If not, it sends the response Event queue empty.

This way, the slave can signal events to the master very easily. If no events occurred, no user data are exchanged. In this case, transmission is completed very quickly.

Since there is no correlation in time between triggering of the event (insertion in the queue) and event polling (read-out of the queue by the master), a time stamp is stored in the data set providing information regarding the time when the event occurred. Therefore, a data set can only contain a maximum of 14 data words and the last two words contain the time stamp.

The CM574-RCOM can store a maximum of 20 events in the queue. Further events are rejected with an error message.

The figure below shows the event-driven transmission.



The slave detects an event and transfers the data set to its event queue (1.). The RCOM master polls the slave (2.) and receives the corresponding data set contained in the slaves response telegram (3.).

Planning

To simplify the use of the communication module, connection elements are provided for all required services.

- RCOM_INIT block for initializing the communication module.
- Connection elements for the system services
  - cold start service (RCOM_COLDST)
  - warm start service (RCOM_WARMST)
  - normalization (RCOM_NORMAL)
  - event polling (RCOM_POLL)
  - telephone dialing (RCOM_DIAL)
  - telephone hang-up (RCOM_HANGUP)
- Connection elements RCOM_TRANSMIT (transmit data set) and RCOM_REC (receive data set) for writing data sets. These connection elements are also used for event-driven transmission. For event polling, the RCOM_POLL connection element has to be used in the master to trigger polling of the slave.
- The RCOM master uses the RCOM_READ connection element for reading data sets. The addressed slave provides the data to be read in the RCOM_READ_SLV connection element.

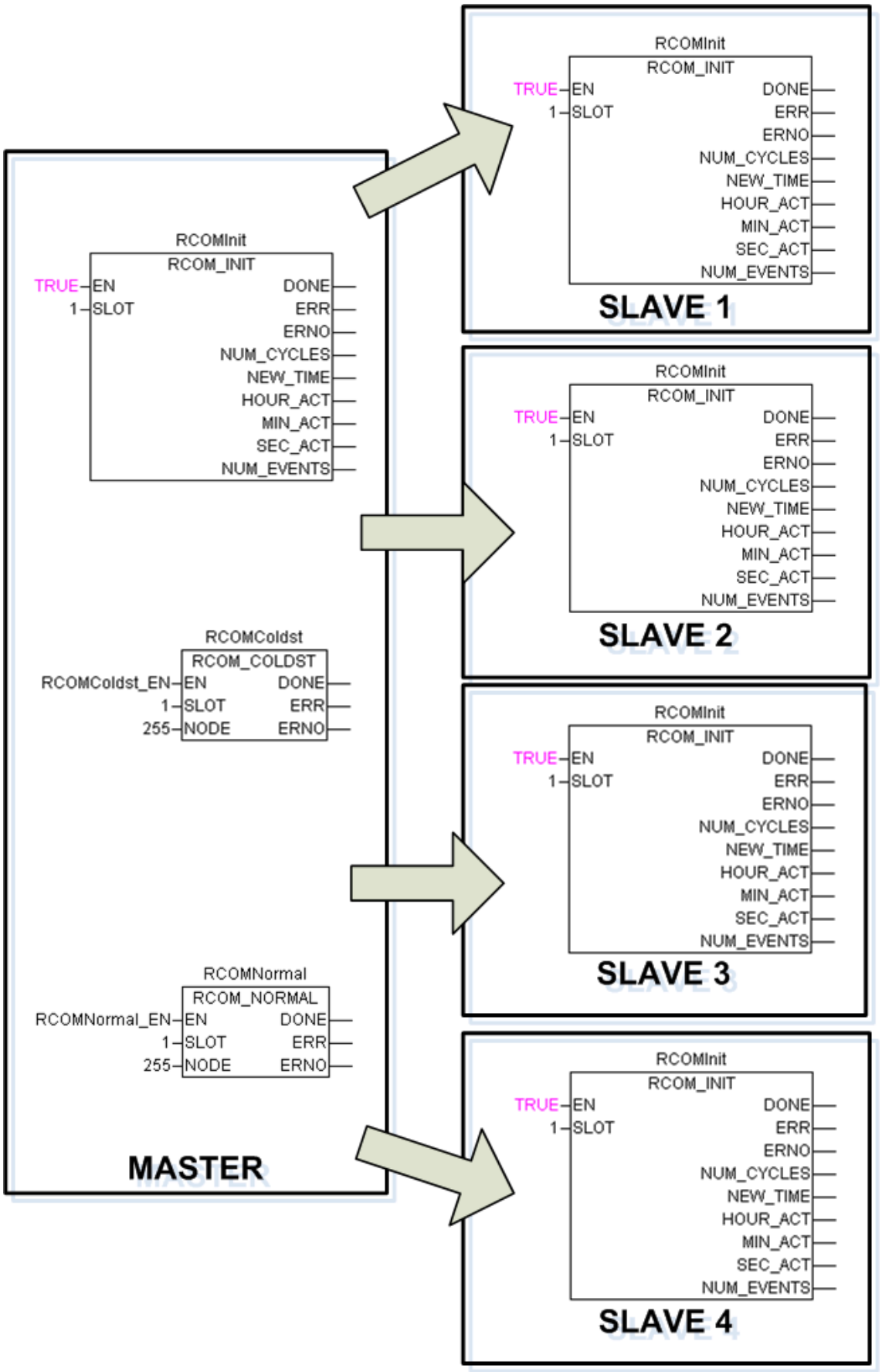
Initialization, cold start and normalization

The figure below shows the application of the connection elements RCOM_INIT, RCOM_COLDST and RCOM_NORMAL. These connection elements have to be used for initializing the communication processors and for starting the RCOM protocol.

Each RCOM user is initialized by the RCOM_INIT connection element, i.e. the transmission parameters, the network address and the timeout times etc. are defined.

The RCOM master then has to perform an RCOM cold start service (RCOM_COLDST) and data transmission has to be enabled by normalization (RCOM_NORMAL). In the example, cold start and normalization are implemented by broadcast telegrams (NODE = 255) so that all slaves are addressed simultaneously.

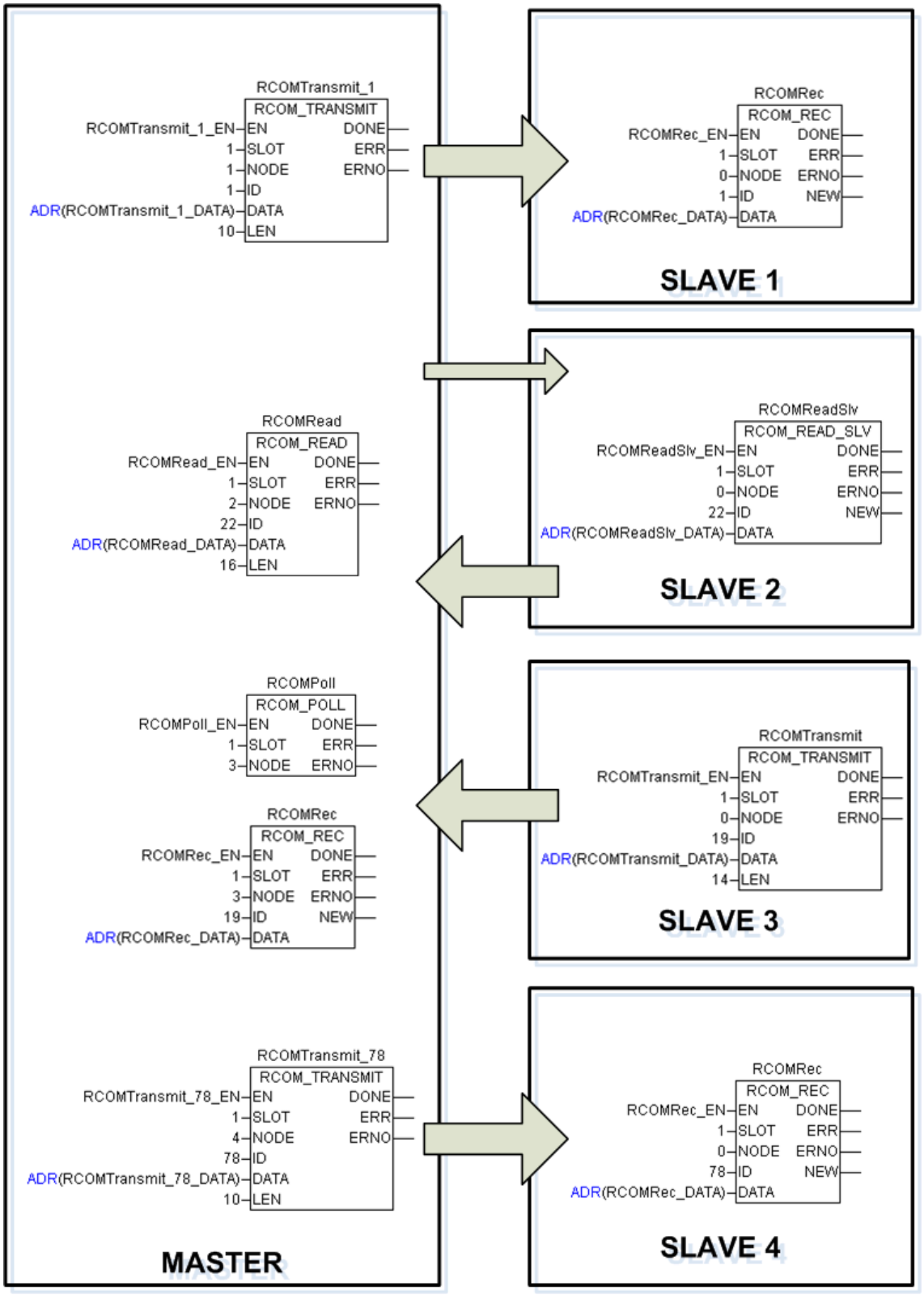
The RCOM network is ready for data transfer after the procedure above has been performed.



In the application shown above, the connection elements for initialization and for the RCOM services cold start and normalization are displayed in simplified form.

Data transmission

The figure below shows an example of transmission of data sets. It illustrates the relationship between connection elements and the significance of address and data set number ("ID").



In the application shown above, the connection elements for data transmission are shown in simplified form.

The following transmissions can be performed in the example:

- Data set 1 is transmitted from the master to slave 1.
- Data set 22 is read by slave 2.

- Data set 19 is sent from slave 3 to the master (event-driven). The master must first poll slave 3 (connection element RCOM_POLL).
- Data set 78 is sent from the master to slave 4.

The figure above does not show the logic for enabling of the connection elements.

For each connection element in the master, a corresponding partner is available in the slave. These two connection elements have the same data set number ("ID").

The data applied to the inputs and outputs of the connection element are not the actual user data but only parameters referencing the storage location of the user data instead.

All data sets in the slave are addressed by zero ("NODE") because jobs can be initiated only by the master.

A specific sequence of system services has to be observed for starting an RCOM network. This sequence ensures the correct initialization of the RCOM protocol.

All system services are triggered by the master using the corresponding connection elements. Nothing needs to be planned in the slaves for the system services. The CM574-RCOM configured as slave handles all system services independently.

Terms

Do not confuse the RCOM services cold start / warm start and the corresponding PLC commands referring to the hardware state. In this section, the terms "cold start" and "warm start" always refer to the RCOM system services and therefore only affect the protocol state.

Cold start

A cold start service has to be performed after the initialization of the RCOM master. The cold start can be transmitted either by broadcast to all slaves simultaneously or to each slave individually.

A cold start requires reinitialization of the entire protocol mechanism and clearing of the event queue contents. For this, a special cold start event is triggered in the addressed RCOM slaves. This event is required when operating ABB MasterPiece systems. In case of pure Advant Controller networks, the event is only indicated when polling.

After a cold start, always normalization has to be performed. Otherwise it is not possible to transmit data sets.

Warm start

By executing a warm start service, it is possible to clear the event queue of a slave (or all slaves). A warm start can be used to resume communication after transmission errors. This permits master and slave to resynchronize.

After a warm start, always normalization has to be performed. Otherwise it is not possible to transmit data sets.

Normalization

A slave has to be normalized after a cold start or a warm start. Normalization enables the transmission of data sets and events. If a slave is not normalized, it cannot trigger events. The RCOM_TRANSMIT connection element then displays a corresponding error message.

If a master polls a non-normalized slave, the RCOM_POLL connection element signals a corresponding error.

Set clock

When the CM574-RCOM is switched on, its software clock is set to 0:00 hours. You can use the RCOM_CLOCK connection element to set the clock of the RCOM master and the clocks of all slaves to the same time. This is important for evaluating time stamps in the case of event-driven data transmission.

Clock setting should always be performed after a cold start and should be repeated cyclically (e.g. every 24 h), if necessary.

The CM574-RCOM contains a software clock for generating time stamps. You can also use this software clock in the PLC program of your CPU module. The time information is made available at the outputs NEW_TIME, HOUR_ACT, MIN_ACT and SEC_ACT of the RCOM_INIT connection element and is updated there approximately every 5 seconds.

The RCOM time starts with 00:00.00 when the Communication Module is set to the RUN state.

The connection element RCOM_CLOCK sets the RCOM clock of the master and sends a set clock telegram to all slaves ("NODE" must be set to 255 for this purpose).

Proceed as follows to set the RCOM time: Read the actual time from the real-time clock and start the RCOM_CLOCK connection element in the RCOM master with this time (NODE = 255 in order to address all slaves). The new time is then transferred to the RCOM clock in the master and in all slaves and output NT is set to "1" for approx. 5 seconds. If individual slaves also use real-time clocks, you can use the edge of NEW_TIME to set these clocks. The master and all slaves use the same time then.

You should use the RCOM_CLOCK connection element even if the RCOM master does not have a real-time clock (e.g. with time 00:00.00 hours). All time stamps in events are then calculated relative to this arbitrary RCOM time.

When planning, you should first precisely analyze the required communication relationships in order to avoid subsequent modifications.

You should consider the following questions:

- How many slaves are necessary? Determine the slave numbers ("NODE").
- How many data words have to be transmitted for each slave? Define the subdivision of the data into data sets. Determine flag ranges for each data set.
- How do the individual data sets have to be transmitted? Cyclically? At the request of the PLC program? Event-driven? Define the communication sequences. Do not forget the required system services (e.g. cold start and normalization). Chapter Planning examples shows how such sequences can be implemented.
- How must the PLC program of the CPU module respond to transmission errors? The example program shows a possible solution.

Important planning rules

The following rules have to be observed during planning for the CM574-RCOM:

- "ID" (data set identifier) can have values between 1 and 255.
- Only an even number of data words can be transmitted in a data set.
- Connection elements must not be skipped once they have been started. This would disturb the logic sequence between the connection element and the CM574-RCOM and cause the connection element to block.
- The connection elements must not be started before the RCOM_INIT connection element has been executed successfully (initialization of the Communication Module). This would cause the connection elements to block.
- Data transmission with RCOM_READ jobs takes approximately twice as long as event-driven transmission. Consequently, you should prefer event-driven transmission in case of time-critical transmissions.
- Max. 14 words are permitted per data set for event-driven transmission. The time stamp is included in two data words, directly after the user data. The master must know the number of words transmitted, if it wishes to evaluate the time stamp (plan a fixed length).

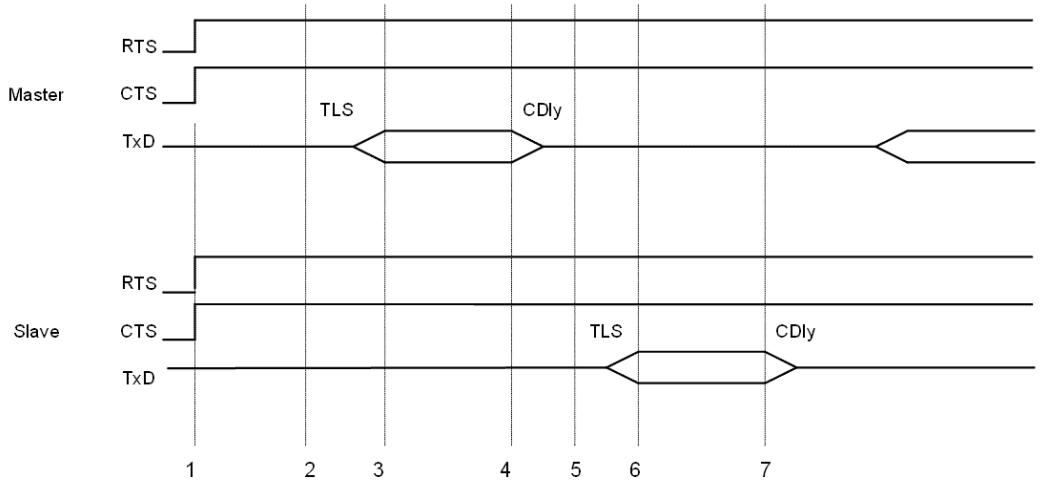
Full duplex

In case of full duplex mode, the RTS line of the Communication Module is set to "1" after initialization.

Prior to the transmission of characters, the Communication Module expects a valid CTS line. The modem may set CTS to zero during transmission in order to stop data flow.

Full duplex should be used for transmission links that provide a separate channel for each transmission direction, e.g. modem-zero cables, telephone connections or modem LS-01 of Messrs. Hedin-Tex.

The figure below shows full-duplex data communication on a CM574-RCOM used as RCOM slave.



1. The CM574-RCOM sets RTS to "1" during initialization. The modem responds with CTS = 1.
2. The CPU starts the job on the CM574-RCOM. The Communication Module waits for TLS.
3. The Communication Module checks CTS and starts transmission of the job telegram.
4. The telegram is finished. The Communication Module waits for CDly.
5. The slave recognizes the job and processes the telegram. TLS is expected before the response is transmitted.
6. The slave starts transmission of the response telegram.
7. After the telegram has been transmitted, the slave waits for CDly. This terminates transmission.

TLS is always expected before transmission of a telegram. CDly is expected after the transmission of a telegram. For error-free communication, TLS has to be greater than CDly of the remote station .

Delay times

Delays can increase the transmission reliability and can be planned before and after the telegram in full duplex mode and in half duplex mode.

Parameter TLS ("line stab. time") indicates the time expected before the transmission of a telegram and after the activation of the modem carrier with CTS = "1" (in case of half duplex only).

Parameter CDLY ("carrier delay") indicates the delay after the telegram.

The following condition has to be observed for full duplex and half duplex mode in order to guarantee reliable transmission: Own TLS > CDLY of the remote station.

The two delay times are entered as a number of characters (duration of transmission of a character at the given transmission rate) so that longer delays result in case of lower transmission rates.

Since the internal clock runs with a clock rate of 10 ms, only multiples of 10 ms are practical.

Examples:

- Transmission rate = 9600 baud -> 1 character = approx. 1 ms, practical time values: 10, 20, 30 etc.
- Transmission rate = 4800 baud -> 1 character = approx. 2 ms, practical time values: 5, 10, 15 etc.

Recommended values for TLS and CDLY at 1200 baud:

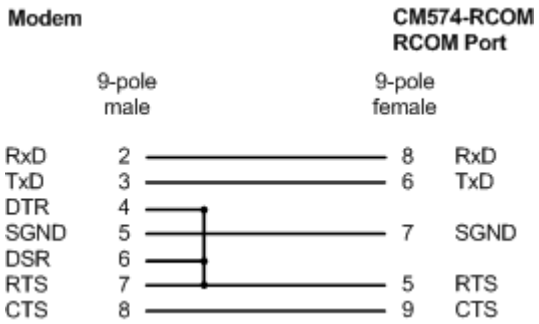
- with half duplex: TLS = 3 characters, CDLY = 2 characters
- with telephone connections: TLS = 2 characters, CDLY = 2 characters
- with full duplex (multidrop or point-to-point): TLS = 2 characters, CDLY = 0 characters.

Please note that these values depend on the modem used and should be determined experimentally, in particular in case of half-duplex links.

Using dial-up modems

The CM574-RCOM communication processor is able to handle communication via the public switched telephone network. For this purpose, it can control Hayes-compatible modems (controlled by AT commands).

The figure below shows an example pin assignment between a modem (RS232C interface) and the RCOM interface of the CM574-RCOM:



The figure below shows an example pin assignment between a modem (RS232C interface) and the RCOM interface of the CM574-RCOM:

- Use only Hayes-compatible modems. Deactivate any MNP options possibly
- Deactivate any MNP options possibly available on the modem. In case of MNP transmission, the correlation in time between telegrams is lost and transmission is disturbed.
- For transmission, a physical connection is necessary that allows the transmission of breaks and binary characters without loosing coherence in time (duration of the break signal, intervals of the characters). For most modems, these operating mode is called "direct mode".
- In the PLC program of the CPU module, connection has to be established (RCOM_DIAL connection element) prior to the transmission of RCOM services, and has to be terminated again afterwards (RCOM_HANGUP).

During commissioning, you should first attempt to address the modem with the operator command "MOD" (refer to chapter "Operator"). If you enter the command OPERATOR>MOD ATI4<CR>, you should see a table of the most important modem parameters. If this is not the case, modem configuration is probably incorrect (transmission rate, parity, etc.).

Communication sequence on the RCOM master

Observe the following sequence in the PLC program of the RCOM master for data transmission:

- Set modem parameter type to "Hayes compatible dial modem" in the PLC configuration.
- Initialize the CM574-RCOM using the RCOM_INIT connection element.
- Call the remote station: RCOM_DIAL. If RCOM_DIAL is completed and no error is signaled:
- Perform a cold start or warm start at the remote station, if required. Cold start and warm start delete the event queue of the called slave. Do not use broadcast telegrams with these services. Address the slave explicitly instead.
- Then perform normalization. This service always has to be performed in order to initialize the protocol mechanism for data transmission. Do not use broadcast telegrams with these services. Address the slave explicitly instead.

- Only if normalization does not signal an error, you may write and read data sets (RCOM_TRANSMIT and RCOM_READ) and poll the slave (RCOM_POLL and RCOM_REC).
- Then you have to terminate the connection (RCOM_HANGUP).

In case of transmission errors (which may occur particularly during normalization) you should terminate the connection using RCOM_HANGUP and start a new dialing attempt.

Communication sequence on RCOM slaves

No special connection elements need to be planned in the PLC program for regular data transmission (master calls and starts services) in case of RCOM slaves.

The slave "picks up" the telephone when it rings and then expects telegrams from the master. If no further telegrams arrive after the waiting time has expired (RCOM_HANGUP time), the slave "hangs up" automatically.

Event transmission: DIAL in slave

The slave can call the master, if it wants to transmit events to the master.

For this purpose, a DIAL connection element needs to be started in the slave. Communication is performed as follows:

- The slave calls the master using RCOM_DIAL.
- The master answers the telephone.
- After a short waiting time, the master starts to normalize all slaves configured in the telephone directory. Since only one slave can be the caller, only the calling slave will answer correctly.
- The master now automatically polls the recognized slave until it signals that the event queue is empty or until the number of polls defined by parameter "Maximum polls" set in the PLC configuration is reached. The received data sets are transferred to the RCOM_REC connection elements in the master.
- The master then "hangs up".
- The slave also "hangs up" after a waiting time ("Hang-up time").

The master automatically attempts to poll all slaves when it is called. No RCOM_POLL connection element is required for this purpose. The RCOM_POLL connection element is only required, if the master calls the slave.

No RCOM_HANGUP connection element needs to be planned in the slave, since the slave does not know when transmission is completed. The slave "hangs up" automatically after expiry of the "hang-up time" (refer to section "Timeouts").

The master and the slave strictly monitor whether RCOM telegrams are actually transmitted over the established connection ("telephone off hook").

If no telegrams were received by the slave or if no services were started in the master after expiry of the "hang-up time", the telephone connection is terminated again.

This prevents "wrong callers" who have dialed the wrong number for instance from blocking the telephone permanently.

Since the RCOM slave is not able to completely monitor the status of the modem (control only with RCOM_DIAL), you should select a short "hang-up time" for the slave, e.g. 10 seconds.

On the master, the timeout should never respond since the modem can be monitored completely by the PLC program of the CPU module (RCOM_DIAL and RCOM_HANGUP). Consequently, you can select a long "hang-up time", e.g. 30 seconds.

Correct settings for the modem are very important for error-free communication.

Certain parameters can be stored in a non-volatile memory in the modem. All other parameters can be stored on the CM574-RCOM in the Init string of the modem setup. They are then transferred to the modem when the CM574-RCOM is initialized.

To enable the CM574-RCOM to control the modem correctly, the following parameters always have to be configured:

- Commands are echoed by the modem
- Acknowledgements from the modem on
- Acknowledgements in plain text
- Break does not clear down connection
- MNP options off
- Data compression off (direct mode)
- Dialing with DTR off
- RTS/CTS handshake between modem and CM574-RCOM
- Automatic call acceptance off
- Escape character: &;;+’

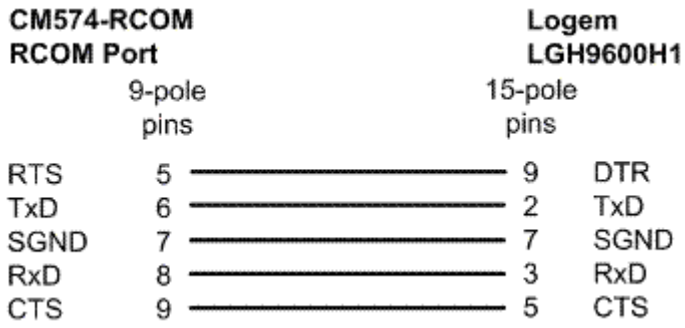
Special features

In "direct mode" the Logem LGH 9600H1 (in the following called LGH) cannot react on the sequence "+++". Consequently, switching from online to command mode for hang-up is not possible.

Therefore, the CM574-RCOM provides an operating mode enabling hang-up by means of the DTR signal (S1/108) of the modem. If the DTR signal is switched from active to passive, the modem hangs-up immediately and switches to the command mode.

Call acceptance and establishment of requested connections is only possible while the DTR signal is active.

Because the CM574-RCOM is not able to make an independent DTR signal available, the RTS signal is used (operating mode "RTS as DTR"). So the cable for connecting the Logem LGH 9600H1 to the CM574-RCOM is as follows:



Settings at the Logem LGH 9600H1

The following parameters have to be set permanently for the Logem LGH 9600H1, e.g. via a terminal directly connected to the Logem LGH 9600H1:

```
at&v
Version 2.02  D
F2 E1 L1 M1 Q0 X4 V1 P \Q2 \G0 \A1 \C0 \L0 \N1 \X1 \K0 \B3 %C1 %E1 %M0 &L0 &I0
&Y0 &X0 &G0 &M0 &C2 &D2 &R1 &S1
S00=000 S01=000 S02=043 S03=013 S04=010 S05=008 S07=100 S08=002 S10=002 S12=050
S20=255 S26=004 S28=000 S37=001 S39=017 S40=019 S45=000 S46=060 S50=002 S51=004
S60=000 S61=000 S80=000 S81=000 S100=042 S101=000 S102=000

OK
```

After entering the parameter "ATF2", ensure that the Logem LGH 9600H1 and the terminal are permanently set to 1200 Baud.

Save the parameters in the non-volatile memory of the Logem LGH 9600H1 by entering "AT&W"

The DIL switches at the Logem LGH 9600H1 have to be set in a way that the basic setting "0" is set in the software mode (all switches S5.1 to S5.5 at the front panel of the unit to "OFF").

Error codes

CAUTION!
For compatibility reasons, the error codes at output ERNO of the RCOM function blocks are not RCOM specific. To obtain the RCOM error code, 8000 hex has to be subtracted from the original value.

Various errors displayed by numbers can occur when using the RCOM Communication Module. The error number is indicated at output ERNO of the corresponding connection elements.

The errors can be divided into two groups: recoverable errors and fatal errors.

The recoverable errors include all errors which occur during transmission of telegrams (character losses, parity errors etc.). In some cases, the Communication Module attempts retransmission in case of transmission errors. The number of attempts can be specified via the parameter "Retransmissions" in the PLC configuration. Further information can be found in section Basic configuration of CM574-RCOM *Chapter 1.6.5.2.6.4.1 “Module parameters” on page 5907*.

In case of recoverable errors, the "RUN" LED flashes cyclic. The LED turns into normal operation mode ("ON") as soon as the error has been fixed.

Fatal errors

If fatal errors occur, the Communication Module terminates communication with the AC500 CPU and the RCOM partners. Fatal erros are indicated by the "ERR" LED.

A reset is required in order to reactivate the Communication Module, e.g. by switching it off and on again.

Table of error codes

The following table contains all possible error codes (hexadecimal and decimal), the error cause and the type of error (fatal or recoverable error).

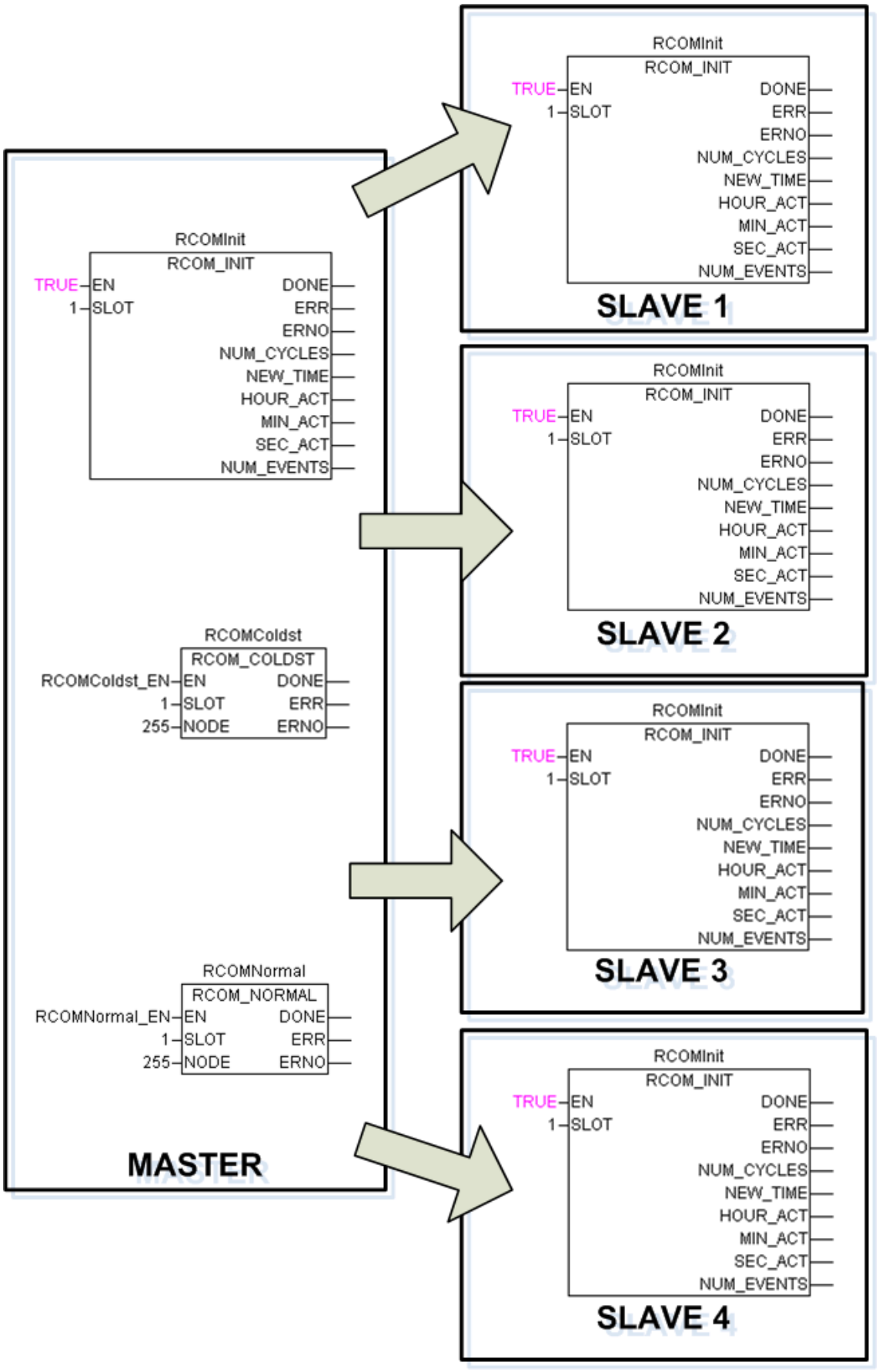
Error code		Cause
(hex)	(dec)	
0000	0	No error
0001	1	CPU does not respond (possible cause: missing connection element or CPU stopped)
1001	4097	Event queue full
2001	8193	Initialization error: Incorrect address (parameter NODE)
2002	8194	Initialization error: Incorrect address (parameter NODE)
2005	8197	Initialization error: Incorrect parity
2006	8198	Initialization error: Incorrect duplex mode
2007	8199	Initialization error: Incorrect line stab. Time (TLS)
2008	8200	Initialization error: Incorrect carrier delay
2009	8201	Initialization error: Incorrect character timeout
2010	8208	Initialization error: Incorrect turnaround time
2011	8209	Initialization error: Incorrect retransmissions
2012	8210	Initialization error: Incorrect number of preambles
2015	8213	Initialization error: Incorrect type of modem
2018	8216	Initialization error: Incorrect debug level
3001	12289	No CTS during transmission

Error code		Cause
(hex)	(dec)	
3002	12290	Timeout, no response telegram
3003	12291	Telegram error (incorrect length code)
3004	12292	Telegram error (incorrect checksum)
3005	12293	Incorrect slave responding
3006	12294	Telegram error (incorrect response code)
4001	16385	Service not known
4002	16386	Incorrect length entry (input LEN)
4003	16387	Incorrect length entry (input LEN)
4004	16388	Incorrect network number (currently unused)
4005	16389	Incorrect data set number (input ID)
4006	16390	Response telegram does not match service
4007	16391	Slave not normalized
4008	16392	Event queue blocked (not normalized)
4010	16400	Invalid time (connection element RCOM_CLOCK)
4020	16416	Slave responds: “application part not ready” (e.g. EN = FALSE)
4030	16432	Slave responds: “application part not ready” (e.g. EN = FALSE)
5000	20480	Communication Module not initialized (after reset) Remark: Requires restart of the AC500 PLC (abort and restart of the program
6001	24577	Telephone directory or setup data not found
6002	24578	Incorrect entry in telephone directory or setup data
6003	24579	No telephone modem planned
6004	24580	Modem not yet online (service started without RCOM_DIAL)
6005	24581	Modem already online (repeated RCOM_DIAL without RCOM_HANGUP)
FFFF	-1	Internal error (fatal)

Operator terminal

The second serial interface "CONSOLE" of the CM574-RCOM serves as a monitoring and debugging interface. The user can enter commands to get information about the protocol parameters or check modem connections and monitor the current RCOM messages.

The pin assignment to connect the interface to the COM-Port of a PC is shown below:


RCOMInit
RCOM_INIT
TRUE
EN
1
SLOT
DONE
ERR
ERNO
NUM_CYCLES
NEW_TIME
HOUR_ACT
MIN_ACT
SEC_ACT
NUM_EVENTS
SLAVE 1
SLAVE 2
SLAVE 3
SLAVE 4
RCOMColdst
RCOM_COLDST
RCOMColdst_EN
255
NODE
RCOMNormal
RCOM_NORMAL
RCOMNormal_EN
MASTER

It is recommended to provide several PCs for checking the data traffic via the console interfaces of the Communication Modules. Terminal programs are required (PCPLUS or Hyper Terminal; settings: 19200 Baud/s, 8 data bits, 1 stop bit, parity none).

Operator commands

A Communication Module can execute commands, if an according command is entered following the "OPERATOR>". E.g. "HELP" lists all available commands.

```
OPERATOR> help
available commands:

   help...................this text

   time...................show RCOM's system time

   event..................show event-queue

   rcom...................show RCOM parameters

   rcsw...................show RCOM status word

   show setup.............show setup-file

   show phone.............show telephone directory

   hangup.................hangup phone

   dial <slave>...........dial a slave

   mod <command>..........send a command to modem

   debug <level>..........show/set debug level
```

The following operator commands can be entered on the console:

HELP Displays a help text on the available commands.

TIME Displays the current RCOM system time.

EVENT Displays a table containing the events currently listed in the event queue. The table lists an event number, the event type (40: event with data set, 41: cold start event), data set number and length and the time when the event occurred.

RCOM Displays a table containing the current RCOM parameters. IMPORTANT: All times are specified in milliseconds. Some of the times are increased slightly by the Communication Module.

RCSW Displays the RCOM status word (RCSW).

SHOW SETUP Displays the setup data. Refer to section "Setups for dial-up modems".

SHOW PHONE Displays the telephone directory. Refer to section "Setups for dial-up modems".

HANGUP Hangs up the telephone (only if a dial-up modem is connected).

DIAL n Dials the telephone number stored in entry "n" in the telephone directory (only if a dial-up modem is connected).

MOD string Sends "string" to the modem (only if a dial-up modem is connected). The responses of the modem are displayed. Example: "MOD AT14 <ENTER>" -> the modem responds with a table of the most important parameters.

DEBUG<n> Sets the debug level to "n". This remains valid until the next initialization by the PLC. If you do not enter "n", the current debug level is displayed.

- In the first level, a message containing the most important parameters is output for each telegram received or transmitted by a slave and for each service started by the PLC.
- In the second level, the Communication Module outputs a message with each important action (including internal operations), e.g. to display received data words and transmitted telegrams, status changes in the RCSW (RCOM status word), status changes in the event queue, etc. The second debug level will probably be of interest in a few cases only. For testing (debugging) the PLC program, the first level is sufficient.

After initialization, the debug level can be set to "1" by entering the command

```
OPERATOR> debug 1
```

(corresponding to level 2). Deactivate the messages with

```
OPERATOR> debug 0
```

After commissioning, you should deactivate the messages for normal operation of the Communication Module.

Operator messages All operator messages have the following appearance:

typ-I-identification text

typ-W- identification text

typ-E- identification text

typ-F- identification text

Where:

- Type: Three letters indicating the origin of the message, i.e. "MST" for services performed by a master, "TEL" for telegrams received by a slave, "RPL" for responses transmitted by a slave, etc.
- I/W/E/F provides information on the type of message:
  - 'I':Information used to trace the sequence (debug levels 1 and 2).
  - 'W': Warning. Occurs, if e.g. telegrams arrive for which no connection element is planned. Warnings do not disturb the sequence in the Communication Module but do indicate a planning error.
  - 'E': An error message of the Communication Module, i.e. if an addressed slave does not respond. Errors disturb the Communication Module when processing the current service but can often be remedied by repeating the service. The "RUN" LED flashes cyclically in case of such an error. If it was possible to fix the error, the LED stops flashing and lights up continuously.
  - 'F': Fatal error, remedy not possible. Communication via RCOM to the PLC is aborted by the Communication Module and only operator entries are possible then. The "ERR" LED indicates that a fatal error has occurred.
- "identification": Abbreviation of the error message.
- "text": Actual message in plain text.

The following tables list the messages possibly output at the operator interface and their significance.

Certain messages contain designations for services that are specified in the following table.

Abbreviated	**Name of service**	**Triggered in the master with CONNECTION ELEMENT**	**Handled in the slave with CONNECTION ELEMENT**
Norm comm part	Normalize communictaion part	*1)	*3)
Quer comm part	Status query communication part	*1)	*3)
Cold start	Cold start	RCOM_COLDST	*3)
Warm start 1	Block all blocks	RCOM_WARMST	*3)
Warm start 2	Block unique blocks	*1)	*3)
Set clock	Set clock	RCOM_CLOCK	*3)
Norm user part	Normalize user part	*1)	*3)
Norm all blocks	Normalize all blocks	RCOM_NORMAL	*3)
Norm sep blocks	Normalize separate blocks	*1)	*3)
Write dataset	Write dataset	RCOM_TRANSMIT	RCOM_REC
Write cntrl	Write data to control register	*1)	*3)
Read dataset	Read dataset	RCOM_READ	RCOM_READ_SLV
Event request	Event request	RCOM_POLL	*3)
Repeat read	Repeat read command	*2)	*3)
Repeat write	Repeat write command	*2)	*3)
Dial	Dial up slave	RCOM_DIAL	*3)
Hangup	Hang up phone	RCOM_HANGUP	*3)

Remarks: *1) not used as master on CM574-RCOM

*2) performed automatically in case of transmission errors

*3) handled automatically in the CM574-RCOM

In the following table, the first column specifies the actual message of the Communication Module. The second column shows a description, the third column specifies the significance of the message, and the fourth column shows remarks indicating the cause of the error.

Message	Description	Significance	Remark
-EPL-E-CPUTO	Service timeout, no CPU reaction	An occurring event was not fetched by the CPU. RCOM_REC connection element missing	*1)
-EPL-I-EVENT	Event - ds: .., len: .., ..	Event arrived	
-EPL-I- CPUACC	Service accepted by CPU	Service accepted by CPU	
-EPL-I-SYSMES	System message	System event occurred (is ignored)	
-EPL-W-CPUREJ	Service rejected by CPU	Event rejected by CPU (EN in case of RCOM_REC = FALSE)	
-ERR-F-FATAL	Fatal error, communication canceled	Fatal error occurred, communication terminated	
-EVT-I-BLOCK	Blocking event queue	Blocking event queue	
-EVT-I-CLEAR	Clearing event queue	Clearing event queue	
-EVT-I-DEBLK	Deblocking event queue	Enable event queue	
-EVT-I-GET	Event queue empty	Event queue is empty	
-EVT-I-GET	Getting event	Fetching event from queue	
-EVT-I-PUT	Putting event	Inserting event in queue	
-EVT-W-PUT	Event-queue full	Event queue is full	
-INI-E-COMGRP	Error reading com group, code	Error reading special flags for communication processors	*2)
-INI-E-EXTINI	External init error, ...	Error during initialization	*2)
-INI-E-GRESI	Error resetting CPU communication, ...	It was not possible to reset system bus communication	*2)
-INI-E-MOD	Error initializing telephone modem, ...	It was not possible to initialize the telephone modem	*3)
-INI-E-OCCUP	Error occupying CPU, ...	It was not possible to assign the PLC	*2)
-INI-E-OPER	Error initializing operator, ...	It was not possible to initialize the commissioning interface	*3)

Message	Description	Significance	Remark
-INI-E-PARAM	Error in parameter, ...	Error in parameter	*1)
-INI-E-RCOM	Error during RCOM-init, ...	It was not possible to initialize the RCOM mechanism	
-INI-E-RCOM	Error initializing RCOM-channel, ...	It was not possible to initialize the RCOM interface	*3)
-INI-E-READ	Error reading parameters, ...	It was not possible to read the RCOM parameters	*2)
-INI-E-RS	Error reading RUN/STOP, ...	It was not possible to read the RUN/STOP switch	*2)
-INI-E-RSINI	Error initializing PLC communication, ...	It was not possible to initialize the system bus communication	*2)
-INI-I-CHECK	Checking RCOM-parameters	Checking RCOM parameters	
-INI-I-COMGRP	Waiting for valid com group	Waiting for valid communication area (in special flag for communication processors)	
-INI-I-EXTINI	External init done	External initialization completed	
-INI-I-EXTINI	Waiting for external init	Waiting for external initialization	
-INI-I-GRES	Resetting CPU communication	Resetting system bus communication	
-INI-I-RUN	Waiting for RUN-switch	Waiting for RUN/STOP switch = RUN	
-KPM-I-EXIT	Exit main loop, reason code ...	Quitting RCOM communication; cause ...	*4)
-KPM-I-GOODM	Good morning!!	It is midnight	
-KPM-I-LCNT	Lifecount ...	Cycle counter set to ...	
-MOD-E-DIAL	Cannot connect	It was not possible to establish the dial-up connection	*5)
-MOD-E-DIAL	Modem already connected	Modem is already on line	*5)
-MOD-E-DIAL	No modem available (modem type = 0)	No modem planned	*1)
-MOD-E-ENTRY	Bad entry in phone file	Entry error in telephone directory	*5)
-MOD-E-HANGUP	Cannot hang up	It was not possible to hang up	*3)
-MOD-E-INIT	Error during modem init	Error during modem initialization	*3)
-MOD-E-NOFILE	No valid files on EEPROM	No valid setup/telephone files on EEPROM	*5)

Message	Description	Significance	Remark
-MOD-E-RING	No modem available (modem type = 0)	No modem planned	*1)
-MOD-I-ANS	Answer ...	Modem response: ...	
-MOD-I-ANSCMP	Compare ... - ...	Comparing modem response with ...	
-MOD-I-ANSCMP	-MOD-I-ANSCMP	Strings are identical	
-MOD-I-DIAL	Connected	It was possible to set up connection (dial)	
-MOD-I-DIAL	Dialing node ...	Dialing station ...	
-MOD-I-DIAL	Ring (...)	Dialing station ...	
-MOD-I-HANGUP	Hangup phone	Hanging up	
MOD-I-INIT	Answer ...	Modem response:...	
-MOD-I-INIT	Init modem (...)	Initializing modem	
-MOD-I-RING	Connected	It was possible to set up connection (going off hook)	
-MOD-I-RING	Ring received	Telephone ringing	
-MOD-W-DIAL	Retry ...	Retry dialing	
-MOD-W-RING	Cannot connect	It was not possible to set up connection (going off hook)	*5)
-MST-E-ADDR	Error reading reply (data), ...	It was not possible to read the header from the response	*6)
-MST-E-LCODE	Illegal length-code in reply	Incorrect length code in response	*6)
-MST-E-POSTA	Error reading reply (checksum/post-ambles),...	It was not pos-sible to read the checksum/trailer from the response	*6)
-MST-E-PREA	Error reading reply (preambles), ...	It was not possible to read the preamble (leader) from the response	*6)
-MST-E-RES	Command not reset by CPU	Command not acknowledged by CPU	*7)
-MST-E-SEND	Error sending tele-gram, ...	It was not possible to transmit the job	*3)
-MST-E-SUM	Checksum error in reply	Checksum error in response	*6)
-MST-I-POLL	Checking slave %3d	Checking whether slave ... has dialed	
-MST-I-POLL	Polled slave %3d, result ...	It was possible to poll the slave ...; result ...	
-MST-I-RESULT	..., result ...	Service ... terminated; result ...	
-MST-I-SERV	...	Service ... started	

Message	Description	Significance	Remark
-MST-W-NOSRV	No services within hang up time	No service within the hang up time, now hanging up	*1)
-MST-W-RETRY	Retry ...	Retrying job telegram	*6)
-OPR-E-CMD	Unknown command ...	Unknown operator command	
-OPR-E-OCCUP	Error occupying CPU, ...	It was not possible to assign the CPU	*2)
-OPR-E-RELEA	Error releasing CPU, ...	It was not possible to release the CPU	*2)
-OPR-I-INIT	Operator init done	Initialization commissioning interface terminated	
-PLC-E-GETEND	Error reading line area from PLC, ...	Error in system bus communication	*2)
-PLC-E-GETEND	Error reading rx area from PLC, ...	Error in system bus communication	
-PLC-E-GETEND	Error reading tx area from PLC, ...	Error in system bus communication	
-PLC-E-GETREQ	Error reading com group from PLC, ...	Error in system bus communication	
-PLC-E-GETREQ	Error reading line area from PLC, ...	Error in system bus communication	
-PLC-E-GETREQ	Error reading rx area from PLC, ...	Error in system bus communication	
-PLC-E-GETREQ	Error reading tx area from PLC, ...	Error in system bus communication	
-PLC-E-SETEND	Error writing control block to PLC, ...	Error in system bus communication	
-PLC-E-SETEND	Error writing rx area to PLC, ...	Error in system bus communication	
-PLC-E-SETEND	Error writing tx area to PLC, ...	Error in system bus communication	
-PLC-E-SETREQ	Error writing control block to PLC, ...	Error in system bus communication	
-PLC-E-SETREQ	Error writing rx area to PLC, ...	Error in system bus communication	
-PLC-E-SETREQ	Error writing tx area to PLC, ...	Error in system bus communication	
-PLC-W-GETEND	Timeout while reading com group from PLC, abort: ...	Timeout during system bus communication, error during abort: ...	*2)
-PLC-W-GETEND	Timeout while reading line area from PLC, abort: ...	Timeout during system bus communication, error during abort: ...	
-PLC-W-GETEND	Timeout while reading rx area from PLC, abort: ...	Timeout during system bus communication, error during abort: ...	

Message	Description	Significance	Remark
-PLC-W-GETEND	Timeout while reading tx area from PLC, abort: ...	Timeout during system bus communication, error during abort: ...	
-PLC-W-SETEND	Timeout while writing control block to PLC, abort: ...	Timeout during system bus communication, error during abort: ...	
-PLC-W-SETEND	Timeout while writing rx area to PLC, abort: ...	Timeout during system bus communication, error during abort: ...	
-PLC-W-SETEND	Timeout while writing tx area to PLC, abort: ...	Timeout during system bus communication, error during abort: ...	
-RCS-I-SET RCSW	Set to ...	RCOM status word set to ...	
-RDS-E-CPUTO	Service timeout, no CPU reaction	PLC not responding to read job (RCOM_READ_SLV connection element missing)	
-RDS-I-CPUACC	Service accepted by CPU	Service read data set accepted by CPU	
-RDS-W-CPUREJ	Service rejected by CPU, ...	Service read data set rejected by CPU (EN with RCOM_READ_SLV = FALSE)	
-RPL-E-REPLY	Internal error: ...	Internal error .... when setting up response	*7)
-RPL-E-REPLY	Reply error: ...	Error in response telegram	*5),*6)
-RPL-I-LEN	... data bytes in reply	... data bytes in response	
-RPL-I-REPLY	...	Response: ...	
-RPL-I-REPLY	..., result ...	Error ... in response	*6)
-SCL-I-TIME	Date: %ld, time %ld	New RCOM time ... arrived	
-SLV-E-ADDR	Error reading telegram (address), ...	It was not possible to read the address from the job	*6)
-SLV-E-BREAK	Error checking for break	It was not possible to check BREAK	*3)
-SLV-E-DATA	Error reading telegram (data), ...	It was not possible to read data from the job	*6)
-SLV-E-HEADR	Error reading telegram (header), ...	It was not possible to read the header from the job	*6)
-SLV-E-LCODE	Illegal length-code in telegram	Incorrect length code in job	*6)

Message	Description	Significance	Remark
-SLV-E-POSTA	Error reading telegram (checksum/postambles), ...	Incorrect checksum/ trailer in job	*6)
-SLV-E-PREA	Error reading telegram (preambles), ...	It was not possible to read the preamble (leader) from the job	*6)
-SLV-E-RES	Command not reset by CPU	Command was not cancelled by CPU	*7)
-SLV-E-SEND	Error sending reply, ...	It was not possible to send response	*3)
-SLV-E-SUM	Checksum error	Checksum error	*6)
-SLV-I-ADR	Not my address (...)	Job not for me but for slave ...	
-SLV-I-NOREP	No reply sent (broadcast request)	No response transmitted (broadcast job)	
-SLV-I-RESULT	..., result ...	Service terminated; result ...	
-SLV-I-SERV	...	Service detected	
-SLV-W-MODE	Event queue blocked	Event queue barred (normalization missing)	*1)
-SLV-W-MODE	Slave mode program	Data transmission barred (normalization missing)	*1)
-SLV-W-NOSRV	No services within hang-up time	No jobs arrived within the hang-up time, no hanging up	
-TEL-E-SERV	Internal error: ...	Internal error	*8)
-TEL-I-LEN	... data bytes in request	... data bytes in job	
-TEL-I-SERV	...	Service ... detected	
-TEL-I-SERV	... Ds: ..., Len: ...	Service ... detected; data set ...; length ...	
-WDS-E-CPUTO	Service timeout, no CPU reaction	PLC not responding to write job (RCOM_REC connection element missing)	
-WDS-I-CPUACC	Service accepted by CPU	Service Write data set accepted by CPU	
-WDS-W-CPUREJ	Service rejected by CPU, ...	Service Write data set rejected by CPU (EN with RCOM_REC = FALSE)	

Remarks

*1) A planning error has probably occurred. Check whether all required connection elements are present and whether the correct parameters have been assigned to them.

*2) Error during system bus communication. Leads to fatal errors, if not remedied automatically. May be triggered by PS501, e.g. when transmitting programs.

*3) There is probably a fault in the cable. Check the wiring of RTS and CTS.

*4) The following causes are possible:

- Reinitialization was performed by the CPU module (RCOM_INIT connection element started)
- RUN/STOP switch was set to STOP
- A fatal error occurred

*5) There is probably an error in modem control. Check the module "dial-up modem" in your PLC configuration and its sub modules for errors concerning the setup or the phone book.

*6) A transmission error occurred. Check all RCOM parameters and timeout times in the PLC configuration.

*7) This occurs if the CPU is set to STOP.

*8) Internal error. Attempt to reset the unit and reinitialize it.

Planning examples

Example 1: Direct connection

This example shows how to realize data transmission between two RCOM Communication Modules using a direct serial connection in operation mode RCOM+. The transmission speed of the connection in the example is set to 9600 Baud/s (odd parity).

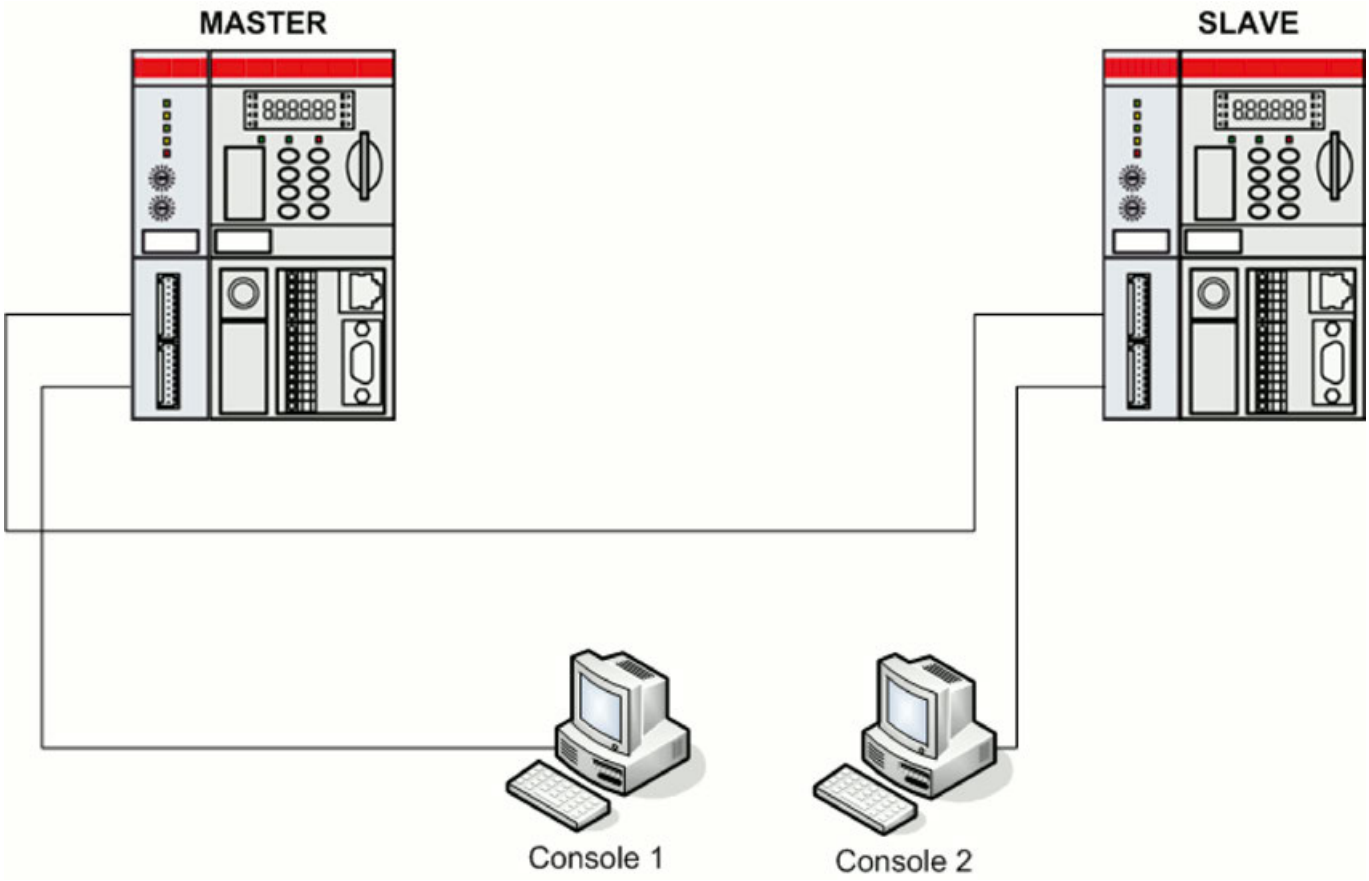
The set up of the AC500 PLCs will be described as well as the projecting of the master and slave devices.

The corresponding projects can be found in the examples folder:

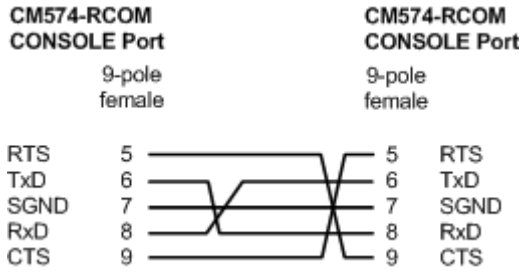
- RCOM Master: AC500_RCOM_DIRECT_MASTER_PM5xx_Vyy.pro
- RCOM Slave: AC500_RCOM_DIRECT_SLAVE_PM5xx_Vyy.pro

The basic hardware setup consists of two AC500 PLCs, each using

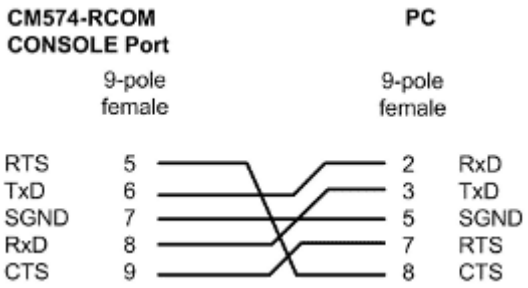
- 1 processor module
- 1 terminal base with at least one communication module slot
- 1 communication module CM574-RCOM.



To connect the communication modules, a serial cable with the following pin assignment is required (cross cable):



In addition to that, a cable is recommended that connects the CONSOLE port to a PC with a terminal emulation software to view protocol information or error messages.



Master-Slave-Arrangement

Configuring the master

To configure the CM574-RCOM as a master device, the parameter "Operation mode" must be set to "RCOM+ Master". The address or node number must be set to 0 while "Parity" is set to "Odd". All other parameters can remain default.

If the CONSOLE port is used (recommended), the parameter "Debug level" of module "CONSOLE" should be set to "Level 2" to obtain detailed information and error messages of the protocol.

Configuring the slave

To configure the CM574-RCOM as a slave device the parameter "Operation mode" must be set to "RCOM+ Slave". The parameter address must be set to a value greater than 0. In the example select 1 for the slave. "Parity" is set to "Odd". All other parameter can remain default.

If the CONSOLE port is used (recommended) the parameter "Debug level" of module "CONSOLE" should be set to "Level 2" to obtain detailed information and error messages of the protocol.

Implementation of the user program

The example shows how to write a dataset with word length 2 to the slave and how to read a dataset from the slave with word length 14.

Implementation of the master

The first step is the implementation of an instance of the function block RCOM_INIT that is used to initialize the RCOM communication module at the specified slot (1 in this case).

```
0001 RCOM_INIT(EN:= TRUE, SLOT:= 1);                    (*coupler initialization*)
0002 IF RCOM_INIT.DONE AND NOT RCOM_INIT.ERR THEN       (*check if done successfully*)
0003     byStep                          := 1;
0004 ELSIF RCOM_INIT.DONE AND RCOM_INIT.ERR THEN
0005     byStep                          := 0;
0006 END_IF
0007 IF byStep = 0 THEN                                 (*on init error disable all RCOM fb's
```

The function block must be called cyclically to ensure proper functionality of the RCOM communication module. The outputs DONE and ERR must be checked to retrieve the current state of initialization. Once the output DONE is TRUE and ERR is FALSE, the communication module is initialized and ready for further operation.

After a successful initialization the slave must go through the RCOM specific initialization sequence that consists of a cold start, a warm start and a normalization. These operations will ensure that the slave is ready to process incoming transmission requests of the master. Input NODE of all function blocks used is set to 1 to address the directly connected slave device.

```
0014 IF byStep = 1 THEN                                    (*do cold start*)
0015     RCOM_COLDST(EN:= TRUE, SLOT:= 1, NODE := 1);
0016     IF RCOM_COLDST.DONE AND NOT RCOM_COLDST.ERR THEN
0017         byStep                  := 2;
0018     ELSIF RCOM_COLDST.DONE AND RCOM_COLDST.ERR THEN
0019         byStep                  := 1;
0020     END_IF
0021 END_IF
0022 IF byStep = 2 THEN                                    (*do warm start*)
0023     RCOM_WARMST(EN := TRUE, SLOT := 1, NODE := 1);
0024     IF RCOM_WARMST.DONE AND NOT RCOM_WARMST.ERR THEN
0025         byStep                  := 3;
0026     ELSIF RCOM_WARMST.DONE AND RCOM_WARMST.ERR THEN
0027         byStep                  := 1;
0028     END_IF
0029 END_IF
0030 IF byStep = 3 THEN                                    (*do normalization*)
0031     RCOM_NORMAL(EN := TRUE, SLOT := 1, NODE := 1);
0032     IF RCOM_NORMAL.DONE AND NOT RCOM_NORMAL.ERR THEN
0033         byStep                  := 4;
0034     ELSIF RCOM_NORMAL.DONE AND RCOM_NORMAL.ERR THEN
0035         byStep                  := 1;
0036     END_IF
0037 END_IF
```

Once the normalization step has finished successfully, the calls of the data transmission function blocks can be started. In the example, the first step of data transmission is the execution of a write request to transfer a data set of word length 2 to the connected slave. The function block RCOM_TRANSMIT is used to realize this operation. The data that is written to the slave is located in the word array arwDataOut[0..1]. The input DATA of the function block is set to the array's address using the ADR operator. The input ID is set to 2 what specifies the data set number which must equal a corresponding data set on the slave side. The length of the data set is set to value 2 at input LEN.

After this step a read request will be started by using the function block RCOM_READ. The requested data set is specified at input ID (3 in this case). A corresponding data set is projected in the slave project described later. The received data is written to word array arwDataIn[0..13]. Input DATA must be set to the address of the array via the ADR operator. The length in words is specified at input LEN.

```
0038 IF byStep = 4 THEN                                    (*write request (data set 2)*)
0039     RCOM_TRANSMIT_DS_2(EN := TRUE, SLOT := 1, NODE := 1, ID := 2, DATA := ADR(arwDataOut[0]), LEN := 2);
0040     IF RCOM_TRANSMIT_DS_2.DONE AND NOT RCOM_TRANSMIT_DS_2.ERR THEN
0041         byStep                  := 5;
0042     ELSIF RCOM_TRANSMIT_DS_2.DONE AND RCOM_TRANSMIT_DS_2.ERR THEN
0043         byStep                  := 3;
0044     END_IF
0045 END_IF
0046 IF byStep = 5 THEN                                    (*read request (data set 3)*)
0047     RCOM_READ_DS_3(EN := TRUE, SLOT := 1, NODE := 1, ID := 3, DATA := ADR(arwDataIn[0]), LEN := 14);
0048     IF RCOM_READ_DS_3.DONE AND NOT RCOM_READ_DS_3.ERR THEN
0049         byStep                  := 4;
0050     ELSIF RCOM_READ_DS_3.DONE AND RCOM_READ_DS_3.ERR THEN
0051         byStep                  := 3;
0052     END_IF
0053 END_IF
```

In case of an error during data transmission, the slave must be normalized to reset its state and to maintain its ability to process incoming request telegrams of the master.

Implementation of the slave

As shown in the master implementation, the slave also requires the creation and call of an instance of the function block RCOM_INIT.

```
0001 RCOM_INIT(EN:= TRUE, SLOT:= 1);                          (*coupler initialization*)
0002 IF RCOM_INIT.DONE AND NOT RCOM_INIT.ERR THEN             (*check if done successfully*)
0003     byStep                          := 1;
0004 ELSIF RCOM_INIT.DONE AND RCOM_INIT.ERR THEN
0005     byStep                          := 0;
0006 END_IF
0007 IF byStep = 0 THEN                                       (*on init error disable all RCOM fb's
```

The 2 requests that are sent by the master require a corresponding implementation of the function blocks RCOM_READ_SLV and RCOM_REC which react to the read or write request telegrams of the master.

An instance of function block RCOM_READ_SLV with input ID set to 3 reacts to the read request projected in the master. The data that is read from the master is located in word array arwDataOut[0..13]. Its address is set at input DATA of the function block.

An instance of function block RCOM_REC with input ID set to 2 reacts to the incoming write request projected in the master. The data contained in the master's telegram is copied to word array arwDataIn[0..1]. Its address is set at input DATA of the function block.

```
0011 IF byStep = 1 THEN                                       (*normal operation: call RCOM fb's*)
0012     RCOM_REC_DS_2(EN:= TRUE, SLOT:= 1, NODE := 0, ID := 2, DATA := ADR(arwDataIn[0]));
0013     IF RCOM_REC_DS_2.NEW THEN
0014         dwInCnt                     := dwInCnt + 1;
0015     END_IF
0016     RCOM_READ_SLV_DS_3(EN := TRUE, SLOT:= 1, NODE:= 1, ID:= 3, DATA:= ADR(arwDataOut[0]));
0017     IF RCOM_READ_SLV_DS_3.NEW THEN
0018         dwOutCnt                    := dwOutCnt + 1;
0019     END_IF
0020 END_IF
```

Example 2: Dial-Up connection

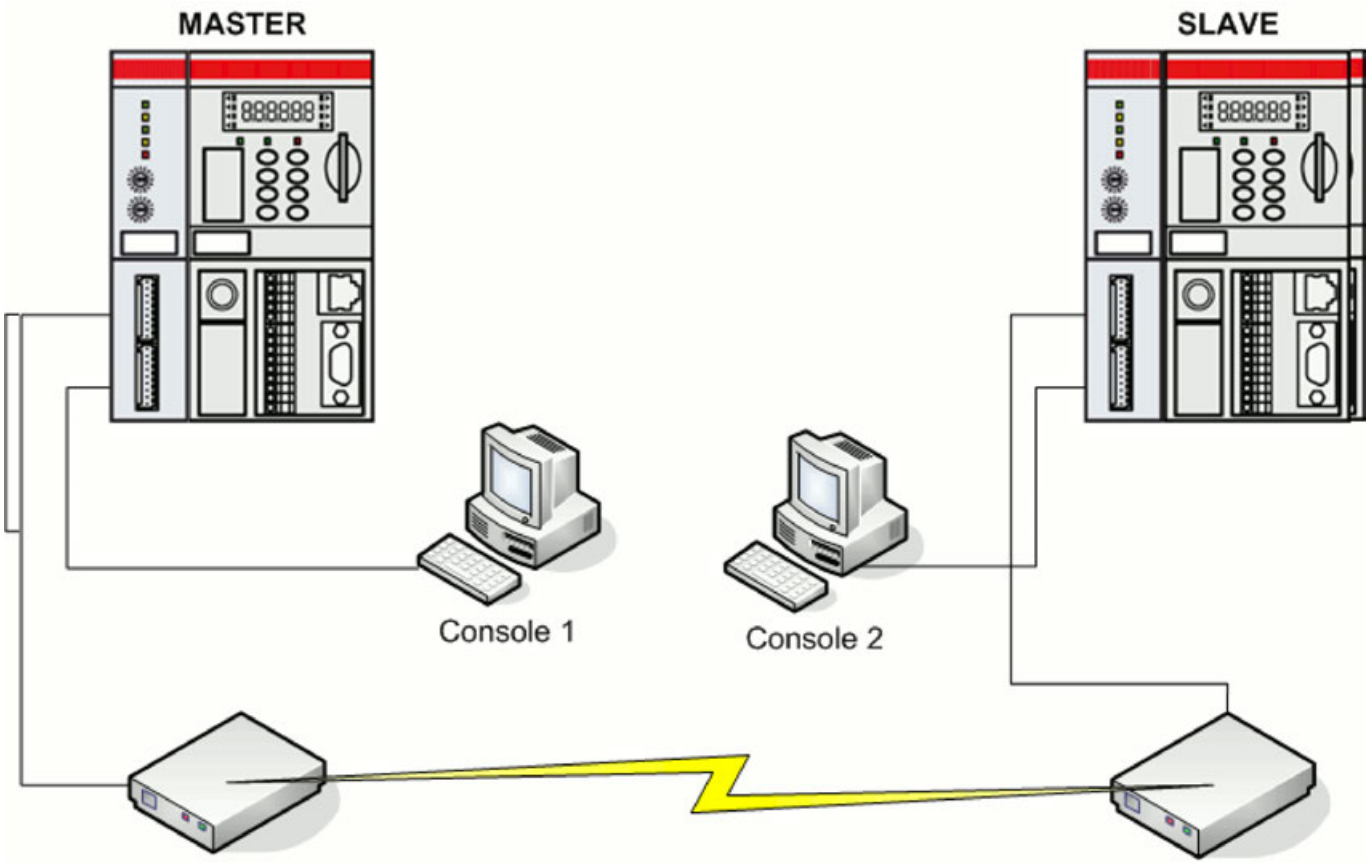
This example shows how to realize data transmission between two RCOM Communication Modules using a modem based dial-up connection in operation mode RCOM+. The transmission speed of the connection in the example is set to 9600 Baud/s (no parity).

The corresponding projects can be found in the examples folder:

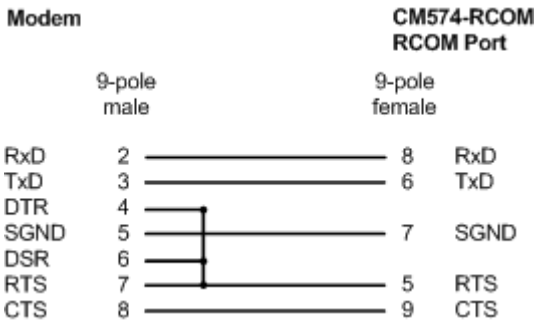
- RCOM Master: AC500_RCOM_DIAL_UP_MASTER_PM5xx_Vyy.pro
- RCOM Slave: AC500_RCOM_DIAL_UP_SLAVE_PM5xx_Vyy.pro

The basic hardware setup consists of two AC500 PLCs, each using

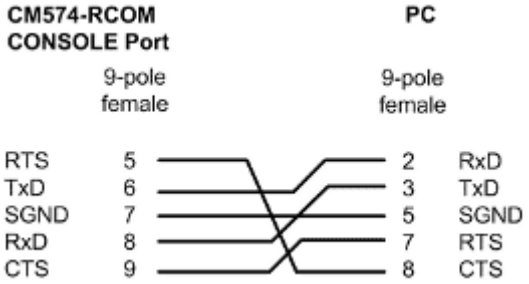
- 1 processor module
- 1 terminal base with at least 1 communication module slot
- 1 communication module CM574-RCOM



The connection between a modem and a CM574-RCOM communication module requires a cable with the following pin assignment:



In addition to that a cable is recommended that connects the CONSOLE port to a PC with a terminal emulation software to view protocol information or error messages.



Master-Slave-Arrangement

Configuring the master

To configure the CM574-RCOM as a master device the parameter "Operation mode" must be set to "RCOM Master". The address or node number must be set to 0 while "Parity" should be "None". Ensure that "Hayes compatible dial modem" is selected as "Type of modem". The "Character timeout" should be set to 50 ms. The value for the "Turnaround time" should be increased to 6000 ms.

If the CONSOLE port is used (recommended) the parameter "Debug level" of module "CONSOLE" should be set to "Level 2" to obtain detailed information and error messages of the protocol.

Configuring the slave

To configure the CM574-RCOM as a slave device the parameter "Operation mode" must be set to "RCOM Slave". The parameter address must be set to a value greater than 0. In the example select 1 for the slave. "Parity" should be "None". Ensure that "Hayes compatible dial modem" is selected as "Type of modem". The "Character timeout" should be set to 50 ms. The value for the "Turnaround time" should be increased to 6000 ms.

If the CONSOLE port is used (recommended) the parameter "Debug level" of module "CONSOLE" should be set to "Level 2" to obtain detailed information and error messages of the protocol.

Configuring the modem

The example uses ABB H&IT HSM-ECO remote modems that provide the ability to use both operation modes, RCOM and RCOM+.

Setting up the phone book

For both projects the telephone numbers of the RCOM devices must be configured. To use a dial-up modem, a corresponding module has to be appended to the RCOM/RCOM+ module in the PLC configuration as described in the PLC configuration steps of the CM574-RCOM.

For consistency reasons, the phone book of the master as well as the phone book of the slave(s) must be equal.

To add a new entry to the phone book, append a telephone number to the modem. The tab "Module parameters" provides the possibility to set up the corresponding dial prefix identifier and the corresponding telephone number of the node that should be configured.

Implementation of the User program

The example shows how to write a dataset with word length 2 to the slave and how to read a dataset from the slave with word length 14. In addition to that event polling is used to transfer a dataset from the slave to the master.

Implementation of the master

The first step is the implementation of an instance of function block RCOM_INIT that is used to initialize the RCOM communication module at the specified slot (1 in this case). To process incoming events of the slave a RCOM_REC function block is projected and called every cycle as soon as the initialization of the local Communication Module has been finished.

```
0001 RCOM_INIT(EN:= TRUE, SLOT:= 1);                                    (*coupler initialization*)
0002 IF RCOM_INIT.DONE AND NOT RCOM_INIT.ERR THEN                       (*check if done successfully*)
0003     RCOM_REC_DS_4(EN := TRUE, SLOT:= 1, NODE:= 1, ID := 4, DATA := ADR(arwDataRec[0]));
0004     IF RCOM_REC_DS_4.NEW THEN
0005         dwRecCnt                := dwRecCnt + 1;
0006     END_IF
0007     IF bTrigger THEN
0008         bTrigger                := FALSE;
0009         RCOM_DIAL_1(EN := FALSE, SLOT := 1);
0010         RCOM_COLDST_1(EN := FALSE, SLOT := 1);
0011         RCOM_WARMST_1(EN := FALSE, SLOT := 1);
0012         RCOM_NORMAL_1(EN := FALSE, SLOT := 1);
0013         RCOM_TRANSMIT_DS_2(EN := FALSE, SLOT := 1);
0014         RCOM_READ_DS_3(EN := FALSE, SLOT := 1);
0015         RCOM_POLL_1(EN := FALSE, SLOT := 1);
0016         byStep                  := 1;
0017         byStepNext              := 2;
0018     END_IF
0019 ELSIF RCOM_INIT.DONE AND RCOM_INIT.ERR THEN
0020     byStep                      := 0;
0021 END_IF
```

The function block must be called cyclically to ensure proper functionality of the RCOM communication module. The outputs DONE and ERR must be checked to get the initialization state. Once output DONE is TRUE and ERR is FALSE the communication module is initialized and ready for further operation.

After a successful initialization the variable bTrigger must be manually set to start the dialing process represented by function block RCOM_DIAL. Otherwise the master waits for incoming calls of the slave to show how the automatic polling process is executed.

```
0023 IF byStep = 1 THEN                                                 (*dial node 1*)
0024     RCOM_DIAL_1(EN := TRUE, SLOT := 1, NODE := 1);
0025     IF RCOM_DIAL_1.DONE AND NOT RCOM_DIAL_1.ERR THEN
0026         byStep                  := byStepNext;
0027     ELSIF RCOM_DIAL_1.DONE AND RCOM_DIAL_1.ERR THEN
0028         byStep                  := 1;
0029         byStepNext              := 2;
0030     END_IF
0031 END_IF
```

In case of a finished dialing process triggered by the master, the RCOM initialization function blocks for cold start, warm start and normalization are started in the next steps to initialize the slave and prepare it for further request telegrams that are used for data transmission.

```
0040 IF byStep = 2 THEN                                    (*do cold start*)
0041     RCOM_COLDST_1(EN:= TRUE, SLOT:= 1, NODE := 1);
0042     IF RCOM_COLDST_1.DONE AND NOT RCOM_COLDST_1.ERR THEN
0043         byStep                  := 3;
0044     ELSIF RCOM_COLDST_1.DONE AND RCOM_COLDST_1.ERR THEN
0045         byStepNext              := 2;
0046         byStep                  := 8;
0047     END_IF
0048 END_IF
0049 IF byStep = 3 THEN                                    (*do warm start*)
0050     RCOM_WARMST_1(EN := TRUE, SLOT := 1, NODE := 1);
0051     IF RCOM_WARMST_1.DONE AND NOT RCOM_WARMST_1.ERR THEN
0052         byStep                  := 4;
0053     ELSIF RCOM_WARMST_1.DONE AND RCOM_WARMST_1.ERR THEN
0054         byStepNext              := 2;
0055         byStep                  := 8;
0056     END_IF
0057 END_IF
0058 IF byStep = 4 THEN                                    (*do normalization*)
0059     RCOM_NORMAL_1(EN := TRUE, SLOT := 1, NODE := 1);
0060     IF RCOM_NORMAL_1.DONE AND NOT RCOM_NORMAL_1.ERR THEN
0061         byStep                  := 5;
0062     ELSIF RCOM_NORMAL_1.DONE AND RCOM_NORMAL_1.ERR THEN
0063         byStepNext              := 2;
0064         byStep                  := 8;
0065     END_IF
0066 END_IF
```

Once the normalization step has finished successfully the data transmission can be executed. In the example the first step of data transmission is the call of a write request to transfer a data set of word length 2 to the connected slave. The function block RCOM_TRANSMIT is used to realize this operation. The data that is written to the slave is located in the word array arwDataOut[0..1]. The input DATA of the function block is set to the array's address using the ADR operator. The input ID is set to 2 what specifies the data set number which must equal a corresponding data set on the slave side. The length of the data set is set to value 2 at input LEN.

After this step a read request will be started by using the function block RCOM_READ. The requested data set is specified at input ID (3 in this case). A corresponding data set is projected in the slave project that is also described in this documentation. The received data is written to word array arwDataIn[0..13]. Input DATA must be set to the address of the array via the ADR operator. The length in words is specified at input LEN.

The next step triggers an event poll (RCOM_POLL) on the slave to check for new events that will be read from the slave's event queue and processed by local RCOM_REC function blocks.

```
0067 IF byStep = 5 THEN                                          (*write request (data set 2)*)
0068     RCOM_TRANSMIT_DS_2(EN := TRUE, SLOT := 1, NODE := 1, ID := 2, DATA := ADR(arwDataOut[0]), LEN := 2);
0069     IF RCOM_TRANSMIT_DS_2.DONE AND NOT RCOM_TRANSMIT_DS_2.ERR THEN
0070         byStep                  := 6;
0071     ELSIF RCOM_TRANSMIT_DS_2.DONE AND RCOM_TRANSMIT_DS_2.ERR THEN
0072         byStepNext              := 4;
0073         byStep                  := 8;
0074     END_IF
0075 END_IF
0076 IF byStep = 6 THEN                                          (*read request (data set 3)*)
0077     RCOM_READ_DS_3(EN := TRUE, SLOT := 1, NODE := 1, ID := 3, DATA := ADR(arwDataIn[0]), LEN := 14);
0078     IF RCOM_READ_DS_3.DONE AND NOT RCOM_READ_DS_3.ERR THEN
0079         byStep                  := 7;
0080     ELSIF RCOM_READ_DS_3.DONE AND RCOM_READ_DS_3.ERR THEN
0081         byStepNext              := 4;
0082         byStep                  := 8;
0083     END_IF
0084 END_IF
0085 IF byStep = 7 THEN                                          (*do event poll*)
0086     RCOM_POLL_1(EN := TRUE, SLOT := 1, NODE := 1);
0087     IF RCOM_POLL_1.DONE AND NOT RCOM_POLL_1.ERR THEN
0088         byStep                  := 5;
0089     ELSIF RCOM_POLL_1.DONE AND RCOM_POLL_1.ERR THEN
0090         byStepNext              := 4;
0091         byStep                  := 8;
0092     END_IF
0093 END_IF
```

In case of an error the, master hangs up the phone (function block RCOM_HANGUP) and waits for incoming calls or a new dialing process that is started manually.

```
0094 IF byStep = 8 THEN                                          (*hangup phone*)
0095     RCOM_HANGUP(EN := TRUE, SLOT := 1, NODE := 1);
0096     IF RCOM_HANGUP.DONE THEN
0097         byStep                  := 0;
0098     END_IF
0099 END_IF
```

Implementation of the slave

As shown in the master implementation the slave also requires the creation and call of an instance of the function block RCOM_INIT.

The two requests that will be sent by the master require corresponding implementation of the function blocks RCOM_READ_SLV and RCOM_REC which will react to incoming read or write request telegrams of the master:

- An instance of function block RCOM_READ_SLV with input ID set to 3 reacts to the read request projected in the master. The data that will be read from the master is located in word array arwDataOut[0..13]. Its address is set at input DATA of the function block.
- An instance of function block RCOM_REC with input ID set to 2 reacts to the incoming write request projected in the master. The data contained in the master's telegram will be copied to word array arwDataIn[0..1]. Its address is set at input DATA of the function block.

Both function blocks are called cyclically to ensure proper reaction to the incoming telegrams. In case of data reception or the completion of a read request the output NEW of the corresponding function block is set to TRUE (until next call of the function block).

```
0001 RCOM_INIT(EN:= TRUE, SLOT:= 1);                                    (*coupler initialization*)
0002 IF RCOM_INIT.DONE AND NOT RCOM_INIT.ERR THEN                       (*check if done successfully*)
0003     byStep                          := 1;
0004 ELSIF RCOM_INIT.DONE AND RCOM_INIT.ERR THEN
0005     byStep                          := 0;
0006 END_IF
0007 IF byStep = 0 THEN                                                 (*on init error disable all RCOM fb's*)
0008     RCOM_REC_DS_2(EN := FALSE);
0009     RCOM_READ_SLV_DS_3(EN := FALSE);
0010     RcOM_TRANSMIT_DS_4(EN := FALSE);
0011 END_IF
0012 IF byStep = 1 THEN                                                 (*normal operation: call RCOM fb's*)
0013     RCOM_REC_DS_2(EN:= TRUE, SLOT:= 1, NODE := 0, ID := 2, DATA := ADR(arwDataIn[0]));
0014     IF RCOM_REC_DS_2.NEW THEN
0015         dwInCnt                     := dwInCnt + 1;
0016     END_IF
0017     RCOM_READ_SLV_DS_3(EN := TRUE, SLOT:= 1, NODE:= 1, ID:= 3, DATA:= ADR(arwDataOut[0]));
0018     IF RCOM_READ_SLV_DS_3.NEW THEN
0019         dwOutCnt                    := dwOutCnt + 1;
0020     END_IF
0021     IF bPutEvent THEN                                              (*manually triggered: put event to queue*)
0022         RCOM_TRANSMIT_DS_4(EN := TRUE, SLOT:= 1, NODE:= 0, ID:= 4, DATA:= ADR(arwDataSnd[0]), LEN:= 14);
0023         IF RCOM_TRANSMIT_DS_4.DONE THEN
0024             bPutEvent               := FALSE;
0025             IF NOT RCOM_TRANSMIT_DS_4.ERR THEN
0026                 bDialMaster         := TRUE;
0027             END_IF
0028             RCOM_TRANSMIT_DS_4(EN := FALSE);
0029         END_IF
0030     END_IF
0031 END_IF
```

The variable bPutEvent can be set to TRUE to add an event that contains data for the master to the slave's event queue with the function block RCOM_TRANSMIT. After the function block is done the program dials the master to trigger an event poll in case of a successful connection to the master.

1.6.4.3 System technology of the communication interface modules

1.6.4.3.1 Modbus communication interface module

Overview

The Modbus TCP communication interface module CI52x-MODTCP is used as decentralized I/O module in Modbus TCP networks. The network connection is performed via 2 RJ45 connectors which are integrated in the terminal unit.

CI521-MODTCP I/O channels properties:

- 4 analog inputs (1.0...1.3)
- 2 analog outputs (1.5...1.6)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	from the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the last BYTE of the IP (00h to FFh)
Analog inputs	4 (configurable via software)
Analog outputs	2 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)

CI522-MODTCP I/O channels properties:

- 8 digital configurable inputs/outputs in 1 group (1.0...1.7)
- 8 digital inputs 24 V DC in 1 group (2.0...2.7)
- 8 digital outputs 24 V DC in 1 group (3.0...3.7)

Functionality

Parameter	Value
Interface	Ethernet
Protocol	Modbus TCP
Power supply	from the process supply voltage UP
Supply of the electronic circuitry of the I/O expansion modules attached	Through the I/O bus interface (I/O bus)
Rotary switches	For setting the last BYTE of the IP (00h to FFh)
Configurable digital inputs/outputs	8 (configurable via software)
Digital inputs	8 (24 V DC; delay time configurable via software)
Digital outputs	8 (24 V DC, 0.5 A max.)
LED displays	For system displays, signal states, errors and power supply
External supply voltage	Via terminals ZP, UP and UP3 (process supply voltage 24 V DC)

The inputs/outputs are galvanically isolated from the Ethernet network. There is no potential separation between the channels.

The configuration of the inputs/outputs is performed by software.

For usage in enhanced ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Modbus TCP registers

Register layout for CI52x-MODTCP

The registers can be divided in 4 sections:

- Information data section 0x0000 to 0x0D50 (for acyclic use)
- I/O data and diagnosis section 0x0FFA to 0x2B00 (for cyclic use)
- Parameter data section 0x3000 to 0x3B00 (for acyclic use)
- Special functionality section 0x5A00 to 0x6A00 (for acyclic use)

Information data section (Acyclic data)

The information data section can be used to read out common and module specific information.

This section is read only.

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
0	Device and FW information CI	3	x
50	Production data CI	3	x
100	Device and FW information 1. EXP	3	x
125	Device and FW information 1. Hot swap terminal unit	3 *)	x
150	Production data 1. EXP	3	x
175	Production data 1. Hot swap terminal unit	3 *)	x
...	...		x
A00	Device and FW information 10. EXP	3	x
A25	Device and FW information 10. Hot swap terminal unit	3 *)	x
A50	Production data 10. EXP	3	x
A75	Production data 10. Hot swap terminal unit	3 *)	x
D00	Common device information	3	x

*) supported from CI52x firmware version V3.2.0 (device index F0)

This section can be divided again in two sections:

- The module specific section (containing information for each module CI52x-MODTCP and expansion modules and hot swap terminal units)
- The common device information block

Module specific information registers

For each module (CI52x device, expansion modules and hot swap terminal units) the following data can be read out:

- Device and FW information
  This section consists of 20 WORDs per module and contains information on each module using the following structure:

Data	DATA TYPE	Description
Module ID	WORD	The module ID of the requested module
Module name	ARRAY [1..10] OF BYTE	The module name of the requested module
Version 1st processor	ARRAY [1..4] OF BYTE	The version of the 1st processor of the requested module
Version 2nd processor	ARRAY [1..4] OF BYTE	The version of the 2nd processor of the requested module
Version 3rd processor	ARRAY [1..4] OF BYTE	The version of the 3rd processor of the requested module

Data	DATA TYPE	Description
Version 4th processor	ARRAY [1..4] OF BYTE	The version of the 4th processor of the requested module
Hardware version [1])	ARRAY [1..4] OF BYTE	The hardware version of the 4 processors
Reserved	ARRAY [1..8] OF BYTE ARRAY [1..4] OF BYTE [2])	Reserved
Number input data	WORD	Number of input data of the requested module in BYTES
Number output data	WORD	Number of output data of the requested module in BYTES

[1]) supported from CI52x firmware version V3.2.0 (device index F0)

[2]) from CI52x firmware version V3.2.0 (device index F0) “Reserved” is ARRAY [1..4] OF BYTE

- Production / Traceability data:
  This section consists of 25 WORDs per module and contains the traceability data for each module using following structure:
  - Article number: Byte 01..15
  - Index: Byte 16..17
  - Name: Byte 18..29
  - Production date: Byte 30..33
  - Key number: Byte 34..38
  - Site: Byte 39..40
  - Year: Byte 41..42
  - Serial number: Byte 41..50 (The serial number implies the year)
- Production / Traceability data from CI5x2 firmware version V3.2.0 (device index F0):
  This section consists of 26 WORDs per module and contains the traceability data for each module using following structure:
  - Article number: Byte 01..15
  - Index: Byte 16..17
  - Name: Byte 18..31
  - Production date: Byte 32..35
  - Key number: Byte 36..40
  - Site: Byte 41..42
  - Year: Byte 43..44
  - Serial number: Byte 42..52 (The serial number implies the year)

Common device information registers

Common device information block

This section consists of 80 WORDs (90 WORDs from CI52x firmware version V3.2.0 (device index F0)) and contains cluster wide information (CI52x device and connected expansion modules using the following structure:

Data	DATA TYPE	Description
Device state	BYTE	The actual state of the device: 0: STATE_PREOP (device booting) 1: STATE_OPERATION (device in operational, no bus supervision active) 2: STATE_ERROR (device detected a bus error, bus supervision active) 3: STATE_IP_ERROR (the device has a IP address error) 4: STATE_CYCLIC_OPERATION (device in operational, bus supervision active)
Parameter state	BYTE	The actual parameter state of the device: 0: PARA_STATE_NO_PARA (the device has no parameters) 1: PARA_STATE_PARA_ACTIVE (parameterization process running) 2: PARA_STATE_PARA_DONE (the uses valid parameters) 3: PARA_STATE_ERROR (The device has invalid
Module ID CI device	WORD	Module ID of the CI52x device itself
Module ID 1st expansion	WORD	Module ID of the 1st connected expansion module
Module ID 2nd expansion	WORD	Module ID of the 2nd connected expansion module
...		
Module ID 10th expansion	WORD	Module ID of the 10th connected expansion module
Expansion bus error count	DWORD	Global telegram error count over all expansion modules
Good count onboard I/O	DWORD	Telegram good count onboard I/Os
Good count 1st expansion	DWORD	Telegram good count 1st expansion module
Good count 2nd expansion	DWORD	Telegram good count 2nd expansion module
...		
Good count 10th expansion	DWORD	Telegram good count 10th expansion module
Error count onboard I/O	DWORD	Telegram error count onboard I/Os
Error count 1st expansion	DWORD	Telegram error count 1st expansion module
Error count 2nd expansion	DWORD	Telegram error count 2nd expansion module
...		
Error count 10th expansion	DWORD	Telegram error count 10th expansion module
Input address onboard I/O	WORD	Modbus TCP register address for inputs of the onboard I/Os
Input address 1st expansion	WORD	Modbus TCP register address for inputs of the 1st expansion module
Input address 2nd expansion	WORD	Modbus TCP register address for inputs of the 2nd expansion module

Data	DATA TYPE	Description
...		
Input address 10th expansion	WORD	Modbus TCP register address for inputs of the 10th expansion module
Output address onboard I/O	WORD	Modbus TCP register address for outputs of the onboard I/Os
Output address 1st expansion	WORD	Modbus TCP register address for outputs of the 1st expansion module
Output address 2nd expansion	WORD	Modbus TCP register address for outputs of the 2nd expansion module
...		
Output address 10th expansion	WORD	Modbus TCP register address for outputs of the 10th expansion module
Module ID 1st hot swap terminal unit *)	WORD	Module ID of the 1st connected hot swap terminal unit *)
Module ID 2nd hot swap terminal unit *)	WORD	Module ID of the 2nd connected hot swap terminal unit *)
...		
Module ID 10th hot swap terminal unit *)	WORD	Module ID of the 10th connected hot swap terminal unit *)

*) supported from CI52x firmware version V3.2.0 (device index F0)

I/O / Process data and diagnosis section (Cyclic data)

Table 710: The cyclic data section for CI52x-MODTCP

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
FCE *)	Module state	3,4, 23	x
FFA	Diagnosis	3,4, 23	x
1000	Inputs CI	3, 4, 23	x
1100	Inputs 1.EXP	3, 4, 23	x
...	...		x
1A00	Inputs 10.EXP	3, 4, 23	x
2000	Outputs CI	3, 23	6, 16, 23
2100	Outputs 1.EXP	3, 23	6, 16, 23
...	...		
2A00	Outputs 10.EXP	3, 23	6, 16, 23
2B00	Dummy output	3, 23	6, 16, 23

*) supported from CI52x firmware version V3.2.0 (device index F0)

This section can be divided again in three sections:

- Module state (containing the state of connected expansion modules and hot swap terminal units)
- Diagnosis data (containing diagnosis data in AC500 specific format)
- Process data (containing I/O data)

Module state

The module state section consists of 44 WORDs and contains the module state of connected expansion modules and hot swap terminal units using the following structure:

Data	DATA TYPE	Description
Module ID	WORD	Module ID of the CI52x
Expected module ID	WORD	Expected (configured) module ID of the CI52x
Module state	BYTE	The current module state of the CI52x: 0: NO_MOD (no module detected) 1: MOD_INIT (module detected, module is in initialization phase) 2: MOD_RUN (module detected and running or in failsafe state, input data are valid) 3: WRONG_MOD (wrong module detected, module ID doesn’t match expected module ID) 4: MOD_REMOVED (module removed or defective on hot swap terminal unit, no communication to module possible) 5: MOD_ERROR (module defective on hot swap terminal unit, no communication to module possible) 6: MOD_LOST (lost communication to module on not hot swap capable terminal unit) 7: UNKNOWN (module detected but not configured)
Diagnosis flag	BYTE	Diagnosis flag for the CI52x: 0: NO_DIAG (no diagnosis available from CI52x I/O cards) 1: DIAG_AVAILABLE (diagnosis available for CI52x I/O cards)
Terminal unit state	BYTE	Terminal unit state for the CI52x: 0: NO_HOTSWAP_TU (not hot swap terminal unit detected) 1: HOTSWAP_TU_RUNNING (hot swap terminal unit detected and working) 2: HOTSWAP_TU_ERROR (hot swap terminal unit detected, but communication errors for hot swap terminal unit detected)
Parameter state	BYTE	Parameter state of the CI52x: 0: NO_PARA (module is in initialization phase and not ready for parameterization) 1: WAIT_PARA (module awaits parameterization) 2: PARA_RUN (parameterization running) 3: LEN_ERR (length of parameters not correct) 4: ID_ERR (module ID inside parameters not correct) 5: PARA_DONE (parameterization finished without errors)
Module ID	WORD	Module ID of the 1st connected expansion module

Data	DATA TYPE	Description
Expected module ID	WORD	Expected (configured) module ID of the 1st connected expansion module
Module state	BYTE	The current module state of the 1st connected expansion module
Diagnosis flag	BYTE	Diagnosis flag for the 1st connected expansion module 0: NO_DIAG (no diagnosis available for expansion module) 1: DIAG_AVAILABLE (diagnosis available for expansion module)
Terminal unit state	BYTE	Terminal unit state for the 1st connected expansion module
Parameter state	BYTE	Parameter state of the 1st connected expansion module
...		
Module ID	WORD	Module ID of the 10th connected expansion module
Expected module ID	WORD	Expected (configured) module ID of the 10th connected expansion module
Module state	BYTE	The current module state of the 10th connected expansion module
Diagnosis flag	BYTE	Diagnosis flag for the 10th connected expansion module
Terminal unit state	BYTE	Terminal unit state for the 10th connected expansion module
Parameter state	BYTE	Parameter state of the 10th connected expansion module

Diagnosis data

The diagnosis data section contains one diagnosis message with the following structure (according to AC500 diagnosis):

Byte Number	Description	Possible Values
1	Diagnosis Byte, slot number	31 = CI52x-MODTCP (e. g. error at integrated 8 DI / 8 DO)
		1 = 1st connected S500 I/O Module
		...
		10 = 10th connected S500 I/O Module
2	Diagnosis Byte, module number	According to the I/O bus specification passed on by modules to the fieldbus master
3	Diagnosis Byte, channel	According to the I/O bus specification passed on by modules to the fieldbus master
4	Diagnosis Byte, error code	According to the I/O bus specification Bit 7 and Bit 6, coded error class 0 = E1 1 = E2 2 = E3 3 = E4 Bit 0 to Bit 5, coded error description
5	Diagnosis Byte, flags	According to the I/O bus specification Bit 7: 1 = coming error Bit 6: 1 = leaving error
6	Reserved	0

If a diagnosis message is read out, the next one will be automatically filled in.

If no more diagnosis messages are available the buffer will be reset to zero.

This ensures that each diagnosis message can be delivered to the Modbus TCP client/slave and no diagnosis will be lost.

I/O data

The I/O data section can use two different formats according to the module parameter “I/O Mapping Structure” (see ↬ *Chapter 1.6.2 “Device specifications” on page 3785* for details).

- Fixed I/O mapping
  In case of fixed I/O mapping each module has a predefined register range for each Inputs and Outputs.
- Dynamic I/O mapping
  In case of dynamic I/O mapping the mapping is build according to the actual configuration.

The dummy output at the end of the I/O data section can be used to retrigger the bus supervision and has no effect on the HW outputs.

Fixed I/O mapping

In case of fixed I/O mapping the following predefined register table is used:

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
1000	Inputs CI	3, 4, 23	x
1100	Inputs 1.EXP	3, 4, 23	x
...	...		x
1A00	Inputs 10.EXP	3, 4, 23	x
2000	Outputs CI	3, 23	6, 16, 23
2100	Outputs 1.EXP	3, 23	6, 16, 23
...	...		
2A00	Outputs 10.EXP	3, 23	6, 16, 23
2B00	Dummy output	3, 23	6, 16, 23

If a certain expansion module has no inputs or outputs the corresponding registers remain empty.

Dynamic I/O mapping

In case of dynamic mapping only the start addresses of inputs and outputs are predefined:

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
1000	Inputs CI	3, 4, 23	x
...	...		x
2000	Outputs CI	3, 23	6, 16, 23
...	...		
2B00	Dummy output	3, 23	6, 16, 23

The register addresses of the connected expansion modules are calculated dynamically based on the number of inputs and outputs of the previous modules (each module starts directly on the next register after the previous module).

The register addresses of each module can be read out via the common device register (see ↳ *Chapter 1.6.4.3.1.2.2.2 “Common device information registers” on page 5654*).

Comparative example

The difference between fixed I/O mapping and dynamic I/O mapping is shown in the following table.

For this comparison a cluster with CI522, AX522, DC532, AX521, DC523, DC532, AO523, AI523, DI524, AX522 and DC523 is used.

Fixed Mapping					Dynamic Mapping			
Register (hex)	**Description**	**Type**	**Data**		**Register (hex)**	**Description**	**Type**	**Data**
1000	Inputs CI	8 DC, 8 DI, FC	4 BYTE + 4 WORD		1000	Inputs CI	8 DC, 8 DI, FC	4 BYTE + 4 WORD
1100	Inputs AX522	8 AI	8 WORD		1006	Inputs AX522	8 AI	8 WORD
1200	Inputs DC532	16 DI, 16 DC	4 BYTE		100E	Inputs DC532	16 DI, 16 DC	4 BYTE
1300	Inputs AX521	4 AI	4 WORD		1010	Inputs AX521	4 AI	4 WORD
1400	Inputs DC523	24 DC	3 BYTE		1014	Inputs DC523	24 DC	3 BYTE
1500	Inputs DC532	16 DI, 16 DC	4 BYTE		1016	Inputs DC532	16 DI, 16 DC	4 BYTE
1600	Inputs AO523	---	---		---	Inputs AO523	---	---
1700	Inputs AI523	16AI	16 WORD		1018	Inputs AI523	16AI	16 WORD
1800	Inputs DI524	32 DI	4 BYTE		1028	Inputs DI524	32 DI	4 BYTE
1900	Inputs AX522	8 AI	8 WORD		102A	Inputs AX522	8 AI	8 WORD
1A00	Inputs DC523	24 DC	3 BYTE		1032	Inputs DC523	24 DC	3 BYTE
2000	Outputs CI	8 DC, 8DO, FC	4 BYTE + 8 WORD		2000	Outputs CI	8 DC, 8DO, FC	4 BYTE + 8 WORD
2100	Outputs AX522	8 AO	8 WORD		200A	Outputs AX522	8 AO	8 WORD
2200	Outputs DC532	16 DC	2 BYTE		2012	Outputs DC532	16 DC	2 BYTE
2300	Outputs AX521	4 AO	4 WORD		2013	Outputs AX521	4 AO	4 WORD
2400	Outputs DC523	24 DC	3 BYTE		2017	Outputs DC523	24 DC	3 BYTE
2500	Outputs DC532	16 DC	2 BYTE		2019	Outputs DC532	16 DC	2 BYTE
2600	Outputs AO523	16 AO	16 WORD		201A	Outputs AO523	16 AO	16 WORD
2700	Outputs AI523	---	---		---	Outputs AI523	---	---
2800	Outputs DI524	---	---		---	Outputs DI524	---	---
2900	Outputs AX522	8 AO	8 WORD		202A	Outputs AX522	8 AO	8 WORD
2A00	Outputs DC523	24 DC	3 BYTE		2032	Outputs DC523	24 DC	3 BYTE

Process data structure CI521-MODTCP

Table 711: I/O data (Inputs 19 BYTEs)

Signal	DATA TYPE	Description
AI0	WORD	Input value of the 1st analog input
AI1	WORD	Input value of the 2nd analog input
AI2	WORD	Input value of the 3rd analog input
AI3	WORD	Input value of the 4th analog input
DI	BYTE	Input value of the DI channels
Fast counter actual value counter 1	DWORD	↳ *Chapter 1.6.4.1.10.1 "Fast counters in AC500 devices" on page 5498*
Fast counter actual value counter 2	DWORD	
Fast counter state counter 1	BYTE	
Fast counter state counter 2	BYTE	

Table 712: I/O data (Outputs 23 BYTEs)

Signal	DATA TYPE	Description
AO0	WORD	Output value of the 1st analog output
AO1	WORD	Output value of the 2nd analog output
DO	BYTE	Output value of the DO channels
Fast counter start value counter 1	DWORD	↳ *Chapter 1.6.4.1.10.1 "Fast counters in AC500 devices" on page 5498*
Fast counter end value counter 1	DWORD	
Fast counter start value counter 2	DWORD	
Fast counter end value counter 2	DWORD	
Fast counter control counter 1	BYTE	
Fast counter control counter 2	BYTE	

Process Data Structure CI522-MODTCP

Table 713: I/O data (Inputs 12 BYTEs)

Signal	DATA TYPE	Description
DC	BYTE	Input value of the DC channels
DI	BYTE	Input value of the DI channels
Fast counter actual value counter 1	DWORD	↳ *Chapter 1.6.4.1.10.1 "Fast counters in AC500 devices" on page 5498*
Fast counter actual value counter 2	DWORD	
Fast counter state counter 1	BYTE	
Fast counter state counter 2	BYTE	

Table 714: I/O data (Outputs 20 BYTEs)

Signal	DATA TYPE	Description
DC	BYTE	Output value of the DC channels
DO	BYTE	Output value of the DO channels
Fast counter start value counter 1	DWORD	↳ *Chapter 1.6.4.1.10.1 "Fast counters in AC500 devices" on page 5498*
Fast counter end value counter 1	DWORD	
Fast counter start value counter 2	DWORD	
Fast counter end value counter 2	DWORD	
Fast counter control counter 1	BYTE	
Fast counter control counter 2	BYTE	

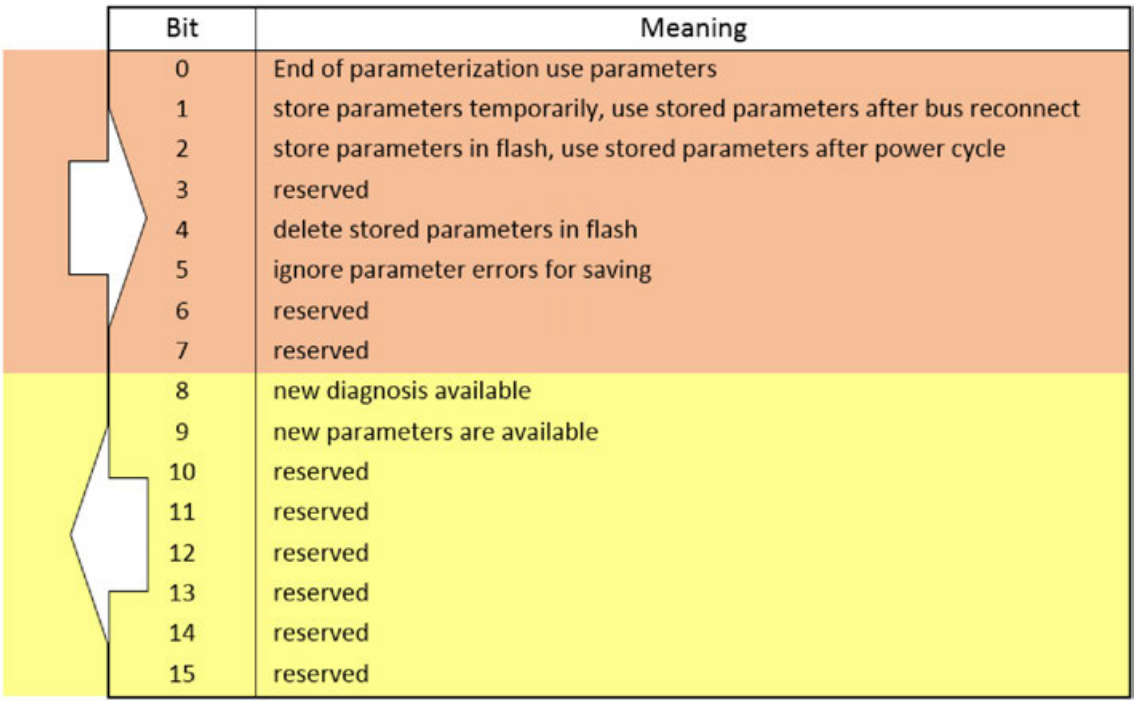
Parameter data (Acyclic data)

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
3000	Parameters CI	3	6, 16
3080	Stored parameters CI	3	x
3100	Parameters 1. EXP	3	6, 16
3180	Stored parameters 10. EXP	3	x
...			
3A00	Parameters 10. EXP	3	6, 16
3A80	Stored parameters 10. EXP	3	x
3B00	controlword/statusword	3	6, 16

For each connected module the following parameter data are defined (the parameters are represented as ARRAY OF BYTE):

- Actual used parameter for each module
  In these sections the actual parameters are stored. This section is also used to write parameters to the module (For a description on how to parameterize see ↳ *Chapter 1.6.4.3.1.3.2 "Parameterization" on page 5670*).
- Stored parameters for each module
  If the module has stored nonvolatile parameters these can be read out using the corresponding registers.

The controlword/statusword is used to trigger a parameterization process. The single bits have the following meaning:



Bit	Meaning
0	End of parameterization use parameters
1	store parameters temporarily, use stored parameters after bus reconnect
2	store parameters in flash, use stored parameters after power cycle
3	reserved
4	delete stored parameters in flash
5	ignore parameter errors for saving
6	reserved
7	reserved
8	new diagnosis available
9	new parameters are available
10	reserved
11	reserved
12	reserved
13	reserved
14	reserved
15	reserved

The direction of the first 8 bits is client to server (master to slave).

The direction of the second 8 bits is server to client (slave to master). A description of the bits can be found in chapter behavior ↳ *Chapter 1.6.4.3.1.3.2 “Parameterization” on page 5670*.

The parameter register sections (actual and stored parameters) have the structure as explained in the of the corresponding module ↳ *Chapter 1.6.2 “Device specifications” on page 3785*.

Short description of the CI521-MODTCP parameters

Parameter	Single parameter index	Description	Additional Info
0		Module ID (high Byte)	Fixed, must be 16#1C
1		Module ID (low Byte)	Fixed, must be 16#E8
2		Ignore Module	Reserved, must be 0
3		Length of following parameter block	Fixed, must be 16#3F
4	0	Error LED / Failsafe	See ↳ *Chapter 1.6.2 “Device specifications” on page 3785*
5	1	Master IP Byte 0	IP Address for write restrictions (↳ *“Configurable write restriction” on page 5672*)
6		Master IP Byte 1	
7		Master IP Byte 2	
8		Master IP Byte 3	
9	2	Master IP 1 Byte 0	IP Address for write restrictions (↳ *“Configurable write restriction” on page 5672*)
10		Master IP 1 Byte 1	
11		Master IP 1 Byte 2	
12		Master IP 1 Byte 3	
13	3	Master IP 2 Byte 0	IP Address for write restrictions (↳ *“Configurable write restriction” on page 5672*)
14		Master IP 2 Byte 1	

Parameter	Single parameter index	Description	Additional Info
15		Master IP 2 Byte 2	
16		Master IP 2 Byte 3	
17	4	Master IP 3 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
18		Master IP 3 Byte 1	
19		Master IP 3 Byte 2	
20		Master IP 3 Byte 3	
21	5	Master IP 4 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
22		Master IP 4 Byte 1	
23		Master IP 4 Byte 2	
24		Master IP 4 Byte 3	
25	6	Master IP 5 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
26		Master IP 5 Byte 1	
27		Master IP 5 Byte 2	
28		Master IP 5 Byte 3	
29	7	Master IP 6 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
30		Master IP 6 Byte 1	
31		Master IP 6 Byte 2	
32		Master IP 6 Byte 3	
33	8	Master IP 7 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
34		Master IP 7 Byte 1	
36		Master IP 7 Byte 2	
36		Master IP 7 Byte 3	
37	9	Timeout	Timeout for bus supervision in 10ms steps if set to 0 no bus supervision is active
38	10 (read only)	I/O Mapping Structure	See *Chapter 1.6.2 “Device specifications” on page 3785*
39	11	Reserved	Reserved, must be 0
40	12	Reserved	Reserved, must be 0
41	13	Reserved	Reserved, must be 0
42	14	Check supply	See *Chapter 1.6.2 “Device specifications” on page 3785*
43	15	Analog data format	Reserved, must be 0
44	16	Input delay	See *Chapter 1.6.2 “Device specifications” on page 3785*
46	17	Fast counter	
46	18	Short circuit detection	
47	19	Behavior binary outputs at com. fault	
48	20	Substitute value binary outputs	
49	21	Overvoltage monitoring	

Parameter	Single parameter index	Description	Additional Info
50	22	Behavior analog outputs	
51	23	Channel Config AI0	
52	24	Check Channel AI0	
53	25	Channel Config AI1	
54	26	Check Channel AI1	
55	27	Channel Config AI2	
56	28	Check Channel AI2	
57	29	Channel Config AI3	
58	30	Check Channel AI3	
59	31	Channel Config AO0	
60	32	Check Channel AO0	
61	33	Substitute value AO0 (high Byte)	
62		Substitute value AO0 (low Byte)	
63	34	Channel Config AO1	
64	35	Check Channel AO1	
65	36	Substitute value AO1 (high Byte)	
66		Substitute value AO1 (low Byte)	

Short description of the CI522-MODTCP parameters

Parameter	Single parameter index	Description	Additional Info
0		Module ID (high Byte)	Fixed, must be 16#1C
1		Module ID (low Byte)	Fixed, must be 16#ED
2		Ignore Module	Reserved, must be 0
3		Length of following parameter block	Fixed, must be 16#2F
4	0	Error LED / Failsafe	See *Chapter 1.6.2 “Device specifications” on page 3785*
5	1	Master IP Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
6		Master IP Byte 1	
7		Master IP Byte 2	
8		Master IP Byte 3	
9	2	Master IP 1 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
10		Master IP 1 Byte 1	
11		Master IP 1 Byte 2	
12		Master IP 1 Byte 3	
13	3	Master IP 2 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
14		Master IP 2 Byte 1	

Parameter	Single parameter index	Description	Additional Info
15		Master IP 2 Byte 2	
16		Master IP 2 Byte 3	
17	4	Master IP 3 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
18		Master IP 3 Byte 1	
19		Master IP 3 Byte 2	
20		Master IP 3 Byte 3	
21	5	Master IP 4 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
22		Master IP 4 Byte 1	
23		Master IP 4 Byte 2	
24		Master IP 4 Byte 3	
25	6	Master IP 5 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
26		Master IP 5 Byte 1	
27		Master IP 5 Byte 2	
28		Master IP 5 Byte 3	
29	7	Master IP 6 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
30		Master IP 6 Byte 1	
31		Master IP 6 Byte 2	
32		Master IP 6 Byte 3	
33	8	Master IP 7 Byte 0	IP Address for write restrictions (*“Configurable write restriction” on page 5672*)
34		Master IP 7 Byte 1	
36		Master IP 7 Byte 2	
36		Master IP 7 Byte 3	
37	2	Timeout	Timeout for bus supervision in 10ms steps if set to 0 no bus supervision is active
38	3 (read only)	I/O Mapping Structure	See *Chapter 1.6.2 “Device specifications” on page 3785*
39	4	Reserved	Reserved, must be 0
40	5	Reserved	
41	6	Reserved	
42	7	Check supply	See *Chapter 1.6.2 “Device specifications” on page 3785*
43	8	Input delay	
44	9	Fast counter	See *Chapter 1.6.2 “Device specifications” on page 3785*
46	10	Short circuit detection	
46	11	Behavior binary outputs at com. fault	
47	12	Substitute value binary outputs (high byte)	
48		Substitute value binary outputs (low byte)	

Parameter	Single parameter index	Description	Additional Info
49	13	Voltage feedback monitoring	
50	14	Overvoltage monitoring	

Parameters of connected expansion modules

The parameters of the connected expansion modules are represented as byte array (the parameters valid for “CPU” in the ↳ *Chapter 1.6.2 “Device specifications” on page 3785* of the corresponding module are used):

Parameter	Description	Additional Info
0	Module ID (high byte)	Fixed, see ↳ *Chapter 1.6.2 “Device specifications” on page 3785* of corresponding module (the module ID of FBP is used)
1	Module ID (low byte)	Fixed, see of corresponding module (the module ID of FBP is used) ↳ *Chapter 1.6.2 “Device specifications” on page 3785*
2	Ignore module	Reserved must be 0
3	Length of following parameter block	Fixed, see ↳ *Chapter 1.6.2 “Device specifications” on page 3785* of corresponding module
4...	The rest of the parameter are described in the corresponding module	

Special functionality

This section contains special services like firmware update or single parameterization.

Register (hex)	Description	Readable by Modbus function code	Writeable by Modbus function code
4000	Firmware download	3	16
4100	Firmware download state	3	x
5000	Write single parameterization of CI	x	16
5100	Write single parameterization of 1. EXP	x	16
...			
5A00	Write single parameterization of 10. EXP	x	16
6000	Read single parameterization of CI	3	16
6100	Read single parameterization of 1. EXP	3	16
...			
6A00	Read single parameterization of 10. EXP	3	16

Behavior

IP address assignment

The delivery IP address of the CI52x-MODTCP is 192.168.0.xx (xx is the hardware address switch position of the device.

The devices support BOOTP, DHCP and fixed IP address setting (these can be set individual or together). If BOOTP and DHCP are enabled the following priority takes place:

- If DHCP configuration fails, the device will fall back to BOOTP.
- In case of a BOOTP failure, the fixed IP address will be used.

A new IP address (or changing of BOOTP and DHCP) can be set in two different ways:

- With the address switches of the corresponding module
- With the ↬ *Chapter 1.6.5.2.2.2.2 “Configuration of the IP settings with the IP configuration tool” on page 5816*

Using the address switches

With the address switches only the last byte of the IP address can be changed.

The IP address can only be set via the address switches in case of factory default or in case of the last byte of the IP address is set to zero with the ↬ *Chapter 1.6.5.2.2.2.2 “Configuration of the IP settings with the IP configuration tool” on page 5816*. The not allowed IP addresses are mapped as followed:

- Address switch position 255 is mapped to fixed IP 192.168.0.254 independent of other stored settings (by IP Configuration Tool).

  This is a backup so the module can always get a valid IP address and can be configured by the IP Configuration Tool.
- Address switch position 0 is mapped to last byte equal 1 and DHCP enabled.

Using the IP configuration tool

With the *Chapter 1.6.5.2.2.2.2 “Configuration of the IP settings with the IP configuration tool” on page 5816* a network scan can be executed, and the found devices can be assigned with new settings, e.g. enable BOOTP or DHCP and set a new fixed IP. If the last byte of the IP address of the CI52x-MODTCP devices is set to 0 with the IP Configuration Tool the address switch position is used instead (see *Chapter 1.6.4.3.1.3.1.1 “Using the address switches” on page 5669*).

Parameterization

The parameterization is done via the corresponding registers explained in the Modbus TCP registers *Chapter 1.6.4.3.1.2.4 “Parameter data (Acyclic data)” on page 5663*.

In addition to that the parameters can be directly transferred via Automation Builder (see documentation of Automation Builder for that).

There are two different parameter sections with different behavior.

Actual used parameters

After startup this section contains the following data:

- Default parameters (only module id and parameter length set all others zero) if no valid stored parameters are available (no or invalid parameters stored).
- Actual used / stored parameters if valid parameters are stored nonvolatile.

These parameters can be read out and changed by reading or writing of the corresponding registers, but will not be used automatically after writing them, the use of new written parameters has to be triggered by writing the parameter control word with the corresponding bits set (see below).

Stored parameters

This section always contains a copy of the nonvolatile stored parameters, if no parameters are stored nonvolatile this sections will be 0.

Controlword/statusword parameter

This parameter can be used to trigger and save new parameters.

The direction of the first 8 bit is client to server (master to slave). The direction of the second 8 bits is server to client (slave to master).

Bit	Description	
0	Use parameters / start parameterization	If this bit is set the CI Device starts the parameterization with the parameters in the actual parameters registers.
1	Store parameters volatile	If this bit is set the CI device will use the parameters temporarily, which means after a bus error detection and reconnection the parameters will be used again. **This bit should always be set.** **This bit is only evaluated when bit 0 is set.**
2	Store parameters nonvolatile	If this bit is set the CI device will store the parameters nonvolatile, which means after a power cycle the stored parameter data will be used again. **This bit is only evaluated when bit 0 is set.**
3	Reserved	-
4	Delete nonvolatile stored parameters	If this bit is set the CI device will delete its nonvolatile stored parameters. **This bit is only evaluated when bit 0 is set**.
5	Ignore parameter error for nonvolatile parameter storage	If this bit is set a parameter error during nonvolatile storage of parameters will be ignored, and the parameters will be stored. **This bit can only be set in combination with bit 0 and bit 2.**
6	Reserved	-
7	Reserved	-
8	New diagnosis available	The device will set this bit if new diagnosis data are available in the diagnosis data section.
9	New parameters available	The device will set this bit if new parameters are available in the actual parameter data section and these were not activated by setting bit 0 in the control word.
10...15	Reserved	-

Cyclic I/O data exchange

The I/O data can be exchanged cyclic by the master by reading, writing the corresponding registers.

I/O data exchange is only possible after successful parameterization of the device.

For writing of outputs **bus failure detection** can be activated by setting the corresponding parameter. This bus failure detection is described in the following chapter.

Bus failure detection

If the parameter "*"timeout"*" in the module parameters of the CI52x-MODTCP is set, the module will supervise the Modbus TCP "write telegrams".

After the first "write telegram" the bus will be supervised. If no new "write telegram" arrives at the CI52x-MODTCP within the configured time, the module will detect a bus failure and switch off its outputs or switch them to the configured failsafe state (see module parameter CI521 ↳ *Chapter 1.6.2.8.5.1.7 "Parameterization" on page 4884* and CI522 ↳ *Chapter 1.6.2.8.5.2.7 "Parameterization" on page 4914* for details).

Configurable write restriction

With the module parameters “Master IP”- “Master IP 7” it is possible to set write restrictions on the CI52x-MODTCP device.

If none of the parameters is set, all masters / clients in the network have read and write rights on the CI52x-MODTCP device and its connected expansion modules.

If at least one parameter is set only the configured masters / clients have write rights on the CI52x-MODTCP device.

All other masters / clients still have read access to the CI52x-MODTCP device.

Diagnosis behavior

Each diagnosis message signals if this error is coming or going , so it is possible to create a list in the master of actual pending diagnosis.

Diagnosis messages will be transferred again after a bus failure detection and reconnection.

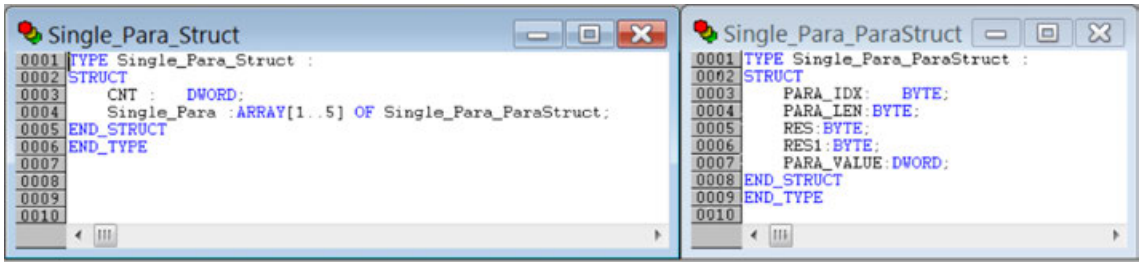
Diagnosis messages can be read out with function code 3,4,23. Function codes 3 and 4 can always read out diagnosis messages, function code 23 can only read out after successful parameterization of the device. See also table *Chapter 1.6.4.3.1.2.3.2 “Diagnosis data” on page 5659*.

Single parameterization

The single parameterization services can be used to read or write parameters during run time of device without the need of triggering a new parameterization process.

For indexes used for single parameterization services see parameter lists in section Modbus TCP registers of this document.

The read and write parameterization services are explained below, for each module (CI52x-MODTCP and connected expansion modules) a different section for read and write is defined see chapter Modbus TCP registers in this document). Both services are using the following data structure:



The length of the read / write service depends on the count of parameters that should be transferred (length = 4+ count*8).

Reading of single parameters

The read single parameterization works in two steps:

- Writing of a request list containing the indexes that should be read using the structure explained above.
  Only CNT and PARA_IDX has to be set.
  Up to 5 parameters can be requested with one telegram.
  The length of the write service depends on the count of parameters that should be transferred (length = 4+ count*8).
- Reading of the parameters list with the same length then the previous write request.
  If the internal reading process inside the CI52x-MODTCP device is done the data will be read out.
  If the internal reading process inside the CI52x-MODTCP device is not yet finished the read service will be rejected with Modbus TCP exception code 6 (device busy).

Writing of single parameters

For writing of single parameters only one step is necessary, the parameters are transferred with one write request using the structure described above.

The length of the write service depends on the count of parameters that should be transferred (length = 4+ count*8).

In case of write of single parameters the following values have to be set:

- CNT: number of parameters to be set
- And for each parameter:
  Parameter index
  Parameter length
  New parameter value

Written single parameters are not stored volatile and not stored nonvolatile. That means after a bus reconnection or power cycle the written parameters will be discarded.

Commissioning example

Set IP Address:

- The setting of the IP address is the first step to integrate the CI52x-MODTCP devices into a running system.
- The setting of the IP address of the CI52x-MODTCP devices is described in the chapter *Chapter 1.6.4.3.1.3.1 “IP address assignment” on page 5669* in this document.

Set Parameters (optional read parameters):

- The second step in configuring the CI52x-MODTCP devices is to set the module and channel parameters.
- A read of parameters is optional but can be used the get the module IDs and the parameter length.
- The reading and or writing of parameters is described in chapter *Chapter 1.6.4.3.1.3.2 “Parameterization” on page 5670*.

Set Control Word:

- After setting the parameter data these have to be activated by writing the control word.
- The meaning and usage of the control word is described in chapter *Chapter 1.6.4.3.1.3.2 “Parameterization” on page 5670*.

Exchange data:

- After setting and activating the parameters the CI52x-MODTCP device is ready for data exchange.
- The registers for data exchange are described in chapter *Chapter 1.6.4.3.1.2.3 “I/O / Process data and diagnosis section (Cyclic data)” on page 5656*.

Hot swap

With hot swap for AC500 and S500 it is possible to exchange expansion modules (with same type) during run time.

Preconditions for using hot swap

Information about preconditions for using hot swap see *“Hot swap” on page 5463*.

Compatibility of hot swap

	Modbus remote I/O
I/O module on TU5xx-H connected to I/O bus master	CI521-MODTCP or CI522-MODTCP
Required version of I/O bus master	Module index as of F0 Firmware as of V3.2.3
Fieldbus master when used as remote I/O with AC500 V2	Any AC500 V2 CPU as of PM57x with on-board Ethernet or CM597-ETH with MODTCP
When used as remote I/O on third party controller (PLC or DCS)	No limitation known

Hot swap behavior

The following table describes the behavior in case of I/O attached to communication interface module for Modbus TCP, CI521-MODTCP or CI522-MODTCP.

Hot Swap Behavior	Modbus TCP remote I/O
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H	Remote I/O station is not starting As of device index F4 and Automation Builder Version 2.4.1 it is possible to configure the startup in case of missing modules on hot swap terminal units. If configured, the remote I/O station is starting up with missing or damaged I/O module, if the module is plugged later or replaced it will be automatically parameterized and I/O data will be exchanged. As the Automation Builder checks that all modules are available during configuration process, it is necessary that all I/O modules are available and in working order during configuration via Automation Builder. As the parameters are stored nonvolatile inside the CI52x devices later one the parameters have effect for power cycle or reconnection operations.
Start-up behavior with wrong I/O module type on any terminal unit	Remote I/O station is not starting
Diagnosis of presence of hot swap terminal unit	Information is available in Modbus registers of the communication interface module which can be accessed by the application program As of device index F4 and Automation Builder Version 2.4.1 it is possible to configure a list of required hot swap terminal units. If a required hot swap terminal unit is missing (normal one plugged) this will not prevent a normal operation but a diagnosis message will be generated for the corresponding slot.
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	Information can be obtained by reading Modbus registers in the communication interface module. Those Modbus registers contain: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis In case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration ● Production data and version index of the modules
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	Diagnosis is available in Modbus registers in the communication interface module
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO
Diagnosis after plugging the I/O module on the hot swap terminal unit	Diagnose "diagnosis gone" is available in Modbus registers in the communication interface module

System behavior

If an expansion module is removed or defective during run time, the input data of this module will be set to “0” and the module state will be set to the corresponding value (see *Chapter 1.6.4.3.1.2.3 “I/O / Process data and diagnosis section (Cyclic data)” on page 5656*). A diagnosis message will be created in that case (see hardware description of *Chapter 1.6.2.8.5.1 “CI521-MODTCP” on page 4864* / *Chapter 1.6.2.8.5.2 “CI522-MODTCP” on page 4904* for diagnosis messages).

In case a module is replaced, the new module will automatically be parameterized with the last parameters of the removed module (if single parameters were written to the previously removed module, this parameters will be ignored).

During pulling or plugging of a certain module, all other module will continue to operate with one limitation: The reaction time of modules connected to the right of the affected module will be bigger in that case (up to 50 ms).

If the bus failure detection is active for CI52x and failsafe is configured (see *Chapter 1.6.4.3.1.3.3 “Cyclic I/O data exchange” on page 5671*) the following behavior applies if a module is removed and replugged during failsafe condition:

- Last value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will remain at its last value.
  - If the module is removed and plugged again, the output will remain off, and not be kept its last value, as the last value of the new module is “0” in that case.
- Substitute value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again.
- Substitute value for x seconds configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value for the configured time.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again, and the configured time starts again.

Mandatory rules for hot swapping

Mandatory rules for hot swapping:

- Between two pull and / or plug operations of I/O modules a pause of at least 1 second must be observed.
  - That means if a module is pulled or plugged there has to be at least a break of 1 second before the next module is pulled or plugged.
- At boot up of CI52x all configured expansion modules have to be physically available.
  - Start up with missing modules is not supported.
- In the application program it is possible to detect if a hot swap terminal unit is mounted in a specific position on the I/O bus. The information is available in the common device information registers. These can be accessed when the version of the communication interface module supports hot swap.
  - This has to be checked by application:

    Best way for checking if a hot swap terminal unit is available or not, is reading out the common device information registers (see *Chapter 1.6.4.3.1.2.2 “Information data section (Acyclic data)” on page 5653*). If the CI52x rejects this read out the CI52x doesn’t support hot swap at all.

1.6.4.3.2 PROFIBUS communication interface module

Hot swap

With hot swap for AC500 and S500 it is possible to exchange expansion modules (with same type) during run time.

Preconditions for using hot swap

Information about preconditions for using hot swap see *“Hot swap” on page 5463*.

Compatibility of hot swap

	PROFIBUS remote I/O
I/O module on TU5xx-H connected to I/O bus master	CI541-DP or CI542-DP
Required version of I/O bus master	Module index as of F0 Firmware as of V3.2.12
Fieldbus master when used as remote I/O with AC500 V2	Any AC500 V2 CPU as of PM57x with CM592-DP
When used as remote I/O on third party controller (PLC or DCS)	No limitation known

Hot swap behavior

The following table describes the behavior in case of I/O attached to communication interface module for PROFIBUS, CI541-DP or CI542-DP.

Hot Swap Behavior	PROFIBUS remote I/O with AC500 V2 CPU and CM592-DP as master	PROFIBUS remote I/O with third party controller (GSD used for configuration)
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H a	Remote I/O station is not starting	Remote I/O station is not starting As of device index F1 and Automation Builder Version 2.5.0 it is possible to configure the startup in case of missing modules on hot swap terminal units. If configured, the remote I/O station is starting up with missing or damaged I/O module, if the module is plugged later or replaced it will be automatically parameterized and I/O data will be exchanged.
Start-up behavior with wrong I/O module type on any terminal unit	Remote I/O station is not starting	Remote I/O station is not starting
Diagnosis of presence of hot swap terminal unit	Information is available via acyclic services	Information is available via acyclic services As of device index F1 and Automation Builder Version 2.5.0 it is possible to configure a list of required hot swap terminal units. If a required hot swap terminal unit is missing (normal one plugged) this will not prevent a normal operation but a diagnosis message will be generated for the corresponding slot.

Hot Swap Behavior	PROFIBUS remote I/O with AC500 V2 CPU and CM592-DP as master	PROFIBUS remote I/O with third party controller (GSD used for configuration)
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	Diagnosis is transmitted as vendor specific diagnosis and can be accessed with the function block *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration Production data and version index of the modules is accessible via acyclic services	Diagnosis is transmitted as vendor specific diagnosis: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration Production data and version index of the modules is accessible via acyclic services
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	Diagnosis is transmitted as vendor specific diagnosis "diagnosis gone" and can be accessed with the function block *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*	Diagnosis is transmitted as vendor specific diagnosis.
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO	Input = ZERO
Diagnosis after plugging of the I/O module on the hot swap terminal unit	Diagnosis is transmitted as vendor specific diagnosis "diagnosis gone" and can be accessed with the function block *Chapter 1.5.4.26.1.5 “DPM_SLV_DIAG” on page 1765*	Diagnosis is transmitted as vendor specific diagnosis "diagnosis gone"

System behavior

If an expansion module is removed or defective during run time, the input data of this module will be set to “0” and the module state will be set to the corresponding value. A diagnosis message will be created in that case (see hardware description of *Chapter 1.6.2.8.6.1 “CI541-DP” on page 4930* / *Chapter 1.6.2.8.6.2 “CI542-DP” on page 4969* for diagnosis messages).

In case a module is replaced, the new module will automatically be parameterized with the last parameters of the removed module (if single parameters were written to the previously removed module, this parameters will be ignored).

During pulling or plugging of a certain module, all other modules will continue to operate with one limitation: The reaction time of modules connected to the right of the affected module will be bigger in that case (up to 50 ms).

If the bus failure detection is active for CI54x and failsafe is configured the following behavior applies if a module is removed and replugged during failsafe condition:

- Last value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will remain at its last value.
  - If the module is removed and plugged again, the output will remain off, and not be kept its last value, as the last value of the new module is “0” in that case.
- Substitute value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again.
- Substitute value for x seconds configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value for the configured time.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again, and the configured time starts again.

Mandatory rules for hot swapping

Mandatory rules for hot swapping:

- Between two pull and / or plug operations of I/O modules a pause of at least 1 second must be observed.
  - That means if a module is pulled or plugged there has to be at least a break of 1 second before the next module is pulled or plugged.
- At boot up of CI54x all configured expansion modules have to be physically available.
  - Start up with missing modules is not supported.
- In the application program it is possible to detect if a hot swap terminal unit is mounted in a specific position on the I/O bus. The information is available via acyclic read requests. These can be accessed when the version of the communication interface module supports hot swap.
  - This has to be checked by application:

    Best way for checking if a hot swap terminal unit is available or not, is reading out the module info. If the CI54x rejects this read out the CI54x doesn’t support hot swap at all.

1.6.4.3.3 PROFINET communication interface module

Hot swap

With hot swap for AC500 and S500 it is possible to exchange expansion modules (with same type) during run time.

Preconditions for using hot swap

Information about preconditions for using hot swap see *“Hot swap” on page 5463*.

Compatibility of hot swap

	PROFINET remote I/O
I/O module on TU5xx-H connected to I/O bus master	CI501-PNIO or CI502-PNIO
Required version of I/O bus master	Module index as of F0 Firmware as of V3.2.10
Fieldbus master when used as remote I/O with AC500 V2	Any AC500 V2 CPU as of PM57x with CM579-PNIO
When used as remote I/O on third party controller (PLC or DCS)	Note: alarms must be acknowledged by fieldbus master. GSDML as of version GSDML-V2.3-ABB-S500-CI501-PNIO-20180822.xml or GSDML-V2.3-ABB-S500-CI502-PNIO-20180822.xml needed for full scope of vendor specific diagnosis.

Hot swap behavior

The following table describes the behavior in case of I/O attached to communication interface module for PROFINET, CI501-PNIO or CI502-PNIO.

Hot Swap Behavior	PROFINET remote I/O with AC500 V2 CPU and CM579-PNIO as master	PROFINET remote I/O with third party controller (GSDML used for configuration)
Start-up behavior with missing or damaged I/O module on hot swap terminal unit TU5xx-H	Remote I/O station is not starting	Remote I/O station is not starting As of device index F1 and Automation Builder Version 2.4.1 it is possible to configure the startup in case of missing modules on hot swap terminal units. If configured, the remote I/O station is starting up with missing or damaged I/O module, if the module is plugged later or replaced it will be automatically parameterized and I/O data will be exchanged.
Start-up behavior with wrong I/O module type on any terminal unit	Remote I/O station is not starting	Remote I/O station is not starting
Diagnosis of presence of hot swap terminal unit	Information is available either: ● via acyclic services or ● as cyclic state information in the process image Requires Automation Builder version as of 2.2.	Information is available either: ● via acyclic services or ● as cyclic state information in the process image As of device index F1 and Automation Builder Version 2.4.1 it is possible to configure a list of required hot swap terminal units. If a required hot swap terminal unit is missing (normal one plugged) this will not prevent a normal operation but a diagnosis message will be generated for the corresponding slot.

Hot Swap Behavior	PROFINET remote I/O with AC500 V2 CPU and CM579-PNIO as master	PROFINET remote I/O with third party controller (GSDML used for configuration)
Diagnosis of hot swap capability of I/O module mounted on hot swap terminal unit	Diagnosis is transmitted as alarm and can be accessed with the function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration Production data and version index of the modules is accessible via acyclic services	Diagnosis is transmitted as vendor specific PROFINET channel diagnosis: ● Diagnosis in case that a not hot-swappable I/O module is plugged on a hot swap terminal unit ● Diagnosis in case that in a mixed configuration with at least one hot swap terminal unit an I/O module, that must not be used in a hot swap configuration, is mounted on any terminal unit of the configuration Production data and version index of the modules is accessible via acyclic services
Diagnosis while hot swap module is pulled or module (mounted on hot swap terminal unit) has stopped working	Diagnosis is transmitted as alarm and can be accessed with the function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*. PROFINET standard "pull alarm" is generated and must be acknowledged with the function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*	PROFINET channel diagnosis is generated together with standard "pull alarm" which must be acknowledged
Input state in process image of controller while module is pulled or module is not operational	Input = ZERO	Input = ZERO In addition a standard PROFINET state information is transmitted saying "inputs not valid"
Diagnosis after plugging of the I/O module on the hot swap terminal unit	PROFINET standard "plug alarm" is generated and must be acknowledged with the function block *Chapter 1.5.4.27.1.1 “PNIO_DEV_ALARM” on page 1794*	PROFINET channel diagnosis is generated together with standard "plug alarm" which must be acknowledged

System behavior

If an expansion module is removed or defective during run time, the input data of this module will be set to “0” and the module state will be set to the corresponding value. A diagnosis message will be created in that case (see hardware description of *Chapter 1.6.2.8.7.2 “CI501-PNIO” on page 4995* / *Chapter 1.6.2.8.7.3 “CI502-PNIO” on page 5035* for diagnosis messages).

In case a module is replaced, the new module will automatically be parameterized with the last parameters of the removed module (if single parameters were written to the previously removed module, this parameters will be ignored).

During pulling or plugging of a certain module, all other module will continue to operate with one limitation: The reaction time of modules connected to the right of the affected module will be bigger in that case (up to 50 ms).

If the bus failure detection is active for CI50x and failsafe is configured the following behavior applies if a module is removed and replugged during failsafe condition:

- Last value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will remain at its last value.
  - If the module is removed and plugged again, the output will remain off, and not be kept its last value, as the last value of the new module is “0” in that case.
- Substitute value configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again.
- Substitute value for x seconds configured for output:
  - After a bus failure is detected, failsafe will be activated and the output will be according to the configured substitute value for the configured time.
  - If the module is removed and plugged again now, the output will be set according to the configured substitute value again, and the configured time starts again.

Mandatory rules for hot swapping

Mandatory rules for hot swapping:

- Between two pull and / or plug operations of I/O modules a pause of at least 1 second must be observed.
  - That means if a module is pulled or plugged there has to be at least a break of 1 second before the next module is pulled or plugged.
- At boot up of CI50x all configured expansion modules have to be physically available.
  - Start up with missing modules is not supported.
- In the application program it is possible to detect if a hot swap terminal unit is mounted in a specific position on the I/O bus. The information is available in the process data area or can be read out via acyclic read. These can be accessed when the version of the communication interface module supports hot swap.
  - This has to be checked by application:

    Best way for checking if a hot swap terminal unit is available or not, is checking the corresponding information inside the process image.

1.6.4.4 System technology of the AC500 function modules

1.6.4.4.1 DC541-CM interrupt and counter module

Cycle time modification

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation *Chapter 1.5.4.11 “DC541 library” on page 1103*.

Access to the channels configured as normal inputs and outputs is performed using the function block Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139.

The module's cycle time is set automatically depending on its channel configuration. The following values are possible for the cycle time:

CYCLE

Data type	Default value	Range	Unit
WORD	-	-	µs

CYCLE (cycle time) output displays the cycle time of the device. The cycle time is set during the device configuration and can have the following values depending on the channel configuration:

Parameter		Description	Value
IO device			50 µs
Counting device		1-2 functions	50 µs
		3-4 functions	100 µs
		5-8 functions	200 µs
"Functions"			
	PWM	Pulse-width modulator	
	FREQ	Time and frequency measurement	
	FREQ_OUT	Frequency output	
	32BIT_CNT	32-bit counter	
	FWD_CNT	32-bit count up counter	
	LIMIT	Limit value monitoring for the 32-bit counter	

The used cycle time can be read at output CYCLE of the block *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132*.

The following table shows an overview of all possible combinations.

Configured as	Function/ can be con- figured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
Mode 1: Interrupt function; mutually exclusive with mode 2 (counting functions).								
Interrupt	Dig. input	1	1	1	1	4	8	Each channel can be configured individually as interrupt input or output.
	Interrupt inp.	1	1	1	1	4	8	
	Dig. output	1	1	1	1	4	8	
Mode 2: Counting functions and multifunctional I/Os; mutually exclusive with mode 1 (interrupt functions).								
Multi-function I/Os, PWM, counters, time and frequency measuring	Dig. input	1	1	1	1	4	8	Normal input
	Dig. output	1	1	1	1	4	8	Normal output
	PWM, resolution 10 kHz	1	1	1	1	4	8	Outputs a pulsed signal with an adjustable on-off ratio.
	Frequency output, resolution 2.5 kHz	1	1	1	1	4	8	Outputs an adjustable frequency (endless output or output of a specified number of pulses).

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Up/down counter, 50 kHz	1	1	OK *1)	OK *1)	OK *1)	2	*1) a) Both channels (0 and 1) config-ured as 50 kHz counter => Chan-nels 2 to 7 can be config-ured as digital I/Os. b) Only one channel (0 or 1) config-ured as 50 kHz counter => Second channel can be config-ured as counter < 50 kHz or for time/ fre-quency meas-urement with a max. resolu-tion of 200 µs. The remainin g chan-nels (2 to 7) can be con-figured as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Up/down counter, 5 kHz	1	1	1	1	OK *2)	4	*2) a) Four channels (0 to 3) config-ured as 5 kHz counter => Chan-nels 4 to 7 can be config-ured as digital I/Os. b) Only a portion of the 4 channels (0 to 3) config-ured as 5 kHz counter => The other ones (of channels 0 to 3) can be config-ured as desired: as 2.5 kHz counter or for time/ fre-quency meas-urement with a max. resolu-tion of 200 µs or as digital I/Os. The remainin

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
								g chan-nels (4 to 7) can be con-figured as digital I/Os.
	Up/down counter, 2.5 kHz	1	1	1	1	4	8	
	Time/ fre-quency meas-urement, resolu-tion 50 µs	1	OK *3)	OK *3)	OK *3)	OK *3)	1	*3) Channel 0 config-ured for a max. resolu-tion of 50 µs => Chan-nels 1 to 7 can be config-ured as digital I/Os.

Configured as	Function/ can be configured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
	Time/ frequency measurement, resolution 100 µs	1	1	OK *4)	OK *4)	OK *4)	2	*4) a) Two channels (0 and 1) configured for a max. resolution of 2x100 µs => Channels 2 to 7 can be configured as digital I/Os. b) Only one channel (0 or 1) configured for a max. resolution of 50 µs => Second channel (0 or 1) can be configured as counter < 50 kHz or for time/ frequency measurement with a max. resolution of 200 µs. The remaining channels (2

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
								to 7) can be con-figured as digital I/Os.
	Time/ fre-quency meas-urement, resolu-tion 200 µs	1	1	1	1	4	8	Times, frequen-cies and rota-tional speeds are meas-ured with a max. resolu-tion of 200 µs.

Configured as	Function/ can be con- figured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
High-speed counter	Up/down 32-bit counter, 50 kHz max.	Channels 0 to 3: Track A, track B, zero track, touch trigger				OK *6)	1	For connection of an incremental transmitter. For signals up to 50 kHz. This frequency corresponds to a motor with a rotational speed of 3000 rpm. The counter always uses the first 4 channels (0 to 3). *6) The remaining channels (4 to 7) can be configured as limit values, as 5 kHz counters, for time/ frequency measurement with a resolution of 200 µs or as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Axis of rotation (endless counting)	1				OK *7)	1	"End-less" for-ward counting. An over-flow occurs corre-sponding to the 32-bit value. *7) The remainin g chan-nels can be con-figured as limit values, as 5 kHz coun-ters, for time/ fre-quency meas-urement with a resolu-tion of 200 µs or as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	32-bit counter incl. sign	1				OK *8)	1	*8) The remainin g chan-nels can be con-figured as limit values, as 5 kHz coun-ters, for time/ fre-quency meas-urement with a resolu-tion of 200 µs or as digital I/Os.
	Limit values for 32-bit counter	OK *9)				1	1	Various counting values of the 32-bit counter can be dis-played directly via these outputs. *9) In this case, the channels 0 to 3 are used as 32-bit coun-ters.

Usage as interrupt I/O module

Creating an interrupt task for the interrupt inputs

If one or more channels of DC541-CM are configured as interrupt inputs, a corresponding interrupt task has to be created to enable the processing of the interrupt(s).

For this purpose, a new task has to be added in the task configuration of Automation Builder:

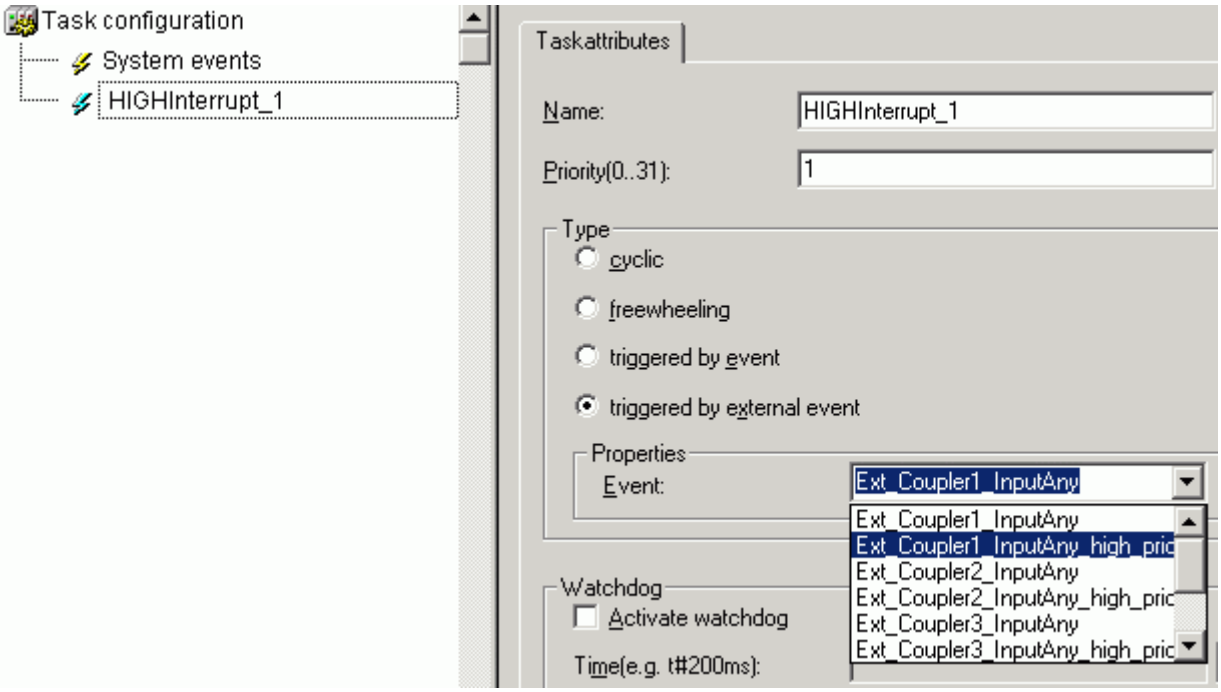
- Enter the task name
- Set the task type to "triggered by external event"
- Specify the event that triggers the task

For each Communication Module slot, two types of interrupt tasks are available in the Event list box:

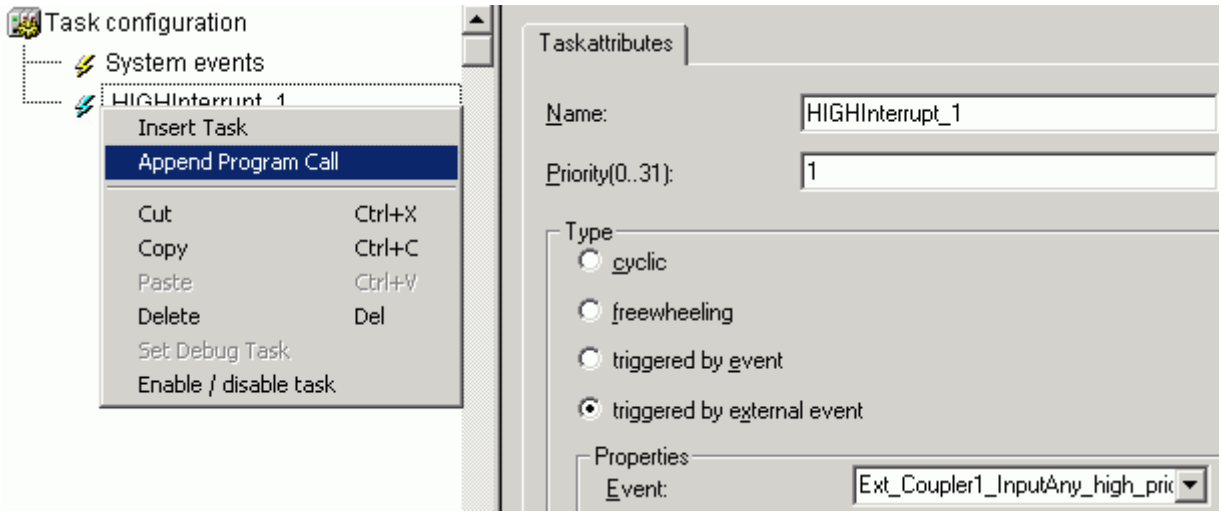
- Ext_Communication ModuleX_InputAny:
  The task is triggered by any interrupt from Communication Module slot X with the priority specified in the Priority field (0...31).
- Ext_Communication ModuleX_InpuAny_high_prio:
  The task is triggered by any interrupt from Communication Module slot X with highest priority, i.e. with a priority higher than the max. adjustable "0" and higher than the priority of the communication task. In this case, the priority (0...31) specified in the Priority field is without any significance.

> *If the interrupt task is started with highest priority, the program execution time must not be longer than approx. 400 µs. Otherwise online access is no longer possible.*

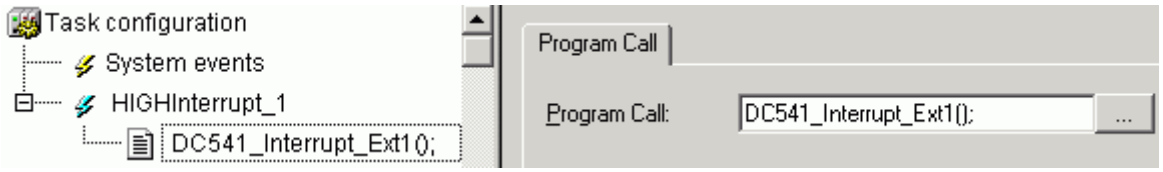
In the example below, the task is named HIGHInterrupt_1, meaning that it is a high-priority interrupt from Communication Module slot 1. The task type is "external event triggered" and the event to trigger the task is "Ext_Communication Module1_InputAny_high_priority".



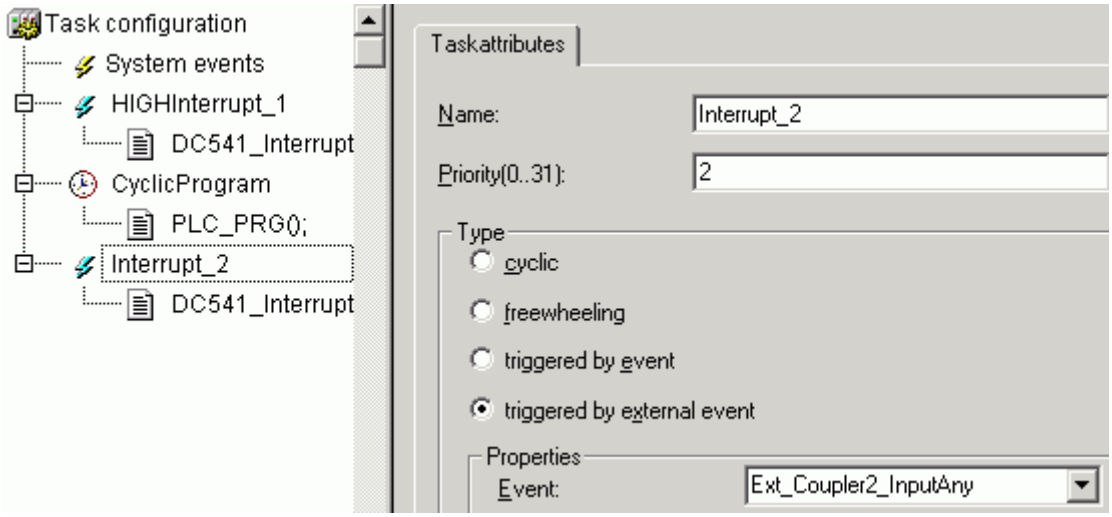
Like for all other tasks, a program call has to be assigned to the task.



In the example, the program DC541_Interrupt_Ext1() shall be started with any interrupt from Communication Module slot 1.



The task configuration for an AC500 equipped with two DC541-CM modules inserted in the Communication Module slots 1 and 2 and containing one cyclically running "background program" PLC_PRG could for example look as follows. Here, an interrupt from slot 1 should start the program DC541_Interrupt_Ext1 with high priority, an interrupt from slot 2 should start the program DC541_Interrupt_Ext2 with priority 2:



Structure of the interrupt program

The following blocks contained in the library DC541_AC500_V11.lib are available for the interrupt program:

- ↳ *Chapter 1.5.4.11.1.1.7 “DC541_INT_IN” on page 1136* Determination of the interrupt initiating source
- ↳ *Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139* Reading and writing of channels C0...C7

It is possible to start one interrupt task per Communication Module slot. This task can be started by any channel (C0...C7) configured as interrupt input. Therefore, it is necessary for the interrupt program to differentiate which channel(s) triggered the interrupt in order to enable the processing of the corresponding actions.

The information whether a channel (C0...C7) has triggered an interrupt since the last call of the block is provided by the outputs IN0...IN7 of the block *Chapter 1.5.4.11.1.1.7 “DC541_INT_IN” on page 1136*. This is why this block always has to be called at the beginning of the interrupt program, if more than one channel is configured as interrupt input.

The access to the channels configured as inputs or outputs is done using the block DC541_IO. Therefore, it makes sense to call this block at the beginning of the interrupt program in order to read the inputs and at the end of the interrupt program in order to write the outputs.

--Configuration example: DC541-CM used as interrupt I/O device

Hardware configuration

The example control system shall have the following configuration:

- Terminal base TB521 (two Communication Module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH
- I/O module DC532 on the I/O Bus

Wiring

The channels are connected as follows:

DC532 / C16 ---------- DC541 / C0

DC532 / C17 ---------- DC541 / C1

DC532 / C18 ---------- DC541 / C2

DC532 / C19 ---------- DC541 / C3

DC532 / C20 ---------- DC541 / C4

DC532 / C21 ---------- DC541 / C5

PLC configuration

- DC541-CM in slot 1, operating mode "IO mode"

- Configuration:	Channels	C0...C4	Interrupt input
	Channel	C5	Input
	Channels	C6...C7	Outputs

- Specification of the Ethernet communication module as internal communication module (if available)
- DC532 on the I/O bus

Task configuration

- Task 1: Cyclic program / Prio = 10 / Interval = t#10ms / PLC_PRG
- Task 2: HIGHInterrupt_1 / DC541_Interrupt_Ext1()

DC541_Interrupt_Ext1()

The interrupt program should fulfill the following functionality:

- Counting of all interrupts
- Counting of the interrupts per input
- Calculation of the interrupt frequency in [Int/s]
- Reporting of the number of interrupts per input
- Input C4: Resetting the counters
- Input C5: Input

- Output C6: Status of input C5
- Output C7: Toggle output

The declaration part of the program looks as follows:

PROGRAM DC541_Interrupt_Ext1			
VAR			
	dwIntCount	: DWORD;	(* count all interrupts *)
	dwIntCountOld	: DWORD;	(* start value for next measure *)
	tActual	: TIME;	(* systemtick in ms *)
	tStart	: TIME;	(* start value of systemtick for next calculation *)
	dwUsedTime	: DWORD;	(* time for 1000 interrupts in ms *)
	dwFrequenz	: DWORD;	(* interrupt frequency in [Int / sec] *)
	DC541_IntSource	: DC541_INT_IN;	(* instance FB: read interrupt source *)
	DC541_Ios	: DC541_IO;	(* instance FB: read/ write inputs/outputs *)
	dwCount_InX	: ARRAY[0..cbyDC541_IntInp] OF DWORD;	(* count interrupts of In0..In3 *)
	dwCount_InXOld	: ARRAY[0..cbyDC541_IntInp] OF DWORD;	(* start value for next 1000 interrupts *)
	dwIntHisto	: ARRAY[0..cbyDC541_IntInp, 0..cbyDC541_MaxHist] OF DWORD;	(* histo data C0...C3 *)
	wIndex	: WORD;	(* index for histo data *)
	byInd	: BYTE;	(* loop index *)
END_VAR			
VAR CONSTANT			
	cbyDC541_SLOT	: BYTE := 1;	(* SLOT number of DC541 *)
	cbyDC541_MaxHist	: BYTE := 9;	(* max number of histo entries *)
	cbyDC541_IntInp	: BYTE := 4;	(* number of interrupt inputs -1 *)
END_VAR			

The instruction part looks as follows:

At the beginning, the interrupts are counted in dwIntCount. After each 1000 interrupts, a calculation of the frequency is performed and the counting values for the interrupts per input are stored.

dwIntCount := dwIntCount + 1;		(* count all interrupts *)
IF dwIntCount - dwIntCountOld >= 1000 THEN		(* after 1000 interrupts -> calculate frequency *)
	dwIntCountOld := dwIntCount;	(* save dwIntCount for next call *)
	tActual := TIME();	
	dwUsedTime := TIME_TO_DWORD(tActual - tStart);	(* duration in ms for 1000 interrupts *)
	dwFrequenz := 1000000 / dwUsedTime;	(* [Interrupt / sec] 1000 Int * 1000 ms/sec *)
	tStart := tActual;	(* for next measure *)
	dwIntHisto[0,wIndex] := dwCount_InX[0] - dwCount_InXOld[0];	(* IN0 interrupts of last 1000 *)
	dwCount_InXOld[0] :=dwCount_InX[0];	(* start value for next measure *)
	dwIntHisto[1,wIndex] := dwCount_InX[1] - dwCount_InXOld[1];	(* IN1 interrupts of last 1000 *)
	dwCount_InXOld[1] :=dwCount_InX[1];	(* start value for next measure *)
	dwIntHisto[2,wIndex] := dwCount_InX[2] - dwCount_InXOld[2];	(* IN2 interrupts of last 1000 *)
	dwCount_InXOld[2] :=dwCount_InX[2];	(* start value for next measure *)
	dwIntHisto[3,wIndex] := dwCount_InX[3] - dwCount_InXOld[3];	(* IN3 interrupts of last 1000 *)
	dwCount_InXOld[3] :=dwCount_InX[3];	(* start value for next measure *)
	wIndex := wIndex + 1;	(* increase index *)
	IF wIndex > cbyDC541_MaxHist THEN wIndex := 0; END_IF;	(* reset index, if >1000 *)
END_IF; (* 1000 Interrupts *)		

After this, the block DC541_INT_IN is called to identify the interrupt source and then the interrupt counters of the channels are updated depending on the outputs of this block.

```
(* Read interrupt source --> if output  = TRUE --> interrupt since last call *)
DC541_IntSource(EN := TRUE, SLOT := cbyDC541_SLOT);
(* count the interrupts for each interrupt input C0..C3 *)
dwCount_InX[0] := dwCount_InX[0] + BOOL_TO_DWORD(DC541_IntSource.IN0);
dwCount_InX[1] := dwCount_InX[1] + BOOL_TO_DWORD(DC541_IntSource.IN1);
dwCount_InX[2] := dwCount_InX[2] + BOOL_TO_DWORD(DC541_IntSource.IN2);
dwCount_InX[3] := dwCount_InX[3] + BOOL_TO_DWORD(DC541_IntSource.IN3);
dwCount_InX[4] := dwCount_InX[4] + BOOL_TO_DWORD(DC541_IntSource.IN4);
```

In case of an interrupt on channel 4, the counters are reset.

IF DC541_IntSource.IN4 THEN		(* Input channel C4 = TRUE *)
	dwIntCount := dwIntCoun-tOld := 0;	(* reset count all interrupts *)
	FOR byInd := 0 TO cbyDC541_IntInp-1 DO	(* reset channel interrupt counters C0..C3 *)
	dwCount_InX[byInd] := dwCount_InXOld[byInd] := 0;	
	END_FOR; (* byInd *)	
	wIndex := 0;	(* start historical data from 0 *)
END_IF; (* C4 = TRUE *)		

At the end, the static inputs and outputs are processed, i.e.:

- reading the inputs,
- execution of actions
- writing the outputs.

(* Read inputs of DC541 *)		
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT);		
DC541_IOs.OUT6 := DC541_IOs.IN5;	(* C6 := state of input channel C5 *)	
DC541_IOs.OUT7 := NOT DC541_IOs.OUT7;	(* toggle channel C7 *)	
(* Write outputs to DC541*)		
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT);		

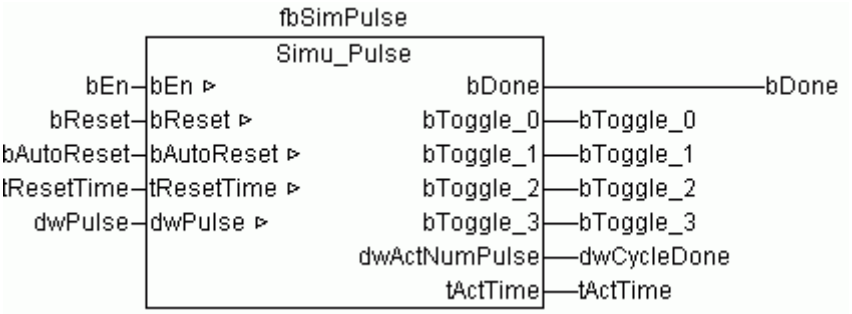
Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Cycles counter dwC := dwC + 1;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE - Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Simulation of the interrupts for the DC541 Calling of block Simu_Pulse

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the Function Block Diagram (FBD) is as follows:



The meanings of the block's inputs and outputs are as follows:

Instance		fbSimuPulse	Instance name
bEn	Input/Output	BOOL	Enabling of the pulse output
bAutoReset	Input/Output	BOOL	Automatic reset of the pulse counter after the specified number of pulses have been output and after expiration of tResetTime
bReset	Input/Output	BOOL	Reset of the pulse counter
tResetTime	Input/Output	TIME	Time until the reset is initiated after the specified number of pulses is reached, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: =0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE > 0: Cyclic mode (output of the specified number of pulses)
bDone	Output	BOOL	Completion message after tResetTime has expired or bReset = TRUE for 1 cycle
bToggle_0	Output	BOOL	Provides a FALSE->TRUE edge with each 2nd call (i.e. the output is toggled with each call)
bToggle_1	Output	BOOL	Provides a FALSE->TRUE edge with each 4th call
bToggle_2	Output	BOOL	Provides a FALSE->TRUE edge with each 8th call
bToggle_3	Output	BOOL	Provides a FALSE->TRUE edge with each 16th call
dwActPulse	Output	DWORD	Displays the number of pulses output (corresponds to the number of edges at bToggle_0)
tActTime	Output	TIME	Displays the elapsed time while tResetTime is running

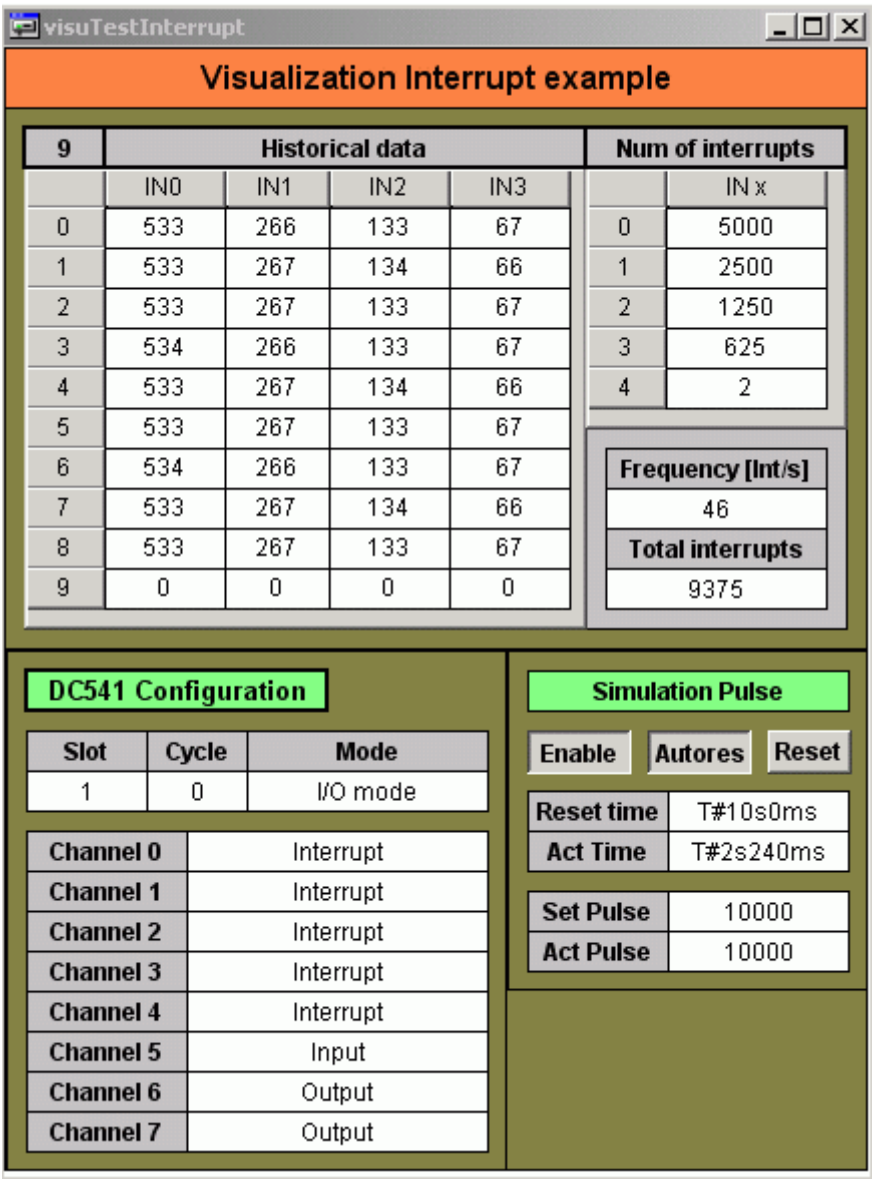
In the example, bEn: = bAutoReset: = TRUE. 10000 pulses are output (dwSetPulse). After the specified number of pulses has been reached, a wait time of 10 seconds is applied and then counting is started from the beginning.

The example has a visualization implemented which can be used to operate the program. After 10000 pulses, the visualization looks as follows:

9375 interrupts are generated:

5000 x C0 + 2500 x C1 + 1250 x C2 + 625 x C3 = 9375

Act Pulse					**Triggers the following interrupts:**
Value	**IN 3** **8**	**IN 2** **4**	**IN 1** **2**	**IN 0** **1**	
0	0	0	0	0	none
1	0	0	0	1	IN 0 -> in every 2. cycle (10000 : 2 = 5000)
2	0	0	1	0	IN 1 -> in every 4. cycle (10000 : 4 = 2500)
3	0	0	1	1	IN 0
4	0	1	0	0	IN 2 -> in every 8. cycle (10000 : 8 = 1250)
5	0	1	0	1	IN 0
6	0	1	1	0	IN 1
7	0	1	1	1	IN 0
8	1	0	0	0	IN 3 -> in every 16. cycle (10000 : 16 = 625)
9	1	0	0	1	IN 0
10	1	0	1	0	IN 1
11	1	0	1	1	IN 0
12	1	1	0	0	IN 2
13	1	1	0	1	IN 0
14	1	1	1	0	IN 1
15	1	1	1	1	IN 0
16	0	0	0	0	none



Usage as counter module

32-Bit up/down counter of module DC541-CM

The 32-bit bidirectional counter functionality is provided by the function block ↳ *Chapter 1.5.4.11.1.1.1 “DC541_32BIT_CNT” on page 1103*.

The 32-bit counter is a count up/count down counter with a directional discriminator. The counter can be used in two counting modes:

- EN_UD = FALSE: Encoder mode
  Connection of an incremental transmitter (track A / track B, offset by 90°)
  It is possible to count signals up to approx. 60 kHz. This corresponds to a motor with a rotational speed of 3.600 rpm and a transmitter with 1.000 pulses per rotation. Pulse multiplication (x2 or x4) is not used.
- EN_UD = TRUE: Up / down mode
  Up-/down counter
  It is possible to count signals up to approx. 60 kHz. Count-up for signals on channel C1, count-down for signals on channel C0.

The counter always uses the channels C0...C3 of the DC541:

- C0: Track A of the incremental transmitter.
- C1: Track B of the incremental transmitter.
- C2 and C3: Reference cam or touch trigger.

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

32-Bit forward counter of module DC541-CM

The 32-bit forward counter functionality is provided by the block ↳ *Chapter 1.5.4.11.1.1.5 “DC541_FWD_CNT” on page 1127*.

The function block DC541_FWD_CNT provides a 32-bit count up counter which is able to count a maximum frequency of 50 kHz at the inputs C0 and C1 or 5 kHz at the inputs C2-C7. In the DC541, the counter is implemented as a 16 bit counter. The actual counter value ACT_CNT is bulit inside the function block by adding the counter differences that occur within the individual cycles. In order not to loose any counting pulses, the function block has to be called cyclically.

- Channel 0-1: 50 kHz max. -> 32767 / 50 = 655 ms
- Channel 2-7: 5 kHz max. -> 32767 / 5 = 6550 ms

Using the counter e.g. in a 100 ms task will prevent any loss of counting pulses.

Operating modes

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32-bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

The device DC541 must be configured as counting device (counter mode).

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

Configuration example: 32-Bit forward counter

All of the 8 channels of the DC541-CM can be used as count up counter. In the configuration example, all 8 channels of the DC541-CM are configured as 32-bit forward counter (count-up). The channels C0...C3 operate as infinite counters (endless mode), the channels C4...C7 as limit counters (limit mode).

The 32-bit count up counter configured as infinite counter (endless mode) corresponds to mode 1 (1 count up counter) of the high-speed counter of the digital input/output modules. In the configuration example, the counting pulses for the first forward counter are therefore applied in parallel to input C0 of the DC541-CM and counting input C24 of the DC532.

Hardware configuration

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet Communication Module
- I/O module DC532 on the I/O bus

Wiring

The channels are connected as follows:

- DC532 / C16 ---------- DC541 / C0
- DC532 / C17 ---------- DC541 / C1
- DC532 / C18 ---------- DC541 / C2
- DC532 / C19 ---------- DC541 / C3
- DC532 / C20 ---------- DC541 / C4
- DC532 / C21 ---------- DC541 / C5
- DC532 / C22 ---------- DC541 / C6
- DC532 / C23 ---------- DC541 / C7
- DC532 / C16 ---------- DC532 / C24

PLC configuration

- DC541-CM in slot 1, operating mode "Counter mode"
- Configuration: - Channel C0..C7 Forward counter
- Specification of the Ethernet communication module as internal communication module (if available)
- DC532 on the I/O bus / parameter "Fast counter" = 1-1 count up counter

Task configuration

- Task 1: Cyclic program / Prio = 10 / Interval = t#100ms / PLC_PRG
- Task 2: Simulation / Prio = 15 / Interval = t#5ms / Simulation_Task

The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for the counters Calling of program proForwardCounter

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation ↬ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation ↬ *Chapter 1.5.4.19 “Internal system library” on page 1500*.

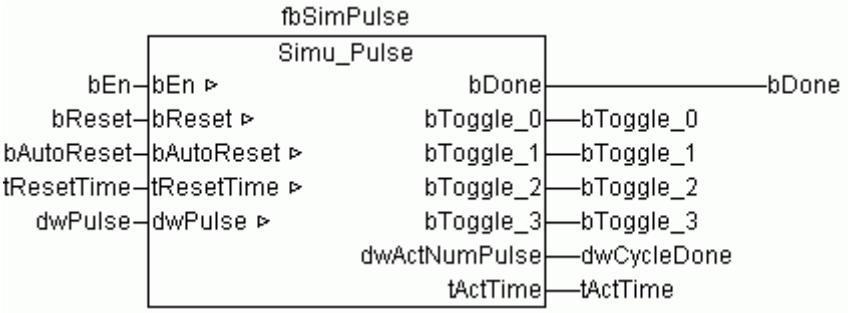
The actual execution of the 32-bit forward counter functionality is implemented in the program proForwardCounter.

Purpose of the program proForwardCounter:

The program proForwardCounter executes the following step chain:

Counter block	**DC541_FWD_CNT CNT_IO**								**CNT_IO**
Step | Channel	**C0**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**1**
0 | Action	Init: SET = 0, endless counter, limit values MIN = 300 / MAX = 1300				Init: SET = 0, limit counter, limit values MIN = 300 / MAX = 1300				Init
| Value	0	0	0	0	0	0	0	0	0
1 | Action	Reset of SET input								
| Value	0	0	0	0	300	300	300	300	0
2 | Action	Start of pulse output - 2000 pulses								
| Value	0	0	0	0	300	300	300	300	0
3 | Action	Wait until pulse output is completed								
| Value	2000	1000	500	250	1299	1300	800	550	2000
4 | Action	Selection last step: byStep = 249								
| Value	2000	1000	500	250	1299	1300	800	550	2000
200 | Action	Manual operation								
| Value	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
249 | Action	Wait time 5 seconds, then restart from step 0								
| Value	2000	1000	500	250	1299	1300	800	550	2000

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the Function Block Diagram (FBD) is as follows:



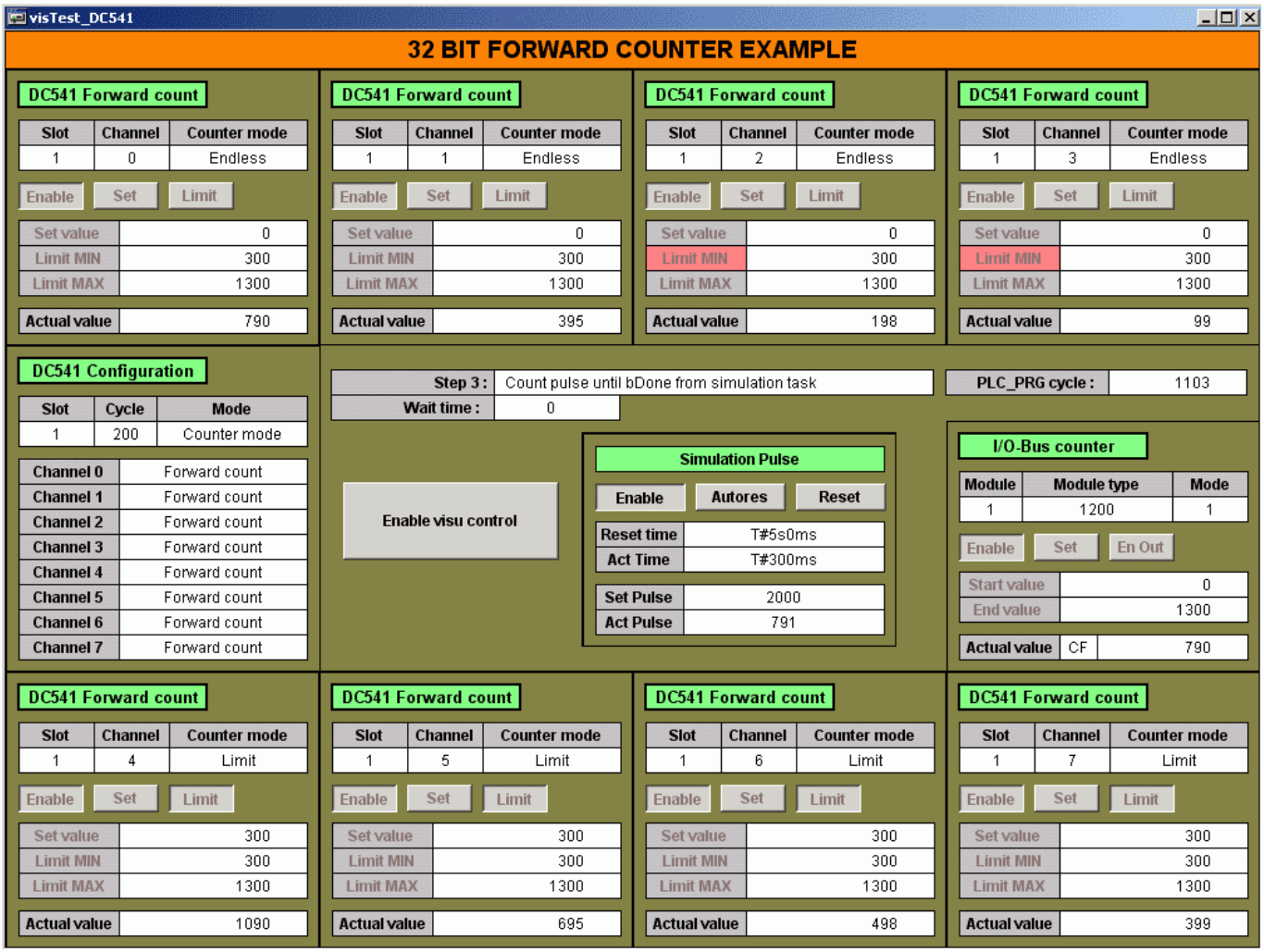
Instance		**fbSimuPulse**	**Instance name**
Ben	Input/Output	BOOL	Enabling of the pulse output
bReset	Input/Output	BOOL	TRUE = Reset of the pulse counter, bDone = TRUE

Instance		fbSimuPulse	Instance name
bAutoReset	Input/Output	BOOL	TRUE and cyclic mode - The time tResetTime is started when the number of pulses set with dwPulse is reached. After this time, the pulse output is restarted again.
tResetTime	Input/Output	TIME	Wait time until restart, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: = 0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE) > 0: Cyclic mode (output of the specified number of pulses)
Bdone	Output	BOOL	Completion message after the number of pulses specified at dwPulse or after bReset if dwPulse = 0
bToggle_0	Output	BOOL	Output: Edge with each clock cycle
bToggle_1	Output	BOOL	Output: Edge with each 2. clock cycle
bToggle_2	Output	BOOL	Output: Edge with each 4. clock cycle
bToggle_3	Output	BOOL	Output: Edge with each 8. clock cycle
dwActNumPulse	Output	DWORD	Number of pulses output
tActTime	Output	TIME	Elapsed time in [ms] while tResetTime is running

In the example, the block Simu_Pulse is called in a 5 ms task. The pulse output is enabled or stopped via input bEn. If input dwPulse = 0, the output of pulses is performed continuously. If dwPulse > 0, only the specified number of pulses is output. When the specified number of pulses is reached, output bDone is set to TRUE.

In the example, the block is called with dwPulse = 2000. The wait time function is not used.

The example program has a visualization implemented that displays all states:



Clicking on the button <Enable visu control> (bEnVisuControl = TRUE) causes the program to jump from the current step to step 200 (manual operation). Then, the operation of the blocks is done via the corresponding buttons/switches of the individual blocks. When manual operation is switched off again (bEnVisuControl = FALSE), the program jumps to step 249 and restarts from step 0 after the wait time.

Usage for pulse width modulation

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 IO mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The pulse width modulation functionality of the DC541 is provided by the block *Chapter 1.5.4.11.1.1.10 “DC541_PWM” on page 1146*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The function block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio. The on and off times are adjusted as 8 bit numbers.

The minimum switching time is specified at input CYCLE, i.e. if an output has been switched to FALSE or TRUE by the PWM, this output remains in this state for at least this time (CYCLE µs).

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the function block *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132* (output CYCLE).

Configuration example: Pulse width modulation (PWM)

In the configuration example, channel 0 of the DC541 is configured for pulse width modulation (PWM). The output signal is measured using the function Time and frequency measurement *Chapter 1.6.4.4.1.5 “Usage for time and frequency measurement” on page 5712* of the DC541-CM.

The following on-off ratio shall be used:

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
Cycle time of DC541 = 100 µs			
1	2	2000	20 x TRUE / 40 x FALSE / 20 x TRUE / 40 x FALSE / … i.e. 2000 µs = TRUE and 4000 µs = FALSE

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541 in communication module slot 1 (first slot on the left of the CPU)
- PM591-ETH
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 ---------- DC541 / C1

PLC configuration:

- DC541-CM in slot 1, operating mode "counter mode"			
- Configuration:	- Channel	C0	PWM
		C1	FREQ
		C2...C7	Input
- Specification of the Ethernet communication module as internal communication module (if available)			

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#1ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

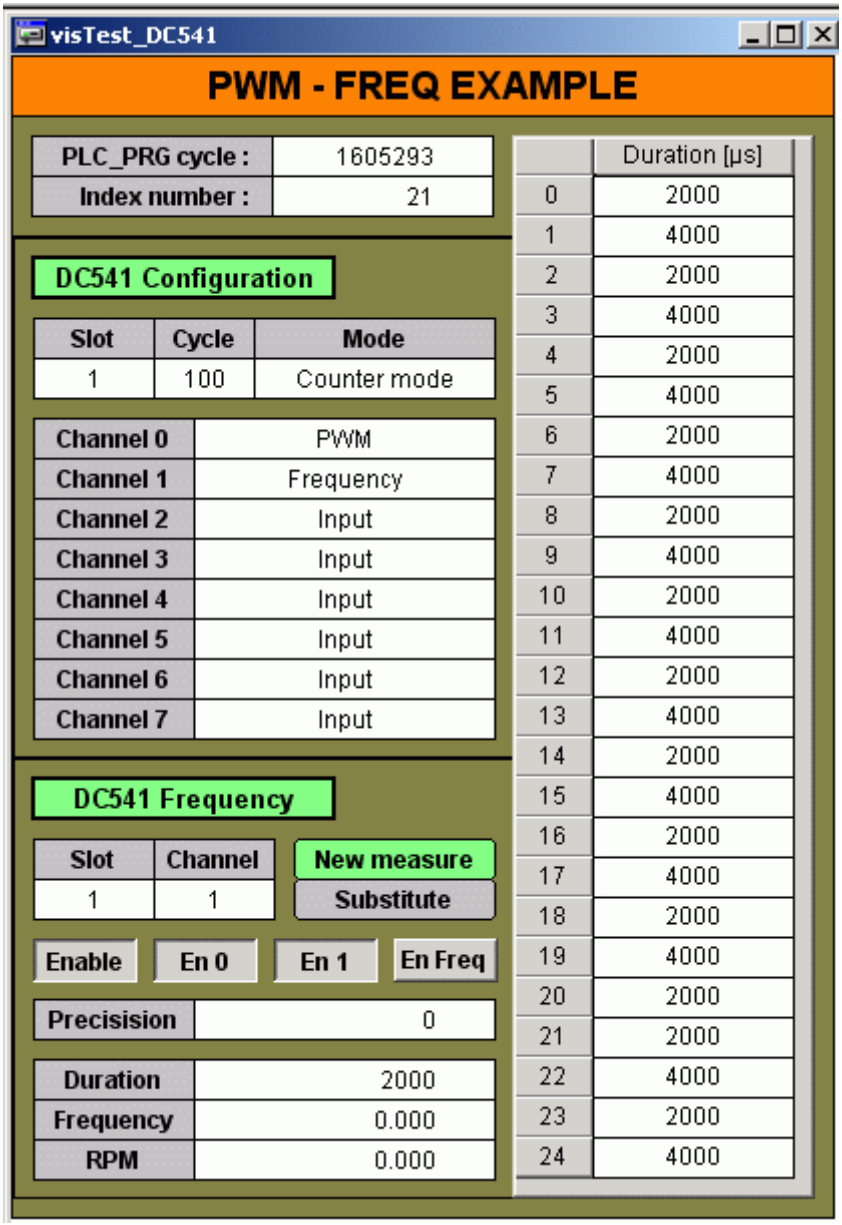
- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for PWM and FREQ Calling of program proPWM_FREQ

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 ↳ *Chapter 1.5.4.19 “Internal system library” on page 1500* and described in detail in the corresponding documentation.

Calling the pulse width modulation functionality as well as measurement and acquisition of measured values are performed in the program proPWM_FREQ. The program proPWM_FREQ contains the calls for the function blocks ↳ *Chapter 1.5.4.11.1.1.10 “DC541_PWM” on page 1146* and ↳ *Chapter 1.5.4.11.1.1.2 “DC541_FREQ” on page 1111* as well as the acquisition of the measured values. The function block DC541_FREQ is configured in a way that it measures the time between each edge change.

The example program has a visualization implemented that displays all states:



Input EN_VISU of the function block DC541_FREQ is TRUE. Therefore, the inputs of the block can be modified using the buttons <Enable>, <En 0>, <En 1> and <En Freq> in the visualization.

The measured values are 2000, 4000 or 6000 µs depending on which edges were considered for measurement.

Usage for time and frequency measurement

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 IO mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The time and frequency measurement functionality of the DC541-CM is provided by the block ↳ *Chapter 1.5.4.11.1.1.2 "DC541_FREQ" on page 1111*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 "DC541 library" on page 1103*.

The function block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

It is able to measure frequencies from 0 to 2000 Hz (2 kHz). In order to obtain a precise measurement of frequencies > 50 Hz, a correspondingly high accuracy setting has to be chosen. It is recommended to use an accuracy of PREC = 1000, i.e. 0.001.

This function block has to be called cyclically, one time per second at least.

The inputs EN_0, EN_1 and EN_FREQ are used to determine the edges to be measured. If input EN_FREQ = TRUE, the frequency and the rotational speed are calculated in addition to the time measurement.

Configuration example: Frequency output

In the configuration example, channel 0 of the DC541-CM is configured for frequency output ↳ *Chapter 1.6.4.4.1.6 "Usage for frequency output" on page 5715*. The output signal is measured using the function "Time and frequency measurement" of the DC541-CM.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet communication module
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 ---------- DC541 / C1

PLC configuration:

DC541-CM in slot 1, operating mode "counter mode"			
Configuration	**Channel**	**C0**	**Frequency output**
		C1	Frequency measure-ment
		C2...C7	Input
Specification of the Ethernet communication module as internal communication module (if available)			

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#5ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

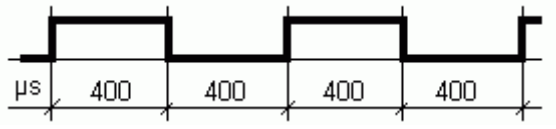
The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for frequency output and measurement Calling of program proFrequency
-

The blocks *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132*, *Chapter 1.5.4.11.1.1.11 “DC541_STATE” on page 1151* and *Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139* are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation *Chapter 1.5.4.11 “DC541 library” on page 1103*.

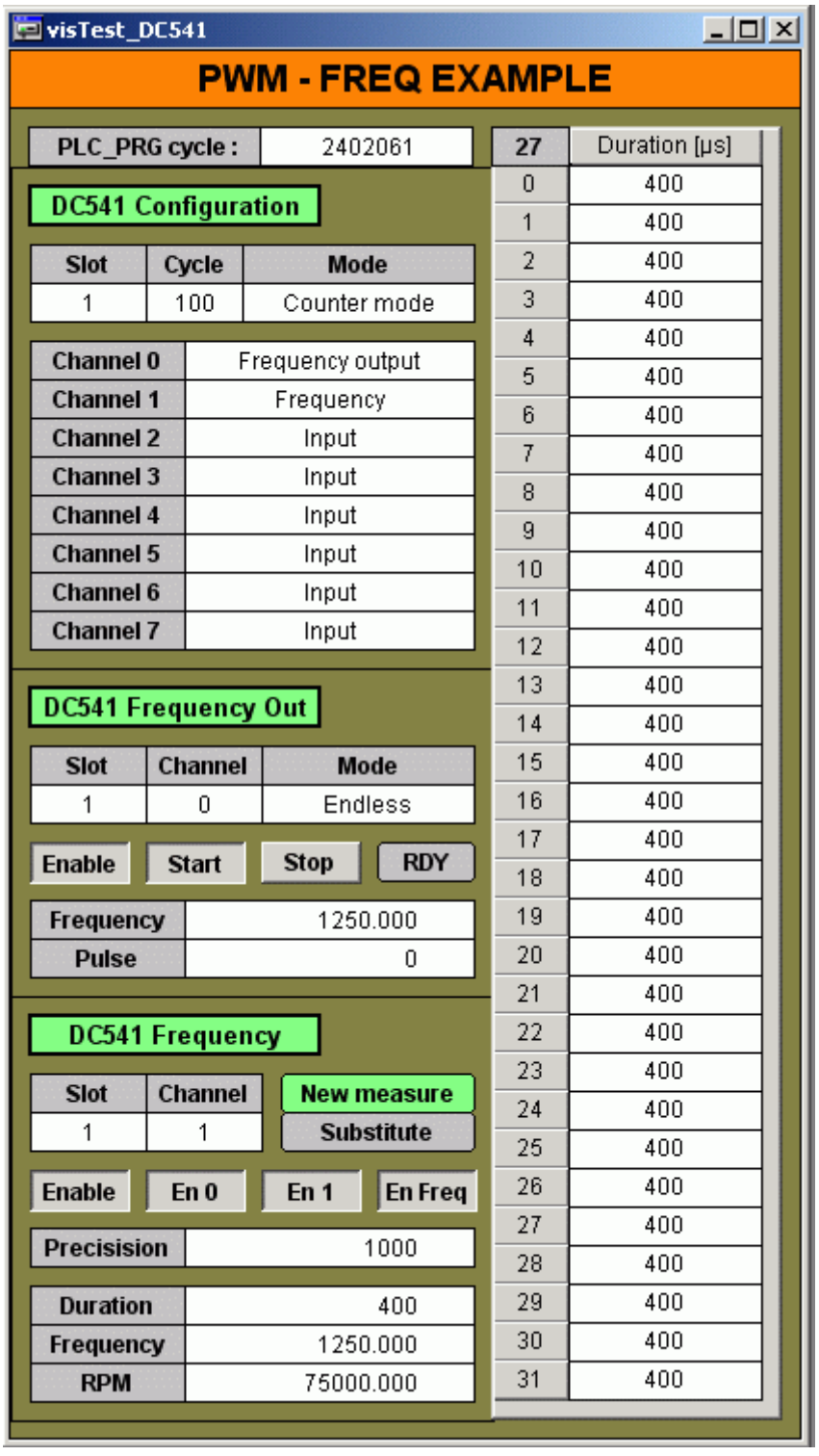
The block *Chapter 1.5.4.19.3.22 “TASK_INFO read number of completed task cycles” on page 1620* is contained in the library SysInt_AC500_V1.0 *Chapter 1.5.4.19 “Internal system library” on page 1500* and described in detail in the corresponding documentation.

The calling of the frequency output functionality as well as the measurement and acquisition of measured values are performed in the program proFrequency. The program proFrequency contains the calls for the function blocks *Chapter 1.5.4.11.1.1.4 “DC541_FREQ_OUT” on page 1123* and *Chapter 1.5.4.11.1.1.2 “DC541_FREQ” on page 1111* as well as the acquisition of the measured values.



The example program has a visualization implemented that displays all states:

Input EN_VISU of the function blocks DC541_FREQ_OUT and DC541_FREQ is TRUE. Therefore, the block inputs can be controlled using the buttons in the visualization.


visTest_DC541
PWM - FREQ EXAMPLE
PLC_PRG cycle :
2402061
DC541 Configuration
Slot
Cycle
Mode
1
100
Counter mode
Channel 0
Frequency output
Channel 1
Frequency
Channel 2
Input
Channel 3
Input
Channel 4
Input
Channel 5
Input
Channel 6
Input
Channel 7
Input
DC541 Frequency Out
Slot
Channel
Mode
1
0
Endless
Enable
Start
Stop
RDY
Frequency
1250.000
Pulse
0
DC541 Frequency
Slot
Channel
New measure
1
1
Substitute
Enable
En 0
En 1
En Freq
Precisision
1000
Duration
400
Frequency
1250.000
RPM
75000.000
27
Duration [µs]
0 400
1 400
2 400
3 400
4 400
5 400
6 400
7 400
8 400
9 400
10 400
11 400
12 400
13 400
14 400
15 400
16 400
17 400
18 400
19 400
20 400
21 400
22 400
23 400
24 400
25 400
26 400
27 400
28 400
29 400
30 400
31 400

Usage for frequency output

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 IO mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The frequency output functionality of the DC541 is provided by the block ↳ *Chapter 1.5.4.11.1.1.4 "DC541_FREQ_OUT" on page 1123*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 "DC541 library" on page 1103*.

The function block DC541_FREQ_OUT is used to output pulses with a fixed frequency on one channel of the device DC541. It is able to output pulses with a frequency between 0.2 and 2.5 kHz. The pulse jitter depends on the cycle time of the DC541. The pulse length is always a multiple of the cycle time of the DC541.

In case of a presetting of PULSE = 0, the output of pulses is infinite. The pulse output is started with a positive edge at input START. The output is aborted if START = FALSE. A positive edge at input STOP interrupts the pulse output. The output is continued if STOP = FALSE.

If input PULSE > 0, the function block outputs the number of pulses specified at input PULSE with the frequency specified at input FREQ on the channel specified at input CH. After the function block has output the number of pulses specified at PULSE, the output RDY becomes TRUE.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency output.

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

For frequency output, the same configuration example is used as for the time and frequency measurement ↳ *Chapter 1.6.4.4.1.5 "Usage for time and frequency measurement" on page 5712*.

Application examples

Application examples of the DC541-CM Interrupt and Counter Module can be found in the *PLC Download Center*.

1.6.4.4.2 CD522 encoder and PWM module

Functionality of the CD522 module

The encoder and PWM module CD522 can be used at the following devices:

- Communication interface modules (e. g. CI501-PNIO, CI541-DP)
- Processor modules

Features:

- 2 independent counting functions with up to 12 configurable modes (including incremental position encoder and frequency input up to 300 kHz)
- 2 independent PWM (pulse-width modulator) or pulse outputs with push-pull driver
- Dedicated inputs/outputs for specific counting functions (e.g. touch, set, reset)
- All unused inputs/outputs can be used with the specifications of standard inputs/outputs range

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Depending on the configuration used, some inputs and outputs are dedicated to specific counting functions (touch, set, reset…). All unused inputs and outputs can be used with the specification of standard inputs/outputs range.

There are special function blocks available to manage and control the function of the CD522 Module. These function blocks are contained in the ↬ *Chapter 1.5.4.8 “CD522 library” on page 972* which is available with a runtime system of version V1.0.2 or above. The library is automatically included into the project when adding a CD522 Module to the Automation Builder project. Details on the hardware is provided in the device descriptions ↬ *Chapter 1.6.2.7.2.1 “CD522 - Encoder, counter and PWM module” on page 4635*.

Special features

The specific functionality is processed within CD522. It works independently of the user program and therefore it is able to response quickly to external signals. A simultaneous counting operation of several expansion modules is possible.

Each module counter can be configured for 1 mode out of 12 possible ones. The desired operating mode is selected in the PLC configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. While integrating a module containing counters in the PLC configuration, the necessary operands are created and reserved immediately. Thus, a counter implementation carried out later does not cause an address shift.

Operating modes

Inputs and outputs, which are not used by the counters, are available for other tasks. In the following table, A means Input Channel A, B means Input Channel B and Z means Output Channel Z.

Operating Mode	Function	Used inputs	Description
0	No counter	None	This operating mode is selected, if the integrated fast counter is not needed.
1	Up/Down counter (A)	A = Counting input	One Up/Down (dynamic changes) counter with set and reset input, end value reached indicator, touch/catch value and overflow flag.
2	Up/Down with release input (B)	A = Counting input B = Enable input	One count up counter with enable input via terminal, counting is valid when input B is true. Dynamic Up/Down count possibility, end value reached indicator, Touch/catch value and Overflow flag
3	Up/Down counters (A,B)	A = Counting input 0 B = Counting input 1	2 counters with separate Up/Down and reset input
4	Up/Down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 counters (counting on falling edge of input B) with separate Up/Down and reset input
5	Up/Down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical input.
6	Up/Down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical input.
7	Reserved	None	---
8	Up/Down with release (B), 0 cross detection	A = Counting input B = Enable input	One 16 bit counter (in range of -32768 to 32767) with zero cross over detection, counting valid when input B is true

Operating Mode	Function	Used inputs	Description
9	Reserved	None	---
10	Reserved	None	---
11	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for encoder x1 count, touch/catch value, RPI function, reset and set
12	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x2 count, touch/catch value, RPI function, reset and set
13	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x4 count, touch/catch value, RPI function, reset and set
14	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface
15	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation

CD522 used as encoder device

Incremental encoder

The function block *Chapter 1.5.4.8.1.1 “CD522_32BIT_ENCODER” on page 972* can be used to control an encoder device for relative positioning with the CD522 Module.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function encoder of module CD522, different operating modes are available. The function block CD522_32BIT_ENCODER should be used with one of these operating modes:

Operating Mode 11	"Incremental encoder"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x1 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 12	"Incremental encoder X2"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x2 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 13	"Incremental encoder X4"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x4 count, with possibility of touch/catch value, RPI function, set and reset actions.	

Absolute SSI encoder

The function block *Chapter 1.5.4.8.1.5 “CD522_SSI_CNT” on page 1006* can be used to control SSI absolute encoder function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_SSI_CNT should be used with one of these operating modes:

Operating Mode 14	"SSI, absolute encoder"
Should be specified in PLC Configuration; parameter mode counter in order to use absolute encoder with SSI interface.	

CD522 used as counter device

32-Bit bidirectional counter

The function block *Chapter 1.5.4.8.1.2 “CD522_32BIT_CNT” on page 982* can be used to control one 32-bit bidirectional counter function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_32BIT_CNT should be used with one of these operating modes:

Operating Mode 1	"Up/Down counter (A)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use one up/down 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 2	"Up/Down counter with release input (B)"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 5	"Up/Down dynamic set (B)/rising edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical.	
Operating Mode 6	"Up/Down dynamic set (B)/falling edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical.	

The module CD522 provides 2 Up/Down 32-bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

16-Bit bidirectional counter

The function block *Chapter 1.5.4.8.1.3 “CD522_16BIT_CNT” on page 991* can be used to control one 16-bit bidirectional counter function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_CNT should be used with one of these operating modes:

Operating Mode 8	**"Up/Down with release (B), 0 cross detection"**
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 16 bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	

The module CD522 provides 2 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

Two 16-bit bidirectional counter

The function block *Chapter 1.5.4.8.1.4 “CD522_16BIT_2CNT” on page 998* can be used to control two 16-bit bidirectional counter functions.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_2CNT should be used with one of these operating modes:

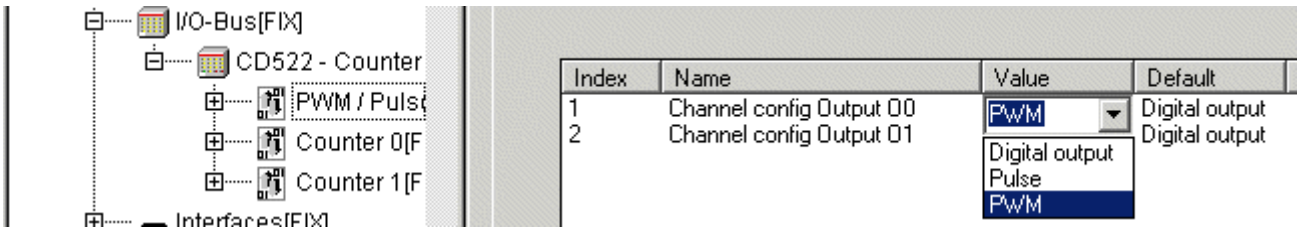
Operating Mode 3	"Up/Down counters (A,B)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use 2 Up/Down 16 bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	
Operating Mode 4	"Up/Down (A, B on falling edges)"
Should be specified in PLC Configuration, parameter mode counter in order to use two Up/Down 16 bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and overflow flag.	

The module CD522 provides 4 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 and B0 for counter A and A1 and B1 for counter B.

CD522 used as PWM output device

To use CD522 as PWM output device, function block *Chapter 1.5.4.8.1.6 “CD522_PWM_OUT” on page 1013* is required.

The module CD522 can be used to control one output pulsing signal (Max= 100 KHz) with an adjustable duty cycle (ON/OFF ratio, max=100%). The PWM operating mode is configured in Automation Builder.



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

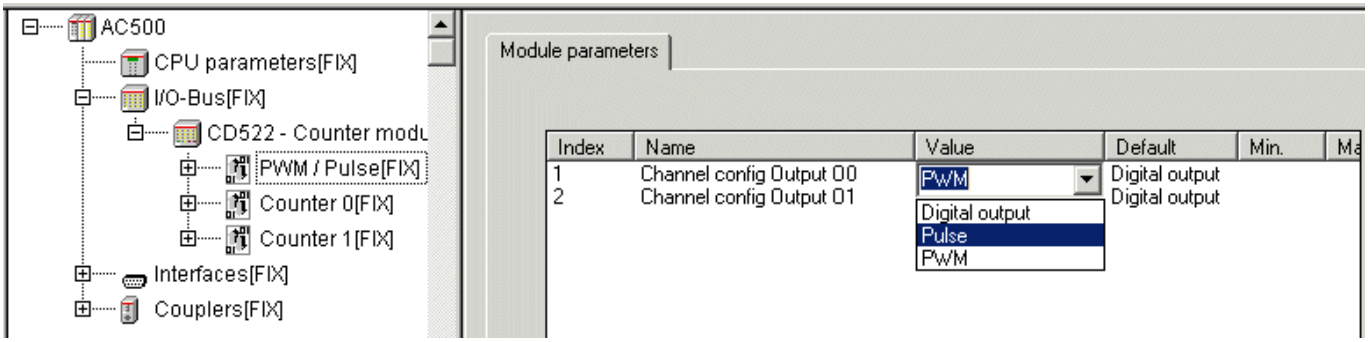
The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_PWM_OUT should be used to control with input EN_PWM, configure the frequency with input FREQ and the input duty cycle DUTY_CYCLE of PWM outputs (pulse-width modulator).

CD522 used as pulse output device

To use CD522 as pulse output device, function block *Chapter 1.5.4.8.1.7 “CD522_PULSE_OUT” on page 1016* is required.

The module CD522 can be used to control one output pulses signals with a fixed duty cycle (ON/OFF ratio 50 %) and number of pulses sent with a fixed frequency (can be modified) .The PULSE operating mode is configured in PLC Configuration using module parameters:



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed by using input and output operands. These necessary operands are created and reserved automatically, when one CD522 module is added into the I/O bus configuration.

The module CD522 provides two independents outputs used in PULSE mode (O0 and O1). Both have the same specification and can work separately.

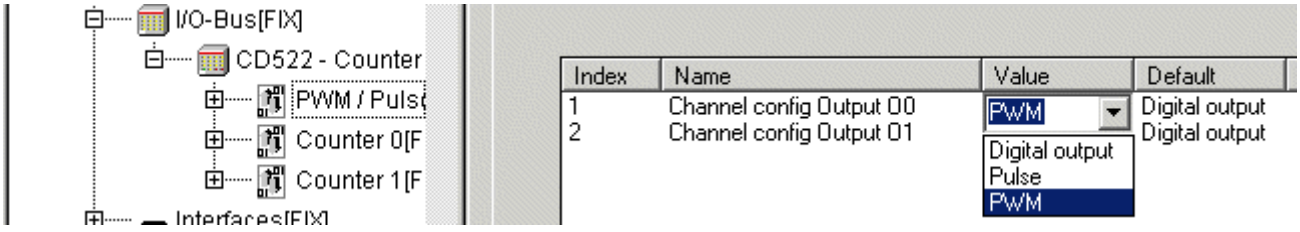
The function block CD522_PULSE_OUT should be used to control the pulse output, with input EN_FREQ, configure the frequency with input FREQ and the number of pulses with input NUM. The number of pulses sent can be displayed in percentage (from 0 % to 100%).

On the fast outputs O0 or O1, the brightness of yellow LED depends on the number of pulse emitted (from 0 and 100%), When the value 100% is obtained, the yellow LED status is off.

CD522 used as frequency output device

To use CD522 as frequency output device, function block *Chapter 1.5.4.8.1.8 “CD522_FREQ_OUT” on page 1020* is required.

The module CD522 can be used to control one output pulses signals with an fixed duty cycle (ON/OFF ratio 50 %).The PWM operating mode is configured in PLC Configuration using module parameters:



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_FREQ_OUT should be used to control with input EN_FREQ and configure the frequency with input FREQ of frequency outputs (1 kHz to 100 kHz).

CD522 used as time frequency meter

To use CD522 to measure times, frequency and rotation speeds on channel Z0 or Z1, function block *Chapter 1.5.4.8.1.9 “CD522_FREQ_SCAN” on page 1024* is required.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_FREQ_SCAN should be used with one of these operating modes:

Operating Mode 15	**"Time frequency meter (Z)"**
Should be specified in PLC Configuration with the parameter mode counter.	

The module CD522 provides 2 channels (Z0 and Z1) which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs. Both have the same specification and can work separately.

The function block CD522_FREQ_SCAN should be used to control with input EN_CNT, configure the capture on falling edge with input EN_0 or rising edge with input EN_1 of signal, and the specification of the mode of the measurement (time, frequency and Rpm) with input EN_FREQ.

The table shows values measured according to configuration input parameters and this example of timing.

NOTICE!

Risk of malfunctions!

Never use the time measurement (bit EN_FREQ=FALSE) mode if the CD522 is connected to a CS31 communication interface module, e. g. CI592.

Depending on the input parameters of function block, the result of time measurement can be measured in time in µs, frequency in Hz or speed of rotation in rotation per minute.

1.6.4.4.3 FM502-CMS function module

Condition monitoring

Components of the condition monitoring system:

- Hardware
  - Function module FM502-CMS for condition monitoring, protection or as precise I/O module *Chapter 1.6.2.7.2.2 “FM502-CMS - Analog measurements” on page 4658*.
  - Function module terminal base TF5x1-CMS *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*
  - Processor module PM592-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
- Configuration of FM502-CMS *Chapter 1.6.4.4.3.2 “FM502-CMS function module” on page 5724*.
- FM502-CMS library: The FM502-CMS library contains function blocks to manage and control the function of the Function Module FM502-CMS. The FM502-CMS library consists of the WAV-File library and the CMS-IO library. Once a Function Module has been added to the configuration, the libraries are automatically included with the next compilation of the project. *Chapter 1.5.8 “FM502-CMS library” on page 2519*.

Introduction to condition monitoring

Condition Monitoring (CM) is a broad term, which can be understood in different ways. For the FM502-CMS, CM means the acquisition and analysis of high-frequency data. CM does usually not occur in real-time. The data is analysed afterwards. If, however, online CM is controller integrated, real-time reaction, e.g. protection, is possible within the same device and using the same sensors. This feature is supported by AC500 and FM502.

The focus is often merely on mechanical CM. This is due to the fact that the movement or rotations of large masses which are connected to a motor (e.g. electric machine) via a shaft pose the greatest danger. Machines in operation inevitably generate measurable vibration, both free or forced, even in the normal operating states and in absence of any damage.

Yet, in electrical CM, electrical high-frequency quantities like currents, voltages or partial discharges can be measured by suitable sensors and can be analyzed in order to detect electrical failure patterns, e.g. inside electrical machines or equipment (transformers). Some electrically measurable failure patters can be induced by mechanical issues e.g. vibration.

Effective fault detection will only be possible if the data patterns indicating an arising defect can be singled out among the data collected.

Monitoring e.g. the vibration characteristics of a machine in operation gives an understanding of the "'health" condition of the machine and its development over time and load. This information can be used to detect arising problems at an early stage. Operating a machine until it breaks down might be acceptable if the machine were a "'disposable" one and the lifetime was very high and known for sure. But many failures can be considered statistical outliers and occur very early and spontaneously.

However, most machines are not "disposable" due to their cost. Therefore, regular monitoring of a machine’s condition can reveal potential problems. Subsequently, counter measures can be taken at an early stage in order to minimize damage and associated cost. If monitoring is permanent and controller integrated, even spontaneous failures can be detected in real-time, in order to prevent substantial damage to a larger part of the equipment and its environment.

Use cases of FM502-CMS

- Condition monitoring: Longer data-stream analyses
- Protection: Fast reaction to e.g. direct or RMS values based on limits
- Fast and precise analog measurements as with any other AC500 I/O module, but even more precise and faster
- Data logging: Fast, efficient data storage

Condition Monitoring typically means acquisition of longer data streams, also called “time series” or “signals”. These can be analyzed after measurement in the time domain (e.g. envelope, statistical analysis) or frequency domain (e.g. spectrum analysis).

The Function Module FM502-CMS works independently of the user program and therefore it is able to response quickly to external signals and fast acquisition of the analog channel values.

The analog channels can be configured in different modes. It is also possible to change individually the sample and acquiring settings for each channel. The counter can be configured in different modes. After that, it is activated during the initialization phase (power-on, cold start, warm start).

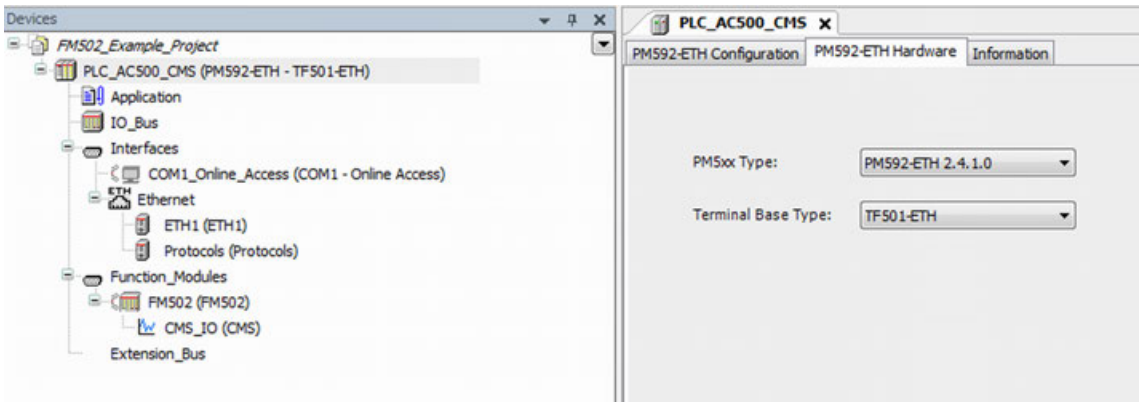
The data exchange to and from the user program is performed using input and output operands. While integrating a module containing counters in the PLC configuration, the necessary operands are created and reserved immediately. For the fast data acquisition the values are stored inside the module in a file and transferred via file transfer to the PLC after the measurement sequence is finished.

FM502-CMS function module

Preconditions

The hardware structure is automatically generated in configuration.

You can change the type of the TF5x1-CMS in the hardware configuration of the processor module:



Only one FM502-CMS can be connected to a processor module.

> *FM502-CMS cannot be used as an I/O on a remote communication interface.*

Configuration of FM502-CMS

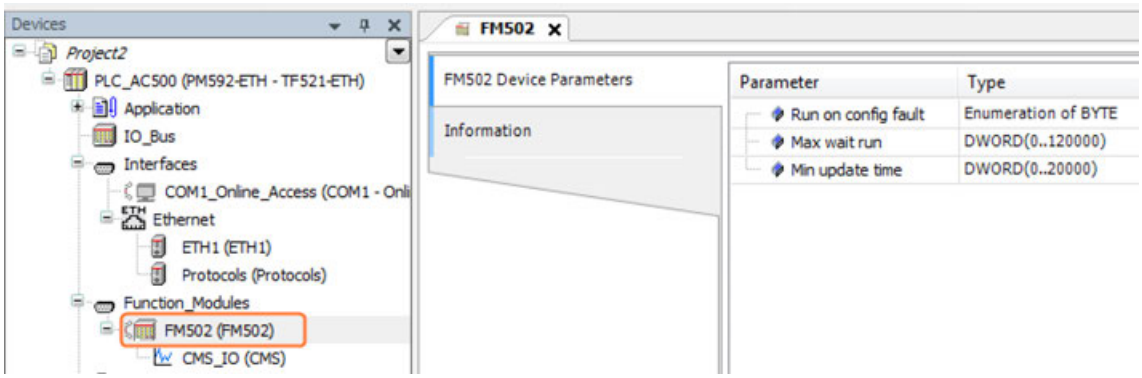


Fig. 1139: Change the behavior on the internal bus.

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not starting.
		Yes	The user program runs independent of a faulty I/O bus configuration
Max wait run	3000 [ms]	-	Maximum wait time for valid inputs (Do not change this parameter)
Min update time	10 [ms]	-	Cycle time for data exchange to IEC program The “Min update time” defines the time how often the I/O image will updated for the IEC program. If the IEC task is faster, the I/O image is not updated on every cycle.
Watchdog	400 [ms]	-	Watchdog time (Do not change this parameter)

Parameterization

The FM502-CMS does not store configuration data itself. The digital inputs and outputs can have specific functions depending on the selected configuration mode. All not otherwise configured inputs and outputs can be used with the specification of standard input/output range.

To change the parameterization you can choose the default parameterization in Automation Builder or the parameterization during run time in CODESYS.

The functionality of the module is directly influenced by the parameterization.

Parameterization - Automation Builder

For non-standard applications, it is necessary to adapt the parameters to your system configuration. After every startup of the device the default parameter set will be downloaded to the module.

The parameterization is divided in the following sections:

- Parameter for encoder/counter functionality
- Parameter for analog channel functionality
- Parameter for digital I/O configuration

Parameterization - CODESYS

In CODESYS for the parameterization you can use function blocks of FM502-CMS library ↳ *Chapter 1.5.8.2.1.9 “CMS_IO_CFG_WRITE” on page 2552* ↳ *Chapter 1.5.8.2.1.8 “CMS_IO_CFG_READ” on page 2550*.

Parameter set

Table 715: Encoder/Counter

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
145	Mode	↳ *“Operation modes” on page 5728*	0-15	BYTE	0	0	15
146	Frequency limit	No filter	0	BYTE	0	0	4
		50 Hz	1				
		500 Hz	2				
		5 kHz	3				
		20 kHz	4				
147	Input level	0 - 24 V DC	0	BYTE	0	0	3
		0 - 5 V DC	1				
		Differential	2				
		1 Vss sinus	3				
148	SSI frequency	200 kHz	2	BYTE	2	2	
		500 kHz	3				
		1 MHz	4				
149	SSI resolution	8 Bit	0	BYTE	1	0	3
		16 Bit	1				
		24 Bit	2				
		32 Bit	3				
150	SSI code type	Binary input	0	BYTE	0	0	0
151	SSI polling time	x	x	BYTE	10 ms	0	255

Table 716: Analog channels

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
1	Available Channel	Disabled	0	BYTE	0	0	1
		Enabled	1				
2	Analog Mode	IEPE	0	BYTE	0	0	1
		+/- 10 V	1				
3	Synchronized encoder file	Disabled	0	BYTE	0	0	1
		Enabled	1				
4	DC Filter	Disabled	0	BYTE	0	0	1
		Enabled	1				
5	Sample rate	50 kHz	0	BYTE	0	0	9
		25 kHz	1				
		12,50 kHz	2				
		6,25 kHz	3				
		3,13 kHz	4				
		1,56 kHz	5				
		0,78 kHz	6				
		0,39 kHz	7				
		0,20 kHz	8				
		0,10 kHz	9				
		(Reserved)	10...15				
6	Start condition	Immediate	0	BYTE	0	0	4
		Delayed	1				
		Binary Input	2				
		Zero Input	3				
		Encoder Value	4				
7	Start condition value (dependend on Start condition)	x	x	DWORD	0	0	4294967296
		I1	0	-	0	0	3
		I2	1				
		C3	2				
		C4	3				
8	Edge type	Rising edge	0	BYTE	0	0	1
		Falling edge	1				
9	Record length value (samples)	x	x	DWORD	5000	1	4294967296

Table 717: I/O configuration: Inputs

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
152	Input delay	0.1 ms	0	BYTE	0	0	3
		1 ms	1				

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
		8 ms	2				
		32 ms	3				
153-156	DI0/DI1/DC2/DC3 use	Digital input	0	BYTE	-	0	5
		Touch	1				
		Reset	2				
		Reset 2nd Bit Counter	3				
		Set	4				
		RPI	5				

Table 718: I/O configuration: Outputs

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
157	Behavior on STOP	Off	0	BYTE	0	0	2
		Last value	1				
		Substitute value	2				
158	Substitute value 10 s	x	x	WORD	0	0	3
159/160	DC2/DC3 use	Digital output	0	BYTE	0	0	3
		Analog Channel failure	1				
		Module failure	2				
		End value	3				

Operation modes

Inputs and outputs which are not used by the counters are available for other tasks.

Table legend: A = input channel A, B = input channel B, Z = output channel Z.

Operation Mode	Function	Used inputs	Description	Function block
0-1	No counter	None	This operating mode is selected, if the integrated high-speed counter is not needed.	-
1-1	Up/down counter (A)	A = Counting input	1 bidirectional 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT

Operation Mode	Function	Used inputs	Description	Function block
2-1	Up/down with release input (B)	A = Counting input B = Enable input	1 bidirectional 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT
3-2	Up/down counters (A,B)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	CMS_IO_16BIT_2CNT
4-2	Up/down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and overflow flag.	CMS_IO_16BIT_2CNT
5-1	Up/down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge sets START_VALUE.	CMS_IO_32BIT_CNT
6-1	Up/down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge sets START_VALUE.	CMS_IO_32BIT_CNT
7-1	Reserved	None	-	-
8-1	Up/down with release (B), 0 cross detection	A = Counting input B = Enable input	1 bidirectional 16-bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	CMS_IO_16BIT_CNT
9-1	Reserved	None	-	-
10-1	Reserved	None	-	-

Opera-tion Mode	Function	Used inputs	Description	Function block
11-1	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for encoder x1 count, touch/ catch value, RPI function, reset and set Function block counts rising edges at input A.	CMS_IO_32BIT_ENCODER
12-1	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x2 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A.	CMS_IO_32BIT_ENCODER
13-1	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x4 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A and B.	CMS_IO_32BIT_ENCODER
14-1	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface	CMS_IO_SSI_CNT
15-1	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation	CMS_IO_FREQ_SCAN

Process image (I/O data)

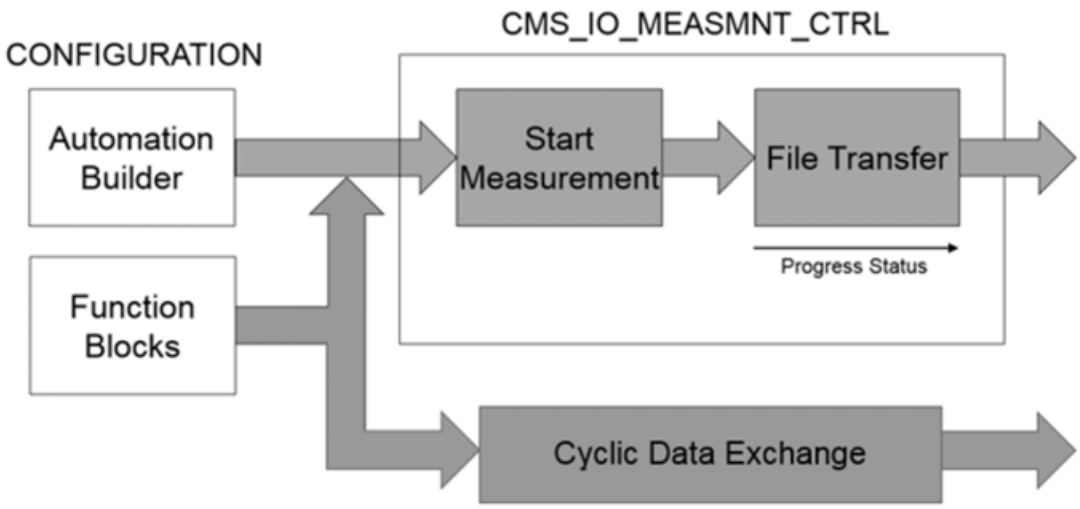
The cyclic data can be accessed to the addresses or variables defined in Automation Builder **I/O Mapping** tab.

FM502-CMS analog measurement

Possibilities to use the analog input signal values:

- The values of the configured analog channels in CODESYS can be used by referencing the I/O mapping variables. The refresh time for the cyclic data exchange of the analog input data is according to the minimum update time in the configuration.
- For detailed data analysis you can record the analog input data into WAV files and store them. In these files, every input data sample is stored.

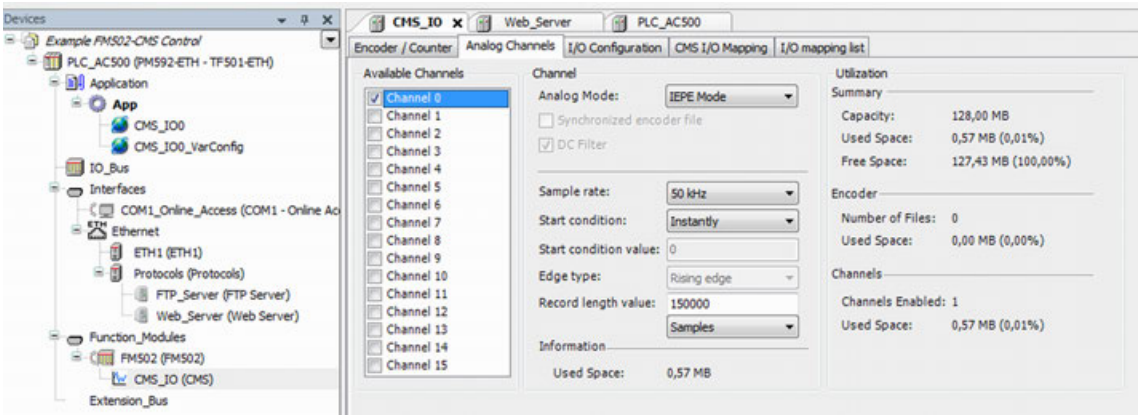
Both possibilities can be used simultaneously.



The DC filter needs 10 seconds to tune in after system power up. During that time the AI values can be accessed by the user program, but the readings are out of tolerance limit.

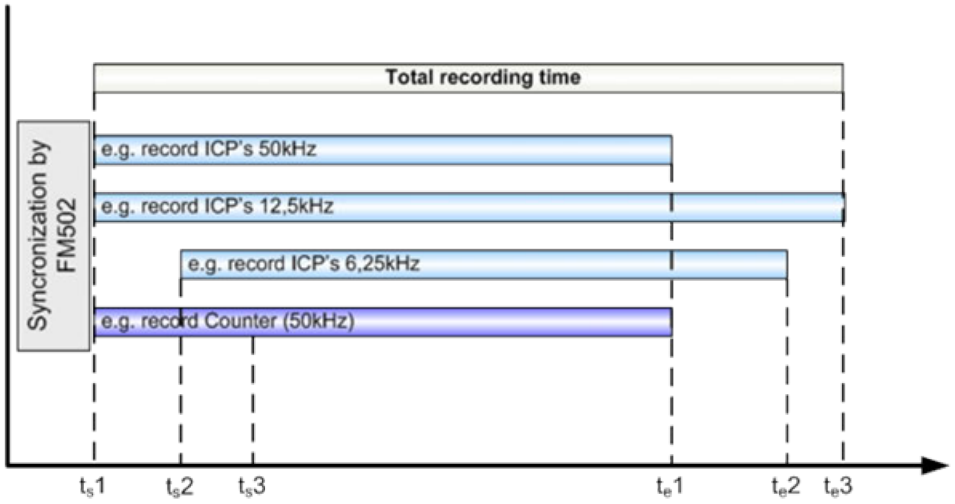
Configuration for analog measurements

▷ Configure at least one analog channel for data acquisition.



The measurement file size depends on the record length value and number of available channels. The maximum capacity of one measurement file is limited by the internal memory. The recording time for each channel is calculated by: recording time = record length value/sample rate.

The total recording time is determined by the earliest measurement start trigger of a channel to the measurement end of the last channel in the configuration.

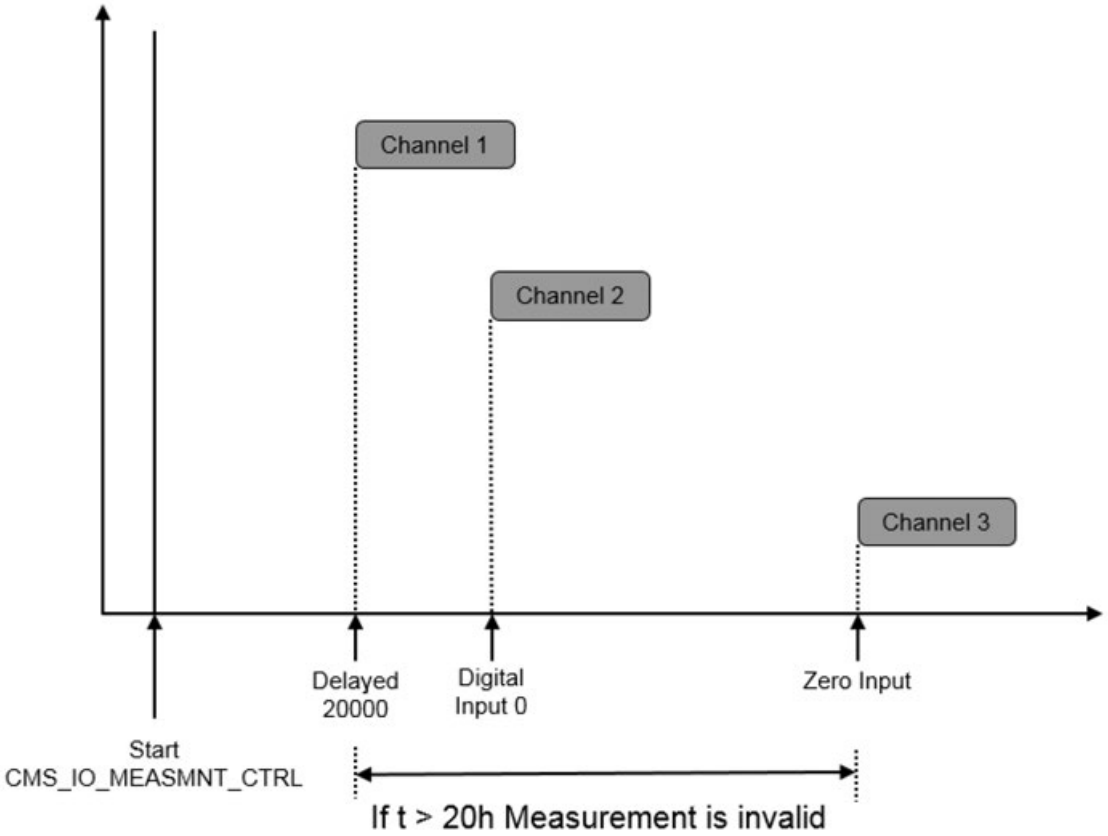


Make sure to configure the right analog mode for each channel. Take care that the sample rate is in relation to the maximum measured frequency.

Start conditions for each individual channel:

- Instantly: Starts measurement instantly after setting input EN to TRUE.
- Delayed: Measurement will start when input EN was set to TRUE and the start condition value [samples] is over.
- Digital input: Starts the measurement when input EN and the digital input was set to TRUE.
- Zero input: Starts the measurement when input EN was set to TRUE and the input Z+ gets a rising edge.
- Encoder value: Starts the measurement when input EN was set to TRUE and the start condition value is satisfied.

The start criteria of all analog channels in one measurement have to be in a 20 hour time window. Otherwise, the measurement is invalid and will be aborted.



The process image (I/O data) of the function module can be mapped for analog measurements.

Measurement files

The measurement data will be stored in the WAV file format. One WAV file will be created for each active channel.

Table 719: RIFF header

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	0 (0x00)	bfChunkID	"RIFX"
DWORD	Little	4	4 (0x04)	dwChunkSize	Data length - 8
BYTE[4]	Big	4	8 (0x08)	bfRiffType	"WAVE"

Table 720: Format chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	12 (0x0C)	bfChunkID	"fmt"
DWORD	Little	4	16 (0x10)	dwChunkSize	Data length - 8
INT	Little	2	18 (0x12)	wFormatTag	0x0001 (PCM)
INT	Little	2	20 (0x14)	wChannels	0x0001 (1 ch.)

Data type	Endian	Length	File offset	Identifier	Value
DWORD	Little	4	24 (0x18)	dwSamples-PerSec	100 Hz - 50.000 kHz
DWORD	Little	4	28 (0x1C)	dwBytes-PerSec	Sample rate * block align
WORD	Little	2	32 (0x1E)	wBlockAlign	4 byte
WORD	Little	2	34 (0x20)	wBitsPer-Sample	32 bit

Table 721: Data chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	36 (0x24)	bfChunkID	"data"
DWORD	Little	4	40 (0x28)	dwChunkSize	Data length - 8
BYTE[]	Big	Undefined	44 (0x2C)	bfData	Measurement data

Table 722: Label chunk

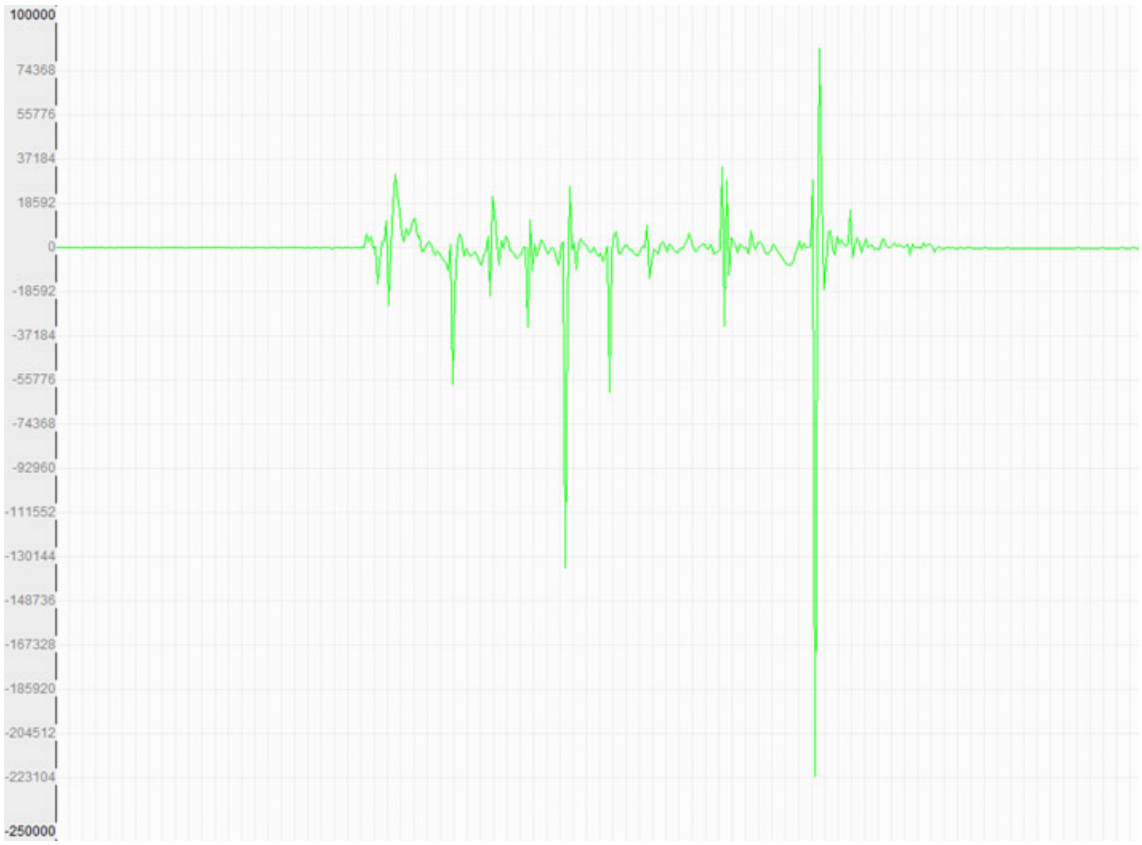
Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	44+sz(bfData)	bfChunkID	“labl”
DWORD	Little	4	48+sz(bfData)	dwChunkSize	Data length -8
DINT	Little	4	52+sz(bfData)	dwIdentifier	Identifier
BYTE[256]	Little	255	56+sz(bfData)	bfText	„Label Text“

The WAV files will be stored in an uncompressed ZIP file at the destination path of CMS_IO_MEASMNT_CTRL. The file names for of the WAV files are given by the FM502-CMS and are directly corresponding to the analog channel and encoder configuration of the FM502-CMS.

Example With no encoder, the files are named: CH00_nEN.wav, CH01_nEN.wav, ... CH15_nEN.wav and stored in a ZIP file.

Programming

In CODESYS, the data of the configured analog channels can be seen and used with the I/O mapping variables.



Use the function block CMS_IO_MEASMNT_CTRL to start a measurement.

> *The name of the measurement ZIP file has to be in 8.3 file format. Example: abcdefgh.zip*

After the measurement is finished, the ZIP file is transferred from the FM502-CMS to the PM592-ETH via communication module bus. The progress of the data transfer can be seen at the output PROGRESS in percent. After a successful measurement and data transfer, the output DONE will change to TRUE.

FM502-CMS used as counter device

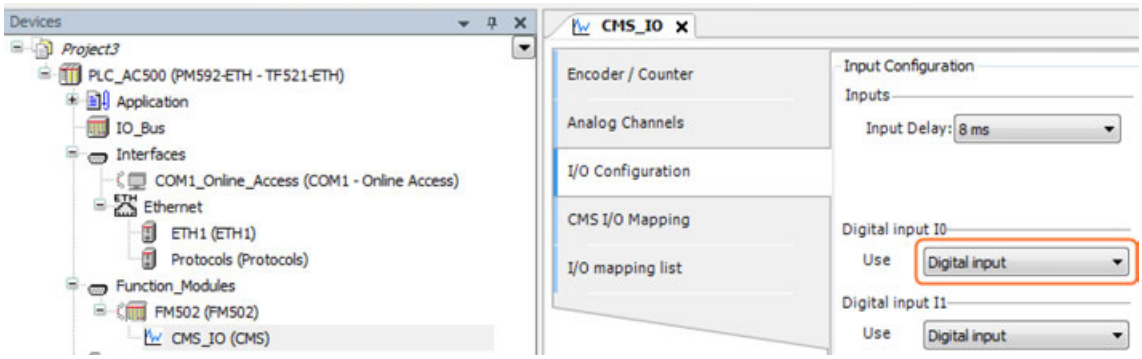


Fig. 1140: Input configuration.

32-bit bidirectional counter

The function block CMS_IO_32BIT_CNT can be used to control one 32-bit bidirectional counter function *Chapter 1.5.8.2.1.3 “CMS_IO_32BIT_CNT 32-bit counter” on page 2534*. A signal used for pulse count is identified by A+. Another signal used for enable or dynamic set is identified by B+.

Possible operation modes: 1-1, 2-1, 5-1, 6-1 *Table on page 5728*

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

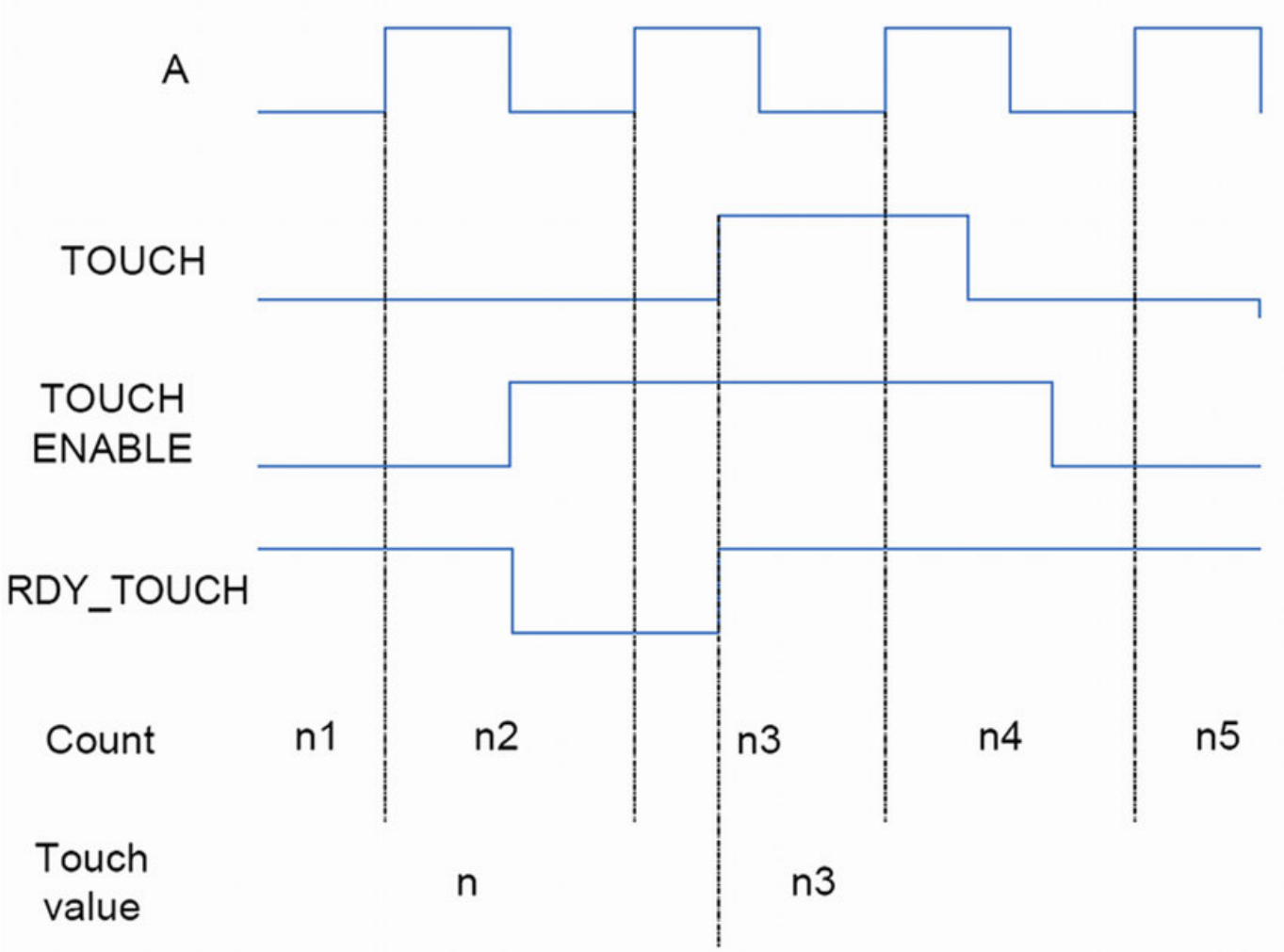


Fig. 1141: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

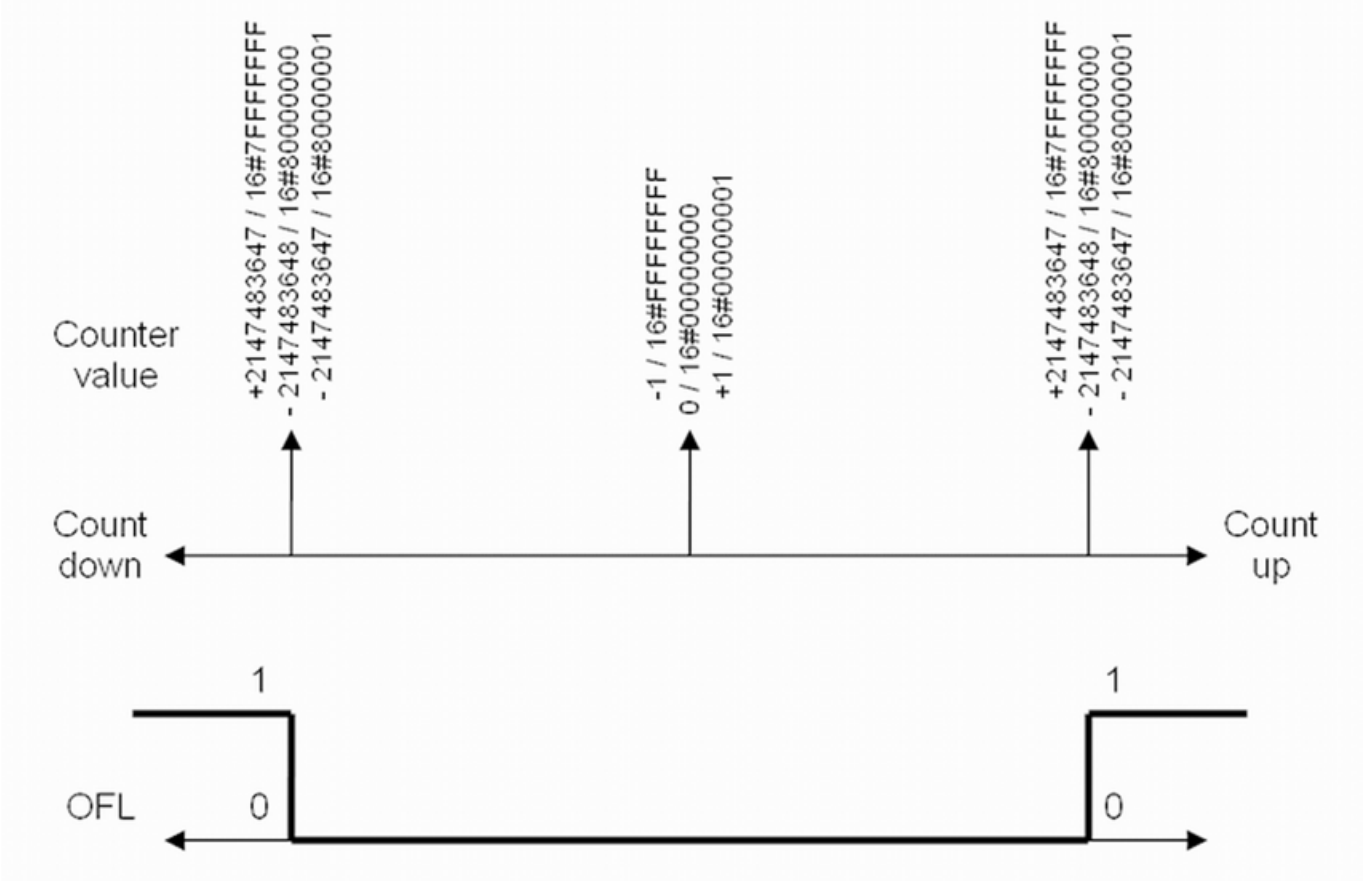
A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

SET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Overflow operation The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



16-bit bidirectional counter

The function block CMS_IO_16BIT_CNT can be used to control one 16 bit bidirectional counter function *Chapter 1.5.8.2.1.2 “CMS_IO_16BIT_CNT” on page 2530*. A signal, used for pulse count, is identified by A+.

Possible operation modes: 8-1 *Table on page 5728*

Touch/Catch operation The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

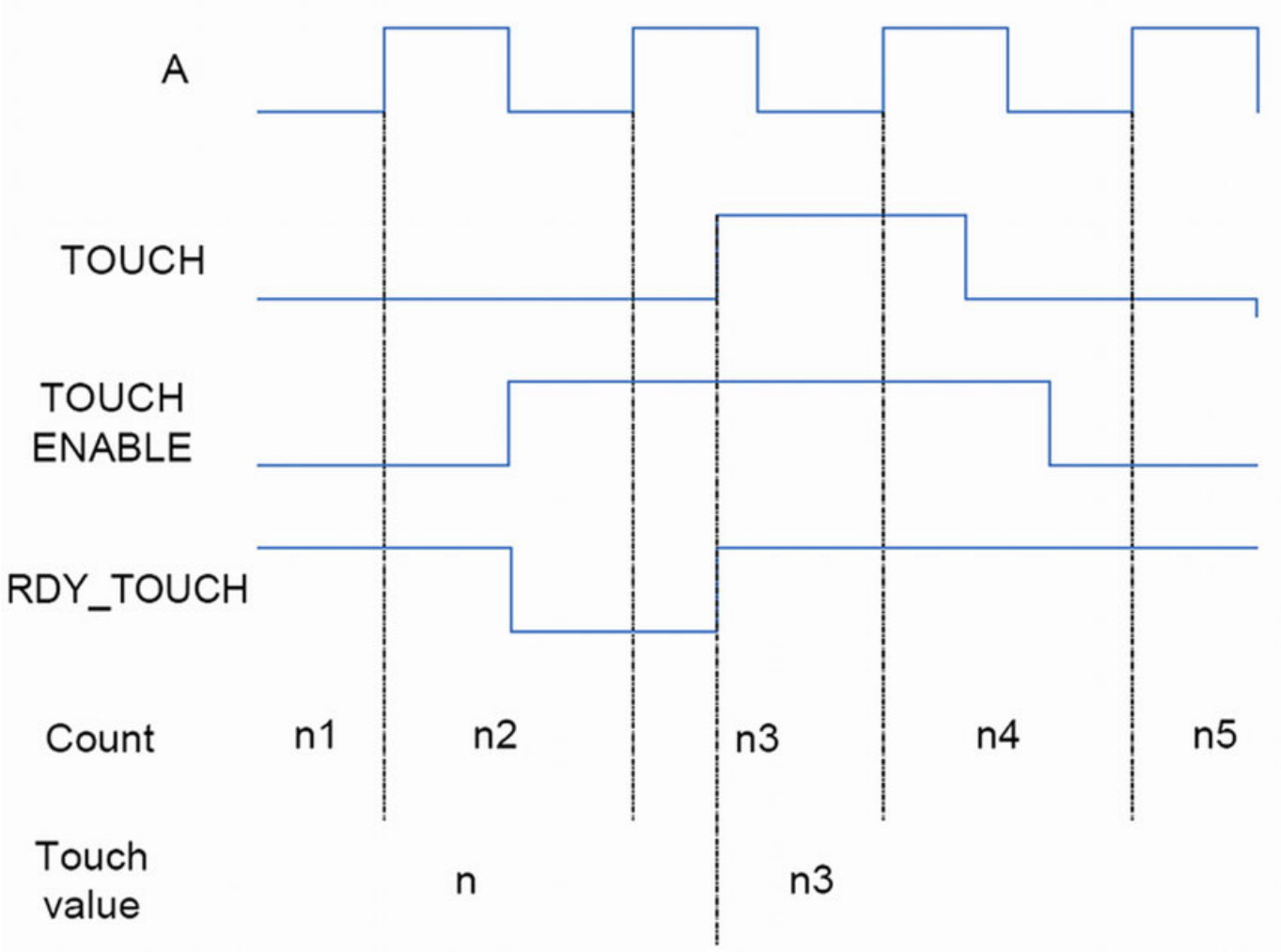


Fig. 1142: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

SET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Two 16 bit up/down counters

The function block CMS_IO_16BIT_2CNT can be used to control two independent 16 bit bidirectional counter functions ↬ *Chapter 1.5.8.2.1.1 “CMS_IO_16BIT_2CNT” on page 2526*. A signal, used for pulse count, is identified by A+ and B+.

Possible operation modes: 3-2, 4-2 ↬ *Table on page 5728*

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

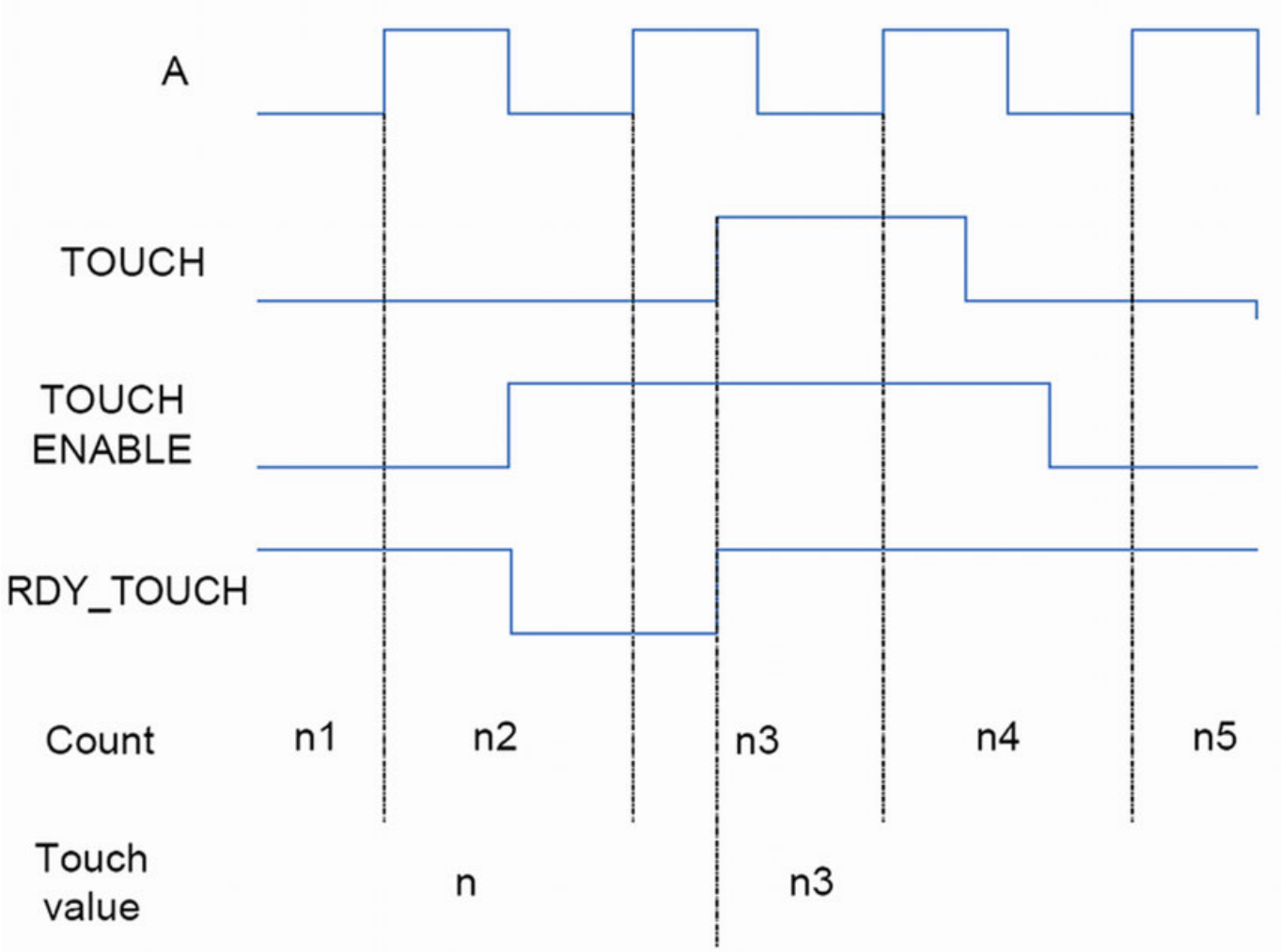


Fig. 1143: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 which is configured as RESET causes the function block to reset the value at output ACT1. A rising edge at input DI0, DI1, DC2 or DC3 which is configured as Reset 2nd Bit counter causes the function block to reset the value at output ACT2.

Overflow operation

The counter operates as an infinite counter. It is set to TRUE, when an overflow occurs, i.e. the counter value ACT1 or ACT2 goes up to value 16#FFFF= -1. Any exceeding or falling below of this value (depending to up use and down use) will set OFL1 = TRUE or OFL2 = TRUE.

FM502-CMS used as encoder device

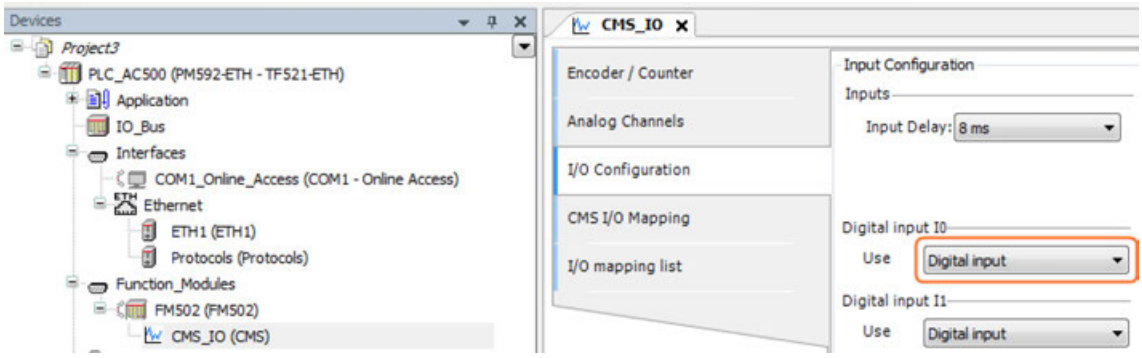


Fig. 1144: Input configuration.

Incremental encoder

The function block CMS_IO_32BIT_ENCODER can be used to control an encoder device for relative positioning with 3 signals ↳ *Chapter 1.5.8.2.1.4 “CMS_IO_32BIT_ENCODER” on page 2538*. 2 signals are used for rotation discrimination and pulse count, identified by A+ and B+. The third one is used in multi-turn encoder to count the number of rotation (mechanical zero), identified by Z+.

The rotation is identified with a shift angle (90°) between A and B signal. In the Function Module, the clockwise rotation is identified with A signal in advance to B.

Possible operation modes: 11-1, 12-1, 13-1 ↳ *Table on page 5728*

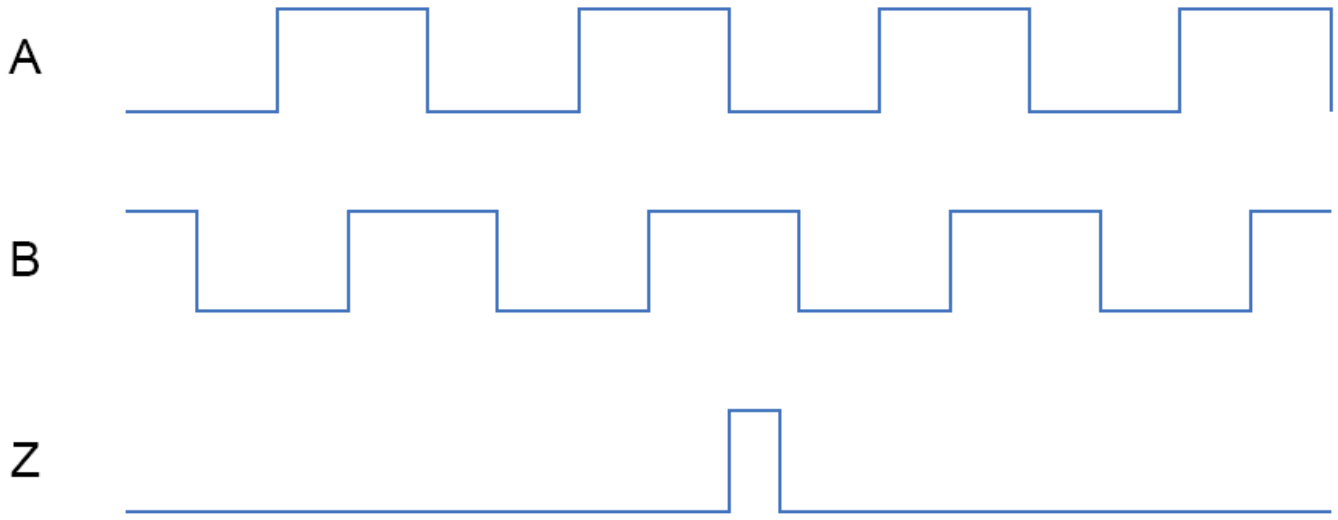


Fig. 1145: Clockwise rotation in the Function Module: A signal ahead from B signal

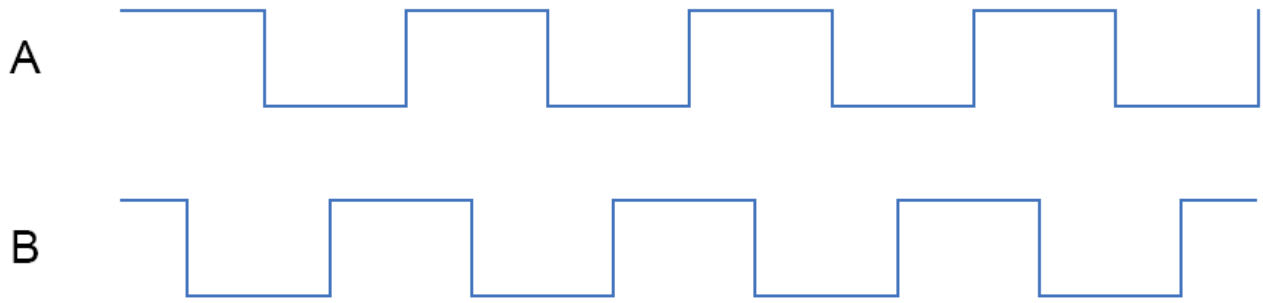


Fig. 1146: Counter-clockwise rotation in the Function Module: A signal late from B signal

Depending on choosen operation mode, the counting procedure will be x1, x2 or x4 count. Basically the x1 counting mode is used (operation mode 11-1). The encoder module discriminates the rotating way and count one pulse for each rising edge of the A signal.

With clockwise rotation, function block CMS_II_32BIT_ENCODER counts downwards. With counter-clockwise rotation, function block counts upwards.

In order to increase resolution, the x2 counting mode can be specified (operation mode 12-1). The encoder module counts one pulse on each rising and falling edge of A signal.

The resolution could be multiplied by 4, using the x4 counting mode (operation mode 13-1). The encoder module counts a pulse on both rising and falling edge of A signal and B signal.

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

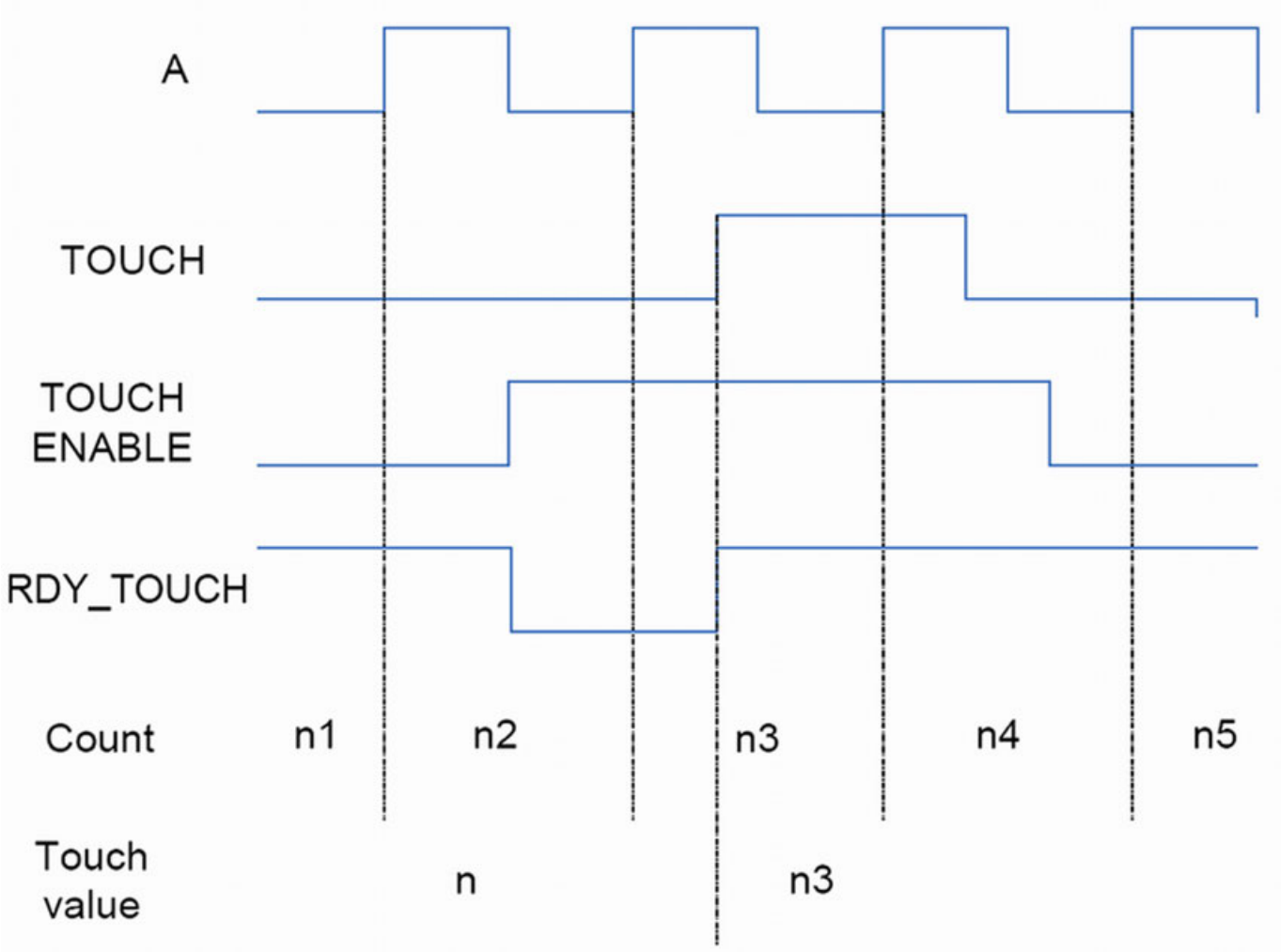


Fig. 1147: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

RPI procedure

The RPI (Reference Point Initialization) is used to synchronize the counter value with the mechanical zero reference based on signal Z.

RPI procedure is enabled with control bit (EN_RPI). If this control bit is set, the module checks for the Z signal. When the signal appears, the set value is copied in the current counter value and RDY_RPI is set (see figure below).

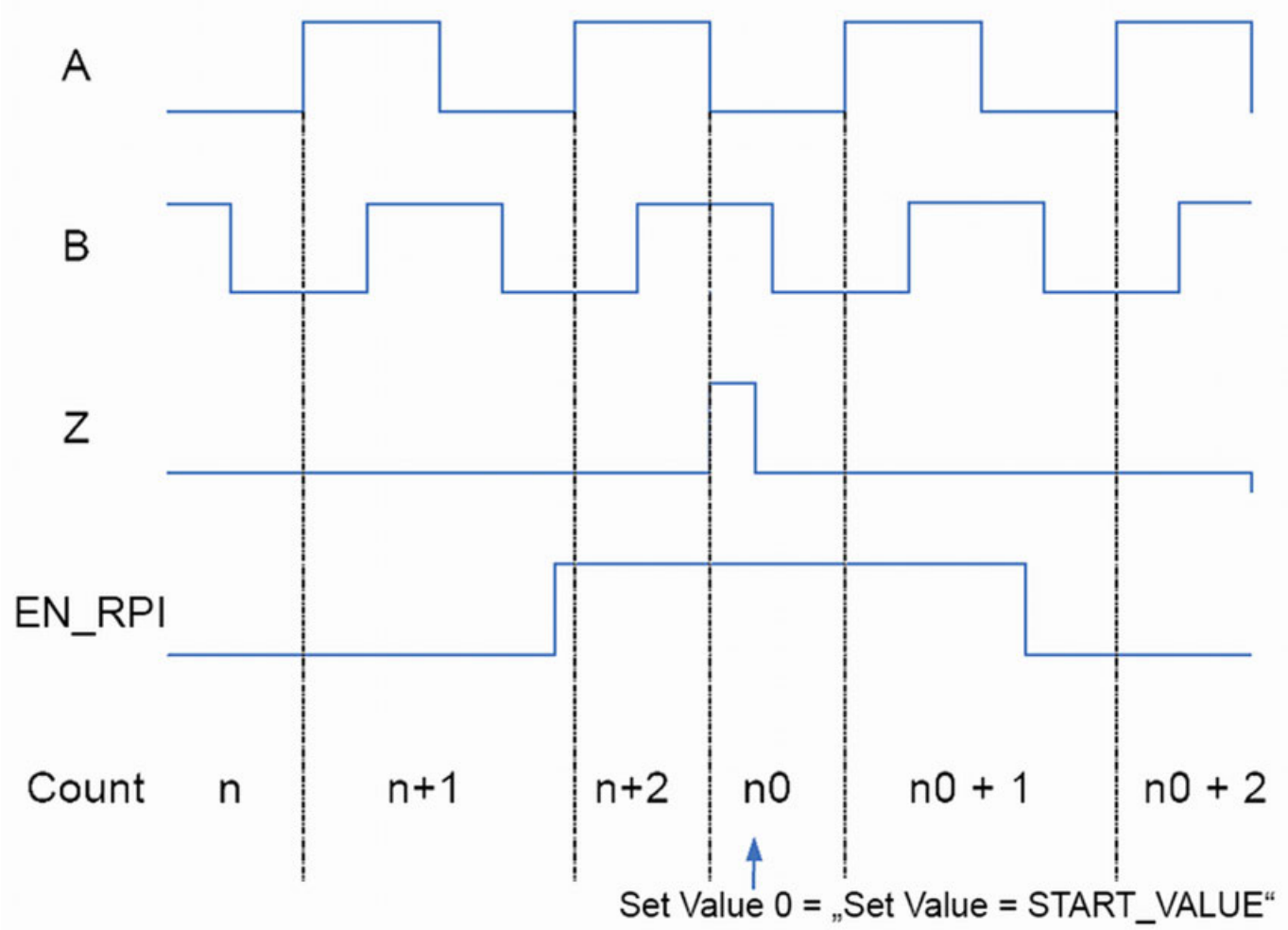


Fig. 1148: RPI operation

A rising edge at the digital inputs DI0, DI1, DC2 or DC3 activates the counter value capture and the counter reset during the capture.

Only one function may be enabled at one time, either the RPI (reference point indicator) or TOUCH (touch trigger measurement) function. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, the RPI function will have a higher priority than the TOUCH function.

SET operation

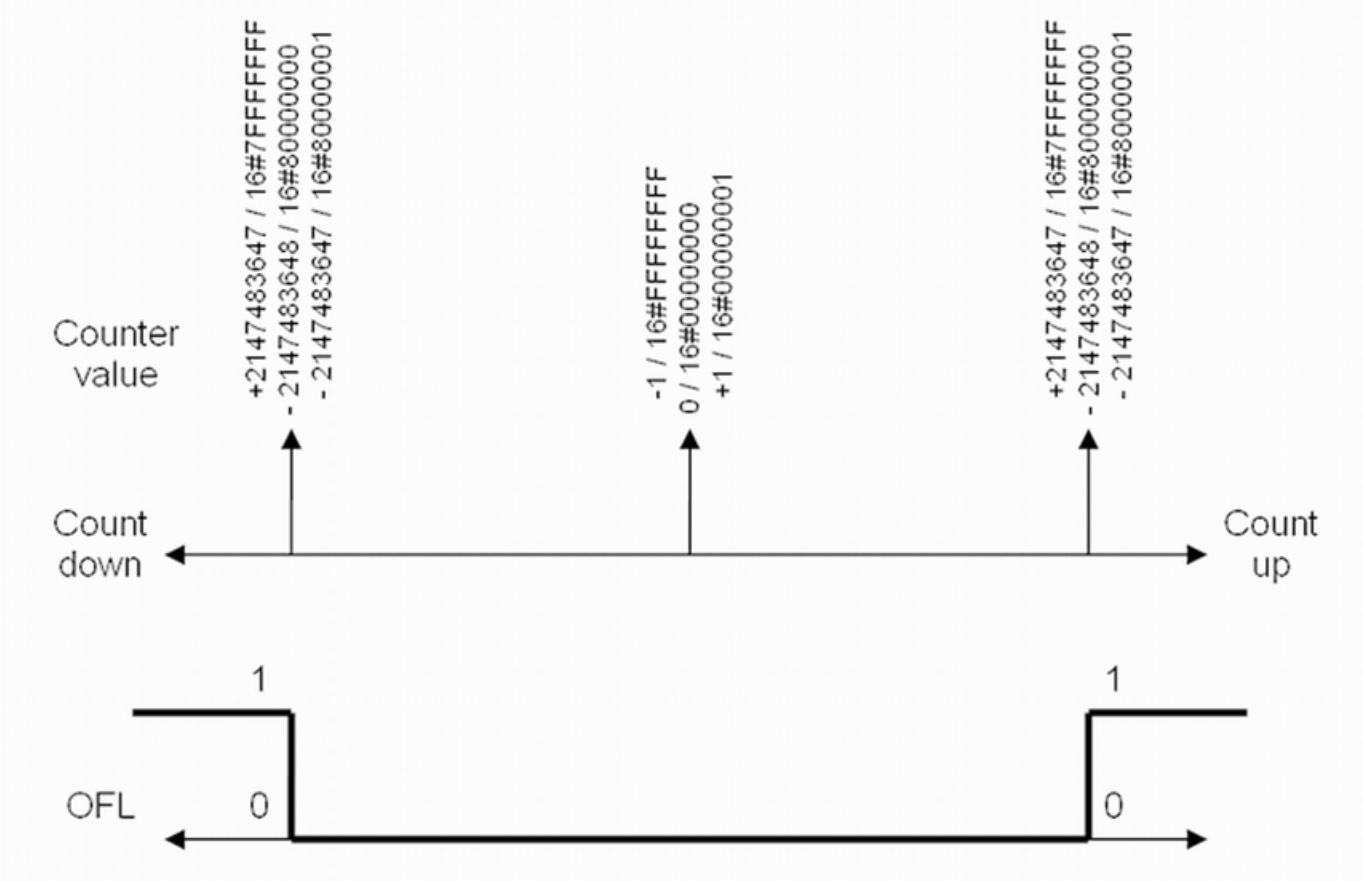
A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Overflow operation

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



Absolute SSI encoder

The function block CMS_IO_SSI_CNT can be used to control SSI absolute encoder function ↳ *Chapter 1.5.8.2.1.6 "CMS_IO_SSI_ENC" on page 2545*. There is an interface for absolute angle and linear encoders (displacement measurement systems).

It allows the transmission of absolute position information through a serial data transfer.

The transmission is based on synchronous serial communication. The device sends a clock signal to the encoder and synchronously, the encoder returns the positioning data from the most significant to the less significant bit.

The synchronization for a new data stream is based on time without clock pulse. This quiet time depends on the encoder.

Possible operation modes: 14-1 ↳ *Table on page 5728*

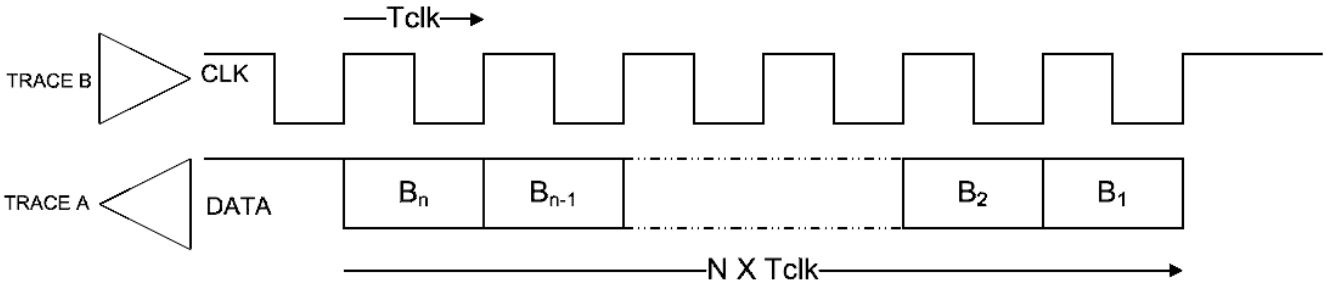


Fig. 1149: Chronogram with data organization with the clock pulse

Resolution For the resolution of the encoder device see technical data from manufacturer. The resolution can be set in configuration under *"SSI parameters ➔ SSI resolution"*.

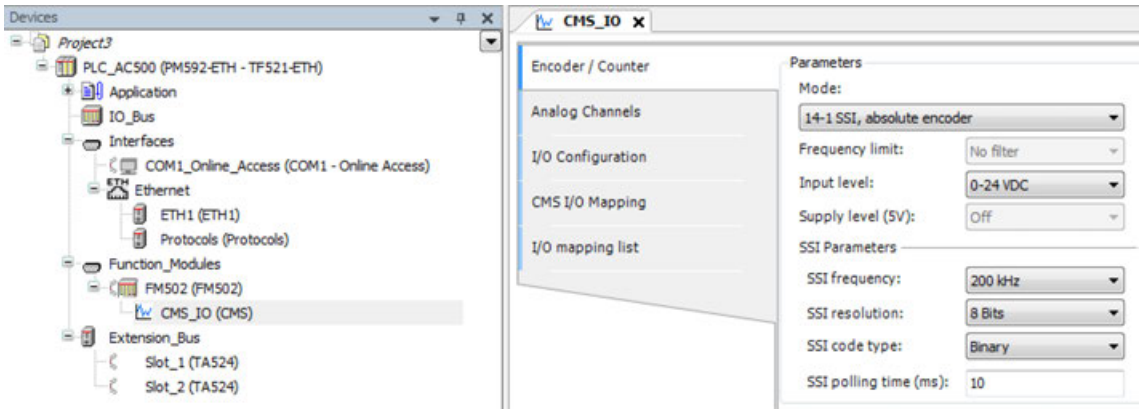


Fig. 1150: SSI parameters

The trace B of the FM502-CMS is switched as output signal (differential). On the rising edge of the signal, the sensor shifts a new value, starting from the most significant bit.

The clock frequency can be specified under *"SSI parameters ➔ SSI frequency"*.

CAUTION!
Risk of malfunctions!
The clock frequency is only an approximately value.

Do not use the clock frequency for any other purposes, e.g. time measurements.

SSI polling time definition

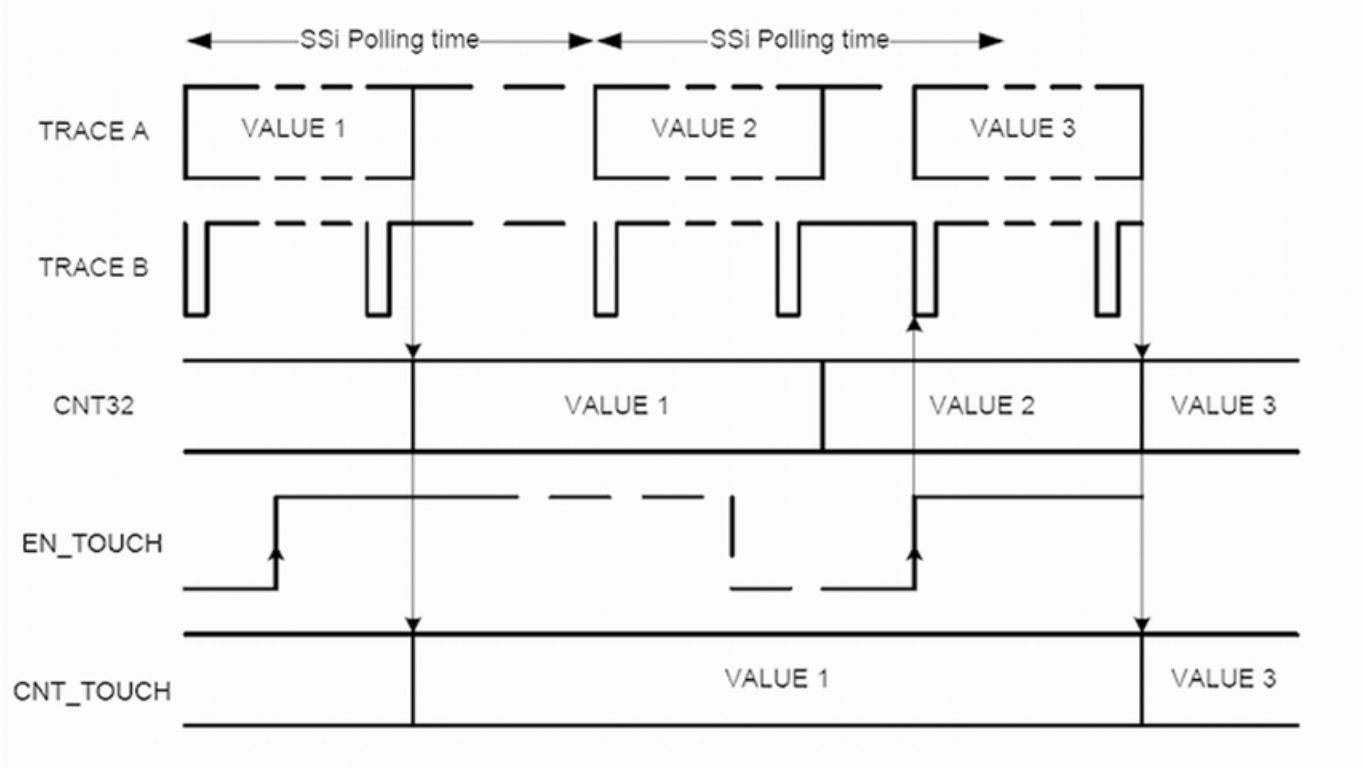
The complete read sequence is launched regularly by the Function Module. The interval between each sequence can be configured under *"SSI parameters ➔ SSI polling time"*.

SSI and touch/ catch operation

Touch operation is valid with SSI sensor. The goal of touch operation is to synchronize sensors with the same hardware signal. In the SSI mode the management is different depending on the reading procedure is running or not.

If the reading procedure has already started while the touch signal becomes active, the reading procedure finishes normally and the last read value is stored in the touch register.

If the reading procedure has not started, the encoder is in the interval between 2 measurements. The reading procedure is started one time more and the result of the last reading is stored in the touch register.

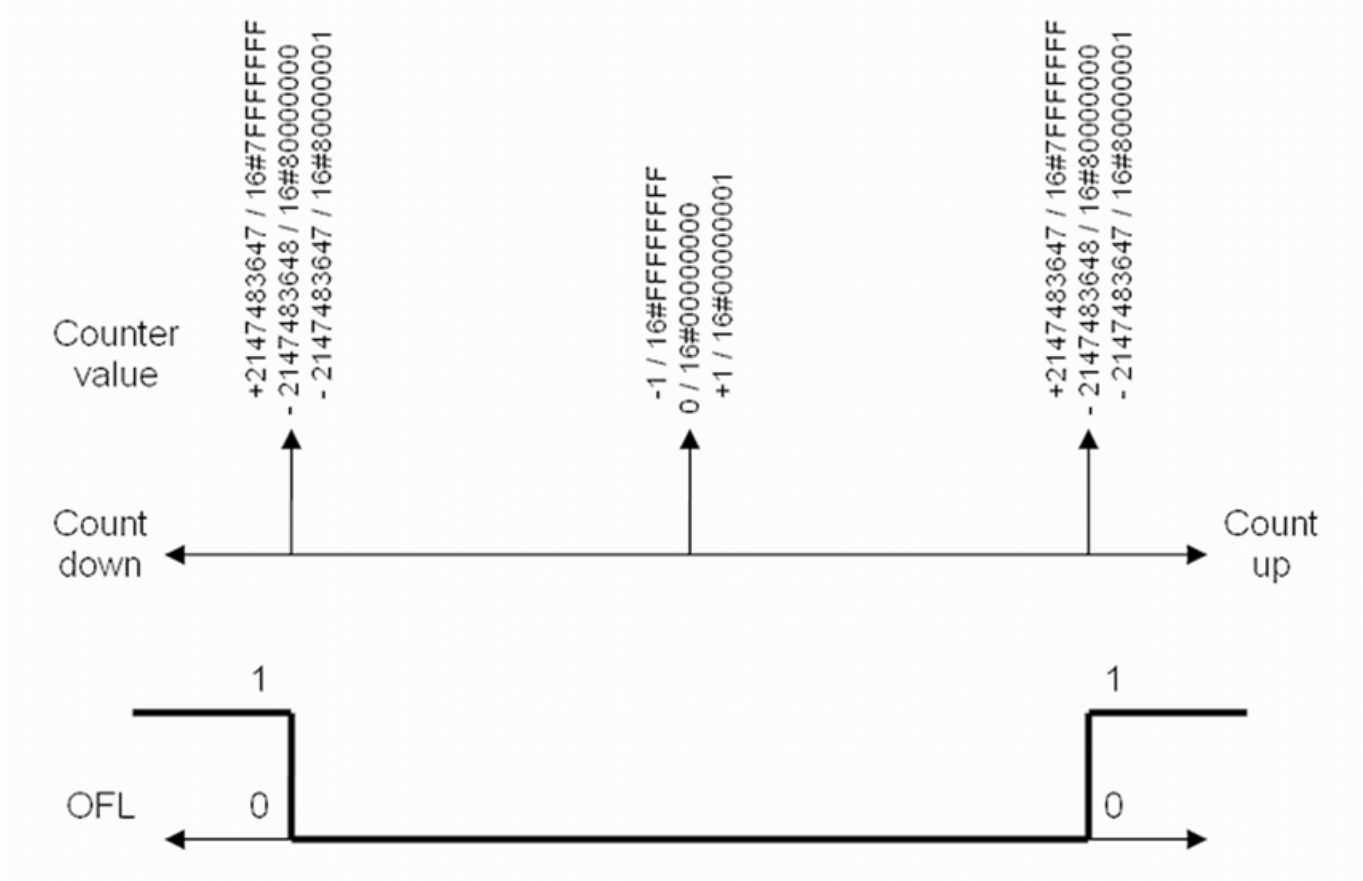


Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the current counter value CNT32 (ACT) and to display this value at output CNT_TOUCH.

Overflow operation

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



FM502-CMS used as time frequency meter

The function block CMS_IO_FREQ_SCAN is used to measure times, frequency and rotation speeds on channel Z+ of the Function Module *Chapter 1.5.8.2.1.5 “CMS_IO_FREQ_SCAN” on page 2542*.

Possible operation modes: 15-1 *Table on page 5728*

The Function Module provides one channel (Z+) which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs.

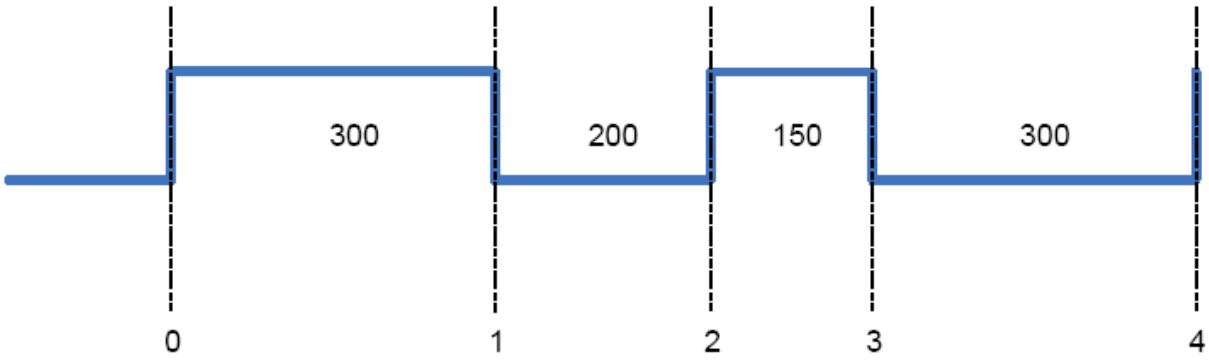


Fig. 1151: Example of timing

Table 723: Values measured according to configuration input parameters and example

EN_0	EN_1	EN_FREQ	Type	1	2	3	4
FALSE	FALSE	TRUE	No measurement	0	0	0	0
FALSE	TRUE	TRUE	Between 2 falling edges		500		450
TRUE	FALSE	TRUE	Between 2 rising edges			350	
TRUE	TRUE	TRUE	Between any 2 edges	300	200	150	300
FALSE	FALSE	FALSE	No measurement	0	0	0	0
FALSE	TRUE	FALSE	Between the rising edge and the subsequent falling edge	300		150	
TRUE	FALSE	FALSE	Between the falling edge and the subsequent rising edge		200		300
TRUE	TRUE	FALSE	Between any 2 edges (start between edge 0 and 1) *)		200		300
TRUE	TRUE	FALSE	Between any 2 edges (start between edge 1 and 2) *)			150	

*) The timing measurement is a single shot process. The function block manages renewal of the measurement as soon as the enable input is valid. Because of timing required to exchange management bits on the bus, it is not possible to provide the time measurement between two adjacent edges. Therefore, the time measured depends on when the measurement is started.

Depending on the input parameters of the function block, the result of the measurement can be read as time in µs, frequency in Hz or speed of rotation in rotation per minute.

FM502-CMS used with synchronized counter/encoder files

Refer to the operation modes and used function blocks ↬ *Table on page 5728*.

Configuration

▷ Configure at least an encoder mode and one analog channel which should be recorded.

⇨ When encoder mode is set, on the **analog channels** tab the check box *Synchronized encoder file* is selected.

The encoder track will only be recorded when the correct encoder function block is used and enabled.

Measurement files

The measurement data will be stored in the WAV file format. One WAV file will be created for each active channel.

Table 724: RIFF header

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	0 (0x00)	bfChunkID	"RIFX"
DWORD	Little	4	4 (0x04)	dwChunkSize	Data length - 8
BYTE[4]	Big	4	8 (0x08)	bfRiffType	"WAVE"

Table 725: Format chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	12 (0x0C)	bfChunkID	"fmt"
DWORD	Little	4	16 (0x10)	dwChunkSize	Data length - 8
INT	Little	2	18 (0x12)	wFormatTag	0x0001 (PCM)
INT	Little	2	20 (0x14)	wChannels	0x0001 (1 ch.)
DWORD	Little	4	24 (0x18)	dwSamples-PerSec	100 Hz - 50.000 kHz
DWORD	Little	4	28 (0x1C)	dwBytes-PerSec	Sample rate * block align
WORD	Little	2	32 (0x1E)	wBlockAlign	4 byte
WORD	Little	2	34 (0x20)	wBitsPer-Sample	32 bit

Table 726: Data chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	36 (0x24)	bfChunkID	"data"
DWORD	Little	4	40 (0x28)	dwChunkSize	Data length - 8
BYTE[]	Big	Undefined	44 (0x2C)	bfData	Measurement data

Table 727: Label chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	44+sz(bfData)	bfChunkID	“labl”
DWORD	Little	4	48+sz(bfData)	dwChunkSize	Data length -8
DINT	Little	4	52+sz(bfData)	dwIdentifier	Identifier
BYTE[256]	Little	255	56+sz(bfData)	bfText	„Label Text“

The WAV files will be stored in an uncompressed ZIP file at the destination path of CMS_IO_MEASMNT_CTRL. The name of the WAV files is given by the Function Module and is corresponds directly with the analog channel and encoder configuration. For every encoder synchronized shannel with different sample rate, there will be a new encoder track which will be used for synchronization.

Example

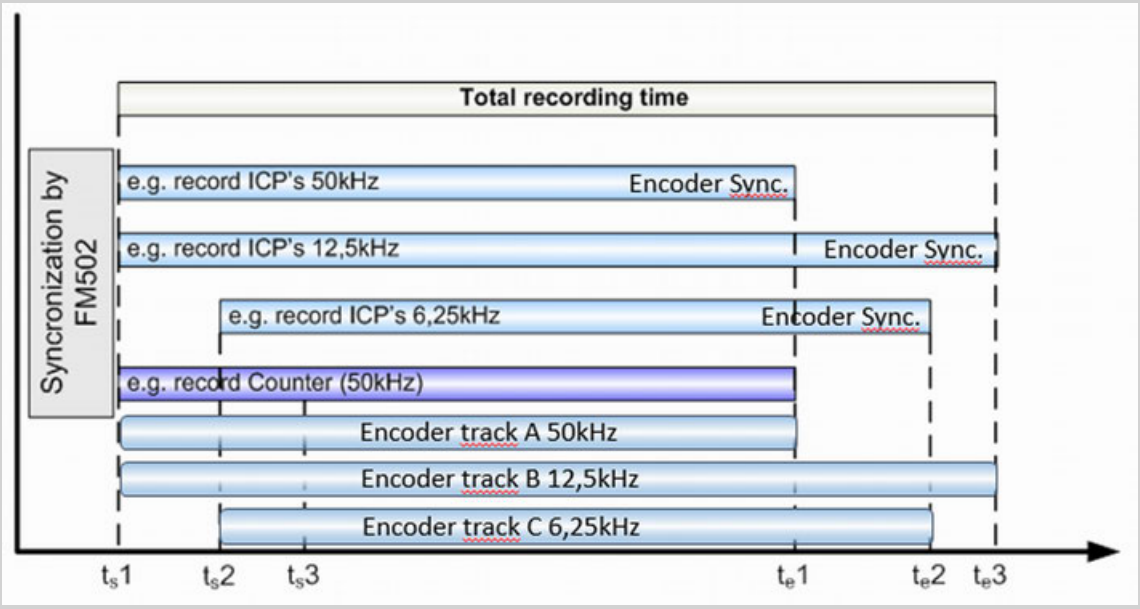
Encoder is configured as:

Channel 0: Sample rate 50 kHz, synchronized encoder file

Channel 1: Sample rate 12,5 kHz, synchronized encoder file

Channel 2: Sample rate 6,25 kHz, synchronized encoder file

Result: The ZIP file contains the WAV files: CH00_ENA.wav, CH01_ENB.wav, CH02_ENC.wav, Enc_A.wav, Enc_B.wav, Enc_C.wav



Example

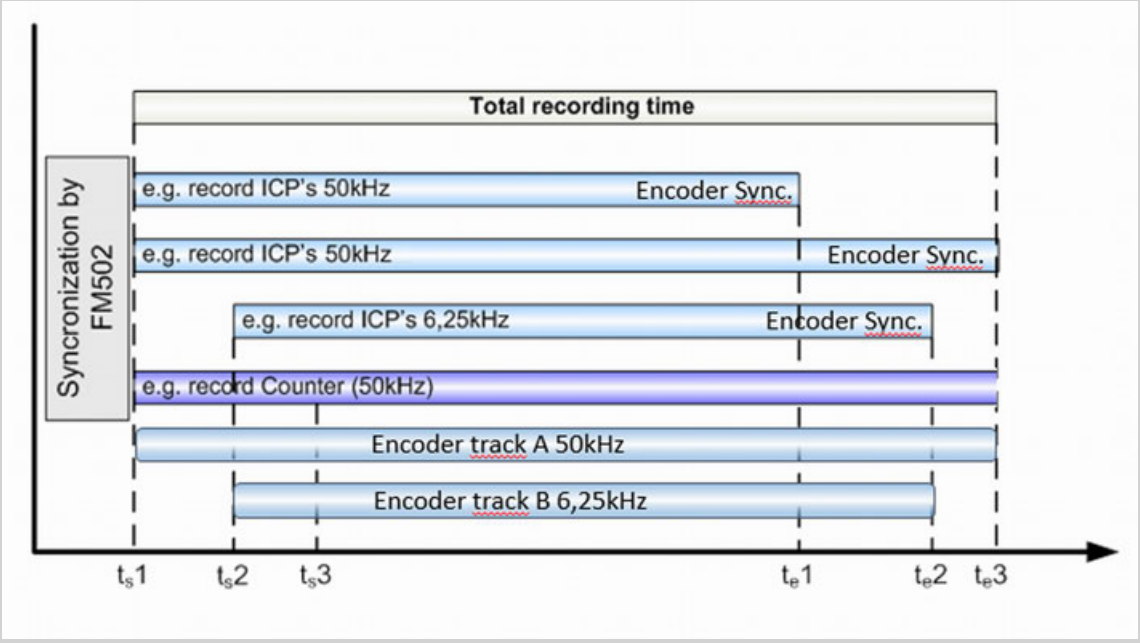
Encoder is configured as:

Channel 0: Sample rate 50 kHz, synchronized encoder file

Channel 1: Sample rate 50 kHz, synchronized encoder file

Channel 2: Sample rate 6,25 kHz, synchronized encoder file

Result: The ZIP file contains the WAV files: CH00_ENA.wav, CH01_ENA.wav, CH02_ENB.wav, Enc_A.wav, Enc_B.wav



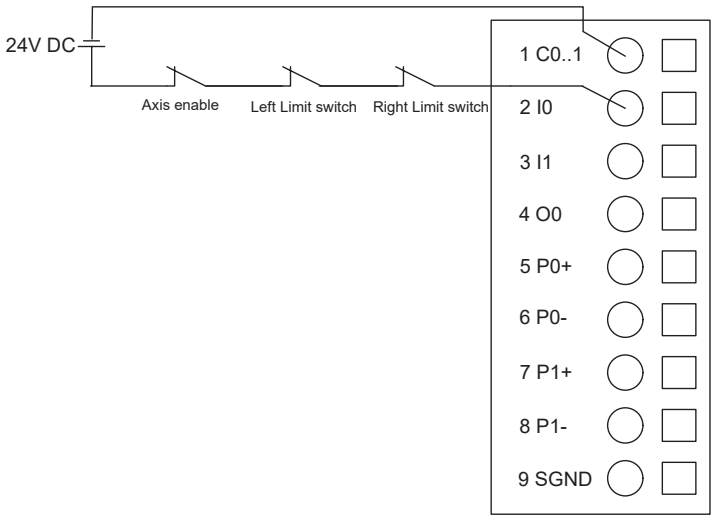
The encoder/counter value at the output of the function block will reset when configuration data of the Function Module will be written in CODESYS.

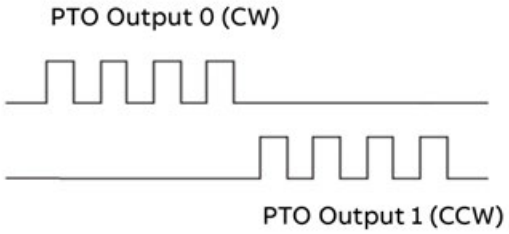
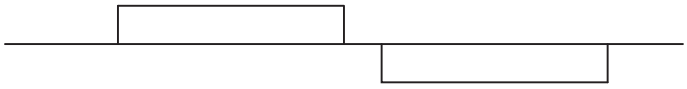
1.6.4.4.4 FM562 module

Information on configuration in Automation Builder: *Chapter 1.6.5.2.8.4 "FM562 module" on page 6056*

Functionality of the FM562 Module

Basic features of the PTO module FM562

Digital inputs	4 inputs (2 inputs per axis) ● Input 0, Input 2: Enable input used for limit switches and enable signal (3 signals in 1 chain)  ● Input 1, Input 3: Stop/Registration input used as interrupt input ● Stop: emergency stop and give the actual position information, the deceleration value can be set via the function block which is provided by the library. ● Registration: give the actual position information, number can be viewed via the function block which is provided by the library, the registration input can be used for homing.

PTO outputs	PTO output mode can be configured by parameters. ● 2-axis control in 1 module ● PTO output type: RS-422 differential output ● PTO frequency: 0..250 kHz ● Configurable PTO output mode: Pulse + Direction / CW + CCW ● CW/CCW (clockwise/counterclockwise): pulse string is set out on 1 of the 2 differential channels according to motor rotation direction. The output channel is defined by a function block. Pulse+ / Pulse- (CW/CCW)  ● Pulse/direction: 1 differential channel for pulse string and 1 differential channel to set direction of movement Pulse + Direction (Pulse / Direction) PTO Output 0 (Pulse)  PTO Output 1 (Direction) 
Power supply for encoders	● Input voltage range: 18..31.2 V DC ● Reverse polarity protection: withstand 10 s ● Surge voltage: 35 V for 0.5 s ● Allowed interruptions of power supply :interruption < 10 ms, time between 2 interruptions > 1 s ● Isolation: 1.2/50 μs Impulse peak 500 V
LED displays	● 1 green LED for PWR (I/O Bus power) ● 1 red LED for ERR ● 1 yellow indicator for each digital input/output channel ● 1 yellow LED indicator for each pulse output channel

The function blocks used with FM562

The following function blocks are used for realizing positioning and speed control with PTO FM562.

The function blocks mentioned below are part of PS552-MC-E Libraries. These libraries are not part of Automation Builder and must be ordered separately.

Classification	Name	Description
PTO Function blocks	PTO_FM562_ACCESS	This function block is used to adapt specific PLCopen blocks to FM562 module and also adds other specialized function for FM562.
PLCopen Function blocks	MC_Power	This function block controls the power stage (on or off).
	MC_Stop	This function block aborts any ongoing function block execution.
	MC_Reset	This function block resets the current axis position to 0.
	MC_MoveAbsolute	This function block commands a controlled motion to a specified absolute position
	MC_MoveRelative	This function block commands a controlled motion of a specified distance relative to the actual position at the time of the execution.
	MCA_MoveVelocityContinuous	This function block commands a never ending controlled motion at a specified velocity and changes velocity on fly.

Latency of function block *PTO_FM562_ACCESS*

When the command for a movement is given, the function block transmits the parameters in several I/O bus cycles. Then the movement is started. The maximum reaction time can be calculated with the following formula:

Max. reaction time = 3 x (T + 3.32 ms)

T = Application program cycle time in ms (set in the configuration)

Example:

- Application program cycle time = 1ms.
  Max. reaction time = 3 x (1 ms + 3.32 ms) = 13 ms
- Application program cycle time = 10 ms.
  Max. reaction time = 3 x (10 ms + 3.32 ms) = 40 ms

Special features

FM562 provides a square wave output for a specified number of pulses and a specified cycle time. It can be programmed to produce either 1 train of pulses or a pulse profile consisting of multiple trains of pulses. For example, a pulse profile can be used to control a stepper motor through a simple ramp up, run, and ramp down sequence or more complicated sequences. The control positioning is achieved according to an open loop mode, meaning without the need for feedback information on the real position. The position loop is integrated in the servo-drive.

Parameter configuration

No.	Operation Object	Function	Description
1	Input 0	No Function	No Function
		Axis enable/Limit switch	When the input is high (24 V), axis can output pulses; when the input is low (0 V), axis cannot output pulses.
2	Input 1	No Function	No Function
		Stop	Stop the pulse output when the signal is high
		Registration	Triggered by the rising edge of the signal, the actual position information can be viewed via the function block which is provided by the library, the registration input can be used for homing.
3	Output	CW/CCW	Pulse string is set out on 1 of the 2 differential channels according to motor rotation direction
		Pulse/Direction	1 differential channel for pulse string and 1 differential channel to set direction of movement

1.6.4.5 System technology for AC31 adapter I/O modules

1.6.4.5.1 KP9x devices

Interrupt handler for KP9x devices

In case a KP9X coupler requires handling of incoming interrupts, an interrupt handler has to be configured.

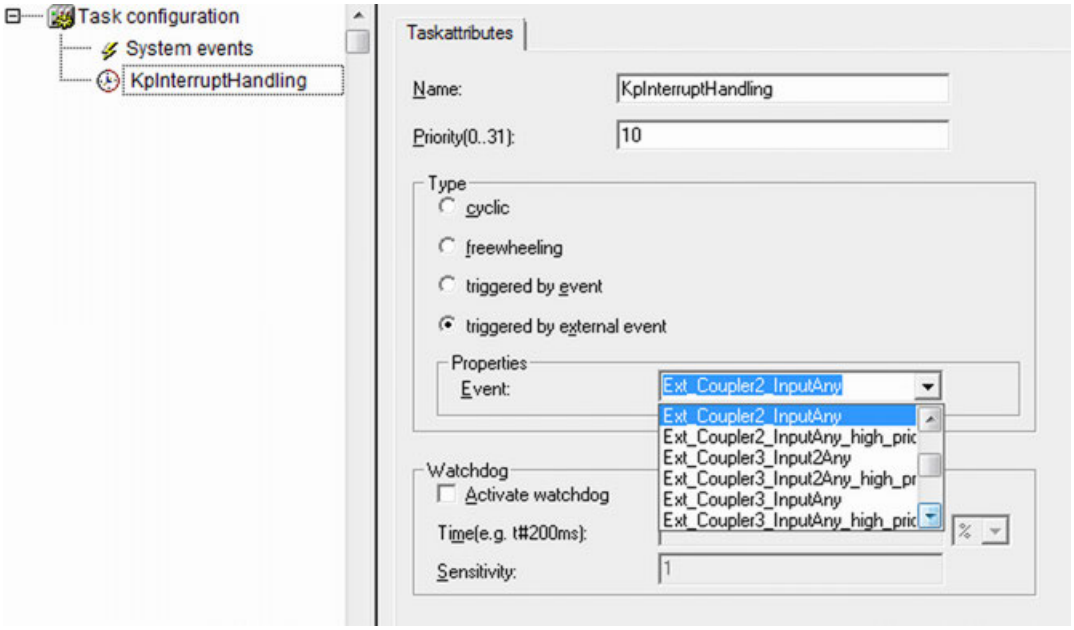
For this purpose, a new task has to be added in the task configuration of the Automation Builder. Enter the task name, set the task type to "external event triggered" and specify the event that triggers the task.

For each I/O bus slot, two types of interrupt tasks are available in the "Event" list box:

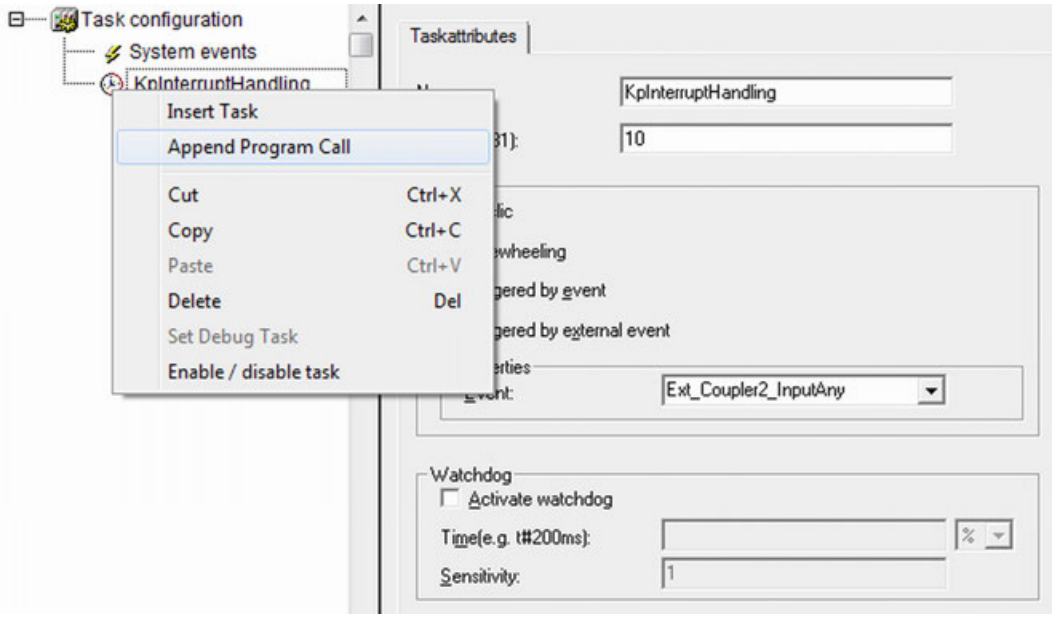
- Ext_CouplerX_InputAny:
  The task is triggered by any interrupt from I/O bus module slot X with the priority specified in the Priority field (0...31).
- Ext_CouplerX_InpuAny_high_prio:
  The task is triggered by any interrupt from I/O bus module slot X with highest priority, i.e. with a priority higher than the max. adjustable "0" and higher than the priority of the communication task. In this case, the priority (0...31) specified in the Priority field is without any significance.

NOTICE!
If the interrupt task is started with high priority (Ext_CouplerX_InpuAny_high_prio), the program execution time must not be longer than approx. 400 µs. Otherwise online access is no longer possible.

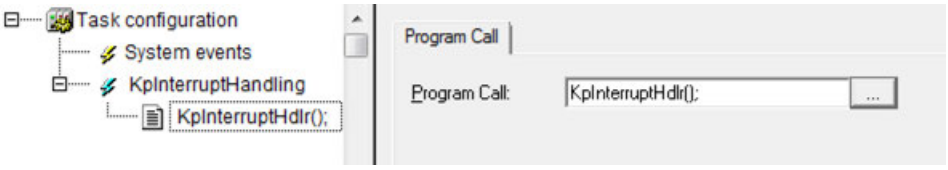
In the example below, the task is named KPInterruptHdlr_2, meaning that it is an interrupt from I/O bus located at slot 2 (where AC31 I/O bus is located for AC31 adapter systems). The task type is "external event triggered" and the event to trigger the task is "Ext_Coupler2_InputAny".



Like for all other tasks, a program call has to be assigned to the task.



In the example, the program KpInterruptHdlr() shall be started with any interrupt from Communication Module slot 1.



1.6.5 Configuration in Automation Builder for AC500 V2 products

1.6.5.1 General settings

This chapter describes the device configuration of AC500 product family with Automation Builder. Basic information on Automation Builder handling can be found in the *Chapter 1.2 "Getting started" on page 12*.

1.6.5.1.1 Project handling

What is a project?

- A project contains the objects which are necessary to create a controller program ("application"):
  - Pure POUs, for example programs, function blocks, functions, and GVLs.
  - Objects that are also required to be able to run the application on a PLC. For example, task configuration, Library Manager, symbol configuration, device configuration, visualizations, and external files.
- In a project, you can program multiple applications and connect multiple controller devices.
- CODESYS manages device-specific and application-specific POUs in the *"Devices"* view ("device tree") and project-wide POUs in the *"POUs"* view.
- For the creation of projects, there are templates that already contain certain objects.
- Basic configurations and information for the project are defined in the *"Project Settings"* and *"Project Information"*. For example:
  - Compiler settings
  - User management
  - Author
  - Data about the project file

  There are settings for the version compatibility of the project in the configuration dialogs in the *"Project Environment"*.

- You save a project as a file in the file system. As an option, you can pack it together with project-relevant files and information into a project archive. It is also possible to save files in a source code management system such as SVN.
- Each project contains the information about the CODESYS version with which it was created. When you open it in another version, CODESYS will notify you about possible or necessary updates regarding file format, library versions, etc.
- You can compare, import/export projects, and create documentation for them.
- You can protect a project from being changed, or even completely protect it from being read. By using user management, you can selectively control the access to the project and even to individual objects in the project.

Creating a new project

1. Select *"File ➔ New Project"*.

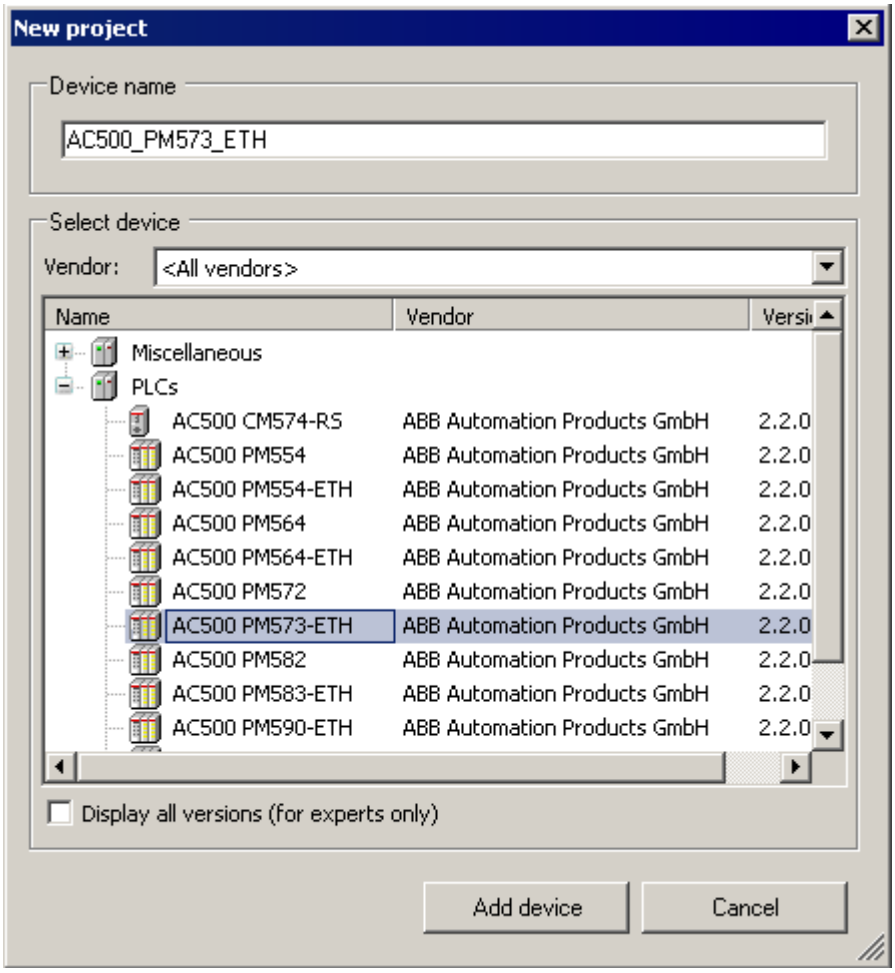
   If the used Automation Builder version is not the latest version, an information is displayed.

   - Select *"Change to newest installed version"* to create a project with the latest installed version of Automation Builder.
   - Select *"Continue to work with version: XXX"* to create a project in the current software version.

2. Select *"AC500 project"*, enter a project name and specify the storage location for the new project.

   With *"Empty project"* a project without a PLC is created.

3. Select the device type for the new project and click *[Add device]*.



⇨ A new project is created and can be configured.

Opening an existing project

NOTICE!

Risk of damaging Automation Builder projects!

Projects created with Automation Builder are incompatible with CODESYS V2.3.9.x. Do not open projects with CODESYS V2.3.9.x as this can cause corrupted Automation Builder projects.

Automation Builder performs an integrity check for the PLC configuration before generating the configuration.

Opening a project

1. Select *"File ➔ Open Project"*.
   ⇨ The *"Open Project"* dialog appears.
2. Select a previously saved project from the file system.
   ⇨ Automation Builder switches to the version of the project and opens the project.

Exporting and importing a project

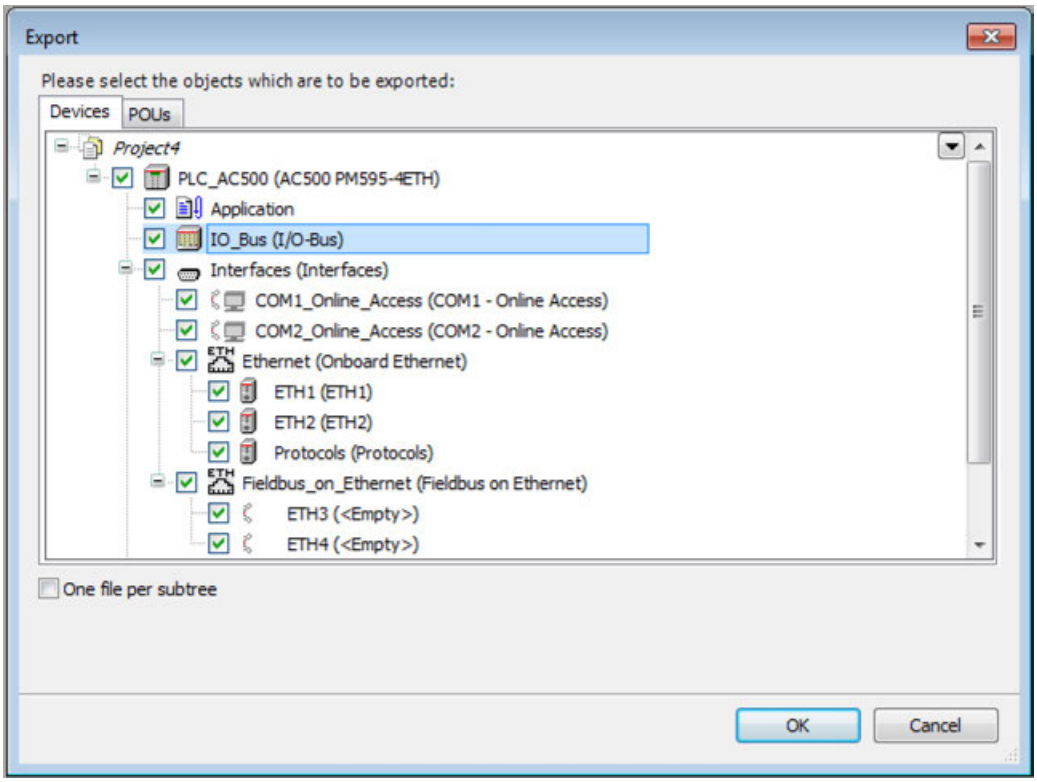
Configuration of a complete PLC or of single devices can be reused within the same project by copy-and-paste the desired nodes in the device tree.

In order to reuse a PLC configuration cross-over projects, the project configuration can be exported and imported afterwards into another project.

An exported project configuration can only be imported to a project with the same Automation Builder version. If the versions are not the same, the import fails with an error message.

Automation Builder performs an integrity check for the PLC configuration before generating the configuration.

Project export From the menu, select *"Project ➔ Export ➔ Project"*. Select the objects to be exported. The configuration of the selected items will be added to an export file (*.export).



"One file per subtree": If this option is activated, all objects belonging to the same subtree will be exported into the same export file, otherwise a separate file will be created for each particular object.

Project import

For importing a project a basic and an advanced function is available.

Basic project import: Users with a basic or a standard Automation Builder license can perform a **basic project import**. Command: *"Project ➔ Import ➔ Project"*.

> *A previously exported project configuration is imported into the current project. With this, the current project configuration is overwritten.*
>
> *In order to supplement the current project with the project configuration of a previously exported project, use the compare function. Command: "Project ➔ Compare".*
>
> *Chapter 1.6.5.1.1.6 "Comparing projects" on page 5765*

Advanced project import: Users with a premium Automation Builder license can perform an **advanced project import**. Command: *"Project ➔ Import ➔ Project with compare"*. This command allows to compare two projects, to check on differences and to adapt single parts of the project configuration easily.

Basic project import

1. From the menu, select *"Project ➔ Import ➔ Project"*.

> *A previously exported project configuration is imported into the current project. With this, the current project configuration is overwritten.*
>
> *In order to supplement the current project with the project configuration of a previously exported project, use the compare function. Command: "Project ➔ Compare".*
>
> *Chapter 1.6.5.1.1.6 "Comparing projects" on page 5765*

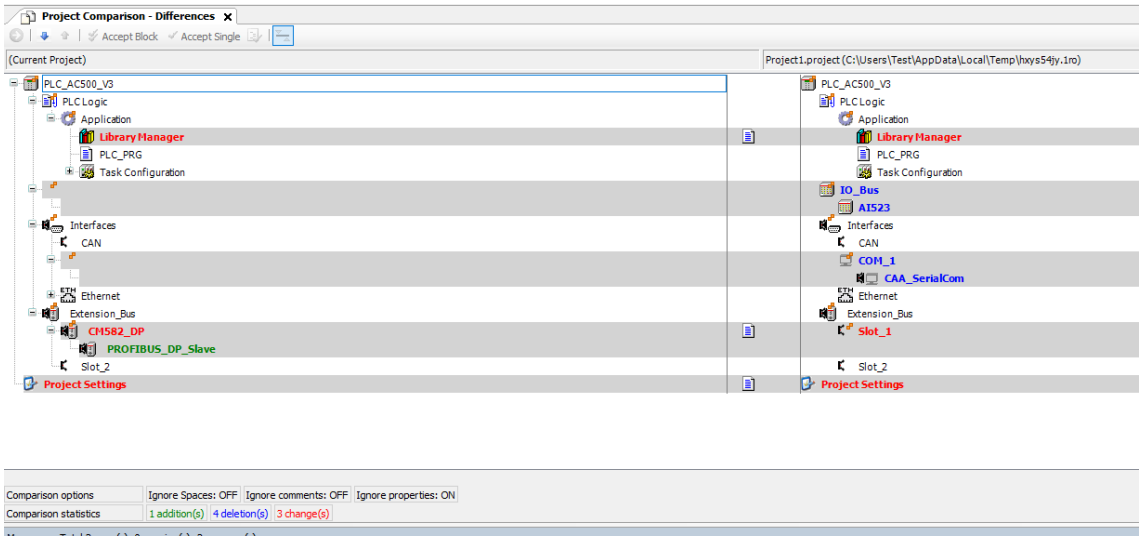
2. Select the export file from the file system and click *[Open]* to import the project configuration.

> *An exported project configuration can only be imported to a project with the same Automation Builder version. If the versions are not the same, the import fails with an error message.*

Advanced project import

Perform an advanced project import in order to compare two projects, to check on differences and to adapt single parts of a previously exported project configuration easily.

1. From the menu, select *"Project ➔ Import ➔ Project with compare"*.
2. Select the export file from the file system and click *[Open]* to import the project configuration.
   ⇨ The project import is started.
3. Once the project file is imported, a compare view is displayed. The left pane represents the current project, the right pane represents the imported project.



⇨ Differences between the current project and the imported project are highlighted in red color.

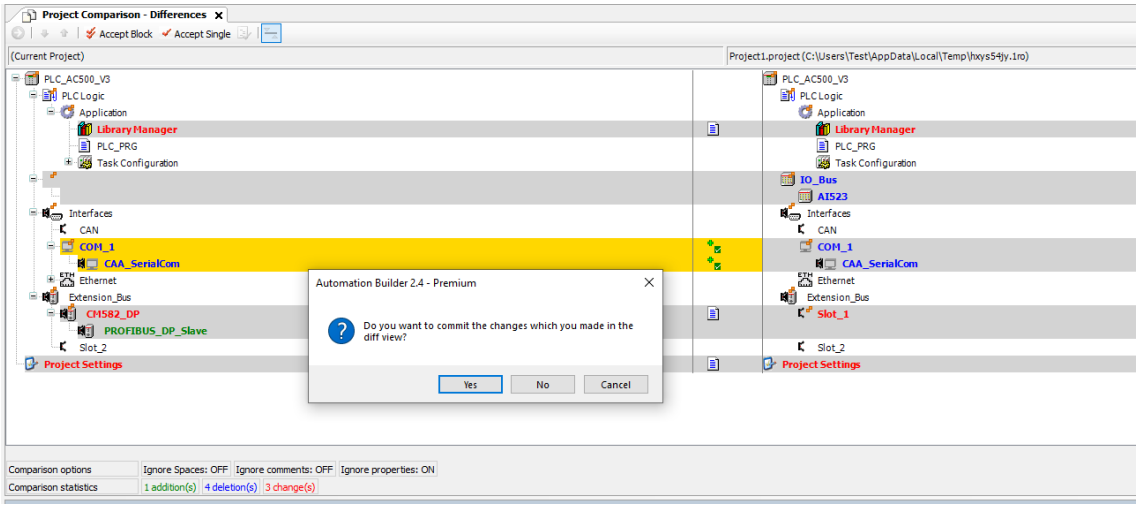
⇨ Additional modules in the current project that are not available in the imported project are highlighted in green color.

⇨ Additional modules in the the imported project or deleted modules in the current project are highlighted in blue color.

⇨ A summary of all differences within the projects is given in the *"Comparison statistics"* under the device tree.

4. Every highlighted item of both projects can be handled individually and can either be transferred to the current project or skipped.

   - *[Accept Block]*: All items of the selected node are transferred to the current project with one click. Use this function for example to copy all nodes of a PLC configuration from the imported project to the current project (select *"I/O_Bus"* node).
   - *[Accept Single]*: Only a single item from a node is transferred to the current project. Use this function for example to copy certain I/O modules from the imported project to the current project.

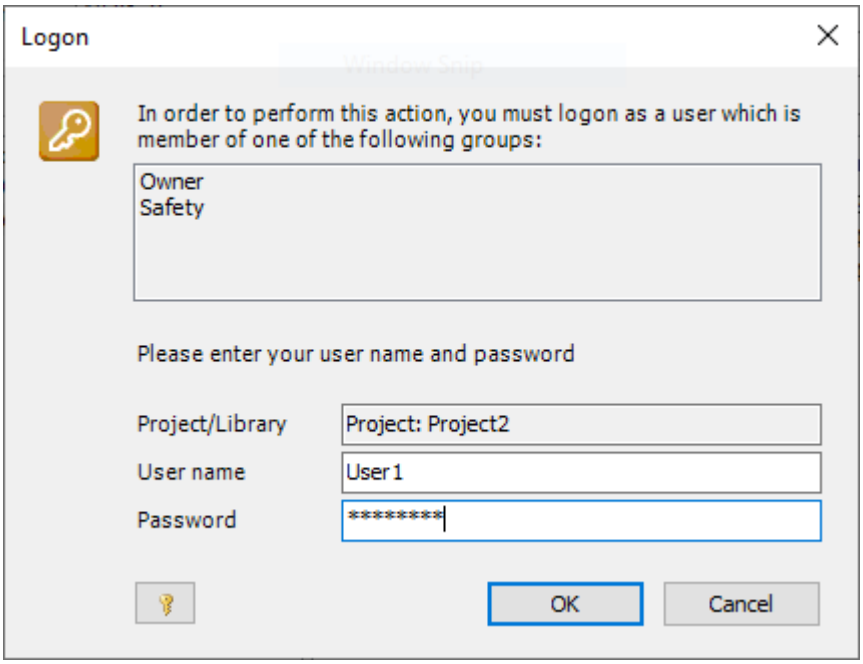


   ⇨ All accepted items are highlighted in the current project in yellow color.

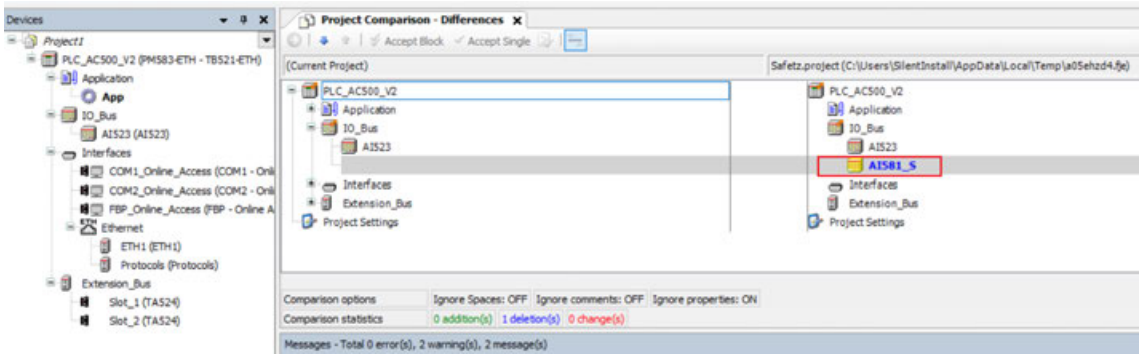
   ⇨ To undo a selection, again, click *[Accept Block]* or *[Accept Single]*.

   ⇨ To accept all changes on the current project, close the *"Project Comparison - Differences"* tab and confirm the prompted dialog.

5. If in the import project the PLC contains an AC500-S safety module, a security check is performed which requires user authentication:



6. After a successful user authentication the AC500-S safety modules are added to the compare view and can be imported to the current project.



Upgrading/ updating a project to a new Automation Builder version or profile

When upgrading or updating Automation Builder a previously configured project can be converted in order to be used in a new Automation Builder version or with a new Automation Builder profile.

Definition:

*Automation Builder upgrade: changing over to a major Automation Builder version (e.g. from version 2.**3**.1 to version 2.**4**.1).*

*Automation Builder update: changing over to a minor Automation Builder version (e.g. from version 2.4.**0** to version 2.4.**1**).*

Further, a project that has been configured for an AC500 V2 PLC can be converted to a project for an AC500 V3 PLC.

Chapter 1.6.5.6 “Converting an AC500 V2 project to an AC500 V3 project” on page 6330

Before the upgrade/update

Project archive

*Create a project archive before updating Automation Builder. Project archives contain all project data, including data that is not stored with a *.project file, e.g. device description files for third party devices.*

Chapter 1.6.5.1.1.7.1 “Creation of an archive ” on page 5768

RobotStudio station

RobotStudio integration has been discontinued as of Automation Builder 2.1.0. It is recommended to externally store the link to the RobotStudio station and to remove the RobotStudio station object prior to the upgrade.

Automation Builder profile

To use the Automation Builder profile of an older project, the old profile must have been installed. The installation of older Automation Builder profiles can be activated in the device dialog during the upgrade process.

Upgrading/ Updating a project

1. With opening a project Automation Builder automatically detects the project version. In case of an outdated project version a dialog is prompted.

   ⇨ If the update is confirmed, the project is automatically updated to the latest Automation Builder version.

   Automation Builder updates the complete project (complete device tree) to the latest version. Success messages, warnings and errors are described in the section *"All messages"*.

   ⇨ If the update is declined, the project is closed unchanged.

   In order to initiate a project update or upgrade later on, select *"Project ➔ Update Project"*.

   ⇨

   To keep an older project, it must be opened with the same Control Builder Plus/ Automation Builder version the project has been created. For this, the appropriate Control Builder Plus/ Automation Builder profil must be selected.

   In this mode, new Automation Builder features cannot be used.

   It is not possible to downgrade a project to an earlier Automation Builder version.

   Automation Builder performs an integrity check for the PLC configuration before generating the configuration.

AC500 V2 libraries

2. When upgrading Automation Builder, new available AC500 V2 system libraries are installed automatically. In difference to AC500 V3 libraries the AC500 V2 libraries are not versioned. Hence, after an Automation Builder update login to a PLC might only be possible after a rebuild and with an online change. This might be required although the application has not been changed and the previous version profile is still in use.

   To avoid this, add the AC500 V2 libraries to the Automation Builder project. The procedure on how to add a AC500 V2 (system) library to a project is described exemplarily.

Migrate third party devices

3. During the project upgrade, an option for migration of third party devices can be selected. If this option was not selected during the upgrade procedure, migration can be initiated manually after an Automation Builder upgrade in order to migrate all third party devices to the project.

   ↳ *Chapter 1.6.5.1.10 "Migration of third party devices" on page 5807*

4. Exception, for the CANopen device CM598-CN:

> *Usually, when upgrading Automation Builder or an existing project, new AC500 V2 system libraries are installed automatically and older library versions are removed.*
>
> *As an exception, for the CANopen device CM598-CN both library versions are available in the Library Manager due to compatibility reasons. However, coexistence of a new library version and an older library version is not possible. In order to avoid compile errors remove the older library version.*

Login

5. After the Automation Builder upgrade login to the PLC from Automation Builder: right-click *"Application ➔ App"* and select *"Login [PLC_AC500_V2]"*.

   ⇨ The firmware on the devices is upgraded.

> *Depending on the currently installed firmware versions, a login from CODESYS V2.3 might be impossible prior to the firmware update.*

Updating PLC devices

To update all devices of a PLC project, right-click the PLC node and select *"Update objects"*. In the dialog enable *"update subtree"* option to update all sub-objects. Otherwise only the processor module object is updated.

To update a specific device only, the command *"Update objects"* can be executed individually at the specific node.

I/O mapping export and import

Export I/O mapping

To exchange information on I/O mapping only, data can be exported as .csv file. This allows maintenance of I/O data outside Automation Builder, e.g. in MS Excel.

Right-click the *"Processor Module"* node or *"I/O_Bus"* node in the device tree and select *"Export -> IO mapping"*. To export the I/O Mapping for the complete project, e.g. with more than one configured processor modules, I/O data of the complete project can be exported *"Project -> Export -> I/O mapping"*.

Import I/O mapping

A previously exported .csv file can be imported to the project: *"Project -> Import -> I/O mapping"*.

Comparing projects

You can compare the currently open project with another project – a reference project. The differences in contents, properties, or access rights are detected and shown in a comparison view.

Clicking *"Project ➔ Compare"* opens the *"Project Compare"* dialog for you to configure and run the comparison. Then the result is shown in the comparison view *"Project Compare - Differences"* where the objects are aligned in a tree structure. Objects that indicate differences from the respective reference object are identified by colors and symbols. This is how you detect whether or not the contents, properties, or access rights are different.

For differences in the contents, you can also open the detailed compare view *"Project Compare - <object name> Differences"* in order to zoom into the object. In the detailed compare view, the contents of the object and reference object are displayed or their source code aligned. The detected differences are marked. Previously opened views are not closed. In this way, you can have any number of comparison views open and read them, in addition to the project compare view.

You can accept the detected differences from the reference project into the current project. This is possible only from the reference project into the open project. To do this, you activate differences (for example in the code) that should be accepted in the current project with the commands , , or in the active comparison view for accepting. These positions are highlighted in yellow. Make sure that any other open compare views are inactive (write-protected, read-only). therefore, you can activate differences to be accepted in exactly one comparison view only. When exiting the active compare view, if you confirm that the differences that are activated for acceptance are actually accepted into the current project, then the current project is modified.

In order to exit the project comparison completely, close the project compare view.

Creating a comparison view

Requirement: You have made changes in your current project and wish, for example, to compare it with the last-saved version. In the meantime, for example, you have added further POUs, removed a POU, changed single lines of code or the object properties in function blocks.

1. Select the command *"Project ➔ Compare"*.
   ⇨ The *"Project Comparison"* dialog box opens.
2. Enter the path to the reference project, for example the path to the last-saved version of your current project.
3. Leave the activation of the comparison option *"Ignore Spaces"* as it is.
4. Click on *"OK"*.
   ⇨ The comparison view opens. Title: *" Project Comparison – Differences"*. The Device trees of the current project and the reference project are displayed alongside each other and the changed objects are marked in color.
5. Select an object marked in blue in the tree of the reference project (right). The current project no longer contains this object.
   Click on *"Accept Single"*
   ⇨ The object is added to the tree of the current project (left). The line has a yellow background. appears in the middle column.
6. Select an object marked in green in the tree of the current project (left). The reference project does not contain this object.
   Click on *"Accept Single"*
   ⇨ The object is removed again from the tree of the current project (left). The line has a yellow background. appears in the middle column.
7. If changes are detected in the content of an object that is contained in both the current project and the reference project, this is indicated by red lettering. You can then switch to the detailed comparison view for the object by double-clicking on the object.
8. Close the comparison view and answer the query whether the changes made are to be saved with *"Yes"*.
   ⇨ The changes become effective in the project.

Opening the detailed compare view

Requirement: For example, a user modified the code in a POU of the current project. You have performed the project comparison by clicking *"Project ➔ Compare"*. The project compare view shows this POU highlighted in red in the aligned in the project tree.

1. Double-click the line of the aligned POU versions.
   ⇨ The compare view switches to the detailed compare view of the POU. The modified code lines are highlighted in gray and written in red.
2. Click .
   ⇨ Code lines with changes (red) are extended by two lines: an line with insert (left, green) and a line with delete (right, blue).
3. Click again.
   ⇨ The code line is marked again as modified.
4. Move the mouse pointer to the code line marked as modified and click *"Accept Single"*.
   ⇨ The code line from the reference project is activated for acceptance into the current project.
5. Click .
   ⇨ The project compare view opens for the entire project. It is write-protected (read-only) to prevent you from activating differences for acceptance. The link highlighted in yellow above the tree view also indicates this.
6. Click the link: *"Project compare view is read only because there are uncommitted changes in another view. Click here to switch to the modified view."*
   ⇨ The detailed compare view opens again. The unconfirmed changes are highlighted in yellow.
7. Click ✕ in the tab of the view and confirm that the changes should be saved.
   ⇨ The detail project view is closed and the POU is overwritten. Now it corresponds to the POU of the reference project. The project view is active again so that you can continue working with project compare.

> *If you do not click the link, but click ✕ instead to close the editor of the project compare view, then you will also confirm the acceptance of changes into the current project. The detail changes are accepted and then the project compare is closed completely.*

See also

- *Chapter 1.4.1.1.4 "Comparing projects" on page 146*
- *Chapter 1.6.5.1.1.6.1 "Creating a comparison view" on page 5766*

Project archive

Automation Builder supports the creation and the import of project archive files. Archive files contain all relevant project data including the PLC configuration, the project files of the CODESYS and all device descriptions. This allows exchanging Automation Builder projects without taking care of the target environment.

Creation of an archive

☑ The following steps describe the creation archive file from an Automation Builder project:

1. Select *“File ➔ Project Archive ➔ Save/Send Archive”*.
2. Select the information which should be included in the archive file from the list box.

Section/Control	Parameter	Description
Information selection list box	Options	Not supported
	Referenced devices	The referenced devices can be selected by expanding the "Referenced devices" item of the list box. It is strongly recommended to include all devices in the project archive to maintain consistency.
Additional files	-	Not supported
Comment	-	Opening a control window which allows the input of a comment to the project archive.
Save	-	Opening a dialog window to determine the path and the file name of the project archive and storing it to the file system.
Send	-	Not supported
Cancel	-	Canceling the operation and closing the dialog window.

With *[Comment]* additional information can be added to the project archive, for example to add a brief description or some information concerning the project.

3. Proceed with *[Save...]*.

It is strongly recommended to keep the default settings.

Section “Options” of the list box is not support. Do not enable this option.

Extraction of an archive

The currently loaded project will be closed automatically when extracting the selected project archive. It is recommended to open a new instance of Automation Builder before starting the extraction process.

☑ The following steps describe the extraction of an archive file and the import to Automation Builder.

1. Select *"File ➔ Project Archive ➔ Extract Archive"*.
2. Select the desired project file and click *[Open]*.

Section/ Control	Parameter	Description
Locations	Extract into the same folder where the archive is located	The project archive will be extracted to the same path where the archive is located.
	Extract into the following folder	Path to which the project archive should be extracted.
	Button ...	Opening a folder selection dialog which allows selecting the desired path.
Contents	Items	Select the items which should be extracted.
	Comment	Displaying comments included inside the Project archive file.
	Extract	Triggering the extraction process. Automation Builder extracts the archive and creates a project from out the archive. After creating the project Automation Builder checks the version of the project. If the project version and the activated Automation Builder version is not identical the workflow is the same as described in "Opening an Existing Project".
	Cancel	Closing the Extract Project Archive dialog and canceling the extraction process.

1.6.5.1.2 User and access rights management

User and access rights

The 'User Management' provide functions for defining user accounts and configure the access rights within a project. The rights to access project objects via specified actions are assigned only to user groups, not to a single user account. So each user must be member of a group.

User management

Before setting up users and user groups, notice the following: The configuration of users and groups is done in the Project Settings dialog *Chapter 1.6.5.1.2.3 "Project Settings - Users and groups" on page 5772*.

- Automatically there is always a group "Everyone" and by default primarily each defined user or other groups are members of this group. Thus each user account at least automatically is provided with defined default settings. Group "Everyone" cannot be deleted, just renamed, and no members can be removed from this group.
- Also automatically there is always a group "Owner" containing one user "Owner". Users can be added to or removed from this group, but at least one user must remain. This group also cannot be deleted and always has all access rights. Thus it is not possible to make a project unusable by denying the respective rights to all groups. Both group and user "owner" might be renamed.
- When starting the programming system resp. a project, primarily no user is logged on the project. But then the user optionally might log on via a defined user account with user name and password in order to have a special set of access rights.

Notice that each project has its own user management!
So, for example to get a special set of access rights for a library included in a project, the user must separately log on to this library. Also users and groups, set up in different projects, are not identical even if they have identical names.

CAUTION!
The user passwords are stored irreversibly!
If a password gets lost, the respective user account gets unusable. If the "Owner"-password gets lost, the entire project might get unusable!

Access right management

User management in a project is only useful in combination with the access right management. Notice the following:

- In a new project basically all rights are not yet defined explicitly but set to a default value. This default value usually is: "granted".
- In the further run of working on the project each right can be explicitly granted or denied resp. set back to default. The access right management of a project is done in the Permissions dialog *"Permissions" on page 5771*.
- Access rights on objects get "inherited". If an object has a "father" object (example: if an action is assigned to a program object, that is inserted in the structure tree below the program, then the program is the "father" of the action object) , the current rights of the father automatically will become the default settings of the child. Father-child relations of objects concerning the access rights usually correspond with the relations shown in the POUs or Devices tree and are indicated in the Permissions dialog by the syntax "<father object>.<child object>".

Example Action ACT is assigned to POU object PLC_PRG. So in the POUs window ACT is shown in the objects tree indented below PLC_PRG. In the Permissions dialog ACT is represented by "PLC_PRG.ACT" indicating that PLC_PRG is the "father" of ACT. If the "modify" right would be denied explicitly for PLC_PRG and a certain user group, the default value of the "modify" right for ACT automatically also would be "denied".

User management commands

The 'User Management UI' plug-in provides commands for command category 'User Management'.

These are used for:

- Configuration of access rights on the project objects
- Logging on or off to/from the project via a defined user account in order to get the access rights which are associated to this account

The configuration of user accounts and groups is done in the Project Settings subdialog User Management *Chapter 1.6.5.1.2.3 "Project Settings - Users and groups" on page 5772*.

By default the following commands are part of submenu 'User Management' in the 'Project' menu: Logon, Logoff, Permissions.

Logon Symbol: 

This command opens the Logon dialog for logging on to a project or library via a defined user account.

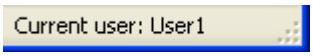
Logging on with a certain user account means to log on with those object access rights which are granted to the group which the user belongs to. The configuration of user accounts and groups is done in the Project Settings subdialog User Management.

To log on select the project or an included library from the selection list in the Project/Library field. Enter User name and Password of a valid user account, noticing that each project or library has an own user and access rights management. Log on with OK.

If already another user is logged on the project, this one will be logged out automatically by the new log-on action.

When you are logged on to a project or library and try to perform an action for which you have no right, automatically a Logon dialog will be opened, giving the possibility to log on with another user account provided with the appropriate rights.

The status bar always displays which user currently is logged on the project.



Logoff

Symbol: 

This command logs off the currently logged on user. If no user had been logged on to the currently opened project or to a referenced library an appropriate message will appear when trying to log off.

If the user currently is logged on to more than one project or referenced library (not necessarily with the same user account) a Logoff dialog will appear when trying to log off.

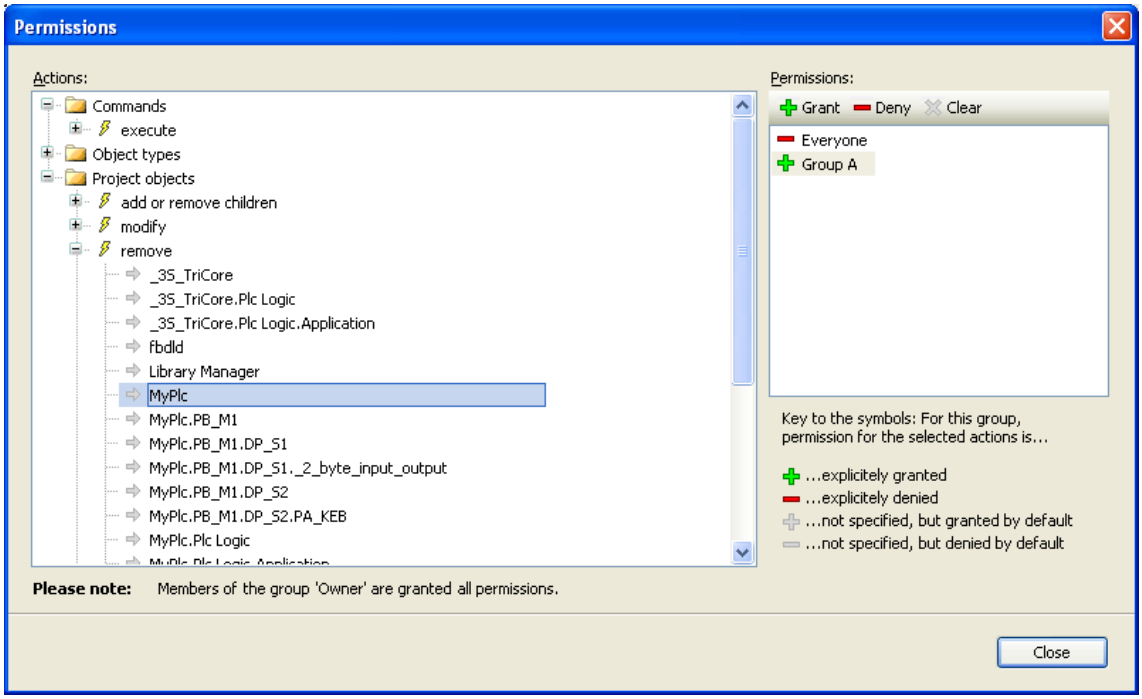
From the Project/Library selection list choose those project/library for which you want to log off. The name of the Current user is displayed just for information.

The status bar always displays which user currently is logged on the project.

Permissions

This command opens the Permissions dialog, where the rights to work on objects or to perform commands in the current project can be configured.

> *Any changes made in this dialog will be applied immediately.*



The Actions window displays all possible rights, that is all actions which might be performed on any object of the current project.

The tree is structured in the following way:

- Top-level see the names of some categories, which have been set up just for the purpose of optical structuring the rights management.
  They are grouping concerning the execution of Commands, the configuration of User accounts and Groups, the creation of Object Types, the viewing, editing, removing and handling of child objects of Project Objects.
- Below each category node there are nodes for the particular actions which might be performed on the command, user account, group, object type or project object. These nodes also only have optical function. Possible Actions:
  - execute (execution of a menu command)
  - create (creating a new object in the current project)
  - add or remove children (adding or removing of "child" objects to an existing object)
  - modify (editing an object in an editor)
  - remove (deleting or cutting an object)
  - view (viewing an object in an editor)

Below each action node find the possible targets, that is project objects, of the respective action.

The Permissions window provides a list of all currently available user groups (except the "Owner" group) and a toolbar for configuring rights to a group.

Select the group and configure it´s permissions.

Left to each group name one of the following icons indicates the currently assigned permission concerning the target which is currently selected in the Actions window:

- : The action(s) for the target(s) currently selected in the Actions window are granted for the selected group.
- : The action(s) for the target(s) currently selected in the Actions window are denied for the selected group.
- : The right to perform the action(s) which are currently selected for the selected target(s) in the Actions window, has not been granted explicitly, but is granted by default, for example because the corresponding right has been granted to the "father" object. (Example: The group has got the right for object "myplc", thus it by default it also has got it for object "myplc.pb_1".) Basically this is the default setting for all rights which not explicitly have been configured.
- : The right to perform the action(s) which are currently selected for the selected target(s) in the Actions window, has not been denied explicitly, but is denied by default, for example in case because the corresponding right has been assigned to the "father" object.

If currently multiple actions are selected in the Actions window, which do not have unique settings referring to the currently selected group, no icon will be displayed.

To configure the rights for a group select the desired action(s) and target in the Actions window and the desired group in the Permissions window. Then use the appropriate button in the toolbar of the Permissions window:

Grant: Explicit granting.

Deny: Explicit denying.

Clear: The currently granted right for the action(s) currently selected in the Actions window will be deleted, that is set back to the default.

Project Settings - Users and groups

The Project Settings dialog in category 'Users and Groups' provides three subdialogs for the user management for the current project: Users, Groups, Settings. For a general description on users and access rights management see help page *Chapter 1.6.5.1.2.1 “User and access rights” on page 5769*.

Users dialog

The currently registered users are listed in a tree structure. The ownerships of each user is displayed and each user is a member of a group by default *Chapter 1.6.5.1.2.1 "User and access rights" on page 5769*.

Define a new user account

1. Click *"Add"* to open the *"Add User"* dialog.
2. Define the user credentials and click OK to set up the new user. If there are incorrect entries (no login name, password mismatch, user already existing) you will get an appropriate error message.

Modify a user account Click *"Edit"* to open the *"Edit User"* dialog. The entry fields are the same as in the *"Add User"* dialog. The password fields however - for security reasons - will show 32 * characters. After having modified the desired entries close the dialog with OK to get applied the new settings.

Remove user accounts Enable the entries to be removed in the Users list and click *"Remove"*. Note that you will get no further inquiry! An error message appears if you try to delete all users from a group. At least one entry must remain.

Groups dialog

Add a group The currently available groups are displayed in a tree structure. A member also might be a group.

1. Click *"Add"* to open the *"Add Group"* dialog.
2. Define a name for the new group and enable all entries (single users or groups) which should be members of the new group.
3. Click OK to set up the new group. If there are incorrect entries (no name defined, group already existing, in Members having selected a group which would cause a "group cycle", you will get an appropriate error message.

Modify a group Click *"Edit"* to open the *"Edit User"* dialog. The entry fields are the same as in the 'Add Group' dialog (see above). After having modified the desired entries close the dialog with OK to get applied the new settings.

Remove groups Enable the entries to be removed in the groups tree and click *"Remove"*. Note that you will get no further inquiry! The members of the deleted groups will remain unmodified. An error message appears if you try to delete the groups "Everyone" and/or "Owner".

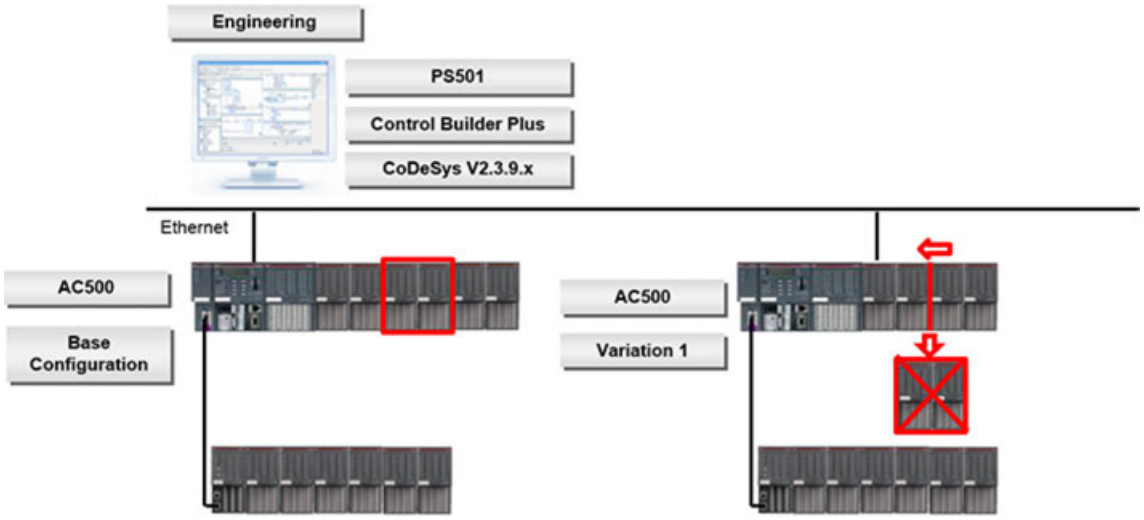
Settings dialog

The following basic options and settings concerning the user accounts can be made:

- Maximum number of authentication trials: If activated, the user account will be set invalid after the specified number of trials to log in with a wrong password. If not activated, the number of erroneous trials is unlimited. Default: option activated, number of trials: 3; permissible values: 1-10.
- Automatically log out after time of inactivity: If activated, the user account will be logged out automatically after the specified number of minutes of inactivity (no user actions via mouse or keyboard registered in the programming system). Default: option activated, time: 10 minutes; permissible time values: 1-180 minutes.

1.6.5.1.3 Flexible AC500 configuration

As of Automation Builder version 1.1 feature for flexible AC500 configuration is introduced. This feature enables you to use only one Automation Builder project for multiple different hardware configurations. Especially for product lines with many hardware variants, flexible configuration is a cost and time saving way for AC500 configuration.

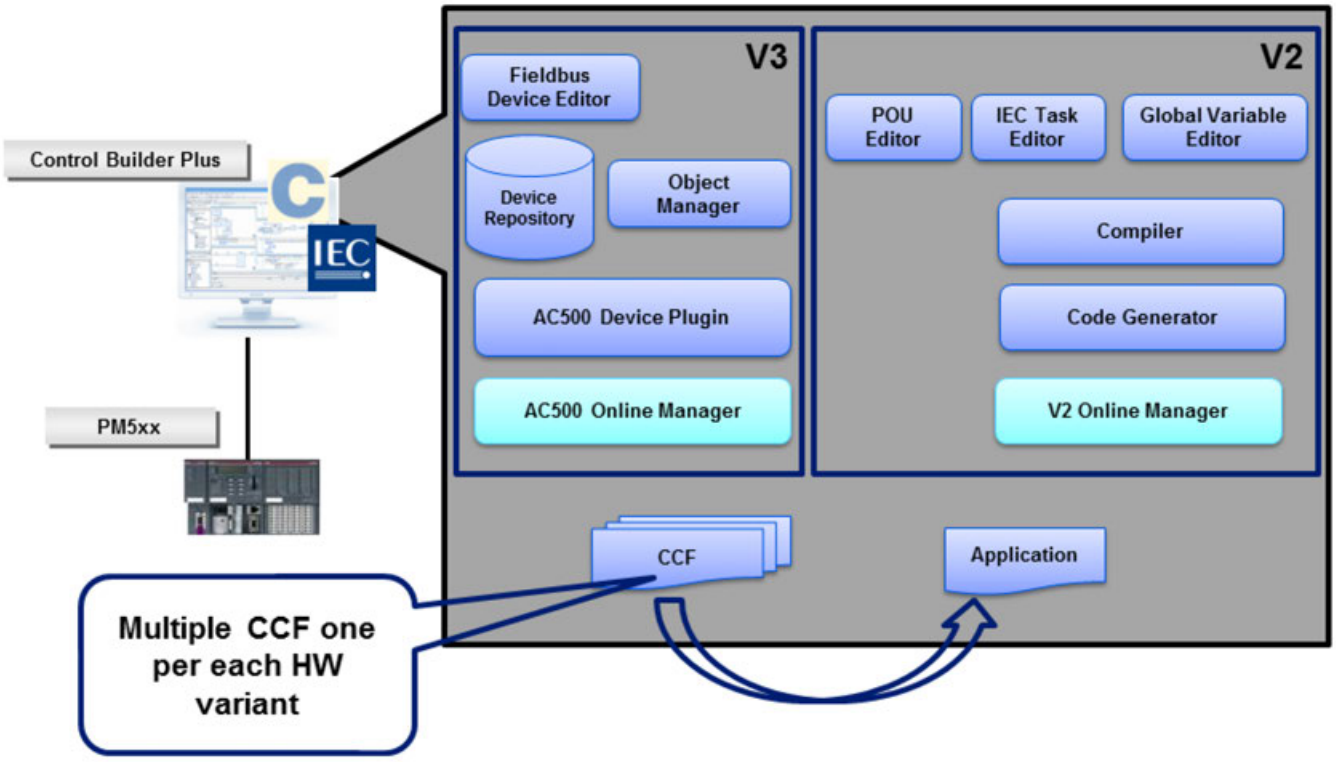


Flexible configuration is not supported for CM574-RS/RCOM and SM560 devices.

An Automation Builder project consists of CODESYS V3 hardware configuration and CODESYS V2 IEC application. Usually, from hardware configuration a *.ccf file is created and downloaded with IEC application to the PLC. Whenever an application is loaded to the PLC firmware, this *.ccf file is evaluated by I/O drivers.

By default, for configuration Config.ccf with FlexConfID 0 is created. When flexible configuration is enabled, Automation Builder creates one individual *.ccf file for each variant of hardware configuration: Cfg1.ccf with FlexConfID 1, Cfg2.ccf with FlexConfID2 etc. IEC application loads all available *.ccf files to the PLC, however, only one *.ccf file can be analyzed by PLC firmware at a time. The user decides which *.ccf file to be used by activation of the desired file. The active configuration is represented by the FlexConfID.

*Maximum number of manageable *.ccf files is limited to 255. FlexConfID 0 is occupied by the default configuration Config.ccf. Cfg0.ccf is not allowed.*



Application configuration and hardware configuration are separated but linked via I/O Mapping. Hence, flexible configuration is only operable in combination with an I/O bus. In order to use several hardware configurations with one application configuration I/O Mapping must be identical.

It is currently not possible to have alternative configurations of safety devices in a project with flexible configuration. However, if configuration of the safety devices is equal in all alternative configurations, flexible configuration is operable.

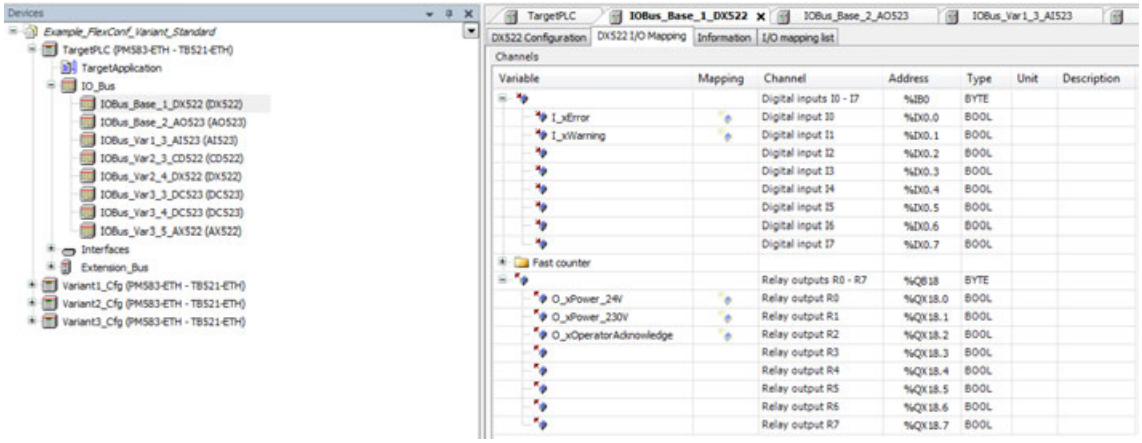
Software configuration

Depending on the use case and on your requirements on the hardware configuration of your variants, perform a standard or a advanced configuration.

- Standard Configuration: Use this configuration if the hardware configuration of your variants is close to the hardware configuration of your primary PLC and contains the same I/O modules in the same order. The parameterization for the hardware variants can vary. I/O modules that are unnecessary for one of your hardware variants can be deactivated easily with the Ignore Module parameter later on. However, the preset module order from the primary configuration must not be changed, as a I/O module can only be ignored but not replaced by another type ↬ *Chapter 1.6.5.1.3.1.1 “Standard configuration” on page 5776*.
- Advanced Configuration: Use this configuration if the order of the I/O modules must be changed, or if the variants require different I/O modules, or if your variants require different addresses for already preset channels. Note: Advanced Configuration requires additional manual actions: a) modifications in CODESYS for I/O Mapping b) file transfer to the file system ↬ *Chapter 1.6.5.1.3.1.2 “Advanced configuration” on page 5778*.

Standard configuration

1. Create an Automation Builder project with a PLC device. This PLC contains your primary hardware configuration (in the example: TargetPLC). Flexible configuration is only operable in combination with a I/O module. We recommend you, to add all S500 I/O modules that are required for the primary PLC or for a PLC of one of your hardware variants. Configure all PLCs in your primary hardware configuration.



   ⇨ Naming concept in the example:

   IOBus_Base_1: This I/O module is at first position of the PLC. It is valid for the primary configuration and for all variants.

   IOBus_Base_2: This I/O module is at second position of the PLC. It is valid for the primary configuration and for all variants.

   IOBus_Var_1_3: This I/O module is at third position of the PLC. It is only valid for the first hardware variant (Variant1_Cfg).

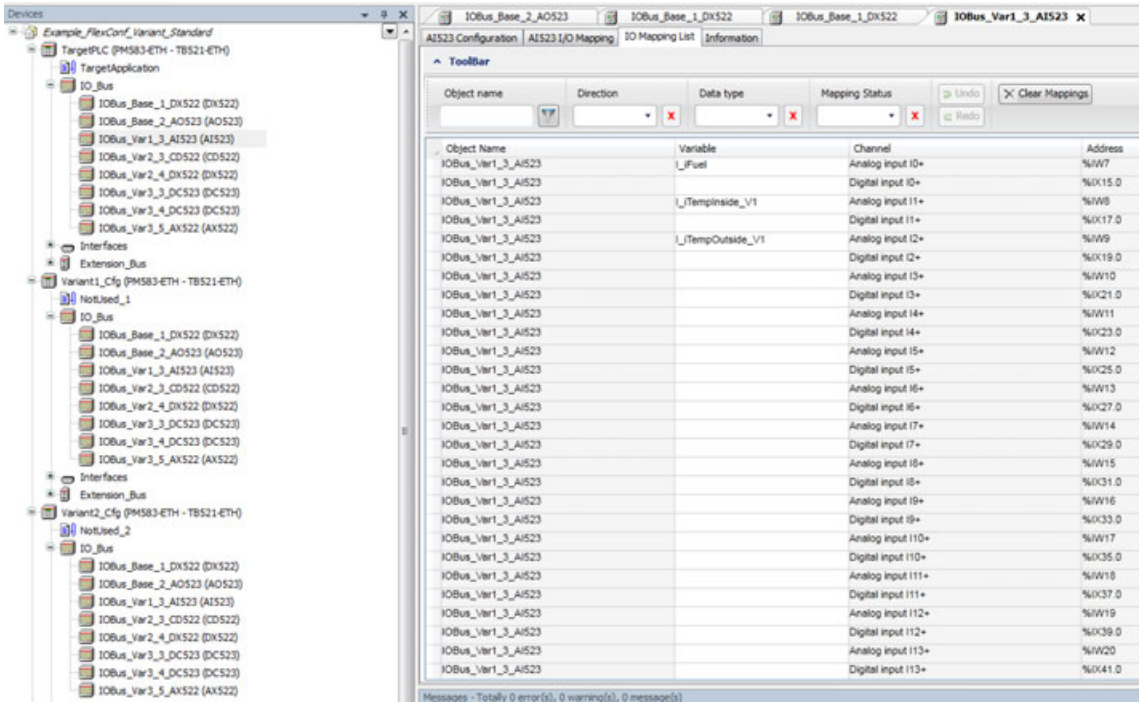
   IOBus_Var_2_3: This I/O module is at third position of the PLC. It is only valid for the second hardware variant (Variant2_Cfg).

2. In the I/O Mapping tab, define the Mapping for the IEC addresses. The variables of the I/O modules can be changed or even deactivated by enabling the parameter Ignore Module.

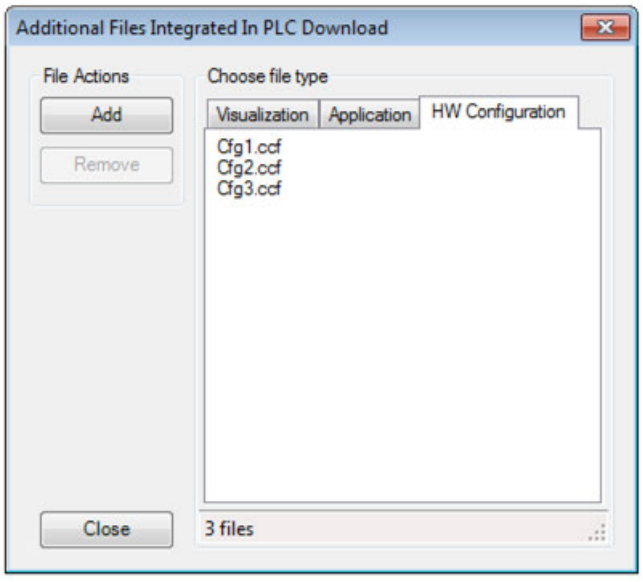
> *IEC addresses must not be changed in the configuration for the variants. If this is required, perform a advanced configuration ↳ Chapter 1.6.5.1.3.1.2 “Advanced configuration” on page 5778.*

3. Copy and paste your complete primary PLC configuration to your project (right-click on PLC_AC500). With this, variants of your hardware configuration are created.

4. Rename the copied PLC configurations. In the example: Variant1_Cfg ... Variant3_Cfg.



5. In order to manage all available *.ccf files, export the PLC hardware configuration to any folder in the file system: *“Export ➔ Export PLC Hardware configuration”*.
6. Repeat this step for each PLC variant to export all *.ccf files to one folder.
7. Right-click *“TargetApplication ➔ Manage additional files for PLC”* to load all *.ccf files to the PLC.
8. In the dialog open the *“HW Configuration”* tab. With *“Add”* select the *.ccf files from the folder previously defined. Ensure that the files to be added follow the syntax Cfg<FlexConfID>.ccf with a continuous FlexConfID number.



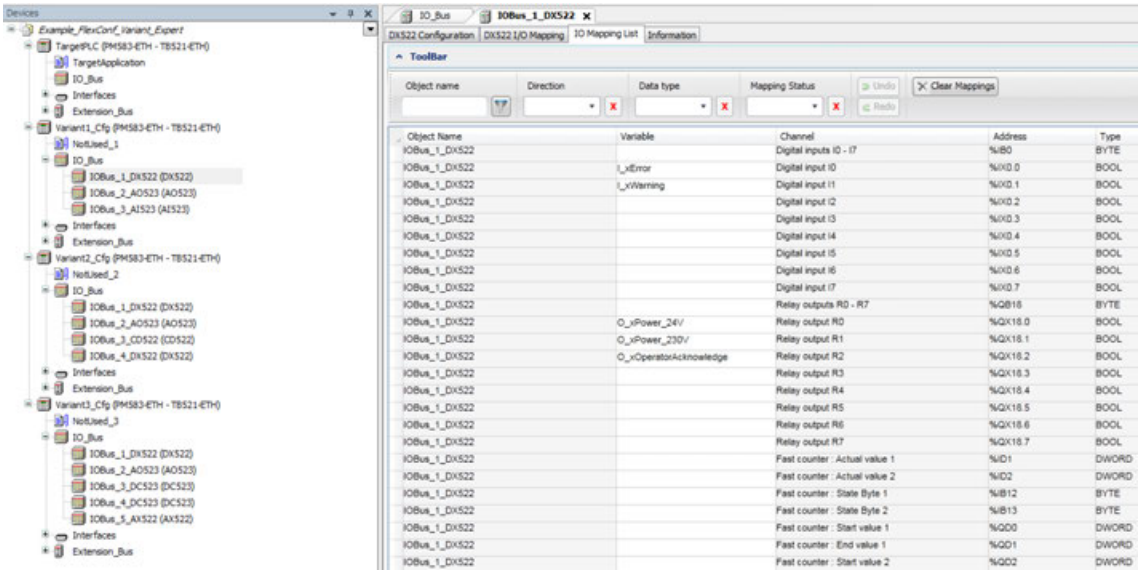
9. Close the dialog, compile the project and create a boot project to finish configuration. Then, activate the desired hardware configuration that shall be used ↳ *Chapter 1.6.5.1.3.2 “ Activating another hardware variant (Configuration file) ” on page 5780*.

Advanced configuration

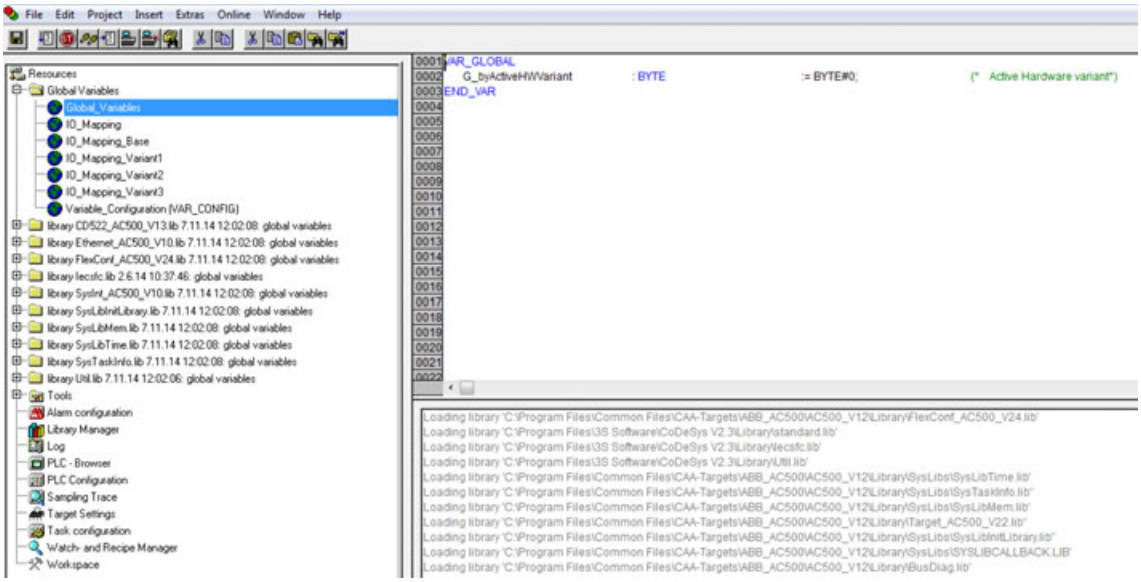
In a standard configuration the global variables for I/O Mapping are set automatically. As this is not possible for hardware variants with complex channel/address settings, global variables must be defined in CODESYS to map the channels in the applications. Advanced configuration allows to operate independent hardware configurations.

1. Create a Automation Builder project with a PLC device. This PLC contains your primary hardware configuration (in the example: TargetPLC).
2. Create as many PLC hardware variants as required.

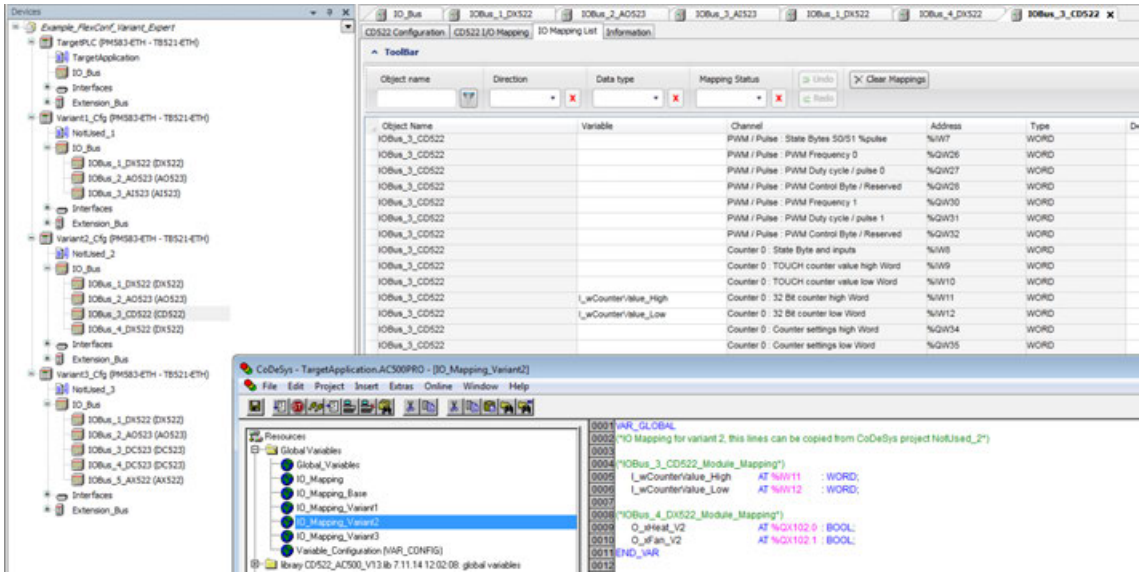
> *The main program that includes the hardware configuration of all your variants is defined in the TargetApplication object of your TargetPLC (not in the application objects of your variants (NotUsed_x)).*



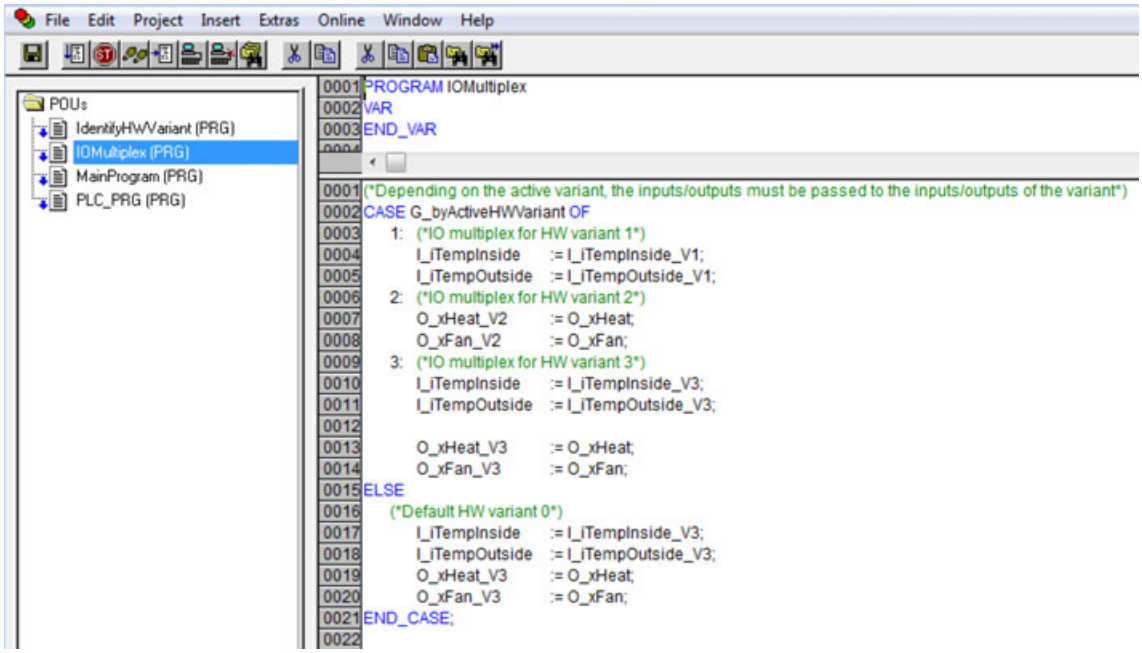
3. Double-click *"TargetApplication"* to define I/O Mapping for the IEC addresses in CODESYS. Due to a different module configuration this cannot be done in the *"I/O Mapping"* tab of Automation Builder.
4. Under *"Resources"* define a global variable that can be used to identify the active hardware variant (in the example G_byActiveHWVariant).



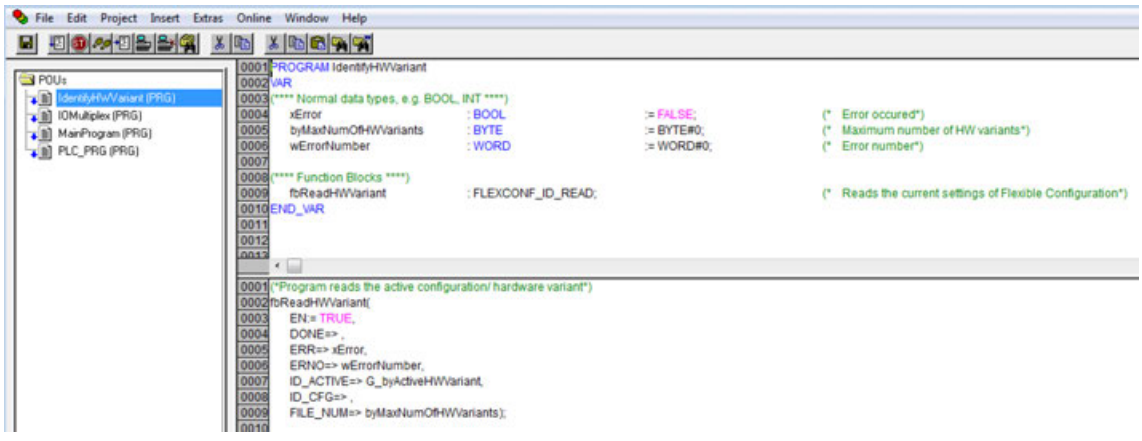
5. For a better readability we recommend you to create I/O Mapping objects for your variants which lists the address mapping for the inputs and outputs of a I/O module.



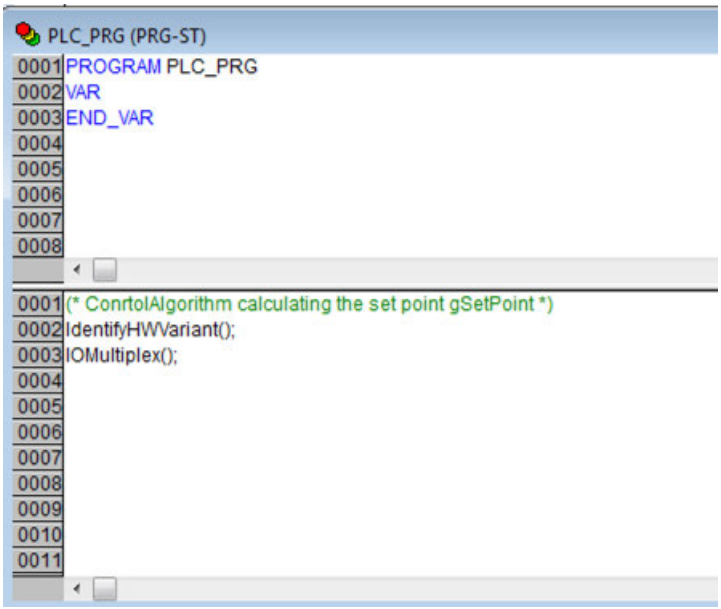
6. Under *“POUs”* create a new program (in the example IOMultiplex) which describes the addresses for inputs and outputs to be used for each hardware variant.



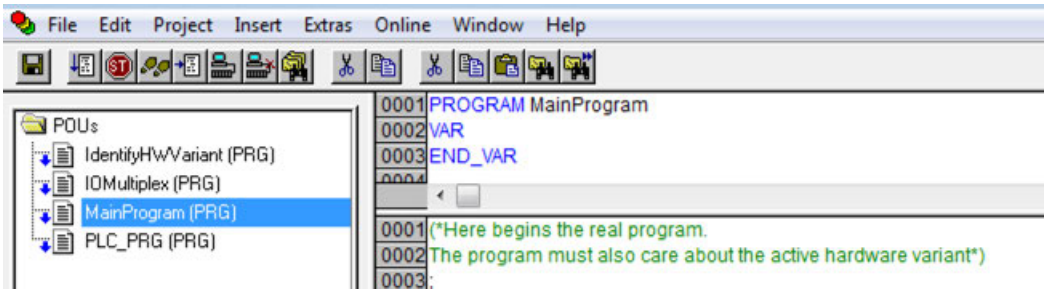
7. Under *“POUs ”* create a new program (in the example IdentifyHWVariant).



8. Under *"POUs"* create a new program (in the example PLC_PRG) which refers to the previously defined program (IOMultiplex) and your global variable (G_byActiveHWVariant).



9. Under *"POUs"* create a new program (in the example MainProgram).



10. Save and close your CODESYS configuration. Proceed with exporting the PLC configuration as described under Step: Exporting the PLC.

Activating another hardware variant (Configuration file)

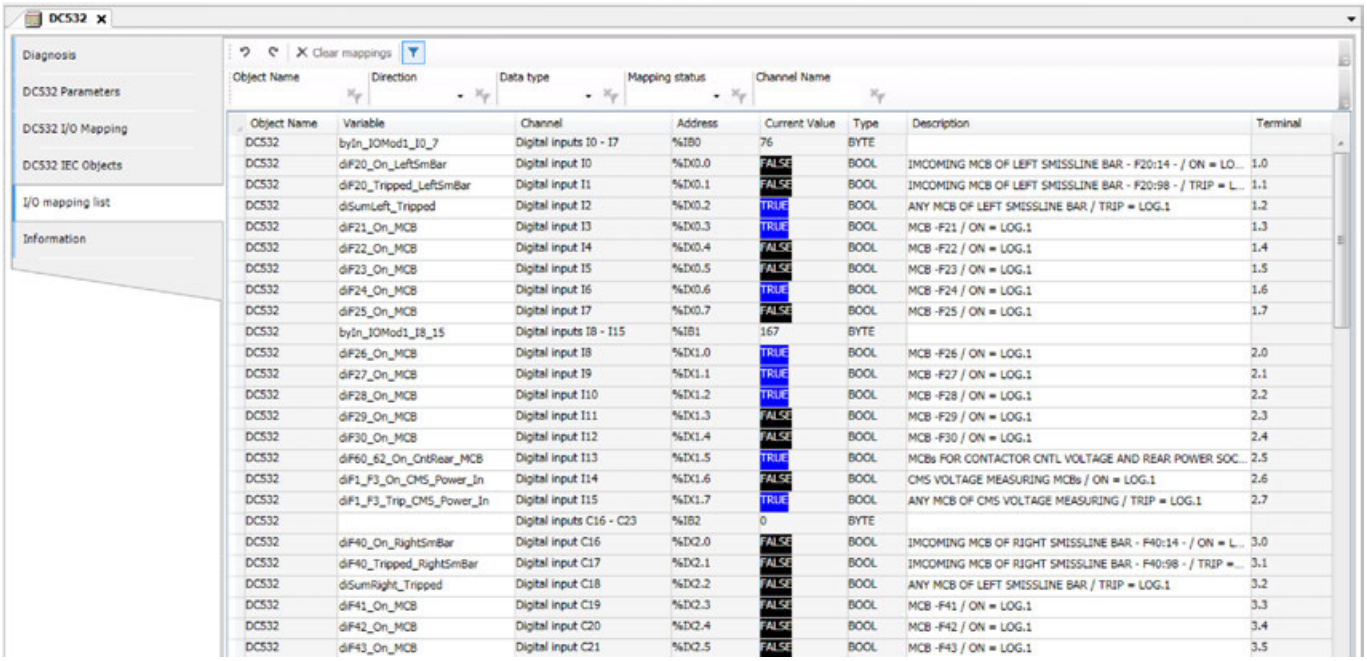
Only one hardware configuration (*.ccf file) can be loaded and analyzed by AC500 firmware at a time. Flexible configuration allows you to switch between hardware variants by using one of the following methods:

- Function block in the IEC application. See Flexible Configuration Library.
- AC500 display (CFG button until FL 000 is displayed). See Display and Operating Elements on the Front Panel of a processor module.
  Note: At the moment this configuration method is not supported for PM595-4ETH.
- A PLC browser command. Possible commands:
  – fcidget: Shows the current values of the settings for flexible configuration. Possible values: Active FlexConfID, Configured FlexConfID, Number of configuration files.
  – fcidset <flexconfid>: Changes the value of the FlexConfID in the confdata file.

> *Activation becomes valid only after PLC reboot - independent from the used method. Information on active configuration is stored in the confdata file in the PLC's flash memory. The online functions of Automation Builder are only supported for the configuration loaded to the PLC.*

1.6.5.1.4 I/O mapping list

Automation Builder contains an I/O mapping list feature for creating mapping variables with better usability support compared to the tree structured view. Details on the tree structured view is provided in the CODESYS Development System.



Functionalities of the I/O mapping list:

- Displays I/O mappings for current node and all valid subsequent child nodes.
- Displays channel information with additional columns.
- Supports keyboard functions such as *cut*, *copy*, *paste*, *delete*, and *select all* within the editor and within Excel spreadsheet (for bulk editing).
- Contains a toolbar for various actions, e.g. filtering, undo/redo and clear mappings.
- Supports single click edit and easy navigation using arrow keys.
- Improvised error handling:
  - Allows to enter invalid mapping variables. This provides flexibility in bulk editing. Only when saving the project, the errors - according to IEC 61131 standard - are displayed.
  - In the message window, the error log is visible. The user can track the errors to their corresponding channel in the editor.
- Allows multi-selection of rows and columns. (Random selection is not allowed.)

Configuring I/O mapping list

Automation Builder supports tree and list based editors for creating I/O mapping variables.

1. From the **Tools** menu, select **Options**.
2. Under **Automation Builder**, select the **Editors** tab.
3. Choose your desired mapping dialog and click **OK**.
   - Choose **tree based** to display the I/O mapping in tree structure.
   - Choose **list based** to display the I/O mapping as list with the functionalities of the ToolBar.
   - Choose **both** to display both the tree structure (**I/O Mapping** tab) and the list view (**I/O mapping list** tab).

Available channel information

The I/O mapping list displays the channel information in offline and online mode. In online mode, all columns are read-only. In offline mode, some columns are editable.

The order of the devices in I/O mapping list is synchronized with the order in the device tree.

The channels of a device are ordered by the device description file. If channels have a section, the channel information is represented in a specific format.

Example: Fast counter: Actual value 1. These channels are listed at last position of a device.

Editing I/O mapping list

1. In the device tree, double-click **IO_Bus** to configure entire I/O mapping list of different I/O devices.
2. Enter the variables and descriptions to map the I/O devices.

Do not start variable names with a number or a special character. When saving the project, this generates an error. Example: 12input3, @input4.

3. Click **Save Project** to save the I/O mapping changes.

Toolbar

Filtering

Especially in case of long I/O mapping lists, it might be helpful to filter the I/O mappings. For this, click the *"Filter"* icon to display all available criteria for filter options.

When reducing the width of the editor, some filters might be hidden.

Undo, redo and clear

- **Undo**: Cancels the last change.
- **Redo**: Repeats the last change.
- **Clear mappings**: Deletes all variables and descriptions.

1.6.5.1.5 Setting standard configuration

If the target setting configuration is changed, standard configuration can be restored:

1. Open CODESYS Development System.
2. In the *"Resources"* tab, double-click *"PLC Configuration"*.
3. Select *"Menu Extras ➔ Standard Configuration"*.

1.6.5.1.6 Later change-over of a target system

Changing the processor module type

In a project, you can change the target system by changing the type of processor module or terminal base type. If possible, the device configuration of fieldbusses and interfaces is kept and switched over to the device configuration of the new module.

Target change options:

- between platforms: from V2 platform to V3 platform (and vice versa)
- between module types: from AC500 (standard) to AC500-eCo (and vice versa)
- a combination of changed platform and changed module type

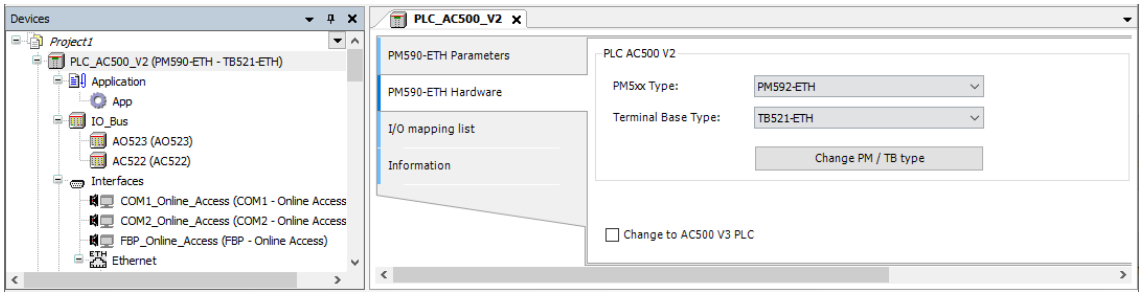
Target change from a V2 processor module to another V2 processor module

Target change options:

- AC500 V2 processor module ➜ AC500 V2 processor module
- AC500 V2 processor module ➜ AC500-eCo V2 processor module
- AC500-eCo V2 processor module ➜ AC500 V2 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V2 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *"PM5<...> Hardware"* tab.
3. Select the desired V2 processor module from the *"PM5xx Type"* drop-down list.



4. Ensure the correct *"Terminal Base Type"* is selected and click *[Change PM / TB type]*.
   ⇨ The new V2 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

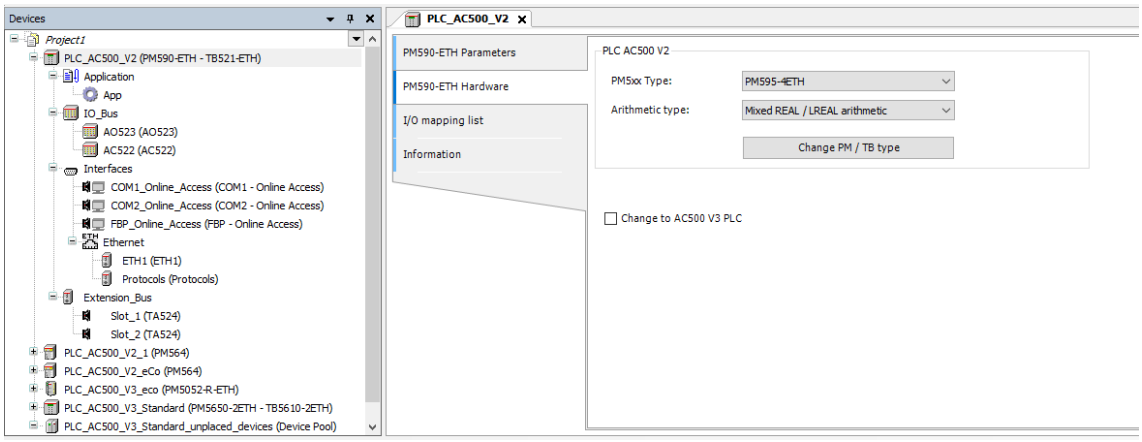
Target change to PM595

Target change options:

- AC500 V2 processor module ➜ PM595
- AC500-eCo V2 processor module ➜ PM595

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node.
3. Open the *"PM5<...> Hardware"* tab and select 'PM595-4ETH' from the *"PM5xx Type"* drop-down list.



4. With the *Arithmetic type* item, processing of Structured Text can be modified.
   - *"Mixed REAL/LREAL arithmetic"* (default value):
     Calculation of LREAL variables is extended to the extended co-domain of 64 bit. In general, we recommend to keep the default setting as this setting provides enough accuracy for code calculation.
   - With *"Only REAL arithmetic"* the LREAL variables are processed as REAL variables (co-domain of 32 bit).
5. Click *[Change PM / TB type]*.
   ⇨ The PM595 is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

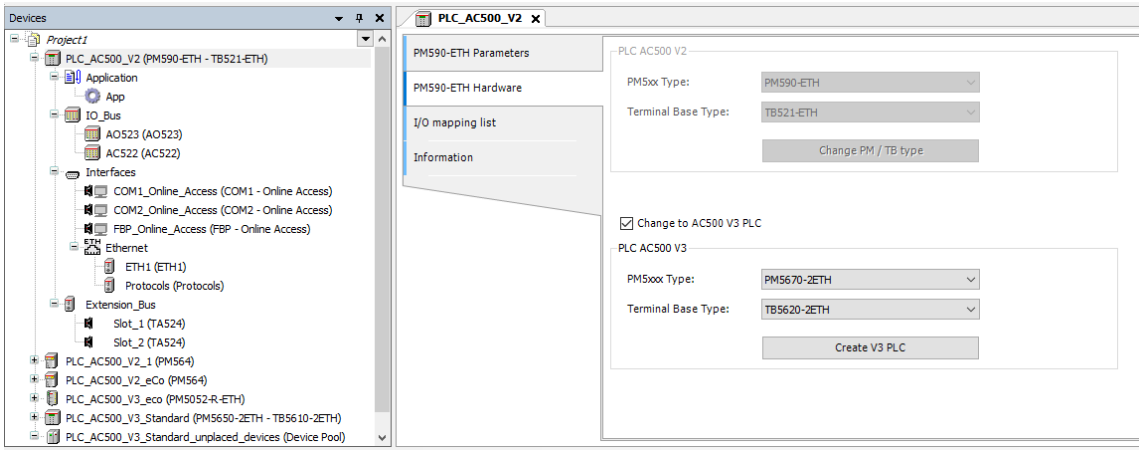
Target change from a V2 processor module to a V3 processor module

Target change options:

- AC500 V2 processor module ➜ AC500 V3 processor module
- AC500 V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500 V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *"PM5<...> Hardware"* tab.
3. Enable *"Change to AC500 V3 PLC"* and select the desired V3 processor module from the *"PM5xx Type"* drop-down list.



4. Click *[Create V3 PLC]*.
   ⇨ The new V3 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

> *In case of a target change from AC500-eCo V2 to AC500-eCo V3, the I/O bus and Ethernet configuration is kept.*

Customer libraries

CODESYS for AC500 V2 products contains different types of libraries:

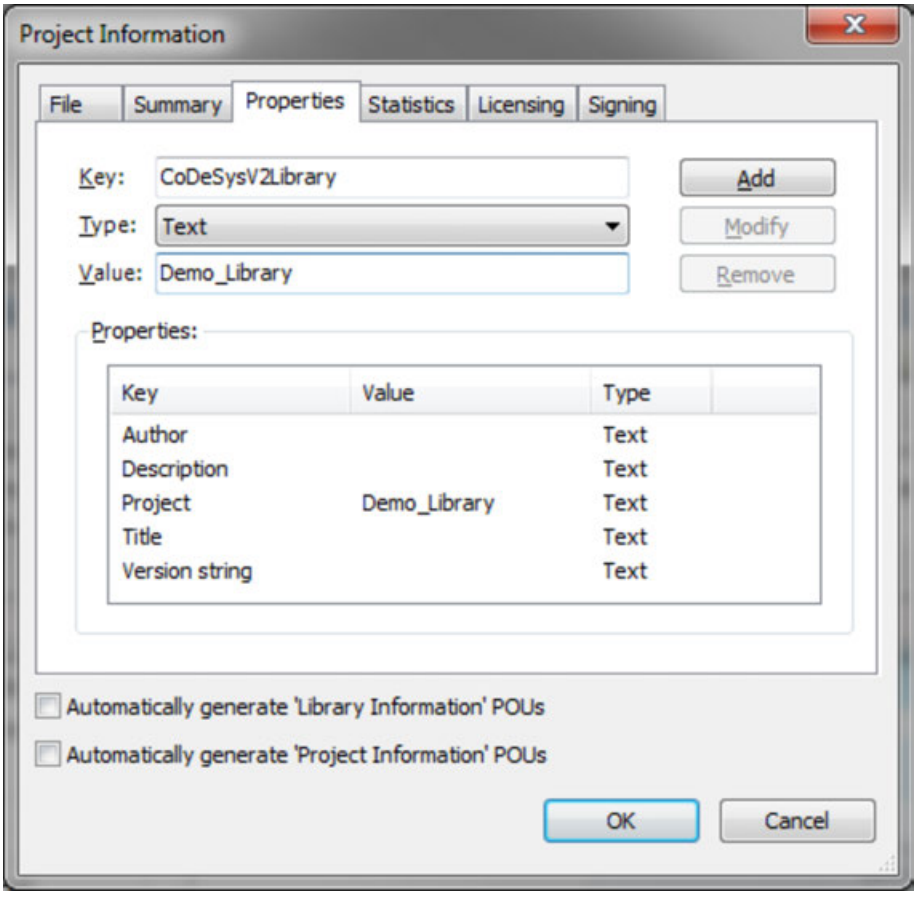
- Standard CODESYS libraries
- Specific AC500 libraries
- Customer libraries

In general, the Standard CODESYS libraries and the AC500 libraries are automatically converted during a target change from AC500 V2 to AC500 V3. Those libraries that cannot be converted (e. g. because there is no matching in V3) are created automatically in the V3 Library Manager and must be manually deleted by the user after the target change.

The customer libraries have to be converted manually using the Library Converter integrated into the Automation Builder installation:

1. In Automation Builder click *“File ➔ Open project”*.
2. Select the CODESYS library for AC500 V2 products which has to be converted.
3. After conversion of the library, open the view POUs in the device navigator and double-click *“Project Information”*.
4. To have the library automatically available in the V3 project, enter “Company”, “Title” and “Version” in the specific fields of the dialog.

   Then, open the *“Properties”* tab. For the target change the new *“Key”* "CoDeSysV2Library" has to be added. Under *“value”*, enter the name of the CODESYS library and click the *“Add”* button.



Click *“File ➔ Save project”* and install into the library repository.

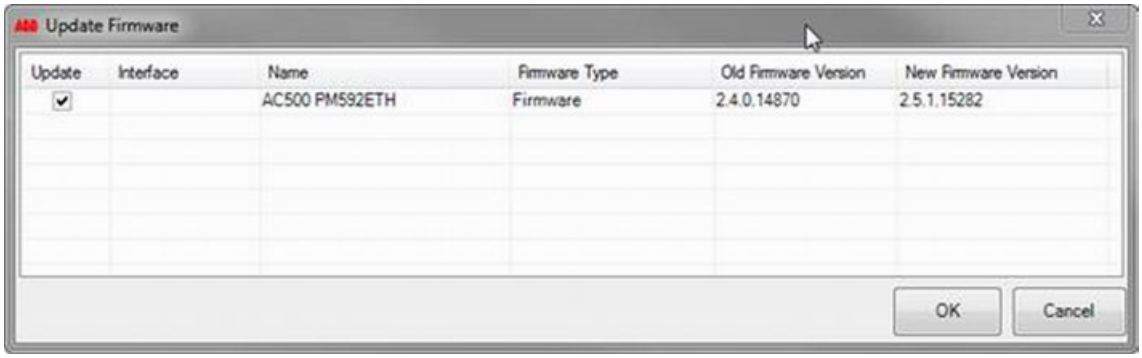
1.6.5.1.7 Firmware identification and update

> *Without direct access to the internet, a firmware update with the memory card is also possible.* *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*

General information

- Firmware for AC500 CPUs and communication modules is provided with Automation Builder
- Firmware versions specific for profile (Automation Builder major.minor version)
- Firmware download from Automation Builder in addition to other mechanisms possible
- Firmware can be stored to an *Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339*. This is done using a standard PC card reader with memory card interface.
- Firmware image from the Internet: The firmware- and bootcode files can be downloaded from the following ABB website: www.abb.com/plc. Use the according AC500 update description.

After pressing “Update Firmware” the following dialog will be shown. It displays the firmware that can be updated. If the CPU firmware version is lower than 2.5.0.0 the CPU firmware must be updated before any Communication Module firmware can be updated.



If multiple firmware types of a device can be updated, all firmware types must be updated in one step, thus selection in column update is possible on device level. Currently multiple firmware types exist only for CPUs.

After pressing “OK” the firmware files are downloaded to the RAM disk of the CPU and the update is triggered. Firmware update is only possible when:

- PLC is in “Stop”.
- The RAM disk has enough free space for the largest firmware file to download plus about one kByte for control files.
- The gateway settings in the CODESYS V2.3 project are correct.

The PLC must be rebooted after the firmware update has been completed. This could be done i.e. by power cycle or with PLC browser command “reboot”.

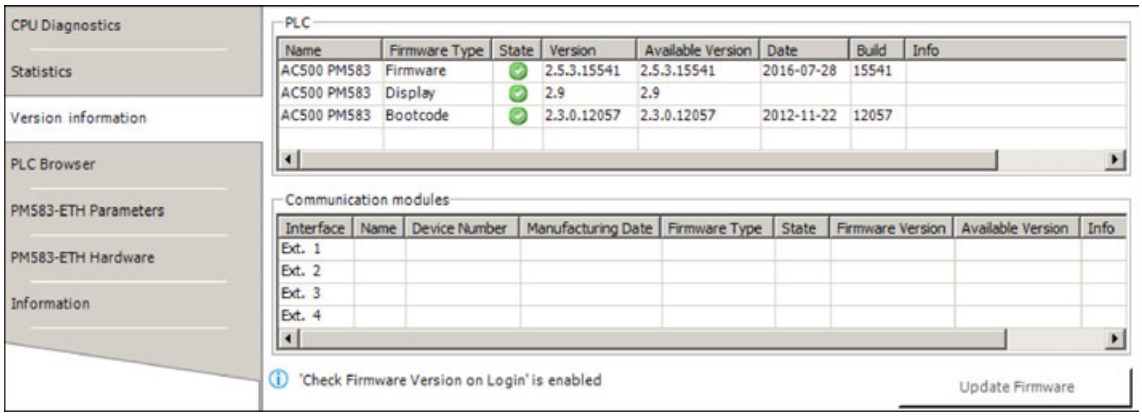
When updating the CPU firmware from a version below 2.5.0.0 to 2.5.x.x or higher the end of the firmware update can’t be detected automatically. The user must observe the PLC to check that firmware update has been completed before rebooting the PLC. See the documentation for the CPU type on how completion of update is indicated.

Version information

Information on the firmware versions of the processor modules or communication modules, is provided on the *“Version information”* tab.

Remarks:

- The *"Version information"* tab displays the version identified on the device and the version provided with Automation Builder.
- The firmware on the devices must match to the Automation Builder version. Upgrade or downgrade to version supplied with Automation Builder is recommended (especially for CPUs) to ensure correct functionality.
- The firmware type can be changed to the type required by the hardware configuration for devices that support changing the firmware type. E.g., the onboard field bus communication modules of PM595 that may be used as PROFINET, Ethernet or EtherCAT communication module.



State icons

	Firmware version on device matches version supplied with Automation Builder.
	Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Only for communication modules if CPU firmware must be updated first. This happens when CPU firmware has version below 2.5.0.0. Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Identified device is different from configured device, thus no firmware update is possible. Happens only for Communication Modules.
No icon	Firmware of device is not updateable or no newer firmware than the initial version is available.

The [Update Firmware] button to download the new firmware is only enabled if there is updateable firmware.

Updating the firmware of AC500 devices from the memory card

There are two options to update a device's firmware from a file stored on a memory card:

- Update automatically during system start.
- Trigger the update via a PLC Browser command.

The following table shows the update possibilities per device:

Device / File	From memory card during System start	From memory card with PLC-Browser command	Comment
CPU Bootcode	no	yes -> sdboot x with x = according version; e.g sdboot 2_3_1	
CPU Firmware	yes	yes -> sdfirm x with x = according version; e.g sdfirm 2_3_1	
CPU Display	yes	yes -> sddisplay x with x = according version; e.g sddis-play 2_3	only for PM57x, PM58x, PM59x
Communication Module Firmware	yes	yes -> sdcoupler x with x = [1|2|3|4] = external Communi-cation Module [1|2|3|4] (also CM574-RS)	
OnboardIO Firm-ware	yes	yes -> sdonboardio x x = according version; e.g sdonboardio 1_1_6	only for PM55x, PM56x
RTC Firmware	yes	yes -> sdrtcbat x x = according version; e.g sdrtcbat 1_1_7	only for TA561-RTC or TA562-RS-RTC as option of PM55x or PM56x

Please be aware, that during the update triggered with a PLC Browser Com-mand the file SDCARD.INI on the memory card is evaluated too. For additional details see Chapter 1.6.6.2 “Memory card in AC500 V2” on page 6339:

- *Memory Card File System on the general memory card structure.*
- *The Command File SDCARD.INI for details on the SDCARD.ini file and its keys.*
- *File content as of version V2.x for a detailed description of the behavior of the update keys.*

NOTICE!

No POWER OFF during flash process!

During the flash process it is not allowed to switch the power off; otherwise, the CPU could be damaged and unavailable anymore.

During the display update it is powered off and on automatically!

Update from memory card during system start

To update the firmware of the AC500 CPU via memory card without control by PLC-Browser commands, proceed as follows:

1. Download and extract the update files on your memory card or prepare the memory card, see *Chapter 1.6.6.2.4 “Storing/Loading the Firmware to the memory card for AC500 V2 products” on page 6354*.
2. Insert prepared memory card into the CPU. The file *SDCARD.INI* on the memory card contains settings that automatically perform the update (details see *Chapter 1.6.6.2.2.2 “Command file SDCARD.INI for AC500 V2 Products” on page 6342*).
3. Switch power on.

   The individual steps are indicated as follows:

Process	Indication	Remark
Reading the firmware	RUN LED flashes fast	If you remove the memory card during reading, the previously stored firmware version is kept.
Flashing the firmware	RUN LED and ERR LED flash fast	Warning: If the control voltage is switched off during flashing, the firmware will be corrupted!
Firmware update completed successfully	RUN LED flashes slow (app. 1Hz)	
Incorrect firmware update	ERR LED flash slow (app. 1Hz)	

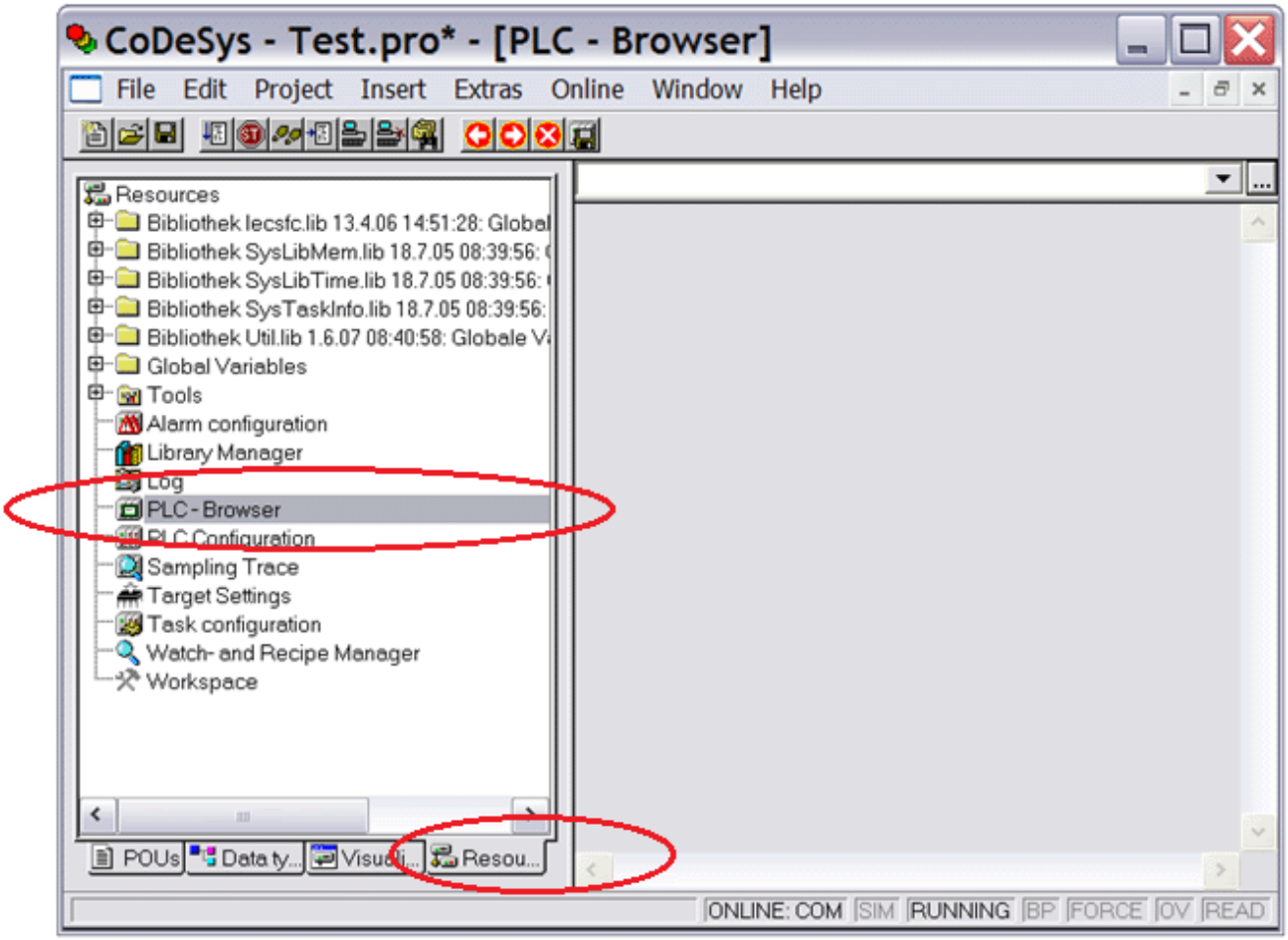
   ⇨ Updates operation starts (green LED blinking = reading from SD card / red and green LEDs blinking fast = flash process).
4. Reboot the CPU to load the updated firmware.

A specific firmware version can be loaded. This is done by setting the parameter CPUPM5x1=2 or 3 and creating an according key for the CPU. The firmware has to be copied to the according directory. See chapter The Command File *Chapter 1.6.6.2.2.2 “Command file SDCARD.INI for AC500 V2 Products” on page 6342* .

If the file SDCARD.INI contains the parameter setting FunctionOfCard=3 (firmware update / load user program), first the firmware and then the user program are read from the memory card and then stored in the according Flash memory.

Update from memory card with PLC browser commands (Online mode)

To start a firmware update from the PLC browser, it is necessary to be logged in into the CPU with your PC.



1. Open the PLC browser from the Resources tab (see above).
2. Insert the memory card with the correctly set *SDCARD.INI* file (see *Chapter 1.6.6.2.2.2 “Command file SDCARD.INI for AC500 V2 Products” on page 6342*) and corresponding firmware files on it into the PLC (see *Chapter 1.6.6.2.4 “Storing/Loading the Firmware to the memory card for AC500 V2 products” on page 6354*).
3. Write the requested command into the PLC browser.
4. The corresponding command is now activated, for example *sdfirm 2_5_1* command.
5. If the firmware update was successful, DONE is shown in the PLC Browser.
6. Remove the memory card and restart the CPU to load the new firmware.

NOTICE!

The memory card has to be removed before restart!

To get information about the firmware version of the CPU enter the command *rtsinfo* into the PLC-Browser.

Only for AC500-eCo PM5x4 modules with firmware version < 2.x: First the CPU firmware has to be updated from V1.3.x to V2.0.x. After doing the update the CPU has to be restarted.

Then any module can be updated as described in this chapter!

Update via FTP

With AC500 Firmware above V2.1 it is possible to update a CPU via the FTP server.

Only PLCs with onboard Ethernet (i.e. AC500 CPU with Ethernet) and firmware version 2.1.0 or higher support the FTP server!

Update steps

1. To update your CPU via FTP, you first must configure and activate the FTP server with the set checkbox "Allow Firmware update".

   Details: *Chapter 1.6.5.3.8.3 “Configuration of FTP server (>= CBP 2.4) ” on page 6191*.
2. Connect to your CPU's system RAM disk (i.e. memory location "ramdisk").

   Details: *Chapter 1.6.5.3.8.4 “Connection to a PLC running a FTP server” on page 6192*.
3. Switch your CPU to STOP.
4. Copy the according *.gza* file to the system RAM disk (= memory location *ramdisk*):

Product	Name of Firmware File
PM55x-ETH, PM56x-ETH	Pm55xE.gza
PM57x-ETH, PM58x-ETH	Pm58xN.gza
PM59x, PM59x-ETH	Pm59xRD.gza
PM595-x-4ETH	Pm595.gza

   ⇨ The CPU will automatically update its firmware with the downloaded file, flashing the RUN and ERR LEDs.

   As of V2.5 copy file *SDCARD.INI* to the system RAM disk. As of V2.5 all firmware updates are triggered by the command file *SDCARD.INI.* This is independent from the way of the firmware update (memory card, FTP, *write file to plc*, …). In addition a result file of the firmware update is generated (*SDCARD.RDY*, identical path as *SDCARD.INI*). The evaluation of this file shows the results of the updates.
5. Wait until all LEDs ceased toggling.
6. Power cycle your PLC and the new firmware will be loaded.

Potential problems:

- If you cannot connect to your CPU, check if the FTP server is configured and running correctly.
- If you cannot download the file, increase your configured "Sessions". Some FTP clients require more than one FTP connection for their operations, because logins and downloads are handled in separate sessions.
- If you downloaded the *.gza* file into the RAM disk, but the CPU does not start to update (i.e. the LEDs do not toggle):
  - Check, if the option *Allow firmware update* is set for the FTP server.
  - Check, if the CPU is in STOP state; in RUN state no firmware update is allowed.
  - Check, if you copied the correct firmware file.
  - Check, if you copied the file into the correct memory location (correct is *ramdisk* - incorrect is "sramdisk", "userdisk", "flashdisk", "sdcard").

Update PM595 firmware

Firmware update of the CPU

To update the firmware of the PM595 CPU proceed as follows:

1. Login on the CPU using Automation Builder.
2. Choose *“Version Information”* Menu.
3. Click *“Update”* at the bottom of the Menu
   ⇨ Run and ERR LED are blinking.
4. Wait until the blinking stops.
5. Reboot the CPU.
6. Update the internal communication module to actual PROFINET Firmware.

Firmware update of internal communication module for PROFINET

ABB recommends that users carry out the firmware update via Automation Builder. ↬ Chapter 1.6.5.1.7.2 “Version information” on page 5786

To update the internal communication module to actual PROFINET Firmware proceed as follows:

1. Unzip the file „Update_PNIO.zip“ on memory card.
2. Insert the memory card into the PM595 CPU.
3. Switch Power Off/On:
   ⇨ Run LED blinks

   Run and Rdy LEDs on the left side of the PM595 (left column) for 1st. Internal communication module blink

   RUN LED blinks

   Run and Rdy LED on the left side of the PM595 (right column) for 2nd. Internal communication module blink

   RUN LED blinks slowly

   The procedure should require 3 – 5 minutes
4. Remove the memory card.
5. Switch Power Off/On.
6. Login in Automation Builder:

   Navigate to Version Informations.
   ⇨ Firmware V 2.8.1.2 or newer for Internal communication module must be displayed.
7. Update Internal communication module on EtherCAT Firmware

Firmware update internal communication module of PM595 for EtherCAT

ABB recommends that users carry out the firmware update via Automation Builder. *Chapter 1.6.5.1.7.2 “Version information” on page 5786*

To update the internal communication module to actual EtherCAT Firmware proceed as follows:

1. Unzip the file „Update_ETHCAT.zip“ on memory card.
2. Insert the memory card into the PM595 CPU.
3. Switch Power Off/On.

   ⇨ Run LED blinks o Run and Rdy LEDs on the left side of the PM595 (left column) for 1st. Internal communication module blink

   RUN LED blinks

   Run and Rdy LED on the left side of the PM595 (right column) for 2nd. Internal communication module blink

   RUN LED blinks slowly

   The procedure should require 3 – 5 minutes.
4. Remove the memory card.
5. Switch Power Off/On.
6. Login in Automation Builder:

   Navigate to Version Informations.

   ⇨ Firmware V 4.2.23 (2) or newer for Internal communication module must be displayed.

Update CI52x-Modbus firmware

Requirement: A firmware update file is available, e.g. AC500_CI52x_Firmware_V3.2.8.bin.

The CI52x Modbus firmware update is only available in the Automation Builder IP Configuration Tool.

Installation of the IP configuration tool

1. In Automation Builder click *“Tools* ➔ *Installation Manager”* to start the Installation Manager.
2. Close any other running instances of Automation Builder. Then, click *“Modify”* in the Installation Manager.
3. Select the option *“IP Configuration Tool”* from the list and start the installation of the IP Configuration Tool.

Firmware update procedure

1. In the IP Configuration Tool click *“Scan”* to initialize a device scan.
2. From the list select the CI52x-MODTCP device(s) which shall be updated and click *“FW Update”*.
3. Select the firmware update file (e.g. AC500_CI52x_Firmware_V3.2.8.bin) to initialize a signature check and start the update procedure.
4. After the update, click *“Scan”* again to retrieve the firmware version of the device.

Troubleshooting

After the IP Configuration Tool has been installed, the firmware update of the CI devices can be initialized. If the CI firmware update fails, check the troubleshooting hints and follow the instructions.

General hints

- Close all unused applications on the update PC and do not open Automation Builder or any other applications during the firmware update.
- Stop the communication between AC500 PLC and the CI52x devices and disconnect the Ethernet connection of the update PC and the CI Modbus device(s).
- Do not close the IP Configuration Tool during a firmware update and do not switch off a CI Modbus device during the firmware update.

During a firmware update the operation of the device(s) is stopped. After the update, all outputs are set to zero.

Erroneous firmware update

Error	Solution
Error 1: Package Timeout Due to a primitive firmware update protocol a fast and stable network connection is required. Otherwise the update packages cannot be transferred within the requested time and a timeout occurs.	Locate the PC on which the update is performed as near as possible to the stationed CI Modbus devices. Avoid network switches.
Error 2: Unable to read device status After the firmware update the IP Configuration Tool reads out the status of the updated device in order to check if the update was successful.	Rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 3: IP is not unique If more than one device hold the same IP address, a firmware update is not possible as the update command is IP based.	Correct the IP address, rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 4: Internal Error An internal error on the CI52x Modbus device occurred during the firmware update.	Rescan and repeat the update. If this doesn't work, power cycle the device and retry the update.
Error 5: Cannot connect to device The TCP communication is not sufficient for a connection. Increase the connection quality.	See Error 1: Package Timeout.

Signature check failed

After the selection of the firmware file (*.bin) a signature check is performed. If either the firmware file or the signature file is corrupt, the signature check fails. In the event of an erroneous signature check, perform the following steps:

- Ensure the signature file is stored in the same directory as the firmware file.
- Check the file names. The name of the signature file must be the same as the firmware file + attached ".sig”.

File names	Name of the firmware file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.bin Correct name of the signature file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.sig Wrong name of the signature file: c:\AC500\AC500_CI52x_Firmware_V3.2.8.sig

Indeterminate device firmware version

If the firmware version of the device cannot be determined, an error occurs. In this case, check that the device and the update PC are located in the same subnet and ping the device. If the ping is successful you can use the IP Configuration Tool to retrieve the device firmware version.

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

1.6.5.1.8 MultiOnlineChange tool

Introduction

The MultiOnlineChange tool/plugin for Automation Builder enables firmware update, download and online change of the same project to several PLCs.

Technically, the tool creates and executes a CODESYS 2.3 command file (*.cmd file) and several batch files, and deletes both types after an operation is finished.

The *.cmd file selects the configured gateway, loads the *.ri file for the PLC and performs the selected task (firmware update, online change, online access or multi download).

The batch files copy the *.ri file for each PLC to a specified directory.

Preconditions

- An Automation Builder project has been created and tested.
- A V2.3 CODESYS project in the master PLC has been created.
- 1 download has been issued to create all the files needed for the MultiOnlineChange tool.

The MultiOnlineChange tool may not work with the autoload option enabled in CODESYS V2.3.

Please check your current autoload option in CODESYS V2.3 in “ Project ➔ Options ➔ Load&Save” disable the option "Auto load".

> *The MultiOnlineChange tool relies on CODESYS V2.3 runtime system.*
> *As a consequence it will not work with AC500 V3 products.*

Usage of the MultiOnlineChange tool

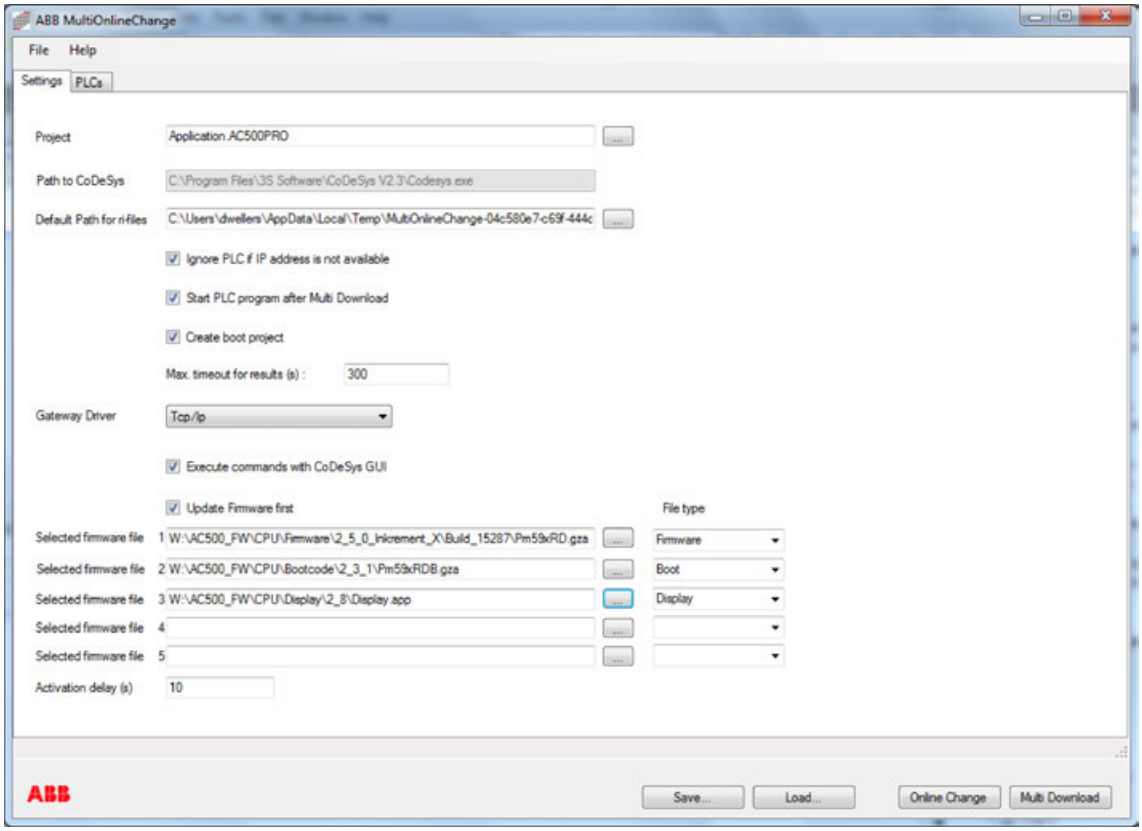
Overview

General options

- Defining of PLCs with IP address and a folder where the **.ri* file shall be stored for each PLC.
  The PLC on position one of the *"PLC"* tab list must be the master PLC. This is indicated by an orange frame. ↯ *" "PLCs" tab" on page 5798*
- Loading the given firmware file first to the PLC, and reboot it afterwards to start with the new firmware. ↯ *""Settings" tab" on page 5796*
- Performing *"Multi Download"* to download the project to all PLCs. ↯ *Chapter 1.6.5.1.8.4 "Performing a multi download" on page 5799*
- Changing the project in the master PLC. It is recommended to use the MultiOnlineChange tool to start CODESYS. ↯ *Chapter 1.6.5.1.8.6 "Editing the master PLC" on page 5802*
- Performing *"Online Change"* for all PLCs. ↯ *Chapter 1.6.5.1.8.7 "MultiOnlineChange" on page 5802*
- Showing the results of the download/online changes. ↯ *Chapter 1.6.5.1.8.5 "Verifying the download/online change success" on page 5800*

Step-by-step guide

***"Settings"* tab**



- In the *“Project”* text box, you can enter the project, which shall be distributed to all PLCs. This can either be done via opening a project file from the harddisk (in MOC stand-alone tool, see figure above) or via selection of the project, if the MOC was started from within Automation Builder.
- The field *“Path to CODESYS”* shows the path, where the file *CODESYS.exe* is available. This path is read from the registry. If the path cannot be retrieved from the registry, the tool will immediately exit after start-up.
- The field *“Default Path for *.ri files”* is an optional field. It is recommended that you define a folder here.

> *Due to compatibility reasons do not change the proposed path for Vista/Windows 7.*

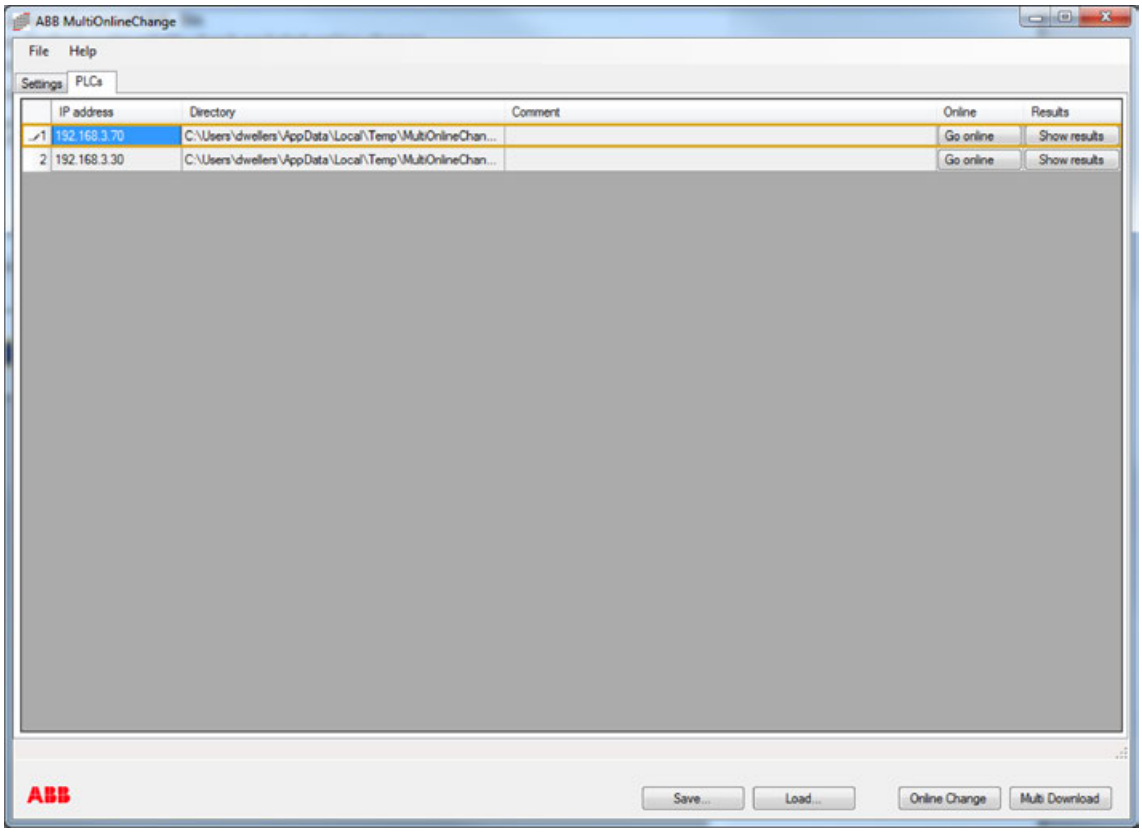
- If *“Ignore PLC if IP address is not available”* check box is enabled, then all functions will ignore any PLC, which does not reply to a *ping* command (the IP state of the PLC is shown by different background colours). In normal operation, this check box should be enabled. Otherwise the tool will try to access the defined IP in any case.
- If *“Start PLC program after Multi Download”* check box is enabled, the program is started on each PLC after a multidownload.
- If *“Create boot project”* check box is enabled, a boot project is created.
  If required, change the default setting for the *Max. timout for results*.
- The field *“Gateway Driver”* allows the selection of one of the following two options:
  – ABB TCP/IP Level 2 AC
  – TCP/IP
- If *“Execute commands with CODESYS GUI”* check box is enabled, CODESYS V2.3 UI will open up while the commands are executed. If the check box is disabled, all commands will be handled in background.
  “ Execute commands with CODESYS GUI” check box is only visible in the Stand-Alone tool.
- *“Update Firmware first”* check box shall be enabled, if the user wants to update the PLC firmware before sending any configuration to the PLC. The tool supports the update of up to 5 firmware files per PLC:
- The user has to enter the name of each firmware file in the text box *Selected firmware file* 1-5.
- The user has to select the type of any given firmware file (e.g. firmware, boot code or display) within the related box *File type*.
- The timeout to download one file and to reboot the PLC after all files have been downloaded may be extended in the box *Activation delay(s)*.

Buttons and status bar

- *Save*: After the definition of PLCs and settings is finished, all data can be saved in an XML file.
- *Load*: Load a previously saved XML file.
- *Online Change*: Perform a plausibility check and start online change.
- *Multi Download*: Perform a plausibility check and start multi download.

The status bar below the buttons shows information about the status of operations.

"PLCs" tab



The first row of the table defines the master PLC, which will be used for changing/testing the project before a download or online change is performed to the other PLCs listed in the table.

In the columns of the table, the following data is entered:

- Column *IP address*: Enter the IP address of the PLC.
- Column *Directory*: Define the directory, in which the **.ri* file will be stored. Each PLC must have its own directory. If you type a directory which does not exist yet, it will be created later during multi download.

> *If you have defined a default folder for *.ri files on the "Settings" tab, this column will be completed automatically by entering an IP address. In this case, the defined folder is the default folder and a subfolder with IP of the PLC (dots in the IP address are replaced by "-").*

> *Due to compatibility reasons do not change the proposed path for Vista/Windows 7.*

- Column *Comment*: The entered text has no influence on the operation of the MultiOnline-Change tool, but will be stored in the data XML file.
- Column *Go online*: By pressing this button CODESYS is started. You can log in to the PLC defined in this row.
- Column *Show results* will be explained in ↬ *Chapter 1.6.5.1.8.5 "Verifying the download/online change success" on page 5800*.

Each IP address and each directory must be unique. Do not define the same IP or directory twice. If you do so, you will get an error message when you try to perform an action.

The context menu of the table can be opened by right-click and contains the following functions:

- *Add single PLC line*: Adds 1 row to enter a new PLC.
- *Delete selected PLC lines*: Deleted selected rows.

- *Add IP range*: A range of IP addresses is added by defining the start IP address, the end IP address and the increment. E.g. if you select start IP 192.168.3.1 and end IP 192.168.3.10 and increment 3, the following IP addresses will be created:
  – 192.168.3.1
  – 192.168.3.4
  – 192.168.3.7
  – 192.168.3.10
- *Clear Data*: Deletes all entries in the table.
- *Check IPs*: Performs a check, using the *ping* command to see, if the defined PLCs are available. After the check, the IP addresses of all available PLCs are shown with green background, unavailable PLCs with a red background. White background means *state unknown*.

Performing a multi download

After you made settings and entered PLCs you want to download the project to, you can start a multi download by clicking the button Multi Download.

Important!

A multi download should be the first action you perform (after the configuration) when you start using the MultiOnlineChange tool. The multi download builds necessary infrastructure by performing the following tasks:

– *It creates all directories defined in the PLC table (this is the first and only time when folder are created).*
– *It copies the *.ri files to the folders.*

Without folder and the corresponding *.ri files in them, online access and online change are not possible. In fact, an online change will be executed even if folders or files are not available, but any PLCs without a defined folder or *.ri file will be skipped.

An error message will be shown if there are:

- Rows without IP address
- Rows without a defined directory
- Duplicate IPs or directories

During the multi download, no other operation is possible and the tool can no longer be operated until the operation is finished.

Multi download (as well as online change) might take a long time, depending on the number of PLCs. Each PLC requires approximately 15 seconds (without firmware update), i.e. 100 PLCs would require about half an hour. Therefore the timeout parameter in the tabpage Settings has to be adapted according to the project dimension.

Important!

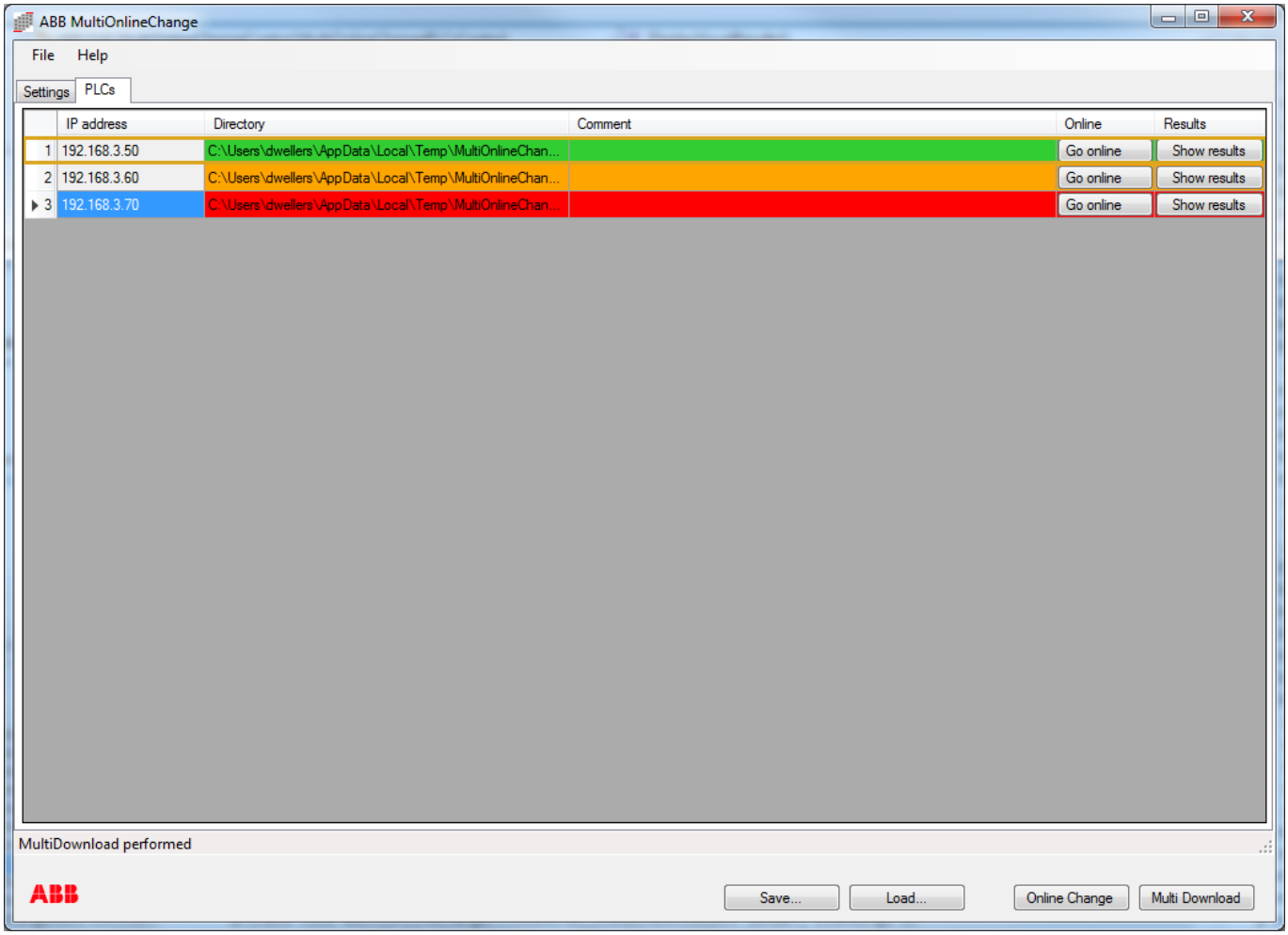
*If you change the IP addresses of a PLC or the folder where the corresponding *.ri file is stored, a multi download should be performed again to rebuild the infrastructure (folders etc.).*

Verifying the download/online change success

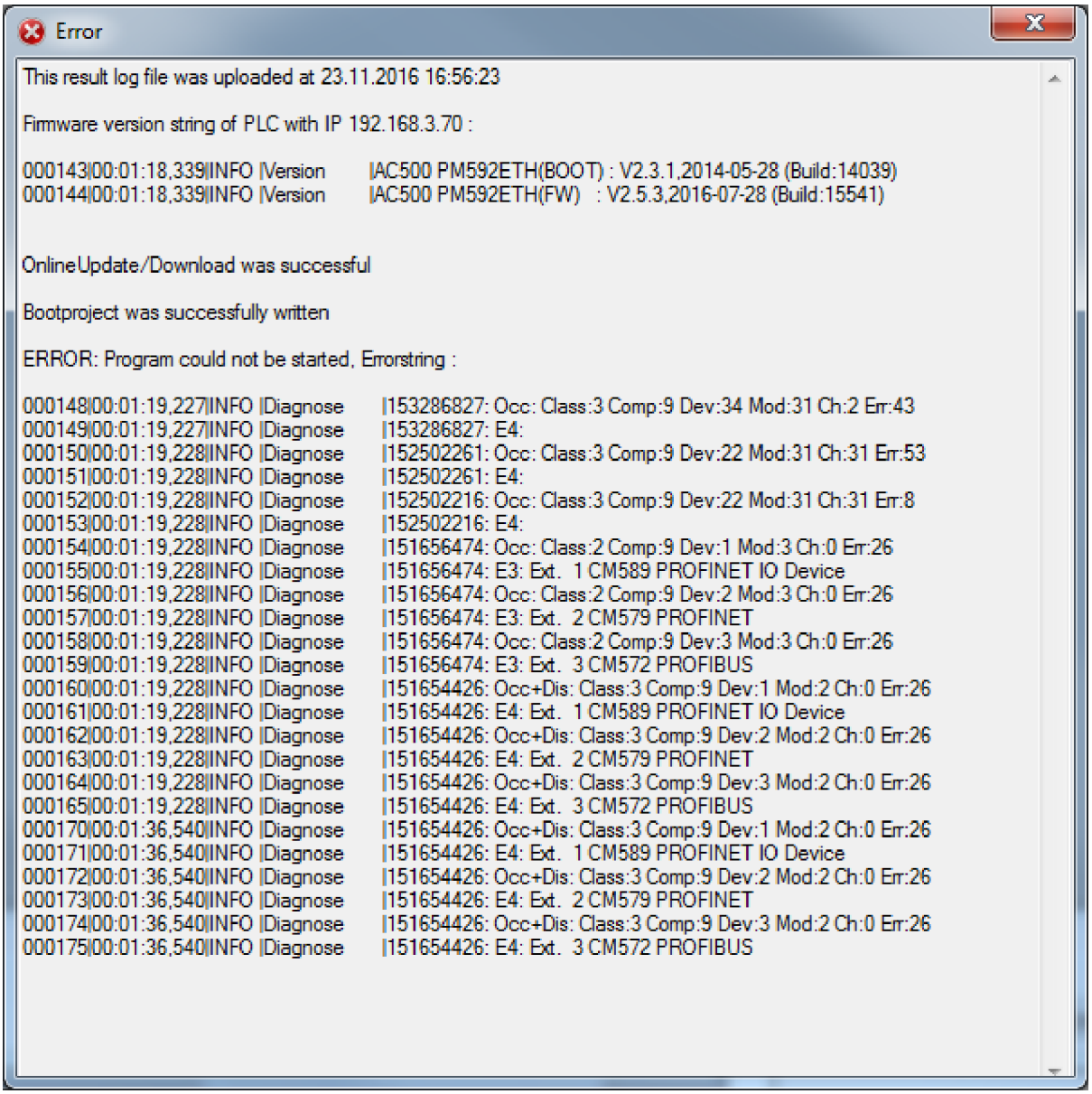
The verification is done via upload of a *.log file from each PLC where all operations are logging their success or failure. For diagnostic purposes these *.log files are not deleted afterwards. They are stored in the installation directory of the Automation Builder.

After starting the download/online change the PLC tab is active and will indicate the success or failure of the operation at the end by using different colored lines:

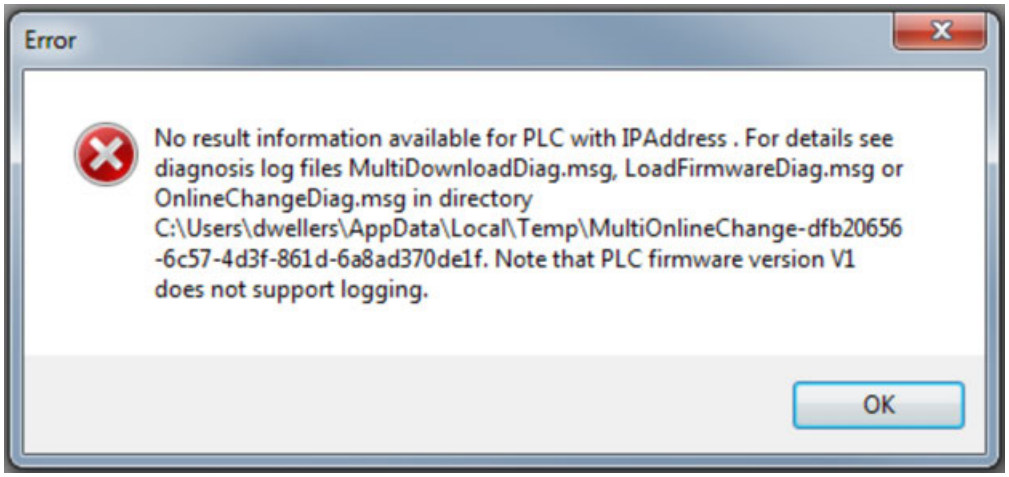
Example view after a run:



- GREEN: All successfully done.
- ORANGE: The download was done, but there were diagnostic errors, e.g. configuration faults or run time errors.
  If the button *"Show Results"* is pressed the contents of the selected PLC log file is displayed:



- RED colored line: Perhaps the PLC is offline, the IP address is wrong or the project has a wrong target (so download will be rejected immediately). In these special cases there is no detail information available because a log file could not be uploaded.
  If the button *"Show Results"* is pressed the following window is displayed:



- "MultiDownloadDiag.msg" is the log file for multi download operation.
- "LoadFirmwareDiag.msg" is the name of the log file of the firmware update operation.
- "OnlineChangeDiag.msg" is the name of the log for the online change operation.

Editing the master PLC

After downloading the project to all PLCs, you can change the project. To do so, click the "Go Online" button of the first row in the PLC table. CODESYS will start with the defined project and log in.

You can now log off, change the project, compile, log in, perform an online change etc. as usual. When you have finished, close CODESYS.

The tool cannot be operated during master edit.

MultiOnlineChange

If you made no breaking changes to your project (and thus only altered the program in the master PLC with online change "" no clean, rebuild, etc …), you can now perform an online change on all connected PLCs.

To do so, click the button *"Online Change"*. During the operation, the MultiOnlineChange tool is deactivated.

Online access

You can log in to any PLC by clicking the button *"Go Online"* in the table row of the PLC. You can log in in more than 1 PLC, as long as you do not log-in to the master PLC. If you do so, the tool operation is deactivated again (see *Chapter 1.6.5.1.8.5 "Verifying the download/online change success" on page 5800*). If the corresponding *.ri file is not available, online access cannot be performed and an error message is shown.

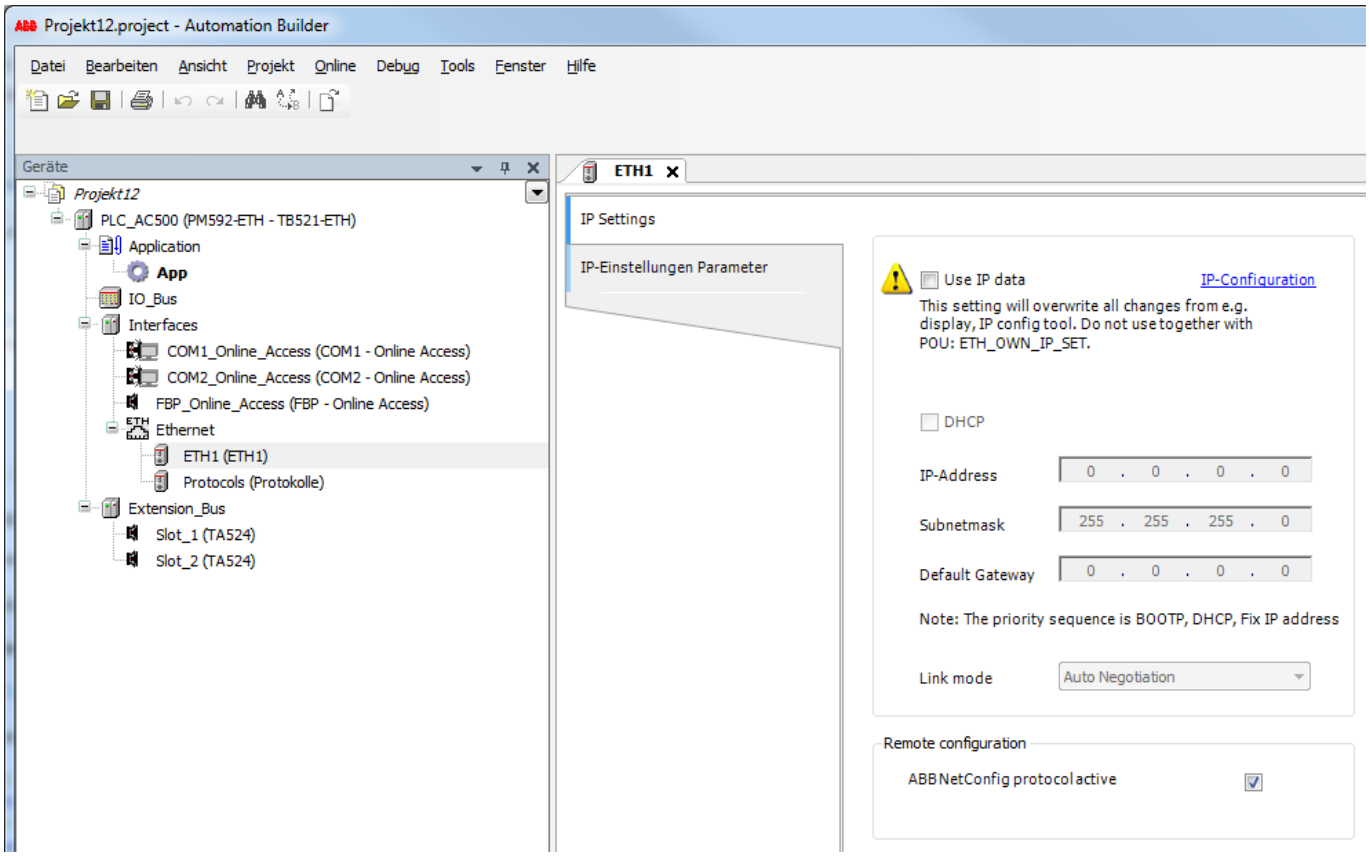
Important!

Do not edit any PLC except the master PLC!

Limitations in V2.x

Do not use fixed IP data!

In the IP settings parameter dialog of an Ethernet Communication Module there is an option to use the given IP data. If this option is used in the master project the MultiOnlineChange tool is not capable of loading the project to other PLCs.



Modify wait time of MultiDownload/Online change

Depending on the network infrastructure the MultiOnlineChange (MOC) might run into errors due to timeouts. The user will not be informed about these timeouts directly, but each timeout might result in errors.

Examples:

“Logfile cannot be read” (red line in PLC list even though the download/change was successful).

“Online Change not possible” (PLC changes state from “run” to “stop” during Online Change, because the wait time for loading the project was not sufficient).

To avoid these timeouts and the resulting errors the user can extend the wait times of the MultiOnlineChange plug-in/tool. These times can be adapted in the template XML file `"CmdTemplate.xml"` by increasing the values in the `delay` lines. The file has the following content:

```
<CmdTemplates>
...
<DownloadPLC>
  device instance "MyDriver"
  device parameter "Address" @@@ipAddress
  device parameter "Port" @@@port
  device parameter "Motorola byteorder" Yes
  onerror continue
  delay 1000
  query off no
  online login
  delay 5000
  online logout
  waitevent ONL_LOGGEDOUT
  query off no
  online login
```

```
  delay 3000
  online filewrite @@@iniFileForLogging
  waitevent ONL_FILEWRITTEN
  online filewrite @@@uniqueIdent
  waitevent ONL_FILEWRITTEN
  online logout
  waitevent ONL_LOGGEDOUT
  query off ok
  online login
  waitevent ONL_PROGRAMLOADED
  waitevent ONL_FILEWRITTEN
  delay 3000
  @@@bootproject online bootproject;
  waitevent ONL_FILEWRITTEN
  delay 10000
  online logout
  waitevent ONL_LOGGEDOUT
  query off no
  online login
  delay 15000 @@@enable online run
  delay 20000 online fileread @@@uploadLogFile
  waitevent ONL_FILEREAD
  delay 5000 online logout
  file save
  delay 1000
  system "@@@handleRiBatch"
  delay 2000
</DownloadPLC>
...
<OnlineChangePLC>
  project loadcompileinfo "@@@plcDirectory\@@@projectName.ri"
  device instance "MyDriver"
  device parameter "Address" @@@ipAddress
  device parameter "Port" @@@port
  device parameter "Motorola byteorder" Yes
  delay 10000 onerror continue
  query off no
  online login
  delay 10000 online filewrite @@@iniFileForLogging
  waitevent ONL_FILEWRITTEN
  online filewrite @@@uniqueIdent
  waitevent ONL_FILEWRITTEN
  online logout
  waitevent ONL_LOGGEDOUT
  query off ok
  delay 1000
  online login
  waitevent ONL_PROGRAMLOADED
  delay 3000
  @@@bootproject online bootproject
  waitevent ONL_FILEWRITTEN
  delay 10000
  online logout
  waitevent ONL_LOGGEDOUT
  query off no
  online login
  delay 15000
  online fileread @@@uploadLogFile
  waitevent ONL_FILEREAD
  delay 3000 online logout
  file save
```

```
  delay 1000
  system "@@@handleRiBatch"
  delay 2000
</OnlineChangePLC>
</CmdTemplates>
```

Each delay defines a wait time in milliseconds, which can be adapted by the user. As an example some of the delays are emphasized above:

- The three emphasized delay times in tag `<DownloadPLC>` can be increased to ensure that the MOC waits long enough for the log-file to be accessible after reboot before reading it.
- The four emphasized delay times in tag `<OnlineChangePLC>` can be increased to ensure that MOC waits long enough for loading the project, logging in to PLC and reading the log-file.

1.6.5.1.9 Embedding of AC500 V2 libraries

Situation

With new Automation Builder releases often new AC500 V2 system libraries are being installed.

In difference to AC500 V3 libraries, the AC500 V2 libraries are not versioned. An update of the Automation Builder might lead to the situation that a login to the PLC is only possible after a rebuild and with an online change – although the application has not been changed and the user is working with a previous version profile.

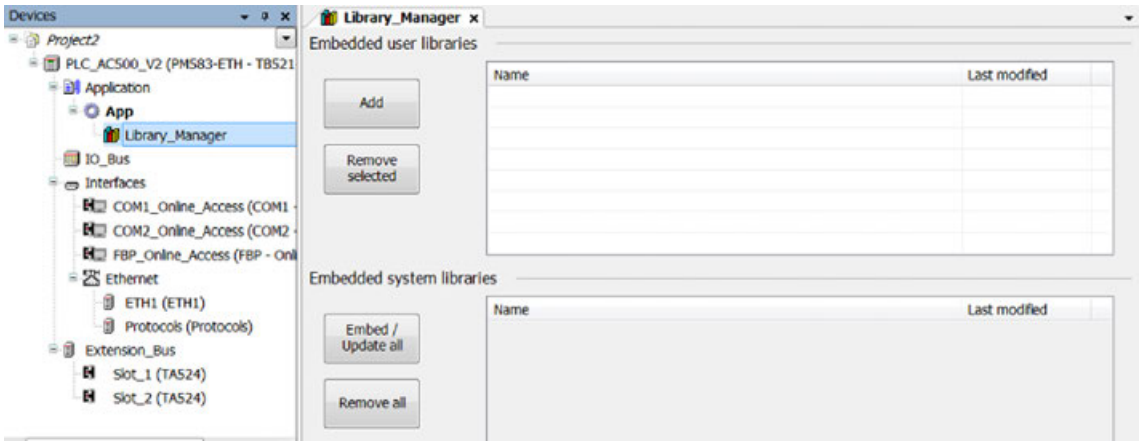
Avoidance

To avoid this situation, the AC500 V2 libraries can be embedded into the Automation Builder project.

> *The feature "Embedding of AC500 V2 libraries" is available as of Automation Builder 2.1.1.*

Embed user defined libraries

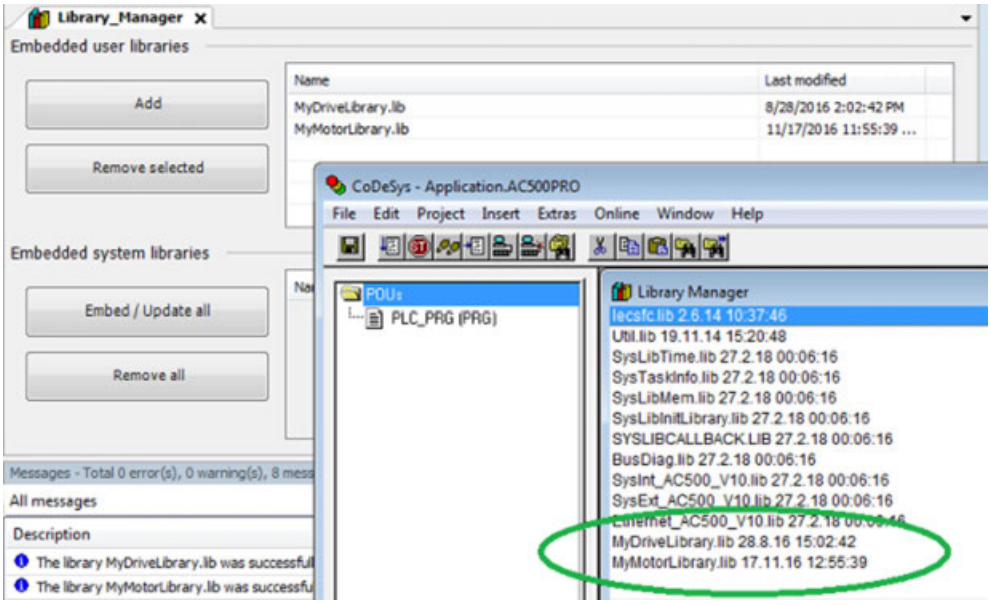
1. Right-click on the *"App"* node of your *"PLC_AC500_V2 <...>"* project and click *"Add object"*.
2. Select *"Library Manager"* in the appearing window and click *[Add object]*.
   ⇨ *"Library_Manager"* appears below your *"App"* node
3. Double-click on the *"Library_Manager"* object in the device tree.
   ⇨ The Library manager editor appears.



4. Click the *[Add]* button and embed your user defined libraries.

Success or failure of the embedding will be displayed in the message window below.

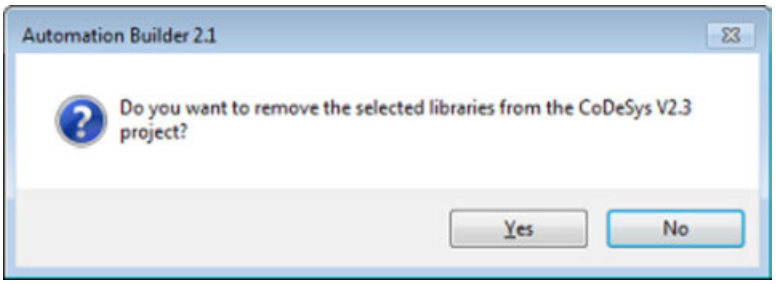
In case of success the libraries are automatically added to the corresponding CoDeSys V2.3 project



Remove user defined libraries

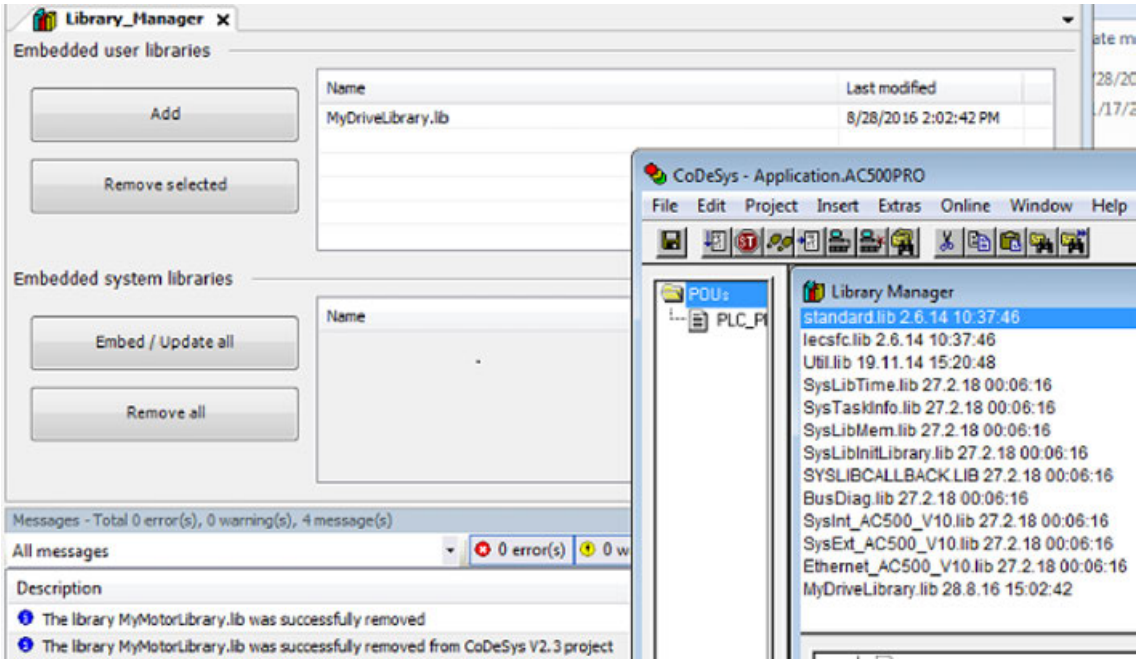
1. Select your libraries to remove (one or more libraries can be removed) in the library manager editor and click the *[Remove selected]* button.

   ⇨ A confirmation prompt appears.



2. Click *[Yes]*

   ⇨ Information messages are displayed in the message window and the selected libraries are removed from the CoDeSys V2 project.



Embed system libraries

- Click *[Embed / Update all]* in the Library Manager editor.

> *Via the [Embed / Update all] button all the system libraries are embedded into the Automation Builder project.*
>
> *The system libraries are not added into the CoDeSys V2 project.*
>
> *The CoDeSys V2 project will use automatically (for the referenced system libraries) these embedded system libraries instead of the system libraries of the latest Automation Builder installation.*

Remove system libraries

- Click *[Remove all]* in the Library Manager editor.

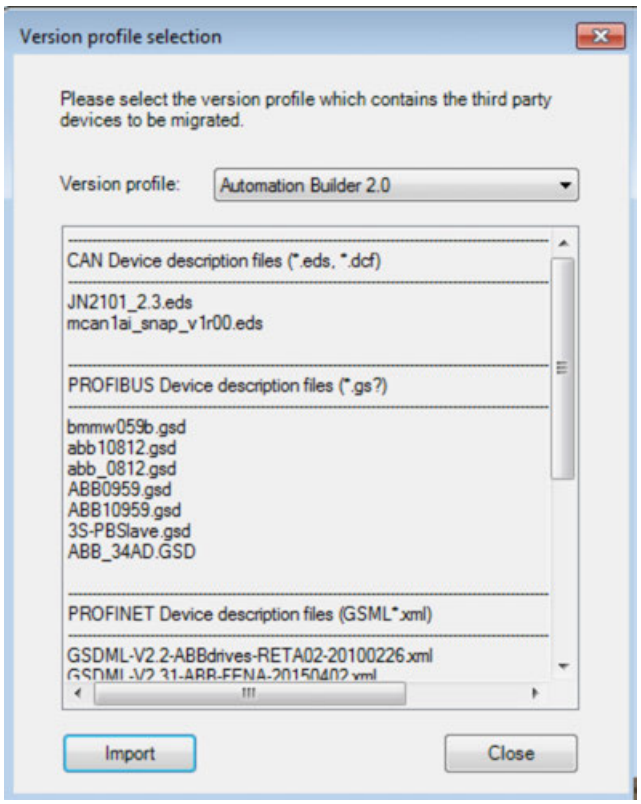
> *Via the [Remove all] button the system libraries will be removed from the Automation Builder project.*
>
> *The system libraries won't be removed from the CoDeSys V2 project.*
>
> *The CoDeSys V2 project will use automatically (for the referenced system libraries) the system libraries of the newest Automation Builder installation.*

1.6.5.1.10 Migration of third party devices

After an update of Automation Builder the device repository contains only ABB devices. The third party devices which were installed into previous versions of Automation Builder are not automatically installed in the newest version profile. This has to be triggered by the user.

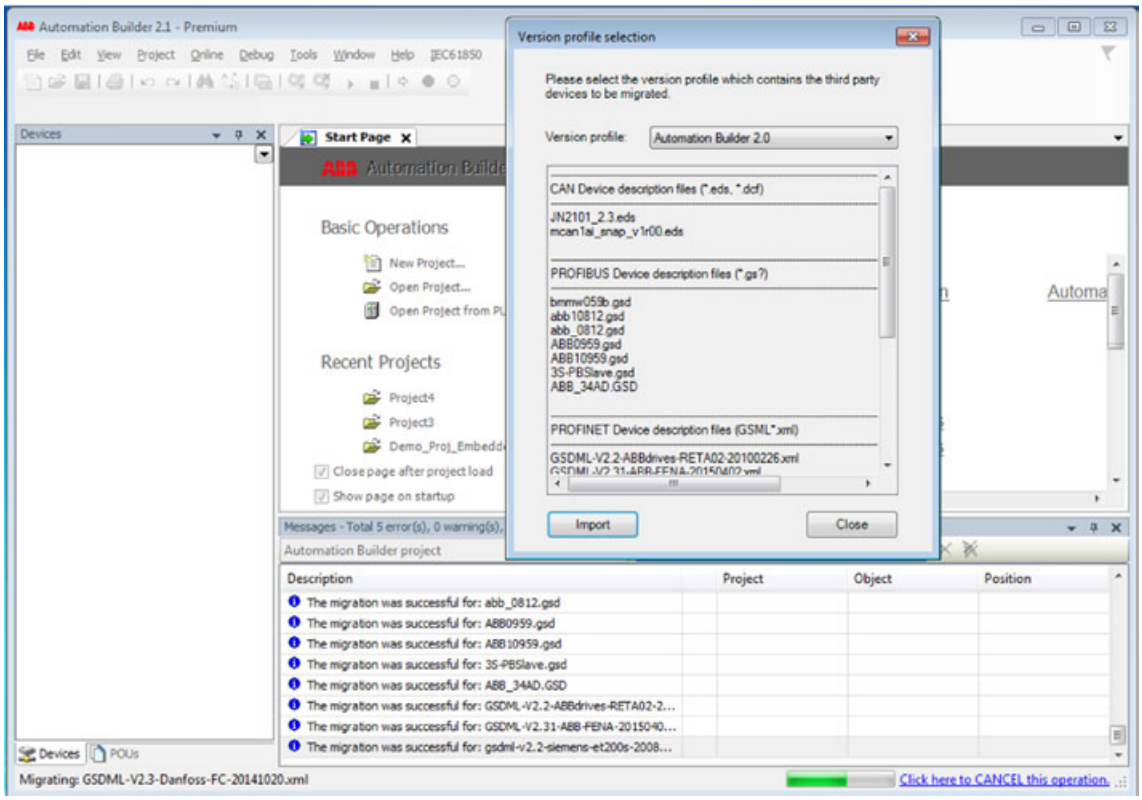
> *The feature "Migrate third party devices" is available as of Automation Builder 2.1.1.*

1. Click *"Tools"* in the main menu of Automation Builder.
2. Click *"Migrate third party devices"* in the drop-down list.
   ⇨ The window *Version profile selection* appears.
3. Select a version profile in the drop-down list containing previous Automation Builder / Control Builder Plus profiles. The active profile does not appear in the list.
   ⇨ After selection of a previous version profile, all the third party devices which have been installed inside this version profile are listed.



It is not possible to select or deselect some third party devices. Importing will affect all the third party devices which are listed in the list view.

4. Select *[Import]*.



⇨ During the migration the message window displays success or failure of device migration.

> *In case of failure during the migration the affected third party device description has to be installed manually via main menu "Tools ➔ Device Repository ➔ Install".*

In the status bar, the third party device which is on *Migrating:* <...> is displayed on the left side.

The import operation can be cancelled by clicking the *"Click here to CANCEL this operation"* link on the right side of the status bar. This becomes effective when the migration of the just migrating third party device is finished.

5. To close the dialog select the *[Close]* button of the *Version profile selection*.

1.6.5.1.11 Advanced IO device handling

Automation Builder provides the Advanced IO Device Handling feature for configuring identical IO device types at multiple instances.

This feature is supported by the following commands that works with IO devices only.

- Generate DUT
- Map to Existing DUT
- Release DUT mapping

These commands work on individual nodes and on CI (communication interface) level nodes.

Generating DUT

Each device generates two DUTs. One for the input and one for the output. Some devices contain only input or output type. In such cases, the device generates only one DUT of the relevant type.

- Right-click on the desired IO device and select *"Generate DUT"* to generate a DUT for an IO device.

The following example shows how to generate DUTs at CI level node.

- In the device tree, right-click on a master node such as PNIO_Controller and select *"Generate DUT"* to create DUTs for the child nodes.
- The DUTs of child nodes are generated in *"Application ➔ App ➔ IO_Device_Generated_Items"* folder.
- Generated DUT considers channels with BYTE datatype as members. If channels with BYTE datatype are not present in the given hierarchy, it adds the members with another higher datatype.
- Channels with BOOL datatype are not considered.

Mapping to existing DUT

This command is enabled for the IO device when the IO device is not mapped and when DUTs of matching size (calculated based on device channel list) are available in *"Application ➔ App ➔ IO_Device_Generated_Items"* folder.

1. Right-click on an IO device and select *"Map to Existing DUT"*.

   ⇨ Enter Instance Name dialog is displayed.

2. Enter the instance name which satisfies IEC naming validations and unique name in global scope.
3. Click *"OK"* to create a global variable associated with the mappings in DI (PRG).

   If you want to view mapped instances, double-click *"DI (PRG)"*.

With the 'Map to Existing DUT' command:

- Any device can be mapped only to one input DUT and one output DUT. If you have already mapped an input DUT, only the output DUT is shown in the options list and vice-versa.
- Mapping is also supported at CI level nodes. To create global variables for CI level nodes, the address of the first child is considered.

Releasing DUT mapping

This command is enabled on an IO device only when an IO device is mapped either to input, output or both DUTs. You can use this command to release (or revert) mappings and to delete global variables created during 'Map to Existing DUT'.

Right-click on an IO device and select *"Release DUT Mapping"*. The mapped DUT instance is deleted.

Using DUT variables in CODESYS application

1. In the Automation Builder project, double-click *"Application"* to launch CODESYS application.

   ⇨ CODESYS application is launched. CODESYS application contains mapped DUT instances.

2. Double-click *"PLC_PRG"* to create DUT variables.
3. Add DUT variables based on mapped DUTs.

For further information on mapping DUTs, see section *Chapter 1.6.5.1.11.2 "Mapping to existing DUT" on page 5810*.

For example, in the PLC_PRG, add analog I/O and digital I/O. If you insert a dot at a position where an identifier should be inserted, then a selection list is open, offering all the input and output variables which are found in the project.

After adding DUT variables, rebuild the program in CODESYS application using *"Project ➔ Rebuild"*.

Support for CI level node

The user can create DUTs for the entire hierarchy of CI level node (for example, IO_BUS), by right-clicking on the desired CI level node and by selecting *"Generate DUT"*. Further, all the DUTs are generated in *"Application ➔ App ➔ IO_Device_Generated_Items"* folder.

- The command generates DUT for the node itself and also for all child nodes.
- The DUT generated for the CI level node contains generated DUTs for the child nodes as their members.
- For every execution, the command checks, if any new child node is added and generates DUT.

If you delete child nodes in CI level node (for example, IO_BUS), the DUTs generated for these child nodes are not deleted automatically. You should delete the DUTs manually in the *"Application ➔ App ➔ IO_Device_Generated_Items"* folder if desired.

Configuration check

Configuration check for size is enabled to ensure that all devices are mapped with DUTs of the correct size. In case of any changes in the mapped DUT, configuration check verifies the size of the DUT. If it fails, an error message is displayed in Automation Builder messages window and does not allow to launch the application. This check can be performed in *"Create configuration data"*.

1.6.5.2 PLC devices and components

1.6.5.2.1 Device repository

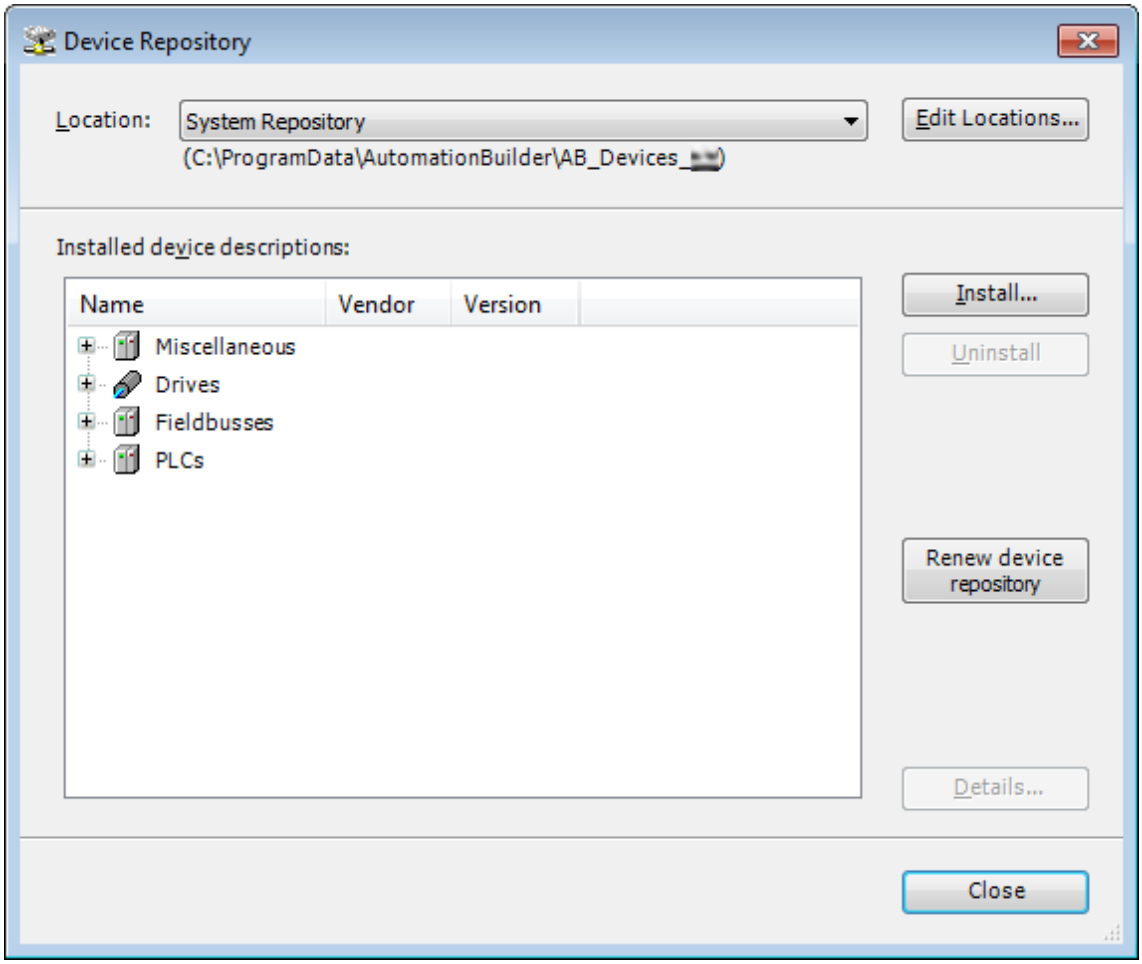
The Device Repository of Automation Builder manages the pool of devices that can be used in the PLC configuration.

You install or uninstall devices in the *"Device Repository"* dialog box. The system installs a device by reading the device description files, which define the device properties for configurability, programmability, and possible connections to other devices.

You can use the devices provided in the device repository by adding them to the device tree of your project.

Dialog device repository

1. Click *"Tools ➔ Device Repository"*.
   ⇨ The *"Device Repository"* dialog box opens.



[Edit Locations]: Changes the default repository location. The devices can be managed at different locations.

[Install] / *[Uninstall]*: Installs or uninstalls devices.

[Renew device repository]: Updates the device list, e.g. after uninstallation of a device.

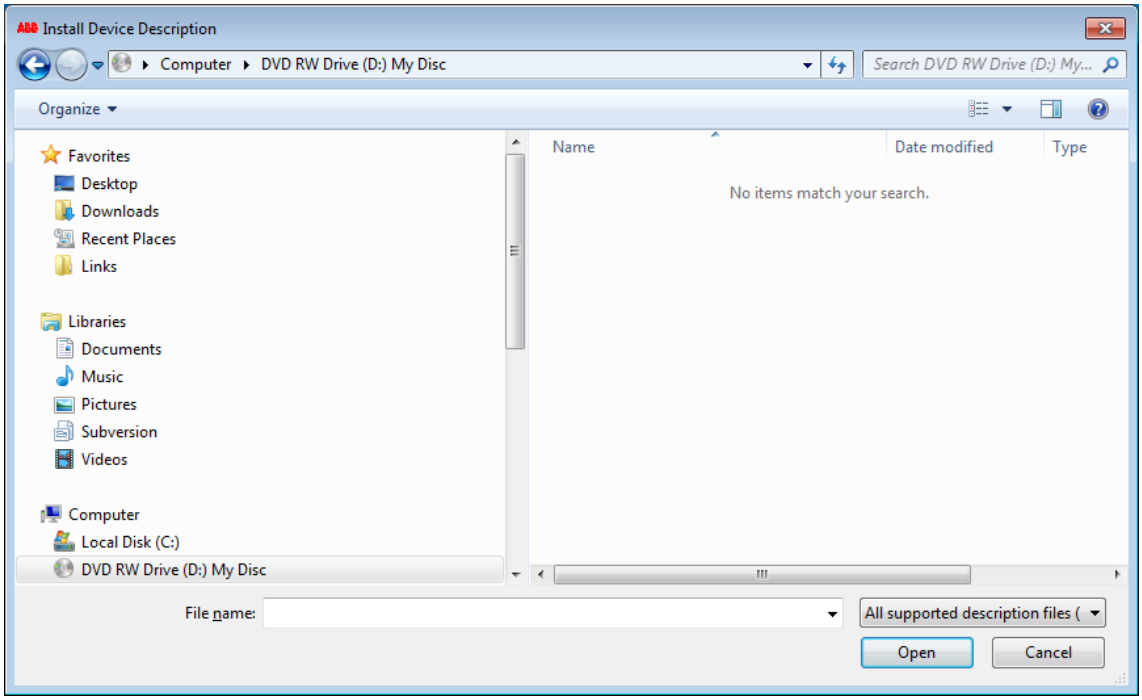
[Details]: Provides technical details on the selected device.

2. Select the install location. *"System Repository"* is set by default.

Installing devices

The device repository cannot be changed manually, e.g. by copying or deleting files. Use always the Device Repository dialog to add or remove devices.

1. Click *[Install]* and select the appropriate file format.
   ⇨ The *"Install Device Description"* dialog box opens.



2. Select the file path of the device description.
3. Select the file type filter of the required device description.
   ⇨ All device descriptions of the selected file type are listed.
4. Select the required device description and click *"Open"*.
   ⇨ Automation Builder adds the device description to the matching category of your device repository.

   If errors occur during installation (for example, missing files that are referenced by the device description), then Automation Builder displays them in the lower part of the device repository dialog box.

> *During the installation the device description files and all additional files referenced by that description will be copied to an internal location. Altering the original files will have no further effects to an internal location.*
>
> *The changes take only effect after reinstalling the corresponding device(s). The version number shown in the information section of the device should be verified.*

Uninstalling devices

Select the device you want to remove and click *[Uninstall]*.

The device is removed from the list.

> *Uninstalled devices which are used in existing projects are indicated by the symbol ❓. The device will not be configured properly.*

1.6.5.2.2 PLC start-up

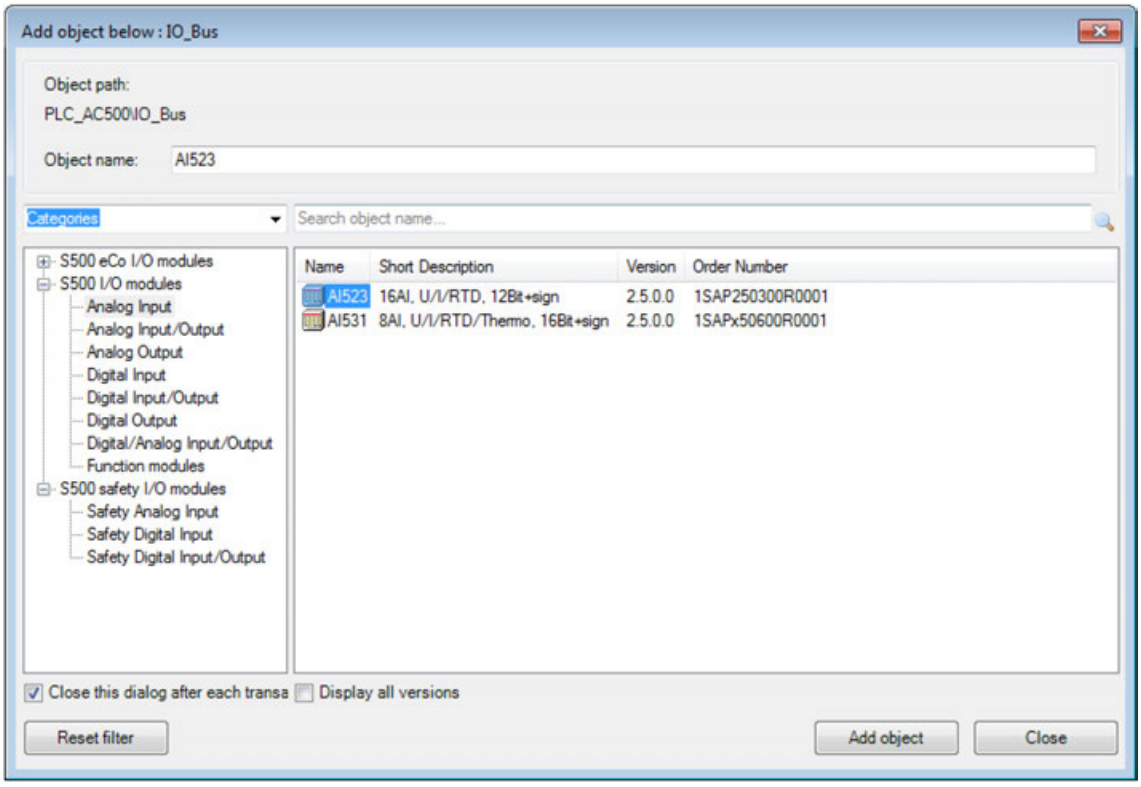
Connection of devices

All installed devices that are available in Automation Builder are listed in the ↳ *Chapter 1.6.5.2.1 "Device repository" on page 5811*.

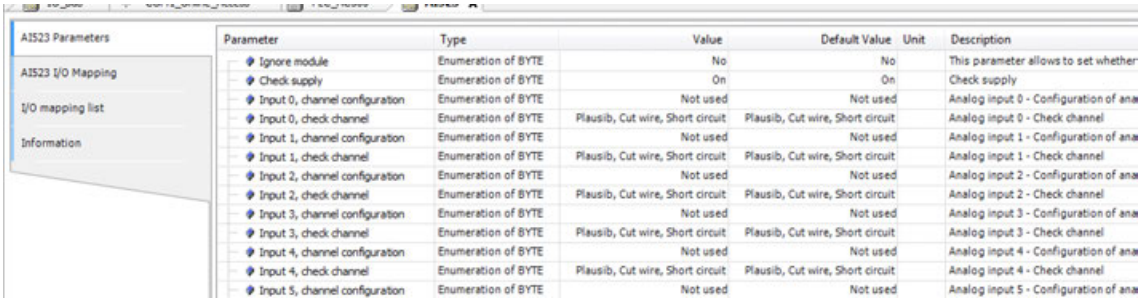
Configuring devices

Modify your Automation Builder project by adding device objects. Preset items can be replaced in the same way.

1. In the device tree, right-click an item node. Select *"Add object"*.



2. Select the desired object and click *[Add object]*.
3. Double-click the new object in the device tree to configure the device settings. Depending on the selected item different configuration tabs are available.



Symbolic names for variables, inputs and outputs

The IEC naming rules are not checked during input in Automation Builder.

Input and output mapping

Devices with I/Os provide an I/O Mapping tab in their configuration editor where the available I/O channels can directly be mapped to a global variable.

The corresponding variable declarations are automatically created in a global variables object in a subfolder of the Global Variables section in the CODESYS project.

All available I/O channels can easily be assigned to a variable.

Channels

Variable	Mapping	Channel	Address	Type	Unit	Description
		Digital inputs I0 - I15	%IW0			
w_Input_IW0		Digital inputs I0 - I15	%IW0	WORD		Digital inputs 0-15
		Bytes	%IB0			
		Digital inputs C16 - C31	%IW1			
		Digital outputs C16 - C31	%QW0			
Fast counter						

The variable is automatically added to the Global Variables in the CODESYS project after recreating the configuration data *Chapter 1.6.5.4.1.1 “Creating configuration data” on page 6196*.

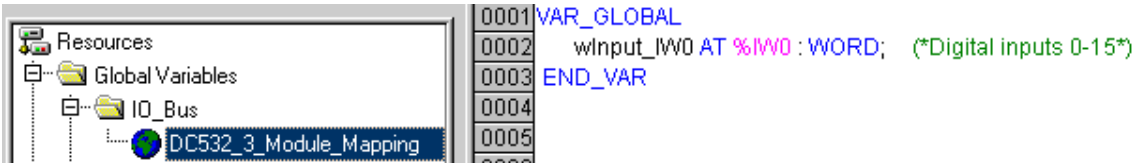
AC500 uses Motorola Byte Order (Big Endian).

The numbers in column Channel correspond to the channel numbers only and not to the bit position inside the WORD variable.

Only entries with a data type set in column "Type" can be mapped. These entries can be expanded to show the available I/O channels.

If the project has been imported from a previous Automation Builder version, all variables should be checked to avoid inconsistencies concerning the I/O mapping.

The variable is automatically added to the global variables in the CODESYS project after (re)creating the configuration data:



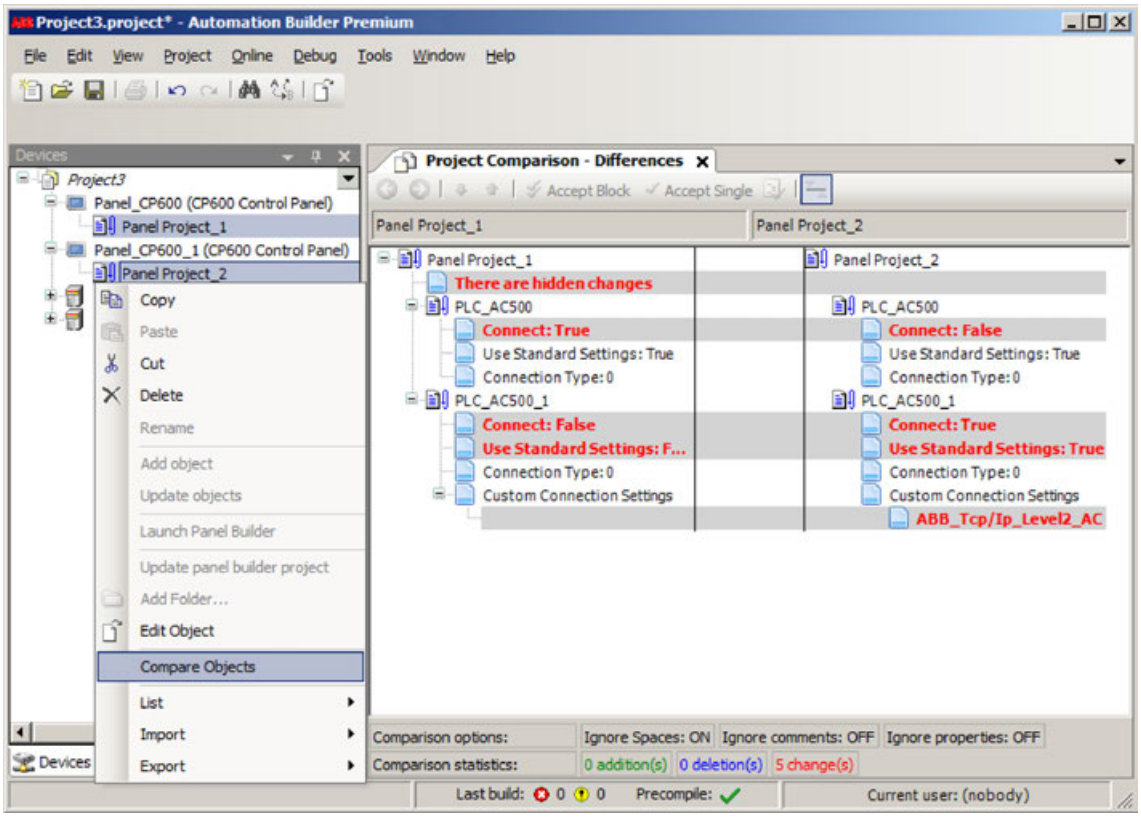
An additional GVL (Global Variables List) can be created and transferred to CODESYS V2.3. Editing of lists created in CODESYS V2.3 is not possible.

Update of AC500 devices

Perform a firmware update to update AC500 V2 devices. *Chapter 1.6.5.1.7 “Firmware identification and update” on page 5786*

Comparing objects

To compare similar objects within a project (such as the project configuration) select both objects. Right-click and select **Compare Objects** to see the differences.



IP settings

Configuration of the IP settings with the LED display

The IP settings for the PLC can be set directly on the processor module via keypad and LED display.

See *Chapter 1.6.4.1.5.4.3 “Configuration” on page 5428*.

Configuration of the IP settings with the IP configuration tool

The IP address for AC500 devices can be set or changed in Automation Builder using

- the IP configuration tool which is described in the following.
- the 'Communication Settings'. *Chapter 1.6.5.2.2.2.3 “Configuration of communication via Ethernet (TCP/IP)” on page 5829*

As an alternative the IP address can be changed at the hardware device itself. *Chapter 1.6.4.1.5.4 “Description of the function keys” on page 5426*

The IP configuration tool:

The IP configuration tool can be used

- to set or change the IP address of devices.
  Chapter 1.6.5.2.2.2.2.2.2 “Changing the IP address” on page 5821
- to scan the network for available hardware devices.
  Chapter 1.6.5.2.2.2.2.2.1 “Network scan” on page 5819

- to update the firmware of devices.
  This functionality is only supported if the IP configuration tool is used stand-alone.
  ↳ *Chapter 1.6.5.2.2.2.2.2.3 “Firmware update” on page 5822*
- to activate certain functionality on hardware devices.
  This feature is only available on AC500 V3 devices.
  ↳ *Chapter 1.6.5.2.2.2.2.2.4 “Blink functionality” on page 5826*

The IP configuration tool is part of Automation Builder and can be called via *“Tools ➔ IP-Configuration”*.

Further the IP configuration tool can be used stand-alone without an Automation Builder application running. The stand-alone variant requires a separate installation via the Installation Manager ↳ *Chapter 1.6.5.2.2.2.2.1 “Stand-alone installation” on page 5817*.

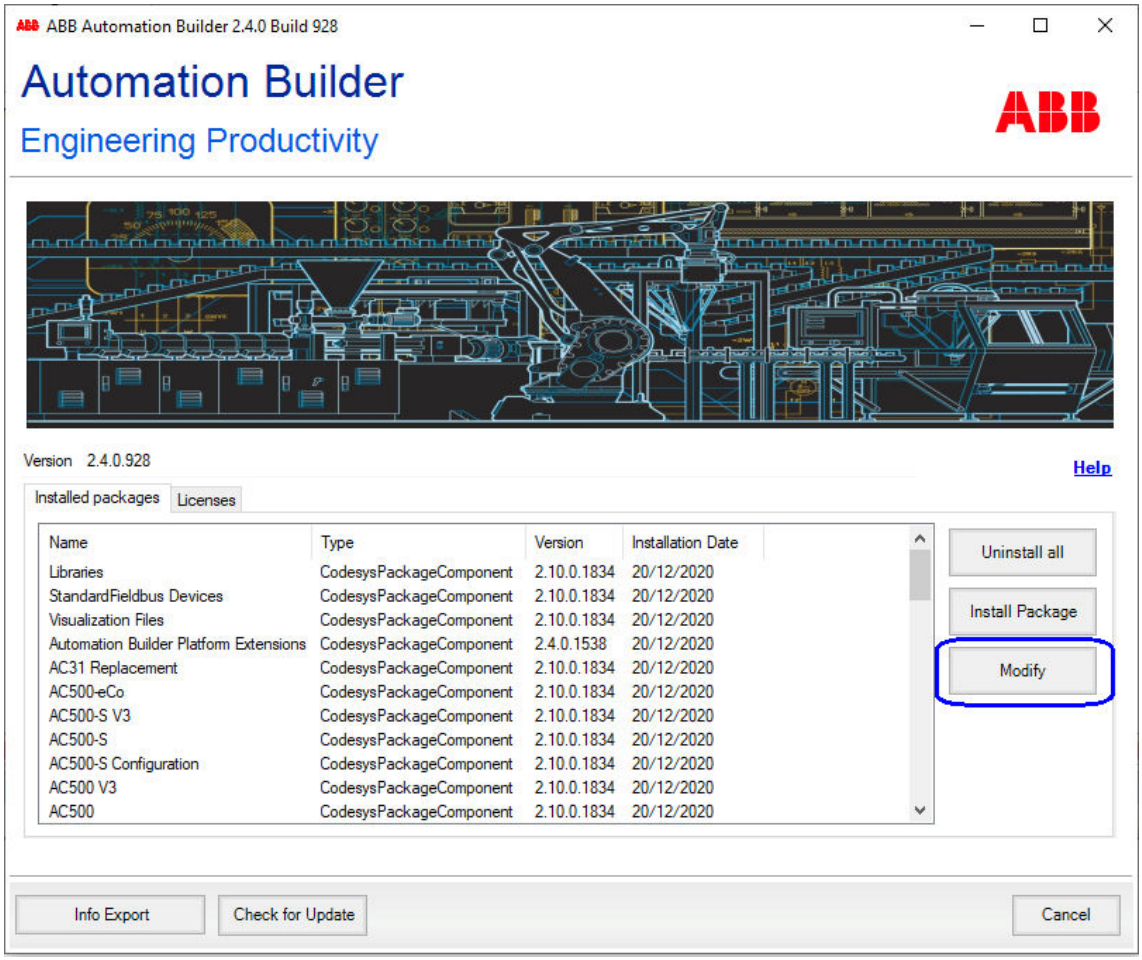
After the installation, the IP configuration tool is started via .exe file / desktop icon.

Some functionality is only supported if the IP configuration tool is used stand-alone, e.g. for firmware updates for communication interface devices.

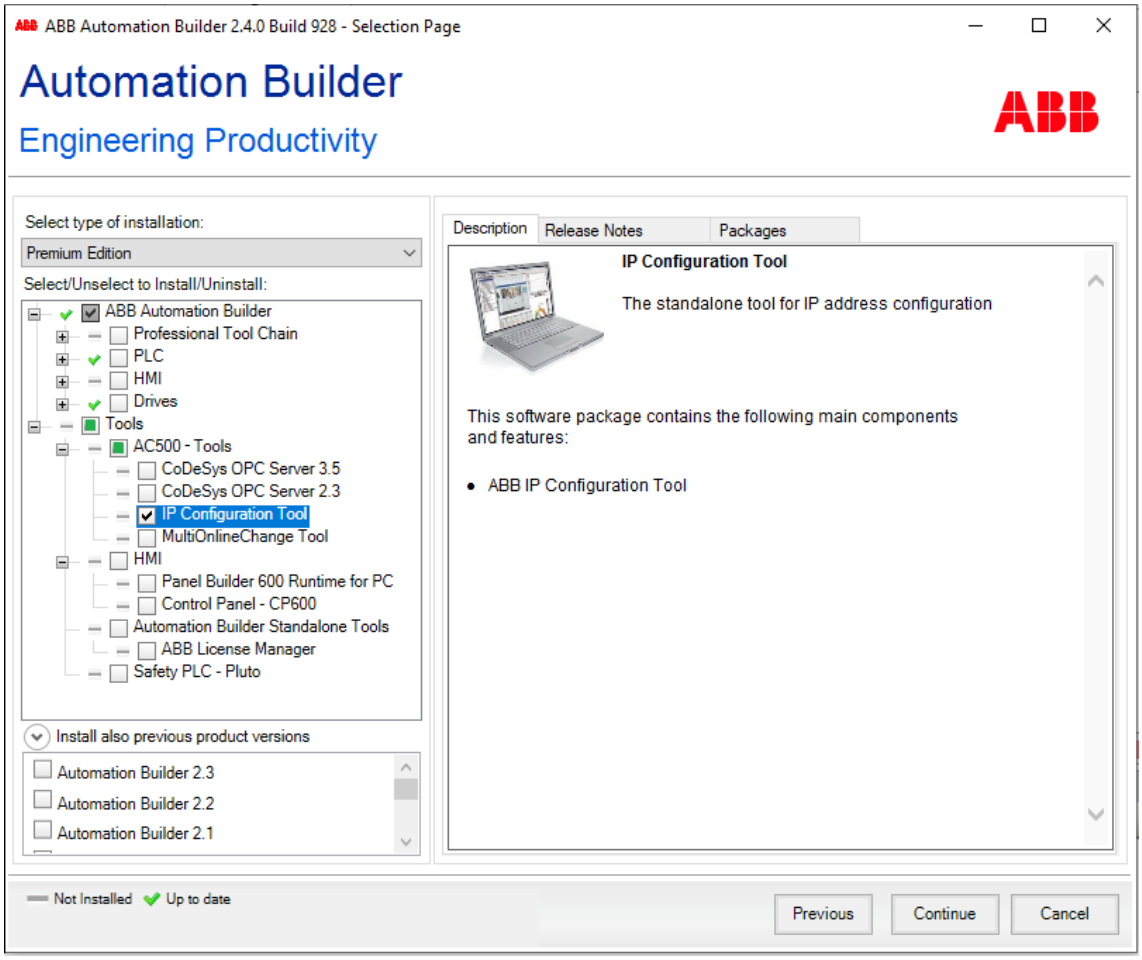
Stand-alone installation

The IP configuration tool is part of Automation Builder and can be called via “Tools ➔ IP-Configuration”. A separate installation is only required if the IP configuration tool shall be used stand-alone.

1. Open the Installation Manager in Automation Builder: *"Tools ➔ Installation Manager"*.
2. Close all other instances of Automation Builder as only one instance of the program can be executed at a time.



3. Click *"Modify"* and select the *"IP Configuration Tool"* from the structure tree.



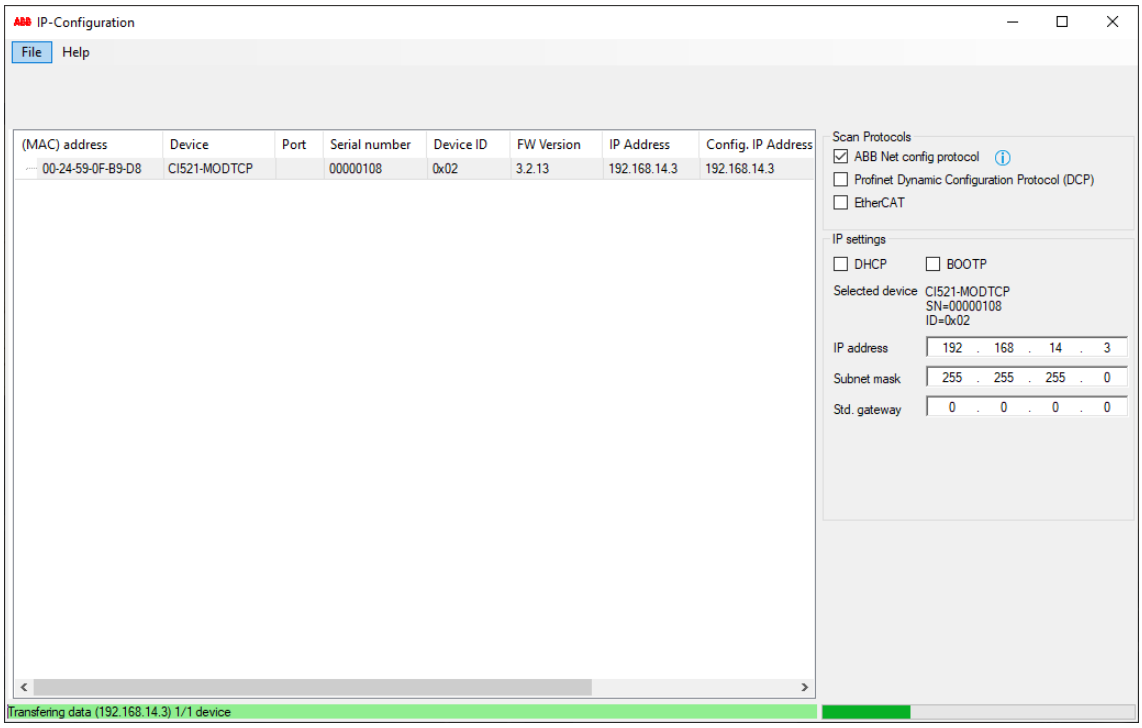
4. Click *"Continue"* to start the installation.
   - ⇨ After a successful installation the IP configuration tool is available as stand-alone tool (.exe).
   - ⇨ To start the IP configuration tool, click the new created desktop icon.

Using the tool functions

Network scan

With a network scan all devices that have been found in the network by the scan process are listed, i.e. ABB devices such as AC500 processor modules, AC500 communication interface modules or ABB Drives.

1. Start the IP configuration tool in Automation Builder (*"Tools ➔ IP-Configuration"*) or start it stand-alone (.exe).
2. The *"IP-Configuration"* dialog opens. Define the device type for the network scan by selecting the desired option under *"Scan Protocol"*:
   - *"ABB Net config protocol"*:
     Use this option for AC500 devices such as processor modules, CI5xx-Modbus devices or ABB Drives. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection.
   - *"Profinet Dynamic Configuration Protocol (DCP)"*:
     Use this option for PROFINET communication interface modules. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection (not via CM579).
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 5824
   - *"EtherCAT"*:
     Use this option for EtherCAT communication interface modules. The Ethernet cable must be connected directly to the first EtherCAT slave device of the EtherCAT fieldbus. Ensure that no EtherCAT master device is available on the bus when a scan is performed.
     "Emergency" option: Enable this option to check on failures in the EtherCAT assembly during the scan process, i.e. a frame loss or interchanged ports. Errors are displayed.
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 5824
3. Click *[Scan]* to start the scan process.



4. All devices that have been found in the network are listed including hardware and connection details. The following details can be changed under *"IP settings"*:

   ⇨ ● *"IP Address"*:
   Current IP address of the device.
   ● *"Conf. IP Address"*:
   Configured IP address of the device. A changed IP address will update this column.
   ● *"FW Version"*:
   Current installed firmware version of the device. This field is visible not until a first network scan. If this field is still empty after a network scan, check on connection errors.
   Chapter 1.6.5.2.2.2.2.3.1 "Trouble-shooting for firmware update" on page 5827

The IP address of some devices, e.g. EtherCAT devices cannot be changed.

Changing the IP address

1. In order to change the IP address of devices perform a network scan.

   Chapter 1.6.5.2.2.2.2.2.1 "Network scan" on page 5819

2. Select a device from the list and select the appropriate protocol under *"Scan protocol"*.

   "DHCP" or *"BOOTP"* option: If required, DHCP or BOOTP can be used to receive the IP address for the device from the server.

   "IP address", *"subnet mask"*, *"Std. gateway"*: Use these fields to change the IP address settings including the settings for the subnet mask and the standard gateway. Ensure that the combination of connection settings is correct.

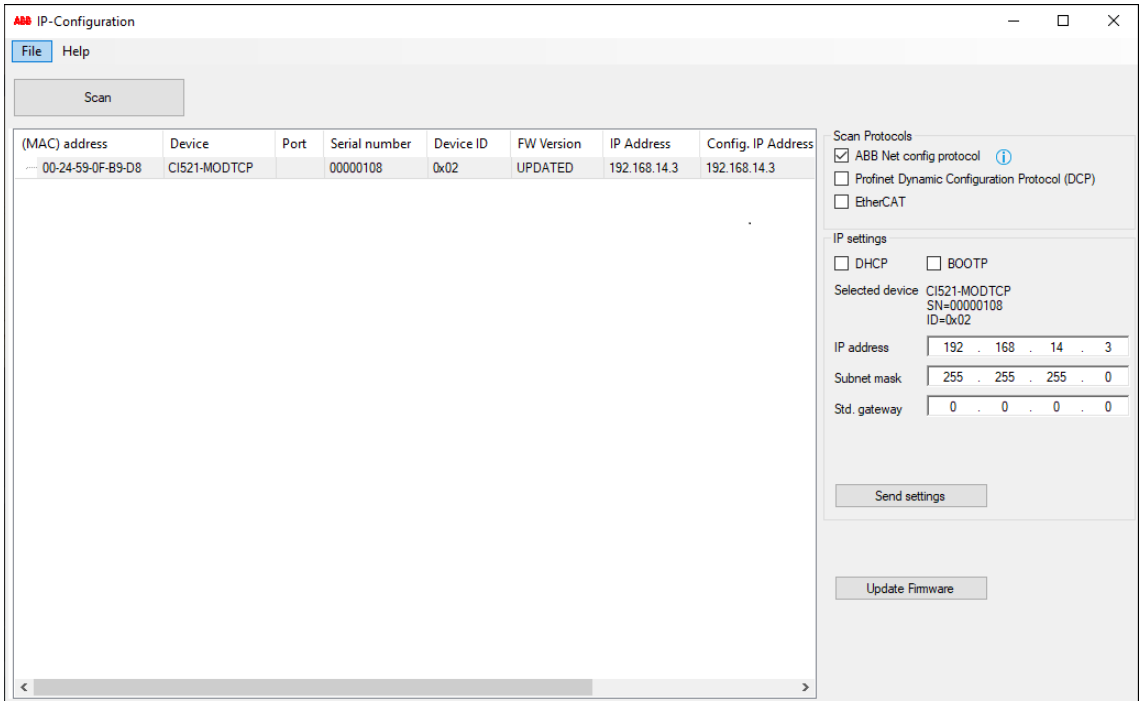
   "Check subnet configuration" on page 5827

Note for CI52x-Modbus devices

Consider the behavior of CI52x-Modbus devices if the last number of the IP address is set to "0".

"Check last number of IP address" on page 5828

3. Change the settings for the IP configuration and click *[Send settings]* to transmit the data to the device.



⇨

Note for PROFINET devices

The device name of PROFINET devices can be edited. If changing the name, ensure the following rules apply:

- *Labels must be separated by "."*
- *Total length: 1 to 240*
- *Label length: 1 to 63*
- *Labels can consist of characters [a-z] and numbers [0-9]*
- *Labels are not allowed to start with "-"*
- *Labels are not allowed to end with "-"*

4. In order to keep all IP changes after a power cycle, the settings can be stored permanently. Confirm the prompted message during the scan process.

Firmware update

The firmware of AC500 communication interface modules can be updated with the IP configuration tool.

For this, the IP configuration tool must be used as stand-alone variant.

↳ *Chapter 1.6.5.2.2.2.1 “Stand-alone installation” on page 5817*

It is not possible to perform a firmware update out of Automation Builder.

- *For PROFINET communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.3.3.*
- *For EtherCAT communication interface modules a firmware update is only supported for devices with firmware version ≥ 2.1.4.*
- *For Modbus communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.2.13.*

Requirements: **Before the firmware update**

- Ensure a fast and stable network connection
- Close all unused applications on the executing PC
- Stop the communication between AC500 PLC and the communication interface module that shall be updated

During the firmware update

- Do not close the IP configuration tool
- Do not open Automation Builder software or any other application
- Do not switch-off the communication interface module that shall be updated
- Do not disconnect the Ethernet connection of a communication interface module or the executing PC

The firmware update will stop the operation of the affected device(s). Hence, the device(s) will become unresponsive for 1 - 2 minutes.

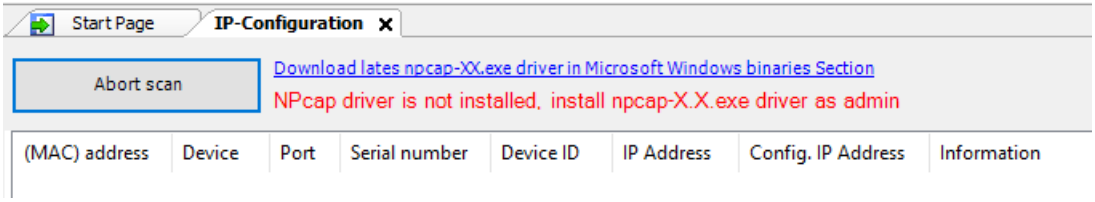
Procedure:

1. Start the IP configuration tool stand-alone (.exe).
2. Perform a network scan.

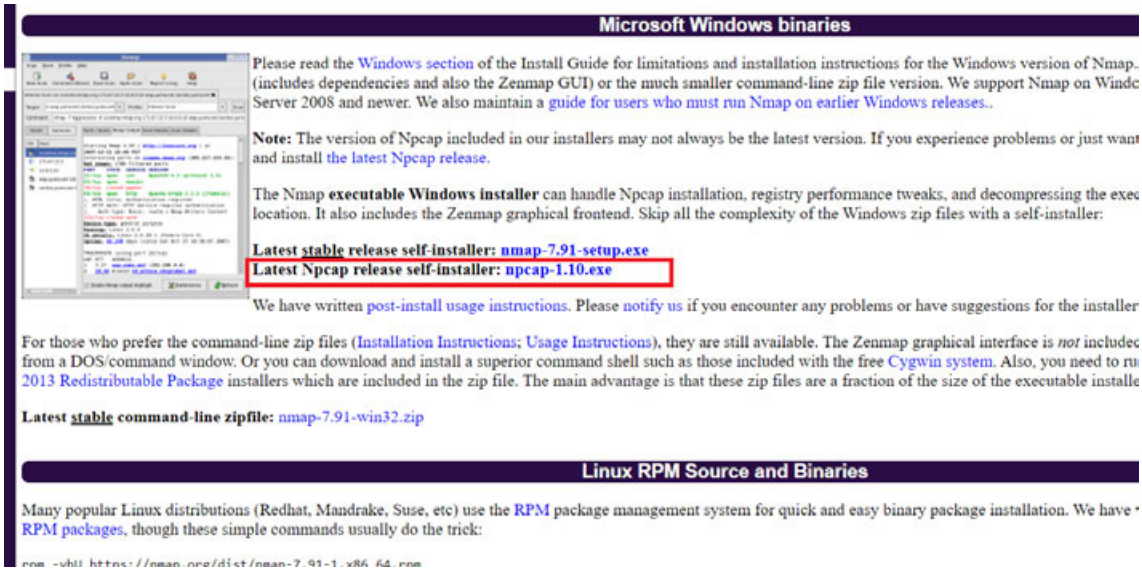
   Chapter 1.6.5.2.2.2.2.2.1 “Network scan” on page 5819
3. Select the devices that shall be updated from the list and click *[Scan]* to trigger the scan process.

   A multiple selection of several devices is possible via control key, however, ensure to select only devices of the same protocol at a time. Otherwise the firmware update fails.

4. This step is only required for devices that require an installed NPcap driver. In this case an appropriate message including a download link is prompted in the IP-Configuration dialog:



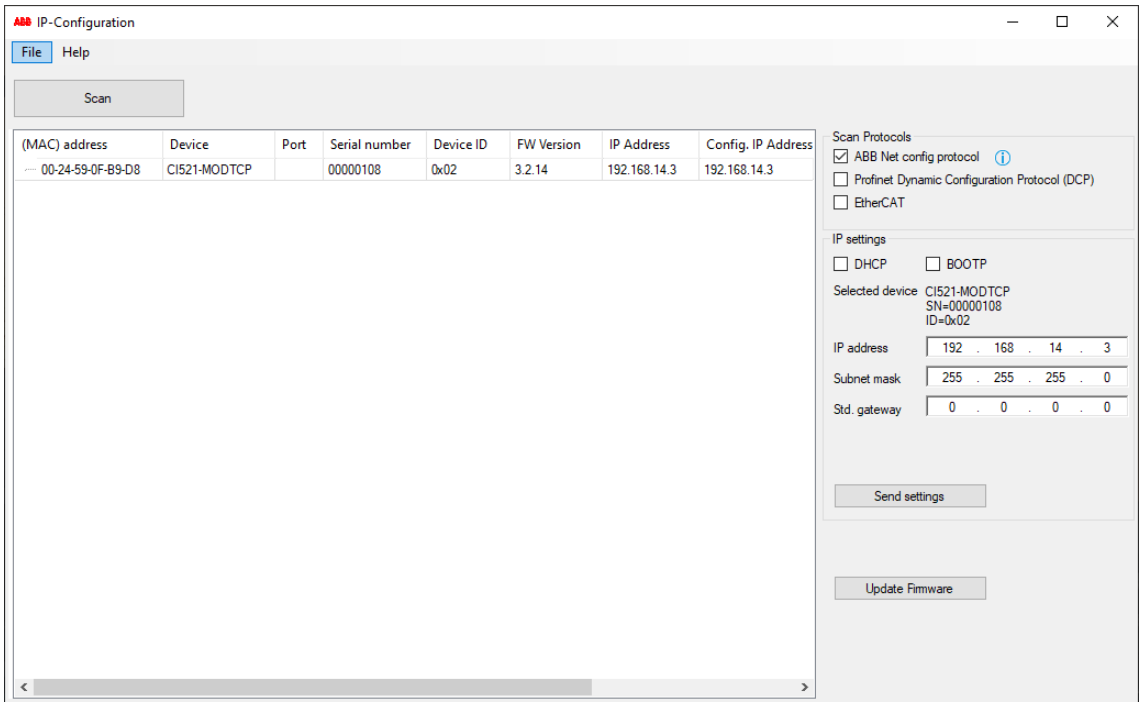
⇨ Click on the displayed link *https://nmap.org/download.html* and download the latest version of the *npcap-X.X.exe* file.



⇨ After the download, execute the file as administrator and restart the scan process.

⇨ The devices that have been scanned are listed.

5. Click *[Update Firmware]* to start the firmware update for the selected devices.



6. For CI50x, CI51x and CI52x devices a signature check is started. Select the appropriate firmware update file (*.bin) for the device(s). Example: C:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.

   After a successful signature check the firmware update file (*.bin) and the respective signature file (*.bin.sig) are transferred to the device. This can last up to 3 minutes.

   If the signature check fails, check the availability of the *.bin file and the *bin.sig file.

   ↳ *"Signature check" on page 5828*

7. A status check followed by a device reboot followed by a second status check is performed automatically.

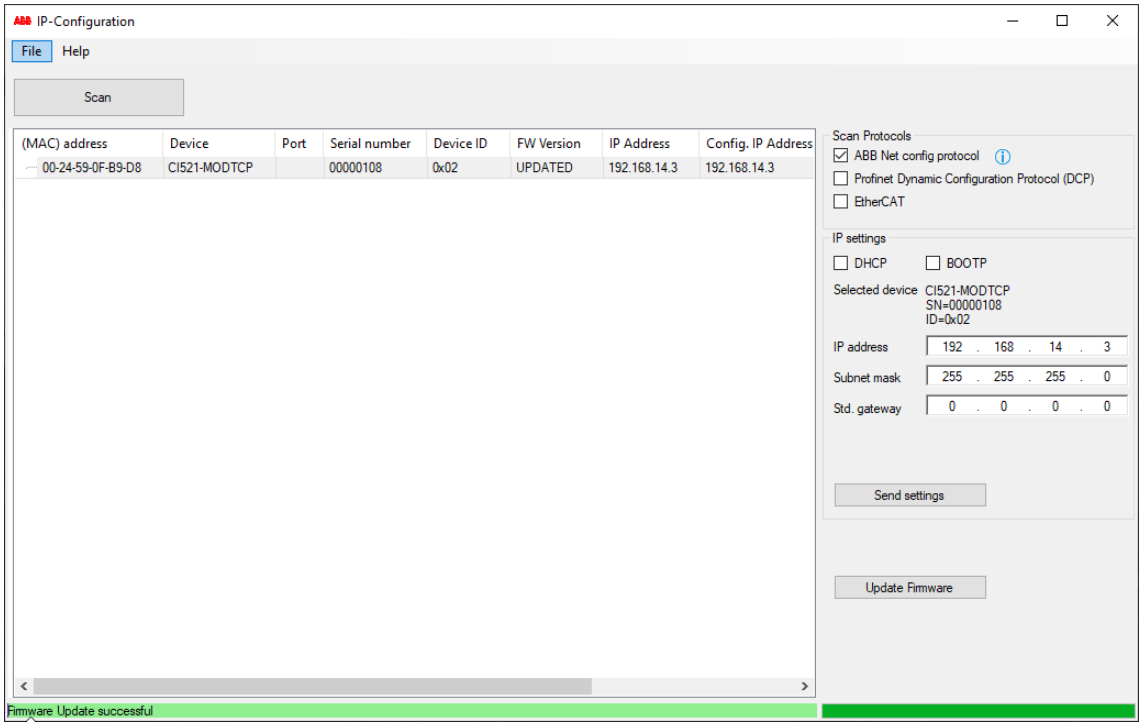
   > *After the firmware update all outputs of the updated devices are set to '0'.*

8. After a successful firmware update the update status or the new firmware version is displayed in the *"FW Version"* field.

   If this field is empty, there possibly is a connection error between the device and the executing PC.

   ↳ *"Error: Can't connect to device" on page 5829*

   Exception: For EtherCAT devices an empty *"FW Version"* field does not indicate a connection error.



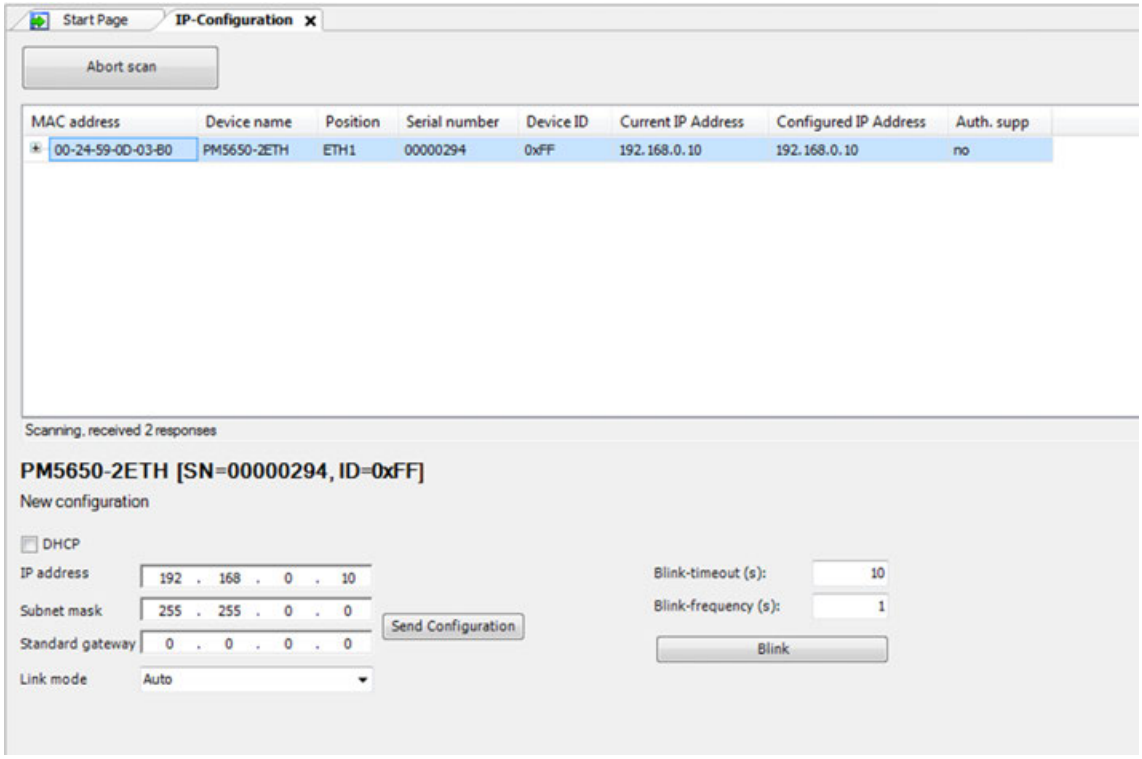
⇨ If the firmware update fails

- check the requirements for the update procedure.
  ↳ *"Requirements:" on page 5823*
- check the hints for trouble-shooting.
  ↳ *Chapter 1.6.5.2.2.2.2.3.1 "Trouble-shooting for firmware update" on page 5827*
- perform a network scan and repeat the update. If the error still persists power cycle the device and try the update again.

Blink functionality

This function activates flashing of the backlight of an AC500 LED display.

1. From the menu, select *"Tools ➔ IP-Configuration"*.
2. Click *[Scan]* to trigger the scan process for devices in the network.
   ⇨ A progress bar shows the progress. The IP settings of a selected device is displayed below the list and can be edited.
3. Adjust your desired time and click *[Blink]* to activate flashing.



Trouble-shooting for IP configuration tool

Firewall exceptions:

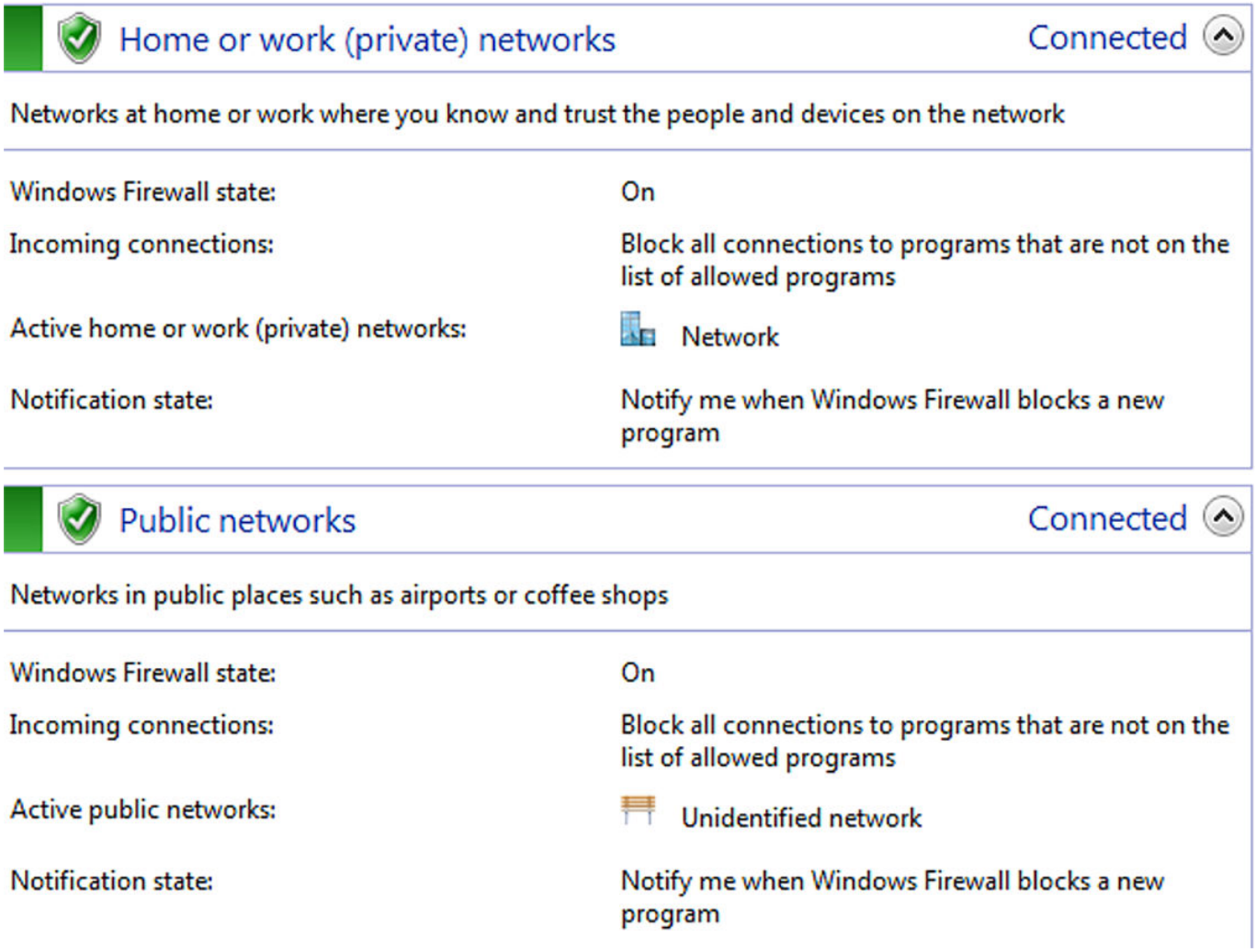
On a standard Windows 7 installation without third party firewall or security tools installed the IP configuration tool should work properly.

The Automation Builder setup installs rules or exceptions for the built-in Windows firewall to allow IPConfig to receive the responses for the IPConfig scan.

To check the Windows firewall is set correctly check the firewall settings.

Windows 7/ Windows 10:

On the network that is used for communication with the PLC, set *"Incoming connections"* to "Block all connections to programs that are not on the list of allowed programs".



If a third party firewall is used these exceptions must be configured manually.

> *Either exceptions for applications can be entered: Automation Builder and IP configuration tool must be added as application.*
>
> *Or the protocol and the port number must be given (for IPConfig: UDP protocol and port number 24576).*

Trouble-shooting for firmware update

Check the requirements

Ensure that all requirements have been considered before and during the update procedure.
↳ *"Requirements:" on page 5823*

Check subnet configuration

This hint is only valid for Modbus devices and PROFINET devices.

If the *"FW Version"* field is empty after the network scan or if the firmware version has not been updated after the update procedure, there possibly is a connection error between the device and the executing PC.

Ping the device from the executing PC. If no connection can be established, check whether the device and the PC are in the same subnet.

Example

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

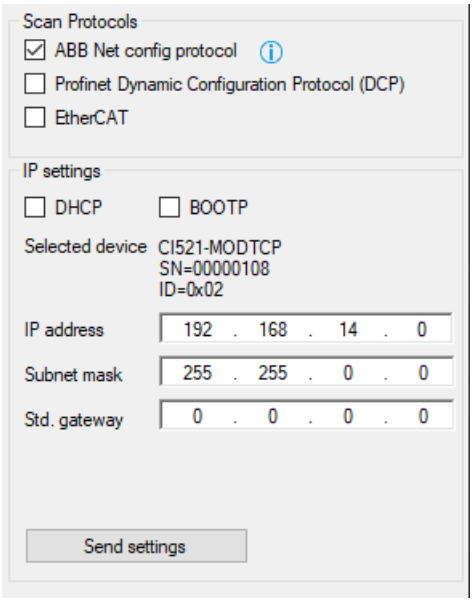
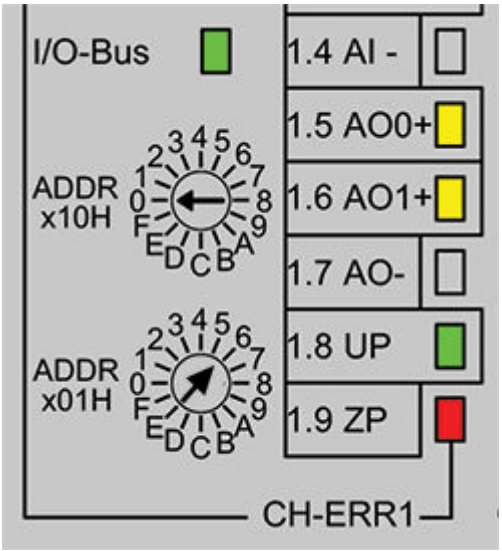
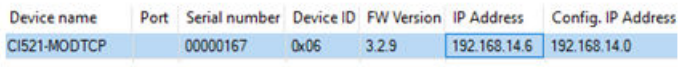
Click *[Scan]* again to restart the network scan. If the connection is successful a newer firmware version is displayed in the *"FW Version"* column.

Check last number of IP address

This hint is only valid for CI52x-Modbus devices.

Check the last number of the IP address. If it is set to "0", the IP address setting for this last number will be used from the rotary switches on the hardware device.

Example:

Automation Builder	AC500 communication interface module (rotary switch)
IP address: 192.168.14.0	IP address: 6
	
As a result, in the field *"IP Address"* the last number is set to "6": 	

Signature check

During the firmware update of CI50x, CI51x and CI52x devices a signature check is started.

The update procedure expects a firmware update file (**.bin*) and a signature file (**.bin.sig*) in the same directory. Without a signature file the signature check will fail.

Example:

Firmware update file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin***

Signature file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin.sig***

Error: Package timeout

A timeout error may occure due to an instable network.

Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

Error: Unable to read device status

A read error may occure due to errors in the firmware update protocol.

After the firmware update the IP configuration tool reads out the status of the updated device in order to check if the update was successful.

Error: IP is not unique

If an IP address is obtained by more than one device an error occures. A firmware update is not possible.

Error: Error State

Internal device error during the firmware update.

Solution:

Step 1: Scan again and repeat the firmware update.

Step 2: If this does not work, power cycle the device, scan again and repeat the firmware update.

Error: Can’t connect to device

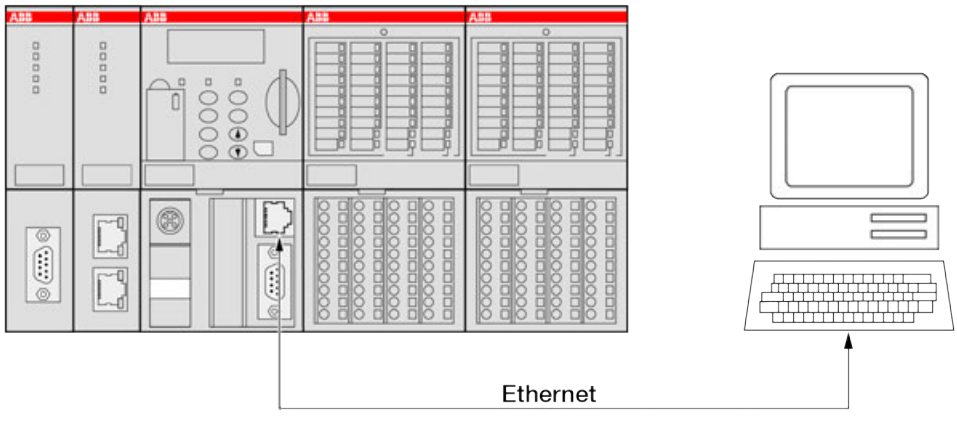
The TCP communication is not sufficient. Increase the connection quality.

Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

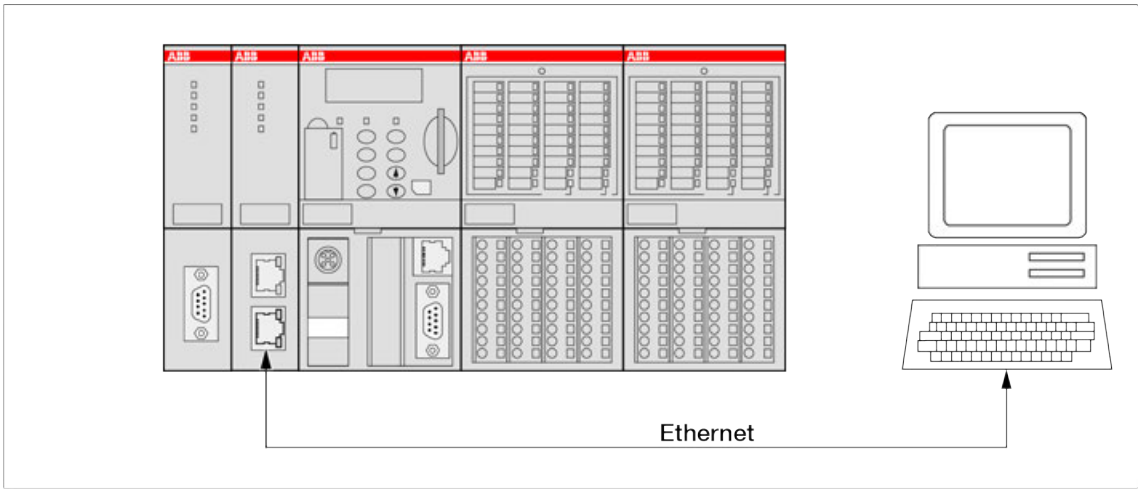
Configuration of communication via Ethernet (TCP/IP)

Programming via Ethernet is only possible on a PC with Ethernet board and installed network. Programming can be done via the internal and external Ethernet communication module.

Programming via internal (onboard) Ethernet communication module:



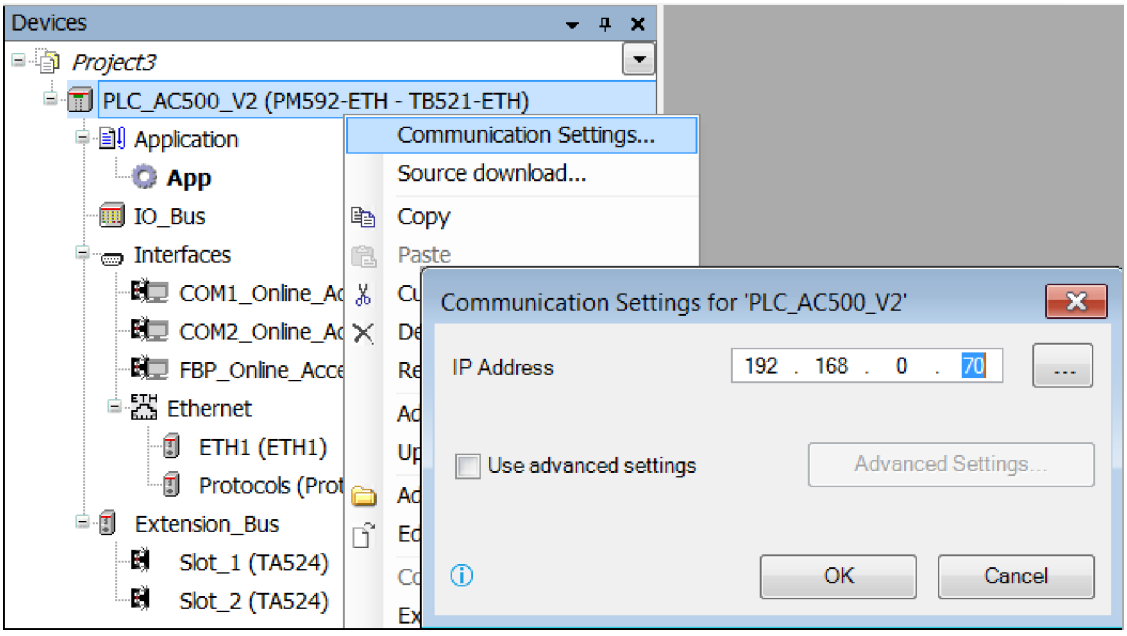
Programming via external Ethernet communication module (in the example communication module 1 in slot 1):



Enter a known PLC IP address

1. Right-click the top node *“PLC_AC500 <...>”* and select *“Communication Settings”* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.

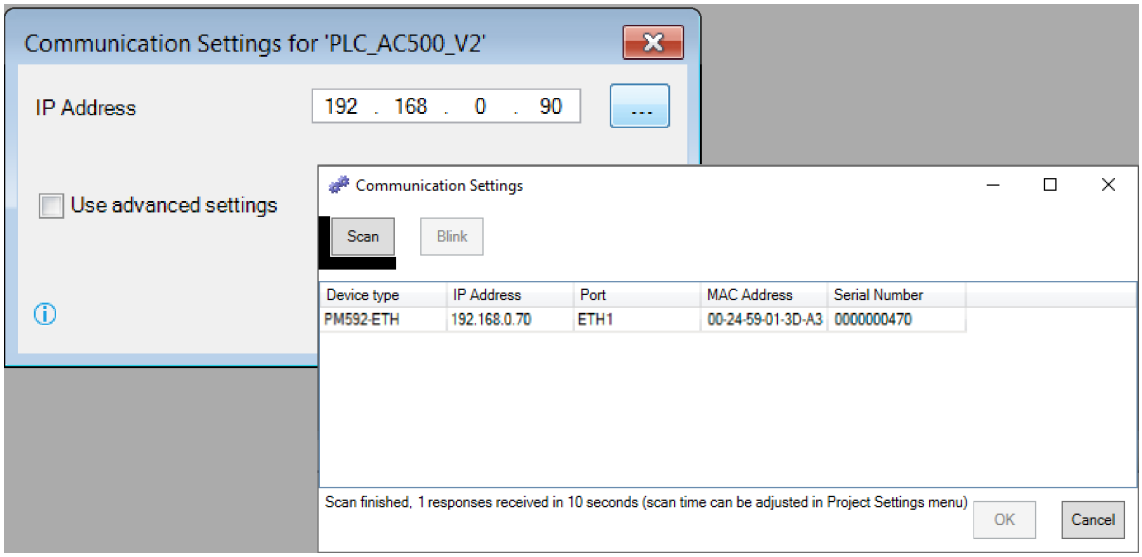


2. Enter your PLC *IP Address* and click *[OK]*.

Enter PLC IP address by scanning devices

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.



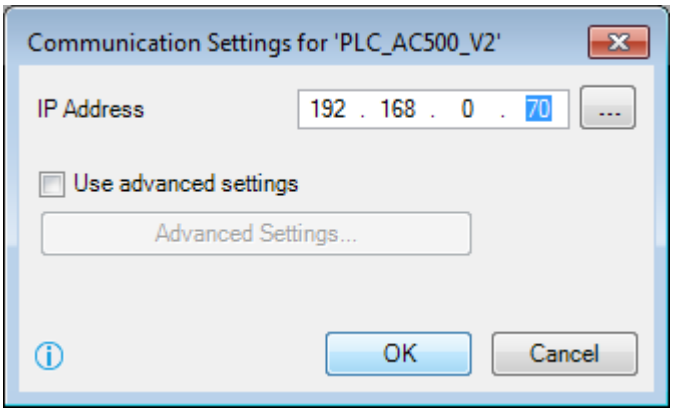
2. Click *[...]*.

   ⇨ Dialog box *Communication Settings <...>* appears.

3. Click *[Scan]*, select your desired PLC and click *[OK]*.

   ⇨ Entry is transferred to the dialog box *Communication Settings <...>*.

   Click *[OK]*.



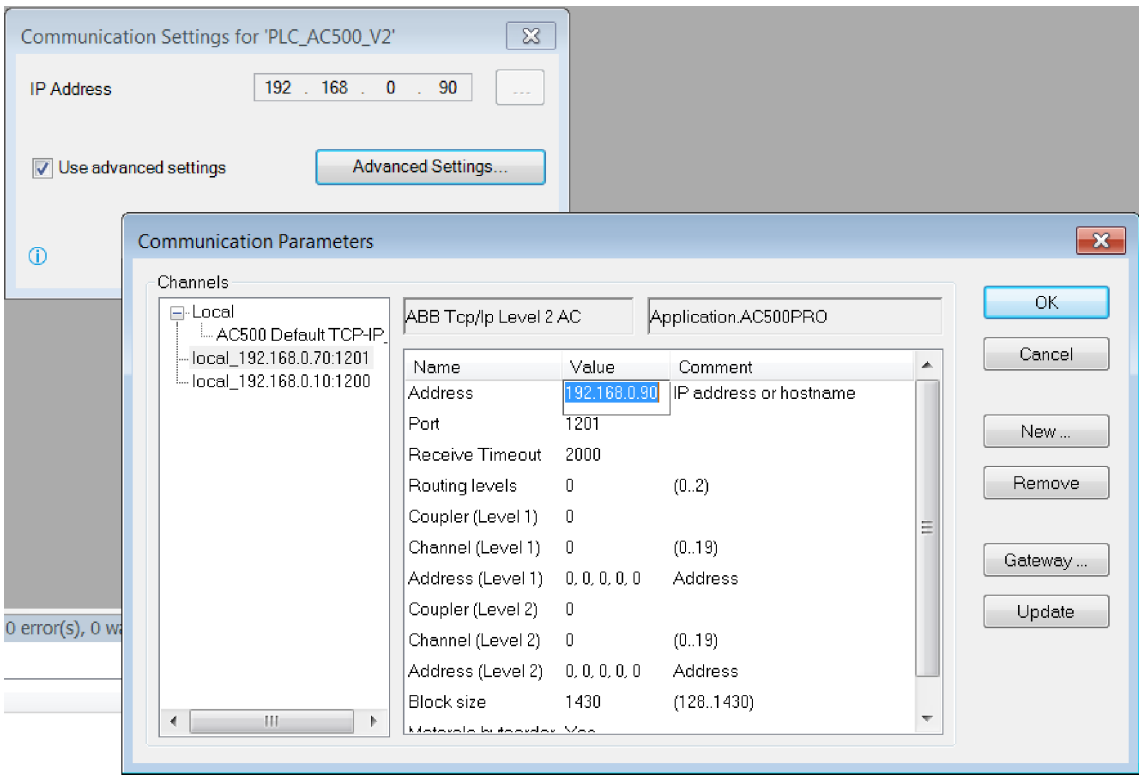
4. Click to log in the *"PLC_AC500_V2"* project.

Enter PLC IP address by *[Advanced Settings...]*

If a remote gateway instead of a local one has to be used it can be configured in the *[Advanced Settings...]*.

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings* <...> appears.



2. Enable checkbox *Use advanced settings* and click *[Advanced Settings]*.

   ⇨ Dialog box *Communication Parameters* appears.

3. Select your channel, enter your PLC IP address and click *[OK]*.

   ⇨ Entry is transferred to the dialog box *Communication Settings* <...>.

   Click *[OK]*.

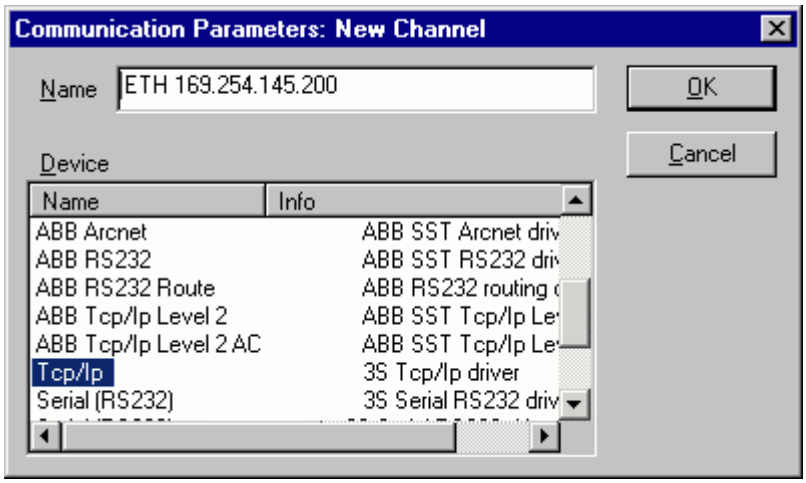
4. Click to log in the *"PLC_AC500_V2"* project.

Ethernet driver "TCP/IP"

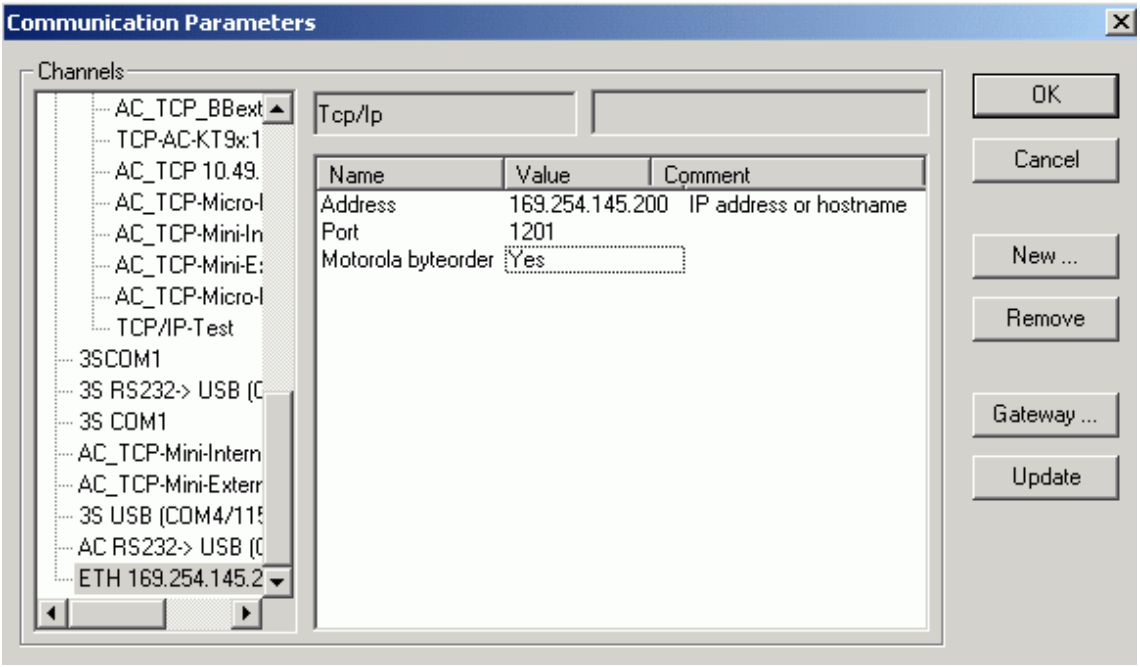
Programming AC500 controllers with internal and/or external Ethernet communication module via Ethernet can be done by using the driver "TCP/IP". This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "TCP/IP" from the device list.



The following communication parameters can be set for the Ethernet driver "TCP/IP":



Parameter	Possible values	Description
Address	0.0.0.0	IP address or hostname of the PLC
Port	1201	Port 1201
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

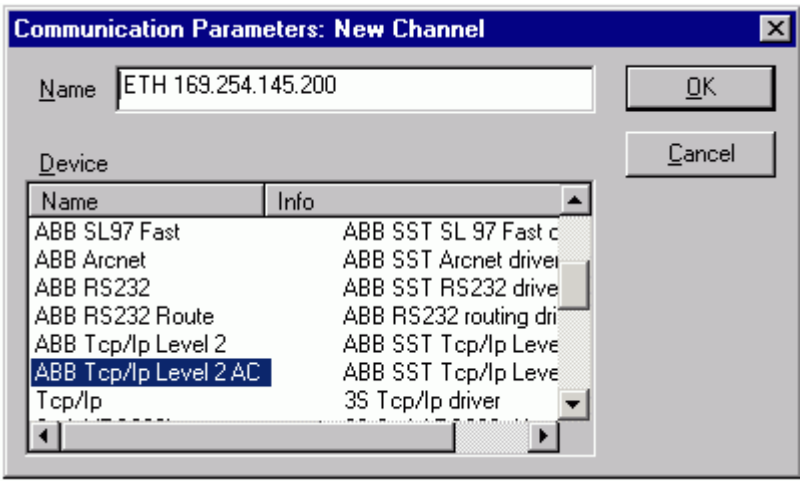
Ethernet driver "ABB TCP/IP Level 2 AC"

As of version V1.2, the driver "ABB TCP/IP Level 2 AC" is available for programming AC500 controllers with internal and/or external Ethernet communication module via Ethernet. This driver provides the following functions:

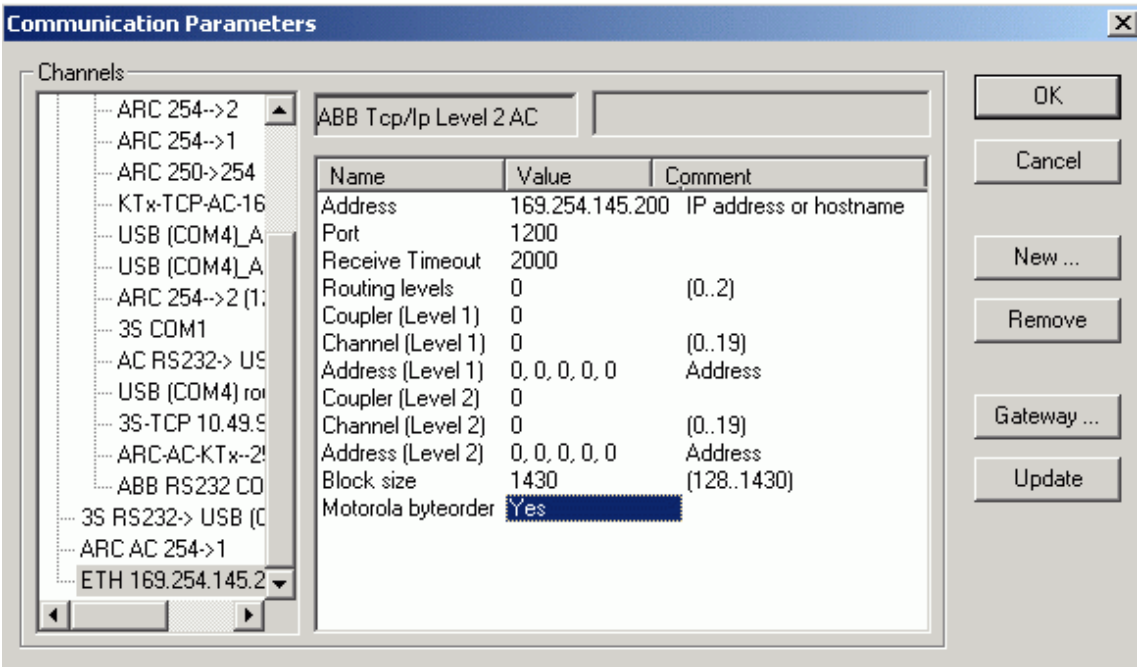
- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and OPC server

- Parallel operation of Control Builder instances with several PLCs
- Online operation of PLCs connected via ARCNET. One PLC equipped with Ethernet communication module and one PLC with ARCNET communication module (Routing Ethernet -> ARCNET), as of version V2.x

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.



The following communication parameters can be set for the Ethernet driver "ABB TCP/IP Level 2 AC":

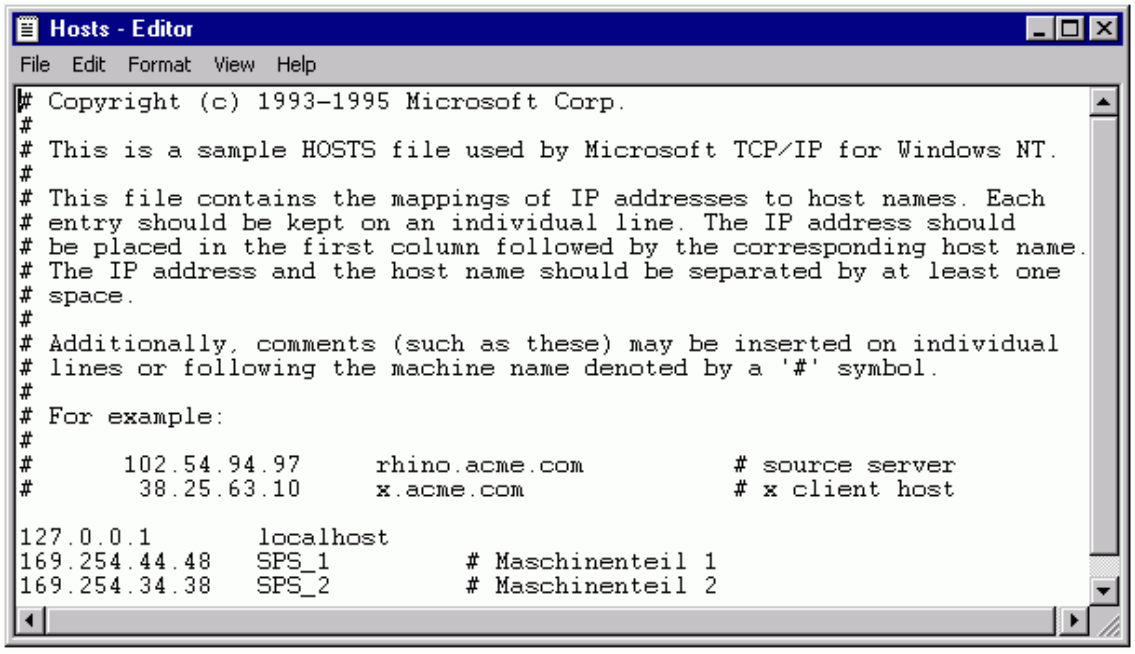


Parameter	Possible values	Description
Address	0.0.0.0	IP address or hostname of the PLC
Port	1200	Port 1200
Timeout (ms)	>= 2000	Timeout [ms] for response
Routing levels	0...2	Routing levels (0 = none)

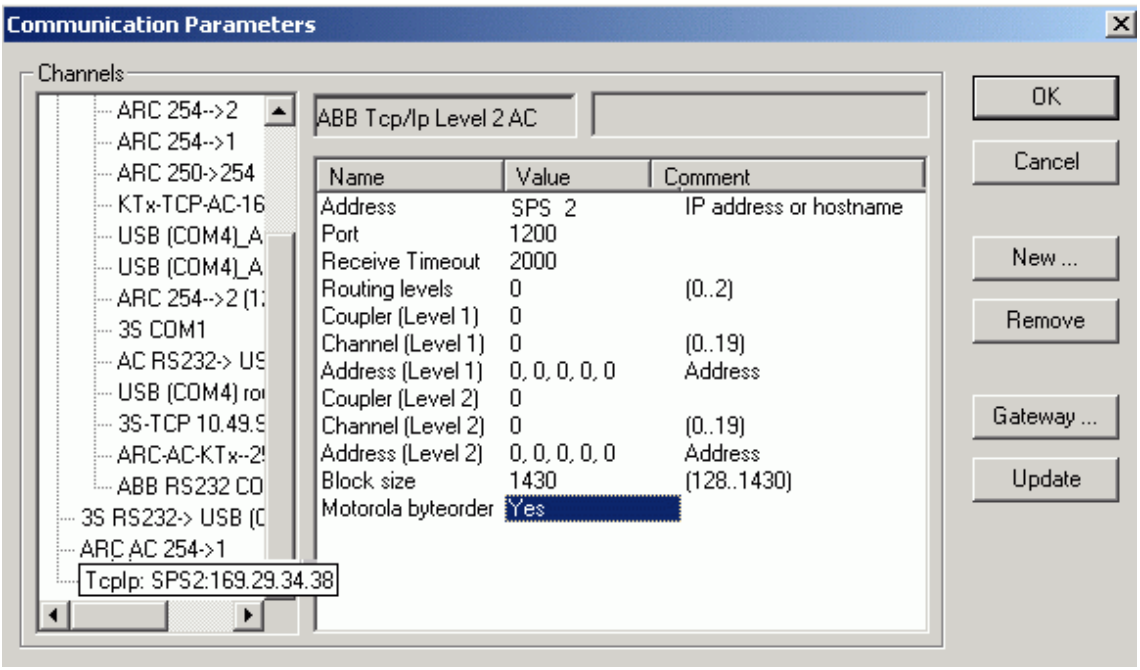
Parameter	Possible values	Description
Communication Module (Level 1)	0, line 0...line 4	Communication module for level 1
Channel (Level 1)	0...19	Channel on communication module level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 1
Communication Module (Level 2)	0, line 0...line 4	Communication module for level 2
Channel (Level 2)	0...19	Channel on Communication Module level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 2
Block size	1430 (128...1430)	Bytes per telegram (unallowed 227..245)
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

If you want to use the Ethernet driver to directly access the PLC, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

The "Address" parameter sets the IP address or hostname of the PLC. To be able to use hostnames, the names have to be added to the file "Hosts". Under Win2000, this file is located in the directory "WINNT\System32\drivers\etc".



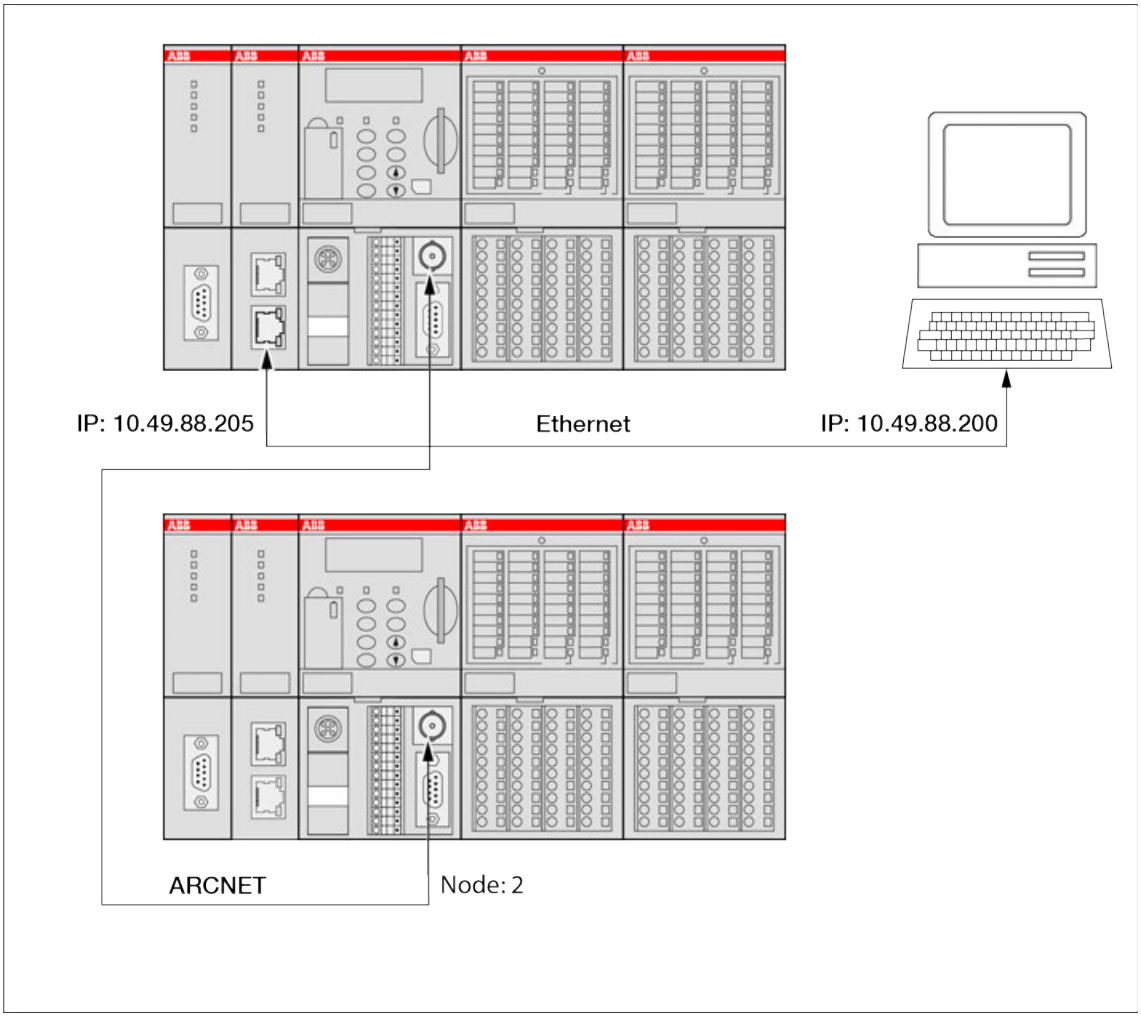
If you have changed the "Hosts" file accordingly, you can enter the symbolic name for the "Address" parameter instead of the IP address. In the following figure, the IP address "169.254.34.38" is replaced by the hostname "SPS_2".



Ethernet ARCNET routing

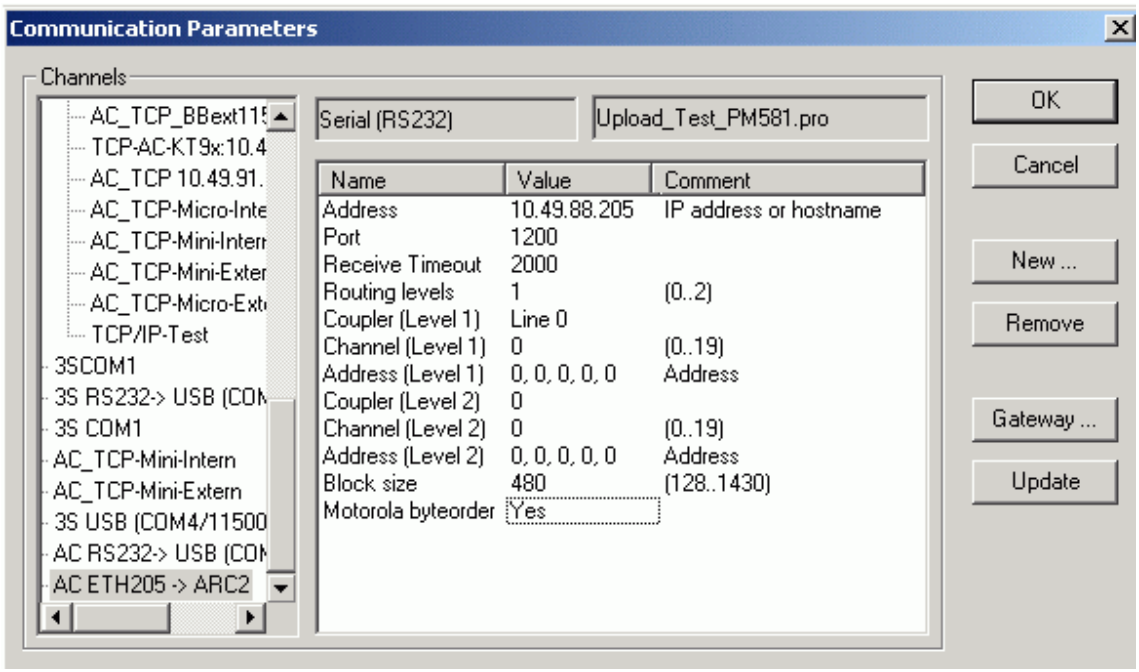
Routing is available as of PLC firmware version V1.3.

For controllers with Ethernet and ARCNET communication module, the PLCs connected via ARCNET can be programmed using the PLC Ethernet interface.



For each PLC connected via ARCNET, one gateway channel has to be defined. To do this, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example TcpIp: PLC1:169.29.44.48 -> ARC_2) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.

For example, set the communication parameters as follows for the configuration shown above:



Parameter	Possible values	Description
Address	10.49.88.205	IP address of PLC 1
Port	1200	Port 1200
Timeout (ms)	2000	Timeout [ms] for response
Routing levels	1	Single-level routing
Communication Module (Level 1)	Line 0	Communication module for level 1 (internal: ARCNET)
Channel (Level 1)	0	Channel on Communication module level 1
Address (Level 1)	2, 0, 0, 0, 0	ARCNET node of the target PLC (Node 2)
Communication Module (Level 2)	0	No level 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Block size	480	Bytes per block: 128...1430
Motorola byteorder	Yes	

For the parameter "Communication Module (Level 1)", enter the slot where the ARCNET communication module "Line 0" is inserted (the ARCNET Communication Module is always the internal communication module).

The ARCNET communication module has only one communication channel. Thus, the "Channel" value must always be 0.

For the ARCNET communication module, 1 byte is required for the subscriber address (node). The address (Node=2) of the target PLC is entered to the first byte of the address byte.

The default value for the block size is 1430. If routing on ARCNET is required (and "large ARCNET packages" are enabled for the target PLC), the block size can be increased to 480 bytes. Values in the range of 227 .. 245 are not allowed.

1.6.5.2.3 Processor modules

Configure a processor module in the device tree

To configure your PLC in the Automation Builder:

1. Add a processor module to your project. ↳ *Chapter 1.6.5.1.1.1 “Creating a new project ” on page 5758*
2. Double-click the PLC node in the device tree.
   ⇨ This will open a new window with tabs for the device configuration:
   - *“PMxxx Parameters”* ↳ *Chapter 1.6.5.2.3.3 “Parameters of the processor module” on page 5839*
   - *“PM5xx Hardware”* ↳ *Chapter 1.6.5.2.3.4 “Changing the processor module type” on page 5844*
   - *“I/O mapping list”* ↳ *Chapter 1.6.5.1.4 “I/O mapping list” on page 5781*
   - *“Information”* General information about the device (name, vendor, version etc.)
3. Select the *“PMxxx Parameters”* tab to configure the parameters for the processor module. ↳ *Chapter 1.6.5.2.3.3 “Parameters of the processor module” on page 5839*
4. Select the *“I/O mapping list”* tab to create mapping variables with better usability support compared to the tree structured view. ↳ *Chapter 1.6.5.1.4 “I/O mapping list” on page 5781*

Processor modules with onboard interfaces: PM5xy-ETH, PM5xy-2ETH, PM5xy-ARC

- PM5xy-ETH: Processor module with network interface Ethernet RJ45 (onboard Ethernet).
- PM5xy-2ETH: Processor module with 2 network interfaces Ethernet RJ45 (onboard Ethernet).
- PM5xy-ARC: Processor module with network interface ARCNET BNC (onboard ARCNET)

Processor modules with the extension "-ETH" or "-ARC" provide an onboard interface for direct communication via Ethernet or ARCNET without using an additional communication module.

Ethernet for example can be used by using either a processor module with onboard Ethernet (e.g. PM591-ETH) or a communication module which supports Ethernet (e.g. CM597-ETH). Details on the configuration of processor modules with onboard Ethernet is provided in the configuration description for devices with onboard Ethernet. ↳ *Chapter 1.6.5.2.4.1.1 “Parameterization of PM5xy-ETH” on page 5860*

Details on the configuration of processor modules with onboard ARCNET is provided in the configuration description for devices with onboard ARCNET. ↳ *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*

Parameters of the processor module

Add the desired processor module to the device tree, then double-click the *“Processor Module”* node. In the editor window open the *“PMxxx Parameters”* tab and check whether the default configuration is to be changed.

For Processor Modules with onboard Ethernet see the Ethernet parameters and IP Settings ↳ Chapter 1.6.5.2.4.1.1 “Parameterization of PM5xy-ETH” on page 5860.

For Processor Modules with onboard ARCNET see the ARCNET parameters ↳ Chapter 1.6.5.2.3.6 “Parameters of PM5x1-ARCNET (onboard ARCNET)” on page 5846

The following parameters are available:

Parameter	Default	Value	Description
Auto run ↳ *Chapter 1.6.5.2.3.3.1 “Remark 1: Setting the parameters auto run and MOD using the display/keypad” on page 5841*	On	On	If the Flash memory contains a valid project, the project will be loaded into the RAM memory and executed when switching on the CPU.
		Off	If the Flash memory contains a valid project, this project will be loaded into the RAM memory but not executed when switching on the CPU.
Error LED ↳ *Chapter 1.6.5.2.3.3.2 “Remark 2: Error LED” on page 5842*	On	On	The error LED lights up for errors of all classes, no failsafe function activated.
		Off by E4	Warnings (E4) are not indicated by the error LED, no failsafe function activated.
		Off by E3	Warnings (E4) and minor errors (E3) are not indicated by the error LED, no failsafe function activated.
		On + Failsafe	The error LED lights up for errors of all classes and the failsafe function of the I/O Bus is activated.

Parameter	Default	Value	Description
		Off by E4 + Failsafe	Warnings (E4) are not indicated by the error LED, (no error Display with "diagnostic" button on the LCD possible), the failsafe function of the I/O Bus is activated.
		Off by E3 + Failsafe	Warnings (E4) and minor errors (E3) are not indicated by the error LED. The failsafe function of the I/O bus is activated.
Check battery	On	On	The presence of the battery and the battery status are checked. If no battery is available or the battery is empty, a warning (E4) is generated and the ERR LED lights up.
		Off	The presence of the battery is not checked. No warning (E4) is generated. The LCD display "Batt" (triangle) can not be acknowledged! This also applies if a battery is installed but empty.
Behavior of outputs in stop *Chapter 1.6.5.2.3.3.3 "Remark 3: Behavior of outputs in stop" on page 5842*	Off in hardware and online	Off in hardware and online	In case of STOP, all outputs at the hardware and in the online display are set to FALSE or 0.
		Off in hardware and actual state online	In case of STOP, all outputs at the hardware are set to FALSE or 0. The online display indicates the status from the last cycle of the user program.
		Actual state in hardware and online	The status of the last cycle of the user program is kept for the outputs at the hardware and in the online display.
Stop on error class	E2	E2	In case of a fatal or severe error (E1-E2), the user program is stopped.
		E3	In case of a fatal, severe or minor error (E1-E3), the user program is stopped.
		E4	In case of a fatal, severe or minor error (E1-E3) or a warning (E4) the user program is stopped.
Warmstart *Chapter 1.6.5.2.3.3.4 "Remark 4: Warmstart" on page 5842*	Off	Off	In case of a fatal error (E2), no warmstart is performed.
		On after E2 error	In case of a fatal error (E2), a warmstart is performed automatically.
		On after short voltage dip	A warmstart is performed after a short voltage dip.
		On after E2 or short voltage dip	In case of a fatal error (E2) or after a short voltage dip, a warmstart is performed automatically.
Reaction on floating point exceptions *Chapter 1.6.5.2.3.3.5 "Remark 5: Reaction on floating point exceptions" on page 5843*	E2 failure	E2 failure	Only for PM59X: If a floating point exception occurs, an E2 error (Err=38) is triggered. The CPU goes to STOP.
		No failure	Only for PM59X: If a floating point exception occurs, no E2 error is triggered. Using the block FPU_EXINFO in the user program allows to react on a possibly occurred exception.
Flexible configuration	None	None	No flexible configuration is used.

Parameter	Default	Value	Description
Chapter 1.6.5.1.3 “Flexible AC500 configuration ” on page 5774		Flash	FlexConf.ini is loaded from Flash.
		User Program	FlexConf.ini is loaded by user program.
		FTP file	FlexConf.ini is loaded from FTP.
Flexible configuration timeout *Chapter 1.6.5.1.3 “Flexible AC500 configuration ” on page 5774*	1000	0...65535	FlexConf.ini timeout in seconds [s].
Free wheeling pause	10	0...255	Free wheeling pause in milliseconds [ms]. *)
Task compatibility mode	File Handling priorized	Ethernet Handling priorized	Task compatibility mode.
		Balanced Handling of Ethernet and file operations	
		File Handling priorized	
Start PERSISTENT %R0x *Chapter 1.6.5.2.3.3.6 “Remark 6: Start PERSISTENT %Rsegment.x and end PERSISTENT %Rsegment.x” on page 5844*			Start offset for buffered area in PERSISTENT area %R0x
End PERSISTENT %R0x			End offset for buffered area in PERSISTENT area %R0x
...			
Start PERSISTENT %R7.x			Start offset for buffered area in PERSISTENT area %R7.x
End PERSISTENT %R7.x			End offset for buffered area in PERSISTENT area %R7.x

*) Setting this parameter to '0' causes the CPU to work with the default setting 10 ms.

Remark 1: Setting the parameters auto run and MOD using the display/keypad

Loading and running the user program also depends on the setting for the parameter MOD using the display/keypad (in AC500 CPUs PM57x, PM58x and PM59x). The display/keypad setting always has the higher priority.

The following applies:

MOD 00:	User program will be loaded and run according to the setting for the CPU parameter "Auto run" (default setting).
MOD 01:	User program will not be loaded/will not run.
MOD 02:	User program will be loaded and run independent of the setting for the CPU parameter "Auto run".

Keeping the RUN key pressed when booting the PLC automatically activates MOD 01, i.e. the user program is not loaded/does not run. Thus, it is possible to boot the PLC in Stop status. This may be required if, for example, both serial interfaces are set to Modbus and therefore no access with the Control Builder software is possible via the serial interface.

On CPUs PM55x and PM56x, a RUN/STOP switch is designed for the MOD function.

If the RUN/STOP switch is set to STOP mode before the CPU module is powered on, the program in the CPU will not run after the CPU is powered on. The program can only be executed when the switch is set to RUN mode and the program is downloaded on the CPU again.

If the serial interface COM1 or COM2 is set to communication mode in a running program (e.g. Modbus communication), it can be changed back to other mode (e.g. online access to Control Builder software) only by setting the switch to STOP mode.

Remark 2: Error LED

In addition to setting the behavior of the CPU's error LED ERR, this parameter is used to set the failsafe behavior of the I/O Bus.

Remark 3: Behavior of outputs in stop

The setting of the parameter Behavior of outputs in stop directly influences the failsafe function of the outputs of the S500 I/O Devices.

Remark 4: Warmstart

The parameter *“warmstart”* allows to set the behavior of the CPU in case of

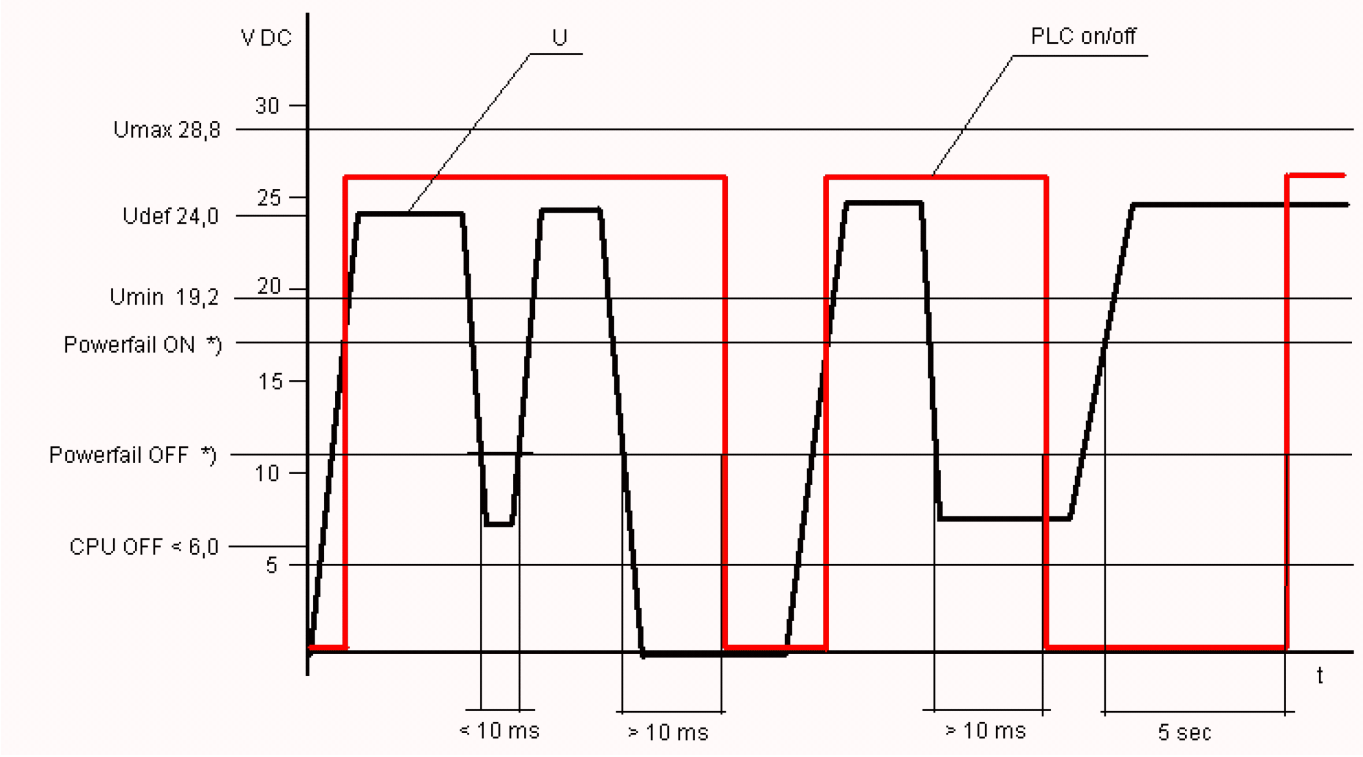
- severe errors (class E2) and
- short voltage dips (only variants with 24 V DC supply)

If the default setting is used, the CPU changes to STOP mode if a severe error occurs. The CPU is switched off for voltage dips >10 ms. The display shows AC500.

The new settings allow performing a warmstart of the CPU after a severe error or after short voltage dips or in case of both events.

The parameter “warmstart” after E2 error takes effect not before the configuration data of the project has been evaluated. If an E2 error occurs prior to this, it will be recorded into the diagnosis system, but no restart will be performed. The intention of this parameter is to raise the availability of the PLC during RUN mode (RUN) when non-systematic E2 errors occur.

The following figure shows the behavior of the CPU for different control voltage signals.



Short voltage dips, i.e., the control voltage falls below a value lower than "Powerfail OFF" (<11 V DC) for less than 10 ms, are bridged by the PLC, i.e., the CPU remains on.

If the control voltage is switched off, the CPU remains on for > 10 ms.

If the control voltage is lower than 11 V DC (but > 6 V DC) for longer than 10 ms and then goes back to the normal value, the behavior of the CPU depends on the setting for the parameter *"Warmstart"*. If the parameter is set to "Off", the CPU remains in power fail mode, i.e. it does not restart. A restart of the CPU can only be done by switching the control voltage OFF/ON. If the parameter is set to "On after short voltage dip" or "On after E2 or short voltage dip", the CPU is restarted when the control voltage is greater than 17 V DC for 5 seconds. However, if the control voltage falls once more below 11 V DC within these 5 seconds, the time is restarted. Thus, the control voltage must have a value > 17 V DC for 5 seconds.

Remark 5: Reaction on floating point exceptions

Behavior of PM59x regarding floating point exceptions can be set. In standard case, any floating point exception triggers an E2 error: class=E2, err=38, d1=9, d2=31, d3=31.

The CPU switches to STOP.

CPUs without floating point processor PM57x and PM58x do not trigger a floating point exception.

If the parameter Reaction on floating point exceptions is set to No failure, no error is triggered in case of a floating point exception. The CPU remains in RUN mode.

By means of the function block ↬ *Chapter 1.5.4.19.2.19 "FPU_EXCEPTION_INFO" on page 1551* (contained in SysInt_AC500_V10.LIB) it can be determined whether a floating point exception occurred during calculation. Depending on the result, either the calculation can be continued with default values or the machine can be shut down.

Remark 6: Start PERSISTENT %Rsegment.x and end PERSISTENT %Rsegment.x

The parameters "Start PERSISTENT %Rsegment.x" and "End PERSISTENT %Rsegment.x" are used to buffer this area. In the particular segment, "Start PERSISTENT %Rsegment.x" specifies the start byte and "End PERSISTENT %Rsegment.x" the end byte of the area to be buffered.

Changing the processor module type

In a project, you can change the target system by changing the type of processor module or terminal base type. If possible, the device configuration of fieldbusses and interfaces is kept and switched over to the device configuration of the new module.

Target change options:

- between platforms: from V2 platform to V3 platform (and vice versa)
- between module types: from AC500 (standard) to AC500-eCo (and vice versa)
- a combination of changed platform and changed module type

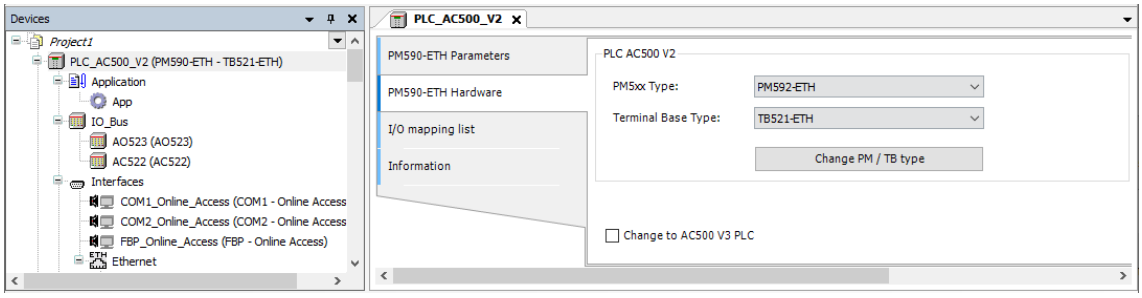
Target change from a V2 processor module to another V2 processor module

Target change options:

- AC500 V2 processor module ➜ AC500 V2 processor module
- AC500 V2 processor module ➜ AC500-eCo V2 processor module
- AC500-eCo V2 processor module ➜ AC500 V2 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V2 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *“PM5<...> Hardware”* tab.
3. Select the desired V2 processor module from the *“PM5xx Type”* drop-down list.



4. Ensure the correct *“Terminal Base Type”* is selected and click *[Change PM / TB type]*.
   ⇨ The new V2 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

Target change to PM595

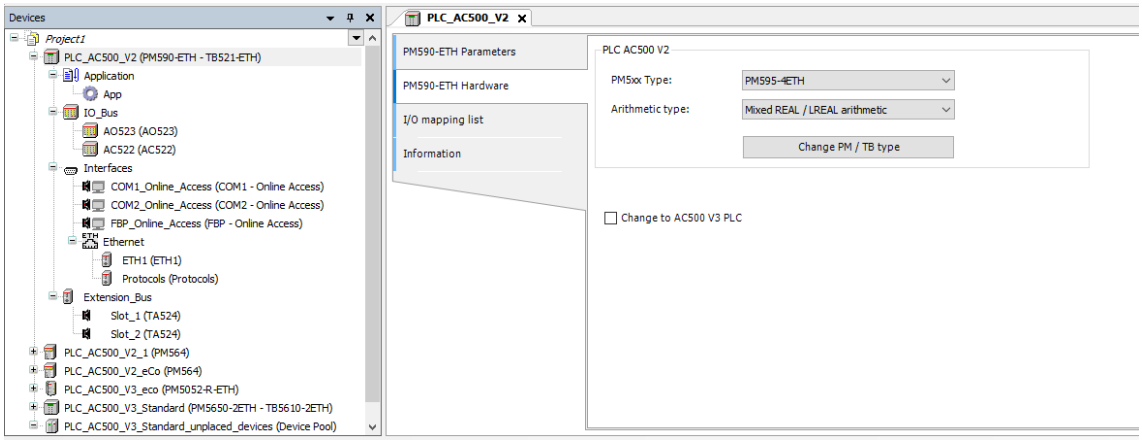
Target change options:

- AC500 V2 processor module ➜ PM595
- AC500-eCo V2 processor module ➜ PM595

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node.

3. Open the *"PM5<...> Hardware"* tab and select 'PM595-4ETH' from the *"PM5xx Type"* drop-down list.



4. With the *Arithmetic type* item, processing of Structured Text can be modified.
   - *"Mixed REAL/LREAL arithmetic"* (default value):
     Calculation of LREAL variables is extended to the extended co-domain of 64 bit. In general, we recommend to keep the default setting as this setting provides enough accuracy for code calculation.
   - With *"Only REAL arithmetic"* the LREAL variables are processed as REAL variables (co-domain of 32 bit).
5. Click *[Change PM / TB type]*.
   ⇨ The PM595 is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

Target change from a V2 processor module to a V3 processor module

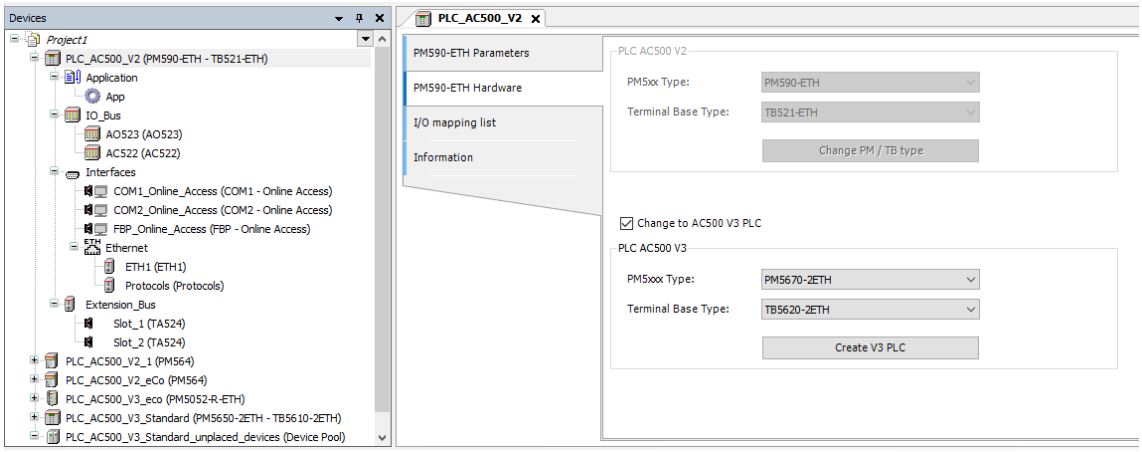
Target change options:

- AC500 V2 processor module ➜ AC500 V3 processor module
- AC500 V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500-eCo V3 processor module
- AC500-eCo V2 processor module ➜ AC500 V3 processor module

Procedure:

1. Close CODESYS.
2. Double-click the *PLC_AC500_V2 <...>* node and open the *"PM5<...> Hardware"* tab.

3. Enable *“Change to AC500 V3 PLC”* and select the desired V3 processor module from the *“PM5xx Type”* drop-down list.



4. Click *[Create V3 PLC]*.
   ⇨ The new V3 processor module is displayed in the navigation tree.
   ⇨ Change the node name of the processor module, if desired.

> *In case of a target change from AC500-eCo V2 to AC500-eCo V3, the I/O bus and Ethernet configuration is kept.*

PM5xy-ETH onboard Ethernet

Onboard Ethernet is provided for device types with -ETH extension, e.g. PM5xy-ETH. Further information is provided in the Onboard Ethernet Configuration section ↳ *Chapter 1.6.5.2.4.1.1 “Parameterization of PM5xy-ETH” on page 5860*.

Parameters of PM5x1-ARCNET (onboard ARCNET)

Add a processor module with onboard ARCNET (e.g. PM590-ARC) to the device tree. Double-click the node *“PM5x1_ARC_Internal_ARCNET”* to open the parameter settings for onboard ARCNET. Check whether the default configuration is to be changed.

The following parameters are available:

Parameter	Default value	Value	Description
Run config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the internal ARCNET Communication Module.

Parameter	Default value	Value	Description
Address see remark 1 ↳ *Chapter 1.6.5.2.3.6.1 "Remark 1: Setting behavior of the ARCNET node ID" on page 5848*	0	0...255	Address (node ID) of the ARCNET Communication Module
Tranmission rate see remark 2 ↳ *Chapter 1.6.5.2.3.6.2 "Remark 2: Tranmission rate of the ARCNET communication module" on page 5848*	2.5 MB/s	2.5 MB/s	Transmission set for the ARCNET Communication Module
		1.25 MB/s	
		625 kB/s	
		312.5 kB/s	
Extended timeout ET1/ET2	Very small net	Very small net (=0)	ARCNET timeout setting. The following applies: Bit 0 configures ET2 of the Communication Module Bit 1 configures ET1 of the Communication Module Value ET1 ET2 Meaning ---------- 0 0 0 Max. network expansion 2 km 1 1 0 2 0 1 3 1 1 for large networks
		Small net (=1)	
		Big net (=2)	
		Very big net (=3)	
Long packets	Enable	Enable	Enable long data packets (512 bytes)
		Disable	Incoming long data packets are received and dismissed. The SEND block indicates an error in case of long data packets.
Evaluate DIN on receipt see remark 3 ↳ *Chapter 1.6.5.2.3.6.3 "Remark 3: Check of DIN identifier on receipt" on page 5848*	Enable	Enable	Enable check of DIN identifier on receipt
		Disable	Disable check of DIN identifier on receipt

Remark 1: Setting behavior of the ARCNET node ID

Configured ARCNET Node ID			
#	In Project	In Display	Effect
1	0 [default]	0 [default]	Configuration error: Class=3 Comp=9 Dev=10 Mod=3 Ch=0 Err=26
2	0 [default]	!=0	No error
3	!=0 && != display setting	!=0	Configuration error: Class=3 Comp=9 Dev=10 Mod=3 Ch=0 Err=26
4	!=0 && == display setting	!=0	No error
5	!=0	0	No error, but display shows "Addr 0"

Remark 2: Tranmission rate of the ARCNET communication module

If the transmission rate set for the ARCNET Communication Module differs from the default value (2.5 MB/s), programming via ARCNET using the SoHard-ARCNET PC boards is no longer possible. The same transmission rate has to be set for all subscribers of the ARCNET network. The ARCNET PC boards are firmly set to 2.5 MB/s.

Remark 3: Check of DIN identifier on receipt

If the parameter "Evaluate DIN on receipt" is enabled (default setting), the following DIN identifiers are reserved:

DIN identifier		**Protocol**
Hex	**Dec**	
4F	79	"Online access" - Programming/OPC with IEC 61131-3 programming / AC1131
5F	95	5F_ARCNET (Ethernet functions for ARCNET)

DIN identifier		Protocol
Hex	**Dec**	
6F	111	PC331 programming (not used for AC1131/IEC 61131-3 programming)
7F	127	Default DIN identifier for data exchange with function blocks: ARC_REC, ARC_SEND, ARC_STO, ARC_INFO All DIN identifiers except the reserved identifiers can be used for data exchange.

If the parameter "Evaluate DIN on receipt" is disabled, programming and/or OPC via ARCNET is not possible!

"ARCNET data exchange" and "5F_ARC"

The protocols "ARCNET data exchange" and "5F_ARC" can be appended.

Right-click *“PM5x1_ARC_Internal_ARCNET”* node and select *[Add object]*.

The following parameters can be set for the "ARCNET data exchange" protocol:

Parameter	Default value	Value	Description
Size of receive buffer	8192	512...65535	Receive buffer size in bytes. The minimum size is equal to the maximum size of an UDP telegram.
Size of transmit buffer high prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with high priority.
Size of transmit buffer low prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with low priority.
Size of timeout buffer	2048	0...65535	Size of buffer (in bytes) for timeout data packets.
Number of header data	10	0...1464	Number of header data to be copied to the timeout buffer for timeout packages (in bytes).
Receive broadcast	Disable	Disable	Reception of broadcast telegrams disabled (data packets to all stations).

Parameter	Default value	Value	Description
		Enable	Reception of broadcast telegrams enabled (data packets to all stations).
Behavior on receive buffer overflow	Overwrite	Overwrite	Behavior on overflow of the receive buffer. The oldest data packets stored in the receive buffer are overwritten with the new incoming data packets.
		Reject	Behavior on overflow of the receive buffer. New incoming data are dismissed.

The following parameters can be set for the "5F_ARC" protocol:

Parameter	Default value	Value	Description
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

PM595-4ETH fieldbus communication

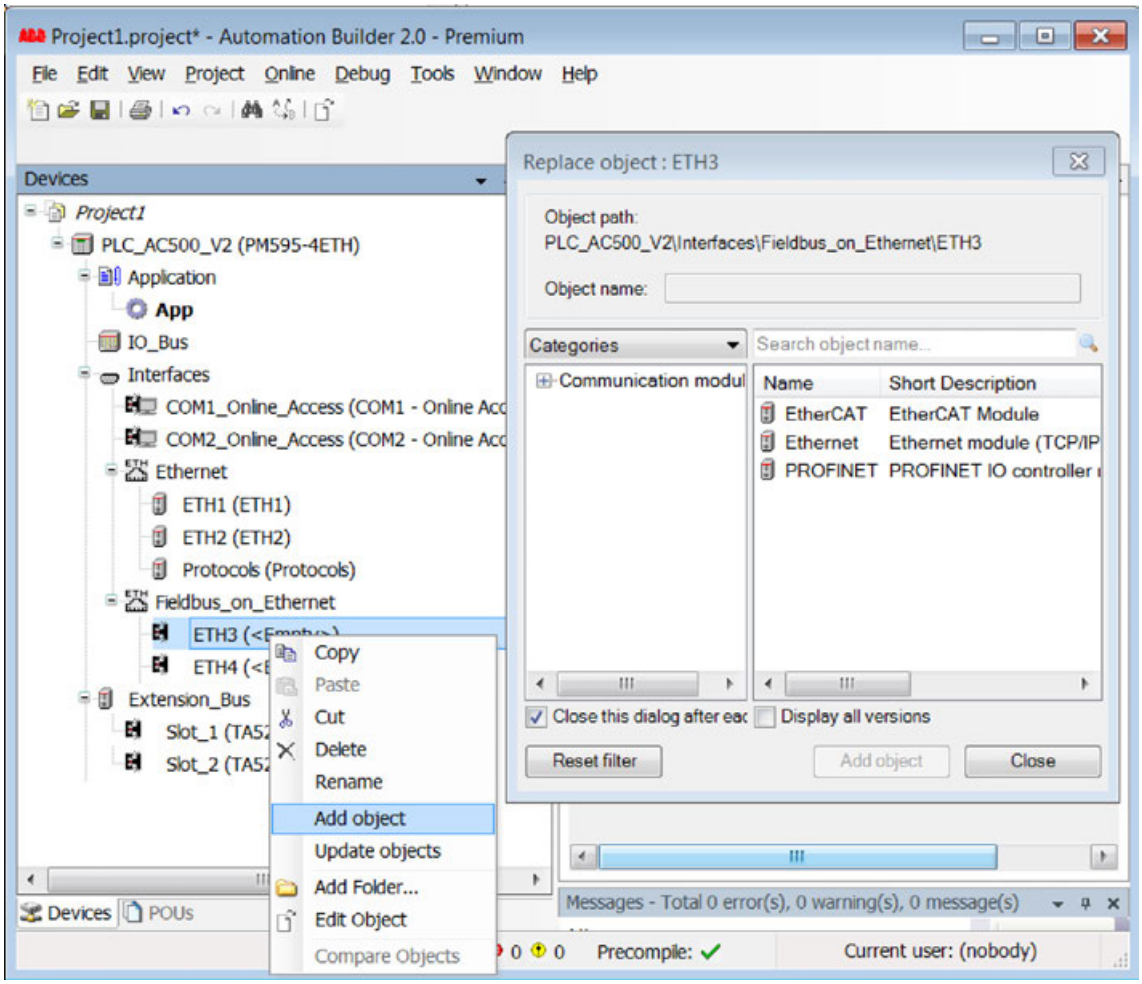
Configuration in Automation Builder

PM595-4ETH communication is provided either via a Communication Module (*“Extension_Bus”*) or onboard via *“Fieldbus_on_Ethernet”* -> node *“ETH3”* and *“ETH4”*.

In Automation Builder nodes ETH3 and ETH4 support 3 fieldbusses:

- EtherCAT
- Ethernet
- PROFINET

If one of them is selected the respective device will be added. This device can be extended with several communication devices. Those are identical to the Communication Modules, which can be added via *“Extension_Bus”*.



The configuration parameters are the same as for any other bus device.

Update firmware for fieldbusses on ETH3/4

After the fieldbus device has been added to the device tree, the user has to update the firmware of ETH3 or ETH4 in order to support the fieldbus in the PLC firmware.

The user can trigger the firmware update of ETH3/4 in the online view of the PLC (see ↳ *Chapter 1.6.5.1.7 “Firmware identification and update” on page 5786*).

Update Automation Builder 1.1 projects to Automation Builder 1.2

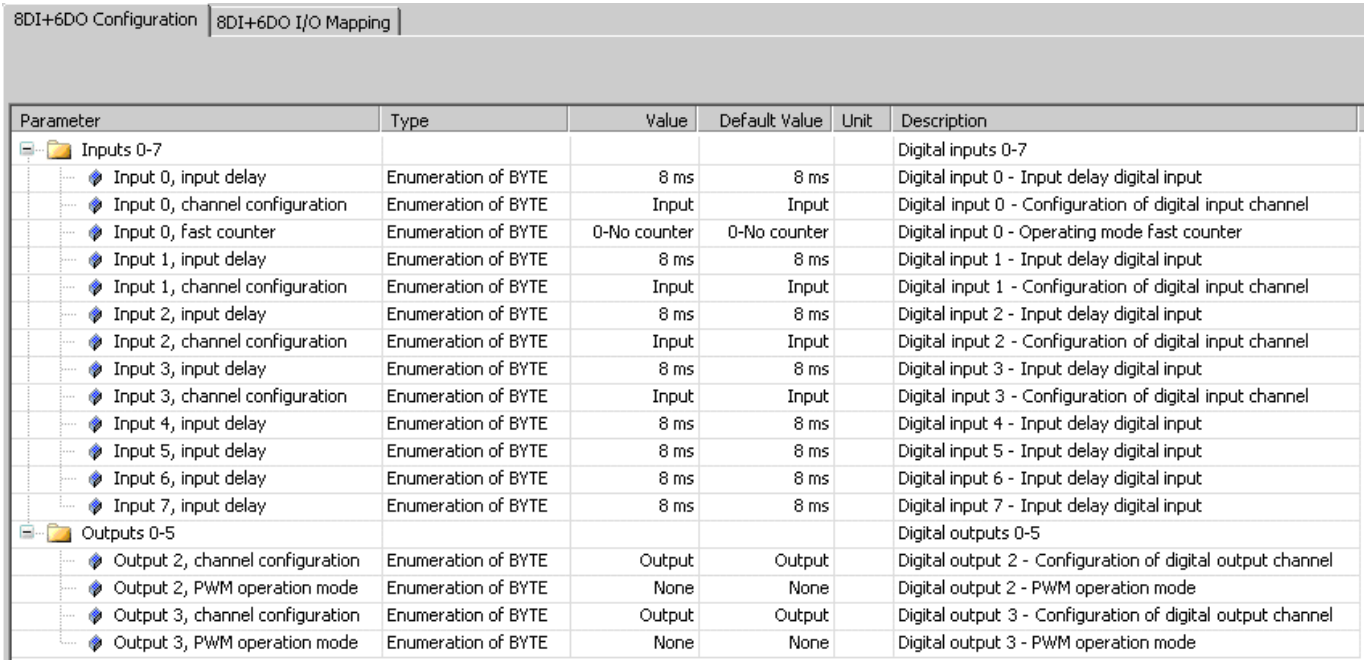
In Automation Builder 1.1 the ETH3/4 field bus can be configured as "<Empty>" or “PROFINET”. On project update from Automation Builder Version 1.1 to Version 1.2 the Automation Builder automatically copies any existing PROFINET configuration below ETH3/4 to the new structure in Automation Builder 1.2.

AC500-eCo onboard I/Os

Parameterization of the onboard I/Os for PM55x-xP

The inputs and outputs channels can be provided as WORD, BYTE and BOOL.

Double-click on *"OBIO (Onboard IO: 8DI+6DO)"* to open the 6DI+6DO configuration in the editor window.



Parameter	Type	Value	Default Value	Unit	Description
Inputs 0-7					Digital inputs 0-7
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 0 - Input delay digital input
Input 0, channel configuration	Enumeration of BYTE	Input	Input		Digital input 0 - Configuration of digital input channel
Input 0, fast counter	Enumeration of BYTE	0-No counter	0-No counter		Digital input 0 - Operating mode fast counter
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 1 - Input delay digital input
Input 1, channel configuration	Enumeration of BYTE	Input	Input		Digital input 1 - Configuration of digital input channel
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 2 - Input delay digital input
Input 2, channel configuration	Enumeration of BYTE	Input	Input		Digital input 2 - Configuration of digital input channel
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 3 - Input delay digital input
Input 3, channel configuration	Enumeration of BYTE	Input	Input		Digital input 3 - Configuration of digital input channel
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 4 - Input delay digital input
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 5 - Input delay digital input
Input 6, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 6 - Input delay digital input
Input 7, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 7 - Input delay digital input
Outputs 0-5					Digital outputs 0-5
Output 2, channel configuration	Enumeration of BYTE	Output	Output		Digital output 2 - Configuration of digital output channel
Output 2, PWM operation mode	Enumeration of BYTE	None	None		Digital output 2 - PWM operation mode
Output 3, channel configuration	Enumeration of BYTE	Output	Output		Digital output 3 - Configuration of digital output channel
Output 3, PWM operation mode	Enumeration of BYTE	None	None		Digital output 3 - PWM operation mode

The 8DI+6DO Onboard I/Os support the following channel functions:

Onboard I/O Type	Channel Function	Max. Number	Channel Name
Digital Onboard I/O	Digital Input	8	Channel 0..7
	Fast counter	2	Channel 0, 1
	Interrupt input	4	Channel 0..3
	Digital output	6	Channel 0..5
	PWM output	2	Channel 2..3

The following channel parameters for onboard I/Os can be configured:

Onboard I/O Type	Parameter	Channel Name	Default Value	Value	Description
Digital inputs	Input X, input delay	Channel 0..7	8 ms	0.1 ms	Configures 0.1 ms input delay
				1 ms	Configures 1 ms input delay
				8 ms	Configures 8 ms input delay
				32 ms	Configures 32 ms input delay

Onboard I/O Type	Parameter	Channel Name	Default Value	Value	Description
	Input X, channel configuration	Channel 0..7	Input	Input	Configures the channel as normal digital input
		Channel 0..3		Interrupt on rising edge	Triggers interrupt task when detecting the rising edge on the input channel
				Interrupt on falling edge	Triggers interrupt task when detecting the falling edge on the input channel
		Channel 0		Fast Counter	Configures the channel as fast counter
	Fast Counter *Chapter 1.6.5.2.3.8.4 "Fast counters in the onboard I/Os" on page 5856*	Channel 0, 1	No counter	0	No counter
				1	1 count up counter
				2	1 count up counter with release input
				3	2 UpDown counters
				4	2 UpDown counters (2nd on falling edge)
				5	1 UpDown counter dynamic set/ rising edge
				6	1 UpDown counter dynamic set/ falling edge
				7	1 UpDown counter directional discriminator
				8	Reserved
				9	1 UpDown directional discriminator x2

Onboard I/O Type	Parameter	Channel Name	Default Value	Value	Description
				10	1 UpDown directional discriminator x4
Digital outputs	Output X, channel configuration	Channel 2, 3	Output	Output	Configures the channel as normal digital output
				PWM	Configures the channel as PWM output
	Output X, PWM operation mode ↳ *Chapter 1.6.5.2.3.8.6 "Configuration of PWM outputs" on page 5858*		Millisec	Millisec	Configures ms as PMW time base
				Microsec	Configures µs as PMW time base

Parameterization of the onboard I/O for PM56x-xP

The 2 analog onboard inputs can be configured as 2 digital onboard inputs, so the maximal digital inputs number is 8. The input and output channels can be provided as WORD, BYTE and BOOL.

Double-click on *"I/O (onboard IOs)"* to open the DI+6DO+2AI+1AO configuration in the editor window:

8DI+6DO+2AI+1AO Configuration | 8DI+6DO+2AI+1AO I/O Mapping

Parameter	Type	Value	Default Value	Unit	Description
Digital + analog inputs					
Input 0, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 0 - Input delay digital input
Input 0, channel configuration	Enumeration of BYTE	Input	Input		Digital input 0 - Configuration of digital input channel
Input 0, fast counter	Enumeration of BYTE	0-No counter	0-No counter		Digital input 0 - Operating mode fast counter
Input 1, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 1 - Input delay digital input
Input 1, channel configuration	Enumeration of BYTE	Input	Input		Digital input 1 - Configuration of digital input channel
Input 2, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 2 - Input delay digital input
Input 2, channel configuration	Enumeration of BYTE	Input	Input		Digital input 2 - Configuration of digital input channel
Input 3, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 3 - Input delay digital input
Input 3, channel configuration	Enumeration of BYTE	Input	Input		Digital input 3 - Configuration of digital input channel
Input 4, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 4 - Input delay digital input
Input 5, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input 5 - Input delay digital input
Input AI0, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input AI0 - Input delay digital input
Input AI0, channel configuration	Enumeration of BYTE	Analog input 0...10 V	Analog input 0...10 V		Digital input AI0 - Configuration of analog input channel
Input AI1, input delay	Enumeration of BYTE	8 ms	8 ms		Digital input AI1 - Input delay digital input
Input AI1, channel configuration	Enumeration of BYTE	Analog input 0...10 V	Analog input 0...10 V		Digital input AI1 - Configuration of analog input channel
Digital + analog outputs					
Output 2, channel configuration	Enumeration of BYTE	Output	Output		Digital output 2 - Configuration of digital output channel
Output 2, PWM operation mode	Enumeration of BYTE	None	None		Digital output 2 - PWM operation mode
Output 3, channel configuration	Enumeration of BYTE	Output	Output		Digital output 3 - Configuration of digital output channel
Output 3, PWM operation mode	Enumeration of BYTE	None	None		Digital output 3 - PWM operation mode
Output 0, channel configuration	Enumeration of BYTE	Analog output 0...10 V	Analog output 0...10 V		Analog output 0 - Configuration of analog input channel

The 6DI+6DO+2AI+1AO onboard I/Os support the following channel functions:

Onboard I/O Type	Channel Function	Max. Number	Channel Name
Digital onboard I/O	Digital Input	8	Digital input channel 0..5 Analog input channel AI0..AI1 *
	Fast counter	2	Input channel 0, 1
	Interrupt input	4	Input channel 0..3
	Digital output	5	Output channel 0..5
	PWM output	2	Output channel 2..3
Analog onboard I/O	Analog input	2	Input channel AI0..AI1 *
	Analog output	1	Output channel AO0

* These 2 analog inputs can be configured as 2 digital inputs

Processor module PM56x-xP has a maximum of 8 digital onboard inputs and 6 digital onboard outputs. The channel parameters for onboard digital I/Os can be found in table onboard I/Os for PM554 *Table on page 5852*. Analog input channels AI0&AI1 on PM564 are for digital input channels 5&7 on PM554.

Table 728: PM564: Channel parameters for onboard analog I/Os

Onboard I/O Type	Parameter	Channel Name	Default Value	Value	Description
Analog inputs	Input delay digital input	Channel AI0&A1	8 ms	0.1 ms	Configure 0.1 ms input delay
				1 ms	Configure 1 ms input delay
				8 ms	Configure 8 ms input delay
				32 ms	Configure 32 ms input delay
	Channel configuration		Digital input	Digital input	Configure the channel as normal digital input
				Analog input 0..10 V	Configure the channel as 0..10 V analog input
Analog outputs	Channel configuration	Channel AO0	Analog output 0..10 V	Analog output 0..10 V	Configure the channel as 0..10 V analog output

Onboard I/O Type	Parameter	Channel Name	Default Value	Value	Description
				Analog output 0..20mA	Configure the channel as 0..20 mA analog output
				Analog output 4..20 mA	Configure the channel as 4..20 mA analog output

Default value of channel AI0&AI1 is digital input. The user has to change the configuration parameter to enable the analog input function.

Mapping of the I/O channels

Double-click on IO (Onboard IOs) to open the editor window.

->The tab xDI+yDO I/O Mapping shows the current settings of the I/O mapping.

See Symbolic Names for Variables, Inputs and Outputs for further details on mapping
↳ *Chapter 1.6.5.2.2.1.2 “Symbolic names for variables, inputs and outputs” on page 5815*.

Fast counters in the onboard I/Os

General details on fast counters see ↳ *Chapter 1.6.4.1.10 “Fast counters” on page 5498*.

Details on the configuration see ↳ *Chapter 1.6.5.2.9.8.2 “Configuration for onboard I/Os” on page 6070*.

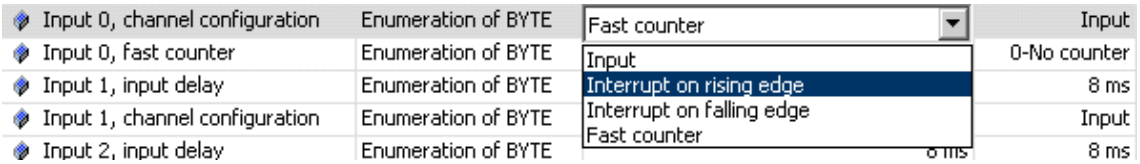
Configuration of interrupt inputs

The processor module PM55x-xP and PM56x-xP provide 4 onboard interrupt inputs. For using the interrupt functionality, the channels I0…I3 have to be configured accordingly.

In the channel parameters of onboard I/O / digital inputs, the channels I0...I3 can be specified as interrupt inputs. 2 interrupt values can be chosen for each channel:

- Interrupt on rising edge
- Interrupt on falling edge

The interrupt mode can be selected in parameter Input X, fast counter of the parameters list:



Creating an interrupt task

If one or more channels of the onboard I/Os are configured as interrupt inputs, a corresponding interrupt task has to be created to enable the processing of the interrupt(s).

For this purpose, a new task has to be added in the task configuration:

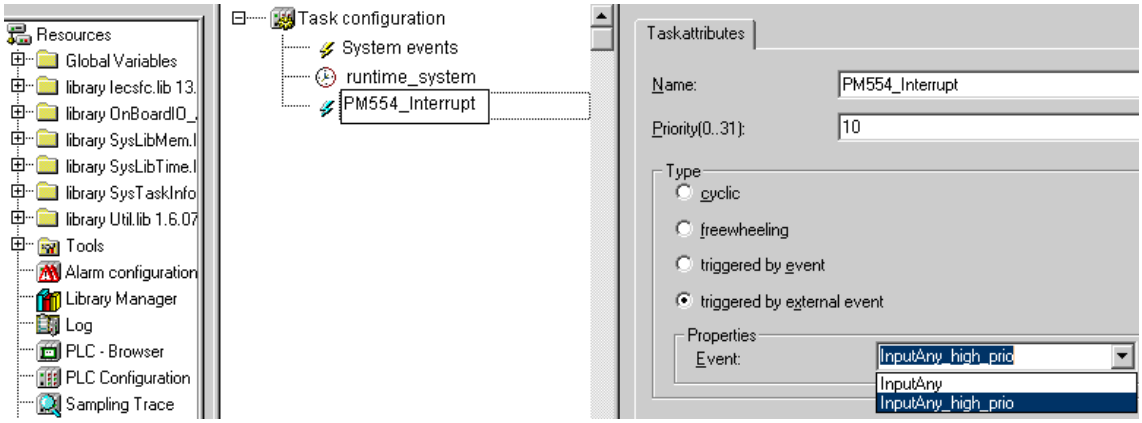
- Enter the task name
- Set the task type to triggered by external event
- Specify the event that triggers the task

2 types of interrupt tasks are available in the Event list box:

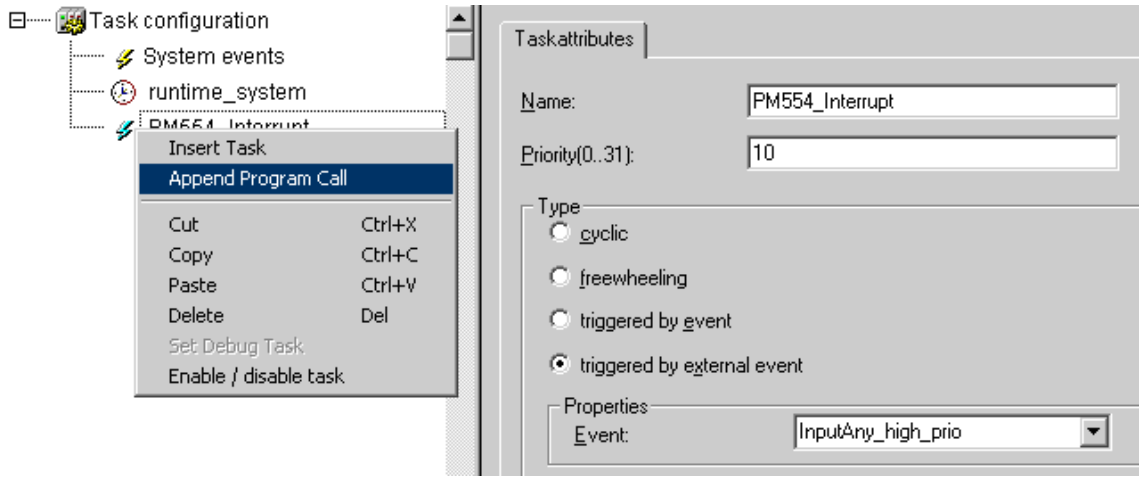
- InputAny
  The task is triggered by any interrupt with the priority specified in the Priority field (0...31).
- InpuAny_high_prio
  The task is triggered by any interrupt with highest priority, i. e. with a priority higher than the max. adjustable "0" and higher than the priority of the communication task. In this case, the priority (0...31) specified in the Priority field has no significance.

> *If the interrupt task is started with highest priority, the program execution time must not be longer than approx. 400 µs. Otherwise online access is no longer possible.*

In the example, the task is named PM554_Interrupt. It is configured as a high-priority interrupt. The task type is triggered by external event and the event to trigger the task is InputAny_high_priority.



Like for all other tasks, a program call has to be assigned to the task.



In the example, the program PM554_Interrupt_1() shall be started with any interrupt from onboard I/Os.

Structure of the interrupt program

The following block contained in the library OnBoardIO_AC500_V13.lib is available for the interrupt program:

- ONB_IO_INT_IN: Determination of the interrupt initiating source

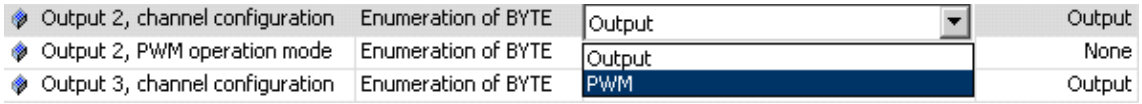
It is possible to start one interrupt task. This task can be started by any channel (I0...I3) configured as interrupt input. Therefore, it is necessary for the interrupt program to differentiate which channel(s) triggered the interrupt in order to enable the processing of the corresponding actions.

The information whether a channel (I0...I3) has triggered an interrupt since the last call of the block is provided by the outputs IN0...IN3 of the block ONB_IO_INT_IN. This is why this block always has to be called at the beginning of the interrupt program, if more than one channel is configured as interrupt input.

The interrupt program will be called every time if a pulse edge is detected. But if pulse edges are occurring simultaneously on several inputs which are configured as interrupt inputs, the interrupt program can be called once.

Configuration of PWM outputs

The processor module PM55x-xP and PM564-xP provide up to 2 PWM output channels with a maximum frequency of 20 KHz. The parameter of PWM output channel of Onboard I/O must be configured before it can be used. User should take these steps to configure the PWM output function.

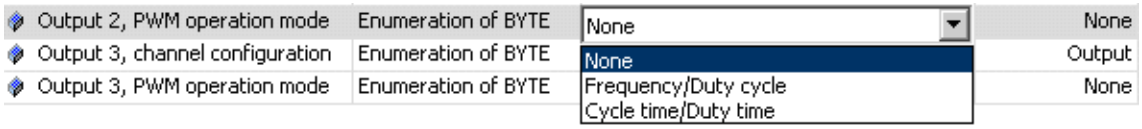


The 2 outputs can be controlled in 2 different configuration modes:

- Frequency mode
- Cycle time mode

Relay-type outputs of Onboard I/Os cannot be used for PWM functionality.

The mode can be configured with the parameter Output X, PWM operation mode in the parameters list:



In frequency mode, by setting the FREQ and DUTY_CYCLE input variables of PWM channel, the Onboard I/O can generate 125 Hz to 20 kHz frequency with 0..100 % duty cycle waveform in output channels 2 and 3.

In the cycle time mode, by setting the CYCLE_TIME and CYCLE_DUTY input variables of PWM channel, the Onboard I/O can generate 125 Hz to 20 kHz frequency with 0..100 % duty cycle waveform in output channels 2 and 3.

The PWM can be controlled via the corresponding mapped parameters in tab xDI+yDO I/O Mapping of the editor window. See Symbolic Names for Variables, Inputs and Outputs for further details on mapping ↳ *Chapter 1.6.5.2.2.1.2 “Symbolic names for variables, inputs and outputs” on page 5815*.

PWM						
		State byte PWM 2	%IB4002	BYTE		PWM 2 - State byte
		State byte PWM 3	%IB4003	BYTE		PWM 3 - State byte
		Control byte PW...	%QB4001	BYTE		PWM 2 - Control byte
		PWM 2, frequen...	%QW2001	WORD		PWM 2 - Frequency / Cycle time
		PWM 2, duty cy...	%QW2002	WORD		PWM 2 - Duty cycle / Duty time
		Control byte PW...	%QB4006	BYTE		PWM 3 - Control byte
		PWM 3, frequen...	%QW2004	WORD		PWM 3 - Frequency / Cycle time
		PWM 3, duty cy...	%QW2005	WORD		PWM 3 - Duty cycle / Duty time

Table 729: Channel description:

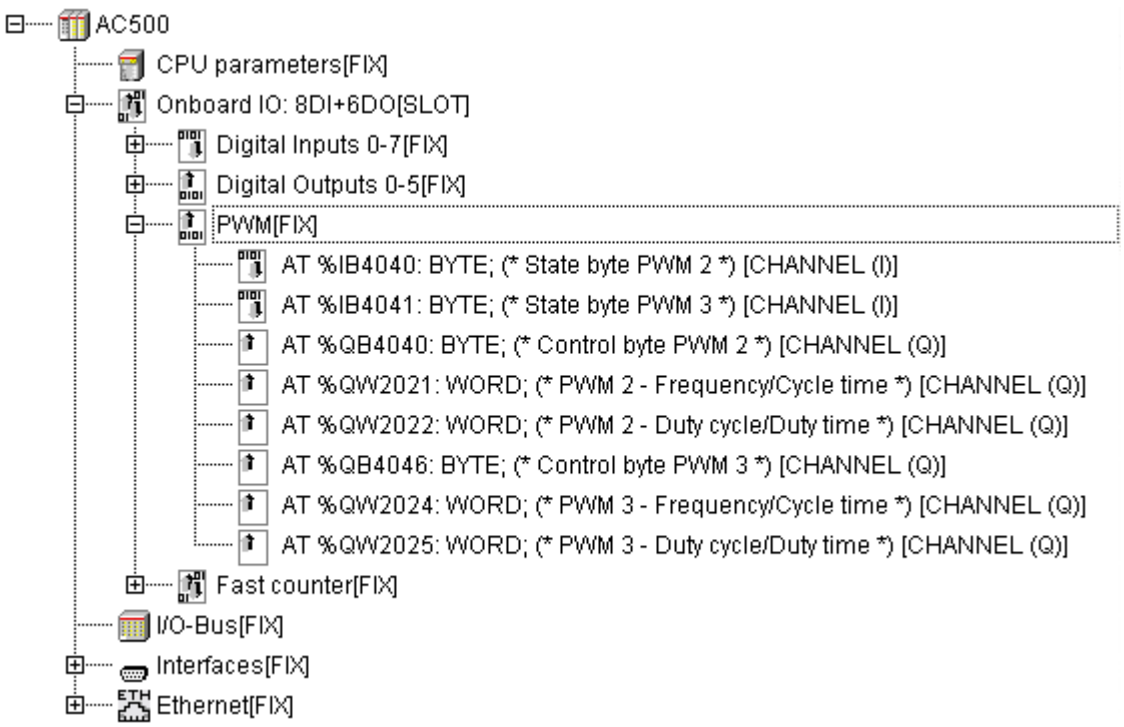
Channel	Direction	Width
State byte PWM X	Input	BYTE
Control byte PWM X	Output	BYTE
PWM X, frequency / cycle time	Output	DWORD
PWM X, duty cycle / duty time	Output	DWORD

Operating the PWM output with user program

The following methods can be used to configure PWM outputs:

- Configuration variables
- Function blocks

Using a configuration variable



Using function blocks

The following function blocks of the library OnBoardIO_AC500_V13.lib can be used to operate the PWM outputs with help of user program.

Group: PWM_OBIO	
ONB_IO_PWM_FREQ	Generate PWM Signal with Frequency and Duty Cycle on Onboard I/O
ONB_IO_PWM_TIME	Generate PWM Signal with Cycle Time and Duty Cycle on Onboard I/O

1.6.5.2.4 Onboard Ethernet configuration

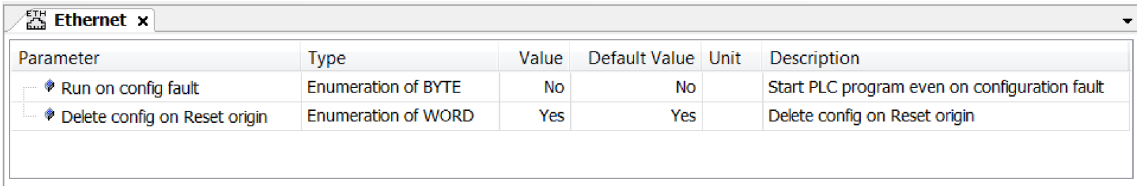
PM5xy-ETH - Onboard Ethernet

Processor modules with 2 onboard Ethernet interfaces:

- *Configure only 1 interface as default gateway.*
- *In case of 2 configured interfaces only the latter will be executed.*

Parameterization of PM5xy-ETH

Double-click the *"Ethernet"* node to open the onboard Ethernet configuration in the editor window:



Processor modules with 2 onboard Ethernet interfaces:

- *Configure only 1 interface as default gateway.*
- *In case of 2 configured interfaces only the latter will be executed.*

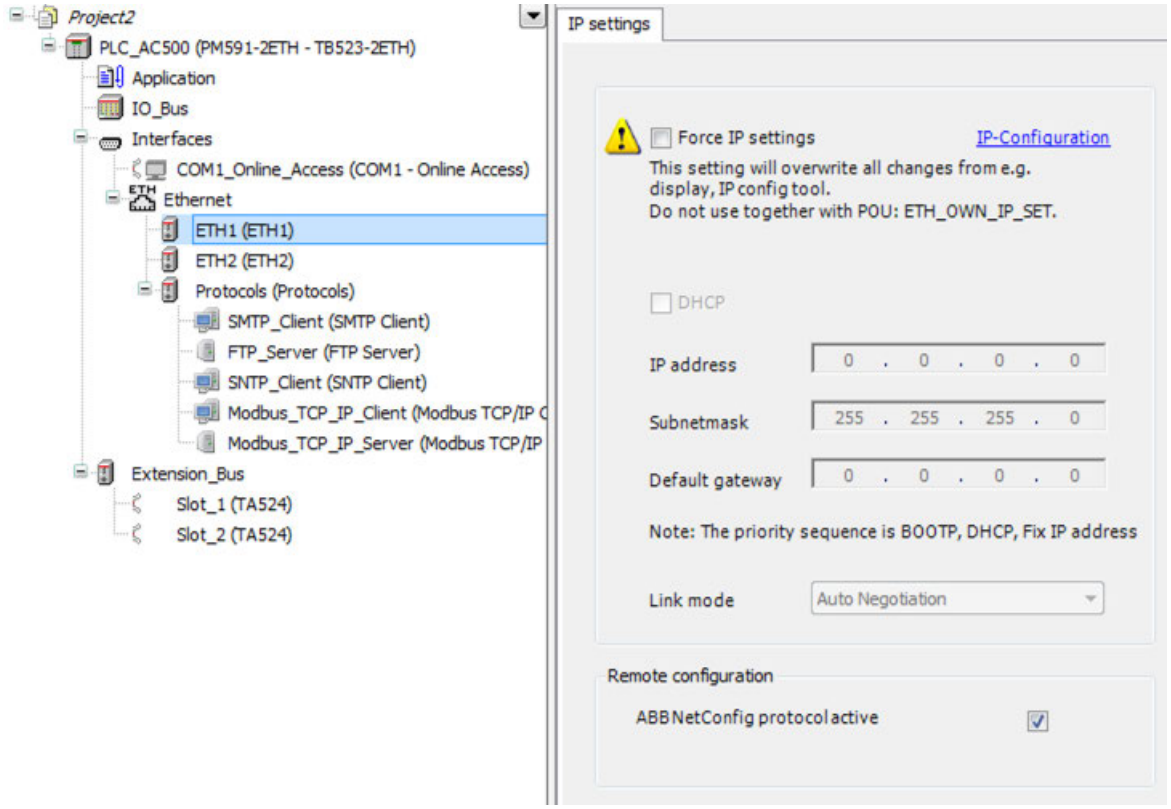
The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the Onboard Ethernet Module.

Parameter	Default value	Value	Description
Delete config on Reset origin	Yes	Yes	The Ethernet configuration is deleted after Reset origin.
		No	The Ethernet configuration (e.g. IP address) is still available after Reset origin.

Configuration of the IP settings

In the Automation Builder project, double-click the Ethernet node to open the *"IP settings"*:



Processor modules with 2 onboard Ethernet interfaces:

- *Configure only 1 interface as default gateway.*
- *In case of 2 configured interfaces only the latter will be executed.*

Table 730: Parameters:

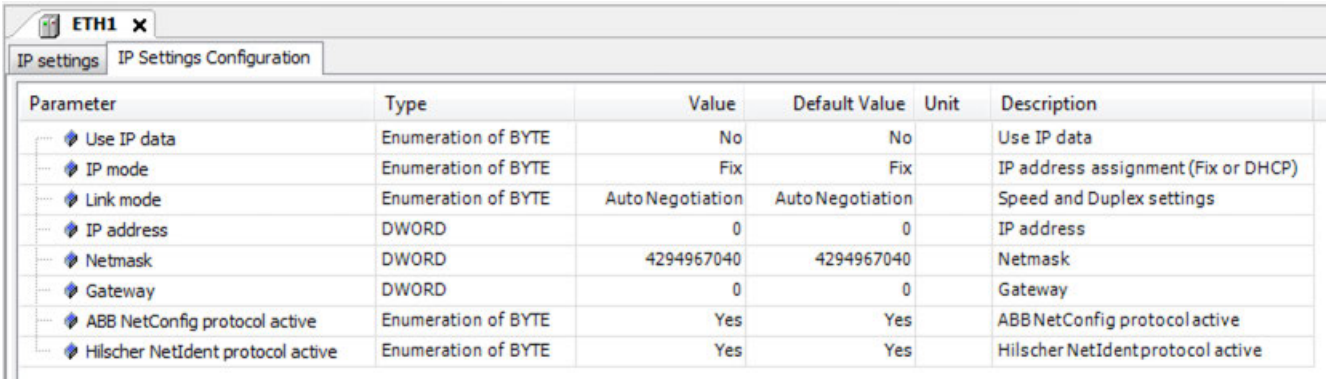
Parameter	Default	Value	Description	Parameter [1]) ↯ *Further information on page 5861*)
Force IP settings	Disabled	Disabled	The IP settings from this editor are not used.	Use IP data
		Enabled	The IP settings from this editor are used. After download of the user application and subsequent logoff, the new IP settings are used temporarily until power off the device. For a permanent usage of the IP settings, create a boot project.	
DHCP	Disabled	Disabled	The DHCP client services are disabled. A static IP address must be used.	IP mode (+ DHCP)
		Enabled	The DHCP client services are enabled. The IP address of the device will be set by a DHCP server in the network.	
IP address	0.0.0.0	Valid IP address	IP address of the device.	IP address
Subnetmask	255.255.255.0	Valid subnet mask	Subnet mask for the device.	Netmask
Default gateway	0.0.0.0	Valid gateway address	Default gateway address for the device.	Gateway
Link mode	Auto negotiation	Auto negotiation	The link mode will be detected automatically.	Link mode
		10 Mbit Half-Duplex	The link mode is statically set to 10 Mbit half-duplex.	
		10 Mbit Full-Duplex	The link mode is statically set to 10 Mbit full-duplex.	

Parameter	Default	Value	Description	Parameter [1]) ↳ *Further information on page 5862*)
		100 Mbit Half-Duplex	The link mode is statically set to 100 Mbit half-duplex.	
		100 Mbit Full-Duplex	The link mode is statically set to 100 Mbit full-duplex.	
ABBNetConfig protocol active	Enabled	Enabled	Remote configuration: ABB NetConfig protocol is enabled.	
		Disabled	Remote configuration: ABB NetConfig protocol is disabled.	

Remarks:

[1]): Generic Device Configuration View Parameters

For a list of all available parameters, open the *"IP Settings Parameters"*. This tab is only visible after enabling parameter 'Show generic device configuration views' under *"Tools ➔ Options"*:



Configuration of the IP settings with the IP configuration tool

The IP address for AC500 devices can be set or changed in Automation Builder using

- the IP configuration tool which is described in the following.
- the 'Communication Settings'. ↳ *Chapter 1.6.5.2.2.2.3 "Configuration of communication via Ethernet (TCP/IP)" on page 5829*

As an alternative the IP address can be changed at the hardware device itself. ↳ *Chapter 1.6.4.1.5.4 "Description of the function keys" on page 5426*

The IP configuration tool:

The IP configuration tool can be used

- to set or change the IP address of devices.
  ↳ *Chapter 1.6.5.2.4.1.3.2.2 "Changing the IP address" on page 5868*
- to scan the network for available hardware devices.
  ↳ *Chapter 1.6.5.2.4.1.3.2.1 "Network scan" on page 5866*

- to update the firmware of devices.
  This functionality is only supported if the IP configuration tool is used stand-alone.
  ↳ *Chapter 1.6.5.2.4.1.3.2.3 “Firmware update” on page 5869*
- to activate certain functionality on hardware devices.
  This feature is only available on AC500 V3 devices.
  ↳ *Chapter 1.6.5.2.4.1.3.2.4 “Blink functionality” on page 5873*

The IP configuration tool is part of Automation Builder and can be called via *“Tools ➔ IP-Configuration”*.

Further the IP configuration tool can be used stand-alone without an Automation Builder application running. The stand-alone variant requires a separate installation via the Installation Manager ↳ *Chapter 1.6.5.2.4.1.3.1 “Stand-alone installation” on page 5864*.

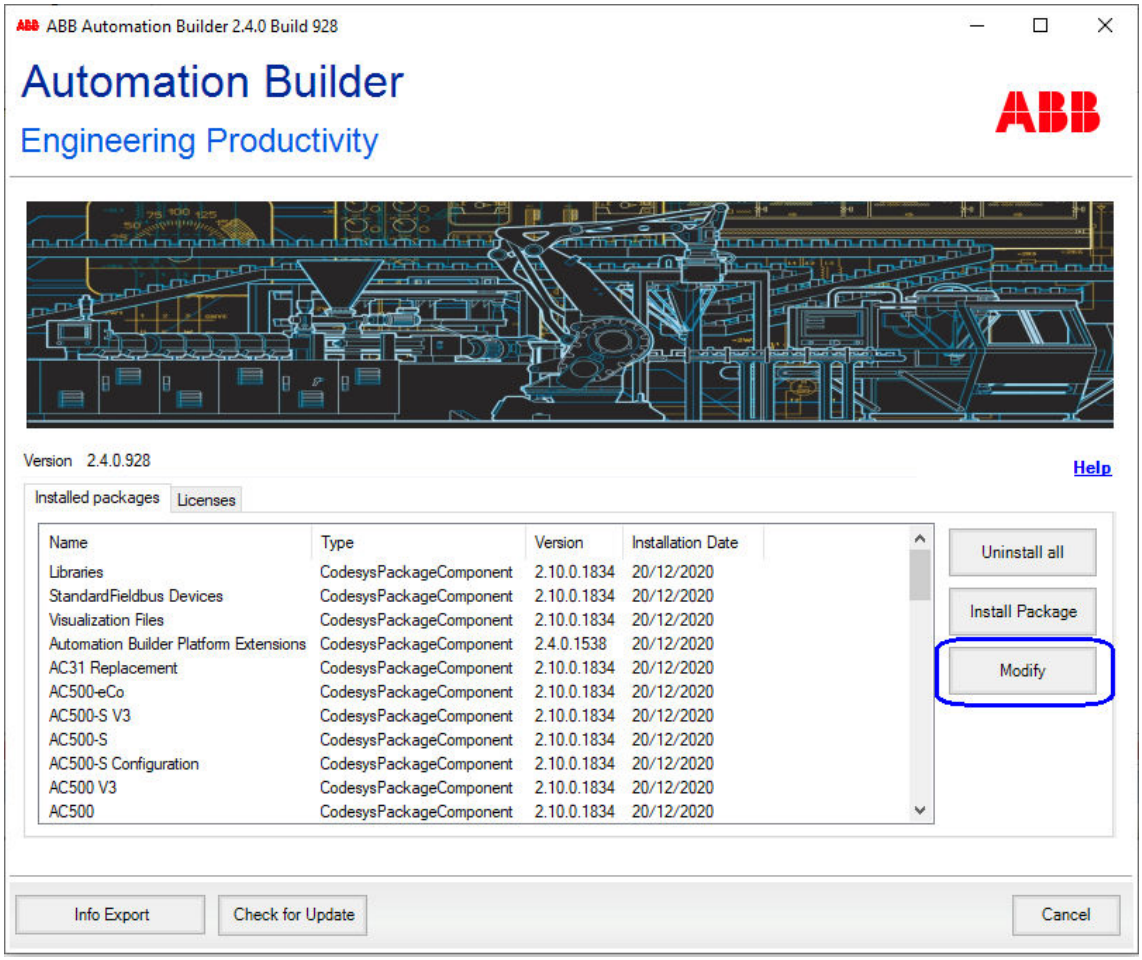
After the installation, the IP configuration tool is started via .exe file / desktop icon.

Some functionality is only supported if the IP configuration tool is used stand-alone, e.g. for firmware updates for communication interface devices.

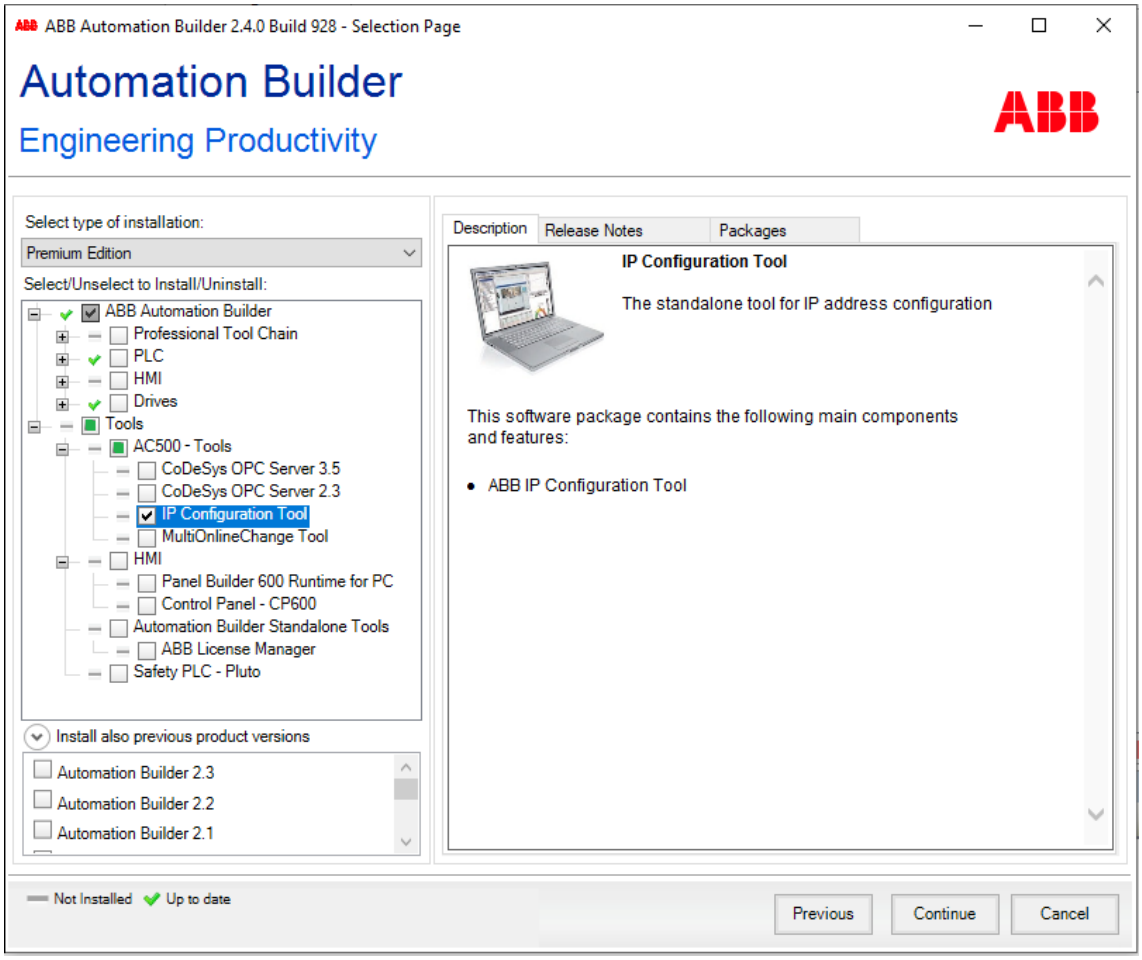
Stand-alone installation

The IP configuration tool is part of Automation Builder and can be called via “Tools ➔ IP-Configuration”. A separate installation is only required if the IP configuration tool shall be used stand-alone.

1. Open the Installation Manager in Automation Builder: *"Tools ➔ Installation Manager"*.
2. Close all other instances of Automation Builder as only one instance of the program can be executed at a time.



3. Click *"Modify"* and select the *"IP Configuration Tool"* from the structure tree.



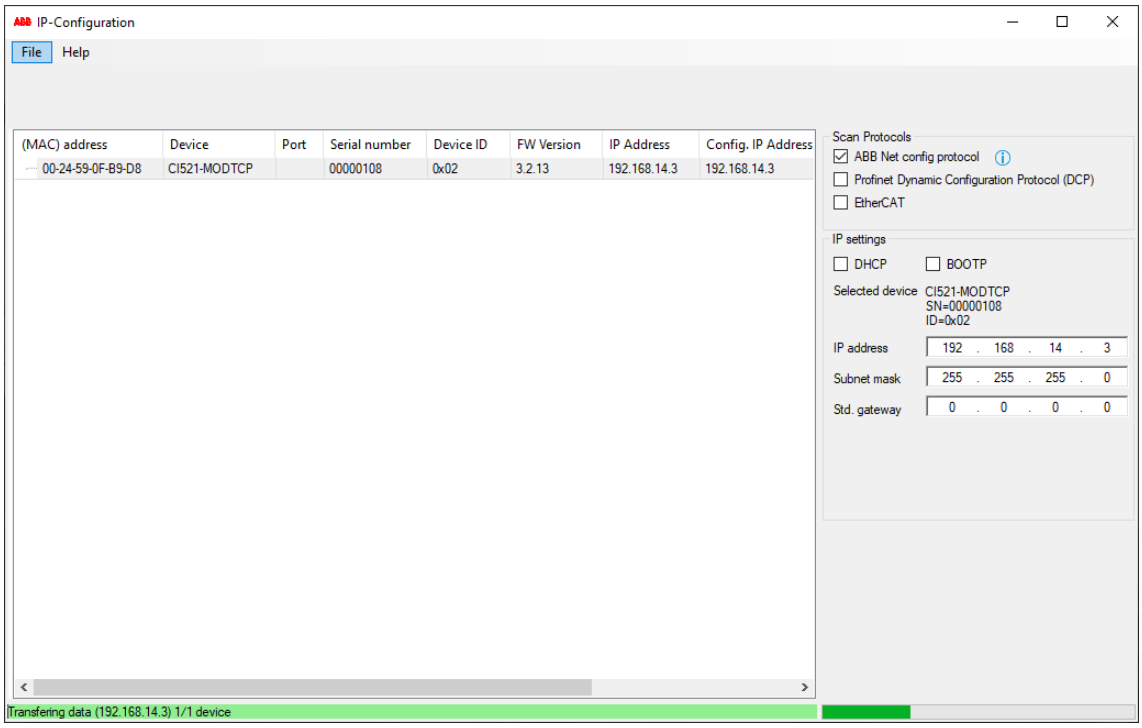
4. Click *"Continue"* to start the installation.
   - ⇨ After a successful installation the IP configuration tool is available as stand-alone tool (.exe).
   - ⇨ To start the IP configuration tool, click the new created desktop icon.

Using the tool functions

Network scan

With a network scan all devices that have been found in the network by the scan process are listed, i.e. ABB devices such as AC500 processor modules, AC500 communication interface modules or ABB Drives.

1. Start the IP configuration tool in Automation Builder (*"Tools ➔ IP-Configuration"*) or start it stand-alone (.exe).
2. The *"IP-Configuration"* dialog opens. Define the device type for the network scan by selecting the desired option under *"Scan Protocol"*:
   - *"ABB Net config protocol"*:
     Use this option for AC500 devices such as processor modules, CI5xx-Modbus devices or ABB Drives. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection.
   - *"Profinet Dynamic Configuration Protocol (DCP)"*:
     Use this option for PROFINET communication interface modules. The device(s) to be scanned must be connected to the PC via a direct Ethernet connection (not via CM579).
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 5871
   - *"EtherCAT"*:
     Use this option for EtherCAT communication interface modules. The Ethernet cable must be connected directly to the first EtherCAT slave device of the EtherCAT fieldbus. Ensure that no EtherCAT master device is available on the bus when a scan is performed.
     "Emergency" option: Enable this option to check on failures in the EtherCAT assembly during the scan process, i.e. a frame loss or interchanged ports. Errors are displayed.
     For the scan, a NPcap driver needs to be installed separately.
     Step 4 on page 5871
3. Click *[Scan]* to start the scan process.



4. All devices that have been found in the network are listed including hardware and connection details. The following details can be changed under *"IP settings"*:

   ⇨ ● *"IP Address"*:
   Current IP address of the device.
   ● *"Conf. IP Address"*:
   Configured IP address of the device. A changed IP address will update this column.
   ● *"FW Version"*:
   Current installed firmware version of the device. This field is visible not until a first network scan. If this field is still empty after a network scan, check on connection errors.
   Chapter 1.6.5.2.4.1.3.3.1 "Trouble-shooting for firmware update" on page 5874

The IP address of some devices, e.g. EtherCAT devices cannot be changed.

Changing the IP address

1. In order to change the IP address of devices perform a network scan.

   Chapter 1.6.5.2.4.1.3.2.1 "Network scan" on page 5866

2. Select a device from the list and select the appropriate protocol under *"Scan protocol"*.

   "DHCP" or *"BOOTP"* option: If required, DHCP or BOOTP can be used to receive the IP address for the device from the server.

   "IP address", *"subnet mask"*, *"Std. gateway"*: Use these fields to change the IP address settings including the settings for the subnet mask and the standard gateway. Ensure that the combination of connection settings is correct.

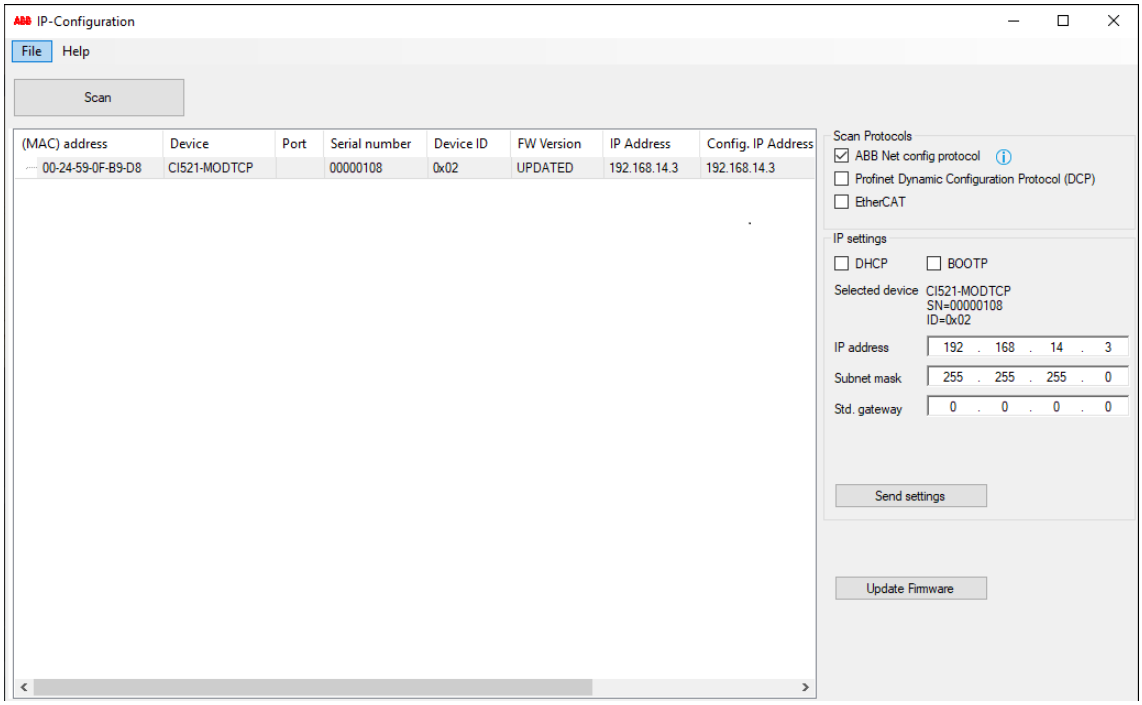
   "Check subnet configuration" on page 5874

Note for CI52x-Modbus devices

Consider the behavior of CI52x-Modbus devices if the last number of the IP address is set to "0".

"Check last number of IP address" on page 5875

3. Change the settings for the IP configuration and click *[Send settings]* to transmit the data to the device.



⇨

Note for PROFINET devices

The device name of PROFINET devices can be edited. If changing the name, ensure the following rules apply:

- *Labels must be separated by "."*
- *Total length: 1 to 240*
- *Label length: 1 to 63*
- *Labels can consist of characters [a-z] and numbers [0-9]*
- *Labels are not allowed to start with "-"*
- *Labels are not allowed to end with "-"*

4. In order to keep all IP changes after a power cycle, the settings can be stored permanently. Confirm the prompted message during the scan process.

Firmware update

The firmware of AC500 communication interface modules can be updated with the IP configuration tool.

For this, the IP configuration tool must be used as stand-alone variant.

↳ *Chapter 1.6.5.2.4.1.3.1 “Stand-alone installation” on page 5864*

It is not possible to perform a firmware update out of Automation Builder.

- *For PROFINET communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.3.3.*
- *For EtherCAT communication interface modules a firmware update is only supported for devices with firmware version ≥ 2.1.4.*
- *For Modbus communication interface modules a firmware update is only supported for devices with firmware version ≥ 3.2.13.*

Requirements: **Before the firmware update**

- Ensure a fast and stable network connection
- Close all unused applications on the executing PC
- Stop the communication between AC500 PLC and the communication interface module that shall be updated

During the firmware update

- Do not close the IP configuration tool
- Do not open Automation Builder software or any other application
- Do not switch-off the communication interface module that shall be updated
- Do not disconnect the Ethernet connection of a communication interface module or the executing PC

The firmware update will stop the operation of the affected device(s). Hence, the device(s) will become unresponsive for 1 - 2 minutes.

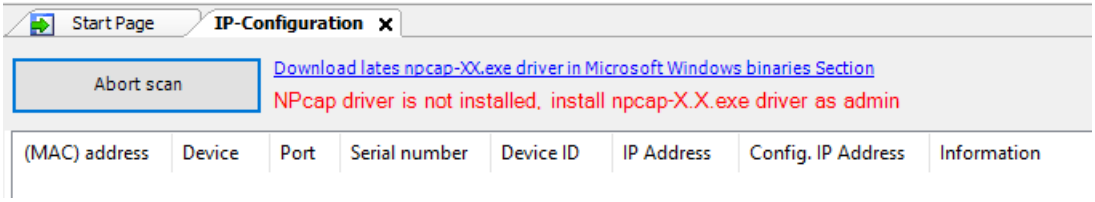
Procedure:

1. Start the IP configuration tool stand-alone (.exe).
2. Perform a network scan.

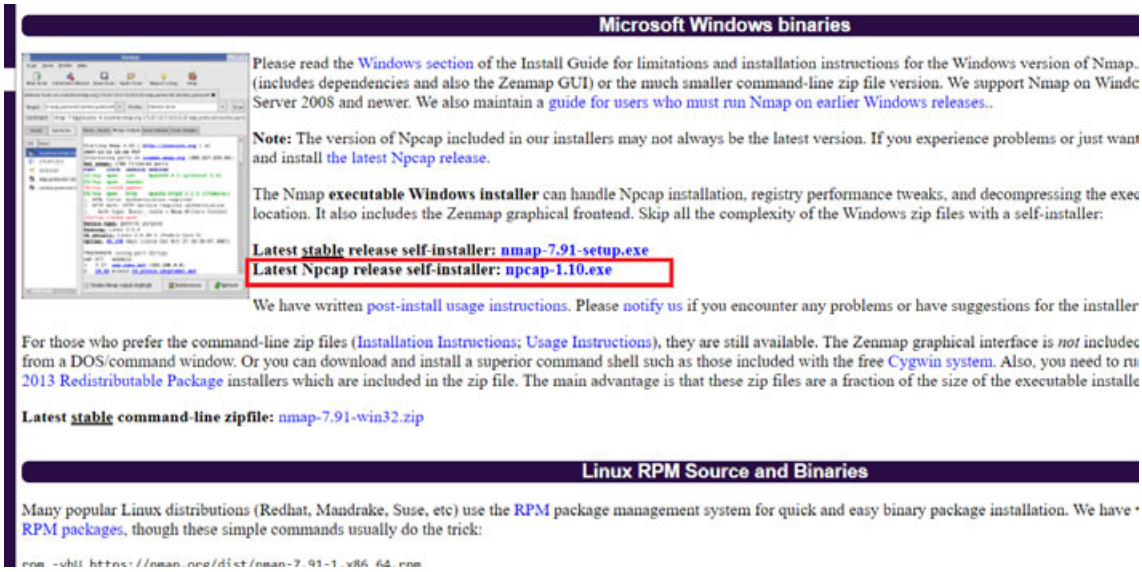
   Chapter 1.6.5.2.4.1.3.2.1 “Network scan” on page 5866
3. Select the devices that shall be updated from the list and click *[Scan]* to trigger the scan process.

   A multiple selection of several devices is possible via control key, however, ensure to select only devices of the same protocol at a time. Otherwise the firmware update fails.

4. This step is only required for devices that require an installed NPcap driver. In this case an appropriate message including a download link is prompted in the IP-Configuration dialog:



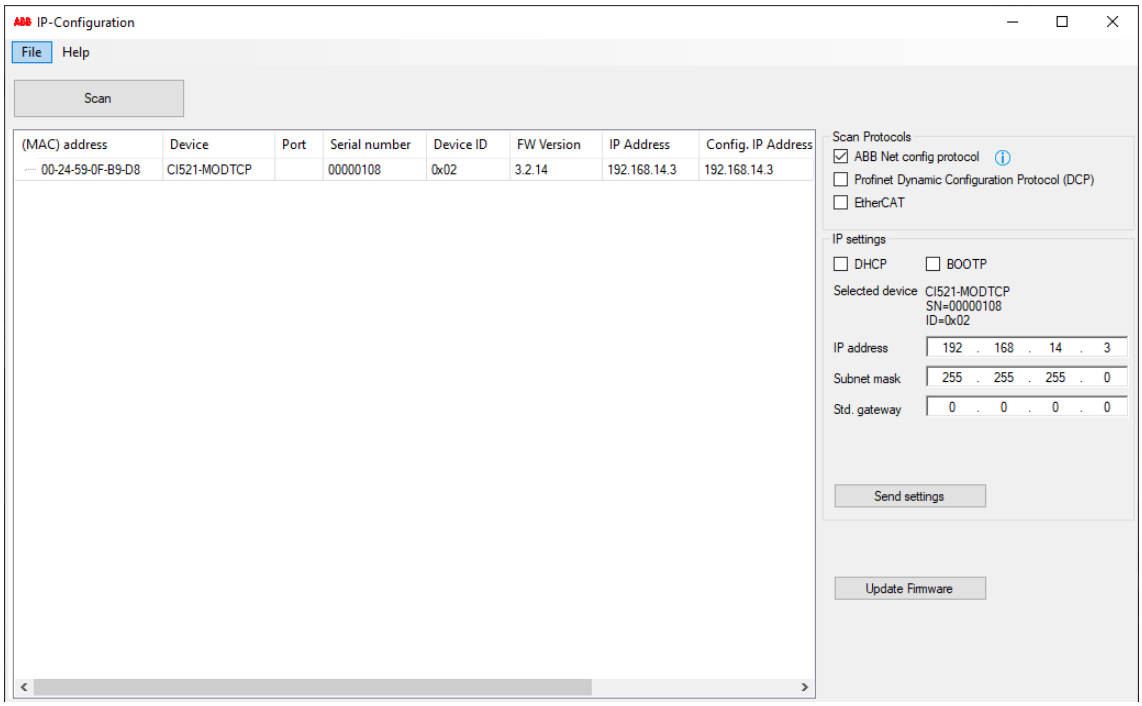
⇨ Click on the displayed link *https://nmap.org/download.html* and download the latest version of the *npcap-X.X.exe* file.



⇨ After the download, execute the file as administrator and restart the scan process.

⇨ The devices that have been scanned are listed.

5. Click *[Update Firmware]* to start the firmware update for the selected devices.



6. For CI50x, CI51x and CI52x devices a signature check is started. Select the appropriate firmware update file (*.bin) for the device(s). Example: C:\AC500\AC500_CI52x_Firmware_V3.2.8.bin.

   After a successful signature check the firmware update file (*.bin) and the respective signature file (*.bin.sig) are transferred to the device. This can last up to 3 minutes.

   If the signature check fails, check the availability of the *.bin file and the *bin.sig file.

   ↳ *"Signature check" on page 5875*

7. A status check followed by a device reboot followed by a second status check is performed automatically.

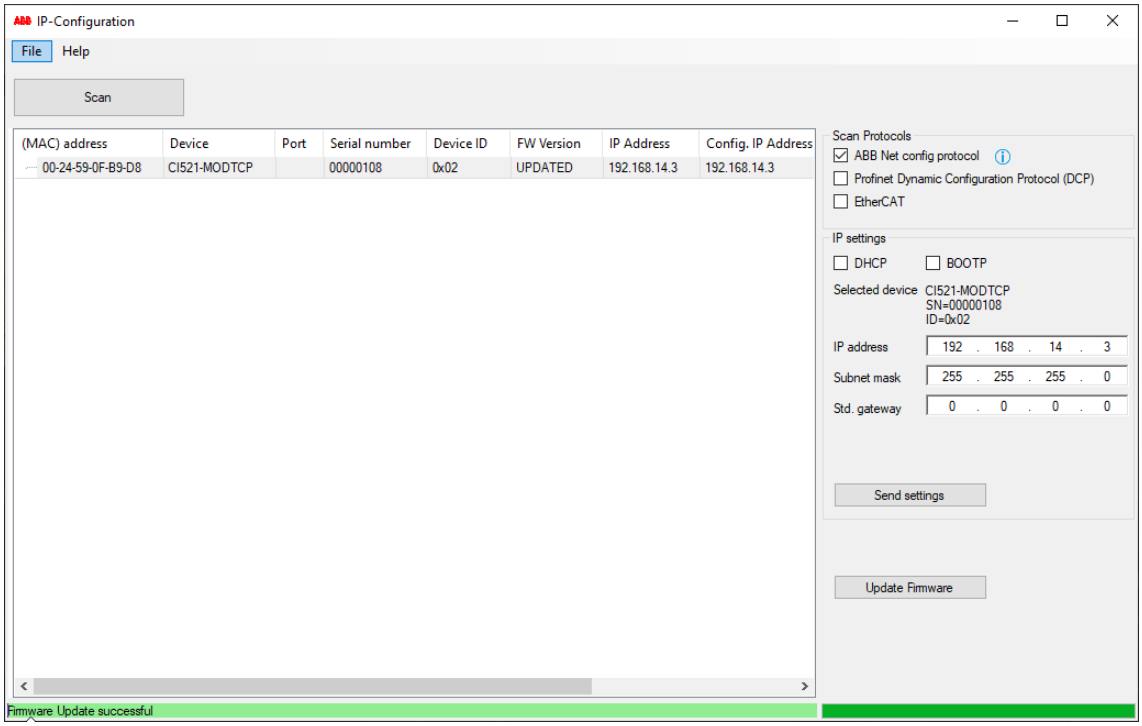
   > *After the firmware update all outputs of the updated devices are set to '0'.*

8. After a successful firmware update the update status or the new firmware version is displayed in the *"FW Version"* field.

   If this field is empty, there possibly is a connection error between the device and the executing PC.

   ↳ *"Error: Can't connect to device" on page 5876*

   Exception: For EtherCAT devices an empty *"FW Version"* field does not indicate a connection error.



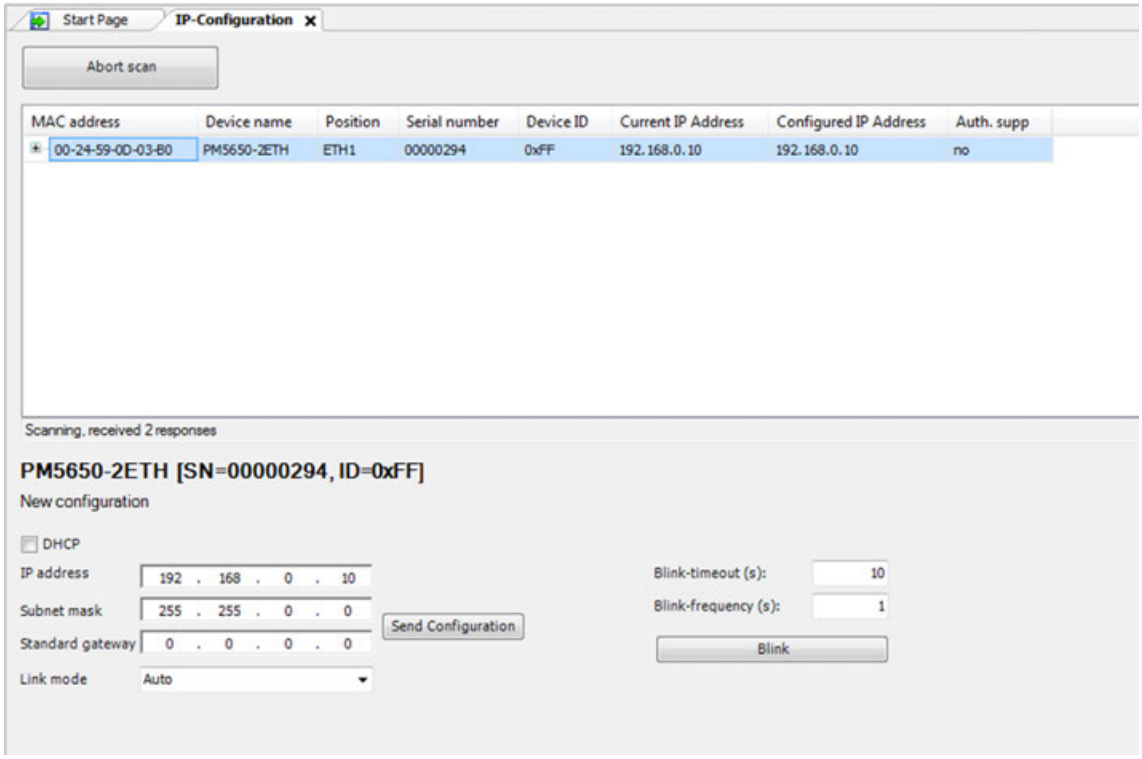
⇨ If the firmware update fails

- check the requirements for the update procedure.
  ↳ *"Requirements:" on page 5870*
- check the hints for trouble-shooting.
  ↳ *Chapter 1.6.5.2.4.1.3.3.1 "Trouble-shooting for firmware update" on page 5874*
- perform a network scan and repeat the update. If the error still persists power cycle the device and try the update again.

Blink functionality

This function activates flashing of the backlight of an AC500 LED display.

1. From the menu, select *"Tools ➔ IP-Configuration"*.
2. Click *[Scan]* to trigger the scan process for devices in the network.
   ⇨ A progress bar shows the progress. The IP settings of a selected device is displayed below the list and can be edited.
3. Adjust your desired time and click *[Blink]* to activate flashing.



Trouble-shooting for IP configuration tool

Firewall exceptions:

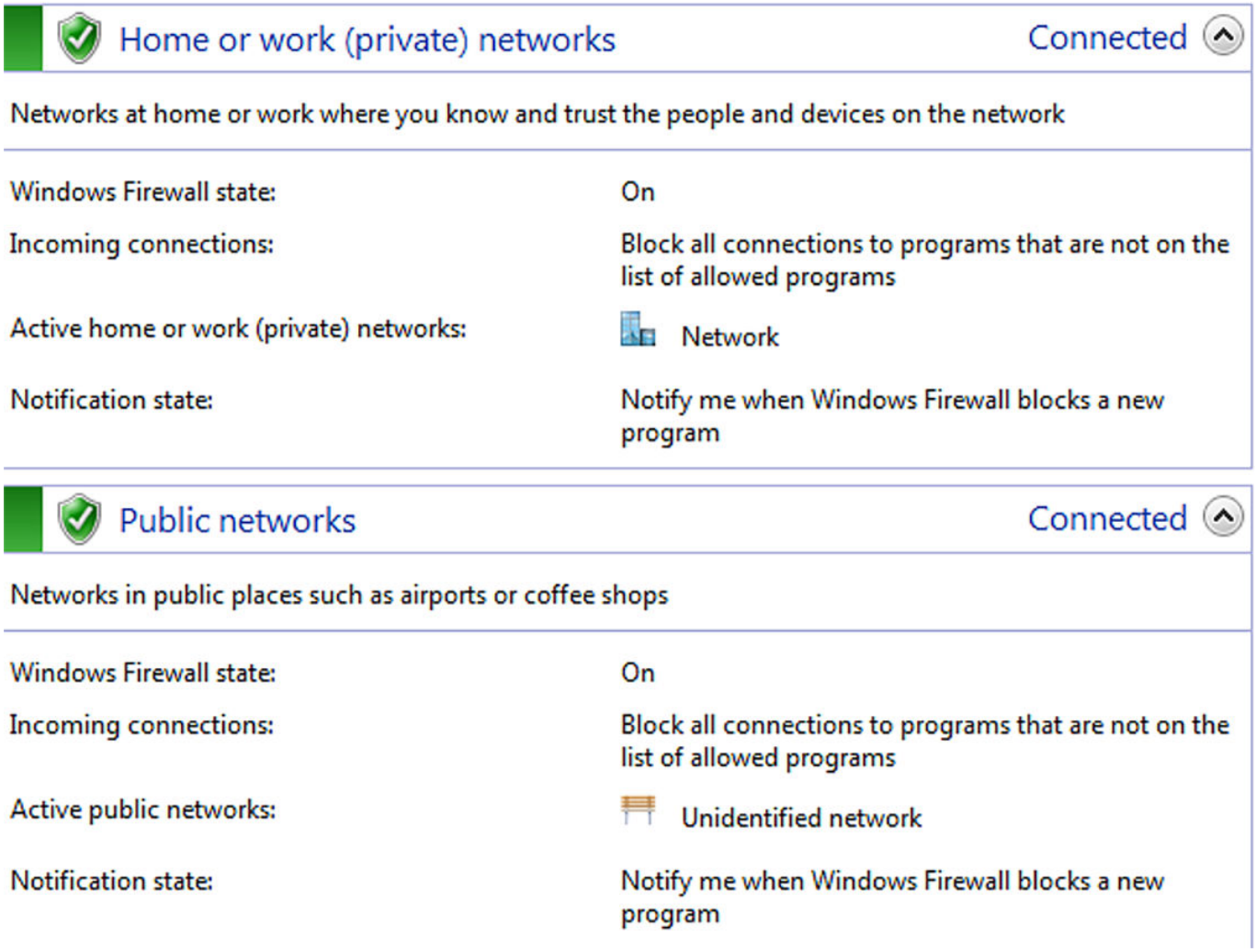
On a standard Windows 7 installation without third party firewall or security tools installed the IP configuration tool should work properly.

The Automation Builder setup installs rules or exceptions for the built-in Windows firewall to allow IPConfig to receive the responses for the IPConfig scan.

To check the Windows firewall is set correctly check the firewall settings.

Windows 7/ Windows 10:

On the network that is used for communication with the PLC, set *"Incoming connections"* to "Block all connections to programs that are not on the list of allowed programs".



If a third party firewall is used these exceptions must be configured manually.

> *Either exceptions for applications can be entered: Automation Builder and IP configuration tool must be added as application.*
>
> *Or the protocol and the port number must be given (for IPConfig: UDP protocol and port number 24576).*

Trouble-shooting for firmware update

Check the requirements

Ensure that all requirements have been considered before and during the update procedure.
“Requirements:” on page 5870

Check subnet configuration

This hint is only valid for Modbus devices and PROFINET devices.

If the *“FW Version”* field is empty after the network scan or if the firmware version has not been updated after the update procedure, there possibly is a connection error between the device and the executing PC.

Ping the device from the executing PC. If no connection can be established, check whether the device and the PC are in the same subnet.

Example

PC	Device	Result
192.168.14.71 / 255.255.255.0	192.168.14.10 / 255.255.255.0	OK
192.168.**10**.71 / 255.255.**255**.0	192.168.**14**.10 / 255.255.**255**.0	ERROR
192.168.10.71 / 255.255.0.0	192.168.14.10 / 255.255.0.0	OK

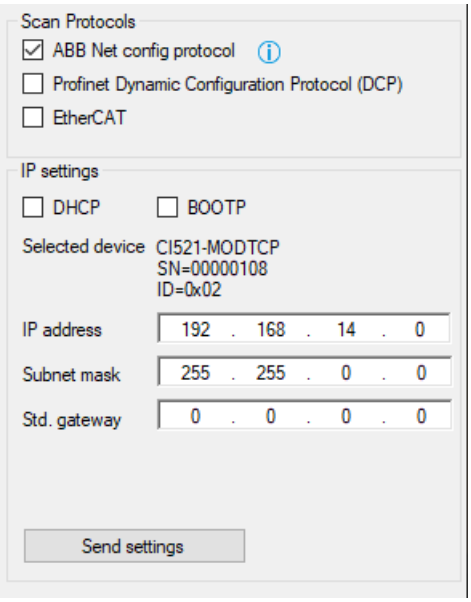
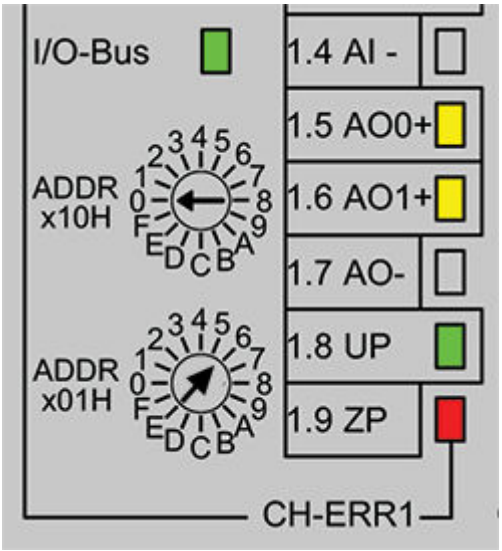
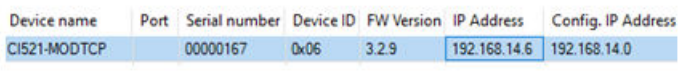
Click *[Scan]* again to restart the network scan. If the connection is successful a newer firmware version is displayed in the *"FW Version"* column.

Check last number of IP address

This hint is only valid for CI52x-Modbus devices.

Check the last number of the IP address. If it is set to "0", the IP address setting for this last number will be used from the rotary switches on the hardware device.

Example:

Automation Builder	AC500 communication interface module (rotary switch)
IP address: 192.168.14.0	IP address: 6
	
As a result, in the field *"IP Address"* the last number is set to "6": 	

Signature check

During the firmware update of CI50x, CI51x and CI52x devices a signature check is started.

The update procedure expects a firmware update file (**.bin*) and a signature file (**.bin.sig*) in the same directory. Without a signature file the signature check will fail.

Example:

Firmware update file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin***

Signature file:
*C:\AC500\AC500_CI52x_Firmware_V3.2.8.**bin.sig***

Error: Package timeout

A timeout error may occure due to an instable network.

Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

Error: Unable to read device status

A read error may occure due to errors in the firmware update protocol.

After the firmware update the IP configuration tool reads out the status of the updated device in order to check if the update was successful.

Error: IP is not unique

If an IP address is obtained by more than one device an error occures. A firmware update is not possible.

Error: Error State

Internal device error during the firmware update.

Solution:

Step 1: Scan again and repeat the firmware update.

Step 2: If this does not work, power cycle the device, scan again and repeat the firmware update.

Error: Can’t connect to device

The TCP communication is not sufficient. Increase the connection quality.

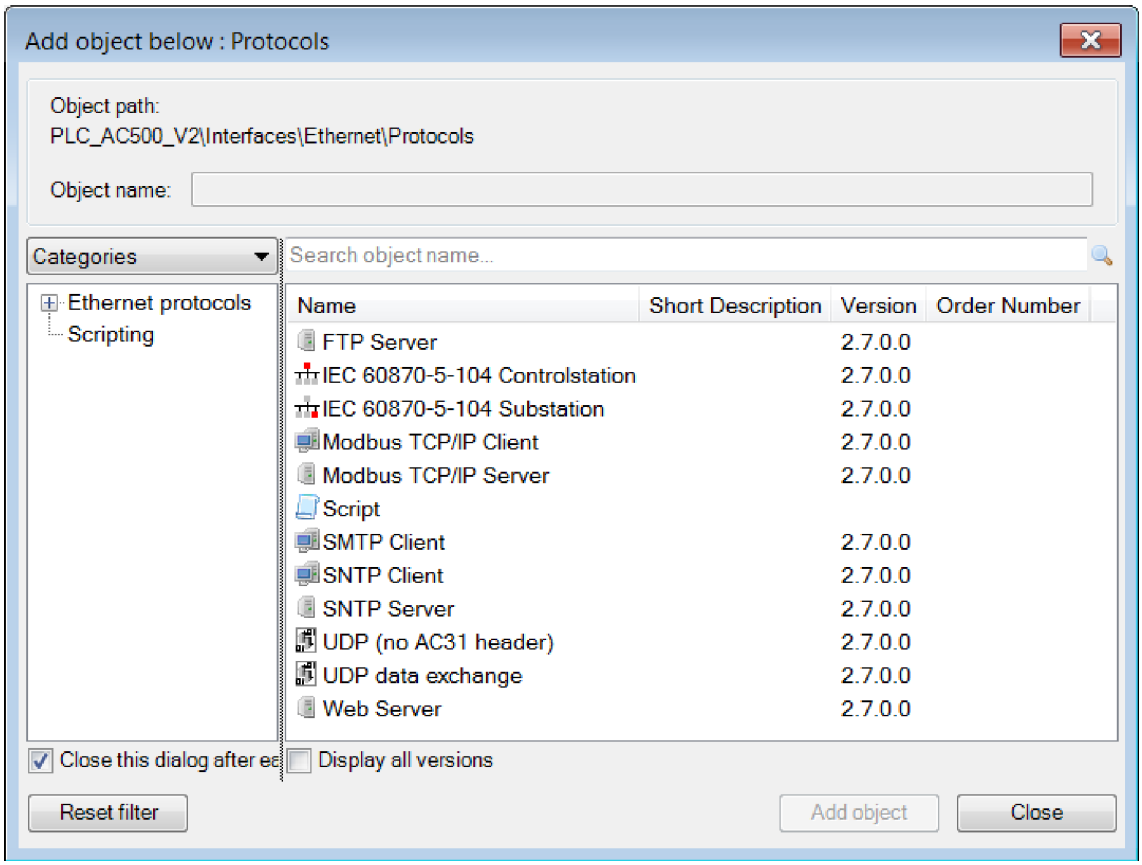
Solution: Keep the executing PC as near as possible to the devices that shall be updated. Avoid network switches.

CM5xy-ETH - External Ethernet communication module

See the description of the IP settings for Ethernet Communication Modules ↯ *Chapter 1.6.5.2.6.5.2 “Configuration of the external communication module CM597-ETH (IP data)” on page 5911*

Ethernet protocols

In the device tree, right-click the *“Protocols”* node under *“Ethernet”* and select *[Add object]*.



From the editor window select the desired protocol. Details on the available protocols are provided in the section *Chapter 1.6.4.1.6.1.1 "Ethernet protocols and ports for AC500 V2 products" on page 5442*.

1.6.5.2.5 Onboard ARCNET configuration

Parameters of PM5x1-ARCNET (onboard ARCNET)

Add a processor module with onboard ARCNET (e.g. PM590-ARC) to the device tree. Double-click the node *"PM5x1_ARC_Internal_ARCNET"* to open the parameter settings for onboard ARCNET. Check whether the default configuration is to be changed.

The following parameters are available:

Parameter	Default value	Value	Description
Run config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the internal ARCNET Communication Module.

Parameter	Default value	Value	Description
Address see remark 1 ↳ *Chapter 1.6.5.2.5.1.1 “Remark 1: Setting behavior of the ARCNET node ID” on page 5879*	0	0...255	Address (node ID) of the ARCNET Communication Module
Tranmission rate see remark 2 ↳ *Chapter 1.6.5.2.5.1.2 “Remark 2: Tranmission rate of the ARCNET communication module” on page 5879*	2.5 MB/s	2.5 MB/s	Transmission set for the ARCNET Communication Module
		1.25 MB/s	
		625 kB/s	
		312.5 kB/s	
Extended timeout ET1/ET2	Very small net	Very small net (=0)	ARCNET timeout setting. The following applies: Bit 0 configures ET2 of the Communication Module Bit 1 configures ET1 of the Communication Module Value ET1 ET2 Meaning ---------- 0 0 0 Max. network expansion 2 km 1 1 0 2 0 1 3 1 1 for large networks
		Small net (=1)	
		Big net (=2)	
		Very big net (=3)	
Long packets	Enable	Enable	Enable long data packets (512 bytes)
		Disable	Incoming long data packets are received and dismissed. The SEND block indicates an error in case of long data packets.
Evaluate DIN on receipt see remark 3 ↳ *Chapter 1.6.5.2.5.1.3 “Remark 3: Check of DIN identifier on receipt” on page 5879*	Enable	Enable	Enable check of DIN identifier on receipt
		Disable	Disable check of DIN identifier on receipt

Remark 1: Setting behavior of the ARCNET node ID

Configured ARCNET Node ID			
#	In Project	In Display	Effect
1	0 [default]	0 [default]	Configuration error: Class=3 Comp=9 Dev=10 Mod=3 Ch=0 Err=26
2	0 [default]	!=0	No error
3	!=0 && != display setting	!=0	Configuration error: Class=3 Comp=9 Dev=10 Mod=3 Ch=0 Err=26
4	!=0 && == display setting	!=0	No error
5	!=0	0	No error, but display shows "Addr 0"

Remark 2: Tranmission rate of the ARCNET communication module

If the transmission rate set for the ARCNET Communication Module differs from the default value (2.5 MB/s), programming via ARCNET using the SoHard-ARCNET PC boards is no longer possible. The same transmission rate has to be set for all subscribers of the ARCNET network. The ARCNET PC boards are firmly set to 2.5 MB/s.

Remark 3: Check of DIN identifier on receipt

If the parameter "Evaluate DIN on receipt" is enabled (default setting), the following DIN identifiers are reserved:

DIN identifier		**Protocol**
Hex	**Dec**	
4F	79	"Online access" - Programming/OPC with IEC 61131-3 programming / AC1131
5F	95	5F_ARCNET (Ethernet functions for ARCNET)

DIN identifier		Protocol
Hex	**Dec**	
6F	111	PC331 programming (not used for AC1131/IEC 61131-3 programming)
7F	127	Default DIN identifier for data exchange with function blocks: ARC_REC, ARC_SEND, ARC_STO, ARC_INFO All DIN identifiers except the reserved identifiers can be used for data exchange.

If the parameter "Evaluate DIN on receipt" is disabled, programming and/or OPC via ARCNET is not possible!

"ARCNET data exchange" and "5F_ARC"

The protocols "ARCNET data exchange" and "5F_ARC" can be appended.

Right-click *“PM5x1_ARC_Internal_ARCNET”* node and select *[Add object]*.

The following parameters can be set for the "ARCNET data exchange" protocol:

Parameter	Default value	Value	Description
Size of receive buffer	8192	512...65535	Receive buffer size in bytes. The minimum size is equal to the maximum size of an UDP telegram.
Size of transmit buffer high prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with high priority.
Size of transmit buffer low prio	4096	0...65535	Size of transmit buffer (in bytes) for telegrams with low priority.
Size of timeout buffer	2048	0...65535	Size of buffer (in bytes) for timeout data packets.
Number of header data	10	0...1464	Number of header data to be copied to the timeout buffer for timeout packages (in bytes).
Receive broadcast	Disable	Disable	Reception of broadcast telegrams disabled (data packets to all stations).

Parameter	Default value	Value	Description
		Enable	Reception of broadcast telegrams enabled (data packets to all stations).
Behavior on receive buffer overflow	Overwrite	Overwrite	Behavior on overflow of the receive buffer. The oldest data packets stored in the receive buffer are overwritten with the new incoming data packets.
		Reject	Behavior on overflow of the receive buffer. New incoming data are dismissed.

The following parameters can be set for the "5F_ARC" protocol:

Parameter	Default value	Value	Description
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

1.6.5.2.6 Communication modules

Communication modules (CM5xx)

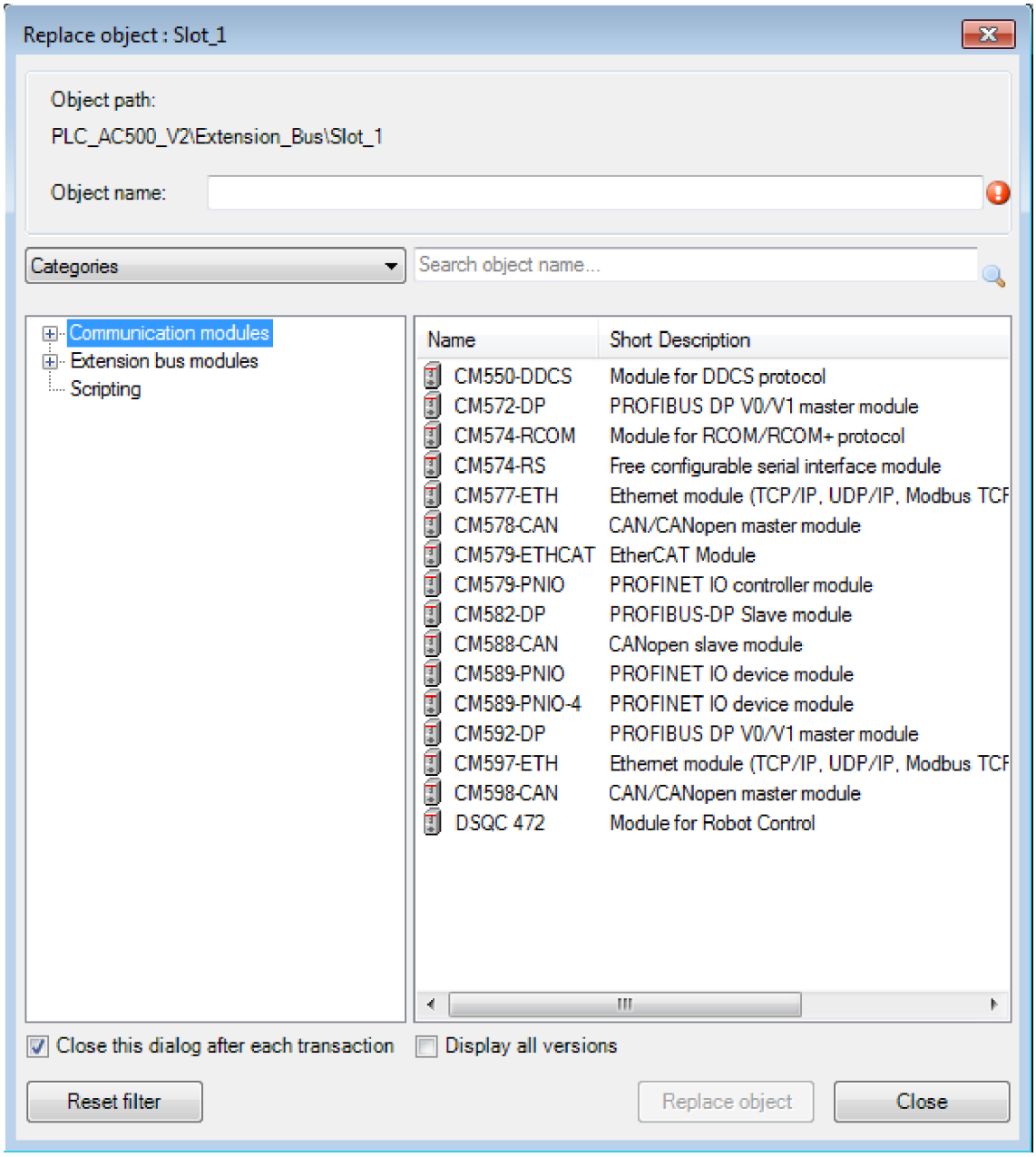
For a processor module with communication module devices, the communication modules must be specified in Automation Builder in the same order as they are installed in the AC500 PLC system:

- Slot 0 corresponds to the Internal communication module (installed in the processor module's housing)
- Slot 1 is the communication module installed in the first slot on the left of the processor module
- Slot 2 is the communication module installed in the second slot on the left of the processor module
- Slot 3 is the communication module installed in the third slot on the left of the processor module
- Slot 4 is the communication module installed in the fourth slot on the left of the processor module

The allocation of inputs/outputs for the communication modules is done slot-oriented and independent of the communication module type.

To append a communication module, add the communication module to the *"Extension_Bus"* node.

1. Right-click the desired *"Slot"* and select *"Add object"*.



2. Select the communication module from the list and click *[Replace object]*.

3. In the device tree, double-click the communication module node and configure the parameters according to the selected communication module.

PROFIBUS

The configuration of PROFIBUS Field Bus has to be done in the following steps:

- Parameterization of the Communication Modules (master and slave) ↳ *Chapter 1.6.5.2.6.2.1.1 “Parameterization of the CM582-DP/CM592-DP communication module” on page 5883*.
- Configuration of the PROFIBUS DP master ↳ *Chapter 1.6.5.2.6.2.1.2 “Configuration of a PROFIBUS DP master” on page 5883*.
- Configuration of the PROFIBUS DP slave ↳ *Chapter 1.6.5.2.6.2.2.1 “Configuration of PROFIBUS DP slave” on page 5894*
  - Configuration of I/O Data Objects ↳ *Chapter 1.6.5.2.6.2.2.1.1 “Configuration of I/O data objects” on page 5895*
  - Mapping of the PROFIBUS slave I/Os ↳ *Chapter 1.6.5.2.6.2.2.1.2 “Mapping of the I/Os” on page 5896*
- Optional:
  - Configuration of the PROFIBUS DP slaves (3rd party devices) ↳ *Chapter 1.6.5.2.6.2.1.3 “Configuration of 3rd party PROFIBUS DP slaves” on page 5887*.
  - Changing the target ↳ *Chapter 1.6.5.2.6.2.1.5 “Changing the target of a device” on page 5893*.

CM592-DP- PROFIBUS DP master communication module

Parameterization of the CM582-DP/CM592-DP communication module

To append a Communication Module, add the Communication Module to the *“Extension_Bus”* node.

- Right-click the desired slot and select *“Add object”*.
- Select the Communication Module from the list and click *[Replace object]*.

- Double-click the new node to open the CM582/CM592 - PROFIBUS DP (Interface) configuration in the editor window.

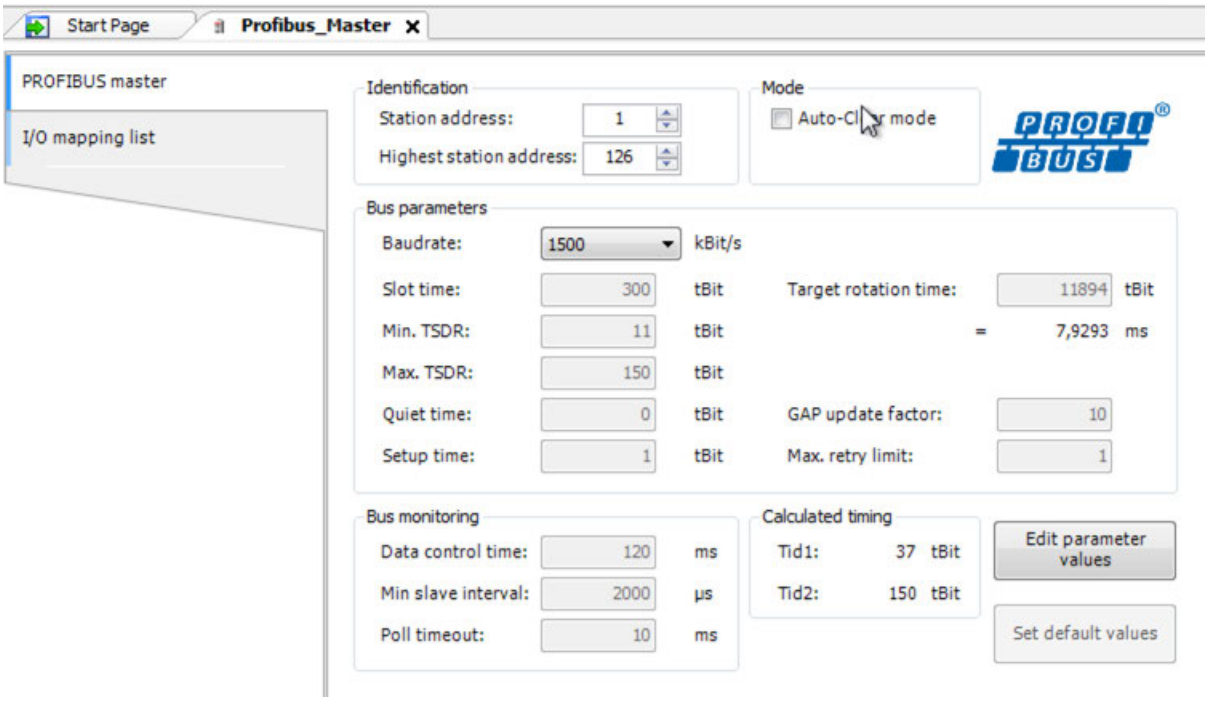
The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the PROFIBUS Communication Module.
Min update time	10	0...20000	Minimum update time of inputs and outputs in [ms].

Configuration of a PROFIBUS DP master

Double-click on *“Profibus_Master_x (Profibus_Master)”* to open the *“Profibus_Master”* configuration in the editor window:

Click on tab *"General"* if not already opend.



Most of the parameters are calculated automatically. Click *[Edit parameter values]* to change the parameters manually. Once edited, click *[Set optimal values]* to calculate the optimal values. *[Set default values]* can be used to revert all parameters to the default values.

> *All times for the PROFIBUS parameters are given in bit time [tBit]. The bit time is the result of the reciprocal of the transmission rate:*
>
> *tBit = 1 / transmission rate in [bit/s]*
>
> *The conversion from milliseconds into a bit time is shown in following formula:*
>
> *tBit = Time in [ms] * transmission rate in [bit/s]*

The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1)
Identification				
Station address	1	0...125	The individual device address of the master device on the bus.	DpParameter -> Station address
Highest station address	126	0...126	The highest bus address up to which a master searches for another master at the bus in order to pass on the token. This station address must on no account be smaller than the master station address.	DpParameter -> Highest station address
Mode				

Parameter	Default	Value	Description	Parameter (Remark 1)
Auto-Clear mode	Enabled	Disabled	The master operation mode will stay in the mode 'Operate' and the communication to all available slaves is kept up.	AutoClear-Supported
		Enabled	The masters operation mode will change from 'Operate' to 'Clear' and it shuts down the communication to all assigned slaves, if at least 1 slave is not responding within the data control time.	
Bus parameters				
Transmission rate see *Chapter 1.6.2.4.8.2.2 "Connections" on page 4080*	1500	9.6 19.2 45.45 93.75 187.5 500 1500 3000 6000 12000	Data transfer speed in [kBits/s]. The baud rate must be set to the same value for all devices on the bus. The result of changing the baud rate is that all other parameters must be recalculated.	DpParameter -> Transmission rate
Slot time	300	37.. 65535	Monitoring time of the sender (requester) of a telegram for the acknowledgement of the recipient (responder). After expiration, a retry occurs in accordance with the value of maximum telegram retries.	DpParameter -> TSL
Min. TSDR	11	1...65535	Shortest time period that must elapsed before a remote recipient (responder) may send an acknowledgement of a received query telegram. The shortest time period between reception of the last bit of a telegram to the sending of the first bit of a following telegram.	DpParameter -> Min. TSDR
Max. TSDR	150	1...65535	Longest time period that must elapse before a sender (requestor) may send a further query telegram. Greatest time period between reception of the last bit of a telegram to the sending of the first bit of a following telegram. The sender (requestor, master) must wait at least for this time period after the sending of an unacknowledged telegram (e.g. broadcast only) before a new telegram is sent.	DpParameter -> Max. TSDR

Parameter	Default	Value	Description	Parameter (Remark 1)
Quiet time	0	0...127	Time delay that occurs for modulators (modulator-trip time) and repeaters (repeater-switch time) for the change over from sending to receiving.	DpParameter -> TQUI
Setup time	1	0...255	Minimum period reaction time between the receipt of an acknowledgement to the sending of a new query telegram (reaction) by the sender (requestor).	DpParameter -> TSET
Target rotation time	11894	1.. 2 -1 (=16777215)	Pre-set nominal token cycle time within the sender authorization (token). The available time for the master to send data telegrams to the slaves depends on the difference between the nominal and the actual token cycle time. The Target rotation time (TTR) is shown in Bit times [tBit] like the other bus parameters. Below the displayed bit time, the Target rotation time is also displayed in [ms]. The default value depends on the number of slaves attached to the master and their module configuration.	DpParameter -> TTR
GAP update factor	10	0...255	Factor for determining after how many token cycles an added participant is accepted into the token ring. After expiry of the time period G*TTR, the station searches to see whether a further participant wishes to be accepted into the logical ring.	DpParameter -> GAP update factor
Max. retry limit	1	1...15	Maximum number of repeats in order to reach a station.	DpParameter -> max. retry limit
Bus monitoring				
Data control time	120	1..2^{24}-1	Defines the time in [ms] within the Data_Transfer_List is updated at least once. After the expiration of this period, the master (class 1) reports its operating condition automatically via the Global_Control command. The default value depends on the transmission rate.	DpParameter -> Data control time

Parameter	Default	Value	Description	Parameter (Remark 1)
Min. slave interval	2000	1...65535	Defines the minimum time period between two slave list cycles in [µs]. The maximum value the active stations require is always given. The default value depends on the slave types.	DpParameter -> min. slave interval
Poll timeout	10		Sets the maximum period of time in [ms] during which the response has to be received.	DpParameter -> Poll timeout
Calculated timing				
Tid1 (read-only)	37	37	Tid1 starts after the initiator has received an acknowledgement, answer or a token telegram. Tid1 = max (T_{QUI} + 2 * T_{SET} + 2 + T_{SYN} , min T_{SDR}) T SYN : This is the minimum time that must be available to each device as a rest condition before it is allowed to accept the start of a query and it is determined at 33 bit times.	-
Tid2 (read-only)	150	150	Tid2 starts after the initiator has send a telegram which is not acknowledged.. Tid2 = max (T_{QUI} + 2 * T_{SET} + 2 + T_{SYN} , max T_{SDR}) T_{SYN}: This is the minimum time that must be available to each device as a rest condition before it is allowed to accept the start of a query and it is determined at 33 bit times.	-

Remark 1:

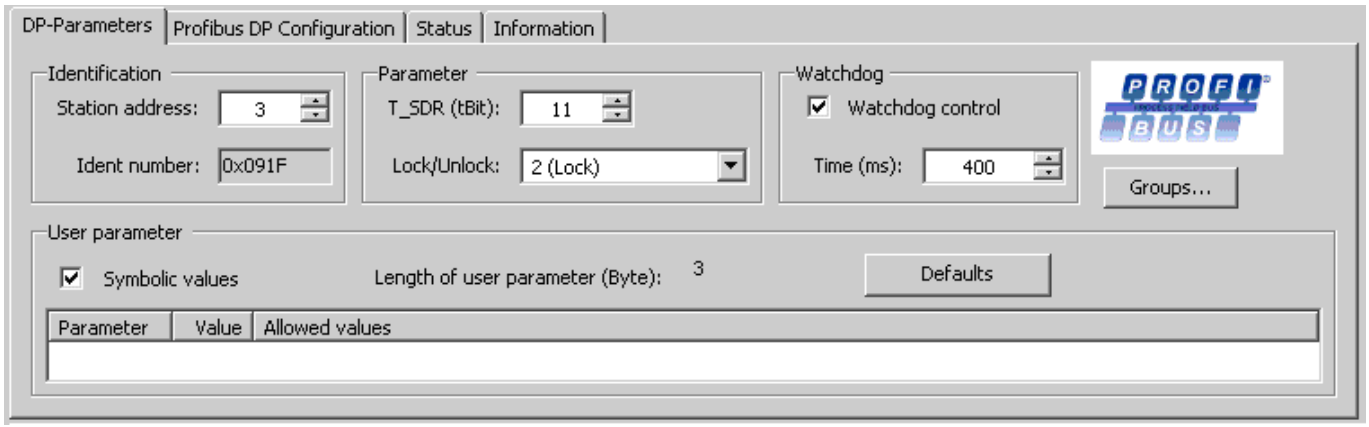
To display the parameters of this column, enable the option *“Show generic device configuration views”* under *“Tools ➔ Options ➔ Device editor”*.

Configuration of 3rd party PROFIBUS DP slaves

A PROFIBUS DP slave can be added by right-clicking on *“Profibus_Master_x (Profibus_Master)”* and selecting *“Add object”*.

If the desired device is not listed it can be installed via the *“Device Repository”* (menu item *“Tools” -> “Device Repository”*).

The slave configuration parameters can be edited in slave related editor window. To open this editor window, double-click the corresponding slave in the device tree.



All times for the PROFIBUS parameters are given in bit time [tBit]. The bit time is the result of the reciprocal of the transmission rate:

tBit = 1 / transmission rate in [bit/s]

The conversion from milliseconds into a bit time is shown in following formula:

*tBit = Time in [ms] * transmission rate in [bit/s]*

The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1)
Identification				
Station address	1	0...126	Station address of the PROFIBUS DP slave device.	StationAddress
Ident number	GSD file specific	FALSE	Station address of the PROFIBUS DP slave device.	SlavePrmData -> identNumber
Parameter				
T_SDR (tBit)	11	11...255	The parameter T_SDR (tBit) represents the minimum station delay of a responder (time a responder waits before generating the reply frame).	SlavePrmData -> minTsdr
Max. TSDR	2 (Lock)	0 (T_SDR unlock)	The TSDR and slave-specific parameter may be overwritten.	Bit 6 = 0 and bit 7 = 0 of bitmask SlavePrmData -> stationStatus
		1 (Will be unlocked)	The slave is released to other masters.	Bit 6 = 1 and bit 7 = 0 of bitmask SlavePrmData -> stationStatus

Parameter	Default	Value	Description	Parameter (Remark 1)
		2 (Lock)	The slave is locked to other masters, all parameters are accepted.	Bit 6 = 0 and bit 7 = 1 of bitmask SlavePrmData -> stationStatus
		3 (Unlock)	The slave is released to other masters.	Bit 6 = 1 and bit 7 = 1 of bitmask SlavePrmData -> stationStatus
Watchdog				
Watchdog control	Enabled	Disabled	The PROFIBUS slave does not utilize the Watchdog Control Time setting.	SlavePrmData -> wdFact1
		Enabled	The PROFIBUS slave utilizes the Watchdog Control Time setting in order to detect communication errors to the assigned master. When the slave finds an interruption of an already operational communication, defined by a Watchdog time, then the slave carries out an independent Reset and places the outputs into the secure condition.	
Time (ms)	400	0...2540	Watchdog time in [ms]. The default value depends on the number of slaves attached to the master and their configuration.	SlavePrmData -> wdFact2
User parameter				
Symbolic values	Enabled	Disabled	No symbolic names for the user parameters.	-
		Enabled	The values for the parameters are shown with symbolic names.	-
Length of user parameter (Byte)	3	Device-specific	The length of the user parameters in [bytes]. By default this value is 3 due to the existing reserved values.	-
Defaults	-	-	The button restores the default values of the user parameters.	-

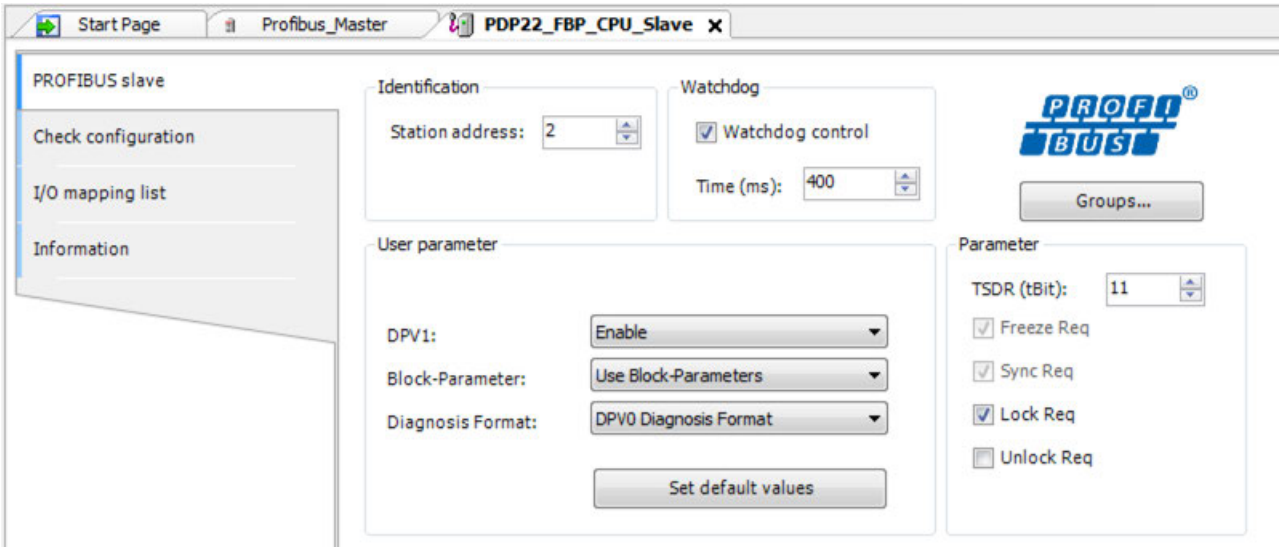
Remark 1:

To display the parameters of this column, enable the option *“Show generic device configuration views”* under *“Tools ➔ Options ➔ Device editor”*.

Configuration of the PROFIBUS DP slaves connected via FBP

A PROFIBUS DP slave can be added by right-clicking on Profibus_Master_x (Profibus_Master) and selecting *“Add object”*.

The slave configuration parameters can be edited in tab *“PROFIBUS slave”*:



All times for the PROFIBUS parameters are given in bit time [tBit]. The bit time is the result of the reciprocal of the transmission rate:

tBit = 1 / transmission rate in [bit/s]

The conversion from milliseconds into a bit time is shown in following formula:

*tBit = Time in [ms] * transmission rate in [bit/s]*

The following parameters are available:

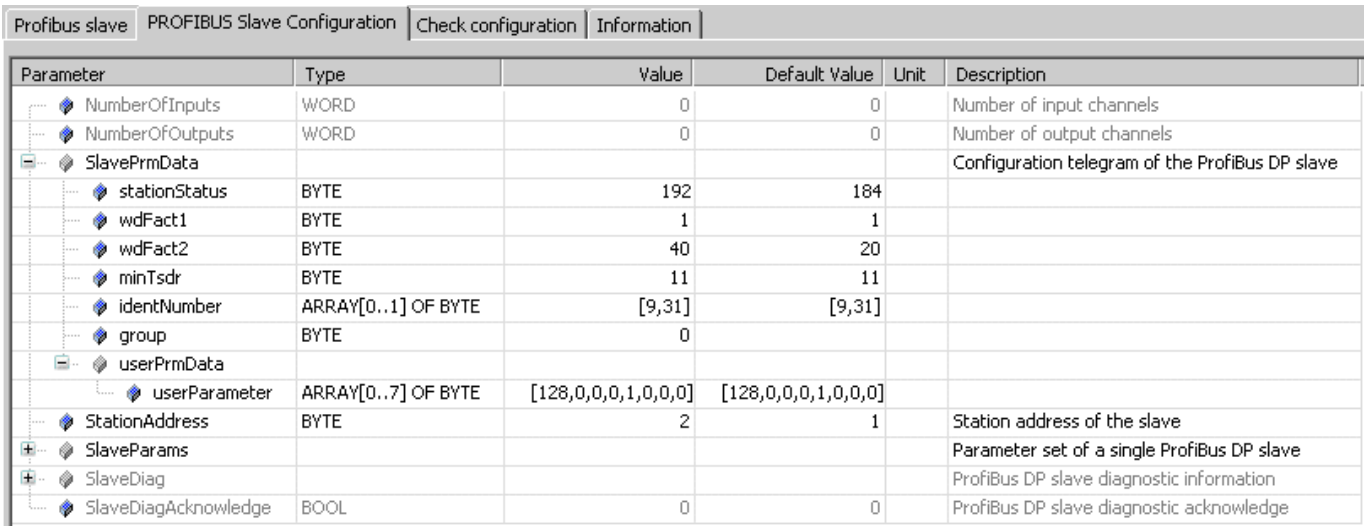
Parameter	Default	Value	Description	Parameter (Remark 1 *Further information on page 5889*)
Groups... Remark 2 *Further information on page 5889*	BUTTON	-	Opens the Group Properties dialog window	Bitmask Slave-PrmData -> group
Identification				
Station address	1	0...126	Station address of the PROFIBUS DP slave device.	StationAddress
Watchdog				
Watchdog control	Enabled	Disabled	The PROFIBUS slave does not utilize the Watchdog Control Time setting.	SlavePrmData -> wdFact1

Parameter	Default	Value	Description	Parameter (Remark 1 ↳ *Further information on page 5889*)
		Enabled	The PROFIBUS slave utilizes the Watchdog Control Time setting in order to detect communication errors to the assigned master. When the slave finds an interruption of an already operational communication, defined by a Watchdog time, then the slave carries out an independent Reset and places the outputs into the secure condition.	
Time (ms)	400	0...2540	Watchdog time in [ms]. The default value depends on the number of slaves attached to the master and their configuration.	SlavePrmData -> wdFact2
User parameter				
DPV1	Enabled	Disabled	DPV1 is disabled.	Item 7 of array SlavePrmData -> userPrmData -> userParameter (enabled = 128, disabled = 0)
		Enabled	DPV1 is enabled.	
Block Parameter	Use Block-Parameters	Use Block-Parameters	Use block parameters.	Item 4 of array SlavePrmData -> userPrmData -> userParameter (enabled = 1, disabled = 0)
		Ignore Block-Parameters	Ignore configured block parameters.	
Diagnosis format	DPV1 Diagnosis Format	DPV0 Diagnosis Format	Use DPV0 diagnosis format.	Item 3 of array SlavePrmData -> userPrmData -> userParameter (enabled = 1, disabled = 0)
		DPV1 Diagnosis Format	Use DPV1 diagnosis format.	
Set default values ...	-	-	The button restores the default values of the user parameters.	-
Parameter				
TSDR (TBit)	11		The TDSR in [tBit] represents the station delay of a responder (time a responder waits before generating the reply frame).	minTSDR

Parameter	Default	Value	Description	Parameter (Remark 1 ↳ *Further information on page 5889*)
Freeze Req	Enabled	Disabled	Freeze request is disabled.	Bit 4 of Slave-PrmData -> stationStatus
		Enabled	Freeze request is enabled.	
Sync Req	Enabled	Disabled	Sync request is disabled.	Bit 5 of Slave-PrmData -> stationStatus
		Enabled	Sync request is enabled.	
Lock Req	Enabled	Disabled	Lock request is disabled.	Bit 7 of Slave-PrmData -> stationStatus
		Enabled	Lock request is enabled.	
Unlock Req	Disabled	Disabled	Unlock request is disabled.	Bit 6 of Slave-PrmData -> stationStatus
		Enabled	Unlock request is enabled.	

Remark 1:

The parameters in this column are shown in tab PROFIBUS slave Configuration which is only visible if the parameter Show generic device configuration views is activated (open the Options dialog with *"Tools ➔ Options ➔ section Device editor"*):



The table shows all configuration parameters which are available for the device.

Remark 2:

Click *[Groups...]* to open the Group Properties dialog window. The relevant group(s) for the slave device can be selected by activating the corresponding item in the dialog window. The selection(s) can be confirmed with *[OK]*. To cancel the selection, use the cross symbol on the right of window title bar.

I/O configuration check

Tab *"Check configuration"* can be used to check the current I/O configuration. The device specific number of available I/Os can be compared with the actually configured I/Os to detect a faulty I/O configuration before downloading the program.

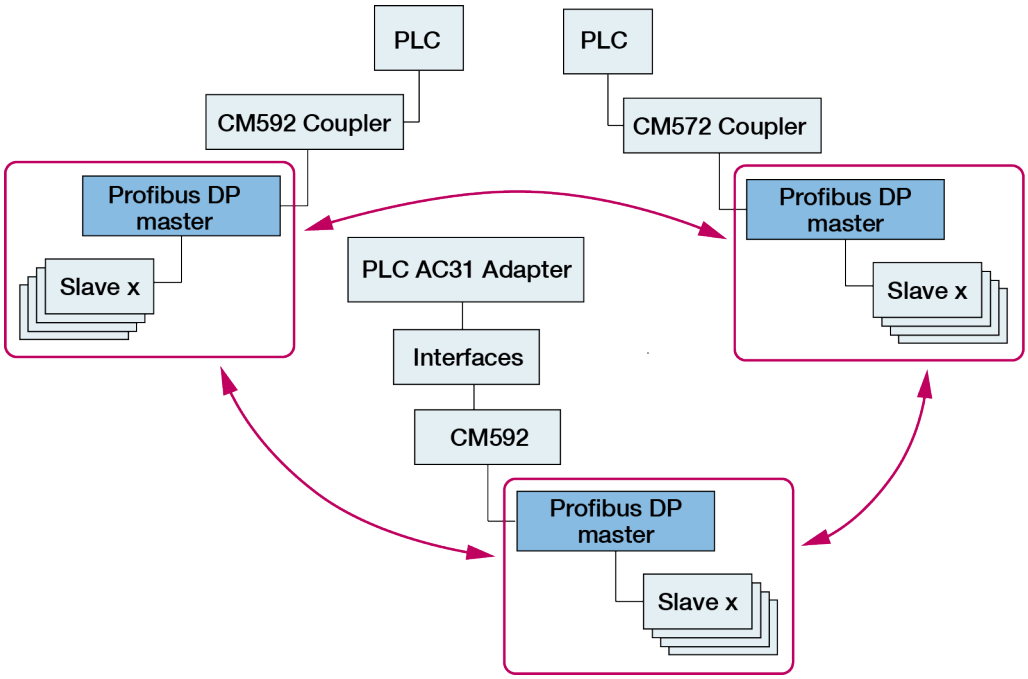
Parameter	Default value	Value	Description
I/O length			
	Device-specific	0...244	Length of the configured input data in [bytes].
	Device-specific	0...244	Length of the configured output data in [bytes].
	Device-specific	0...368	Length of the configured input/output data in [bytes].
Restriction on [device name]			
AI	Device-specific	0...128	Available analog inputs of the device in [words].
AO	Device-specific	0...128	Available analog outputs of the device in [words].
DI	Device-specific	0...32	Available digital inputs of the device in [bytes].
DO	Device-specific	0...32	Available digital outputs of the device [bytes].
Parameter length			
Parameter length	Device-specific	0...244	Parameter length in [bytes].

Changing the target of a device

As of Automation Builder 1.2, the support of Communication Module CM572-DP is discontinued by the manufactorer Hilscher.

In Automation Builder, you can update CM572-DP to CM592-DP. Automation Builder will preserve the protocol specific configuration after the device update.

Alternatively, you can copy the existing PROFIBUS protocol configuration to any new PROFIBUS device, e.g. from CM572-DP to CM592-DP or vice versa. Also the CM592-DP module of the AC31-Adapter is supported.



Update CM572-DP to CM592-DP

To update CM572-DP device to a CM592-DP, proceed as follows:

- Right-click *"CM572-DP"* node in the device tree and select *"Update objects"*.
- From the dialog select the CM592-DP device and select *"Update objects"*.
- Rename the node to CM592-DP.

Automation Builder configuration will be preserved after the device update.

Protocols in PROFIBUS devices

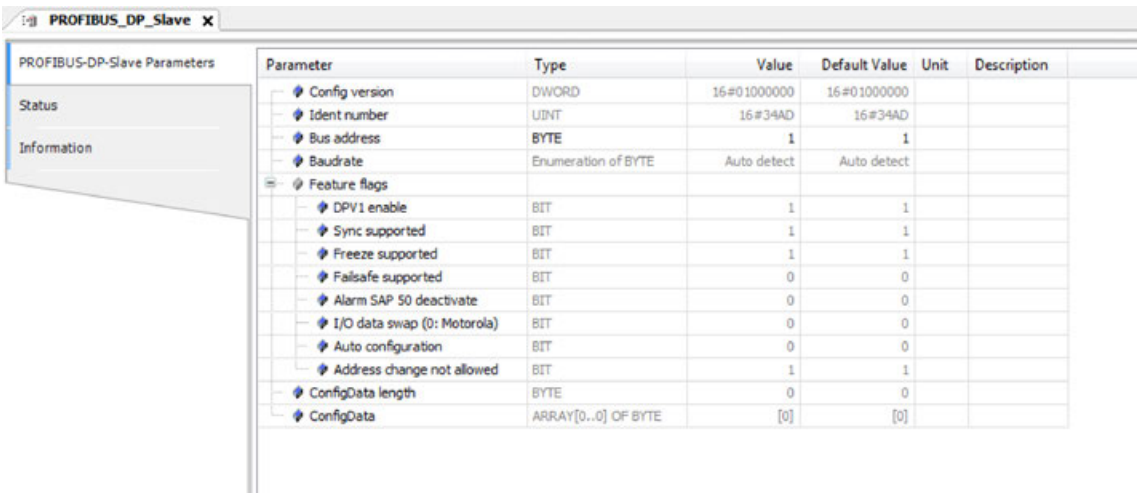
An existing PROFIBUS protocol configuration can be copied to any other PROFIBUS device, e.g. from a CM572-DP device to a CM592-DP device. For this, right-click the device node to be transferred and click copy. Right-click the device node to be changed and click paste.

Automation Builder configuration of CM592-DP is changed.

CM582-DP PROFIBUS DP slave communication module

Configuration of PROFIBUS DP slave

Double-click on *"PROFIBUS_DP_Slave"* to open the PROFIBUS slave configuration in the editor window:



The following parameters can be modified:

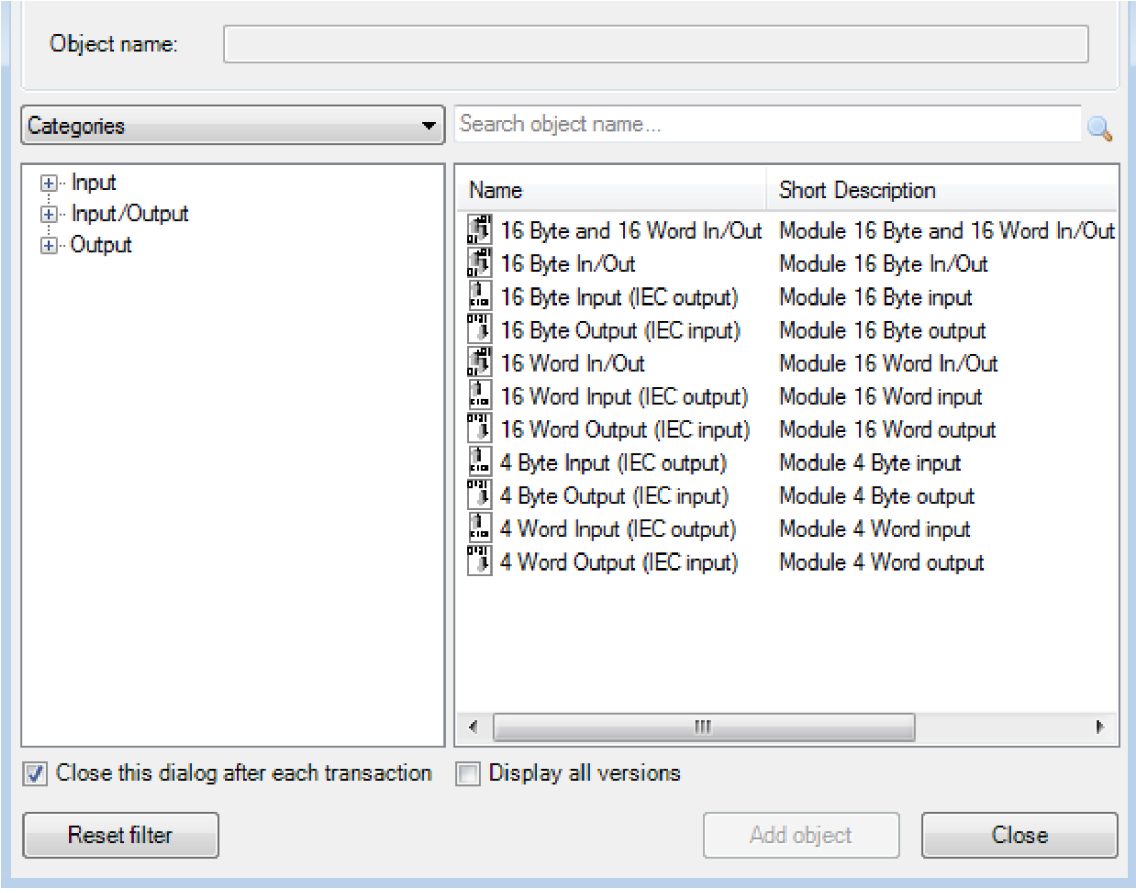
Parameter	Default value	Value	Description
Bus address	1	0...126	The bus address is the individual device address of the slave device on the bus.

Configuration of I/O data objects

To append I/O data, add the desired input / output objects to the Communication Module node.

Right-click the Communication Module node and select *"Add object"*.

Different types of data objects group I/O variables by size and direction. The I/O driver of the PLC firmware copies the amount of data bytes configured by these data objects cyclically. The time which is required for data copying is defined by the parameter *"Min. update time"* ↳ *Further information on page 5883*.



Select the desired I/O objects from the list and click *[Add object]*.

> *To keep basic load of PLC low, only configure as much I/O data objects as actually required. If further I/O variables need to be added later, additional data objects can be inserted.*

Technical details on the device such as the maximum amount of bytes used for I/O data is described in the device specification for ↳ *Chapter 1.6.2.4.8.1 "CM582-DP - PROFIBUS DP slave" on page 4075*.

Double-click an added I/O object node to open the preset configuration. As the I/O objects do not need user configuration all parameters in the *"Parameters"* tab are read-only.

Open the *"I/O Mapping"* tab to configure the mapping configuration for the I/O object.

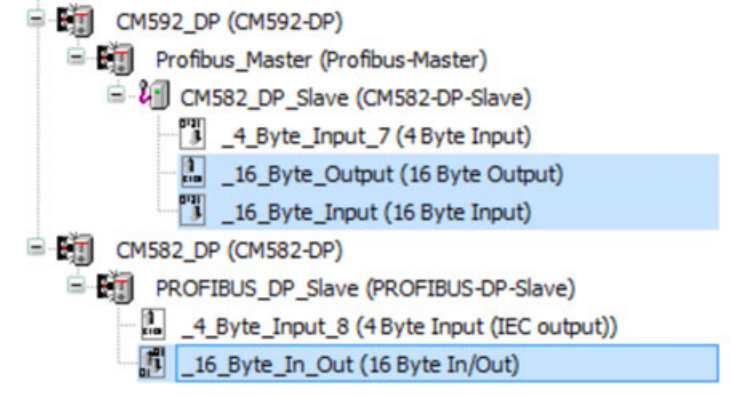
Possible inconsistency

On using CM582-DP slave device configured with modules types combining input and output data the following situation may happen:

Example

CM582-DP Communication Module configuration uses module type 16 Byte In/Out.

The device representation assigned to CM592-DP master uses module types 16 Byte Output and 16 Byte Input at the same place instead.



This mismatch will not be detected; neither by Automation Builder nor by PROFIBUS master and slave.

However, the communication will run stable and I/O data exchange is executed successfully.

Reason:

AC500 defines modules combining input and output directions to be split to two separated module configurations internally with output direction first.

Thus in AC500 the PROFIBUS configuration data for one module of type 16 Byte In/Out looks the same as for the combination of module types 16 Byte Output and 16 Byte Input.

Mapping of the I/Os

Double-click on the desired I/O data object in the device tree to show current I/O mapping connected to this data object.

See chapter Symbolic Names for Variables, Inputs and Outputs for further details on mapping inputs and outputs ↬ *Chapter 1.6.5.2.2.1.2 "Symbolic names for variables, inputs and outputs" on page 5815*.

CM574-RS - Programmable serial communication module

Configuration of the CM574-RS in the AC500 CPU project

To append a Communication Module, add the Communication Module to the *"Extension_Bus"* node.

- Right-click the desired slot and select *"Add object"*.
- Select the Communication Module from the list and click *[Replace object]*.

- Double-click on CM574_RS (CM574-RS) to open the CM574-RS configuration.

The following parameters are available:

Parameter	Default value	Value	Description
Run config fault	No	No	The user program is not started in case of a configuration error.
		Yes	The user program is started independent of a faulty configuration of the CM574.
Max. wait run	3000	0...120000	Max wait time for valid inputs.
Min. update time	10	10...2000	Minimum update time of inputs and outputs in [ms].
Enable debug	On	Off	The user program of the CPU only runs if the communication module is in RUN mode. The user program of the communication module only runs when the CPU is in RUN mode. Debug commands on the communication module are not permitted.
		On	The user program of the CPU runs independent of the communication module status, i.e., all debug commands are permitted on the communication module.
Watchdog	400	400..60000	Watchdog in [ms]

Basically 2 different modes can be configured at the sub modules of the CM574-RS communication module. Either cyclic data exchange (default setting) or remotely controlled COM port usage is available.

Using cyclic data exchange

The submodules Channel 1 and Channel 2 can be used to transfer I/O data cyclically between the processor module and the CM574-RS communication module. Input and output modules can be attached to these submodules. The procedure corresponds to the one of the *Chapter 1.6.4.1.1.3 “Processing of inputs and outputs in the multitasking system” on page 5399*.

Per channel, a maximum of 500 Bytes of I/O data can be exchanged.

A maximum of 32 sub modules can be attached. There is no automatic monitoring of the 500 bytes being exceeded. For this reason, it needs to be checked whether the last module exceeds the highest permissible input or output address. These addresses are:

Channel	Communication Module inputs in slot x			Communication Module outputs in slot x		
	%IBx.y	%IWx.y	%IDx.y	%QBx.y	%QWx.y	%QDx.y
Channel 1	%IBx.499	%IWx.249	%IDx.124	%QBx.499	%QWx.249	%QDx.124
Channel 2	%IBx.999	%IWx.499	%IDx.249	%QBx.999	%QWx.499	%QDx.249

In the project of the CM574-RS, the same I/O modules have to be configured for data exchange with the processor module.

Using COM protocols

In the CM574-RS project the corresponding COM port which should be remotely controlled by the AC500 CPU project must be set to COMx - Shared. At least an empty boot project with this setting must be deployed on the CM574-RS communication module.

If the cyclic data exchange is unused or not required for the project the corresponding channel can be switched to a COM protocol that is configured at a COM interface of the CM574-RS communication module and remotely controlled by the AC500 CPU. See Remote Control of COM ports of the CM571-RS communication module for further information ↳ *Chapter 1.6.5.2.6.3.3 "-Remote control of COM ports of the CM574-RS communication module" on page 5905*.

The protocol selection is done in the same way as it is known for the local COM interfaces ↳ *Chapter 1.6.5.2.11.1 "Setting up the protocol of a serial interface" on page 6098*.

Only the protocol CS31, Modbus and ASCII are allowed to be used remotely.

Shared Modbus Modus

Overview

CM547 as an additional CPU

Protocols of the serial interfaces of the CM574-RS

- Online Access
- CS31
- ASCII
- Modbus RTU
- Multi
- SysLibCom

View the different protocols here, ↳ *Chapter 1.6.4.2.6.2 "Protocols of the serial interfaces of the CM574-RS" on page 5594*

Communication Mode

The cyclic and acyclic data can be adjusted in the function blocks.

- Cyclic - ↳ *Chapter 1.6.4.2.6.2.6.2 "Cyclic data exchange CM574-RS/CS31 bus <-> AC500 CPU" on page 5597*
- Acyclic - ↳ *Chapter 1.6.4.2.6.3.1 "Function blocks for acyclic data exchange CM574-RS/ AC500 CPU" on page 5607*

Communication Settings

↳ *Chapter 1.6.3.6.4 “Connection and wiring” on page 5337*

Program in CODESYS 2.3

Download

CM574 in shared Mode

Configuration in CM574

↳ *Chapter 1.6.5.2.6.3.1 “Configuration of the CM574-RS in the AC500 CPU project” on page 5896*

• Shared Mode

Communication Settings

↳ *Chapter 1.6.3.6.4 “Connection and wiring” on page 5337*

Download in CODESYS 2.3

Program in Main PLC

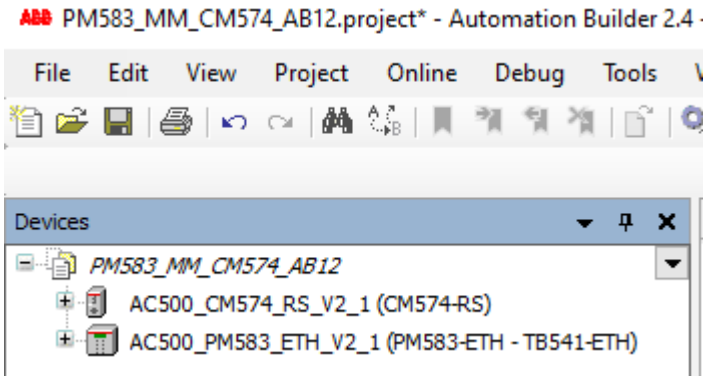
Download Project in Main PLC

Example project

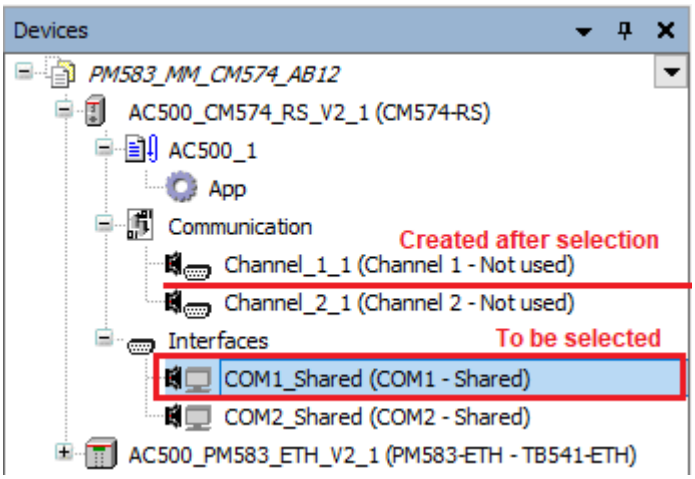
In the following, the configuration of the Modbus communication between a AC500 CPU and the CM574-RS via a common channel is described using an example project.

- Configuration protocols in CM574
  - CPU PM583
    CM574-RS mounted in the first slot
- Wiring
  - CM574-RS COM1 - CPU PM578-ETH COM1
- Functionality
  - CM574-RS is a Modbus Server. The COM1 of a CPU is a Modbus Client.
- Purpose
  - Direct access from the CPU program to the Modbus Clients wired to CM574-RS

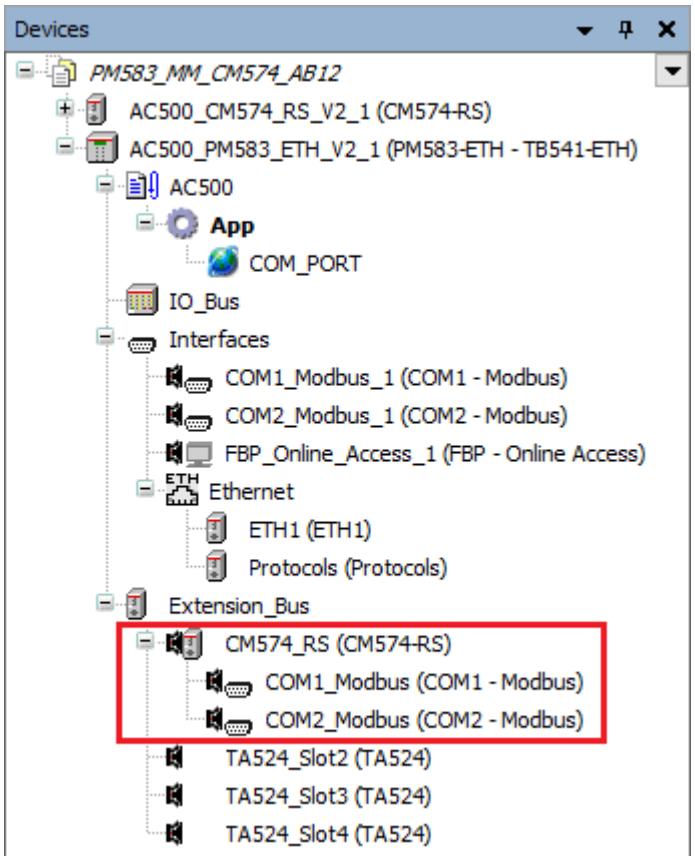
The project contains two sub-projects, one for the communication module CM574-RS and another one for the CPU.



1. Configure the settings in the CM574-RS project. See *Chapter 1.6.5.2.6.3.1 "Configuration of the CM574-RS in the AC500 CPU project" on page 5896*.



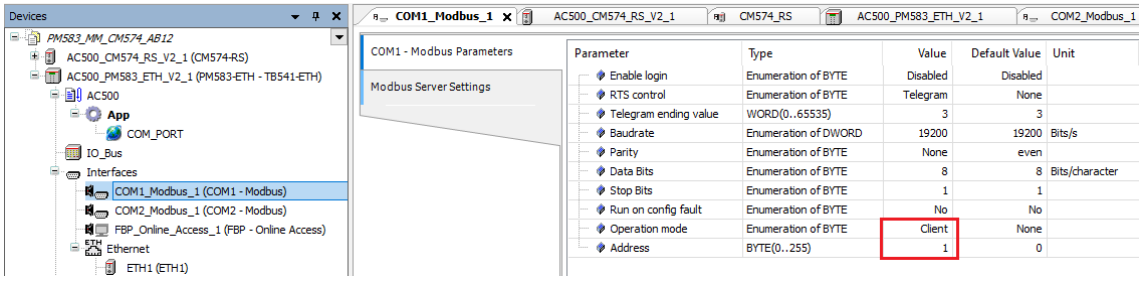
2. Configure the settings in the CPU project.



3. Configure the COM 1 port as Modbus Client. *Chapter 1.6.5.2.6.3.3 "-Remote control of COM ports of the CM574-RS communication module" on page 5905*
4. In the *"Device"* interface double-click on the *"COM1_Modbus_1 "*port.
   ⇨ The tab with the adjustable *"COM1 - Modbus Parameters"* opens on the right.

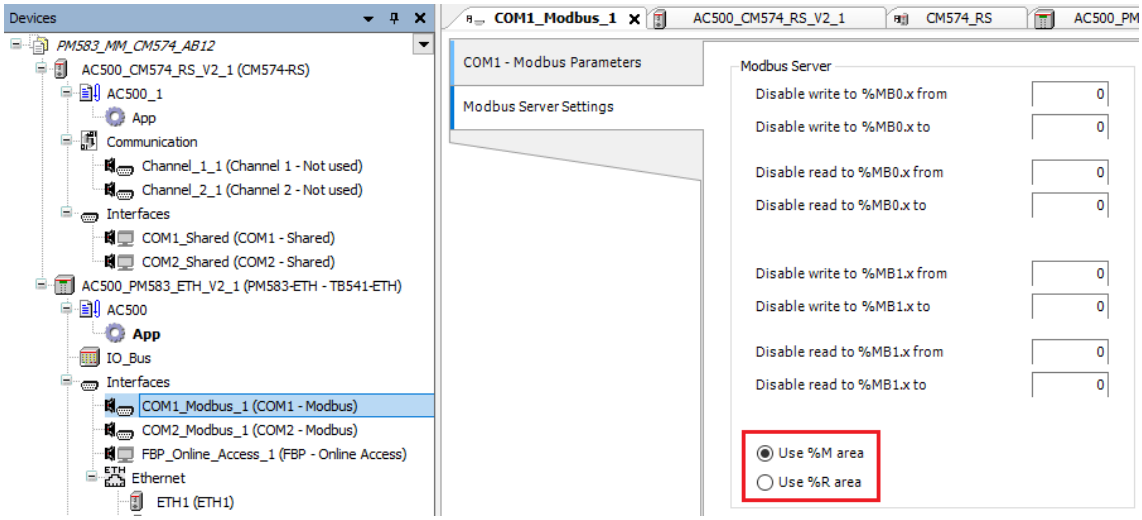
5. Double-click on the *"Value"* in the *"Operation mode"* to change the setting to *"Client"*.

   ⇨



6. Click to change to the *"Modbus Server Settings"*.

7. Compare the settings with the following figure, if necessary change the settings.



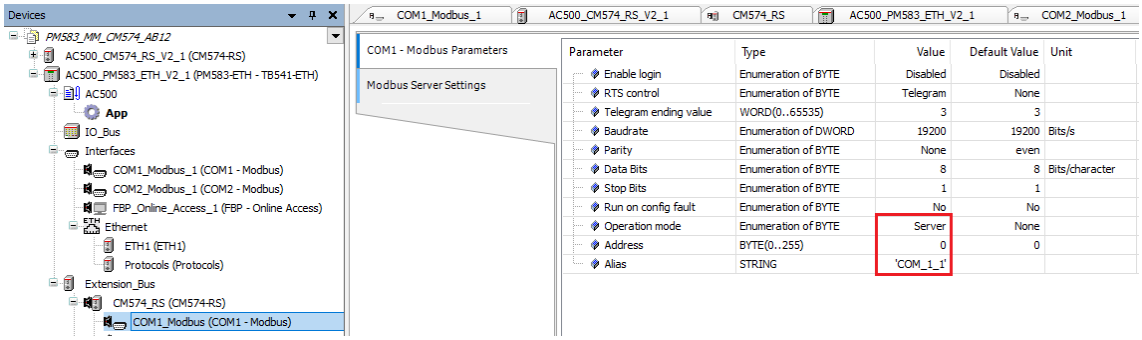
8. Set the CM574-RS as Modbus server in the CPU project.

9. In the *"Device"* interface double-click on the *"COM1_Modbus"* port.

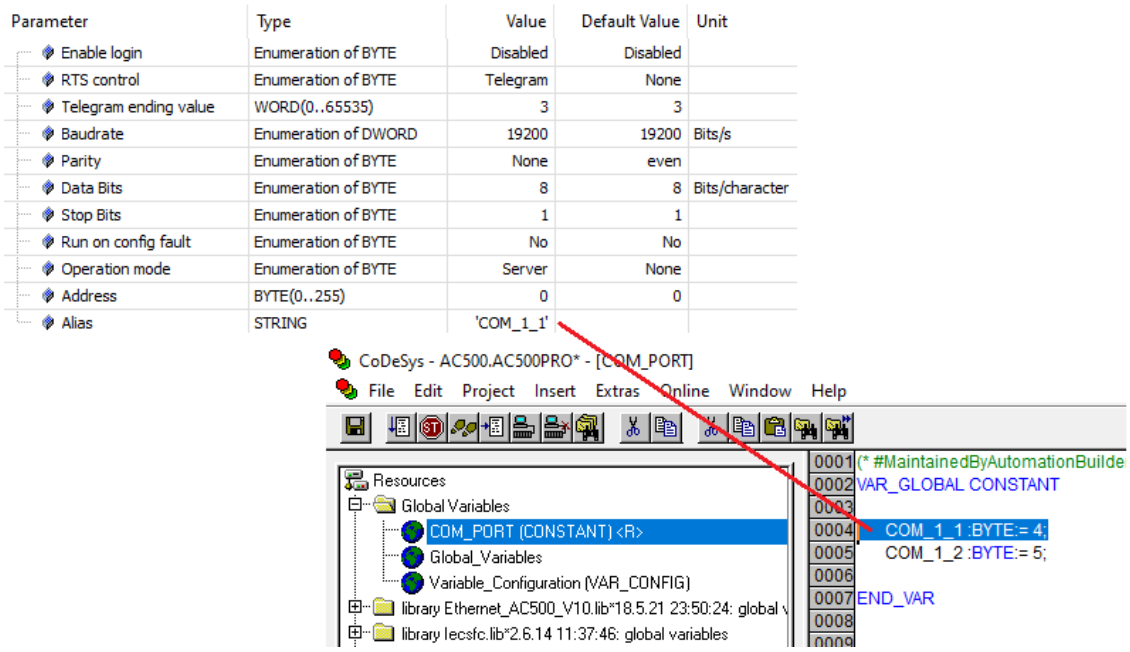
   ⇨ The tab with the adjustable *"COM1 - Modbus Parameters"* opens on the right.

10. Double-click on the *“Value”* in the *“Operation mode”* to change the setting to *“Server”*.

⇨



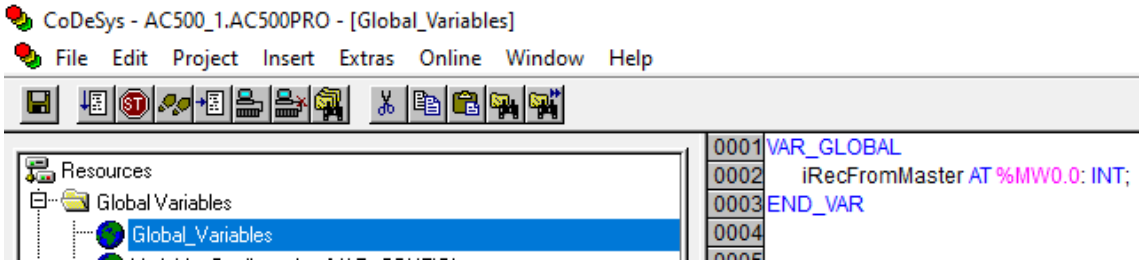
⇨ The *“COM_PORT”* for RCM574-RS COM 1 are now created.



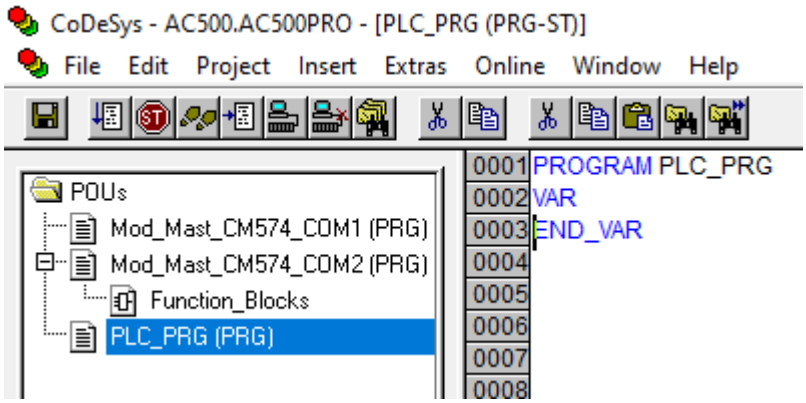
CPU project

- No program in the CM574-RS necessary
- The Server FB for the CM574-RS COM 1 is contained in the CPU program
- The COM number is given by the “alias” variable
- SLAVE=1
  communication with the client addressed with 1
- FCT=15
  master writes (sends) n words
- ADDR=0
  data from master will be stored in the slave from address %MW0.0
- NB=1 1
  word will be sent
- DATA=
  address of begin the data to be sent

Indication of the received data by client 1 in the CPU project.



In the CM574-RS project no programm is available.



Configuration of CM574-RS

A project for the serial communication module CM574-RS is created in the same way as an AC500 CPU project. See *Chapter 1.6.5.1.1.1 “Creating a new project ” on page 5758* for further details.

All interfaces of the Communication Module are pre-assigned with default values.

Parameterization

Double-click on *“CPU_parameters”* (CPU parameters) to open the CPU parameters configuration in the editor window.

Parameter	Default value	Value	Description
Error LED	On	On	The Error LED lights up in case of non-acknowledged errors in the diagnosis system.
		Off	The Error LED does not automatically light up in case of non-acknowledged error messages.
Flexible configuration	None	None	No flexible configuration is used.
		Flash	FlexConf.ini is loaded from Flash.
		User program	FlexConf.ini is loaded by user program.

Parameter	Default value	Value	Description
		FTP file	FlexConf.ini is loaded from FTP.
Flexible configuration timeout	1000	0...65535	FlexConf.ini timeout in seconds [s].
Free wheeling pause	10	0...255	Free wheeling pause.

If the Error LED should be activated by the user program, the parameter has to be set to "Off".

The library "SysIntExt_AC500_V10.lib" contains the function block LED_SET for controlling the LEDs of the CM574-RS. See description of the function block LED_SET of the Chapter 1.5.4.20 “Extended internal system library” on page 1626.

Configuration of the cyclic data exchange between CM574-RS and the processor module

Configuration of the cyclic data exchange of the CM574-RS with the AC500 CPU via inputs/ outputs is carried out in the entry CPU Communication.

In the project of the CM574-RS the same I/O channels have to be configured for data exchange with the AC500-CPU as they are configured in the AC500 CPU project.

The data exchange is cyclically but might be asynchronous. It cannot be ensured that all signals are present with chronological synchronism on the communication partners. This particularly applies in case of several tasks with different cycle times.

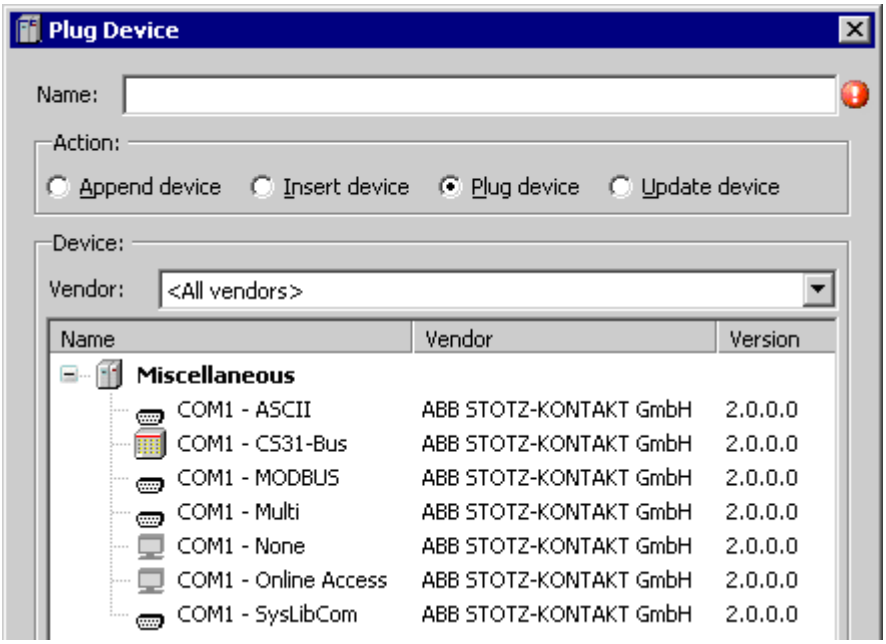
Configuration of the serial interfaces of the CM574-RS

If a COM port should be remotely controlled by the AC500 CPU project it must be set to COMx shared. At least an empty boot project with this setting must be deployed on the CM574-RS Communication Module.

In the standard configuration, the Serial Interfaces COM1 and COM2 of the CM574-RS are pre-assigned with the Online access protocol. Thus, online access from CODESYS can occur via one of the two interfaces with the serial driver Serial (RS-232) or ABB RS232 AC.

To set another protocol, proceed as follows:

Right-click the respective interface and select *“Plug Device”*.



Select the desired protocol and click *[OK]*.

The protocols have the same properties as for the serial interfaces of the AC500 CPUs PM57x, PM58x and PM59x.

The current interface settings can be displayed with the PLC browser command 'com settings'.

-Remote control of COM ports of the CM574-RS communication module

> *In order to use remote COM protocols for a CM574-RS serial port, the corresponding COM must be set to COMx- Shared within the CM574-RS CPU project.*

CM574-RS Communication Module's COM ports can be configured and used directly in the CPU project as if they were local COM interfaces. Each communication channel can be replaced by a COM port which is linked as follows:

- Channel 1 --> COM1 on CM574-RS Communication Module
- Channel 2 --> COM2 on CM574-RS Communication Module

The following protocols are available:

- ASCII
- Modbus RTU
- CS31 bus

The protocol selection as well as the configuration of the available parameters is done in the same way as it is known for a local COM interface. See Setting Up the Protocol of a Serial Interface *Chapter 1.6.5.2.11.1 “Setting up the protocol of a serial interface” on page 6098*.

Shared ports in the CM574-RS Project

In any case the COM port which should be remotely controlled must be "shared" by selecting the port to COMx-Shared within the CM574-RS Communication Module project. This option automatically sets the CPU Communication Module for the corresponding communication channel to Not Used. The ability to program the CM574-RS Communication Module is not affected.

CAUTION!
It is not possible to combine cyclic DPRAM-I/O data exchange with remote protocol functionality.

CS31 bus I/O configuration and access

The CS31 bus is configured as described for the local interfaces ↳ *Chapter 1.6.5.2.10.1 "Configuration of CS31 bus master" on page 6078*. The I/O mapping is done in exact the same way so that it is possible to access the CS31 bus I/Os locally in the project of the AC500 CPU.

When using the remotely controlled CS31 bus in combination with a CM574-RS Communication Module the I/O area size for each port is restricted to 500 bytes input and 500 bytes output data (for local interfaces 1000 bytes in each direction are available). Automation Builder automatically calculates the configured I/O sizes and warns as soon as the available size has been reached.

Alias parameter

The protocols ASCII ↳ *Chapter 1.6.5.2.11.3 "Setting COMx - ASCII" on page 6100* and Modbus ↳ *Chapter 1.6.5.2.11.4 "Setting COMx - Modbus" on page 6108* work with function blocks which require the target COM port number as an input. The COM port number of the remote controlled port is an internal identifier and might differ because it depends on the existing remote COM interfaces of the project. To address a remote COM interface the parameter Alias which represents the corresponding COM interface is used:

Parameter	Default	Value	Description
Alias	'COM_[x]_[y]'	STRING(80)	The alias of the COM port which represents the unique identifier of the COM port. By default the alias is generated by Automation Builder and consists of the slot number of the Communication Module (x) and the COM port's index at the Communication Module (y).

CAUTION!
It must be ensured that the alias string represents an IEC-conform variable name. Otherwise it will not be possible to compile the IEC project.

Automation Builder automatically adds a new section RemoteInterface to the Global Variables of CODESYS V2.3.9.x project after (re)creating the configuration data. This section contains the generated aliases as constant variables which can be used to address each remote COM interface at the existing function blocks of the protocols (COM_SEND, COM_REC, COM_MOD_MAST).

CM574-RCOM - RCOM/RCOM+ Communication module

Module parameters

The RCOM/RCOM+ Communication Module CM574-RCOM provides an implementation of the serial remote protocol RCOM/RCOM+ in master and slave mode.

1. In the device tree under *"Extension_Bus"*, right-click *"Slot_1"* or *"Slot_2"* node and select *[Add object]*. The slot corresponds to the slot of the Terminal Base.
2. Select *"CM574-RCOM"* from the list and click *[Replace object]*.
3. In the device tree, double-click the *"CM574-RCOM"* node.
   ⇨ The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of configuration fault the user program will not be launched.
		Yes	The user program will be also launched in case of configuration error on the I/O Bus.
Min. update time	10	10...20000	Minimum update time of the in- and outputs in [ms].
Watchdog	400	400...60000	Watchdog time [ms].

For each interface of the Communication Module, a module is added in the project structure.

- RCOM (RCOM/RCOM+): represents the RCOM protocol interface ↳ *Chapter 1.6.5.2.6.4.3 "Configuration of the RCOM protocol interface" on page 5908*.
- Console (Console): represents the debugging terminal interface (operator terminal) ↳ *Chapter 1.6.5.2.6.4.2 "Configuration of the operator terminal" on page 5907*.

Configuration of the operator terminal

▷ In the device tree, double-click the *"Console (Console)"* node.
   ⇨ The following parameters are available for debugging the terminal interface (operator terminal):

Parameter	Default value	Value	Description
RTS control	None	None	RTS control is not enabled.
Transmisson rate	19200	19200	The transmission rate of the Operator terminal is 19200 Baud/sec (read-only).
Parity	None	None	The operator console COM port does not use a parity check (read-only).
Data Bits	8	8	The number of data bits is 8 (read-only).
Stop Bits	1	1	One stop bit is used (read-only).
Debug level	No messages	No messages	Only error messages are printed on the terminal.
		Level 1	Brief messages of protocol events like data transmissions are printed.
		Level 2	Detailed information of protocol events is printed.

Configuration of the RCOM protocol interface

▷ In the device tree, double-click the *“RCOM (RCOM/RCOM+)”* node.

⇨ The following parameters are available:

Parameter	Default value	Value	Description
Operation mode	RCOM+ master	RCOM+ Master	The Communication Module works as a master device in an RCOM+ network.
		RCOM+ Slave	The Communication Module works as a slave device in an RCOM+ network.
		RCOM Master	The Communication Module works as a master device in an RCOM network.
		RCOM Slave	The Communication Module works as a slave device in an RCOM network.
Address (NODE)	0	0...254	Address (node number) of the Communication Module.
Transmission rate	9600 baud/s	300 600 1200 2400 4600 9600 19200	Transmission rate in Baud per second.
Parity	None	None	No parity check is used for telegram transmission.
		Odd	Odd parity check is used for telegram transmission.
		Even	Even parity check is used for telegram transmission.
CDLY	0	0	Value for carrier delay. The value will be converted internally into [ms] according to the specified transmission rate.
Character timeout	30	30	Value for character timeout. The value is internally converted into [ms] according to the specified transmission rate.
Turnaround time	4000	4000	Turnaround time to calculate timeout [ms].
Type of modem	Direct connection	Direct connection	Setting for direct connections.
		Fixed line modem	Setting for usage of fixed line modems.
		Installed RS-485 interface	Settings for usage of Hayes compatible dial-up modems. This requires the configuration of a telephone list as described below (module Telephone number).
		Installed RS-485 interface	Setting for installed RS-485 interface.

Parameter	Default value	Value	Description
		Modem 23WT90 half duplex	Settings for usage of a 23WT90 modem in half duplex operation mode.
Retransmissions	2	0...255	Number of telegram retransmissions after a reply was not received from the corresponding slave.
Max. number of polls	3	0...255	Number of event polls that are sent by the master during one event polling phase.
Number of preambles	1	1...255	Number of preamble bytes.
Number of postambles	1	1...255	Number of postamble bytes. This value must be equal to the number of preambles. If the values are different, the Communication Module automatically sets both parameters to the configured number of preambles.

The values of the parameters TLS (line stab time) and CDLY (carrier delay) are dependent on the length of the line and the type of modems used and might vary from system to system.

Parameter TLS always must be higher than parameter CDLY.

If a dial-up connection is used for the RCOM network, proceed with dial-up configuration. If no dial-up configuration is used, skip this section.

Dial-up configuration

In order to use dial-up connections, a modem module has to be appended to the RCOM/ RCOM+ Communication Module in the project structure.

ABB recommends HSM Eco modems for usage with CM574-RCOM Communication Modules.

1. Right-click the *"RCOM (RCOM/RCOM+)"* node and select *"Add object"*.
2. Select *"Dial-up modem"* and click *[Add object]*.
3. In the device tree, double-click the new *"Dial-up modem object"*. The module contains all commands required for controlling the modem. It is preconfigured for the HSM Eco modem:

Parameter	Default value	Value	Description
Modem init	ATZ^M~~ATI4^M~	String	Hayes compatible AT command to initialize connected modem.
Dial prefix 1...6	ATDT	String	Hayes compatible AT command to dial a number. 6 different dial prefixes can be configured.
Dial suffix	^M	String	Hayes compatible AT command string that is appended to the dial string.
Connect answer	CONNECT	String	Message of the connected modem, if connection is established.

Parameter	Default value	Value	Description
Modem ring	RING	String	Message of the connected modem, if incoming call is registered.
No carrier	NO CARRIER	String	String Message of the connected modem, if no carrier is detected.
Command mode	~~+++~~	String	Hayes compatible AT command to switch modem to command mode.
Modem answer	ATA^M	String	Hayes compatible AT command to answer incoming call.
Modem hangup	ATA^M~~~~~	String	Hayes compatible AT command to hang up (terminate) an existing connection.
Max ring time	70	0...255	Maximum time [s] until a call is cancelled.
Max number of calls	3	0...255	Maximum number of calls executed to establish a connection to a communication partner.
Call delay	2	0...255	Delay time [s] between two calls.
Hangup time	30	0...255	Time until a hang up process is considered as completed.

After configuring the modem, a list of telephone numbers (phone book) has to be created for the nodes of the RCOM network. This is done by adding new sub entries to the Dial-up modem entry.

1. In the device tree, right-click the *“Dial_up_modem (Dial-up modem)”* node and select *“Add object”*. Select *“Telephone number”* and click *[Add object]*.
2. In the device tree, double-click the *“Telephone_number”* node to open the telephone number parameters.

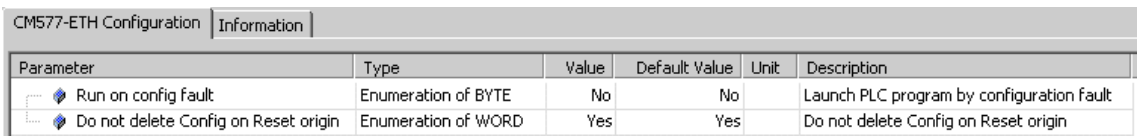
   ⇨ The following parameters are available:

Parameter	Default value	Value	Description
Dial prefix number	1	1...6	Identifier for the corresponding prefix. Prefixes can be configured under module "Dial-up modem" as described above.
Telephone number	-	Valid telephone number	Telephone number as a string.

CM597-ETH

Parameterization of the external communication module CM597-ETH

Double-click the *“CM597_ETH”* node:



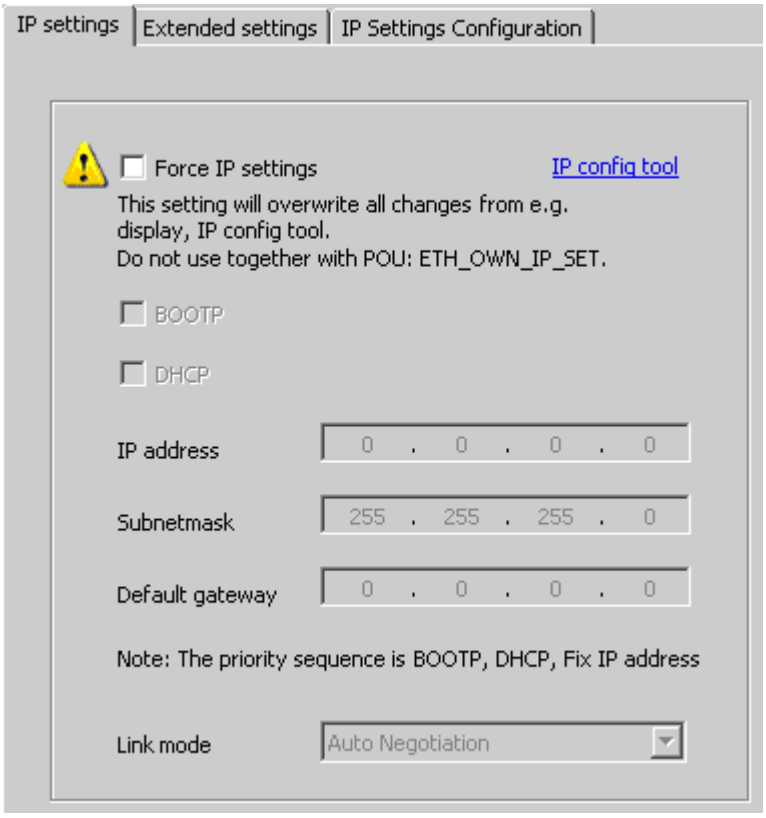
The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the Internal Ethernet Communication Module.
Do not delete Config on Reset origin	0	0	The Ethernet configuration is deleted after Reset origin.
		1	The Ethernet configuration (e.g. IP address) is still available after Reset origin.

Configuration of the external communication module CM597-ETH (IP data)

The IP data are used with highest priority. Existing settings done by the display or any other method will be overwritten.

Double-click the *"IP_Settings"* node to open the IP settings in the editor window:

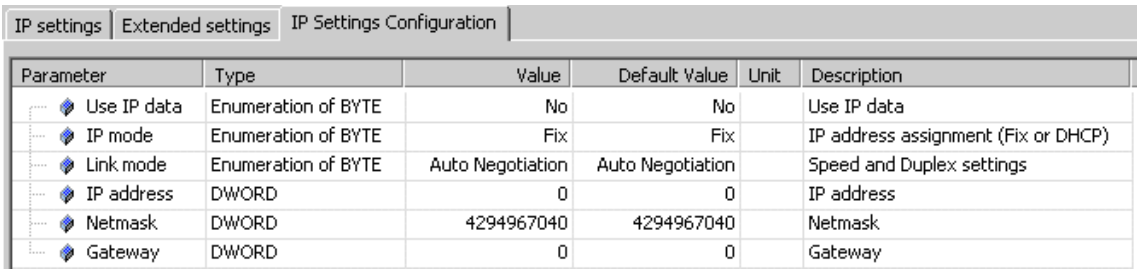


The following parameters are available:

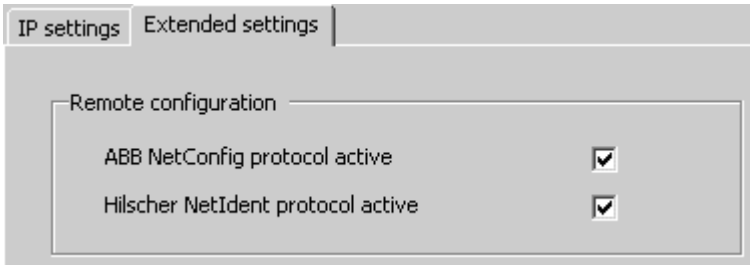
Parameter	Default	Value	Description	Parameter (Remark 1 ↳ *Further information on page 5911*)
Force IP settings	Enabled	Disabled	The IP data is not used. The device must be configured by another means of settings the IP data.	Use IP data
		Enabled	The IP data set in this editor is used for the device.	
DHCP	Disabled	Disabled	The DHCP client services are disabled. A static IP address must be used.	IP mode (+ DHCP)
		Enabled	The DHCP client services are enabled. The IP address of the device will be set by a DHCP server in the network.	
IP address	0.0.0.0	Valid IP address	IP address of the device.	IP address
Subnet-mask	255.255.255.0	Valid subnet mask	Subnet mask for the device.	Netmask
Default gateway	0.0.0.0	Valid gateway address	Default gateway address for the device.	Gateway
Link mode	Auto negotiation	Auto negotiation	The link mode will be detected automatically.	Link mode
		10 Mbit Half-Duplex	The link mode is statically set to 10 Mbit half-duplex.	
		10 Mbit Full-Duplex	The link mode is statically set to 10 Mbit full-duplex.	
		100 Mbit Half-Duplex	The link mode is statically set to 100 Mbit half-duplex.	
		100 Mbit Full-Duplex	The link mode is statically set to 100 Mbit full-duplex.	

Remark 1: Generic Device Configuration View Parameters

Tab IP Settings Configuration shows a list of all available parameters which is only visible if parameter 'Show generic device configuration views' is activated (in the menu click *“Tools ➔ Options ➔ section 'Device editor'”*).



Configuration of the external communication module CM597-ETH (Extended settings)



The following parameters are available:

Parameter	Default	Value	Description
Remote configuration			
ABB NetConfig protocol active	Enabled	Disabled	ABB NetConfig protocol is disabled. IP Config Tool is not supported in this case.
		Enabled	ABBNetConfig protocol is enabled. IP Config Tool can be used to configure the device.
Hilscher NetIdent protocol active	Enabled	Disabled	The Hilscher NetIdent protocol is disabled. Hilscher IP Config Tool is not supported in this case.
		Enabled	The Hilscher NetIdent protocol is enabled. Hilscher IP Config Tool can be used to configure the IP of the device temporarily.

ABB NetConfig protocol is more secure than Hilscher NetIdent protocol. It is recommended to disable the Hilscher NetIdent protocol if not needed.

CANopen

CM598-CN - CANopen master communication module

CM598-CN - CANopen manager communication module

The configuration of the CANopen field bus has to be done in the following steps:

- Parameterization of the AC500 Communication Module Bus Interface
- Parameterization of the CAN bus

Optionally the following steps are required to complete the configuration. These steps depend on the desired functionality:

- Configuration of the CAN2A
- Configuration of the CANopen Manager
- Configuration of the CANopen Slaves

Parameterization of CM598-CN

Presetting

– *Click menu “Tools ➔ Options ” and select “Device editor” in the Options window.*
– *Enable first checkbox “Show generic device configuration views” and click [OK].*

To append a Communication Module, add the Communication Module to the *“Extension_Bus”* node.

- Right-click the desired slot and select *“Add object”*.
- Select the Communication Module from the list and click *[Replace object]*.

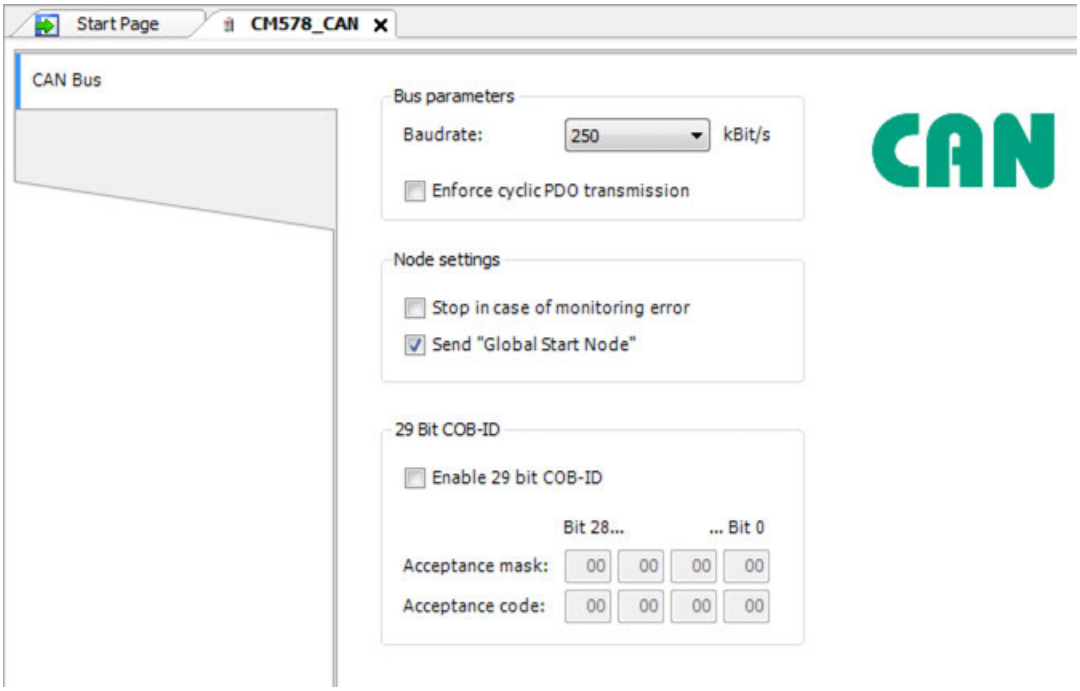
- Double-click on CM598_CAN to open the CM598-CAN configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the CANopen communication module.
Min update time	10	0...20000	Minimum update time of inputs and outputs in [ms].

Parameter	Default value	Value	Description
CANopen Sync mode	Sync Bus only	Sync Bus only	The CANopen Manager sends Sync messages (if enabled) to CAN bus only. The properties of the sync message are configured in the CANopen Manger parameters.
		Sync Bus and Task	CANopen Manager sends Sync messages to CAN bus and triggers IEC Task. The associated task has to be added manually in the task configuration. If there is any problem with CANopen Sync, check the following points in configuration: 1. CANopen Sync mode has to be set to Sync Bus and Task. 2. Sync Message has to be enabled. 3. A task of type "triggered by external event" has to be configured. The external event has to be "Ext_CouplerX_InputAny". 4. PDO transmission type has to be "cyclic synchronous".

The tab *CAN Bus* contains the basic CAN bus parameters:



Parameter	Default value	Value	Description
Bus Parameters			
Transmission rate	250 kBit/s	10 kBit/s 20 kBit/s 50 kBit/s 100 kBit/s 125 kBit/s 250 kBit/s 500 kBit/s 800 kBit/s 1000 kBit/s	Transmission speed in [kBit/s]
Node settings			
Stop in case of monitoring error	Disabled	Disabled	The manager does not stop if a monitoring error (Node Guard or Heartbeat Error) appears. This function defines the behavior of the manager if the communication is interrupted to at least one node. If this function is enabled, the manager will also stop the communication to all further nodes which were still responding and active. If this function is disabled, a lost communication to one node has no influence on the communication of the still present nodes. For all the error affected nodes the master remains in the state to try the reestablishment of the communication again.
		Enabled	
Send "Global Start Node"	Enabled	Disabled	No "Global Start Node" message is sent after configuring the nodes.
		Enabled	After the manager started all nodes configured individually first, it sends a "Global Start Node" afterwards, in order to synchronize all nodes again.
29 Bit COB ID			
Enable 29 bit COB-ID	Disabled	Disabled	The receive filter of CAN2.0 B protocol is disabled. The 29-bit COB-ID is disabled. An 11-bit COB-ID is used.
		Enabled	Enable this option in order to receive CAN2.0 B frames when using CAN 2.0 B protocol. The 29-bit COB-ID is enabled.

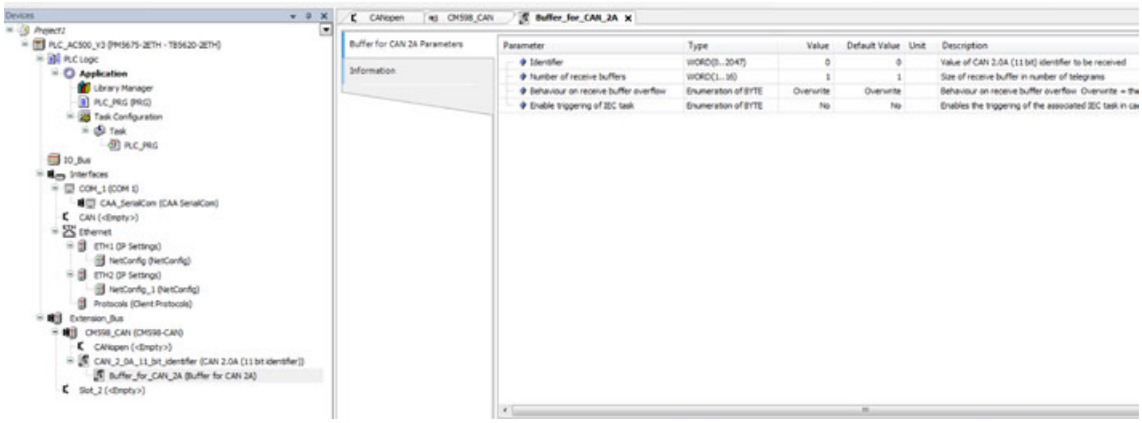
Parameter	Default value	Value	Description
Acceptance mask	0	29 bit	Here it is possible to define the bits, the filter uses. In other words: All not set bits will not be filtered out. The parameter Enable 29 bit COB-ID must be enabled.
Acceptance code	0	29 bit	Those are the bits set to filter the IDs. Those bits must have the value '1' in the acceptance code and the received COB ID to pass the filter. If a bit is not set in the acceptance mask, the filter will pass the message anyway. The parameter Enable 29 bit COB-ID must be enabled.

Configuration of the CAN protocols

The Communication Module CM598-CAN can be used to realize CAN bus based networks in combination with library CANopen_AC500_V11.lib.

1. Plug the communication module CM598-CAN to the desired slot of the communication module bus.
2. Right-click *"CM598_CAN (CM598-CAN)"* and select *"Add device"* in the context menu.
3. Select the desired *"CAN 2.0A"* protocol or *"CAN 2.0B"* protocol from the list.
4. The CAN data transmission requires a buffer for the incoming data.

   Right-click *"CAN_2_0A_11_bit_identifier_ (CAN 2.0A)"* or *"CAN_2_0B_29_bit_identifier_ (CAN 2.0B)"* and select *"Add object"*. Select the corresponding receive buffer for your selected CAN protocol from the list.



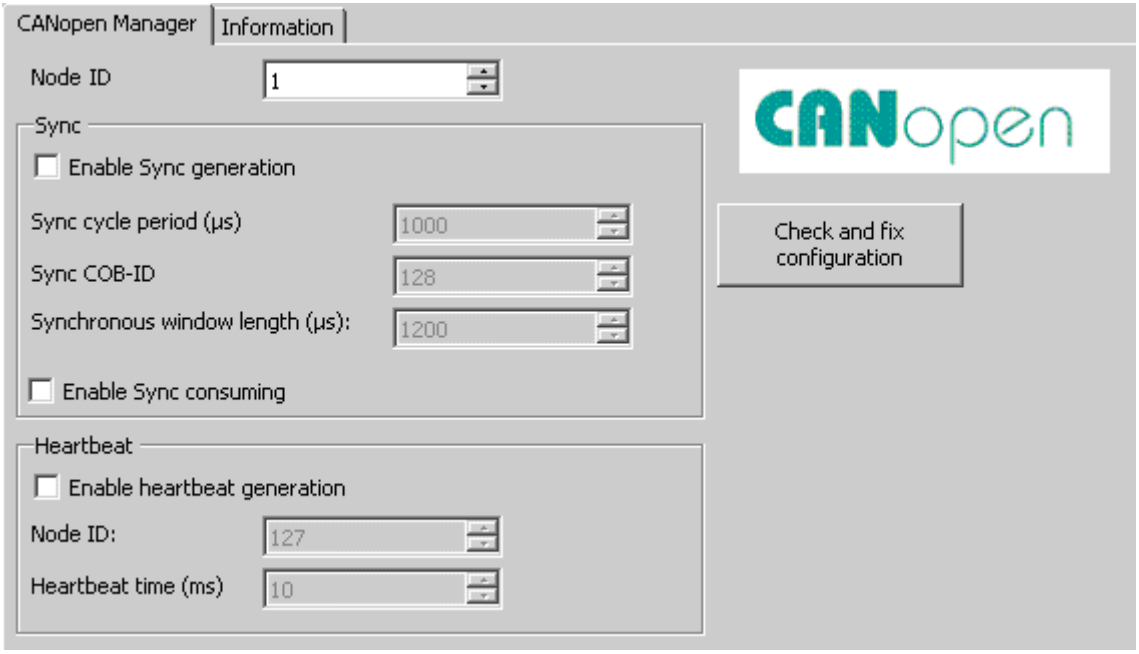
5. In the device tree, double-click on *"Buffer_for_CAN_2A (Buffer for CAN2A)"* or *"Buffer_for_CAN_2B (Buffer for CAN2B)"* to open the Buffer configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Identifier	0	CAN 2A: 0...2047 CAN 2B: 0...536870911	The value of the CAN identifier that is compared with the identifier of the incoming telegrams. The telegrams will be added to the buffer if the identifier matches.
Number of receive buffers	1	1..16	The size of the buffer in number of telegrams.
Behaviour on receive buffer overflow	Overwrite	Overwrite	The oldest telegram in the buffer is overwritten by the incoming telegram.
		Discard	The incoming telegrams are discarded as long as the buffer is full.
		Wait	The incoming telegram is stored as soon as a telegram is read out of the buffer.
Enable triggering of IEC task (only for CM598-CN)	No	No	No triggering of IEC task
		Yes	On receiving of a CAN object with this identifier the CAN manager triggers an IEC task. A task of type "triggered by external event" has to be configured. The external event has to be "CAN2A event"/"CAN2B event".

Configuration of CANopen manager

In the device tree, double-click *"CANopen_Master"* to open the configuration in the editor window. The following parameters are available on the the *"General"* tab:

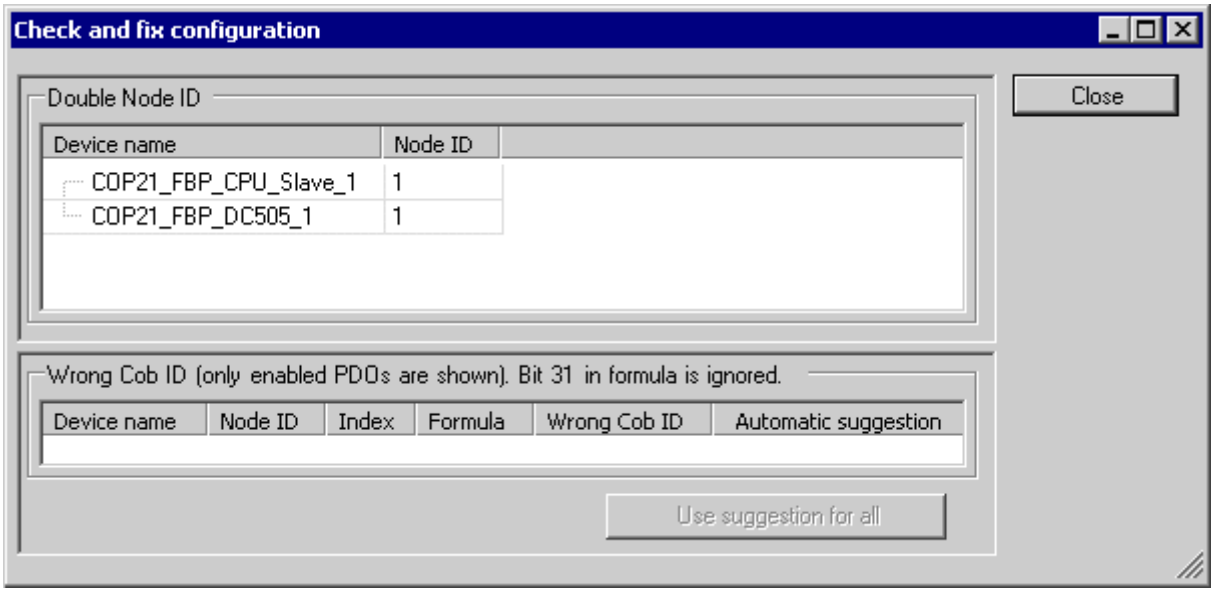


The following parameters are available:

Parameter	Default value	Value	Description
Node ID	127	1...127	Node ID of the communication module
Sync			
Sync cycle period [µs]	1000	100.. 2^{32}-1	Determines the interval in which the synchronization message will be sent.
Sync COB-ID	128	128 1664 1759 1761 1792 1920 2047	Sets the communication object identifier (COB-ID) which identifies the synchronization message.
Synchronous window length [µs]	1200	0.. 2^{32}-1	Determines the length of the time window for synchronous PDOs.
Check and fix configuration Remark 1 *Further information on page 5918*	-	-	This button opens the dialog that allows checking and fixing the current configuration.
Heartbeat			
Enable Heartbeat Producing	Disabled	Disabled	The heartbeat generation is disabled.
		Enabled	The master sends heartbeats according to the interval defined in parameter Heartbeat time. If new slaves with heartbeat functionality are added their behavior will automatically be enabled and configured appropriately. If the heartbeat generation of the CANopen manager is not enabled, nodeguarding is enabled instead. The CANopen slaves can be configured as heartbeat producers.
Node ID	127	1...127	Parameter Node ID determines the identifier of the heartbeat producer on the bus.
Heartbeat time [ms]	10	1...65535	Time interval for the heartbeat generation in [ms].

Remark 1: Check and fix configuration

The dialog Check and fix configuration shows an overview of the current conflicts of the configured node IDs and the COB-IDs. It detects double node IDs and shows the wrong COB-IDs of the PDOs of the remote devices:



The dialog gives suggestions to resolve the existing conflicts that can be accepted by clicking on the corresponding button. Alternatively the configuration must be adapted manually for the faulty devices.

Configuration of the CANopen remote devices

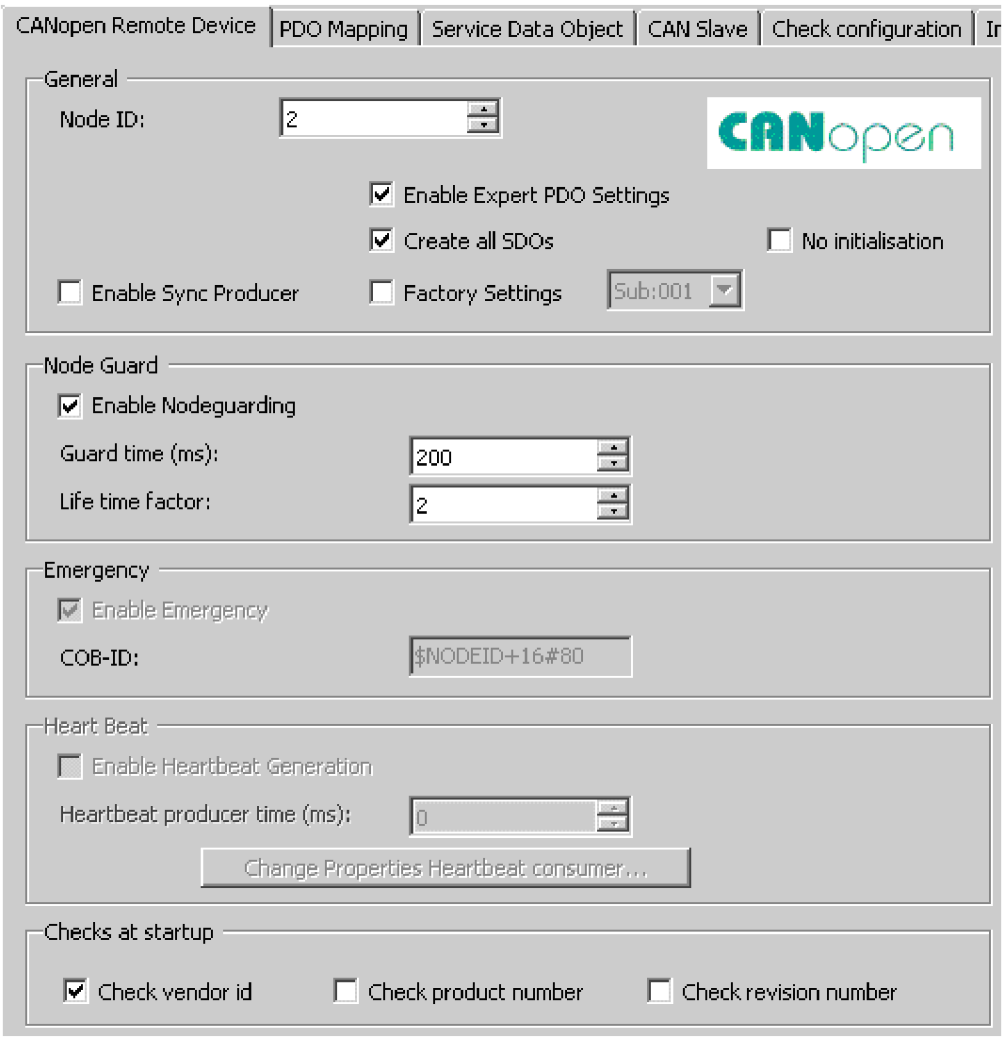
CANopen remote devices can be added to the module CM598_CAN (CM598-CAN).

Double-click on a remote device to open the corresponding CANopen remote device configuration in the editor window. The basic parameters of a remote device can be set up in *"CANopen Remote Device"* tab. See the following section for more information on the parameters.

> *If "Enable Expert PDO Settings" check box is enabled additional parameters to set PDO are displayed.*

Configuring the expert PDO settings

> *If "Enable Expert PDO Settings" check box is enabled additional parameters to set PDO are displayed.*



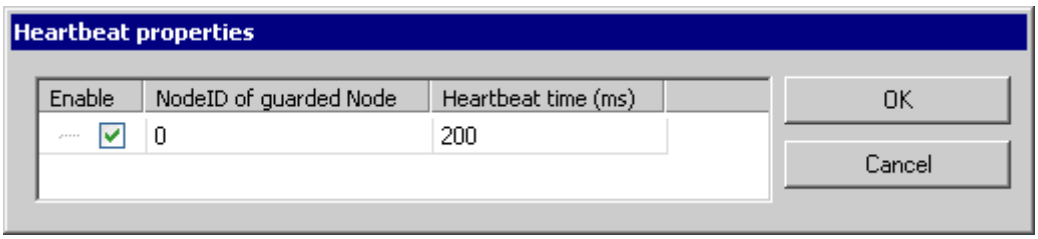
The following parameters are available:

Parameter	Default value	Value	Description
General			
Node ID	1	1...127	Note ID of the remote device
Enable Expert PDO Settings	Disabled	Disabled	Expert PDO settings are disabled.
		Enabled	Expert PDO settings are enabled.
Create all SDOs	Enabled	Disabled	Creation of all SDOs is disabled.
		Enabled	Creation of all SDOs is enabled.
No initialization	Disabled	Disabled	No initialization option is disabled.
		Enabled	No initialization option is enabled.
Enable Sync Producer	Disabled	Disabled	Sync producer is disabled.
		Enabled	Sync producer is enabled.
Factory Settings	Disabled	Disabled	No factory settings are used.

Parameter	Default value	Value	Description
		Enabled	Factory settings are used. The corresponding factory setting can be selected in the combo box right of the check box.
Node Guard			
Enable Node-guarding	Enabled	Disabled	Nodeguarding is disabled.
		Enabled	Nodeguarding is enabled. This function can only be enabled if Heartbeat Generation is disabled.
Guard time [ms]	200	0...65535	The parameter Guard time sets the cycle time for the Node-guarding in [ms].
Life time factor	2	0...255	The Life time factor sets the factor when the connection should be applied as lost. Warning: To reach a stable communication of the node on the CANopen, the Life time factor has to be set to minimal 2.
Emergency			
Enable Emergency (read only)	Enabled	Enabled	CANopen Remote Device sends emergency telegrams.
COB-ID (read only)	$NODEID+16#80	$NODEID+16#80	Communication Object Identifier of the remote device.
Heart Beat			
Enable Heartbeat Generation	Disabled	Disabled	Heartbeat generation is disabled.
		Enabled	Heartbeat Generation is enabled. This option can only be enabled if Nodeguarding is disabled.
Heartbeat producer time [ms]	10	0...65535	The Heartbeat producer time represents the cycle time for the Heartbeat Generation in [ms].
Change Properties Heartbeat consumer Remark 1 ↳ *Further information on page 5920*	-	-	The button Change Properties Heartbeat consumer opens the dialog Heartbeat Properties where the heartbeat parameters can be set.
Checks at startup			
Check vendor id	Enabled	Disabled	The vendor id will not be checked on startup.
		Enabled	The vendor id will be checked on startup.
Check product number	Disabled	Disabled	The product number will not be checked on startup.
		Enabled	The product number will be checked on startup.

Parameter	Default value	Value	Description
Check revision number	Disabled	Disabled	The revision number will not be checked on startup.
		Enabled	The revision number will be checked on startup.

Remark 1: Button *"Change Properties Heartbeat consumer"*



The dialog shows the nodes defined in the EDS files which can be selected for getting guarded. For this purpose set a check in the Enable field and enter the desired value in milliseconds in the Heartbeat time field (double-clicking on this field opens a selection box of time values). If the Heartbeat Consumer option is activated, then the corresponding module will listen to heartbeats which are sent by the master. As soon as no more heartbeats are received, the module will switch off the I/Os.

To reach a stable communication of the node on the CANopen bus the parameter Life Time factor has to be set to 2 at least.

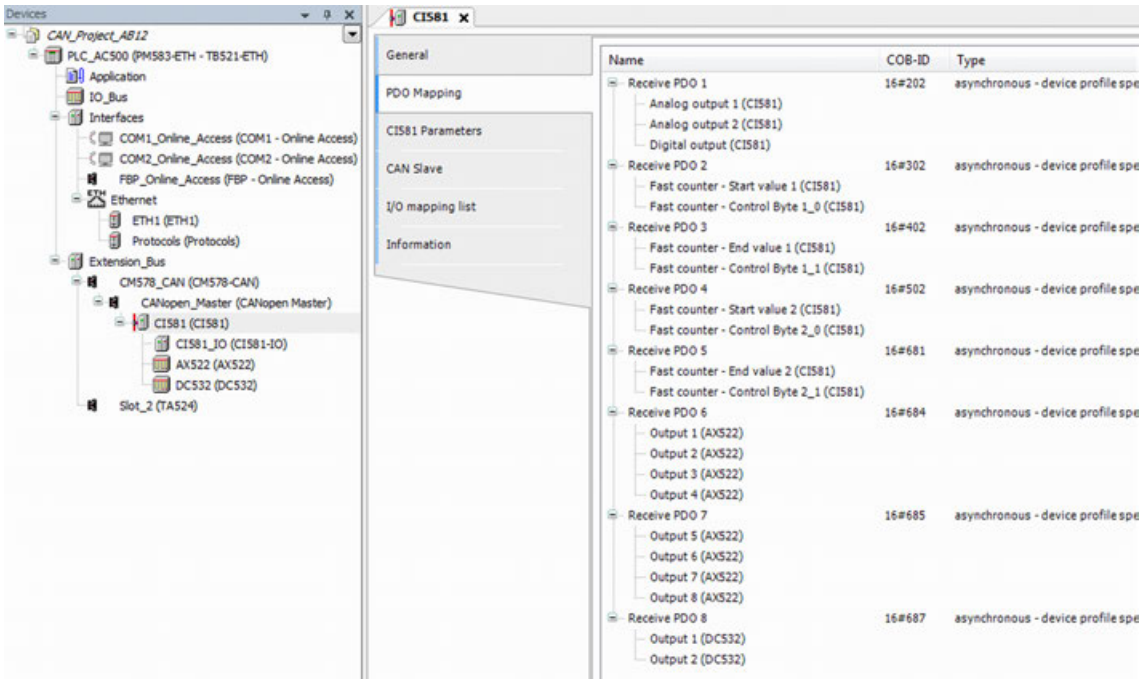
A life guarding can only be used if the master carries out a Nodeguarding. That means Life-Guarding presumes Nodeguarding.

PDO mapping editor

Overview

The PDO Mapping editor contains tree views respectively for the Receive and Transmit PDOs.

On opening the editor, all the PDO communication parameters entries are are displayed. Each PDO Mapping channel is displayed below the corresponding PDO:



Per default all PDOs are activated and preconfigured with a valid COB-ID.

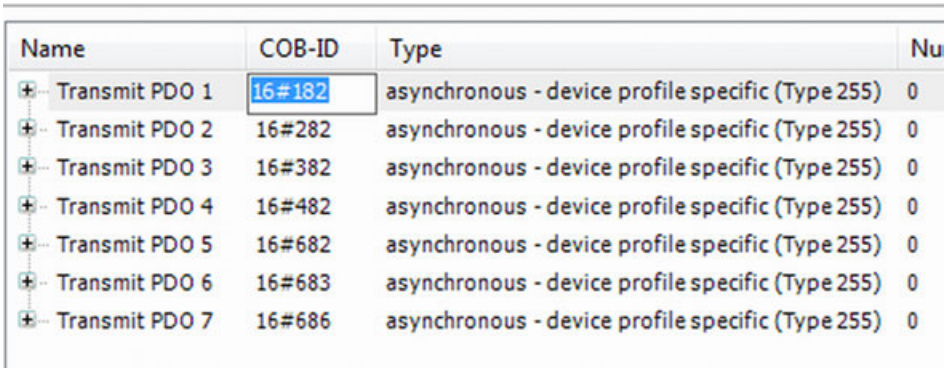
PDO configuration

Only the configurable elements are present at the user interface:

- For the Receive PDOs: COB-ID, Type and Num of syncs
- For the Transmit PDOs: COB-ID, Type, Num of syncs, Inhibit time, Event timer and RTR

The settings can be directly changed inside this editor. No additional dialog will be opened for this.

COB-ID

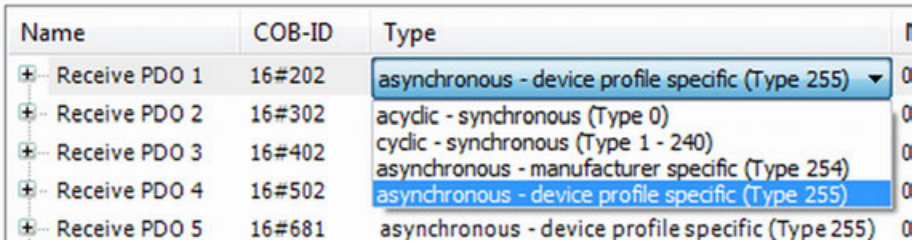


If an entered value is not correct, an error message is displayed and the entry will be reset to its original value.

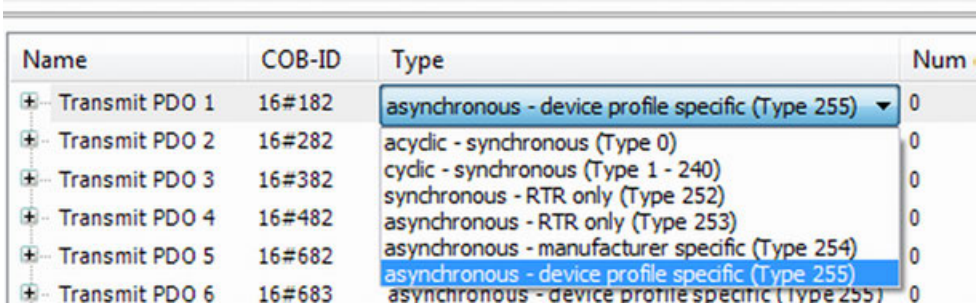
Transmission type

The allowed values for “Transmission type” depends on the type of PDO. For Receive PDOs, only Type 0, Type 1-240, Type 254 and Type 255 are allowed. The enumeration box displays in this case also only the configurable values:

- Receive PDO:

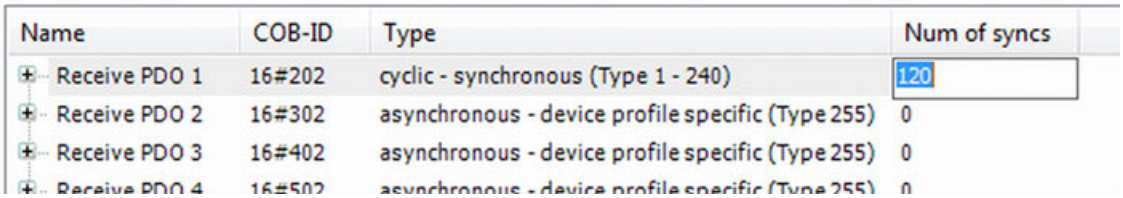


- Transmit PDO:



Number of syncs

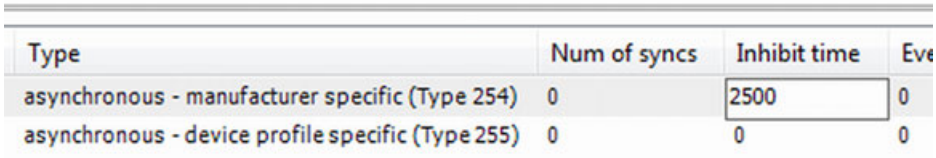
The field “Num of syncs” is only editable when the transmission type is configured to “Type 1 – 240”:



The allowed values for this field are 1 – 240. Entering a value outside this range generates an error message for the user. The value will be reset to its original value.

Inhibit time

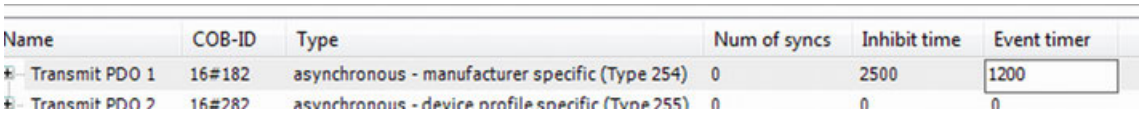
Inhibit time is only configurable for the Transmit PDOs and only when transmission type is configured to “Type 254” or “Type 255”.



The allowed values for this field are 0 – 65535. Entering a value outside this range generates an error message for the user. The value will be reset to its original value.

Event timer

Event timer is only configurable for the Transmit PDOs and only when transmission type is configured to “Type 254” or “Type 255”.



The allowed values for this field are 0 – 65535. Entering a value outside this range generates an error message for the user. The value will be reset to its original value.

RTR

RTR is only configurable for the Transmit PDOs.

When the field “RTR” is checked, the RTR flag for the PDO is set.

Configuring the service data object

During the initialization of the CANbus, the current configuration values will be transferred via Service Data Objects (SDO) to the CAN module.

In the Service Data Object configuration dialog, the desired SDOs can be configured and it can be defined in which sequence they should be transferred and what should happen in case of an incomplete transfer.

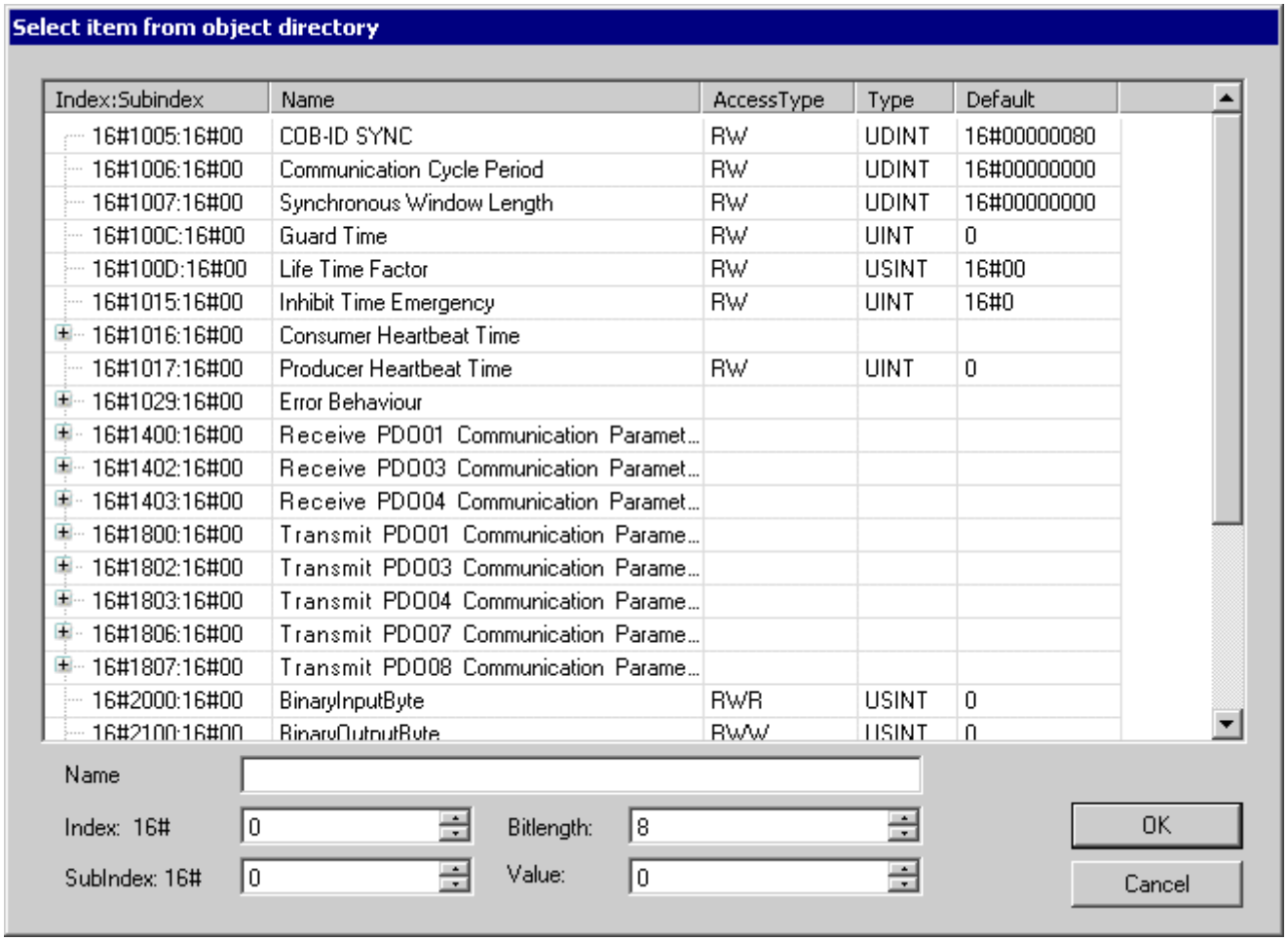
> *If option Enable expert PDO settings is not activated for the current device only the user-defined SDOs will be listed. for more information, see Enable Expert PDO Settings ↳ Chapter 1.6.5.2.6.6.1.5 “Configuration of the CANopen remote devices” on page 5920.*

The *“Service Data Object”* tab can be used to configure the Service Data objects. The view shows all parameters of the Service Data Object of the current device including the device-specific values and sizes.

Parameter	Default value	Value	Description
SDO table columns			
Line	Device-specific	Device-specific	Line number of the parameter.
Index:Subindex	Device-specific	Device-specific	Index and the subindex of the parameter.
Name	Device-specific	Device-specific	Name of the parameter.
Value	Device-specific	Device-specific	Configured value of the parameter.
Bitlength	Device-specific	Device-specific	Length of the parameter in [bit].
Abort if error	Device-specific	Device-specific	If this parameter is activated, transfer will be terminated on error detection.
Jump to line if error	Device-specific	Device-specific	If this parameter is activated, in case of an error the transfer will be continued with the line which is set in parameter 'Next line'.

Parameter	Default value	Value	Description
Next line	0	1...number of lines	In case of an error this parameter determines the line with which the transfer will be continued if the parameter 'Jump to line if error' is activated.
Comment	''	Device-specific	Shows a comment text of the corresponding line
Buttons			
Move up	-	-	The order (top-down) in the SDO table represents the sequence in which the SDOs will be transferred to the module. 'Move up' shifts up the selected entry.
Move down	-	-	The order (top-down) in the SDO table represents the sequence in which the SDOs will be transferred to the module. 'Move down' shifts up the selected entry.
New... see Remark 1	-	-	Opens the 'Select item from object directory' to add an SDO entry to the table.
Delete...	-	-	Deletes the currently selected line from the table.
Edit... see Remark 1	-	-	Opens the 'Select item from object directory' dialog where it can be edited.

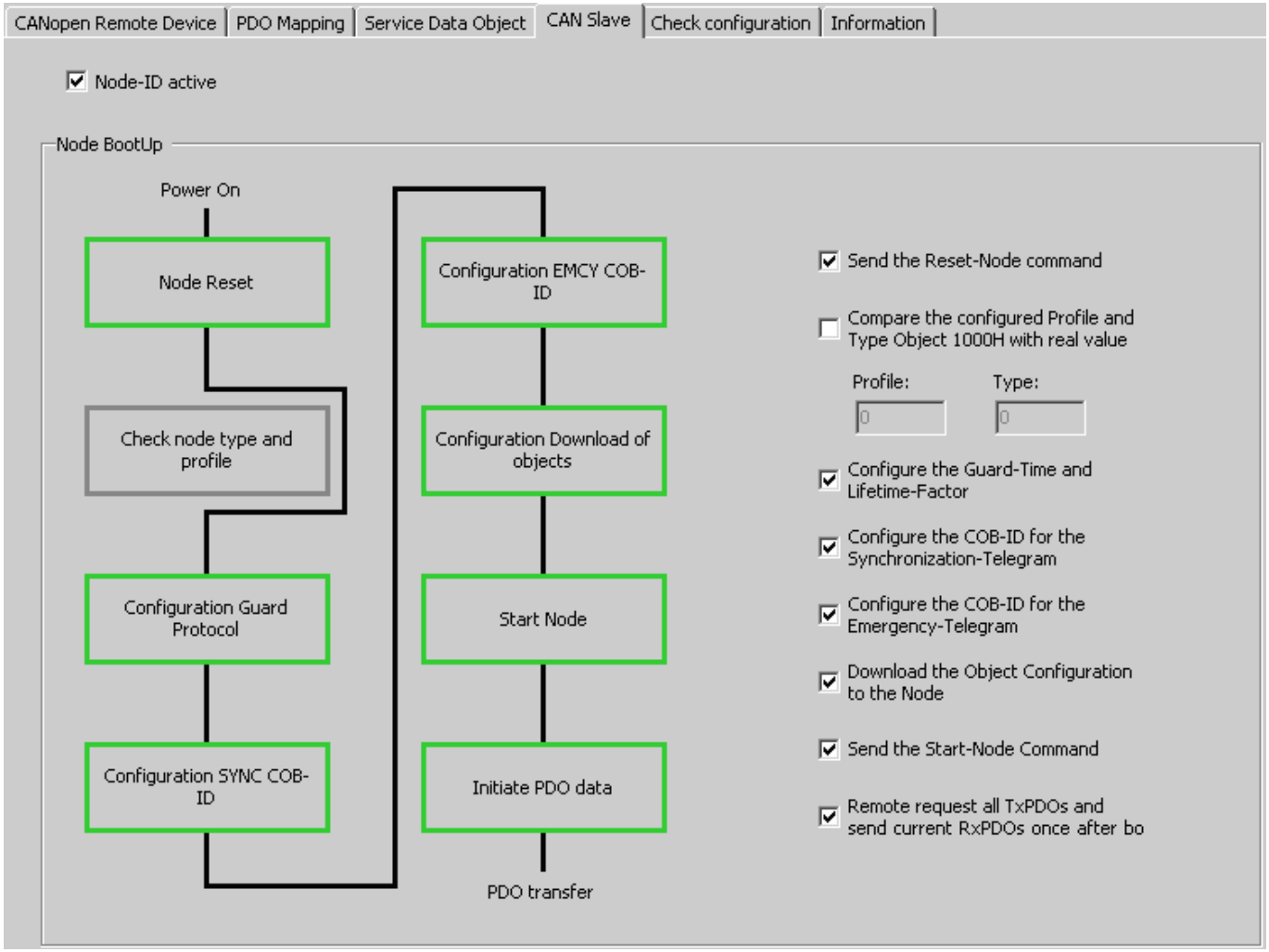
Remark 1: The dialog 'Select item from object directory' shows all entries defined in the EDS file:



The dialog allows editing the available SDOs before adding them to the table. New entries can be created and edited which are not defined in the EDS file.

Configuring the CAN slave boot up

"CAN slave" tab allows the configuration of the boot up behavior of the CAN slave:



The following options are available and can be set by either clicking on the graphics or selecting the corresponding check box on the right side of the editor window. The graphic illustrates the order of execution:

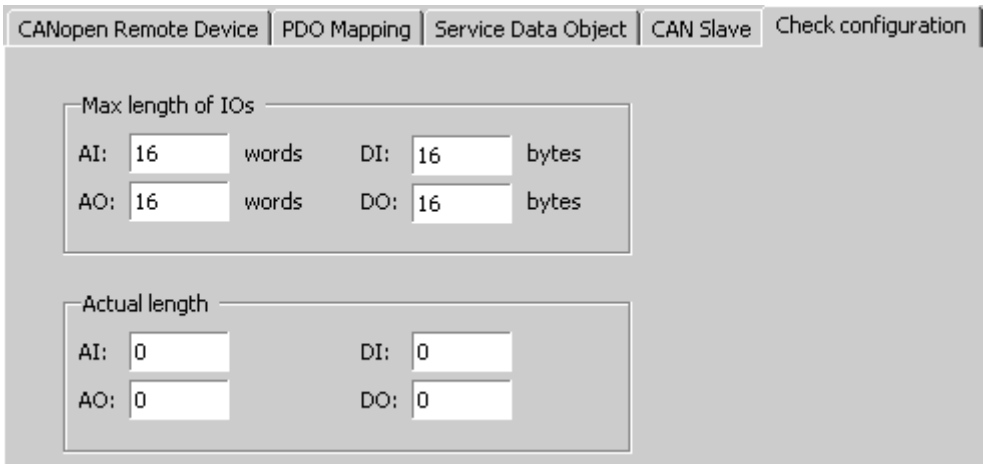
Parameter	Default value	Value	Description
Node ID-active	Enabled	Disabled	The node is inactive.
		Enabled	The node is active.
Node BootUp			
Node Reset (Send the Reset-Node command)	Enabled	Disabled	No specific node reset communication command is sent.
		Enabled	The master sends the CANopen specific node reset communication command.
Check node type and profile (Compare the configured Profile and Type Object 1000H with real value)	Disabled	Disabled	The content of the node object 1000H are not compared with the current parameters.
		Enabled	The master compares the content of the mandatory node object 1000H. If the values are different, the master will report a parameterization error.

Parameter	Default value	Value	Description
Configuration Guard Protocol (Configure the Guard-Time and Lifetime-Factor)	Enabled	Disabled	The guard time and Life-Time factor of the node configuration are not written.
		Enabled	A CANopen has 2 specific register responsible for the Node guarding protocol. The master writes the guard time and Life-Time factor of the node configuration into the corresponding objects of the node during startup.
Configuration SYNC COB-ID (Configure the COB-ID for the Synchronization-Telegram)	Enabled	Disabled	The master does not write the SYNC COB-ID of the configuration into the corresponding objects of the node.
		Enabled	The master will write the SYNC COB-ID of the configuration into the corresponding objects of the node during startup.
Configuration EMCY COB-ID (Configure the COB-ID for the Emergency-Telegram)	Enabled	Disabled	The master does not write the EMCY COB-ID of the configuration into the corresponding of the node.
		Enabled	The master writes the EMCY COB-ID of the configuration into the corresponding objects of the node during startup.
Configuration Download of objects (Download the Object Configuration to the Node)	Enabled	Disabled	
		Enabled	To get a PDO communication to a node working, the master has to send all relevant configuration objects to the Node. For example, the COB-IDs of PDOs are covered here in the mapping table. If enabled, all these parameter and also the user specific objects which are added manually in the Node object configuration window are written down to the Node by the master.
Start Node (Send the Start-Node Command)	Enabled	Disabled	No start node command is sent.
		Enabled	To reach the operational state in CANopen, a node has to get the CANopen specific start node command. The master sends the start node command to the node at the end of the boot-up procedure.

Parameter	Default value	Value	Description
Initiate PDO data (Remote request all TxPDOs and send current RxPDOs once after	Enabled	Disabled	PDOs are not written and read by the master automatically.
		Enabled	This item selects if the installed PDOs shall be automatically written and read by the master directly after the startup once. This ensures that the latest output data which can be found within the master's output process data area is sent to the node and that the latest node input data is read from the node and be placed into the input process data area.

Checking the I/O configuration

Tab *"Check configuration"* can be used to check the current I/O configuration. The device specific number of available I/Os can be compared with the actually configured I/Os to detect a faulty I/O configuration before downloading the program:



Parameter	Default value	Value	Description
Max. I/O length			
AI	Device specific	0...128	Available analog inputs of the device in [words].
AO	Device specific	0...128	Available analog outputs of the device in [words].
DI	Device specific	0...32	Available digital inputs of the device in [bytes].
DO	Device specific	0...32	Available digital outputs of the device [bytes].
Actual length			
AI	0	Read-only	Actually configured length of analog inputs in [words].
AO	0	Read-only	Actually configured length of analog outputs of the device in [words].

Parameter	Default value	Value	Description
DI	0	Read-only	Actually configured length of digital inputs in [bytes].
DO	0	Read-only	Actually configured length of digital outputs [bytes].

CM588-CN - CANopen slave communication module

Configuration steps

The configuration of the CM588-CN CANopen slave module has to be done in the following steps:

- Parameterization of the AC500 Communication Module interface ↳ *Chapter 1.6.5.2.6.6.2.2 "Parameterization of the CM588-CN communication module interface" on page 5932*
- Configuring the module CM588-CN for use as CANopen Slave device ↳ *Chapter 1.6.5.2.6.6.2.2 "Parameterization of the CM588-CN communication module interface" on page 5932*
- Parameterization of the CANopen slave protocol stack ↳ *Chapter 1.6.5.2.6.6.2.3 "Parameterization of the CANopen slave protocol stack" on page 5933*
- Configuring I/O data objects ↳ *Chapter 1.6.5.2.6.6.2.4 "Configuration of I/O data objects" on page 5933*
- Mapping of the CANopen Slave I/Os ↳ *Chapter 1.6.5.2.6.6.2.5 "Mapping of the I/Os" on page 5934*

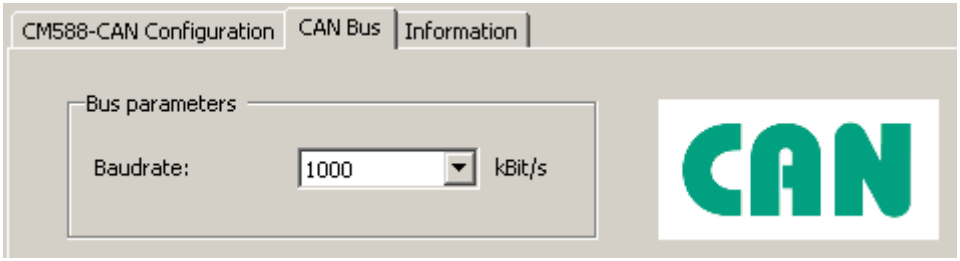
Parameterization of the CM588-CN communication module interface

Double-click on CM588_CAN (CM588-CAN) to open the CM588-CAN configuration in the editor window.

The following parameters are available:

Parameter	Default	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even in case of configuration error.
Min. update time	10 ms	0 ms ... 20000 ms	Update time of inputs and outputs
Transmission rate	1000 kBit/s	1000 kBit/s 800 kBit/s 500 kBit/s 250 kBit/s 125 kBit/s 100 kBit/s 50 kBit/s 20 kBit/s 10 kBit/s	Supported transmission rates of the Communication Module

Parameter transmission rate can be configured also by using the *"CAN Bus"* tab.



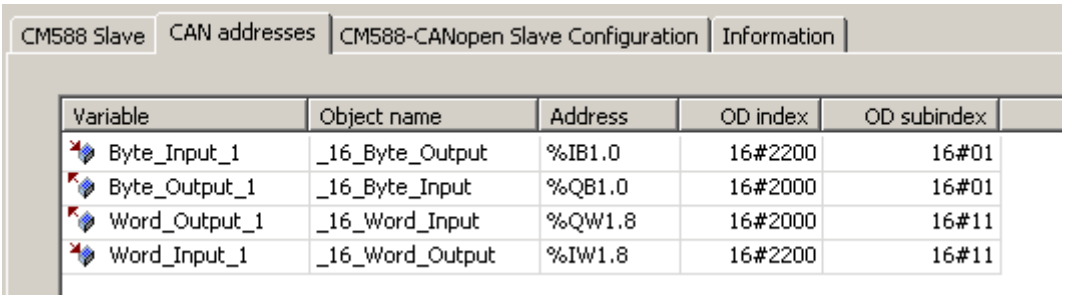
Parameterization of the CANopen slave protocol stack

Double-click on CM588_CANopen_Slave (CM588-CANopen slave) to open the *"CM588 Slave"* tab.

The following parameters are available:

Parameter	Default	Value	Description
Node-ID	1	1...127	The node-ID the CANopen Slave Communication Module uses on acting at the network.

"CAN addresses" tab provides overview of assigned I/O variables and how they are mapped to CANopen objects located in slave object dictionary. Find these object indexes and sub-indexes on looking at the PDO configuration of the CM588 CANopen slave from a CANopen Master configuration point of view.



"CAN addresses" tab only shows read-only data.

Further parameters are available in the *"CM588-CANopen Slave Configuration"* tab. Activate parameter *"Show generic device configuration views"* to get this tab visible. This setting is available under *"Tools ➔ Options"*.

Configuration of I/O data objects

Data objects added to CM588-CN CANopen Slave configures I/O data. Inserting variable names at the data objects maps the variables to dedicated I/O channels.

Several types of data object groups I/O variables by size and direction. The I/O driver of the PLC firmware copies the amount of data bytes configured by these data objects cyclically. The copy cycle time is defined by the parameter *"Min. update time"*.

Keep basic load of PLC low by configuring only as much data objects as really needed. If further I/O variables need to be added later additional data objects can be inserted.

The maximum amount of bytes used for I/O data is limited to 512 bytes for input data and 512 bytes for output data.

Inserting new data objects checks these limits and generates an error message if the new object does not fit. Pasting a single or a number of copied data objects checks these limits also. In case of a limit violation the paste operation is canceled completely and no data object is inserted.

Mapping of the I/Os

Double-click on the desired I/O data object in the device tree to show current I/O mapping connected to this data object.

See chapter Symbolic Names for Variables, Inputs and Outputs for further details on mapping inputs and outputs ↳ *Chapter 1.6.5.2.2.1.2 “Symbolic names for variables, inputs and outputs” on page 5815*.

PROFINET

CM579-PNIO - PROFINET communication module

The configuration of the PROFINET fieldbus has to be done in the following steps:

- Parameterization of the AC500 Communication Module Bus Interface ↳ *Chapter 1.6.5.2.6.7.1.1 “Parameterization” on page 5934*
- Configuration of the PROFINET IO controller ↳ *Chapter 1.6.5.2.6.7.1.2 “Configuration of the PROFINET IO controller” on page 5935*
- Configuration of the PROFINET IO devices:
  ABB PROFINET IO devices
  – ↳ *Chapter 1.6.5.2.7.2 “CI504-PNIO/CI506-PNIO” on page 5970*
  – ↳ *Chapter 1.6.5.2.6.7.1.3 “Configuration of ABB PROFINET IO devices” on page 5936*

  3rd party devices)
  – ↳ *Chapter 1.6.5.2.6.7.1.4 “Configuration of 3rd party PROFINET IO devices” on page 5941*
- Configuration of the PROFINET IO device names
- Configuration and mapping of the I/Os ↳ *Chapter 1.6.5.2.6.7.1.7 “Mapping of the PROFINET IO devices” on page 5950*
- Name Assignment of a PROFINET IO device ↳ *Chapter 1.6.5.2.6.7.1.6 “Name assignment of a PROFINET IO device” on page 5949*
- ↳ *Chapter 1.7.2 “Online diagnosis in Automation Builder” on page 6374*

Parameterization

To append a Communication Module, add the Communication Module to the *“Extension_Bus”* node.

- Right-click the desired slot and select *“Add object”*.
- Select the Communication Module from the list and click *[Replace object]*.

- Double-click on *"CM579_PNIO (CM579-PNIO)"* to open the CM579-PNIO configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the CM579-PNIO communication module.
Min update time	10	0...20000	Minimum update time of inputs and outputs in [ms].
Inhibit error signaling on LED STA2	Off	On/Off	Inhibit error signaling on LED STA2 of PROFINET IO controller CM579-PNIO

Configuration of the PROFINET IO controller

Double-click on *"PNIO_Controller"* to open the PROFINET IO controller configuration in the editor window.

The following parameters are available:

Parameter	Default	Value	Description	Parameter
Identification				
IP-Address	192.168.0.1	Valid IP address	IP address of the PROFINET IO controller station.	IP address
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO controller station.	Subnet Mask
Default gateway	0.0.0.0	Valid gateway address	Gateway address of the PROFINET IO controller station.	Default gateway
Station name	controller	Up to 240 characters	Network name of the PROFINET IO controller station. Must be a valid hostname.	Station name
Address settings for devices				
First IP-Address	192.168.0.2	Valid IP address	First IP address of the PROFINET IO devices. This parameter determines the address range of the PROFINET IO devices in combination with parameter Last IP address.	First IP address

Parameter	Default	Value	Description	Parameter
Last IP-Address	192.168.0.254	Valid IP address	Last IP address of the PROFINET IO devices. This parameter determines the address range of the PROFINET IO devices in combination with parameter First IP address.	Last IP address
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO devices.	Default subnet mask
Default gateway	0.0.0.0	Valid gateway	Gateway address of the PROFINET IO devices.	Default gateway address

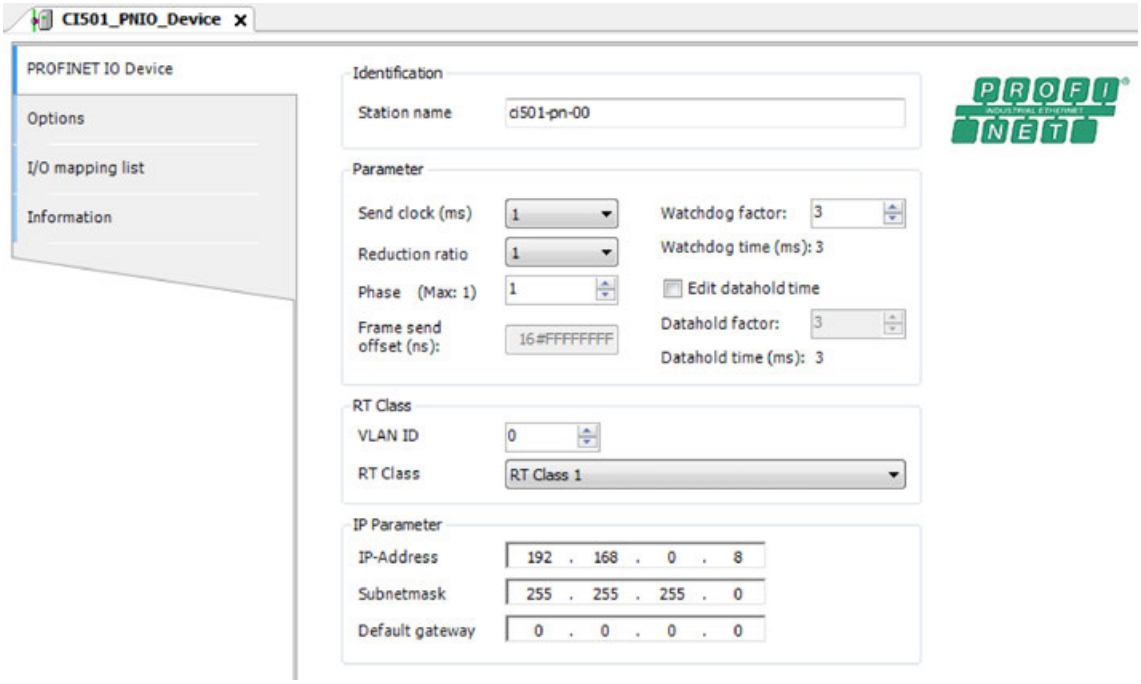
Chapter 1.6.5.2.6.7.1.3 “Configuration of ABB PROFINET IO devices” on page 5936

Configuration of ABB PROFINET IO devices

A PROFINET IO device can be added by right-clicking on a master module, e.g. *CM589-PNIO* and selecting *“Add device”*.

PROFINET IO device

Edit the communication parameters:



The following parameters are available:

Parameter	Default	Value	Description	Parameter
Identification				
Station name	Device-specific	Up to 240 characters	This is a system wide unique name for addressing the device. Must be a valid host-name.	Slave parameters -> Identification -> Station name

Parameter	Default	Value	Description	Parameter
Parameter				
Send clock (ms)	Device-specific	0.25 0.5 1 2 4	Parameter Send clock determines the SendCycle. SendCycle = Send clock x Reducation ratio <= 512ms x	Slave parameters -> Reduction ratio
Reduction ratio	Device-specific	1...16384	The Reduction ratio determines the factor for calculating the cycle time. Cycle time = Send clock x Reducation ratio	Slave parameters -> Reduction ratio
Phase	1	1…Reduction ratio	Defines the part of the SendCycle at which an IO frame is sent.	Phase
Watchdog factor	3	1...65535	The Watchdog time is calculated as Watchdog time = SendCycle * Watchdog factor. The transfer of a IO telegram is always checked of the consumer side. Within this time the next IO telegram must be received by a consumer. Otherwise it is checked if the Datahold has been expired too.	Watchdog factor
Datahold factor	3	1...65535	The Data Hold Time is calculated as Datahold time = SendCycle * Datahold factor. If the Watchdog time has been expired, the Data Hold Time will checked. If the Data Hold Time also has been expired, the substitution values of the IOs will be used. For RT_Class_1 communication Data Hold Time and the Watchdog time are usually configured with the same value.	Datahold factor
Frame send offset	0xFFFFFFFF (means: as soon as possible)	1...2^32 - 1	Only available for RT Class 3 Data-RTC-PDU	Frame send offset
RT Class				
RT Class	RT Class 1 Data-RTC-PDU	RT Class 1 Data-RTC-PDU RT Class 2 Data-RTC-PDU RT Class 3 Data-RTC-PDU	Defines the Realtime Class of cyclic data. Currently only RT Class 1 (legacy) and RT Class 1 are supported.	Slave parameters -> RT Class

Parameter	Default	Value	Description	Parameter
		RT Class UDP-RTC-PDU		
VLAN ID	0	0..4095 or 0..32767	In case of VLAN usage the parameter VLAN ID represents the ID of the virtual network. For VLAN type 802.1Q the range is 0..4095 while VLAN type ISL accepts values from 0 to 32767. The supported type depends on the used device.	Slave parameters -> VLAN ID
IP Parameter				
IP-Address	192.168.0.8	Valid IP address	IP address of the PROFINET IO Controller station.	Slave parameters -> Identification -> IP address
Subnetmask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO Controller station.	Slave parameters -> Identification -> Subnet mask
Default gateway	0.0.0.0	Valid gateway address	Default gateway address of the PROFINET IO Controller station.	Slave parameters -> Identification -> Default gateway address

Chapter 1.6.5.2.6.7.1.3 “Configuration of ABB PROFINET IO devices” on page 5936

PROFINET IO timing

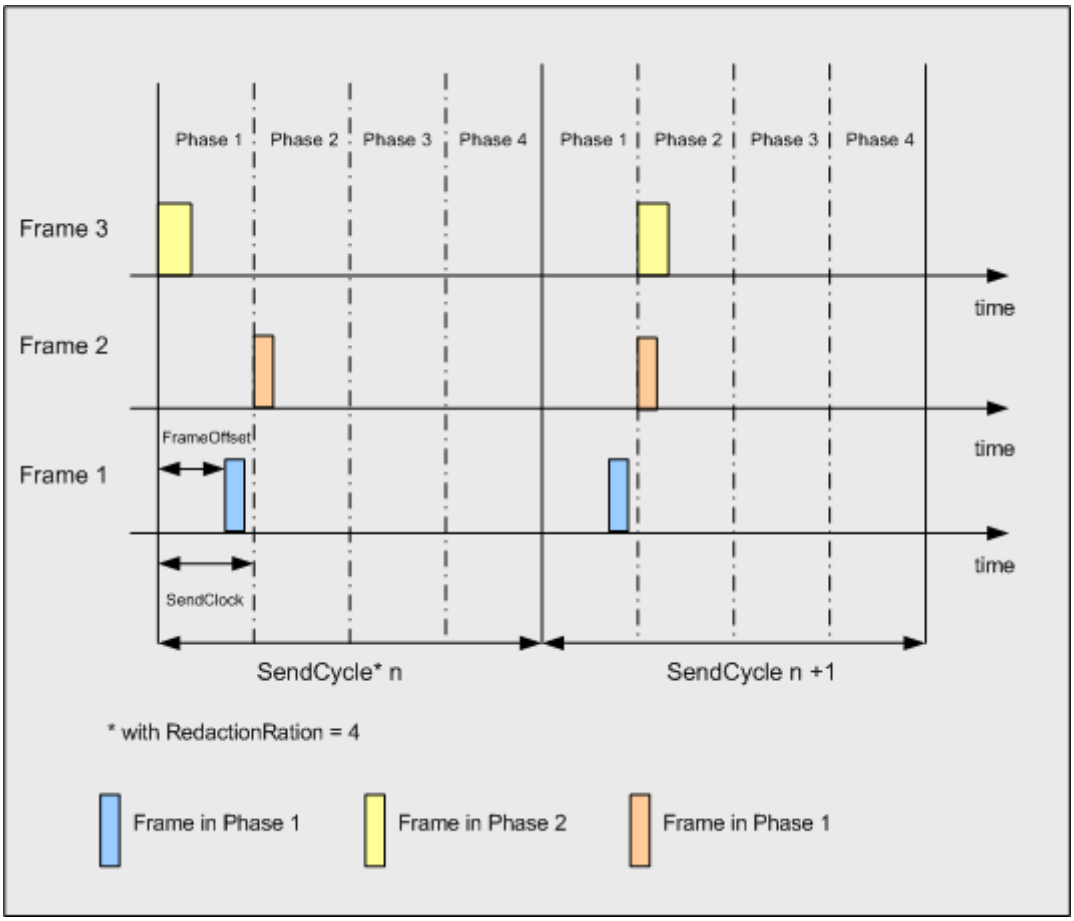
In the current implementation per device 2 *Communication Relations* (CR) between the controller and the device are defined. One CR describes the IO telegram from the controller to the device (outputs) and the other the IO telegram from the device to the controller (inputs).

The timing of the corresponding IO telegrams can be defined separately for each device.

The editable timing parameters are:

- Send clock
- Reduction ratio
- Phase
- FrameSendOffset.

The relation between these parameters is shown in the following drawing.



For each device a SendCycle must be configured, which determines the sending interval of IO frames.

It is based on a time base of 31.25 µs und is calculated as:

SendClock [ms] = SendClockFactor * 31.25 µs / 1000.

The cycle time of an IO telegram is defined by the SendCyle. It's calculated as:

SendCyle [ms] = SendClock [ms] * Reduction Ratio.

The values of the individual parameters are limited by the maximum value 512 ms of the SendCycle.

The following table summarizes the relation of the timing parameters.

Parameter	Description	Relation	Range
SendCycle	Is the cycle time of a RT telegrams.	SendCyle = Send-Clock * Reduction ratio.	1ms ..512 ms
SendClock	The SendCycle is divided into several time slots. The SendClock defines the size of a time slot within the SendCyle.	SendClock = SendClock factor * 31.25 µs;	SendClock * Reduction ratio ≤ 512 ms
SendClock factor	Is multiplied with the time base 31.25 µs to calculate the SendClock.	SendClock factor = Send-Clock / 31.25 µs	1..128
Reduction ratio	The reduction ratio defines the number of time slots within the SendCyle.	SendClock * Reduction factor ≤ 512 ms	1..16384

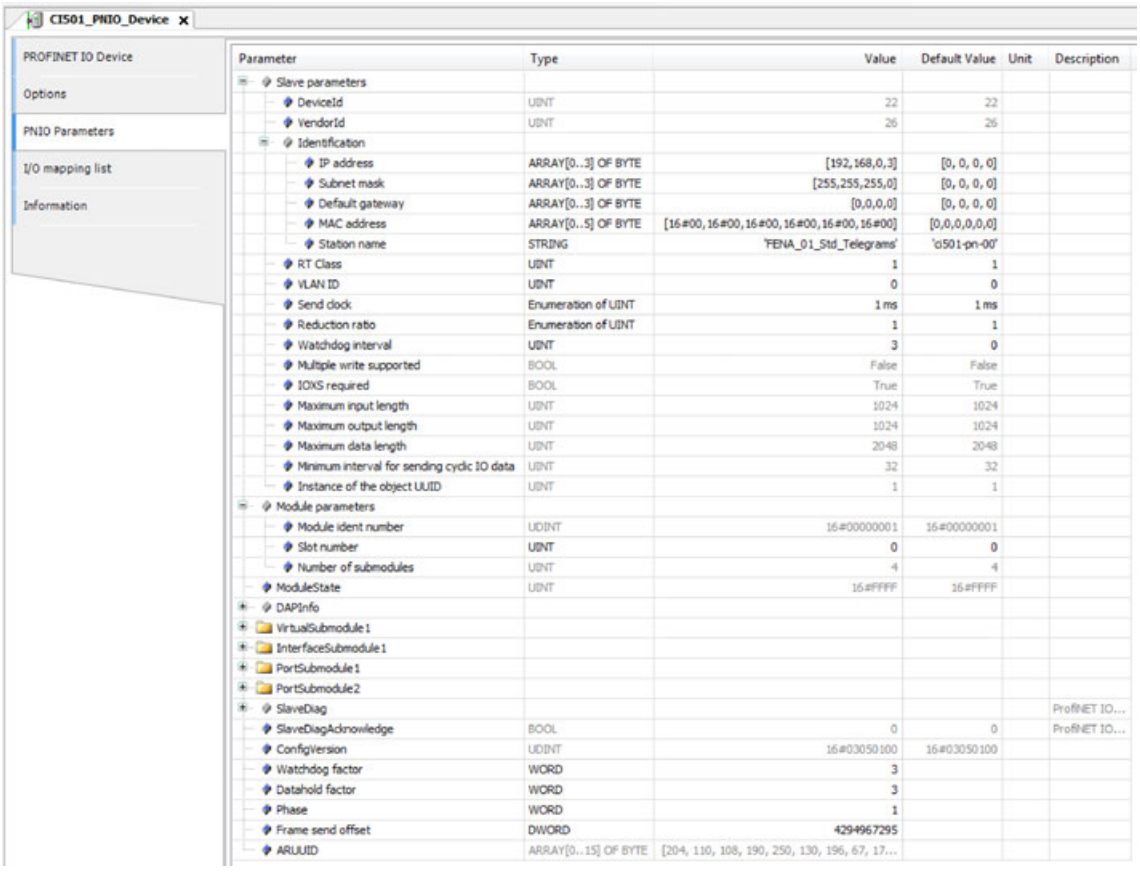
Parameter	Description	Relation	Range
Phase	The time slot in which the IO frame is sent.	A integer value of the range 1 … Reduction ratio	1..16384
FrameSen-dOffset	It's an offset value relative to the beginning of a time slot. For all RT_Classes the value 0xFFFFFFFF means sending as soon as possible.	Only for RT_CLASS_3 it is necessary to sup-port this param-eter.	0x0 … 0xFFFFFFFF

PNIO configuration

This tab is a generic view of all PROFINET IO device parameters. It is normally hidden and is normally not needed for configuration.

Use this dialog only, if you need to change a parameter, which is not visible in other dialogs.

This dialog could be activated with the parameter *"Show generic device configuration views"* in the Automation Builder options (*"Tools ➔ Options ➔ Device editor"*).



CI504-PNIO/CI506-PNIO

Configuration of CI504-PNIO and CI506-PNIO is described in the PROFINET IO Controller configuration ↳ *Chapter 1.6.5.2.7.2 "CI504-PNIO/CI506-PNIO" on page 5970*.

Configuration of 3rd party PROFINET IO devices

A *PROFINET IO device* can be added by right-clicking on *CM579_Master (CM579-PNIO master)* and selecting *"Add device"*.

The *"Add Device"* dialog appears where all available PROFINET IO devices are listed.

If the desired device is not listed you can add this device to the *"Device Repository"* by installing the associated *GSDML-File* (via *"Tools ➔ Device Repository ➔ Install"*).

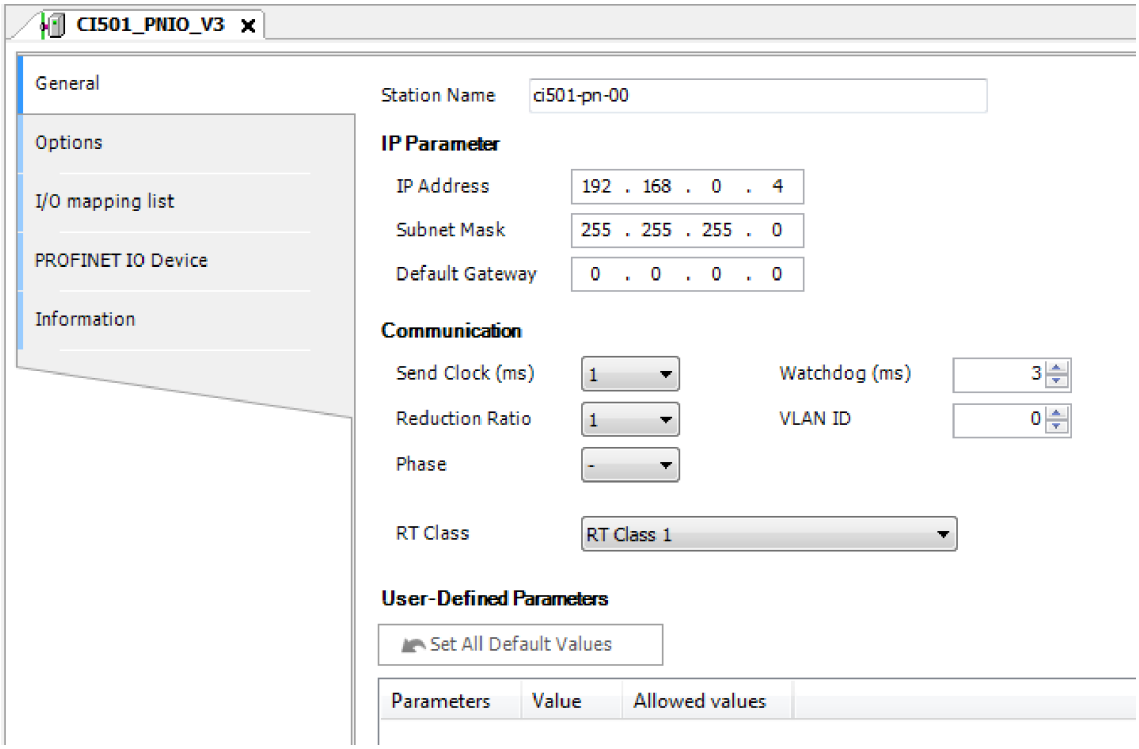
The device parameters are edited in the PROFINET IO device dialog, which is divided into 2 tab-sheets:

- PNIO Parameters
- PNIO Configuration.

To open the PROFINET IO device dialog, double-click on the desired device in the configuration tree.

PNIO parameters

In this *"General"* tab-sheet the *"IP Parameter"*, the *"Communication"* and the *"User-defined Parameters"* will be edited.



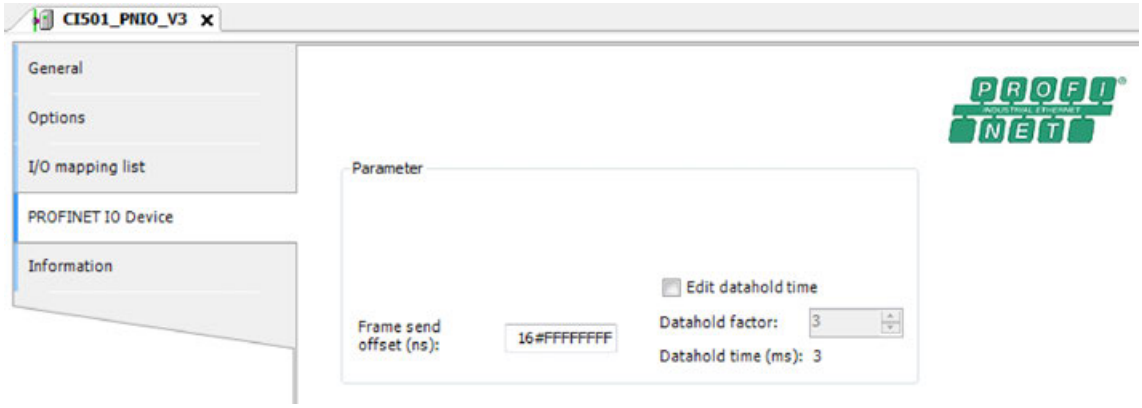
The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1)
Station name	Device-specific	Up to 240 characters	This is a system wide unique name for addressing the device. Must be a valid host-name.	Slave parameters -> Identification -> Station name
IP Parameter				

Parameter	Default	Value	Description	Parameter (Remark 1)
IP Address	192.168.0.4	Valid IP address	IP address of the PROFINET IO controller station.	Slave parameters -> Identification -> IP address
Subnet mask	255.255.255.0	Valid subnet mask	Network mask of the PROFINET IO controller station.	Slave parameters -> Identification -> Subnet mask
Default Gateway	0.0.0.0	Valid gateway address	Default gateway address of the PROFINET IO controller station.	Slave parameters -> Identification -> Default gateway address
Communication				
Send Clock (ms)	Device-specific	Device-specific	Parameter Send clock determines the sending interval in [ms].	Slave parameters -> Send clock
Reduction Ratio	Device-specific	1...512	The Reduction ratio determines the factor for calculating the SendCycle. Cycle time = Send clock x Reducation ratio <= 512ms	Slave parameters -> Reduction ratio
Phase	-	1…Reduction ratio	Defines the part of the SendCycle at which an IO frame is sent.	Phase
RT Class	RT Class 1 Data-RTC-PDU	RT Class 1 Data-RTC-PDU	Defines the real-time Class of cyclic data. Currently only RT Class 1 (legacy) and RT Class 1 are supported.	Slave parameters -> RT Class
		RT Class 2 Data-RTC-PDU		
		RT Class 3 Data-RTC-PDU		
		RT Class 1 UDP-RTC-PDU		
Watchdog [ms]	3	1...65535	The Watchdog time is calculated as Watchdog time = SendCycle * Watchdog factor. The transfer of a IO telegram is always checked of the consumer side. Within this time the next IO telegram must be received by a consumer. Otherwise it is checked if the Datahold has been expired too.	Watchdog time

Parameter	Default	Value	Description	Parameter (Remark 1)
VLAN ID	0	0..4095 or 0..32767	In case of VLAN usage the parameter VLAN ID represents the ID of the virtual network. For VLAN type 802.1Q the range is 0..4095 while VLAN type ISL accepts values from 0 to 32767. The supported type depends on the used device.	Slave parameters -> VLAN ID
User-Defined Parameters				
User-Defined Parameters	Device-specific	Device-specific	The parameters of a PROFINET device are stored as RecordData items. The table "User parameter" is a generic view of the elements of a RecordData item. The Record Data item and it's default values are read from *GSDML-File*.	-
Set All Default Values	-	-	The button restores the default values of the user parameters.	-

In this *"PROFINET IO device"* tab-sheet the following *"Parameter"* will be edited.



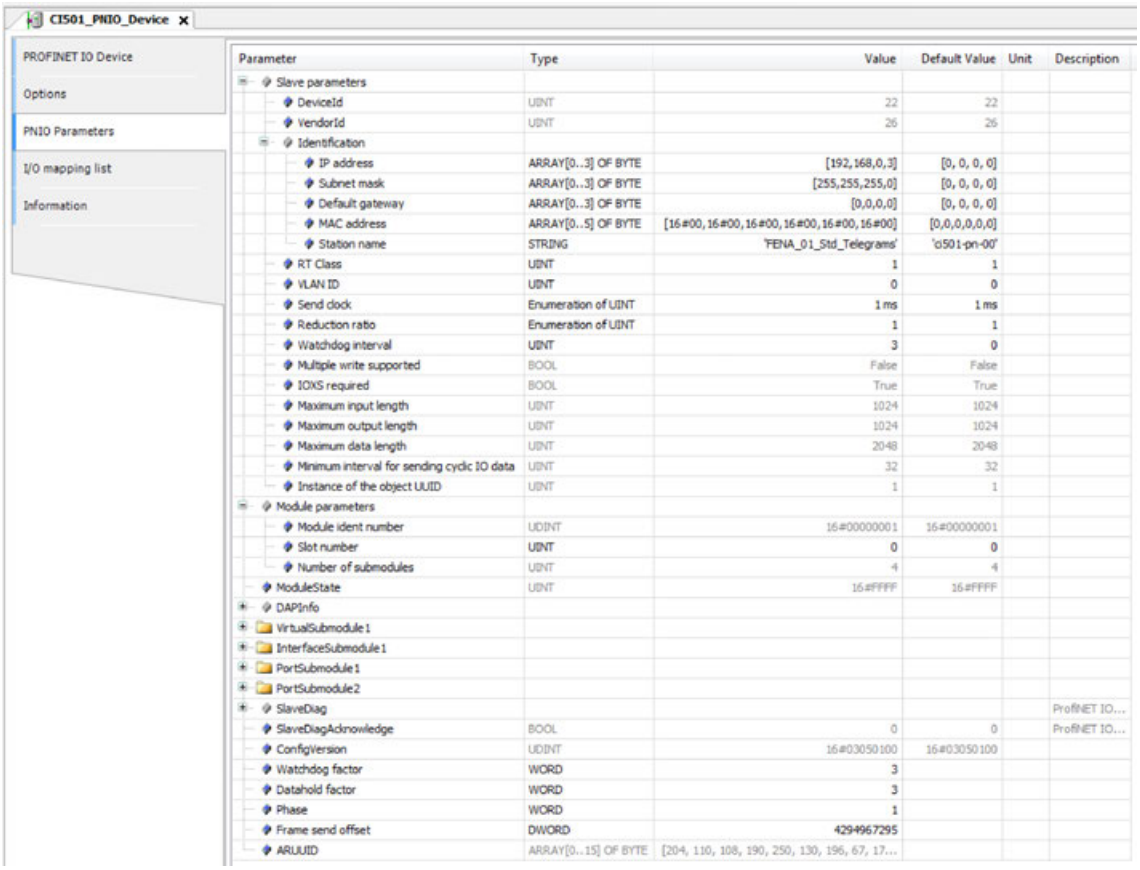
Parameter	Default	Value	Description	Parameter (Remark 1)
Frame send offset [ns]	0xFFFFFFFF (means: as soon as possible)	1...2^32 - 1	Only available for RT Class 3 Data-RTC-PDU	Frame send offset
Datahold factor	3	1 ...65535	The Data Hold Time is calculated as Datahold time = SendCycle * Datahold factor. If the Watchdog time has been expired, the Data Hold Time will checked. If the Data Hold Time also has been expired, the substitution values of the IOs will be used. For RT_Class_1 communication Data Hold Time and the Watchdog time are usually configured with the same value.	Datahold factor

PNIO configuration

This tab is a generic view of all PROFINET IO device parameters. It is normally hidden and is normally not needed for configuration.

Use this dialog only, if you need to change a parameter, which is not visible in other dialogs.

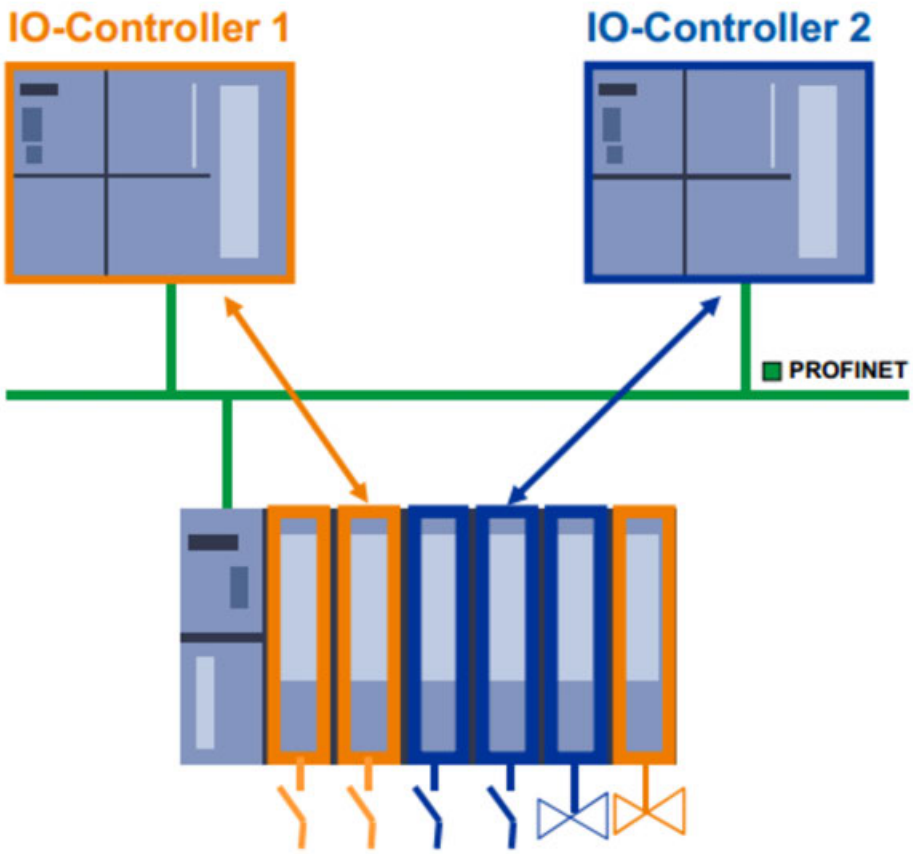
This dialog could be activated with the parameter *"Show generic device configuration views"* in the Automation Builder options (*"Tools ➔ Options ➔ Device editor"*).



Configuration of PROFINET shared device functionality

"PROFINET Shared Device Functionality" enables an IO device to be connected to more than one IO controller at the same time. In this situation the IO controllers share the IO devices IO modules/submodules to have access to the IO data. Each IO controller exclusively connects a set of IO modules/submodules as defined in its configuration. Accessing the same IO device modules/submodules by more than one IO controller in parallel is not allowed for Shared Device.

Following figure taken from *"PROFINET IO System Description"* as provided by PNO gives an impression about the concept of *"Shared Device Functionality"*.

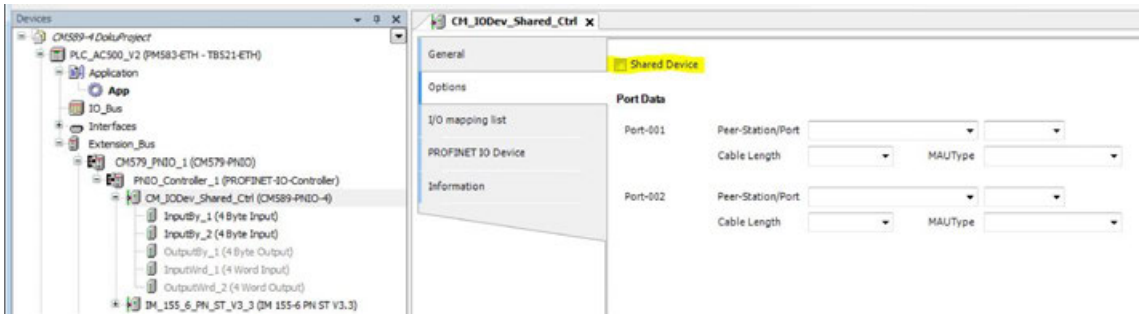


IO Controller 1 connects the modules labeled with orange color.
IO Controller 2 connects the modules labeled with blue color.

Each IO controller that should connect the shared IO device has to have this device configured in its device list. In Automation Builder all configurations of the shared IO device have to look the same at all IO controllers; all modules/submodules have to be configured in the same order. Each IO controller has to be configured which modules/submodules should be connected.

One of the IO controllers has to be defined as responsible to connect the IO device access point (DAP) with its assigned submodules. To do so Automation Builder provides the parameter *"Shared Device"*. This parameter can be found in the IO device configuration as part of the device list sub-tree of the IO controller. Parameter *"Shared Device"* defines which IO controller will connect the DAP and which IO controllers use the IO device just as shared device.

To connect the DAP this parameter must not be checked; at all other IO controllers this parameter has to be checked.



If this rule is not followed accurately the PROFINET communication will start anyway but the IO controllers / IO devices may signal errors.

Two possible error cases have to be considered:

Error Case 1	Effect
All IO controllers have parameter 'Shared Device' selected. Thus, no IO controller will connect the DAP to load related parameters.	No error will be signaled; PROFINET network will run with default parameters.

Error Case 2	Effect
More than one IO controller has parameter 'Shared Device' unselected. These IO controllers try to connect to the DAP in parallel.	One IO controller will succeed. The other IO controllers will fail but further modules/submodules can be connected with success. Communication will run to connected modules/submodules. The IO device responds with error signaling DAP's submodules as already connected.

In **Error case 2** IO controller and IO device specific diagnosis pages in Automation Builder, will show error information. See ↳ *Chapter 1.7.2.7.5.1 "IO controller views" on page 6408* and ↳ *Chapter 1.7.2.7.6 "CM579-PNIO PROFINET IO device views" on page 6413*.

In display *"CM579-PNIO PROFINET IO Device Views"* field *"Diagnosis state"* shows a text message describing the error. Basically this text depends on the type of device signaling the error. In case of *CM589-PNIO-4* the text *"The IO-Device reported a ModuleDiffBlock during connection establishment"* will be found.

Take care to have parameter "Shared Device" unselected at one IO controller device and have it selected at all other involved IO controller devices.

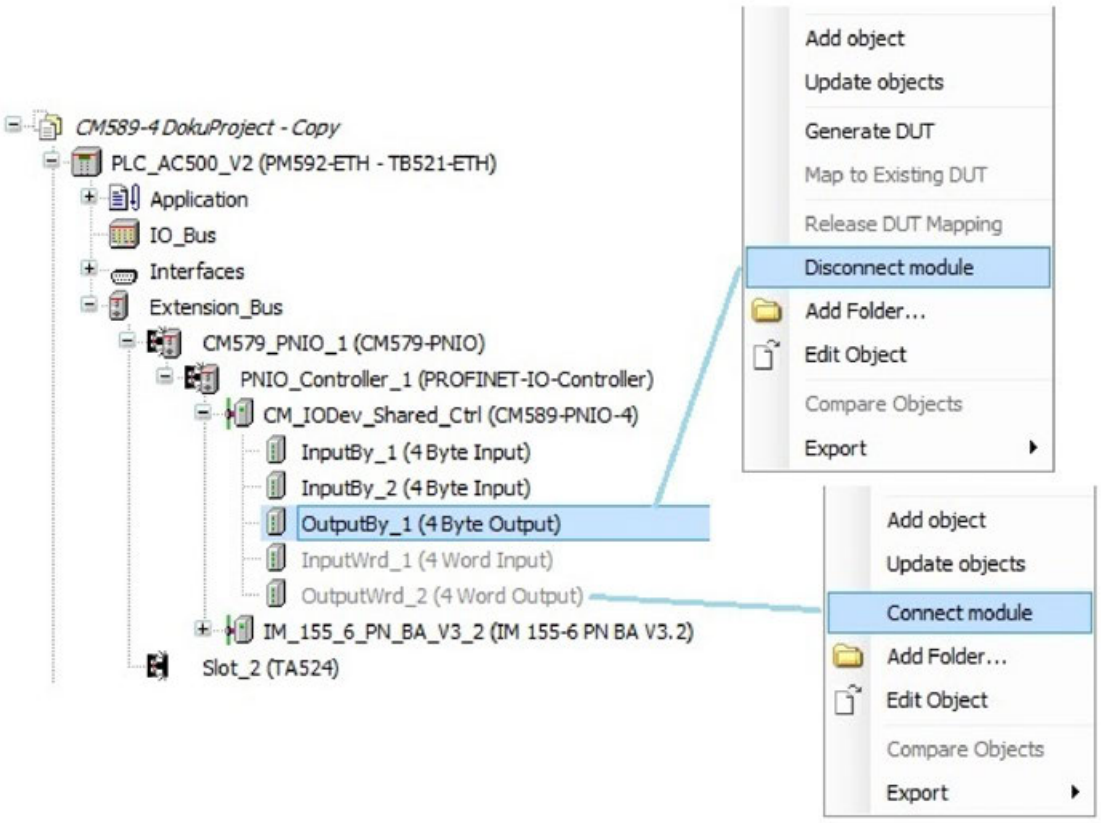
This rule also applies in case of using different Automation Builder projects or 3rd party IO controller devices.

The following workflow applies to configure an IO device to be connected by more than one IO controller as shared device:

- Add IO device at first IO controller
- Add list of IO modules as defined at IO device configuration itself
- Define set of IO modules to be connected by this IO controller
  - use function *"connect/disconnect"*
- Add same IO device the same way at further IO controllers
- Define set of IO modules to be connected for each of the further IO controllers
- Define one IO controller to be responsible to connect DAP
  - this may be a different IO controller outside this configuration
  - parameter *"Shared Device"* in Automation Builder; may be different approach at 3rd party tools

Inserting an IO module at a *"PROFINET IO device"* assigned to a *"CM579 IO controller"* defines this module as to be connected by default. If the module should not be connected by this IO controller this setting has to be adjusted. Via context menu at IO device module sub-tree Automation Builder provides the function *"disconnect module"* to define a certain module as not to be connected.

To define a not connected module as to be connected again the context menu shows the function *"connect module"*.



The module nodes that are configured to be connected will be shown in active state (black style). Modules not to be connected will be shown in in-active state (gray style).

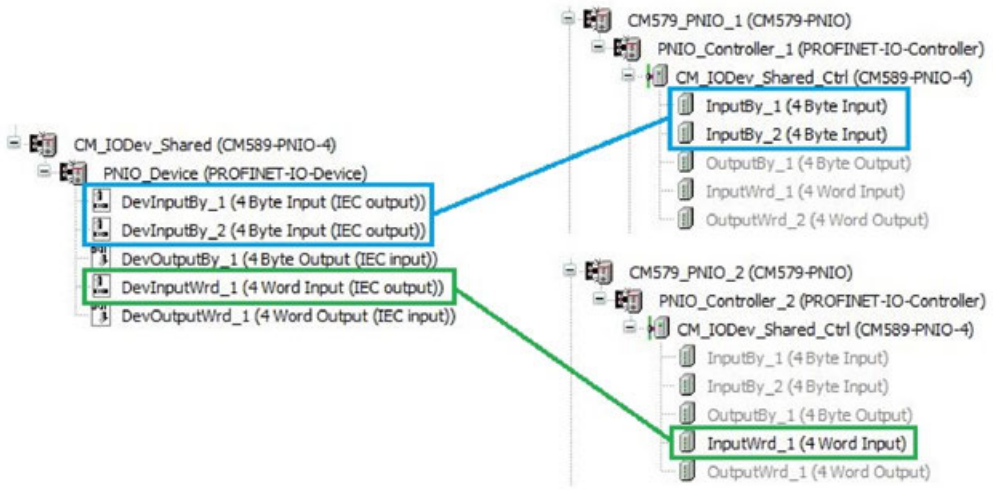
Once a PROFINET IO device is configured as child at an IO controller the IO device sub-tree (complete IO device or its module configuration) may be copied from one IO controller node to a different. The selection which modules should be connected has to be adjusted to the needs at the new position.

The following example shows a configuration of *"Shared Device"* usage by using one AC500 *"CM589-PNIO-4 IO device"* and two IO controllers represented by two ABB *"CM579-PNIO IO controller"* modules.

"CM579_PNIO_1" connects the modules *"InputBy_1"* and *"InputBy_2"* only; represented by *"CM589 PNIO_Device"* modules *"DevInputBy_1"* and *"DevInputBy_2"*.

"CM579_PNIO_2" connects module *"InputWrd_1"*; represented by *"PNIO_Device"* modules *"DevInputWrd_1"*.

The other modules are not connected here but may be connected from outside this configuration by an external IO controller.



In case of connecting the same module/submodule by more than one IO controller only the first connect will have success. All further connect attempts will fail. The IO device will reply to those attempts with a connect error. The effected IO controllers will show this error at their diagnosis interface.

See ↬ Chapter 1.7.2.7.5.1 “IO controller views” on page 6408 on how Automation Builder will show this diagnosis information.

In display *“CM579-PNIO PROFINET IO Device Views”* field *“Diagnosis state”* shows a text message describing the error. Basically this text depends on the type of device signaling the error. In case of *“CM589-PNIO-4”* the text *The IO-Device reported a ModuleDiffBlock during connection establishment* will be found.

Rules to be followed on using PROFINET “Shared Device” configurations:

- *All configurations of the IO device at IO controllers using this IO device have to look the same*
- *IP configuration (IP address, subnet mask) have to be the same*
- *Station name has to be the same*
- *Configure each IO module as to be connected at one single IO controller only*
- *Automation Builder is not able to check these details for consistency*

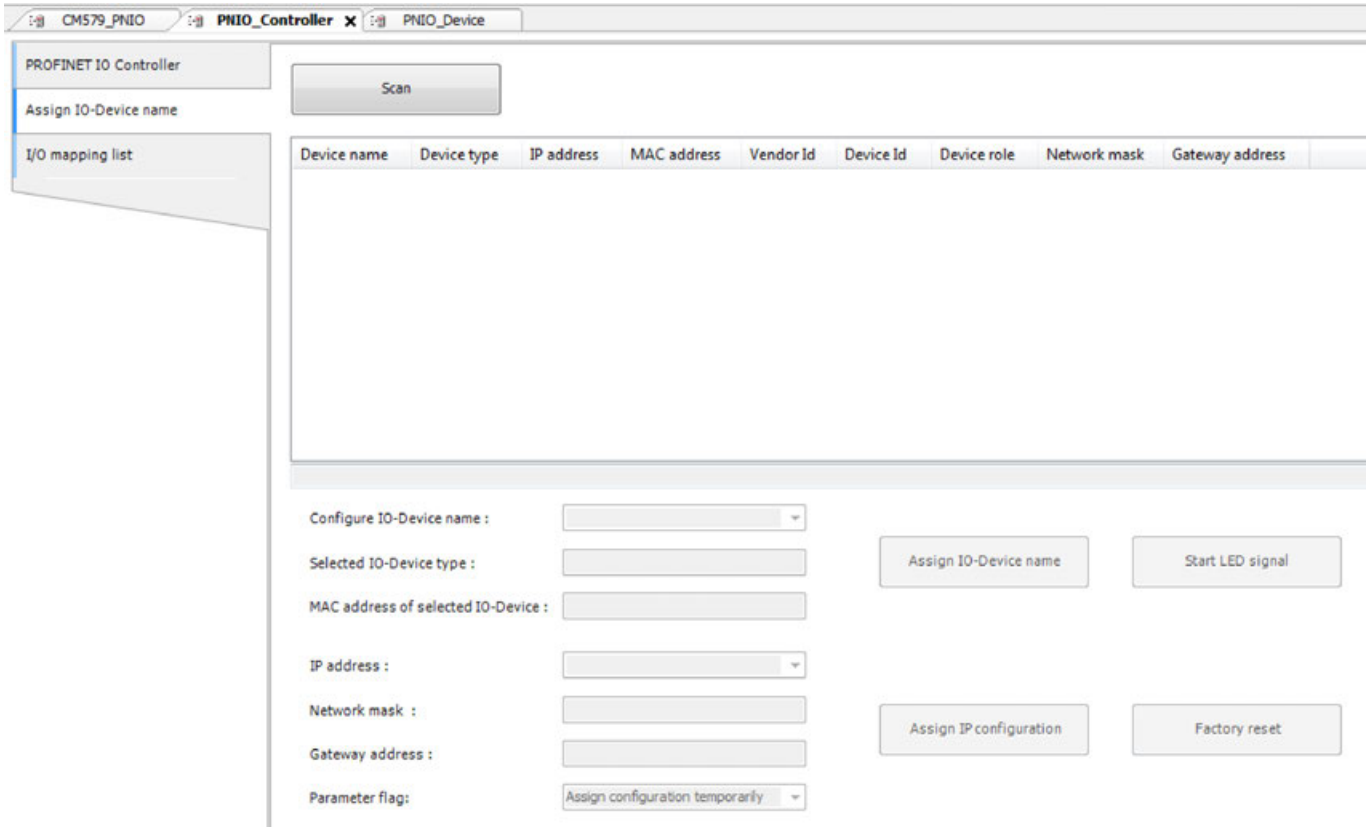
In case of using more than one single IO controller device in the same network take care to assign unique station names and IP addresses to these IO controller devices.

In case of having a configuration with overlapping module usage it cannot be foreseen which IO controller will connect to which module. The result of connecting the modules/submodules depends on the start-up sequence of the complete system. The IO controller running first will get the connection.

Name assignment of a PROFINET IO device

A *"PROFINET IO device"* is addressed with a system wide unique name.

This name is permanently stored in the device and must be assigned during the initial start-up of a PROFINET IO system.



Parameter	Description
Configure IO-Device name	Configure I/O device name
Selected IO-Device type	Selected I/O device type
MAC address of selected IO-Devices	MAC address of selected I/O devices
IP address	IP address
Network mask	Network mask
Gateway address	Gateway address
Parameter flag	Values: • Assign configuration temporarily: Store the IP configuration (IP address, network mask, gateway) temporarily. After powering off/on the configuration will be lost. • Assign configuration permanently: Store the IP configuration (IP address, network mask, gateway) permanently.

For name assignment to a PROFINET IO Device the following steps are necessary:

1. Configure the PROFINET IO controller and the PROFINET IO devices of your system as described in the previous sections of this chapter.
2. Configure a name for each *"PROFINET IO device"* in the PROFINET IO device parameter editor.
3. Configure a communication connection to AC500 in communication gateway.

4. Open the PROFINET IO controller parameter editor by double-clicking on the corresponding device. Open *“Assign IO-Device”* name.
5. Use the button *“Connect to PLC”* to open an online connection.
6. Perform PROFINET IO bus scan. All PROFINET IO devices, which are connected to the PROFINET IO controller, will appear in the device list after a while.
7. Select the desired PROFINET IO devices in the device list.
8. Select the desired name from the pull-down-box Configure station name. This box contains all names, which were configured in the PROFINET IO device parameter editor.
9. Press the button Assign station name to assign the desired name to the selected device.
10. Perform the steps 7 to 9 for all other devices, which have not yet a device name.

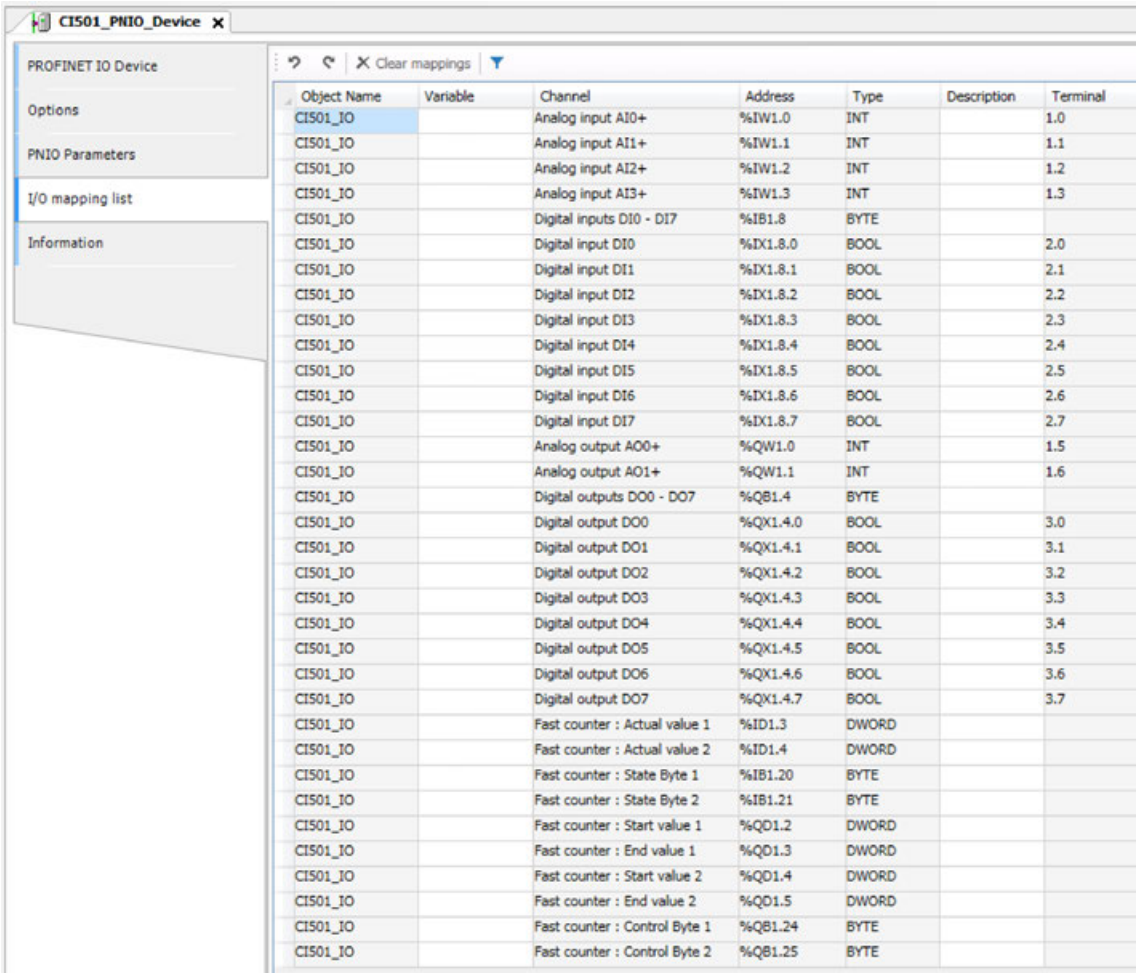
In addition to the name assignment, this dialog has the following functions:

- Start LED Signal:
  The button Start LED Signal can be used to find the device in the system. If it’s pressed the selected device will start blinking with it's LED.
- Factory reset:
  With this function a PROFINET IO device can be reset to factory settings.
- Assign IP configuration:
  The assignment of an ip address, network mask and gateway address can be performed in the same way as the name assignment. These values will be stored permanently in a device.

Mapping of the PROFINET IO devices

Doube-click on the *“PROFINET IO device”* or module in the configuration tree to open the configuration dialog.

Then select the mapping tab to show the current I/O mapping.



Object Name	Variable	Channel	Address	Type	Description	Terminal
CI501_IO		Analog input AI0+	%IW1.0	INT		1.0
CI501_IO		Analog input AI1+	%IW1.1	INT		1.1
CI501_IO		Analog input AI2+	%IW1.2	INT		1.2
CI501_IO		Analog input AI3+	%IW1.3	INT		1.3
CI501_IO		Digital inputs DI0 - DI7	%IB1.8	BYTE		
CI501_IO		Digital input DI0	%IX1.8.0	BOOL		2.0
CI501_IO		Digital input DI1	%IX1.8.1	BOOL		2.1
CI501_IO		Digital input DI2	%IX1.8.2	BOOL		2.2
CI501_IO		Digital input DI3	%IX1.8.3	BOOL		2.3
CI501_IO		Digital input DI4	%IX1.8.4	BOOL		2.4
CI501_IO		Digital input DI5	%IX1.8.5	BOOL		2.5
CI501_IO		Digital input DI6	%IX1.8.6	BOOL		2.6
CI501_IO		Digital input DI7	%IX1.8.7	BOOL		2.7
CI501_IO		Analog output AO0+	%QW1.0	INT		1.5
CI501_IO		Analog output AO1+	%QW1.1	INT		1.6
CI501_IO		Digital outputs DO0 - DO7	%QB1.4	BYTE		
CI501_IO		Digital output DO0	%QX1.4.0	BOOL		3.0
CI501_IO		Digital output DO1	%QX1.4.1	BOOL		3.1
CI501_IO		Digital output DO2	%QX1.4.2	BOOL		3.2
CI501_IO		Digital output DO3	%QX1.4.3	BOOL		3.3
CI501_IO		Digital output DO4	%QX1.4.4	BOOL		3.4
CI501_IO		Digital output DO5	%QX1.4.5	BOOL		3.5
CI501_IO		Digital output DO6	%QX1.4.6	BOOL		3.6
CI501_IO		Digital output DO7	%QX1.4.7	BOOL		3.7
CI501_IO		Fast counter : Actual value 1	%ID1.3	DWORD		
CI501_IO		Fast counter : Actual value 2	%ID1.4	DWORD		
CI501_IO		Fast counter : State Byte 1	%IB1.20	BYTE		
CI501_IO		Fast counter : State Byte 2	%IB1.21	BYTE		
CI501_IO		Fast counter : Start value 1	%QD1.2	DWORD		
CI501_IO		Fast counter : End value 1	%QD1.3	DWORD		
CI501_IO		Fast counter : Start value 2	%QD1.4	DWORD		
CI501_IO		Fast counter : End value 2	%QD1.5	DWORD		
CI501_IO		Fast counter : Control Byte 1	%QB1.24	BYTE		
CI501_IO		Fast counter : Control Byte 2	%QB1.25	BYTE		

See chapter Symbolic Names for Variables, Inputs and Outputs ↳ *Chapter 1.6.5.2.9.6 “Symbolic names for variables, inputs and outputs” on page 6060* for further details on mapping of inputs and outputs.

CM589-PNIO / CM589-PNIO-4 - PROFINET IO device communication module

CM589-PNIO / CM589-PNIO-4 - PROFINET IO device communication module

The configuration of the CM589-PNIO PROFINET IO device module has to be done in the following steps:

- Parameterization of the AC500 communication module interface ↳ *“Parameterization - CM interface” on page 5952*
- Parameterization of the PROFINET IO device protocol stack ↳ *“Parameterization - PROFINET IO stack” on page 5952*
- Configuring PROFINET IO device module structure ↳ *“Configuring PROFINET IO structure” on page 5952*
- Parameterization of the PROFINET IO device modules ↳ *“Parameterization - PROFINET IO device modules” on page 5953*
- Mapping of the I/Os ↳ *“Mapping of the I/Os” on page 5954*
- Configuration specific to CM589-PNIO-4 ↳ *“Configuration CM589-PNIO-4” on page 5955*

Configuration procedure for CM589-PNIO and CM589-PNIO-4 are basically the same. Thus this description refers to CM589-PNIO only. Differences between CM589-PNIO and CM589-PNIO-4 will be highlighted explicitly if necessary!

Parameterization - CM interface

For connecting a PLC as *"PROFINET IO device"*, plug *"CM589-PNIO / CM589-PNIO-4"* at the *"Extension_Bus"* node.

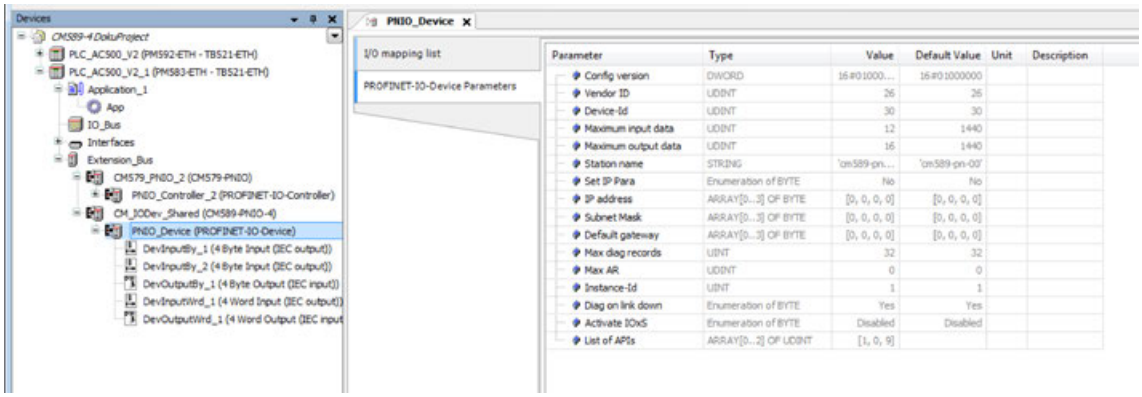
Double-click on *"CM589-PNIO"* to open the CM589-PNIO configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even in case of configuration error.
Min update time	10	0...20000	Update time of inputs and outputs in [ms].

Parameterization - PROFINET IO stack

"PROFINET IO device" protocol does not need user configuration. All needed parameters are set automatically by Automation Builder. Double-click on *"PROFINET IO device"* in tree view will show the parameter set in tab *"PROFINET IO device parameters"*. The parameters are displayed just for information and in read-only mode.

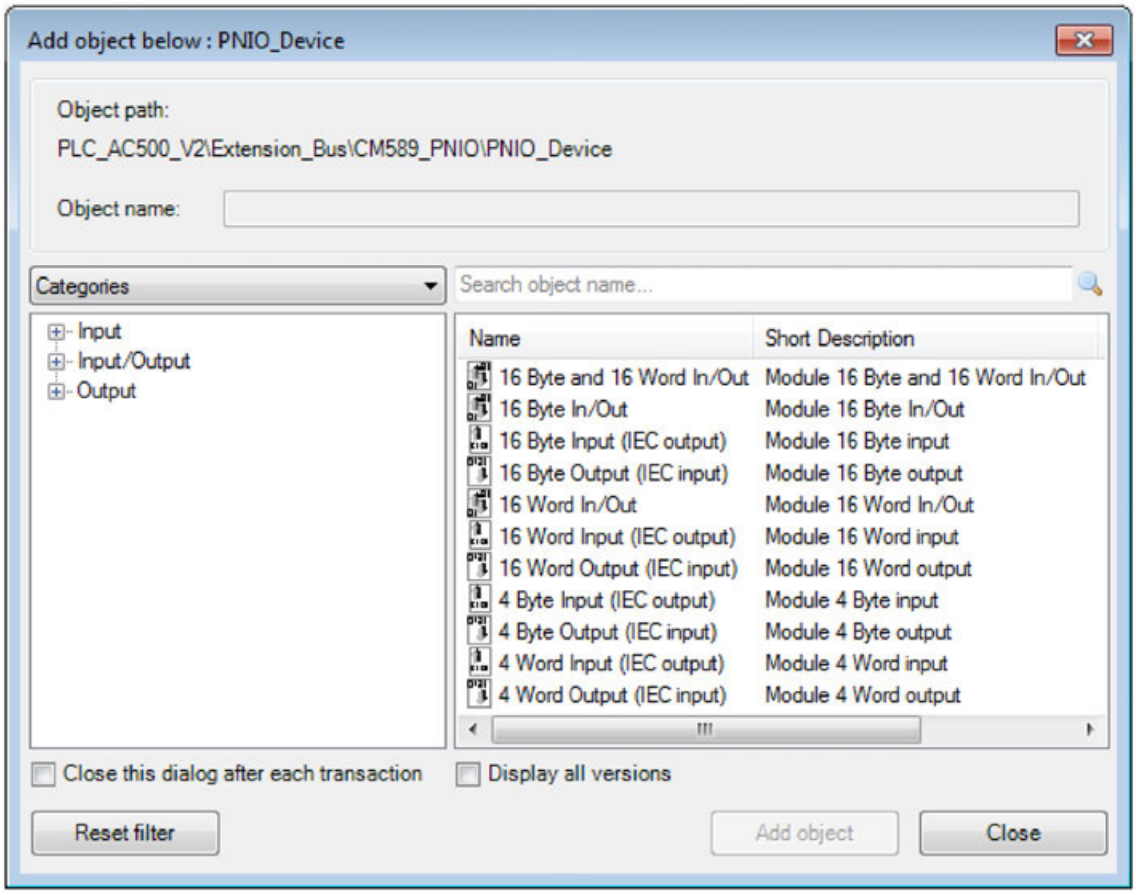


- *Station name: the default name is displayed. The real name used on acting at the field bus is combined out of this default name and the used setting of the rotary switches at the CM589 module ("cm589-pnio-00", "00" will be replaced by rotary switch value) or the name set via PROFINET set name service.*
- *Parameter "IP address", "Subnet Mask", "Default Gateway": the default values are displayed here. These values are not used as communication settings. "PROFINET IO controller" supplies the IO devices with IP settings on communication establishing.*

Configuring PROFINET IO structure

CM589-PNIO provides I/O data as modules with different data types and directions. Create an application specific I/O structure by compiling an appropriate combination of modules.

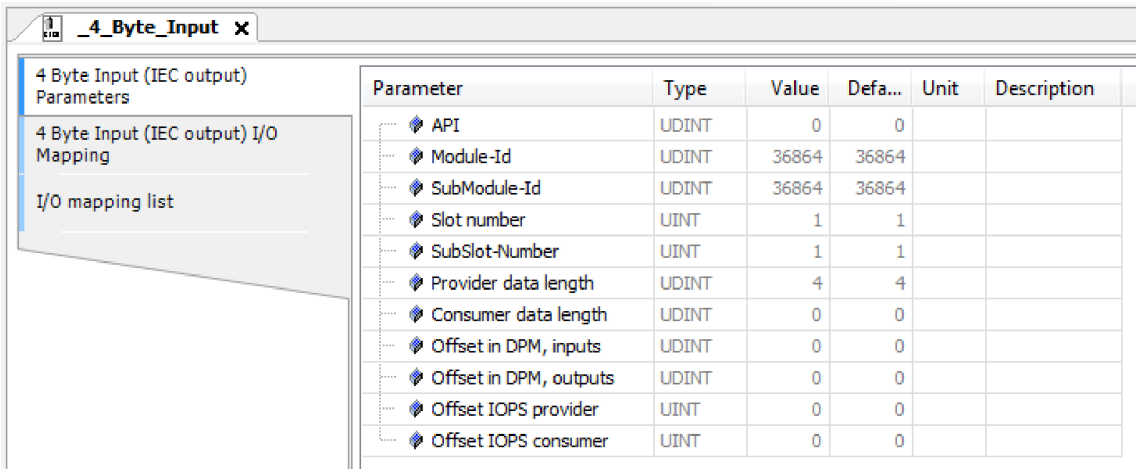
To assign I/O modules to *"PROFINET-IO-device"* node open *"Add Object"* dialog.



The maximum image size available for I/O data depends on CM589-PNIO modules FW revision (for details see *Chapter 1.6.2.4.9.2 “CM589-PNIO(-4) - PROFINET IO RT with 4 devices” on page 4089*). If FW revision run by connected CM589-PNIO module does not support the number of I/O data configured, download configuration will fail. See *“Calculating size of I/O data” on page 5954* how to calculate number of I/O data occupied by certain configuration.

Parameterization - PROFINET IO device modules

PROFINET-IO device modules do not need user configuration. All needed parameters are set automatically by Automation Builder. Double-click on a module node shows the parameter set just for information. This parameter set is identical for all module types and is displayed in read-only mode.



- *API: shows the API which is used by the CM589-PNIO modules. As API 0 is supported only CM589-PNIO modules do not provide configuration capabilities for this parameter*
- *Slot number, Sub-Slot-Number, Offset in DPM, inputs/outputs: will be set to default values on inserting a module. On creating configuration data Automation Builder calculates real values and overwrites the defaults*
- *Offset IOPS provider/consumer: not used and set to 0 values*

Calculating size of I/O data

PROFINET defines IO data and status information to be exchanged between IO controller and IO device. The status information is called *"Provider Status"* and *"Consumer Status"*. Both (IO data and status information) have to be considered on calculating allocated memory in input and output image.

- The number of status bytes depends on the type of module used.
- The different types of modules input, output and in/output have to be considered different.
- Some status bytes are reserved for predefined submodules have to be considered additionally.

A configured IO module allocates memory space at the corresponding IO image for data and status bytes. Additionally memory is allocated at the opposite directions IO image to store further status bytes. E.g. an input module allocates memory at the input image but additionally it allocates one byte for status at the output image. Summarized size of input and output data and status has to fit to the corresponding image.

See following table for an overview of IO module types and corresponding status bytes:

Module Type	**Input Data**			**Output data**		
	Inputs	**Provider Status Inputs**	**Consumer Status Outputs**	**Outputs**	**Provider Status Outputs**	**Consumer Status Inputs**
Reserved	0 Input Bytes	4 Bytes	0 Bytes	0 Bytes	0 Bytes	4 Bytes
Input Module (e.g. 4 Byte Input)	n Input Bytes	1 Byte	0 Bytes	0 Bytes	0 Bytes	1 Byte
Output Module	0 Input Bytes	0 Bytes	1 Byte	n Output Bytes	1 Byte	0 Bytes
Input/ Output Module	n Input Bytes	1 Byte	1 Byte	n Output Bytes	1 Byte	1 Byte

Following expressions calculate allocated sizes of input and output data:

Size Input =	**Input + Status + 4 bytes (reserved status)**
Size Output =	**Output + Status + 4 bytes (reserved status)**

- Input = summarized number input bytes all modules
- Output = summarized number output bytes all modules
- Status = count input modules + count output modules + 2 * count input/output modules

Mapping of the I/Os

Double-click on the desired *"PROFINET-IO-device"* module object in the device tree to show current I/O mappings connected to this module.

See chapter Symbolic Names for Variables, Inputs and Outputs *Chapter 1.6.5.2.9.6 "Symbolic names for variables, inputs and outputs" on page 6060* for further details on mapping inputs and outputs.

Configuration CM589-PNIO-4

CM589-PNIO-4 adds PROFINET *"Shared Device"* feature to already described CM589-PNIO device functionality. Thus CM589-PNIO-4 is able to communicate to 4 different PROFINET IO controllers in parallel.

Shared device usage of a CM589-PNIO-4 does not need to be considered on doing the configuration of the CM589-PNIO-4 communication module itself. It has to be considered only on doing the configuration of CM589-PNIO-4 as child to the CM579-PNIO IO controller. See *Chapter 1.6.5.2.6.7.1.5 "Configuration of PROFINET shared device functionality" on page 5945*.

CM579-ETHCAT - EtherCAT master communication module

AC500 CM579-ETHCAT - Communication module EtherCAT master

The configuration of the EtherCAT field bus has to be done in the following steps:

- Parameterization of the AC500 Communication Module Interface *Chapter 1.6.5.2.6.8.2 "Parameterization of the CM579-ETHCAT communication module interface" on page 5955*
- Configuration of the EtherCAT master *Chapter 1.6.5.2.6.8.3 "Configuration of the EtherCAT master" on page 5956*
- Configuration of the EtherCAT slaves *Chapter 1.6.5.2.6.8.4 "Configuration of the EtherCAT slaves" on page 5959*
- Mapping of the EtherCAT slave I/Os *Chapter 1.6.5.2.6.8.5 "Mapping of the EtherCAT slave I/Os" on page 5966*
- EtherCAT Sync - Synchronization of a PLC Task with the IO Image *Chapter 1.6.5.2.6.8.7 "EtherCAT Sync - Synchronization of a PLC task with the IO image" on page 5966*

Parameterization of the CM579-ETHCAT communication module interface

To append a Communication Module, add the Communication Module to the *"Extension_Bus"* node.

- Right-click the desired slot and select *"Add object"*.
- Select the Communication Module from the list and click *[Replace object]*.

- Double-click on *"CM579_ECAT (CM579-ECAT)"* to open the CM579-ECAT configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started independent of a faulty configuration of the EtherCAT Communication Module.
Max wait run	3000	3000	Maximum wait time for the Master to build up the communication relation to the slaves. A restart to build up the communication is initiated as long as BootUpTime has not run out.

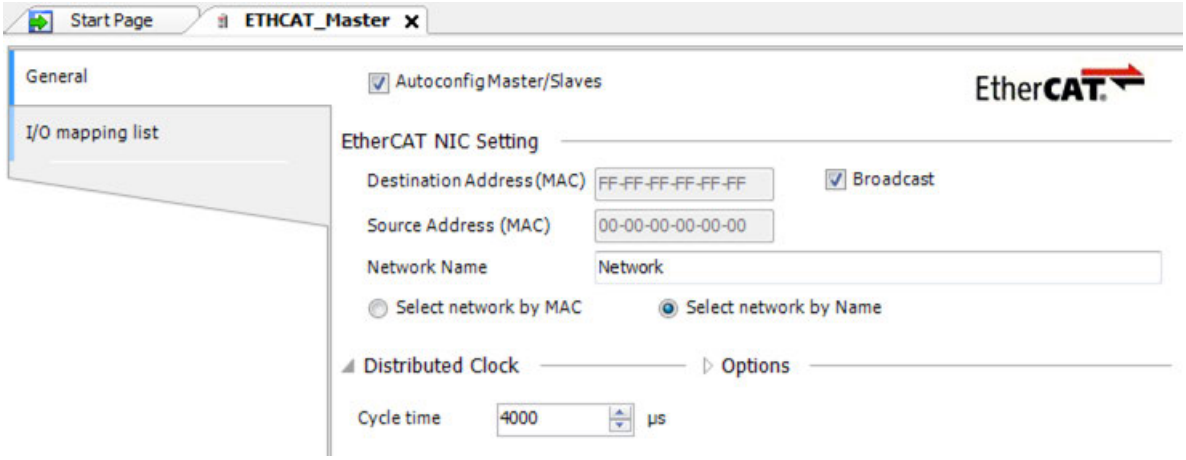
Parameter	Default value	Value	Description
Min update time	10	0...20000	Priority of the data exchange between CPU and Communication Module. This parameter should never be set to values under 10, otherwise important sequences in the CPU might be influenced.
Broken slave behavior	Leave all broken slaves down	Leave all broken slaves down	Broken slaves will not be served.
		Leave addressless slaves down	Only slaves without address will be left down.
		Leave no slaves down	Broken slaves will be ignored.
Distributed clocks	Inaktiv	Inactive	Distributed clocks are inactive.
		Active	Distributed clocks are active.
BootUpTime	10000	10000	Maximum wait time for the slaves to boot completely. This absolute time value must be a multiple of Max wait run. It defines the time in which Max wait run is restarted to wait for the slaves to boot up completely. The multiplication ratio between Max wait run and BootUpTime can be interpreted as number of trials to boot up the slaves.

Configuration of the EtherCAT master

To append a Communication Module, add the Communication Module to the *“Extension_Bus”* node.

- Right-click the desired slot and select *“Add object”*.
- Select the Communication Module from the list and click *[Replace object]*.

- Double-click the added object to open the Master configuration in the editor window.



The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1 *Further information on page 5956*)
Autoconfig Master/Slaves Remark 2 *Further information on page 5956*:	Enabled	Disabled	The automatic configuration is disabled.	Autoconfig
		Enabled	The configuration parameters are set automatically based on the given device description and network topology. Normally the parameters do not have to be modified for standard applications.	
Ethercat NIC Setting				
Destination Address (MAC)	FF-FF-FF-FF-FF-FF	Valid MAC address	Parameter Destination Address (MAC) determines the MAC address of the participant of the EtherCAT network which should receive the telegrams. If option Broadcast is activated, no specific address must be entered.	DestAddress1 and DestAddress 2
Source Address (MAC)	00-00-00-00-00-00	Valid MAC address	Parameter Source Address (MAC) determines the MAC address of the system.	SrcAddress1 and SrcAddress2
Network Name	empty	String	Parameter Network Name determines the network name of the system.	NetworkName
		Select network by MAC	If this option is selected, the network is selected by the MAC address determined by parameter Source Address (MAC).	SelectNetworkByName = FALSE
		Select network by Name	If this option is selected, the network is selected by the network name determined by parameter Network Name.	SelectNetworkByName = TRUE
Broadcast	Enabled	Disabled	The destination address is not the broadcast address. The parameter Destination Address (MAC) must be edited.	-
		Enabled	The destination address is set to broadcast address FF-FF-FF-FF-FF-FF. The parameter Destination Address (MAC) can not be edited.	
Distributed Clock				

Parameter	Default	Value	Description	Parameter (Remark 1 *Further information on page 5956*)
Cycle time	4000	0...10000	Period of time in [µs] after which a new data telegram is to be sent on the bus. If the Distributed Clock functionality is activated, the master cycle time will be transferred to the slave clocks. Thus an accurate synchronization of data exchange can be reached, which is particularly important in cases where spatially distributed processes require simultaneous actions (e.g. in applications where several servo axes carry out coordinated movements simultaneously). So a very precise network-wide timebase with a jitter of significantly less then 1 microsecond can be achieved.	MasterCycleTime
Options				
Use LRW instead of LWR/LRD	Disabled	Disabled	Slave-to-slave communication is not allowed.	MasterUseLRW
		Enabled	Slave-to-slave communication is allowed.	
Enable messages per task	Disabled	Disabled	This parameter is currently not supported.	EnableTaskMessage
		Enabled		
Auto restart slaves	Disabled	Disabled	This parameter is currently not supported.	
		Enabled		
Master Setting (only available if AutoconfigMaster/Slaves option is disabled)				
Image In Address	16#10880	16#00000000... 16#FFFFFFFF	Parameter Image In Address determines the first logical address of the first slave for input data.	ImageInAddress and ImageInLength
Image Out Address	16#10000	16#00000000... 16#FFFFFFFF	Parameter Image Out Address determines the first logical address of the first slave for output data.	ImageOutAddress and ImageOutLength

Remark 1: Tab CM579-Master Configuration

The parameters in this column are shown in tab EtherCAT Configuration which is only visible if the parameter Show generic device configuration views is activated (open the Options dialog window with menu item Tools -> Options, parameter is located under section Device editor):

Master | EtherCAT Configuration | Information

Parameter	Type	Value	Default Value	Unit	Description
Autoconfig	DWORD	0	1		Automatic configuration
MasterCycleTime	DWORD	10000	4000		Master Cycle Time
MasterUseLRW	BOOL	FALSE	FALSE		Master uses LRW command
SlaveAutorestart	BOOL	FALSE	FALSE		Slave restarts automatically
SlaveCheckMode	USINT	0	0		Mode for vendor product check
NetworkName	STRING(100)	''	''		Name of the network card
SelectNetworkByName	BOOL	True	FALSE		Select network by name
EnableTaskMessage	BOOL	FALSE	FALSE		Enable transmission per task
NumberOfOutputSlaves	DWORD	0	0		Number of Slaves with an Output
NumberOfInputSlaves	DWORD	0	0		Number of Slaves with an Input
SrcAddress1	DWORD	0	0		Two zero Bytes and the First Two Bytes of the Source Address
SrcAddress2	DWORD	0	0		Lower 4 Bytes of the Source Address
DestAddress1	DWORD	65535	65535		Two zero Bytes and the First Two Bytes of the Destination Address
DestAddress2	DWORD	4294967295	4294967295		Lower 4 Bytes of the Destination Address
ImageOutAddress	DWORD	1048575	65536		Address of the Output
ImageOutLength	DWORD	0	0		Length of the Output
ImageInAddress	DWORD	4294967295	67584		Address of the Input
ImageInLength	DWORD	0	0		Length of the Input

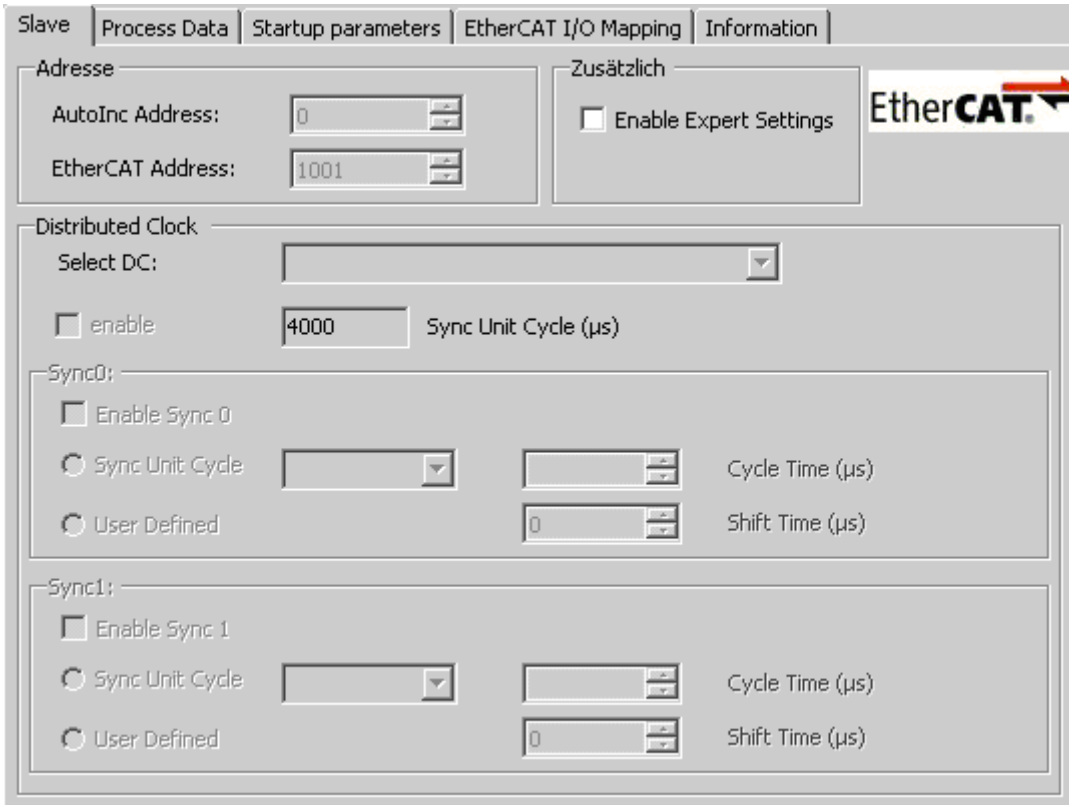
The table shows all configuration parameters.

Remark 2: Autoconfig

The autoconfig mode is activated by default and usually sufficient and strongly recommended for standard applications. If the option is deactivated, all configuration settings for the Master and the Slave(s) will have to be done manually and expert knowledge is required. For the configuration of a slave-to-slave communication the autoconfig option has to be deactivated.

Configuration of the EtherCAT slaves

Double-click on a EtherCAT slave module in the device tree:



All parameters are set to the device-specific standard values by default. To edit the parameters enable the *“Enable Expert Settings”* check box. This option also displays detailed parameters.

The following parameters are available:

Parameter	Default	Value	Description	Parameter (Remark 1 ↯ *Further information on page 5959*)
Address				
AutoInc Address	0	0...65535	Autoincremental address (16-bit), defined by the position of the slave in the network. This address is only used during startup, when the master is assigning the EtherCAT addresses to the slaves. When for this purpose the first telegram runs through the slaves, each run-through slave increases its AutoInc Address by 1. The slave with address 0 finally will receive the data. Possible input for example "-8".	General -> AutoIncr Address of the Slave
EtherCAT Address	0	0...65535	Final address of the slave, assigned by the master during startup. This address is independent from the position in the network.	General -> Physical Address of the Slave
Additional				
Enable Expert Settings	Disabled	Disabled	Expert settings are not editable.	-
		Enabled	Expert settings are editable.	
Distributed Clock				
Select DC	-	-	The drop-down menu provides all settings for distributed clocks provided by the device description file. Currently this parameter is deactivated.	-
Enable	Disabled	Disabled	The distributed clock is not enabled. The parameter setting for Sync0 and Sync1 are disabled and unused.	Distributed_Clock -> DC enable
		Enabled	The distributed clock is enabled. The parameter settings for Sync0 and Sync1 are enabled and can be edited.	

Parameter	Default	Value	Description	Parameter (Remark 1 ↳ *Further information on page 5959*)
Sync Unit Cycle (µs)	Device-specific	Device-specific	If the Distributed Clock functionality is enabled, the data exchange cycle time, displayed in the field Sync Unit Cycle (µs) will be determined by the master cycle time. Thus the master clock can synchronize the data exchange within the network. The master cycle time for the distributed clock can be determined by editing parameter Cycletime of section Distributed Clock in the Master editor window.	Parameter of the Master
Sync 0				
Enable Sync 0	Disabled	Disabled Enabled	Sync 0 is currently not supported.	Distributed_Clock -> DC sync0 enable
Sync Unit Cycle	x 1	/ 16 .. x 16		Distributed_Clock -> DC sync0 factor
Cycle time (µs)	-	0..2^{32}-1		Distributed_Clock -> DC sync0 cycletime
User defined	-	-		-
Shift Time (µs)	0	0..2^{32}-1		Distributed_Clock -> DC sync0 shift time
Sync 1				
Enable Sync 1	Disabled	Disabled Enabled	Sync 1 is currently not supported.	Distributed_Clock -> DC sync1 enable
Sync Unit Cycle	-	-		Distributed_Clock -> DC sync1 factor
Cycle time (µs)	-	0..2^{32}-1		Distributed_Clock -> DC sync1 cycletime
User defined	-	-		-
Shift Time (µs)				Distributed_Clock -> DC sync1 shift time

Parameter	Default	Value	Description	Parameter (Remark 1 ↳ *Further information on page 5959*)
Startup checking				
Check Vendor ID	Enabled	Disabled	The vendor ID is not checked against the current configuration settings.	General -> CheckVendorID
		Enabled	The vendor ID is checked against the current configuration settings. If a mismatch is detected the bus will be stopped and no further actions will be executed.	
Check Product ID	Enabled	Disabled	The product ID is not checked against the current configuration settings.	General -> CheckProductID
		Enabled	The product ID is checked against the current configuration settings. If a mismatch is detected the bus will be stopped and no further actions will be executed.	
Timeouts				
SDO Access	1000	0...10000	Timeout for sending SDO list at system startup in [ms].	General -> Timeout SDO Access
I -> P	1000	0...10000	Timeout for switch from mode Init to mode Preoperational in [ms].	General -> Timeout Init
P -> S / S -> O	10000	0...10000	Timeout for switch from mode Preoperational to Safe Operational resp. from Safe Operational to Operational in [ms].	General -> Timeout Preop to Safeop
DC cyclic unit control: assign to local µC				
Cyclic Unit	Disabled	Disabled	DC cyclic unit control is currently not supported.	-
		Enabled		
Latch Unit 0	Disabled	Disabled		
		Enabled		
Latch Unit 1	Disabled	Disabled		
		Enabled		
Station alias				
Enable (read-only)	Disabled	Disabled	Station alias is currently not supported.	-
Value (read-only)	0	0		-

Remark 1:

The parameters in this column are shown in tab EtherCAT Configuration which is only visible if parameter Show generic device configuration views is activated (open the Options dialog window under *"Tools ➔ Options ➔ Device editor"*).

Slave | FMMU/Sync | Expert Process Data | Process Data | Startup parameters | EtherCAT Configuration | EtherCAT I/O Mappi

Parameter	Type	Value	Default Value	Unit	Description
General					
Number of O...	DWORD	10	10		Number of Outputs
Number of In...	DWORD	10	10		Number of Inputs
Number of In...	DWORD	10	10		Number of Infos about Outputs
Physical Addr...	DWORD	3840	3840		Physical Address of 1. Output
Length of the...	DWORD	1	1		Length of the first Output
StartBit of the...	DWORD	0	0		StartBit of the first Output
EndBit of the...	DWORD	7	7		EndBit of the first Output
Number of In...	DWORD	10	10		Number of Infos about Inputs
Physical Addr...	DWORD	3840	3840		Physical Address of 1. Input
Length of the...	DWORD	1	1		Length of the first Input
StartBit of the...	DWORD	0	0		StartBit of the first Input
EndBit of the...	DWORD	7	7		EndBit of the first Input
CheckVendorID	BOOL	TRUE	TRUE		CheckProductID
CheckVendorID	BOOL	TRUE	TRUE		CheckProductID
Timeout SDO...	DWORD	100000			
Timeout Init t...	DWORD	100000			
Timeout Preo...	DWORD	100000			
PDOAssign	BOOL	FALSE	FALSE		PDOAssign
PDOConfig	BOOL	FALSE	FALSE		PDOConfig
Number of Id...	DWORD	4	4		Number of Identity Parameters
Vendor Id of t...	DWORD	570	570		Vendor Id of the Slave
Product Code...	DWORD	2020894049	2020894049		Product Code of the Slave
Revision Num...	DWORD	65539	65539		Revision Number of the Slave
Serialnumber ...	DWORD	0	0		Serialnumber of the Slave
Physical Addr...	DWORD	65535	0		Physical Address of the Slave
AutoIncr Addr...	DWORD	4294901761	0		AutoIncr Address of the Slave
SyncManager					
FMMUs					
RxPDO					
TxPDO					
Distributed_Clocks					
SDOs					

Export process data

Tab Expert Process Data is only shown if the parameter Show generic device configuration views is activated (open the Options dialog window under *"Tools ➔ Options ➔ Device editor"*):

Slave | Process Data | Startup parameters | EtherCAT I/O Mapping | Information

Select the outputs

Name	Type	Index
16#1600 Outputs		
CAM_release	DINT	16#2000:01
CAM_mux	BYTE	16#2000:02
reserve1	BYTE	16#2000:03
CAM_On_Position	WORD	16#2000:04
CAM_Off_Position	WORD	16#2000:05
CAM_Comp_Time	INT	16#2000:06
DO	BYTE	16#2000:07
reserve4	BYTE	16#2000:08
AO 0	INT	16#2000:09
AO 1	INT	16#2000:10

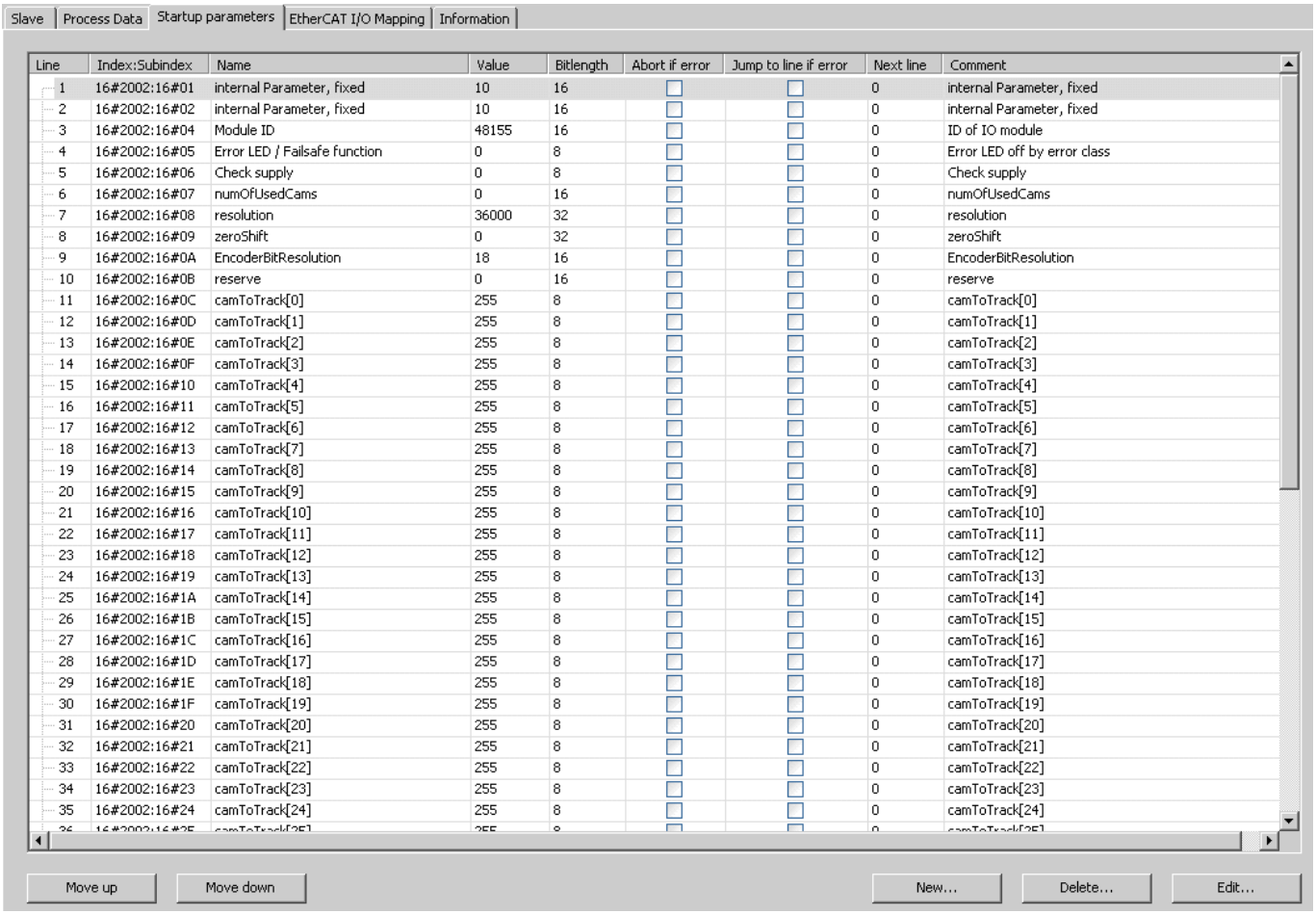
Select the inputs

Name	Type	Index
16#1A00 Inputs		
CAM_state	DWORD	16#2001:01
CAM_track	WORD	16#2001:02
QUIT_mux	BYTE	16#2001:03
reserve2	BYTE	16#2001:04
DI	BYTE	16#2001:05
reserve3	BYTE	16#2001:06
AI 0	INT	16#2001:07
AI 1	INT	16#2001:08
AI 2	INT	16#2001:09
AI 3	INT	16#2001:10

The Expert Process Data dialog allows the set up of the PDO assignment and the PDO configuration.

Startup parameters

Double-click on the EtherCAT slave module in the device tree and open the *"Startup parameters"* tab:



Line	Index:Subindex	Name	Value	Bitlength	Abort if error	Jump to line if error	Next line	Comment
1	16#2002:16#01	internal Parameter, fixed	10	16	☐	☐	0	internal Parameter, fixed
2	16#2002:16#02	internal Parameter, fixed	10	16	☐	☐	0	internal Parameter, fixed
3	16#2002:16#04	Module ID	48155	16	☐	☐	0	ID of IO module
4	16#2002:16#05	Error LED / Failsafe function	0	8	☐	☐	0	Error LED off by error class
5	16#2002:16#06	Check supply	0	8	☐	☐	0	Check supply
6	16#2002:16#07	numOfUsedCams	0	16	☐	☐	0	numOfUsedCams
7	16#2002:16#08	resolution	36000	32	☐	☐	0	resolution
8	16#2002:16#09	zeroShift	0	32	☐	☐	0	zeroShift
9	16#2002:16#0A	EncoderBitResolution	18	16	☐	☐	0	EncoderBitResolution
10	16#2002:16#0B	reserve	0	16	☐	☐	0	reserve
11	16#2002:16#0C	camToTrack[0]	255	8	☐	☐	0	camToTrack[0]
12	16#2002:16#0D	camToTrack[1]	255	8	☐	☐	0	camToTrack[1]
13	16#2002:16#0E	camToTrack[2]	255	8	☐	☐	0	camToTrack[2]
14	16#2002:16#0F	camToTrack[3]	255	8	☐	☐	0	camToTrack[3]
15	16#2002:16#10	camToTrack[4]	255	8	☐	☐	0	camToTrack[4]
16	16#2002:16#11	camToTrack[5]	255	8	☐	☐	0	camToTrack[5]
17	16#2002:16#12	camToTrack[6]	255	8	☐	☐	0	camToTrack[6]
18	16#2002:16#13	camToTrack[7]	255	8	☐	☐	0	camToTrack[7]
19	16#2002:16#14	camToTrack[8]	255	8	☐	☐	0	camToTrack[8]
20	16#2002:16#15	camToTrack[9]	255	8	☐	☐	0	camToTrack[9]
21	16#2002:16#16	camToTrack[10]	255	8	☐	☐	0	camToTrack[10]
22	16#2002:16#17	camToTrack[11]	255	8	☐	☐	0	camToTrack[11]
23	16#2002:16#18	camToTrack[12]	255	8	☐	☐	0	camToTrack[12]
24	16#2002:16#19	camToTrack[13]	255	8	☐	☐	0	camToTrack[13]
25	16#2002:16#1A	camToTrack[14]	255	8	☐	☐	0	camToTrack[14]
26	16#2002:16#1B	camToTrack[15]	255	8	☐	☐	0	camToTrack[15]
27	16#2002:16#1C	camToTrack[16]	255	8	☐	☐	0	camToTrack[16]
28	16#2002:16#1D	camToTrack[17]	255	8	☐	☐	0	camToTrack[17]
29	16#2002:16#1E	camToTrack[18]	255	8	☐	☐	0	camToTrack[18]
30	16#2002:16#1F	camToTrack[19]	255	8	☐	☐	0	camToTrack[19]
31	16#2002:16#20	camToTrack[20]	255	8	☐	☐	0	camToTrack[20]
32	16#2002:16#21	camToTrack[21]	255	8	☐	☐	0	camToTrack[21]
33	16#2002:16#22	camToTrack[22]	255	8	☐	☐	0	camToTrack[22]
34	16#2002:16#23	camToTrack[23]	255	8	☐	☐	0	camToTrack[23]
35	16#2002:16#24	camToTrack[24]	255	8	☐	☐	0	camToTrack[24]

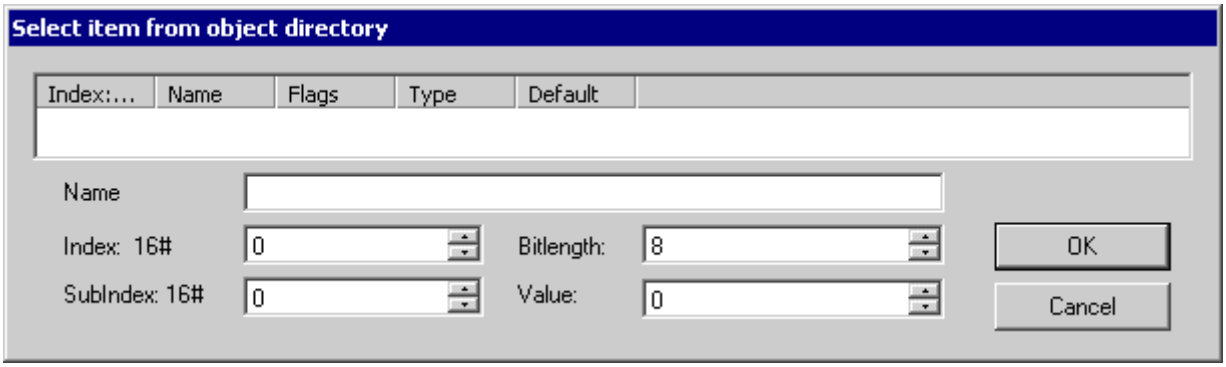
The following parameters are available:

Parameter	Default value	Value	Description
Startup parameters table columns			
Line	Device-specific	Device-specific	The line number of the parameter.
Index:Subindex	Device-specific	Device-specific	The index and the subindex of the parameter.
Name	Device-specific	Device-specific	The name of the parameter.
Value	Device-specific	Device-specific	The configured value of the parameter.
Bitlength	Device-specific	Device-specific	The length of the parameter in [bit].
Abort if error	Device-specific	Device-specific	If parameter Abort if error is activated the transfer will be terminated on error detection.

Parameter	Default value	Value	Description
Jump to line if error	Device-specific	Device-specific	If parameter Jump to line if error is activated, the transfer will be continued with the line which is set in parameter Next line in case of an error.
Next line	0	1 ... number of lines	The parameter Next line determines the line with which the transfer will be continued in case of an error if the parameter Jump to line if error is activated.
Comment	"	String	The parameter Comment shows a comment text of the corresponding line.
Buttons			
Move up	-	-	The order (top-down) in the startup parameters table represents the sequence in which the parameters will be transferred to the module. Button Move up shifts up the selected entry.
Move down	-	-	The order (top-down) in the startup parameters table represents the sequence in which the startup parameters will be transferred to the module. Button Move down shifts up the selected entry.
New... Remark 1 ↳ *Further information on page 5959*	-	-	The button New... opens the Select item from object directory to add a startup parameter entry to the table.
Delete...	-	-	The button Delete... deletes the currently selected line from the table.
Edit Remark 1 ↳ *Further information on page 5959*	-	-	The button Edit... opens the Select item from object dictionary dialog where it can be edited.

Remark 1

The dialog *"Select item from object dictionary"* shows all entries defined in the EtherCAT XML file:



The dialog allows editing the available startup parameters before adding them to the table. New entries can be created and edited which are not defined in the XML file.

Mapping of the EtherCAT slave I/Os

Double-click on the EtherCAT slave device in the device tree and select the mapping tab to show the current I/O mapping.

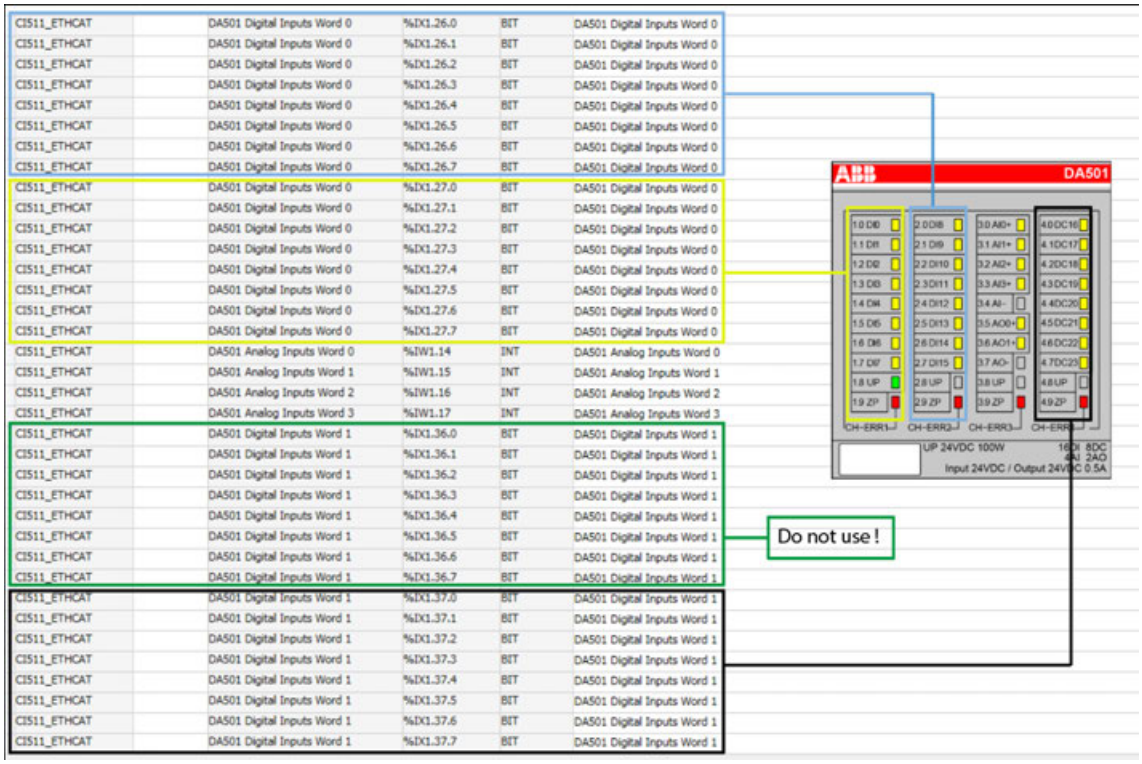
See Symbolic Names for Variables, Inputs and Outputs for further details on mapping of inputs and outputs *Chapter 1.6.5.2.9.6 “Symbolic names for variables, inputs and outputs” on page 6060*.

EtherCAT Master configuration of DA501

NOTICE!

Special attention is required for configuration of DA501 devices on EtherCAT networks:

- The order of the I/O channels in the Automation Builder I/O mapping editors is different from the order on the device
- The Automation Builder I/O mapping editors contain internal I/O channels that are not available on the device and must not be used in the configuration and application



EtherCAT Sync - Synchronization of a PLC task with the IO image

The EtherCAT master CM579-ETHCAT supports the synchronization of a PLC task with the IO image. For the configuration of EtherCAT Sync Automation Builder and the task configuration is used. Configuration of EtherCAT Sync is described in *Chapter 1.6.4.2.4.6.2 “EtherCAT sync” on page 5569*.

Configuration

☑ To configure EtherCAT synchronization perform the following steps:

1. Configure the EtherCAT master. ↳ *Chapter 1.6.5.2.6.8.3 “Configuration of the EtherCAT master” on page 5956*
2. Configure the EtherCAT slaves. This is identically to configuration in unsynchronized mode. ↳ *Chapter 1.6.5.2.6.8.4 “Configuration of the EtherCAT slaves” on page 5959*
3. ↳ *Chapter 1.6.5.2.6.8.7.1.1 “Configuration of the bus cycle time” on page 5967*
4. ↳ *Chapter 1.6.5.2.6.8.7.1.2 “Configuration of the master synchronization mode” on page 5967*
5. ↳ *Chapter 1.6.5.2.6.8.7.1.3 “Configuration of PLC task and synchronization mode” on page 5967*

Configuration of the bus cycle time

The bus cycle time can be set with the parameter "bus cycle time" which is part of the node CM579-ETHCAT of PLC configuration tree. It has a range from 0.5 ms to 20 ms.

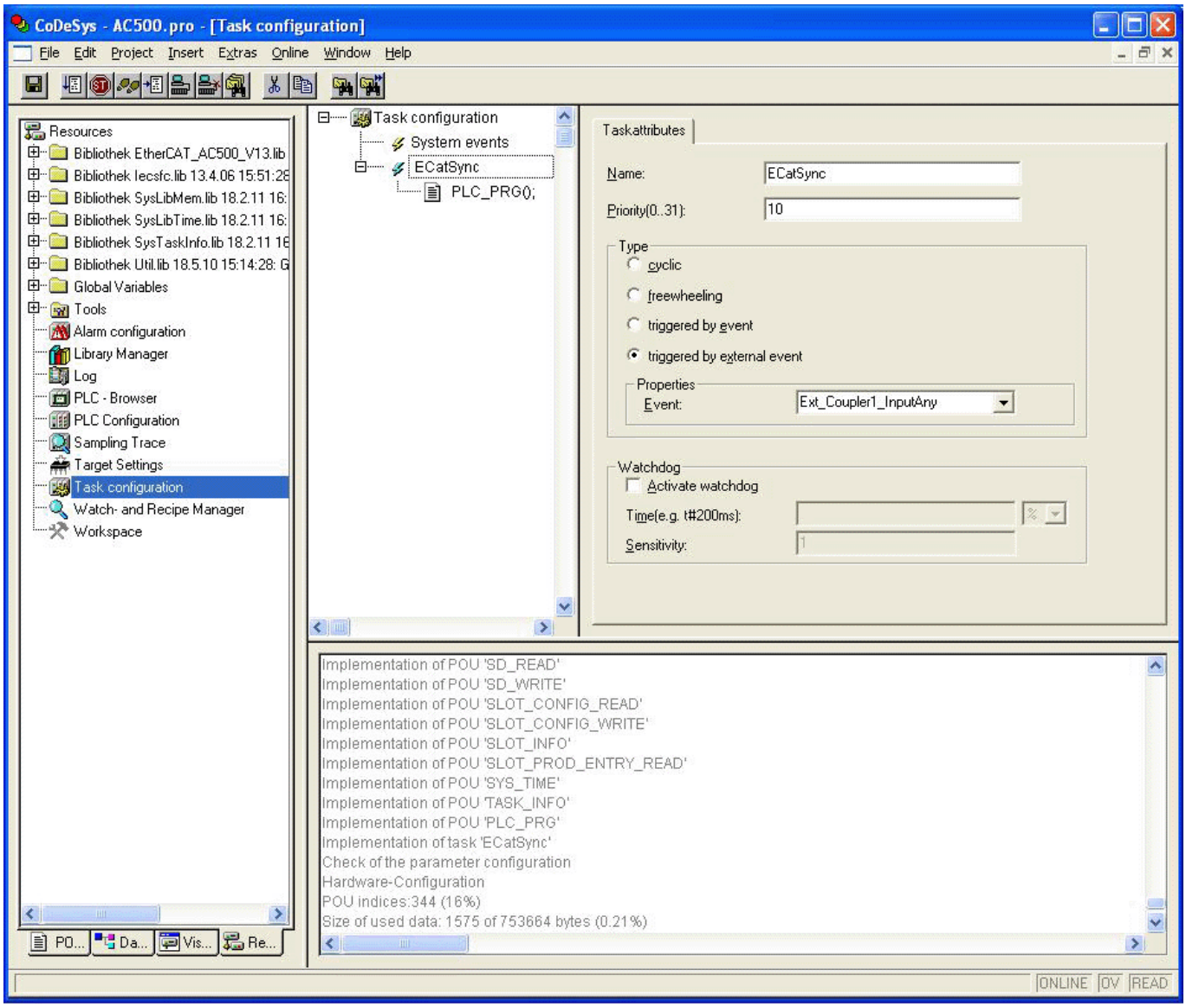
Configuration of the master synchronization mode

The EtherCAT can operate in "Cyclic mode" or "DC mode". The operation mode can be selected with the parameter "distributed clock" which is part of node CM579-ETHCAT of PLC configuration tree.

Parameter	Value	Description
Distributed clocks	Inactive	Cyclic mode
	Active	DC mode

Configuration of PLC task and synchronization mode

In Automation Builder the PLC program is written and assigned to a PLC task. For task creation and parameterization the dialog "task configuration" is used.



The task configuration is evaluated when a PLC project is loaded and the corresponding sync mode is activated for the EtherCAT master. The synchronization will be activated if the parameter task type is set to the value "triggered by external event". The mode of synchronization can be selected with the parameter event. Between the value of the parameter event and the synchronization mode exists the following relationship:

Parameter	Value	Description
Event	Ext_CouplerX_InputAny	Sync mode 1
	Ext_CouplerX_Input2Any	Sync mode 2
	Ext_CouplerX_InputAny_high_prio	Sync mode 1 with increased priority
	Ext_CouplerX_Input2Any_high_prio	Sync mode 2 with increased priority

1.6.5.2.7 Communication interface modules

Configuration of communication interface modules

Automation Builder can be used to configure the parameters of CI5xx devices.

Configuration of S500 I/O modules can be performed without CI5xx devices connected.

Adding CI5xx device to the device tree

1. Right click in the device tree on the node *“Slot1”* or *“Slot2”* of the *“Extension_Bus”* and click *“Add object”*.
   ⇨ The window *Replace object : Slot <...>* opens.
2. Select your *CM5xx master module* and click *[Add object]*.
   ⇨ The *CM5xx master* appears in the Slot.
3. Right click on the *CM5xx master module* and click *“Add object”*.
   ⇨ The window *Add object below : <...>_Master* opens.
4. Select your *“CI5xx”* device and click *[Add object]*.
   ⇨ The *“CI5xx”* device appears in your device tree.

Adding S500 I/O modules

1. Right click on your *“CI5xx”* device and click *[Add object]*.
   ⇨ The window *Add object below:* opens.
2. Select your I/O module and click *[Add object]*.
   ⇨ The I/O module is added.

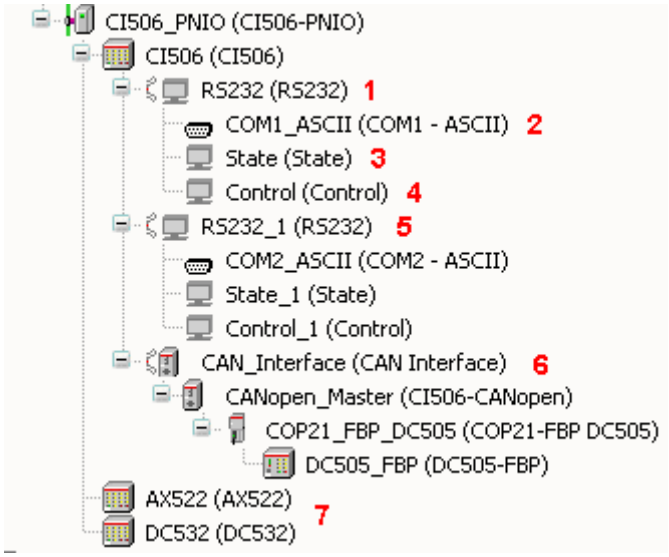
Configure parameters

▷ Double-click the *“CI5xx”* device to open editors and select the *“CI5xx_IO Parameters”* tab.

This editor shows the parameters that can be set for each device. For more information see ↳ *Chapter 1.6.2.8 “Communication interface modules (S500)” on page 4681*, and ↳ *Chapter 1.6.2.6 “I/O modules” on page 4124*.

CI504-PNIO/CI506-PNIO

Device tree



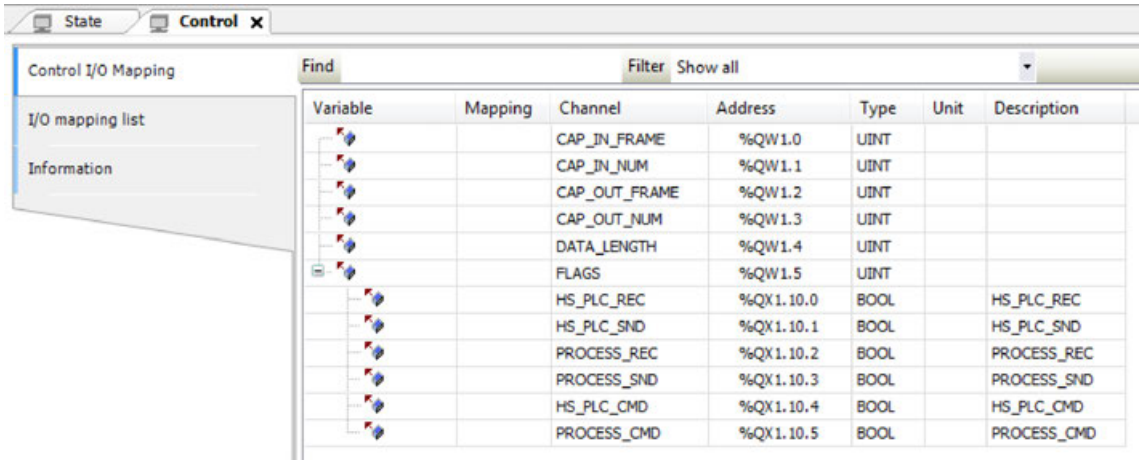
Remarks:

- The different hardware layers (RS-232, RS-485, RS-422) are configurable by plugging of corresponding module.
- For configuration of "COM1 ASCII", see The Setting "COMx ASCII" ↳ *Chapter 1.6.5.2.11.3 “Setting COMx - ASCII” on page 6100*.
- For inputs of serial (or raw CAN) interface see explanation below.
- For outputs of serial (or raw CAN) interface see explanation below.
- For configuration of "COM2 ASCII", see The Setting "COMx ASCII" ↳ *Chapter 1.6.5.2.11.3 “Setting COMx - ASCII” on page 6100*.
- For configuration of the CANopen master and underlying CANopen network, see Communication Module CANopen manager ↳ *Chapter 1.6.5.2.6.6.1.1 “CM598-CN - CANopen manager communication module” on page 5913*.

  => CANopen master is not available for CI504-PNIO. Instead a 3rd serial interface is available (configuration is the same as configuration of COM1)

  => CANopen master can be replaced by raw CAN (CAN 2A / CAN2B) at CI506-PNIO

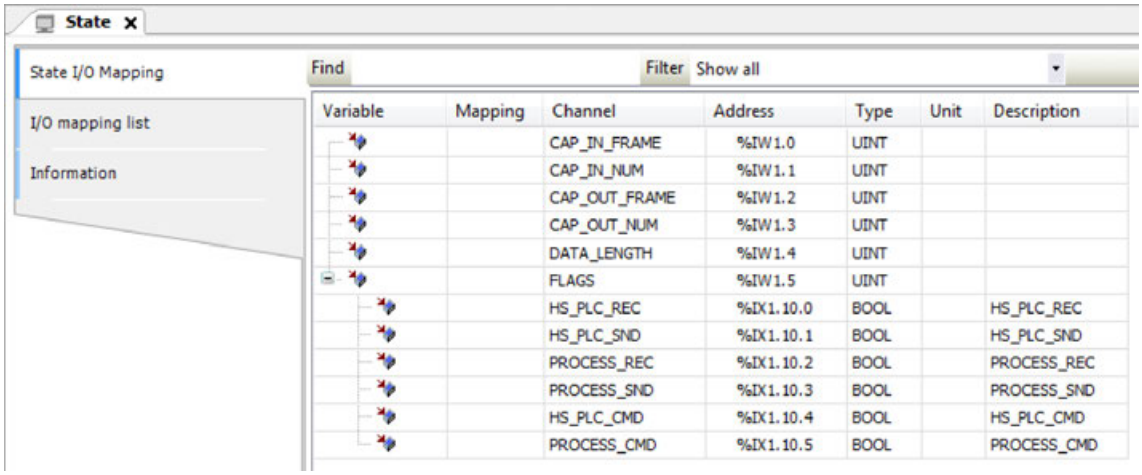
Outputs of serial (or Raw CAN) interface



Variable	Mapping	Channel	Address	Type	Unit	Description
		CAP_IN_FRAME	%QW1.0	UINT		
		CAP_IN_NUM	%QW1.1	UINT		
		CAP_OUT_FRAME	%QW1.2	UINT		
		CAP_OUT_NUM	%QW1.3	UINT		
		DATA_LENGTH	%QW1.4	UINT		
		FLAGS	%QW1.5	UINT		
		HS_PLC_REC	%QX1.10.0	BOOL		HS_PLC_REC
		HS_PLC_SND	%QX1.10.1	BOOL		HS_PLC_SND
		PROCESS_REC	%QX1.10.2	BOOL		PROCESS_REC
		PROCESS_SND	%QX1.10.3	BOOL		PROCESS_SND
		HS_PLC_CMD	%QX1.10.4	BOOL		HS_PLC_CMD
		PROCESS_CMD	%QX1.10.5	BOOL		PROCESS_CMD

Name of Output	Type of Output	Description
CAP_IN_FRAME	UINT	Reserved for future use
CAP_IN_NUM	UINT	Reserved for future use
CAP_OUT_FRAME	UINT	Reserved for future use
CAP_OUT_NUM	UINT	Reserved for future use
DATA_LENGTH	UINT	Reserved for function blocks - DO NOT USE
FLAGS	UINT	Reserved for function blocks - DO NOT USE

Inputs of serial (or Raw CAN) interface



Name of Input	Type of Input	Description
CAP_IN_FRAME	UINT	Remaining free capacity in the serial transmission FIFO (in frames)
CAP_IN_NUM	UINT	Remaining free capacity in the serial transmission FIFO (in bytes)
CAP_OUT_FRAME	UINT	Occupied level in serial reception FIFO (in frames)
CAP_OUT_NUM	UINT	Occupied level in serial reception FIFO (in bytes)
DATA_LENGTH	UINT	Reserved for function blocks - DO NOT USE
FLAGS	UINT	Reserved for function blocks - DO NOT USE

CI521-MODTCP/CI522-MODTCP

Unbundled CI52x-MODTCP configuration

Automation Builder can be used to configure the parameters of CI52x-MODTCP devices.

A direct Ethernet connection is required between the PC running Automation Builder and the CI52x-MODTCP module.

Configuration of S500 I/O modules can be performed without CI52x-MODTCPs modules connected.

Start a project from template

1. Select *"New Project"* in menue item *"File"*.
   ⇨ The window *"New Project"* appears.
2. Select the *"CI52x-MODTCP Configuration Project"* and click *"OK"*.
   ⇨ The window *"Select PLC"* opens.
3. Select a *"CI52x-MODTCP"* device and click *"Add device"*.
   ⇨ A project is created. More modules can be added.

Add CI52x-MODTCP to a project

1. Right click in the device tree on the root of the *"Project"* and click *"Add object"*.
   ⇨ The window *"Add object below"* opens.
2. Select *"Modbus devices"* and click *"Add object"*.
   ⇨ The node*"Modbus_devices"* appears in your device tree.
3. Right click on the node *"Modbus_devices"* and click *"Add object"*.
   ⇨ The window *"Select PLC"* opens.
4. Select your *"CI52x-MODTCP"* device and click *"Add device"*.
   ⇨ The *"CI52x-MODTCP"* device appears in your device tree.

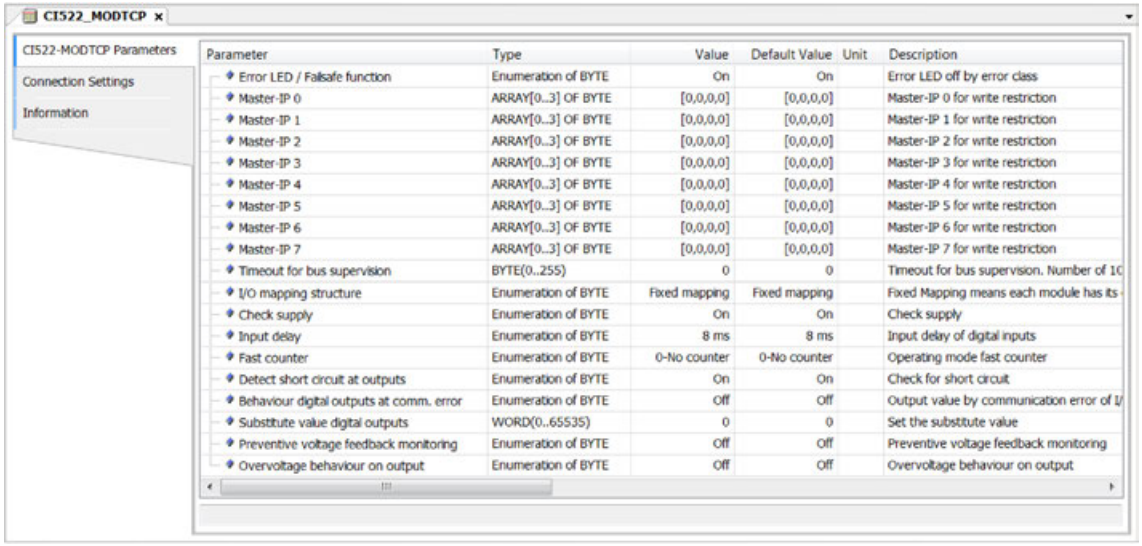
Add S500 I/O modules

1. *"Add object"* to your *"CI52x-MODTCP"* device.
   ⇨ The window *"Add object below: CI52x-MODTCP"* opens.
2. Select your I/O module and click *"Add object"*.
   ⇨ The I/O module is added.

Configure parameters

▷ Double-click the device to open editors and select the *"CI52x-MODTCP Parameters"* tab.

This editor shows the parameters that can be set for each device. For more information see *Chapter 1.6.2.8.5.1.7.1 "Parameters of the module" on page 4884* CI521, *Chapter 1.6.2.8.5.2.7.1 "Parameters of the module" on page 4914* CI522 and *Chapter 1.6.2.6 "I/O modules" on page 4124*.

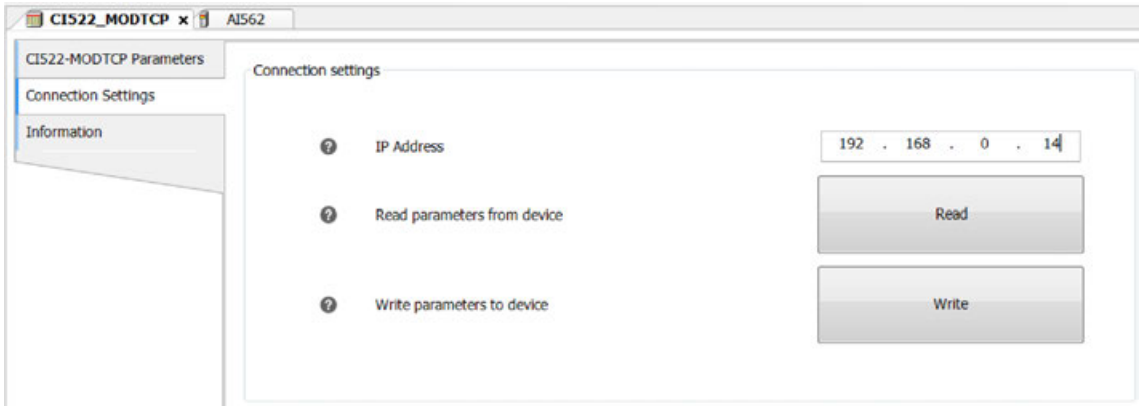


Connect to device

To read or write parameters, the CI52x-MODTCP module must be connected to the PC with an Ethernet connection.

See *Chapter 1.6.2.8.5.1.5 "Addressing" on page 4883* CI521 and *Chapter 1.6.2.8.5.2.5 "Addressing" on page 4914* CI522 of the CI52x-MODTCP hardware documentation for information on configuring the IP address of the device.

On the CI52x-MODTCP device editor, the *"Connection Settings"* tab allows the IP address of the device to be entered.



Read Reads the parameters from the CI52x-MODTCP and also for the attached S500 I/O modules.

Write Sends the parameters from the editors to the CI52x-MODTCP and also the S500 I/O modules.

Device checking

The CI52x-MODTCP module knows which I/O modules are attached.

While reading and writing parameters, the project must match the physical hardware. Otherwise an error will be given.

Communication errors will also result in error messages.

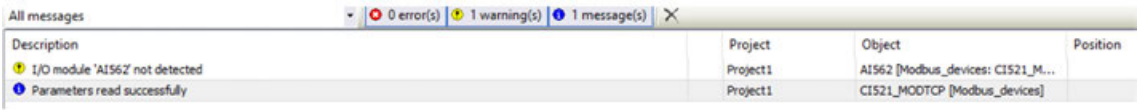
When the parameters have been read or written correctly, a message is seen in the *"All messages"* window:

Attached S500 modules

It is possible to read and write parameters when the S500 I/O modules are not attached to the CI52x-MODTCP module.

> *To perform a read, the project structure must still match the configuration of CI52x-MODTCP.*

A warning will be shown if an I/O module is not detected:



When writing parameters, the CI52x-MODTCP configuration is overwritten so the current configuration of missing (unplugged) modules does not matter.

If the I/O modules are attached, then the project must match the hardware, otherwise an error will be given.

Firmware update

As of Automation Builder 2.2.1, the IP Configuration Tool can be used to perform firmware updates for CI52x-MODTCP devices.

1.6.5.2.8 Function modules

DC541-CM interrupt and counter module

The function module DC541-CM can be used for a wide variety of control tasks. It can be configured to act for example as an interrupt I/O, as a fast counter or to provide pulse width modulation functionality.

Configuration in Automation Builder is mainly identical, however, programming with the provided function blocks differ. All necessary function blocks for DC541-CM programming are available in the *Chapter 1.5.4.11 “DC541 library” on page 1103*.

Details on the hardware is provided in the device descriptions *Chapter 1.6.2.6.1.2.4 “DC541-CM - Digital input/output module” on page 4290*.

General Automation Builder configuration

1. In the device tree, add the object *“DC541-CM”* (Extension Bus -> Slot).
2. Double-click the added object and configure the I/O channels.

   The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not started.
		Yes	The user program is started even if the internal Ethernet communication module is configured incorrectly.
Do not delete config on Reset (original)	On	On	The configuration of the DC541-CM is not deleted in case of a reset (original).
		Off	In case of a reset (original), the configuration of the DC541-CM is deleted, too.
Num edges ignore input 0	0	0...255	Number of edges that may occur at input 0 without initiating the interrupt task, if channel C0 is configured as interrupt input.
Watchdog	On	On	Mutual time monitoring between the CPU and the DC541-CM is switched on.
		Off	No time monitoring.

Depending on the desired use case of the function module DC541-CM, the appropriate operating mode is to be defined. For this, add a new object to the *“DC541-CM”* node and select an operating mode.

Usage as interrupt I/O device

Cycle time modification

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↬ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

Access to the channels configured as normal inputs and outputs is performed using the function block ↬ *Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139.*

The module's cycle time is set automatically depending on its channel configuration. The following values are possible for the cycle time:

CYCLE

Data type	Default value	Range	Unit
WORD	-	-	µs

CYCLE (cycle time) output displays the cycle time of the device. The cycle time is set during the device configuration and can have the following values depending on the channel configuration:

Parameter		Description	Value
IO device			50 µs
Counting device		1-2 functions	50 µs
		3-4 functions	100 µs
		5-8 functions	200 µs
"Functions"			
	PWM	Pulse-width modulator	
	FREQ	Time and frequency measurement	
	FREQ_OUT	Frequency output	
	32BIT_CNT	32-bit counter	
	FWD_CNT	32-bit count up counter	
	LIMIT	Limit value monitoring for the 32-bit counter	

The used cycle time can be read at output CYCLE of the block *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132*.

The following table shows an overview of all possible combinations.

Configured as	Function/ can be configured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
Mode 1: Interrupt function; mutually exclusive with mode 2 (counting functions).								
Interrupt	Dig. input	1	1	1	1	4	8	Each channel can be configured individually as interrupt input or output.
	Interrupt inp.	1	1	1	1	4	8	
	Dig. output	1	1	1	1	4	8	
Mode 2: Counting functions and multifunctional I/Os; mutually exclusive with mode 1 (interrupt functions).								
Multi-function I/Os, PWM, counters,	Dig. input	1	1	1	1	4	8	Normal input
	Dig. output	1	1	1	1	4	8	Normal output

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
time and fre-quency meas-uring	PWM, resolu-tion 10 kHz	1	1	1	1	4	8	Outputs a pulsed signal with an adjust-able on-off ratio.
	Fre-quency output, resolu-tion 2.5 kHz	1	1	1	1	4	8	Outputs an adjust-able fre-quency (endless output or output of a speci-fied number of pulses).

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Up/down counter, 50 kHz	1	1	OK *1)	OK *1)	OK *1)	2	*1) a) Both channels (0 and 1) config-ured as 50 kHz counter => Chan-nels 2 to 7 can be config-ured as digital I/Os. b) Only one channel (0 or 1) config-ured as 50 kHz counter => Second channel can be config-ured as counter < 50 kHz or for time/ fre-quency meas-urement with a max. resolu-tion of 200 µs. The remainin g chan-nels (2 to 7) can be con-figured as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Up/down counter, 5 kHz	1	1	1	1	OK *2)	4	*2) a) Four channels (0 to 3) config-ured as 5 kHz counter => Chan-nels 4 to 7 can be config-ured as digital I/Os. b) Only a portion of the 4 channels (0 to 3) config-ured as 5 kHz counter => The other ones (of channels 0 to 3) can be config-ured as desired: as 2.5 kHz counter or for time/ fre-quency meas-urement with a max. resolu-tion of 200 µs or as digital I/Os. The remainin

Configured as	Function/ can be configured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
								g channels (4 to 7) can be configured as digital I/Os.
	Up/down counter, 2.5 kHz	1	1	1	1	4	8	
	Time/ frequency measurement, resolution 50 µs	1	OK *3)	OK *3)	OK *3)	OK *3)	1	*3) Channel 0 configured for a max. resolution of 50 µs => Channels 1 to 7 can be configured as digital I/Os.

Configured as	Function/ can be configured for channel	C0	C1	C2	C3	C4 to C7	Max. number of channels for this function	Remark and reference to alternative combinations (a and b)
	Time/ frequency measurement, resolution 100 µs	1	1	OK *4)	OK *4)	OK *4)	2	*4) a) Two channels (0 and 1) configured for a max. resolution of 2x100 µs => Channels 2 to 7 can be configured as digital I/Os. b) Only one channel (0 or 1) configured for a max. resolution of 50 µs => Second channel (0 or 1) can be configured as counter < 50 kHz or for time/ frequency measurement with a max. resolution of 200 µs. The remaining channels (2

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
								to 7) can be con-figured as digital I/Os.
	Time/ fre-quency meas-urement, resolu-tion 200 µs	1	1	1	1	4	8	Times, frequen-cies and rota-tional speeds are meas-ured with a max. resolu-tion of 200 µs.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
High-speed counter	Up/down 32-bit counter, 50 kHz max.	Channels 0 to 3: Track A, track B, zero track, touch trigger				OK *6)	1	For con-nection of an incre-mental trans-mitter. For sig-nals up to 50 kHz. This fre-quency corre-sponds to a motor with a rota-tional speed of 3000 rpm. The counter always uses the first 4 channels (0 to 3). *6) The remainin g chan-nels (4 to 7) can be con-figured as limit values, as 5 kHz coun-ters, for time/ fre-quency meas-urement with a resolu-tion of 200 µs or as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	Axis of rotation (endless counting)	1				OK *7)	1	"End-less" for-ward counting. An over-flow occurs corre-sponding to the 32-bit value. *7) The remainin g chan-nels can be con-figured as limit values, as 5 kHz coun-ters, for time/ fre-quency meas-urement with a resolu-tion of 200 µs or as digital I/Os.

Config-ured as	Func-tion/ can be con- fig-ured for channel	C0	C1	C2	C3	C4 to C7	Max. number of chan-nels for this function	Remark and ref-erence to alter-native combi-nations (a and b)
	32-bit counter incl. sign	1				OK *8)	1	*8) The remainin g chan-nels can be con-figured as limit values, as 5 kHz coun-ters, for time/ fre-quency meas-urement with a resolu-tion of 200 µs or as digital I/Os.
	Limit values for 32-bit counter	OK *9)				1	1	Various counting values of the 32-bit counter can be dis-played directly via these outputs. *9) In this case, the channels 0 to 3 are used as 32-bit coun-ters.

Creating an interrupt task for the interrupt inputs

If one or more channels of DC541-CM are configured as interrupt inputs, a corresponding interrupt task has to be created to enable the processing of the interrupt(s).

For this purpose, a new task has to be added in the task configuration of Automation Builder:

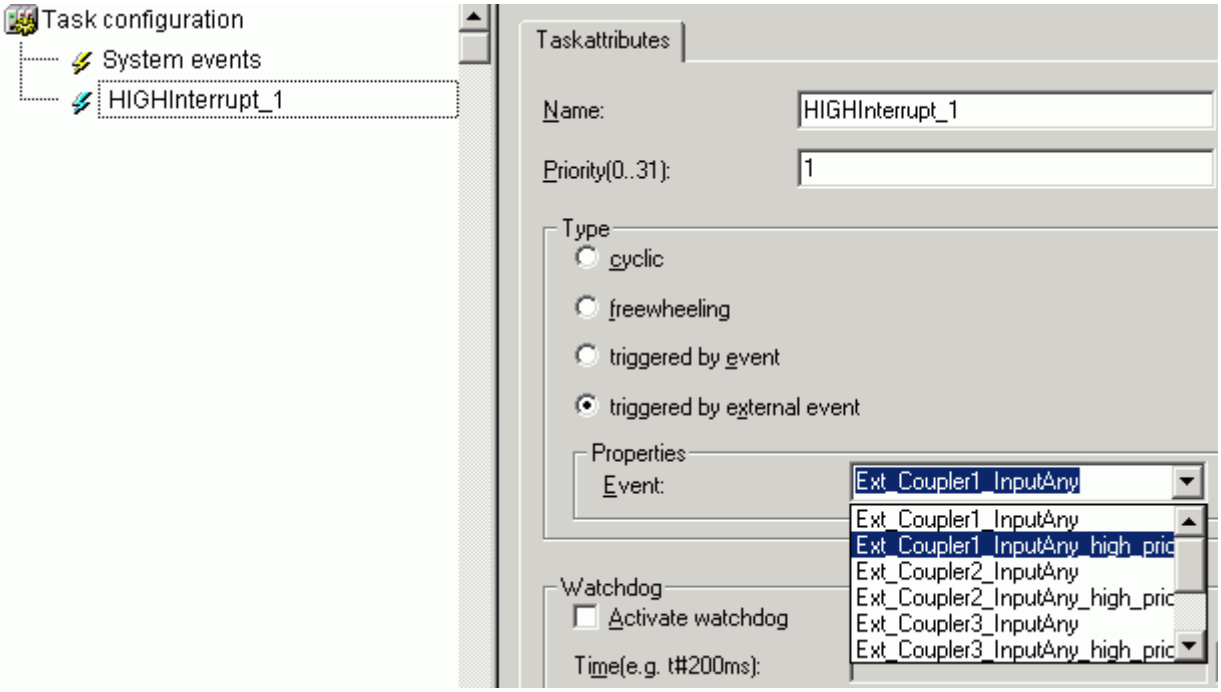
- Enter the task name
- Set the task type to "triggered by external event"
- Specify the event that triggers the task

For each Communication Module slot, two types of interrupt tasks are available in the Event list box:

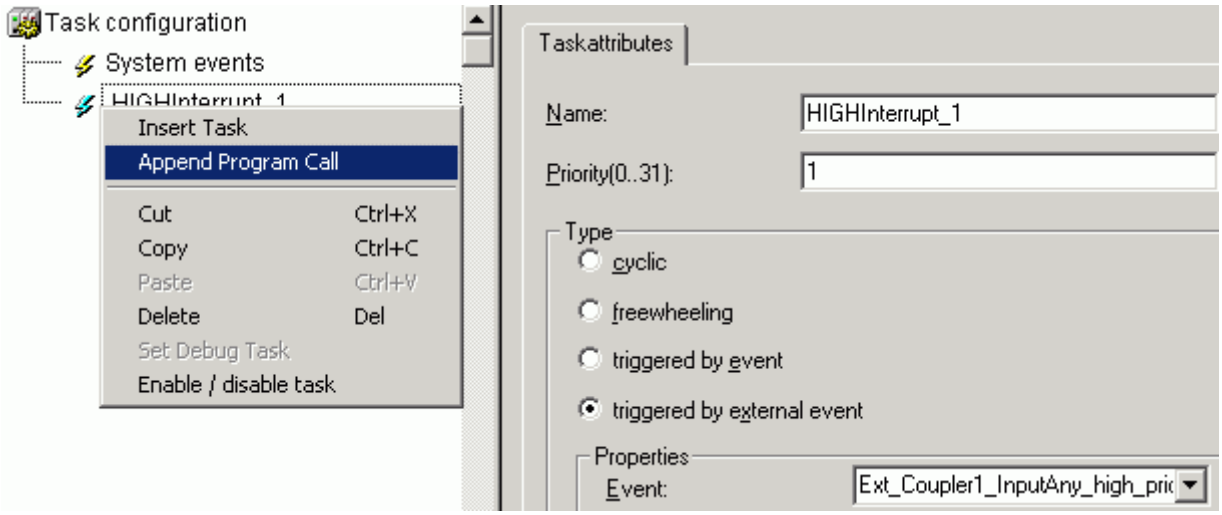
- Ext_Communication ModuleX_InputAny:
  The task is triggered by any interrupt from Communication Module slot X with the priority specified in the Priority field (0...31).
- Ext_Communication ModuleX_InpuAny_high_prio:
  The task is triggered by any interrupt from Communication Module slot X with highest priority, i.e. with a priority higher than the max. adjustable "0" and higher than the priority of the communication task. In this case, the priority (0...31) specified in the Priority field is without any significance.

> *If the interrupt task is started with highest priority, the program execution time must not be longer than approx. 400 µs. Otherwise online access is no longer possible.*

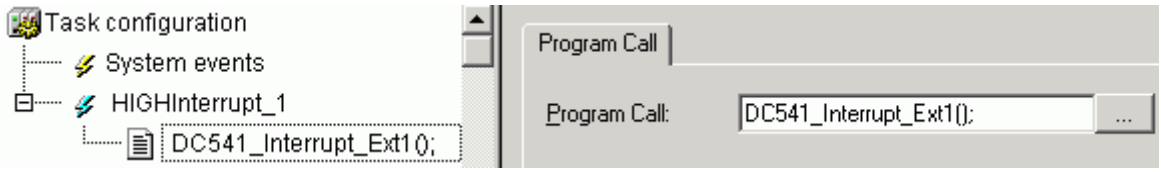
In the example below, the task is named HIGHInterrupt_1, meaning that it is a high-priority interrupt from Communication Module slot 1. The task type is "external event triggered" and the event to trigger the task is "Ext_Communication Module1_InputAny_high_priority".



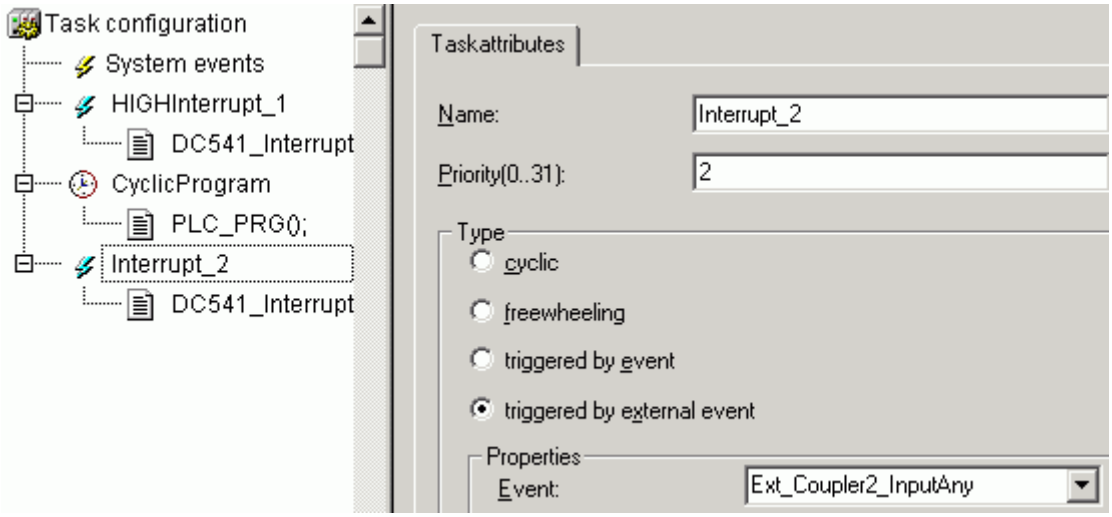
Like for all other tasks, a program call has to be assigned to the task.



In the example, the program DC541_Interrupt_Ext1() shall be started with any interrupt from Communication Module slot 1.



The task configuration for an AC500 equipped with two DC541-CM modules inserted in the Communication Module slots 1 and 2 and containing one cyclically running "background program" PLC_PRG could for example look as follows. Here, an interrupt from slot 1 should start the program DC541_Interrupt_Ext1 with high priority, an interrupt from slot 2 should start the program DC541_Interrupt_Ext2 with priority 2:



Structure of the interrupt program

The following blocks contained in the library DC541_AC500_V11.lib are available for the interrupt program:

- ↳ *Chapter 1.5.4.11.1.1.7 “DC541_INT_IN” on page 1136* Determination of the interrupt initiating source
- ↳ *Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139* Reading and writing of channels C0...C7

It is possible to start one interrupt task per Communication Module slot. This task can be started by any channel (C0...C7) configured as interrupt input. Therefore, it is necessary for the interrupt program to differentiate which channel(s) triggered the interrupt in order to enable the processing of the corresponding actions.

The information whether a channel (C0...C7) has triggered an interrupt since the last call of the block is provided by the outputs IN0...IN7 of the block *Chapter 1.5.4.11.1.1.7 “DC541_INT_IN” on page 1136*. This is why this block always has to be called at the beginning of the interrupt program, if more than one channel is configured as interrupt input.

The access to the channels configured as inputs or outputs is done using the block DC541_IO. Therefore, it makes sense to call this block at the beginning of the interrupt program in order to read the inputs and at the end of the interrupt program in order to write the outputs.

--Configuration example: DC541-CM used as interrupt I/O device

Hardware configuration

The example control system shall have the following configuration:

- Terminal base TB521 (two Communication Module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH
- I/O module DC532 on the I/O Bus

Wiring

The channels are connected as follows:

DC532 / C16 ---------- DC541 / C0

DC532 / C17 ---------- DC541 / C1

DC532 / C18 ---------- DC541 / C2

DC532 / C19 ---------- DC541 / C3

DC532 / C20 ---------- DC541 / C4

DC532 / C21 ---------- DC541 / C5

PLC configuration

- DC541-CM in slot 1, operating mode "IO mode"

- Configuration:	Channels	C0...C4	Interrupt input
	Channel	C5	Input
	Channels	C6...C7	Outputs

- Specification of the Ethernet communication module as internal communication module (if available)
- DC532 on the I/O bus

Task configuration

- Task 1: Cyclic program / Prio = 10 / Interval = t#10ms / PLC_PRG
- Task 2: HIGHInterrupt_1 / DC541_Interrupt_Ext1()

DC541_Interrupt_Ext1()

The interrupt program should fulfill the following functionality:

- Counting of all interrupts
- Counting of the interrupts per input
- Calculation of the interrupt frequency in [Int/s]
- Reporting of the number of interrupts per input
- Input C4: Resetting the counters
- Input C5: Input

- Output C6: Status of input C5
- Output C7: Toggle output

The declaration part of the program looks as follows:

```
PROGRAM DC541_Interrupt_Ext1
VAR
    dwIntCount          : DWORD;                    (* count all interrupts *)
    dwIntCountOld       : DWORD;                    (* start value for next measure *)
    tActual             : TIME;                     (* systemtick in ms *)
    tStart              : TIME;                     (* start value of systemtick for next calculation *)
    dwUsedTime          : DWORD;                    (* time for 1000 interrupts in ms *)
    dwFrequenz          : DWORD;                    (* interrupt frequency in [Int / sec] *)
    DC541_IntSource     : DC541_INT_IN;             (* instance FB: read interrupt source *)
    DC541_Ios           : DC541_IO;                 (* instance FB: read/write inputs/outputs *)
    dwCount_InX         : ARRAY[0..cbyDC541_IntInp] OF DWORD;   (* count interrupts of In0..In3 *)
    dwCount_InXOld      : ARRAY[0..cbyDC541_IntInp] OF DWORD;   (* start value for next 1000 interrupts *)
    dwIntHisto          : ARRAY[0..cbyDC541_IntInp, 0..cbyDC541_MaxHist] OF DWORD;   (* histo data C0...C3 *)
    wIndex              : WORD;                     (* index for histo data *)
    byInd               : BYTE;                     (* loop index *)
END_VAR
VAR CONSTANT
    cbyDC541_SLOT       : BYTE := 1;                (* SLOT number of DC541 *)
    cbyDC541_MaxHist    : BYTE := 9;                (* max number of histo entries *)
    cbyDC541_IntInp     : BYTE := 4;                (* number of interrupt inputs -1 *)
END_VAR
```

The instruction part looks as follows:

At the beginning, the interrupts are counted in dwIntCount. After each 1000 interrupts, a calculation of the frequency is performed and the counting values for the interrupts per input are stored.

dwIntCount := dwIntCount + 1;		(* count all interrupts *)
IF dwIntCount - dwIntCountOld >= 1000 THEN		(* after 1000 interrupts -> calculate frequency *)
	dwIntCountOld := dwIntCount;	(* save dwIntCount for next call *)
	tActual := TIME();	
	dwUsedTime := TIME_TO_DWORD(tActual - tStart);	(* duration in ms for 1000 interrupts *)
	dwFrequenz := 1000000 / dwUsedTime;	(* [Interrupt / sec] 1000 Int * 1000 ms/sec *)
	tStart := tActual;	(* for next measure *)
	dwIntHisto[0,wIndex] := dwCount_InX[0] - dwCount_InXOld[0];	(* IN0 interrupts of last 1000 *)
	dwCount_InXOld[0] :=dwCount_InX[0];	(* start value for next measure *)
	dwIntHisto[1,wIndex] := dwCount_InX[1] - dwCount_InXOld[1];	(* IN1 interrupts of last 1000 *)
	dwCount_InXOld[1] :=dwCount_InX[1];	(* start value for next measure *)
	dwIntHisto[2,wIndex] := dwCount_InX[2] - dwCount_InXOld[2];	(* IN2 interrupts of last 1000 *)
	dwCount_InXOld[2] :=dwCount_InX[2];	(* start value for next measure *)
	dwIntHisto[3,wIndex] := dwCount_InX[3] - dwCount_InXOld[3];	(* IN3 interrupts of last 1000 *)
	dwCount_InXOld[3] :=dwCount_InX[3];	(* start value for next measure *)
	wIndex := wIndex + 1;	(* increase index *)
	IF wIndex > cbyDC541_MaxHist THEN wIndex := 0; END_IF;	(* reset index, if >1000 *)
END_IF; (* 1000 Interrupts *)		

After this, the block DC541_INT_IN is called to identify the interrupt source and then the interrupt counters of the channels are updated depending on the outputs of this block.

```
(* Read interrupt source --> if output  = TRUE --> interrupt since last call *)
DC541_IntSource(EN := TRUE, SLOT := cbyDC541_SLOT);
(* count the interrupts for each interrupt input C0..C3 *)
dwCount_InX[0] := dwCount_InX[0] + BOOL_TO_DWORD(DC541_IntSource.IN0);
dwCount_InX[1] := dwCount_InX[1] + BOOL_TO_DWORD(DC541_IntSource.IN1);
dwCount_InX[2] := dwCount_InX[2] + BOOL_TO_DWORD(DC541_IntSource.IN2);
dwCount_InX[3] := dwCount_InX[3] + BOOL_TO_DWORD(DC541_IntSource.IN3);
dwCount_InX[4] := dwCount_InX[4] + BOOL_TO_DWORD(DC541_IntSource.IN4);
```

In case of an interrupt on channel 4, the counters are reset.

IF DC541_IntSource.IN4 THEN		(* Input channel C4 = TRUE *)
	dwIntCount := dwIntCoun-tOld := 0;	(* reset count all interrupts *)
	FOR byInd := 0 TO cbyDC541_IntInp-1 DO	(* reset channel interrupt counters C0..C3 *)
	dwCount_InX[byInd] := dwCount_InXOld[byInd] := 0;	
	END_FOR; (* byInd *)	
	wIndex := 0;	(* start historical data from 0 *)
END_IF; (* C4 = TRUE *)		

At the end, the static inputs and outputs are processed, i.e.:

- reading the inputs,
- execution of actions
- writing the outputs.

(* Read inputs of DC541 *)		
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT);		
DC541_IOs.OUT6 := DC541_IOs.IN5;	(* C6 := state of input channel C5 *)	
DC541_IOs.OUT7 := NOT DC541_IOs.OUT7;	(* toggle channel C7 *)	
(* Write outputs to DC541*)		
DC541_IOs(EN := TRUE, SLOT := cbyDC541_SLOT);		

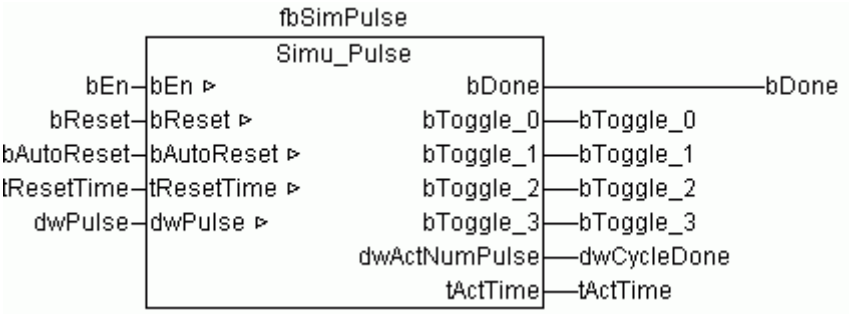
Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

- Cycles counter dwC := dwC + 1;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE - Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Simulation of the interrupts for the DC541 Calling of block Simu_Pulse

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the Function Block Diagram (FBD) is as follows:



The meanings of the block's inputs and outputs are as follows:

Instance		fbSimuPulse	Instance name
bEn	Input/Output	BOOL	Enabling of the pulse output
bAutoReset	Input/Output	BOOL	Automatic reset of the pulse counter after the specified number of pulses have been output and after expiration of tResetTime
bReset	Input/Output	BOOL	Reset of the pulse counter
tResetTime	Input/Output	TIME	Time until the reset is initiated after the specified number of pulses is reached, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: =0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE > 0: Cyclic mode (output of the specified number of pulses)
bDone	Output	BOOL	Completion message after tResetTime has expired or bReset = TRUE for 1 cycle
bToggle_0	Output	BOOL	Provides a FALSE->TRUE edge with each 2nd call (i.e. the output is toggled with each call)
bToggle_1	Output	BOOL	Provides a FALSE->TRUE edge with each 4th call
bToggle_2	Output	BOOL	Provides a FALSE->TRUE edge with each 8th call
bToggle_3	Output	BOOL	Provides a FALSE->TRUE edge with each 16th call
dwActPulse	Output	DWORD	Displays the number of pulses output (corresponds to the number of edges at bToggle_0)
tActTime	Output	TIME	Displays the elapsed time while tResetTime is running

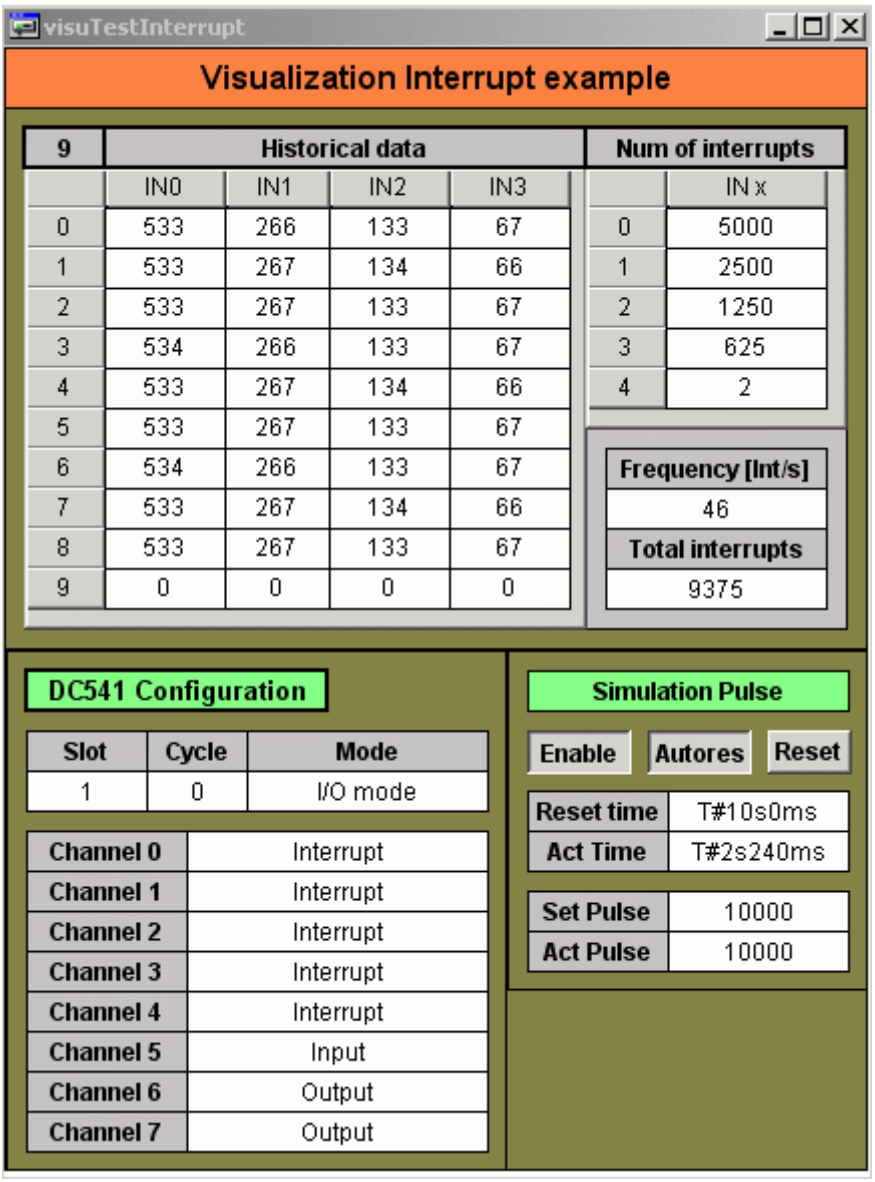
In the example, bEn: = bAutoReset: = TRUE. 10000 pulses are output (dwSetPulse). After the specified number of pulses has been reached, a wait time of 10 seconds is applied and then counting is started from the beginning.

The example has a visualization implemented which can be used to operate the program. After 10000 pulses, the visualization looks as follows:

9375 interrupts are generated:

5000 x C0 + 2500 x C1 + 1250 x C2 + 625 x C3 = 9375

Act Pulse					**Triggers the following interrupts:**
Value	**IN 3** **8**	**IN 2** **4**	**IN 1** **2**	**IN 0** **1**	
0	0	0	0	0	none
1	0	0	0	1	IN 0 -> in every 2. cycle (10000 : 2 = 5000)
2	0	0	1	0	IN 1 -> in every 4. cycle (10000 : 4 = 2500)
3	0	0	1	1	IN 0
4	0	1	0	0	IN 2 -> in every 8. cycle (10000 : 8 = 1250)
5	0	1	0	1	IN 0
6	0	1	1	0	IN 1
7	0	1	1	1	IN 0
8	1	0	0	0	IN 3 -> in every 16. cycle (10000 : 16 = 625)
9	1	0	0	1	IN 0
10	1	0	1	0	IN 1
11	1	0	1	1	IN 0
12	1	1	0	0	IN 2
13	1	1	0	1	IN 0
14	1	1	1	0	IN 1
15	1	1	1	1	IN 0
16	0	0	0	0	none



Usage as counting device

32-Bit up/down counter of module DC541-CM

The 32-bit bidirectional counter functionality is provided by the function block ↳ *Chapter 1.5.4.11.1.1.1 “DC541_32BIT_CNT” on page 1103*.

The 32-bit counter is a count up/count down counter with a directional discriminator. The counter can be used in two counting modes:

- EN_UD = FALSE: Encoder mode
  Connection of an incremental transmitter (track A / track B, offset by 90°)
  It is possible to count signals up to approx. 60 kHz. This corresponds to a motor with a rotational speed of 3.600 rpm and a transmitter with 1.000 pulses per rotation. Pulse multiplication (x2 or x4) is not used.
- EN_UD = TRUE: Up / down mode
  Up-/down counter
  It is possible to count signals up to approx. 60 kHz. Count-up for signals on channel C1, count-down for signals on channel C0.

The counter always uses the channels C0...C3 of the DC541:

- C0: Track A of the incremental transmitter.
- C1: Track B of the incremental transmitter.
- C2 and C3: Reference cam or touch trigger.

The counter can be used in two operating modes:

- Infinite counter (endless mode)
- Limiting counter (limit mode)

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

32-Bit forward counter of module DC541-CM

The 32-bit forward counter functionality is provided by the block ↳ *Chapter 1.5.4.11.1.1.5 “DC541_FWD_CNT” on page 1127*.

The function block DC541_FWD_CNT provides a 32-bit count up counter which is able to count a maximum frequency of 50 kHz at the inputs C0 and C1 or 5 kHz at the inputs C2-C7. In the DC541, the counter is implemented as a 16 bit counter. The actual counter value ACT_CNT is bulit inside the function block by adding the counter differences that occur within the individual cycles. In order not to loose any counting pulses, the function block has to be called cyclically.

- Channel 0-1: 50 kHz max. -> 32767 / 50 = 655 ms
- Channel 2-7: 5 kHz max. -> 32767 / 5 = 6550 ms

Using the counter e.g. in a 100 ms task will prevent any loss of counting pulses.

Operating modes

- Infinite counter (endless mode)
- Limiting counter (limit mode)

The operating mode is selected at input EN_LIM.

If EN_LIM = FALSE, the counter operates as an infinite counter (endless mode). An overflow occurs corresponding to the 32-bit value at 16#FFFFFFFF = 4 294 967 295. In this mode, any exceeding of the limit value LIM_MAX or falling below the limit value LIM_MIN is displayed at the outputs MAX_LIM or MIN_LIM.

If EN_LIM = TRUE (limit mode), the counting range is between the limit values LIM_MIN and LIM_MAX. In case of an overflow, i.e. if LIM_MAX is reached, the counter restarts again at LIM_MIN.

The upper limit value LIM_MAX has to be higher than the lower limit value LIM_MIN. If LIM_MIN is higher than LIM_MAX, an error is displayed.

The device DC541 must be configured as counting device (counter mode).

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

Configuration example: 32-Bit forward counter

All of the 8 channels of the DC541-CM can be used as count up counter. In the configuration example, all 8 channels of the DC541-CM are configured as 32-bit forward counter (count-up). The channels C0...C3 operate as infinite counters (endless mode), the channels C4...C7 as limit counters (limit mode).

The 32-bit count up counter configured as infinite counter (endless mode) corresponds to mode 1 (1 count up counter) of the high-speed counter of the digital input/output modules. In the configuration example, the counting pulses for the first forward counter are therefore applied in parallel to input C0 of the DC541-CM and counting input C24 of the DC532.

Hardware configuration

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet Communication Module
- I/O module DC532 on the I/O bus

Wiring

The channels are connected as follows:

- DC532 / C16 ---------- DC541 / C0
- DC532 / C17 ---------- DC541 / C1
- DC532 / C18 ---------- DC541 / C2
- DC532 / C19 ---------- DC541 / C3
- DC532 / C20 ---------- DC541 / C4
- DC532 / C21 ---------- DC541 / C5
- DC532 / C22 ---------- DC541 / C6
- DC532 / C23 ---------- DC541 / C7
- DC532 / C16 ---------- DC532 / C24

PLC configuration

- DC541-CM in slot 1, operating mode "Counter mode"
- Configuration: - Channel C0..C7 Forward counter
- Specification of the Ethernet communication module as internal communication module (if available)
- DC532 on the I/O bus / parameter "Fast counter" = 1-1 count up counter

Task configuration

- Task 1: Cyclic program / Prio = 10 / Interval = t#100ms / PLC_PRG
- Task 2: Simulation / Prio = 15 / Interval = t#5ms / Simulation_Task

The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for the counters Calling of program proForwardCounter

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation ↬ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 and described in detail in the corresponding documentation ↬ *Chapter 1.5.4.19 “Internal system library” on page 1500*.

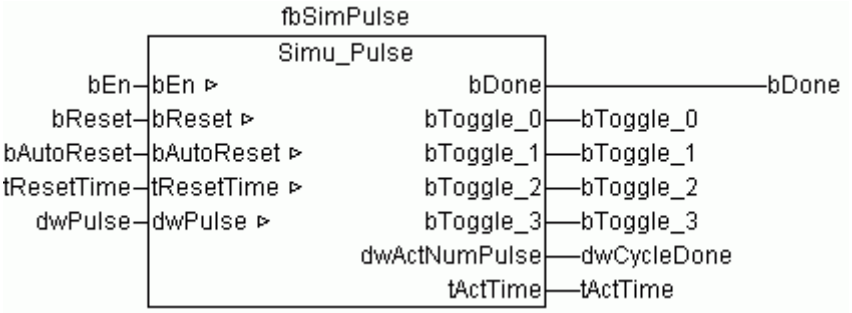
The actual execution of the 32-bit forward counter functionality is implemented in the program proForwardCounter.

Purpose of the program proForwardCounter:

The program proForwardCounter executes the following step chain:

Counter block	**DC541_FWD_CNT CNT_IO**								**CNT_IO**
Step | Channel	**C0**	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**1**
0 | Action	Init: SET = 0, endless counter, limit values MIN = 300 / MAX = 1300				Init: SET = 0, limit counter, limit values MIN = 300 / MAX = 1300				Init
| Value	0	0	0	0	0	0	0	0	0
1 | Action	Reset of SET input								
| Value	0	0	0	0	300	300	300	300	0
2 | Action	Start of pulse output - 2000 pulses								
| Value	0	0	0	0	300	300	300	300	0
3 | Action	Wait until pulse output is completed								
| Value	2000	1000	500	250	1299	1300	800	550	2000
4 | Action	Selection last step: byStep = 249								
| Value	2000	1000	500	250	1299	1300	800	550	2000
200 | Action	Manual operation								
| Value	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx	xxx
249 | Action	Wait time 5 seconds, then restart from step 0								
| Value	2000	1000	500	250	1299	1300	800	550	2000

The block Simu_Pulse is used to generate an adjustable number of pulses. Its representation in the Function Block Diagram (FBD) is as follows:



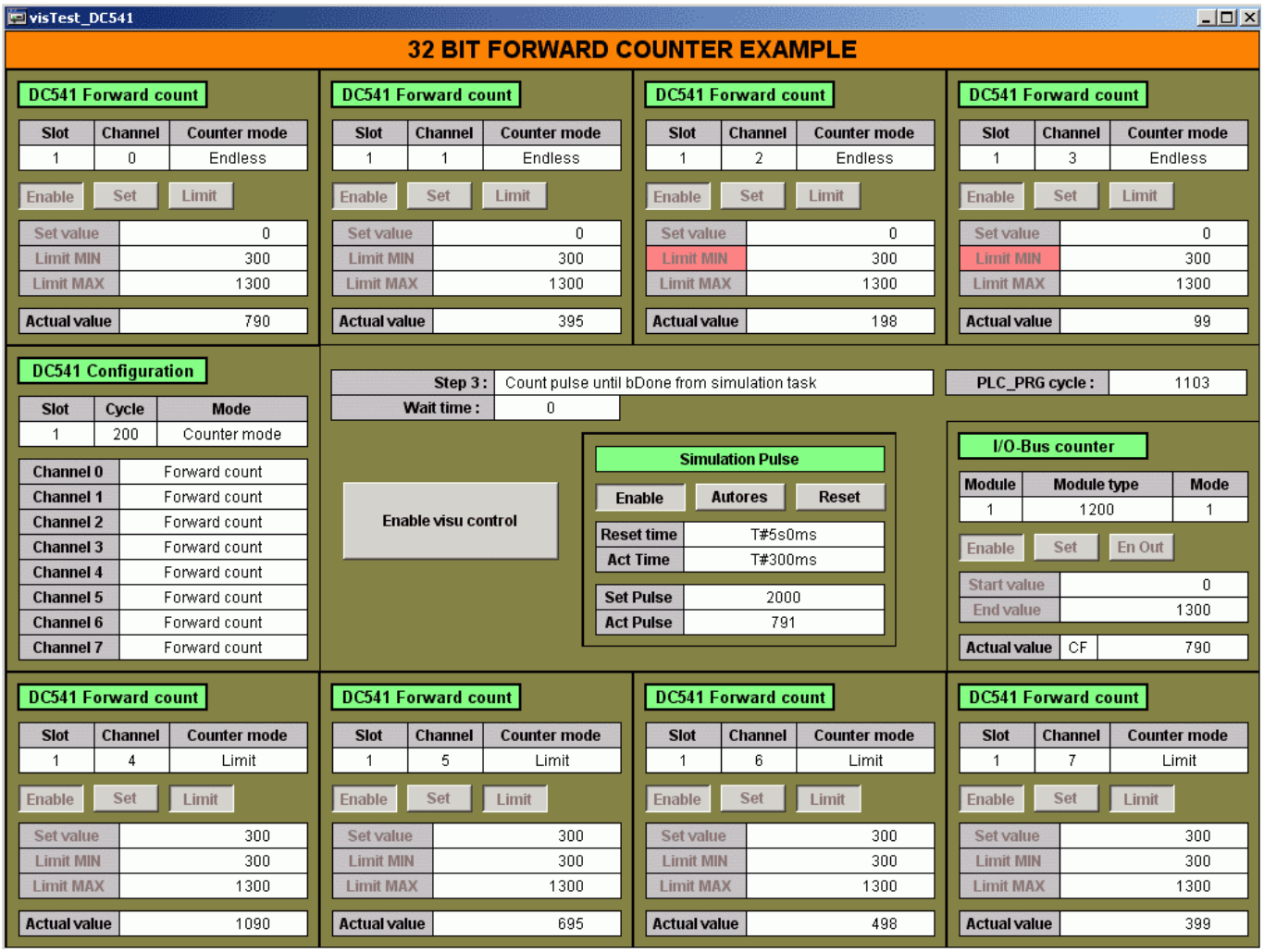
Instance		**fbSimuPulse**	**Instance name**
Ben	Input/Output	BOOL	Enabling of the pulse output
bReset	Input/Output	BOOL	TRUE = Reset of the pulse counter, bDone = TRUE

Instance		fbSimuPulse	Instance name
bAutoReset	Input/Output	BOOL	TRUE and cyclic mode - The time tResetTime is started when the number of pulses set with dwPulse is reached. After this time, the pulse output is restarted again.
tResetTime	Input/Output	TIME	Wait time until restart, if bAutoReset = TRUE
dwPulse	Input/Output	DWORD	Number of pulses to be output: = 0: Endless mode (pulse output continues until bEn = FALSE or bReset = TRUE) > 0: Cyclic mode (output of the specified number of pulses)
Bdone	Output	BOOL	Completion message after the number of pulses specified at dwPulse or after bReset if dwPulse = 0
bToggle_0	Output	BOOL	Output: Edge with each clock cycle
bToggle_1	Output	BOOL	Output: Edge with each 2. clock cycle
bToggle_2	Output	BOOL	Output: Edge with each 4. clock cycle
bToggle_3	Output	BOOL	Output: Edge with each 8. clock cycle
dwActNumPulse	Output	DWORD	Number of pulses output
tActTime	Output	TIME	Elapsed time in [ms] while tResetTime is running

In the example, the block Simu_Pulse is called in a 5 ms task. The pulse output is enabled or stopped via input bEn. If input dwPulse = 0, the output of pulses is performed continuously. If dwPulse > 0, only the specified number of pulses is output. When the specified number of pulses is reached, output bDone is set to TRUE.

In the example, the block is called with dwPulse = 2000. The wait time function is not used.

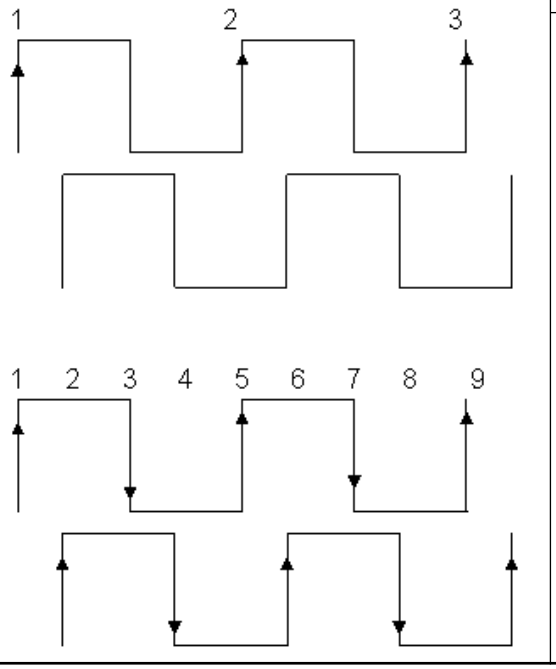
The example program has a visualization implemented that displays all states:



Clicking on the button <Enable visu control> (bEnVisuControl = TRUE) causes the program to jump from the current step to step 200 (manual operation). Then, the operation of the blocks is done via the corresponding buttons/switches of the individual blocks. When manual operation is switched off again (bEnVisuControl = FALSE), the program jumps to step 249 and restarts from step 0 after the wait time.

Usage as X4 counting device

The function module DC541-CM supports an encoder counting mode with channels A/B. This counting mode usually just supports to count 1 edge out of possible 4 edges from the 2 encoder lines. These are connected to C0 and C1 on DC541-CM.

	Counting in X1-mode Counting in X4-mode

For increasing the accuracy, it is possible to use a special mode with counting all 4 edges. This mode just supports a frequency up to 4 kHz. Despite of counting 4 edges on a lower frequency, the functionality is the same as 32-bit encoder on channel 0 and 1 and also the same function block ↳ *Chapter 1.5.4.11.1.1.1 "DC541_32BIT_CNT" on page 1103* has to be used.

Automation Builder configuration

For use of DC541-CM as a X4 counting device, the operating mode of the device has to be configured:

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 Encoder X4 mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 Counter mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

The channels C0..C3 are to be used for the encoder mode. In the module parameters, the channels C4..C7 can be configured as:

Input	-Input
Output	-Output

To do so, select the corresponding value for each channel.

> *The channels C1..C3 are automatically used, too. In the PLC configuration, these channels are left at the default setting "Input".*

The specified configuration can be read using the function block ↳ *Chapter 1.5.4.11.1.1.6 "DC541_GET_CFG" on page 1132*.

Usage for pulse width modulation

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 IO mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The pulse width modulation functionality of the DC541 is provided by the block ↳ *Chapter 1.5.4.11.1.1.10 "DC541_PWM" on page 1146*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 "DC541 library" on page 1103*.

The function block DC541_PWM outputs a pulsed signal with an adjustable on-off ratio. The on and off times are adjusted as 8 bit numbers.

The minimum switching time is specified at input CYCLE, i.e. if an output has been switched to FALSE or TRUE by the PWM, this output remains in this state for at least this time (CYCLE µs).

The minimum time specified at input CYCLE must not be smaller than the cycle time of the device DC541. Depending on its configuration, the cycle time of the DC541 can be 50, 100 or 200 µs. The cycle time can be polled using the function block ↳ *Chapter 1.5.4.11.1.1.6 "DC541_GET_CFG" on page 1132* (output CYCLE).

Configuration example: Pulse width modulation (PWM)

In the configuration example, channel 0 of the DC541 is configured for pulse width modulation (PWM). The output signal is measured using the function Time and frequency measurement ↳ *Chapter 1.6.5.2.8.1.6 “Usage for time and frequency measurement” on page 6004* of the DC541-CM.

The following on-off ratio shall be used:

PULSE	PAUSE	CYCLE	Result (x = number of cycles of the DC541)
Cycle time of DC541 = 100 µs			
1	2	2000	20 x TRUE / 40 x FALSE / 20 x TRUE / 40 x FALSE / … i.e. 2000 µs = TRUE and 4000 µs = FALSE

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541 in communication module slot 1 (first slot on the left of the CPU)
- PM591-ETH
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 ---------- DC541 / C1

PLC configuration:

- DC541-CM in slot 1, operating mode "counter mode"			
- Configuration:	- Channel	C0	PWM
		C1	FREQ
		C2...C7	Input
- Specification of the Ethernet communication module as internal communication module (if available)			

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#1ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

The cyclic program PLC_PRG contains the following functions:

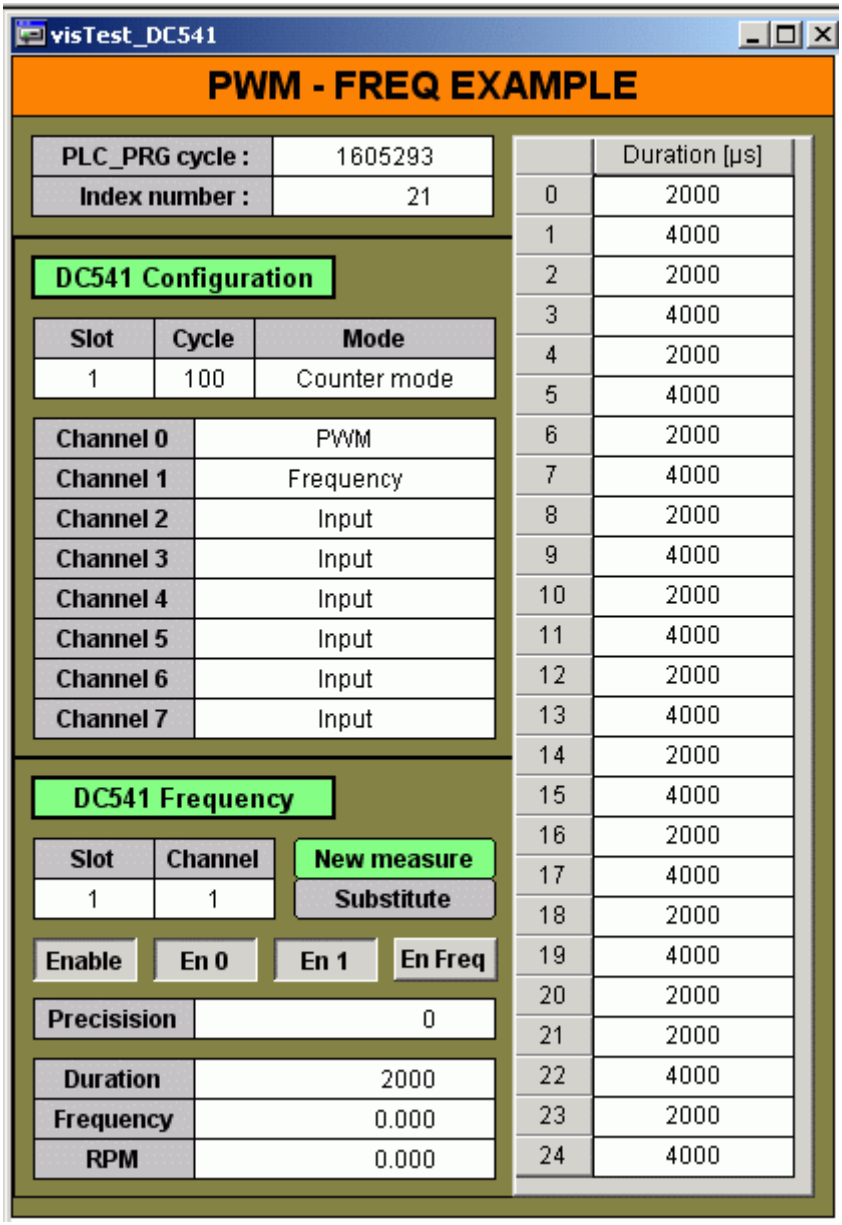
- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for PWM and FREQ Calling of program proPWM_FREQ

The blocks DC541_GET_CFG, DC541_STATE and DC541_IO are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation ↳ *Chapter 1.5.4.11 “DC541 library” on page 1103*.

The block TASK_INFO is contained in the library SysInt_AC500_V1.0 ↳ *Chapter 1.5.4.19 “Internal system library” on page 1500* and described in detail in the corresponding documentation.

Calling the pulse width modulation functionality as well as measurement and acquisition of measured values are performed in the program proPWM_FREQ. The program proPWM_FREQ contains the calls for the function blocks *Chapter 1.5.4.11.1.1.10 “DC541_PWM” on page 1146* and *Chapter 1.5.4.11.1.1.2 “DC541_FREQ” on page 1111* as well as the acquisition of the measured values. The function block DC541_FREQ is configured in a way that it measures the time between each edge change.

The example program has a visualization implemented that displays all states:



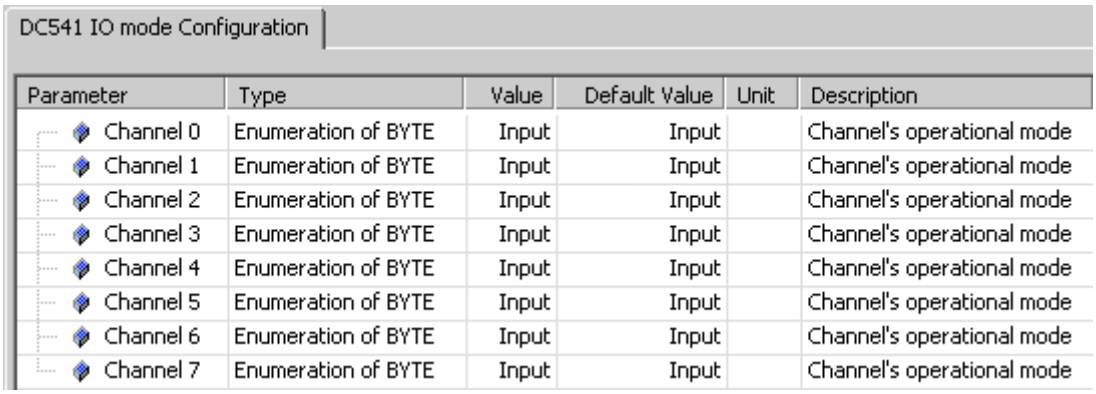
Input EN_VISU of the function block DC541_FREQ is TRUE. Therefore, the inputs of the block can be modified using the buttons <Enable>, <En 0>, <En 1> and <En Freq> in the visualization.

The measured values are 2000, 4000 or 6000 µs depending on which edges were considered for measurement.

Usage for time and frequency measurement

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:



Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The time and frequency measurement functionality of the DC541-CM is provided by the block ↳ *Chapter 1.5.4.11.1.1.2 "DC541_FREQ" on page 1111*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 "DC541 library" on page 1103*.

The function block DC541_FREQ is used to measure times, frequencies and rotational speeds with a resolution of 100 µs.

It is able to measure frequencies from 0 to 2000 Hz (2 kHz). In order to obtain a precise measurement of frequencies > 50 Hz, a correspondingly high accuracy setting has to be chosen. It is recommended to use an accuracy of PREC = 1000, i.e. 0.001.

This function block has to be called cyclically, one time per second at least.

The inputs EN_0, EN_1 and EN_FREQ are used to determine the edges to be measured. If input EN_FREQ = TRUE, the frequency and the rotational speed are calculated in addition to the time measurement.

Configuration example: Frequency output

In the configuration example, channel 0 of the DC541-CM is configured for frequency output ↳ *Chapter 1.6.5.2.8.1.7 "Usage for frequency output" on page 6007*. The output signal is measured using the function "Time and frequency measurement" of the DC541-CM.

Hardware configuration:

The example control system shall have the following configuration:

- Terminal base TB521 (two communication module slots)
- DC541-CM in Communication Module slot 1 (first slot on the left of the CPU)
- PM591-ETH CPU with internal Ethernet communication module
- I/O module DC532 on the I/O bus

Wiring:

The channels are connected as follows:

DC541 / C0 ---------- DC541 / C1

PLC configuration:

DC541-CM in slot 1, operating mode "counter mode"			
Configuration	**Channel**	**C0**	**Frequency output**
		C1	Frequency measurement
		C2...C7	Input
Specification of the Ethernet communication module as internal communication module (if available)			

Task configuration:

- Task 1: Cyclic program / Prio = 10 / Interval = t#5ms / PLC_PRG

Purpose of the cyclic program PLC_PRG:

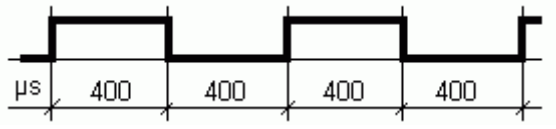
The cyclic program PLC_PRG contains the following functions:

- Reading the cycle of PLC_PRG Calling of block TASK_INFO;
- Reading the configuration of the DC541 Calling of block DC541_GET_CFG
- Reading the status of the DC541 Calling of block DC541_STATE
- Reading/writing the static channels of the DC541 Calling of block DC541_IO
- Calling of the sequence control for frequency output and measurement Calling of program proFrequency
-

The blocks *Chapter 1.5.4.11.1.1.6 “DC541_GET_CFG” on page 1132*, *Chapter 1.5.4.11.1.1.11 “DC541_STATE” on page 1151* and *Chapter 1.5.4.11.1.1.8 “DC541_IO” on page 1139* are contained in the library DC541_AC500_V11.lib and described in detail in the library documentation *Chapter 1.5.4.11 “DC541 library” on page 1103*.

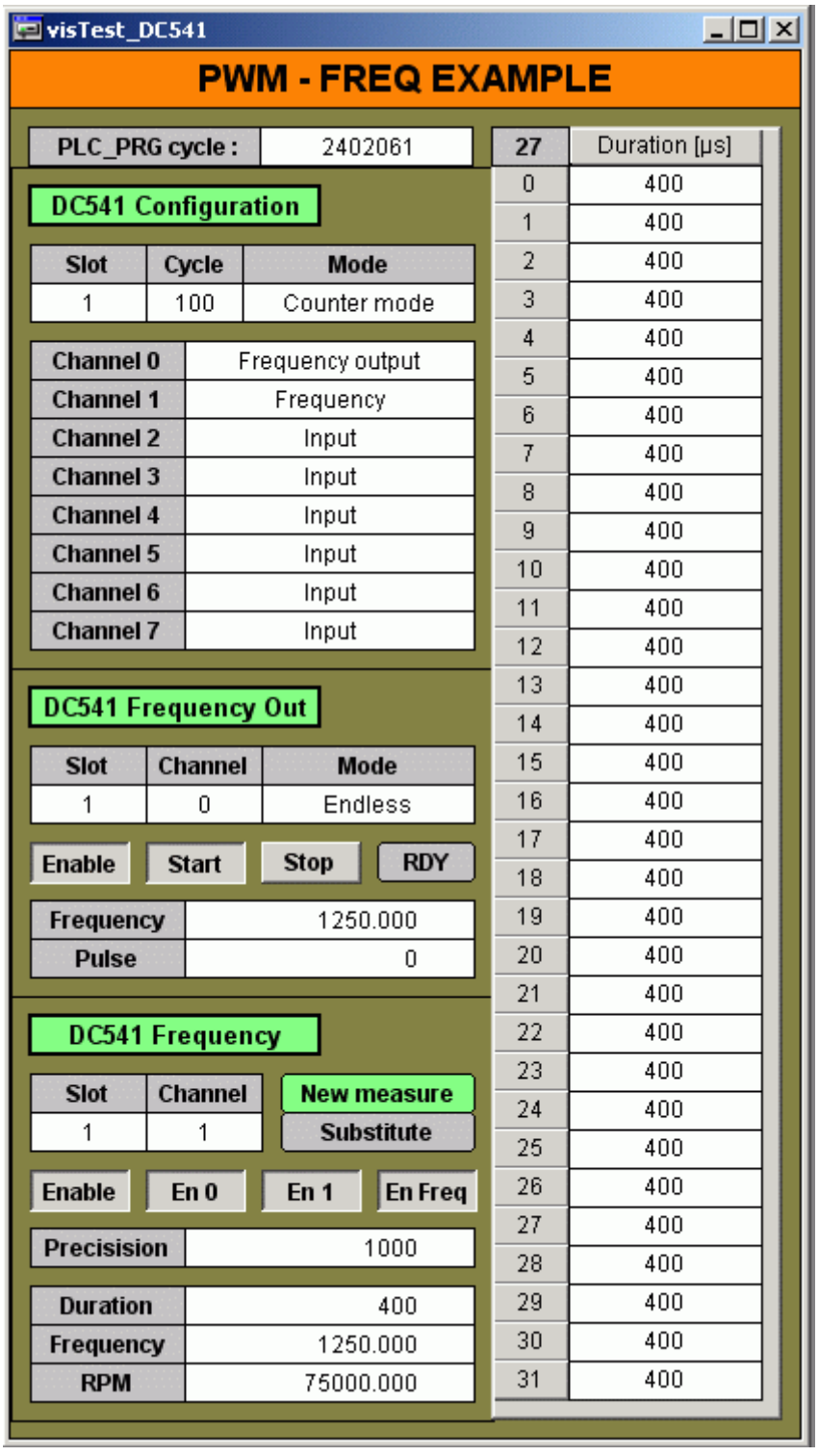
The block *Chapter 1.5.4.19.3.22 “TASK_INFO read number of completed task cycles” on page 1620* is contained in the library SysInt_AC500_V1.0 *Chapter 1.5.4.19 “Internal system library” on page 1500* and described in detail in the corresponding documentation.

The calling of the frequency output functionality as well as the measurement and acquisition of measured values are performed in the program proFrequency. The program proFrequency contains the calls for the function blocks *Chapter 1.5.4.11.1.1.4 “DC541_FREQ_OUT” on page 1123* and *Chapter 1.5.4.11.1.1.2 “DC541_FREQ” on page 1111* as well as the acquisition of the measured values.



The example program has a visualization implemented that displays all states:

Input EN_VISU of the function blocks DC541_FREQ_OUT and DC541_FREQ is TRUE. Therefore, the block inputs can be controlled using the buttons in the visualization.


visTest_DC541
PWM - FREQ EXAMPLE
PLC_PRG cycle :
2402061
DC541 Configuration
Slot
Cycle
Mode
1
100
Counter mode
Channel 0
Frequency output
Channel 1
Frequency
Channel 2
Input
Channel 3
Input
Channel 4
Input
Channel 5
Input
Channel 6
Input
Channel 7
Input
DC541 Frequency Out
Slot
Channel
Mode
1
0
Endless
Enable
Start
Stop
RDY
Frequency
1250.000
Pulse
0
DC541 Frequency
Slot
Channel
New measure
1
1
Substitute
Enable
En 0
En 1
En Freq
Precisision
1000
Duration
400
Frequency
1250.000
RPM
75000.000
27
Duration [µs]
0 400
1 400
2 400
3 400
4 400
5 400
6 400
7 400
8 400
9 400
10 400
11 400
12 400
13 400
14 400
15 400
16 400
17 400
18 400
19 400
20 400
21 400
22 400
23 400
24 400
25 400
26 400
27 400
28 400
29 400
30 400
31 400

Usage for frequency output

Automation Builder configuration

1. In the device tree, add a new object to the *"DC541-CM"* node and select *"DC541 IO mode"* from the list.
2. Double-click the added object and configure the I/O channels:

DC541 IO mode Configuration

Parameter	Type	Value	Default Value	Unit	Description
Channel 0	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 1	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 2	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 3	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 4	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 5	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 6	Enumeration of BYTE	Input	Input		Channel's operational mode
Channel 7	Enumeration of BYTE	Input	Input		Channel's operational mode

- Input
- Output
- Interrupt input

In the module parameters, you can specify the channels C0...C7 as inputs, outputs or interrupt inputs.

Calling the function blocks

The frequency output functionality of the DC541 is provided by the block ↳ *Chapter 1.5.4.11.1.1.4 "DC541_FREQ_OUT" on page 1123*.

Function blocks for the most module functions of DC541-CM are contained in the library DC541_AC500_V11.lib. The library is automatically included into the project and is described in detail in the library documentation ↳ *Chapter 1.5.4.11 "DC541 library" on page 1103*.

The function block DC541_FREQ_OUT is used to output pulses with a fixed frequency on one channel of the device DC541. It is able to output pulses with a frequency between 0.2 and 2.5 kHz. The pulse jitter depends on the cycle time of the DC541. The pulse length is always a multiple of the cycle time of the DC541.

In case of a presetting of PULSE = 0, the output of pulses is infinite. The pulse output is started with a positive edge at input START. The output is aborted if START = FALSE. A positive edge at input STOP interrupts the pulse output. The output is continued if STOP = FALSE.

If input PULSE > 0, the function block outputs the number of pulses specified at input PULSE with the frequency specified at input FREQ on the channel specified at input CH. After the function block has output the number of pulses specified at PULSE, the output RDY becomes TRUE.

The device DC541 must be configured as counting device (counter mode). Channel CH must be configured for frequency output.

The function block has an integrated visualization which can be used to control all function block functions in parallel to the user program, if input EN_VISU = TRUE.

For frequency output, the same configuration example is used as for the time and frequency measurement ↳ *Chapter 1.6.5.2.8.1.6 "Usage for time and frequency measurement" on page 6004*.

CD522 encoder and PWM module

Functionality of the CD522 module

The encoder and PWM module CD522 can be used at the following devices:

- Communication interface modules (e. g. CI501-PNIO, CI541-DP)
- Processor modules

Features:

- 2 independent counting functions with up to 12 configurable modes (including incremental position encoder and frequency input up to 300 kHz)
- 2 independent PWM (pulse-width modulator) or pulse outputs with push-pull driver
- Dedicated inputs/outputs for specific counting functions (e.g. touch, set, reset)
- All unused inputs/outputs can be used with the specifications of standard inputs/outputs range

For use in extreme ambient conditions (e.g. wider temperature and humidity range), a special XC version of the device is available.

Depending on the configuration used, some inputs and outputs are dedicated to specific counting functions (touch, set, reset…). All unused inputs and outputs can be used with the specification of standard inputs/outputs range.

There are special function blocks available to manage and control the function of the CD522 Module. These function blocks are contained in the ↬ *Chapter 1.5.4.8 “CD522 library” on page 972* which is available with a runtime system of version V1.0.2 or above. The library is automatically included into the project when adding a CD522 Module to the Automation Builder project. Details on the hardware is provided in the device descriptions ↬ *Chapter 1.6.2.7.2.1 “CD522 - Encoder, counter and PWM module” on page 4635*.

Special features

The specific functionality is processed within CD522. It works independently of the user program and therefore it is able to response quickly to external signals. A simultaneous counting operation of several expansion modules is possible.

Each module counter can be configured for 1 mode out of 12 possible ones. The desired operating mode is selected in the PLC configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. While integrating a module containing counters in the PLC configuration, the necessary operands are created and reserved immediately. Thus, a counter implementation carried out later does not cause an address shift.

Operating modes

Inputs and outputs, which are not used by the counters, are available for other tasks. In the following table, A means Input Channel A, B means Input Channel B and Z means Output Channel Z.

Operating Mode	Function	Used inputs	Description
0	No counter	None	This operating mode is selected, if the integrated fast counter is not needed.
1	Up/Down counter (A)	A = Counting input	One Up/Down (dynamic changes) counter with set and reset input, end value reached indicator, touch/catch value and overflow flag.
2	Up/Down with release input (B)	A = Counting input B = Enable input	One count up counter with enable input via terminal, counting is valid when input B is true. Dynamic Up/Down count possibility, end value reached indicator, Touch/catch value and Overflow flag
3	Up/Down counters (A,B)	A = Counting input 0 B = Counting input 1	2 counters with separate Up/Down and reset input
4	Up/Down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 counters (counting on falling edge of input B) with separate Up/Down and reset input
5	Up/Down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical input.
6	Up/Down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical input.
7	Reserved	None	---
8	Up/Down with release (B), 0 cross detection	A = Counting input B = Enable input	One 16 bit counter (in range of -32768 to 32767) with zero cross over detection, counting valid when input B is true

Operating Mode	Function	Used inputs	Description
9	Reserved	None	---
10	Reserved	None	---
11	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for encoder x1 count, touch/catch value, RPI function, reset and set
12	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x2 count, touch/catch value, RPI function, reset and set
13	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x4 count, touch/catch value, RPI function, reset and set
14	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface
15	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation

CD522 configuration

CD522 on I/O bus

1. In the device tree, add a new object to the *"I/O-Bus"* node.
2. From the list, select *"S500 I/O modules ➔ Function modules"* and add the CD522 device.

A maximum of 10 CD522 modules can be appended to the I/O bus.

3. Double-click the added object and configure the parameters ↳ *Chapter 1.6.5.2.8.2.3.3 "Parameterization" on page 6011*.

CD522 Modules connected to the I/O bus occupy the address area %IB0 .. %IB999 or %QB0 .. %QB999.

There is no fix assignment between module number and the input/output addresses of the channels.

CD522 on CS31-bus

1. In the device tree, add a new object under *"Interfaces"* to COM1 or COM2.
2. From the list, select a CS31 device e.g. *"DC551-CS31"*.
3. Add a new object to the *"DC551-CS31"* node.
4. From the list, select *"S500 I/O modules ➔ Function modules"* and add the CD522 device.
5. Double-click the added object and configure the parameters ↳ *Chapter 1.6.5.2.8.2.3.3 "Parameterization" on page 6011*.

CD522 modules connected to the CS31-Bus occupy the address area %IB1000 .. %IB1999 or %QB1000 .. %QB1999.

There is no fix assignment between module number and the input/output addresses of the channels.

A maximum of 4 CD522 modules can be appended to the CS31-Bus. If the DC551-CS31 Module is configured with the fast counter function, the maximum number of CD522 Modules is decreased to 3 modules (without other I/O Modules).

Automation Builder does not check the validity/integrity (according to the I/O limitations) of the number of expansion modules configured. The check is only done during project compilation.

Parameterization

The Encoder and PWM Module CD522 does not store configuration data itself. The digital inputs and outputs are used to specific counting functions depending to configuration mode chosen. All unused inputs and outputs can be used with the specification of standard input/ output range.

For non-standard applications, it is necessary to adapt the parameters to your system configuration:

Name	Type	Value	Default	Internal Value	Min.	Max.	EDS Slot Index
Ignore module *)	BYTE	No Yes	No 0x00	0 1			Not for FBP
Check supply	BYTE	Off On	On 0x01	0 1			0x0Y03
Input delay	BYTE	0.1 ms 1 ms 8 ms 32 ms	8 ms 0x02	0 1 2 3	0	3	0x0Y04
Mode counter 0	BYTE	see table below	0x00	0	0	15	0x0Y05

Name	Type	Value	Default	Internal Value	Min.	Max.	EDS Slot Index
Freq limit FC0	BYTE	No filter 50 Hz 500 Hz 5 kHz 20 kHz	No filter 0x00	0 1 2 3 4	0	4	0x0Y06
Input level FC0	BYTE	0-24 V DC 0-5 V DC Differen-tial 1 Vpp sinus	0-24 V DC 0x00	0 1 2 3	0	3	0X0Y07
SSI 0 fre-quency	BYTE	200 kHz 500 kHz 1 MHz	200 kHz 0x02	2 3 4	0	4	0x0Y08
SSI 0 res-olution (in Bit)	BYTE	8 to 32 bit	16 bit 16		8	32	0x0Y09
SSI 0 code type	BYTE	Binary	Binary 0	0	0	0	0x0Y0A
SSI 0 polling time	BYTE	10 ms	10		1	255	0x0Y0B
5 V sensor 0 supply	BYTE	Off On	Off 0x00	0	0	1	0x0Y0C
Mode counter 1	BYTE	see table below	0x00	0	0	15	0x0Y0D
Freq limit FC1	BYTE	No filter 50 Hz 500 Hz 5 kHz 20 kHz	No filter 0x00	0 1 2 3 4	0	4	0x0Y0E
Input level FC1	BYTE	0-24 V DC 0-5 V DC Differen-tial 1 Vpp sinus	0-24 V DC 0x00	0 1 2 3	0	3	0X0Y0F
SSI 1 fre-quency	BYTE	200 kHz 500 kHz 1 MHz	200 kHz 0x02	2 3 4	2	4	0x0Y10
SSI 1 res-olution (in Bit)	BYTE	8 to 32 bit	16 bit 16		8	32	0x0Y11

Name	Type	Value	Default	Internal Value	Min.	Max.	EDS Slot Index
SSI 1 code type	BYTE	Binary	Binary 0	0	0	0	0x0Y12
SSI 1 polling time	BYTE	10 ms	10		1	255	0x0Y13
5 V sensor 1 supply	BYTE	Off On	Off 0x00	0	0	1	0x0Y14
Detection SC and sensors	BYTE	Off On	Off 0x00	0	0	1	0x0Y15
Behaviour outputs at comm. error	BYTE	Off Last value Substitute Last value 5s Substitute 5s Last value 10s Substitute 10s	Off 0x00	0 1 2 3 4 5 6	0	1	0x0Y16
Substitute value	WORD	0	Default 0x0000	0	0	65536	0x0Y17

*) Not with FBP

Table 731: Operating modes for counters 0 and 1:

Internal value	Operating modes of counter
0	No counter / No PWM (default value)
1	1-1 UpDown counter (A)
2	2-1 UpDown with release input
3	3-2 UpDown counters (A, B)
4	4-2 UpDown (A, B on falling edges)
5	5-1 UpDown dynamic set (B) / rising edge
6	6-1 UpDown dynamic set (B) / falling edge
7	Not used
8	8-1 UpDown with release (B), 0 cross detection
9 - 19	Not used
20	11-1 Incremental encoder
21	12-2 Incremental encoder X2
22	13-1 Incremental encoder X4
30	14-1 SSI, absolute encoder
40	15-1 Time frequency meter

Operands

Operating modes

Inputs and outputs, which are not used by the counters, are available for other tasks. In the following table, A means Input Channel A, B means Input Channel B and Z means Output Channel Z.

Operating Mode	Function	Used inputs	Description
0	No counter	None	This operating mode is selected, if the integrated fast counter is not needed.
1	Up/Down counter (A)	A = Counting input	One Up/Down (dynamic changes) counter with set and reset input, end value reached indicator, touch/catch value and overflow flag.
2	Up/Down with release input (B)	A = Counting input B = Enable input	One count up counter with enable input via terminal, counting is valid when input B is true. Dynamic Up/Down count possibility, end value reached indicator, Touch/catch value and Overflow flag
3	Up/Down counters (A,B)	A = Counting input 0 B = Counting input 1	2 counters with separate Up/Down and reset input
4	Up/Down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 counters (counting on falling edge of input B) with separate Up/Down and reset input
5	Up/Down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical input.
6	Up/Down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	One Up/Down counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical input.

Operating Mode	Function	Used inputs	Description
7	Reserved	None	---
8	Up/Down with release (B), 0 cross detection	A = Counting input B = Enable input	One 16 bit counter (in range of -32768 to 32767) with zero cross over detection, counting valid when input B is true
9	Reserved	None	---
10	Reserved	None	---
11	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for encoder x1 count, touch/catch value, RPI function, reset and set
12	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x2 count, touch/catch value, RPI function, reset and set
13	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	One Up/Down counter for position sensor x4 count, touch/catch value, RPI function, reset and set
14	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface
15	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation

Operands for counting function

Input information for the counter	<	Output information for the user program
Counter settings 0	<	Output double word 0
High word	<	Output word 0.0
Low word	<	Output word 0.1
Control Byte 0	<	Output byte 0
Outputs counter 0	<	Output byte 1
Counter settings 1	<	Output double word 1

Input information for the counter	**<**	**Output information for the user program**
High word	<	Output word 2.0
Low word	<	Output word 2.1
Control Byte 1	<	Output byte 2
Outputs counter 1	<	Output byte 3

Input information for the counter:		
Counter settings 0	DWORD	Dependent on status bit 0 within output counter 0
		- SET value for the counter 0 If bit 0 = FALSE, the counter can be set to a start value. The start value is loaded into the counter by the user program using the SET signal.
		- END value for the counter 0 If bit 0 = TRUE, the end value for the counter is stored as comparison value into the module by the user program. The counter compares continuously whether the programmed end value is greater than or equal to its actual counter value or not. When the counter value reaches its programmed end value, the binary outputs C4 to C7 (output counter 0) can be set permanently.
		This value of the double word variable is loaded into the counter 0. Note: In mode 2 16 bit counters, the higher word is the value for counter A and the lower word is the value for counter B.
Counter settings 1	DWORD	Dependent on status bit 0 within output counter 1
		- SET value for the counter 1 If bit 0 = FALSE, the counter can be set to a start value. The start value is loaded into the counter by the user program using the SET signal.

Input information for the counter:		
		- END value for the counter 1 If bit 0 = TRUE, the end value for the counter is stored as comparison value into the module by the user program. The counter compares continuously whether the programmed end value is greater than or equal to its actual counter value or not. When the counter value reaches its programmed end value, the binary outputs C12 to C15 (output counter 1) can be set permanently.
		This value of the double word variable is loaded into the counter 0. Note: In mode 2 16 bit counters, the higher word is the value for counter A and the lower word is the value for counter B.
Control byte 0	BYTE	Control byte for the counter 0: Depending on operating modes, the different bits of Control byte 0 are used to manage and control the counting (see below table)
Control byte 1	BYTE	Control byte for the counter 1: Depending on operating modes, the different bits of Control byte 1 are used to manage and control the counting (see below table)
Output counter 0	BYTE	Outputs for the counters 0:
	Bit 0	Select SET/END value: FALSE= SET and TRUE = END
	Bit 1-3	Digital outputs unused
	Bit 4	Digital output C4, can be configured to indicate END value
	Bit 5	Digital output C5, can be configured to indicate END value
	Bit 6	Digital output C6, can be configured to indicate END value
	Bit 7	Digital output C7, can be configured to indicate END value
Output counter 1	BYTE	Outputs for the counters 1:
	Bit 0	Select SET/END value: FALSE= SET and TRUE = END
	Bit 1-3	Digital outputs unused

Input information for the counter:		
	Bit 4	Digital output C12, can be configured to indicate END value
	Bit 5	Digital output C13, can be configured to indicate END value
	Bit 6	Digital output C14, can be configured to indicate END value
	Bit 7	Digital output C15, can be configured to indicate END value

Control bytes (0 and 1) functions

	Used by operative modes:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Bit 0	EN : FALSE = counter disabled	X	X	X	X	X	X		X			X	X	X	X	X
	EN : TRUE = counter enabled															
Bit 1	SET : TRUE = set the counter A	X	X			X	X		X			X	X	X		
	or															
	EN_0 : TRUE = enable time capture on falling edge															X
	or															
	SET : TRUE = set end value (with bit 0 of output byte set to TRUE)														X	
Bit 2	RESET : TRUE = reset counter A	X	X	X	X	X	X		X			X	X	X		
	or															
	EN_1 : TRUE = enable time capture on rising edge															X
Bit 3	UP/DOWN : FALSE = up counter A; UP/ DOWN : TRUE = down counter A			X	X											
	or															
	FREQ : FALSE = time measure mode; FREQ : TRUE = frequency and rotation per minute (RPM) mode															X
Bit 4	RESET : TRUE = reset counter B			X	X											

	Used by operative modes:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
	or															
	RPI: reference point indicator											X	X	X		
	or															
	RESET_NEW : TRUE = time, frequency, rotation per minute (RPM) is reset and parameter output NEW is cleared															X
Bit 5	UP/DOWN : FALSE = up counter; UP/DOWN : TRUE = down counter	X	X			X	X		X							
	or															
	UP/DOWN : FALSE = up counter B; UP/DOWN : TRUE = down counter B			X	X											
Bit 6	EN_TOUCH : FALSE = no catch operation; EN_TOUCH : TRUE = enable next catch operation	X	X	X	X	X	X		X			X	X	X	X	
Bit 7	EDGE_TOUCH : FALSE = catch on falling edge; EDGE_TOUCH : TRUE = catch on rising edge	X	X	X	X	X	X		X			X	X	X	X	

Output information for the counter	**>**	**Input information for the user program**
32-bit counter 0	>	Input double word 0
High word	>	Input word 0.0
Low word	>	Input word 1
Touch counter value 0	>	Input double word 1
High word	>	Input word 1.0
Low word	>	Input word 1.1
State byte 0	>	Input byte 0
32-bit counter 1	>	Input double word 2
High word	>	Input word 2.0
Low word	>	Input word 2.1
Touch counter value 1	>	Input double word 3
High word	>	Input double word 3.0
Low word	>	Input double word 3.1

Output information for the counter	**>**	**Input information for the user program**
State byte 1	>	Input byte 2
Inputs counter 1	>	Input byte 3

Output information for the counter:		
32-bit counter 0	DWORD	Actual value of the counter 0
32-bit counter 1	DWORD	Actual value of the counter 1
Touch counter value 0	DWORD	Touch/catch value of the counter 0
Touch counter value 1	DWORD	Touch/catch value of the counter 1
State byte 0	BYTE	State byte for the counter 0: Depending to operating modes, the different bits of state byte 0 are used to display and control the counting (see below table)
State byte 1	BYTE	State byte for the counter 1: Depending to operating modes, the different bits of state byte 1 are used to display and control the counting (see below table)
Input counter 0	BYTE	Inputs for the counters 0:
	Bit 0	State corresponding to input A
	Bit 1	State corresponding to input B
	Bit 2	State corresponding to input Z
	Bit 3	State corresponding to input I3
	Bit 4	State corresponding to input I4
	Bit 5	State corresponding to input I5
	Bit 6	State corresponding to input I6
	Bit 7	State corresponding to input I7
Input counter 1	BYTE	Inputs for the counters 1:
	Bit 0	State corresponding to input A
	Bit 1	State corresponding to input B
	Bit 2	State corresponding to input Z
	Bit 3	State corresponding to input I11
	Bit 4	State corresponding to input I12

Output information for the counter:		
	Bit 5	State corresponding to input I13
	Bit 6	State corresponding to input I14
	Bit 7	State corresponding to input I15

Status bytes (0 and 1) functions

	Used by operative modes:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
Bit 0	CF : TRUE = when end value of counter is reached The CF bit is cleared if the counter was disabled or a Set/Reset operation was executed.	X	X			X	X					X	X	X	X	
	CF : TRUE = zero crossover indicator								X							
Bit 1	Not used	X	X		X	X	X		X			X	X	X		
Bit 2	RDY_TOUCH : TRUE = new catch/touch value available	X	X			X	X		X			X	X	X	X	
Bit 3	OVERFLOW counter : TRUE = overflow	X	X			X	X					X	X	X	X	
	or															
	OVERFLOW counter A : TRUE = overflow 0000H <--> FFFFH (65535)			X	X											
Bit 4	SET INPUT counter : TRUE = logical OR function on all inputs (I3 to I7 or I11 to I15) configured as SET input	X	X			X	X		X			X	X	X	X	
	or															
	OVERFLOW counter B : TRUE = overflow 0000H <--> FFFFH (65535)			X	X											
Bit 5	RESET INPUT counter : TRUE = logical OR function on all inputs (I3 to I7 or I11 to I15) configured as RESET input	X	X			X	X		X			X	X	X	X	
	or															
	RESET INPUT counter A : TRUE = logical OR function on all inputs (I3 to I7) configured as RESET input			X	X											
Bit 6	NEW : TRUE = new timing value is available															X
	or															

	Used by operative modes:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	RDY_RPI : TRUE = when the RPI operation is done											X	X	X		
Bit 7	RESET INPUT counter B : TRUE = logical OR function on all inputs (I11 to I15) configured as RESET input			X	X											

Operands for PWM/pulse function

Input information for the output	<	Output information for the user program
PWM frequency 0	<	Output word 0
PWM duty cycle/pulse C0	<	Output word 1
PWM frequency C1	<	Output word 2
PWM duty cycle/pulse C1	<	Output word 3
PWM Control Byte C0	<	Output byte 0
PWM Control Byte C1	<	Output byte 1
PWM/Pulse Outputs	<	Output byte 2
Bit 0	<	Output bit 2.0
Bit 4	<	Output bit 2.4

Input information for the PWM/Pulse function:		
PWM Frequency C0	WORD	Frequency of channel O0 Unit: Hz or 10Hz (depending on bit 0 of PWM control byte 0)
PWM Frequency C1	WORD	Frequency of channel O1 Unit: Hz or 10Hz (depending on bit 0 of PWM control byte 1)
PWM Duty cycle/pulse C0	WORD	PWM duty cycle of channel O0 in 1/10 of percentage or Number of pulses to be send on channel O0
PWM Duty cycle/pulse C1	WORD	PWM duty cycle of channel O1 in 1/10 of percentage or Number of pulses to be send on channel O1

Input information for the PWM/Pulse function:		
PWM control byte 0	BYTE	Control byte for the PWM/ Pulse output O0: Depending on operating modes, the different bits of PWM control byte 0 are used to manage and control the counting (see below table)
PWM control byte 1	BYTE	Control byte for the PWM/ Pulse output O1: Depending to operating modes, the different bits of PWM control byte 1 are used to manage and control the counting (see below table)
PWM/Pulse Outputs	BYTE	Outputs for PWM/Pulse output:
	Bit 0	Output O0
	Bit 1-3	Digital outputs unused
	Bit 4	Output O1
	Bit 5-7	Digital outputs unused

Control bytes (0 and 1) functions

Byte	**Description**
Bit 0	FALSE = frequency multiplier x1 is enable TRUE = frequency multiplier x10 is enable
Bit 1	Not used
Bit 2	Not used
Bit 3	Start pulse emission, if one rising edge => start pulse emission on channel O0 or O1
Bit 4	Not used
Bit 5	Not used
Bit 6	Not used
Bit 7	FALSE = PWM/Pulse function is disabled TRUE = PWM/Pulse function is enabled

Output information of the PWM/Pulse	**>**	**Input information for the user program**
State Byte S0 %pulse	>	Input byte 0
State Byte S1 %pulse	>	Input byte 1

Output information of the PWM/Pulse:		
State Byte S0 %pulse	BYTE	Percentage of pulses already sent on channel O0
State Byte S1 %pulse	BYTE	Percentage of pulses already sent on channel O1

CD522 used as encoder device

Incremental encoder

The function block *Chapter 1.5.4.8.1.1 “CD522_32BIT_ENCODER” on page 972* can be used to control an encoder device for relative positioning with the CD522 Module.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function encoder of module CD522, different operating modes are available. The function block CD522_32BIT_ENCODER should be used with one of these operating modes:

Operating Mode 11	"Incremental encoder"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x1 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 12	"Incremental encoder X2"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x2 count, with possibility of touch/catch value, RPI function, set and reset actions.	
Operating Mode 13	"Incremental encoder X4"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one bidirectional counter for position sensor x4 count, with possibility of touch/catch value, RPI function, set and reset actions.	

Absolute SSI encoder

The function block *Chapter 1.5.4.8.1.5 “CD522_SSI_CNT” on page 1006* can be used to control SSI absolute encoder function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_SSI_CNT should be used with one of these operating modes:

Operating Mode 14	"SSI, absolute encoder"
Should be specified in PLC Configuration; parameter mode counter in order to use absolute encoder with SSI interface.	

CD522 used as counter device

32-Bit bidirectional counter

The function block *Chapter 1.5.4.8.1.2 "CD522_32BIT_CNT" on page 982* can be used to control one 32-bit bidirectional counter function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_32BIT_CNT should be used with one of these operating modes:

Operating Mode 1	"Up/Down counter (A)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use one up/down 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 2	"Up/Down counter with release input (B)"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/catch value and overflow flag.	
Operating Mode 5	"Up/Down dynamic set (B)/rising edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge of physical.	
Operating Mode 6	"Up/Down dynamic set (B)/falling edge"
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 32-bit counter with set and reset input, end value reached indicator, touch/catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge of physical.	

The module CD522 provides 2 Up/Down 32-bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

16-Bit bidirectional counter

The function block *Chapter 1.5.4.8.1.3 "CD522_16BIT_CNT" on page 991* can be used to control one 16-bit bidirectional counter function.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_CNT should be used with one of these operating modes:

Operating Mode 8	**"Up/Down with release (B), 0 cross detection"**
Should be specified in PLC Configuration; parameter "mode counter" in order to use one up/down 16 bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	

The module CD522 provides 2 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 for counter 0 and A1 for counter 1. Another signal used for enable or dynamic set is identified by B0 for counter 0 and B1 for counter 1.

Two 16-bit bidirectional counter

The function block *Chapter 1.5.4.8.1.4 “CD522_16BIT_2CNT” on page 998* can be used to control two 16-bit bidirectional counter functions.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_16BIT_2CNT should be used with one of these operating modes:

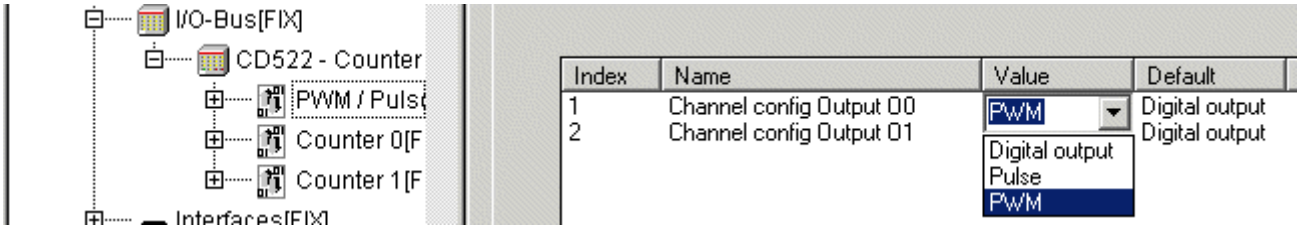
Operating Mode 3	"Up/Down counters (A,B)"
Should be specified in PLC Configuration, parameter "mode counter" in order to use 2 Up/Down 16 bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	
Operating Mode 4	"Up/Down (A, B on falling edges)"
Should be specified in PLC Configuration, parameter mode counter in order to use two Up/Down 16 bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and overflow flag.	

The module CD522 provides 4 Up/Down 16 bit counter functions. A signal used for pulse count is identified by A0 and B0 for counter A and A1 and B1 for counter B.

CD522 used as PWM output device

To use CD522 as PWM output device, function block *Chapter 1.5.4.8.1.6 “CD522_PWM_OUT” on page 1013* is required.

The module CD522 can be used to control one output pulsing signal (Max= 100 KHz) with an adjustable duty cycle (ON/OFF ratio, max=100%). The PWM operating mode is configured in Automation Builder.



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

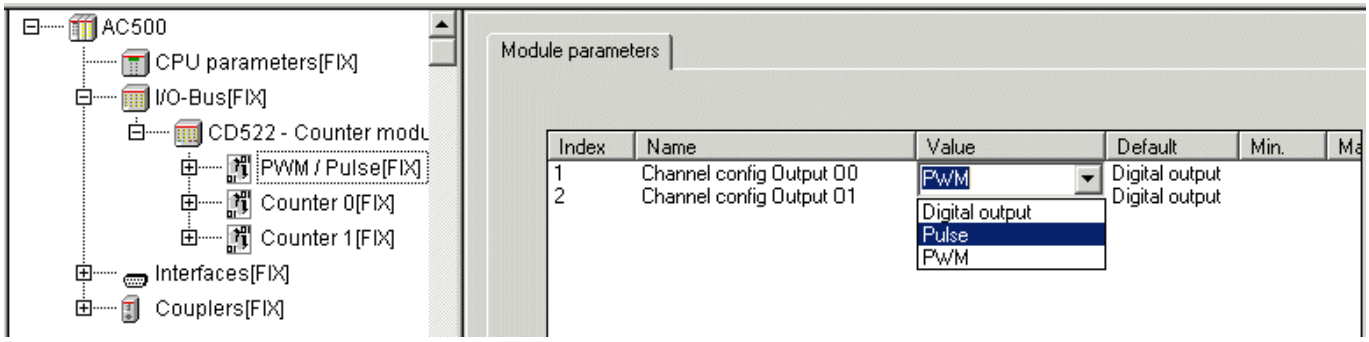
The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_PWM_OUT should be used to control with input EN_PWM, configure the frequency with input FREQ and the input duty cycle DUTY_CYCLE of PWM outputs (pulse-width modulator).

CD522 used as pulse output device

To use CD522 as pulse output device, function block *Chapter 1.5.4.8.1.7 “CD522_PULSE_OUT” on page 1016* is required.

The module CD522 can be used to control one output pulses signals with a fixed duty cycle (ON/OFF ratio 50 %) and number of pulses sent with a fixed frequency (can be modified) .The PULSE operating mode is configured in PLC Configuration using module parameters:



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed by using input and output operands. These necessary operands are created and reserved automatically, when one CD522 module is added into the I/O bus configuration.

The module CD522 provides two independents outputs used in PULSE mode (O0 and O1). Both have the same specification and can work separately.

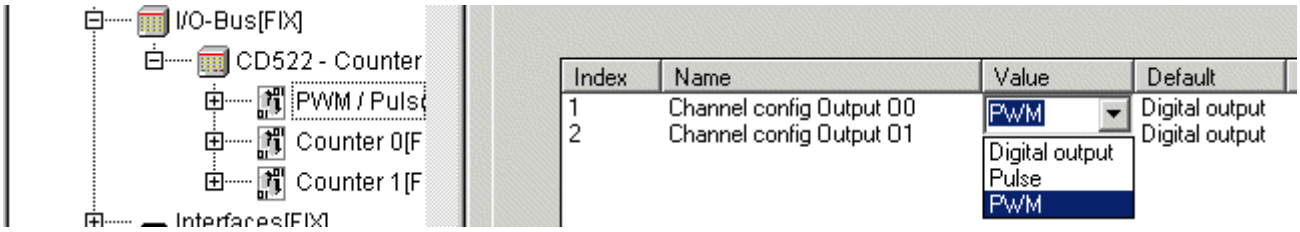
The function block CD522_PULSE_OUT should be used to control the pulse output, with input EN_FREQ, configure the frequency with input FREQ and the number of pulses with input NUM. The number of pulses sent can be displayed in percentage (from 0 % to 100%).

On the fast outputs O0 or O1, the brightness of yellow LED depends on the number of pulse emitted (from 0 and 100%), When the value 100% is obtained, the yellow LED status is off.

CD522 used as frequency output device

To use CD522 as frequency output device, function block *Chapter 1.5.4.8.1.8 “CD522_FREQ_OUT” on page 1020* is required.

The module CD522 can be used to control one output pulses signals with an fixed duty cycle (ON/OFF ratio 50 %).The PWM operating mode is configured in PLC Configuration using module parameters:



After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added in I/O bus configuration.

The module CD522 provides two independent outputs which can be used in PWM mode (O0 and O1). Both have the same specification and can work separately.

The function block CD522_FREQ_OUT should be used to control with input EN_FREQ and configure the frequency with input FREQ of frequency outputs (1 kHz to 100 kHz).

CD522 used as time frequency meter

To use CD522 to measure times, frequency and rotation speeds on channel Z0 or Z1, function block *Chapter 1.5.4.8.1.9 “CD522_FREQ_SCAN” on page 1024* is required.

The module CD522 can be used in 12 different configurable operating modes. The operating mode is configured in PLC Configuration using module parameters. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange from and to the user program is performed using input and output operands. These necessary operands are created and reserved automatically, when one module CD522 is added into the I/O bus configuration.

In order to configure and use the function counter of module CD522, different operating modes are available. The function block CD522_FREQ_SCAN should be used with one of these operating modes:

Operating Mode 15	**"Time frequency meter (Z)"**
Should be specified in PLC Configuration with the parameter mode counter.	

The module CD522 provides 2 channels (Z0 and Z1) which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs. Both have the same specification and can work separately.

The function block CD522_FREQ_SCAN should be used to control with input EN_CNT, configure the capture on falling edge with input EN_0 or rising edge with input EN_1 of signal, and the specification of the mode of the measurement (time, frequency and Rpm) with input EN_FREQ.

The table shows values measured according to configuration input parameters and this example of timing.

NOTICE!

Risk of malfunctions!

Never use the time measurement (bit EN_FREQ=FALSE) mode if the CD522 is connected to a CS31 communication interface module, e. g. CI592.

Depending on the input parameters of function block, the result of time measurement can be measured in time in µs, frequency in Hz or speed of rotation in rotation per minute.

FM502-CMS Function module

Condition monitoring

Components of the condition monitoring system:

- Hardware
  - Function module FM502-CMS for condition monitoring, protection or as precise I/O module *Chapter 1.6.2.7.2.2 “FM502-CMS - Analog measurements” on page 4658*.
  - Function module terminal base TF5x1-CMS *Chapter 1.6.2.2.2 “TF501-CMS and TF521-CMS - Function module terminal bases ” on page 3796*
  - Processor module PM592-ETH *Chapter 1.6.2.3.2.1 “PM57x (-y), PM58x (-y) and PM59x (-y)” on page 3848*
- Configuration of FM502-CMS *Chapter 1.6.5.2.8.3.2 “FM502-CMS function module” on page 6030*.
- FM502-CMS library: The FM502-CMS library contains function blocks to manage and control the function of the Function Module FM502-CMS. The FM502-CMS library consists of the WAV-File library and the CMS-IO library. Once a Function Module has been added to the configuration, the libraries are automatically included with the next compilation of the project. *Chapter 1.5.8 “FM502-CMS library” on page 2519*.

Introduction to condition monitoring

Condition Monitoring (CM) is a broad term, which can be understood in different ways. For the FM502-CMS, CM means the acquisition and analysis of high-frequency data. CM does usually not occur in real-time. The data is analysed afterwards. If, however, online CM is controller integrated, real-time reaction, e.g. protection, is possible within the same device and using the same sensors. This feature is supported by AC500 and FM502.

The focus is often merely on mechanical CM. This is due to the fact that the movement or rotations of large masses which are connected to a motor (e.g. electric machine) via a shaft pose the greatest danger. Machines in operation inevitably generate measurable vibration, both free or forced, even in the normal operating states and in absence of any damage.

Yet, in electrical CM, electrical high-frequency quantities like currents, voltages or partial discharges can be measured by suitable sensors and can be analyzed in order to detect electrical failure patterns, e.g. inside electrical machines or equipment (transformers). Some electrically measurable failure patters can be induced by mechanical issues e.g. vibration.

Effective fault detection will only be possible if the data patterns indicating an arising defect can be singled out among the data collected.

Monitoring e.g. the vibration characteristics of a machine in operation gives an understanding of the "'health" condition of the machine and its development over time and load. This information can be used to detect arising problems at an early stage. Operating a machine until it breaks down might be acceptable if the machine were a "'disposable" one and the lifetime was very high and known for sure. But many failures can be considered statistical outliers and occur very early and spontaneously.

However, most machines are not "disposable" due to their cost. Therefore, regular monitoring of a machine’s condition can reveal potential problems. Subsequently, counter measures can be taken at an early stage in order to minimize damage and associated cost. If monitoring is permanent and controller integrated, even spontaneous failures can be detected in real-time, in order to prevent substantial damage to a larger part of the equipment and its environment.

Use cases of FM502-CMS

- Condition monitoring: Longer data-stream analyses
- Protection: Fast reaction to e.g. direct or RMS values based on limits
- Fast and precise analog measurements as with any other AC500 I/O module, but even more precise and faster
- Data logging: Fast, efficient data storage

Condition Monitoring typically means acquisition of longer data streams, also called “time series” or “signals”. These can be analyzed after measurement in the time domain (e.g. envelope, statistical analysis) or frequency domain (e.g. spectrum analysis).

The Function Module FM502-CMS works independently of the user program and therefore it is able to response quickly to external signals and fast acquisition of the analog channel values.

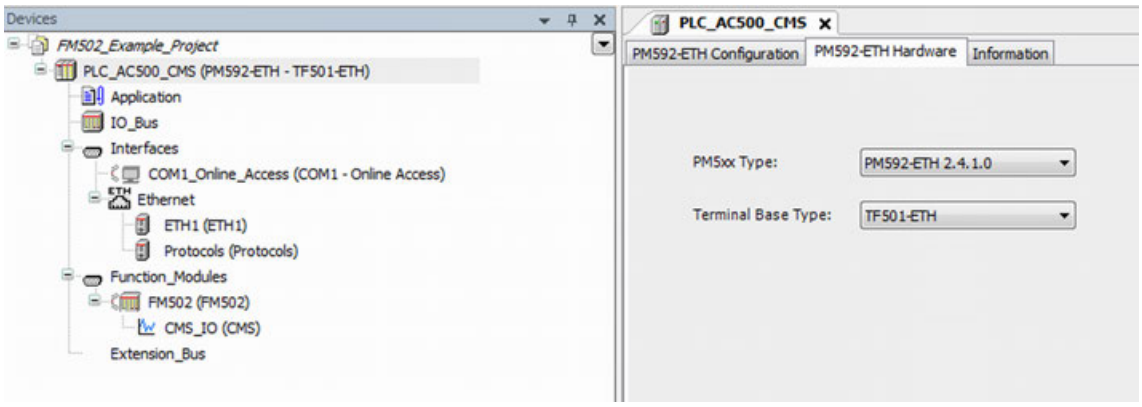
The analog channels can be configured in different modes. It is also possible to change individually the sample and acquiring settings for each channel. The counter can be configured in different modes. After that, it is activated during the initialization phase (power-on, cold start, warm start).

The data exchange to and from the user program is performed using input and output operands. While integrating a module containing counters in the PLC configuration, the necessary operands are created and reserved immediately. For the fast data acquisition the values are stored inside the module in a file and transferred via file transfer to the PLC after the measurement sequence is finished.

FM502-CMS function module

Preconditions The hardware structure is automatically generated in configuration.

You can change the type of the TF5x1-CMS in the hardware configuration of the processor module:



Only one FM502-CMS can be connected to a processor module.

> *FM502-CMS cannot be used as an I/O on a remote communication interface.*

Configuration of FM502-CMS

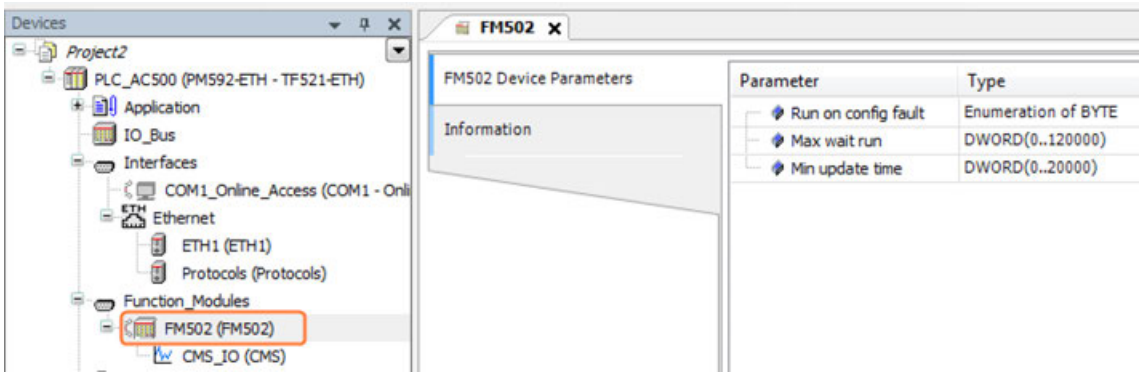


Fig. 1152: Change the behavior on the internal bus.

Parameter	Default value	Value	Description
Run on config fault	No	No	In case of a configuration error, the user program is not starting.
		Yes	The user program runs independent of a faulty I/O bus configuration
Max wait run	3000 [ms]	-	Maximum wait time for valid inputs (Do not change this parameter)
Min update time	10 [ms]	-	Cycle time for data exchange to IEC program The “Min update time” defines the time how often the I/O image will updated for the IEC program. If the IEC task is faster, the I/O image is not updated on every cycle.
Watchdog	400 [ms]	-	Watchdog time (Do not change this parameter)

Parameterization

The FM502-CMS does not store configuration data itself. The digital inputs and outputs can have specific functions depending on the selected configuration mode. All not otherwise configured inputs and outputs can be used with the specification of standard input/output range.

To change the parameterization you can choose the default parameterization in Automation Builder or the parameterization during run time in CODESYS.

The functionality of the module is directly influenced by the parameterization.

Parameterization - Automation Builder

For non-standard applications, it is necessary to adapt the parameters to your system configuration. After every startup of the device the default parameter set will be downloaded to the module.

The parameterization is divided in the following sections:

- Parameter for encoder/counter functionality
- Parameter for analog channel functionality
- Parameter for digital I/O configuration

Parameterization - CODESYS

In CODESYS for the parameterization you can use function blocks of FM502-CMS library ↳ *Chapter 1.5.8.2.1.9 “CMS_IO_CFG_WRITE” on page 2552*↳ *Chapter 1.5.8.2.1.8 “CMS_IO_CFG_READ” on page 2550*.

Parameter set

Table 732: Encoder/Counter

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
145	Mode	↳ *“Operation modes” on page 6034*	0-15	BYTE	0	0	15
146	Frequency limit	No filter	0	BYTE	0	0	4
		50 Hz	1				
		500 Hz	2				
		5 kHz	3				
		20 kHz	4				
147	Input level	0 - 24 V DC	0	BYTE	0	0	3
		0 - 5 V DC	1				
		Differential	2				
		1 Vss sinus	3				
148	SSI frequency	200 kHz	2	BYTE	2	2	
		500 kHz	3				
		1 MHz	4				
149	SSI resolution	8 Bit	0	BYTE	1	0	3
		16 Bit	1				
		24 Bit	2				
		32 Bit	3				
150	SSI code type	Binary input	0	BYTE	0	0	0
151	SSI polling time	x	x	BYTE	10 ms	0	255

Table 733: Analog channels

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
1	Available Channel	Disabled	0	BYTE	0	0	1
		Enabled	1				
2	Analog Mode	IEPE	0	BYTE	0	0	1
		+/- 10 V	1				
3	Synchronized encoder file	Disabled	0	BYTE	0	0	1
		Enabled	1				
4	DC Filter	Disabled	0	BYTE	0	0	1
		Enabled	1				
5	Sample rate	50 kHz	0	BYTE	0	0	9
		25 kHz	1				

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
		12,50 kHz	2				
		6,25 kHz	3				
		3,13 kHz	4				
		1,56 kHz	5				
		0,78 kHz	6				
		0,39 kHz	7				
		0,20 kHz	8				
		0,10 kHz	9				
		(Reserved)	10...15				
6	Start condition	Immediate	0	BYTE	0	0	4
		Delayed	1				
		Binary Input	2				
		Zero Input	3				
		Encoder Value	4				
7	Start condition value (dependend on Start condition)	x	x	DWORD	0	0	4294967296
		I1	0	-	0	0	3
		I2	1				
		C3	2				
		C4	3				
8	Edge type	Rising edge	0	BYTE	0	0	1
		Falling edge	1				
9	Record length value (samples)	x	x	DWORD	5000	1	4294967296

Table 734: I/O configuration: Inputs

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
152	Input delay	0.1 ms	0	BYTE	0	0	3
		1 ms	1				
		8 ms	2				
		32 ms	3				
153-156	DI0/DI1/DC2/DC3 use	Digital input	0	BYTE	-	0	5
		Touch	1				
		Reset	2				
		Reset 2nd Bit Counter	3				
		Set	4				
		RPI	5				

Table 735: I/O configuration: Outputs

No.	Name	Value	Internal value	Internal type	Default	Min.	Max.
157	Behavior on STOP	Off	0	BYTE	0	0	2
		Last value	1				
		Substitute value	2				
158	Substitute value 10 s	x	x	WORD	0	0	3
159/160	DC2/DC3 use	Digital output	0	BYTE	0	0	3
		Analog Channel failure	1				
		Module failure	2				
		End value	3				

Operation modes

Inputs and outputs which are not used by the counters are available for other tasks.

Table legend: A = input channel A, B = input channel B, Z = output channel Z.

Operation Mode	Function	Used inputs	Description	Function block
0-1	No counter	None	This operating mode is selected, if the integrated high-speed counter is not needed.	-
1-1	Up/down counter (A)	A = Counting input	1 bidirectional 32-bit counter on input A (dynamic changes) with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT
2-1	Up/down with release input (B)	A = Counting input B = Enable input	1 bidirectional 32-bit counter with enable input. Counting is valid when input B is TRUE. Dynamic up/down count possibility, with set and reset input operation, end value reached indicator, touch/ catch value and overflow flag.	CMS_IO_32BIT_CNT
3-2	Up/down counters (A,B)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter (on rising edge count) functions, with separate up/down, reset operation and overflow flag.	CMS_IO_16BIT_2CNT

Operation Mode	Function	Used inputs	Description	Function block
4-2	Up/down (A, B on falling edges)	A = Counting input 0 B = Counting input 1	2 bidirectional 16-bit counter functions (with A on rising edge count and B on falling edge count), With separate up/down, reset operation and over-flow flag.	CMS_IO_16BIT_2CNT
5-1	Up/down dynamic set (B) / rising edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on rising edge sets START_VALUE.	CMS_IO_32BIT_CNT
6-1	Up/down dynamic set (B) / falling edge	A = Counting input B = Dynamic set input	1 bidirectional 32-bit counter with set and reset input, end value reached indicator, touch/ catch value and overflow flag. Additional function to mode 1 is the dynamic set input (B) on falling edge sets START_VALUE.	CMS_IO_32BIT_CNT
7-1	Reserved	None	-	-
8-1	Up/down with release (B), 0 cross detection	A = Counting input B = Enable input	1 bidirectional 16-bit counter (in range of -32768 to 32767) with enable input and zero crossover detection (CF). Counting is valid when input B is TRUE. With set and reset input operation and touch/catch value.	CMS_IO_16BIT_CNT
9-1	Reserved	None	-	-
10-1	Reserved	None	-	-
11-1	Incremental encoder	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for encoder x1 count, touch/ catch value, RPI function, reset and set Function block counts rising edges at input A.	CMS_IO_32BIT_ENCODER
12-1	Incremental encoder X2	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x2 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A.	CMS_IO_32BIT_ENCODER

Operation Mode	Function	Used inputs	Description	Function block
13-1	Incremental encoder X4	A = Trace A of the encoder B = Trace B of the encoder Z = Trace Z of the encoder (mechanical zero)	1 bidirectional counter for position sensor x4 count, with possibility of touch/ catch value, RPI function, set and reset actions. Function block counts rising and falling edges at input A and B.	CMS_IO_32BIT_ENCODER
14-1	SSI, absolute encoder	A = Data signal B = Clock signal	Absolute positioning sensor using SSI interface	CMS_IO_SSI_CNT
15-1	Time frequency meter	Z = Input signal	Time measurement of Z signal, rising edge, falling edge, rotation per minute and frequency calculation	CMS_IO_FREQ_SCAN

Process image (I/O data)

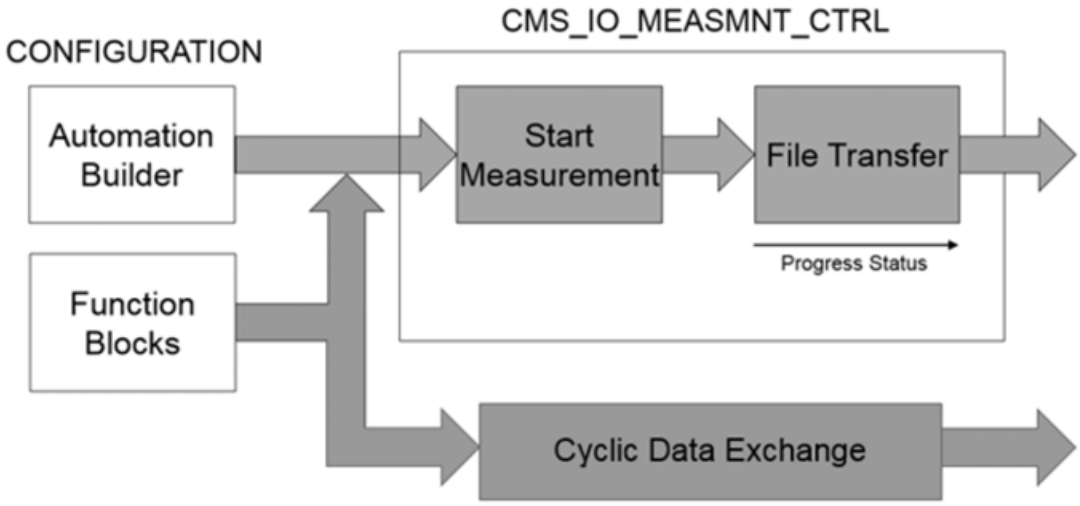
The cyclic data can be accessed to the addresses or variables defined in Automation Builder **I/O Mapping** tab.

FM502-CMS analog measurement

Possibilities to use the analog input signal values:

- The values of the configured analog channels in CODESYS can be used by referencing the I/O mapping variables. The refresh time for the cyclic data exchange of the analog input data is according to the minimum update time in the configuration.
- For detailed data analysis you can record the analog input data into WAV files and store them. In these files, every input data sample is stored.

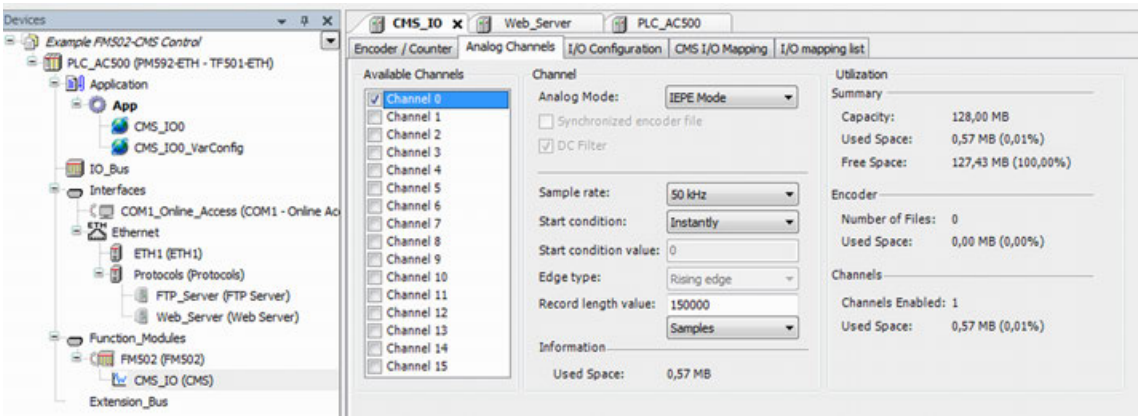
Both possibilities can be used simultaneously.



The DC filter needs 10 seconds to tune in after system power up. During that time the AI values can be accessed by the user program, but the readings are out of tolerance limit.

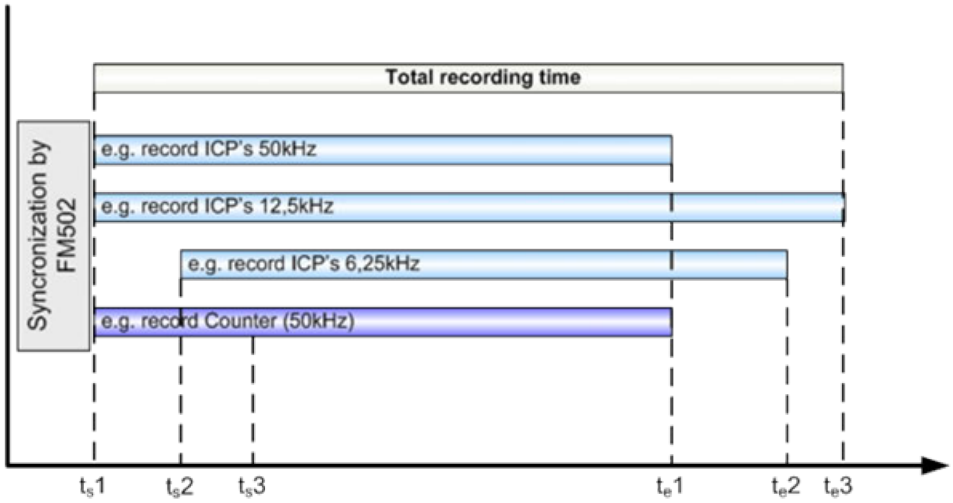
Configuration for analog measurements

▷ Configure at least one analog channel for data acquisition.



The measurement file size depends on the record length value and number of available channels. The maximum capacity of one measurement file is limited by the internal memory. The recording time for each channel is calculated by: recording time = record length value/sample rate.

The total recording time is determined by the earliest measurement start trigger of a channel to the measurement end of the last channel in the configuration.

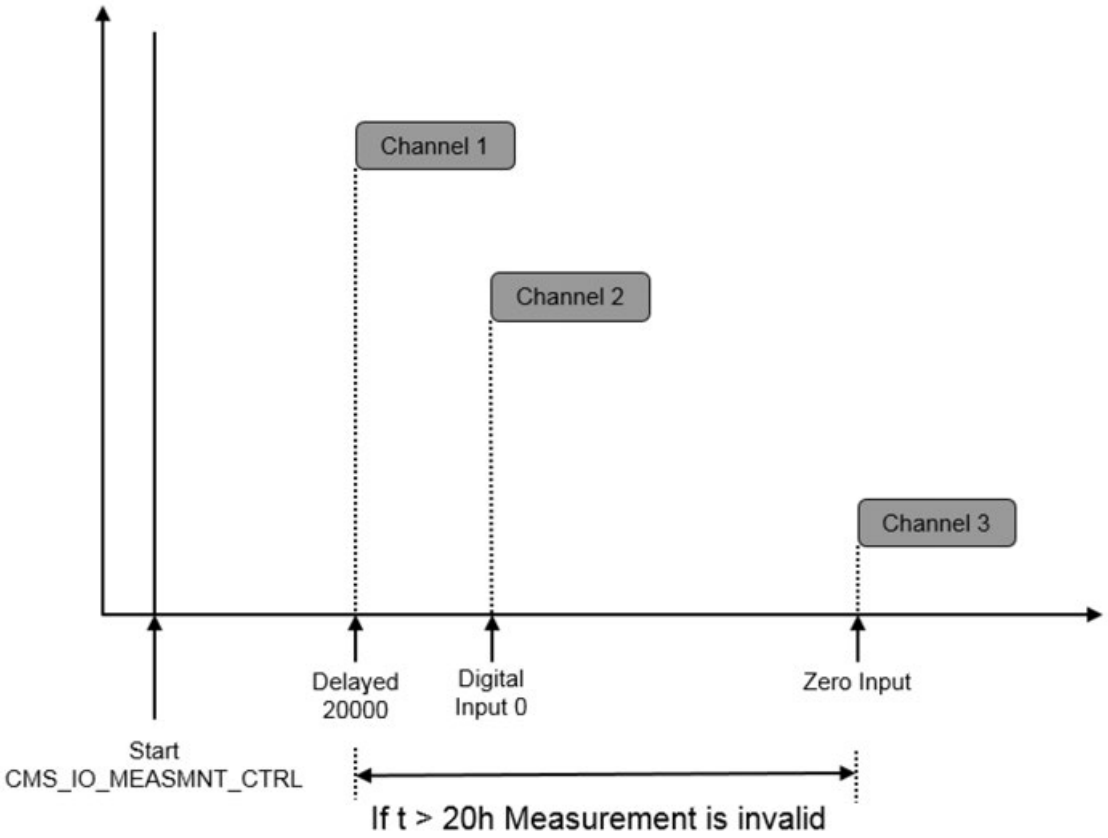


Make sure to configure the right analog mode for each channel. Take care that the sample rate is in relation to the maximum measured frequency.

Start conditions for each individual channel:

- Instantly: Starts measurement instantly after setting input EN to TRUE.
- Delayed: Measurement will start when input EN was set to TRUE and the start condition value [samples] is over.
- Digital input: Starts the measurement when input EN and the digital input was set to TRUE.
- Zero input: Starts the measurement when input EN was set to TRUE and the input Z+ gets a rising edge.
- Encoder value: Starts the measurement when input EN was set to TRUE and the start condition value is satisfied.

The start criteria of all analog channels in one measurement have to be in a 20 hour time window. Otherwise, the measurement is invalid and will be aborted.



The process image (I/O data) of the function module can be mapped for analog measurements.

Measurement files

The measurement data will be stored in the WAV file format. One WAV file will be created for each active channel.

Table 736: RIFF header

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	0 (0x00)	bfChunkID	"RIFX"
DWORD	Little	4	4 (0x04)	dwChunkSize	Data length - 8
BYTE[4]	Big	4	8 (0x08)	bfRiffType	"WAVE"

Table 737: Format chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	12 (0x0C)	bfChunkID	"fmt"
DWORD	Little	4	16 (0x10)	dwChunkSize	Data length - 8
INT	Little	2	18 (0x12)	wFormatTag	0x0001 (PCM)
INT	Little	2	20 (0x14)	wChannels	0x0001 (1 ch.)
DWORD	Little	4	24 (0x18)	dwSamples-PerSec	100 Hz - 50.000 kHz
DWORD	Little	4	28 (0x1C)	dwBytes-PerSec	Sample rate * block align
WORD	Little	2	32 (0x1E)	wBlockAlign	4 byte
WORD	Little	2	34 (0x20)	wBitsPer-Sample	32 bit

Table 738: Data chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	36 (0x24)	bfChunkID	"data"
DWORD	Little	4	40 (0x28)	dwChunkSize	Data length - 8
BYTE[]	Big	Undefined	44 (0x2C)	bfData	Measurement data

Table 739: Label chunk

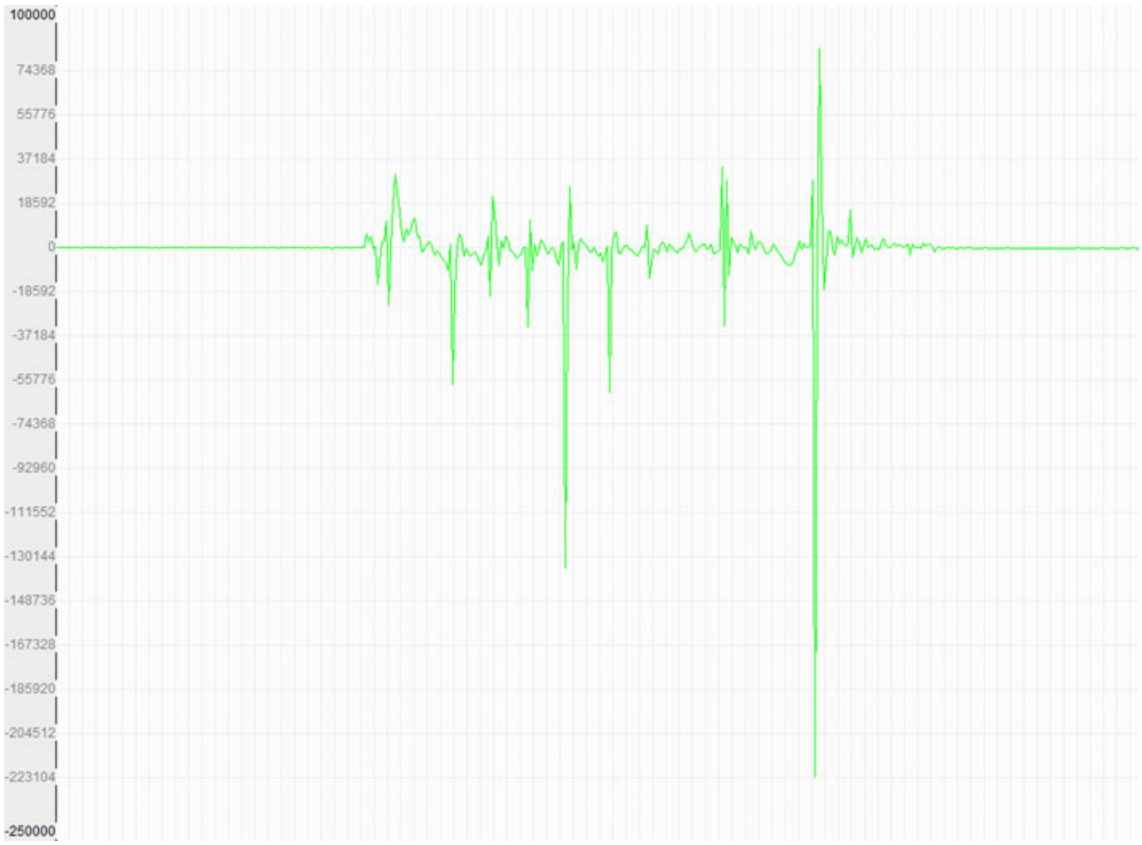
Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	44+sz(bfData)	bfChunkID	“labl”
DWORD	Little	4	48+sz(bfData)	dwChunkSize	Data length -8
DINT	Little	4	52+sz(bfData)	dwIdentifier	Identifier
BYTE[256]	Little	255	56+sz(bfData)	bfText	„Label Text“

The WAV files will be stored in an uncompressed ZIP file at the destination path of CMS_IO_MEASMNT_CTRL. The file names for of the WAV files are given by the FM502-CMS and are directly corresponding to the analog channel and encoder configuration of the FM502-CMS.

Example With no encoder, the files are named: CH00_nEN.wav, CH01_nEN.wav, ... CH15_nEN.wav and stored in a ZIP file.

Programming

In CODESYS, the data of the configured analog channels can be seen and used with the I/O mapping variables.



Use the function block CMS_IO_MEASMNT_CTRL to start a measurement.

> *The name of the measurement ZIP file has to be in 8.3 file format. Example: abcdefgh.zip*

After the measurement is finished, the ZIP file is transferred from the FM502-CMS to the PM592-ETH via communication module bus. The progress of the data transfer can be seen at the output PROGRESS in percent. After a successful measurement and data transfer, the output DONE will change to TRUE.

FM502-CMS used as counter device

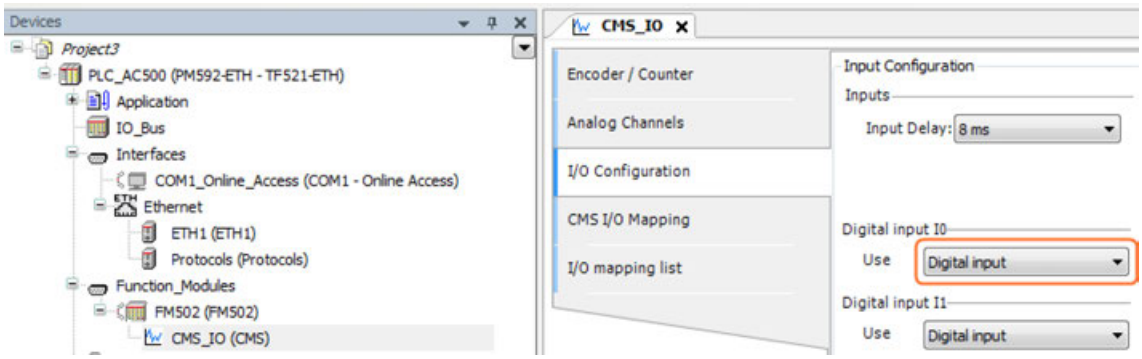


Fig. 1153: Input configuration.

32-bit bidirectional counter

The function block CMS_IO_32BIT_CNT can be used to control one 32-bit bidirectional counter function *Chapter 1.5.8.2.1.3 “CMS_IO_32BIT_CNT 32-bit counter” on page 2534*. A signal used for pulse count is identified by A+. Another signal used for enable or dynamic set is identified by B+.

Possible operation modes: 1-1, 2-1, 5-1, 6-1 *Table on page 6034*

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

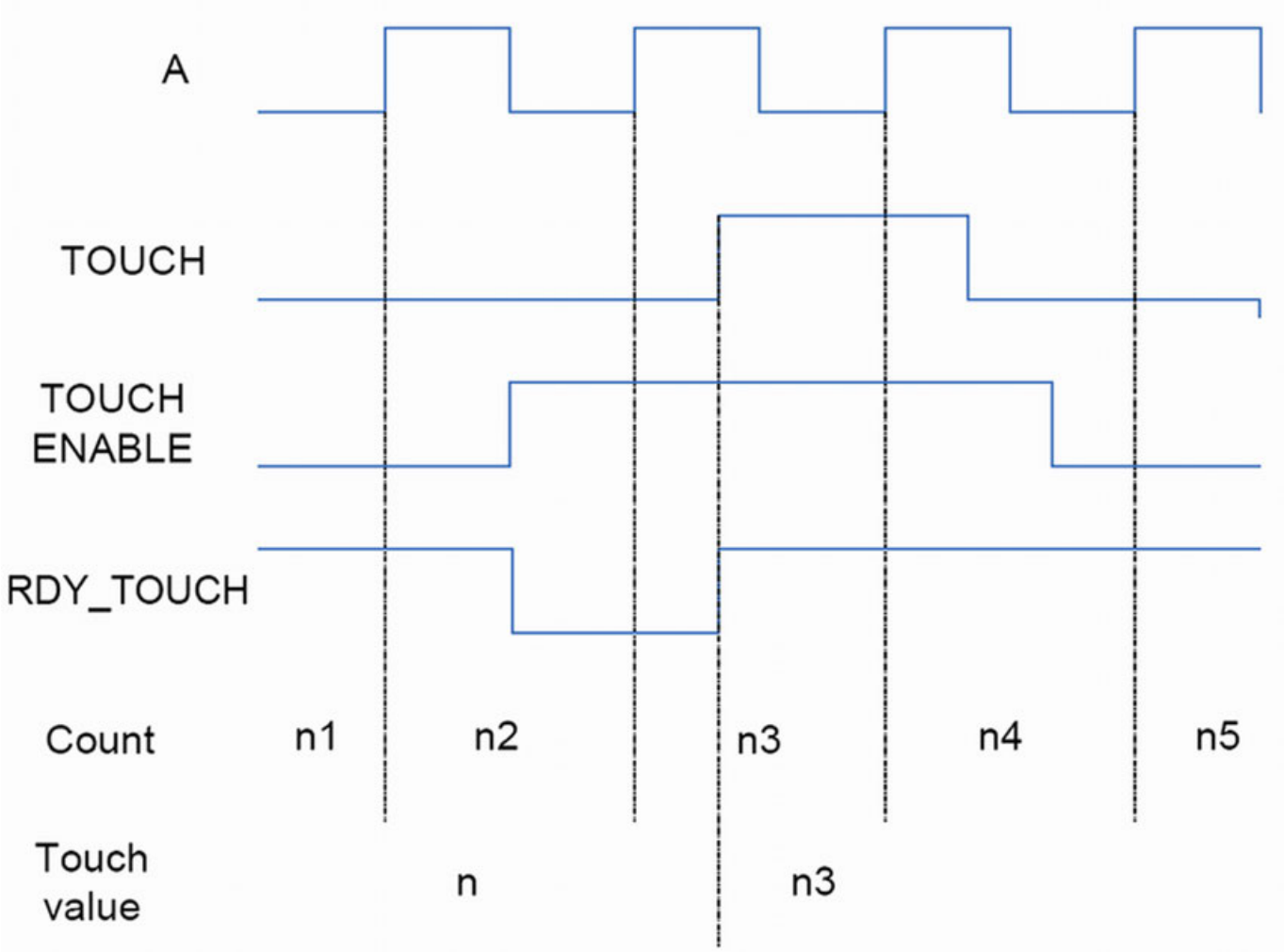


Fig. 1154: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

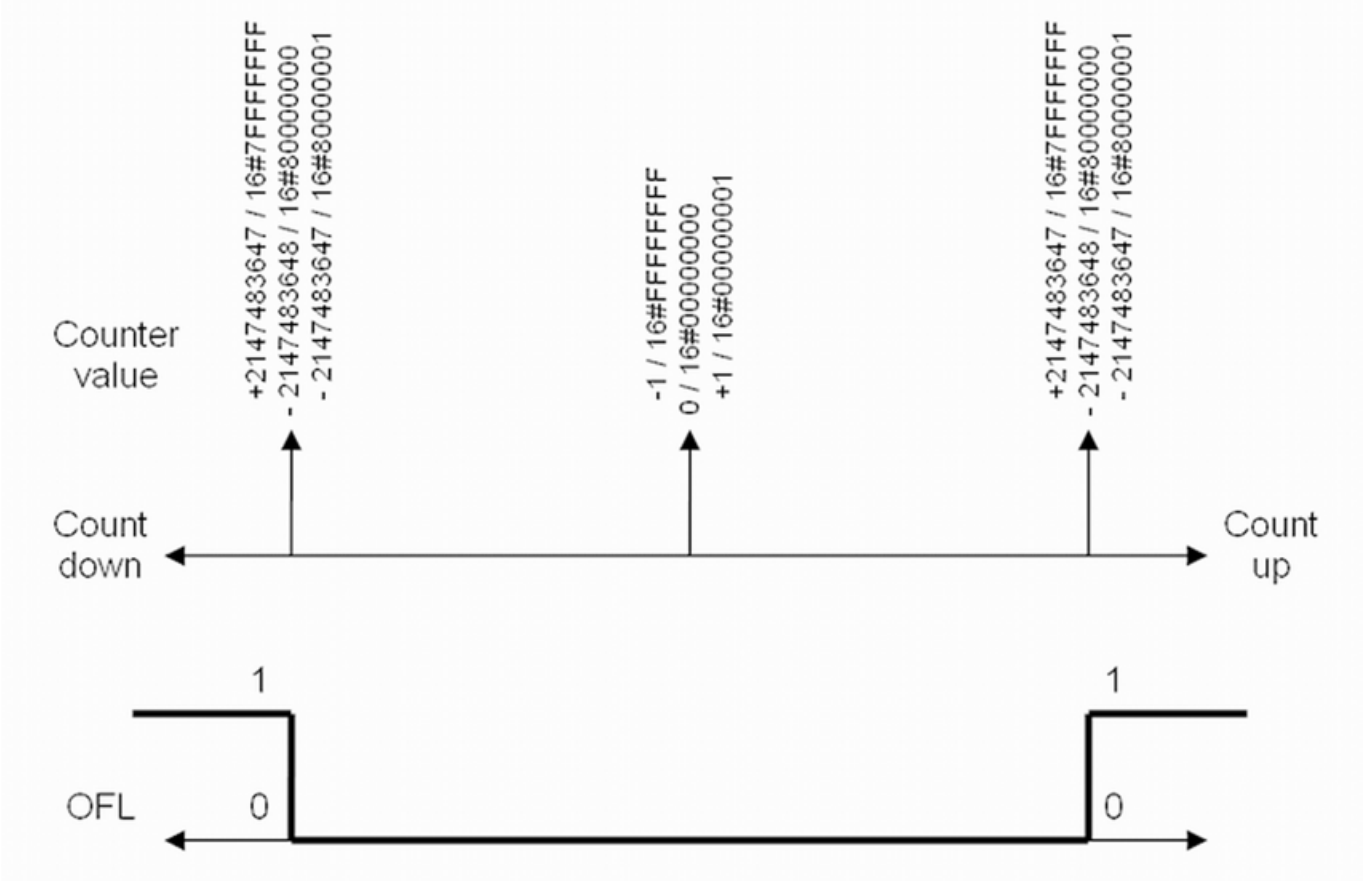
A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

SET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Overflow operation The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



16-bit bidirectional counter

The function block CMS_IO_16BIT_CNT can be used to control one 16 bit bidirectional counter function *Chapter 1.5.8.2.1.2 “CMS_IO_16BIT_CNT” on page 2530*. A signal, used for pulse count, is identified by A+.

Possible operation modes: 8-1 *Table on page 6034*

Touch/Catch operation The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

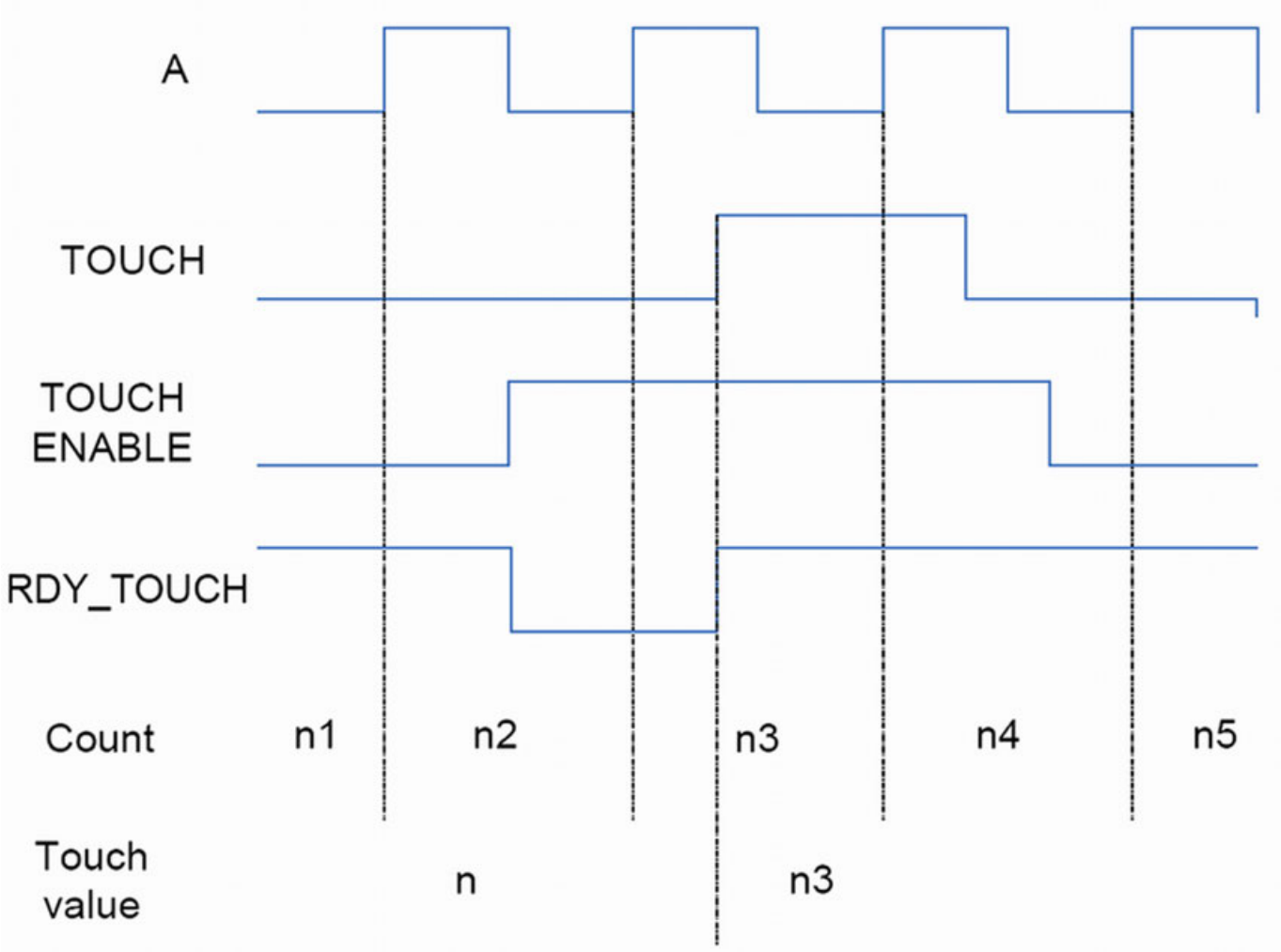


Fig. 1155: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

SET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Two 16 bit up/down counters

The function block CMS_IO_16BIT_2CNT can be used to control two independent 16 bit bidirectional counter functions *Chapter 1.5.8.2.1.1 “CMS_IO_16BIT_2CNT” on page 2526*. A signal, used for pulse count, is identified by A+ and B+.

Possible operation modes: 3-2, 4-2 *Table on page 6034*

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

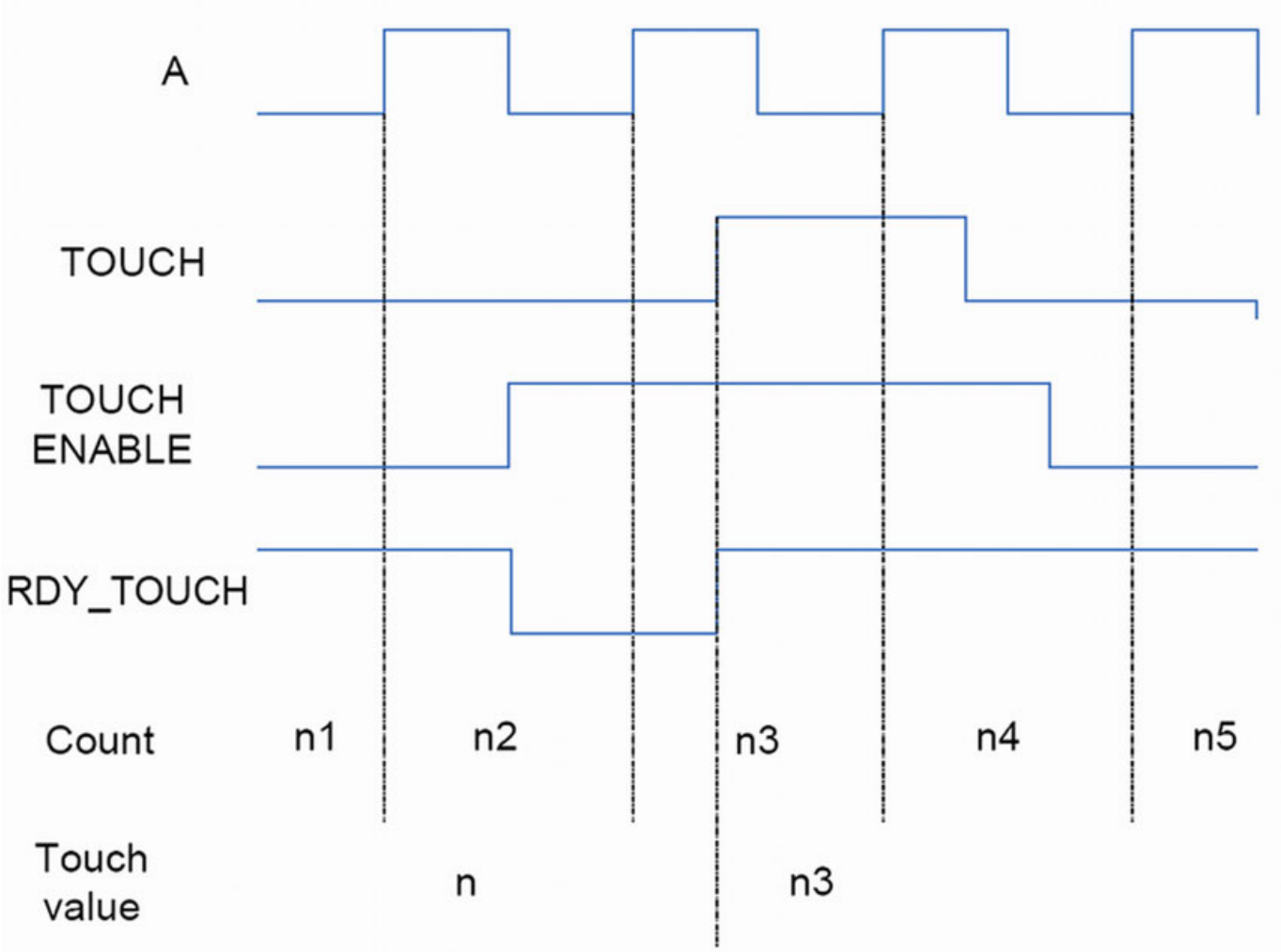


Fig. 1156: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 which is configured as RESET causes the function block to reset the value at output ACT1. A rising edge at input DI0, DI1, DC2 or DC3 which is configured as Reset 2nd Bit counter causes the function block to reset the value at output ACT2.

Overflow operation

The counter operates as an infinite counter. It is set to TRUE, when an overflow occurs, i.e. the counter value ACT1 or ACT2 goes up to value 16#FFFF= -1. Any exceeding or falling below of this value (depending to up use and down use) will set OFL1 = TRUE or OFL2 = TRUE.

FM502-CMS used as encoder device

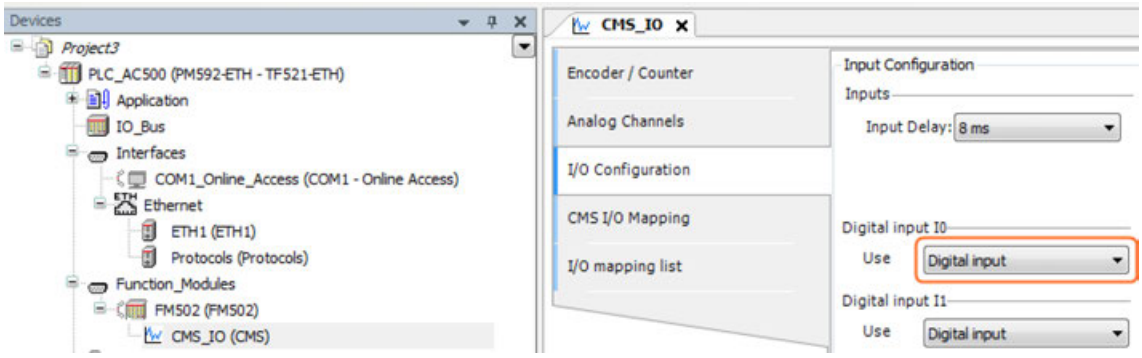


Fig. 1157: Input configuration.

Incremental encoder

The function block CMS_IO_32BIT_ENCODER can be used to control an encoder device for relative positioning with 3 signals ↳ *Chapter 1.5.8.2.1.4 “CMS_IO_32BIT_ENCODER” on page 2538*. 2 signals are used for rotation discrimination and pulse count, identified by A+ and B+. The third one is used in multi-turn encoder to count the number of rotation (mechanical zero), identified by Z+.

The rotation is identified with a shift angle (90°) between A and B signal. In the Function Module, the clockwise rotation is identified with A signal in advance to B.

Possible operation modes: 11-1, 12-1, 13-1 ↳ *Table on page 6034*

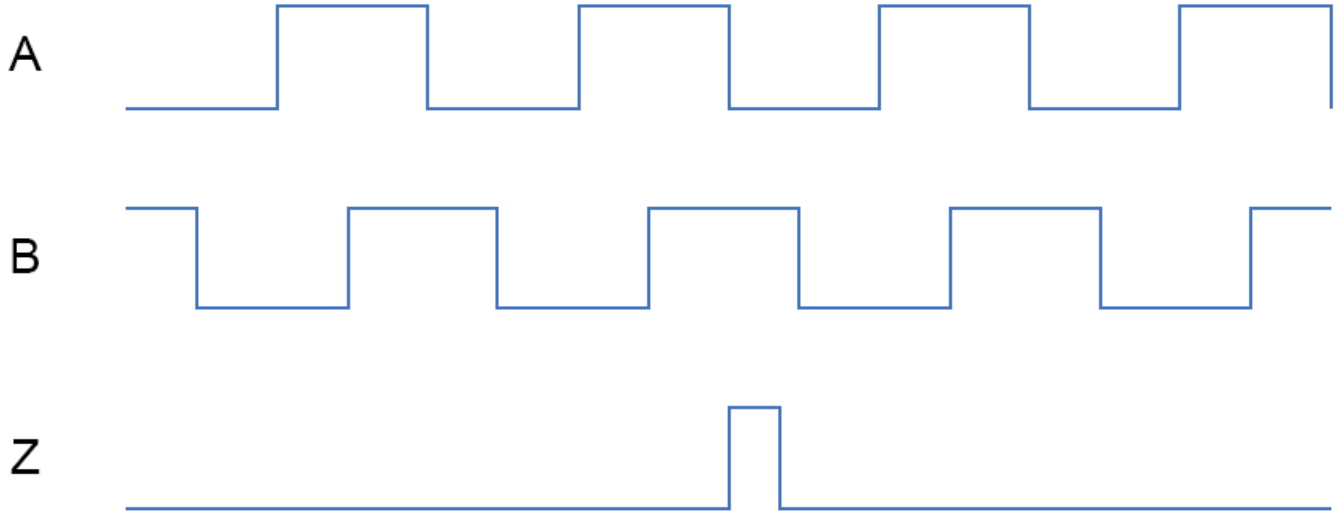


Fig. 1158: Clockwise rotation in the Function Module: A signal ahead from B signal

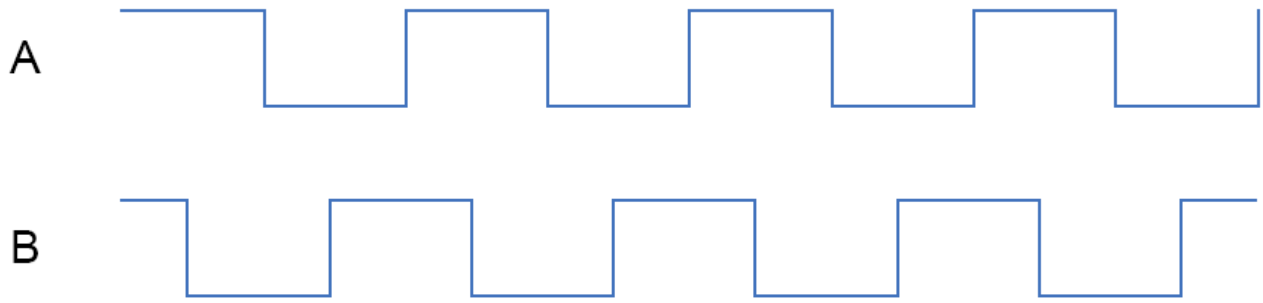


Fig. 1159: Counter-clockwise rotation in the Function Module: A signal late from B signal

Depending on choosen operation mode, the counting procedure will be x1, x2 or x4 count. Basically the x1 counting mode is used (operation mode 11-1). The encoder module discriminates the rotating way and count one pulse for each rising edge of the A signal.

With clockwise rotation, function block CMS_II_32BIT_ENCODER counts downwards. With counter-clockwise rotation, function block counts upwards.

In order to increase resolution, the x2 counting mode can be specified (operation mode 12-1). The encoder module counts one pulse on each rising and falling edge of A signal.

The resolution could be multiplied by 4, using the x4 counting mode (operation mode 13-1). The encoder module counts a pulse on both rising and falling edge of A signal and B signal.

Touch/Catch operation

The touch/catch operation is the way to acquire the counting position synchronously with hardware external signal removing all the latency time of I/O bus and network. This operation allows synchronization between 2 different encoder devices if the same hardware signal is used for touch/catch input.

On edge of the physical external signal, the current counter value (ACT) is stored in a dedicated double word (CNT_TOUCH). The touch/catch operation could be settled on rising or falling edge depending on parameter EDGE_TOUCH.

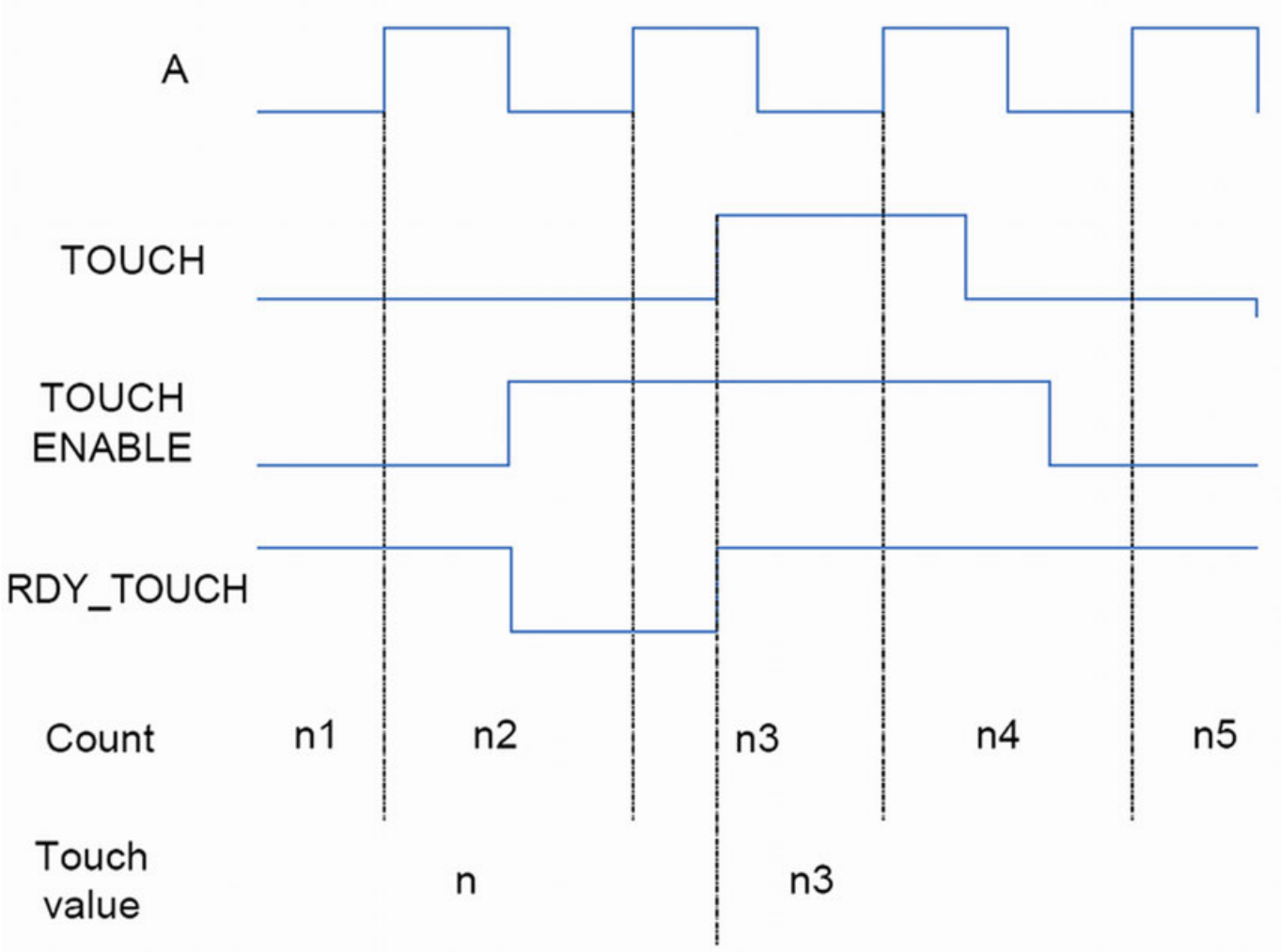


Fig. 1160: Procedure and associated counting value with signal A

Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the actual counter value and to display this value at output CNT_TOUCH.

RPI procedure

The RPI (Reference Point Initialization) is used to synchronize the counter value with the mechanical zero reference based on signal Z.

RPI procedure is enabled with control bit (EN_RPI). If this control bit is set, the module checks for the Z signal. When the signal appears, the set value is copied in the current counter value and RDY_RPI is set (see figure below).

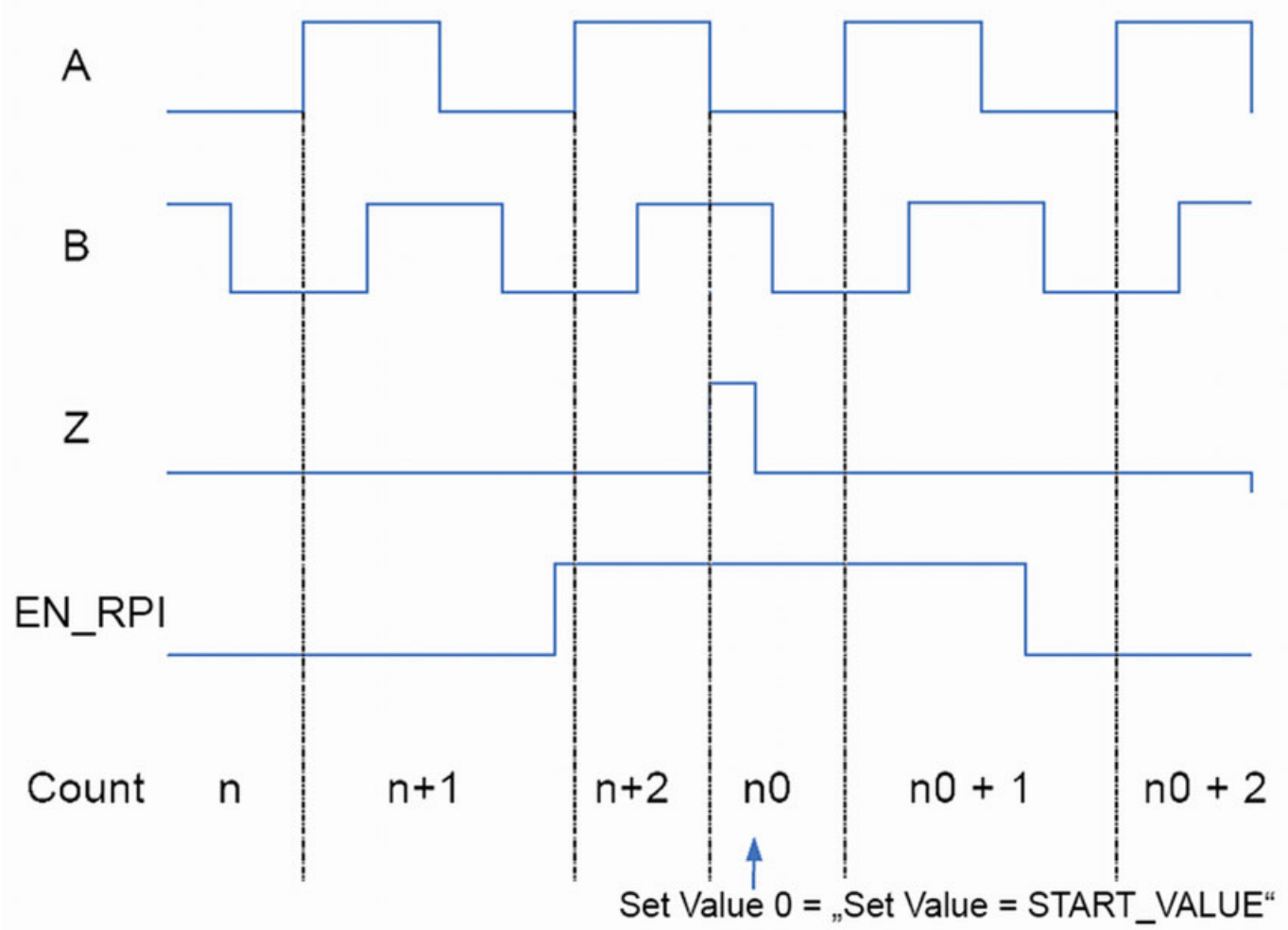


Fig. 1161: RPI operation

A rising edge at the digital inputs DI0, DI1, DC2 or DC3 activates the counter value capture and the counter reset during the capture.

Only one function may be enabled at one time, either the RPI (reference point indicator) or TOUCH (touch trigger measurement) function. If both functions are enabled simultaneously or if the execution of one function is not yet completed when enabling the other function, the RPI function will have a higher priority than the TOUCH function.

SET operation

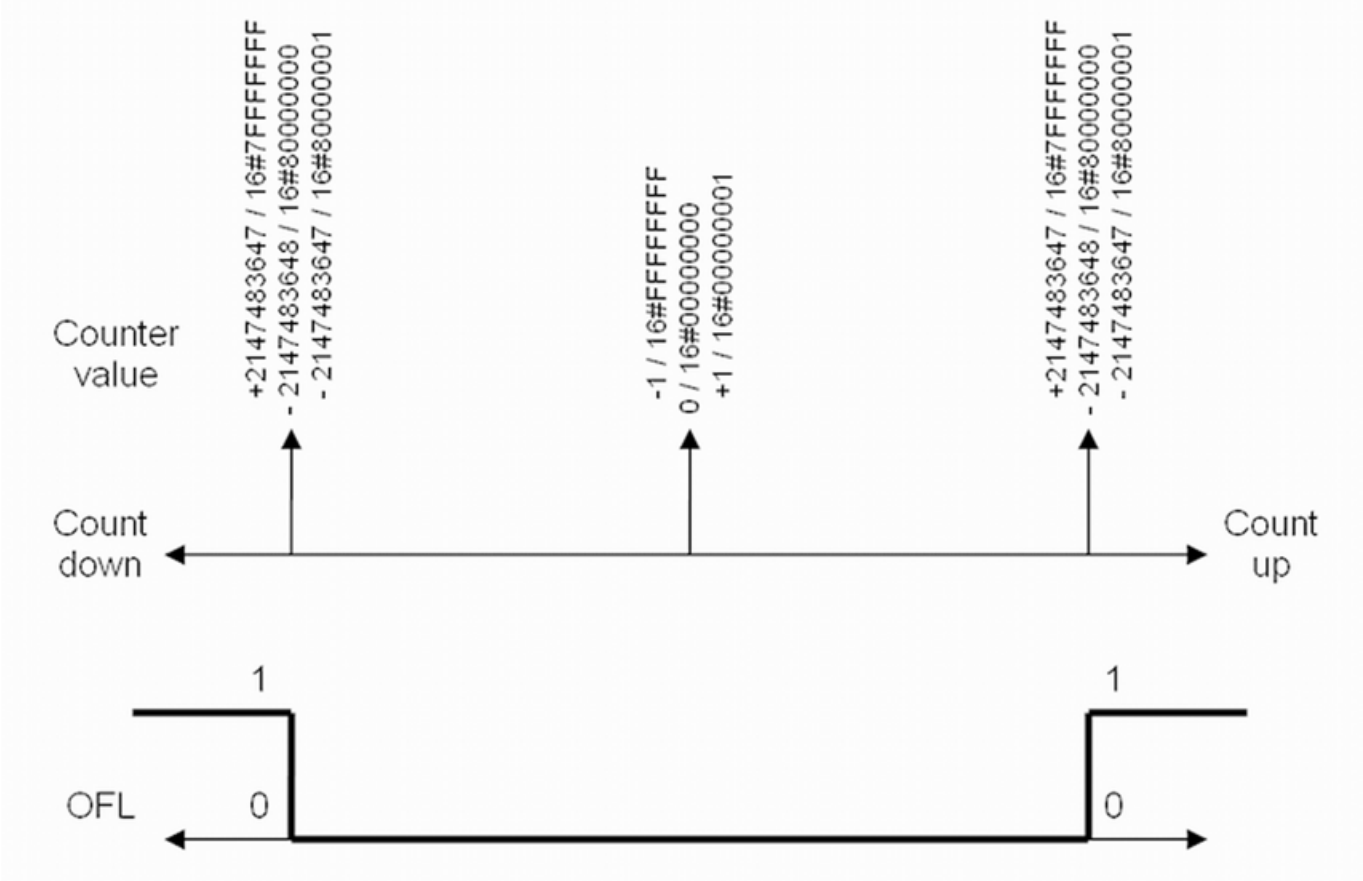
A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the START_VALUE value and to display this value at output ACT.

RESET operation

A rising edge at input DI0, DI1, DC2 or DC3 causes the block to reset the value at output ACT.

Overflow operation

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



Absolute SSI encoder

The function block CMS_IO_SSI_CNT can be used to control SSI absolute encoder function ↳ *Chapter 1.5.8.2.1.6 “CMS_IO_SSI_ENC” on page 2545*. There is an interface for absolute angle and linear encoders (displacement measurement systems).

It allows the transmission of absolute position information through a serial data transfer.

The transmission is based on synchronous serial communication. The device sends a clock signal to the encoder and synchronously, the encoder returns the positioning data from the most significant to the less significant bit.

The synchronization for a new data stream is based on time without clock pulse. This quiet time depends on the encoder.

Possible operation modes: 14-1 ↳ *Table on page 6034*

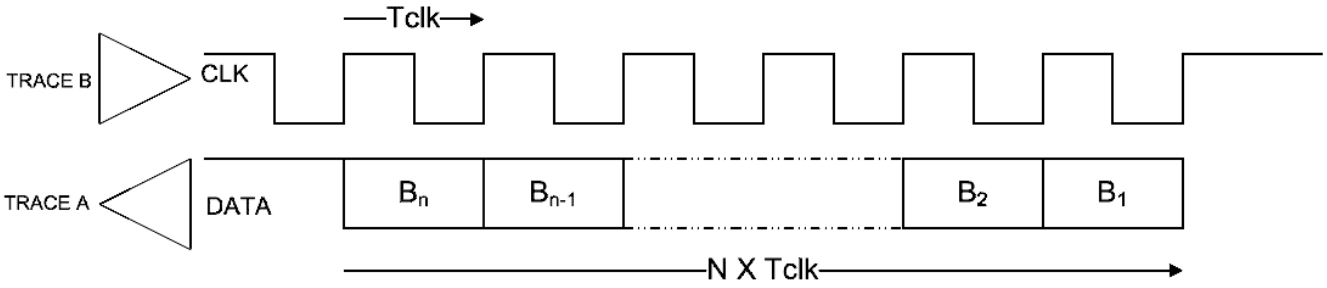


Fig. 1162: Chronogram with data organization with the clock pulse

Resolution For the resolution of the encoder device see technical data from manufacturer. The resolution can be set in configuration under *“SSI parameters ➔ SSI resolution”*.

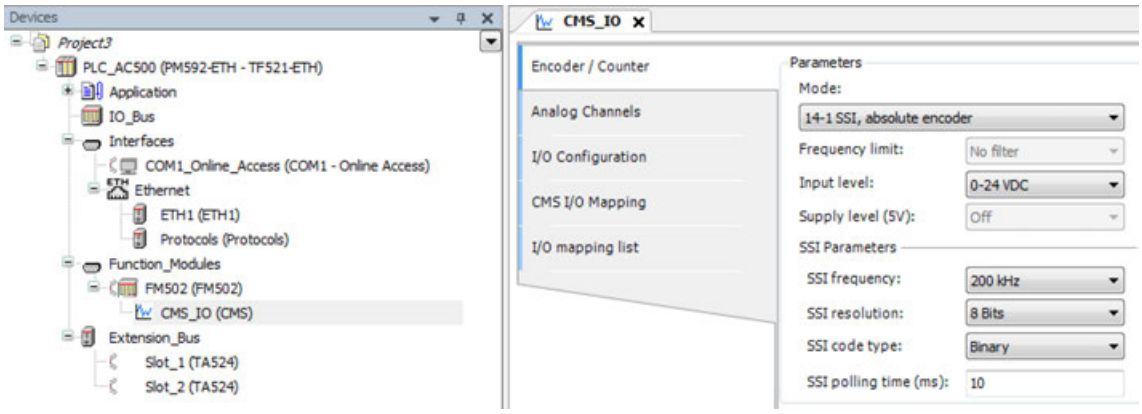


Fig. 1163: SSI parameters

The trace B of the FM502-CMS is switched as output signal (differential). On the rising edge of the signal, the sensor shifts a new value, starting from the most significant bit.

The clock frequency can be specified under *"SSI parameters ➔ SSI frequency"*.

CAUTION!
Risk of malfunctions!
The clock frequency is only an approximately value.

Do not use the clock frequency for any other purposes, e.g. time measurements.

SSI polling time definition

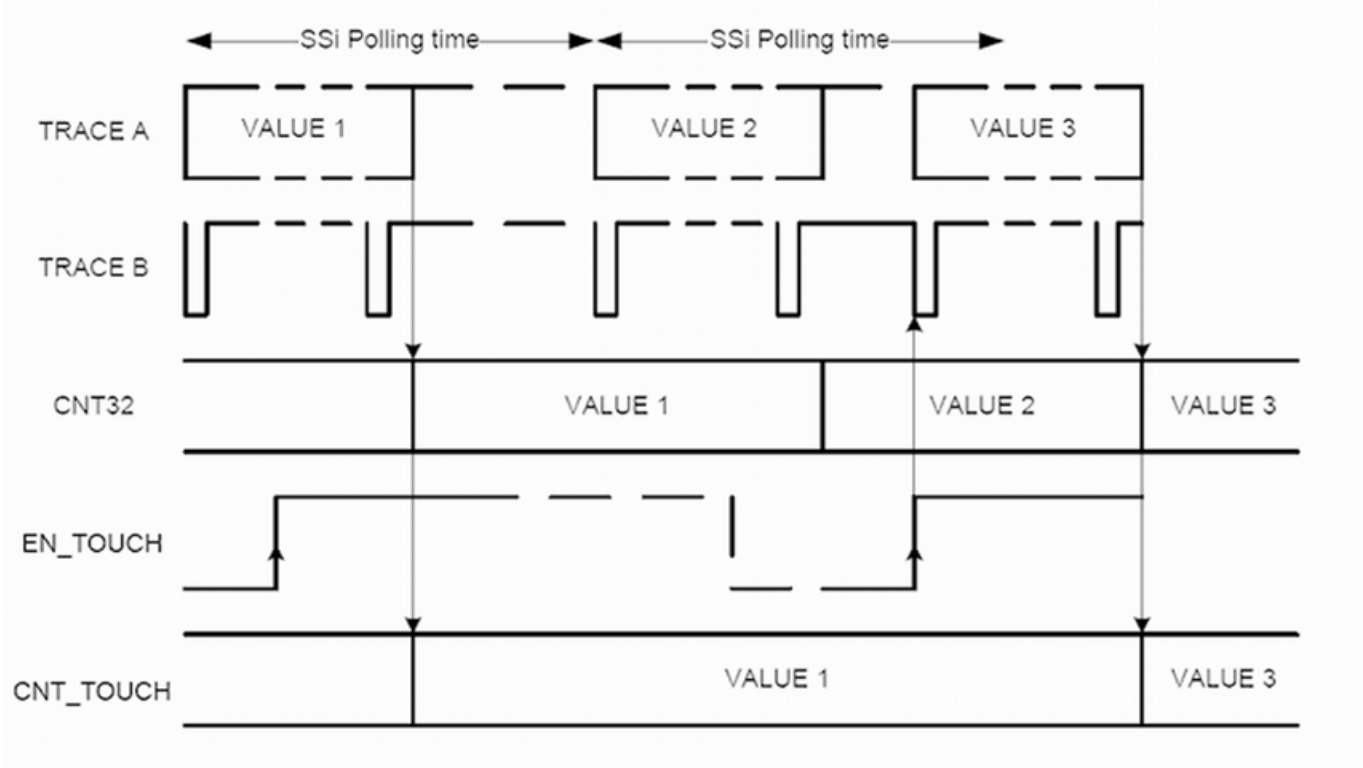
The complete read sequence is launched regularly by the Function Module. The interval between each sequence can be configured under *"SSI parameters ➔ SSI polling time"*.

SSI and touch/ catch operation

Touch operation is valid with SSI sensor. The goal of touch operation is to synchronize sensors with the same hardware signal. In the SSI mode the management is different depending on the reading procedure is running or not.

If the reading procedure has already started while the touch signal becomes active, the reading procedure finishes normally and the last read value is stored in the touch register.

If the reading procedure has not started, the encoder is in the interval between 2 measurements. The reading procedure is started one time more and the result of the last reading is stored in the touch register.

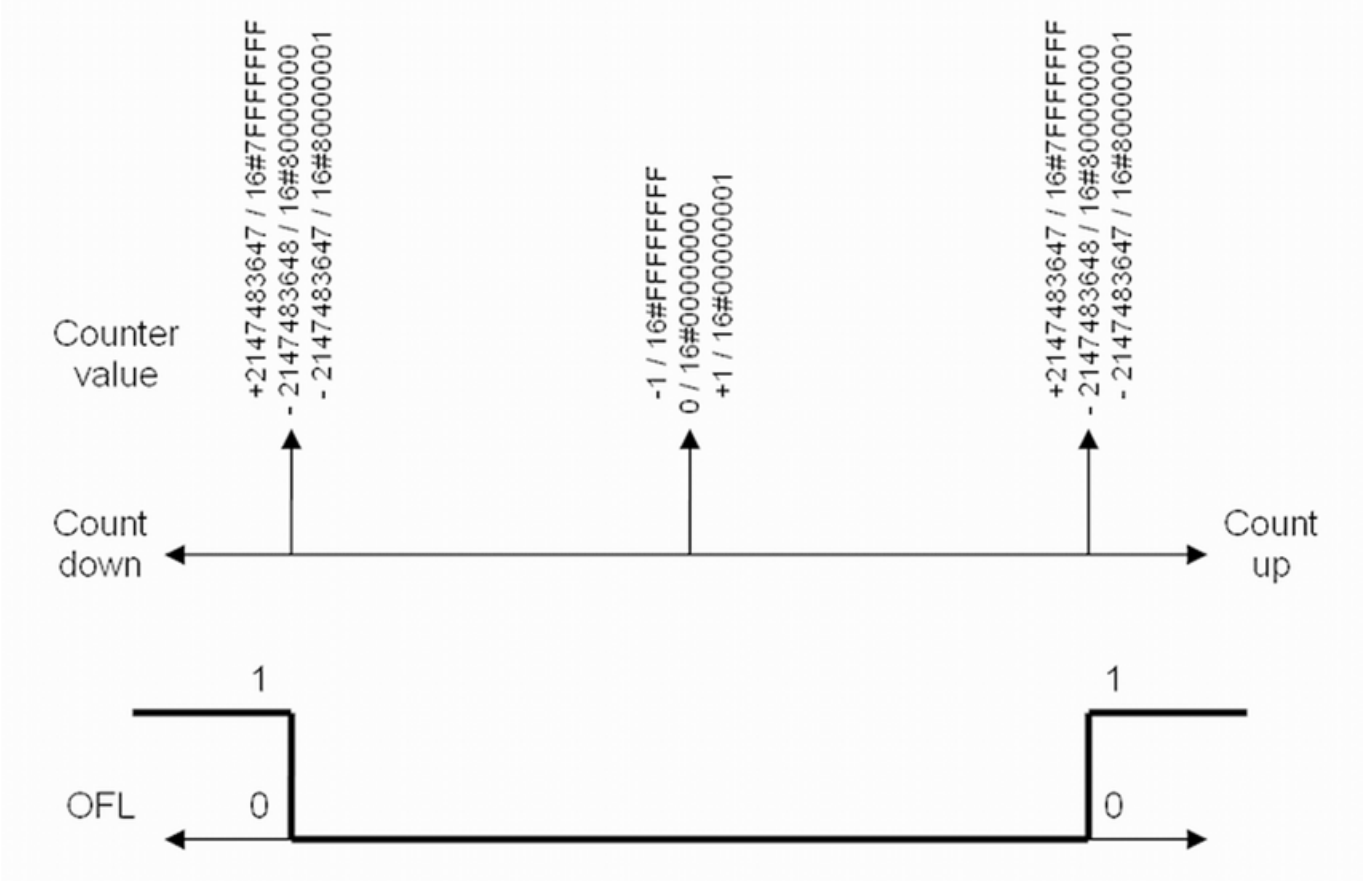


Touch/catch operation is enabled through control bit (EN_TOUCH). This resets the status flag (RDY_TOUCH), when the pre-determined edge occurs; the current counter value is stored in the touch value double word. In the same time, the status RDY_TOUCH is set to TRUE.

A rising edge at input DI0, DI1, DC2 or DC3 causes the function block to store the current counter value CNT32 (ACT) and to display this value at output CNT_TOUCH.

Overflow operation

The counter operates as infinite counter. An overflow occurs corresponding to the 32-bit value at 16#80000000 = 2147483648 and any exceeding or falling below of this value (depending to up and down use) OFL will set to TRUE.



FM502-CMS used as time frequency meter

The function block CMS_IO_FREQ_SCAN is used to measure times, frequency and rotation speeds on channel Z+ of the Function Module *Chapter 1.5.8.2.1.5 “CMS_IO_FREQ_SCAN” on page 2542*.

Possible operation modes: 15-1 *Table on page 6034*

The Function Module provides one channel (Z+) which can be used to measure times, frequencies and rotational speeds with a resolution of 1 µs.

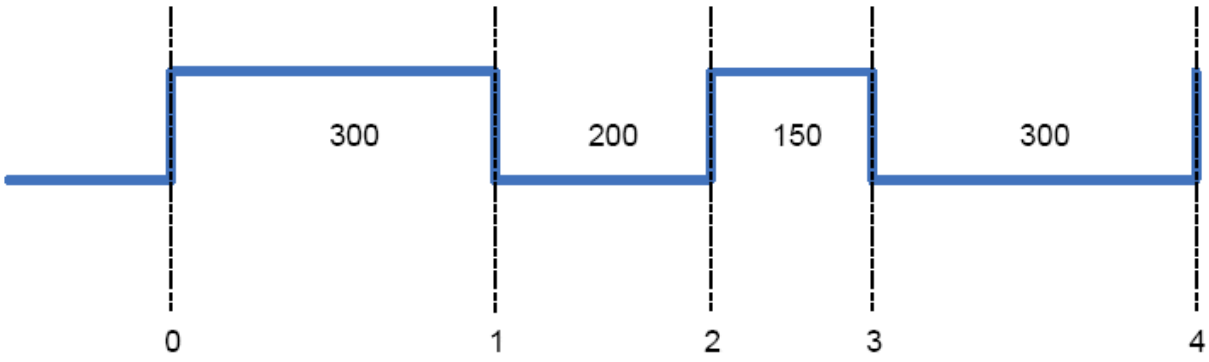


Fig. 1164: Example of timing

Table 740: Values measured according to configuration input parameters and example

EN_0	EN_1	EN_FREQ	Type	1	2	3	4
FALSE	FALSE	TRUE	No measurement	0	0	0	0
FALSE	TRUE	TRUE	Between 2 falling edges		500		450
TRUE	FALSE	TRUE	Between 2 rising edges			350	
TRUE	TRUE	TRUE	Between any 2 edges	300	200	150	300
FALSE	FALSE	FALSE	No measurement	0	0	0	0
FALSE	TRUE	FALSE	Between the rising edge and the subsequent falling edge	300		150	
TRUE	FALSE	FALSE	Between the falling edge and the subsequent rising edge		200		300
TRUE	TRUE	FALSE	Between any 2 edges (start between edge 0 and 1) *)		200		300
TRUE	TRUE	FALSE	Between any 2 edges (start between edge 1 and 2) *)			150	

*) The timing measurement is a single shot process. The function block manages renewal of the measurement as soon as the enable input is valid. Because of timing required to exchange management bits on the bus, it is not possible to provide the time measurement between two adjacent edges. Therefore, the time measured depends on when the measurement is started.

Depending on the input parameters of the function block, the result of the measurement can be read as time in µs, frequency in Hz or speed of rotation in rotation per minute.

FM502-CMS used with synchronized counter/encoder files

Refer to the operation modes and used function blocks ↬ *Table on page 6034*.

Configuration

▷ Configure at least an encoder mode and one analog channel which should be recorded.

⇨ When encoder mode is set, on the **analog channels** tab the check box *Synchronized encoder file* is selected.

The encoder track will only be recorded when the correct encoder function block is used and enabled.

Measurement files

The measurement data will be stored in the WAV file format. One WAV file will be created for each active channel.

Table 741: RIFF header

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	0 (0x00)	bfChunkID	"RIFX"
DWORD	Little	4	4 (0x04)	dwChunkSize	Data length - 8
BYTE[4]	Big	4	8 (0x08)	bfRiffType	"WAVE"

Table 742: Format chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	12 (0x0C)	bfChunkID	"fmt"
DWORD	Little	4	16 (0x10)	dwChunkSize	Data length - 8
INT	Little	2	18 (0x12)	wFormatTag	0x0001 (PCM)
INT	Little	2	20 (0x14)	wChannels	0x0001 (1 ch.)
DWORD	Little	4	24 (0x18)	dwSamples-PerSec	100 Hz - 50.000 kHz
DWORD	Little	4	28 (0x1C)	dwBytes-PerSec	Sample rate * block align
WORD	Little	2	32 (0x1E)	wBlockAlign	4 byte
WORD	Little	2	34 (0x20)	wBitsPer-Sample	32 bit

Table 743: Data chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	36 (0x24)	bfChunkID	"data"
DWORD	Little	4	40 (0x28)	dwChunkSize	Data length - 8
BYTE[]	Big	Undefined	44 (0x2C)	bfData	Measurement data

Table 744: Label chunk

Data type	Endian	Length	File offset	Identifier	Value
BYTE[4]	Big	4	44+sz(bfData)	bfChunkID	“labl”
DWORD	Little	4	48+sz(bfData)	dwChunkSize	Data length -8
DINT	Little	4	52+sz(bfData)	dwIdentifier	Identifier
BYTE[256]	Little	255	56+sz(bfData)	bfText	„Label Text“

The WAV files will be stored in an uncompressed ZIP file at the destination path of CMS_IO_MEASMNT_CTRL. The name of the WAV files is given by the Function Module and is corresponds directly with the analog channel and encoder configuration. For every encoder synchronized shannel with different sample rate, there will be a new encoder track which will be used for synchronization.

Example

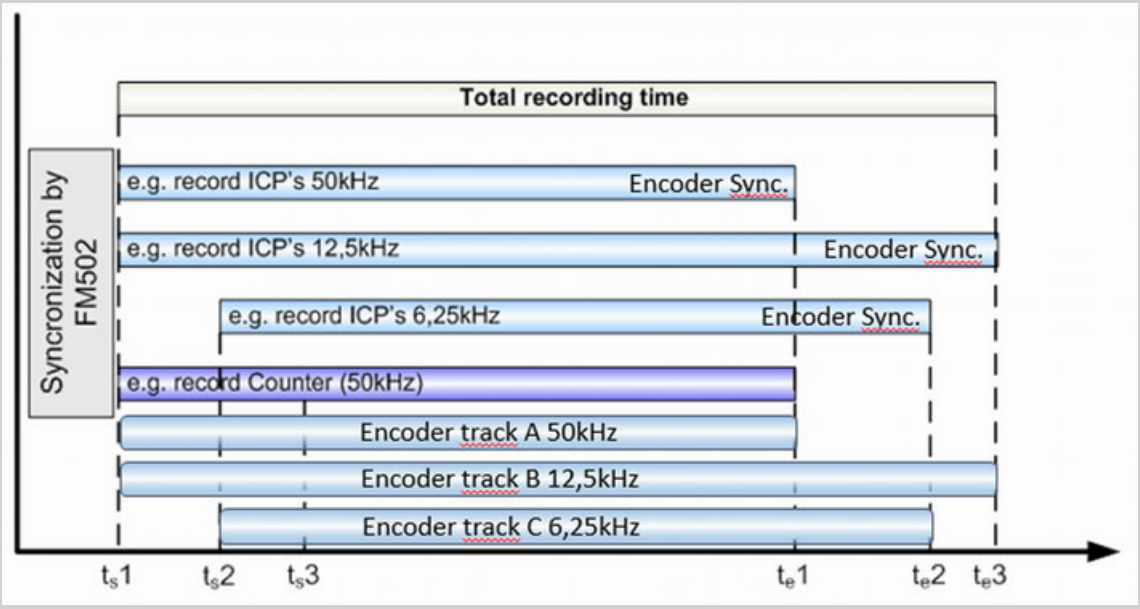
Encoder is configured as:

Channel 0: Sample rate 50 kHz, synchronized encoder file

Channel 1: Sample rate 12,5 kHz, synchronized encoder file

Channel 2: Sample rate 6,25 kHz, synchronized encoder file

Result: The ZIP file contains the WAV files: CH00_ENA.wav, CH01_ENB.wav, CH02_ENC.wav, Enc_A.wav, Enc_B.wav, Enc_C.wav



Example

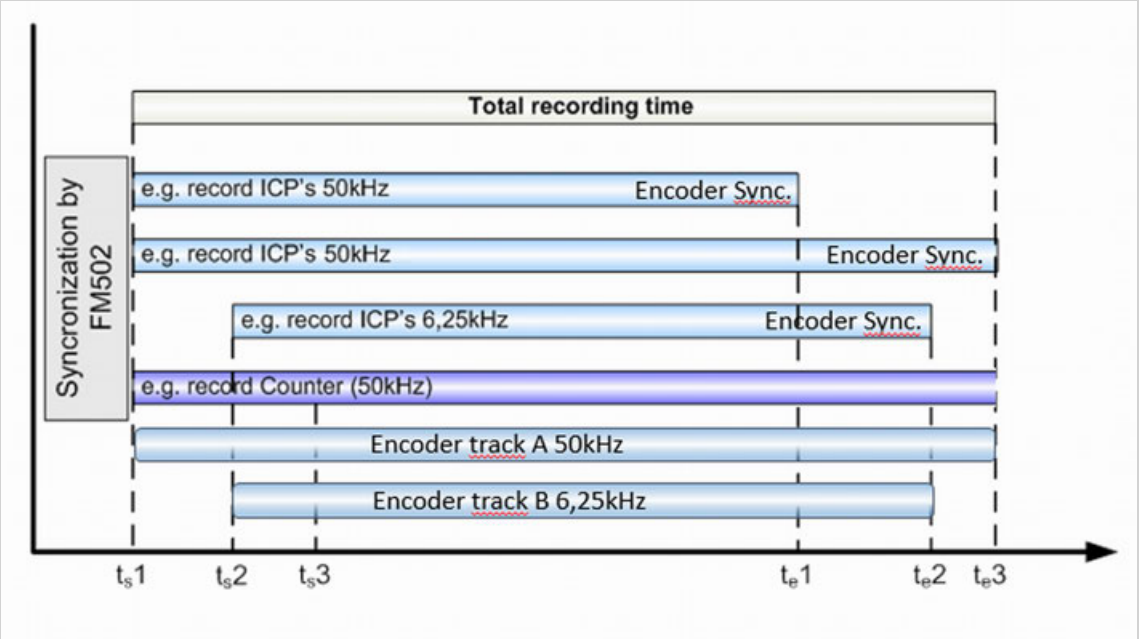
Encoder is configured as:

Channel 0: Sample rate 50 kHz, synchronized encoder file

Channel 1: Sample rate 50 kHz, synchronized encoder file

Channel 2: Sample rate 6,25 kHz, synchronized encoder file

Result: The ZIP file contains the WAV files: CH00_ENA.wav, CH01_ENA.wav, CH02_ENB.wav, Enc_A.wav, Enc_B.wav



The encoder/counter value at the output of the function block will reset when configuration data of the Function Module will be written in CODESYS.

FM562 module

The function module FM562 for pulse train output (PTO) is used for simple positioning tasks with servo drives or stepper drives. FM562 provides 2 axes with 2 inputs and 2 pulse-train outputs each.

It can be used at the following devices:

- Communication interface modules (e. g. CI501-PNIO, CI541-DP)
- Processor modules

It contains the following features:

- 2 axes control
- 2 configurable discrete digital inputs per axis for enable and limit switches signal inputs
- PTO output type: RS-422 differential output (P0, P1, P2 and P3)
- PTO frequency: 10 Hz to 250kHz
- Configurable PTO output mode: CW/CCW (clockwise/counterclockwise), pulse/direction
- Position and speed control with built in motion profile generators. Integration in the application program by PLCopen Motion Control function blocks (PS552-MC-E motion control library is required for programming)

The pulse outputs of the 2 axes are not galvanically isolated from each other.

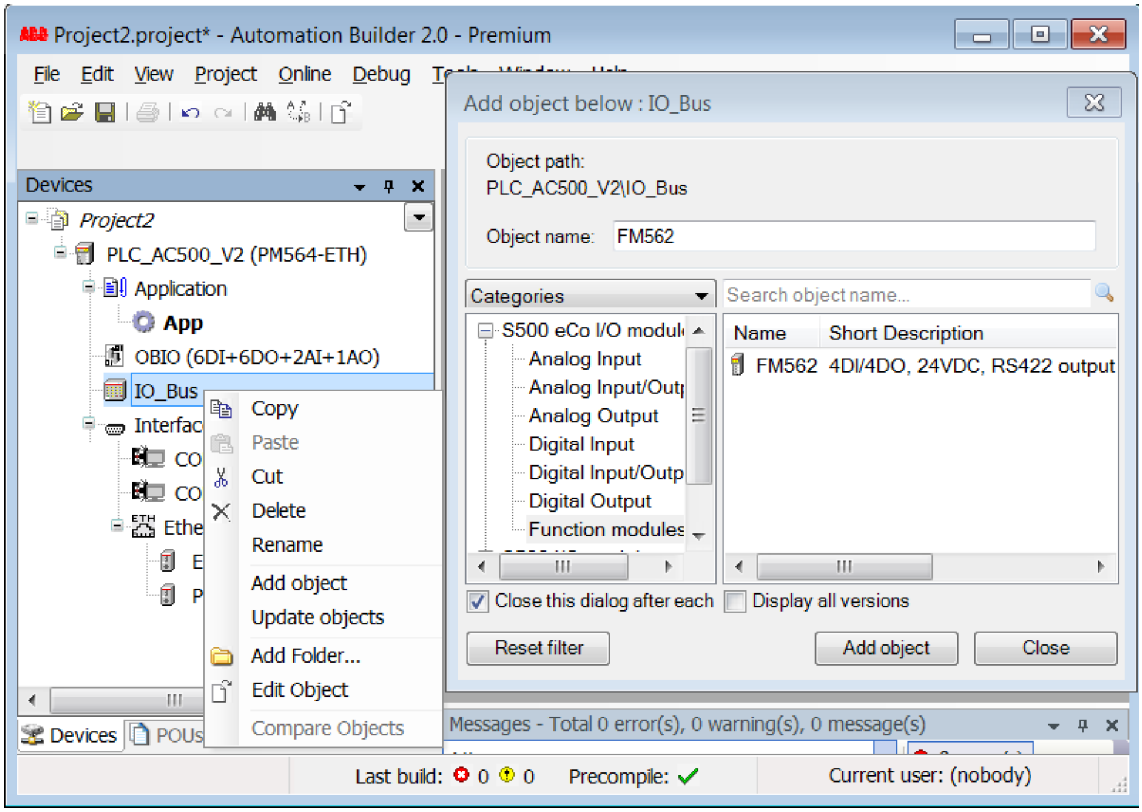
The other circuitry of the module is galvanically isolated from the inputs/outputs.

Special features

FM562 provides a square wave output for a specified number of pulses and a specified cycle time. It can be programmed to produce either 1 train of pulses or a pulse profile consisting of multiple trains of pulses. For example, a pulse profile can be used to control a stepper motor through a simple ramp up, run, and ramp down sequence or more complicated sequences. The control positioning is achieved according to an open loop mode, meaning without the need for feedback information on the real position. The position loop is integrated in the servo-drive.

See also: *Chapter 1.6.4.4.4 “FM562 module” on page 5751*

Inserting FM562 on I/O bus



1. Right-click on the *“IO_Bus”* element in the configuration tree opens the context menu.
2. Click *“Add object”* and open in the *“S500 eCo I/O module”* list the *“Function modules”*.
3. Select *“FM562”* and click *“Add object”*.

See also: *Chapter 1.6.2.7.1.1 “FM562 for pulse train output” on page 4617*

Configuring the Library Manager

> *Ensure that you have installed the library PS552-MC-E before you start using the FM562 module in your application program.*

The following library files of the PLCopen Motion Control Library PS552-MC-E are required for using the FM562 module inside an application:

PTO library file:

- PTO_FM562_MC_support_V22.lib

The PTO Library provides the access function block that contains the interface between the PLCopen function blocks and the FM562 PTO Module.

PLCopen Library files:

- MC_Base_AC500_V11.lib
- MC_Blocks_AC500_V11.lib

The PLCopen Library contains function blocks based on the PLCopen Motion Control standard and can be used for various motion devices, e.g: ACSM1, ACS350, ACS800, E100, E150 and FM562.

See also: *Chapter 1.6.4.4.4.1.2 “The function blocks used with FM562” on page 5753*

1.6.5.2.9 I/O bus and I/O modules

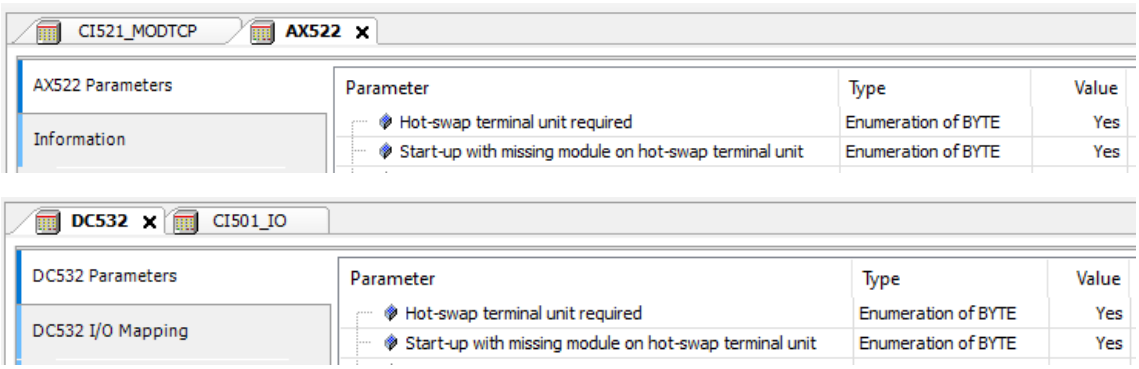
Hot swap configuration

Parameter configuration

I/O extension modules include the below parameters for hot swap configuration

Parameter	Purpose	Value
Hot-swap terminal unit required	To include diagnosis for missing hot-swap terminal unit	Yes: Communication Interface provides extended diagnosis for missing hot-swap terminal unit
		No (default): Extended diagnosis not available
Start-up with missing module on hot-swap terminal unit	Ignore missing module during start-up on hot-swap terminal unit. Incomplete I/O configurations must not prevent the system from starting.	Yes: Module is optional, start-up if there is no module available on hot-swap terminal unit
		No (default): Module is mandatory, start-up only if correct module is available

In Automation Builder projects for Modbus TCP and V2 PLCs, hot-swap parameters can be configured from the Parameters tab of respective I/O module



Parameterization of the I/O bus

Double-click the *"IO_Bus"* node in the device tree to open the I/O bus configuration.

The following parameters are available:

Parameter	Default	Value	Description
Run on config fault	No	No	In case of configuration fault the user program will not be launched.
		Yes	The user program will be also launched in case of configuration error on the I/O Bus.
Max wait run	3000	0...120000	Maximum waiting time for valid inputs.

In case of a digital I/O Module, the channels are provided as WORD, BYTE and BOOL. Because the analog inputs can also be configured as digital inputs, bit 0 of each channel is also available as BOOL.

The symbolic name of a channel can be entered in front of the string "AT" in the channel declaration.

All channels should have a symbolic name and only symbolic names should be used in the program code. If the hardware configuration has changed or if you want to download the project to a PLC with another hardware configuration and thus the PLC configuration has to be changed, the addresses of the inputs and outputs can change. In case of symbolic programming (i.e., symbolic names are used), the program code does not have to be changed.

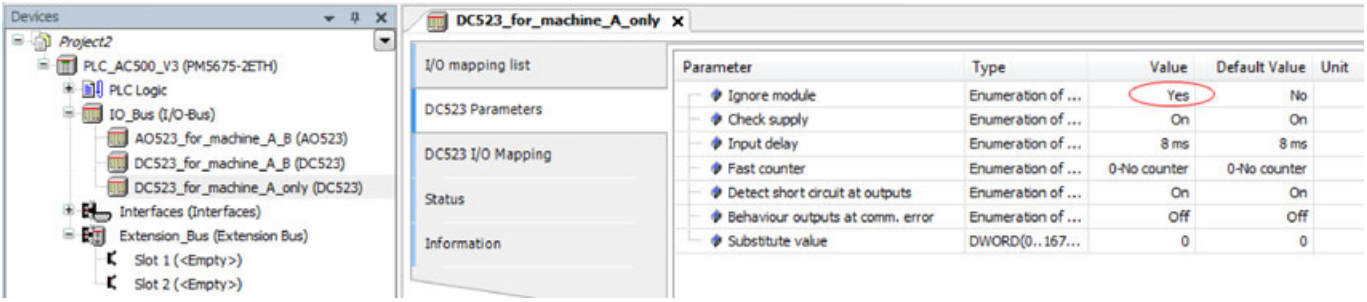
Parameter 'Ignore module'

All I/O devices provide the parameter "Ignore module". This parameter can be used for simulation purposes and determines whether an I/O device is considered or ignored during a PLC configuration check.

This allows to use an existing Automation Builder project/PLC configuration though some hardware devices are not physically available in a hardware installation.

> *I/O devices with one byte input and output cannot be detected if they are configured but not connected to the CS31-Bus.*

Example The Automation Builder project for machine A shall be used for machine B. However, the second DC523 device is missing in the hardware installation of machine B. Hence, for machine B the value for 'Ignore module' is set to 'YES'.



Insertion of S500 I/O devices

1. Right-click *"IO_Bus"* in device tree and select *[Add object]*.

   -> The Add Device dialog window where all available S500 I/O Devices are listed will open.

2. Append the S500 I/O Devices in the same order as they are mounted on the hardware.

   Input and output modules connected to the I/O bus occupy the I/O following area: %IB0 .. %IB999 or %QB0 .. %QB999.

> *AC500-eCo: PM55x-xP and PM56x-xP support up to 10 S500 I/O Devices.*
>
> *AC500 (Standard): PM57x, PM58x, PM59x support up to 10 S500 I/O Devices.*

Configuring the input and output modules and channels

The I/O channel configuration depends on the corresponding S500 I/O Device. See hardware documentation of the I/O Device for more information.

The individual configuration parameters can be opened in the editor window via double-click on the corresponding module and are listed in tab [S500 I/O device name] Configuration.

Symbolic names for variables, inputs and outputs

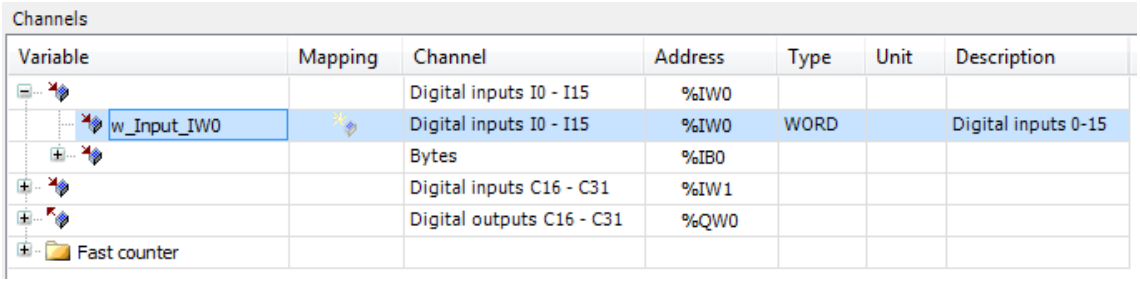
> *The IEC naming rules are not checked during input in Automation Builder.*

Input and output mapping

Devices with I/Os provide an I/O Mapping tab in their configuration editor where the available I/O channels can directly be mapped to a global variable.

The corresponding variable declarations are automatically created in a global variables object in a subfolder of the Global Variables section in the CODESYS project.

All available I/O channels can easily be assigned to a variable.



The variable is automatically added to the Global Variables in the CODESYS project after recreating the configuration data *Chapter 1.6.5.4.1.1 “Creating configuration data” on page 6196*.

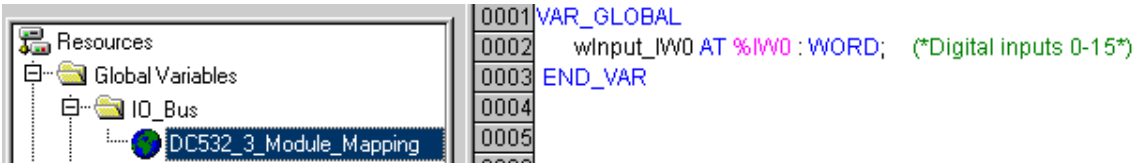
AC500 uses Motorola Byte Order (Big Endian).

The numbers in column Channel correspond to the channel numbers only and not to the bit position inside the WORD variable.

Only entries with a data type set in column "Type" can be mapped. These entries can be expanded to show the available I/O channels.

If the project has been imported from a previous Automation Builder version, all variables should be checked to avoid inconsistencies concerning the I/O mapping.

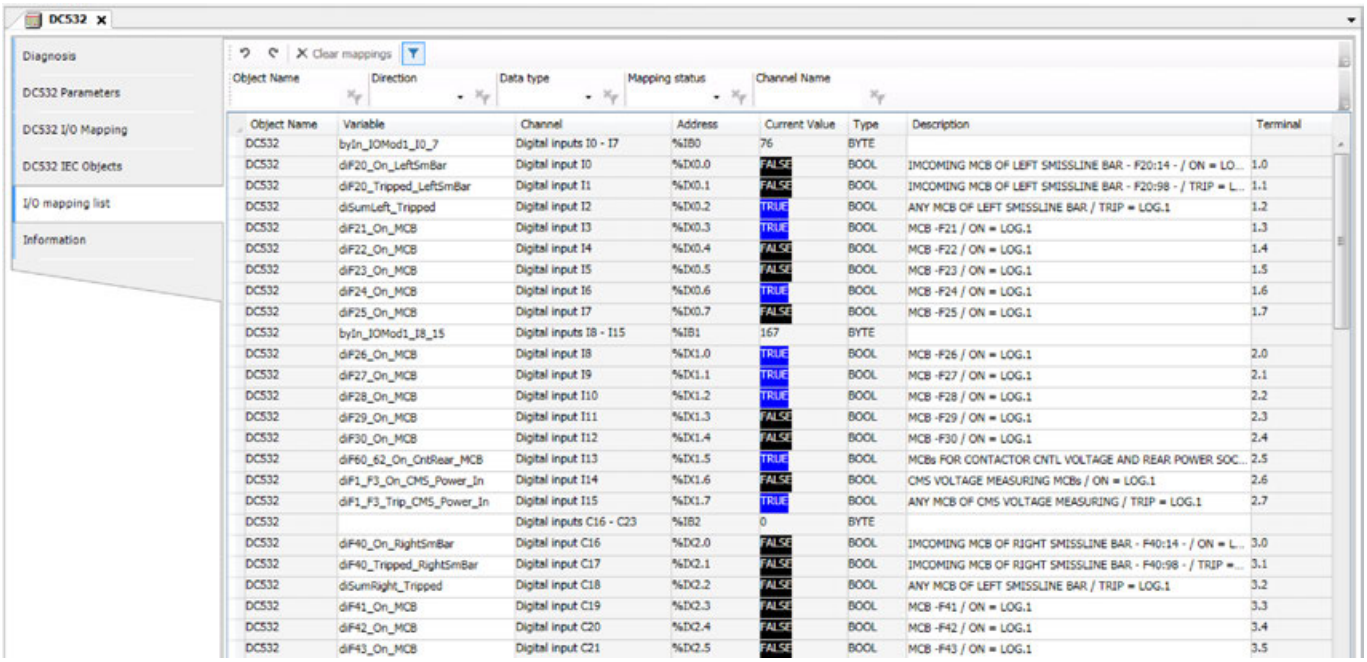
The variable is automatically added to the global variables in the CODESYS project after (re)creating the configuration data:



An additional GVL (Global Variables List) can be created and transferred to CODESYS V2.3. Editing of lists created in CODESYS V2.3 is not possible.

I/O mapping list

Automation Builder contains an I/O mapping list feature for creating mapping variables with better usability support compared to the tree structured view. Details on the tree structured view is provided in the CODESYS Development System.



Functionalities of the I/O mapping list:

- Displays I/O mappings for current node and all valid subsequent child nodes.
- Displays channel information with additional columns.
- Supports keyboard functions such as *cut*, *copy*, *paste*, *delete*, and *select all* within the editor and within Excel spreadsheet (for bulk editing).
- Contains a toolbar for various actions, e.g. filtering, undo/redo and clear mappings.
- Supports single click edit and easy navigation using arrow keys.
- Improvised error handling:
  - Allows to enter invalid mapping variables. This provides flexibility in bulk editing. Only when saving the project, the errors - according to IEC 61131 standard - are displayed.
  - In the message window, the error log is visible. The user can track the errors to their corresponding channel in the editor.
- Allows multi-selection of rows and columns. (Random selection is not allowed.)

Configuring I/O mapping list

Automation Builder supports tree and list based editors for creating I/O mapping variables.

1. From the **Tools** menu, select **Options**.
2. Under **Automation Builder**, select the **Editors** tab.
3. Choose your desired mapping dialog and click **OK**.
   - Choose **tree based** to display the I/O mapping in tree structure.
   - Choose **list based** to display the I/O mapping as list with the functionalities of the ToolBar.
   - Choose **both** to display both the tree structure (**I/O Mapping** tab) and the list view (**I/O mapping list** tab).

Available channel information

The I/O mapping list displays the channel information in offline and online mode. In online mode, all columns are read-only. In offline mode, some columns are editable.

The order of the devices in I/O mapping list is synchronized with the order in the device tree.

The channels of a device are ordered by the device description file. If channels have a section, the channel information is represented in a specific format.

Example: Fast counter: Actual value 1. These channels are listed at last position of a device.

Editing I/O mapping list

1. In the device tree, double-click **IO_Bus** to configure entire I/O mapping list of different I/O devices.
2. Enter the variables and descriptions to map the I/O devices.

> *Do not start variable names with a number or a special character. When saving the project, this generates an error. Example: 12input3, @input4.*

3. Click **Save Project** to save the I/O mapping changes.

Toolbar

Filtering

Especially in case of long I/O mapping lists, it might be helpful to filter the I/O mappings. For this, click the *"Filter"* icon to display all available criteria for filter options.

> *When reducing the width of the editor, some filters might be hidden.*

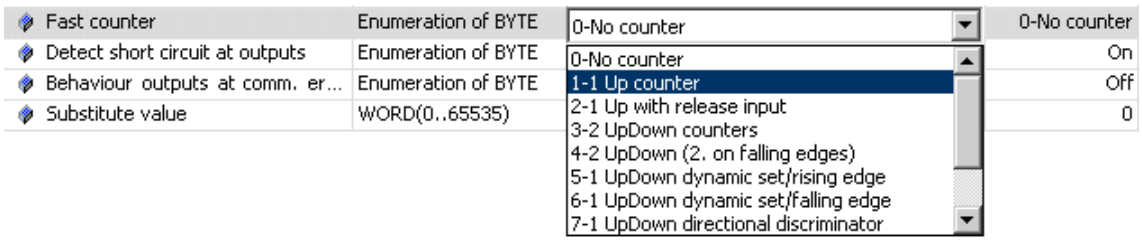
Undo, redo and clear

- **Undo**: Cancels the last change.
- **Redo**: Repeats the last change.
- **Clear mappings**: Deletes all variables and descriptions.

Fast counter

Configuration for S500 I/O modules

1. In the device tree, add a digital I/O module to the *"IO-Bus"* node.
2. Double-click the node for the I/O module, open the *"Parameters"* tab and set the counting mode ↳ *Chapter 1.6.5.2.9.8.2.1 "Counting modes" on page 6071* of the *"Fast counter"* parameter.



3. In the *"I/O Mapping"* tab channel configuration is displayed. ↳ *Chapter 1.6.5.2.9.8.3 "Control of the fast counter" on page 6075*

Operands

Table 745: Input information

Description of the input information	Output information of the user program	Description	
Start value 1	Output double word 0	Double word	Set values for the counters 1 and 2: Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (depending on the operating mode either via a terminal or the bit SET within the control byte 1 or 2), the values of the double word variables are loaded into the counter 1 or 2.
Start value 2	Output double word 1	Double word	
End value 1	Output double word 2	Double word	End value for the counters 1 and 2: The end values for the two counters are stored as comparison values into the module by the user program. Both counters compare continuously whether or not their programmed end value is equal to their actual value. When the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.
End value 2	Output double word 3	Double word	

Description of the input information	Output information of the user program	Description	
Control byte 1 see 1)	Output byte 0	Byte: Bit 0 = UP/DWN Bit 1 = EN Bit 2 = SET Bit 3 = CF_HW Bit 4 to Bit 7 free	Control bytes for the counter 1: **UP/DWN**: In some operating modes, the counter can count downwards, too. If counting down is desired, set the bit UP/DWN to TRUE and the bit SET to 1. When doing so, the counter starts counting downwards from the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. **EN**: Processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **SET**: The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. **CF_HW** 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1.
Control byte 2 see 1)	Output byte 0	Byte: Bit 0 = UP/DWN Bit 1 = EN Bit 2 = SET Bit 3 to Bit 7 free	Control bytes for the counter 2: **UP/DWN**: In some operating modes, the counter can count downwards, too. If counting down is desired, set the bit UP/DWN to TRUE and the bit SET to 1. When doing so, the counter starts counting downwards from the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. **EN**: Processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte.

Description of the input information	Output information of the user program	Description	
			SET: The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table.

1) Only for CI581-CN/CI582-CN: Control bytes 1 and 2 are available twice on grounds of data consistency. Hence, a Start and End evaluation is only effected if the signals "Control Byte1_0" and "Control Byte1_1" or "Control Byte2_0" and "Control Byte2_1" (process image) are identical.

Table 746: Output information

Output information	Input information for the user program	Description	
Actual Value 0	Input double word 0	Double word	Actual value of the counter 0
Actual Value 1	Input double word 1	Double word	Actual value of the counter 1
Status Byte 0	Input byte 0	Byte: Bit 0 = CF Bit 1 to Bit 7 free	CF: When the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only when the counter is set again (set value), CF is reset to FALSE.
Status Byte 1	Input byte 1		

Operating modes

Inputs and outputs which are not used by the counters, are available for other tasks.

Legend:

- A refers to input channel A
- B refers to input channel B
- C refers to output channel C

Operating mode	Function	Used inputs and outputs	Notes
0	No counter	none	This operating mode is selected if the integrated fast counter is not necessary.
1	One count up counter	A = Counting input C = End value reached	The counting input and the output "End value reached) are enabled by the bit EN = TRUE within the control byte.

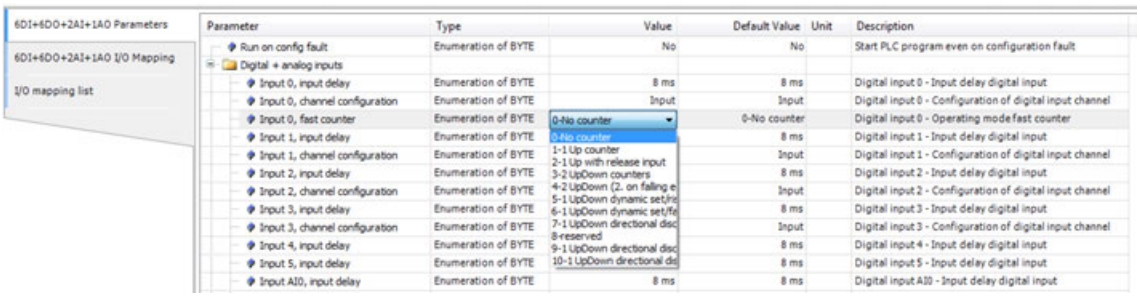
Operating mode	Function	Used inputs and outputs	Notes
2	One count up counter with enable input via terminal	A = Counting input B = Enable input C = End value reached	The enable input enables the counting input and the output "end value reached". The counter is only enabled if the enable input = TRUE (signal 1) AND the bit EN = TRUE within the control byte.
3	Two up/down counters	A = Counting input 0 B = Counting input 1	With this operating mode, two counters exist, which are independent of each other. The state "End value reached" is only readable from the two status bytes. It is not readable from output terminals. The counting direction is defined by the bit UP/DWN within the control byte.
4	Two up/down counters (1 counting input inverted)	A = Counting input 0 B = Counting input 1	This operating mode equals operating mode 3 with one exception: The counting input B (of counter 1) is inverted. It counts the TRUE/FALSE edges at input B.
5	One bidirectional counter with a dynamic set input via terminal	A = Counting input B = Dynamic set input	With this operating mode, one bidirectional counter is available which has a dynamic set input. Dynamic means that the set operation is performed at the FALSE/TRUE signal edge (0/1 edge) of the set input and not while the signal is TRUE. The state "End value reached" is only readable from the status byte, not from an output terminal.
6	One bidirectional counter with a dynamic set input via terminal	A = Counting input B = Dynamic set input	This operating mode equals operating mode 5 with one exception: The dynamic set input operates at the TRUE/FALSE edge (1-0 edge).

Operating mode	Function	Used inputs and outputs	Notes
7	One bidirectional counter for position sensors	A = Trace A of the position sensor B = Trace B of the position sensor	With this operating mode, incremental position sensors can be used which interchange their counting signals on tracks A and B in a 90° phase sequence. Depending on the sequence of the signals at A and B, the counter counts up or down. There is no pulse-multiplier function (e.g. x2 or x4). The position sensor must provide 24 V signals. Signals of 5 V sensors must be converted. Zero traces are not processed. The state "End value reached" is only readable from the state byte 0, not from an output terminal. The bit UP/DWN within the control byte must be FALSE. Otherwise, a parameter error occurrs. In this operating mode, the maximum counting frequency is: I/O modules 35 kHz. Communucation interface modules 50 kHz.
8	Reserved		

Operating mode	Function	Used inputs and outputs	Notes
9	One bidirectional counter for position sensors (pulse multiplier x2)	A = Trace A of the position sensor B = Trace B of the position sensor	This operating mode equals operating mode 7 with one exception: There is a pulse multiplication x2 with the evaluation of the counting inputs. This means, that the counter counts both the positive edges and the negative edges of trace A. This results in the double number of counting pulses. The precision increases correspondingly. In this operating mode, the maximum counting frequency is: I/O modules 30 kHz. Communucation interface modules 35 kHz.
10	One bidirectional counter for position sensors (pulse multiplier x4)	A = Trace A of the position sensor B = Trace B of the position sensor	This operating mode equals operating mode 7 with one exception: There is a pulse multiplication x4 with the evaluation of the counting inputs. This means that the counter counts the positive and negative edges of the traces A and B. This results in the fourfold number of counting pulses. The precision increases correspondingly. In this operating mode, the maximum counting frequency is: I/O modules 15 kHz. Communucation interface modules 20 kHz.

Configuration for onboard I/Os

1. In the device tree, double-click the *"Onboard I/O"* node (OBIO).
2. In the *"Parameters"* tab set the counting mode ↳ *Chapter 1.6.5.2.9.8.2.1 "Counting modes" on page 6071* for the fast counter.



3. In the *"I/O Mapping"* tab channel configuration is displayed. ↳ *Chapter 1.6.5.2.9.8.3 "Control of the fast counter" on page 6075*

Configuring the fast counter

The parameter of the fast counter channels of the Onboard I/O must be configured before they can be used. User should take these steps to configure the fast counter:

Channel	Direction	Width	Description
Actual value X	Input	DWORD	Current value of the fast counter.
State byte X	Input	BYTE	**Bit 0 = CF** If the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only, if the counter is set again (set value), CF is reset to FALSE. **Bit 1 to Bit 7 free**
Start value X	Output	DWORD	Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (dependent on the operating mode either via a terminal or the bit SET within the control byte X), the values of the double word variables are loaded into the counter X.
End value X	Output	DWORD	The end values for the two counters are stored as comparison values into the module by the user program. Both counters compare continuously, whether or not their programmed end value is equal to their actual value. If the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.

Channel	Direction	Width	Description
Control byte 1	Output	BYTE	**Bit 0 = UP/DWN** In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. When doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0, the counter jumps to 4,294,967,295. **Bit 1 = EN** The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **Bit 2 = SET** The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table). **Bit 3 = CF_HW** 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1. **Bit 4 to Bit 7 free**
Control byte 2	Output	BYTE	**Bit 0 = UP/DWN** In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. When doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0, the counter jumps to 4,294,967,295. **Bit 1 = EN** The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. **Bit 2 = SET** The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table). **Bit 3 to Bit 7 free**

Counting modes

The fast counter can be configured as one mode out of 10 possible modes. The desired operating mode is selected in the PLC configuration using configuration parameters. Inputs and outputs which are not used by the counter are available for other tasks. In the following table, A means input channel A, B means input channel B and C means output channel C.

CPUs	Integrated fast counter	Assigned inputs		Assigned Outputs	Remarks
		Channel A	Channel B	Channel C	
PM55x, PM56x	Yes	Input channel 0	Input channel 1	Output channel 0	Only 1 fast counter is available on the module. Input channel 0 is the default channel for fast counter. Input channel 1 can be used as another fast counter channel depending on fast counter mode.

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
0	No counter	None	Fast counter is disabled	-
1	1 count up counter	A = Counter input C = End value reached	Counting up A from 0 to 0xFFFFFFFF When the end value is reached, C will be set to high.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
2	1 count up counter with release input	A = Counter input B = Enable input C = End value reached	Counting up A from 0 to 0xFFFFFFFF The counter is enabled if B is high When the end value is reached, C will be set to high.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
3	2 Up/Down counters	A = Counter input 1 B = Counter input 2	2 independent counters. Status "End value reached" is only readable from the 2 status bytes, not from output terminals. The counting direction is defined by the Boolean parameters UD1 and UD2 of function block ONB_IO_CNT (Handle fast counter on Onboard I/O)	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
4	2 Up/Down counters (2nd on falling edges)	A = Counter input 1 B = Counter input 2	Same as operating mode 3, but counting input B is inverted (counts at TRUE/FALSE edges at input B).	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
5	1 Up/Down counter with dynamic set/ rising edge	A = Counter input B = Dynamic set input	1 Up/Down counter is available which counts on the rising edge of A and has a dynamic set input on B. Dynamic set input will set the start value at the rising edge of B.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)
6	1 Up/Down counter with dynamic set/ falling edge	A = Counter input B = Dynamic set input	1 Up/Down counter is available which counts on the rising edge of A and has a dynamic set input on B. Dynamic set input will set the start value at the falling edge of B.	30 kHz (before firmware V2.0.6) 50 kHz (since firmware V2.0.6)

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
7	1 UpDown directional discriminator	A = Phase A B = Phase B	With this mode, incremental encoders can be used which give their counting signals on phase A and B in a 90° phase sequence to each other. Dependent on the sequence of the signals at A and B, the counter counts up or down. There is no pulse multiplier function.	12 kHz (before firmware V2.0.6) 35 kHz (since firmware V2.0.6)
8	Reserved	-	-	-

Operating Mode	Function	Input channels	Description	Counting frequency (max.) for PM5x4-T and PM5x4-R
9	1 UpDown directional discriminator X2	A = Phase A B = Phase B	This mode is the same as mode 7 with one exception: There is a pulse multiplication x2 with the evaluation of the counting inputs. This means that the counter counts both the positive edges and the negative edges of phase A. This results in the double number of counting pulses. The precision increases correspondingly.	11 kHz (before firmware V2.0.6) 30 kHz (since firmware V2.0.6)
10	1 UpDown directional discriminator X4	A = Phase A B = Phase B	This mode is the same as mode 7 with one exception: There is a pulse multiplication x4 with the evaluation of the counting inputs. This means that the counter counts both the positive edges and the negative edges of phase A and B. This results in the fourfold number of counting pulses. The precision increases correspondingly.	10 kHz (before firmware V2.0.6) 15 kHz (since firmware V2.0.6)

If channel 0 is configured as fast counter, the other channels 1,2 and 3 cannot be configured as interrupt inputs. Otherwise, a configuration error will appear and the CPU will be stopped.

Control of the fast counter

To control the fast counter configuration open the *“I/O Mapping”* tab.

The channels can be mapped as described in Symbolic Names for Variables, Inputs and Outputs and have the following meaning ↳ *Chapter 1.6.5.2.9.6 “Symbolic names for variables, inputs and outputs” on page 6060*:

Fast counter						
		Actual value 1	%ID17	DWORD		Actual value 1
		Actual value 2	%ID18	DWORD		Actual value 2
		State byte 1	%IB76	BYTE		State byte 1
		State byte 2	%IB77	BYTE		State byte 2
		Start value 1	%QD11	DWORD		Start value 1
		End value 1	%QD12	DWORD		End value 1
		Start value 2	%QD13	DWORD		Start value 2
		End value 2	%QD14	DWORD		End value 2
		Control byte 1	%QB60	BYTE		Control byte 1
		Control byte 2	%QB61	BYTE		Control byte 2

Channel	Direction	Width	Description
Actual value X	Input	DWORD	Current value of the fast counter
State byte X	Input	BYTE	Bit 0 = CF If the counter reaches the programmed end value, the counter output is stored permanently as CF = TRUE (end value reached). Only if the counter is set again (set value), CF is reset to FALSE. Bit 1 to Bit 7 free
Start value X	Output	DWORD	Each counter can be set to a start value. Start values are loaded into the counter by the user program. Using the set signal (dependent on the operating mode either via a terminal or the bit SET within the control byte X), the values of the double word variables are loaded into the counter X.
End value X	Output	DWORD	The end values for the 2 counters are stored as comparison values into the module by the user program. Both counters compare continuously whether or not their programmed end value is equal to their actual value. When the counter (actual value) reaches its programmed end value, the binary output CF of the status byte is set permanently.

Channel	Direction	Width	Description
Control byte 1	Output	BYTE	Bit 0 = UP/DWN In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. If doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. Bit 1 = EN The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte. Bit 2 = SET The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. CF = 0 Bit 3 = CF_HW 0 = state of CF is set to hardware channel (only for mode 1 and 2) 1 = normal output is set to hardware channel Bit 3 is evaluated only in control byte of counter 1. Bit 4 to Bit 7 free
Control byte 2	Output	BYTE	Bit 0 = UP/DWN In some operating modes, the counter can count downwards, too. If counting down is desired, the bit UP/DWN must be set to TRUE. If doing so, the counter starts counting downwards at the start value (set value) to the end value (max. from 4,294,967,295 to 0 or hexadecimal from FF FF FF FF to 00 00 00 00). After reaching 0 the counter jumps to 4,294,967,295. Bit 1 = EN The processing of the counter signals must be enabled. Depending on the operating mode, enabling is done via a terminal or by the bit EN = TRUE within the control byte.

Channel	Direction	Width	Description
			Bit 2 = SET The counter can be set to a start value (see the description of the set values for the counters 1 and 2 at the beginning of this table. Bit 3 to Bit 7 free

1.6.5.2.10 CS31 fieldbus

Configuration of CS31 bus master

If in the device tree, for the *"Interfaces"* node the protocol 'CS31-Bus' is selected for the interface COM1, the interface is set as CS31 bus master. CS31 cannot be used as a slave. COM2 cannot be used as CS31 bus interface.

Some modules do not support CS31 functionality.

Double-click *"COMx_CS31_Bus"* node to configure the parameters.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	The PLC program is executed even if there are faults in configuration.
		Yes	The PLC program is not executed if there are faults in configuration.
Operating mode (read only)	Master	Master	Operating mode of the CS31 device. This parameter is read-only (not editable).
Max wait run	0		Max. wait time for valid inputs.
Min update time	10		Cycle time for data exchange to IEC program.

Slave modules can be appended to the CS31 bus master ↳ *Chapter 1.6.5.2.10.2 "Configuration of the slave modules" on page 6078*.

Configuration of the slave modules

DC551-CS31 and S500 I/O devices as slave modules

If in the device tree, a CS31 bus master has been configured ↳ *Chapter 1.6.5.2.10.1 "Configuration of CS31 bus master" on page 6078*, the DC551-CS31 device can be used as CS31 slave.

1. Right-click *"COMx_CS31_Bus ➔ Add object"*.
2. DC551-CS31 module is available in two versions in the PLC configuration:
   - DC551-CS31 8 DI + 16 DC / without fast counter:
     The addresses 00...61 can be set at the module and in the PLC configuration.
   - DC551-CS31 8 DI + 16 DC + 2FC / with 2 fast counters:
     The hardware addresses 70...99 can be set at the module. This corresponds to the module addresses 00...29 with activated counter. In the PLC configuration and at the block CNT_DC551, the module address (00..29) is set *Chapter 1.5.4.9 "Counter library" on page 1037*.

   Select the desired module from the list.
3. Define the parameters in the *"Parameters"* tab.

The following parameters are available:

Table 747: Available parameters

Parameter	Default value	Value	Description
Ignore module *Chapter 1.6.5.2.9.3 "Parameter 'Ignore module'" on page 6059*	No	No	It is checked whether the module exists on the CS31 Bus.
		Yes	Module is not checked. -> available as of CPU firmware V1.2.0 and PS501 V1.2
Module address	0	0...69	Module address of the DC551-CS31 without fast counter
		0...29	Module address of the DC551-CS31 with fast counter
Error-LED *"Failsafe function of CS31" on page 6090*	On	On	The error LED lights up for errors of all classes, no failsafe function activated.
		Off_by_E4	Warnings (E4) are not indicated by the error LED, no failsafe function activated.
		Off_by_E3	Warnings (E4) and minor errors (E3) are not indicated by the error LED. No failsafe function activated.
		On+failsafe	The error LED lights up for errors of all classes and the failsafe function of the CS31 Bus is activated. -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2

Parameter	Default value	Value	Description
		Off_by_E4+failsafe	Warnings (E4) are not indicated by the error LED, the failsafe function of the CS31 Bus is activated. -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Off_by_E3+failsafe	Warnings (E4) and minor errors (E3) are not indicated by the error LED, the failsafe function of the CS31 bus is activated. -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
Check supply	On	On	Control voltage monitoring ON
		Off	Control voltage monitoring OFF
Input delay	8 ms	0.1 / 1 / 8 / 32 ms	Input delay 0.1 / 1 / 8 / 32 ms
Fast counter *“No counter” on page 6090*	0-No counter	0-No counter	Operation mode of the *Chapter 1.6.4.1.10 “Fast counters” on page 5498*
Detection short-circuits at outputs	On	On	Output short-circuit detection ON
		Off	Output short-circuit detection OFF
Behaviour of outputs at communication fault *“Failsafe function of CS31” on page 6090*	Off	Off	Behavior of outputs at communication faults on the CS31 Bus OFF
		Last value	Last value -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Substitute value	Substitute value -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Last value 5 sec.	Last value for 5 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2

Parameter	Default value	Value	Description
		Substitute value 5 sec.	Substitute value for 5 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Last value 10 sec.	Last value for 10 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Substitute value 10 sec.	Substitute value for 10 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
Substitute value *“Failsafe function of CS31” on page 6090*	0	0...65535 0000hex...FFFFhex	Substitute value for the outputs, one bit per output, bit 0=C8 .. bit 15=C23 -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2

No counter

If a DC551 is configured as a fast counter module and in Automation Builder '0 - no Counter' is selected the channel ERR LEDs stays on and the module does not start up. The address was adjusted with '71'. Only the'0- no Counter' mode does not operate. If any other counter is selected e.g. '1-1 Up counter' the module starts up and can be utilized.

Failsafe function of CS31 bus

The setting of the parameter *“Behaviour”* of outputs in communication fault directly influences the failsafe function of the outputs of the S500 I/O modules.

S500/S500-eCo modules as slave modules

S500 / S500-eCo modules can be used as a slave module by appending the desired device to a configured DC551-CS31 device.

In the device tree, right-click the *“DC551-CS31”* node and select *“Add object”*. Select the desired S500 or S500-eCo module from the list.

When addressing S500 I/O devices at the CS31 bus, observe the following peculiarities concerning the CS31 bus in the AC500.

- A CS31 software module can occupy a maximum of -> 15 bytes of inputs and 15 bytes of outputs in the digital area. This corresponds to 15 x 8 = 120 digital inputs and 120 outputs.
- A CS31 software module can allocate a maximum of -> 8 words of inputs and 8 words of outputs in the analog area.
- A maximum of 31 of these CS31 software modules are allowed for connection to the CS31 bus.
- If a device has more than 15 bytes or 8 words of inputs or outputs, it occupies 2 or more of the 31 CS31 software modules.

- The DC551-CS31 can internally manage 2 CS31 software modules in the digital area and 5 CS31 software modules in the analog area. This corresponds to a maximum of: 240 digital inputs (2 x 15 bytes), 240 digital outputs (2 x 15 bytes), 40 analog inputs (5 x 8 words) and 40 analog outputs (5 x 8 words).
- Address setting is done at the DC551-CS31 using two rotary switches at the module's front plate.
- To enable the fast counter of the DC551-CS31, the hardware address (HW_ADR) has to be set to the module address + 70. With activated fast counter, the module addresses 0..28 (hardware address setting 70..98) are allowed. Then, the DC55-CS31 registers as 2 CS31 software modules using the module address (hardware address 70), once in the digital area and once in the analog area.
- CS31 software module 1 in digital area: Registers using the module address.
  CS31 software module 2 in digital area: Registers using module address+7 and bit "Channel >= 7" set.
  CS31 software module 1 in analog area: Registers using the module address.
  CS31 software module 2 in analog area: Registers using module address and bit "Channel >= 7" set.
  CS31 software module 3 in analog area: Registers using the module address+1.
  CS31 software module 4 in analog area: Registers using module address+1 and bit "Channel >= 7" set.
- The DC551-CS31 can manage a maximum of 255 parameters. This does not cause any restrictions in all configurations with the currently available S500 I/O Devices.
- The next free address for a DC551-CS31 is derived from the highest address occupied in the digital area or the analog area of the previous DC551-CS31.
- When connecting several S500 expansion modules to a DC551-CS31 via the I/O Bus, their inputs and outputs follow the DC551-CS31's inputs and outputs without gap. Such a cluster can occupy up to 6 CS31 software modules.

The fast counters of the input/output Modules (e.g. "DC532") are only available if the modules are connected to the CPU's I/O Bus.

Summary of input/output data

Table 748: Available input/output data and parameters with S500 I/O devices

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
CD522	1805	2 DI + 2 DI (encoder) + 8 DC + 2 DO PWM	6	7	8	8	6
DA501	1810	16 DI + 8 DC + 4 AI + 2 AO	3	1	5	2	8
DC551	2716	8 DI + 16 DC	3	2	0	0	15
DC551+FC	2715	8 DI + 16 DC + FC	5	4	4	8	16
AI523	1515	16 AI	0	0	16	0	36
AI531	1535	8 AI	0	0	8	1	36
AO523	1510	16 AO	0	0	0	16	41
AX521	1505	4 AI + 4 AO	0	0	4	4	23

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
AX522	1500	8 AI + 8 AO	0	0	8	8	39
DC522	1220	16 DC	2	2	0	0	8
DC523	1215	24 DC	3	3	0	0	10
DC532	1200	16 DI + 16 DC	4	2	0	0	8
DI524	1000	32 DI	4	0	0	0	4
DX522	1210	8 DI + 8 DX	1	1	0	0	6
DX531	1205	8 DI + 4 DX	1	1	0	0	6

Table 749: Available input/output data and parameters with S500-eCo I/O devices

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
DI561	6105	8 DI, 24 V DC	1	0	0	0	0
DI562	6110	16 DI, 24 V DC	2	0	0	0	0
DI571	6115	8 DI, 100-240 V AC	1	0	0	0	0
DO561	6120	8 DO 24 V DC, 0.5 A, transistor	0	1	0	0	0
DO571	6125	8 DO, up to 240 V AC/DC, 2.0 A, relay	0	1	0	0	1
DO572	6130	8 DO, 100-240 V AC, 0.5 A, triac	0	1	0	0	0
DX561	6135	8 DI 24 V DC + 8 DO 24 V DC, 0.5 A, transistor	1	1	0	0	0
DX571	6140	8 DI 24 V DC + 8 DO, up to 240 V AC/DC, 2.0 A, relay	1	1	0	0	1

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
DC561	6100	24 DC, 24 V DC, 0.1 A, Interfast connection	2	2	0	0	0
AI561	6500	4 AI, U/I. configurable	0	0	4	0	6
AI562	6505	2 AI, RTD, configurable	0	0	2	0	4
AI563	6510	4 AI, thermocouple, configurable	0	0	4	0	6
AO561	6515	2 AO, U/I, configurable	0	0	0	4	4
AX561	6520	4 AI, U/I + 2 AO, U/I, configurable	0	0	8	4	8

Impossible configurations

Due to the peculiarities concerning the CS31 bus and the DC551-CS31 described at the beginning of this chapter, some configurations cannot be realized.

Example: DC551-CS31 + 6 x DC532

This configuration is not possible because the DC551-CS31 can manage a maximum of 30 bytes in the digital area (= 120 inputs/outputs).

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
Total	**120 DI + 128 DC**	**31**	**16**	**0**	**0**	**78**

Example: DC551-CS31 + 6 (or more) AX522

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For 7 AX522, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
Total	**8 DI + 16 DC + 48 AI + 48 AO**	**3**	**2**	**48**	**48**	**255**

Example: DC551-CS31 + 3 (or more) AO523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + 48 AO**	**3**	**2**	**0**	**48**	**141**

Example: DC551-CS31 + 3 (or more) AI523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + 48 AI**	**3**	**2**	**48**	**0**	**126**

Example: DC551-CS31 with FC + 3 (or more) AO523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + FC + 48 AO**	**5**	**4**	**4**	**56**	**142**

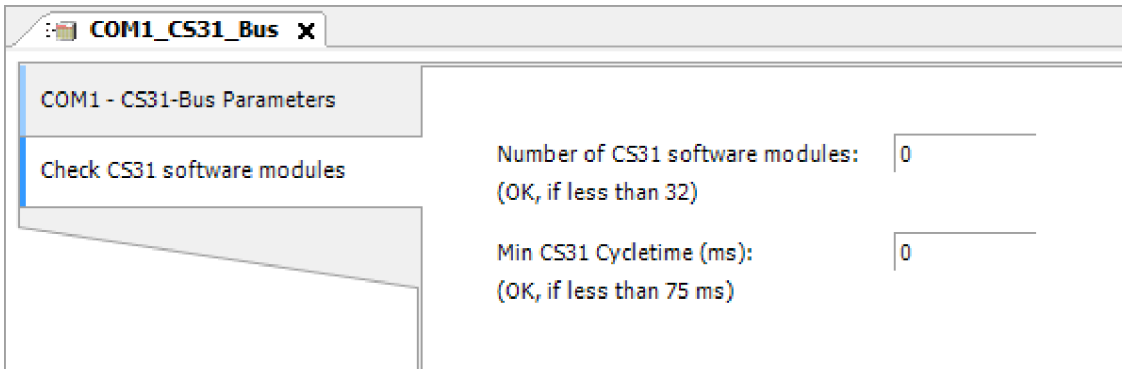
Example: DC551-CS31 with FC + 3 (or more) AI523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + FC + 48 AI**	**5**	**4**	**52**	**8**	**127**

Checking the CS31 modules

Double-click *"COMx_CS31_Bus"*:

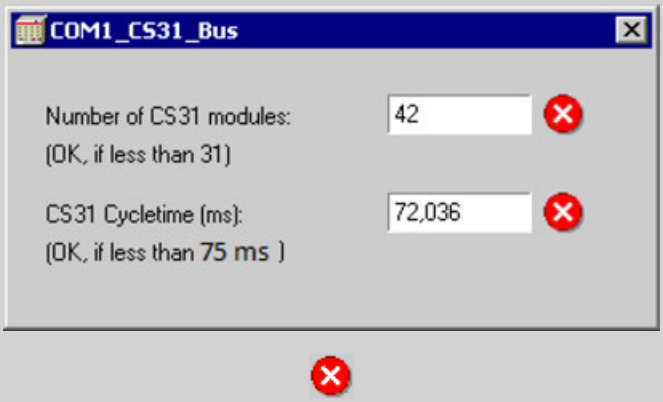


The following information is show in the view:

Parameter	Description
Number of CS31 software modules	The current number of configured CS31 modules at the CS31 master. It is not allowed to attach more than 31 slave devices to the bus.
Min. CS31 Cycletime (ms)	The predicted CS31 cycle time in [ms]. The value may not exceed 75 ms.

In case of a violation of the described parameters an error message appears.

Error message



The icon indicates the parameter that was violated.

The violation message does not block the download of an invalid configuration.

Connecting the DC551 and S500 I/O devices to the CS31 bus

The basic module DC551-CS31 is available in two versions in Automation Builder:

- DC551-CS31 8 DI + 16 DC / without fast counter
  The addresses 00...69 can be set at the module and in Automation Builder.
- DC551-CS31 8 DI + 16 DC + 2FC / with 2 fast counters
  The hardware addresses 70...99 can be set at the module. This corresponds to the module addresses 00...29 with activated counter. In Automation Builder and at the function block *Chapter 1.5.4.9.1.1 “CNT_DC551” on page 1037*, the module address (00..29) is set.

Table 750: Available parameters

Parameter	Default value	Value	Description
Ignore module *Chapter 1.6.5.2.9.3 “Parameter 'Ignore module'” on page 6059*	No	No	It is checked whether the module exists on the CS31 Bus.
		Yes	Module is not checked. -> available as of CPU firmware V1.2.0 and PS501 V1.2
Module address	0	0...69	Module address of the DC551-CS31 without fast counter
		0...29	Module address of the DC551-CS31 with fast counter
Error-LED *“Failsafe function of CS31” on page 6090*	On	On	The error LED lights up for errors of all classes, no failsafe function activated.
		Off_by_E4	Warnings (E4) are not indicated by the error LED, no failsafe function activated.

Parameter	Default value	Value	Description
		Off_by_E3	Warnings (E4) and minor errors (E3) are not indicated by the error LED. No failsafe function activated.
		On+failsafe	The error LED lights up for errors of all classes and the failsafe function of the CS31 Bus is activated. -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Off_by_E4+failsafe	Warnings (E4) are not indicated by the error LED, the failsafe function of the CS31 Bus is activated. -> available as of CPU firmware V1.2.0, DC551 firmware V1.9 and PS501 V1.2
		Off_by_E3+failsafe	Warnings (E4) and minor errors (E3) are not indicated by the error LED, the failsafe function of the CS31 bus is activated. -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
Check supply	On	On	Control voltage monitoring ON
		Off	Control voltage monitoring OFF
Input delay	8 ms	0.1 / 1 / 8 / 32 ms	Input delay 0.1 / 1 / 8 / 32 ms
Fast counter *“No counter” on page 6090*	0-No counter	0-No counter	Operation mode of the *Chapter 1.6.4.1.10 “Fast counters” on page 5498*
Detection short-circuits at outputs	On	On	Output short-circuit detection ON
		Off	Output short-circuit detection OFF
Behaviour of outputs at communication fault *“Failsafe function of CS31” on page 6090*	Off	Off	Behavior of outputs at communication faults on the CS31 Bus OFF

Parameter	Default value	Value	Description
		Last value	Last value -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Substitute value	Substitute value -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Last value 5 sec.	Last value for 5 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Substitute value 5 sec.	Substitute value for 5 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Last value 10 sec.	Last value for 10 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
		Substitute value 10 sec.	Substitute value for 10 seconds -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2
Substitute value *“Failsafe function of CS31” on page 6090*	0	0...65535 0000hex...FFFFhex	Substitute value for the outputs, one bit per output, bit 0=C8 .. bit 15=C23 -> available as of CPU firmware V1.2.0, DC551-CS31 firmware V1.9 and PS501 V1.2

No counter

If a DC551 is configured as a fast counter module and in Automation Builder '0 - no Counter' is selected the channel ERR LEDs stays on and the module does not start up. The address was adjusted with '71'. Only the'0- no Counter' mode does not operate. If any other counter is selected e.g. '1-1 Up counter' the module starts up and can be utilized.

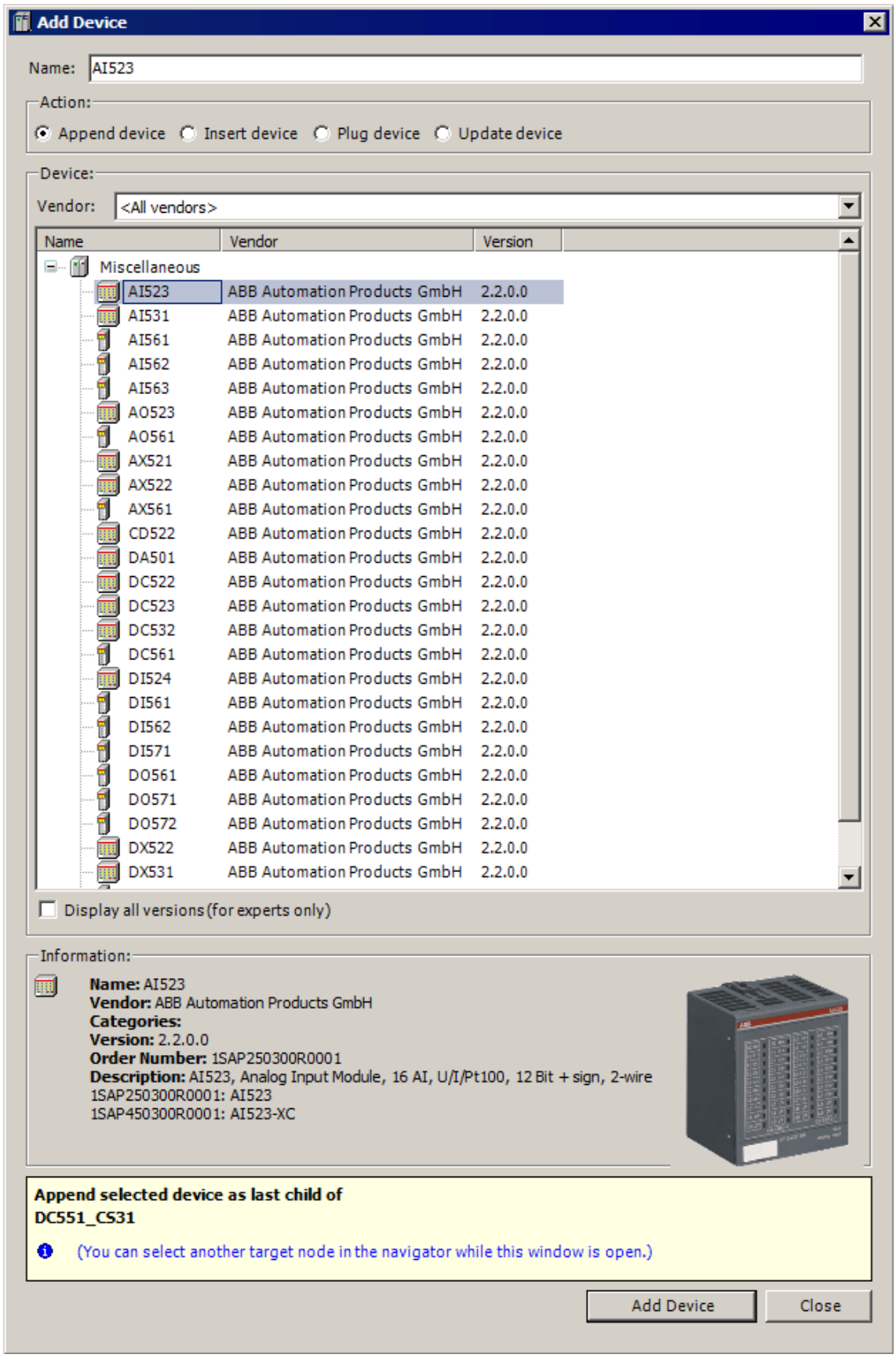
Failsafe function of CS31

The setting of the parameter Behaviour of outputs in communication fault directly influences the failsafe function of the outputs of the S500 I/O devices.

To connect further S500 I/O Modules to the basic module DC551-CS31 via the I/O Bus, see the following steps.

1. Right-click the element DC551-CS31 in the configuration tree and select the menu item *"Add device"*.

   ⇨ The *"Add Device"* dialog window appears. It shows all available I/O Modules that can be added to DC551-CS31.



2. Choose the modules you want to attach to DC551-CS31.

A maximum of 7 expansions with a total of 240 DI / 240 DO and 40 AI / 40 AO can be appended to the module.

When addressing S500 I/O devices at the CS31 bus, observe the following peculiarities concerning the CS31 bus in the AC500.

- A CS31 software module can occupy a maximum of -> 15 bytes of inputs and 15 bytes of outputs in the digital area. This corresponds to 15 x 8 = 120 digital inputs and 120 outputs.
- A CS31 software module can allocate a maximum of -> 8 words of inputs and 8 words of outputs in the analog area.
- A maximum of 31 of these CS31 software modules are allowed for connection to the CS31 bus.
- If a device has more than 15 bytes or 8 words of inputs or outputs, it occupies 2 or more of the 31 CS31 software modules.
- The DC551-CS31 can internally manage 2 CS31 software modules in the digital area and 5 CS31 software modules in the analog area. This corresponds to a maximum of: 240 digital inputs (2 x 15 bytes), 240 digital outputs (2 x 15 bytes), 40 analog inputs (5 x 8 words) and 40 analog outputs (5 x 8 words).
- Address setting is done at the DC551-CS31 using two rotary switches at the module's front plate.
- To enable the fast counter of the DC551-CS31, the hardware address (HW_ADR) has to be set to the module address + 70. With activated fast counter, the module addresses 0..28 (hardware address setting 70..98) are allowed. Then, the DC55-CS31 registers as 2 CS31 software modules using the module address (hardware address 70), once in the digital area and once in the analog area.
- CS31 software module 1 in digital area: Registers using the module address.

  CS31 software module 2 in digital area: Registers using module address+7 and bit "Channel >= 7" set.

  CS31 software module 1 in analog area: Registers using the module address.

  CS31 software module 2 in analog area: Registers using module address and bit "Channel >= 7" set.

  CS31 software module 3 in analog area: Registers using the module address+1.

  CS31 software module 4 in analog area: Registers using module address+1 and bit "Channel >= 7" set.
- The DC551-CS31 can manage a maximum of 255 parameters. This does not cause any restrictions in all configurations with the currently available S500 I/O Devices.
- The next free address for a DC551-CS31 is derived from the highest address occupied in the digital area or the analog area of the previous DC551-CS31.
- When connecting several S500 expansion modules to a DC551-CS31 via the I/O Bus, their inputs and outputs follow the DC551-CS31's inputs and outputs without gap. Such a cluster can occupy up to 6 CS31 software modules.
- A maximum of 7 S500 expansion modules (extensions) can be connected to a DC551-CS31.

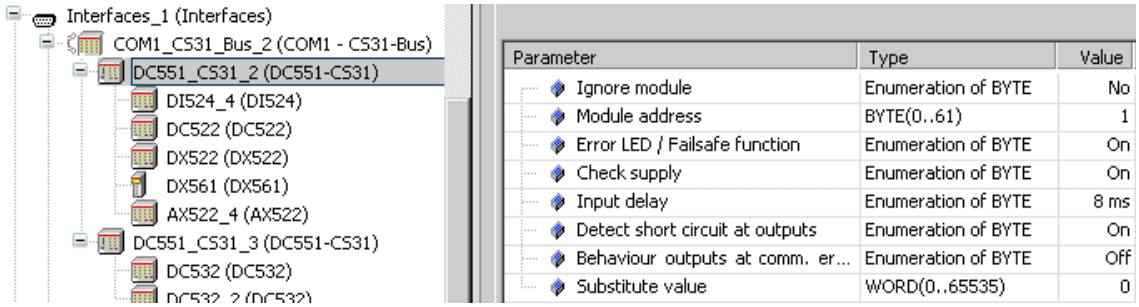


Fig. 1165: Possible configuration consisting of two combined input/output modules.

The fast counters of the input/output Modules (e.g. "DC532") are only available if the modules are connected to the CPU's I/O Bus.

Summary of input/output data

Table 751: Available input/output data and parameters with S500 I/O devices

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
CD522	1805	2 DI + 2 DI (encoder) + 8 DC + 2 DO PWM	6	7	8	8	6
DA501	1810	16 DI + 8 DC + 4 AI + 2 AO	3	1	5	2	8
DC551	2716	8 DI + 16 DC	3	2	0	0	15
DC551+FC	2715	8 DI + 16 DC + FC	5	4	4	8	16
AI523	1515	16 AI	0	0	16	0	36
AI531	1535	8 AI	0	0	8	1	36
AO523	1510	16 AO	0	0	0	16	41
AX521	1505	4 AI + 4 AO	0	0	4	4	23
AX522	1500	8 AI + 8 AO	0	0	8	8	39
DC522	1220	16 DC	2	2	0	0	8
DC523	1215	24 DC	3	3	0	0	10
DC532	1200	16 DI + 16 DC	4	2	0	0	8
DI524	1000	32 DI	4	0	0	0	4
DX522	1210	8 DI + 8 DX	1	1	0	0	6
DX531	1205	8 DI + 4 DX	1	1	0	0	6

Table 752: Available input/output data and parameters with S500-eCo I/O devices

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
DI561	6105	8 DI, 24 V DC	1	0	0	0	0
DI562	6110	16 DI, 24 V DC	2	0	0	0	0

Device	ID	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
			Inputs	Outputs	Inputs	Outputs	
DI571	6115	8 DI, 100-240 V AC	1	0	0	0	0
DO561	6120	8 DO 24 V DC, 0.5 A, transistor	0	1	0	0	0
DO571	6125	8 DO, up to 240 V AC/DC, 2.0 A, relay	0	1	0	0	1
DO572	6130	8 DO, 100-240 V AC, 0.5 A, triac	0	1	0	0	0
DX561	6135	8 DI 24 V DC + 8 DO 24 V DC, 0.5 A, transistor	1	1	0	0	0
DX571	6140	8 DI 24 V DC + 8 DO, up to 240 V AC/DC, 2.0 A, relay	1	1	0	0	1
DC561	6100	24 DC, 24 V DC, 0.1 A, Inter-fast connection	2	2	0	0	0
AI561	6500	4 AI, U/I. configurable	0	0	4	0	6
AI562	6505	2 AI, RTD, configurable	0	0	2	0	4
AI563	6510	4 AI, thermocouple, configurable	0	0	4	0	6
AO561	6515	2 AO, U/I, configurable	0	0	0	4	4
AX561	6520	4 AI, U/I + 2 AO, U/I, configurable	0	0	8	4	8

Impossible configurations

Due to the peculiarities concerning the CS31 bus and the DC551-CS31 described at the beginning of this chapter, some configurations cannot be realized.

Example: DC551-CS31 + 6 x DC532

This configuration is not possible because the DC551-CS31 can manage a maximum of 30 bytes in the digital area (= 120 inputs/outputs).

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
DC532	16 DI + 16 DC	4	2	0	0	9
Total	**120 DI + 128 DC**	**31**	**16**	**0**	**0**	**78**

Example: DC551-CS31 + 6 (or more) AX522

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For 7 AX522, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
AX522	8 AI + 8 AO	0	0	8	8	40
Total	**8 DI + 16 DC + 48 AI + 48 AO**	**3**	**2**	**48**	**48**	**255**

Example: DC551-CS31 + 3 (or more) AO523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + 48 AO**	**3**	**2**	**0**	**48**	**141**

Example: DC551-CS31 + 3 (or more) AI523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC	3	2	0	0	15
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + 48 AI**	**3**	**2**	**48**	**0**	**126**

Example: DC551-CS31 with FC + 3 (or more) AO523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AO523, the number of analog channels increases accordingly.

Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
AO523	16 AO	0	0	0	16	42
Total	**8 DI + 16 DC + FC + 48 AO**	**5**	**4**	**4**	**56**	**142**

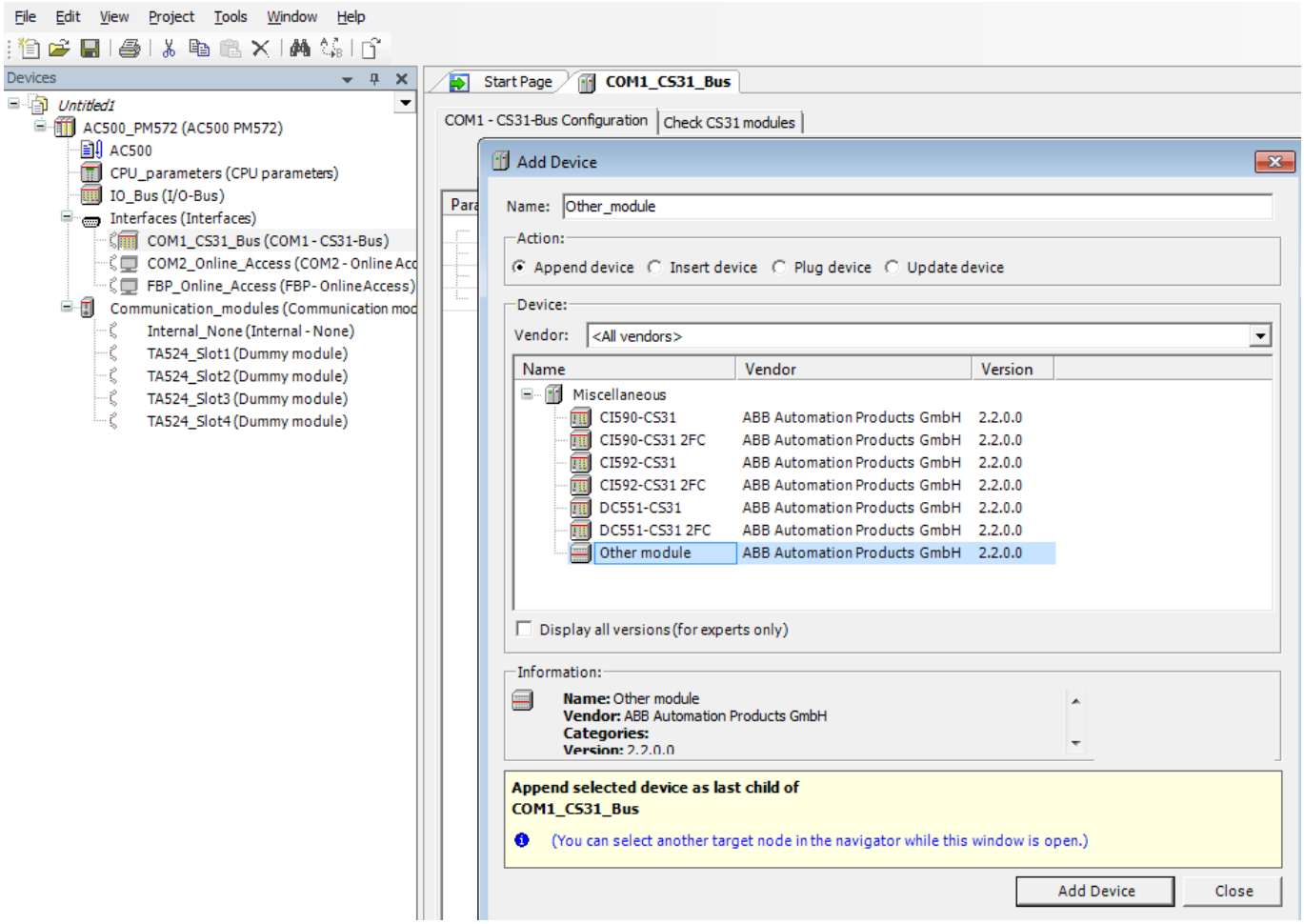
Example: DC551-CS31 with FC + 3 (or more) AI523

This configuration is not possible because the DC551-CS31 can manage a maximum of 40 words in the analog area. For each further AI523, the number of analog channels increases accordingly.

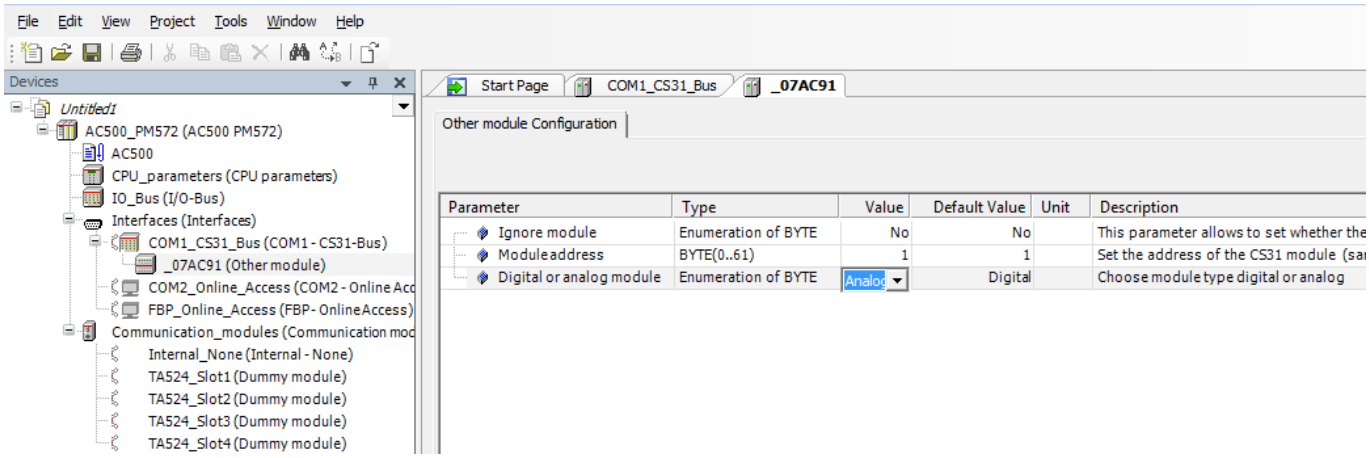
Device	I/O range	Digital area [Byte]		Analog area [Words]		Parameter [Byte]
		Inputs	Outputs	Inputs	Outputs	
DC551	8 DI + 16 DC + FC	5	4	4	8	16
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
AI523	16 AI	0	0	16	0	37
Total	**8 DI + 16 DC + FC + 48 AI**	**5**	**4**	**52**	**8**	**127**

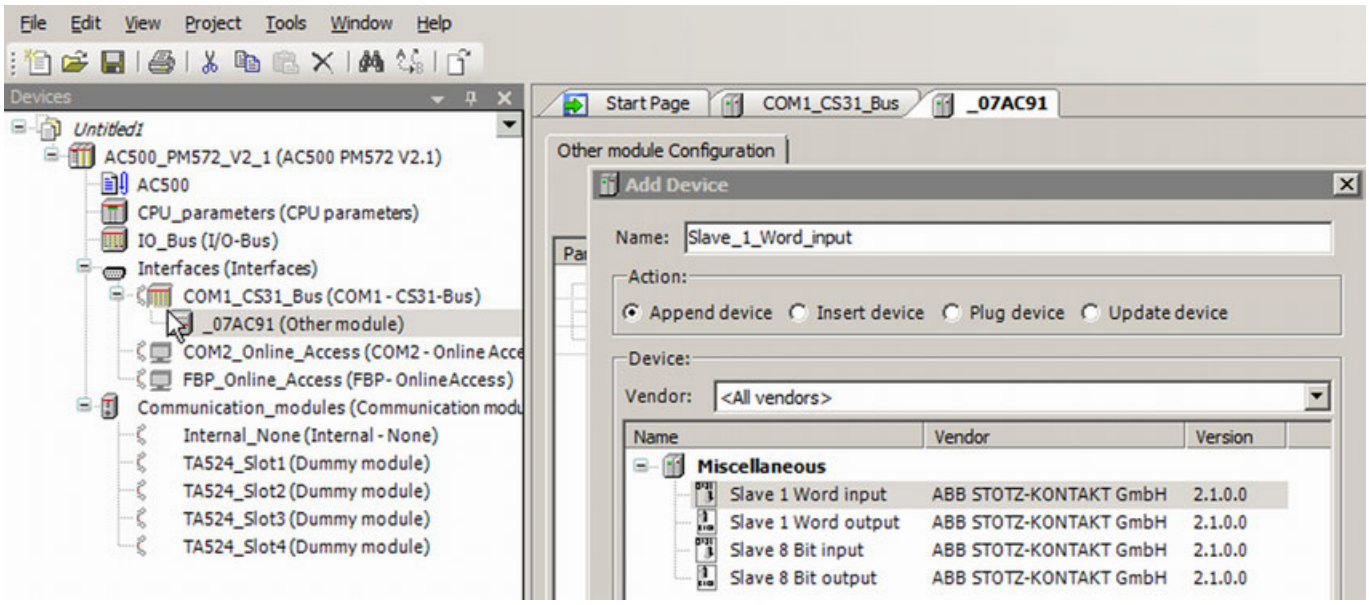
Configuration of other module

1. Right-click on COM1_CS31_Bus => Context menu opens.
2. Select Add Device => menu Add Device opens => select Other module.
3. Click button Add Device.



1. Double-click the moduls name and start configuration.
2. Double-click Digital in column Value =>set Digital or Analog.
3. Right-click the moduls name (Other module) => Add Device => Add Device menu opens.
4. Select In-/Outputs by double-clicking.





1.6.5.2.11 Serial interfaces COM1 and COM2

Setting up the protocol of a serial interface

AC500 CPUs PM57x, PM58x and PM59x are equipped with the two interfaces COM1 and COM2 which can be operated as RS-232 and RS-485.

Some AC500 CPUs such as PM56xy-2ETH are equipped with interface COM1 only.

> *RS-485 operation of an interface is only possible if the parameter RTS control is set to Telegram. The cabling of RS-232 or RS-485 is detected automatically by the hardware.*

AC500 PM55x and PM56x CPUs are equipped with two interfaces COM1 and COM2 (optional) which can be operated as RS-485 only. The parameter RTS control is set by default to Telegram and can not be changed.

Setting COMx - Online access

By default, the serial interfaces are set to 'Online access'. That means the access is done with help of Automation Builder.

If the default mode is set to another value, the interface protocol is directly set in the PLC configuration. No block (such as MODINIT, COMINIT) is required.

The serial interface settings can be read in online mode using the PLC browser commands "com settings" and "com protocols" ↳ *Chapter 1.6.5.4.3 “AC500-specific PLC browser commands” on page 6222*.

Configuration

Double-click *“COMx_Online_Access”* to open the COMx - Online Access Configuration in the editor window.

The following parameters are available:

Parameter	Default value	Value	Description
RTS control	Telegram	None	No RTS control (direction control, not for PM55x and PM56x).
		Telegram	RTS control activated (absolutely necessary for programming via RS-485!).
		RTS/CTS (DTE<->DTE)	RTS and CTS signals are controlled to simulate a DTE device at the PLC communicating with a DTE device.
		RTS/CTS (DTE-->DCE)	RTS and CTS signals are controlled to simulate a DTE device at the PLC communicating with a DCE device.
		RTS/CTS (DCE<--DTE)	RTS and CTS signals are controlled to simulate a DCE device at the PLC communicating with a DTE device.
Transmission rate	19200	300 1200 4800 9600 14400 19200 38400 57600 115200	Transmission rate (Baude)
Parity (read only)	None	None	Parity setting for the Online Access. This parameter is read-only (not editable).
Data bits (read only)	8	8	Number of data bits. This parameter is read-only (not editable).
Stop bits (read only)	1	1	Number of stop bits. This parameter is read-only (not editable).
Run on config fault	No	No	The PLC program is not executed if there are faults in configuration.
		Yes	The PLC program is executed even if there are faults in configuration.

The serial interface settings must match the settings for the serial gateway driver. Refer to the description for programming via the serial interfaces *Chapter 1.6.5.4.2.2 “Programming via the serial interfaces” on page 6200*.

Setting COMx - ASCII

With the selection ASCII, the initialization of the serial interface is done for the "free protocol", i.e. all interface parameters can be set and any protocol can be realized.

Sending and receiving data is done by means of the function blocks COM_SEND and COM_REC which are contained in *Chapter 1.5.4.3 “ASCII communication library” on page 783*.

To be able to receive data using the function block COM_REC, a buffer of the size 272 bytes must be available (for example abyRecData : ARRAY[0..271] OF BYTE).

This is also required if only short telegrams are received.

The operating system provides a total of 32 buffers with 272 bytes each for the transmission and reception of data. If the PLC is in STOP mode (= pause) or the input EN at the block COM_REC is set to FALSE or the function block is not called, these buffers run full.

If the function block COM_REC is called again (with EN = TRUE) before all buffers are used, the data received meanwhile are available.

If all buffers were full, the error Invalid handle with ERR = TRUE and ERNO = 16#2001 = 8193 is reported for one cycle. After this the reception is reset.

Receiving is always reset after a download or the command Online/Reset.

Configuration

Right-click *“COMx_Online_Access ➔ Add object”* and select *“COMx_ASCII ”* from the list.

The parameters define how the serial interface will be initialized. The parameters can be grouped. They are used to initialize the following functions:

- Monitoring the programming login:
  Enable login
- Modem control and RS-485:
  RTS control, TLS, CDLY
- Recognition of telegram ending for reception:
  Character timeout, Telegram ending selection, Telegram ending value, Telegram ending character
- Checksum
- Transmission parameters:
  Transmission rate, Parity, Data bits, Stop bits

The following parameters are available:

Parameter	Default value	Value	Description
Enable login *Chapter 1.6.5.2.11.3.1 “Enable login” on page 6102*	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to Online access.

Parameter	Default value	Value	Description
RTS control s *Chapter 1.6.5.2.11.3.2 "Usage of modems" on page 6103*	None	None	No RTS control (direction control, not for PM55x and PM56x CPUs)
		Telegram	RTS control activated (absolutely necessary for RS-485!)
		RTS/CTS (DTE <--> DTE)	RTS and CTS signals are controlled to simulate a DTE device at the PLC communicating with a DTE device.
		RTS/CTS (DTE --> DCE)	RTS and CTS signals are controlled to simulate a DTE device at the PLC communicating with a DCE device.
		RTS/CTS (DCE <-- DTE)	RTS and CTS signals are controlled to simulate a DCE device at the PLC communicating with a DTE device.
TLS *Chapter 1.6.5.2.11.3.2 "Usage of modems" on page 6103*	0	0...65535	For AC500 CPUs and CM574-RS: Carrier lead time in [ms] (TLS > CDLY)
		0...850	For CI504-PNIO and CI506-PNIO: Carrier lead time in [ms] (TLS > CDLY)
CDLY *Chapter 1.6.5.2.11.3.2 "Usage of modems" on page 6103*	0	0...65535	For AC500 CPUs and CM574-RS: Carrier delay time in [ms] (CDLY <= TLS)
		0...850	For CI504-PNIO and CI506-PNIO: Carrier delay time in [ms] (CDLY <= TLS)
Character timeout *Chapter 1.6.5.2.11.3.3 "Telegram ending identifier" on page 6104*	0	0...65535	For AC500 CPUs and CM574-RS: Character timeout in characters (must be 0 if Telegram ending selection = Character timeout)
		0/32	For CI504-PNIO and CI506-PNIO: Character timeout in bits (must be 0 if Telegram ending selection = Character timeout)
Telegram ending selection *Chapter 1.6.5.2.11.3.3 "Telegram ending identifier" on page 6104* and *Chapter 1.6.5.2.11.3.4 "Checksum" on page 6107*	none	none	No telegram ending identifier
		String (check receive)	2 characters, e.g. <CR><LF> (16#0d, 16#0a -> 16#0d0a) in parameter "Telegram ending value" (not supported with COM2 of AC500-eCo CPUs)
		Telegram length	Telegram ending identifier set by telegram length
		Duration	Telegram ending identifier set by time (not supported with CI504-PNIO and CI506-PNIO)
		Character timeout	Telegram ending identifier set by character timeout
Telegram ending character *Chapter 1.6.5.2.11.3.3 "Telegram ending identifier" on page 6104*	16#0d	0...255	Up to version V1.1.x: Telegram ending character
	0	0...2	As of version V1.2.0: Number of end characters in case of telegram ending selection "String"

Parameter	Default value	Value	Description
Telegram ending value ↳ *Chapter 1.6.5.2.11.3.3 "Telegram ending identifier" on page 6104*	0	0...65535	Up to version V1.1.x: Telegram ending identifier value for settings "Duration" and "Character timeout"
	0	0...65535	As of version V1.2.0: Telegram ending identifier value for settings "Duration", "Character timeout" and "String"
Checksum ↳ *Chapter 1.6.5.2.11.3.4 "Checksum" on page 6107*	none	None	No checksum
		CRC8	CRC8 checksum
		CRC16	CRC16 checksum (Motorola format)
		LRC	Add all values to byte (ignore overflow), result multiplied by -1
		ADD	Add all values to byte (ignore overflow)
		CS31	CS31 bus checksum
		CRC8-FBP	CRC8 FBP field bus neutral protocol
		XOR	XOR all values to byte (ignore overflow)
		CRC16 (Intel)	Like CRC16, result swapped
Transmission rate	19200	300 1200 4800 9600 14400 19200 38400 57600 115200	Transmission rate (Baud)
Parity	none	None	No parity check
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE (Not supported with CI504-PNIO and CI506-PNIO)
		Space	Parity bit := FALSE (Not supported with CI504-PNIO and CI506-PNIO)
Data bits	8	5, 6, 7, 8	Character length in bits/character
Stop bits	1	1, 2	Number of stop bits

Enable login

If Enable login is set to Yes, all received telegrams are checked with regard to the CODESYS login service.

It is recommended to activate the automatic login detection only for those projects for which this function is absolutely required because it slows down communication via the serial interface and also influences the PLC performance.

If the connection is directly made via RS-232, a login telegram will only be detected if the same parameters as used by CODESYS (Transmission rate=19200 Baud, Stop bits=1, Parity=None, Data bits=8 bit) are set when initializing the interface.

The same applies if the connection is made via RS-232/RS-485 interface converters. The login telegram can only be detected if the initialization parameters have the same values as the parameters set in CODESYS. Because for such an application usually more than one devices are connected to the RS-485 transmission line the following has to be observed additionally:

The CODESYS login telegram does not contain a device address. Thus, the service is first identified by all devices connected to the RS-485 transmission line that can be programmed using CODESYS and the interface of which is able to read the login telegram (interface with parameter Enable login set to Yes). Due to this, telegram collisions can occur during the subsequent acknowledgement of the login request by these devices, resulting in an interruption of the communication.

If the connection between CODESYS and the PLC is established via modem, the communication is not influenced by the interface parameters set in the PLC configuration. The parameter values required for the modem used have to be set. Once the initialization is completed, the mode processes the received telegrams according to the parameter settings. Also the assignment between login request and an individual PLC is guaranteed because the connection is established using the modem's phone number or MSN.

The login with CODESYS first causes a reinitialization of the interface. All blocks accessing this interface are locked during the online session, i.e. they do not perform any function. During this period the block outputs have the following values:

DONE = FALSE

ERR = TRUE

ERNO = PROTOCOL_PROTECTED = 16#301F = 12319

The blocks will be reactivated after the logout by CODESYS.

The login monitoring for an interface is only done if CODESYS is not already logged in via another interface (Ethernet, ARCNET or other COM).

Usage of modems

The ASCII protocol considers the special properties of modems, interface converters and repeaters. If these devices are used at a serial interface operated in 'free mode', the compression mechanism possibly supported by these devices has to be deactivated. For detailed information, refer to the operation manual of the used device.

Some repeaters, modems or interface converters require a control signal in order to set the transfer direction. The direction control can be enabled or disabled via the input RTSCTRL.

Various devices additionally require a lead time to stabilize their carrier signal. These devices can only transfer data in send direction after this time has elapsed. This carrier lead time can be set via the input TLS.

Additionally, for some devices it is necessary to sustain the carrier signal in send direction for some time after data transfer is completed. Only if this time has elapsed, the complete transfer of a telegram is ensured and the devices are ready for data transfer in opposite direction. This carrier delay time can be set via the input CDLY.

Carrier lead time (TLS) and carrier delay time (CDLY) must be adjusted for all communication devices connected to the same transmission line.

The times are only considered for RTS control set to telegram.

For CI504-PNIO and CI506-PNIO: For the time settings of TLS/CDLY, the used transmission rate listed in the following table has to be considered:

Transmission rate [Bd]	Max. value for TLS/CDLY [ms]
300	850
1200	212
2400	106
4800	53
9600	26
14400	17
19200	13
38400	6
57600	4
115200	2

Telegram ending identifier

The telegram ending identifier is set using the parameters Character timeout, Telegram ending selection, Telegram ending character and Telegram ending value.

Character silent time monitoring:

Monitoring of the character timeout can be set for all possible telegram ending settings (except Character timeout).

If the parameter Character timeout = 0, no character timeout monitoring is done.

With Character timeout > 0 the character timeout monitoring is activated.

The character silent time is defined in number of characters. The number of characters and the interface parameters (Transmission rate, Parity, Data bits and Stop bits) are used to calculate the silent time.

Example: Transmission rate=9600 Baud, Parity=none, Data bits=8, Stop bits=1, Character timeout=3

This results in a frame of 10 bits/character:

1 start bit + 8 data bits + 0 parity bit + 1 stop bit

Character silent time = 1000 x Character timeout x Frame / Transmission rate [ms]

Character timeout = 1000 x 3 x 10 / 9600 = 3.125 ms ~ 4 ms.

If the time between the reception of two characters exceeds the character silent time, the reception is aborted with an error and the characters received up to this moment are made available.

The following parameter combinations are possible:

Character timeout **see remark on character silent time monitoring**	**Telegram ending selection** **Type of telegram ending identifier**	**Telegram ending character = ignored**	**Telegram ending value = ignored**	**Description**
Number of characters 0 or > 0	None	-	-	No telegram ending identifier, i.e., the characters received since last call are provided. The maximum number of characters is limited to 256. For CI504-PNIO and CI506-PNIO: The data will only be provided after 256 characters have been received.
Number of characters 0 or > 0	String (check receive)	Number of telegram ending characters 1 or 2	2 characters (for example 16#0d0a)	According to value set for "Telegram ending character", it is checked for 1 or 2 ending characters. The ending character(s) is (are) not passed, i.e., they are not contained in DATA area.
	1	1	16#0d = 13dec = <CR>	After reception of 16#0d, telegram received is reported.
	2	2	16#0d0a = 3338dec = <CR><LF>	After reception of 16#0d and subsequently 16#0a, telegram received is reported.
Number of characters 0 or > 0	Telegram length	-	Number of characters > 0 and ≤ 256	Telegram received is reported once the number of characters defined in "Telegram ending value" is received.

Character timeout **see remark on character silent time monitoring**	**Telegram ending selection** **Type of telegram ending identifier**	**Telegram ending character = ignored**	**Telegram ending value = ignored**	**Description**
Number of characters 0 or > 0	Duration	-	Time in [ms]	Telegram received is reported once the time set for "Telegram ending value" (in [ms]) is elapsed. The time starts with the first FALSE -> TRUE edge at input EN of the receive block COM_REC.
0	Character timeout	-	For Onboard COM: Number of characters > 0 and ≤ 256	The number of characters set for Telegram ending value and the interface parameters (Transmission rate, Parity, Data bits and Stop bits) are used to calculate the silent time. Telegram received is reported if the silent time between two characters is ≥ the calculated silent time.
			For CI504-PNIO and CI506-PNIO: Number of bits (0 or 32)	The number of bits set for Telegram ending value define the silent time. Telegram received is reported if the silent time between 2 characters is ≥ the silent time.

AC500-eCo processor module do not support string (check receive) and character timeout at COM1 and COM2 as telegram ending selection.

Checksum

Sending with COM_SEND

With Checksum < > none, the selected checksum is appended when sending. If the parameter Telegram ending selection is set to String (check receive), the checksum of the ending character(s) is entered. The character(s) is/are appended according to the inputs END_LEN and END_CH of the function block COM_SEND.

Receiving with COM_REC

With Checksum < > none, the selected checksum is checked during reception. If the parameter Telegram ending selection is set to String (check receive), the checksum of the ending character(s) is expected.

The ending character(s) and the checksum are not output, i.e., they are not contained in the DATA area.

A telegram should look as follows:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**Check 1**	**[Check 2]**	**End 1**	**[End 2]**

The values enclosed in [ ] are only relevant for 16 bit checksum or 2 ending characters.

At the blocks COM_SEND and COM_REC, the area addressed via the input DATA contains the following values:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

Example

Setting in PLC configuration:

"Telegram ending selection" = String

"Telegram ending character" = 2

"Telegram ending value" = 16#0d0a

"Checksum" = CRC16 (i.e., Motorola format)

Send with COM_SEND:

LEN = n+1 END_LEN = 2 END_CH = 16#0d0a

The area addressed via input DATA contains the following data:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

The following data are sent via the interface:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**CRC16 high**	**CRC16 low**	**16#0d**	**16#0a**

Reception with COM_REC:

The interface receives the following telegram:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**	**CRC16 high**	**CRC16 low**	**16#0d**	**16#0a**

The following data are written to the area addressed via DATA:

Data 0	**Data 1**	**Data 2**	**..**	**Data n**

Setting COMx - Modbus

Right-click *“COMx_Online_Access ➔ Add object ➔ COMx_MODBUS”*.

The following parameters are available:

Parameter	Default value	Value	Description
Enable login	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to 'Online access'. -> available as of firmware V1.2.0
RTS control	None	None	No RTS control (not for PM55x and PM56x CPUs)
		Telegram	RTS control for telegram activated -> available as of firmware V1.2.0
TLS	0	0...65535	Carrier lead time in [ms] or characters (TLS > CDLY) -> available as of firmware V1.2.0
CDLY	0	0...65535	Carrier delay time in [ms] or characters (CDLY <= TLS) -> available as of firmware V1.2.0
Telegram ending value	3	0...65535	Number of characters for character timeout
Handshake	None	None	No flow control
		RTS/CTS	Hardware handshake -> available as of firmware V1.2.0
		XON/XOFF	Software handshake -> Not implemented

Parameter	Default value	Value	Description
Transmission rate	19200	300 1200 4800 9600 14400 19200 38400 57600 115200	Transmission rate (Baud)
Parity	Even	None	No parity
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE
		Space	Parity bit := FALSE
Data bits	8	5, 6, 7, 8	Number of data bits, 5 to 8
Stop bits	1	1, 2	Number of stop bits, 1 or 2
Operation mode	None	None	None
		Server	Server
		Client	Client
Address	0	0...255	Address for Modbus slave
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x
Disable read to %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x
Disable read to %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x
Disable read to %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x
Disable read to %MB1.x to	0	0...65535	Disable read access for segment 1 up to %MB1.x

At AC500-eCo Processor Modules the parameter "Parity" cannot be configured as mark and/or space.

For Modbus slave operation, an area without read and/or write access can be set in the segments %M0.x and %M1.x. reading/writing is disabled beginning at the set address and is valid up to the set end address (inclusive).

The parameter "Data bits" always has to be set to 8 for Modbus.

Setting COMx - CS31

Right-click *"COMx_Online_Access ➔ Add object ➔ COMx_CS31"*.

The serial interface is definitely set as CS31 bus master. COM2 cannot be used as CS31 bus interface.

The following parameters are available:

Parameter	Default value	Value	Description
Run on config fault	No	No	The PLC program is executed even if there are faults in configuration.
		Yes	The PLC program is not executed if there are faults in configuration.
Operating mode (read only)	Master	Master	Operating mode of the CS31 device. This parameter is read-only (not editable).
Max wait run	0		Max. wait time for valid inputs.
Min update time	10		Cycle time for data exchange to IEC program.

For further information on configuring CS31 Modules and I/O channels, see AC500 CS31 Bus ↳ *Chapter 1.6.5.2.10.1 "Configuration of CS31 bus master" on page 6078*.

Setting COMx - SysLibCom

For AC500-eCo processor modules the serial interface COM2 cannot be configured as SysLibCom mode.

Right-click *"COMx_Online_Access ➔ Add object ➔ COMx_SysLibCom"*.

The serial interface COMx is prepared for operation with the blocks contained in the library SysLibCom.lib and the according protocols. For details see ↳ *Chapter 1.4.2.4.2.1 "Overview" on page 564*.

The following parameters are available:

Parameter	Default value	Value	Description
Enable login, see ASCII protocol description ↳ *Chapter 1.6.5.2.11.3.1 “Enable login” on page 6102*	Disabled	Disabled	There is no check with regard to the Control Builder login telegram.
		Enabled	Telegrams received are checked with regard to the Control Builder login sequence. If the sequence is detected, the protocol setting is changed to 'Online access'.
RTS control, see ASCII protocol description ↳ *Chapter 1.6.5.2.11.3.2 “Usage of modems” on page 6103*	None	None	No RTS control (direction control, not for PM55x and PM56x)
		telegram	RTS control activated (absolutely necessary for RS-485!)
TLS, see ASCII protocol description ↳ *Chapter 1.6.5.2.11.3.2 “Usage of modems” on page 6103*	0	0...65535	Carrier lead time in [ms] (TLS > CDLY)
CDLY, see ASCII protocol description ↳ *Chapter 1.6.5.2.11.3.2 “Usage of modems” on page 6103*	0	0...65535	Carrier delay time in [ms] (CDLY <= TLS)
Character timeout ↳ *Chapter 1.6.5.2.11.6.1 “Telegram ending identifier” on page 6112*	0	0...65535	Character timeout in characters (must be 0 if Telegram ending selection = Character timeout)
Telegram ending selection ↳ *Chapter 1.6.5.2.11.6.1 “Telegram ending identifier” on page 6112*	None	None	No telegram ending identifier
		String (check receive)	2 characters, e.g. <CR><LF> (16#0d, 16#0a -> 16#0d0a) in parameter "Telegram ending value" Setting not recommended! (not supported with COM2 of AC500-eCo processor modules)
		Telegram length	Telegram ending identifier set by telegram length Setting not recommended!
		Duration	Telegram ending identifier set by time Setting not recommended!
		Character timeout	Telegram ending identifier set by character timeout
Telegram ending character ↳ *Chapter 1.6.5.2.11.6.1 “Telegram ending identifier” on page 6112*	0	0...1	Number of end characters in case of telegram ending selection "String"
Telegram ending value ↳ *Chapter 1.6.5.2.11.6.1 “Telegram ending identifier” on page 6112*	0	0...65535	Telegram ending identifier value for settings "Duration", "Character timeout" and "String"

Parameter	Default value	Value	Description
Transmission rate	19200	300 1200 4800 9600 19200 38400 57600 115200	Transmission rate (Baude)
Parity	None	None	No parity check
		Odd	Odd parity
		Even	Even parity
		Mark	Parity bit := TRUE
		Space	Parity bit := FALSE
Data bits	8	5, 6, 7, 8	Character length in bits/character
Stop bits	1	1, 2	Number of stop bits

Telegram ending identifier

The telegram ending identifier is set using the parameters Character timeout, Telegram ending selection, Telegram ending character and Telegram ending value.

Character silent time monitoring:

Monitoring of the character timeout can be set for all possible telegram ending settings (except Character timeout).

If the parameter Character timeout = 0, no character timeout monitoring is done.

With Character timeout > 0 the character timeout monitoring is activated.

The character silent time is defined in number of characters. The number of characters and the interface parameters (Transmission rate, Parity, Data bits and Stop bits) are used to calculate the silent time.

Example: Transmission rate=9600 Baud, Parity=none, Data bits=8, Stop bits=1, Character timeout=3

This results in a frame of 10 bits/character:

1 start bit + 8 data bits + 0 parity bit + 1 stop bit

Character silent time = 1000 x Character timeout x Frame / Transmission rate [ms]

Character timeout = 1000 x 3 x 10 / 9600 = 3.125 ms ~ 4 ms.

If the time between the reception of two characters exceeds the character silent time, the reception is aborted with an error and the characters received up to this moment are made available.

The following parameter combinations are possible:

Character timeout **see remark on character silent time monitoring**	**Telegram ending selection** **Type of telegram ending identifier**	**Telegram ending character = ignored**	**Telegram ending value = ignored**	**Description**
Number of characters 0 or > 0	None	-	-	No telegram ending identifier, i.e., the characters received since last call are provided. The maximum number of characters is limited to 256.
Number of characters 0 or > 0	String (check receive)	Number of telegram ending characters 1 or 2	2 characters (for example 16#0d0a)	According to value set for "Telegram ending character", it is checked for 1 or 2 ending characters. The ending character(s) is (are) not passed, i.e., they are not contained in DATA area.
	1	1	16#0d = 13dec = <CR>	After reception of 16#0d, telegram received is reported.
	2	2	16#0d0a = 3338dec = <CR><LF>	After reception of 16#0d and subsequently 16#0a, telegram received is reported.
Number of characters 0 or > 0	Telegram length	-	Number of characters >0 and <=256	Telegram received is reported once the number of characters defined in "Telegram ending value" is received.
Number of characters 0 or > 0	Duration	-	Time in [ms]	Telegram received is reported once the time set for "Telegram ending value" (in [ms]) is elapsed. The time starts with the first FALSE -> TRUE edge at input EN of the receive block COM_REC.
0	Character timeout	-	Number of characters >0 and <=256	The number of characters set for "Telegram ending value" and the interface parameters (Transmission rate, Parity, Data bits and Stop bits) are used to calculate the silent time. Telegram received is reported if the silent time between two characters is >= the calculated silent time.

An example on how data is sent/received with the protocol "SysLibCom" is provided in the System Technology chapter *Chapter 1.6.4.1.11.2 “Sending/Receiving data with SysLibCom protocol” on page 5502*.

Setting COMx - Multi

> *For AC500-eCo processor modules the interface COM2 cannot be configured as “COM2 - Multi”.*

The serial interface COMx is prepared for operation with 2 selectable protocols.

1. Right-click *“COMx_Online_Access* ➔ *Add object”* and select *“COMx - Multi”* from the list.
2. In the device tree, append the desired protocols to the node (*[Add object]*).
3. Modify the parameters for the appended protocol nodes.

   The protocol parameters are identical to the parameters described for the individual protocols.

When restarting the program, i.e. after switching power ON, a download or a reset, the protocol appended first is always active.

Switching between the protocols is done using the function block COM_SET_PROT (contained in the library SysInt_AC500_V10.lib). At the function block input COM, the number of the serial interface is applied and at the input IDX the protocol index is set. The protocol appended first in the PLC configuration has the index 0, the second protocol the index 1.

An example which describes the usage of the function block can be found in the System Technology chapter *Chapter 1.6.4.1.11.1 “Function block COM_SET_PROT” on page 5501*.

1.6.5.2.12 AC500 FBP slave interface

> *Some processor modules do not provide a neutral FBP interface.*

The FBP slave interface is used to connect the AC500 controllers as fieldbus slave via FieldBusPlug (FBP).

The neutral FBP interface is configured as *“FBP_Online_Access”* by default. This allows programming and login via the FBP slave interface using the device UTF21-FBP adapter (USB connection on PC-side).

> *No protocol is set for the neutral FBP interface in the standard configuration (setting "FBP - none").*
>
> *If the FBP slave interface address (ADR > 0) is set using the display/keypad, this address has priority over the PLC configuration setting.*

The FBP slave interface occupies the I/O area:

%IB3000 .. %IB3999 or %QB3000 .. %QB3999.

Configuration of a FBP slave

1. Add the desired FBP module to the FBP node and modify the FBP slave parameters if required:

Parameter	Default value	Value	Description
Run on config fault	No	No	The PLC program is not executed if there are faults in configuration.
		Yes	The PLC program is executed even if there are faults in configuration.
Operation mode (read only)	Slave	Slave	Operation mode of FBP (Slave). This parameter is read-only (not editable).
Address	1	0...255	Address of the FBP slave.

2. Check the I/O configuration. I/O ranges of the different slaves:

FBP	Fieldbus	I/O range
PDP22 DP V1 modular	PROFIBUS DP V0/V1	8 modules, but a total of 32 bytes and 128 words in/out, 244 bytes max. per direction, a total of 368 bytes per slave
DNP21	DeviceNet	1 module with a maximum of 16 bytes and 16 words (inputs or outputs), for example 1 x "Module 16 Byte and 16 Word In/Out"
DNP21 modular	DeviceNet	8 modules, but a total of 16 bytes and 16 words (input or output)
COP21	CANopen	8 modules, but a total of 16 bytes and 16 words (input or output)

Depending on the fieldbus master, the processor module can exchange a different amount of input/output data with the master.

⇨ A maximum of 8 modules can be appended to the FBP slave interface. The size of possible modules depends on the used FBP, fieldbus and Communication Module (fieldbus master).

The byte inputs and outputs are provided as BYTE and BOOL and the word inputs and outputs as WORD, BYTE and BOOL.

The I/O modules saved in the PLC configuration and their addresses must match the entries in the configuration of the respective fieldbus master.

If you want to exchange less data than the maximum allowed amount of I/O data with the fieldbus master, you can setup a configuration consisting of different modules.

3. Set the I/O mapping.

See Symbolic Names for Variables, Inputs and Outputs for further details on mapping
↳ *Chapter 1.6.5.2.9.6 “Symbolic names for variables, inputs and outputs” on page 6060*.

1.6.5.2.13 Gateway configuration

1. In the Automation Builder project, right-click the topmost PLC tree node and select *“Communication Settings”*.

   ⇨ The dialog window Communication Settings appears.

2. Click *“Advanced Settings”* to open the Communication Parameters dialog and to change the communication settings.

   ⇨ This information will be stored in the project file.

3. Click *“Gateway”* to enter the gateway settings . Specify the connection type (e.g. TCP/IP) and the gateway address credentials if required.

   Confirm your settings with *“OK”*.

Gateway settings on windows server 2012

Gateway as a service

To allow multiple concurrent users from different user sessions on the server to connect to PLCs, the user has to run CODESYS gateway as a system service. This is managed by a service called "CoDeSys V2.3 Gateway Service Wrapper". The service starts on system start-up and launch the gateway.

If you want to restart the gateway, use "Services management console" to restart "CoDeSys V2.3 Gateway Service Wrapper".

Gateway settings

You can set the communication settings in the Automation Builder project for every PLC. Otherwise, an error message is displayed while trying to open CODESYS.

See the description for *Chapter 1.6.5.2.13 “Gateway configuration” on page 6116* and select "TCP/IP" under *“Connection”*.

1.6.5.2.14 Open Device Type editor

The Open Device Type editor has reached end-of-service-life with Automation Builder 2.4.0. All data can be accessed, but no further maintenance or support will be given. This option will be removed with Automation Builder 2.6.0.

Automation Builder provides Open Device Type editor to create device descriptions and parameters on any device connected through Modbus RTU/Modbus TCP/IP.

In the Automation Builder main menu click *“Tools ➔ Open Device Type Editor”* to launch the Open Device Type editor. The Open Device Type editor consists of device information, groups and parameters and custom parameter types.

Device information

The *“Device Information”* tab is used to add basic information of a device and to select the supported communication protocols.

See the following table to add basic information and communication settings of a device.

Device information	
Name	Name of the device.
Type	The device type is fixed for open devices.

Device information	
ID	The first four digits are fixed for ABB (i.e. 1020). The other four digits must be unique.
Version	The version field is enabled to edit and to add desired value.
Description	Device description.
Vendor	Name of the vendor.
Order number	Ordering information.
Icon	Default icons are used if not updated.
Image	Default images are used if not updated.

Communication settings	
Supported protocols	Supports through Modbus RTU, Modbus TCP/IP or both.

Groups and parameters

1. In the Open Device Type editor, click *"Groups and Parameters tab ➔ New"* to create parameter data.
2. Enter the basic information, range, scaling and parameter address information and save the changes.
3. Click *"Discard Changes"* to discard the changes while editing the parameter data.
4. To modify already existing parameter data, select the parameter and edit.

See the following table to create parameter data.

Basic information	
Group	Name of the parameter group.
Name	Name of a parameter.
Type	Type of parameter.
Description	Parameter data description.
Parameter ID	Numeric parameter ID.
Unit	Unit of parameter.
Offline Access	Validates the parameter read/write settings.
Online Access	Validates the parameter read/write settings.

Range information	
Minimum value	Minimum value of a parameter.
Maximum value	Maximum value of a parameter.
Default value	Default value of a parameter.

Scaling	
Numerator	Numerator value of a parameter scaling factor.
Denominator	Denominator value of a parameter scaling factor.

Parameter Address	
Register ID	Address used for Modbus (range 0...65536).

Bulk editing

The Open Device Type editor allows to copy-and-paste bulk parameters within the view and from/to MS Excel.

In *"Groups and Parameters"*, right-click on parameter data and do the following:

- Click *"Copy"* to copy bulk parameter data.
- Click *"Paste"* to paste bulk parameter data to the editor.
- Click *"Delete"* to delete bulk parameter data from the editor.

If the fields marked with are empty, a message is displayed.

For example, if the parameter data contains inappropriate data, those parameters are highlighted in red color. By selecting parameter data, the error is highlighted .

Custom parameter types

The *"Custom Parameter Types"* tab is used to create Bit Field and Enum custom parameter types. The custom parameter types are later used as bit field and enum type parameter.

Bit field

1. In the *"Custom Parameter Types"* tab, right-click *"Bit Field Type ➔ Add New"* to create a new bit field type.
2. In the Basic Info editor, add the parameter Name, Default Value and Base Type.
3. In the Member editor, create a parameter type by adding Name, Default Value and Visibility. Save the data.
4. Click *"Previous"* or *"Next"* to enable the bit field for editing.
5. Click *"Change sequence of members"* to change the bit field sequence.

See the following table to create the parameter type.

Basic info	
Name	Parameter name.
Default Value	Default value of a parameter.
Base Type	Parameter type.

Member	
Name	Selected bit field name.
Default Value	Default value of a bit field.
Visibility	Bit field visibility.

The newly created bit field parameter type is visible in the Type options in the *"Groups and parameters"* tab.

Enum type parameter

1. In the *"Custom Parameter Types"* tab, right-click *"Enum Type ➔ Add New"* to create a new Enum parameter type.
2. In the Basic Info editor, enter the parameter Name and select the Type.
3. In the Member editor, click *"New Member"* to add Enum Type values.
4. Save the Enum Type values to the Device description editor.
5. Click *"Previous"* or *"Next"* to enable the Enum value to rename and to change the member fields.
6. Click *"Discard Changes"* to discard the changes.

The newly created Enum type parameter is visible in *"Groups and parameters ➔ Type"*.

Exporting/Importing a device description file

The Open Device Type editor allows to export the device descriptions in DEVDESC format.

In the Open Device Type editor, click *"Export"* and save the file to the file system.

Select a location in the file system to save the file.

For the import of a customized device description file, click *"Import"* in the Open Device Type editor and select the device description file from the file system.

Installing a device description file to the device repository

The device description file can be installed in the Open Device Type editor or by using *"Tools ➔ Device repository"* in Automation Builder main menu.

Configuring a device description file in modbus RTU

1. In the Automation Builder project, right-click on Modbus RTU communication node and select *"Add object"*.
2. Select the installed device description file and click *"Add object"*.
   ⇨ The added device description file is created using Modbus RTU protocol.

Configuring a device description file in modbus TCP/IP

1. In the Automation Builder project, right-click *"Protocols ➔ Add object"*.
2. Select *"Modbus TCP/IP Server ➔ Add object"*.

3. Right-click *"Modbus_TCP_IP_Server ➔ Add object"*.
4. Select the operating data file and click *"Add object"*.
   - By default, the installed device description file is updated in CODESYS application.
   - After adding the device description file (for example, operating data) to Modbus TCP IP server, check the IP configuration in *"Tools ➔ IP configuration"*.
5. Right-click *"PLC_AC500 ➔ Communication Settings"* to set the gateway communication settings.

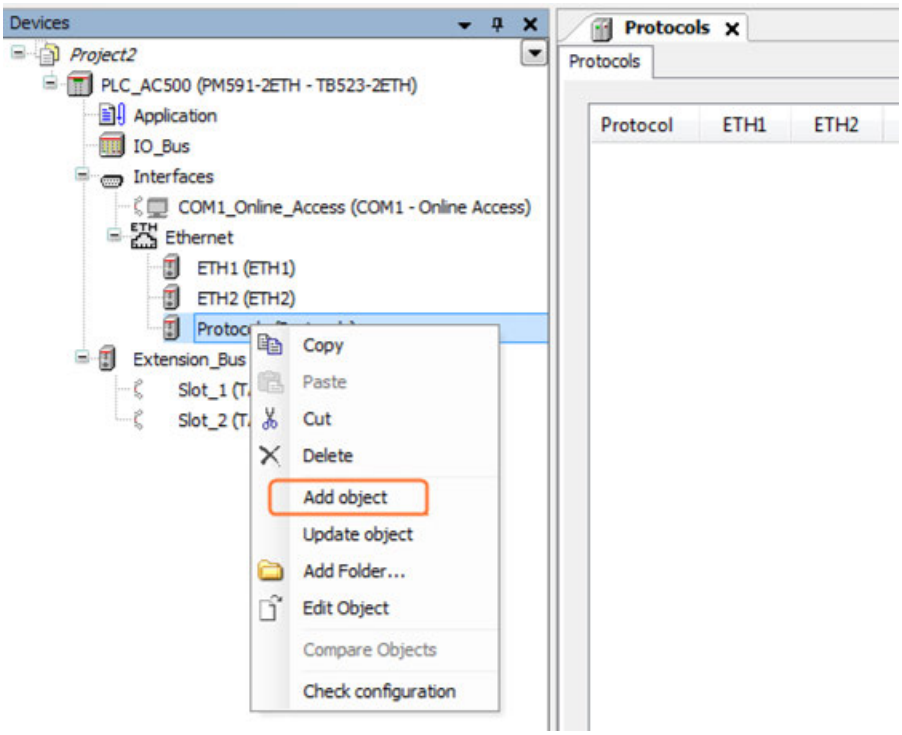
Task configuration

1. In the Automation Builder project, double-click *"Application"* to launch CODESYS application.
2. In the CODESYS application main menu, click *"Online ➔ Login"*.
3. Click *"Resources"* tab and configure the task configuration.
4. In the *"POU"* tab, double-click the device description file and change the status to TRUE.

1.6.5.3 Protocols and special servers

1.6.5.3.1 General configuration of protocols and special servers

As of Automation Builder 1.0 protocol configuration and the configuration of special servers such as FTP server configuration is done at a common device tree location under *"Ethernet ➔ Protocols"*.



Right-click *"Protocols ➔ Add object"* and select the object to be configured from the list.

1.6.5.3.2 IEC60870-5-104 (Telecontrol)

Configuration of IEC 60870-5-104 (Telecontrol)

> *Protocol IEC 60870-5-104 (telecontrol) is only available on AC500 CPUs with onboard Ethernet.*

Protocol IEC 60870-5-104 is described in *Chapter 1.5.4.18 “IEC60870 library” on page 1351*.

For further information on telecontrol, see the following chapters:

Configuration < CBP 2.4 *Chapter 1.6.5.3.2.3.1 “Configuration up to and including CBP 2.3” on page 6123*

Configuration >= CBP 2.4 *Chapter 1.6.5.3.2.4.1 “Configuration changes >= Automation Builder 1.1/CBP 2.4” on page 6138*

General information IEC60870

Introduction

The implemented IEC60870-5-104 protocol allows link-ups between AC500 CPUs with onboard Ethernet and external systems. The link-up takes place via the onboard Ethernet interface of the CPU. The telecontrol protocol according to IEC60870-5 is used.

The CPU can work as both control station and substation. In control direction, setpoints and commands can be set; in monitoring direction the control station sends status values, real values and discrete values to the substation. Via general inquiry, the substation requests the control station to send all status values, real values and discrete values. Otherwise, these values are sent by the control station on a change-driven basis, cyclically or when triggered by an application. Status values, real values and discrete values may contain timestamps. These are filled in with the time of the process station when sent. The CPU can time-synchronize the telecontrol link.

A module accepts the configuration of the physical interface (link layer) and the general protocol parts (application layer).

Send and receive blocks are available for data exchange. These blocks exist for the IEC60870-5 data types setpoint value, command value, double command value, status value, double status value, real value and discrete value. The inputs/outputs of the send and receive blocks are combined with the signals to be communicated. See documentation of IEC60870 library for more information.

Configuration

Note: As of AB 1.1 (CBP 2.4) telecontrol configuration has been changed. Hence, description can be found in the following chapters:

Configuration < CBP 2.4 *Chapter 1.6.5.3.2.3.1 “Configuration up to and including CBP 2.3” on page 6123*

Configuration ≥ CBP 2.4 *Chapter 1.6.5.3.2.4.1 “Configuration changes >= Automation Builder 1.1/CBP 2.4” on page 6138*

Data flow control

Each send or receive block can only process one data message. Ideally, new data are available at each user task run-through or new data can be sent.

If the output OV (send block only) indicates TRUE, the function block computes more quickly than the data can be sent. This can happen if the receive block is not computed quickly enough and has thus not collected all the data.

Alternatively, this block sends either cyclically or if the input value is changed. Ideally, the topical data can be sent via the telecontrol link in connection with every user task run-through.

Data integrity

With IEC60870-5 protocol, a distinction is made between data transmission in the monitoring direction (status values, real values, discrete values) and in the control direction (commands and setpoints).

All data transmissions are acknowledged from the link communication level by the receiver. This acknowledgement is not sent to the sender of the data in every telecontrol link.

For data transmission in control direction, additional acknowledgement (e.g. ACTTERM) is possible. These acknowledgements are not sent by every telecontrol link either. For safe data transmission, it is necessary, in such cases, to configure data readback. The receiver then sends the data received back to the sender via the corresponding send blocks.

Information in the monitoring direction is acknowledged by the receiver on the lowest communication level (link level) when received. This acknowledgement is generated by the telecontrol head itself with some telecontrol heads. In the event of overload/overrun, a data message may be lost. For data in the control direction, so-called ACTTERM acknowledgement can be used. This additional acknowledgement is sent back to the sender when the data have been executed in the process. If data are to be sent in the monitoring direction with guaranteed transmission, it is necessary to read back the sent value via another variable and, after observing a monitoring time, resend in the event of an error.

Data transmission

Send blocks

On the basis of the communication protocol, it is sensible to restrict the data types at one send block to one type. Therefore, there are 5 types of send blocks: send of status values, commands, real values, setpoints and discrete values. These types are mapped to the IEC1131 data types BOOL, REAL and DINT. See documentation of IEC60870 Library for more information.

Operating modes of the send blocks

The send blocks know three operating modes to send their data:

- Caused by request pin (SEND)
- Send in connection with a change of data (AUTO)
- Cyclic send of data (CYCLE)

Send via request pin

The SEND signal is evaluated on the rising edge, the RDY signal remains applied for one computation cycle. If a rising edge is generated again at the SEND signal although no acknowledgement has yet been received from the receiver, the OV pin is set in order to indicate that an overrun has happened. The evaluation of the receive acknowledgement is carried out before the evaluation of whether transmission is to take place. This means, assuming that there is an appropriately fast telecontrol link, that in connection with change-driven and cyclic transmission, a transmission job can be sent in connection with every computation of the block. In connection with send via the request pin it is possible to send only in connection with every second computation (send takes place only with a rising edge).

Change-driven send of data

Data are always sent when the value of the input variables changes. When changes take place, there is an internal simulation that the SEND pin changed from 0 to 1.

In order to prevent unnecessarily frequent send in the event of mild fluctuations in the input value, a threshold value can be configured for real values and setpoints. The input value is not sent until it differs positively or negatively from the value last sent by more than the threshold value.

If the input value changes again although no acknowledgement has yet been received from the receiver, the OV pin is set in exactly the same way as in connection with send via the request pin. If an error occurs during send, the job is automatically retried until the value has been sent without error.

Cyclic send

The data are automatically sent after expiration of a configurable cycle time (SCANDOWN). This cycle time is indicated in multiples of the task cycle time in which the block is computed. In this operating mode, an overrun error can occur if the transmission is faster than the response time of the receiver. For setpoints, it is necessary to ensure that an acknowledgement is generated by the receiver which is not sent until the setpoint is accepted. The send block is not ready for transmission again until after this acknowledgement has been received.

Receive blocks

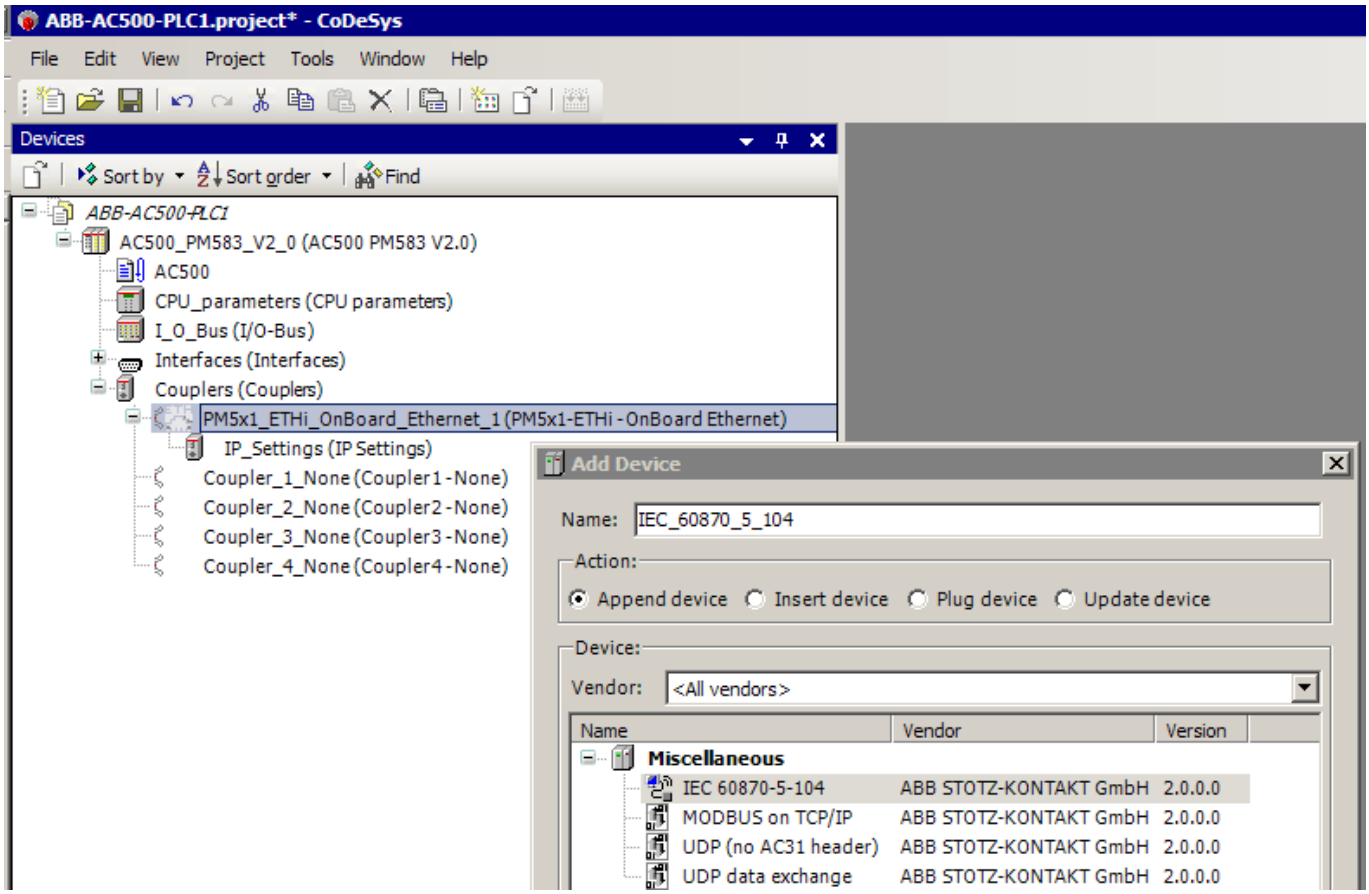
In receive direction, the jobs enter the device module via the interface. The device module selects the correct receive block using the telecontrol address. To this end, during installation the receive blocks pass their parameterized telecontrol addresses to the device module. The device module stores the data received and the receive blocks make the data available at their output pins in connection with the next computation of the user task.

Configuration < Automation Builder 1.1/CBP 2.4

Configuration up to and including CBP 2.3

The IEC 60870 protocol allows link-ups between AC500 CPUs with onboard Ethernet (e.g. PM573-ETH and PM583-ETH) and external systems.

The link-up takes place via the onboard Ethernet interface of the CPU.



For further information on configuration (> CBP 2.4), see the following chapters:

Control and Substations ↳ *Chapter 1.6.5.3.2.3.2 “Control station and substations < Automation Builder 1.1” on page 6124*

Global Address ↳ *Chapter 1.6.5.3.2.3.3 “Global address” on page 6129*

Data Points ↳ *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*

Validity Check of Configuration ↳ *Chapter 1.6.5.3.2.3.5 “Validity check of configuration (< Automation Builder 1.1/CBP 2.4)” on page 6135*

Creating Configuration Data in Automation Builder ↳ *Chapter 1.6.5.3.2.3.6 “Creating configuration data in Automation Builder” on page 6135*

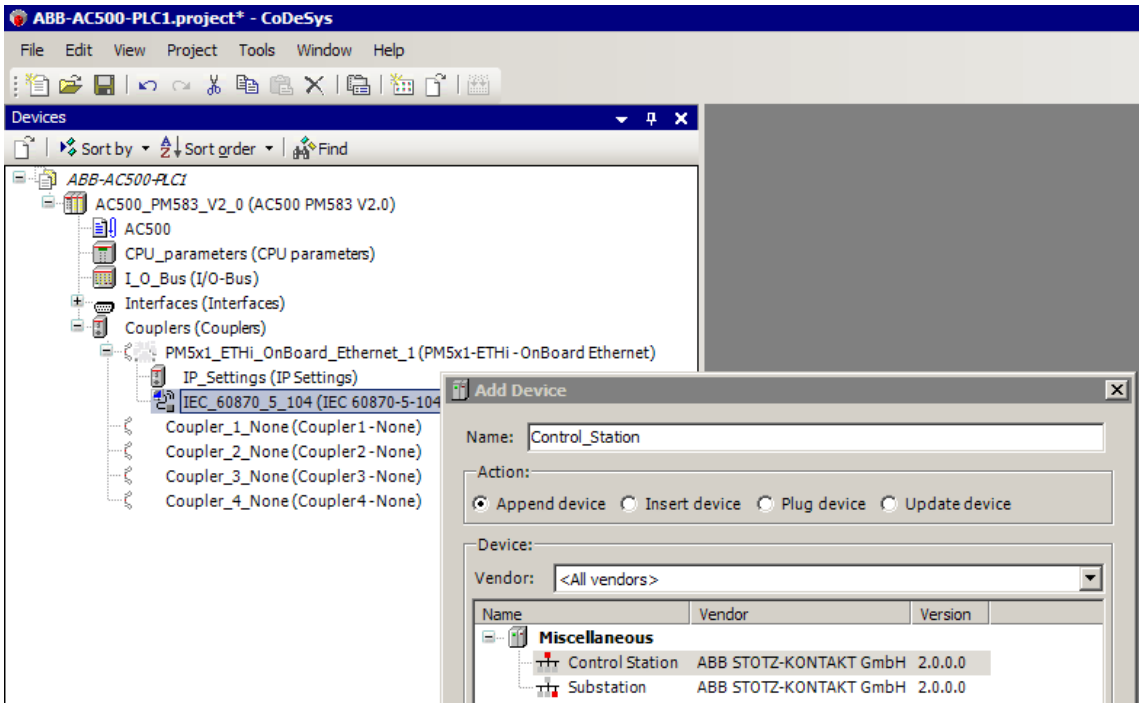
Using Control and Substations in CODESYS ↳ *Chapter 1.6.5.3.2.3.7 “Using control and substations in CODESYS” on page 6137*

Using Data Points in CODESYS ↳ *Chapter 1.6.5.3.2.3.8 “Using data points in CODESYS” on page 6137*

Control station and substations < Automation Builder 1.1

The CPU can work as both control station and substation.

Control station	Master, client, controlling station: Synonyms for a higher-level station (central station, monitors others stations)
Control direction	Data transfer direction from the control station to the substation
Substation	Server, slave, controlled station: synonyms for a subordinate IEC 60870-5-104 telecontrol station (which is monitored)
Monitoring direction	Data transfer direction from the substation to the controlling station



Control station and substation preferences

The application module contains all IEC 60870-5 application layer functionality. Within these functions the ASDU (Application Service Data Unit) is generated and decoded. Data is exchanged between the ASDU (protocol layer) and the telecontrol data points.

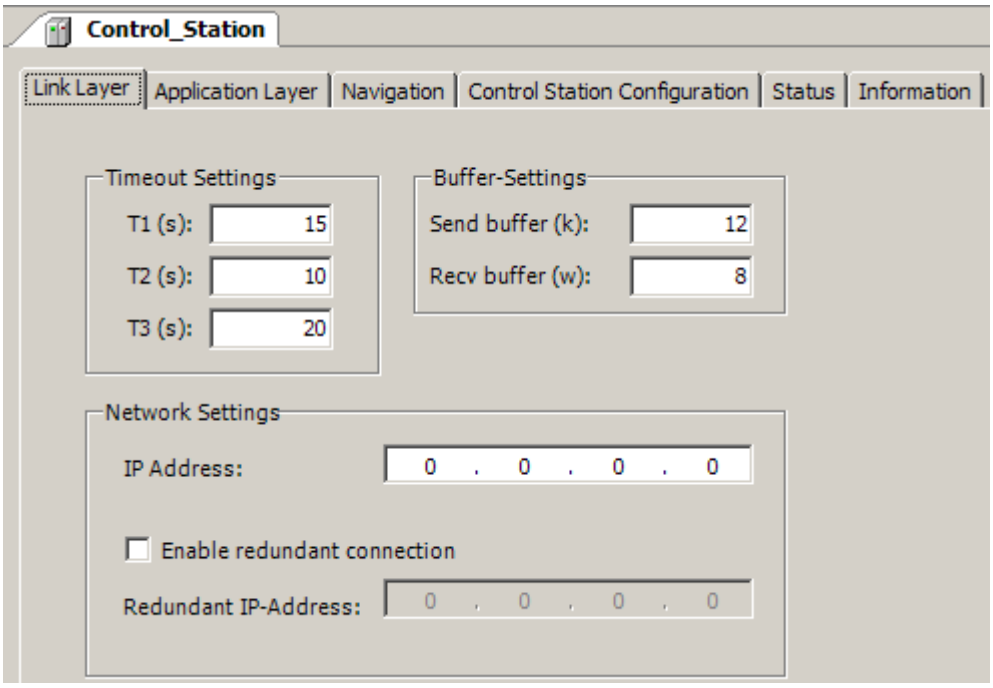
All functions within the IEC 60870 are handled by this code (ACTCON, ACTERM, General Inquiry etc).

To use them it is necessary to select the correct settings due to the Control- /Substation.

Tab link layer

The link layer (link level) is the communication layer which accesses to the Ethernet interface.

The link layer provides the following settings:



Timeout settings T1, T2, T3: The values for the connection control and message replication; timeout1/2/3.

Buffer settings This parameter gives the maximum number of outstanding messages and acknowledgement behaviour.

Send buffer (k):	Maximum difference receive sequence number to send state variable
Rec buffer (w):	Latest acknowledge after receiving w I format APDUs

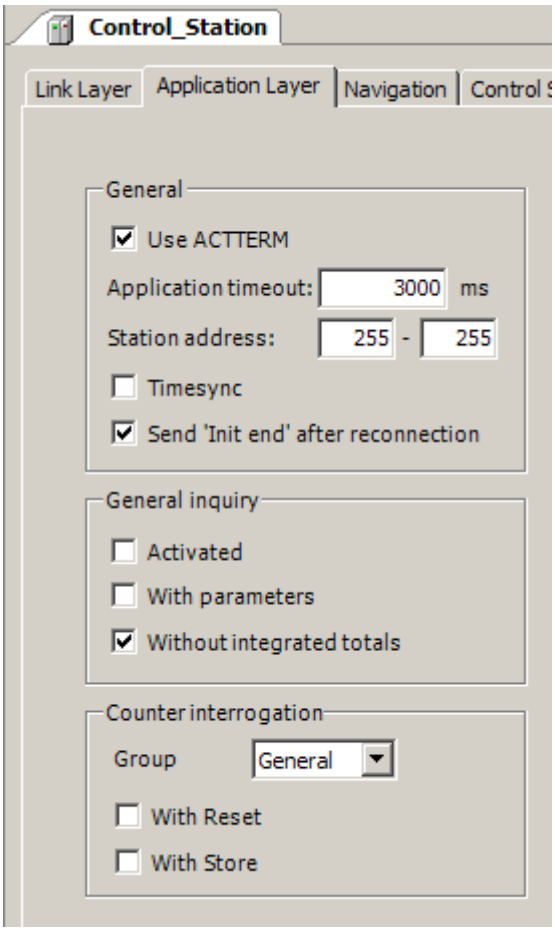
Network settings Network Settings are only available for control stations. The IP address of the control station and if available the IP address of another control station (Redundant IP address) can be selected by the user.

Tab application layer

The application layer is the communication layer with which the send and receive blocks work.

The application layer defines different information objects and services for their transmission.

A data point or information object is identified via a system-wide unambiguous address containing a maximum 5 bytes. ↳ *Chapter 1.6.5.3.2.3.4 “Data points” on page 6131*



General

Use ACTTERM This parameter concerns only setpoints and commands. If this parameter is checked, an acknowledgement with set ‘actterm’ is generated as reason for transmission at the time at which the receive block is computed and outputs its telecontrol data at its output pins. On transmission side, the data block awaits the reception of this ACTTERM acknowledgement and reacts with its corresponding output (see *Chapter 1.5.4.18 “IEC60870 library” on page 1351*) to the reception of this acknowledgement. For commands with execution time, the acknowledgement is generated when the command is terminated, for commands with continuous execution time and for setpoints, the acknowledgement is generated when the data are output to the output pins.

Application timeout This time indicates how long an acknowledgement will be awaited on the application level. An acknowledgement is generated only for commands and setpoints on the application level.

Station address The station address defines which station will be subject to a count query. The values define the 2 bytes for the common telecontrol address (GADU1 and GADU2).

Value	Description
0	The station address is not used.
1...254	The count is queried on the station defined by the station address.
255	The count is queried on all accessible stations.

Timesync After each new establishment of a link and once per hour, a ‘coarse time synchronisation’ message is generated. This time synchronisation is only supported from AC500 to external systems. Time synchronisation from an external system to AC500 is not provided!

Incoming time synchronisation messages are confirmed by the process station but not executed. Greenwich Mean Time (GMT) is used as the time for the synchronisation.

Send 'Init end' After each establishment of a link or only in connection with the first establishment of a link and after reconfiguration, an init end message is generated. After the init end message, there is a general inquiry, if configured.

General inquiry

Activated This parameter concerns only real values, discrete values and status values. The device module generates a general inquiry message after each new establishment of a link. The other side then generates a message with the reason for transmission ‘general inquiry’ for every data point and subsequently an init end message. This procedure ensures that, in the event of a new establishment of a link, all data are available on the reception side in topical form.

With parameters If general inquiry is activated the parameter values are sent.

Without integrated totals With a general inquiry no integrated total values are sent.

Counter interrogation

Group General, 1 .. 4: The count inquiry is executed for a specific group of counters (1 .. 4).

The count inquiry is executed for all groups of counters.

With reset The reset quality bit is sent along with the count inquiry.

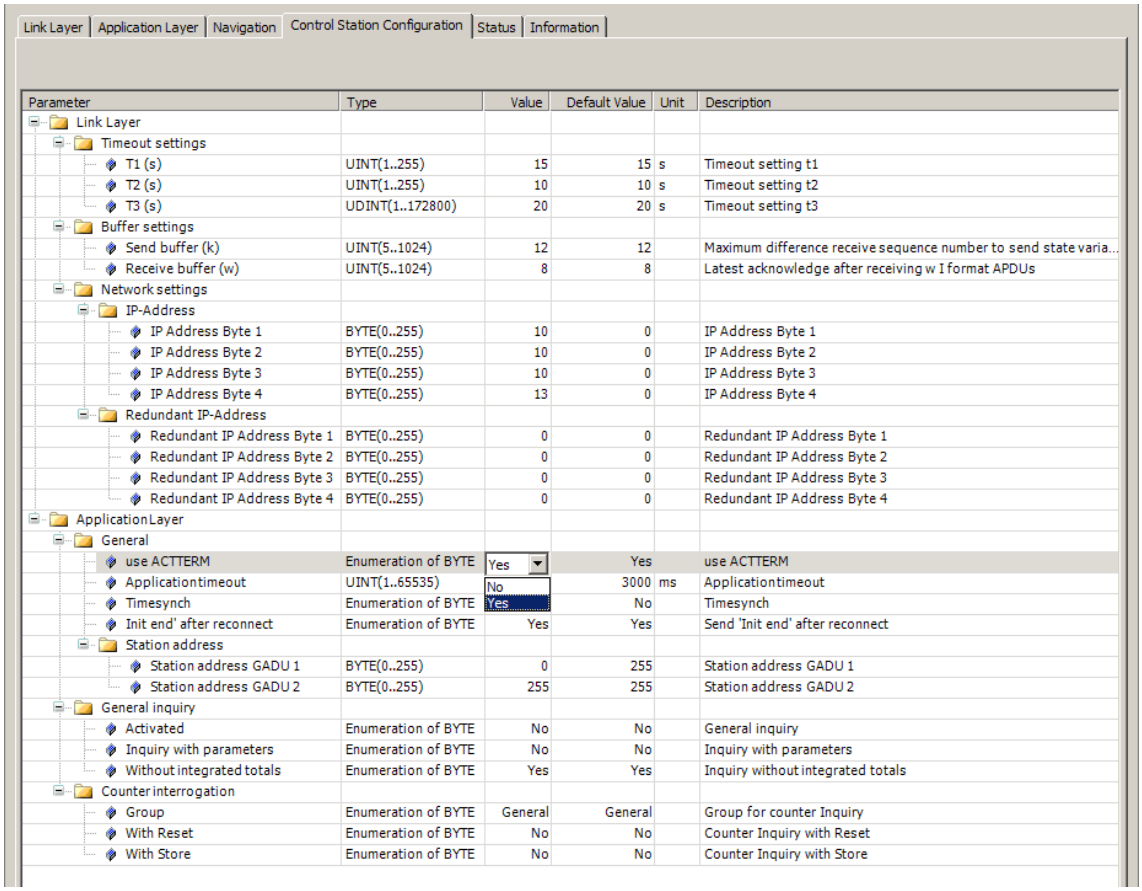
With relocate The relocate quality bit is sent along with the count inquiry.

Tab navigation

The defined Data Points from all defined global addresses were shown. Description of the columns: ↯ *Chapter 1.6.5.3.2.3.3 “Global address” on page 6129*

Tab control station configuration

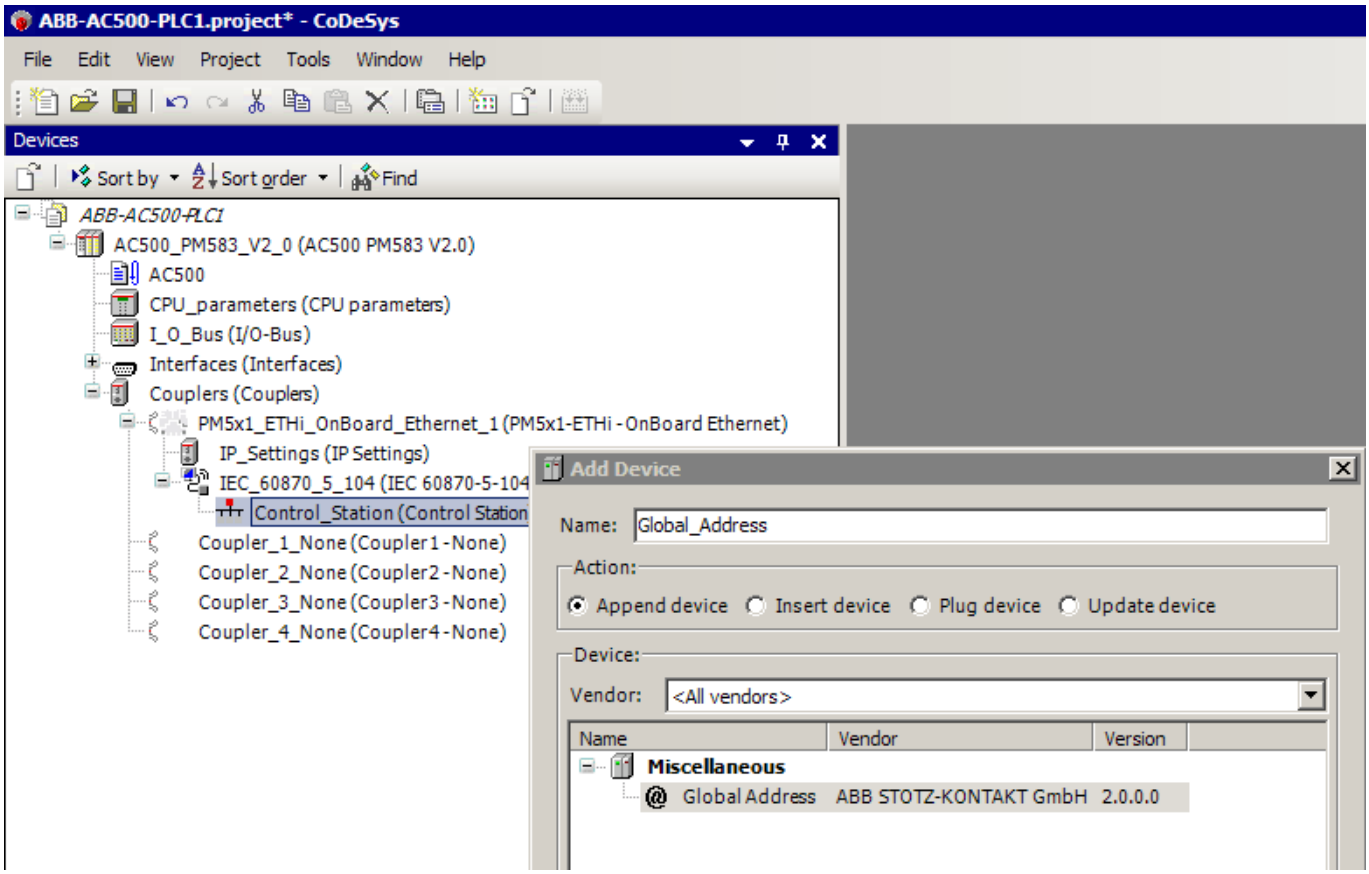
The Control Station Configuration register shows all selectable configurations of the Control and Substation.



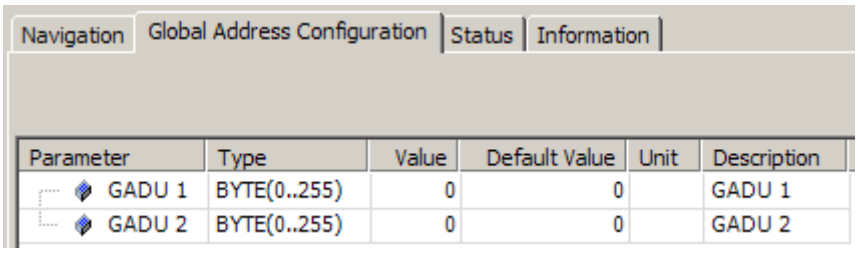
Furthermore it is possible to do the setup directly at this register. Otherwise the configuration can be done as described at the link layer and application layer register.

Global address

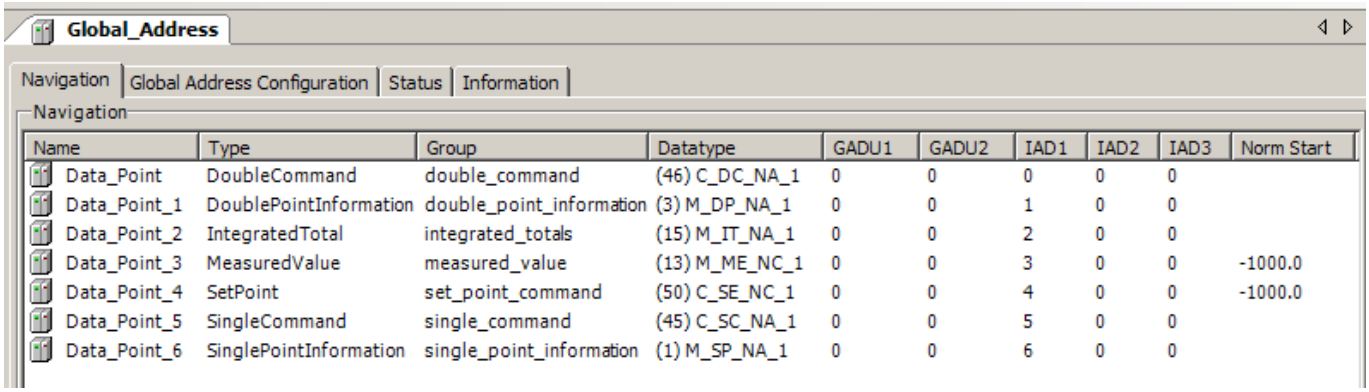
To select the correct Control Station / Substation it is necessary to assign the corresponding global address.



The configuration of the global address takes place by the following 2 global address bytes of the common telecontrol:



The Navigation menu reflects information about all configured data points ↳ *Chapter 1.6.5.3.2.3.4 "Data points" on page 6131*.



Description of the columns

Name: node name of the data point

Type: Type of the data point

Group: node name of the data point group

Data type: Data type of the data point

GADU1 / GADU2: Global address of the data point (parameter of the data group). Byte 1/2 of the common telecontrol address of the block

IAD1, IAD2, IAD3: Local address (Byte 1, 2 and 3) of the data point (parameter of the data point)

Norm Start / End / Threshold: Parameter of data point (only if available)

NORM. START: Low limit (0 %) of the normalized range for real values and setpoints.

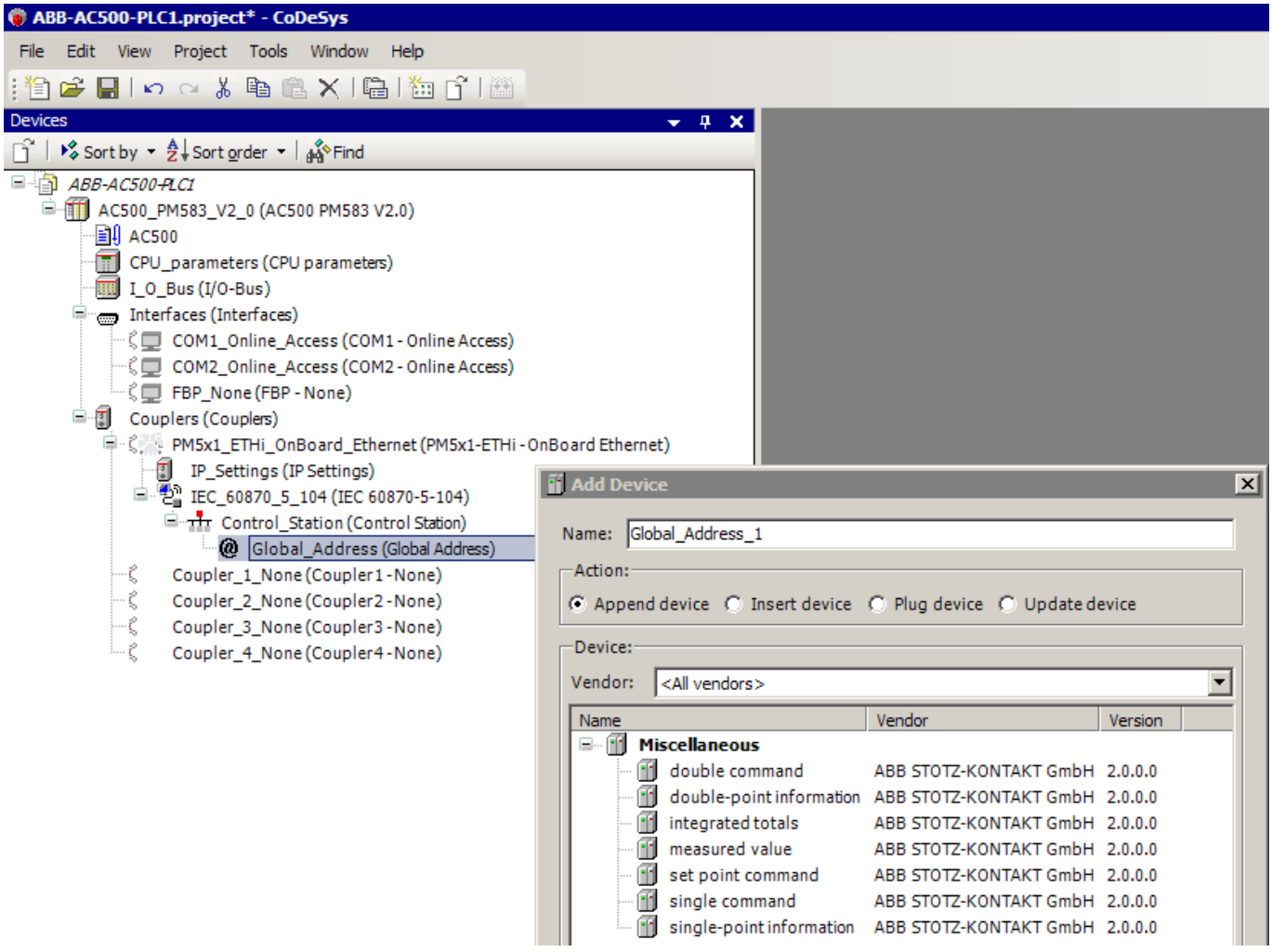
NORM. END: High limit (100 %) of the normalized range for real values and setpoints.

THRESHOLD: Threshold limit beyond which a change of the input value referred to.

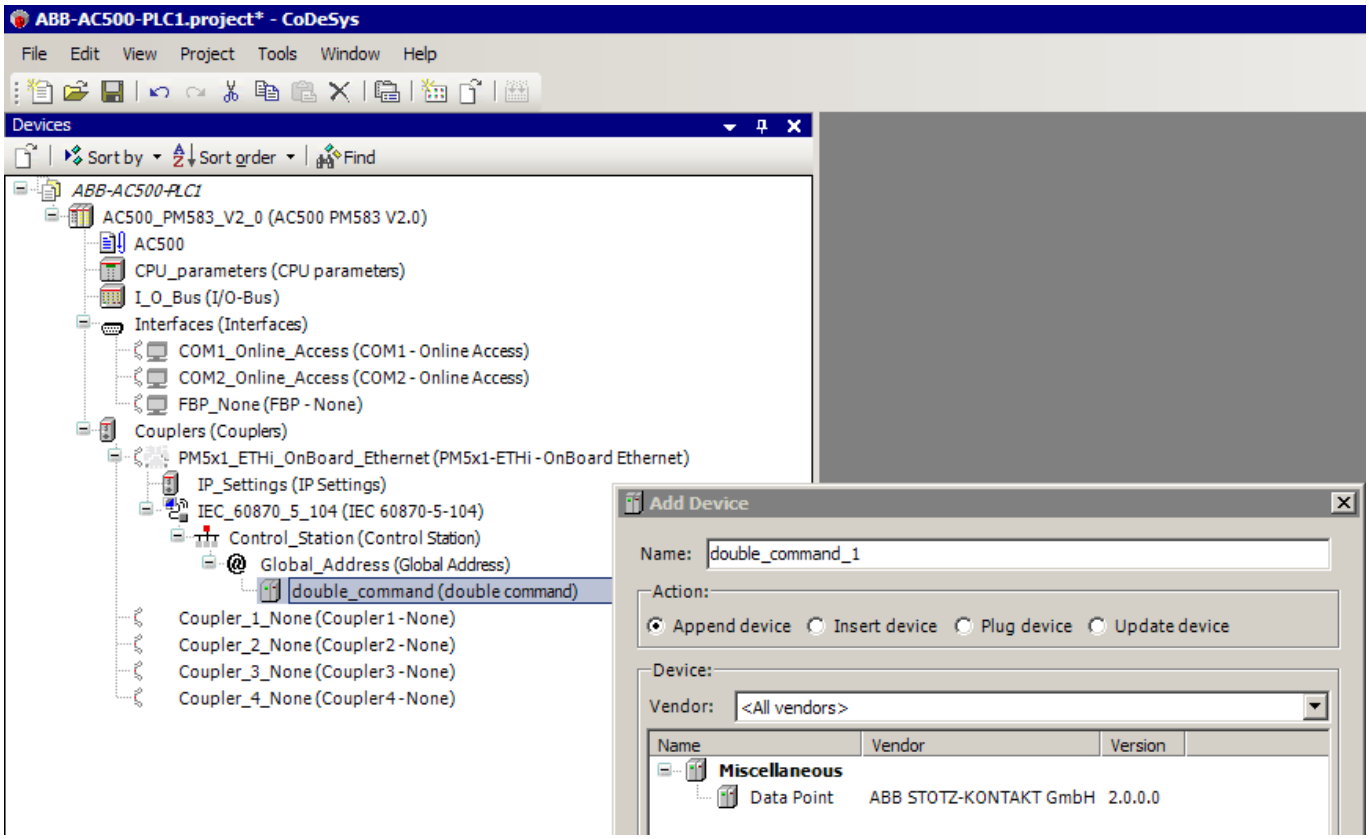
Data points

With the IEC 60870-5 protocol, a distinction is made between data transmission in the monitoring direction (status values, real values, discrete values) and in the control direction (commands and setpoints).

Therefore, the IEC 60870 protocol contains several data point types (called Pin Group), which were used due to their special functionality:



The Pin Groups are used by adding Data Points which have to be configured to use them.



Data point configuration

For a user friendly configuration the Data Points will be structured as described in following table:

Data Point type	Parameters Group	Parameters DP
	Timestamp	IA
single-point information (16)	Priority	
double-point information	Priority	
integrated totals	Group (none -> 4), Priority	
measured value (16)	Type (float/norm), Priority *1)	Threshold, Normalizing
single command		
double command		
set point command	Threshold, Normalizing	Threshold, Normalizing

*1) Priority will only be evaluated if the Data Point group will be sent. If the Data Point is connected with an receive function block the parameter will be ignored. It is also only necessary for an unbalanced mode. So for 5-104 variant this parameter will be hidden in the parameter mask.

All Data Point groups that are marked with (16) can be configured as multiple Data Points where all Data Points are sent in 1 telegram. It is not possible to receive multiple Data Points.

Mapping to IEC 60870-5-104 type identification

CP56Time2a	No timestamp		With timestamp	
Data point type	Norm	Floating point	Norm	Floating point
Measured value	(9) M_ME_NA_1	(13) M_ME_NC_1	(34) M_ME_TD_1	(36) M_ME_TD_1
Set point command	(48) C_SE_NA_1	(50) C_SE_NC_1	(61) C_SE_TA_1	(63) C_SE_TC_1

CP56Time2a Data Point type	No timestamp	Parameters DP
	Timestamp	IA
single-point information (16)	Priority	
double-point information	Priority	
integrated totals	Group (none -> 4), Priority	
measured value (16)	Type (float/norm), Priority *1)	Threshold, Normalizing
single command		
double command		
set point command	Threshold, Normalizing	Threshold, Normalizing

Data point modules

Beneath a Data Point group node the user can configure data point nodes. The Data Point module has parameters depending on the parent node as follows:

Local address (independent from parent node)

Data Point Configuration | Status | Information

Parameter	Type	Value	Default Value	Unit	Description
IA 1	BYTE(0..255)	0	0		local address IA 1
IA 2	BYTE(0..255)	0	0		local address IA 2
IA 3	BYTE(0..255)	0	0		local address IA 3

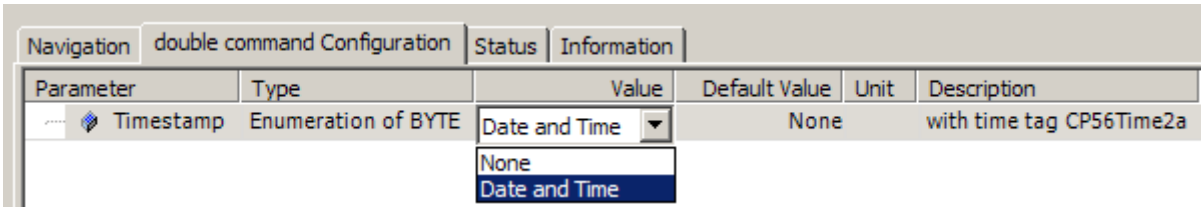
Threshold / Normalizing / Type (only beneath measured value and set point command)

Data Point Configuration | Status | Information

Parameter	Type	Value	Default Value	Unit	Description
IA 1	BYTE(0..255)	3	0		local address IA 1
IA 2	BYTE(0..255)	0	0		local address IA 2
IA 3	BYTE(0..255)	0	0		local address IA 3
Normalize start	REAL	-1000.0	-1000.0		Normalize start
Normalize end	REAL	1000.0	1000.0		Normalize end
Threshold	REAL	0.0	0.0		Threshold

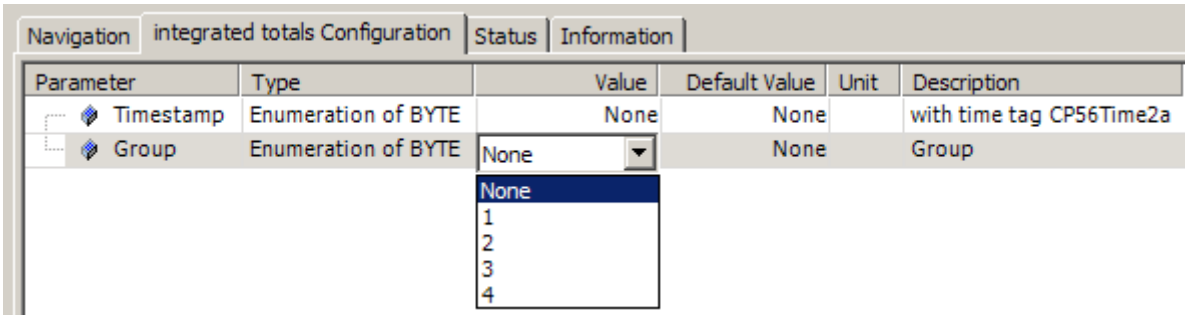
Timestamp

Available for Data Point type:	All
2 possible settings:	None
	Date and Time



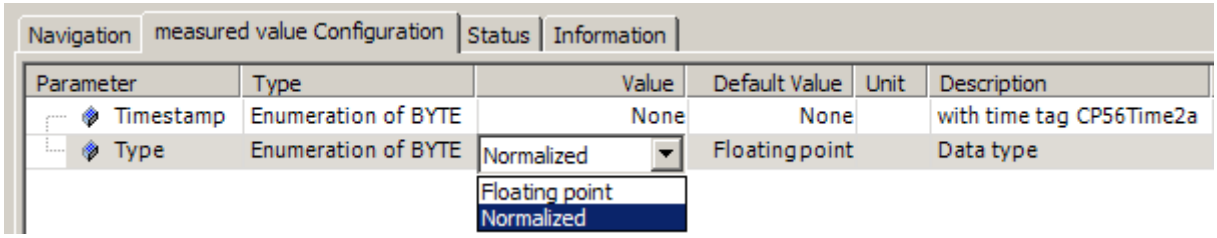
Group

Available for Data Point type:	Integrated totals
5 possible settings:	None
	1
	2
	3
	4



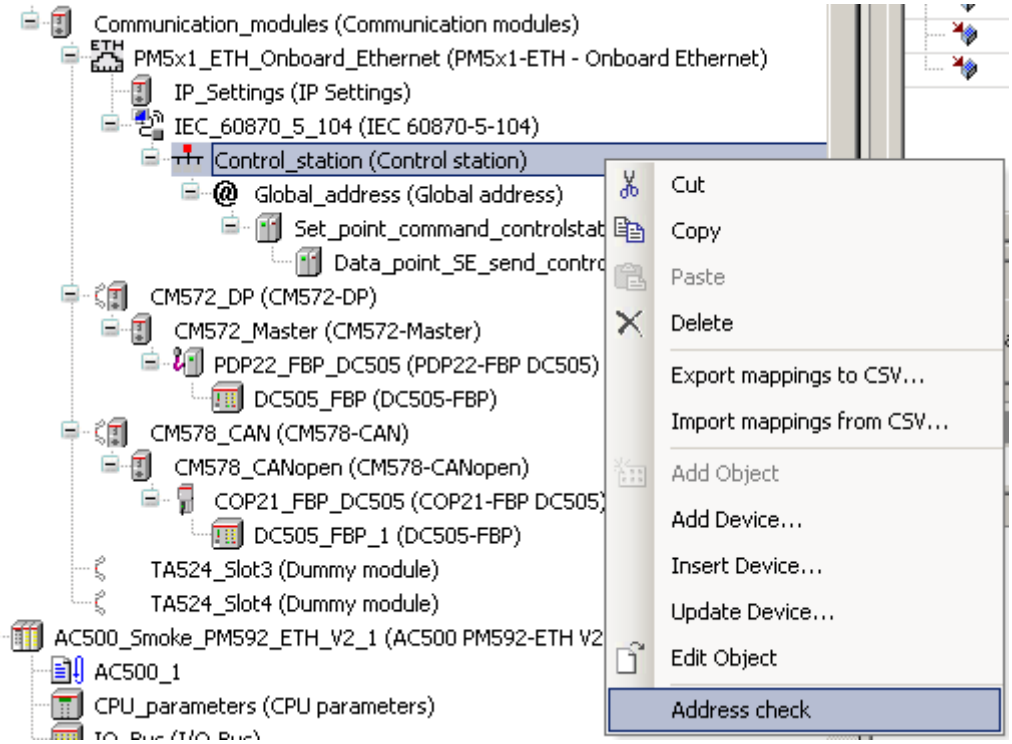
Type

Available for Data Point type:	Measured value
	Set point command
2 possible settings:	Floating point. Send/Rec data message as real value, floating-point number
	Normalized. Send/Rec data message as real value, normalised value



Validity check of configuration (< Automation Builder 1.1/CBP 2.4)

A validity check of the IEC configuration can be issued on different levels of the IEC 60870 tree elements via context menu:

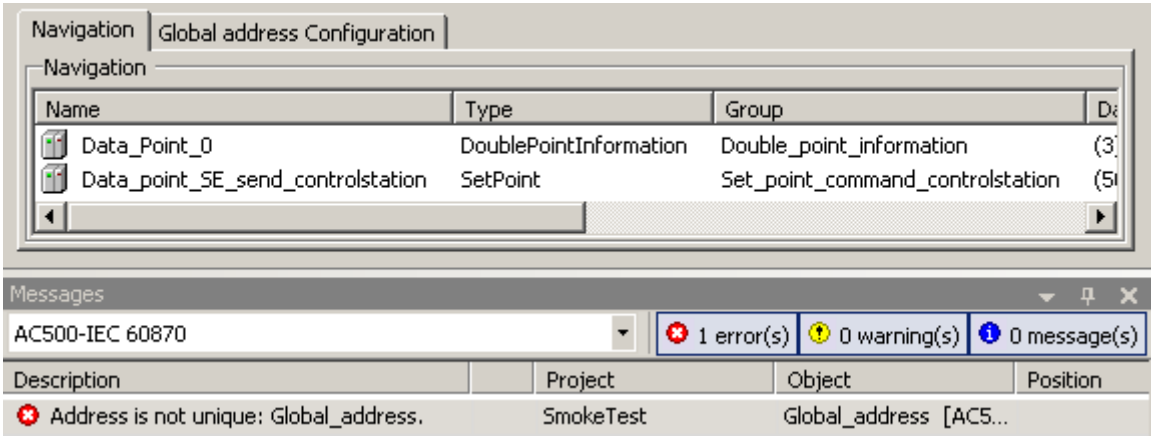


The scope for the check is the actual selected tree element and from there down the tree hierarchy. The overall check can be started from the top IEC60870 tree element.

The check will look for

- duplicate addresses
- stations and global addresses without a sub tree element
- missing data points

When a check finds errors this will be reported in a separate messages view:



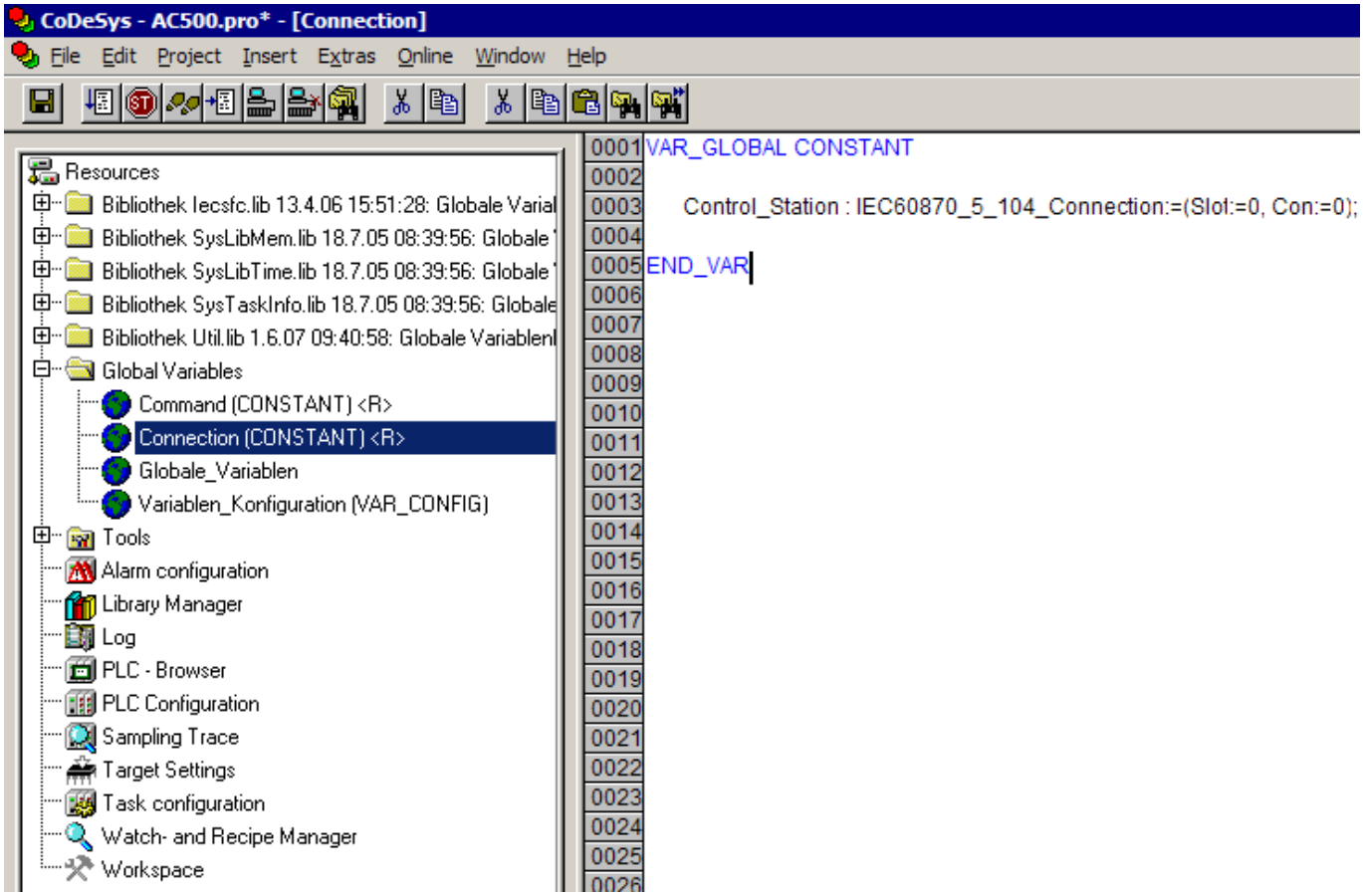
With a double-click on the error line, the part of the configuration with the violation will be opened. Now, you can correct the error.

Creating configuration data in Automation Builder

Information on creating configuration data in Automation Builder see ↬ *Chapter 1.6.5.4.1.1 “Creating configuration data” on page 6196*.

Connection

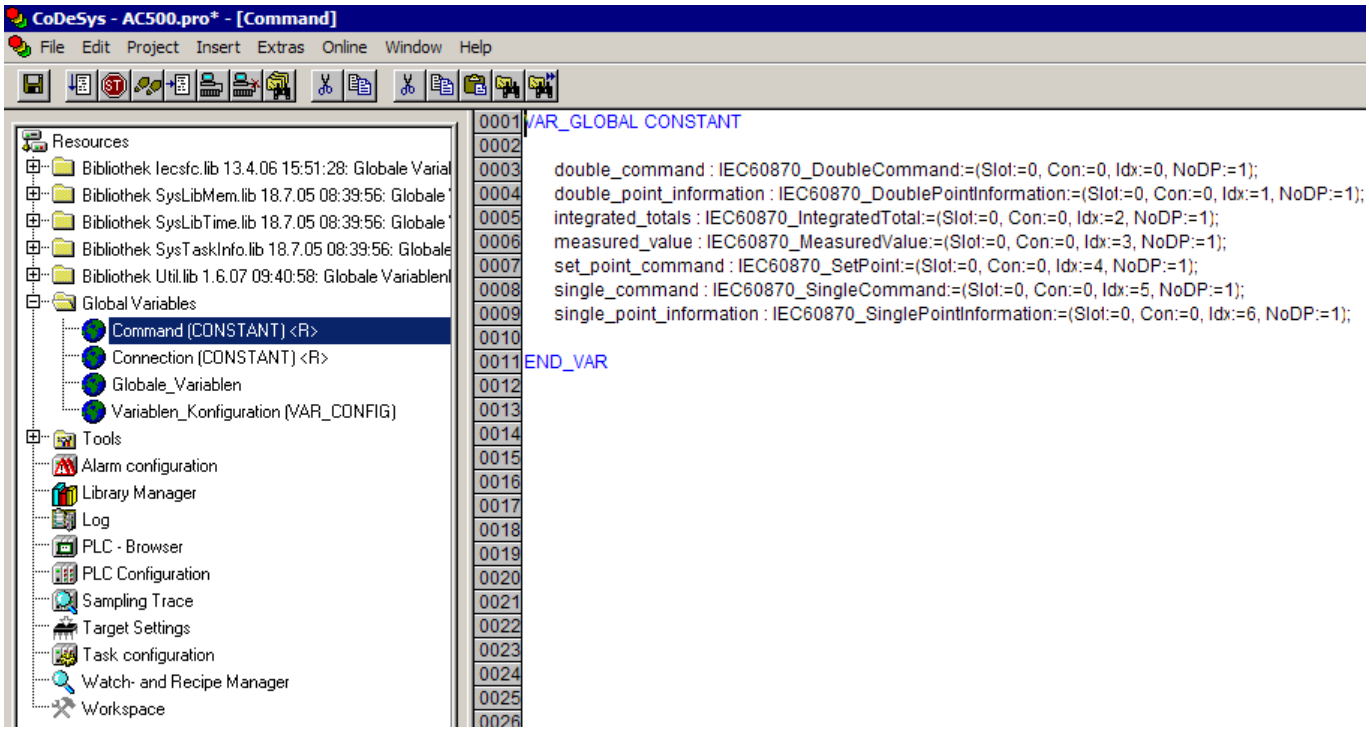
The created Control and Substations are read-only, i.e. the settings previously defined in Automation Builder can not be changed.



Commands

As well as the IEC 60870 Connections the created Data points, named Commands, are also available in CODESYS.

The created Commands are read-only, i.e. the settings previously defined in Automation Builder can not be changed.



Using control and substations in CODESYS

To map the IEC 60870-5-104 Control and Substation to CODESYS, use IEC 60870-5-104 function block library (see ↳ *Chapter 1.5.4.18 “IEC60870 library” on page 1351*).

The created Station (Control and Substation) which is available at the global variable list (see ↳ *Chapter 1.6.5.4.1.1 “Creating configuration data” on page 6196*) has to be assigned to the corresponding Input of the function blocks of the IEC60870 Library.

The created Control and Substation characteristics itself with:

- Unique names: Names of all connections (substation and control station)
- Unique addresses: All telecontrol addresses (consist of GADU1, GADU2, IA1, IA2, IA3) have to be unique within 1 telecontrol connection (all knots beneath 1 connection knot).
- Aliases: For the project planning the plug-in generates constant data structure aliases as global variables (read-only).

For each created connection (Substation and Control Station) the plug-in generates.

Example

```
VAR_GLOBAL CONSTANT
Control_Station : IEC60870_5_104_Connection:=(Slot:=0, Con:=0);
END_VAR
```

Description of the initialization values of the station

Slot: The number of the Communication Module slot (0: internal, 1: first external, ...)

Con: Index of connection (Substation / Control Station) beneath an IEC60870-5-104 node.

Using data points in CODESYS

To map the IEC 60870-5-104 Data Points to CODESYS, use IEC 60870-5-104 function block library (see ↳ *Chapter 1.5.4.18 “IEC60870 library” on page 1351*).

The created Data Points which are available at the global variable list have to be assigned to the corresponding Input of the function blocks of the IEC 60870 Library. See *Chapter 1.6.5.4.1.1 “Creating configuration data” on page 6196*.

For each Data Point group type an equivalent function block type in IEC60870 Library exists.

The implemented functionality of the IEC 60870-5-104 plug-in has the following characteristics:

- Unique names: Names of all connections (Substation and Control Station) and all names of the data groups are unique
- Unique addresses: All telecontrol addresses (consist of GADU1, GADU2, IA1, IA2, IA3) have to be unique within one telecontrol connection (all nodes beneath 1 connection node)
- Aliases: For the project planning the plug-in generates constant data structure aliases as global variables (read only)

For each created connection (Substation / Control Station) at the Automation Builder the plug-in generates 1 data point in CODESYS.

Example

```
VAR_GLOBAL CONSTANT
Rec_sp_inf_16 : IEC60870_SinglePointInformation:=(Slot:=0, Con:=0, Idx:=17, NoDP:=1);
END_VAR
```

Description of the initialization values of the station

Slot: The number of the Communication Module slot (0: internal, 1: first external, ...)

Con: Index of connection (Substation / Control Station) beneath an IEC60870-5-104 node.

Idx: Index of data group beneath a connection

NoDP: number of data points beneath this data group

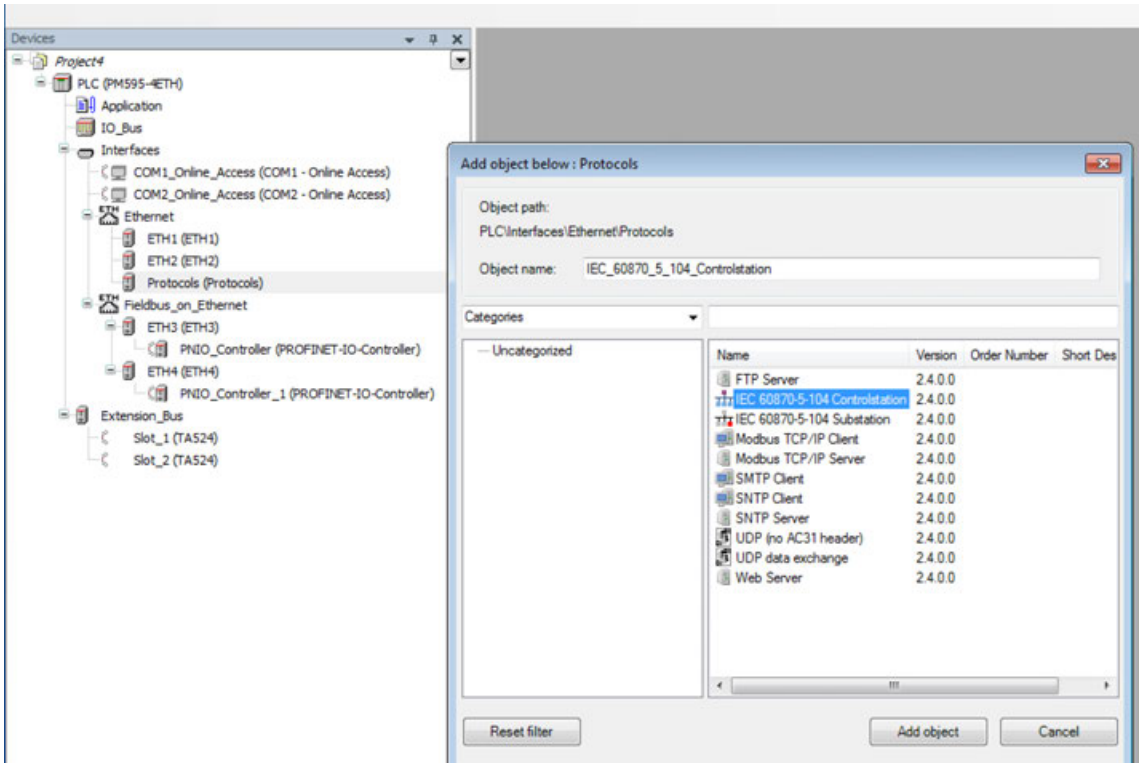
Configuration >= Automation Builder 1.1/CBP 2.4

Configuration changes >= Automation Builder 1.1/CBP 2.4

The IEC 60870 protocol allows link-ups between AC500 CPUs with onboard Ethernet (e.g. PM595-4ETH and PM591-2ETH) and external systems.

The link-up takes place via the onboard Ethernet interface of the CPU. As of Automation Builder Version 1.1 telecontrol is also supported for CPUs that provide more than one Ethernet interface (e.g. PM595-4ETH and PM591-2ETH). This allows to use different Ethernet interfaces for IEC 60870 connections, hence, telecontrol configuration is changed. Further, as of this version terminology is aligned with IEC 60870 standard and provides additional features that are described in this chapter. For a description on principle telecontrol configuration.

See Configuration *Chapter 1.6.5.3.2.3.1 “Configuration up to and including CBP 2.3” on page 6123*.



For further information on configuration changes, see the following chapters:

Control and Substations ≥ CBP 2.4 ↬ *Chapter 1.6.5.3.2.4.2 “Control station and substation configuration” on page 6139*

Import Export ≥ CBP 2.4 ↬ *Chapter 1.6.5.3.2.4.1 “Configuration changes >= Automation Builder 1.1/CBP 2.4” on page 6138*

Validity Check of Configuration ≥ CBP 2.4 ↬ *Chapter 1.6.5.3.2.4.4 “Validity check of configuration” on page 6158*

Control station and substation configuration

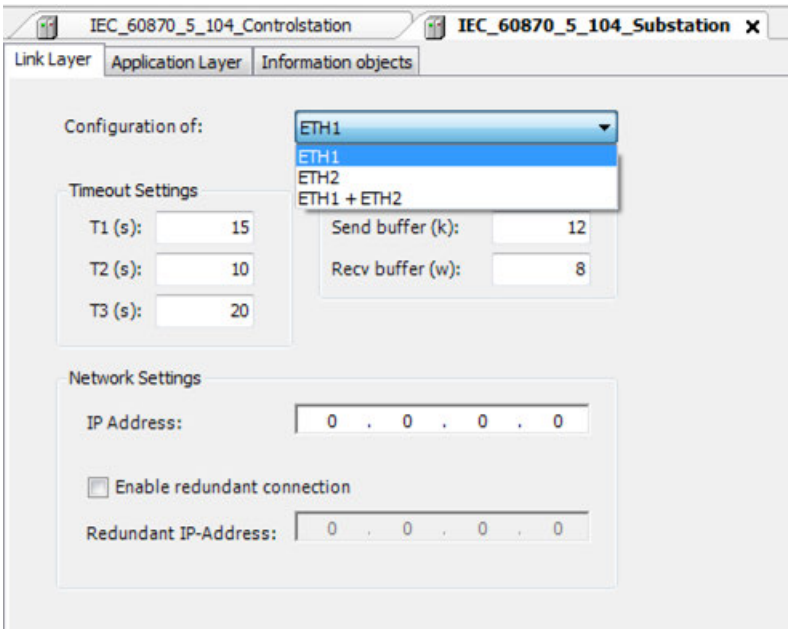
The CPU can work as both, control station and substation.

Control station	Client, master, controlling station: Synonyms for a higher-level station (central station, monitors other stations)
Control direction	Data transfer direction from the control station to the substation
Substation	Server, slave, controlled station: synonyms for a subordinate IEC 60870-5-104 telecontrol station (which is monitored)
Monitoring direction	Data transfer direction from the substation to the controlling station

Configure a control station in the device tree *PLC -> Interfaces -> Ethernet -> Protocols*:

1. Right-click *“Protocols ➔ Add objects”*.
2. Select the control station from the list and click *“Add object”*. Configure substations and further control stations in the same way. As of Automation Builder 1.1 any combination of control stations and substations can be configured, in due consideration of a total number of 10 stations.
3. Double-click the new control station node to open parameter configuration. In the Link Layer tab access to the Ethernet interface is configured.

Tab link layer



In order to provide flexible usage of control stations and substations as of Automation Builder 1.1 configuration of substations has been changed. As several substations can be operated with several Ethernet interfaces, select the Ethernet interface to be used from the pull-down menu. Enter the IP address to the control station and if required to another control station (redundant connection). If no IP address is defined, the substation accepts connection to any control station.

> *This field is not available in the Link Layer tab of control stations. Selection of ETH interface is only possible for substations. The control station is always configured on both interfaces by default.*

Timeout settings T1, T2, T3: The values for the connection control and message replication; timeout1/2/3.

Buffer settings This parameter gives the maximum number of outstanding messages and acknowledgement behavior.

Send buffer (k): Maximum difference receive sequence number to send state variable.

Rec buffer (w): Latest acknowledge after receiving w I format APDUs.

Network settings

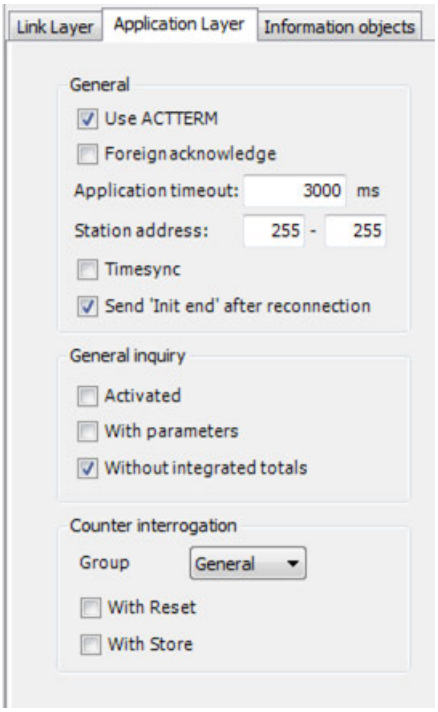
Network settings are available for control stations and for substations. The IP address of the control station and if available the IP address of another control station (redundant IP address) can be selected by the user.

For an overview on the configured Ethernet interfaces for the control stations and substations, double-click the *“Protocols”* node.

Tab application layer

Settings

The application layer is the communication layer with which the send and receive blocks work.



Use ACTTERM This parameter concerns only setpoints and commands. If this parameter is checked, an acknowledgement with set ‘actterm’ is generated as reason for transmission at the time at which the receive block is computed and outputs its telecontrol data at its output pins. On transmission side, the data block awaits the reception of this ACTTERM acknowledgement and reacts with its corresponding output (see *Chapter 1.5.4.18 “IEC60870 library” on page 1351*) to the reception of this acknowledgement. For commands with execution time, the acknowledgement is generated when the command is terminated, for commands with continuous execution time and for setpoints, the acknowledgement is generated when the data are output to the output pins.

ForeignAck-nowledge If this option is not enabled (default), a message that was sent is considered as ok as soon as transmission was successful. If you enable this option, a message that was sent is not considered as ok until a success message (foreign acknowledge) is returned from the receiver.

Application timeout This time indicates how long an acknowledgement will be awaited on the application level. An acknowledgement is generated only for commands and setpoints on the application level.

Station address The station address defines which station will be subject to a count query. The values define the 2 bytes for the common telecontrol address (Common addr.). The values concerned are as follows:

0: The station address is not used.

1...254: The count is queried on the station defined by the station address.

255: The count is queried on all accessible stations.

Timesync After each new establishment of a link and once per hour, a ‘coarse time synchronisation’ message is generated. This time synchronisation is only supported from AC500 to external systems. Time synchronisation from an external system to AC500 is not provided! Incoming time synchronisation messages are confirmed by the process station but not executed. Greenwich Mean Time (GMT) is used as the time for the synchronisation.

Send 'Init end' after reconnection After each establishment of a link or only in connection with the first establishment of a link and after reconfiguration, an init end message is generated. After the init end message, there is a general inquiry, if configured.

General inquiry

Activated This parameter concerns only real values, discrete values and status values. The device module generates a general inquiry message after each new establishment of a link. The other side then generates a message with the reason for transmission ‘general inquiry’ for every data point and subsequently an init end message. This procedure ensures that, in the event of a new establishment of a link, all data are available on the reception side in topical form.

With parameters If general inquiry is activated the parameter values are sent.

Without integrated totals With a general inquiry no integrated total values are sent.

Counter interrogation

Group General, 1 .. 4: The count inquiry is executed for a specific group of counters (1 .. 4). The count inquiry is executed for all groups of counters.

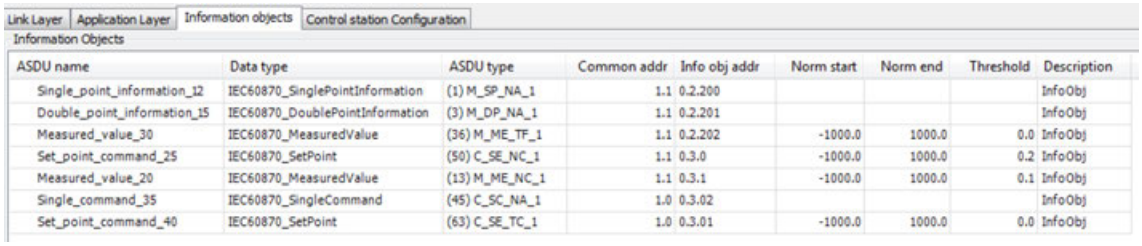
With reset The reset quality bit is sent along with the count inquiry.

With relocate The relocate quality bit is sent along with the count inquiry.

Tab information objects

Open the *“Information object”* tab to configure so called information objects and a common address (known as 'data points' and 'Global address' in former Automation Builder versions). In this tab different information objects and their services for transmission are defined. A data point or information object is identified via a system-wide unambiguous address containing a maximum 5 bytes.

1. Right-click in the empty view and select *“Add Information Object with ASDU”* to add a data group. Select the desired object from the list (e.g. M_SP_NA_1).
   ⇨ An information object with a corresponding ASDU (Application Service Data Unit) is created.
2. Configure the settings in the *“Information Object”* tab to your convenience.



ASDU name	Data type	ASDU type	Common addr	Info obj addr	Norm start	Norm end	Threshold	Description
Single_point_information_12	IEC60870_SinglePointInformation	(1) M_SP_NA_1	1.1	0.2.200				InfoObj
Double_point_information_15	IEC60870_DoublePointInformation	(3) M_DP_NA_1	1.1	0.2.201				InfoObj
Measured_value_30	IEC60870_MeasuredValue	(36) M_ME_TF_1	1.1	0.2.202	-1000.0	1000.0	0.0	InfoObj
Set_point_command_25	IEC60870_SetPoint	(50) C_SE_NC_1	1.1	0.3.0	-1000.0	1000.0	0.2	InfoObj
Measured_value_20	IEC60870_MeasuredValue	(13) M_ME_NC_1	1.1	0.3.1	-1000.0	1000.0	0.1	InfoObj
Single_command_35	IEC60870_SingleCommand	(45) C_SC_NA_1	1.0	0.3.02				InfoObj
Set_point_command_40	IEC60870_SetPoint	(63) C_SE_TC_1	1.0	0.3.01	-1000.0	1000.0	0.0	InfoObj

3. Double-click a table cell to modify pre-set values. For some ASDUs additional sub information objects can be configured. For this, right-click the already existing ASDU and select *“Add Information Object”* to selected ASDU option. This allows configuration of 16 data points at the most (depending on the ASDU type). With *“Remove Information Object”* the selected ASDU is deleted.

Description of the columns

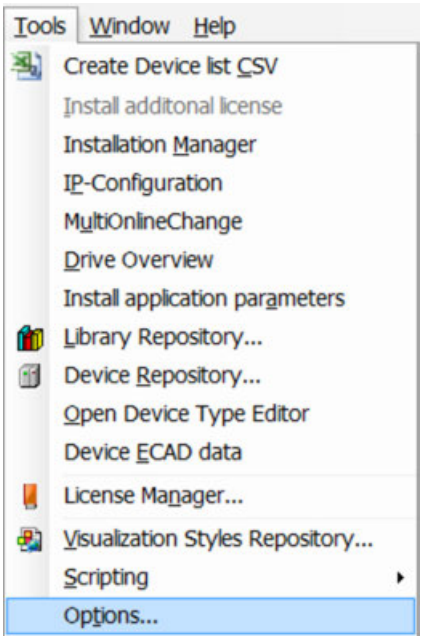
- ASDU name: node name of the information object (name of the ASDU).
- Data type: Data type of the ASDU.
- ASDU type: Type of ASDU.
- Common addr: Common address of the ASDU (known as 'Global Address' in former AB Versions). Byte 1/2 of the common telecontrol address of the block (range: 0...255).
- Info obj addr: Together with common address Info obj addr defines the endpoint (range: 0...255).
- Norm start: Low limit (0 %) of the normalized range for real values and setpoints.
- Norm end: High limit (100 %) of the normalized range for real values and setpoints.
- Threshold: Threshold limit beyond which a change of the input value referred to.
- Description: Table cell for free text. Use this field to describe your configuration settings e.g. differences between configuration variants.

Format of common addr and info obj addr

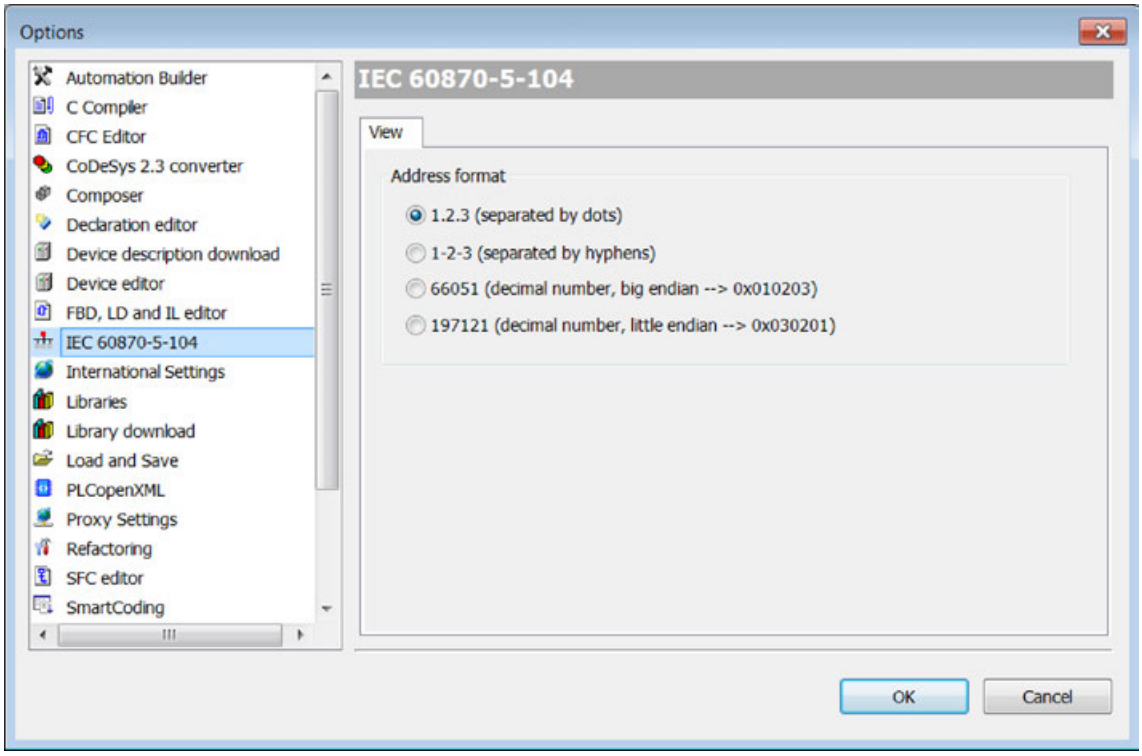
The following adress formats of your entries in the columns *Common addr* and *Info obj addr* of the Tab *Information Objects* are possible:

- 1.2 and 3.4.5 (Default format)
- 1-2 and 3-4-5
- 258 or hex 0x102 and 197637 or hex 0x30405
- 513 or hex 0x201 and 328707 or hex 0x50403

Previously you have to choose your preferred adress format:



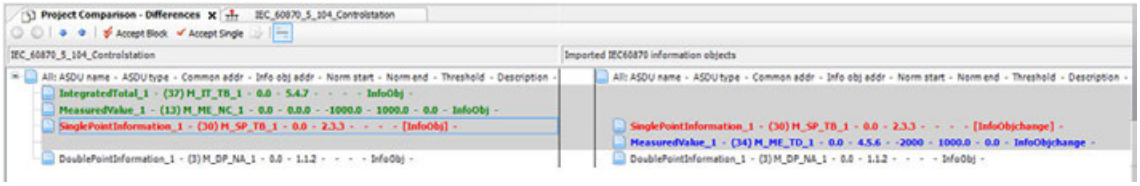
1. Click *“Tools”*and then *“Options...”*
   ⇨ The Window *Options* appears



2. Select *IEC 60870-5-104*, make your choice and click *OK*.

Import options of information objects

The User can accept the imported IEC60870 information objects as single change or change as block.



IEC60870-5-104 Multiple connections

An AC500 with more than one substation connection must be able to identify the corresponding control station clearly. This identification takes place exclusively via the control station’s IP address. In order to make it possible for a non-redundant control station to have redundant access to a substation with 2 Ethernet connections. The local substation address is ignored during connection establishment.

In the following descriptions, the term station must not be confused with the individual connection. One station can have several connections. An IEC60870-5-104 communication always takes place between a control station and a substation. A control station can manage several substations and also simultaneously be a substation for one or several control stations. However, these must then be realized using different stations.

A PLC may **not** be configured for another PLC repeatedly as a substation or a control station unless a disjunctive Ethernet infrastructure is used for this.

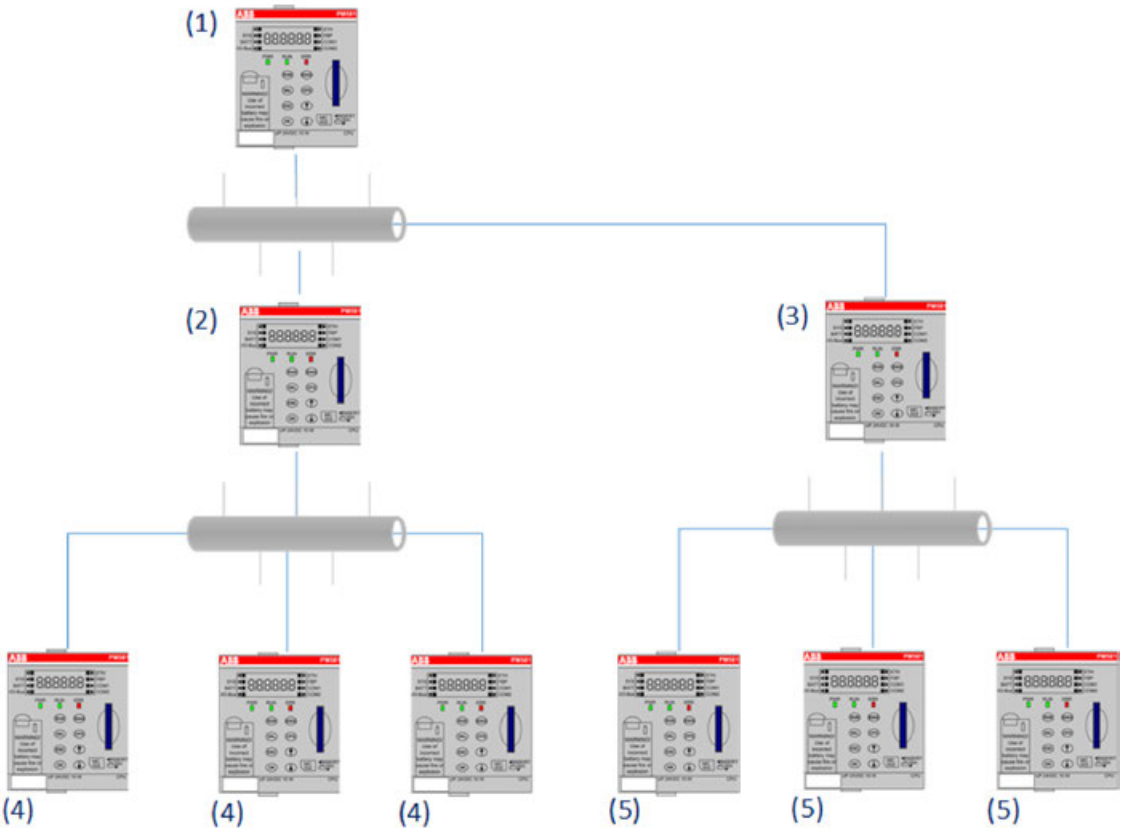
Redundant connections must be specified as such in the configuration.

An AC500 can be used only once as control station for another AC500, it makes no sense to use the same AC500 repeatedly as a control station for the same substation. Such a structure is configured as a redundant control station as long as only one AC500 exists as a control station per substation. However, this control station may have 2 IP addresses. Therefore, this configuration must either have the IP address 0.0.0.0 entered on the substation for the control station, meaning that all IP addresses are accepted and no other control station can access this AC500 or alternatively the possible control station addresses must be specified (ETH1 and ETH2).

Tree constellation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

If you plan to control several substations with the AC500, they can be cascaded. This results in a tree structure.



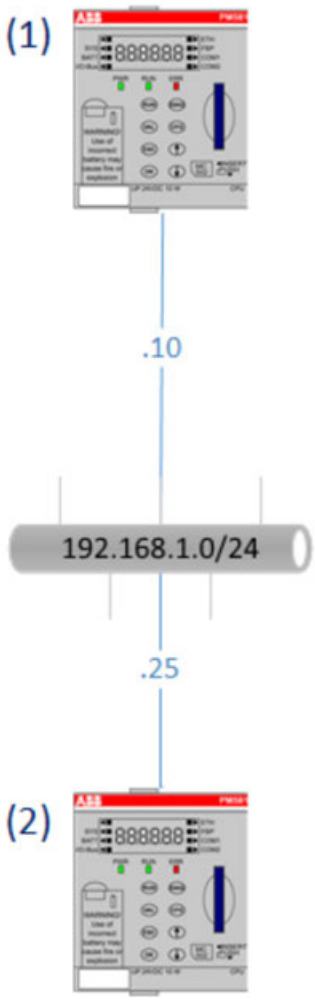
(1) 2 control stations
(2) Substation and 3 control stations
(3) Substation and 3 control stations
(4) Substation
(5) Substation

Structures of connections

In the following, the notation 192.168.1.0/24 is used for TCP/IP networks. Here, the figure /24 specifies the network mask with 255.255.255.0 and 192.168.1.0 describes the network. The valid addresses for this Class C network are 192.168.1.1 to 192.168.1.254! Only the last byte of the address is provided on the respective devices, with e.g. .10. This means that the respective device has the address 192.168.1.10.

Minimal structure

A control station with an Ethernet interface is connected to a substation with an Ethernet interface.



(1) Control station
(2) Substation

Configuration at control station

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The respective substation IP address must be specified at the control station. For this, in the network settings of the control station (1) enter the IP address of the substation (in the example: 192.168.1.25). Option *“Enable redundant connection”* must be disabled.

Configuration at substation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

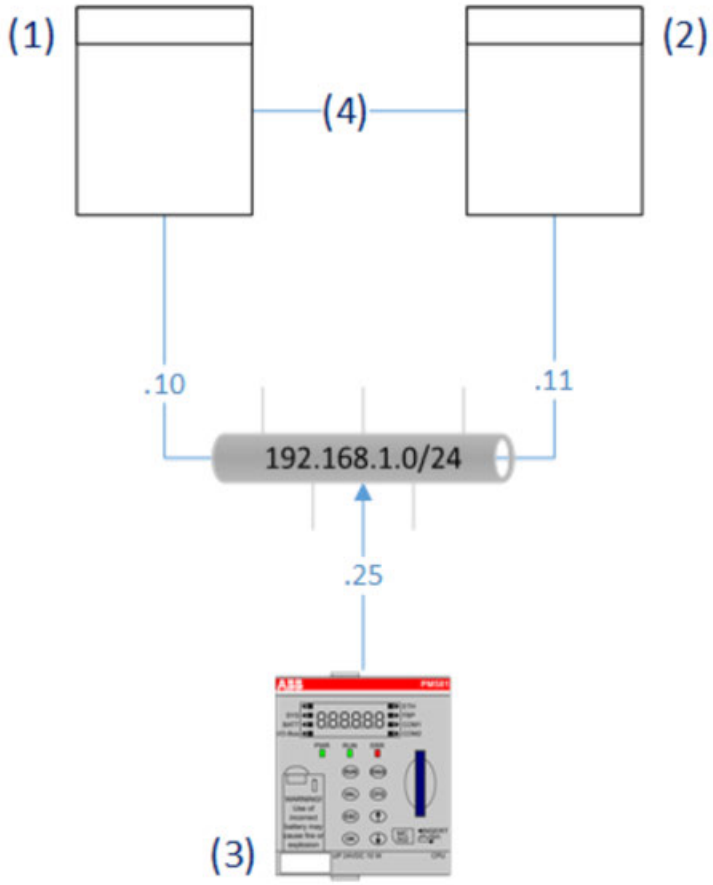
Either the control station IP address or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP address of the control station (in the example: 192.168.1.10). Option *“Enable redundant connection”* must be disabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station can be configured on this controller for a further substation.*

Minimal redundancy structure

The most simple redundant structure with an AC500 consists of a redundant control station (not AC500) which is connected to the AC500 substation with 2 different IP addresses. These redundant control stations must synchronize which control station is active.

Only one control station can be active at any given time.



(1) Control station 1A (Not AC500)
(2) Control station 1B (Not AC500)
(3) Substation
(4) Redundancy link

Configuration at control stations

The respective substation IP address must be specified at the control stations 1 and 2 (not AC500). For this, in the network settings of both control stations enter the IP address of the substation (in the example: 192.168.1.25).

Configuration at substation

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

Either the control station IP addresses or the general address 0.0.0.0 must be specified at the substation (3). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.1.11). Option *“Enable redundant connection”* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station can be configured on this controller for a further substation.*

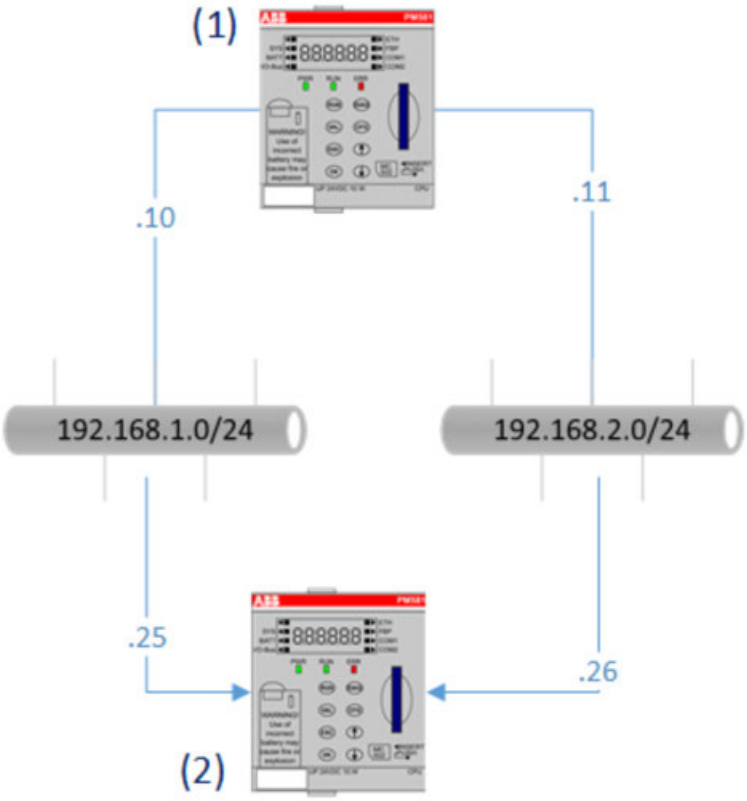
Network redundancy

For network redundancy a control station can reach a substation via 2 paths.

Both the control station and the substation can have 2 different IP addresses. Without special network routing, 2 separate networks should exist, within which both the substation and the control station each have 2 interfaces.

Possible variants of network redundancy are described in the following.

Network redundancy with 2 separate networks



(1) Control station with 2 redundant paths
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26). Option *"Enable redundant connection"* must be enabled.

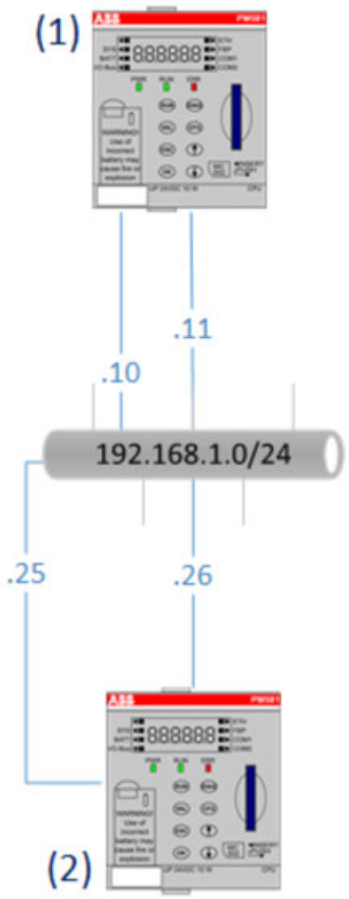
Configuration at substation

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 1 network and 2 Ethernet ports in substation



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.1.26). Option *"Enable redundant connection"* must be enabled.

Configuration at substation

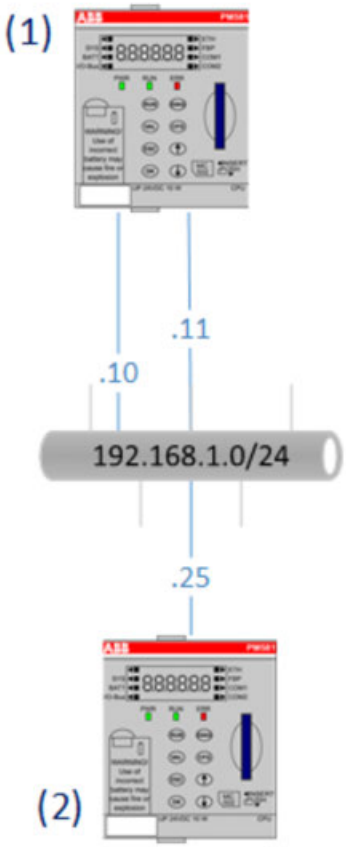
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 1 network and 1 Ethernet port in substation

> *No online redundancy.*
>
> *Only one connection will be established.*



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 1 Ethernet interface

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 0.0.0.0). Option *"Enable redundant connection"* must be disabled.

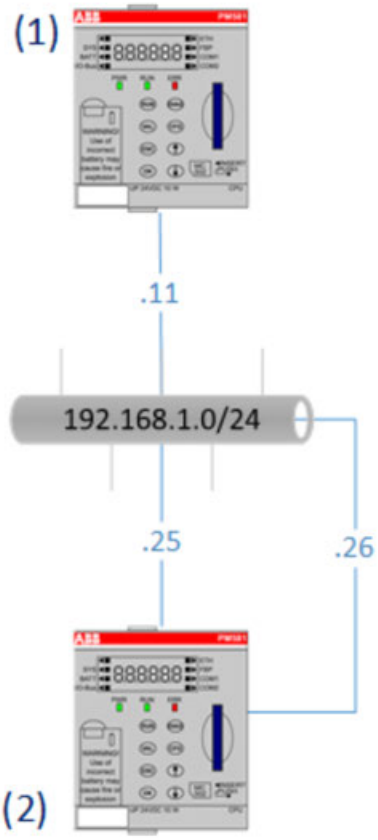
Configuration at substation

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Network redundancy with 2 Ethernet ports in substation



(1) Control station with 2 paths to reach substation
(2) 1 Substation with 2 Ethernet interfaces

Configuration at control stations

PM591-2ETH, PM595-4ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.1.26). Option *"Enable redundant connection"* must be enabled.

Configuration at substation

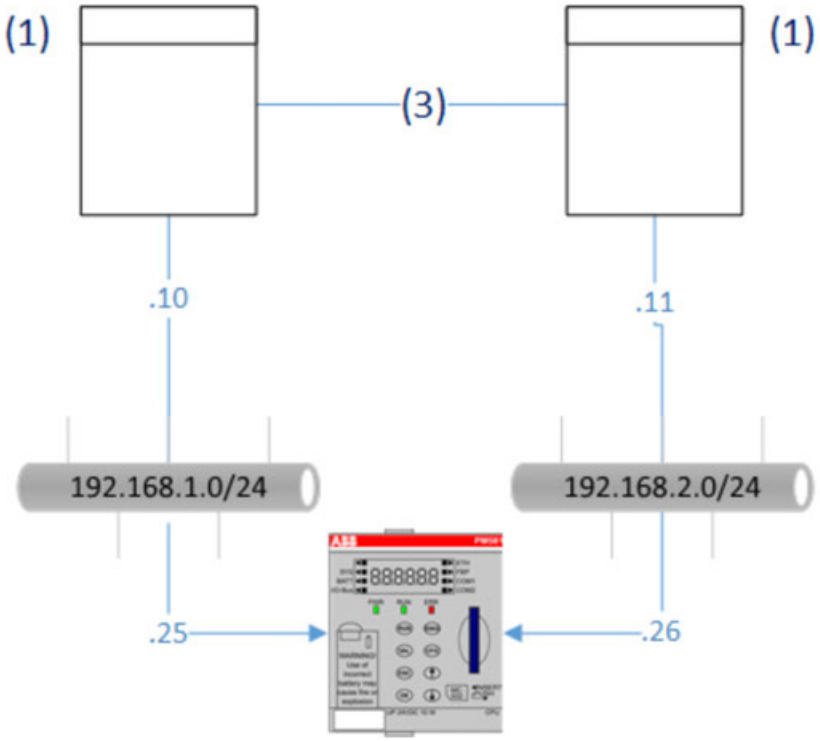
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.11 and 0.0.0.0). Option *"Enable redundant connection"* must be disabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Full control station redundancy

A control stationcan consist of two fully redundant units (not AC500s), which are connected via a redundancy link. These control stations must ensure that only one of them at a time is actively connected to the substation and communicates with it. The inactive control station, however, can establish non-active connection with a substation and monitor it with keep alive packages.



(1) 2 redundant Control stations (Not AC500)
(2) 1 Substation with redundant Control station and 2 Ethernet interfaces (2nd port)
(3) Redundancy link

Configuration at control stations

The substation's IP address must be specified at the control stations (1) (not AC500). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26).

Configuration at substation

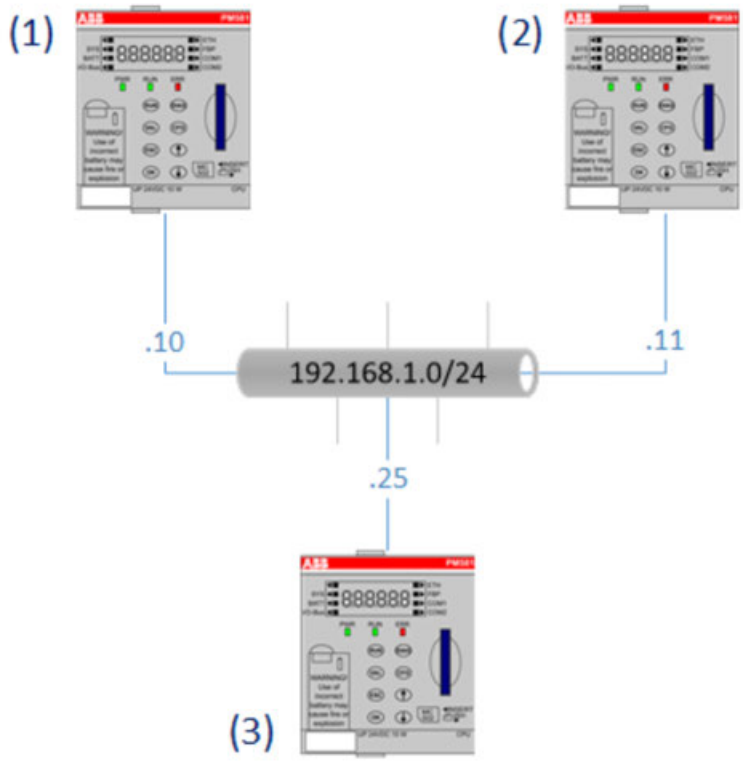
PM591-2ETH, PM595-4ETH, PM5650-2ETH:

Either the control station's IP addresses or the general address 0.0.0.0 must be specified at the substation (2). For this, in the network settings of the substation enter the IP addresses of the control station (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be enabled.

> *If the general address 0.0.0.0 is used at the substation, no further control station on another substation can be configured on this controller. Equally, the substation connection must be activated for both interfaces.*

Multiple control stations on the same network

As of firmware version 2.4, an AC500 can be used as a substation for several control stations. For this, the control stations must be distinguished by their IP addresses. Should a control station have more than one IP address (redundancy), both possible IP addresses should also be entered for the allocated substation connection. As a result, even despite being equipped with several Ethernet interfaces, a device can only be one allocated control station at a time for a determined substation. Thus, several substations can be configured for different control stations on a AC500.



(1) Control station 1
(2) Control station 2
(3) 2 Substations (IEC60870-5-104 2nd Connection)

Configuration at control stations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The substation’s IP address must be specified at the control stations. For this, in the network settings of the control station (1 and 2) enter the IP addresses of the substation (in the example: 192.168.1.25). Option *“Enable redundant connection”* must be disabled.

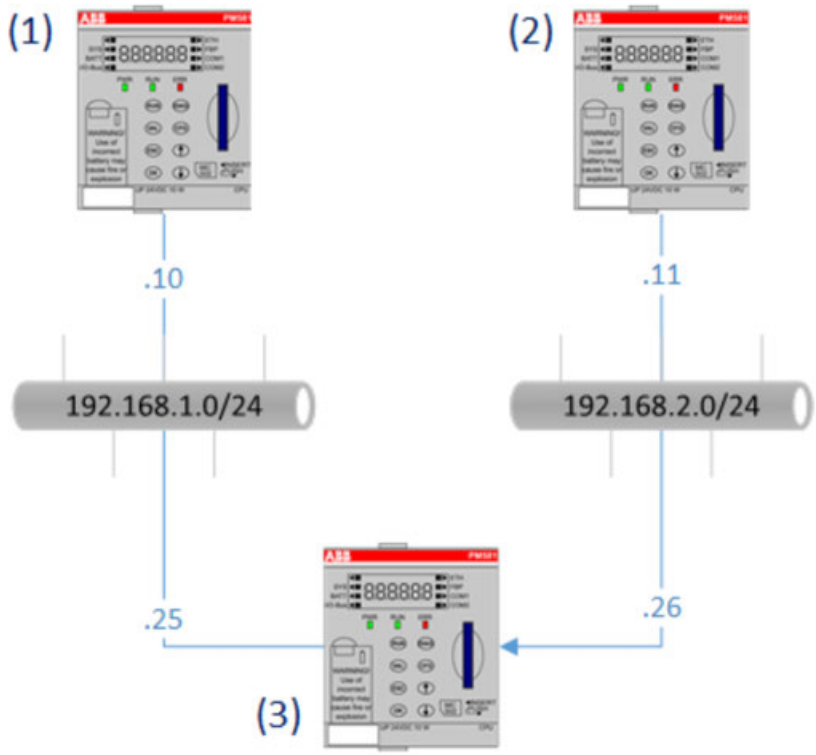
Configuration at substations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

Both control station's IP addresses must be specified at the substation (3). For this, in the network settings of the substation enter the IP addresses of the control stations (in the example: 192.168.1.10 and 192.168.1.11). Option *“Enable redundant connection”* must be disabled.

Multiple control stations on different networks

As of firmware version 2.4, an AC500 can have several local Ethernet interfaces which can be used for separate control station connections. For this, a control station must be identified via its IP address. The substation address used locally is not used to distinguish a connection in order to enable a network and therefore route redundancy. On AC500, the acception of IEC60870-5-104 connections on an interface can only be prevented.



(1) Control station 1
(2) Control station 2
(3) 2 Substations with 2 Ethernet interfaces (2nd port and 2nd connection)

Configuration at control stations

PM57x-/ PM58x-/ PM59x-ETH, PM5650-2ETH:

The substation's IP addresses must be specified at the control stations (1 and 2). For this, in the network settings of the control station enter the IP addresses of the substation (in the example: 192.168.1.25 and 192.168.2.26). Option *"Enable redundant connection"* must be disabled.

Configuration at substations

PM591-ETH, PM595-ETH, PM5650-2ETH:

Both control station's IP addresses must be specified at the substation (3) under both substation connections. For this, in the network settings of the substation enter the IP addresses of the control stations (in the example: 192.168.1.10 and 192.168.2.11). Option *"Enable redundant connection"* must be disabled.

Double connection

*This configuration **does** work.*

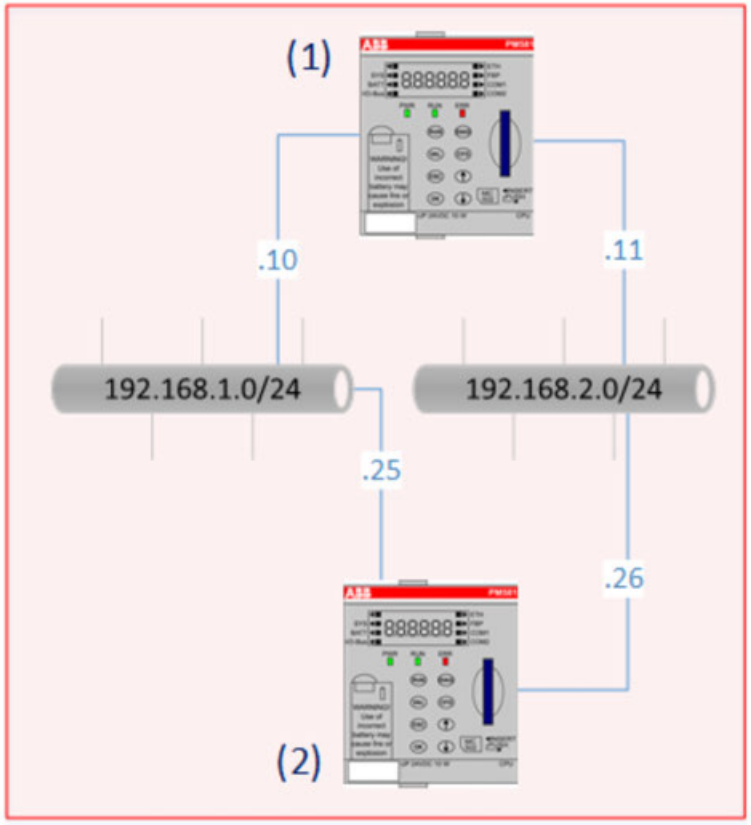
*But it is **senseless!***

It is possible to configure a double connection between 2 stations using 2 separate networks (at least logically separated sub-networks).

However, such a setup has no advantages via-à-vis the minimal structure right at the start *Chapter 1.6.5.3.2.4.2.6.1.1 “Minimal structure” on page 6146*.

For this setup, connection data must be double configured and double resources are also required at the stations, not providing any advantages whatsoever.

Rather the opposite is true, because such configurations are highly prone to errors.



(1) 2 Control stations with 2 Ethernet interfaces
(2) 2 Substations with 2 Ethernet interfaces

Faulty configuration

*This configuration **does not** work!*

If an AC500 is configured as a control station, the interface which is used to reach the substation is not defined.

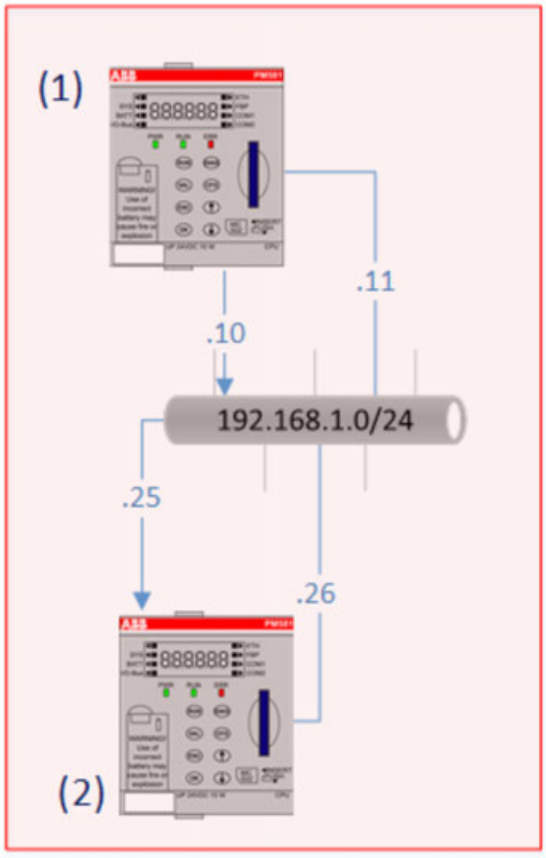
The decision as to which interface is used for this is taken by TCP/IP when running.

It is also dependent on the current network configuration.

Here, the current link status and the order of link recognition may be decisive for the interface to be used.

Such a scenario would not result in stable communication as both substations cannot clearly distinguish the control stations.

Instead, the connection management for a substation will assume that the control station has lost the connection and then establishes a connection.



(1) 2 Control stations with 2 Ethernet interfaces
(2) 2 Substations with 2 Ethernet interfaces

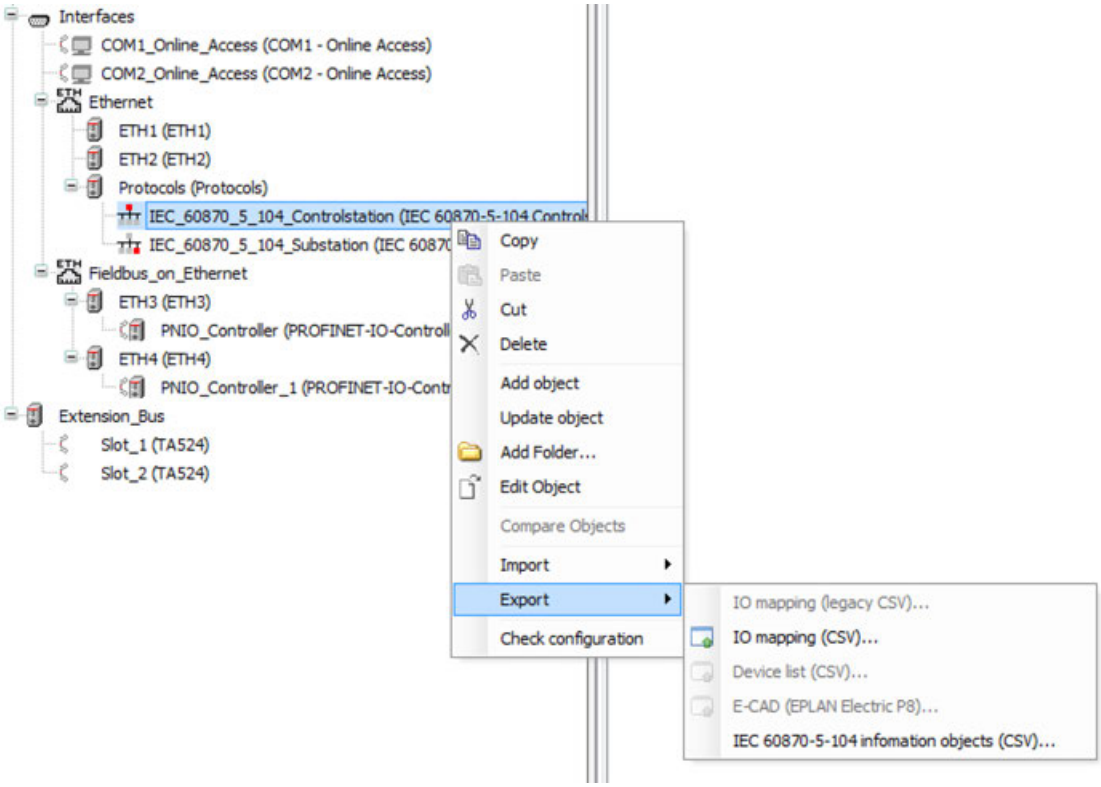
Export a CSV file

As an alternative many values can be modified at a time by exporting the configuration to a CSV file. After modifying the file data, import the CSV file ↳ *Chapter 1.6.5.3.2.4.3 “Import/Export functionality ” on page 6157*.

Import/Export functionality

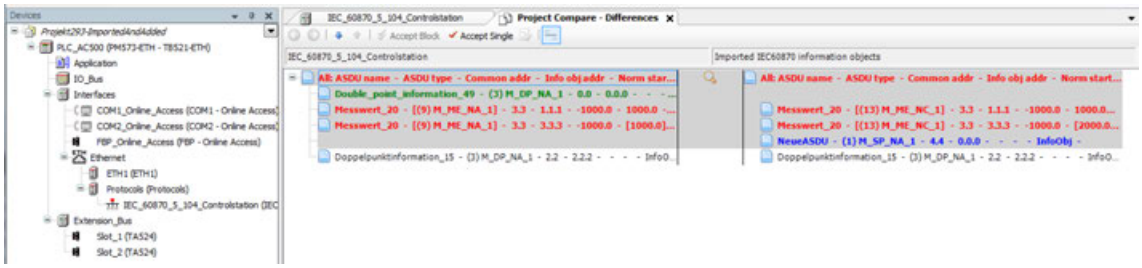
As of Automation Builder 1.1 (CBP >= 2.4) configuration of control stations and substations can be exported/imported via CSV file. Open the CSV file with a spreadsheet software (e.g. Microsoft Excel) and modify the values within the file to your convenience:

1. Export configuration data: right-click the node of the control station or substation to be exported.



2. Click *"Export ➔ IEC 60870-5-104 information objects (CSV)"* and store the CSV file to a desired directory.
3. Open the CSV file with a spreadsheet software (e.g. Microsoft Excel) and change the values to your convenience. Added table columns are only accepted after the last column.
4. Import configuration data: right-click the node of the control station or substation that has been exported previously.
5. Click *"Import ➔ IEC 60870-5-104 information objects (CSV)"* and select the CSV file from the file system. Configuration data is imported.

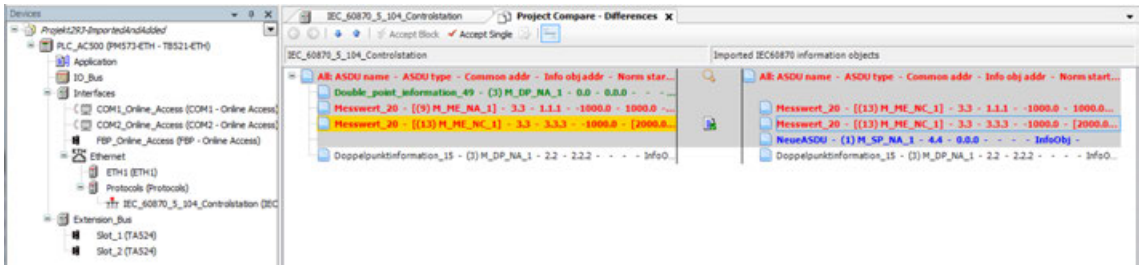
As of Automation Builder 1.1.1 during file import the project data is compared with the project data that is already available. In order to prevent data from being overwritten inadvertently, you can select the data that shall be imported in the *"Project Compare - Differences"* window:



Data on the left side of the window refers to already available project data. This data is displayed under *"Control station ➔ Information objects"* tab. Data on the right side of the window refers to new data that can be imported after your confirmation. Decide whether to import (and overwrite) the data or not.

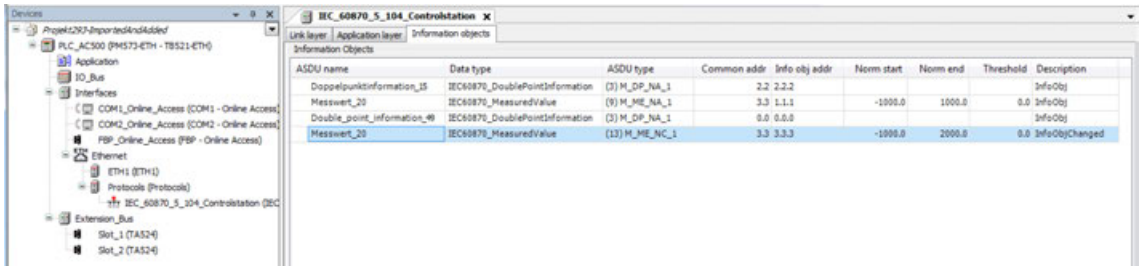
- Data in black color means the existing data and the data to be imported is identical.
- Data in red color means the existing data and the data to be imported differ. Decide whether to import the new data (and to overwrite the existing data) or not.
- Data in blue color means, the data to be imported is new and will be added to the existing data.
- Data that has been confirmed for the import already is displayed in green color (after clicking the *[Accept Single]* button).

In order to move data from one side of the window to another, select the data and click the *[Accept Single]* button. Data is highlighted in yellow.



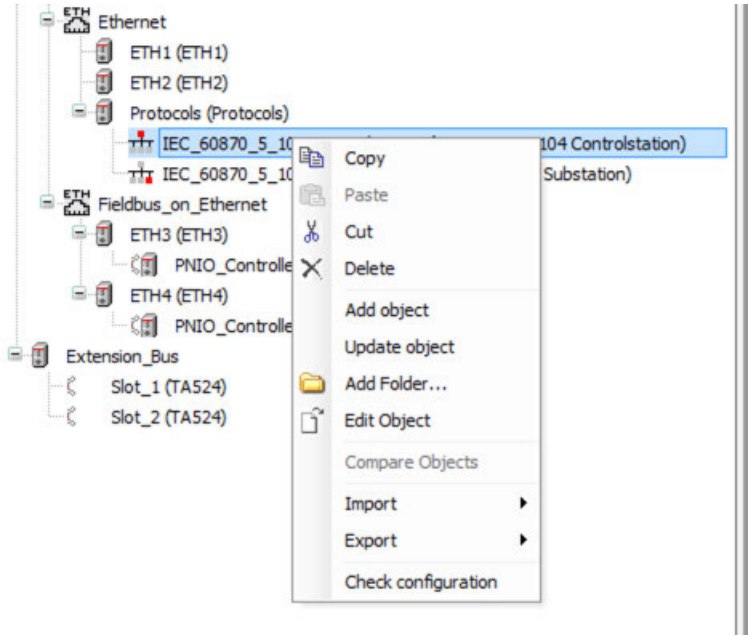
To confirm the import of all new data, click the top entry (here: All: ASDU name - ASDU tpye - Common addr - ...). Then, click the *[Accept Single]* button.

Close the *"Project Compare - Differences"* tab, save your project and confirm the message. The changes are displayed in the *"Information objects"* tab.



Validity check of configuration

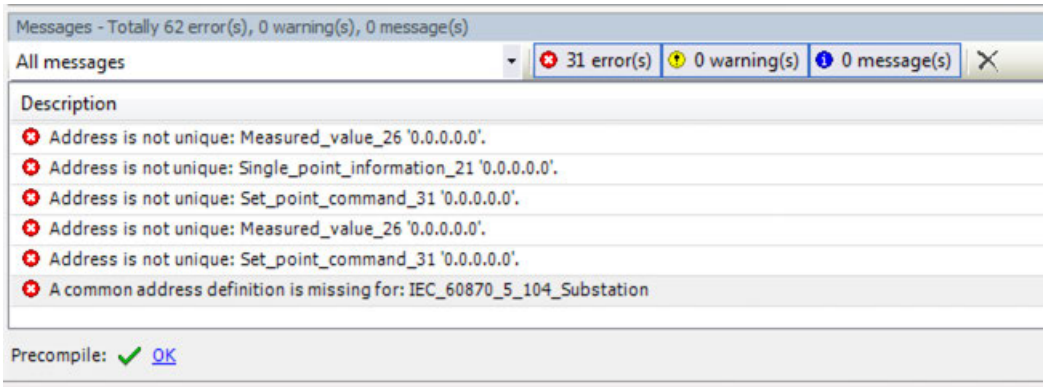
We recommend you to verify the IEC configuration of control stations and substations: Right-click a control station or substation -> Check configuration.



The check will look for the following topics:

- Duplicate addresses.
- Stations without any Information objects.
- ASDU names, which are not unique.

When a check finds errors or incompatibilities this will be reported in a separate messages view at the buttom of the window:



With a double-click on the error line, the part of the configuration with the violation will be opened. Now, you can correct the error.

IEC60870 compatibility list

AC500 V2.4 IEC60870-5-104 Compatibility List

9 Interoperability

This companion standard presents sets of parameters and alternatives from which subsets must be selected to implement particular telecontrol systems. Certain parameter values, such as the choice of "structured" or "unstructured" fields of the INFORMATION OBJECT ADDRESS of ASDUs represent mutually exclusive alternatives. This means that only one value of the defined parameters is admitted per system. Other parameters, such as the listed set of different process information in command and in monitor direction allow the specification of the complete set or subsets, as appropriate for given applications. This clause summarizes the parameters of the previous clauses to facilitate a suitable selection for a specific application. If a system is composed of equipment stemming from different manufacturers, it is necessary that all partners agree on the selected parameters.

The interoperability list is defined as in IEC 60870-5-101 and extended with parameters used in this standard. The text descriptions of parameters which are not applicable to this companion standard are strike-through (corresponding check box is marked black).

NOTE In addition, the full specification of a system may require individual selection of certain parameters for certain parts of the system, such as the individual selection of scaling factors for individually addressable measured values.

The selected parameters should be marked in the white boxes as follows:

☐ Function or ASDU is not used

☒ Function or ASDU is used as standardized (default)

☒ Function or ASDU is used in reverse mode

☒ Function or ASDU is used in standard and reverse mode

The possible selection (blank, X, R, or B) is specified for each specific clause or parameter.

A black check box indicates that the option cannot be selected in this companion standard.

9.1 System or device

(system-specific parameter, indicate definition of a system or a device by marking one of the following with "**X**")

☐ System definition

☐ Controlling station definition (Master)

☐ Controlled station definition (Slave)

9.2 Network configuration

(network-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~Point-to-point~~ ■ ~~Multipoint~~

■ ~~Multiple point-to-point~~ ■ ~~Multipoint star~~

AC500 V2.4 IEC60870-5-104 Compatibility List

9.3 Physical layer

(network-specific parameter, all interfaces and data rates that are used are to be marked "**X**")

Transmission speed (control direction)

Unbalanced interchange Circuit V.24/V.28 Standard	Unbalanced interchange Circuit V.24/V.28 Recommended if >1 200 bit/s	Balanced interchange Circuit X.24/X.27	
~~100 bit/s~~	~~2 400 bit/s~~	~~2 400 bit/s~~	~~56 000 bit/s~~
~~200 bit/s~~	~~4 800 bit/s~~	~~4 800 bit/s~~	~~64 000 bit/s~~
~~300 bit/s~~	~~9 600 bit/s~~	~~9 600 bit/s~~	
~~600 bit/s~~		~~19 200 bit/s~~	
~~1 200 bit/s~~		~~38 400 bit/s~~	

Transmission speed (monitor direction)

Unbalanced interchange Circuit V.24/V.28 Standard	Unbalanced interchange Circuit V.24/V.28 Recommended if >1 200 bit/s	Balanced interchange Circuit X.24/X.27	
~~100 bit/s~~	~~2 400 bit/s~~	~~2 400 bit/s~~	~~56 000 bit/s~~
~~200 bit/s~~	~~4 800 bit/s~~	~~4 800 bit/s~~	~~64 000 bit/s~~
~~300 bit/s~~	~~9 600 bit/s~~	~~9 600 bit/s~~	
~~600 bit/s~~		~~19 200 bit/s~~	
~~1 200 bit/s~~		~~38 400 bit/s~~	

9.4 Link layer

(network-specific parameter, all options that are used are to be marked "X". Specify the maximum frame length. If a non-standard assignment of class 2 messages is implemented for unbalanced transmission, indicate the Type ID and COT of all messages assigned to class 2.)

~~Frame format FT 1.2, single character 1 and the fixed time out interval are used exclusively in this companion standard.~~

Link transmission

- ~~Balanced transmission~~
- ~~Unbalanced transmission~~

Frame length

- ~~Maximum length L (number of octets)~~

Address field of the link

- ~~not present (balanced transmission only)~~
- ~~One octet~~
- ~~Two octets~~
- ~~Structured~~
- ~~Unstructured~~

AC500 V2.4 IEC60870-5-104 Compatibility List

When using an unbalanced link layer, the following ASDU types are returned in class 2 messages (low priority) with the indicated causes of transmission:

■ ~~The standard assignment of ASDUs to class 2 messages is used as follows:~~

Type identification	Cause of transmission
9, 11, 13, 21	<1>

■ ~~A special assignment of ASDUs to class 2 messages is used as follows:~~

Type identification	Cause of transmission

~~Note: (In response to a class 2 poll, a controlled station may respond with class 1 data when there is no class 2 data available).~~

9.5 Application layer

Transmission mode for application data

Mode 1 (Least significant octet first), as defined in 4.10 of IEC 60870-5-4, is used exclusively in this companion standard.

Common address of ASDU
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~ [X] Two octets

Information object address
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~
■ ~~Two octets~~
[X] Three octets

[ ] Structured
[ ] Unstructured

Cause of transmission
(system-specific parameter, all configurations that are used are to be marked "**X**")

■ ~~One octet~~ [X] Two octets (with originator address). Originator address is set to zero if not used

Length of APDU
(system-specific parameter, specify the maximum length of the APDU per system)

The maximum length of APDU for both directions is 253. It is a fixed system parameter.

■ Maximum length of APDU per system in control direction

AC500 V2.4 IEC60870-5-104 Compatibility List

■ Maximum length of APDU per system in monitor direction

Selection of standard ASDUs

Process information in monitor direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<1> :=	Single-point information	M_SP_NA_1
■	~~<2> :=~~	~~Single-point information with time tag~~	~~M_SP_TA_1~~
X	<3> :=	Double-point information	M_DP_NA_1
■	~~<4> :=~~	~~Double-point information with time tag~~	~~M_DP_TA_1~~
	<5> :=	Step position information	M_ST_NA_1
■	~~<6> :=~~	~~Step position information with time tag~~	~~M_ST_TA_1~~
	<7> :=	Bitstring of 32 bit	M_BO_NA_1
■	~~<8> :=~~	~~Bitstring of 32 bit with time tag~~	~~M_BO_TA_1~~
X	<9> :=	Measured value, normalized value	M_ME_NA_1
■	~~<10> :=~~	~~Measured value, normalized value with time tag~~	~~M_ME_TA_1~~
X	<11> :=	Measured value, scaled value	M_ME_NB_1
■	~~<12> :=~~	~~Measured value, scaled value with time tag~~	~~M_ME_TB_1~~
X	<13> :=	Measured value, short floating point value	M_ME_NC_1
■	~~<14> :=~~	~~Measured value, short floating point value with time tag~~	~~M_ME_TC_1~~
X	<15> :=	Integrated totals	M_IT_NA_1
■	~~<16> :=~~	~~Integrated totals with time tag~~	~~M_IT_TA_1~~
■	~~<17> :=~~	~~Event of protection equipment with time tag~~	~~M_EP_TA_1~~
■	~~<18> :=~~	~~Packed start events of protection equipment with time tag~~	~~M_EP_TB_1~~
■	~~<19> :=~~	~~Packed output circuit information of protection equipment with time tag~~	~~M_EP_TC_1~~
■	<20> :=	Packed single-point information with status change detection	M_SP_NA_1
■	<21> :=	Measured value, normalized value without quality descriptor	M_ME_ND_1
X	<30> :=	Single-point information with time tag CP56Time2a	M_SP_TB_1
X	<31> :=	Double-point information with time tag CP56Time2a	M_DP_TB_1
	<32> :=	Step position information with time tag CP56Time2a	M_ST_TB_1
	<33> :=	Bitstring of 32 bit with time tag CP56Time2a	M_BO_TB_1
X	<34> :=	Measured value, normalized value with time tag CP56Time2a	M_ME_TD_1
	<35> :=	Measured value, scaled value with time tag CP56Time2a	M_ME_TE_1
X	<36> :=	Measured value, short floating point value with time tag CP56Time2a	M_ME_TF_1
X	<37> :=	Integrated totals with time tag CP56Time2a	M_IT_TB_1
■	<38> :=	Event of protection equipment with time tag CP56Time2a	M_EP_TD_1
■	<39> :=	Packed start events of protection equipment with time tag CP56Time2a	M_EP_TE_1
■	<40> :=	Packed output circuit information of protection equipment with time tag CP56Time2a	M_EP_TF_1

In this companion standard only the use of the set <30> – <40> for ASDUs with time tag is permitted.

AC500 V2.4 IEC60870-5-104 Compatibility List

Process information in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<45> :=	Single command	C_SC_NA_1
X	<46> :=	Double command	C_DC_NA_1
	<47> :=	Regulating step command	C_RC_NA_1
X	<48> :=	Set point command, normalized value	C_SE_NA_1
	<49> :=	Set point command, scaled value	C_SE_NB_1
X	<50> :=	Set point command, short floating point value	C_SE_NC_1
	<51> :=	Bitstring of 32 bit	C_BO_NA_1
X	<58> :=	Single command with time tag CP56Time2a	C_SC_TA_1
X	<59> :=	Double command with time tag CP56Time2a	C_DC_TA_1
	<60> :=	Regulating step command with time tag CP56Time2a	C_RC_TA_1
X	<61> :=	Set point command, normalized value with time tag CP56Time2a	C_SE_TA_1
	<62> :=	Set point command, scaled value with time tag CP56Time2a	C_SE_TB_1
X	<63> :=	Set point command, short floating point value with time tag CP56Time2a	C_SE_TC_1
	<64> :=	Bitstring of 32 bit with time tag CP56Time2a	C_BO_TA_1

Either the ASDUs of the set <45> – <51> or of the set <58> – <64> are used.

System information in monitor direction
(station-specific parameter, mark with an "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<70> :=	End of initialization	M_EI_NA_1

System information in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<100>:=	Interrogation command	C_IC_NA_1
X	<101>:=	Counter interrogation command	C_CI_NA_1
X	<102>:=	Read command	C_RD_NA_1
X	<103>:=	Clock synchronization command (option see 7.6)	C_CS_NA_1
	<104>:=	~~Test command~~	~~C_TS_NA_1~~
X	<105>:=	Reset process command	C_RP_NA_1
	<106>:=	~~Delay acquisition command~~	~~C_CD_NA_1~~
X	<107>:=	Test command with time tag CP56Time2a	C_TS_TA_1

AC500 V2.4 IEC60870-5-104 Compatibility List

Parameter in control direction
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

X	<110>:= Parameter of measured value, normalized value	P_ME_NA_1
	<111>:= Parameter of measured value, scaled value	P_ME_NB_1
X	<112>:= Parameter of measured value, short floating point value	P_ME_NC_1
	<113>:= Parameter activation	P_AC_NA_1

File transfer
(station-specific parameter, mark each Type ID "**X**" if it is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

	<120>:= File ready	F_FR_NA_1
	<121>:= Section ready	F_SR_NA_1
	<122>:= Call directory, select file, call file, call section	F_SC_NA_1
	<123>:= Last section, last segment	F_LS_NA_1
	<124>:= Ack file, ack section	F_AF_NA_1
	<125>:= Segment	F_SG_NA_1
	<126>:= Directory {blank or X, only available in monitor (standard) direction}	F_DR_TA_1
	<127>:= Query Log – Request archive file	F_SC_NB_1

Type identifier and cause of transmission assignments
(station-specific parameters)

Shaded boxes: option not required.

Black boxes: option not permitted in this companion standard

Blank: functions or ASDU not used.

Mark Type Identification/Cause of transmission combinations:

"**X**" if only used in the standard direction;

"**R**" if only used in the reverse direction;

"**B**" if used in both directions.

Type identification		**Cause of transmission**																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<1>	M_SP_NA_1		X	X		X									X					
~~<2>~~	~~M_SP_TA_1~~																			
<3>	M_DP_NA_1		X	X		X									X					
~~<4>~~	~~M_DP_TA_1~~																			
<5>	M_ST_NA_1																			
~~<6>~~	~~M_ST_TA_1~~																			
<7>	M_BO_NA_1																			
~~<8>~~	~~M_BO_TA_1~~																			
<9>	M_ME_NA_1	X	X	X		X									X					
~~<10>~~	~~M_ME_TA_1~~																			
<11>	M_ME_NB_1																			
~~<12>~~	~~M_ME_TB_1~~																			

AC500 V2.4 IEC60870-5-104 Compatibility List

Type identification		Cause of transmission																		
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<13>	M_ME_NC_1	x	x	x		x									x					
~~<14>~~	~~M_ME_TC_1~~																			
<15>	M_IT_NA_1			x												x				
~~<16>~~	~~M_IT_TA_1~~																			
~~<17>~~	~~M_EP_TA_1~~																			
~~<18>~~	~~M_EP_TB_1~~																			
~~<19>~~	~~M_EP_TC_1~~																			
<20>	M_PS_NA_1																			
<21>	M_ME_ND_1																			
<30>	M_SP_TB_1			x		x														
<31>	M_DP_TB_1			x		x														
<32>	M_ST_TB_1																			
<33>	M_BO_TB_1																			
<34>	M_ME_TD_1			x		x														
<35>	M_ME_TE_1																			
<36>	M_ME_TF_1			x		x														
<37>	M_IT_TB_1			x												x				
<38>	M_EP_TD_1																			
<39>	M_EP_TE_1																			
<40>	M_EP_TF_1																			
<45>	C_SC_NA_1						x	x			x									
<46>	C_DC_NA_1						x	x			x									
<47>	C_RC_NA_1																			
<48>	C_SE_NA_1						x	x			x									
<49>	C_SE_NB_1																			
<50>	C_SE_NC_1						x	x			x									
<51>	C_BO_NA_1																			
<58>	C_SC_TA_1						x	x			x									
<59>	C_DC_TA_1						x	x			x									
<60>	C_RC_TA_1																			
<61>	C_SE_TA_1						x	x			x									
<62>	C_SE_TB_1																			
<63>	C_SE_TC_1						x	x			x									
<64>	C_BO_TA_1																			
<70>	M_EI_NA_1*				x															
<100>	C_IC_NA_1						x	x			x									
<101>	C_CI_NA_1						x	x			x									
<102>	C_RD_NA_1					x														
<103>	C_CS_NA_1						x	x												
~~<104>~~	~~C_TS_NA_1~~																			
<105>	C_RP_NA_1						x													
~~<106>~~	~~C_CD_NA_1~~																			
<107>	C_TS_TA_1																			
<110>	P_ME_NA_1						x	x							x					
<111>	P_ME_NB_1																			
<112>	P_ME_NC_1						x	x							x					
<113>	P_AC_NA_1																			
<120>	F_FR_NA_1																			
<121>	F_SR_NA_1																			
<122>	F_SC_NA_1																			
<123>	F_LS_NA_1																			
<124>	F_AF_NA_1																			
<125>	F_SG_NA_1																			
<126>	F_DR_TA_1*																			

AC500 V2.4 IEC60870-5-104 Compatibility List

Type identification		Cause of transmission																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	20 to 36	37 to 41	44	45	46	47
<127>	F_SC_NB_1*																			
* Blank or X only																				

AC500 V2.4 IEC60870-5-104 Compatibility List

9.6 Basic application functions

Station initialization
(station-specific parameter, mark "**X**" if function is used)

[X] Remote initialization

Cyclic data transmission
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Cyclic data transmission

Read procedure
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Read procedure

Spontaneous transmission
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions)

[X] Spontaneous transmission

Double transmission of information objects with cause of transmission spontaneous
(station-specific parameter, mark each information type "**X**" where both a Type ID without time and corresponding Type ID with time are issued in response to a single spontaneous change of a monitored object)

The following type identifications may be transmitted in succession caused by a single status change of an information object. The particular information object addresses for which double transmission is enabled are defined in a project-specific list.

- [ ] Single-point information M_SP_NA_1, M_SP_TA_1, M_SP_TB_1 and M_PS_NA_1
- [ ] Double-point information M_DP_NA_1, M_DP_TA_1 and M_DP_TB_1
- [ ] Step position information M_ST_NA_1, M_ST_TA_1 and M_ST_TB_1
- [ ] Bitstring of 32 bit M_BO_NA_1, M_BO_TA_1 and M_BO_TB_1 (if defined for a specific project)
- [ ] Measured value, normalized value M_ME_NA_1, M_ME_TA_1, M_ME_ND_1 and M_ME_TD_1
- [ ] Measured value, scaled value M_ME_NB_1, M_ME_TB_1 and M_ME_TE_1
- [ ] Measured value, short floating point number M_ME_NC_1, M_ME_TC_1 and M_ME_TF_1

AC500 V2.4 IEC60870-5-104 Compatibility List

Station interrogation
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] global
- [ ] group 1
- [ ] group 2
- [ ] group 3
- [ ] group 4
- [ ] group 5
- [ ] group 6
- [ ] group 7
- [ ] group 8
- [ ] group 9
- [ ] group 10
- [ ] group 11
- [ ] group 12
- [ ] group 13
- [ ] group 14
- [ ] group 15
- [ ] group 16

Information object addresses assigned to each group must be shown in a separate table.

Clock synchronization
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Clock synchronization
- [ ] Day of week used
- [ ] RES1, GEN (time tag substituted/ not substituted) used
- [ ] SU-bit (summertime) used

optional, see 7.6

Command transmission
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Direct command transmission
- [X] Direct set point command transmission
- [ ] Select and execute command
- [ ] Select and execute set point command
- [ ] C_SE ACTTERM used

- [ ] No additional definition
- [X] Short-pulse duration (duration determined by a system parameter in the outstation)
- [X] Long-pulse duration (duration determined by a system parameter in the outstation)
- [X] Persistent output

- [ ] Supervision of maximum delay in command direction of commands and set point commands
- [ ] Maximum allowable delay of commands and set point commands

AC500 V2.4 IEC60870-5-104 Compatibility List

Transmission of integrated totals
(station- or object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [ ] Mode A: Local freeze with spontaneous transmission
- [ ] Mode B: Local freeze with counter interrogation
- [ ] Mode C: Freeze and transmit by counter-interrogation commands
- [ ] Mode D: Freeze by counter-interrogation command, frozen values reported

- [X] Counter read
- [X] Counter freeze without reset
- [X] Counter freeze with reset
- [X] Counter reset

- [X] General request
- [X] Request counter group 1
- [X] Request counter group
- [X] Request counter group 3
- [X] Request counter group 4

Parameter loading
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Threshold value
- [ ] Smoothing factor
- [ ] Low limit for transmission of measured values
- [ ] High limit for transmission of measured values

Parameter activation
(object-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [ ] Act/deact of persistent cyclic or periodic transmission of the addressed object

Test procedure
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

- [X] Test procedure

AC500 V2.4 IEC60870-5-104 Compatibility List

File transfer
(station-specific parameter, mark "**X**" if function is used).

File transfer in monitor direction

☐ Transparent file
☐ Transmission of disturbance data of protection equipment
☐ Transmission of sequences of events
☐ Transmission of sequences of recorded analogue values

File transfer in control direction

☐ Transparent file

Background scan
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

☒ Background scan

Acquisition of transmission delay
(station-specific parameter, mark "**X**" if function is only used in the standard direction, "**R**" if only used in the reverse direction, and "**B**" if used in both directions).

■ ~~Acquisition of transmission delay~~

Definition of time outs

Parameter	Default value	Remarks	Selected value
t_0	30 s	Time-out of connection establishment	
t_1	15 s	Time-out of send or test APDUs	
t_2	10 s	Time-out for acknowledges in case of no data messages $t_2 < t_1$	
t_3	20 s	Time-out for sending test frames in case of a long idle state	

Maximum range for timeouts t_0 to t_2: 1 s to 255 s, accuracy 1 s.
Recommended range for timeout t_3: 1 s to 48 h, resolution 1 s.
Long timeouts for t_3 may be needed in special cases where satellite links or dialup connections are used (for instance to establish connection and collect values only once per day or week).

Maximum number of outstanding I format APDUs *k* and latest acknowledge APDUs (*w*)

Parameter	Default value	Remarks	Selected value
k	12 APDUs	Maximum difference receive sequence number to send state variable	
w	8 APDUs	Latest acknowledge after receiving *w* I format APDUs	

Maximum range of values *k*: 1 to 32767 (2^{15}–1) APDUs, accuracy 1 APDU

AC500 V2.4 IEC60870-5-104 Compatibility List

Maximum range of values *w*: 1 to 32767 APDUs, accuracy 1 APDU (Recommendation: *w* should not exceed two-thirds of *k*).

Portnumber

Parameter	Value	Remarks
Portnumber	2404	In all cases

Redundant connections

2 Number N of redundancy group connections used

RFC 2200 suite

RFC 2200 is an official Internet Standard which describes the state of standardization of protocols used in the Internet as determined by the Internet Architecture Board (IAB). It offers a broad spectrum of actual standards used in the Internet. The suitable selection of documents from RFC 2200 defined in this standard for given projects has to be chosen by the user of this standard.

☐ Ethernet 802.3

☐ Serial X.21 interface

☐ Other selection from RFC 2200:

List of valid documents from RFC 2200

1. .....
2. .....
3. .....
4. .....
5. .....
6. .....
7. etc.

1.6.5.3.3 Modbus protocol

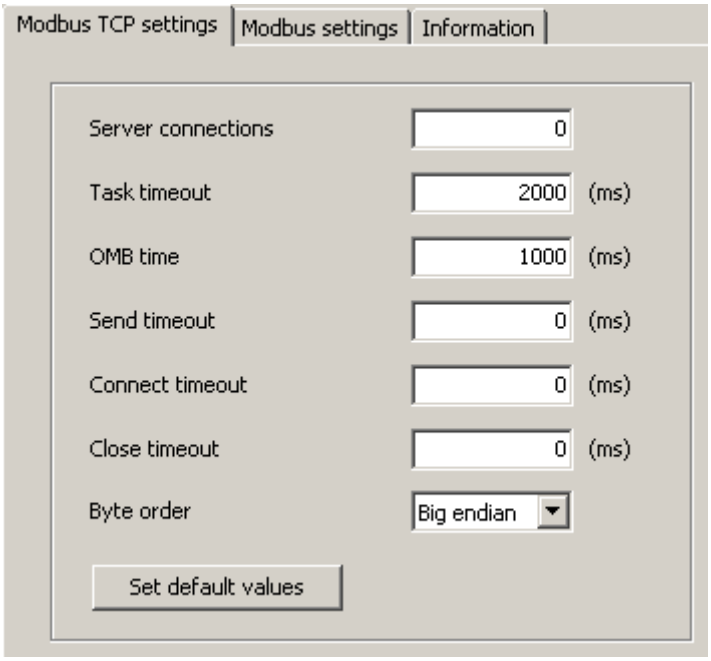
Modbus on TCP/IP protocol

Configuration of Modbus TCP/IP server settings

As of Automation Builder 1.1 (CBP 2.4.) Modbus configuration has been moved to another location within the device tree. Note: Principle of Modbus configuration remains unchanged.

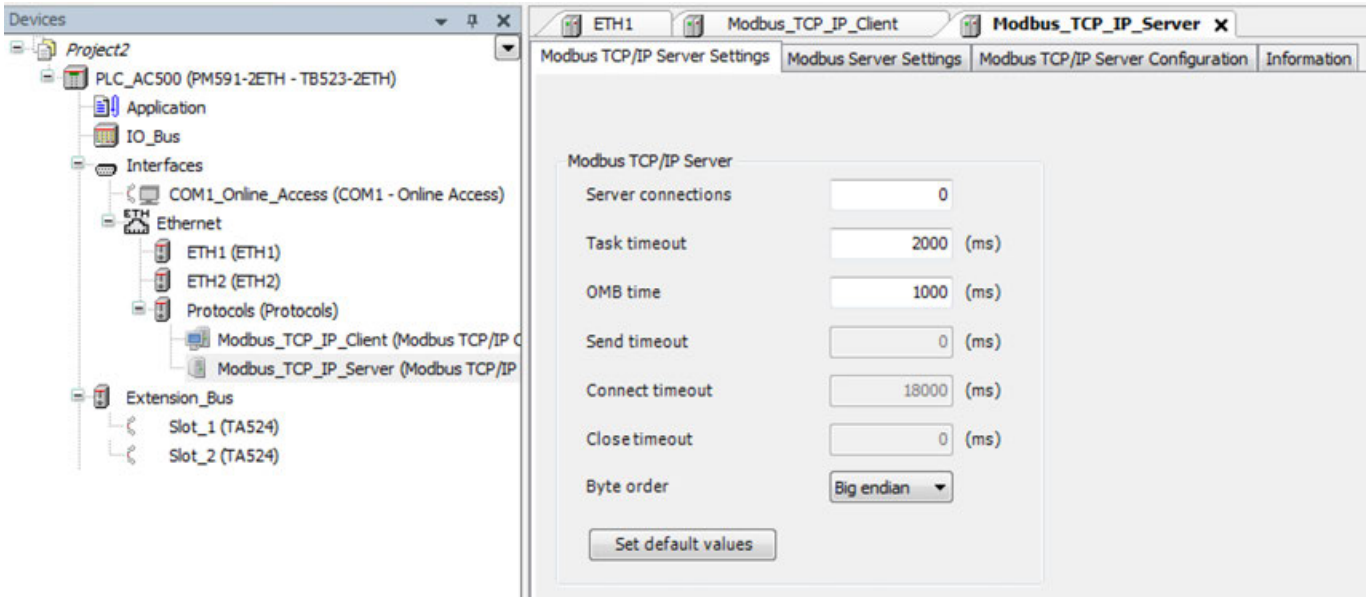
Configuration up to CBP 2.4

Double-click on *"MODBUS_on_TCP_IP (MODBUS on TCP/IP)"* to open the Modbus TCP settings in the editor window:



Configuration as of CBP 2.4

For Modbus TCP/IP protocol configuration add a new object for Modbus Server settings under *"Ethernet Interface ➔ Protocols"*. In the device tree double-click the new added item to open the configuration:



Parameters

Depending on the Automation Builder/CBP version some or all parameters are available:

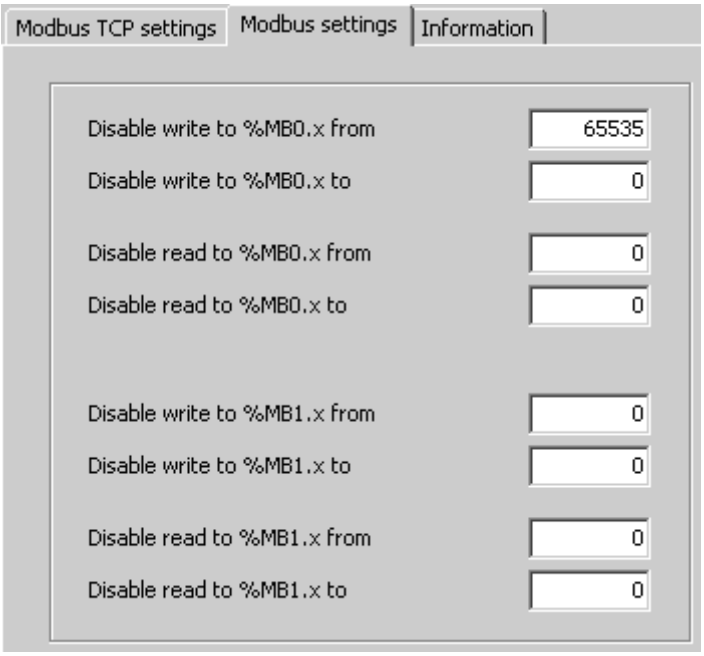
Parameter	Default	Value range	Description
Server connections	0	0...X (X depending on CPU)	Maximum number of logical server connections allowed in parallel. 0 means only client usage (Modbus master). Requests from other clients are ignored.
Task timeout	2000	100...6000000	Used for client mode only: A request is cancelled if no reply from the server is received during the time given in task timeout [in ms].
OMB time	1000	100...6000000	Used for server mode only: determining how long a logical connection in [ms] remains after receiving the reply message of a client. Establishing and terminating of connection takes some time. When the data communication between the controller and the servers should take place only at long intervals, it is useful to close the connection immediately after completion of the data communication.
Send timeout	0	0 ... 2000000000	Used for client mode only: definition, how long [in ms] the PLC tries to send a request to a server. CM579-ETH: 0 is default value 31000 ms Onboard Ethernet: not implemented
Connect timeout	0	0 ... 2000000000	Used for client mode only: definition, how long [in ms] the PLC tries to establish a TCP connection with a server. CM579-ETH: 0 is default value 31000 ms Onboard Ethernet: fixed value 18000 ms
Close timeout	0	0 ... 2000000000	Used for client mode only: definition, how long [in ms] the PLC tries to quit a TCP connection with a server. CM579-ETH: 0 is default value 31000 ms Onboard Ethernet: not implemented
Byte order	Big endian	Little endian Big endian	Valid for client and server mode. Format for the transmission of WORD values (register) within the telegram.
Set default values	-	-	Restoring the default values of all parameters.

Configuration of Modbus server settings

As of AB 1.1 (CBP 2.4.) Modbus configuration has been moved to another location within the device tree. Note: Principle of Modbus configuration remains unchanged.

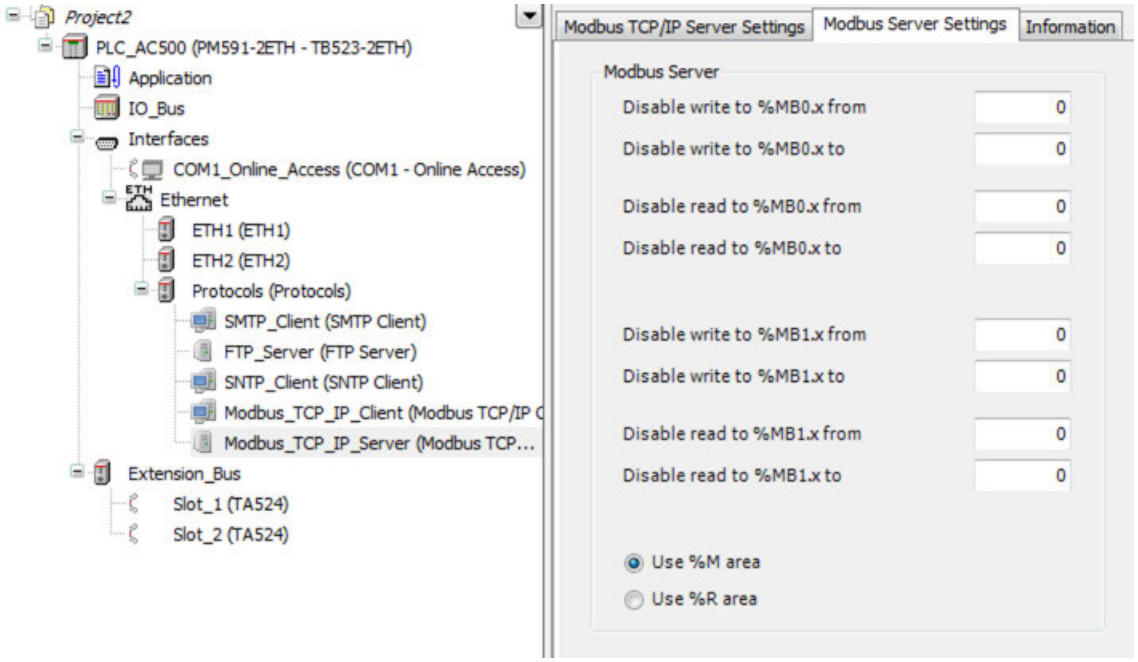
Up to CBP 2.4

Double-click on *"MODBUS_on_TCP_IP (MODBUS on TCP/IP)"* and select the tab *"Modbus settings"* to show the available Modbus settings in the editor window:



As of CBP 2.4

For Modbus TCP/IP protocol configuration add a new object for Modbus Server settings under Ethernet Interface. In the device tree double-click the new added item to open the configuration:



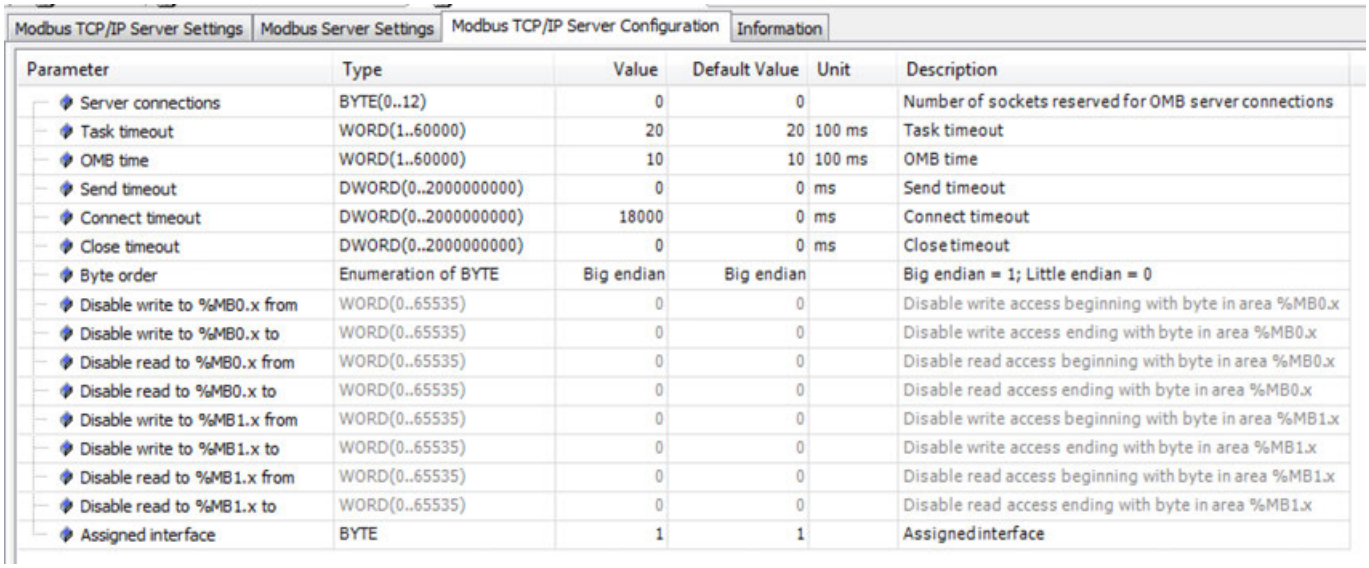
The following parameters are available:

Parameter	Default	Value	Description	Generic see remark 1
Disable write to %MB0.x from	0	0...65535	Disable write access for segment 0 starting at %MB0.x	Disable write to %MB0.x from
Disable write to %MB0.x to	0	0...65535	Disable write access for segment 0 up to %MB0.x	Disable write to %MB0.x to
Disable read to %MB0.x from	0	0...65535	Disable read access for segment 0 starting at %MB0.x	Disable read to %MB0.x from
Disable read to %MB0.x to	0	0...65535	Disable read access for segment 0 up to %MB0.x	Disable read to %MB0.x to
Disable write to %MB1.x from	0	0...65535	Disable write access for segment 1 starting at %MB1.x	Disable write to %MB1.x from
Disable write to %MB1.x to	0	0...65535	Disable write access for segment 1 up to %MB1.x	Disable write to %MB1.x to
Disable read to %MB1.x from	0	0...65535	Disable read access for segment 1 starting at %MB1.x	Disable read to %MB1.x from
Disable read to %MB1.x to	0	0...65535	Disable write access for segment 1 up at %MB1.x	Disable read to %MB1.x to

Like for Modbus RTU on the serial interfaces COMx, it is also possible to disable read and/or write access to individual segments for slave operation. Reading/writing is disabled beginning at the set address and is valid up to the set end address (inclusive).

Remark 1: Generic device configuration view parameters

Tab IP Settings Configuration or Modbus TCP/IP Server Configuration shows a list of all available parameters which is only visible if the parameter Show generic device configuration views is activated (open Options dialog window with menu item Tools -> Options, parameter is located under section Device editor):

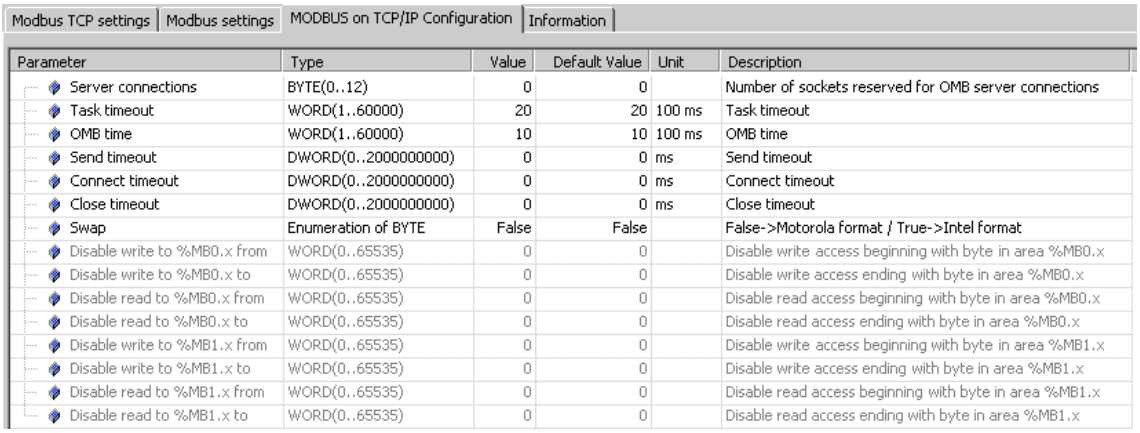


Configuration of Modbus TCP/IP configuration

As of Automation Builder 1.1 (CBP 2.4.) Modbus configuration has been moved to another location within the device tree. Note: Principle of Modbus configuration remains unchanged.

Configuration up to CBP 2.4

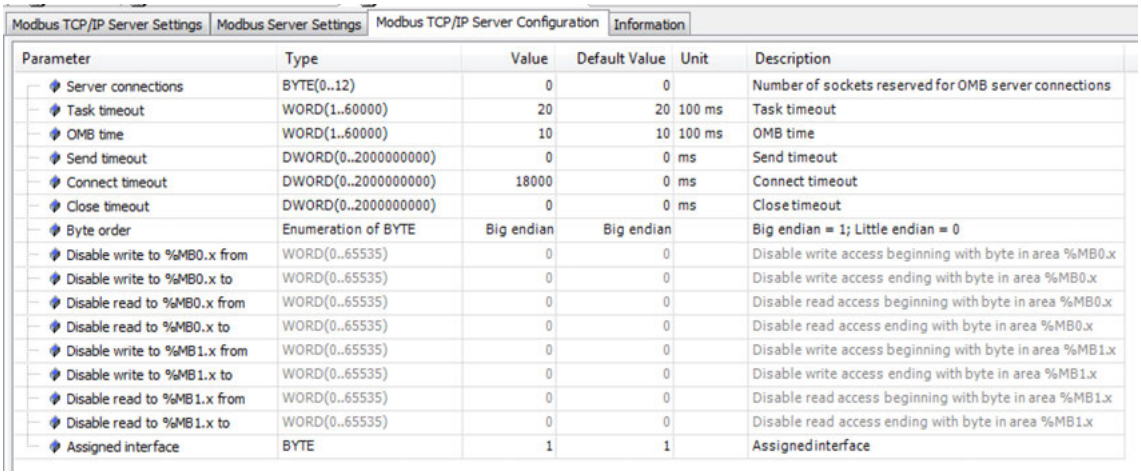
Double-click on *"MODBUS_on_TCP_IP (MODBUS on TCP/IP)"* and select the tab *"MODBUS on TCP/IP Configuration"* to show the available Modbus settings in the editor window:



Parameter	Type	Value	Default Value	Unit	Description
Server connections	BYTE(0..12)	0	0		Number of sockets reserved for OMB server connections
Task timeout	WORD(1..60000)	20	20	100 ms	Task timeout
OMB time	WORD(1..60000)	10	10	100 ms	OMB time
Send timeout	DWORD(0..2000000000)	0	0	ms	Send timeout
Connect timeout	DWORD(0..2000000000)	0	0	ms	Connect timeout
Close timeout	DWORD(0..2000000000)	0	0	ms	Close timeout
Swap	Enumeration of BYTE	False	False		False->Motorola format / True->Intel format
Disable write to %MB0.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB0.x
Disable write to %MB0.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB0.x
Disable read to %MB0.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB0.x
Disable read to %MB0.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB0.x
Disable write to %MB1.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB1.x
Disable write to %MB1.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB1.x
Disable read to %MB1.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB1.x
Disable read to %MB1.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB1.x

Configuration as of CBP 2.4

For Modbus TCP/IP protocol configuration double-click the Modbus item in the device tree and select the tab Modbus TCP/IP Server Configuration:



Parameter	Type	Value	Default Value	Unit	Description
Server connections	BYTE(0..12)	0	0		Number of sockets reserved for OMB server connections
Task timeout	WORD(1..60000)	20	20	100 ms	Task timeout
OMB time	WORD(1..60000)	10	10	100 ms	OMB time
Send timeout	DWORD(0..2000000000)	0	0	ms	Send timeout
Connect timeout	DWORD(0..2000000000)	18000	0	ms	Connect timeout
Close timeout	DWORD(0..2000000000)	0	0	ms	Close timeout
Byte order	Enumeration of BYTE	Big endian	Big endian		Big endian = 1; Little endian = 0
Disable write to %MB0.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB0.x
Disable write to %MB0.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB0.x
Disable read to %MB0.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB0.x
Disable read to %MB0.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB0.x
Disable write to %MB1.x from	WORD(0..65535)	0	0		Disable write access beginning with byte in area %MB1.x
Disable write to %MB1.x to	WORD(0..65535)	0	0		Disable write access ending with byte in area %MB1.x
Disable read to %MB1.x from	WORD(0..65535)	0	0		Disable read access beginning with byte in area %MB1.x
Disable read to %MB1.x to	WORD(0..65535)	0	0		Disable read access ending with byte in area %MB1.x
Assigned interface	BYTE	1	1		Assigned interface

Parameters

Depending on the Automation Builder/CBP version some or all parameters are available:

Parameter	Default	Value	Description
Server connections	0	0...X (X depending on PLC)	Maximum number of logical parallel connections, that are kept for connection requests by clients in operation mode as server. The value 0 means pure client mode (Modbus master) (0..X, depending on the performance of the PLC)
Task timeout	2000	100 ... 6000000	Only valid for client mode. A request is canceled when no response by the server is received within this time in [ms]. (100..6.000,000 ms)

Parameter	Default	Value	Description
OMB time	1000	100 ... 6000000	Only valid for server mode. Determining how long a logical connection in [ms] remains after receiving the reply message of a client. Establishing and terminating of connection takes some time. When the data communication between the controller and the servers should take place only at long intervals, it is useful to close the connection immediately after completion of the data communication. (100..6.000,000 ms)
Send timeout	0	0 ... 2000000000	Only valid for client mode. Determining how long the controller should try to send a request to a server. (0..2.000,000,000 ms)
Connect timeout	0	0 ... 2000000000	Only valid for client mode. Determining how long the controller should try to create a TCP connection with a server. (0..2.000,000,000 ms)
Close timeout	0	0 ... 2000000000	Only valid for client mode. Determining how long the controller should try to close a TCP connection with a server. (0..2.000,000,000 ms)
Byte order	Big endian	Little endian Big endian	Valid for client and server mode. Format of the transfer of word values (registers) in the telegram. Depending on the settings, the byte order will be changed by sending and receiving. Changing the byte order may be required e.g. when devices with different processor types are used.

Modbus on RTU protocol

Protocol description can be found in the chapter for Serial interfaces ↳ *Chapter 1.6.5.2.11.4 “Setting COMx - Modbus” on page 6108*.

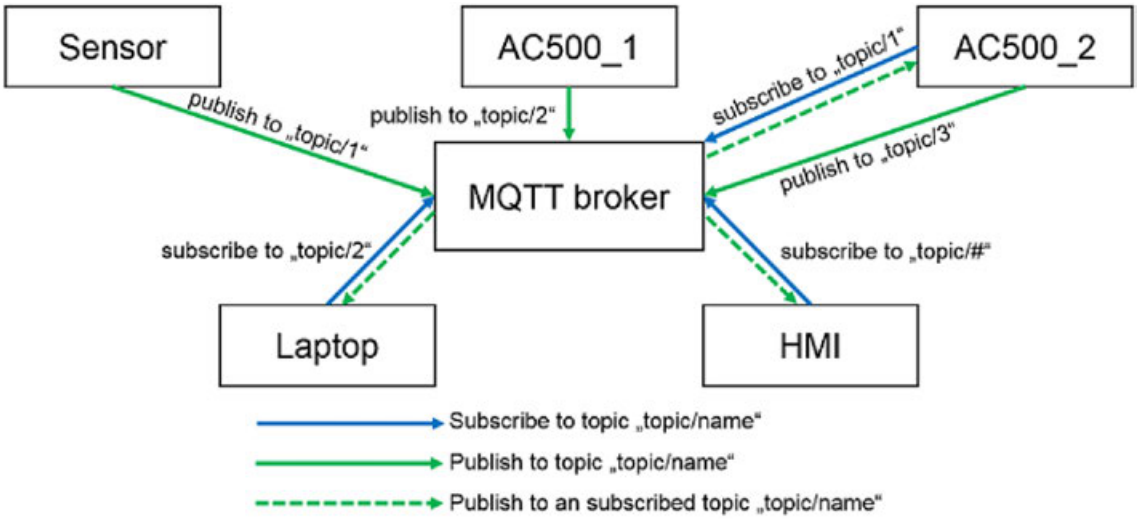
1.6.5.3.4 MQTT client protocol

System technology

The MQTT protocol is a lightweight communication protocol which is widely used on the internet to connect embedded device to the cloud.

The MQTT (Message Queuing Telemetry Transport) client library allows to integrate an AC500 processor module to act as a client in the MQTT protocol. Thus, it is possible to exchange data between the AC500 and other devices connected to the MQTT network.

In the figure below, there is an MQTT network with one broker (MQTT broker in the middle) and five clients. The figure shows the main functions of MQTT to send and receive data: publish and subscribe. The clients can publish messages with a specific topic to send data (e. g. the temperature of a connected sensor with a timestamp) to the MQTT broker. For example, the client “AC500_1” publishes a message to topic “topic/2”. On the other hand side clients can also subscribe to topics to receive data. For example, the client “Laptop” has subscribed topic “topic/2”. So all messages with the topic “topic/2” which has been published to the MQTT broker will be sent immediately to the client “Laptop”. This creates a message flow from the client "AC500_1" to the laptop.

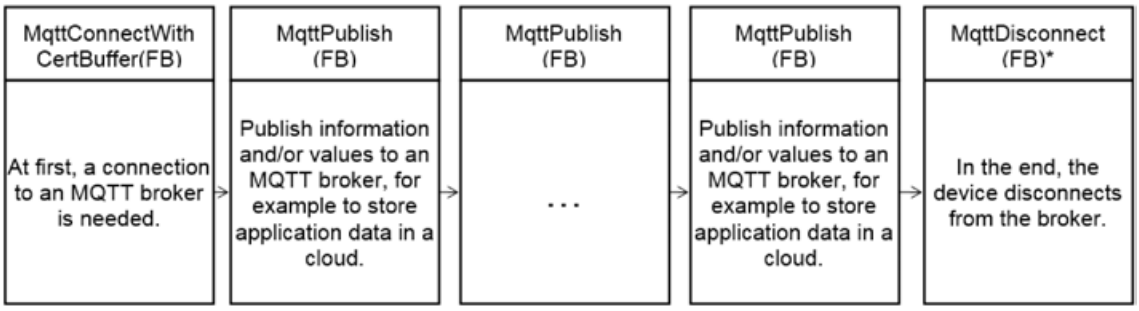


To realize the MQTT behavior, there are several function blocks implemented in the *Chapter 1.5.4.24 “MQTT client library” on page 1710*.

Table 753: Function blocks overview

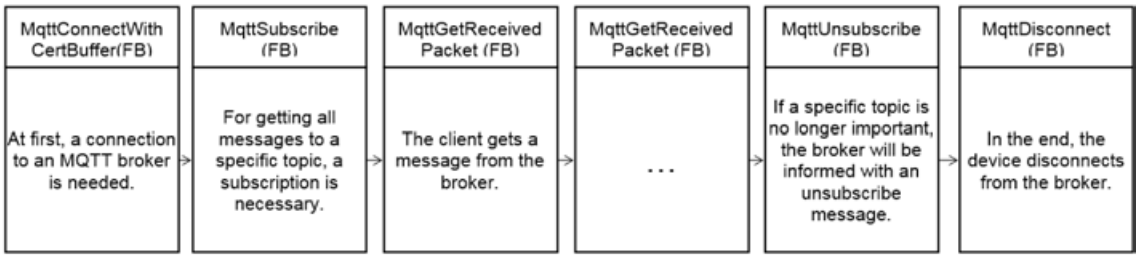
Function Block	Description
MqttConnectWithCertBuffer MqttConnectWithCertFile	Every MQTT use case starts with establishing a connection to an MQTT broker. Therefore, a connection structure needs to be created. The connection structure is used to identify the connection for subsequent operations like publish or subscribe. It is possible to establish an SSL connection. Using an SSL connection, at least a certificate for the server is needed. Certificates can be loaded from a buffer (program variable) or a file which is stored on the PLC.
MqttGetReceivedPacket MqttPing MqttPublish MqttSubscribe MqttUnsubscribe	These function blocks can be used on an established MQTT connection to realize the desired use case.
MqttDisconnect	This function block is the end of each use case.

One MQTT send use case could look like this:



*) It makes sense for several publish messages in a row (e. g. one message per second) not always open a new connection.

One MQTT receive use case could look like this:



TLS version

The MQTT client uses the TLS version 1.2.

Configuration in Automation Builder

For the MQTT client no configuration is needed.

Configuration in CODESYS

All function blocks have to be called in tasks with cyclically processing.

You can use the function blocks with:

- PLC_PRG with automatic task configuration or manual task configuration.
- One single program or different programs.
- One single task or different tasks.

With different programs assigned to different tasks you can define different cycle times and priorities.

Limitations

- No persistent session. After an interrupted connection, the client needs to subscribe on topics again in case of reconnect.
- One connection (MQTT_CONNECTION) cannot be shared between multiple tasks. Different connections can be used by different tasks or even within the same task.
- Only one FB can operate on a single connection at the same time. Always wait for the FB to complete before calling the next FB. To use two different FB's in parallel (like publish and receive) it is necessary to have two different connections, otherwise they must be called one after the other.

Hardware

The MQTT protocol requires AC500 devices with integrated Ethernet.

Examples

Example projects for the libraries can be found in the folder: \Users\Public\Documents\AutomationBuilder\Examples.

MQTT can be used using the MQTT client library or JSON. An introduction to programming with JSON is given in the *application example*.

1.6.5.3.5 SMTP protocol

Introduction of the SMTP protocol

> *SMTP Protocol is only available on the AC500 CPUs with onboard Ethernet.*

SMTP protocol allows a PLC to instruct a mail server in the network to send emails of user specified content to email accounts the SMTP server can reach (directly or via forwarding). The SMTP protocol is accompanied by the function block ETHx_SMTP_EMAIL_SEND within the AC500 Ethernet library. This function block replaces the previous ETH_SMTP_EMAIL_SEND function block of former CBP versions. Note: Input semantics have been changed.

Both function blocks allow users to transfer email content and data to the mail server via SMTP protocol implementation.

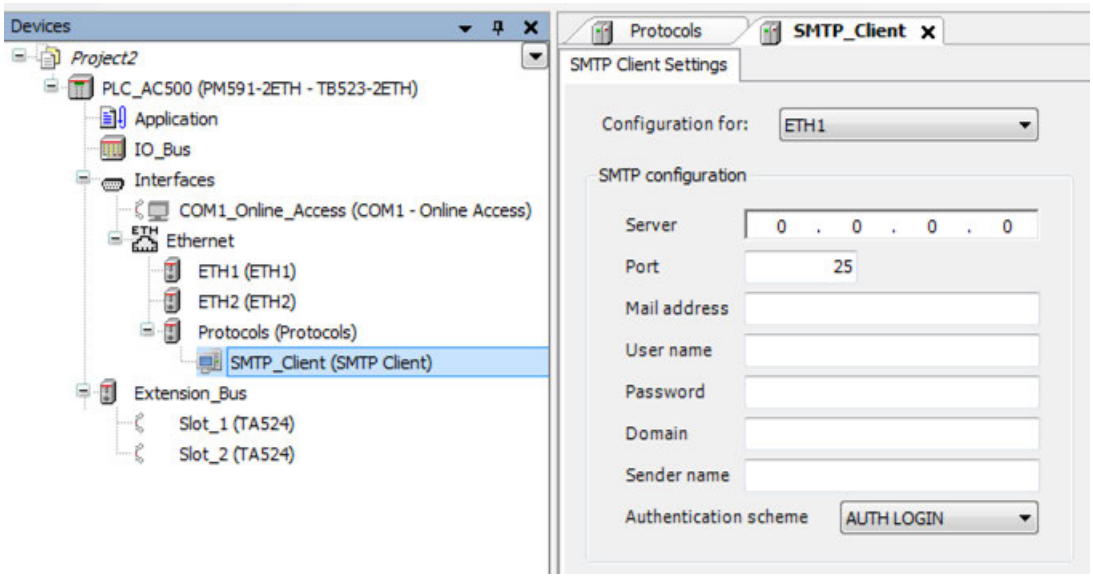
Please consider the following restrictions and constraints in the current AB Version:

- SMTP implementation only covers client functionality (AC500 can not receive emails) *Chapter 1.6.5.3.5.2 “Configuration of the SMTP protocol (>= CBP 2.4)” on page 6181*.
- Number of text lines in the email body is restricted to 20 lines with 255 characters each.
- Number of files attached to an email is restricted to 10 files with a file name of 255 chars and a path of 255 chars.
- SMTP implementation does not support secure connections to the SMTP server (like TLS or SSL).

Configuration of the SMTP protocol (>= CBP 2.4)

> *SMTP Protocol is only available on the AC500 CPUs with onboard Ethernet.*

For SMTP protocol configuration add a new object under Ethernet Interface. In the device tree double-click the new added item to open SMTP configuration:



The following parameters are available:

Parameter	Default	Value	Description
Configuration for SMTP configuration			
Server	0.0.0.0	-	Enter the IP address of the target SMTP server.
Port	25	-	Enter the SMTP port of the target SMTP server.
Mail address	Empty	-	Enter a valid email address.
User name / Password	Empty	-	Enter the user credentials for authentication at the SMTP server.
Domain	Empty	-	Domain of the target SMTP server. If an email address was entered, the data is set automatically.
Sender name	Empty	-	Name that will be used for SMTP mails. If an email address was entered, this address will be set by default.
Authentication scheme	AUTH LOGIN	AUTH LOGIN	With this authentication method, the users' password used for authentication is transferred in plain text to the target SMTP server.
		AUTH CRAM-MD5:	With this authentication method, the users' password used for authentication is transferred via CRAM-MD5 technology (Challenge-Response Authentication Mechanism, Message Digest 5) to the SMTP server.

1.6.5.3.6 SNTP protocol

Introduction of the SNTP protocol

- *SNTP Protocol is only available for AC500 CPUs with onboard Ethernet.*
- *If a high precision of system time is wanted, use a fully functional NTP server or at least a SNTP server with a high-precision time-source (e.g. DCF-77 receiver). Avoid cascading several levels of SNTP server / SNTP clients.*
- *Client requests are normally sent at intervals depending on the frequency tolerance of the client clock and the required accuracy. However, under no conditions requests should be sent at less than one minute intervals (see RFC 4330). Keep that in mind when setting polling-interval of the SNTP client, especially if a huge amount of clients use one single server.*
- *Be sure not to use broadcast or multicast addresses as server or backup-server since current SNTP implementation does not support manycast mode.*
- *If the AC500 PLC acts as SNTP server but is itself not synchronized with an external clock, e.g. a DCF77 device or the internet, the response telegram from the server back to the client will have the following flags:*
  - *Peer Clock Stratum: 15*
  - *Leap Indicator: unknown (clock unsychronized)*

  This might lead to negative effects on or with other devices that have contrary clock settings, e.g. a Peer Clock Stratum < 15. To avoid this, use the function block "RTC_SYNC_DISPLAY" (RTC_AC500_V20.lib) with the following settings:
  - *Peer Clock Stratum: primary reference (1)*
  - *Leap Indicator: no warning*

As of Automation Builder 1.1 SNTP protocol configuration has been moved to another location within the device tree. In former Versions SNTP protocol configuration has been configured under so-called *extended settings*.

As of and including Automation Builder 1.1 SNTP parameters for SNTP client and SNTP server are configured under the *Protocols* item in the device tree.

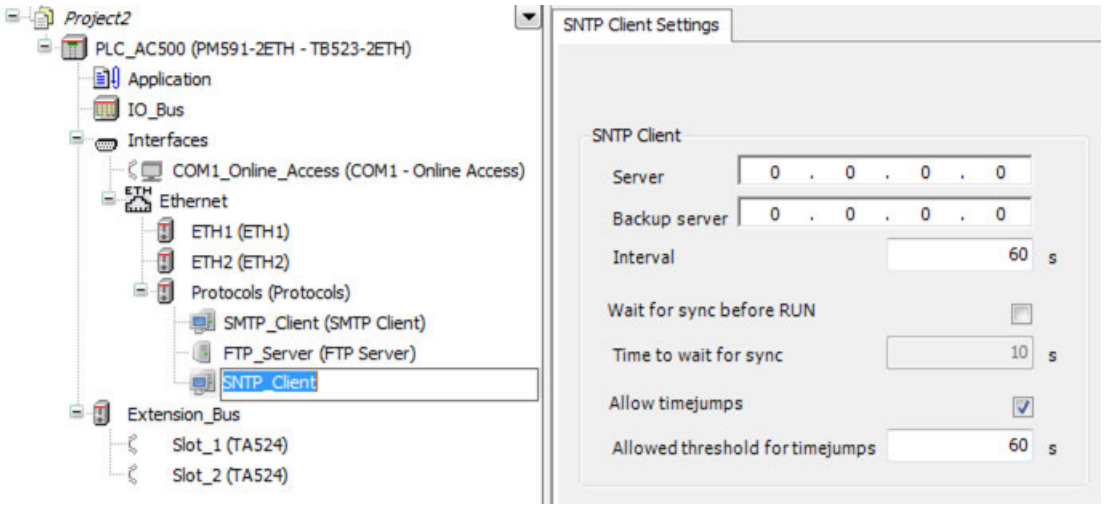
For further information on SNTP protocol configuration, see *Chapter 1.6.5.3.6.2 “Configuration of the SNTP protocol ” on page 6183*.

Configuration of the SNTP protocol

As of Automation Builder 1.1 SNTP protocol configuration has been moved to another location within the device tree. Note: Principle of SNTP configuration remains unchanged.

SNTP client configuration

For SNTP client configuration (Automation Builder 1.1) add a new object *“SNTP Client”* under *“Ethernet Interface ➔ Protocols”*:

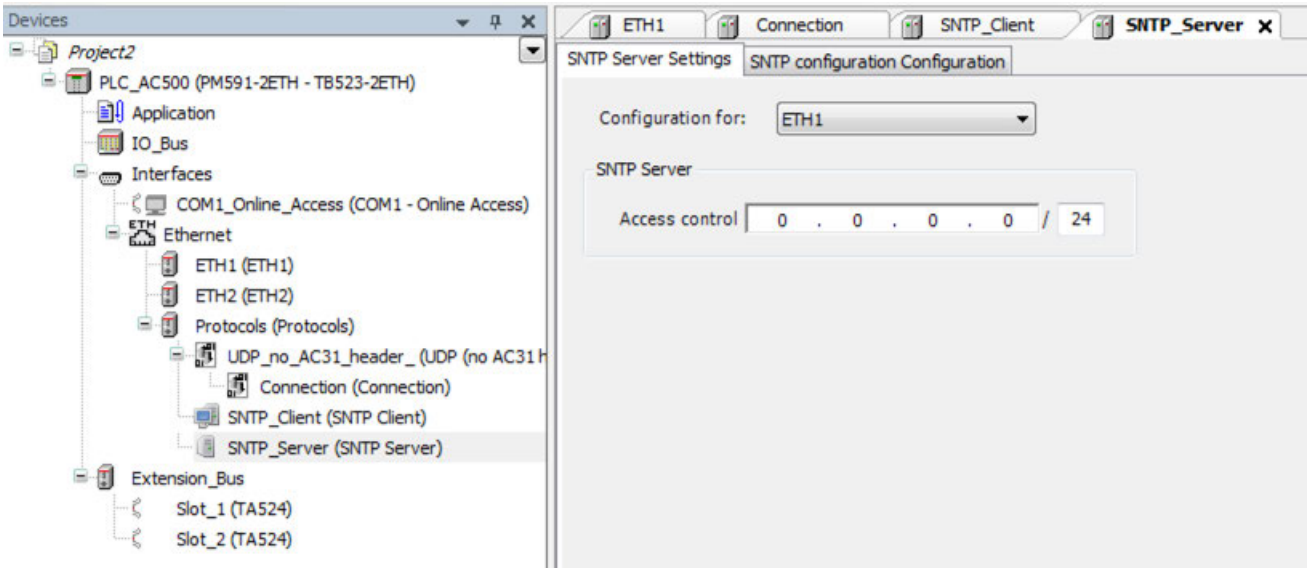


The following parameters are available:

Parameter	Default	Value	Description
SNTP Client			
Server	0.0.0.0	Valid IP address	IP address of the SNTP server.
Backup server	0.0.0.0	Valid IP address	IP address of the SNTP server which will be used if the primary server is unreachable.
Interval	60	15 ... 65535	Sets the interval in [s] for outgoing SNTP requests.
Wait for sync before RUN		Disabled	CPU does not wait for time synchronization from SNTP server and turns to RUN state immediately.
		Enabled	CPU waits for time synchronization from SNTP server and remains in Stop state after the time is over.
Time to wait for sync	10	1 ... 120	The time the CPU waits for receiving the time synchronization from SNTP server. If the CPU does not receive a valid time within the given period, an EC4 error appears.
Allow timejumps	Enabled	Disabled	Timejumps lead to an EC4 error in the CPU.
		Enabled	Timejumps are accepted. Parameter 'Allowed threshold for timejumps' can be adapted to determine the threshold for the timejumps. If the needed timejump exceeds the threshold, the CPU refuses the setting of time received from SNTP server and an EC4 error appears.
Allowed threshold for timejumps	60	1 ... 43200	Represents the threshold which is allowed for timejumps in [s].

SNTP server configuration

For SNTP server configuration (Automation Builder 1.1) add a new object *"SNTP Server"* under *"Ethernet Interface ➔ Protocols"*:



The following parameters are available:

Parameter	Default	Value	Description
Access control	0.0.0.0/24	Valid IP address	Sets the subnet from which incoming SNTP requests are accepted. Requests from other subnets will be ignored. Value 0.0.0.0/24 means IP address/ subnet mask. The subnet mask 24 means 255.255.255.0 The subnet mask 16 means 255.255.0.0

1.6.5.3.7 UDP protocol

Contents of the UDP protocol configuration

The available Ethernet devices support 2 types of UDP data exchange:

- UDP (no AC31 header): Up to 12 free UDP connections can be configured.
- UDP data exchange: up to 1 AC31 compatible UDP connection can be configured.

For further information on FTP server, see the following chapters:

Using UDP (No AC31 Header) *Chapter 1.6.5.3.7.2 "Using UDP (No AC31 header)" on page 6185*

Using UDP Data Exchange *Chapter 1.6.5.3.7.3 "Using UDP data exchange" on page 6187*

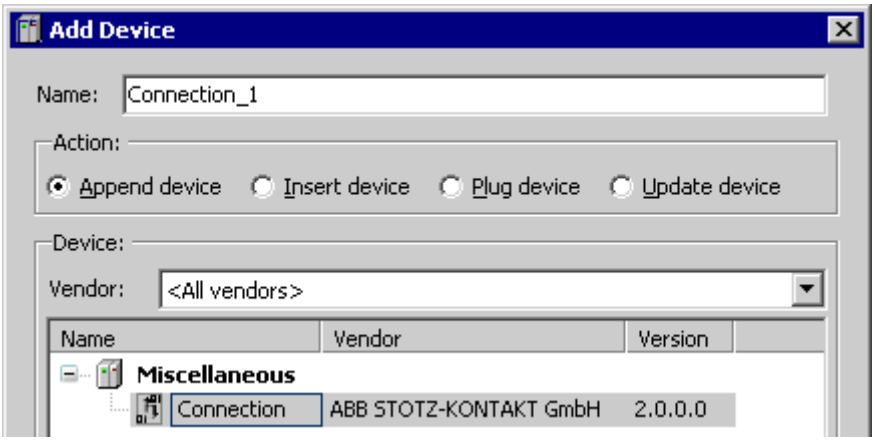
Using UDP (No AC31 header)

The UDP (no AC31 header) data exchange is a free UDP protocol that can be added to the AC500 Ethernet Communication Module. Up to 12 connections can be configured to each Ethernet device.

As of AB 1.1 (CBP 2.4.) UDP configuration has been moved to another location within the device tree. Note: Principle of UDP configuration remains unchanged.

Up to CBP 2.4

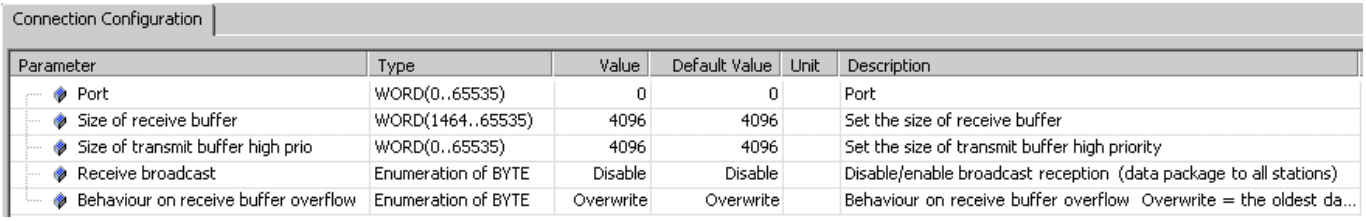
Right-click on the Communication Module and select Add device to open the Add Device dialog where the UDP data exchange module is listed:



Double-click on Connection (Connection) to open the Connection Configuration in editor window.

As of CBP 2.4

For UDP (no AC31 header) configuration add a new object under *“Ethernet Interface ➔ Protocols”*. In the device tree double-click the new added item to open the configuration:



The following parameters are available:

Parameter	Default	Value	Description
Port	0	0...65535	Port that is used for the UDP connection. See note below for further information.
Size of receive buffer	4096	1464...65535	Size of the receive buffer in [Bytes]. The behaviour in case of a full buffer can be determined with parameter Behaviour on receive buffer overflow.
Size of transmit buffer	4096	0...65535	Size of the transmit buffer in [Bytes].
Receive broadcast	Disable	Disable	Reception of broadcast packages is disabled.

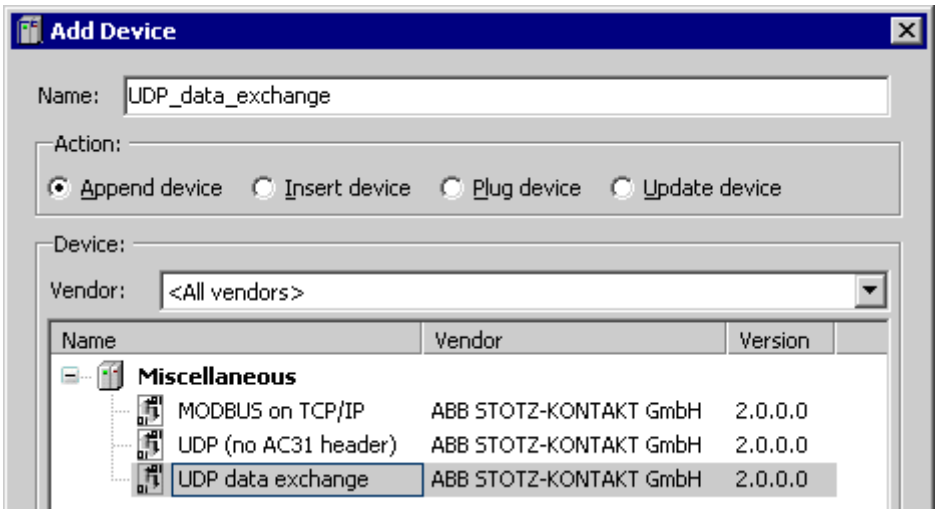
Parameter	Default	Value	Description
		Enable	Reception of broadcast packages is enabled. Broadcast packages will not be ignored by the controller.
Behaviour on receive buffer overflow	Overwrite	Overwrite	Incoming packages will overwrite the oldest packages in case of a full receive buffer.
		Reject	Incoming packages will be rejected in case of a full receive buffer.

The following ports are reserved and will cause a configuration error if selected: 1200, 1201, 123, 80, 24576, 25383.

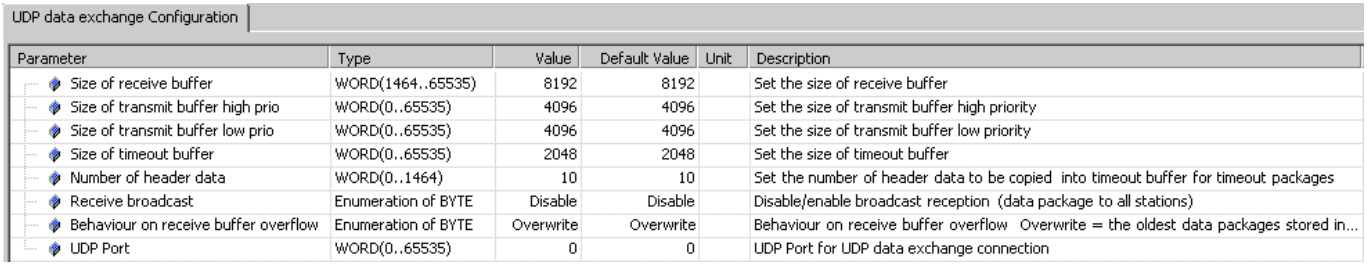
Using UDP data exchange

The UDP data exchange can be used to realize communication with AC31 devices and is an optional module that can be added to the AC500 Ethernet communication module.

Right-click on the communication module and select Add device to open the *Add Device* dialog where the UDP data exchange module is listed:



Double-click on *UDP_data_exchange* (UDP data exchange) to open the UDP data exchange Configuration in the editor window:



The following parameters are available:

Parameter	Default	Value	Description
Size of receive buffer	8192	1464...65535	Size of the receive buffer in [Bytes]. The behavior in case of a full buffer can be determined with parameter Behaviour on receive buffer overflow.
Size of transmit buffer high prio	4096	0...65535	Size of the high priority transmit buffer in [Bytes].
Size of transmit buffer low prio	4096	0...65535	Size of the low priority transmit buffer in [Bytes].
Size of timeout buffer	2048	0...65535	Size of the timeout buffer in [Bytes].
Number of header data	10	0...1464	Number of bytes that are copied into the timeout buffer for timeout packages.
Receive broadcast See note[1)]	Disable	Disable	Reception of broadcast packages is disabled.
		Enable	Reception of broadcast packages is enabled.
Behaviour on receive buffer overflow	Overwrite	Overwrite	Incoming packages will overwrite the oldest packages in case of a full receive buffer.
		Reject	Incoming packages will be rejected in case of a full receive buffer.
UDP Port See note[1)]	0	0...65535	Port used for the UDP connection.

Note[1)]

The following ports are reserved and will cause a configuration error if selected: 1200, 1201, 123, 80, 24576, 25383.

At this operating mode, the function blocks have no input for setting a port, so the port has to be identically at both sides. Otherwise, a communication is not possible.

UDP broadcast is only working properly with broadcast to the address 255.255.255.255.

1.6.5.3.8 FTP server

Preconditions for the use of the FTP server

- The FTP server is available on all AC500 PLCs with onboard Ethernet (i.e. AC500 CPU with Ethernet) and firmware version 2.1.0 or higher.
- To start the FTP server it has to be activated in the Automation Builder ↳ *Chapter 1.6.5.3.8.2 “Configuration of FTP server (< CBP 2.4)” on page 6189*.
- After downloading a project with an activated FTP server, you can connect to the PLC with a standard FTP client ↳ *Chapter 1.6.5.3.8.4 “Connection to a PLC running a FTP server” on page 6192* (e.g. "FileZilla").
- You can log in with different usernames giving access to different drives (for further details please see AC500_CPU_Memory_Locations.doc of the PLC):

Username	Accessible Drives	Comment
userdisk	user ramdisk "userdisk"	Size depends on product
ramdisk	system ramdisk "ramdisk" The system ramdisk should only be used for Firmware updates	Size depends on product
sdcard	SD card "sdcard"	Availability and size depends on inserted memory card
flashdisk	flashdisk "flashdisk"	PM592-ETH only
sramdisk	sramdisk "sramdisk"	Size depends on product
root	All available drives	-

Constraints for the usage of the FTP server are:

- User passwords are limited to 10 characters (no whitespaces).
- Directory and file names have to be in standard DOS 8.3 notation (e.g.: "abcdefgh.123" for a file or "12345678" for a directory) without \ / : * ? " < > |.
- Directory and file names may not contain whitespaces and are not case-sensitive.
- Maximum total path length: 255 characters.
- Maximum number of files in the root directory is limited (depends on memory location).

Configuration of FTP server (< CBP 2.4)

1. Open/Create a project for a PLC with onboard Ethernet (i.e. AC500 CPU with Ethernet).
2. Configure the onboard Ethernet.
3. Select: *“Node "IP_Settings"* ➔ *Tab "Extended settings"* ➔ *checkbox "FTP-Server active""*.
4. Configure the FTP server:

   "Port"

   --> TCP/IP-Port used to connect to the FTP-Server on the PLC.

   "Sessions"

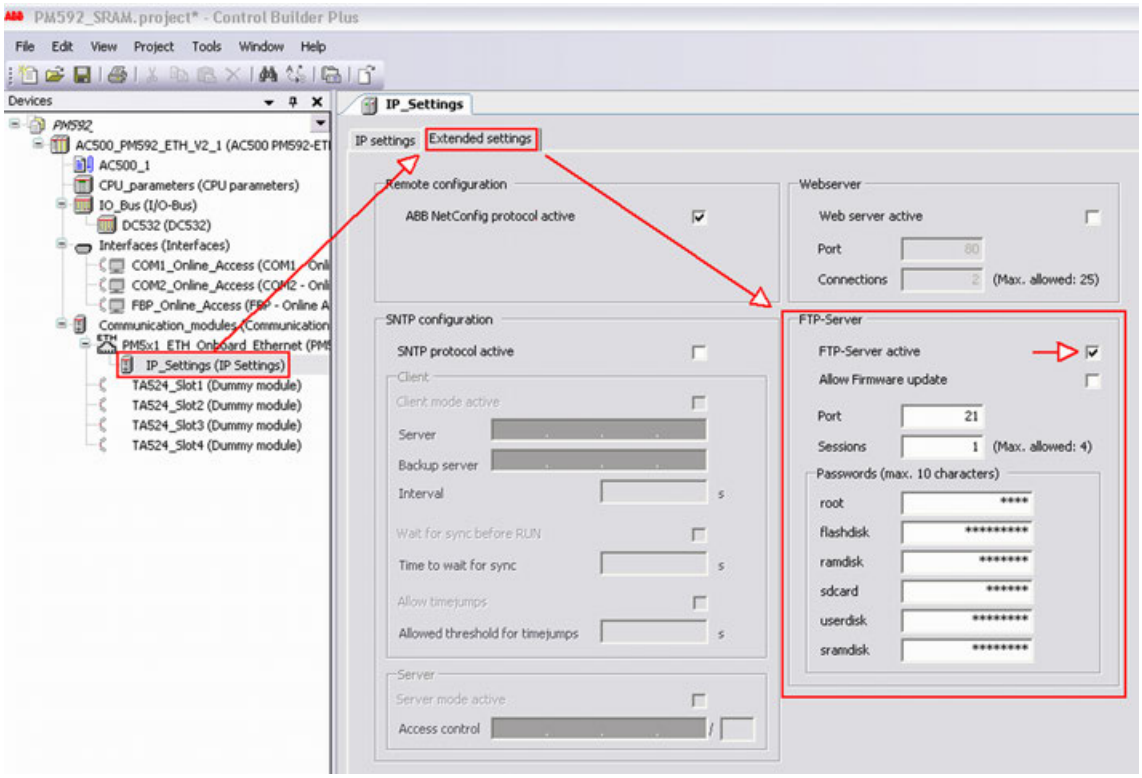
   --> maximum number of allowed simultaneous, parallel connections to the FTP-server

   each session uses one socket

   some FTP clients require several connections to work

   "Passwords"

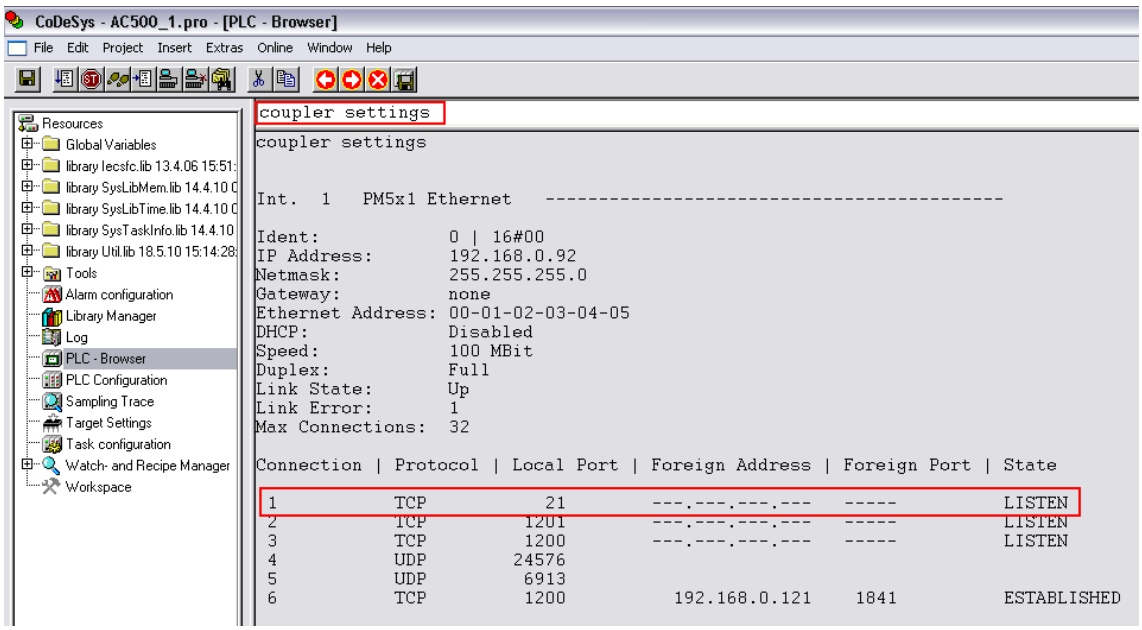
   --> set each user’s passwords for login (no entry means = no password).
5. Create configuration & download project to PLC.



If the checkbox "Allow Firmware update" is activated, you can download a PLC firmware to the RAM disk and the PLC will automatically update the device with the downloaded firmware. Afterwards a restart of the PLC is required. For further details please see FW_UPDATE_DOCUMENTATION.

> *The system RAM disk should only be used for Firmware updates.*

If the TCP protocol listens on the configured FTP port, the server is ready for login.

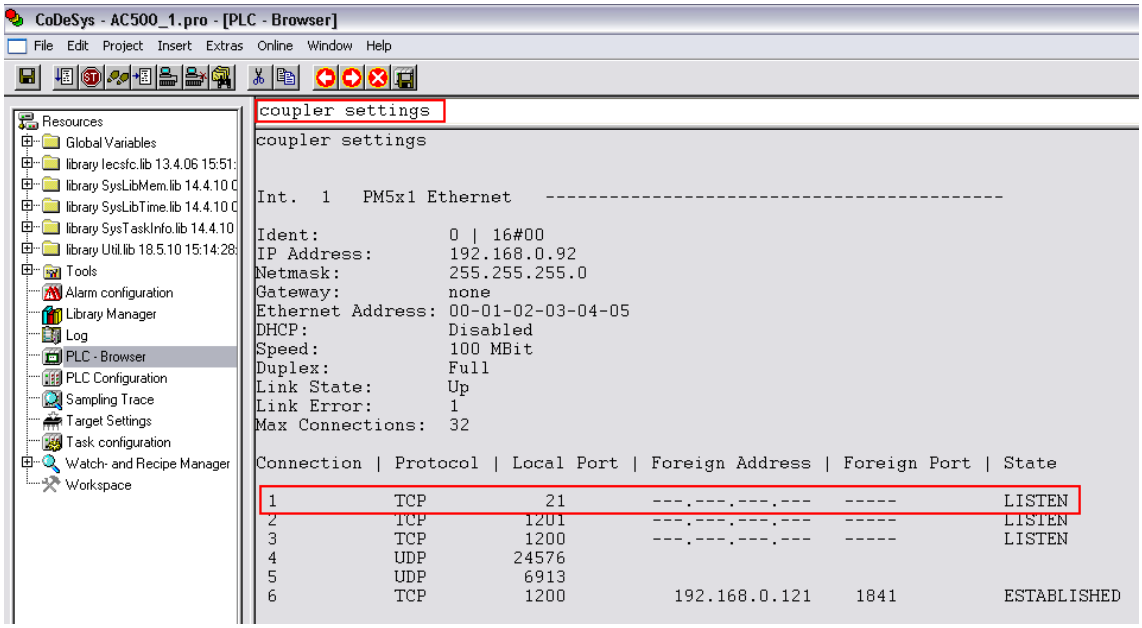


Configuration of FTP server (>= CBP 2.4)

1. Under Ethernet -> Protocols add a new object and select FTP server from the list.
2. Double-click the *"FTP_Server"* item to open FTP server configuration. Depending on the PLC device all or some of the following parameters and options are available:

Parameter		Default	Value	Description
Configuration for		ETH1	ETH1/ ETH2	Select the desired Ethernet connection used for FTP server.
FTP Server				
	Allow Firmware update	Disa-bled	Enabled	CBP firmware/Automation Builder firmware is not updated automatically. You can download a PLC firmware to the RAM disk and the PLC will automatically update the device with the downloaded firmware. Afterwards a restart of the PLC is required. For further details see FW_UPDATE_DOCUMENTATION.
			Disabled	CBP firmware is updated automatically.
	Port	21	1...x	Enter the TCP/IP-Port used to connect to the FTP server on the PLC.
	Sessions	1	1...4	Enter the max. number of allowed simultaneous and parallel connections to the FTP server. Each session uses one socket. Note: Some FTP cli-ents require several connections to work.
Passwords		-	-	Set each user's passwords for login. No entry = no password.
	root	-	-	
	ramdisk	-	-	The system RAM disk should only be used for firmware updates.
	sdcard	-	-	Inserted memory card.
	userdisk	-	-	User section of the flash disk.
	flashdisk	-	-	Only available with PM5675-2ETH
	sramdisk	-	-	

1. Create configuration & download project to PLC.
2. If the TCP protocol listens on the configured FTP port, the server is ready for login.



Connection to a PLC running a FTP server

After activating the FTP server, you can log onto the PLC from a PC with a FTP client. In the following example the web browser Microsoft Internet Explorer has been used:

1. Start the FTP client application.
2. Enter the URL with the syntax:

   "ftp://<USERNAME>:<USER_PASSWORD>@<HOST_IP>:<PORT>" into the address line with:

   "USERNAME" to the drive you want to access;

   "USER_PASSWORD" to the password of the user set in the Automation Builder;

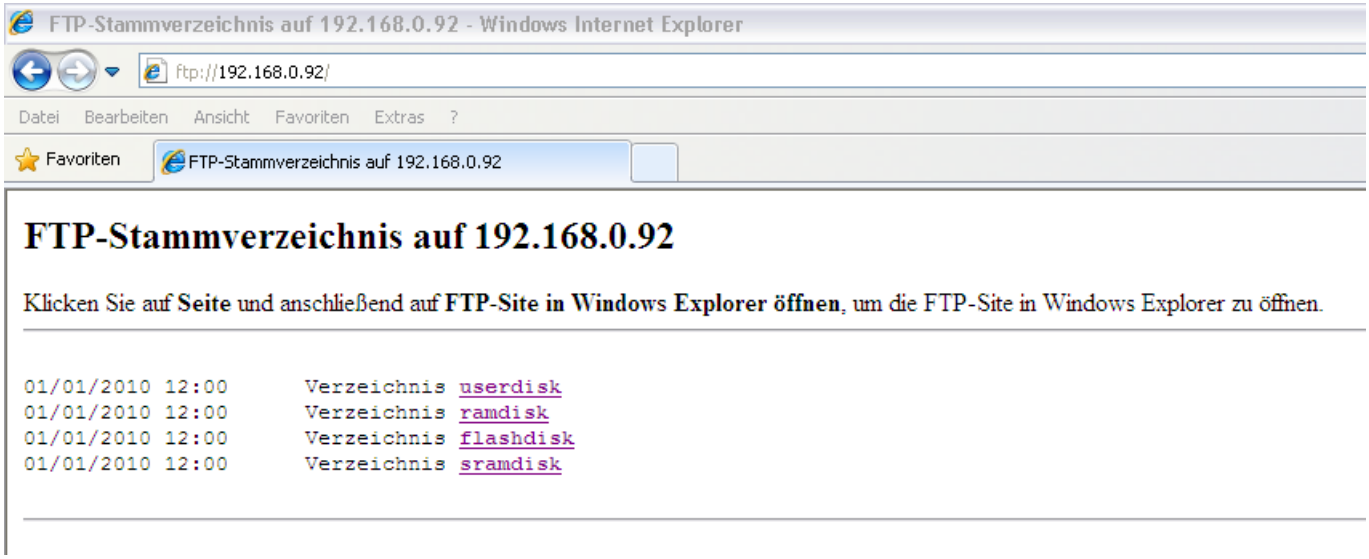
   "HOST_IP" to the IP address of your target PLC running the FTP server;

   "PORT" to the FTP port configured in the Automation Builder;

   for example:

   ftp://root:MyPwd@192.168.0.92:21

   ftp://sramdisk:Pwd123@192.168.0.83:20
3. Connect to the PLC (Press "Enter"-key).



If the connection has been established successfully, Internet Explorer will show the available contents of the root directory of the chosen user (i.e. the available drives is logged in as "root" as shown above). For further details on usage of a FTP client please refer to its documentation.

1.6.5.3.9 Web server

Configuration

Configuration in Automation Builder

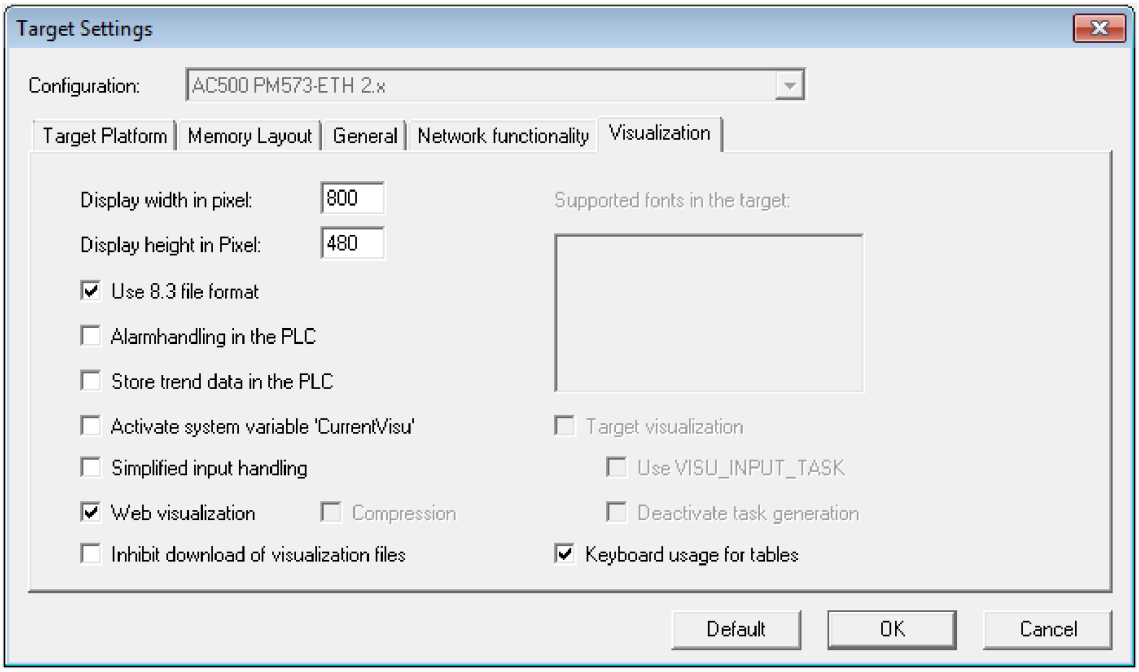
1. In your PLC project, click *"Interfaces ➔ Ethernet ➔ Protocols"* and add a new object *"Web Server"*.
2. Open the web server settings to change the default settings if required.

   Before changing the number of parallel connections for the web server, check the general information on *Chapter 1.6.4.1.6.1.1 "Ethernet protocols and ports for AC500 V2 products" on page 5442*.

Parameter	Default	Value	Description
Port	80	0 ... 65535	Listen port of the web server
Connections	2	1 ... 25 (depending on CPU type)	Number of parallel connections accepted by the web server. Depending on the CPU type and web-visu complexity it might be possible to have more connections to different clients. The connections will be opened and closed one after the other. This might lead to the impression of more parallel connections than configured.

Configuration in CODESYS

1. Open the CODESYS.
2. In the *“Resources”* tab click on *“Target Settings”* and select the *“Visualization”* tab.



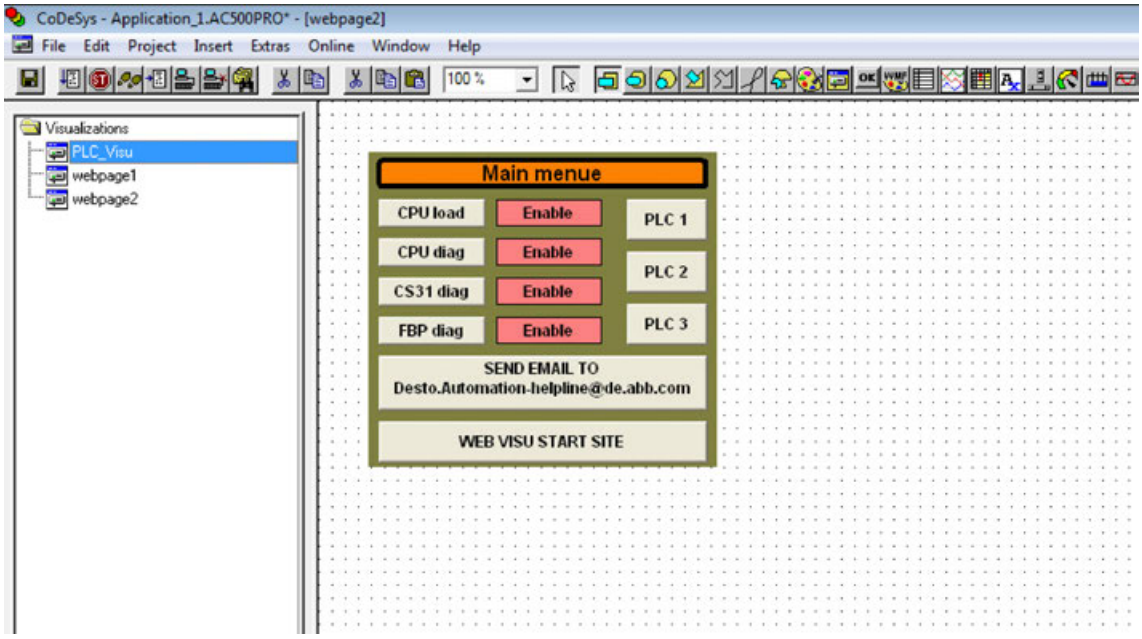
⇨ If required, change *“Display width”* and *“Display height”* for the web pages.

Enable *“Web visualization”* check box, otherwise the webpages are only visible in online mode.

Ensure *“Inhibit download of visualization files”* check box is disabled.

> *If you enable “Activate system variable 'CurrentVisu'” check box and define a global variable “CurrentVisu” [STRING(40)] in your project, then the name of the active visualization will be stored in this variable. You can switch between your visualization pages by changing this value.*

3. In CODESYS open the *"Visualizations"* tab to create the webpages. The first entry is the start page of your visualization project and must be named "PLC_Visu".

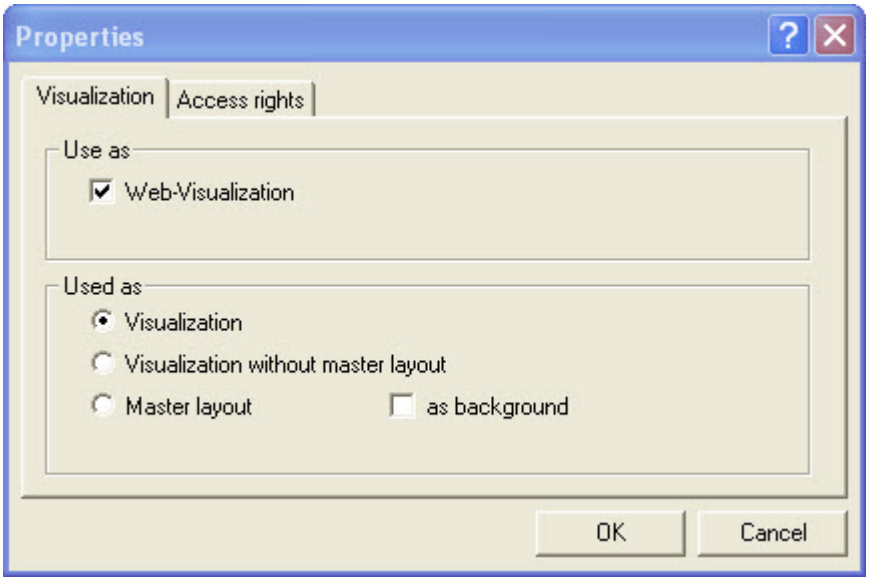


   ⇨ Further information on the creation and formatting of websites is provided in the CODESYS section ↳ *Chapter 1.4.5 "Web visualization" on page 721*. When building up your visualization project, consider the required disk space on the user RAM disk.

4. By default, all objects in the visualizations tab are marked for web visualization. When the project is built, all marked objects are saved as XML files (objectname.xml) into the project.

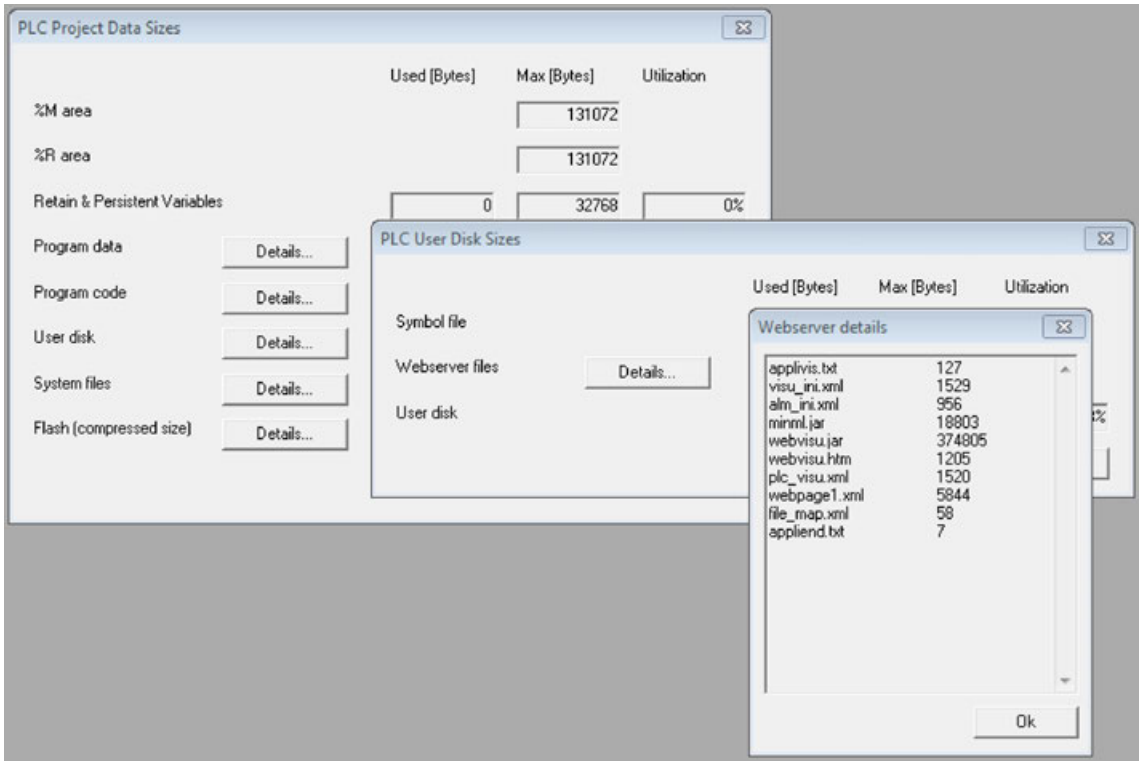
   To exclude a particular visualization object from the web visualization, click *"Project ➔ Object ➔ Properties"*.

   In the *"Visualization"* tab disable *Web-Visualization* check box.



5. Open a web browser and enter the IP address to the PLC (webserver project): http://<IP address/webvisu.htm>.

6. Click *“Online ➔ Show file information”* to check the *“PLC Project Data Sizes”*.



1.6.5.4 Data transfer and programming

1.6.5.4.1 Data transfer and CODESYS programming

Creating configuration data

☑ If the setup of all devices has been finished, you can create the configuration file.

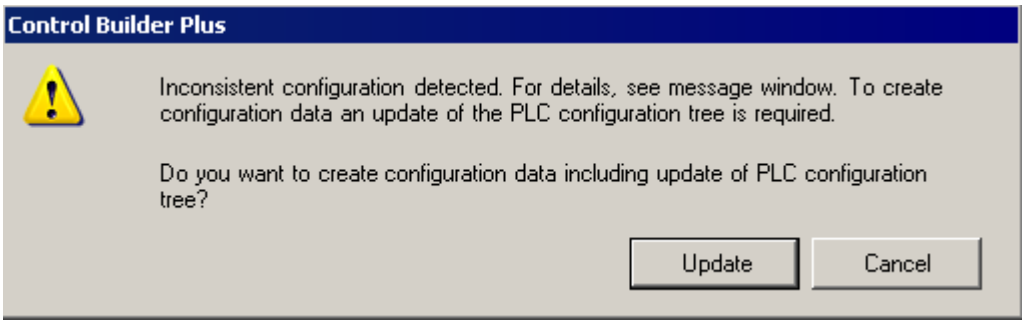
▷ Right-click on *“Application”* and select *“Create configuration data”* from context menu.

⇨ The generation process starts. The progress is shown in the status bar of Automation Builder.

Check integrity Within a Automation Builder version only device versions of this Automation Builder version can be used. By following the use case described in the chapter Use Cases the system takes care about this project integrity. Additionally Automation Builder performs an integrity check for the selected PLC before generating the configuration.

The integrity check can manually be called for the complete project:

1. Select the menu item *"Project ➔ Check integrity"*.
   ⇨ Automation Builder checks the project integrity for the complete project ("Project integrity" checks if all devices in the device tree are installed in the device repository).
   If the integrity check was successful a success message is displayed.
   If the integrity check was not successful the following message is displayed:



2. Click *"Update"* button:
   ⇨ Automation Builder updates the configuration to the latest version, creates configuration data and starts CODESYS V2.3.
3. Click *"Cancel"* button:
   ⇨ Creates configuration data and Automation Builder does not start CODESYS V2.3.

If the integrity check fails the Message Window displays further details.

In case of inconsistent configuration data update correct your project. Update all devices and install all 3rd party devices as used in the project.

Following error messages indicating that devices are not installed properly: "The device XXXXXX was not found in the device repository! Please reinstall this device using the menu item *"Tools ➔ DeviceRepository"*.

Install the device. ↳ *Chapter 1.6.5.2.1 "Device repository" on page 5811*.

"The version of device XXXXXX is not compatible with the current version."

Update to new version.

Launching programming system CODESYS V2.3.9.x

In your PLC project, double-click on *"Application"* to open the CODESYS V2.3.9.x project.

CAUTION!
Risk of damaged Automation Builder projects!
Projects created with Automation Builder are incompatible with CODESYS V2.3.9.x. Opening these projects directly with CODESYS V2.3.9.x may cause corrupted projects. Always use Automation Builder to open these projects.

If several instances of CODESYS V2.3.9.x are opened, double-clicking on the target item brings the corresponding CODESYS V2.3.9.x instance to the front.

CPU name

The name of the used AC500 CPU is shown in the main node of the device tree.

The names of the devices can be edited by selecting the corresponding entry and clicking the entry. The name must be unique and cannot be reused in the same project. Automation Builder automatically appends double entries with an up counting number as an suffix.

The IEC naming rules are not checked during input in Automation Builder.

Source download/upload in Automation Builder

Source download

The *"Source download"* downloads and stores the current project as a Zip-file on the PLC's memory card.

- Right-click *"PLC_AC500_V2 <...>"* node and click *"Source download"*.

Source upload

The *"Source upload"* retrieves a previously downloaded project from memory card to a selectable directory on the Automation Builder PC and optionally opens Automation Builder with this uploaded project.

- Click *"File"* in the main menu of Automation Builder and click *"Source upload"*.

Internally both commands are invoking a hidden instance of CODESYS V2.3 which is then working in batch mode to accomplish the download/upload.

Most of the error messages which could occur during the process are raised by CODESYS V2.3 and will then be displayed in Automation Builder as a result message box (the language of the message box is the installed language of CODESYS V2.3, the messages are sometimes quite "basic").

Before using the commands, check the following:

- *No CODESYS V2.3 instance started from within Automation Builder on the same PLC node should be running.*
- *To perform the download/upload, CODESYS V2.3 must be able to log in/on to the PLC. Therefore the gateway settings should be correct and saved to project file. It is recommended to test once the ability of CODESYS V2.3 to log in/on to the PLC.*
- *A formatted memory card must be mounted in the PLC's memory card slot.*

1.6.5.4.2 Programming and testing

Programming interfaces to the AC500 used by control builder plus / Automation Builder

The AC500 controllers provide the following interfaces for communication with other devices.

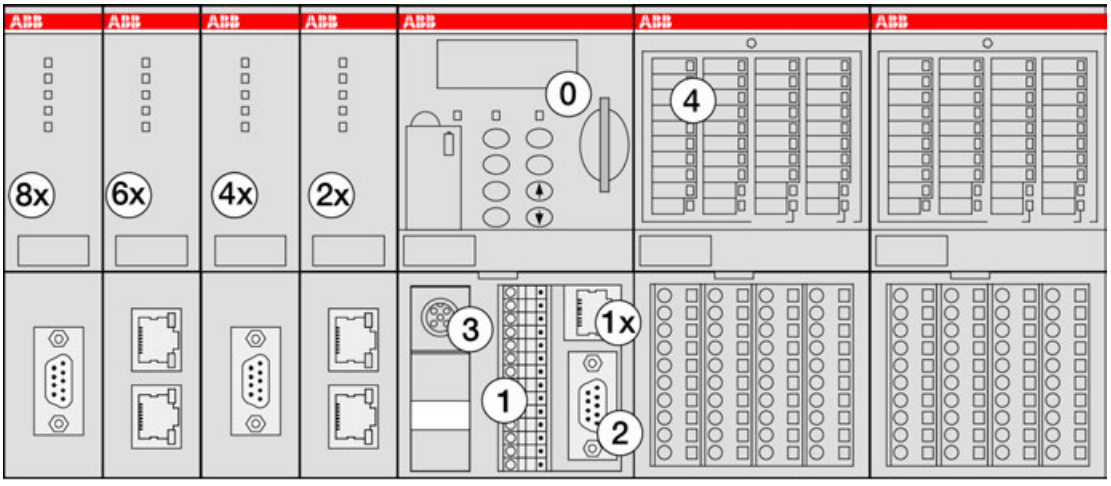
For all AC500 CPUs:

No.	Designation	Interface	Programming access
0	CPU	Own CPU	CPU for online operation
1	COM1	Serial interface COM1	yes
2	COM2	Serial interface COM2	yes

For PM57x, PM58x and PM59x only:

No.	Designation	Interface	Programming access
3	FBP	FBP slave interface	yes (as of FW V1.1.7)
4	I/O bus	I/O bus	no
1x	Line 0	Internal communication module with channel 0 x 9	depends on type
2x	Line 1	Communication module inserted in slot 1 with chan. 0 x 19	depends on type
4x	Line 2	Communication module inserted in slot 2 with chan. 0 x 19	depends on type
6x	Line 3	Communication module inserted in slot 3 with chan. 0 x 19	depends on type
8x	Line 4	Communication module inserted in slot 4 with chan. 0 x 19	depends on type

PM55x and PM56x CPUs only have RS-485 COM-interface, which can be transferred to RS-232 by using a RS-232->RS-485 adapter.



Communication drivers

The following communication drivers are available and recommended for programming the AC500:

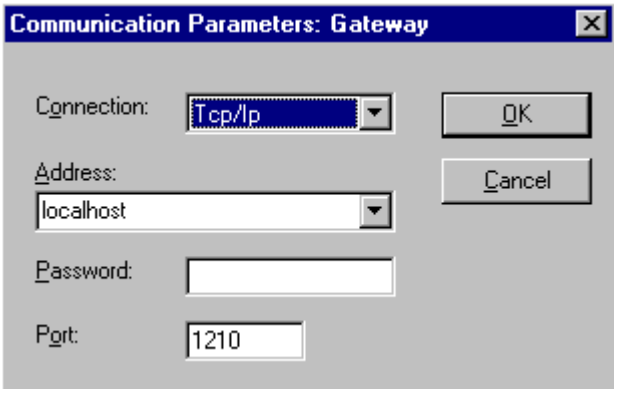
Communication driver	Description
Tcp/Ip	Ethernet driver
ABB Tcp/Ip Level 2 AC	Ethernet driver with routing functionality (as of PS501 V1.2, routing as of firmware V1.3.x)
Serial (RS-232)	Serial interface driver
ABB RS-232 Route AC	Driver for serial interfaces with routing functionality (as of PS501 V1.2, routing as of firmware V1.3.x)
ABB ARCNET AC	Driver for programming via ARCNET with routing and adjustable block size (as of PS501 V1.2, routing as of firmware V1.3.x)
Serial (Modem)	Modem driver for modem connected to serial interface of the PC and PLC
Note: It is recommended not to use the **Tcp/Ip (Level 2 Route)** driver for new projects. This driver is only available for compatibility reasons in order to support easy update of an old project.	

Routing is available with version V1.3.x of the Control Builder and PLC firmware.

Gateway configuration

In Control Builder, select "Online/Communication Parameters/Gateway" and select "Local" from the "Connection" list box:

The following gateway settings apply for the Ethernet driver "ABB Tcp/Ip Level 2 AC":



Communication parameters

The communication parameters and address data are set in the Control Builder by selecting the desired driver and specifying the parameters in the "Communication Parameters" window, which can be opened using the menu item "Online/Communication Parameters".

The following sections describe the drivers listed above and their settings.

Programming via the serial interfaces

The operation modes of the serial interfaces COM1 and COM2 are described in the chapter ↳ *Chapter 1.6.5.2.11 “Serial interfaces COM1 and COM2” on page 6098*. Both serial interfaces COM1 and COM2 are defined as programming interface by default.

The Installation guide provides information how to set the serial interfaces for the different PC operating systems.

For the CPUs PM57x, PM58x and PM59x, PC and PLC are connected via the Programming Cable TK501 or TK502.

For the CPUs PM55x and PM56x, PC and PLC are connected via the Programming Cable TK503 or TK504.

> *The load of PM55x and PM56x should not exceed 80 % in order to obtain a stable communication between the PLC and the PC via the serial interfaces. The load of the CPU can be displayed with the PLC Browser via the command cpuload.*

The interface parameters are set to fixed values and cannot be changed.

Transmission rate: 19200 baud

Parity bit: no

Data bits: 8

Stop bits: 1

Synchronization: none

Motorola byteorder: yes

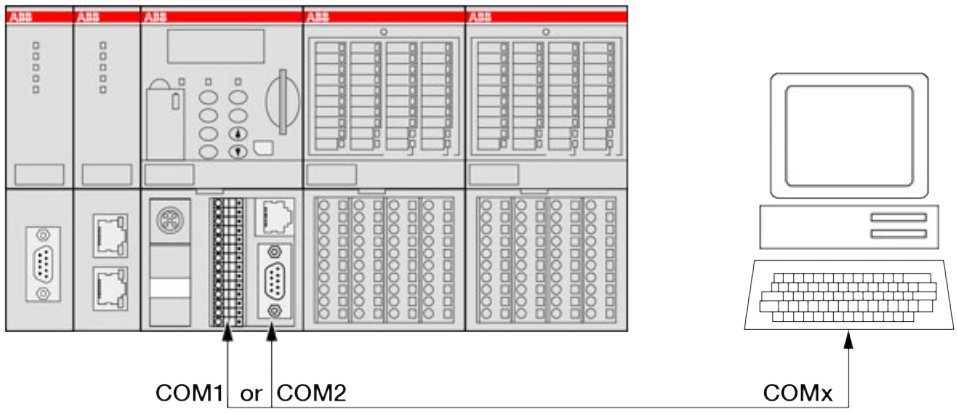
The following drivers are available for programming via the serial interfaces:

- Serial (RS-232)
- ABB RS-232 Route AC (as of PS501 V1.2, routing as of firmware V1.3.x)

Serial driver "Serial (RS-232)"

The serial driver "Serial (RS-232)" provides the following functions:

- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.0
- Parallel operation of Control Builder and OPC server

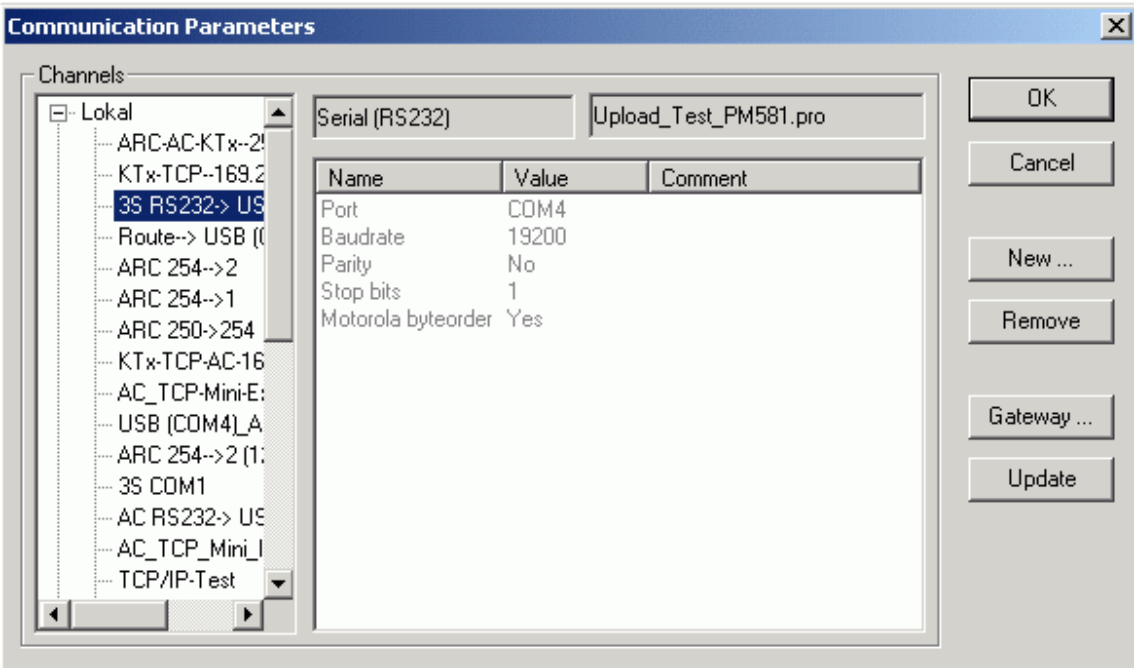


To define a new gateway channel for the serial driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example USB->COM4) and select the driver "Serial RS232" from the device list. Select the desired values by double clicking the corresponding parameter:

Port: COM port on the PC

Transmission rate: 19200 Baud

Motorola byteorder: Yes

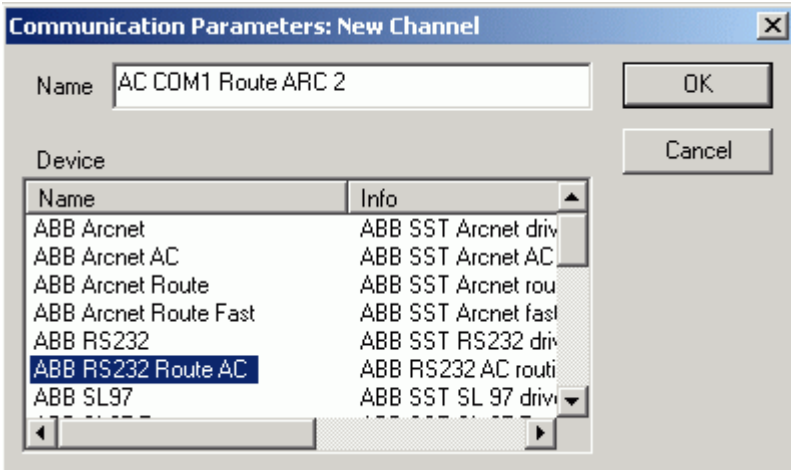


Serial driver "ABB RS232 Route AC"

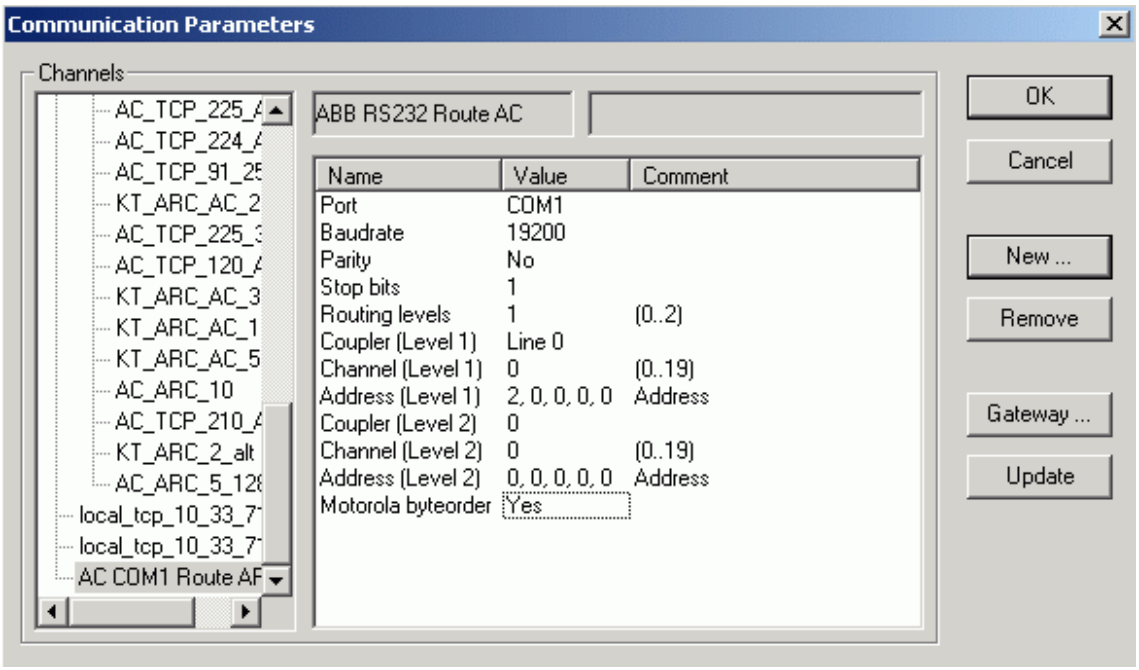
As of version V1.2.0, the serial routing driver "ABB RS232 Route AC" is available in addition to the serial driver. This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.2.0
- Online operation of PLCs connected via Ethernet or ARCNET using the serial interface (Control Builder version V1.3 and later)
- Parallel operation of Control Builder and OPC server

To define a new gateway channel for the serial routing driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example AC COM1 Route ARC 2) and select the driver "ABB RS232 Route AC" from the device list.



The following communication parameters can be set for the serial routing driver "ABB RS232 Route AC":



Parameter	Possible values	Description
Port	COMx (PC dependant)	Serial interface of the PC
Transmission rate	19200	Always 19200 baud
Parity	No	Always no parity
Stop bits	1	Always one stop bit
Routing levels	0...2	Routing levels (0 = none)
Communication Module (Level 1)	0, line 0...line 4	Communication Module for level 1
Channel (Level 1)	0...19	Channel on Communication Module level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target Communication Module level 1
Communication Module (Level 2)	0, line 0...line 4	Communication Module for level 2
Channel (Level 2)	0...19	Channel on Communication Module level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target Communication Module level 2
Motorola byteorder	yes	Selection of Motorola or Intel byte order

If you want to use the serial routing driver in order to directly access the connected CPU, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

Routing is available with version V1.3.x of the Control Builder and PLC firmware.

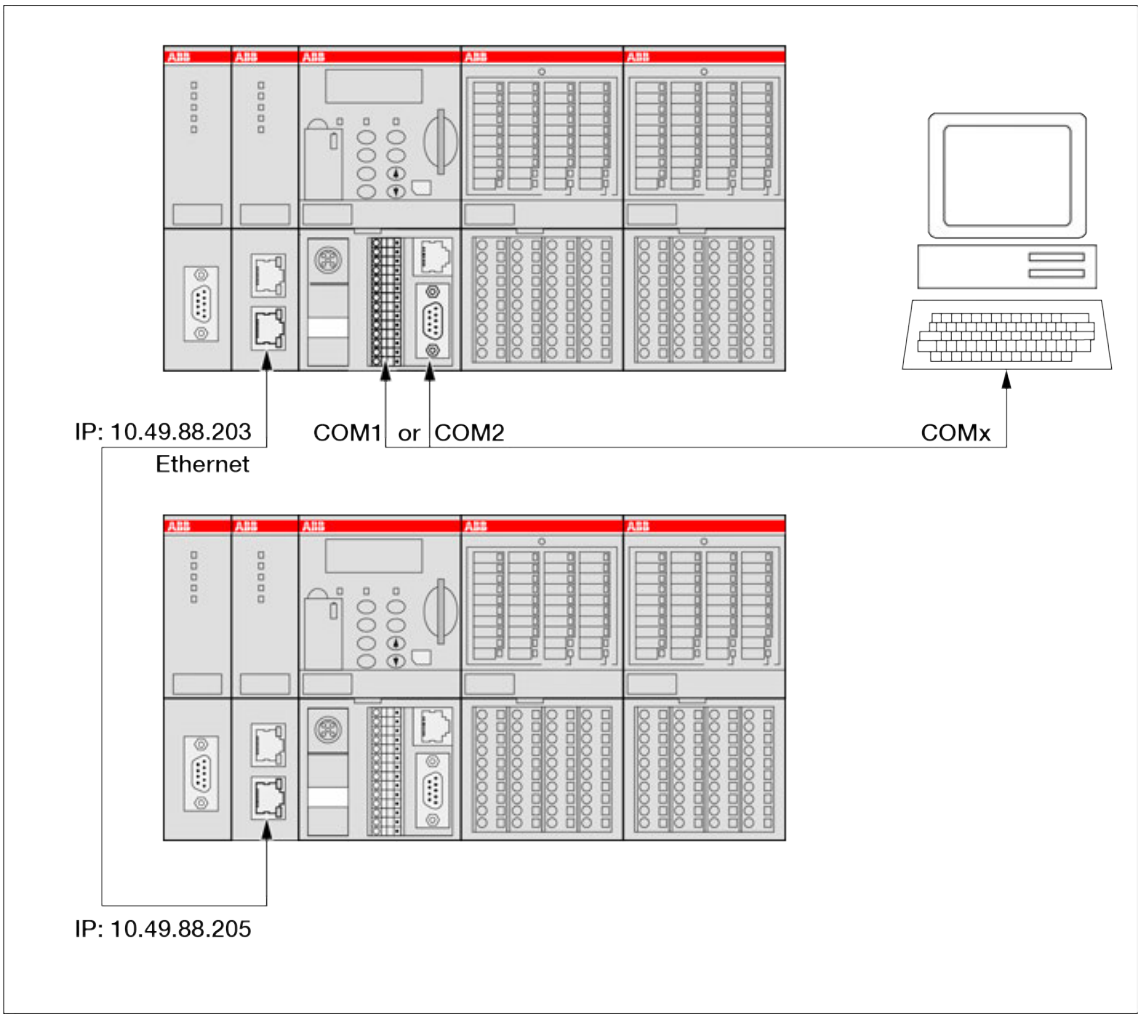
The following applies to the routing levels:

Routing levels = 0 No routing (parameters for Level 1 and Level 2 not set)

Routing levels = 1 Single-level routing (set parameters for Level 1)

Routing levels = 2 Double-level routing (set parameters for Level 1 and Level 2)

Example



Configuration of PLC 2 (IP: 10.49.88.205) shall be performed via the external Ethernet Communication Module (IP: 10.49.88.203) inserted in slot 1 of PLC 1. The serial PC interface COM2 is connected to the serial interface COM1 of PLC 1:

Parameter	Value	Remark
Port	COM2	PC COM2
Transmission rate	19200	
Parity	No	
Stop bits	1	
Routing levels	1	Single-level routing
Communication Module (Level 1)	Line 1	Communication Module in slot 1
Channel (Level 1)	0	Channel 1
Address (Level 1)	10, 49, 88, 205, 0	Subscriber address of the target PLC (Node 2)
Communication Module (Level 2)	0	No level 2

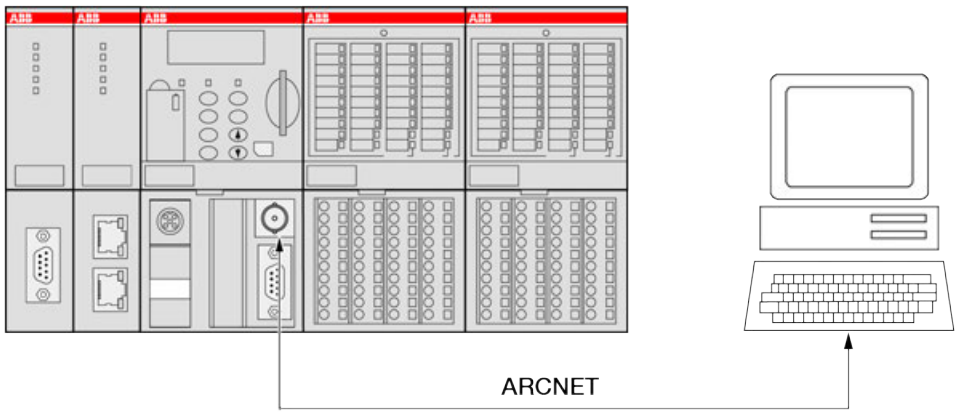
Parameter	Value	Remark
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Motorola byteorder	yes	Motorola byteorder

Programming via ARCNET

The ARCNET Communication Module is available as of Control Builder version V1.2 and PLC firmware version V1.2.0.

Programming via ARCNET is only possible on a PC with installed ARCNET board.

When programming via ARCNET, the PC is an ARCNET node.



The "sender node", i.e., the ARCNET subscriber address of the PC is set in the file:

Arcnet_xx.ini with the parameter NodeID1 = 254.

The file Arcnet_xx.ini is located in the folder where the PC operating system is installed (for example C:\WINNT for Win2000).

For a PC with installed ARCNET board, the file Arcnet_xx.ini contains for example the following entries:

[ARCNET]

DriverAccessName1=FARC

;Default = Farc

NodeID1 = 254

; Default = 254

;DriverAccessName2=FARC1

; Default = Farc1

;NodeID2 = 253

; Default = 253

;DriverAccessName3=FARC11

; Default = Farc11

;NodeID3 = 252

; Default = 252

;DriverAccessName4=FARC111

; Default = Farc111

;NodeID4 = 251

; Default = 251

The current card drivers can be downloaded from the SoHard website. The driver versions listed in the "Driver file" column were tested.

Installation of ARCNET cards

Overview of Supported ARCNET cards

Support of ARCNET cards is available as of version V2.x

The following ARCNET cards are available for the programming of AC500 PLCs in the integrated ARCNET Communication Module.

ARCNET card	Bus type	Operating system	Driver file (on SoHard website)	Driver
SH FARC E3 (K) (with COM20022)	ISA	Win2000/XP	W2K_RAW_DRIVER_V0518.zip ..\ARCNET\Win2000	Farc.sys
SH ARC PCI	PCI	Win2000/XP	W2K_RAW_DRIV-ERSH FARC E3 (K) (with COM20022)_V0518.zip ..\ARCNET\Win2000	Farc.sys
SH ARC PCMCIA with SH KOAX-PCMCIA	PCMCIA	Win2000/XP	W2K_RAW_DRIVER_V0518.zip ..\ARCNET\Win2000	Farc.sys

The current card drivers can be downloaded from the SoHard website. The driver versions listed in the "Driver file" column were tested.

Installation of ARCNET cards under Win2000 / XP

SH ARC PCI - ARCNET card SH ARC PCI - ARCNET Card for PCI Bus under Win2000 / XP

Driver files Internet: W2K_RAW_DRIVER_V05xx.zip

CD V5.x: ..\CD_Menu\Files_V5x\ARCNET\Win2000\farc.sys (sharcraw.inf)

packed: W2K_RAW_DRIVER_V0517.zip

Driver after installation: ..\WINNT\system32\drivers\farcpci.sys

Installing the card Proceed as follows to install the ARCNET card:

1. Delete the possibly existing old ARCNET driver (V4.xx).

   To do so, delete the following path in the registry:

   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\farc (or farc66)
2. Reboot the PC afterwards.
3. Copy the driver without any subdirectory to a floppy disk.
4. Shut down the operating system of the PC and then switch the computer off.
5. Install the ARCNET card to a PCI slot.
6. Reboot the PC. The card is automatically detected since it supports PnP.

   The message "Found new Hardware / Network controller -> Please wait" is displayed.

   The "Hardware Wizard" is executed. This may take some time.
7. After the "Found ..." message is displayed, click on "Have Disk" and select:

   "A:\sharcraw.inf" and "Next". The driver is going to be installed.
8. Reboot the PC.
9. Select "Start => Settings => Control Panel => System => Hardware => Device Manager" and then "SoHard ARCNET / SH ARC PCI (RAW)".
10. In the "Properties", change the DriverAccessName from FARCPCI to FARC.
11. Reboot the PC.

Uninstalling the card

Proceed as follows to remove the ARCNET card:

1. Select "Start => Settings => Control Panel => System => Hardware => Device Manager".
2. Select "SoHard ARCNET / SH ARC PCI (RAW)".
3. Select "Delete".
4. Shut down the operating system of the PC and then switch the computer off.
5. Remove the card and restart your PC.

Tips and tricks

The following notes will help you, if the communication does not work properly after the installation.

- The card SH ARC PCI requires PCI version V2.10 and hardware level V1.1.
- Test program for the drivers:
  The program arcread.exe is supplied together with the drivers. Start it from a DOS shell using the following command:
  arcread farc 254<ENTER>
  The driver will either be started or a corresponding error message will be displayed!
- LED indication:
  If the driver is activated correctly and no ARCNET cable is connected, both LEDs should flash (on/off time approx. 1 s).
  If only the green LED flashes, the interrupt has failed!
- If necessary, use another PCI slot.

SH ARC PCMCIA card

SH ARC PCMCIA - ARCNET-PCMCIA Card under Win2000 / XP

Driver files

Internet: W2K_RAW_DRIVER_V05xx.zip

CD V5.x: ..\CD_Menu\Files_V5x\ARCNET\Win2000\farc.sys (sharcraw.inf) packed: W2K_RAW_DRIVER_V0517.zip

Driver after installation: ..\WINNT\system32\drivers\farcpcmcia.sys

Installing the card

Proceed as follows to install the ARCNET card:

1. Delete the possibly existing old ARCNET driver (V4.xx).
   To do so, delete the following path in the registry:
   HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\farc (or farc66)
   After this, reboot the PC.
2. Copy the driver without any subdirectory to a floppy disk.
3. Shut down the operating system of the PC and then switch the computer off.
4. Install the ARCNET card to a PCMCIA slot.
5. Reboot the PC. The card is automatically detected since it supports PnP.
   The message "Found new Hardware / Network controller -> Please wait" is displayed.
   The "Hardware Wizard" is executed. This may take some time.
6. After the "Found ..." message is displayed, click on "Have Disk" and select:
   "A:\sharcraw.inf" and "Next". The driver is going to be installed.
7. Reboot the PC.
8. Select "Start => Settings => Control Panel => System => Hardware => Device Manager" and then "SoHard ARCNET / SH ARC PCMCIA (RAW)".
9. In the "Properties" change the DriverAccessName from FARCPCMCIA to FARC.
10. Reboot the PC.

Uninstalling the card

Proceed as follows to remove the ARCNET card:

1. Select "Start => Settings => Control Panel => System => Hardware => Device Manager".
2. Select "SoHard ARCNET / SH ARC PCMCIA (RAW)".
3. Select "Delete".
4. Shut down the operating system of the PC and then switch the computer off.
5. Remove the card and restart your PC.

Tips and tricks

The following notes will help you, if the communication does not work properly after the installation.

- Verify that the IRQ and/or the I/O range are not assigned twice.
  To do so, select
  "Start => Settings => Control Panel => Network => Network Adapter => Resources".
- Test program for the drivers:
  The program arcread.exe is supplied together with the drivers. Start it from a DOS shell using the following command: arcread farc 254 <ENTER>
  The driver will either be started or a corresponding error message will be displayed!
- If the error message ..59 (E_FARC_NO_NEXTID) is displayed, the selected interrupt has failed. In this case you should change between master PIC (IRQ 0-7) and slave PIC (IRQ 8-15). IRQ 5 or IRQ 7 should work in almost every case.
- LED indication:
  If the driver is activated correctly and no ARCNET cable is connected, both LEDs should flash (on/off time approx. 1 s). If only the green LED flashes, the interrupt has failed!

SH FARC E3 [K] card

SH FARC E3 [K] - ARCNET Card for ISA Bus under Win2000 / XP

Driver files

Internet: W2K_RAW_DRIVER_V05xx.zip

CD V5.x: ..\CD_Menu\Files_V5x\ARCNET\Win2000\farc.sys

packed: W2K_RAW_DRIVER_V0517.zip

Driver after installation: ..\WINNT\system32\drivers\farc0.sys

Installing the card

Proceed as follows to install the ARCNET card:

1. Delete the possibly existing old ARCNET driver (V4.xx). To do so, delete the following path in the registry: HKEY_LOCAL_MACHINE\System\CurrentControlSet\Services\farc (or farc66) After this, reboot the PC.
2. Determine a free IRQ and I/O range (16 bytes), e.g. by selecting "Start => Programs => Accessories => System Tools => System Information => Hardware Resources".
3. Shut down the operating system of the PC and then switch the computer off.
4. On the card, set the I/O range and the interrupt IRQ according to the enclosed documentation.

   The following settings are set ex works:

   I/O-Address = 120 hex

   IRQ = 10

   KOAX mode

   2.5 MHz

5. Install the ARCNET card to an ISA slot.
6. Reboot the PC.
7. Copy the driver without any subdirectory to a floppy disk.
8. Select "Start => Settings => Control Panel => System => Hardware => Hardware Wizard => OK".
9. Select "Choose a Hardware device" and "Add new device" and then confirm with "Next".
10. A new hardware component is searched. This may take some time. Since the card does not support PnP, the message "Does not find any new devices on your computer" is displayed.
11. Select "Other devices" and click on "Have disk".
12. Insert the floppy disk and confirm with "OK". The message "Install from disk" is displayed.
13. Select "A:\tsharcraw.inf" and confirm with "OK". The driver is going to be installed.
14. Enter the following card properties:

    Controller: COM20022

    DriverAccess: FARC (is entered automatically)

    IRQ: Free IRQ determined in step 2

    I/O Address: Free I/O range determined in step 2
15. Confirm with "Ready" and "Close" and answer the message dialog with "Yes" to restart the PC.

Uninstalling the card

Proceed as follows to remove the ARCNET card:

1. Select "Start => Settings => Control Panel => System => Hardware => Device Manager".
2. Select "SoHard ARCNET / SH FARCxx (RAW)".

3. Select "Delete".
4. Shut down the operating system of the PC and then switch the computer off.
5. Remove the card and restart your PC.

Tips and tricks

The following notes will help you, if the communication does not work properly after the installation.

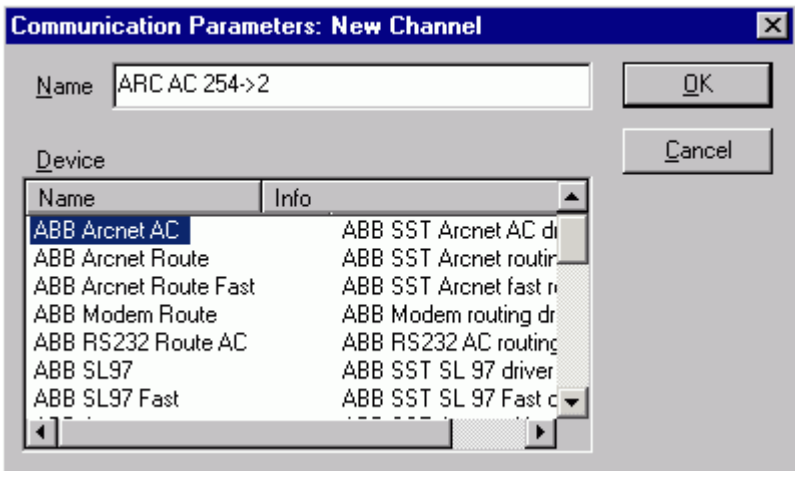
- Verify that the IRQ and/or the I/O range are not assigned twice.
  To do so, select
  "Start => Settings => Control Panel => System => Hardware => Device Manager => SoHard ARCNET => SH FARCxx".
- The interrupt must be enabled for ISA in the BIOS.
- Test program for the drivers:
  The program arcread.exe is supplied together with the drivers. Start it from a DOS shell using the following command: arcread farc 254<ENTER>
  The driver will either be started or a corresponding error message will be displayed!
- If the error message ..59 (E_FARC_NO_NEXTID) is displayed, the selected interrupt has failed. In this case you should change between master PIC (IRQ 0-7) and slave PIC (IRQ 8-15). IRQ 5 or IRQ 7 should work in almost every case.
- LED indication:
  If the driver is activated correctly and no ARCNET cable is connected, both LEDs should flash (on/off time approx. 1 s).
  If only the green LED flashes, the interrupt has failed!

ARCNET driver "ABB ARCNET AC"

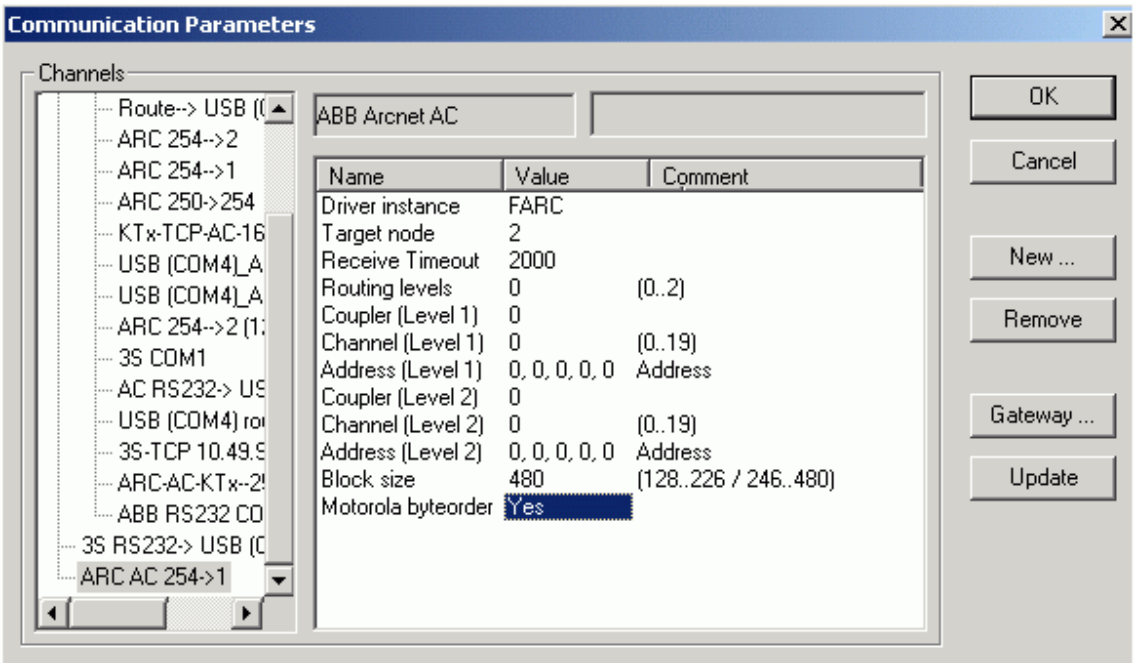
As of runtime system version V1.2.0 and Control Builder version V1.2, the driver "ABB ARCNET AC" is available. This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs

To define a new gateway channel for the ARCNET routing driver, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ARC AC 254 -> 2) and select the driver "ABB ARCNET AC" from the device list.



The following communication parameters can be set for the ARCNET routing driver "ABB ARCNET AC":

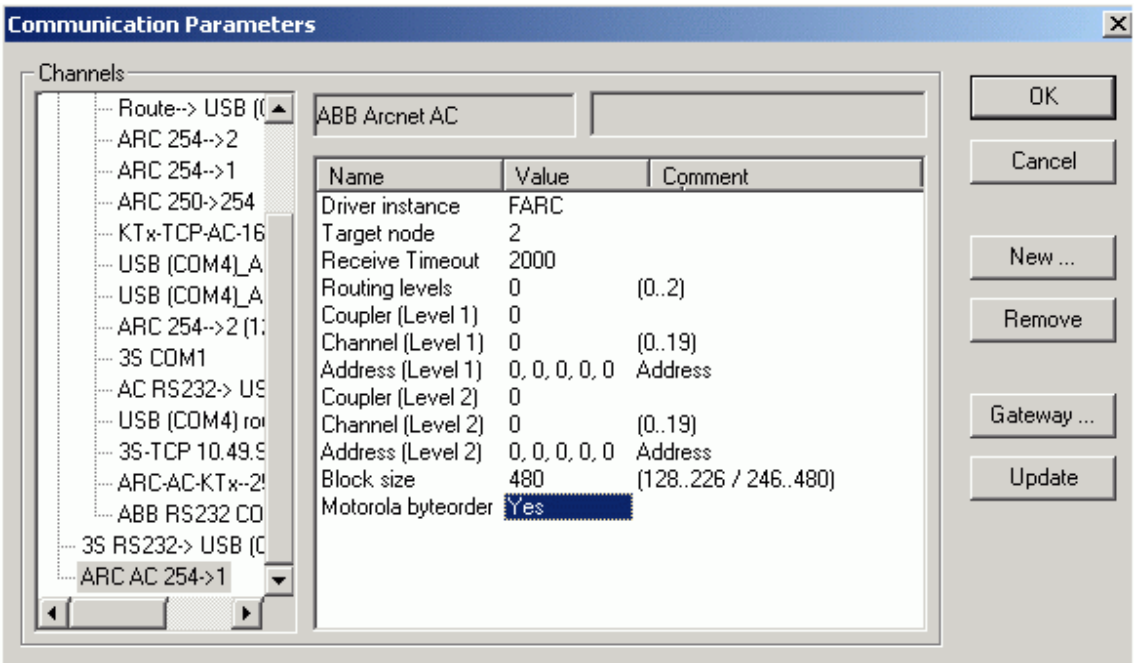


Parameter	Possible values	Description
Driver instance	FARC	DriverAccessName set in Arcnet_xx.ini
Target node	1...255	ARCNET subscriber address of the PLC
Receive Timeout	≥ 2000	Timeout [ms] for response
Routing levels	0...2	Routing levels (0 = none)
Communication Module (Level 1)	0, line 0...line 4	Communication module for level 1
Channel (Level 1)	0...19	Channel on communication module level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 1
Communication Module (Level 2)	0, line 0...line 4	Communication module for level 2
Channel (Level 2)	0...19	Channel on communication module level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 2
Block size	128...226 / 246...480	User data size
Motorola byteorder	yes/no	Motorola or Intel byte order

If you want to use the ARCNET routing driver to directly access the connected CPU, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

The parameter "Block size" (128...480) sets the number of user data within one block. The default value is 480 (this is the maximum allowed block size). Values in the range of 227 .. 245 are not allowed.

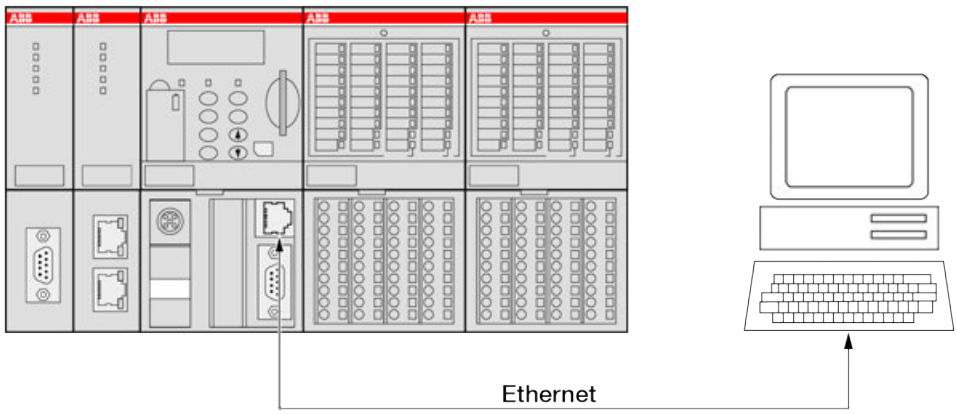
The parameter "Motorola byteorder" must be set to "Yes" for AC500 controllers.



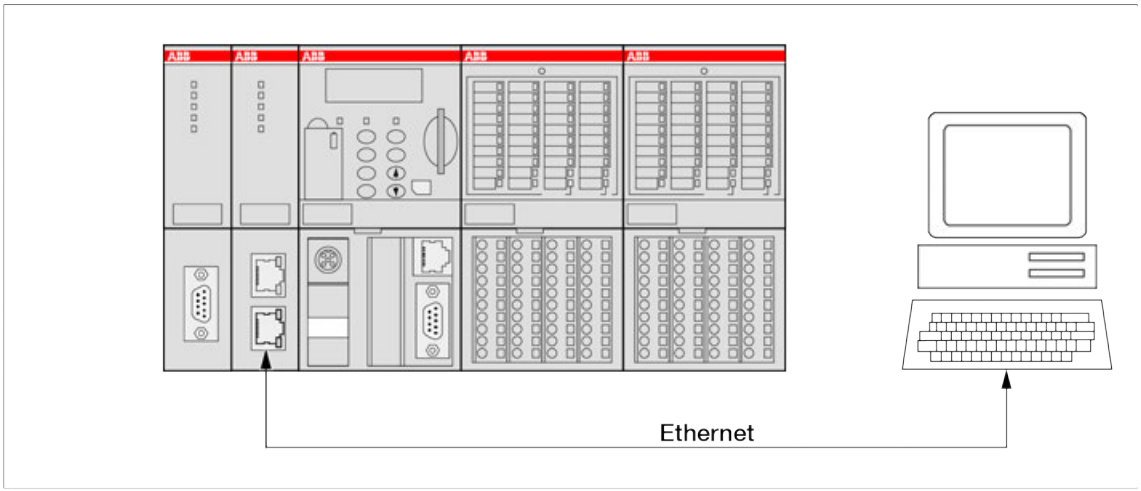
Configuration of communication via Ethernet (TCP/IP)

Programming via Ethernet is only possible on a PC with Ethernet board and installed network. Programming can be done via the internal and external Ethernet communication module.

Programming via internal (onboard) Ethernet communication module:



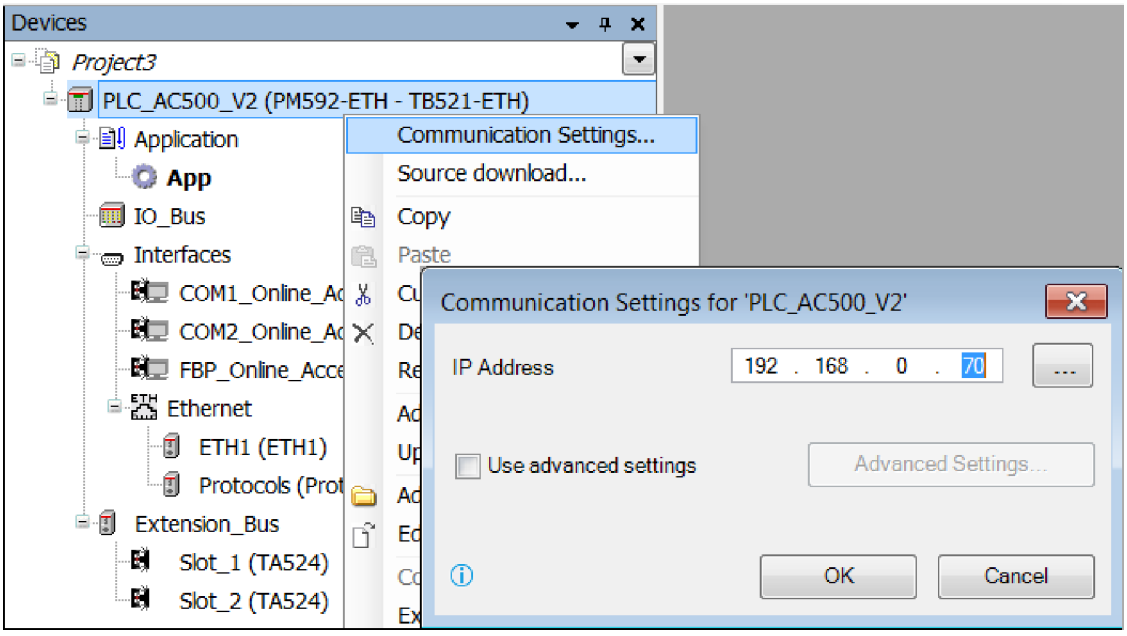
Programming via external Ethernet communication module (in the example communication module 1 in slot 1):



Enter a known PLC IP address

1. Right-click the top node *“PLC_AC500 <...>”* and select *“Communication Settings”* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.

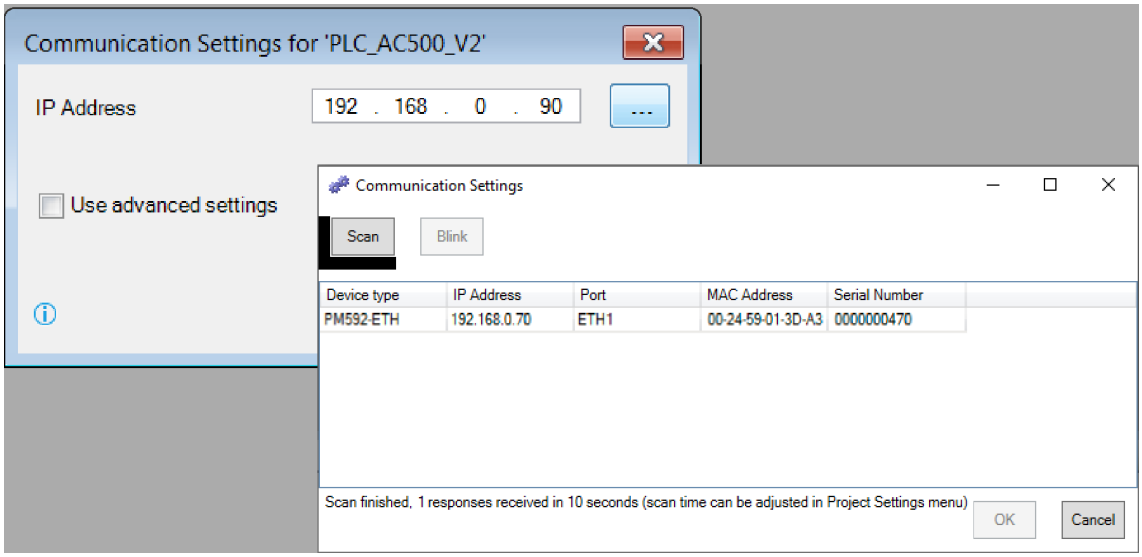


2. Enter your PLC *IP Address* and click *[OK]*.

Enter PLC IP address by scanning devices

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.



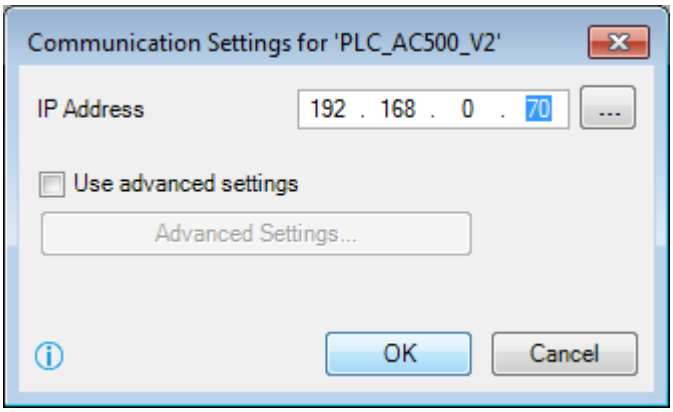
2. Click *[...]*.

   ⇨ Dialog box *Communication Settings <...>* appears.

3. Click *[Scan]*, select your desired PLC and click *[OK]*.

   ⇨ Entry is transferred to the dialog box *Communication Settings <...>*.

   Click *[OK]*.



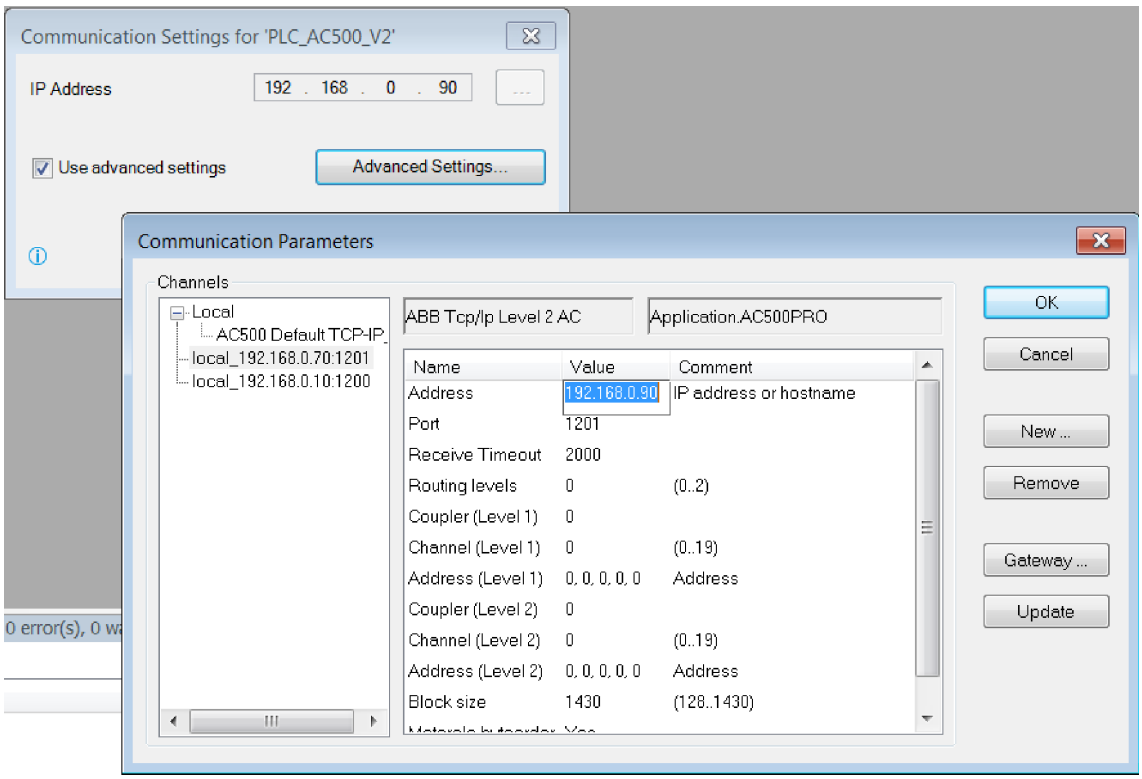
4. Click to log in the *"PLC_AC500_V2"* project.

Enter PLC IP address by *[Advanced Settings...]*

If a remote gateway instead of a local one has to be used it can be configured in the *[Advanced Settings...]*.

1. Right-click the top node *"PLC_AC500 <...>"* and select *"Communication Settings"* from the context menu.

   ⇨ Dialog box *Communication Settings <...>* appears.



2. Enable checkbox *Use advanced settings* and click *[Advanced Settings]*.

   ⇨ Dialog box *Communication Parameters* appears.

3. Select your channel, enter your PLC IP address and click *[OK]*.

   ⇨ Entry is transferred to the dialog box *Communication Settings <...>*.

   Click *[OK]*.

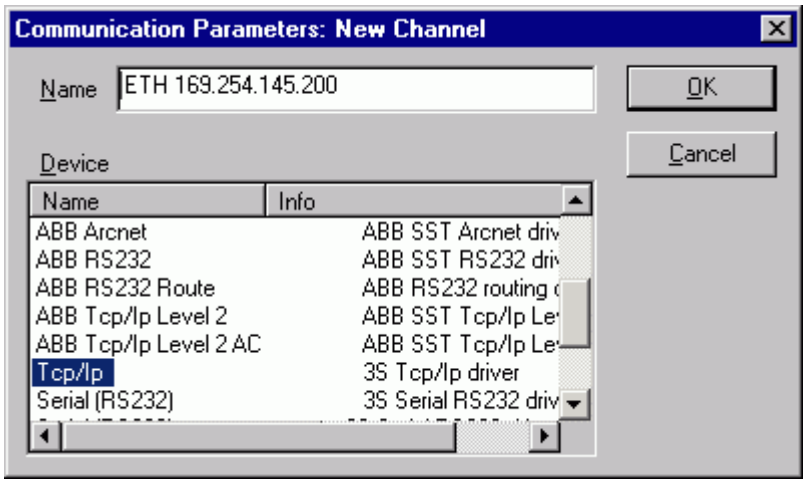
4. Click to log in the *"PLC_AC500_V2"* project.

Ethernet driver "TCP/IP"

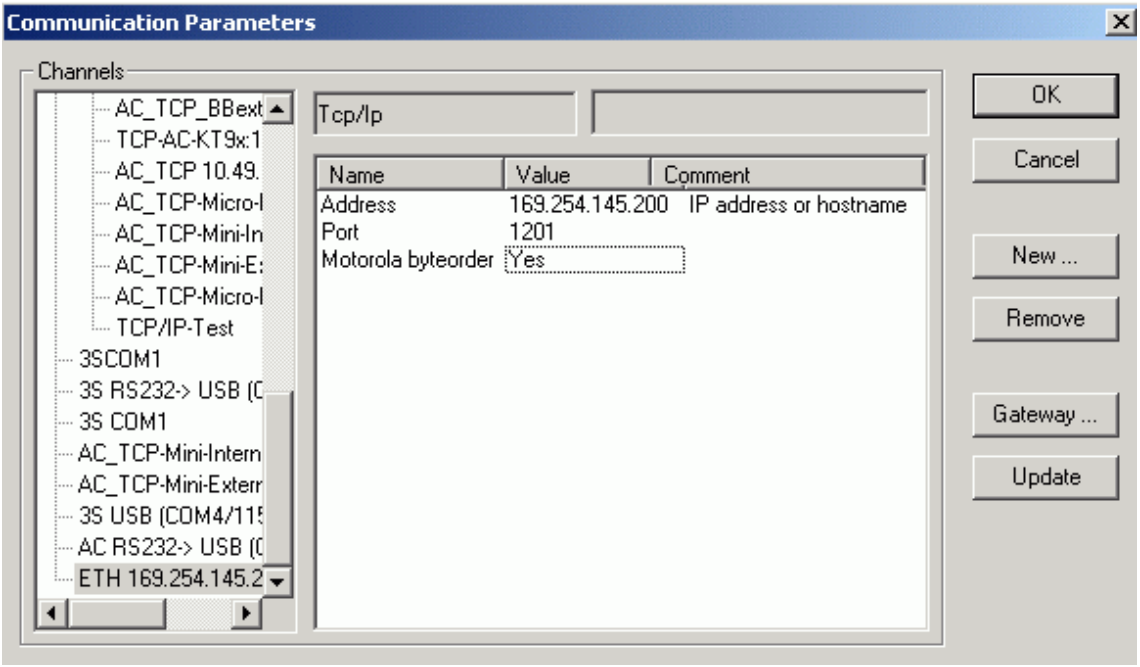
Programming AC500 controllers with internal and/or external Ethernet communication module via Ethernet can be done by using the driver "TCP/IP". This driver provides the following functions:

- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and OPC server
- Parallel operation of Control Builder instances with several PLCs

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "TCP/IP" from the device list.



The following communication parameters can be set for the Ethernet driver "TCP/IP":



Parameter	Possible values	Description
Address	0.0.0.0	IP address or hostname of the PLC
Port	1201	Port 1201
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

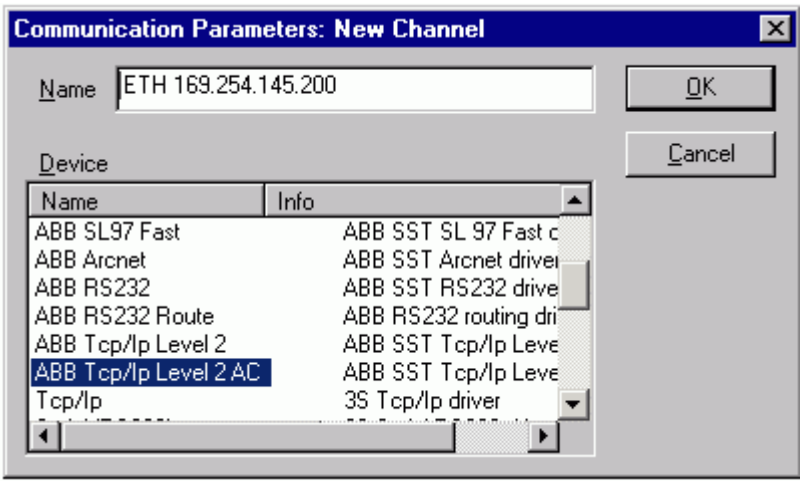
Ethernet driver "ABB TCP/IP Level 2 AC"

As of version V1.2, the driver "ABB TCP/IP Level 2 AC" is available for programming AC500 controllers with internal and/or external Ethernet communication module via Ethernet. This driver provides the following functions:

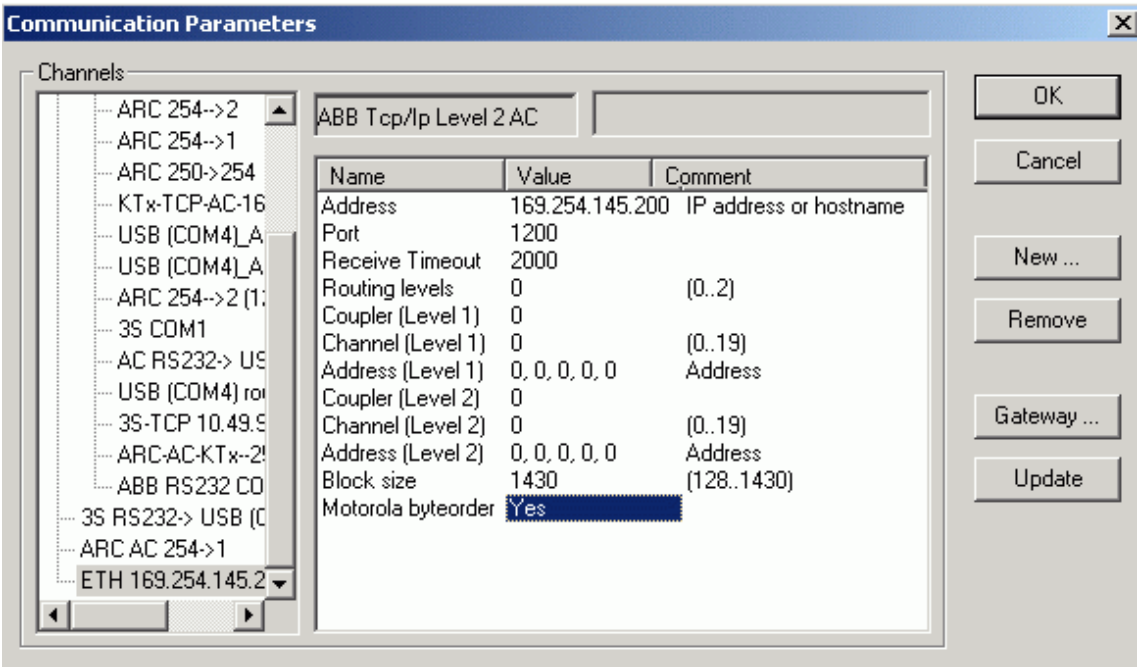
- Online operation of the PLC with the Control Builder
- OPC connection with OPC server, as of version V1.3
- Parallel operation of Control Builder and OPC server

- Parallel operation of Control Builder instances with several PLCs
- Online operation of PLCs connected via ARCNET. One PLC equipped with Ethernet communication module and one PLC with ARCNET communication module (Routing Ethernet -> ARCNET), as of version V2.x

To define a new gateway channel for the Ethernet interface, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example ETH 169.254.145.200) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.



The following communication parameters can be set for the Ethernet driver "ABB TCP/IP Level 2 AC":

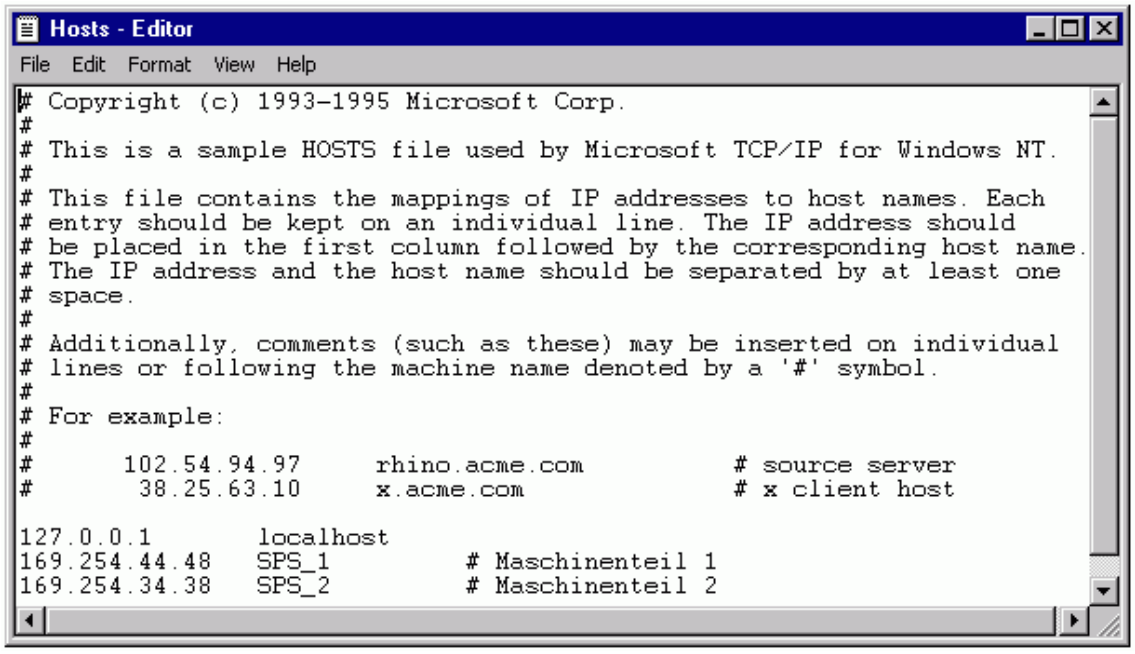


Parameter	Possible values	Description
Address	0.0.0.0	IP address or hostname of the PLC
Port	1200	Port 1200
Timeout (ms)	>= 2000	Timeout [ms] for response
Routing levels	0...2	Routing levels (0 = none)

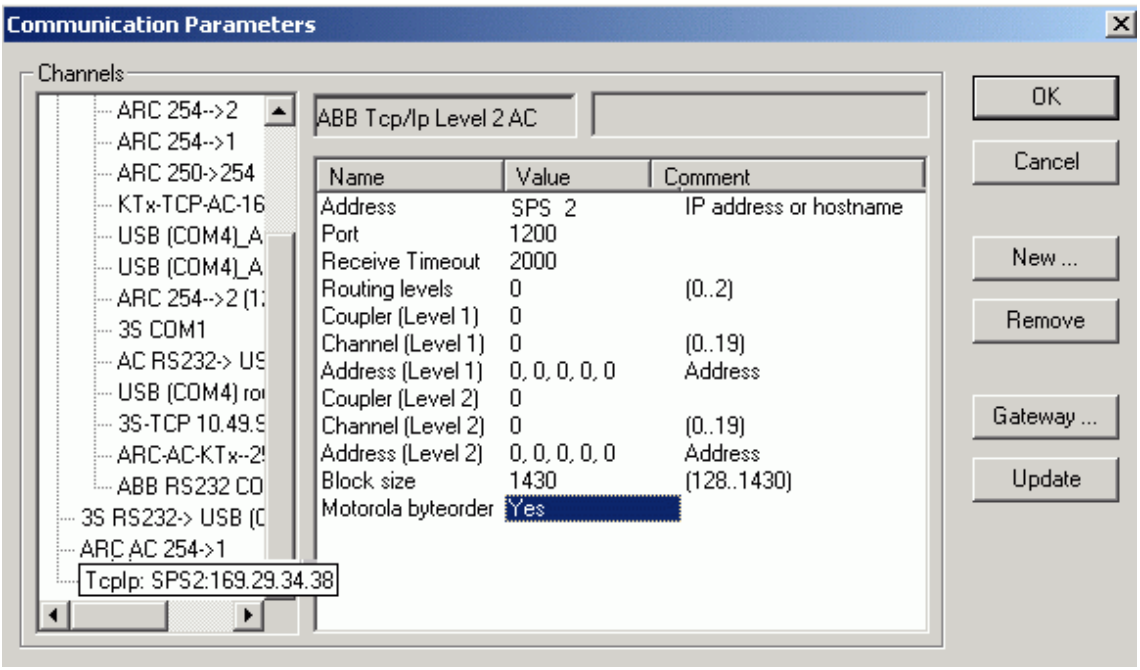
Parameter	Possible values	Description
Communication Module (Level 1)	0, line 0...line 4	Communication module for level 1
Channel (Level 1)	0...19	Channel on communication module level 1
Address (Level 1)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 1
Communication Module (Level 2)	0, line 0...line 4	Communication module for level 2
Channel (Level 2)	0...19	Channel on Communication Module level 2
Address (Level 2)	0, 0, 0, 0, 0 (max. 5 bytes)	Address in target communication module level 2
Block size	1430 (128...1430)	Bytes per telegram (unallowed 227..245)
Motorola byteorder	Yes (Yes/No)	Motorola or Intel byteorder (=Yes for AC500)

If you want to use the Ethernet driver to directly access the PLC, set all routing parameters (parameter Routing levels and following parameters listed in the table above) to 0.

The "Address" parameter sets the IP address or hostname of the PLC. To be able to use hostnames, the names have to be added to the file "Hosts". Under Win2000, this file is located in the directory "WINNT\System32\drivers\etc".



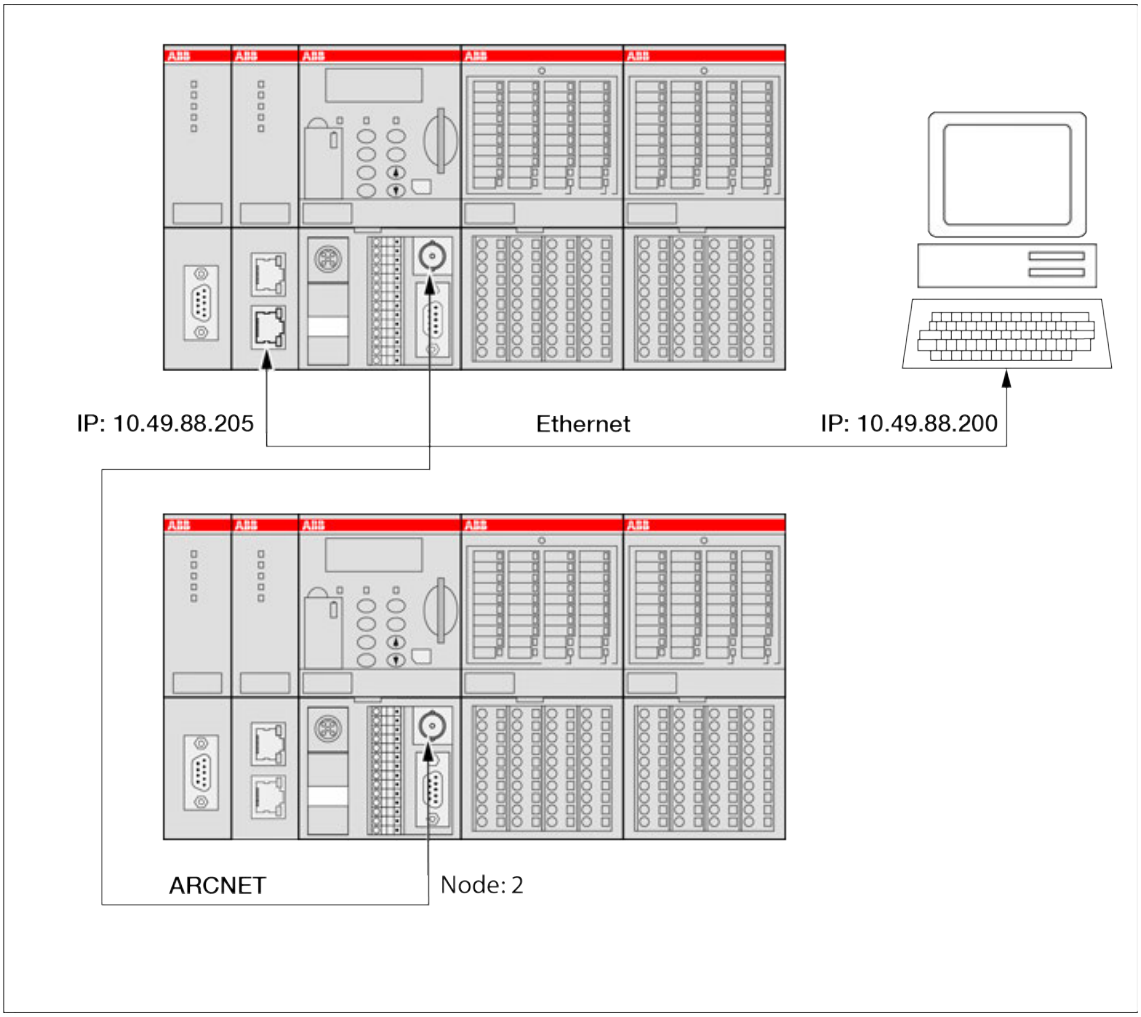
If you have changed the "Hosts" file accordingly, you can enter the symbolic name for the "Address" parameter instead of the IP address. In the following figure, the IP address "169.254.34.38" is replaced by the hostname "SPS_2".



Ethernet ARCNET routing

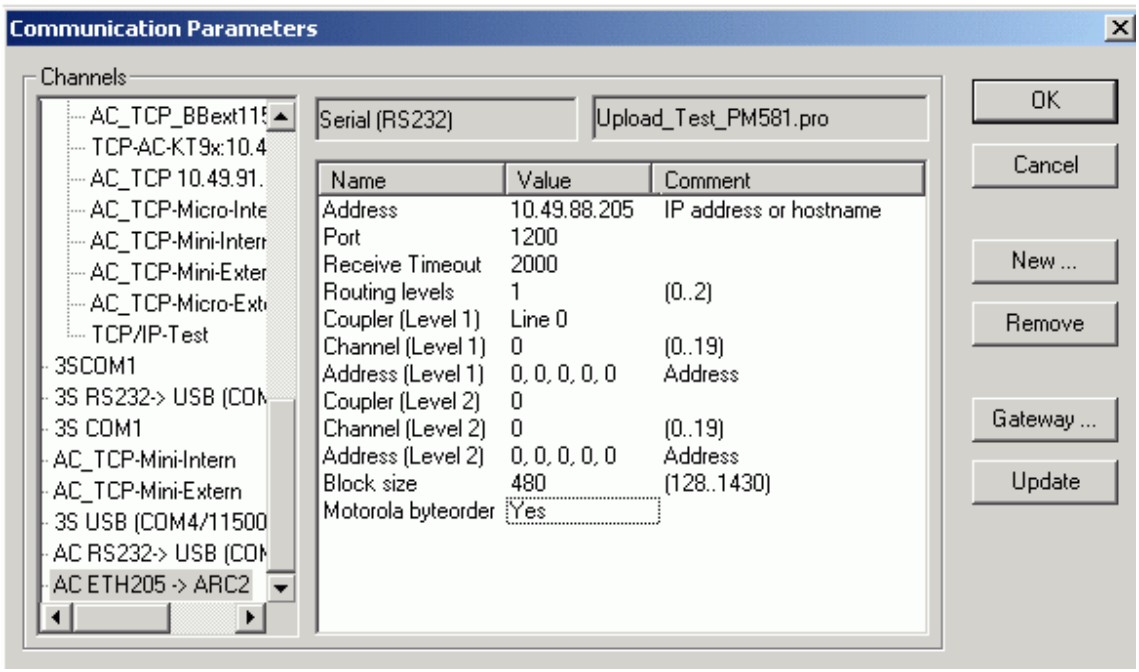
Routing is available as of PLC firmware version V1.3.

For controllers with Ethernet and ARCNET communication module, the PLCs connected via ARCNET can be programmed using the PLC Ethernet interface.



For each PLC connected via ARCNET, one gateway channel has to be defined. To do this, select "Online/Communication Parameters" and press the button "New" in the "Communication Parameters" window. In the appearing window, enter a name for the channel (for example TcpIp: PLC1:169.29.44.48 -> ARC_2) and select the driver "ABB Tcp/Ip Level 2 AC" from the device list.

For example, set the communication parameters as follows for the configuration shown above:



Parameter	Possible values	Description
Address	10.49.88.205	IP address of PLC 1
Port	1200	Port 1200
Timeout (ms)	2000	Timeout [ms] for response
Routing levels	1	Single-level routing
Communication Module (Level 1)	Line 0	Communication module for level 1 (internal: ARCNET)
Channel (Level 1)	0	Channel on Communication module level 1
Address (Level 1)	2, 0, 0, 0, 0	ARCNET node of the target PLC (Node 2)
Communication Module (Level 2)	0	No level 2
Channel (Level 2)	0	
Address (Level 2)	0, 0, 0, 0, 0	
Block size	480	Bytes per block: 128...1430
Motorola byteorder	Yes	

For the parameter "Communication Module (Level 1)", enter the slot where the ARCNET communication module "Line 0" is inserted (the ARCNET Communication Module is always the internal communication module).

The ARCNET communication module has only one communication channel. Thus, the "Channel" value must always be 0.

For the ARCNET communication module, 1 byte is required for the subscriber address (node). The address (Node=2) of the target PLC is entered to the first byte of the address byte.

The default value for the block size is 1430. If routing on ARCNET is required (and "large ARCNET packages" are enabled for the target PLC), the block size can be increased to 480 bytes. Values in the range of 227 .. 245 are not allowed.

1.6.5.4.3 AC500-specific PLC browser commands

Automation Builder provides IEC 61131 standard commands as well as AC500-specific commands.

Online help is available for all commands. The help information is displayed language-dependent by entering command "?" when operating in online mode. The command "?" lists all available firmware commands.

The commands listed in online mode can differ from the commands shown when pressing the button *[...]* as Automation Builder version and firmware version can differ.

Depending on the device, the PLC browser provides the following commands:

Command	Description	Implementation
?	Displays all implemented commands	Standard
mem	Memory dump of an area Usage: mem [from-addr] [to-addr]	Standard
memc	Memory dump relative to code area	Standard

Command	Description	Implementation
memd	Memory dump relative to data area	Standard
reflect	Reflect current command (for test purposes)	Standard
dpt	Displays the data pointer table	Standard
ppt	Displays the block pointer table	Standard
pid	Displays the project ID	Standard

pinf	Displays project information in the format:
	`pinf`
	`Address of Structure: 16#0013CF74`
	`Date: 4213949F`
	`Project Name: MODBUS_Test_BB.pro`
	`Project Title: Test MODBUS`
	`Project Version: V1.0`
	`Project Author: Test User`
	`Project Description:`
	`Test of serial interfaces`
	`End of Project-info.`
		Standard

Command	Description	Implementation

tsk	Displays the IEC task list with task information in the format:
	`tsk`
	`Number of Tasks: 1`
	`Task 0: Main program, ID: 1519472`
	`Cycle count: 45402`
	`Cycle time: 1 ms`
	`Cycle time (min): 1 ms`
	`Cycle time (max): 1 ms`
	`Cycle time (avg): 1 ms`
	`Status: RUN`
	`Mode: CONTINUE`
	`----`
	`Priority: 10`
	`Interval: 5 ms`
	`Event: NONE`
	`----`
	`Function pointer: 16#00601584`
	`Function index: 131`
		Standard

tskclear	Clears IEC Task information (cycle count & overall maximum and minimum cycle time)	Specific as of V2.0
startprg	Starts the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
stopprg	Stops the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
resetprg	Resets the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
resetprgcold	Resets the user program (cold)	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)

Command	Description	Implementation
resetprgorg	Resets the user program (origin)	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
reload	Reloads the boot project from user flash memory	Standard (not supported with CPU firmware V2.2 or higher)

getprgprop	Displays program properties in the format:
	`getprgprop`
	`Name: MODBUS_FBP_Test_BB.pro`
	`Title: Test MODBUS`
	`Version: V1.0`
	`Author: Test User`
	`Date: 4213949F`
		Standard

getprgstat	Displays the program status in the format:
	`getprgstat`
	`Status: Run`
	`Last error: Id 00000000 TimeStamp 000055F3 Parameter 00000000 Text`
	`Flags:`
		Standard

filecopy	File command copy	No
filerename	File command rename	No
filedelete	File command delete	No
filedir	File command dir	No
saveretain	In V1.0 and V1.1: Saves the RETAIN variables to the memory card. As of V1.2: Writes the RETAIN variables to RAM (same as retain save)	Specific
restoreretain	In V1.0 and V1.1: Restores the RETAIN variables from the memory card. As of V1.2: Restores the RETAIN variables from RAM (same as retain restore)	Specific

Command	Description	Implementation
setpwd	Sets the PLC password (required at logon!) Note: From CPU firmware V2.3.0 and higher, this command works as follows: ● If there is no password set, a password can be set with: setpwd <new_password> ● If a [new] password has been set, the old password must also be inserted.: setpwd <old_password> <new_password>	Standard (with CPU firmware to V2.3.0) Specific (with CPU firmware V2.3.0 and higher)
delpwd	Deletes the PLC password Note: From CPU firmware V2.3.0 and higher, this command works only if a password has been set. Also, you have to specify the old password to delete it, i. e. syntax is: delpwd <current_password>	Standard (with CPU firmware to V2.3.0) Specific (with CPU firmware V2.3.0 and higher)
plcload	Displays the PLC utilization (system + IEC + tasks + communication)	Standard

Command	Description	Implementation

rtsinfo	Displays the firmware information (version, driver) in the format:
	
	`rtsinfo`
	`rts version: 2.4.7.24`
	`OS version: SMX smxPPC 3.5.2`
	`uses IO driver interface`
	`rts api version: 2.408`
	`4 driver(s) loaded`
	`driver 1: AC500 CPU driver, device interface version: 2.403`
	`driver 2: AC500 I/O-BUS driver, device interface version: 2.403`
	`driver 3: AC500 COM driver, device interface version: 2.403`
	`driver 4: AC500 Coupler driver, device interface version: 2.403`
	`AC500 PM___(DISP) : V2.1`
	`AC500 PM___(BOOT) : V2.0.5, 2017-10-26 (Build: 9603, 13:55:09, Rel)`
	`AC500 PM___(FW) : V2.0.4, 2017-10-12 (Build: 9530, 14:30:50, Rel)`
		Specific

traceschedon	Enables task tracing	No
traceschedoff	Disables task tracing	No
traceschedstore	Stores task trace to RAM	No
fdir <path>	Show content of a drive or directory <path> (e.g. `fdir userdisk`, `fdir sdcard/userdata`)	Specific as of V2.1
fread <path>	Dump a file's content	Specific as of V2.1
fmove <path>	Move a file to a directory	Specific as of V2.1
mkdir <path>	Create a directory	Specific as of V2.1
deldir <path>	Delete an empty directory	Specific as of V2.1
rndir <old path> <new path>	Rename a directory	Specific as of V2.1

Command	Description	Implementation
ipaddr	Sets the IP address of the CPU	No
basetick	Sets the basetick to µs	No
diagreset	Resets the diagnosis system	Specific
diagack all	Acknowledges all errors	Specific
diagack x	Acknowledges all errors of the class X (with X= 1...4)	Specific

diagshow all	Shows all errors in the format:
	`diagshow all`
	`--- All errors ---`
	`State Class Comp Dev Mod Ch Err`
	`0152502216`
	`active and acknowledged 4 9 22 31 31 8`
	`1970-01-01 00:00:08 occurred`
	`disappeared`
	`1970-01-01 00:00:15 ack.`
	`0152369165`
	`active not acknowledged 49 2031013`
	`1970-01-01 01:19:12 occurred`
	`- disappeared`
	`- ack.`
	`--- end ---`
		Specific

time	Displays and sets the time of the real-time clock. If no battery is inserted in the PLC and the control voltage is switched on, the PLC clock is set to "01. January 1970, 00:00".	Specific
date	Displays and sets the date of the real-time clock. If no battery is inserted in the PLC and the control voltage is switched on, the PLC clock is set to "01. January 1970, 00:00".	Specific
batt	Polls the battery status	Specific

Command	Description	Implementation
sdappl	Saves the boot project to the memory card	Specific
sdclone	Copys the user program and the user data to the memory card.	For AC500-S only!
sdfunc	Displays and changes the memory card function	Specific
sdsys	Save firmware to memory card	Specific
sdboot	Updates the bootcode from the memory card	Specific
sddisplay	Updates the MMI firmware from the memory card	Specific
sdfirm	Updates the firmware from the memory card	Specific
sdcoupler x	Updates the firmware of Communication Module x from the memory card	Specific
cpuload	Displays the CPU load (current, min., max., average)	Specific
delappl	Deletes the user program in the user flash memory	Specific
retain	Saving and restoring the RETAIN variables: retain clear -> Clears all RETAIN variables retain save -> Saves the RETAIN variables to the RAM disk retain restore -> Restores the RETAIN variables from the RAM disk retain export -> Exports the RETAIN variables from the RAM disk to the memory card retain import -> Imports the RETAIN variables from the memory card to the RAM disk	Specific as of V1.2

Command	Description	Implementation
persistent	Saving and restoring the persistent area %R area: persistent clear -> Clears the %R area persistent save -> Saves the buffered %R area to the RAM disk persistent restore -> Restores the buffered %R area from the RAM disk persistent export -> Exports the buffered %R area from the RAM disk to the memory card persistent import -> Imports the buffered %R area from the memory card to the RAM disk	Specific as of V1.2
cfginfo	Print expected and active configuration version. This is for internal use.	Specific
hashappl	Hash the user program	Specific
io-bus stat	Displays the I/O bus statistic	Specific
io-bus desc	Displays the I/O bus configuration	Specific
com protocols	Displays the protocols available for the serial interfaces	Specific
com settings	Displays the serial interface settings	Specific
coupler desc	Displays information on the communication module interfaces (type, firmware, serial number, date)	Specific
coupler settings	Displays the current communication module settings, for example, IP address and socket assignment	Specific as of V1.2
ping	Ping a address, usage: ping <ipaddr> <couplerid> <ms>	Specific as of V2.1.3
reboot	Reboots the PLC (IEC 61131 performs a logout when restarting or logout possible up to 3 seconds after command input) (This command is not available for CM574-RS as of firmware revision V2.1.x)	Specific

Command	Description	Implementation
diskcfg	Access to drive maintenance Command Syntax: `diskcfg [option] [drivename]` Command Options: ● `unlock` : unlock a drive for writing MBR or formating ● `lock` : lock drive again ● `writembr` : write clean MBR (unlock required) ● `format` : write clean file system (unlock required) ● `settings` : show drive configuration details ● `desc` : show drive overview ● `help` : show this help ● `[none]` : no option shows this help Available Drives (not all commands are supported on all drives): ● Flash memory ● Memory card ● RAM disk ● Userdisk ● Flash disk ● SRAM disk	Specific as of V2.1

Command	Description	Implementation

proddata	Shows Production Data of PLC
	`proddata`
	`Production data`
	`---------------`
	`Ident : 1SAP123456R0001`
	`Index : B1`
	`Type : PM___`
	`Date : 0447`
	`BA-Inst : 1S120`
	`Factory : 05`
	`Year : 17`
	Serial No. : 00007134
	MAC-Addr : "
		Specific as of V2.0

confdata	Shows Configuration Data of PLC It is possible to save and load PLC specific configuration of any kind with the function blocks (from SysInt_AC500_V10.lib): *Chapter 1.5.4.19.2.3 “CPU_CONFIG_READ” on page 1508* *Chapter 1.5.4.19.2.4 “CPU_CONFIG_WRITE” on page 1511* The AC500 firmware also uses this INI file for settings like IP addresses.	Specific

1.6.5.4.4 Watch- and recipe manager

General information

Since CODESYS 2.3.9.25 (Control Builder Plus V2.1), the programming environment is extended by the Watch- and Recipe Manager which provides more comfortable online debugging functions.

For general information, see *Chapter 1.4.1.4.9 “Watch- and recipe manager” on page 395*.

Function

In the Watch- and Recipe Manager (Resources tab of the Object Organizer) the current values of specified variables can be viewed in so-called "watch lists (Monitoring).

Further on the variables listed in a watch list can be preset with constant values and this set of values, named "Recipe" be transferred to the PLC. Also the current values of the variables of a watch list can be read from the PLC to the Watch and Recipe Manager as a preset/recipe. In this context regard the possibility to save recipes in files and to reload them to the Recipe Manager when required. For further information on the usage of recipes see *Chapter 1.4.1.4.9.2 “Creating watch lists, recipes” on page 397*.

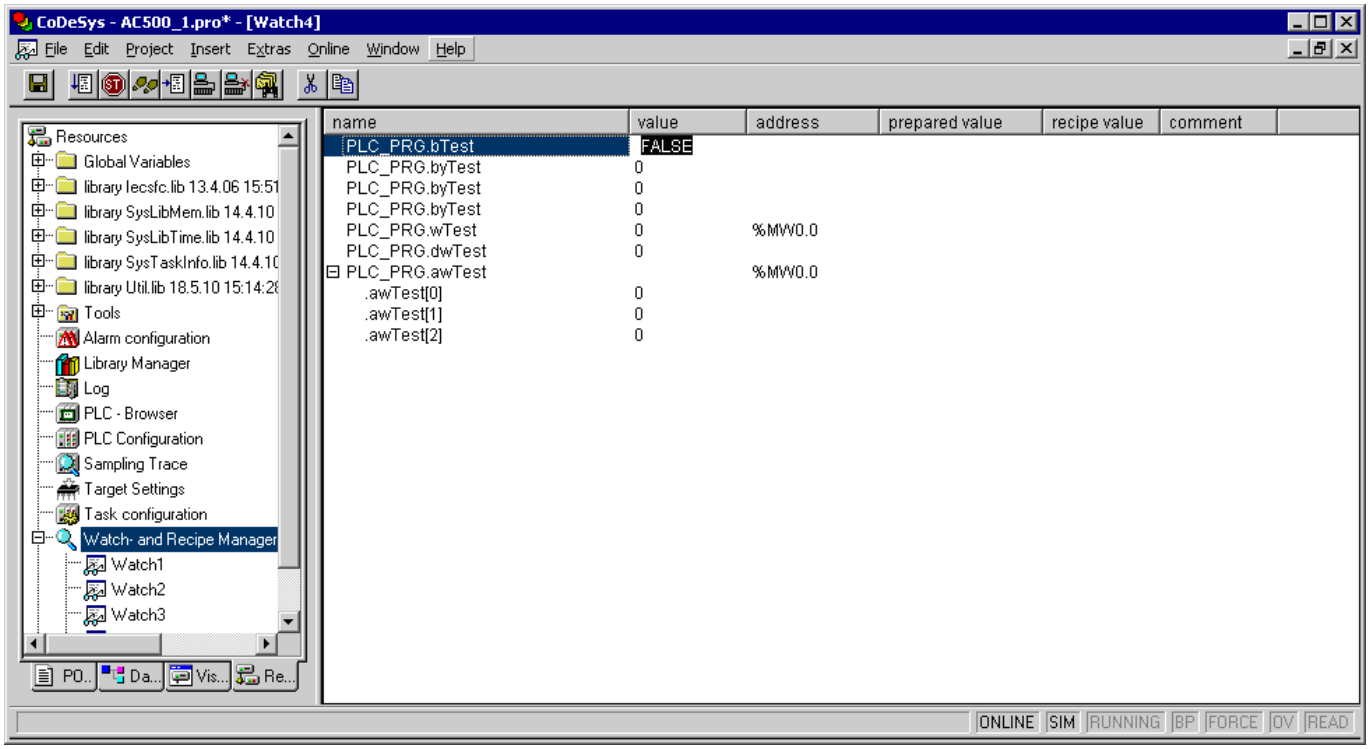
In online mode watch lists as well can be used to Write (*Chapter 1.4.1.2.6.16 “'Online' 'Write values'” on page 286*) and Force (*Chapter 1.4.1.2.6.17 “'Online' 'Force values'” on page 287*) variables.

All these functions for example can be used for logging and setting of control parameters.

Watch- and recipe manager extensions

Activate the Watch- and Recipe Manager with “ Project ➔ Options ➔ Tab Desktop ➔ Tabular watch editor”.

The Watch window displays the values in a table oriented view.



The table display is used in both offline and online mode. It is not necessary to switch on/off monitoring mode to insert or delete a variable. It is possible to insert and append a variable in online and offline mode at any time. In addition it is also possible to add an address range.

The online value is displayed in CODESYS in online mode only.

A "+" will appear in the first column in case of function block instances, structured variables and arrays. It allows to expand and collapse them.

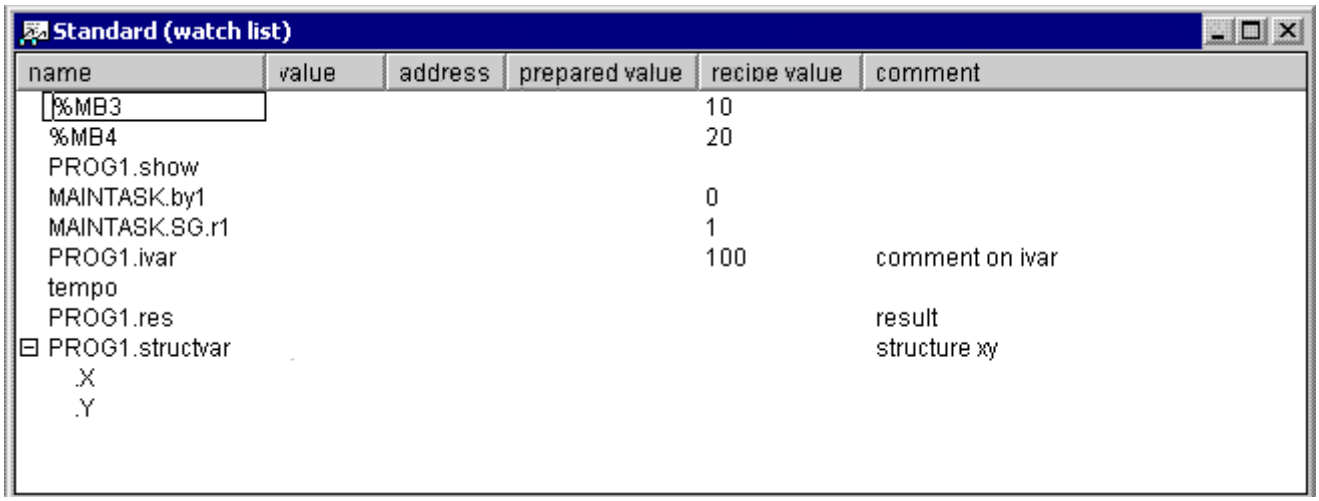
The editor allows manually to

- add watch entries
- delete watch entries
- change the name of a watch entry.

In online mode, the prepared value can be defined with a dialog. More than one prepared value can be selected. It is possible to enter the same value for all of them. One or more prepared values can be sent to the PLC via *"Online ➔ Write Variables or Online ➔ Force variables"*.

Tabular editor

Each watch list is viewed in a separate tabular editor window and multiple windows can be opened at the same time. In this case the available watch lists will be shown as entries in the *"Resources"* tab and can be opened by double-click.

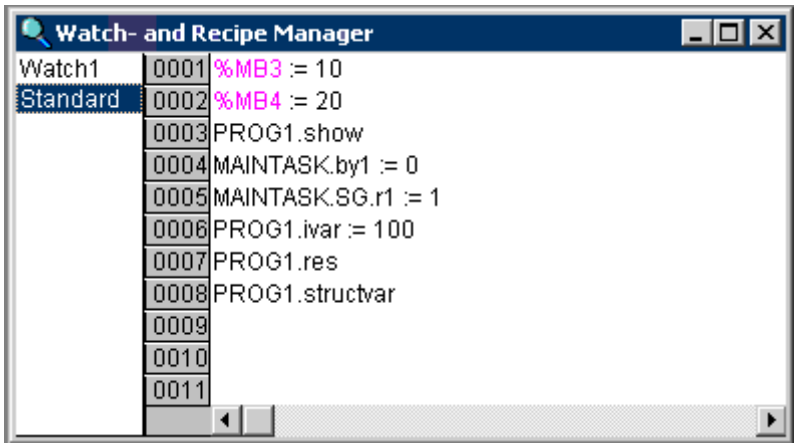


- name: Here a variable identifier according to the following syntax must be entered of an address in standard format:
  <POU name>.<variable name>
  In case of global variables the POU name is dropped. The variable name can be multilevel. Addresses can be entered directly (e.g. "%IB0.0").
  Example for a multilevel variable:
  PLC_PRG.Instance1.Instance2.Structure.Component
  Example for a global variable:
  globalvar.component1
- address, comment: As specified in the declaration of the variable.
- value: In online mode here the current value of the variable is displayed (Monitoring).
- recipe value: Here a value can be entered, which will be transferred to the PLC when command *Chapter 1.4.1.4.9.14 "'Extras' 'Write Recipe'" on page 401* is applied on the whole watch list. The recipe values of all variables of the list can be replaced by the current values from the PLC by using command *Chapter 1.4.1.4.9.15 "'Extras' 'Read Recipe'" on page 402*.
  In case of function block instances and structured variables a plus respectively minus sign appears in front of the name in the first column. It serves to expand resp. collapse the list of components. For function block variables the context menu is extended by the items 'Open function block' and 'Open instance'.

By a double-click on a non-editable position within the editor window, the table gets adapted to the window width and the column widths get optimized.

1-Window-editor

There is only one bipartite editor window, in the left part of which you find all available watch lists. For the list currently selected the right part of the window shows the associated variables. This editor view is opened via object 'Watch- and Recipe Manager' in the *"Resources"* tab in the Object Organizer. The watch variables are entered line by line and a recipe value can be assigned to each by ":=".



Adding variables to watch lists is possible in offline mode by typing, or in online mode directly out of the POU editors. See ↳ *Chapter 1.4.1.4.9.2 "Creating watch lists, recipes" on page 397*

A 'Cross Reference List' can be called directly from a watch list, when one of the watch variables is selected. In this case the command ↳ *Chapter 1.4.1.3.7 "Show cross references" on page 295* in the Extras menu or in the context menu is available.

Adding variables to watch window from language editors

If one or more variables or elements are selected in a language editor or the declaration editor, the following additional menu items will appear in the *"Extras"* menu:

- *"Into new watch list"*
- *"Add to watch list"*

1. Select *"Into new watch list"* to automatically create a new watch list for the selected variables.

2. Select *"Add to watch list"* to open a submenu containing a list of all existing watch lists. After that, select the desired watch list to append the variables to that watch list.

If an area with more than one variables is selected in a language editor (it is dependent on the language editor how multiple selections of elements is realized, see below), all selected variables will be added to the watch list.

It is not necessary to enable/disable monitoring active manually.

To enable/disable monitoring active manually (to add a variable manually), press *[ALT]* + *[X]* + *[M]*.

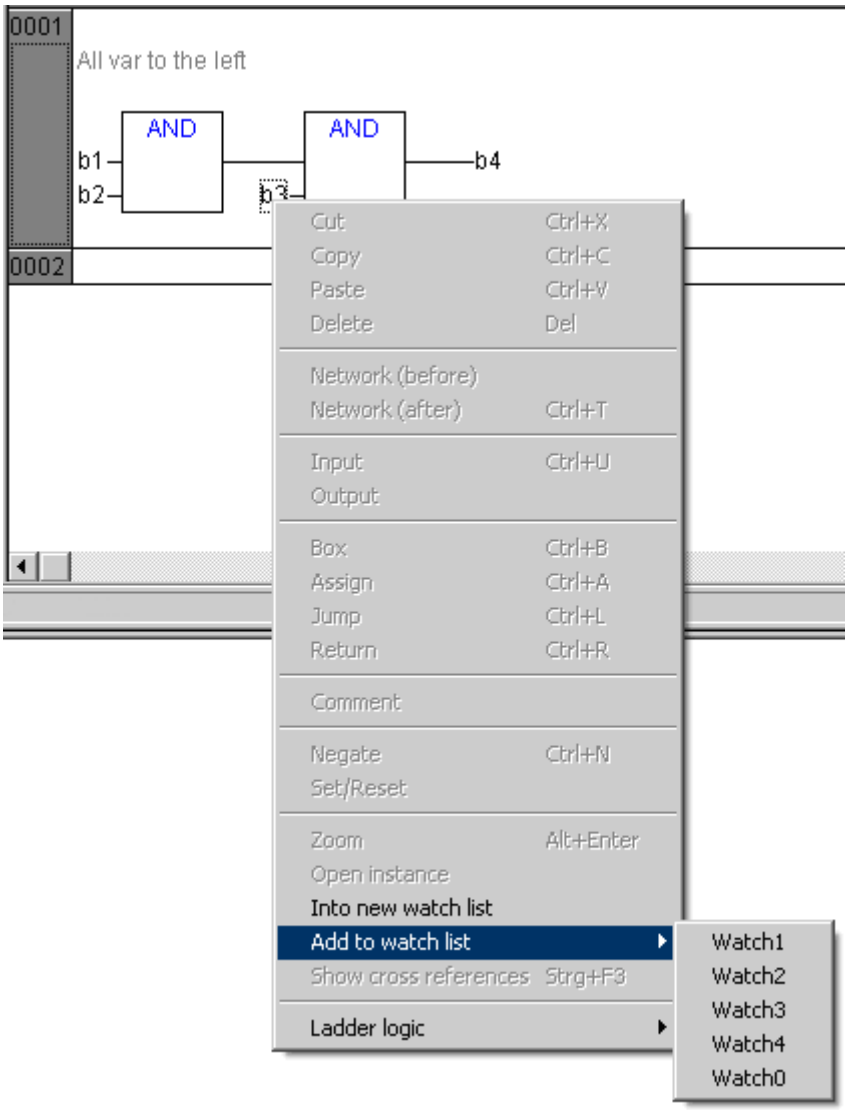
These menu items are available in online mode only, if the full instance path of a variable inside a function block is known.

The possibilities of multiple selections are explained in the following sections.

Function Block Diagram (FBD)

The FBD editor has a tree-oriented selection. By selecting one box in the tree, all elements to the left of it are selected, too.

In the following example, variable b3 is selected.

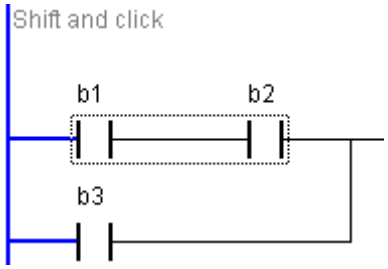


Right-click onto the selected variables to open the context menu and adding them to a watch window.

Ladder Diagram (LD)

Press *[SHIFT]* and click onto the desired elements to do a multiple selection. Only consecutive elements can be selected for a multiple selection.

In the following example, variables b1 and b2 are selected.

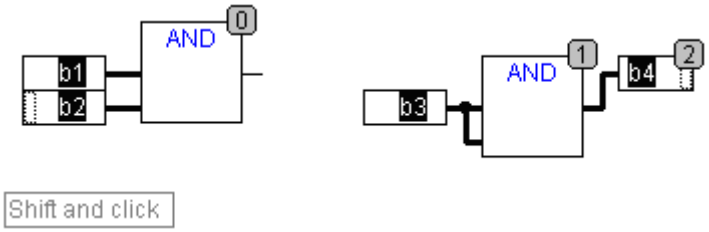


Right-click onto the selected elements to open the context menu and adding them to a watch window.

Continous function chart (CFC)

Press *[SHIFT]* and click onto the desired elements to do a multiple selection.

In the following example, variables b2 and b4 are selected.



Right-click onto the selected variables to open the context menu and adding them to a watch window.

Instruction list (IL)

In Online Mode, the IL editor window displays the IL text on the left side and the variables and their online values on the right side of the window.

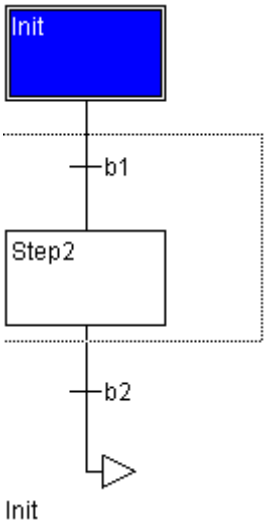
To select multiple variables, mark the text on the left side of the window.

Right-click on the text to open the context menu and adding them to a watch window.

Sequential function chart (SFC)

Multiple selection is possible with consecutive elements.

Press *[SHIFT]* and click onto the desired elements to do a multiple selection.

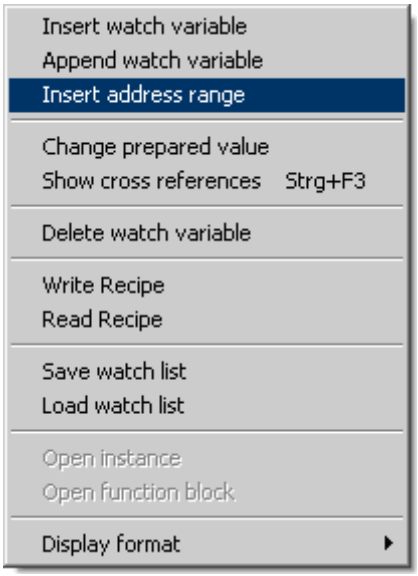


Right-click onto the selected variables to open the context menu and adding them to a watch window.

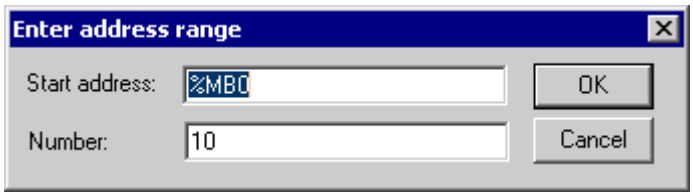
Adding an address range to the watch window

☑ To add an address range to a watch list, proceed as follows:

1. Right-click the Watch Window and select *“Insert address range”*.



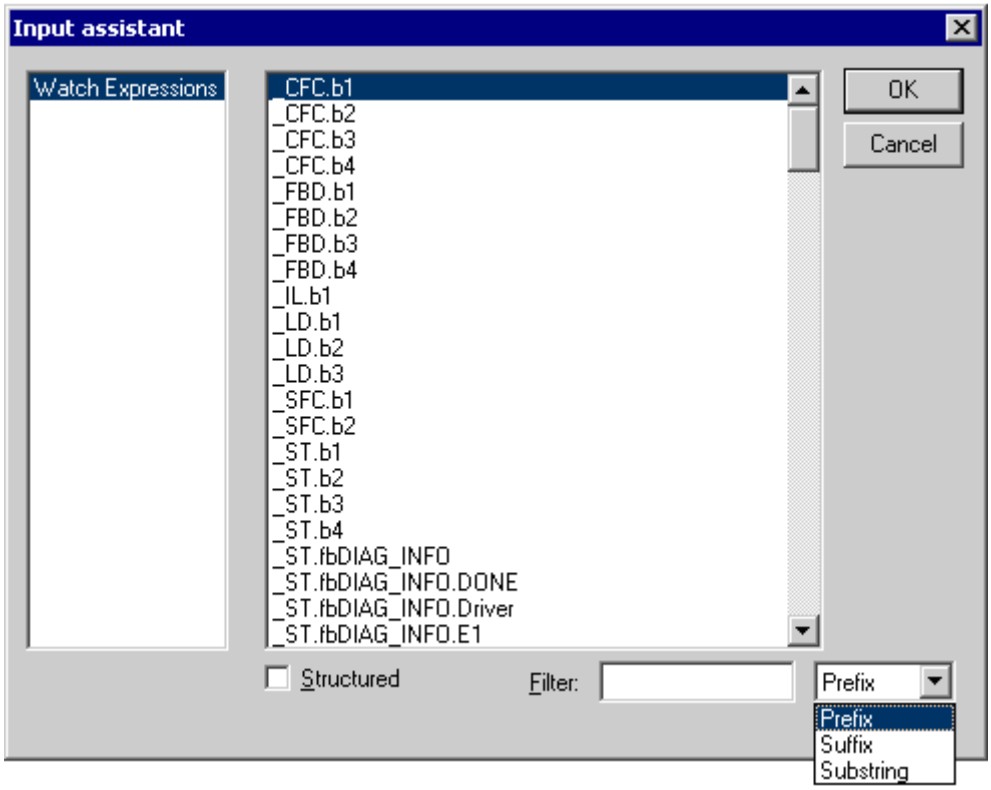
2. In the *Enter address range* dialog box enter the name or address of the variable and the number of element you want to insert.



3. Confirm the dialog box with *[OK]*.

Filtering within input assistant

1. Press *[F2]* within the Watch Window to open the *Input assistant* dialog.



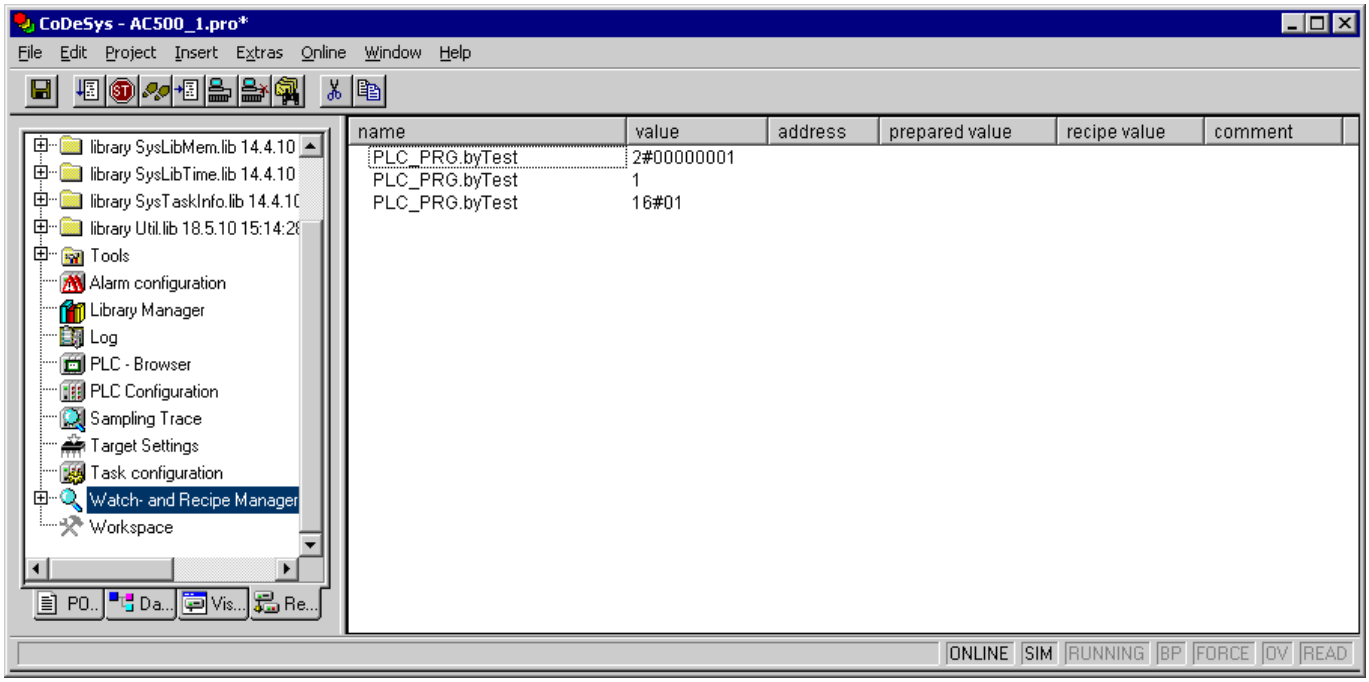
   ⇨ If check box *"Structured"* is enabled, the variables are displayed in a structured way by global variable lists, POUs, libraries and folders.

   If check box *"Structured"* is disabled, all variables are displayed in one list.

2. Enter the filter text in text field 'Filter'. Select one of the 3 filter criteria from the pull-down menu:
   - *"Prefix"*: Beginning of the variables is compared with the filter text.
   - *"Suffix"*: Ending of the variables is compared with the filter text.
   - *"Substring"*: Complete variable name is compared with the filter text.

Defining display format for each watch variable separately

With an additional column in the Watch Window table, the display format of integer variables can be selected. A drop-down list allows to select between decimal, hexadecimal, binary and default display format. This selection can be made for every variable separately. One variable at a time can be selected to change the display format.



Default format means that the display format of the variable is defined by the global setting.

1.6.5.4.5 Cross-reference list in watch- and recipe manager

General information

In this chapter extended features for the cross-reference list are described.

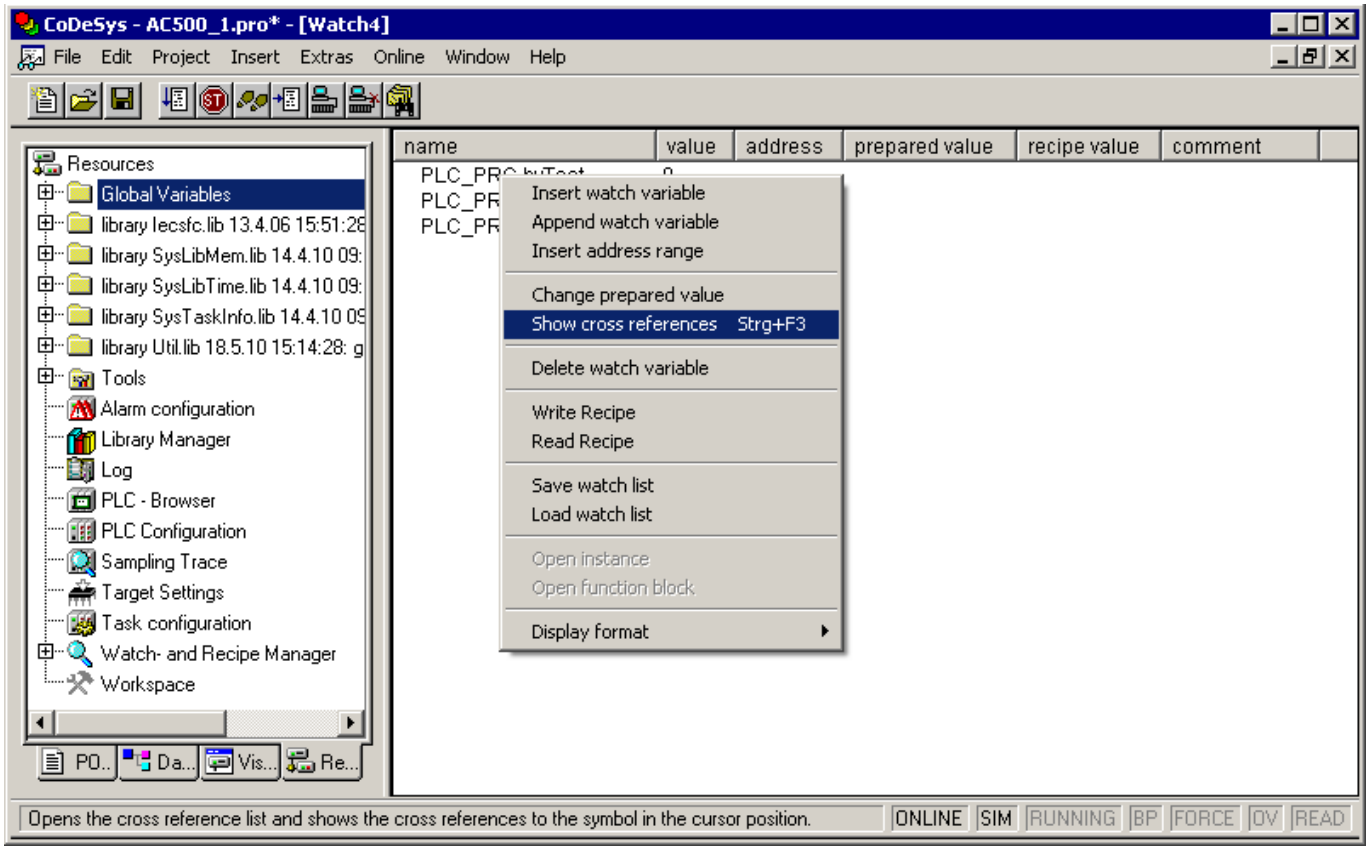
Extended Features:

- Open *"Cross Reference List"* from Watch Windows
- Open *"Cross Reference List"* from Language Editors
- Including Visualizations into *"Cross Reference List"*
- Including arrays, structures and addresses into *"Cross Reference List"*

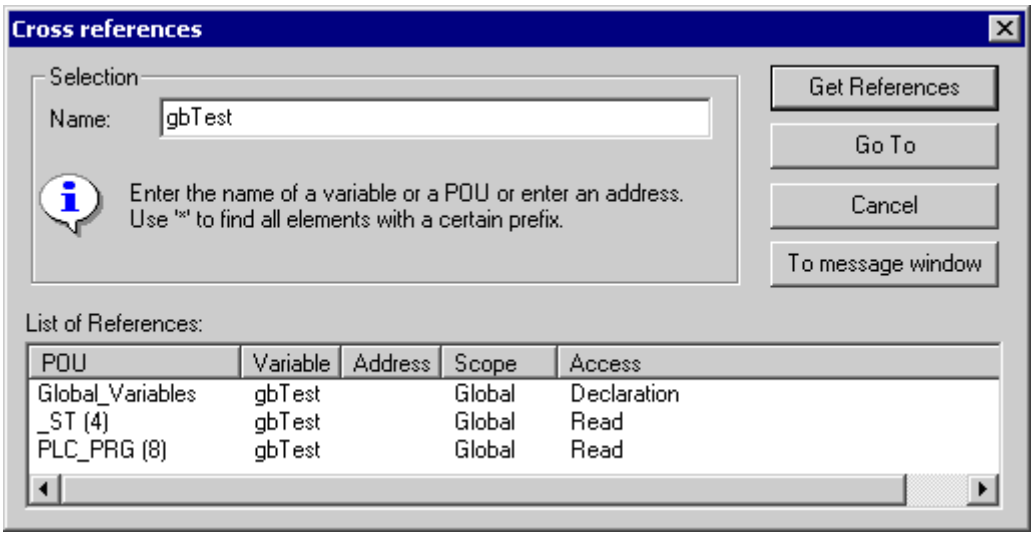
Open *"Cross Reference List"* from watch window

To open the cross-reference list from Watch Window (possible in Online and Offline Mode), proceed as follows:

1. Select the desired variable, e.g. PLC_PRG.ByTest.



2. Right-click on the variable and select Show cross references in the context menu.



⇨ The Cross references window appears which displays the full path names.

Open *"Cross Reference List"* from language editors

To open the cross-reference list from a language editor, proceed as follows:

▷ Right-click on the variable in the language editor window and select Show cross references.

⇨ The Cross references window appears which displays the full path names.

The following language editors are supported:

- FBD
- LD
- CFC
- ST
- IL
- SFC

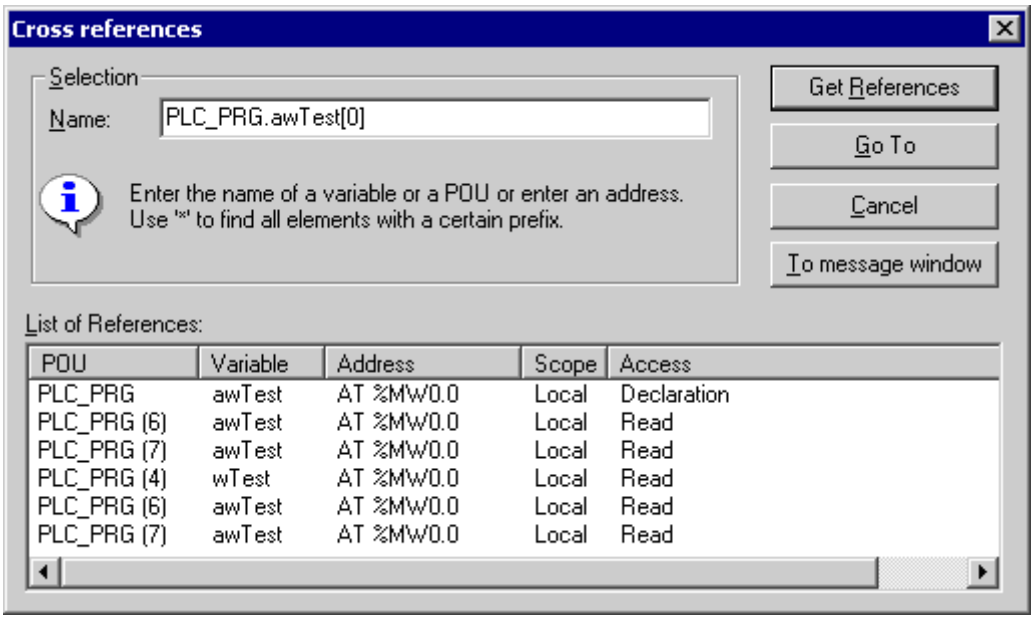
Cross references in visualizations

The Cross reference window will display the usage of a variable also in a visualization. Visualization can use variables in read- and write access.

The visualization will be shown in the POU column of the Cross reference window. Information will be added to indicate it is a visualization. For example: vis1 (VIS).

Cross references for arrays, structures and addresses

The Cross references window displays the usage of variables that are within the address range of a structure or array.



1.6.5.4.6 Reference to libraries

Library configuration is described in the chapter *Chapter 1.5 “Libraries and solutions” on page 735*.

1.6.5.4.7 Programming in C code

Preface

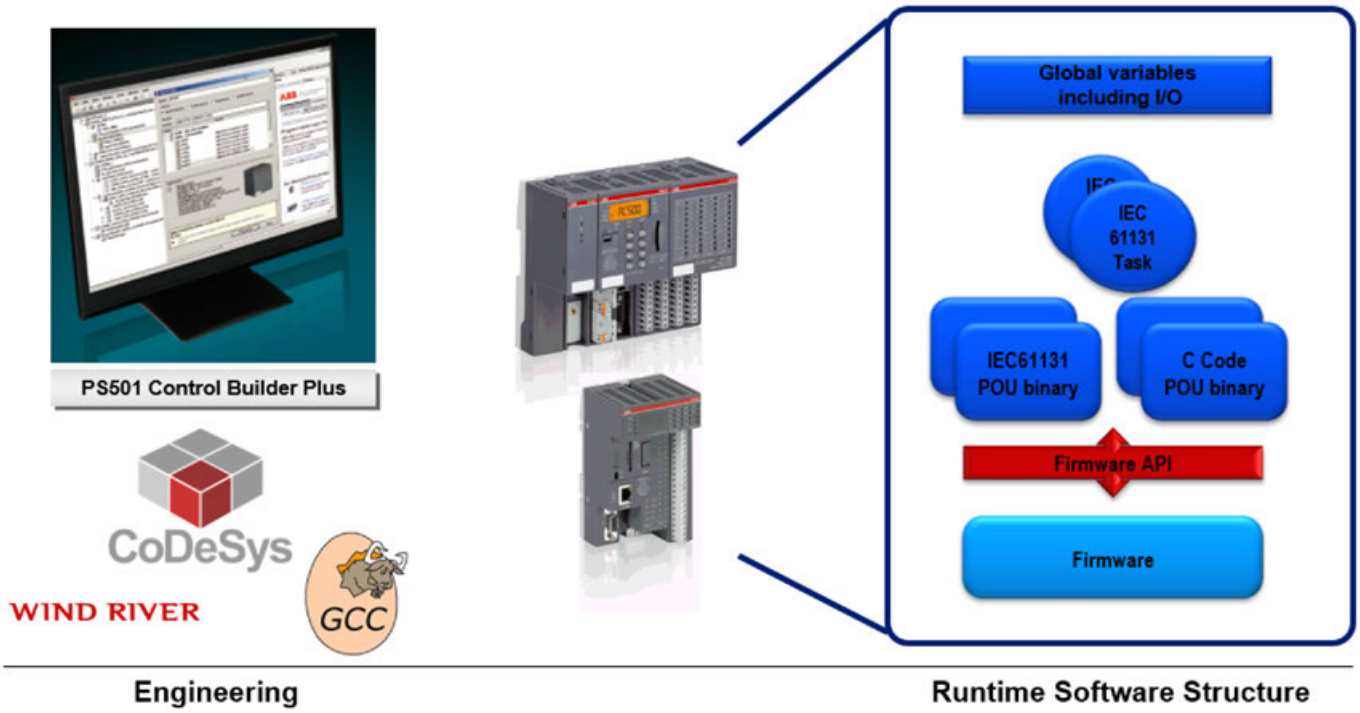
The user shall be able to implement custom application parts in C or C++ code and call them from IEC application. This document shows how to create a simple application with parts of it written in C or C++ code.

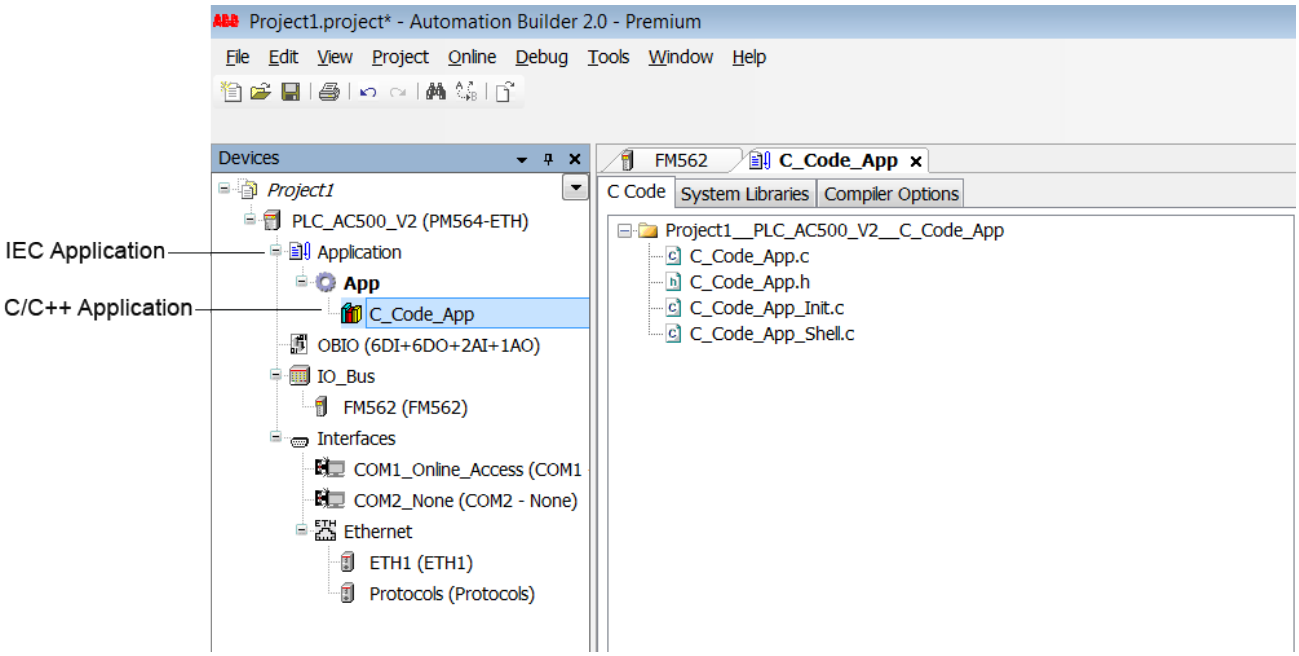
Table 754: Abbreviations

Abbreviation	Definition
OBJ	**OBJ**ect file
FW	**F**irm**W**are
PLC	**P**rogrammable **L**ogic **C**ontroller
ST	**S**tructured **T**ext
AB	**A**utomation **B**uilder
POU	**P**rogram **O**rganization **U**nit

Overview

C/C++ application is part of IEC application and hence seamlessly integrated into Automation Builder engineering tool.





Key facts:

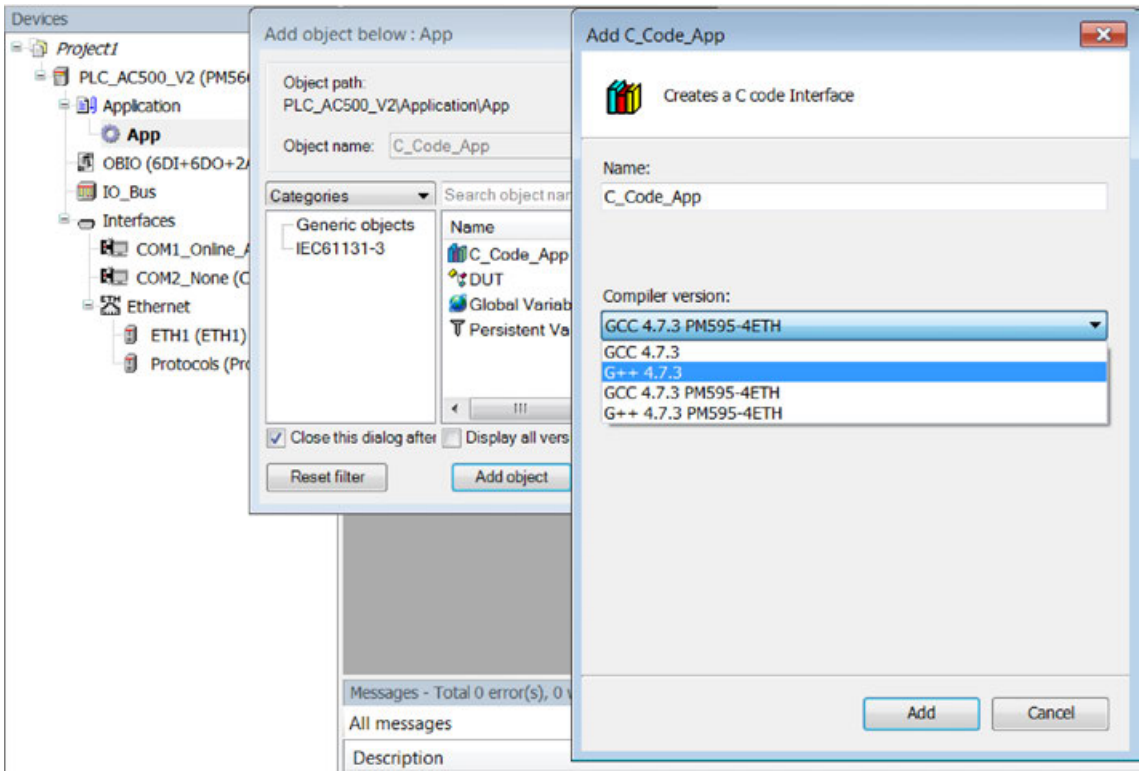
- AC500 and AC500-eCo POUs (Program Organization Units) can be implemented in IEC61131-3 or C/C++ code.
- C/C++ code can use C/C++ standard libraries and can access the PLC functions e.g. floating point calculations, memory allocation, µs-resolution, real-time clock, file access. ↳ *Chapter 1.6.5.4.7.4.2 “Firmware Application Programming Interface (FWAPI)” on page 6254*
- C/C++ compilation with target CPU specific optimization, like FPU. ↳ *Further information on page 6264*
- Application libraries can be implemented in C/C++. As only the object files need to be shipped to the customer, knowledge within the code is protected. ↳ *Chapter 1.6.5.4.7.4.5 “Binary Deployment of C/C++ Application” on page 6257*

How to Create a C/C++ Application

This chapter will guide you through the necessary steps to create an AB project using a simple POU implemented in C/C++.

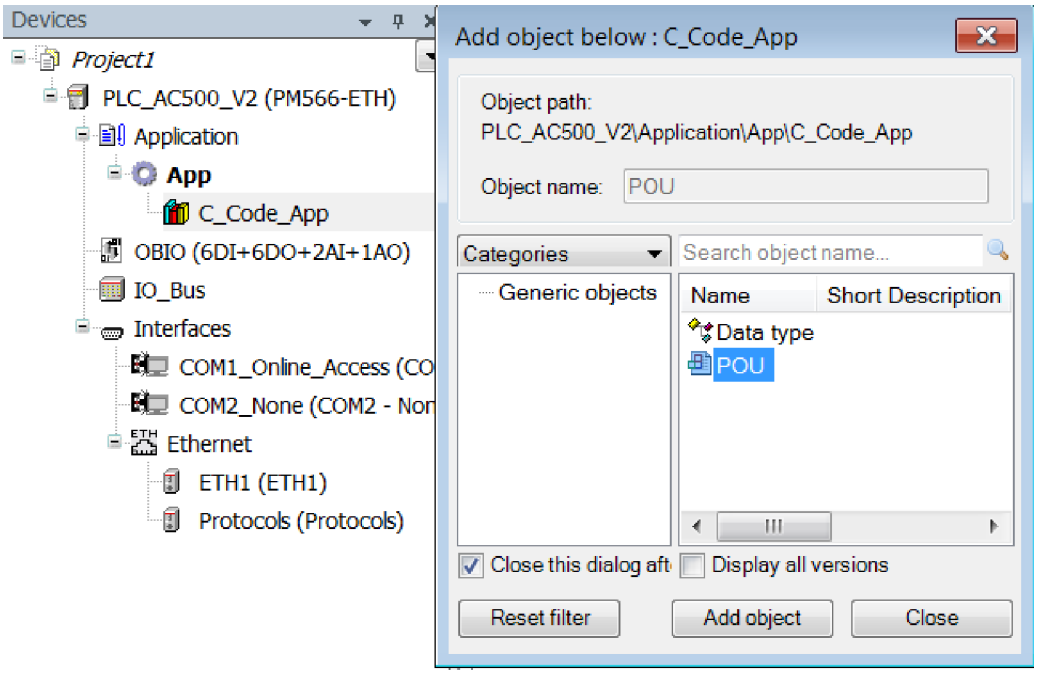
Creating a “Hello World” Application

1. In the tree view right-click on *“App”* node and add a new object.

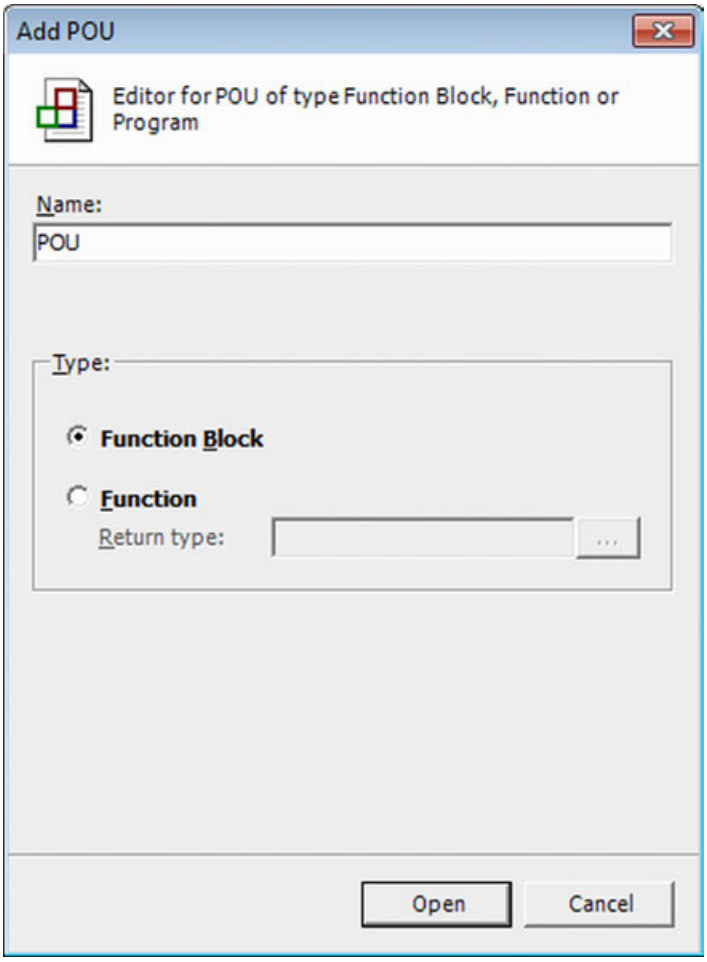


2. In the following dialog select *“C_Code_App”* and click *“Add object”*.

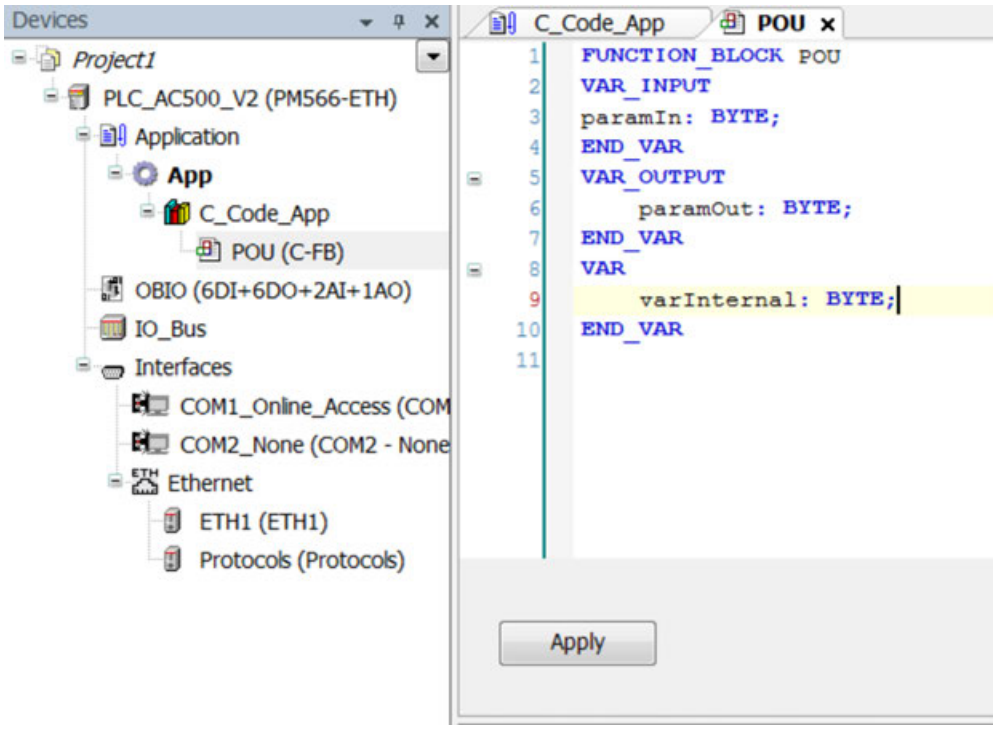
   Select the compiler to be used for the new C/C++ application and click *“Add”*.



3. In the following dialog select *"POU"* and click *"Add object"*.

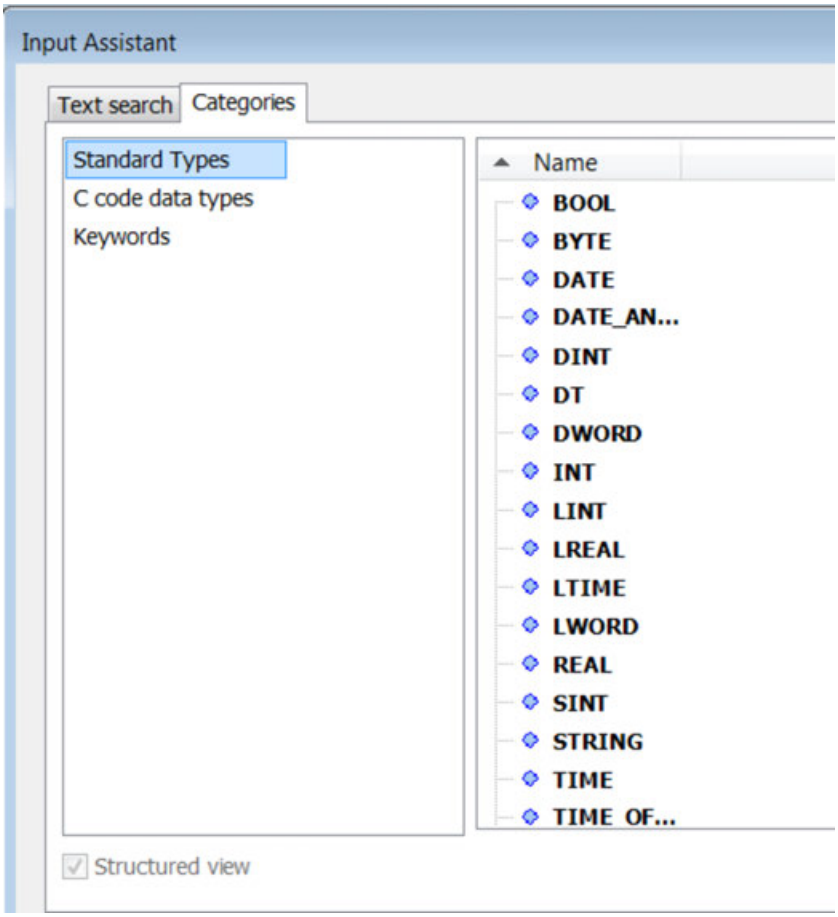


4. Select the type for the POU object and click *"Open"*.

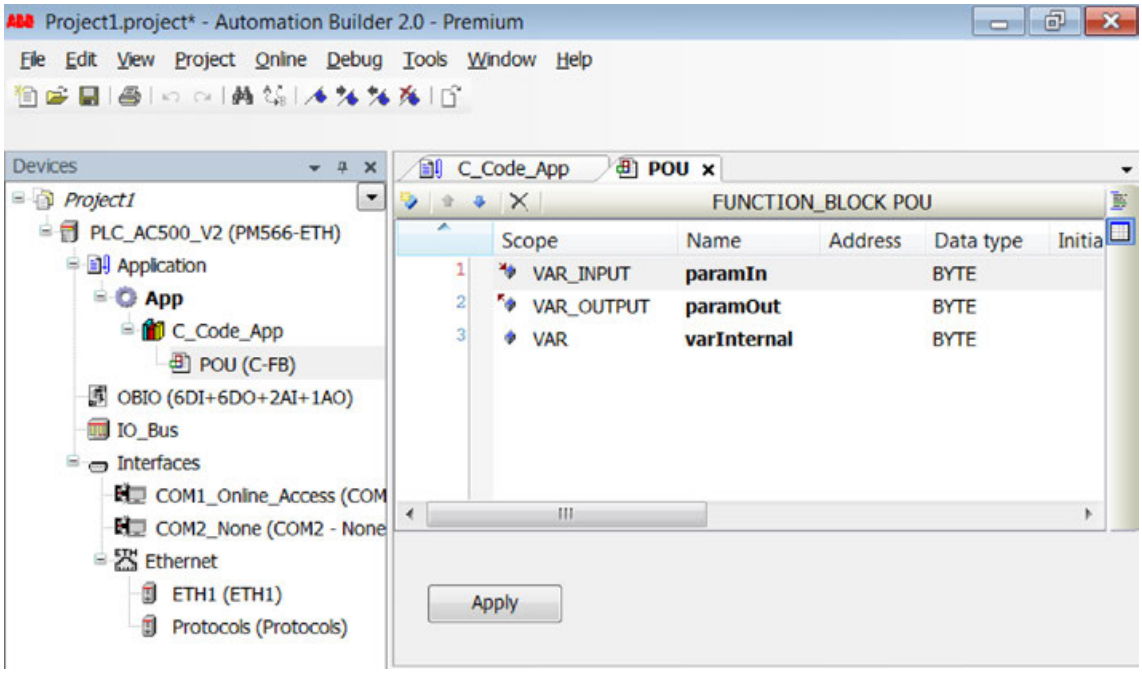


5. Enter a parameter and variable declaration.

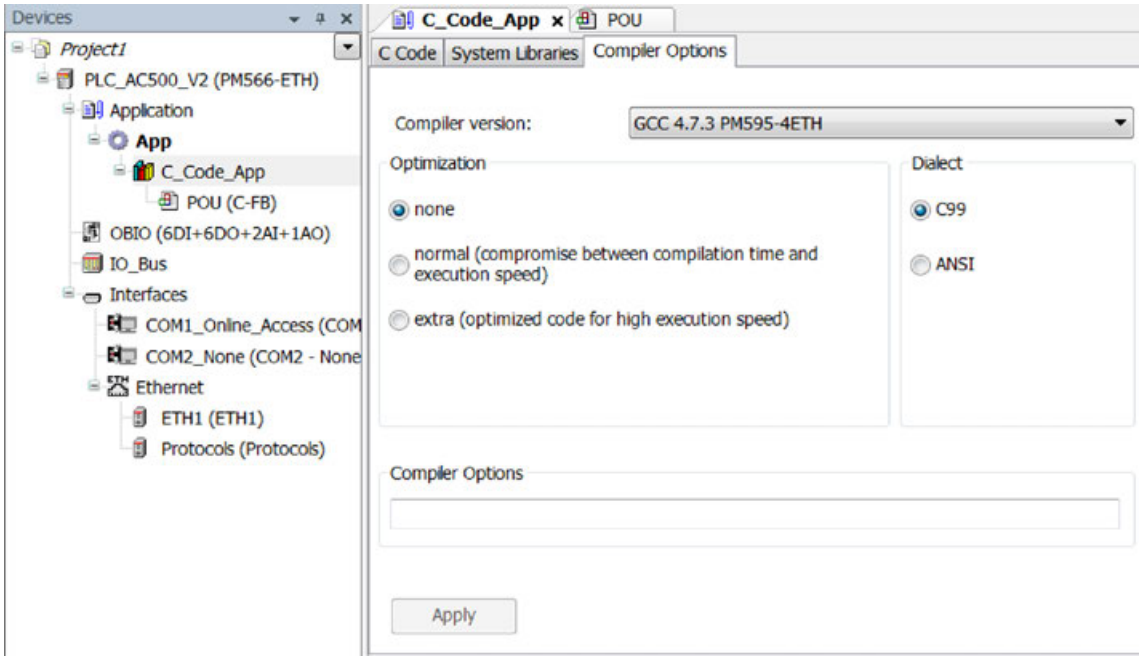
6. Right-click at the desired code position and select *"Input Assistant"* in order to define the data type:



7. Then, click *"Apply"* to update the interface. As an alternative the interface can be displayed tabularly. Tabular view is a comfortable way to edit parameters:

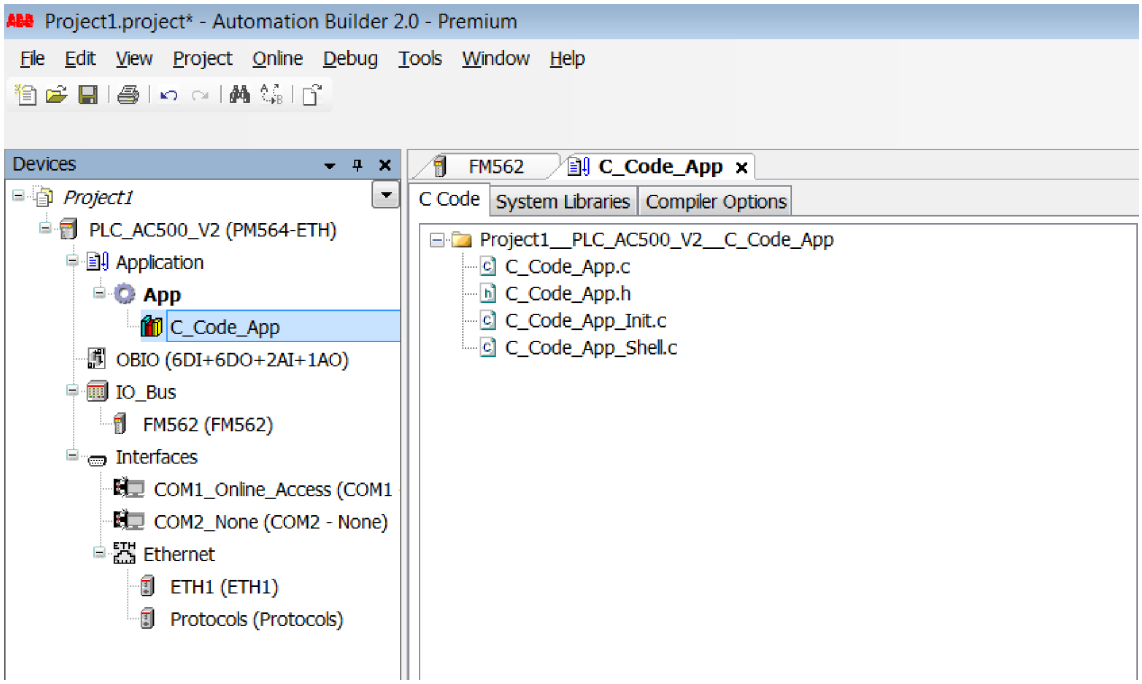


8. On the *"C_Code_App"* tab open the *"Compiler options"* tab. Select a compiler and configure compiler settings for the C/C++ application. These settings enable the compiler to interprete and process the source files listed on the *"C-Code"* tab.



9. Open the *"C-Code"* tab. The following files are listed:

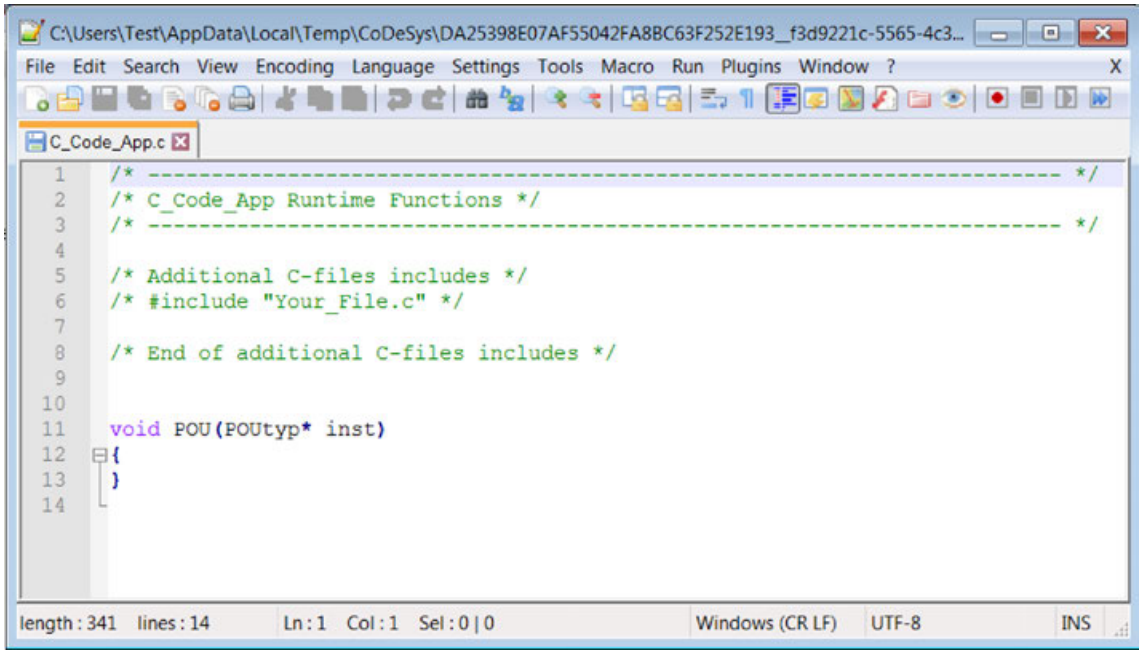
- *C_Code_App.c*: Implementation of POU.
- *C_Code_App.h*: C/C++ Interface of POU.
- *C_Code_App_Init.c*: Initialization of POU variables and parameters.
- *C_Code_App_Shell.c*: Main module.



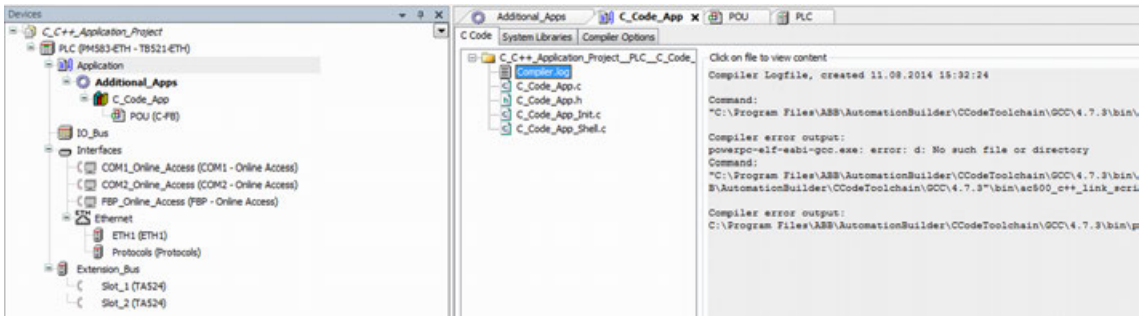
⇨

Double-click the Folder icon to open the directory in Windows Explorer.

10. To edit the POU implementation double-click or right-click a file and select *“Open...)”*. The *.c and *.h files are opened in your operating system’s default editor.

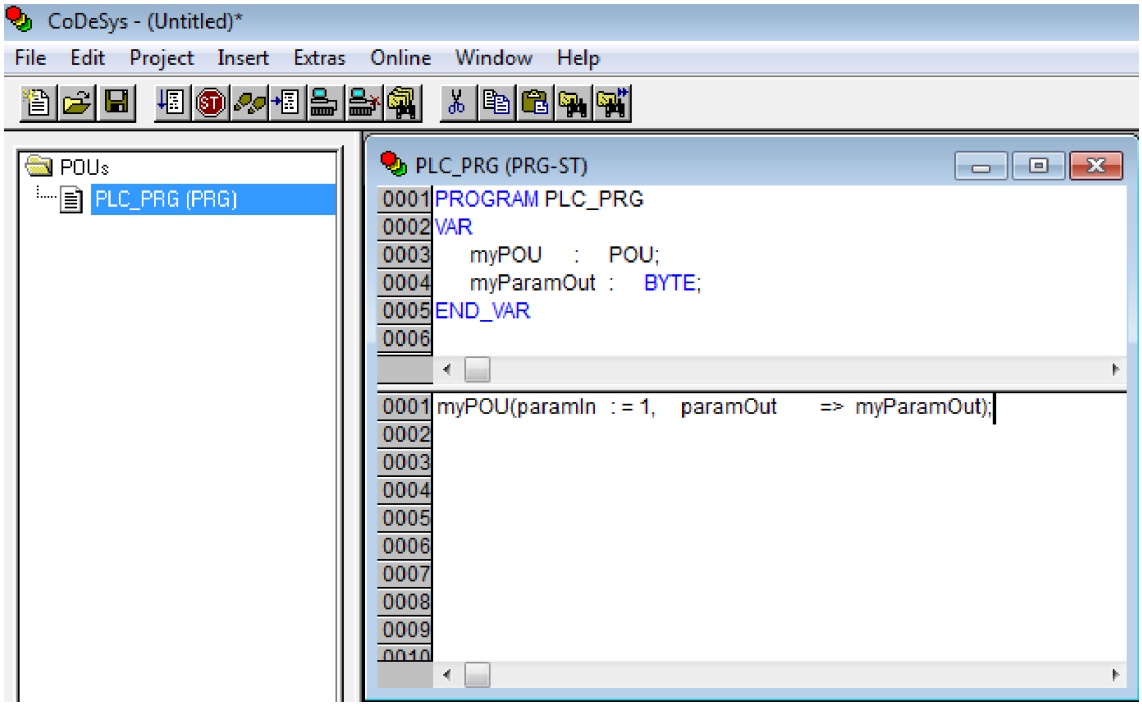


11. Save your settings and click *“Compile”* on the right side of the screen. The object files are generated from source code. Compilation errors are reported within a message window. Details on the error can be found in the *compiler.log* file.



12. Save your project. In the tree view double-click *“Application”*. CODESYS is started with C/C++ interface. From the *“C_Code_App”* node an external library is generated. This library can be found in CODESYS 2.3 in the Library Manager.

13. Call the C/C++ code POU from your IEC application.



⇨

> *In order to lookup C/C++ POUs, press F2 and open section "Standard Functions / Standard Function Blocks" .*

14. Build the project via *"Project ➔ Build"*: C/C++ application parts are linked together with IEC application.

    ⇨ Application is ready for download.

Interface between C/C++ and IEC Code

C/C++ applications may contain POUs as well as data types. Declaration and definition of stubs both are automatically done by Automation Builder when a POU or data type is created by the user. Following sections show how this is done in detail.

POU Type Function

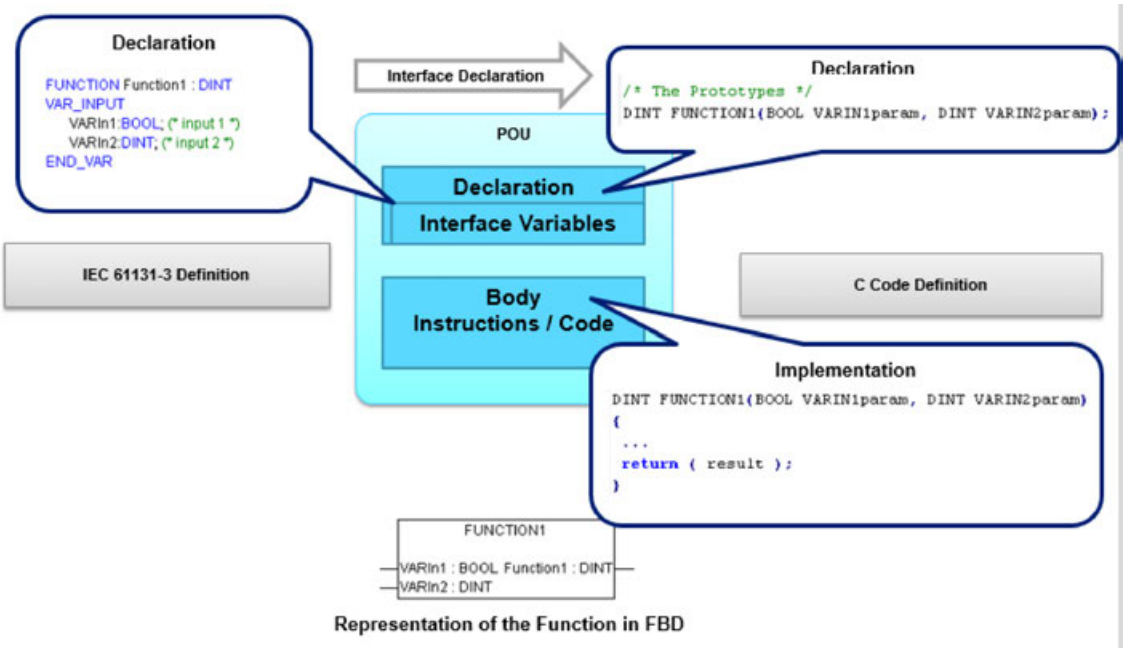


Fig. 1166: Interface between C/C++ and IEC code: POU type function.

POU type function block

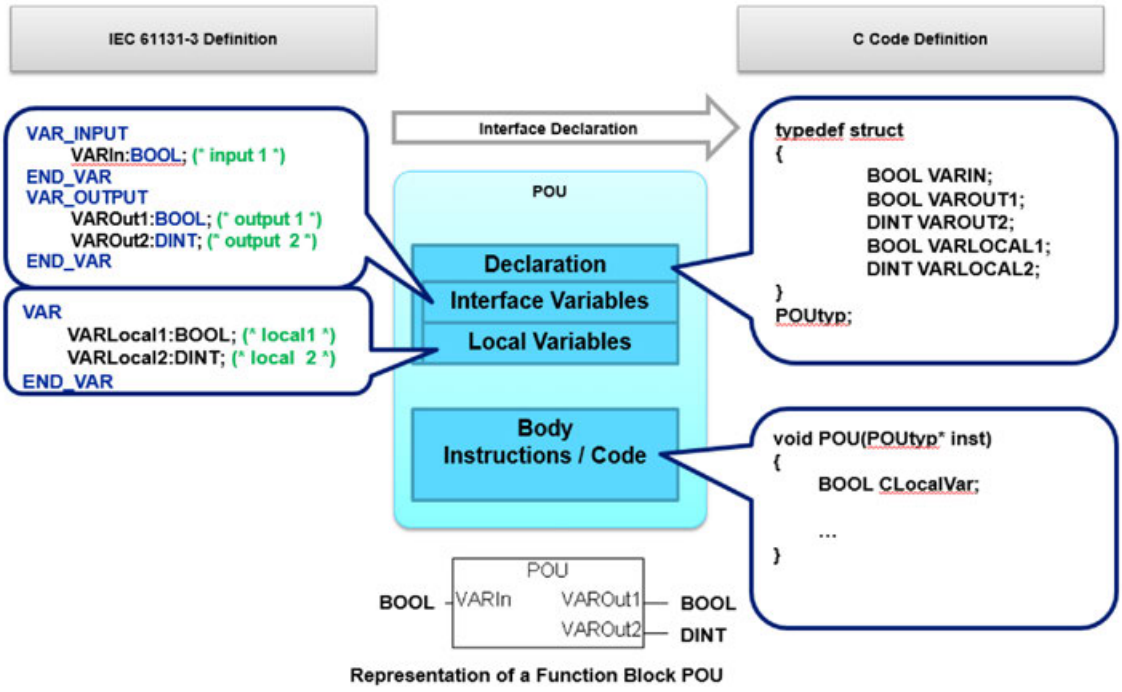


Fig. 1167: Interface between C/C++ and IEC code: POU type function block.

POU Structure with VAR_IN_OUT

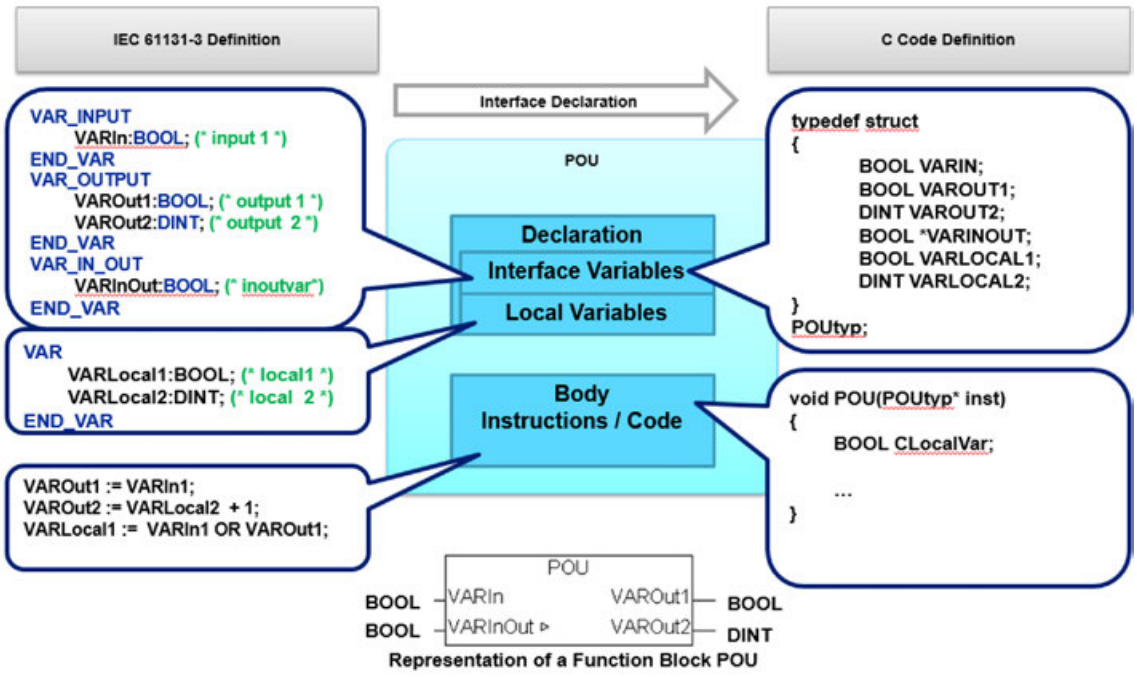


Fig. 1168: Interface between C/C++ and IEC code: POU structure with VAR_IN_OUT.

VAR_IN_OUT parameters are passed as reference to C/C++ implementation (parameter is implemented as pointer).

Function block instance data

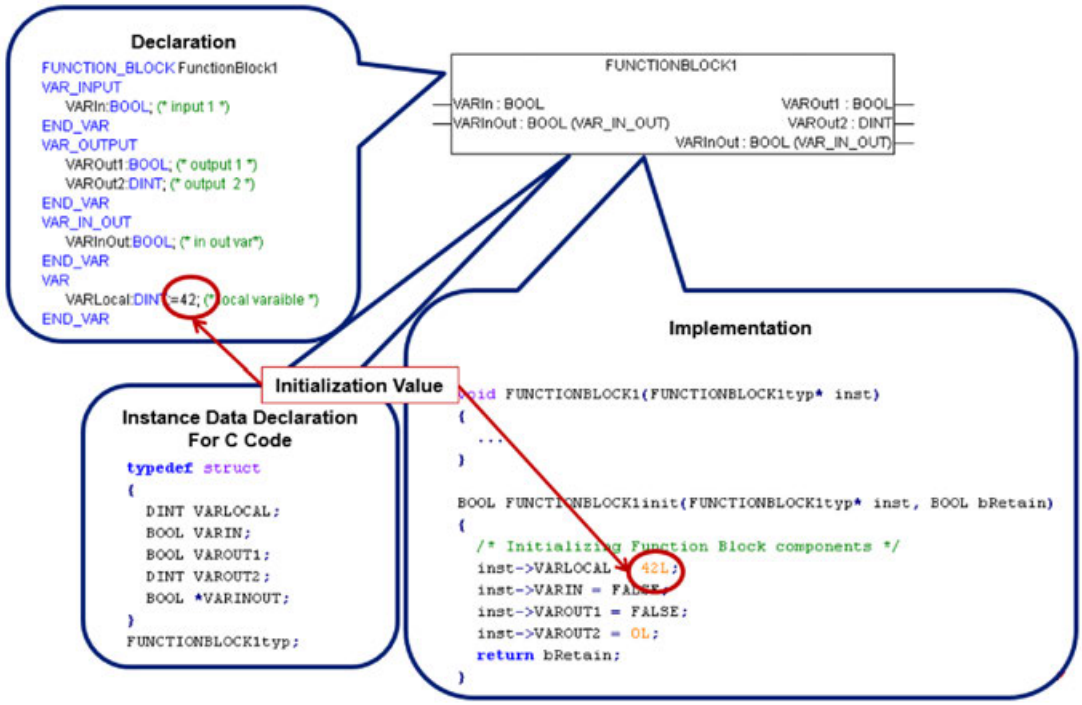


Fig. 1169: Interface between C/C++ and IEC code: function block instance data.

For each POU an *init* function is generated. This function is called on load-time and on reset of IEC application. Within this function, all parameters (VAR_IN, VAR_OUT, VAR_IN_OUT) and instance variables (VAR) are set to zero. If an instance variable is initialized in IEC declaration, this value is automatically taken for initialization inside of *init* function.

Data Type

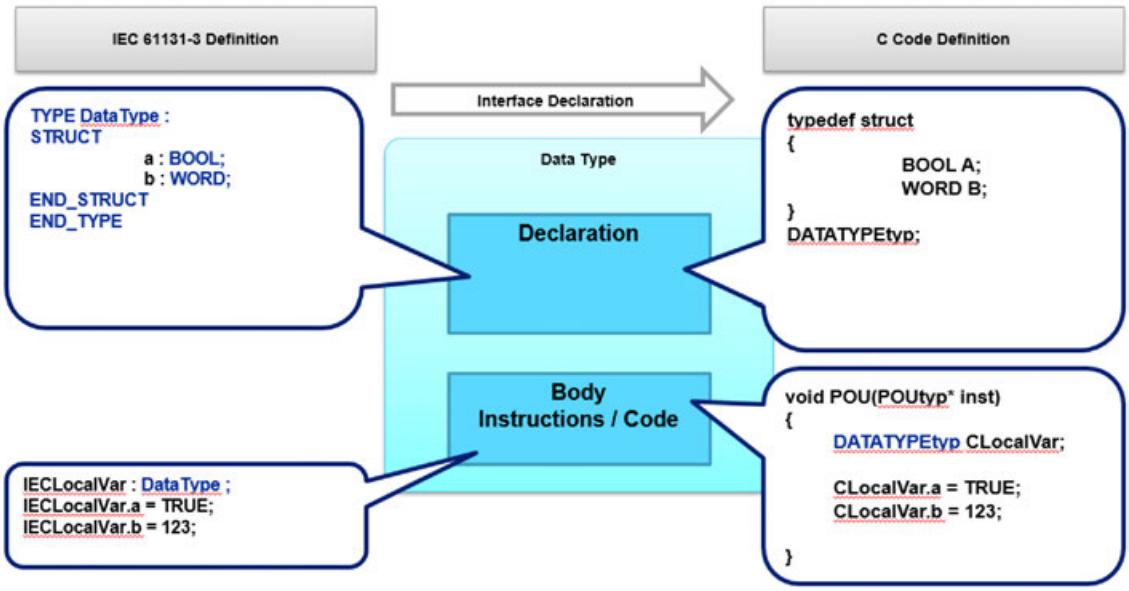


Fig. 1170: Interface between C/C++ and IEC code: Data type.

Advanced Topics

Debugging C/C++ Code

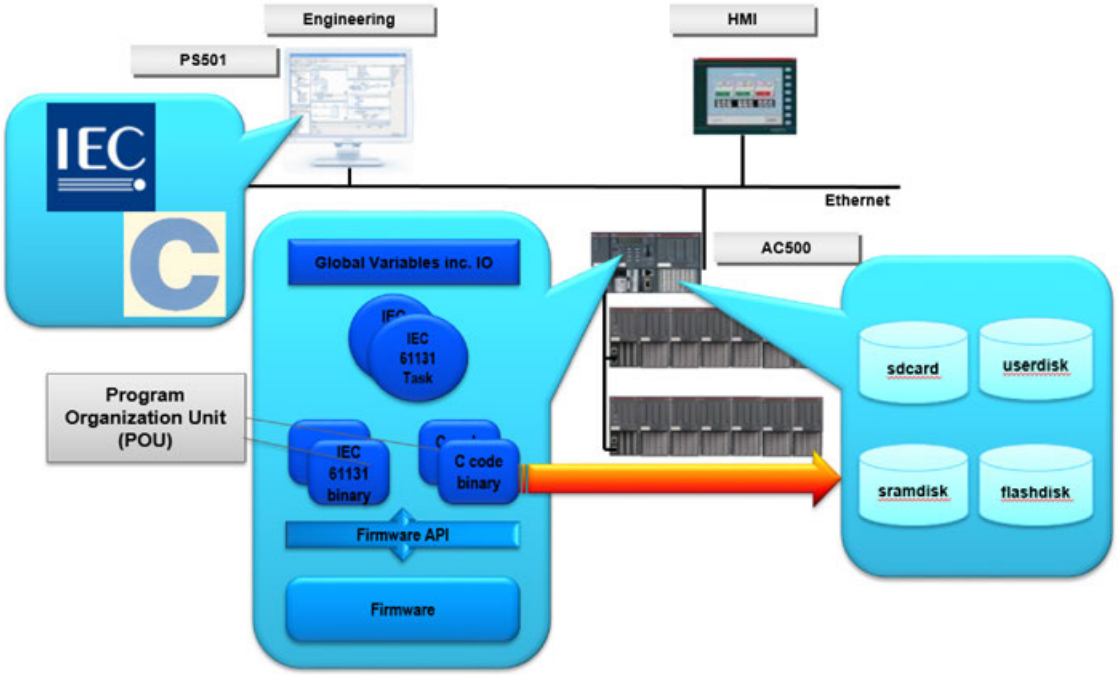


Fig. 1171: Debugging C/C++ code.

Debugging with Local POU Variables

Local POU variables are accessible via CODESYS and OPC server, thus they can be utilized to any means to reflect the state of the C/C++ implementation as desired.

Write User-defined Content to File

You can create your own logging messages. Write them to any memory location available for IEC access e.g. to read the file via the PLC's FTP server.

1. Use the *SysLibFile.h* or *stdio.h* file.
2. For further information, please refer to the CODESYS online help (chapter AC500 CPU Storage Devices).

Send User-defined Content via Ethernet

You can create your own logging messages. Send them to any other PLC or PC for further analysis.

1. Use the *SysLibSocket.h* file.
2. For further information, please refer to the CODESYS online help (chapter AC500 Ethernet Protocols and Ports).

Firmware Application Programming Interface (FWAPI)

Access to AC500 firmware from C/C++ application part is gained via the FWAPI (Firmware Application Programming Interface). It consists of a set of libraries sorted in two categories (C standard libraries and SysLib libraries). This allows a fine-grained selection of needed functionality.

▷ In order to use FWAPI from C/C++ application part, select the **Libraries** tab and click **Apply**.

⇨ `#include` statements are automatically included into source files.

FWAPI implements a subset of C standard library. ↯ *Chapter 1.6.5.4.7.5 "Known Issues and Frequently Asked Questions" on page 6258*

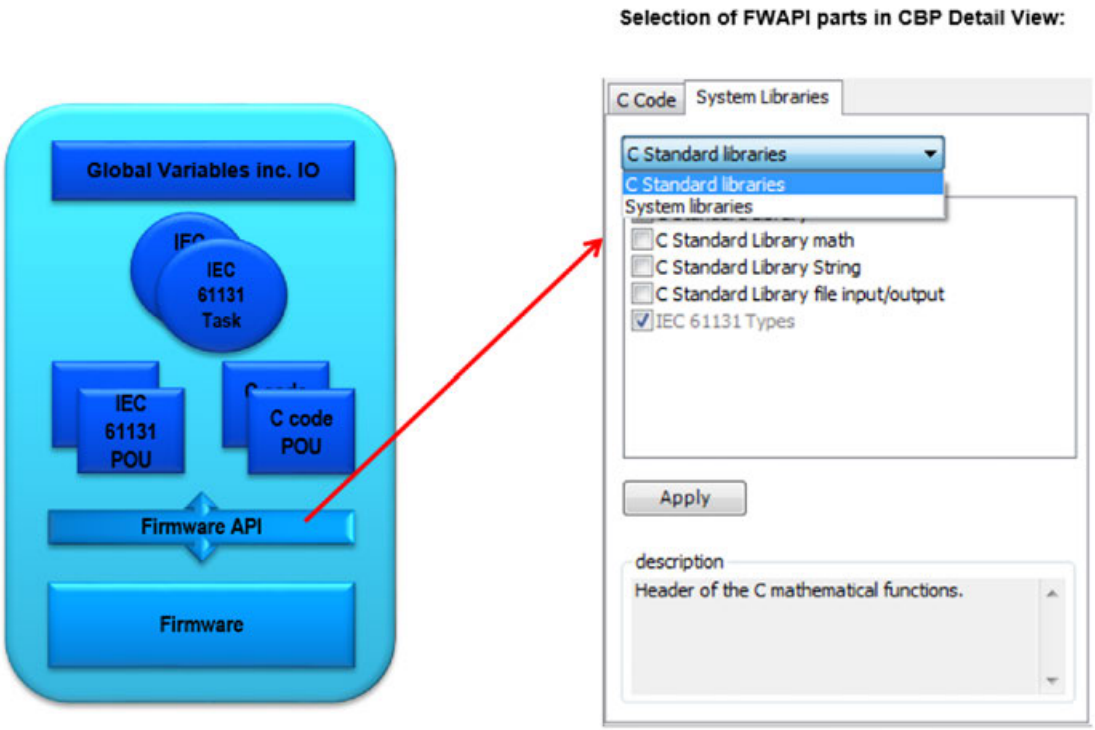


Fig. 1172: FWAPI (Firmware Application Programming Interface).

1. Right-click a list entry to open the context menu.
2. Select *“Display”* to open the header file in your default editor.

Additional Source Code Files in C/C++ Application

The C/C++ application can be split into any number of header and source files, as long as they are assembled via `#include` directives into one of the auto-generated .c files: *<application name>_shell.c* or *<application_name>_app.c*. In V2.3 this is to be done manually.

Adding files to C/C++ application:

1. Copy the files to the project’s source folder. You might have to compile or change the tabs to update the file list in the Automation Builder.
2. Right-click the project’s source folder, select *“Add files”* and use the dialog.
   ⇨ A copy of the selected files is placed in the project’s source folder.

Removing files from the project:

1. Delete the files from the project’s source folder. You might have to compile or change the tabs to update the file list in the Automation Builder.
2. Right-click the file and select *“Delete”*.

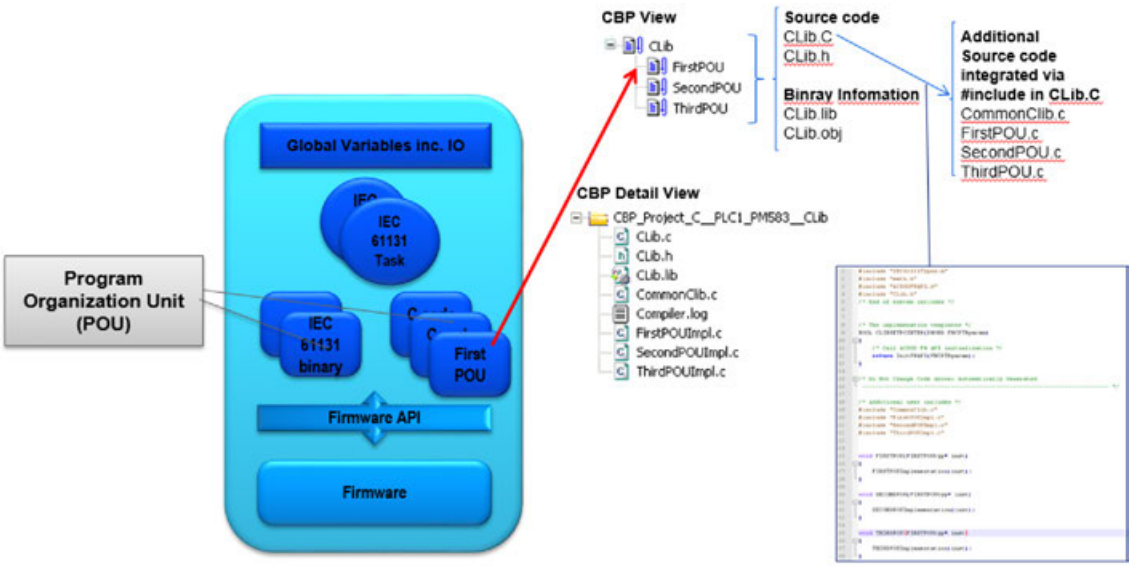
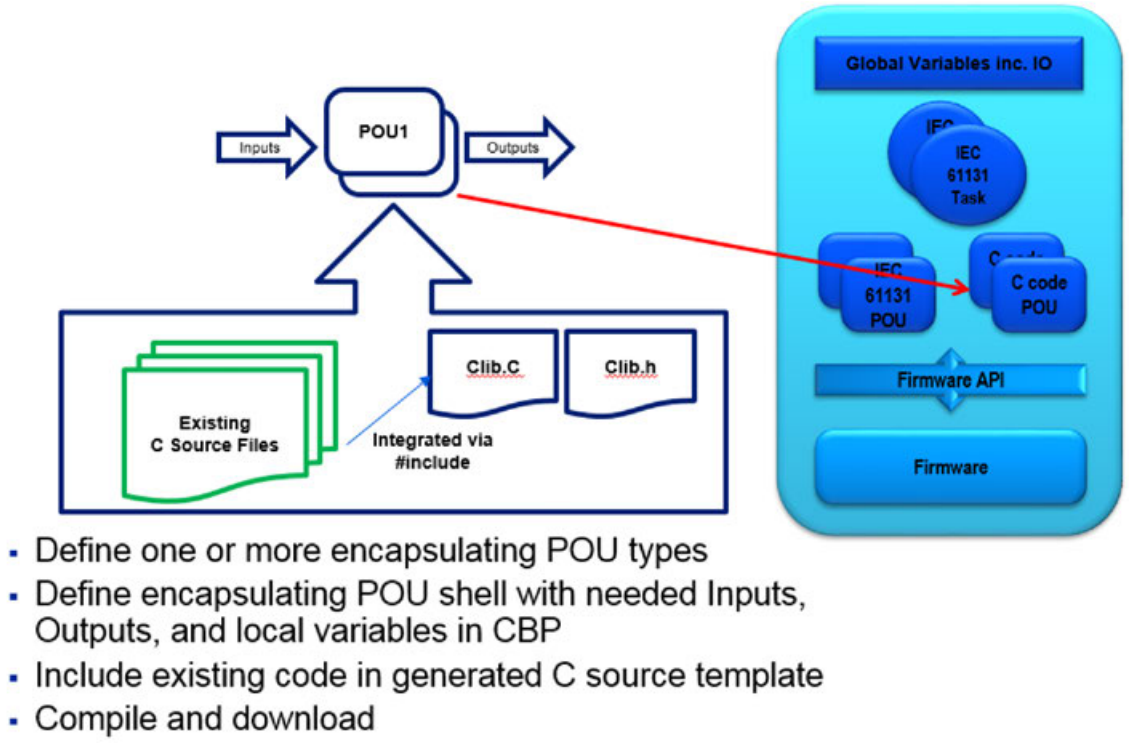


Fig. 1173: Integration of additional source code into C/C++ application.

Migration of Existing Applications

An existing C/C++ application can be used as part of an IEC application. All functions, which shall be called from IEC application have to be wrapped by a POU.

> ***Exception:*** *Functions of an existing application, which shall exclusively be called from other parts of the C/C++ code, do not need to be wrapped by POU definitions.*



- Define one or more encapsulating POU types
- Define encapsulating POU shell with needed Inputs, Outputs, and local variables in CBP
- Include existing code in generated C source template
- Compile and download

Fig. 1174: Migration of existing applications

Wrapping a function by a POU:

1. Define one or more encapsulating POU types for each corresponding function.
2. Define encapsulating POU shell with needed inputs, outputs, and local variables in Automation Builder.
3. Include existing code in generated C source template.
4. Compile and download the function.

Binary Deployment of C/C++ Application

Application libraries can be implemented in C/C++ code. Only the object files need to be shipped to the customer. This protects the knowledge within the code.

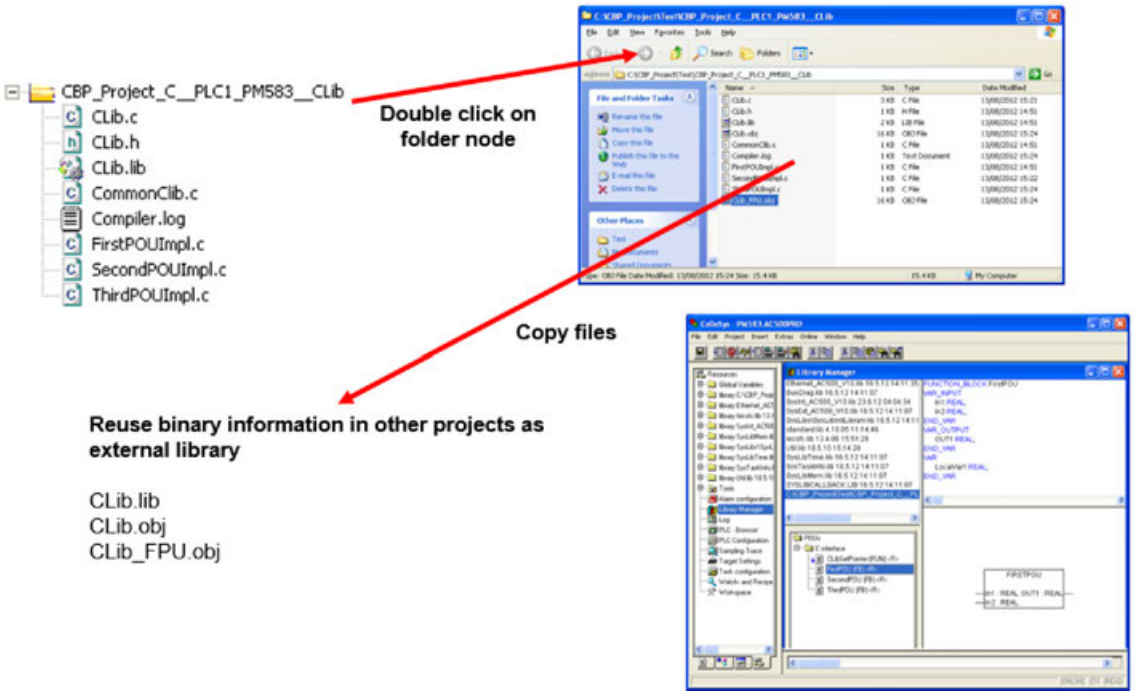


Fig. 1175: Binary deployment of C/C++ application.

Export/Import of C/C++ Code

Export/Import of C/C++ Application

Not supported yet.

Export/Import of Data Type

To import/export an interface of a single data type, right-click on the POU in tree view.

Export/Import of POU Interface

To import/export an interface of a single POU, right-click on the POU in tree view.

Project Backup Mechanism

The .c and .h files are saved in TEMP user directory under CCodeFiles\ProjectName. A copy of these files is always created on interface update (e.g. POU changes - definitions/names). A timestamp is added to the file name by copying in the TEMP user directory.

Location of the default TEMP directory:

Windows 7/8.1: %USERPROFILE%\AppData\Local\Temp. By default, "AppData" is hidden.

These files are never removed by the system. The user can decide when to delete.

Known Issues and Frequently Asked Questions

Supported compiler toolchains:

- GCC 4.7.3

CAUTION!
Support for Windriver DIAB 5.9.2 has been dropped.

Supported compiler toolchains for AC500 PLCs are shipped along with Automation Builder.

If using another version or another C/C++ cross compiler toolchain or to modify the predefined environment settings, the compiler settings must be configured manually:

1. From the **Tools** menu select **Options**. From the menu list select **C compiler settings**.

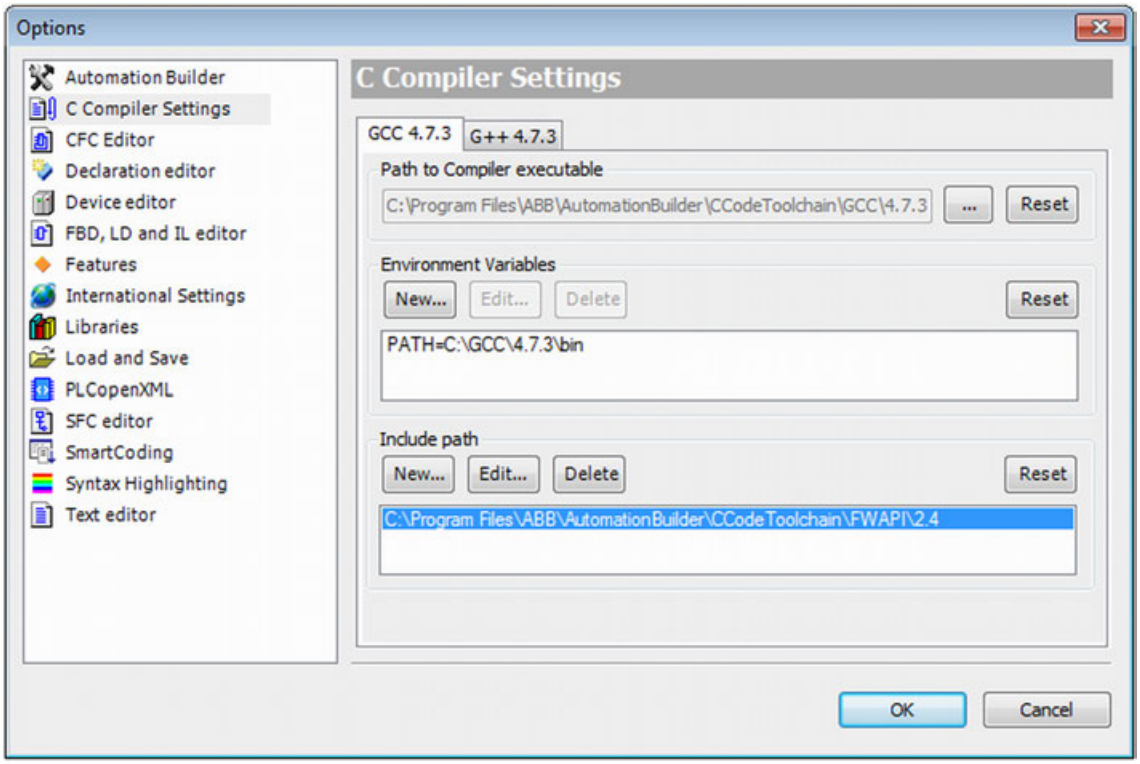


Fig. 1176: Defining compiler settings.

2. Open the **GCC <>** tab.
3. Under **Environment Variables** click **New** to enter the path to compiler executable.
4. Under **Include path** click **New** to add custom include paths. The include path to FWAPI is automatically detected and cannot be changed.

CAUTION!
Changes on the preset FWAPI path lead to ending support service.

Specifying optimization level

When optimizing for size, GCC uses primitives for saving and restoring GPR and FPR registers from *libgcc*. Since C/C++ application is not linked against *libgcc*, this results in link errors (symbol not found). Therefore optimization level –Os is not available in the compiler settings.

Restrictions

Restrictions on C:

Language elements: ANSI C and C99 language extension are supported.

Restrictions on C++ :

- Online change functionality is not available, if C++ toolchain is used. After changes in IEC application part, always a full download will occur.
- Language elements: ISO C++98 and ISO C++2003 C++ dialects are supported as it is supported by GCC –std option.
- We currently do not support:
  – RTTI (Runtime Type Information)
  – Exceptions

  When existing code is imported, please ensure that the code works with these restrictions.

Restrictions for PM595:

Do not use 64-bit data types (LREAL, LINT, ULINT, LWORD) as return value of Function POUs.

Workaround: Use either function block POU, or encapsulate 64-bit return values in a STRUCT declaration and return this data instead.

Problems with accessing function block parameters from C/C++ application:

In order to convert the parameter names of POUs or DUTs automatically, follow the following rules:

- Function block parameters/DUT members get converted to uppercase (e.g. FirstParam -> FIRSTPARAM).
- Function parameters get converted to uppercase and a postfix is added (e.g. abc -> ABC-param).

Using standard output/standard error:

Access to standard streams (e.g. via printf, puts, etc.) is not available. Please use standard file access instead (e.g. fprintf, fputs, etc.).

Supported parts of C standard library

A description on available functions in the standard library (FWAPI functions) is provided under <AB InstallDirectory>\CCodeTool-chain\FWAPI\2.4\doc\index.html.

Header	Description	Sup-ported	Remarks
assert.h	Contains the assert macro, used to assist with detecting logical errors and other error types in debugging versions of a program.	Yes	-
com-plex.h	A set of functions for manipu-lating complex numbers.	Yes	-

Header	Description	Sup-ported	Remarks
ctype.h	Defines a set of functions used to classify characters by their types or to convert between uppercase and lowercase in a way that is independent of the used character set (typically ASCII or one of its extensions, although implementations utilizing EBCDIC are also known).	Partially	Not supported: setchrclass
errno.h	For testing error codes reported by library functions.	Yes	-
fenv.h	Defines a set of functions for controlling floating-point environment.	Yes	-
float.h	Defines macro constants specifying the implementation-specific properties of the floating-point library.	Yes	-
inttypes.h	Defines exact-width integer types.	Yes	-
iso646.h	Defines several macros that are equivalent to some of the operators in C. For programming in ISO-646 variant character sets.	Yes	-
limits.h	Defines macro constants specifying the implementation-specific properties of the integer types.	Yes	-
locale.h	Defines localization functions.	Yes	Standard C/POSIX locales only
math.h	Defines common mathematical functions.	Partially (ANSI and C99)	Not supported: Macros: NAN, HUGE VAL
setjmp.h	Declares the macros setjmp and longjmp, which are used for non-local exits.	Yes	-
signal.h	Defines signal handling functions.	No	-
stdalign.h	For querying and specifying the alignment of objects.	No	Part of C11: Not supported by GCC 4.7.3
stdarg.h	For accessing a varying number of arguments passed to functions.	Yes	-
stdatomic.h	For atomic operations on data shared between threads.	No	Part of C11: Not supported by GCC 4.7.3
stdbool.h	Defines a boolean data type.	Yes	-
stddef.h	Defines several useful types and macros.	Yes	-
stdint.h	Defines exact-width integer types.	Yes	-

Header	Description	Sup-ported	Remarks
stdio.h	Defines core input and output functions.	Partially	Not supported: **Operations on files:** tmpfile, tmpnam **File access:** freopen, setbuf, setvbuf **Formatted input/output:** scanf, vscanf **Character input/output:** getc, getchar, gets, ungetc **Error-handling:** clearerr, ferror, perror **Macros:** BUFSIZ, FILENAME_MAX, FOPEN_MAX, L_tmpnam, TMP_MAX
stdlib.h	Defines numeric conversion functions, pseudo-random numbers generation functions, memory allocation, process control functions.	Partially	Not supported: **Pseudo-random sequence generation:** srand **Environment:** abort, atexit, at_quick_exit, exit, getenv, quick_exit, system, _Exit **Integer arithmetics:** llabs, lldiv **Multibyte characters /strings:** mblen, mbtowc, wctomb, mbstowcs, wcstombs **Macros:** EXIT_FAILURE, EXIT_SUC-CESS, MB_CUR_MAX
stdnore-turn.h	For specifying non-returning functions.	No	Part of C11: Not supported by DIAB C lib and GCC 4.7.3
string.h	Defines string handling functions.	Partially	Not supported: **Comparison:** strcoll, strxfrm **Searching:** strtok (alternative: strtok_r) **Other:** strerror
tgmath.h	Defines type-generic mathematical functions.	Yes	-
threads.h	Defines functions for managing multiple threads as well as mutexes and condition variables.	No	Part of C11: Not supported by GCC 4.7.3
time.h	Defines date and time handling functions.	Partially	Not supported: **Conversion:** localtime
uchar.h	Types and functions for manipulating unicode characters.	No	-
wchar.h	Defines wide string handling functions.	No	-
wctype.h	Defines set of functions used to classify wide characters by their types or to convert between uppercase and lowercase.	No	-

Supported parts

A description on available functions in the standard library (FWAPI functions) is provided under <AB InstallDirectory>\CCodeTool-chain\FWAPI\2.4\doc\index.html.

Header	Description	Supported
<c*> (.h)	C library headers	Yes
array	Containers	No
bitset	Containers	No
deque	Containers	No
forward_list	Containers	No
list	Containers	No
map	Containers	No
queue	Containers	No
set	Containers	No
stack	Containers	No
unordered_map	Containers	No
unordered_set	Containers	No
vector	Containers	No
ios	Input/output base classes	No
streambuf	Stream buffer	No
istream	Input streams	No
ostream	Output streams	No
iostream	Standard input/output streams library	No
fstream	File streams	No
sstream	String streams	No
iomanip	IO manipulators	No
iosfwd	Input/output forwarding	No
atomic	Atomics and threading library	No
condition_variable	Atomics and threading library	No
future	Atomics and threading library	No
mutex	Atomics and threading library	No
thread	Atomics and threading library	No
algorithm	Algorithms	No
chrono	Time	No
complex	Complex numbers	No
functional	Function objects	No
initializer_list	Initializer list	No
iterator	Iterator definitions	No
limits	Numeric limits	No

Header	Description	Supported
memory	Memory elements	No
new	Dynamic memory	Yes
numeric	Generalized numeric operations	No
random	Random	No
ratio	Ratio header	No
regex	Regular expressions	No
string	Strings	No
system_error	System errors	No
tuple	Tuple library	No
typeindex	Type index	No
typeinfo	Type information	No
type_traits	type_traits	No
utility	Utility components	No
valarray	Library for arrays of numeric values	No

Floating point instructions:

Support for floating point instructions (single and double precision) in hardware.

Debugging:

At the moment the development environment does not support debugger, i.e. debugging while running on PLC target is not possible.

Linking third-party binaries or libraries:

At the moment it is not possible to link against third-party binaries or libraries.

Adding third-party source code to C/C++ applications:

Chapter 1.6.5.4.7.4.3 “Additional Source Code Files in C/C++ Application” on page 6255

Compiler/linker options for compiling C/C++ code:

CAUTION!
These compiler/linker settings are activated by default, and cannot be changed by end-users, since otherwise compilation/linking may fail or lead to unexpected results.

In addition to that, following options are configurable by end users via C/C++ compiler settings dialog.

Table 755: Compile options:

Type of PLC	GCC (C)	G++ (C++)
PM55x, PM56x, PM57x, PM58x	-mcpu=860 -fno-common -msdata=none -fno-jump-tables -fno-section-anchors -fno-merge-constants -fno-builtin -nostdlib -Wconversion -Werror-implicit-function-declaration fstack-usage	-
PM590, PM591, PM592	-mcpu=603e -fno-common -msdata=none -fno-jump-tables -fno-section-anchors -fno-merge-constants -fno-builtin -nostdlib -Wconversion -Werror-implicit-function-declaration fstack-usage	-fmessage-length=0 -fno-rtti -fomit-frame-pointer -fno-exceptions -fno-asynchronous-unwind-tables -fno-unwind-tables
PM595	-mcpu=8540 -mdouble-float -mfloat-gprs=double -mspe -mabi=spe -fno-common -msdata=none -fno-jump-tables -fno-section-anchors -fno-merge-constants -fno-builtin -nostdlib -Werror-implicit-function-declaration -Wconversion -fstack-usage	-

Dialect options

Table 756: GCC (C):

Option	GCC (C)
C99	-std=c99
ANSI	-ansi

Table 757: G++ (C++):

Option	GCC (C)
ISO C++98	-std=c++98
ISO C++03	-std=c++03

Optimization options:

Option	GCC (C) and G++ (C++)
none	-
normal	-O2
extra	-O3

Linker options:

Linker options are used in exactly the order as shown in the following table.

C	C++
--emit-relocs	--emit-relocs
--no-check-sections	--no-check-sections
-L%TOOLCHAINPATH%\powerpc-elf-eabi\lib	-L%TOOLCHAINPATH%\powerpc-elf-eabi\lib
-L%TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3	-L%TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3
	%TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3\ecrti.o
	%TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3\crtbegin.o
userapplication.obj	userapplication.obj
-	-lstdc++

C	C++
-	%TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3\crtend.o %TOOLCHAINPATH%\lib\gcc\powerpc-elf-eabi\4.7.3\ecrtn.o
%TOOLCHAINPATH%\powerpc-elf-eabi\lib\lib_a-setjmp.o %FWAPIPATH%\libFWAPI.a" -o user_libname.obj --strip-debug --script=ac500_link_script	%TOOLCHAINPATH%\powerpc-elf-eabi\lib\lib_a-setjmp.o %FWAPIPATH%\libFWAPI.a" -o user_libname.obj --strip-debug --script=ac500_link_script

Why are VAR_IN_OUT parameters of a function block not initialized in corresponding init function?

During initialization of the function block the pointer `BYTE *B;` is not valid. It is getting valid during first call of the function block in IEC, therefore it is first accessible in FB main function code.

CODESYS compiler complains about unresolved symbols _f_sub, __subsf3, …:

Your PLC uses math emulation. Please include math standard library (math.h) to support this.

Type char:

Type `char` is implemented as `unsigned char`. Hence, its definition is equal to IEC type BYTE.

Why does GCC generate code that aborts application when va_list is used?

If you use `va_list`, GCC performs an automatic type promotion of the list members:

- `char/short` ➔ `int`
- `unsigned char/unsigned short` ➔ `unsigned int`
- `float` ➔ `double`

When extracting list members with `va_arg()`, ensure using the **promoted** type (e.g. `int` instead of `char`). We recommend you, to check the compiler warnings. In case of type mismatch, GCC will throw a warning similar to the following:

```
warning: 'unsigned char' is promoted to 'int' when passed through
'...' [enabled by default]
note: (so you should pass 'int' not 'unsigned char' to 'va_arg')
note: if this code is reached, the program will abort
```

CAUTION!
In this case call to `va_arg()`. Otherwise GCC will insert a trap instruction, that halts your application.

Parameters of POU or the members of data types defined in Automation Builder differ from definition in C sources:

On code generation, parameters of POU and members of data types are slightly transformed:

- Names of POU and data types are converted to uppercase (e.g. mypou -> MYPOU).
- Parameters of POUs and members of data types are converted to uppercase.
- Parameters of POU type FUNCTION: A postfix "param" is added (e.g. myByte -> MYBYTE-param).

Number of POUs or data types for a Automation Builder project:

Up to 50 POUs or data types can be created in one single Automation Builder project.

Searching for a POU or data type name in Automation Builder:

Search with *Edit -> Find/Replace -> Find does* not work is not supported yet.

Online change after changes in C/C++ application parts:

After a change in C/C++ application part, online change is not possible. Perform a "Clean All" function before downloading to PLC.

If changes are in IEC code only, online changes are still possible.

Appendix

IEC vs. C/C++ Operators

IEC	C	Remark
a **AND** b	a **&&** b	Logical AND
a **OR** b	a **||** b	Logical OR
a **AND** b	a **&** b	Bitwise AND
a **OR** b	a **|** b	Bitwise OR
NOT a	**!**a	Logical negation
NOT	**~**a	Bitwise negation
XOR	**^**a	Bitwise XOR
SHL(a,b**)**	a **<<** b	Bitwise left shift
SHR(a,b**)**	a **>>** b	Bitwise right shift
a **>** b	a **>** b	Greater than
a **>=** b	a **>=** b	Greater than or equal to
a **<** b	a **<** b	Less than
a **<=** b	a **<=** b	Less than or equal to
a **=** b	a **==** b	Equal to
a **<>** b	a **!=** b	Not equal to
a **:=** b	a **=** b	Assignment
ADR(a**)**	**&**a	Reference (address of)
a**^**	***a**	Indirection (object pointed to by)
a**^.**b	a**->**b	Structure dereference
a**.**b	a**.**b	Structure reference

IEC	C	Remark
a **+** b	a **+** b	Addition
a **–** b	a **–** b	Subtraction
a ***** b	a ***** b	Multiplication
a **/** b	a **/** b	Division
MOD(a,b**)**	a **%** b	Modulo
n/a	**++**a	Pre-increment
n/a	a**++**	Post-increment
n/a	**--**a	Pre-decrement
n/a	a**--**	Post-decrement

IEC vs. C/C++ Types

IEC	C	Remark
BOOL	Unsigned char	-
BYTE	Unsigned char	-
WORD	Unsigned short	-
DWORD	Unsigned long	-
SINT	Signed char	-
USINT	Unsigned char	-
INT	Short	-
UINT	Unsigned short	-
DINT	Long	-
UDINT	Unsigned long	-
REAL	Float	-
LREAL	Double	-
STRING(size)	Char[size+1]	Size specifier is optional. Default size is 80 characters (IEC). Size in C code is +1 due to null termination.
TIME	Unsigned long	Milliseconds starting from 00:00:00
TIME_OF_DAY (TOD)	Unsigned long	Milliseconds starting from 00:00:00
DATE	Unsigned long	Seconds starting from 1.1.1970 at 00:00:00
DATE_AND_TIME (DT)	Unsigned long	Seconds starting from 1.1.1970 at 00:00:00
LWORD	Unsigned long long	-
LINT	Long long	-
ULINT	Unsigned long long	-

C 99 vs. ANSI C

For further information refer to *http://www.open-std.org/jtc1/sc22/WG14/www/docs/n1256.pdf*:

- Restricted pointers: *http://gcc.gnu.org/onlinedocs/gcc/Restricted-Pointers.html*.
- Variable-length arrays: *http://gcc.gnu.org/onlinedocs/gcc/Variable-Length.html*.
- Flexible array members: *http://gcc.gnu.org/onlinedocs/gcc/Zero-Length.html*.
- `static` and type qualifiers in parameter array declarators.
- `long long int` type:*http://gcc.gnu.org/onlinedocs/gcc/Long-Long.html*.
- Additional floating-point characteristics in `<float.h>`.
- Removed implicit `int`.
- Reliable integer division.
- Universal character names (`\u` and `\U`).
- Compound literals:*http://gcc.gnu.org/onlinedocs/gcc/Compound-Literals.html*.
- Non-constant initializers:*http://gcc.gnu.org/onlinedocs/gcc/Initializers.html*.
- Designated initializers:*http://gcc.gnu.org/onlinedocs/gcc/Designated-Inits.html*.
- // comments:*http://gcc.gnu.org/onlinedocs/gcc/C_002b_002b-Comments.html*.
- Extended integer types and library functions in `<inttypes.h>` and `<stdint.h>`.
- Removed implicit function declaration.
- Preprocessor arithmetic done in `intmax_t/uintmax_t`.
- Mixed declarations and code:*http://gcc.gnu.org/onlinedocs/gcc/Mixed-Declarations.html*.
- New block scopes for selection and iteration statements.
- Integer constant type rules.
- Integer promotion rules.
- Macros with a variable number of arguments.
- The vscanf family of functions in `<stdio.h>` and `<wchar.h>`.
- Additional math library functions in `<math.h>`.
- Floating-point environment access in `<fenv.h>`.
- IEC 60559 (also known as IEC 559 or IEEE arithmetic) support.
- Trailing comma allowed in `enum` declaration.
- `%lf` conversion specifier allowed in `printf`.
- Inline functions.
- The snprintf family of functions in `<stdio.h>`.
- Boolean type in `<stdbool.h>`.
- Empty macro arguments.
- New struct type compatibility rules (tag compatibility).
- Additional predefined macro names.
- `_Prama` preprocessing operator.
- Standard pragmas.
- `__func__` predefined identifier.
- `va_copy` macro.
- Additional `strftime` conversion specifiers.
- Deprecate `ungetc` at the beginning of a binary file.
- Removed deprecation of aliased array parameters.
- Conversion of array to pointer not limited to values.
- Relaxed constraints on aggregate and union initialization.
- `return` without expression not permitted in function that returns a value (and vice versa).

C++ 98 vs. C++ 03

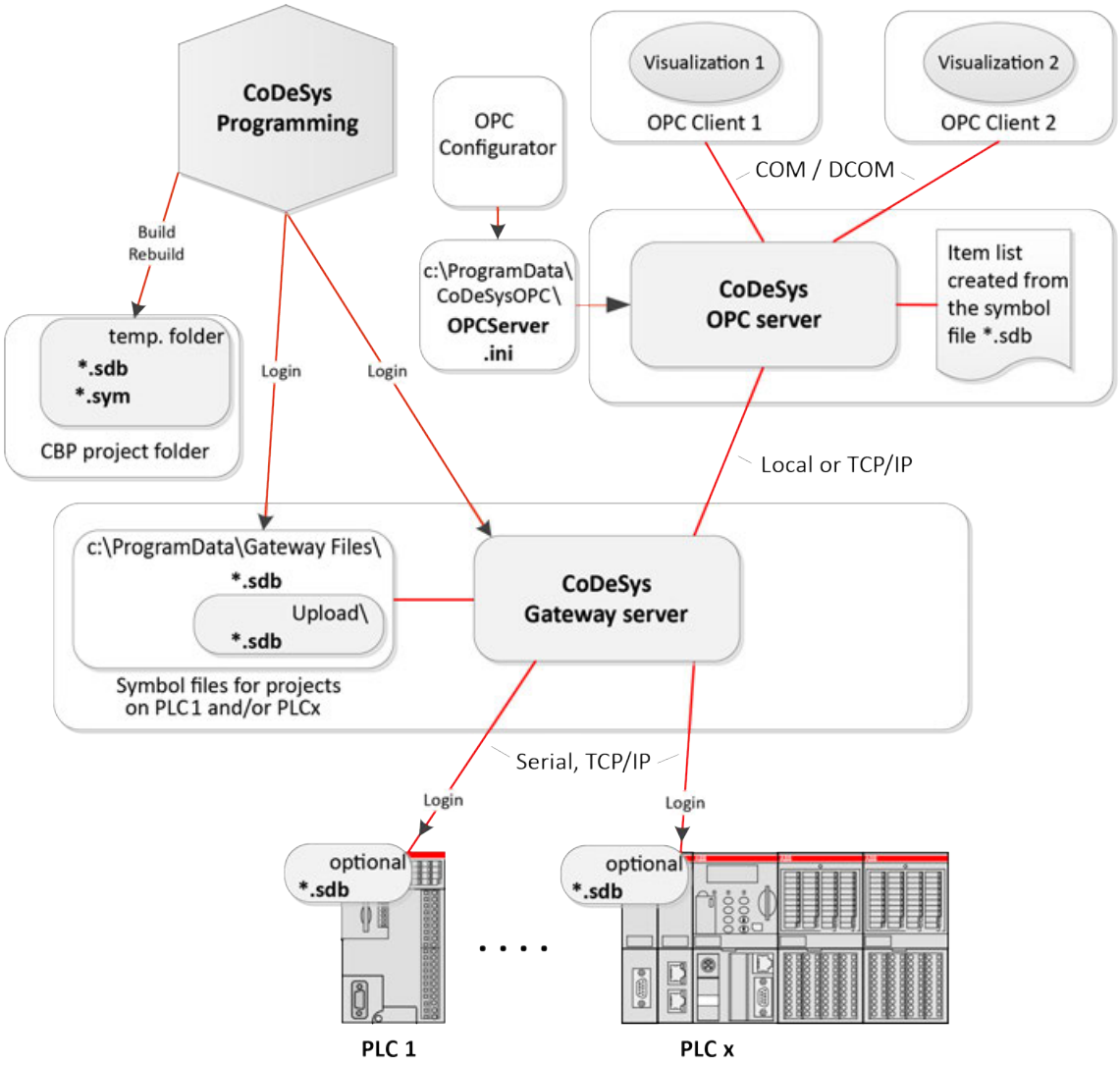
As C++ 98 and C++ 03 can be used interchangeably, programming is identical.

1.6.5.5 Server installation

1.6.5.5.1 OPC server for AC500 V2 products

Introduction

Architecture of the CODESYS OPC server



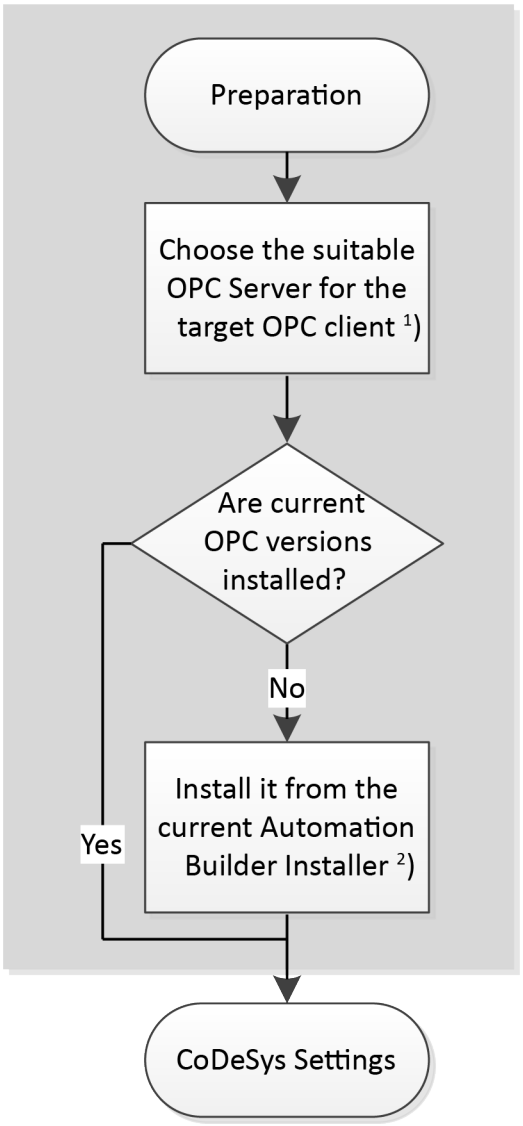
Essential documents

For further information see ↳ *Chapter 1.6.5.5.1.2 “Hints” on page 6277*.

	File name	Com-ment	Where to find
REF1	*OPC_V3_how_to_use_E.pdf* *OPC_V3_how_to_use_D.pdf*	OPC V3	C:\Program Files\ABB\CoDeSys OPC Server 3 AE
REF2	*AeConfigurator_User-Guide.pdf*	OPC V3	C:\Program Files (x86)\3S CODESYS\CODESYS OPC Server 3
REF3	*ReadMe.rtf*	OPC V3	Installation ABB DM Suit 1.0.: \PLC - AC500\OPC Server\OPC-ServerV3.xAE\
REF4	*ReleaseNotesOPCV3 AE for HA*	OPC V3	Installation ABB DM Suit 1.0.: \PLC - AC500\OPC Server\OPC-ServerV3.xAE\
REF5	*OPC_20_how_to_use_E.pdf* *OPC_20_how_to_use_D.pdf*	OPC V2	C:\Program Files (x86)\3S Software\CoDeSysOPC
REF6	*HA_OPC_Example.pdf*	OPC V3 HA	Installation CD PS501: \CD_AC500\Projects\Examples\High_Availability_OPCV3

Work flow

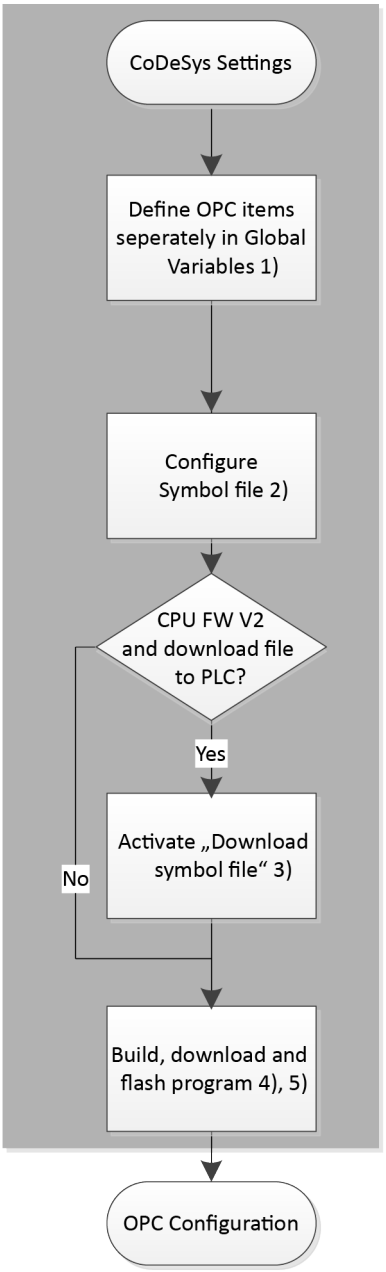
Consideration and preparation



1) *Chapter 1.6.5.5.1.2.1 “When using OPC server V2 or V3” on page 6277*

2) *Chapter 1.6.5.5.1.2.3 “Installation of OPC server” on page 6281*

Commission OPC server



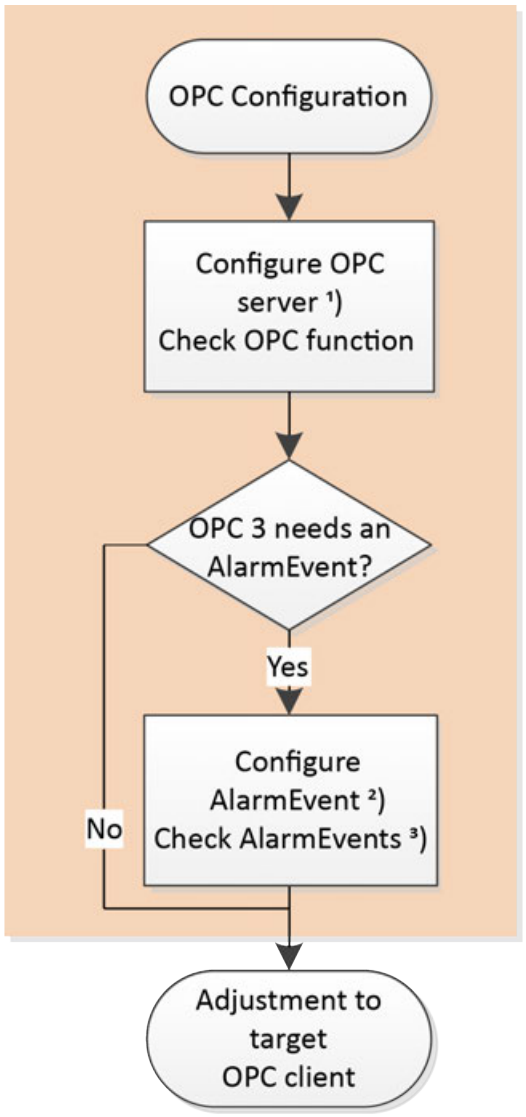
[1]) ↳ *Chapter 1.6.5.5.1.1.2 “Essential documents” on page 6272 REF6*.

[2]) ↳ *Chapter 1.6.5.5.1.2.5.1.1 “Configure a symbol file” on page 6285*

[3]) ↳ *Chapter 1.6.5.5.1.2.5.1.2 “Create and download a symbol file” on page 6287*

[4]) ↳ *Chapter 1.6.5.5.1.2.5.1.1 “Configure a symbol file” on page 6285*

[5]) ↳ *Chapter 1.6.5.5.1.4.1.1 “AC500 project” on page 6309 “step 5”*.

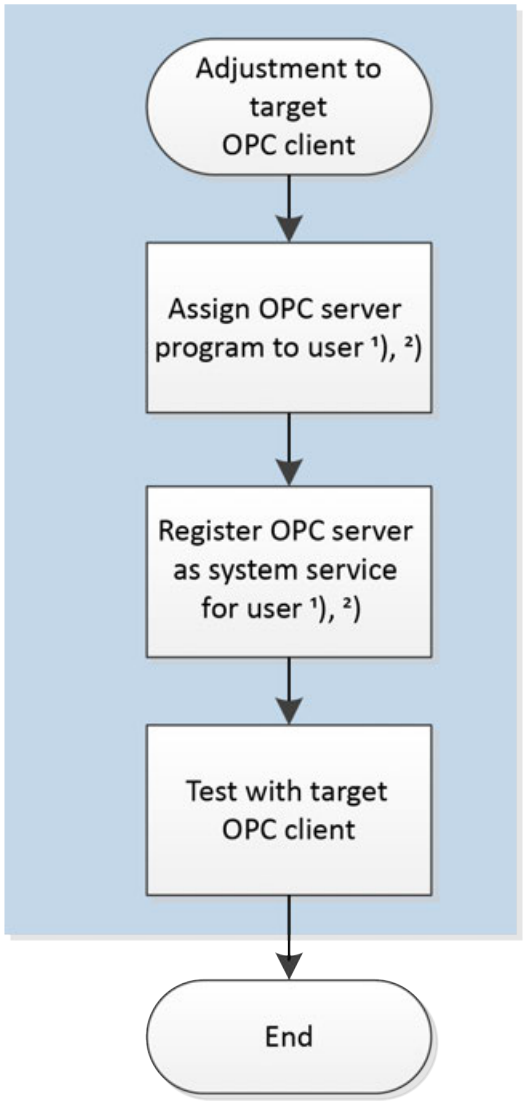


1) *Chapter 1.6.5.5.1.2.6.1 “Configure OPC server V2 (for AC500 V1 and V2)” on page 6289*

2) *Chapter 1.6.5.5.1.1.2 “Essential documents” on page 6272 REF2.*

3) *Chapter 1.6.5.5.1.2.7.1 “Check AlarmEvents” on page 6299*

Adjustment to target OPC client



[1]) *Chapter 1.6.5.5.1.1.2 “Essential documents” on page 6272 REF4*.

[2]) *Chapter 1.6.5.5.1.2.8 “Configure user account for OPC server” on page 6299*

Hints

When using OPC server V2 or V3

Required functions of the OPC client	OPC server V2	OPC server V3	Hints
Windows 7 32-bit, Windows 7 64-bit, Windows Server 2003, Windows Server 2008	X	X	
OPC client runs as service	-	X	
Support alarm/event	-	X	
Support AC500 HA	-	X	
OPC performance	-	Faster	Comparison with OPC server V2 to V3: transmission rate
Support VB, VBA OPC clients (Automation Interface, Automation Wrapper)	X	X	OPC server V3 supports VBA OPC clients. However, a OPC server V2 must be installed additionally as otherwise an essential DLL is missing.
Resource-saving to older OPC clients, which support only the old OPC DA 1.0a (Async I/O 1.0a) groups.	X	X	See hints *Chapter 1.6.5.5.1.3.4 “Behavior OPC server V3 via interface IOPCAsyncIO” on page 6308*
Simulation without AC500	-	X	

Several OPC clients used at the same time, must run in the same session.

See Hints Chapter 1.6.5.5.1.3.3 “Session isolation” on page 6306.

Default folder and contents

Windows 7, Windows Server 2008/2016 (64-bit)

OPC Server V2	Windows 7 64-bit, Windows Server 2008 64-bit, Windows Server 2016 64-bit
CoDeSysOPC.exe *OPCConfig.exe* *OPCConfig_e.exe* *OPC_20_how_to_use_D.pdf* *OPC_20_how_to_use_E.pdf*	C:\Program Files (x86)\3S Software\CoDeSysOPC\
CoDeSysOPC.ini *OPCServer.log*	C:\ProgramData\CoDeSysOPCV2.3
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\SysWOW64\

OPC Server V3	Windows 7 64-bit, Windows Server 2008 64-bit, Windows Server 2016 64-bit
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhandbuch.pdf* *AeConfigurator_UserGuide.pdf*	C:\Program Files (x86)\3S CoDeSys\CoDeSys OPC Server 3\
OPCServer.ini *OPCServerA.ini* *OPCServer.log*	C:\ProgramData\CoDeSysOPC\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\SysWOW64\

Windows 7 (32-bit), Windows Server 2008/2016 (32-bit)

OPC Server V2	**Windows 7 32-bit, Windows Server 2008 32-bit, Windows Server 2016 32-bit**
CoDeSysOPC.exe *OPCConfig.exe* *OPCConfig_e.exe* *OPC_20_how_to_use_D.pdf* *OPC_20_how_to_use_E.pdf*	C:\Program Files\3S Software\CoDeSysOPC\
CoDeSysOPC.ini *OPCServer.log*	C:\ProgramData\CoDeSysOPCV2.3
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

OPC Server V3	**Windows 7 32-bit, Windows Server 2008 32-bit, Windows Server 2016 32-bit**
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhand-buch.pdf* *AeConfigurator_UserGuide.pdf*	C:\Program Files\3S CoDeSys\CoDeSys OPC Server 3\
OPCServer.ini *OPCServerA.ini* *OPCServer.log*	C:\ProgramData\CoDeSysOPC\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\ProgramData\Gateway Files\ After starting the OPC server: C:\ProgramData\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

Windows Server 2008/2016 (32-bit)

OPC Server V2	**Windows Server 2008 32-bit, Windows Server 2016 32-bit**
CoDeSysOPC.exe *OPCConfig.exe* *OPCConfig_e.exe* *OPC_20_how_to_use_D.pdf* *OPC_20_how_to_use_E.pdf* *CoDeSysOPC.ini* *OPCServer.log*	C:\Program Files\3S Software\CoDeSysOPC\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\WINDOWS\Gateway Files\ After start CODESYS OPC server: C:\WINDOWS\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

OPC Server V3	**Windows Server 2008 32-bit, Windows Server 2016 32-bit**
WinCoDeSysOPC.exe *OPCConfig.exe* *AEConfiguration.exe* *CoDeSys_OPC_Server_V3_User_Guide.pdf* *CoDeSys_OPC_Server_V3_Benutzerhand-buch.pdf* *AeConfigurator_UserGuide.pdf* *OPCServer.ini* *OPCServerA.ini* *OPCServer.log*	C:\Program Files\3S CoDeSys\CoDeSys OPC Server 3\
Symbol file **.SDB*, **.SYM*	CBP open, after project build or rebuild all: in the project folder
Symbol file **.SDB*	After login in AC500: C:\WINDOWS\Gateway Files\ After start CODESYS OPC server: C:\WINDOWS\Gateway Files\Upload\
Gateway.exe	C:\Windows\System32\

*If folder **C:\ProgramData** is missing, select "Show hidden files, folders and drives" at "Control Panel ➔ All Control Panel Items ➔ Folder Options ➔ View ➔ Hidden files and folders".*

Installation of OPC server

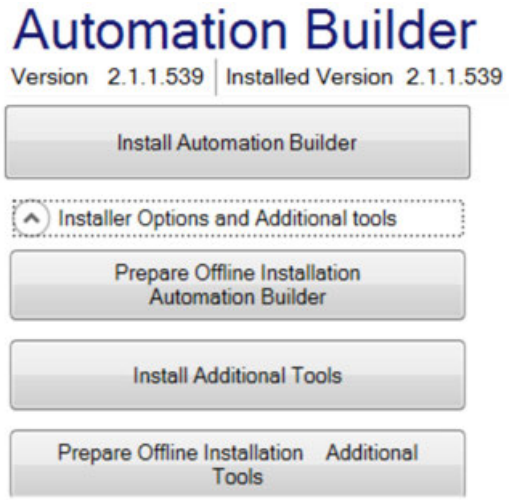
Prerequisites

The following applications are closed:
- *All OPC clients*
- *ABB OPC tunnel*
- *CODESYS gateway server*

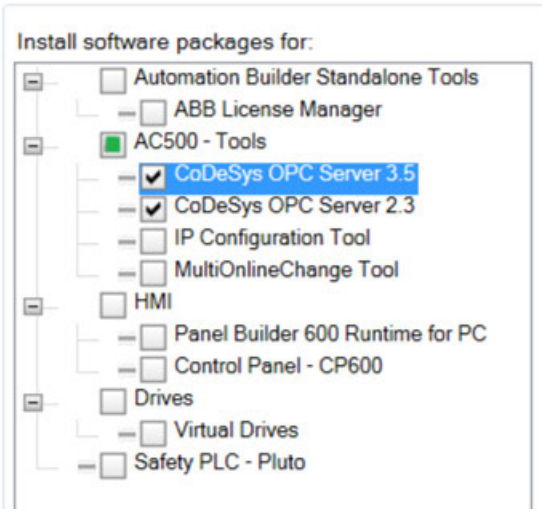
Ensure termination of the following processes:
- *Gateway.exe*
- *CoDeSysOPC.exe*
- *WinCoDeSysOPC.exe*
- *OCTsvc.exe*

Installing with Automation Builder

1. Go to homepage *http://new.abb.com/plc/automationbuilder/platform/software* .
2. Click button of *Latest Automation Builder version (recommended)* and run the installer.



3. Open *"Installer Options and Additional Tools"* and click *[Install Additional Tools]*.
4. Agree to the *"License Terms"*.



5. Select *“Version 2 and/or 3”* and install.

   ⇨ All required files are installed for OPC and the OPC server is registered automatically as user application.

Manual registration and unregistration

It is possible to register or to uninstall the OPC server manually either as COM server (user application) or as a service.

Register the OPC server as interactive software in the Windows registry:

Command for OPC 3: ***`WinCoDeSysOPC/RegServer`***

Command for OPC 2: `CoDeSysOPC/RegServer`

Register the OPC server as system service:

Command for OPC 3: ***`WinCoDeSysOPC/Service`***

Unregister the OPC server from the Windows registry and from the service entry:

Command for OPC 3: ***`WinCoDeSysOPC/UnRegServer`***

Command for OPC 2: `CoDeSysOPC/UnRegServer`

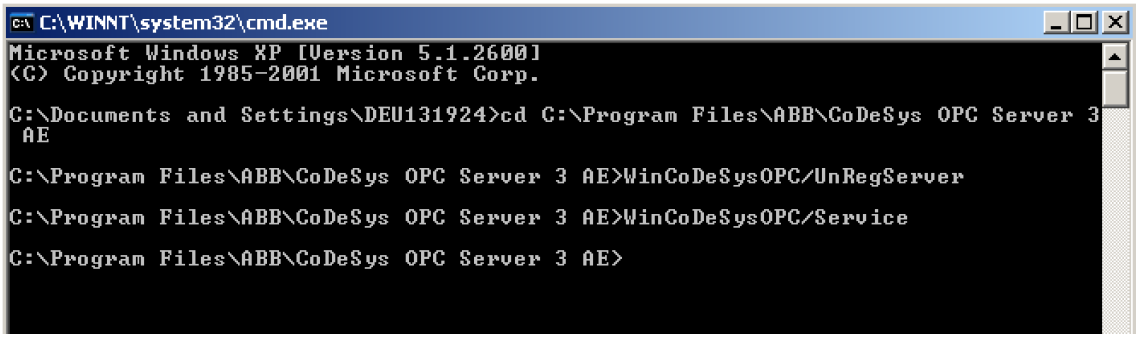
Please see REF1 chapter 3 (OPC 3) and REF6 chapter 2.2 (OPC 2) ↳ Table on page 6272 for details.

Register OPC server V3 as a system service

Prerequisites

- All programs, processes and services which connect to the OPC server are closed.

1. Start the *"Command Prompt"* with command *"cmd"* in the *"Start ➔ Run... "* window.



2. Go to the *CoDeSysOPC V2* installation folder.
3. Unregister the OPC server with `WinCoDeSysOPC/UnRegServer`.
4. Register the OPC server as system service with `WinCoDeSysOPC/Service`.

OPC clients for tests

Free of charge test clients can be found in the web:

https://industrial.softing.com/us/downloads.html

http://www.matrikonopc.com/products/opc-desktop-tools/index.aspx

CODESYS settings

Please refer to REF5 Online Help of PS501 chapter OPC for details ↳ *Table on page 6272.*

Symbol file

AC500 (V1 and V2)

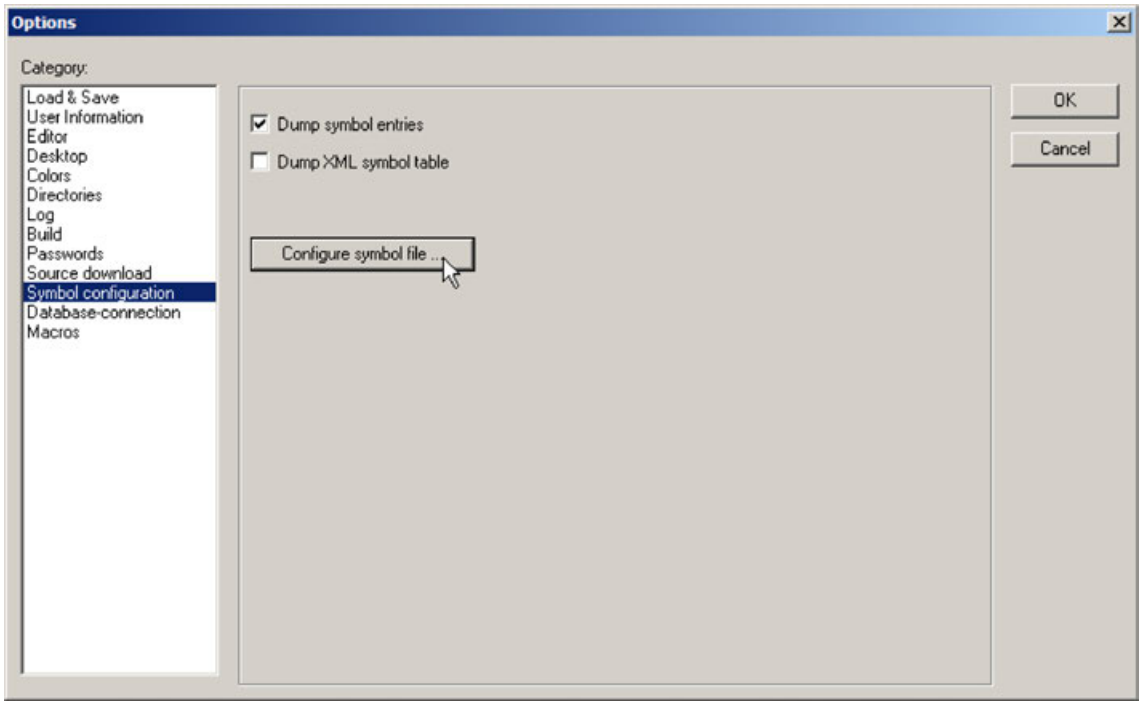
Configure a symbol file

Symbol files contain the items (variables) which are exchanged with the PLC for OPC communication. After the project has been build, the following symbol files are generated and stored in the project (*.pro*) folder:

- *.sdb*
  Important binary file needed by OPC server (computer-readable).
- *.sym*
  Human-readable file with content from *.sdb*. It can be used to check if *.sdb* has been generated correctly.

> *Please refer to REF5 Online Help of PS501 for how to configure a symbol file*
> ↳ *Table on page 6272.*

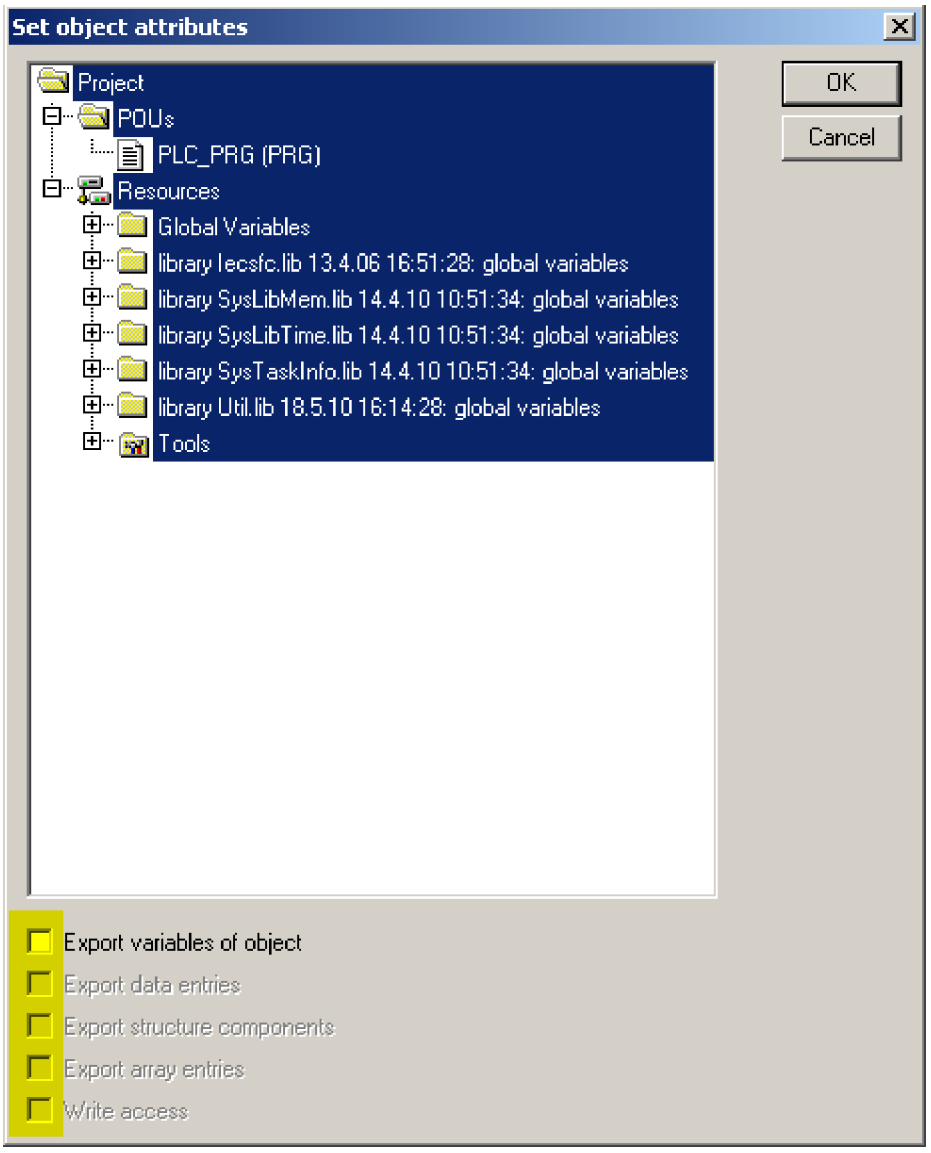
1. Open *CoDeSys - Application.AC500PRO* by double-click *“Application”* in the device tree.
2. Select the *“Symbol configuration”* in menu *“Project ➔ Options”*.



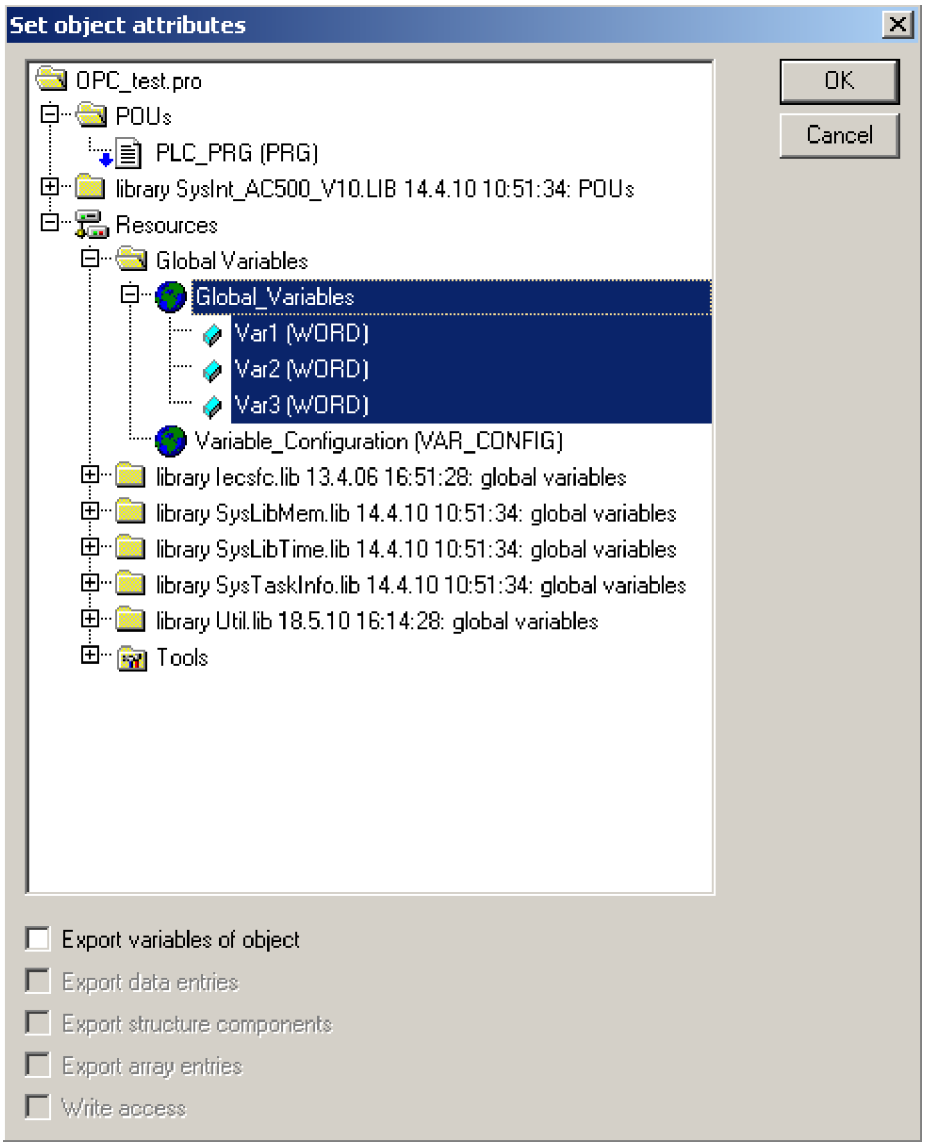
3. Enable checkbox *“Dump symbol entries”* and click *[Configure symbol file]*.

⇨

> *In the case a symbol file gets corrupt and e.g. contains additional symbols, please follow the steps to recreate a clean symbol file.*



1. Disable all checkboxes in window *Set object attributes* and confirm twice with *[OK]*.
2. Go to *“Project ➔ Options” “Symbol configuration”* and click *[Configure symbol file]*.



⇨ Select the variables which should be communicated as symbol:

3. Enable the following checkboxes:
   - *Export variables of object*
   - *Export structure components*
   - *Export array entries*
   - *Write access*
4. Confirm twice with *[OK]*.

   ⇨ Rebuild the project.

Create and download a symbol file

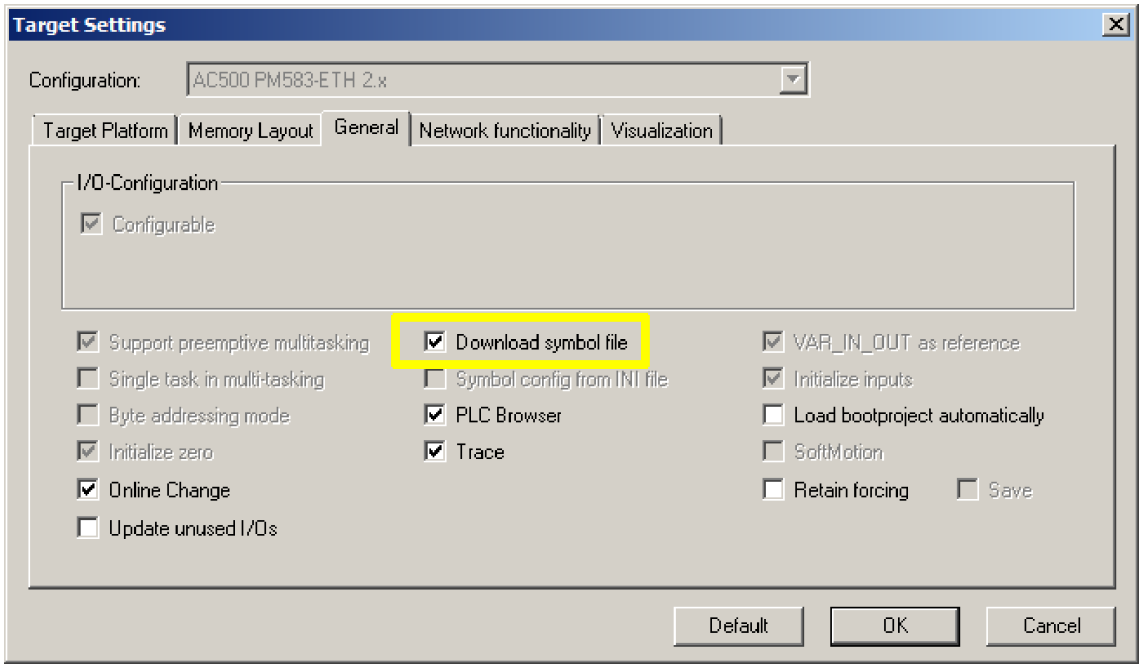
For CPU with FW V1:

If the PLC hardware is available, use *"login / download program"* to copy the *.sdb* file automatically to the gateway folder, e.g. *"C:\WINNT\Gateway Files"*. If the PLC hardware is not available, copy the *.sdb* file manually to the gateway folder.

When the OPC server is started, the *.sdb* file will be copied to the folder for gateway communication, e.g. to *"C:\WINNT\Gateway Files\Upload"*.

For CPU with FW V2:

Following option can be chosen to download the *.sdb* file also to the PLC.



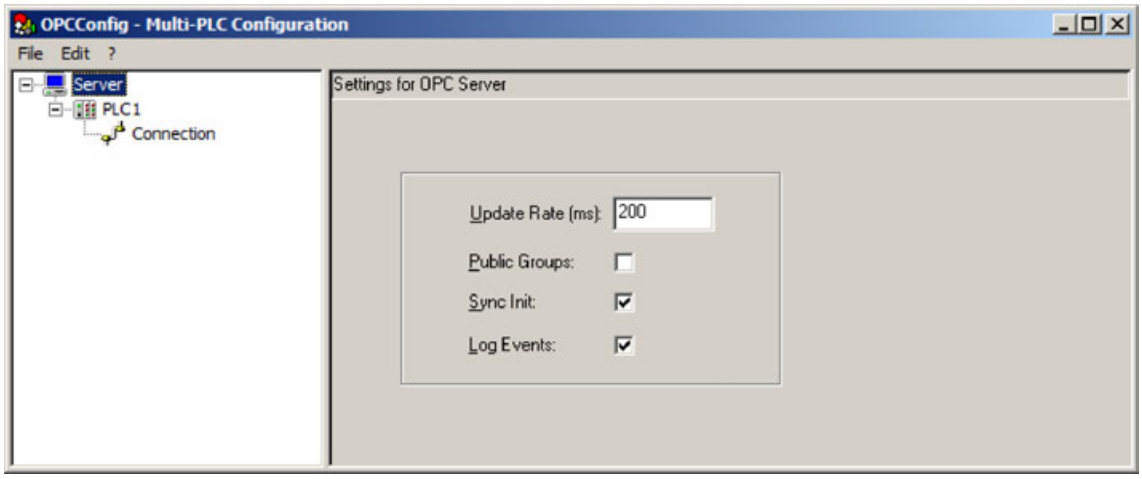
When the OPC server is started, *.sdb* will be copied from the PLC (if available) or from the gateway folder to *"C:\WINNT\Gateway Files\Upload"*.

> *Do not configure the program as a cyclic program, use a task configuration. Call the PLC Browser and have a look at the task time (command "tsk" in the command line). For example, if the program has a cycle time of 40 ms, use a task time of 50 or 60 ms to enable the CPU to answer the OPC request from the OPC server between the tasks.*

Configure OPC server

Configure OPC server V2 (for AC500 V1 and V2)

1. Start the OPC Configurator and select *Server*.



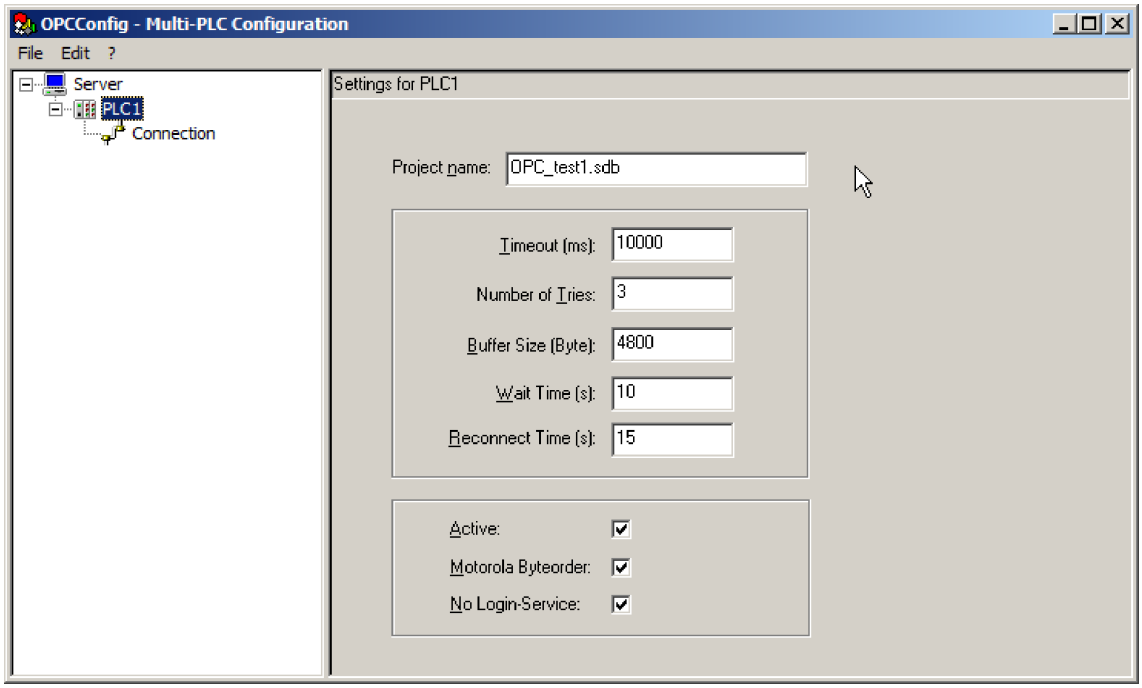
⇨

Update rate

- *The Update Rate may not be 0 (ms)!*
- *The default value of 200 ms is a suitable value of many applications.*
- *The adjustment for the Update Rate depends on the number of symbols (variables).*
- *For a big number of symbols it can be better to increase the Update Rate.*

The checkboxes Sync Init and Log Events must be enabled.

2. Select the 'PLC1' node.

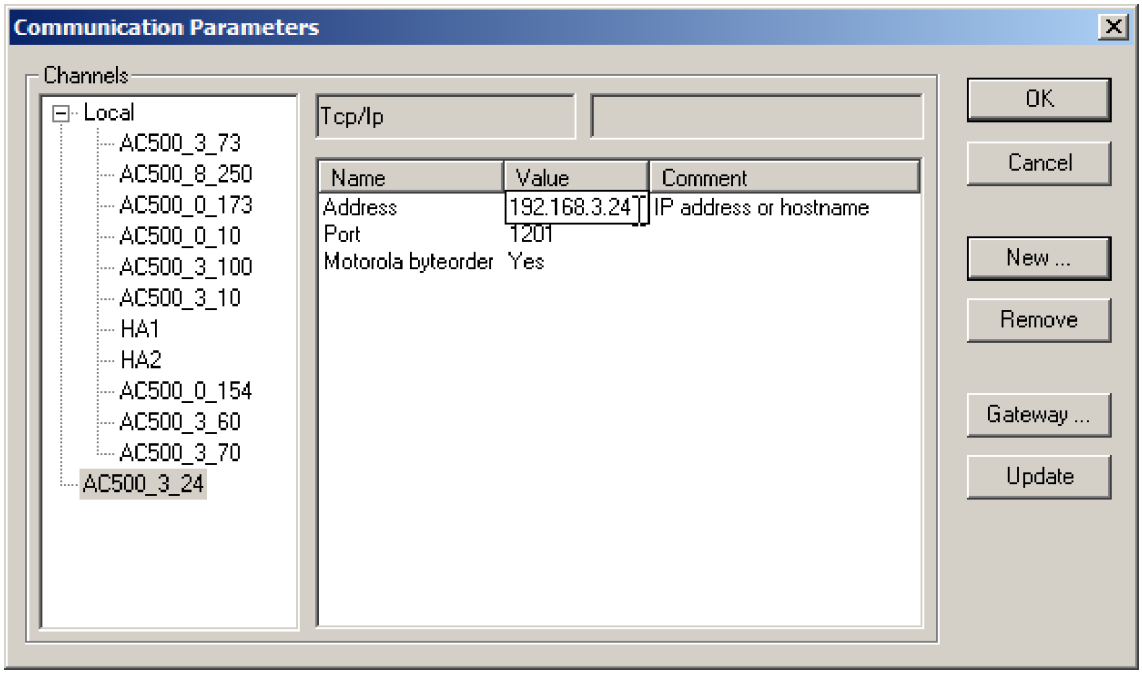


⇨

- *If the *.sdb files should be loaded from the "Gateway Files" directory on PC, the project name must be identical with project name in CODESYS. The extension is not necessary.*
- *If the symbol information should be loaded from V2.x, the project name is not required and can also be empty.*
- *The parameters displayed in the screenshot above are recommended default settings.*
- *The checkboxes Active, Motorola Byteorder and No Login-Service must be enabled.*

3. Select *"Connection"* and click *[Edit]*.
4. Choose a channel of the channel list (normally the channel which is used for programming) or click *"New"*.

5. Define a *Name*, select *"TCP/IP"* and click *[OK]*.



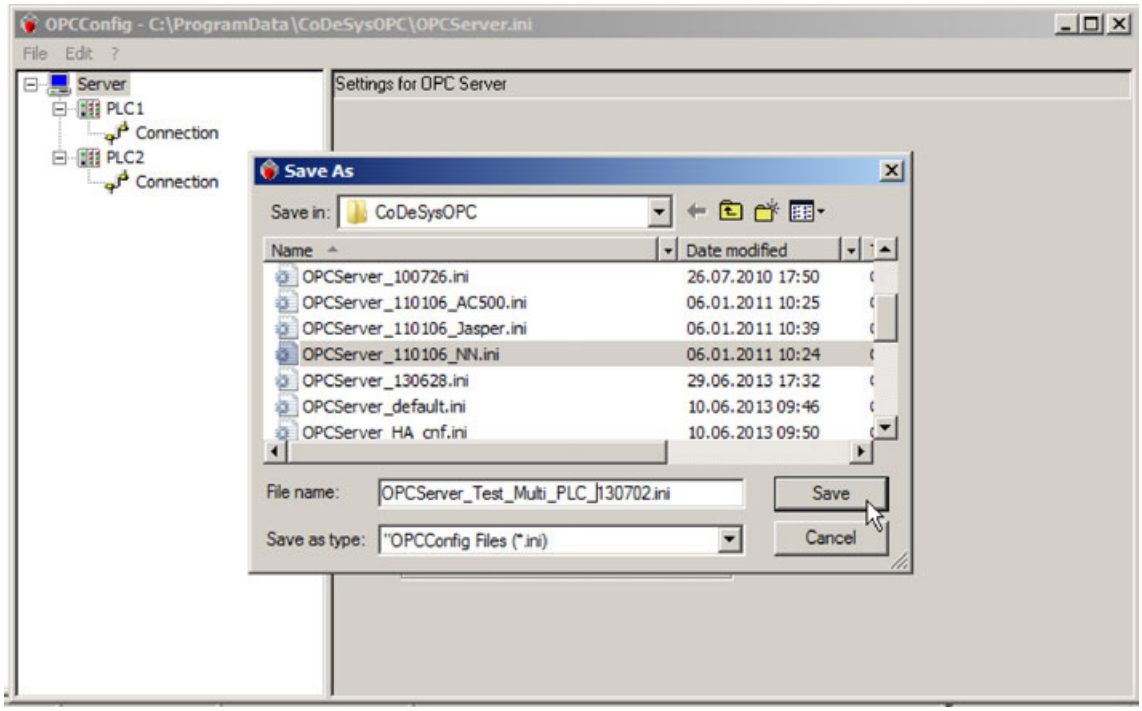
6. Under *Value*, enter the gateway address (192.168.0.10) and click *[OK]*.

If you have more than one PLC, repeat the steps 2 - 5.

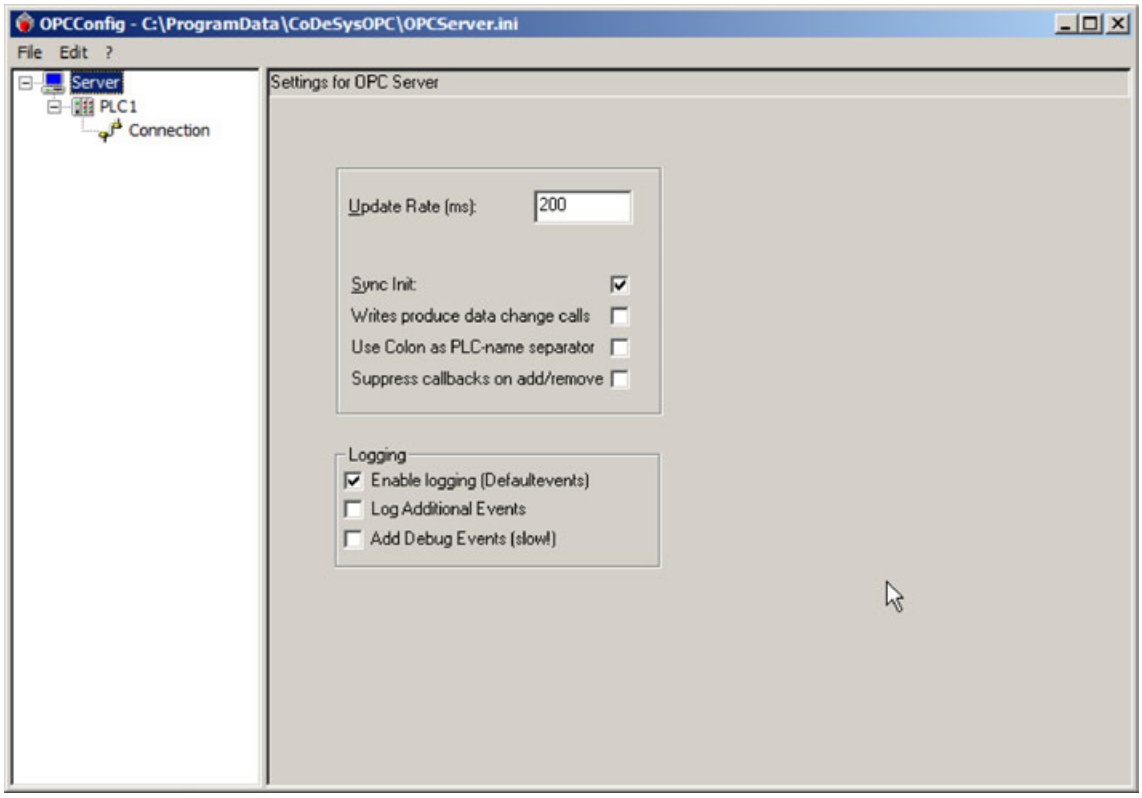
Previous settings of gateway channels are only visible, after the first time the connection has been built up.

Configure OPC Server V3

1. Start CODESYS/ CoDeSysOPC Server V3/OPC Configurator.



2. If the configuration is needed furthermore, save the configuration.



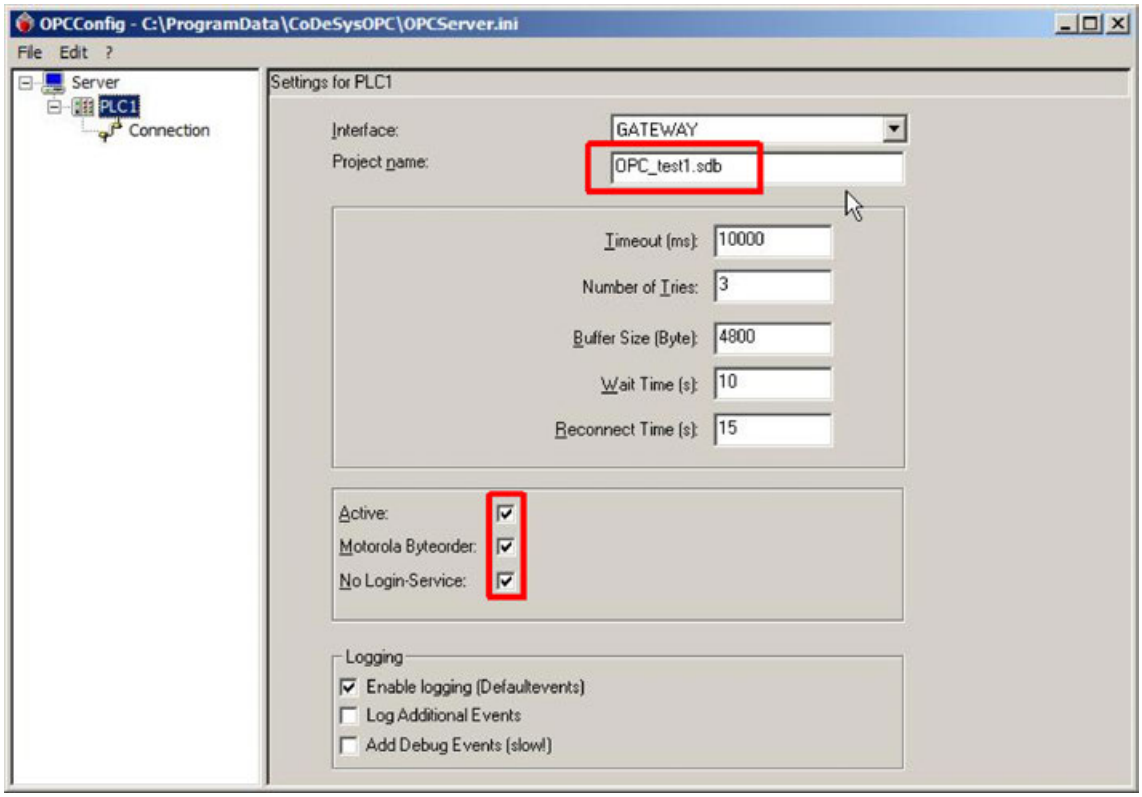
⇨

Update rate

- *The Update Rate may not be 0 (ms)!*
- *The default value of 200 ms is a suitable value of many applications.*
- *The adjustment for the Update Rate depends on the number of symbols (variables).*
- *For a big number of symbols it can be better to increase the Update Rate.*

The checkboxes Sync Init and Enable logging (Defaultevents) must be enabled.

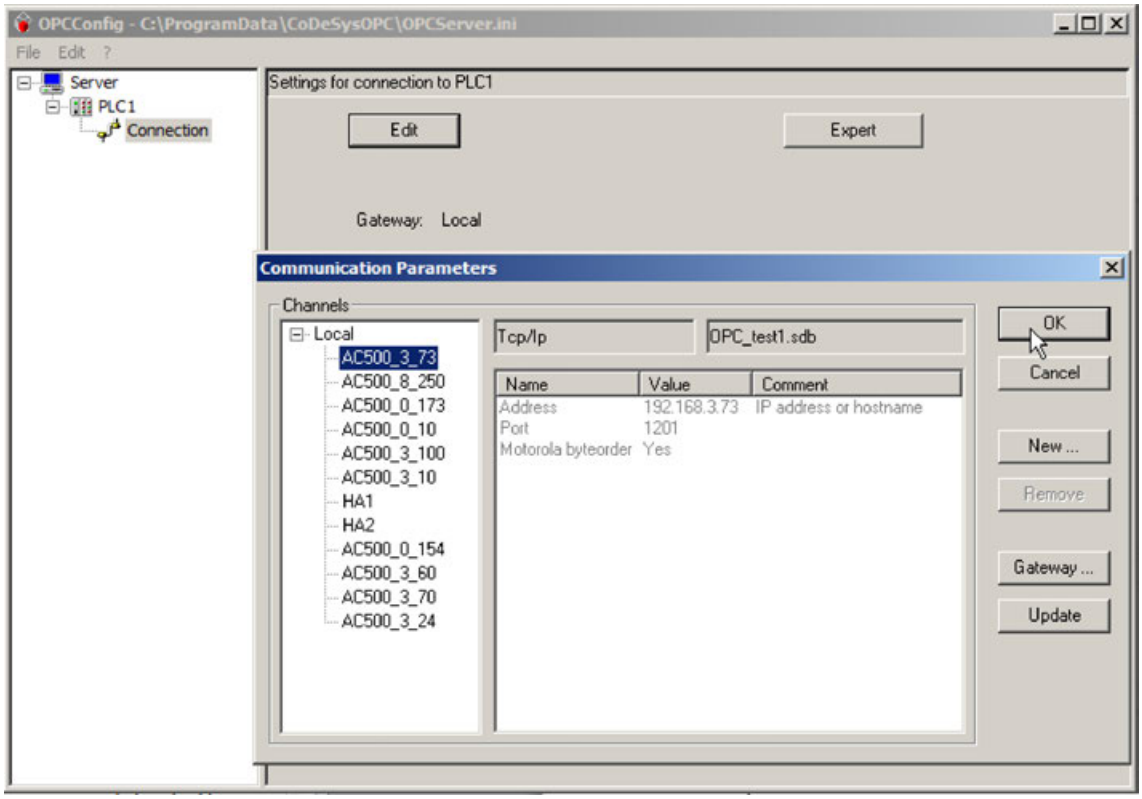
3. Select *“PLC1”*.



⇨

- *If the *.sdb files should be loaded from the “Gateway Files” directory on PC, the project name must be identical with project name in CODESYS. The extension is not necessary.*
- *If the symbol information should be loaded from AC500 V2.x, the project name is not required and can also be empty.*
- *The parameters displayed in the screenshot above are recommended default settings.*
- *The checkboxes Active, Motorola Byteorder and No Login-Service must be enabled.*
- *Enabled checkbox “Enable logging” allows a later diagnosis.*

4. Select *“Connection”* and click *[Edit]*.



5. Choose a channel of the channel list (normally the channel which is used for programming) or click *[New]*.

6. Define *Name* select *“TCP/IP”* and click *[OK]*.

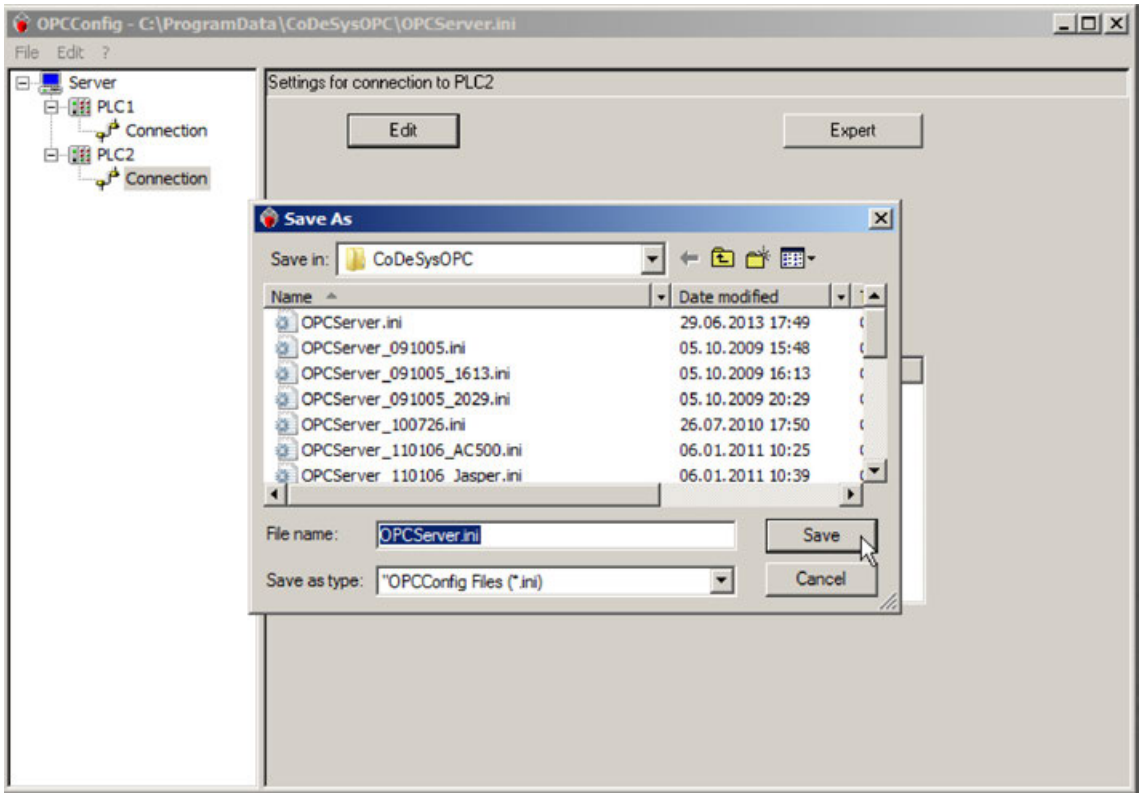
⇨

– *Previous settings of gateway channels are only visible, after the first time the connection has been built up.*
– *See Ref 5 Online Help of PS501 CoDeSys, Help, Contents, System Technology, OPC* *Table on page 6272.*
– *Use of the CODESYS OPC server.*
– *Configure the OPC server with OPCconfig.exe.*

If there are more than one PLC, then repeat for the other PLCs (gateway depends on version of AC500).

7. Save As.

   Click *“File ➔ Save as...” OPCserver.ini “Save”* and *“File ➔ Exit” OPCConfig*.



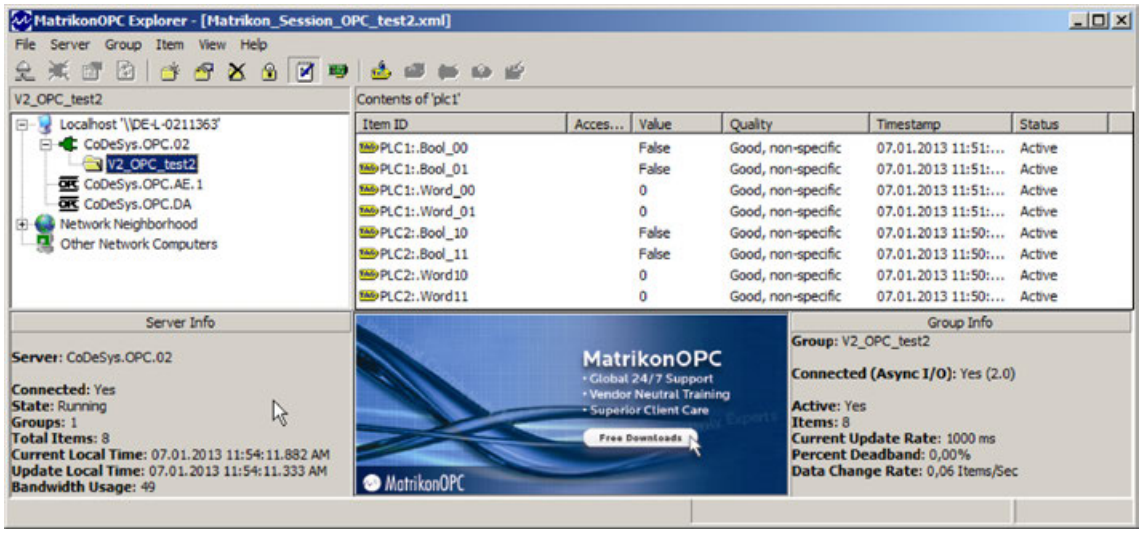
Check OPC function with AC500

It is urgently recommended to check the function of the previous configuration steps.

In order to check the OPC function without AC500, see ↳ Chapter 1.6.5.5.1.4.1 “Test OPC function without AC500” on page 6309.

Check OPC server V2

1. Start *OPCExplorer.exe* and connect *“CoDeSys.OPC.02”*.

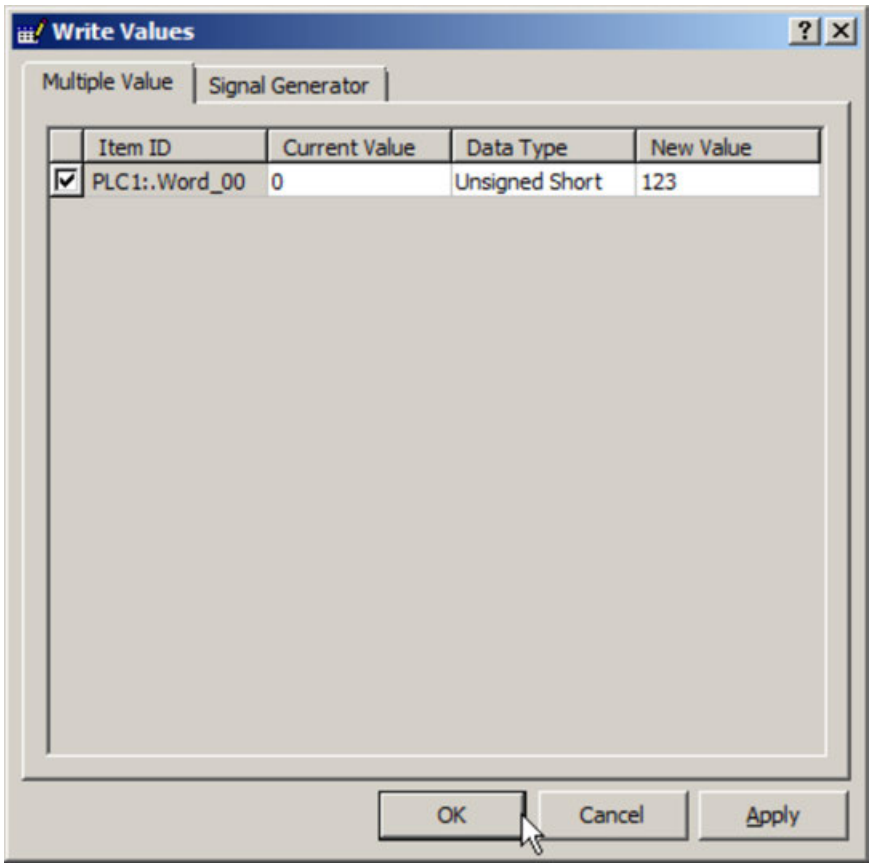


2. Add Group, add Items, select availabe Items in *“Server CoDeSys.OPC.02”*.

   Add to Tag List, close the .

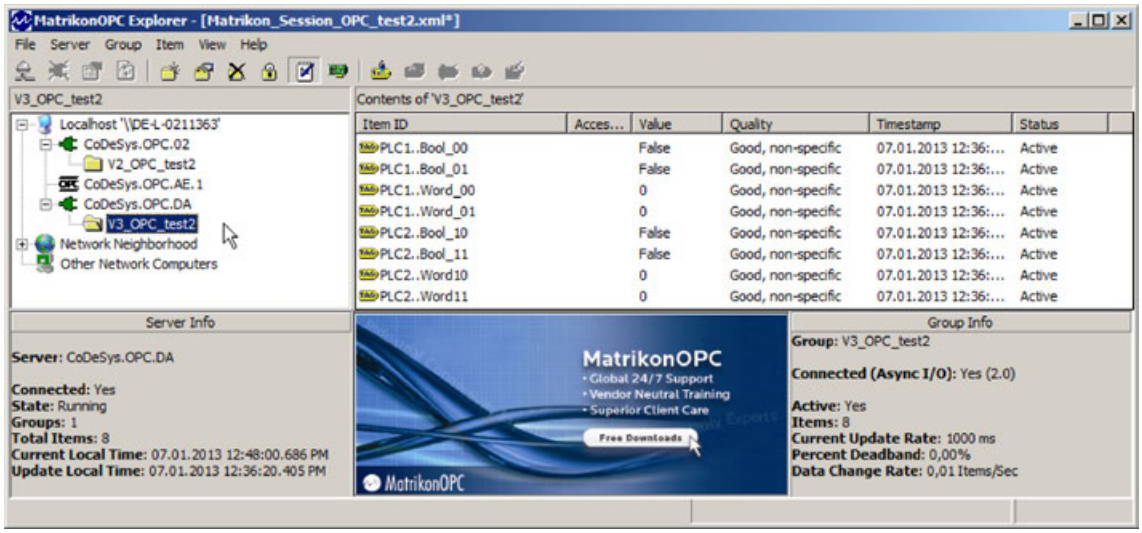
   ⇨ If anything is right, then *“CoDeSys.OPC.02”* is connected, is running and the *“Quality”* of the items is good.

3. With the *“Matrikon”* is it possible to read / write the values of the items.



Check OPC server V3

1. Start *OPCExplorer.exe* and connect *“CoDeSys.OPC.DA”*.

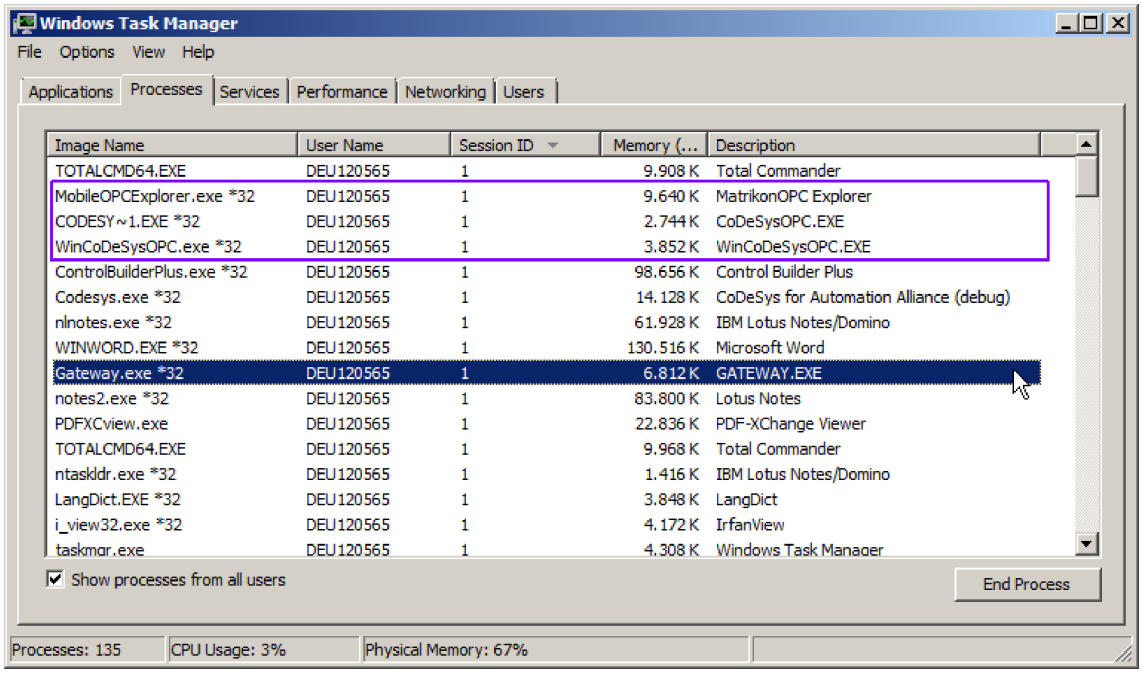


2. Add Group, add Items, select availabe Items in Server *“CoDeSys.OPC.DA”*.

   Add to Tag List, close the Item browser.

   ⇨ If anything is right, then *“CoDeSys.OPC.DA”* is connected, is running and the *“Quality”* of the items is good.

Check processes with windows task manager



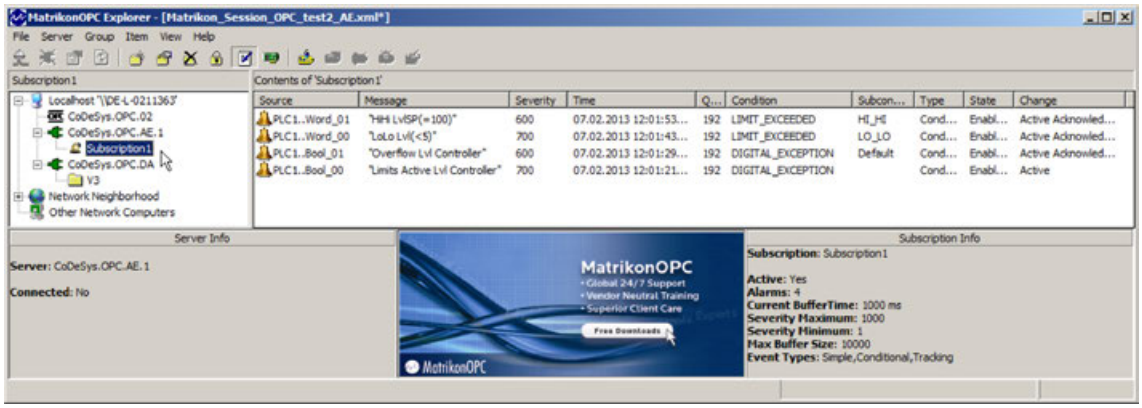
Correct configuration: All processes run with the same *“User Name”* and with the same *“Session ID”*.

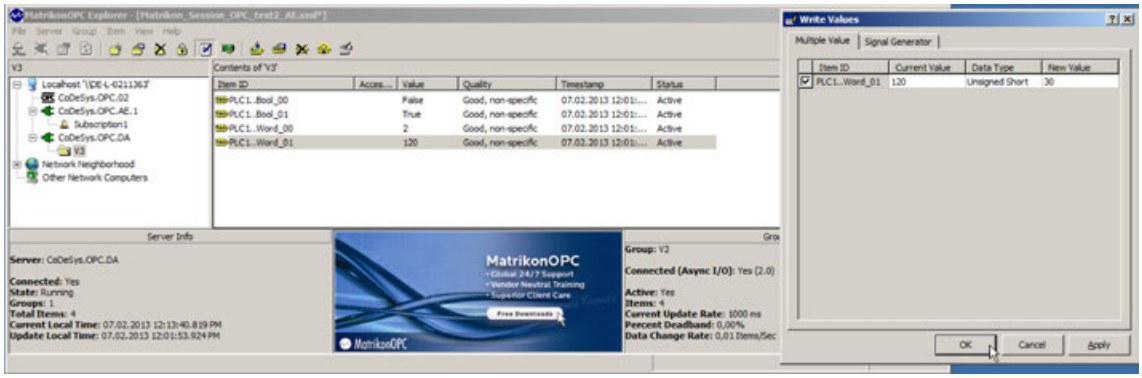
Configure AlarmEvents

Refer to REF2 AeConfigurator_UserGuide.pdf for details ↳ *Table on page 6272.*

Check AlarmEvents

The function of the *“AlarmEvents”* can be checked with *“MatrikonOPC Explorer”*.





The *“AlarmEvents”* can be simulated by writing the value of the Items.

Configure user account for OPC server

Please refer to REF3 ReadMe.rtf and REF4 ReleaseNotes OPCV3 AE for HA ↳ *Table on page 6272.*

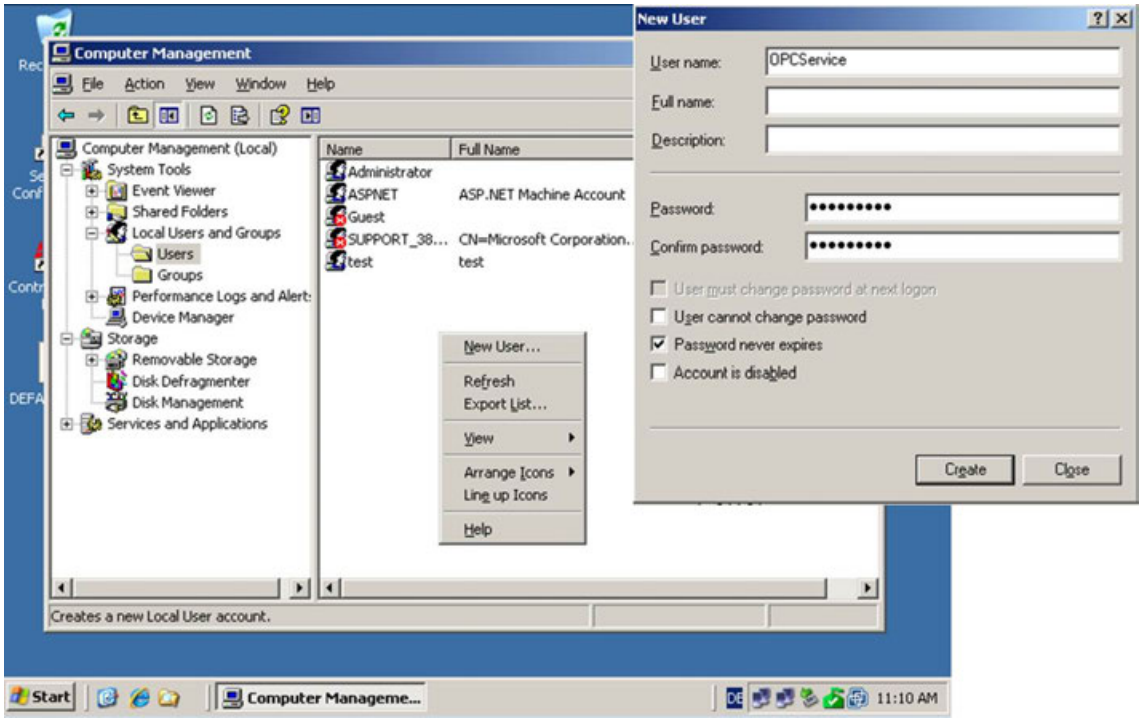
OPC server V3 on Windows Server 2003/ 2008/ 2012/ 2016

When running the OPC server V3 on Windows Server 2003/ 2008/ 2012 /2016 multiple sessions need to be supported. Therefore the installation of the OPC server as service running with a dedicated user account is recommended.

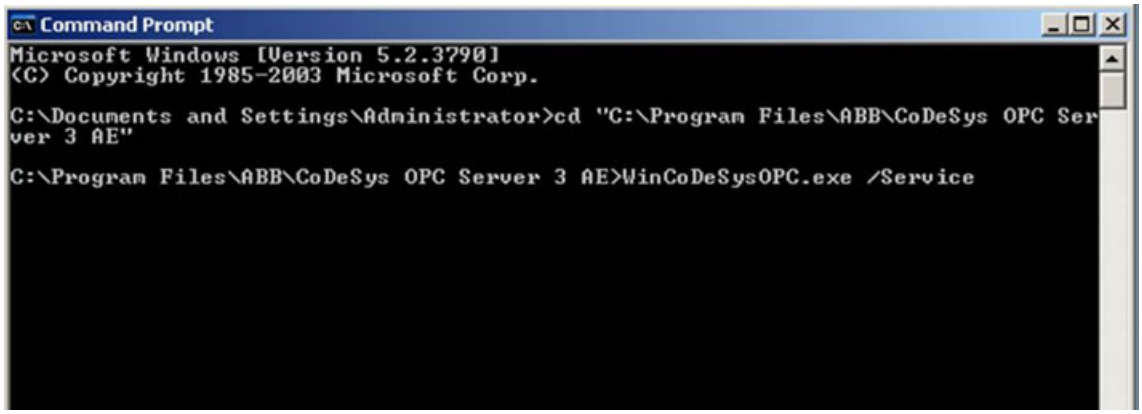
Configuration steps

- Create specific user, no administrator account is required
- Register V3 OPC server as service
- Configure V3 OPC server as service

Create specific user



Registration



Register the OPC Server executable as service from the command line.

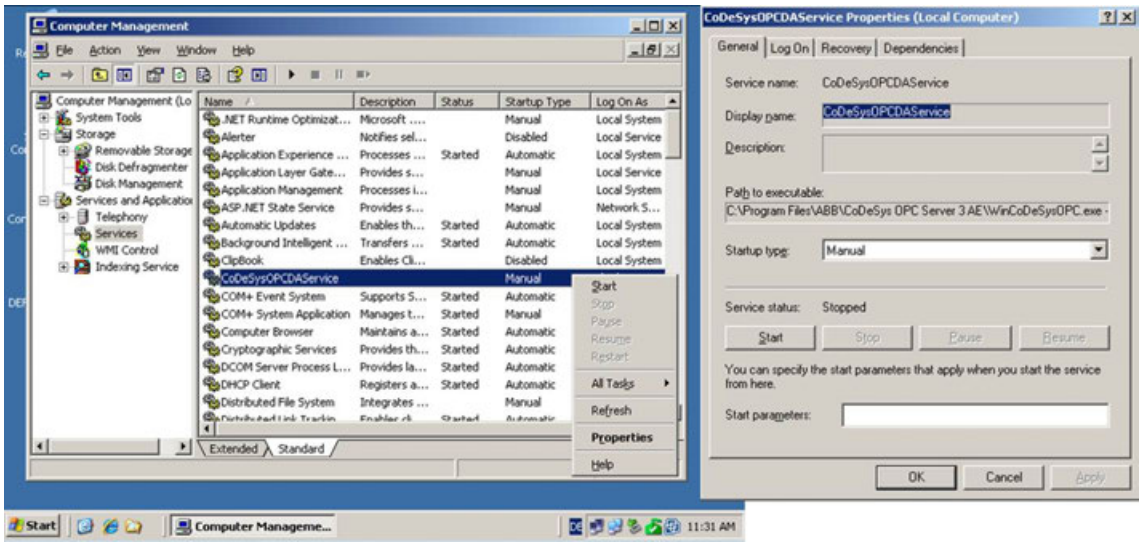
With command “WinCoDeSysOPC /Service” WinCoDeSysOPC.exe gets installed as system service.

Started once, the service will stay “started” until the system gets terminated.

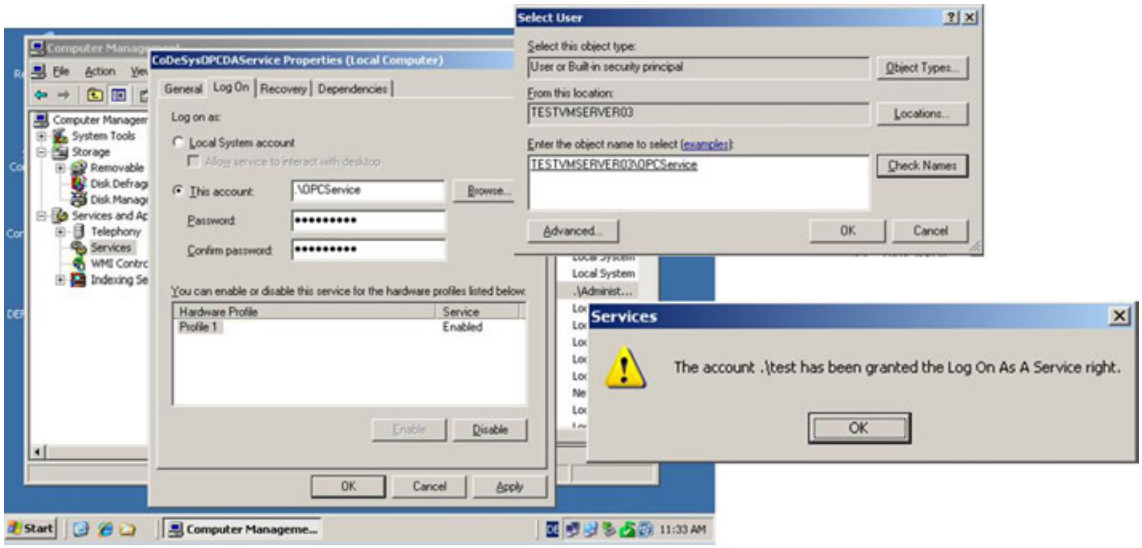
The communication to the configured PLCs survives.

Also here the service gets installed in the current position of WinCoDeSysOPC.exe.

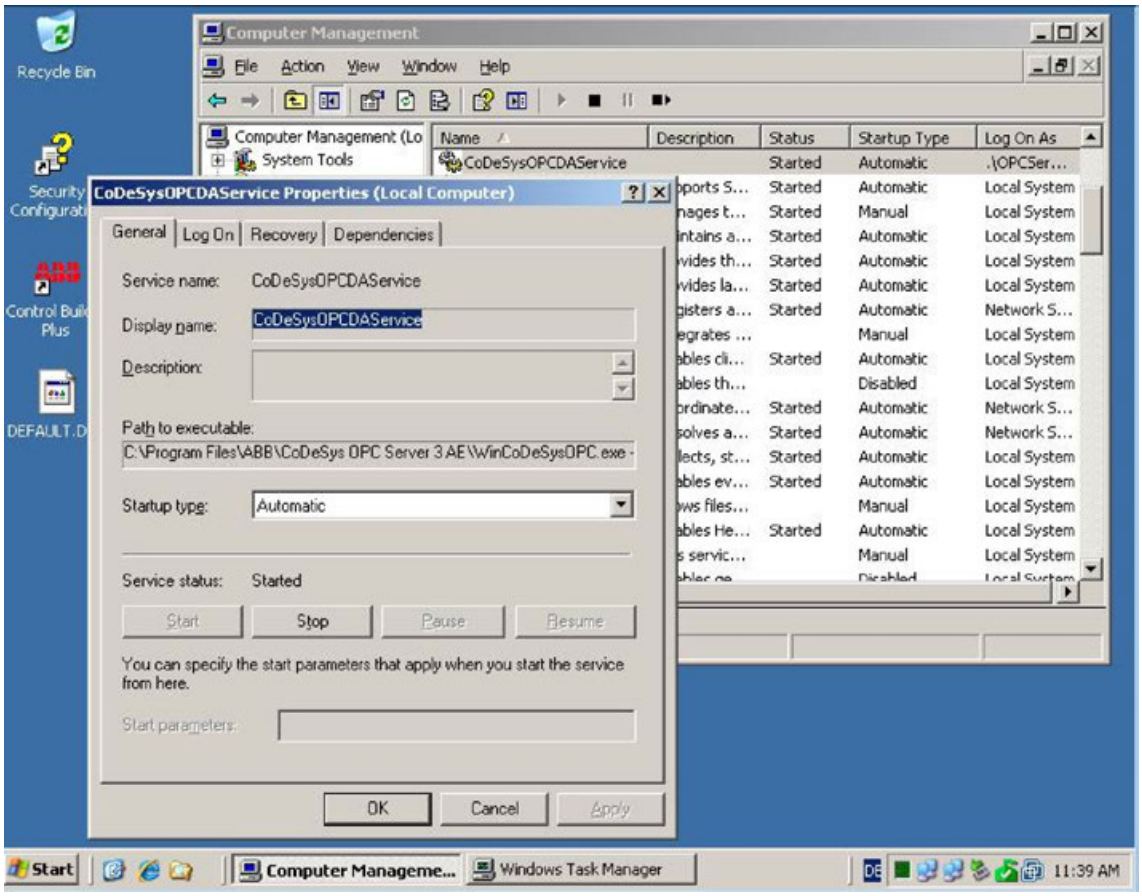
Configuration



At *“Computer Management* ➔ *Services and Applications* ➔ *Services”* open the *“Properties”* of the *“CoDeSysOPCDAService”*.

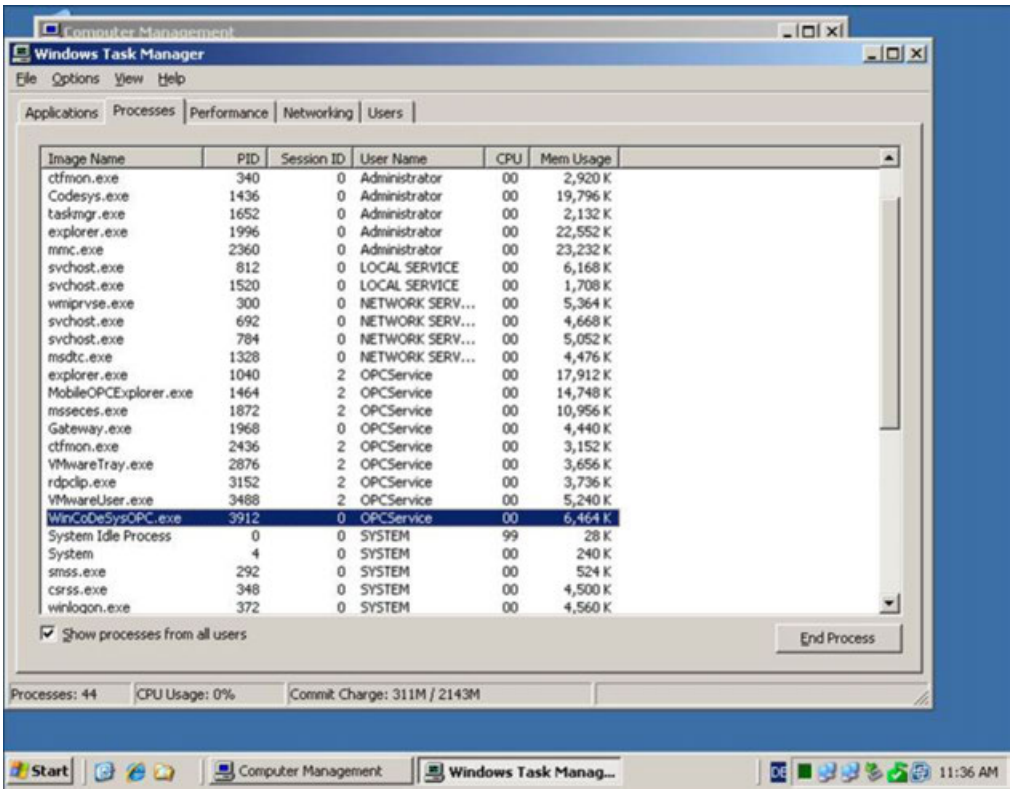


Complete the Service Configuration



Testing

Check Users and Session during Test Cases



Check the *“Session ID”* and *“User Name”* of

- *Gateway.exe*,
- *WinCoDeSysOPC.exe* and
- OPC Client

on different test cases like multi session with terminal service sessions.

Potential issues

Gateway communication not possible

The CODESYS Gateway server uses TCP port 1210 for communication.

> – *The gateway communication is not possible if gateway ports 1210 and 1211 are used by other applications.*

Possible applications that also use port 1210 and/or 1211:

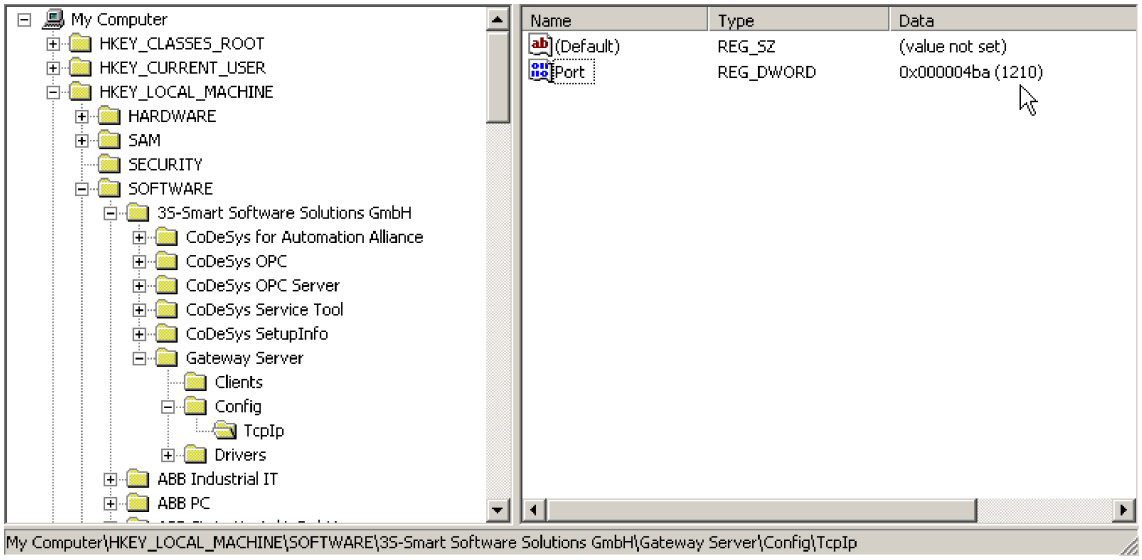
- Java update client
- ABB 800xA System

If there are problems to establish a gateway communication, check the usage of port 1210 (via any port scanning tool, e.g. *SysInternals*) and close the application which uses this port.

ABB 800xA system

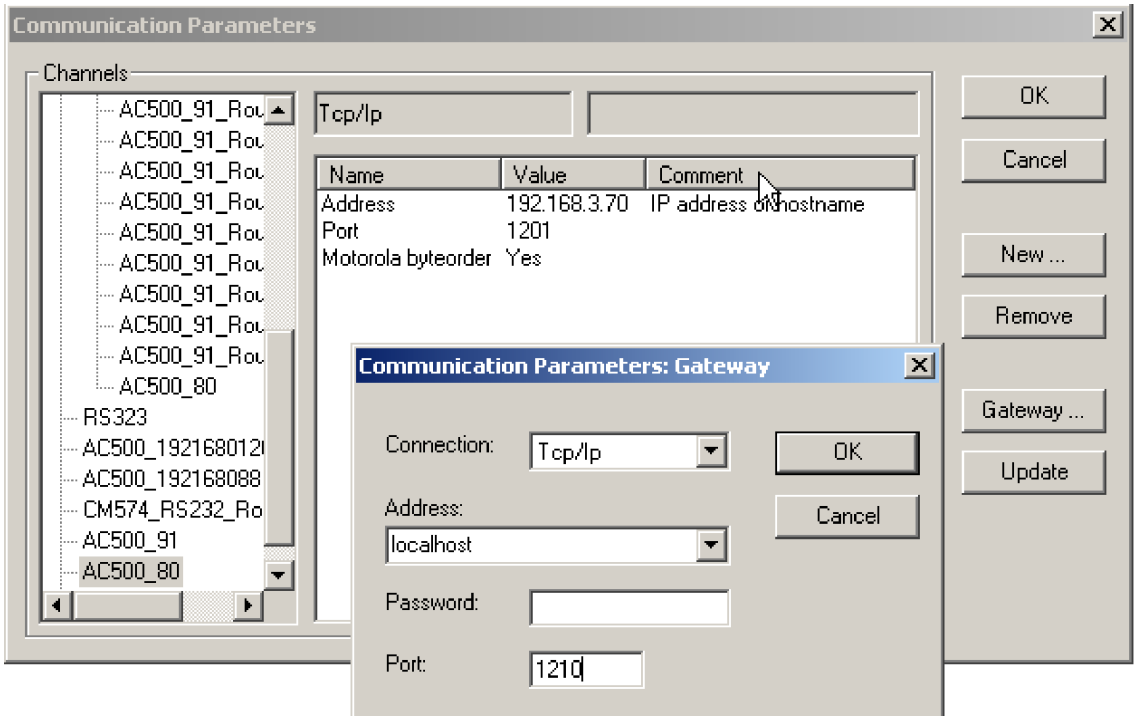
Change port number of ABB 800xA System:

1. Open the registry editor and double-click *“Port”*.



2. Change the *“registry key”* from 1210 to 51000 and click *[OK]*.
3. Restart the *“Gateway Server”* and close all applications.

4. Close the processes *gateway.exe* in Windows Task Manager.



5. Change the port number of Gateway communication parameter from standard port number 1210 to 51000 (in the example).
6. The OPC Server Configuration (*OPCConfig.exe*) must be renewed to enter this change in the ↳ *Chapter 1.6.5.5.1.2.6.2 “Configure OPC Server V3” on page 6292OPCServer.ini*.

Windows Server 2012/ 2016

At Windows Server 2012 /2016 (64-bit) the path for the Reg Key is:

“HKEY_LOCAL_MACHINE ➔ Software ➔ WOW6432Node ➔ 3S-Smart Software Solutions GmbH ➔ Gateway Server ➔ Config ➔ TcpIp”

Symbol file from AC500 not loaded

Situation

OPC server is shown, but no OPC variables are to be found.

OPC with symbol file on AC500 does not work.

OPC server does not load the symbol file (*.sdb*) from AC500 PLC (FW V2) to PC

Possible reason

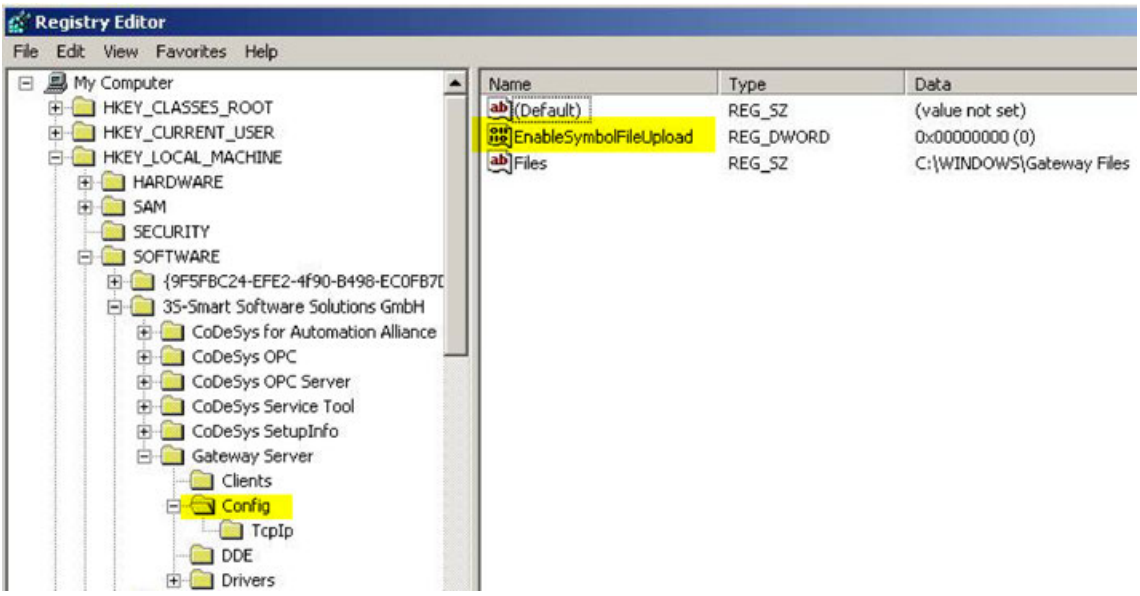
- Programming Software *"907AC1131"* is installed.

Problem solving

1. Check the registry item: *"HKEY_LOCAL_MACHINE\ SOFTWARE\ 3S-Smart Software Solution GmbH\ Gateway Server\ Config\ EnableSymbolFileUpload"*.

   If this item is inside, the symbol file will not be loaded from AC500 PLC to PC. For Control Builder Plus this item must be deleted but for AC1131 this item must be available. You must ckeck this.

2. In Windows, go to *"Start ➔ Run"* type *"regedit"* and click *[OK]*.



3. Open the folder *"Config"* in *"Registry Editor"*.
   - For AC500 FW V2 delete the item *"EnableSymbolFileUpload"*.
   - For AC1131 the item *"EnableSymbolFileUpload"* must be available.

Session isolation

Situation

With Windows Server 2003, Windows Server 2008, Windows Server 2016 the Windows 7 services are alone in session 0. User applications run in session 1 (2 and so on).

Services:

A Windows service is a computer program that operates in the background.

Windows services can be configured to start when the operating system is started or can be started manually and run in the background as long as Windows is running. They can operate when a user is not logged on.

Services are:

Windows operating systems include numerous services. OPC client like S+ OPC scanner PGIM, Aspen CIM-IO Manager, ICONICS, .. can also installed as a service.

User applications are:

Microsoft Word, Notepad, MatrikonExplorer, *ControlBuilderPlus.exe* and *Codesys.exe*

Problem

Service and user application are isolated in their session. They can not communicate with each other directly.

OPC Server uses, like the CBP and CODESYS, the gateway server from CODESYS (*gateway.exe*) for the communication with the AC500 and starts the gateway in their session. That creates undefined behavior, if the OPC Server runs as a service. The gateway server is not able to run in multi sessions.

Resolutions

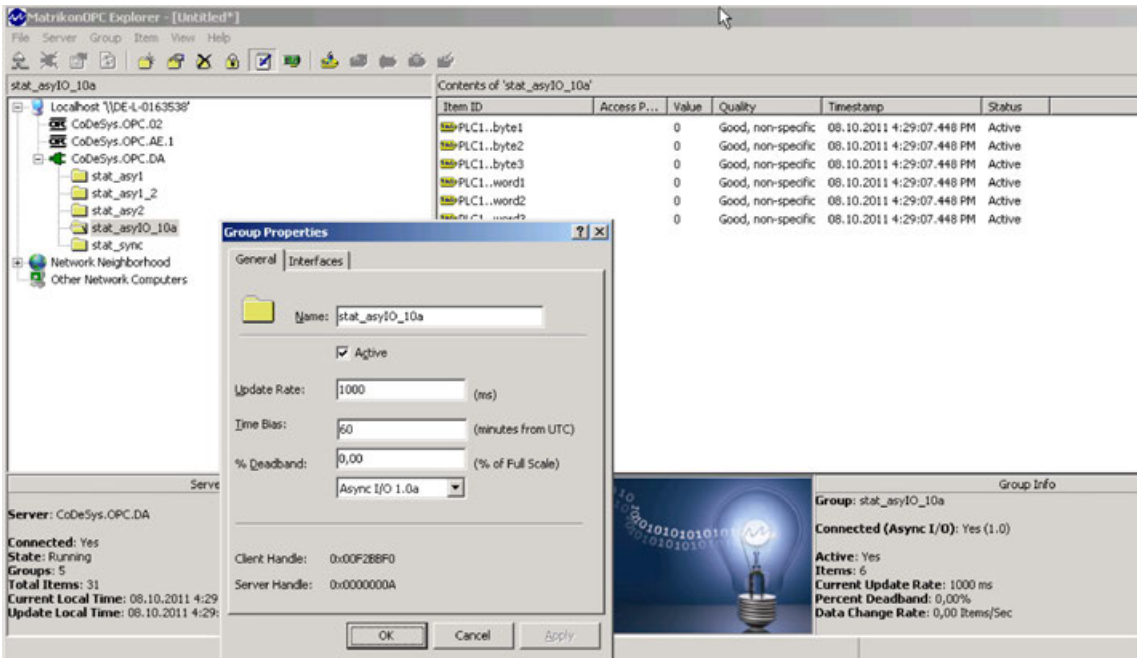
- Install all OPC clients and OPC Server, which use the gateway server, in the same session.
- The OPC Server as a service (session 0) may not be connected at the same time (in parallel) with an OPC server as a user application or CBP or CODESYS (all in session 1) with the AC500. If this function is necessary, different PC or virtual machines must be used.
- Use tools like OPC tunnel. In a DigiVis 500 setup context the OPC server must not be registered as service. The OPC tunnel itself starts the OPC server within its service.

See also http://msdn.microsoft.com/en-us/windows7trainingcourse_sessionisolation_unit .

Behavior OPC server V3 via interface IOPCAsyncIO

Using of an *"OPC client"*[1)] with the older OPC standard Interface IOPCAsyncIO (OPC DA V1.0a) creates a higher communication load on the OPC client, because the OPC Server sends also the unchanged items in every scan cycle to the client.

Test setup



Reason

If OPC Items are registered via Interface IOPCAsyncIO (OPC DA V1.0a), the OPC Server sends mostly with each ready cycle a data change event, including also unchanged values. The change detection is correct when using the interface IOPCAsyncIO2 (OPC DA V2).

Workaround

- Use the interface IOPCAsyncIO2 (OPC DA V2).
- If the OPC client does not support IOPCAsyncIO2 interface, then use the OPC Server V2. The OPC Server does not show this behavior.

"OPC client" [1)]:

Visualization software in VISU PMS (Fa. epro GmbH) uses an older standard OPC with the interface IID_IAdviseSink than data sink.

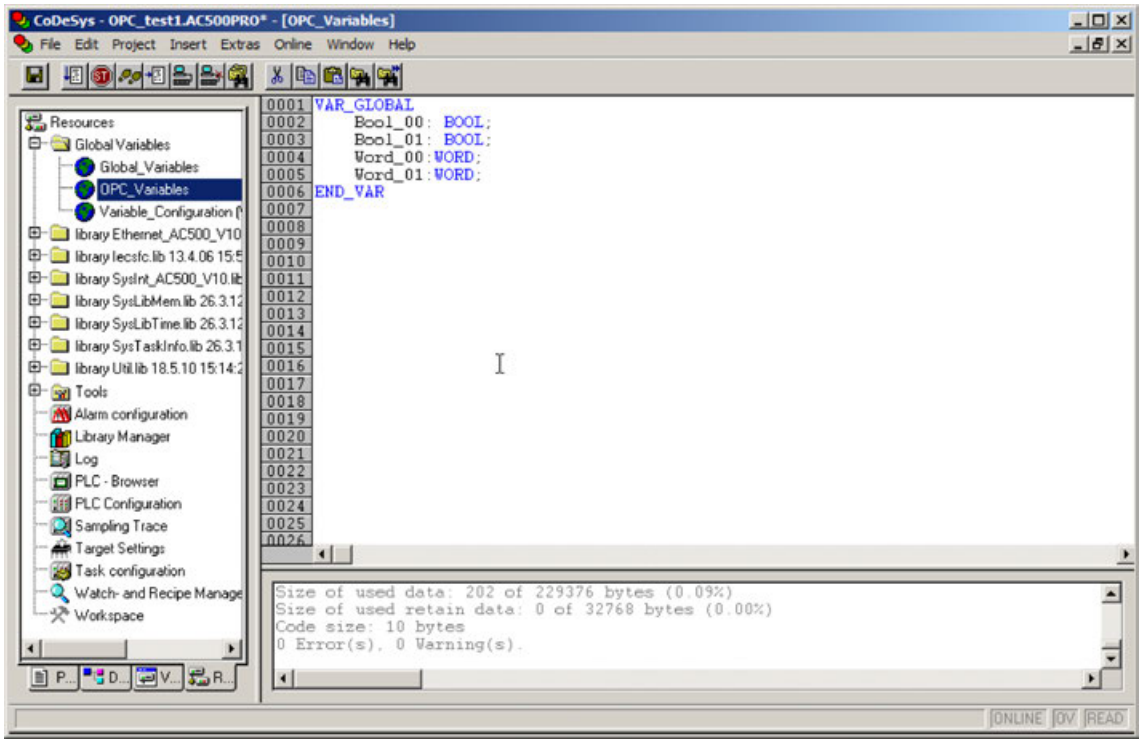
Examples

Test OPC function without AC500

The example shows, how the OPC server V2/V3 can be tested/simulated without available AC500.

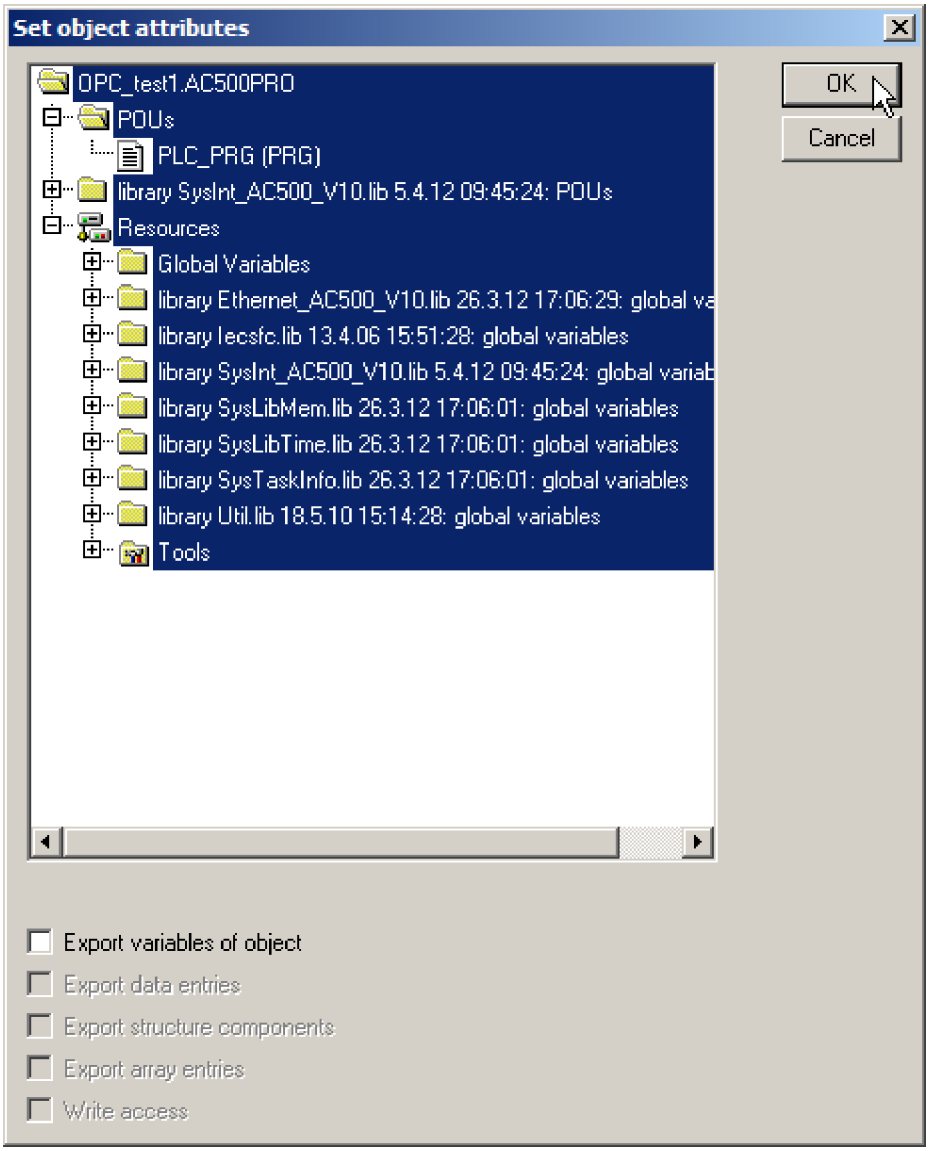
AC500 project

1. Open *CoDeSys Application*.



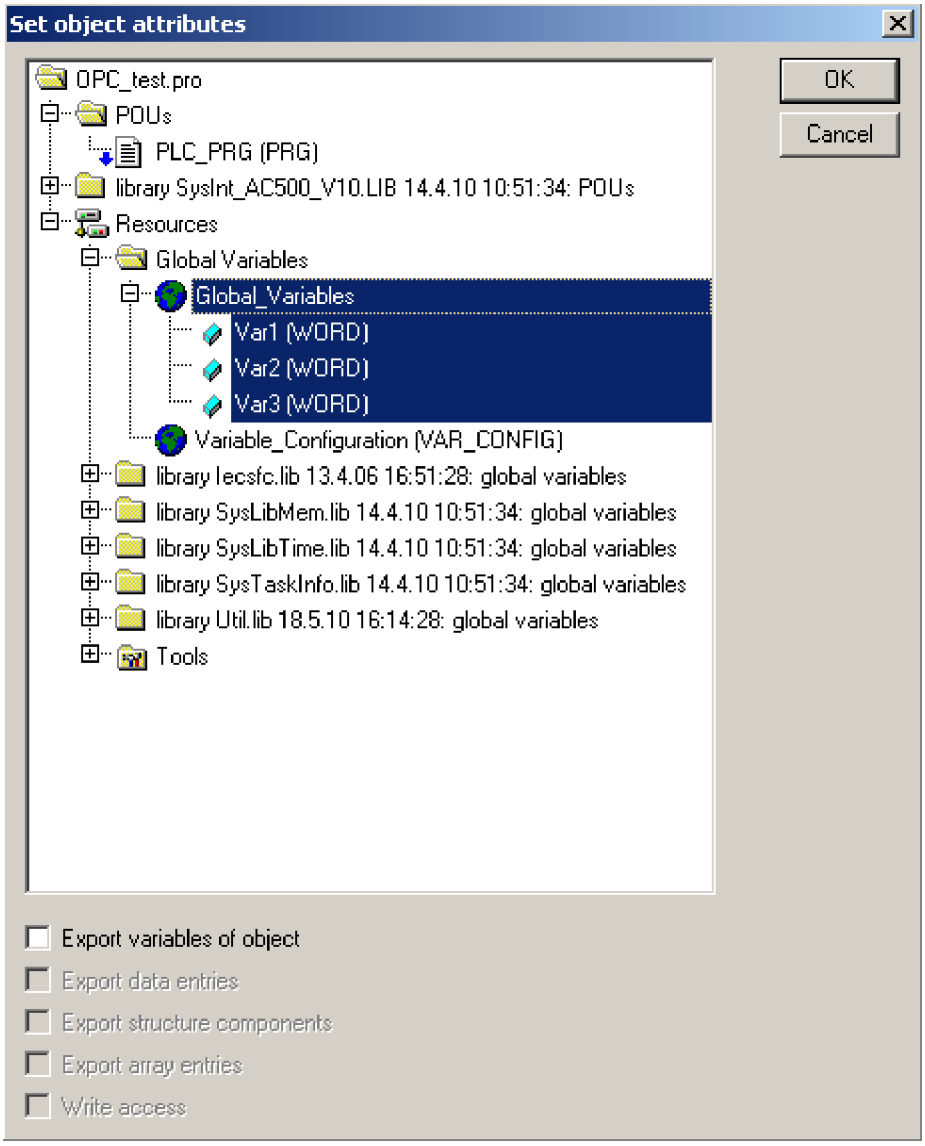
2. Collect all OPC variables in a separate *“Global Variables”* list.

3. Under *"Project ➔ Options "* select the *"Symbol configuration"*.
   Enable checkbox *"Dump symbol entries"* and click *[Configure symbol file]*.



4. Disable all the checkboxes and confirm twice with *[OK]*.

5. Under *"Project ➔ Options " "Symbol configuration"* click *[Configure symbol file]* again.

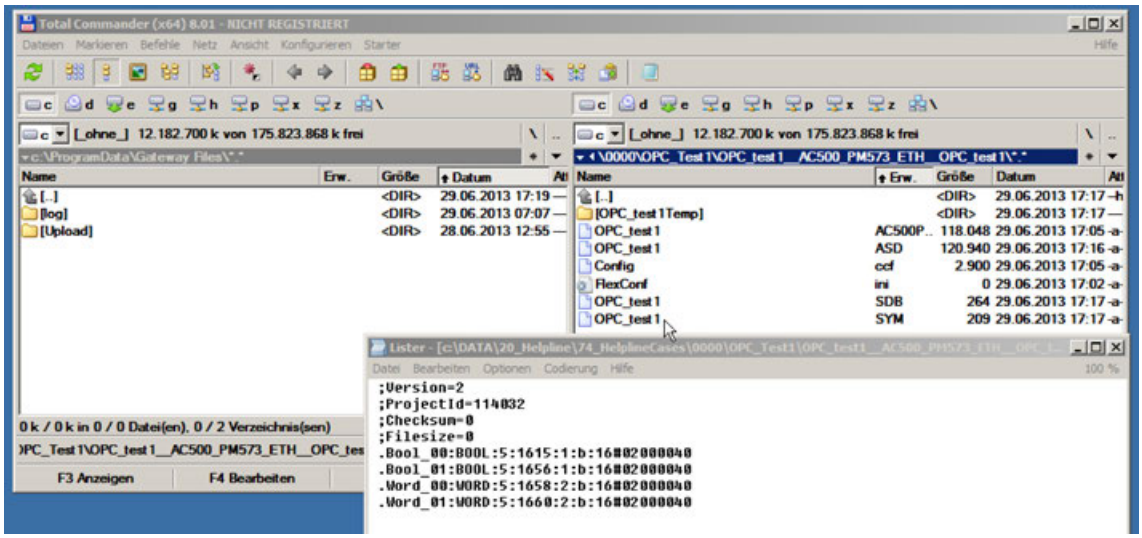


6. Select the variables which should be communicated as symbol.

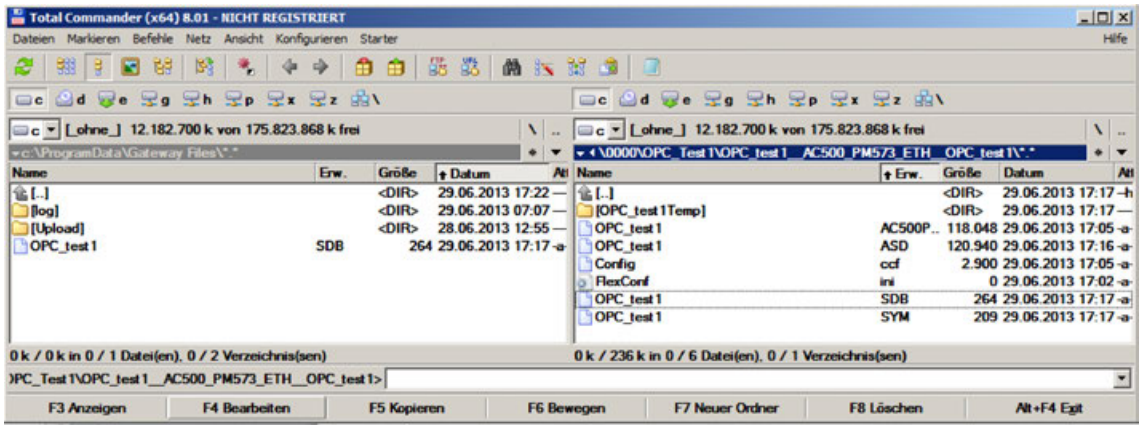
   Enable the following checkboxes:

   - *Export variables of object*
   - *Export structure components*
   - *Export array entries*
   - *Write access*

7. Confirm twice with *[OK]*.
8. Under *"Project ➔ Rebuild all"* rebuild the project.

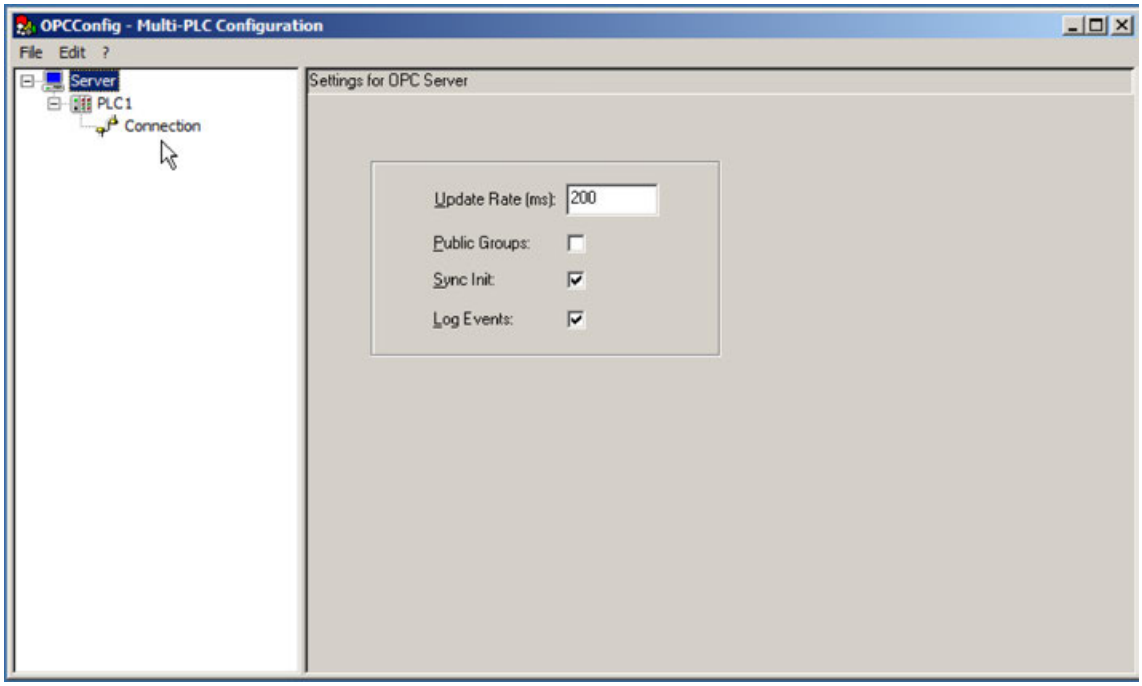


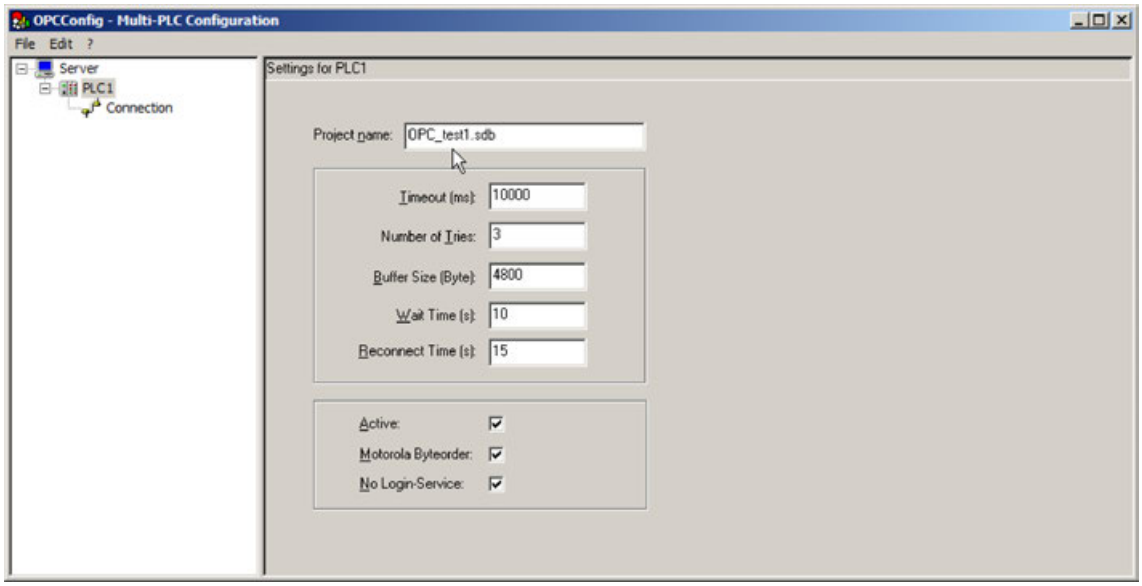
9. In the project folder is the subfolder *"OPC_test1__AC500_PM573_ETH__OPC_test1"*. It contains symbol files **.SYM* and **.SDB* with the time of the *"Rebuild all"*. The items in the file **.SYM* can be checked with Notepad. The binary file **.SDB* contains the items for the OPC server. With *<Online>* *<Login>* it will copied in the gateway files directory and optionally on the AC500.



10. The folder *"OPC_test1__AC500_PM573_ETH__OPC_test1"* is a temporary folder, if the CBP project is opened. For the simulation of the server OPC it is copied **.SDB* by hand.

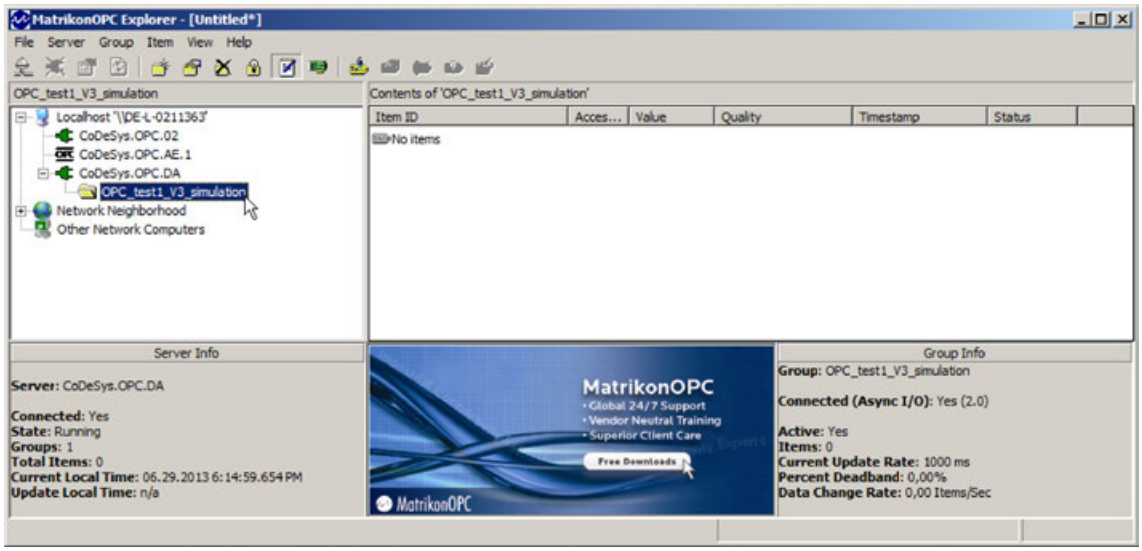
Configure OPC server V2



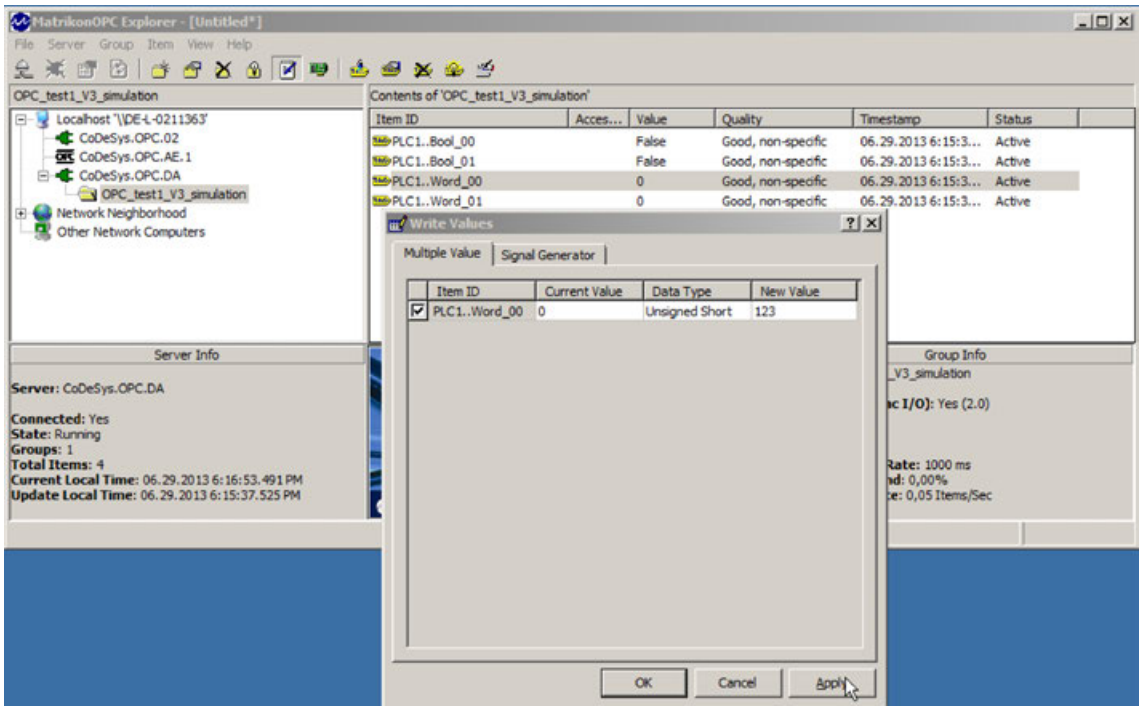


▷ Only the *"Project name"* may be specified.

Check OPC server with MatrikonOPCExplorer

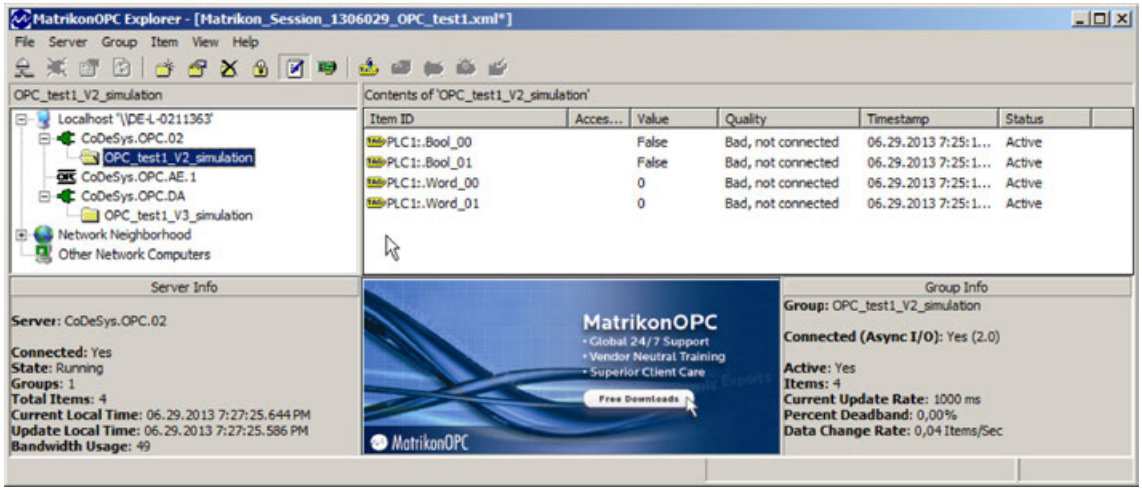


1. OPC Server V3: *"Connect CoDeSys.OPC.DA"*. Add *"Group"*, add *"Items"*, select *"Availabe Tags"* and add to *"Tag List"*.



2. The OPC Server V3 (*"CoDeSys.OPC.DA"*) is connected, running and the *"Quality"* is good.

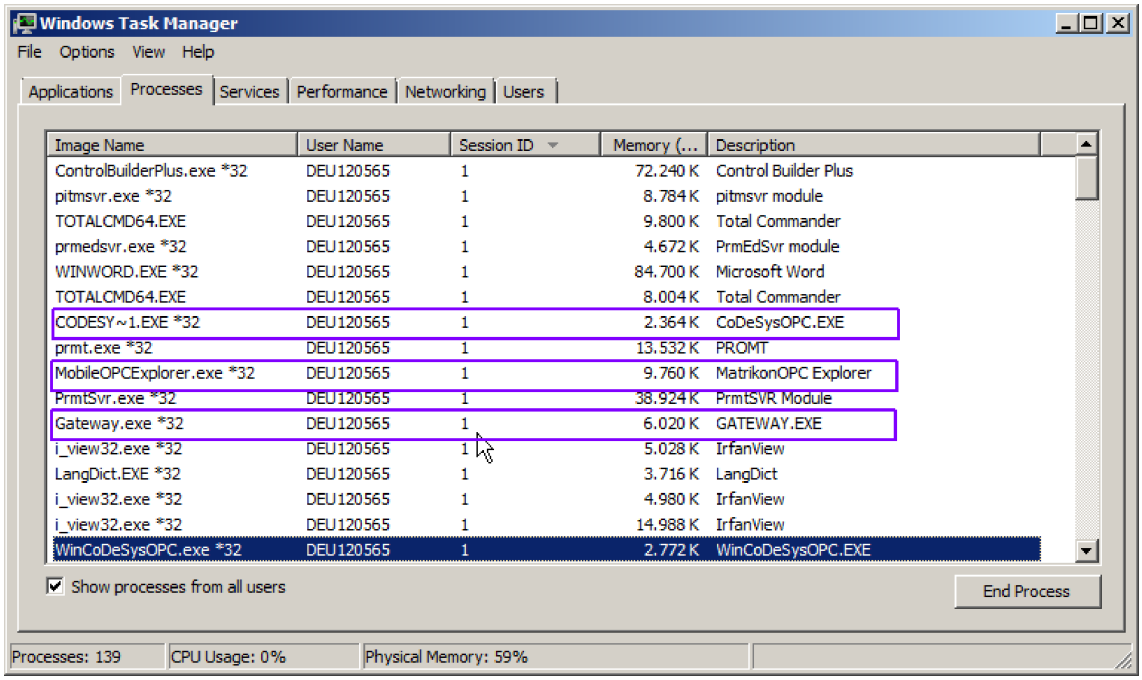
   One OPC client can read / write the values of the items.



3. Similar configuration as above.

   The OPC Server V2 (*"CoDeSys.OPC.02"*) is connected, running and the configured items are found. But the *"Quality"* is bad. One OPC client can not read / write the values of the items.

Check processes with windows task manager



Correct configuration: All *"Processes"* run with the same *"User Name"* and with the same *"Session ID"*.

Summary

The correct function of OPC Server V2 and V3 can be checked without AC500.

With OPC Server V3 with the configuration “SIMULATION” the Project name with the directory name has to be specified. The values of the items can be read and write by one OPC client.

With OPC Server V2, as well as with OPC Server V3 in configuration “GATEWAY”, only the project name may be specified. The configured items are found, but the quality is bad. The values of the items can not be read and not write by one OPC client.

Refer to REF5 Online Help of PS501 chapter OPC for details ↬ Table on page 6272.

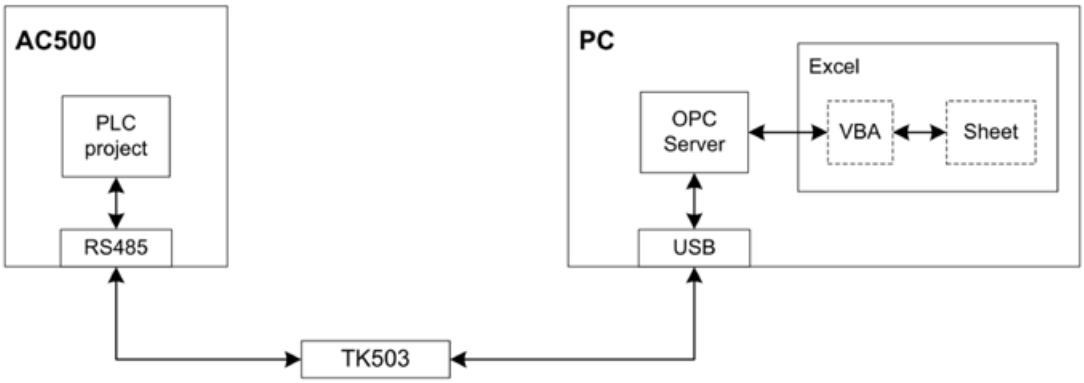
Create an OPC client with Microsoft Excel

How do you create an OPC client with Microsoft Excel?

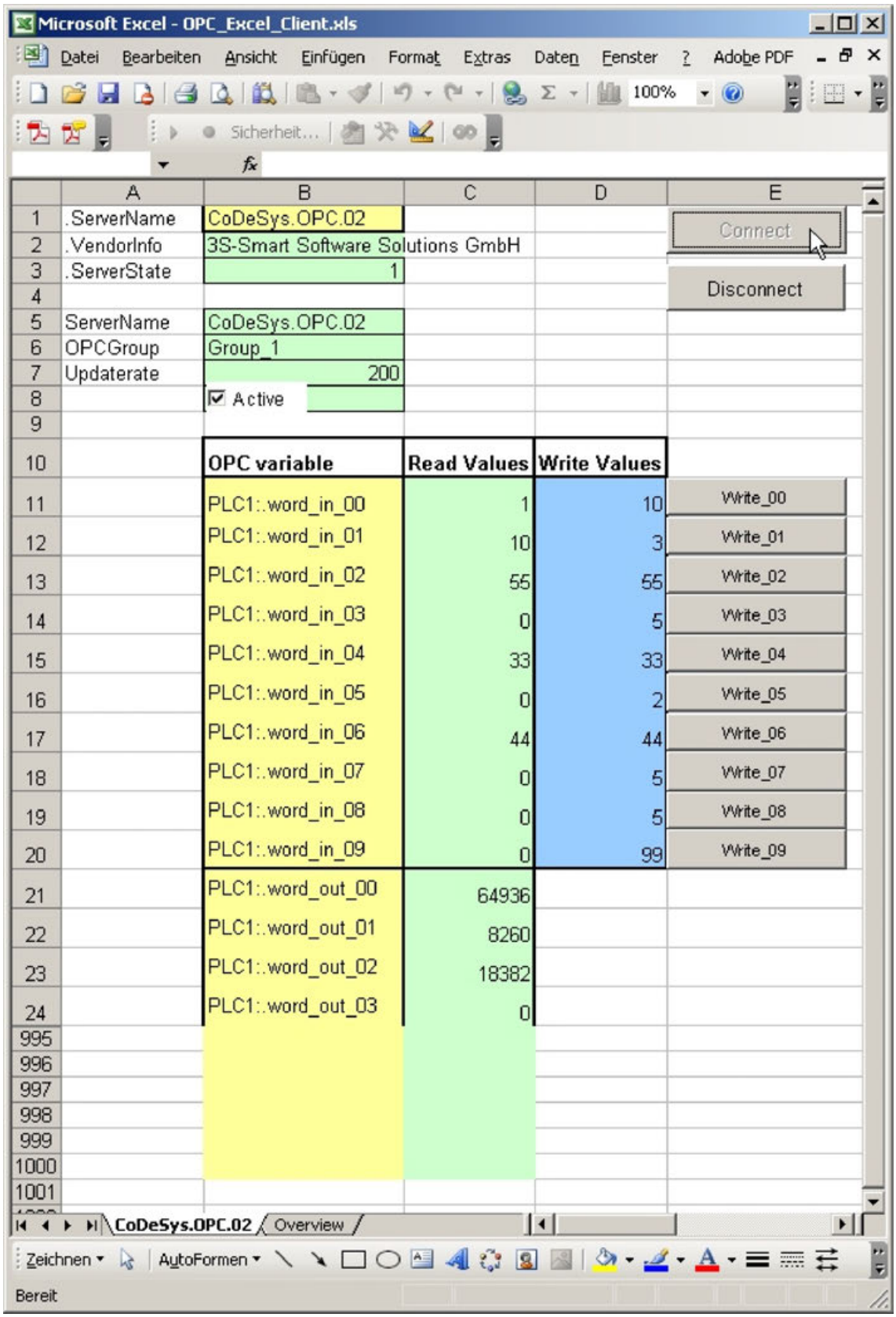
See *Application Example, OPC*.

This application example consists of two parts:

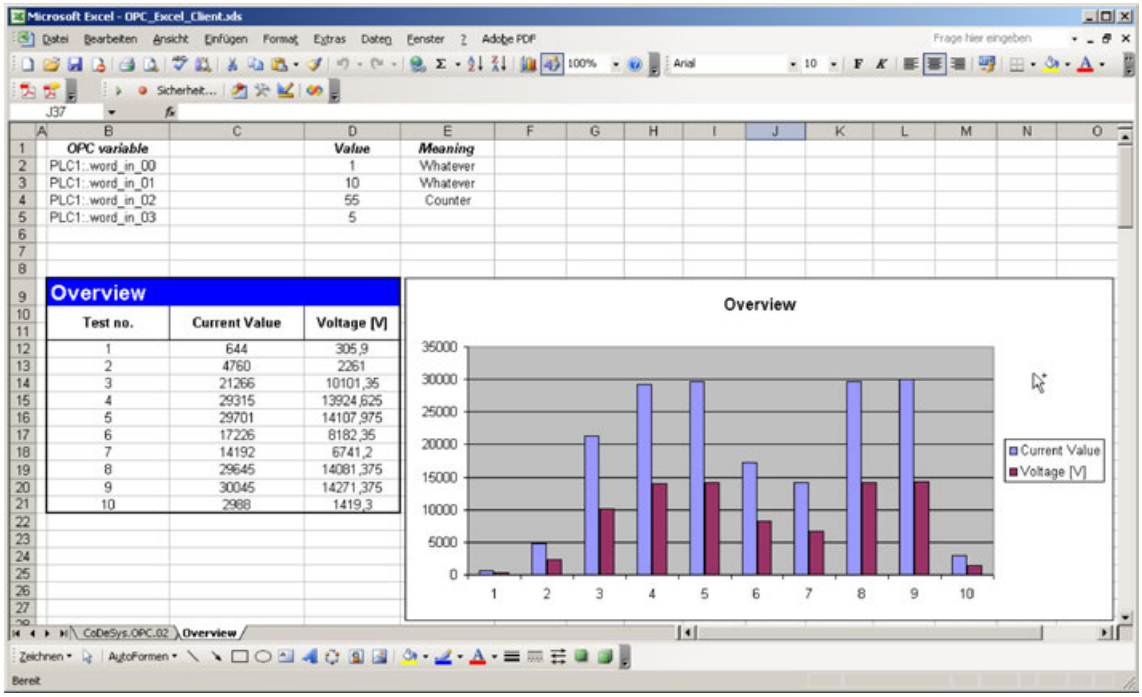
- AC500_to_OPC_Excel_Client.pro: AC500eCo project with symbol and CoDeSysOPC
- OPC_Excel_Client.xls: MS Excel sheet with VBA program



Block Diagram



Worksheet *"Control panel"* for the communication with the OPC-Server.



Worksheet *"Overview"* for visualization.

This works also with OPC Server V3 but because of a missing DLL the OPC Server V2 must be installed also.

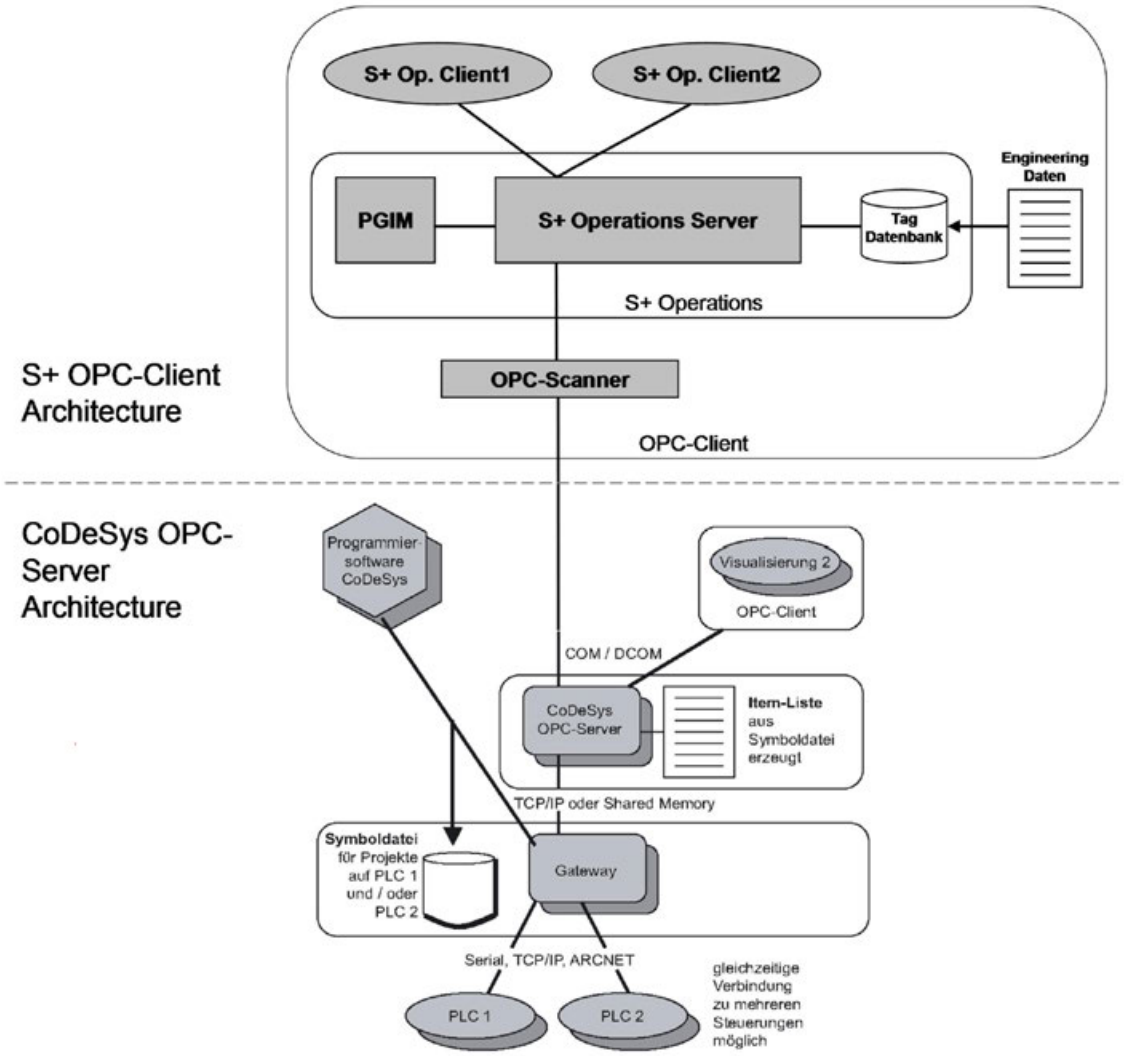
It will be fixed in later Releases as V2.3.

OPC server V3 with S+

Setup

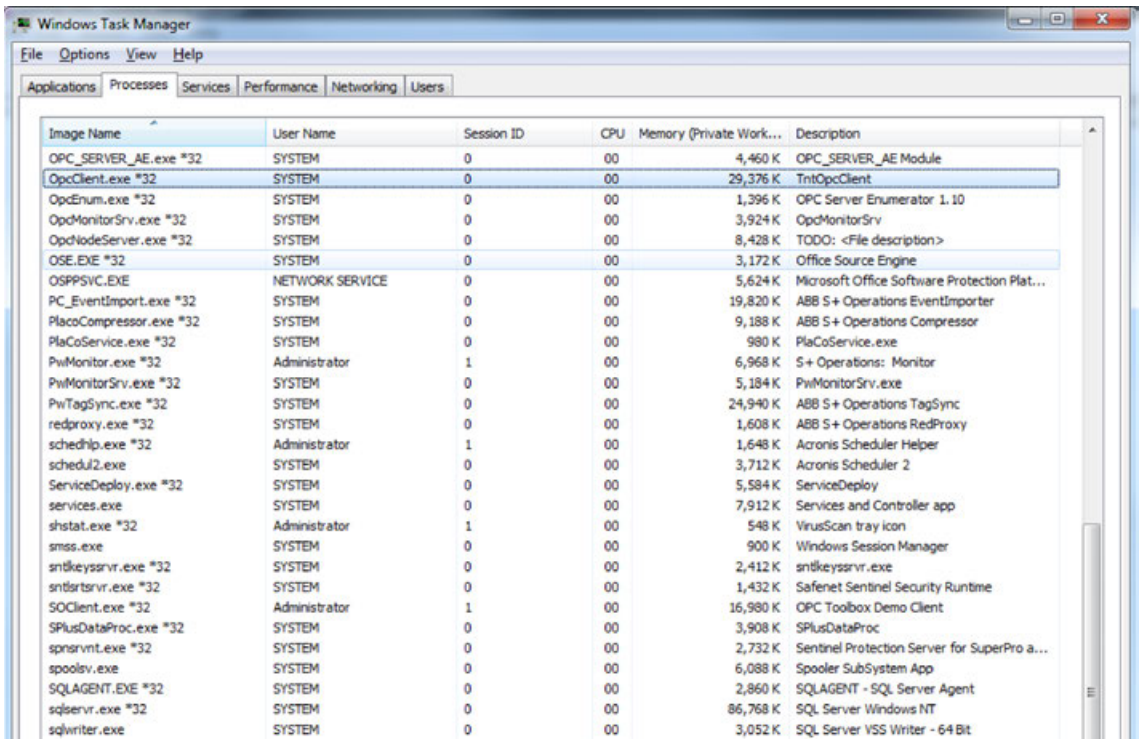
Test with ABB PS Mannheim, 2012. Communication via OPC with AC500.

- Windows Server 2008 64 Bit
- S+
- OPC Server V3



Procedure

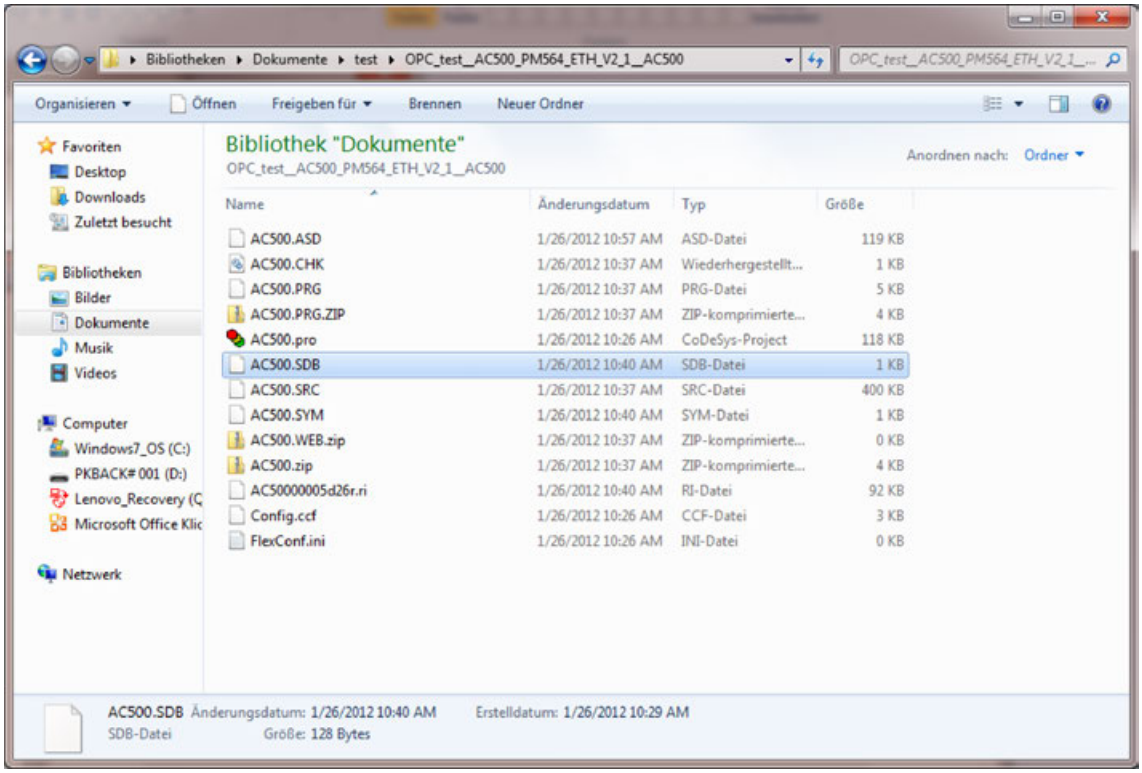
1. Install OPC-Server V3 from folder CBP.
   ⇨ After the installation OPC server runs in session ID: 1
2. Test with Testclients, as *Softing* or *Matrikon OPC*, if the data is able to be called up.
   ⇨ The S+ OPC-Scanner runs as a service.
3. Configure OPC Server V3 according to Hints ↳ *Chapter 1.6.5.5.1.2.6.2 “Configure OPC Server V3” on page 6292*.
4. Configure User account for OPC server ↳ *Chapter 1.6.5.5.1.2.8 “Configure user account for OPC server” on page 6299*.
   ⇨ The OPC server runs then in session ID: 0.



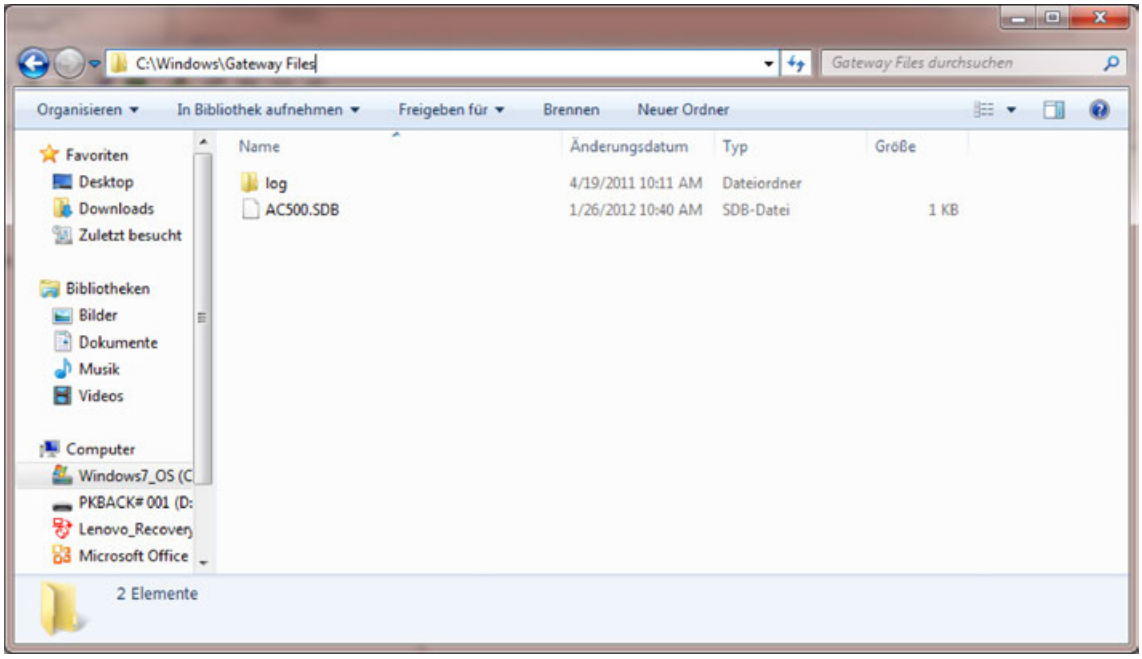
Win7 64 bit OPC server V3 symbol file local

1. Create a new project and take a look at your symbol file.

   The project must be opened to see this file.

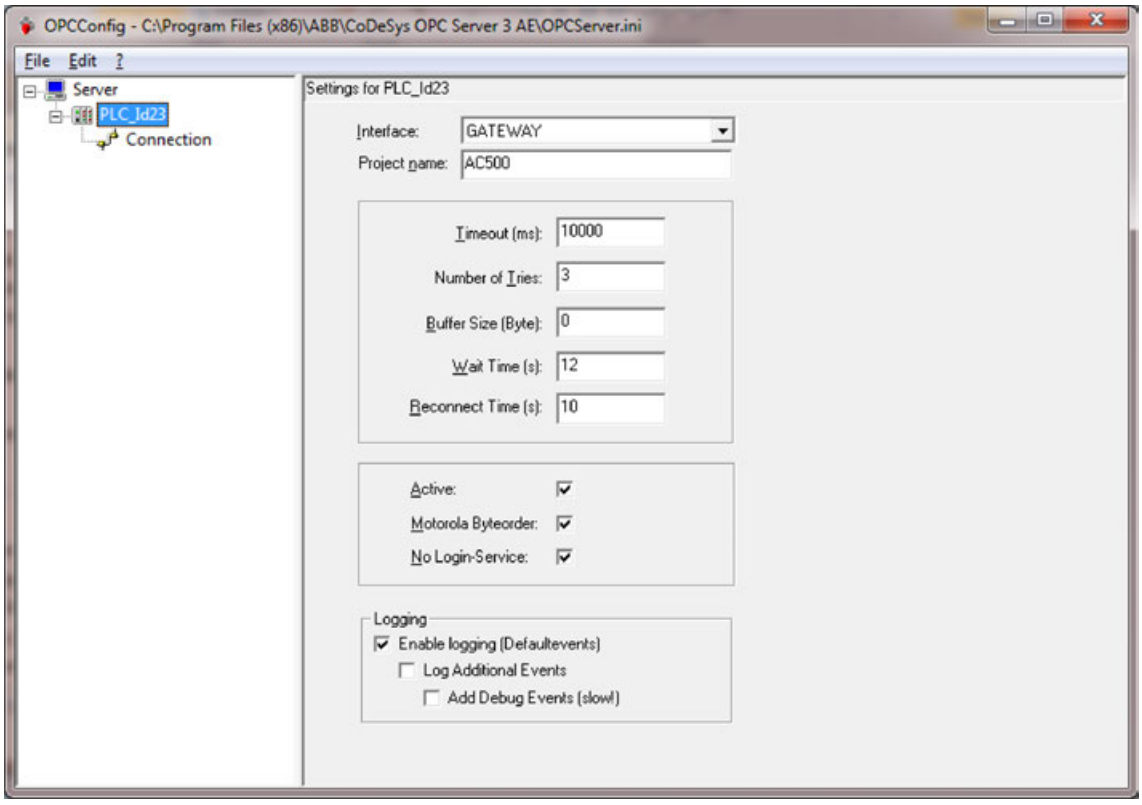


2. Copy your **.sdb* file to the following folder *"C:\Windows\Gateway Files"*.

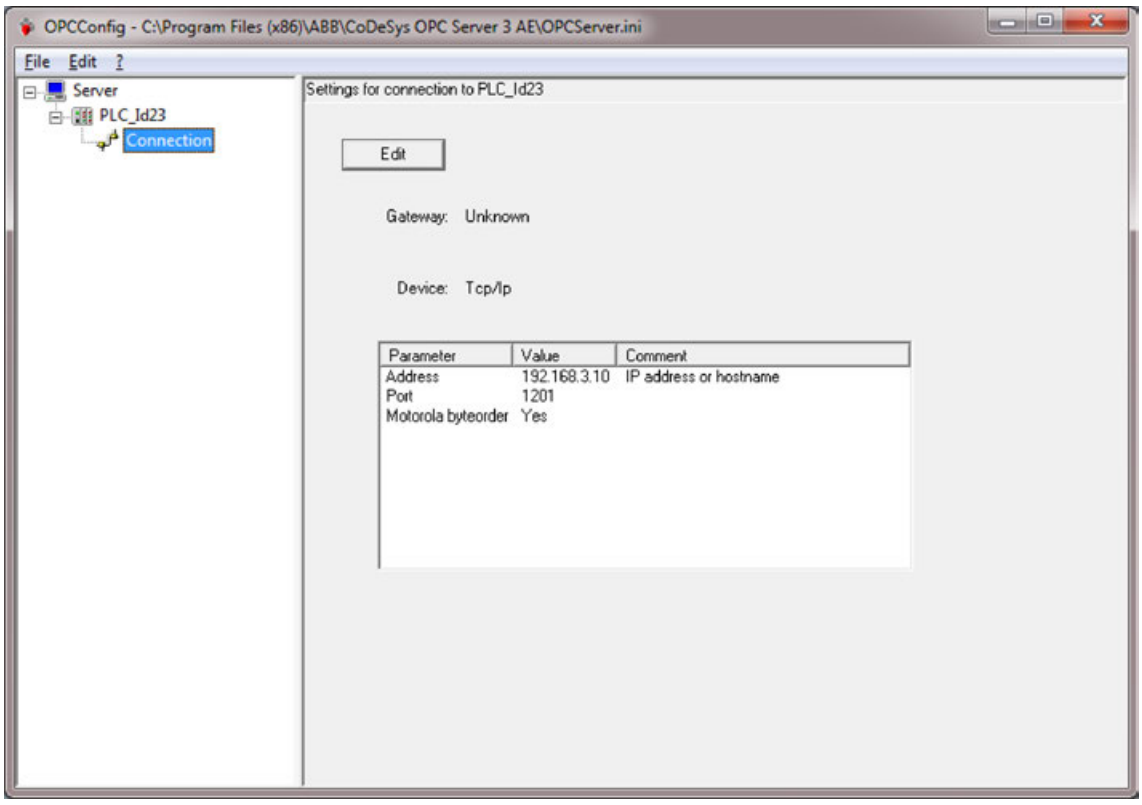


3. Open the OPCConfig and select *"PLC<...>"*. The *Project name* must be the same as the name of the symbol file.

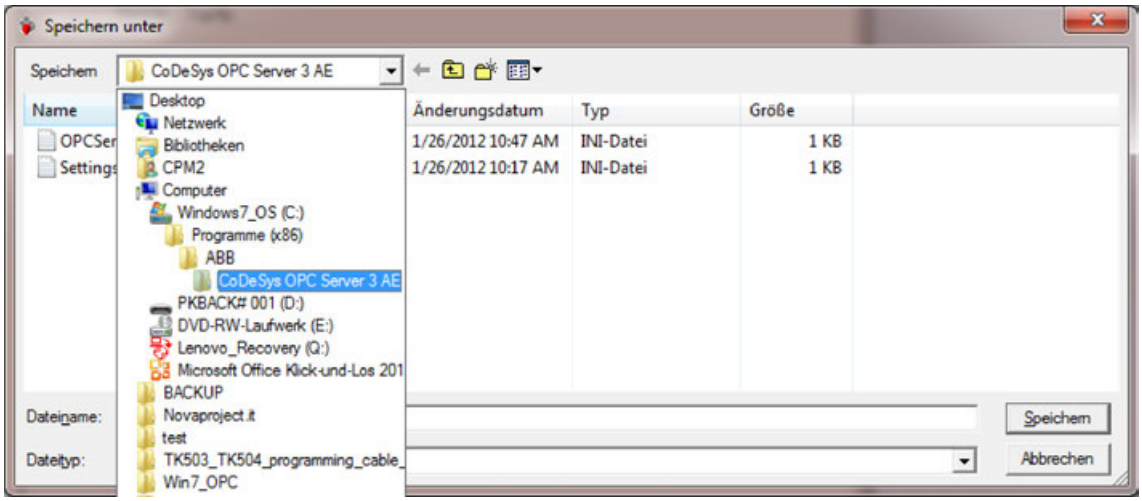
   Enable check boxes *Active*, *Motorola Byteorder*, *No Login-Service* and *Enable logging (Defaultevents)*.



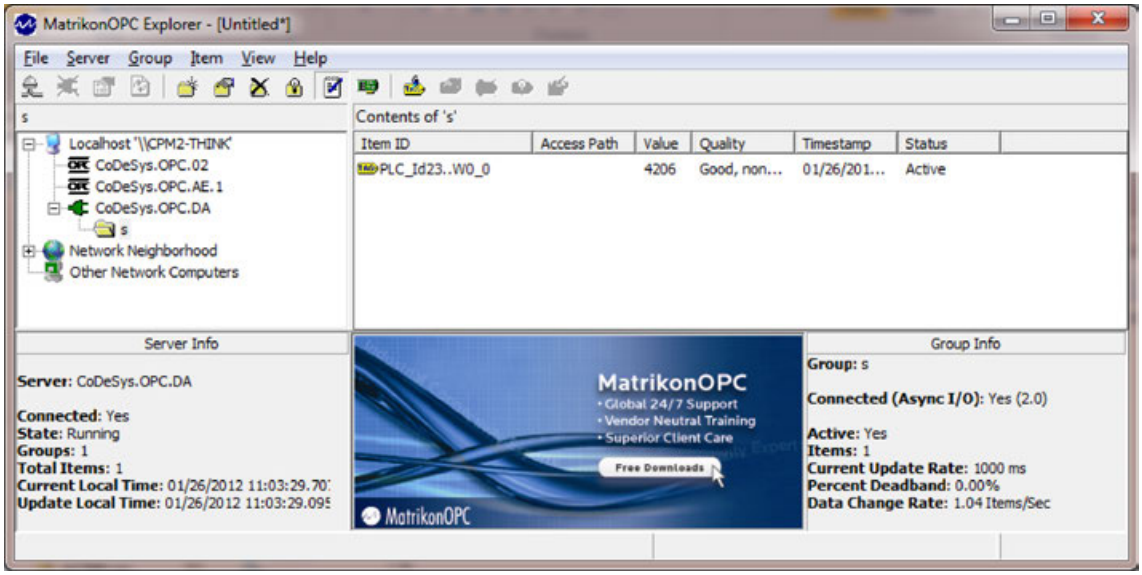
4. Select *"Connection"* to the PLC.



5. Save the current *OPCServer.ini* in the following folder *"C:\Programme (x86)/ABB/CoDeSys OPC Server 3 AE"*.



6. Check the OPC connection with a OPC client e.g. Matrikon.



Appendix

Transmission rate - Comparison with OPC server V2 to V3

Some figures about OPC Server transmission rates of a special test setup of HHZ:

- PC Lenovo T430, Windows 7, 64 bit
  - OPC client (OPC Systemtest Teststand, LabView 8.6 application)
  - OPC Server V2 und V3
- AC500 PM592 (task freewheeling and t=2 ms shown similar values)
- OPC client application: 100 cycles (write item, read item, compare value, increment value)

OPC Server	Item Byte			
	Connect [ms]	**Mean value [ms]**	**Max. value [ms]**	**Disconnect [ms]**
OPC Server V2: write cycle	2	2.374	4	0
OPC Server V2: read cycle	2	127.2	**133**	0
OPC Server V3: write cycle	2	1.838	4	1
OPC Server V3: read cycle	2	96.8889	**99**	1

OPC Server	Item Byte			
	Connect [ms]	**Mean value [ms]**	**Max. value [ms]**	**Disconnect [ms]**
OPC Server V2: write cycle	1	2.333	4	0
OPC Server V2: read cycle	1	127.152	**133**	0
OPC Server V3: write cycle	1	1.616	4	1
OPC Server V3: read cycle	1	97.1414	**99**	1

Performance - Comparison with OPC server V3 and TCPIP drivers

Measured on a Lenovo Thinkpad with Core-I5, Windows 7-64, 8GB RAM using a minimum OPC-Client (console application) written in C# with use of OpcNetApi-Library.

V2.3 project with 5 AC500 PLCs

TCPIP - Driver Name	**Buffer size setting in *opcserver.ini***	**Average CPU Load (PM591) [%]**	**Throughput Cyclic items per second at OPC-Client**
3S TCPIP	0	16	**8500**
ABB TCP/IP Level 2 AC	1000	19	**2886**
ABB TCP/IP Level 2 AC	5000	19	4770
ABB TCP/IP Level 2 AC	7000	19	5202

1.6.5.5.2 Web server

Use of the web server for the AC500

Integrated web server on AC500

As of firmware 2.0 an integrated web server on AC500/AC500-eCo PLCs is available. A web browser communicates with the web server on the PLC and displays the visualization by means of a Java applet using http 1.0.

Prerequisite: The processor module provides onboard Ethernet. No access via the interface of an Ethernet Communication Module.

Features:

- Web visualization (created in CODESYS)
- Download and storage of the webpages as part of the boot project on RAM disk of the CPU
- Access to the integrated AC500 web server with web browsers that contain a Java plug-in
- Access to the integrated AC500 web server with web panels e.g. CP6xx-WEB.
- File transfer from the PLC to the web browser on the PC.
- No special license required.

Usage of the web server occupies storage on the user RAM disk.

Table 758: User RAM disk (required for Java)

PM55x-xP-ETH PM56x-xP-ETH	PM556	PM573-ETH	PM583-ETH	PM590-ETH	PM591-ETH	PM592-ETH	PM595
512 kB (+ 400 kB)	1024 kB (+ 400 kB)	1024 kB (+ 400 kB)	4096 kB (+ 400 kB)	8 MB (+ 400 kB)	8 MB (+ 400 kB)	8 MB (+ 400 kB)	32 MB (+ 400 kB)

Configuration

Configuration in Automation Builder

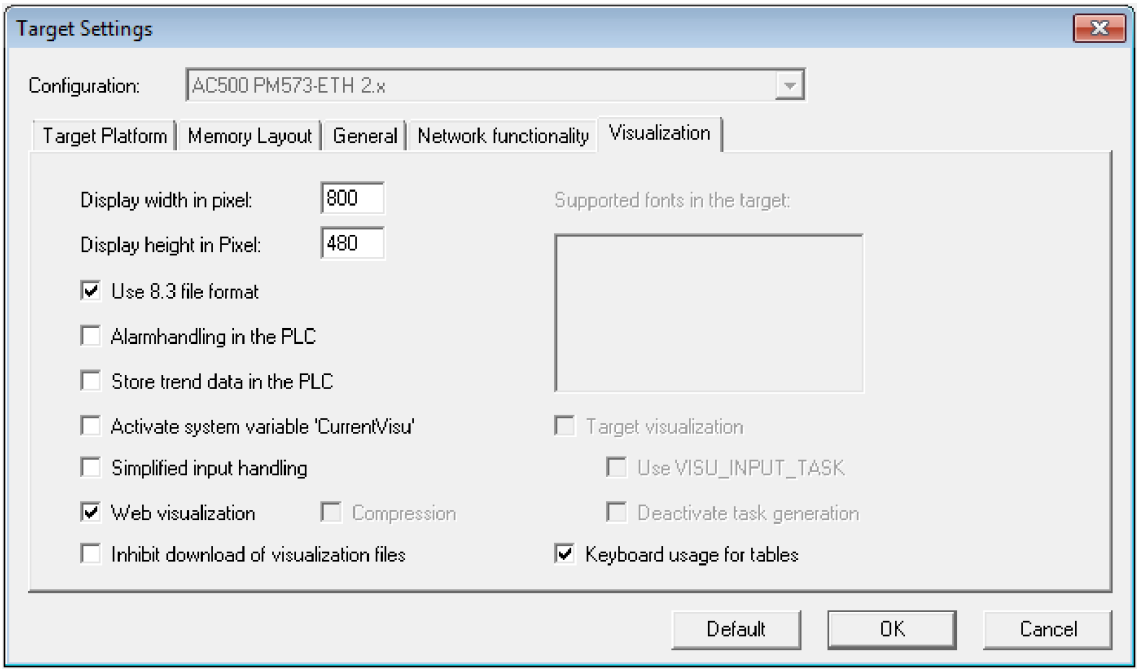
1. In your PLC project, click *“Interfaces ➔ Ethernet ➔ Protocols”* and add a new object *“Web Server”*.
2. Open the web server settings to change the default settings if required.

   Before changing the number of parallel connections for the web server, check the general information on *Chapter 1.6.4.1.6.1.1 “Ethernet protocols and ports for AC500 V2 products” on page 5442*.

Parameter	Default	Value	Description
Port	80	0 ... 65535	Listen port of the web server
Connections	2	1 ... 25 (depending on CPU type)	Number of parallel connections accepted by the web server. Depending on the CPU type and web-visu complexity it might be possible to have more connections to different clients. The connections will be opened and closed one after the other. This might lead to the impression of more parallel connections than configured.

Configuration in CODESYS

1. Open the CODESYS.
2. In the *“Resources”* tab click on *“Target Settings”* and select the *“Visualization”* tab.



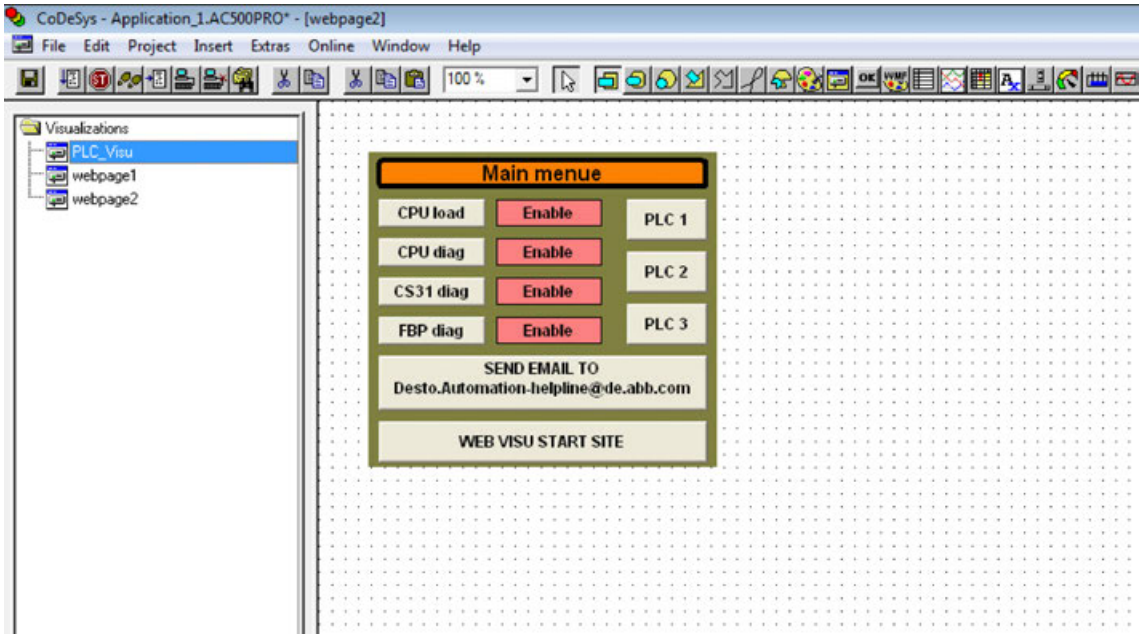
⇨ If required, change *“Display width”* and *“Display height”* for the web pages.

Enable *“Web visualization”* check box, otherwise the webpages are only visible in online mode.

Ensure *“Inhibit download of visualization files”* check box is disabled.

> *If you enable “Activate system variable 'CurrentVisu'” check box and define a global variable “CurrentVisu” [STRING(40)] in your project, then the name of the active visualization will be stored in this variable. You can switch between your visualization pages by changing this value.*

3. In CODESYS open the *"Visualizations"* tab to create the webpages. The first entry is the start page of your visualization project and must be named "PLC_Visu".

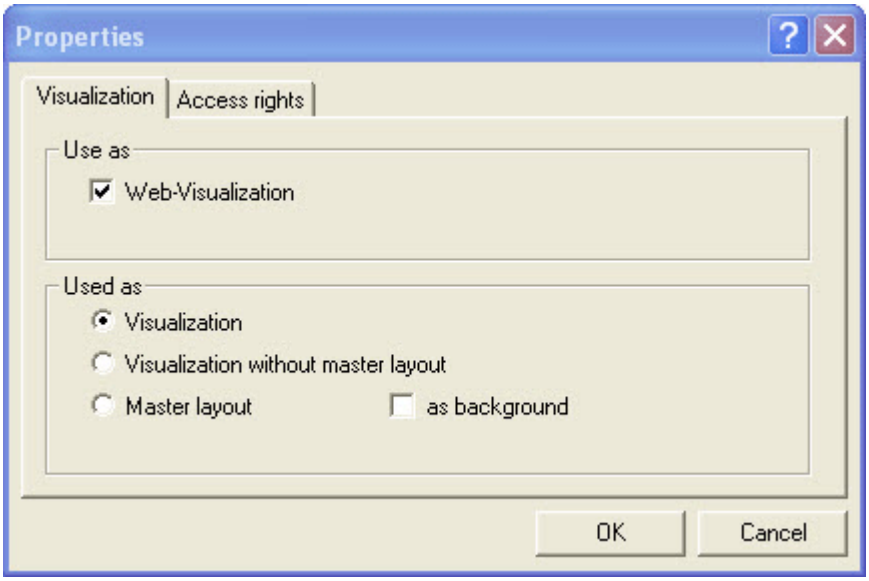


   ⇨ Further information on the creation and formatting of websites is provided in the CODESYS section ↳ *Chapter 1.4.5 "Web visualization" on page 721*. When building up your visualization project, consider the required disk space on the user RAM disk.

4. By default, all objects in the visualizations tab are marked for web visualization. When the project is built, all marked objects are saved as XML files (objectname.xml) into the project.

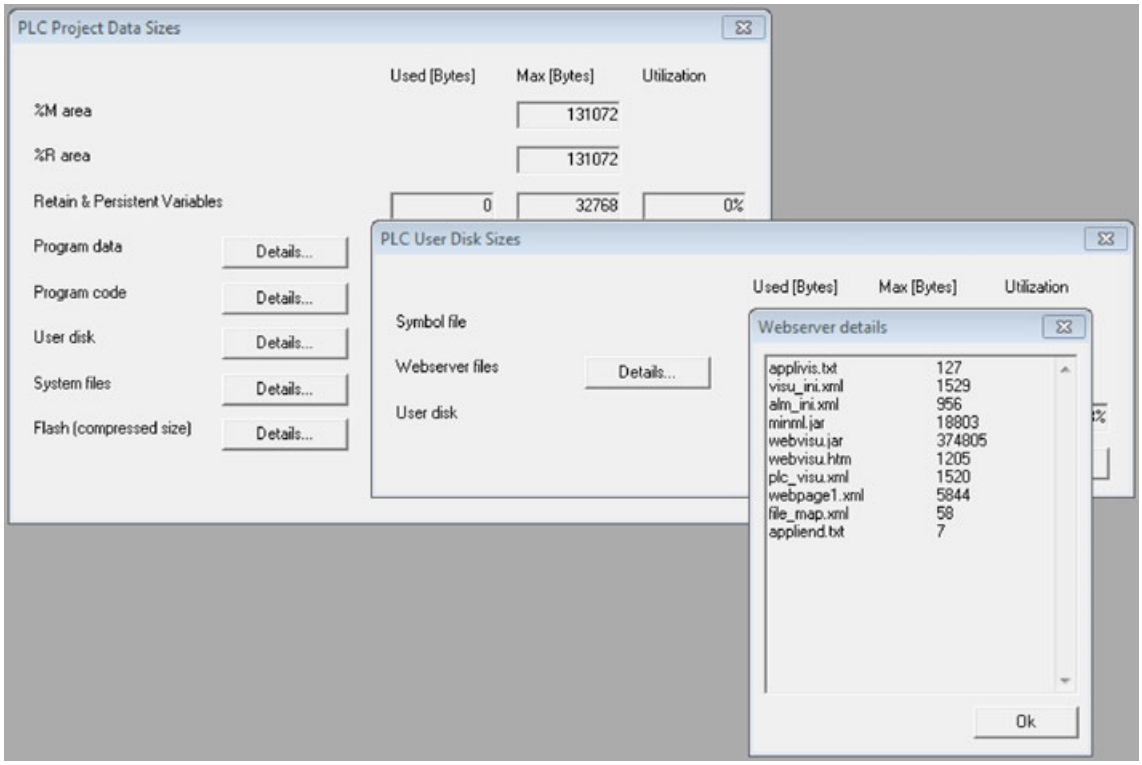
   To exclude a particular visualization object from the web visualization, click *"Project ➔ Object ➔ Properties"*.

   In the *"Visualization"* tab disable *Web-Visualization* check box.



5. Open a web browser and enter the IP address to the PLC (webserver project): http://<IP address/webvisu.htm>.

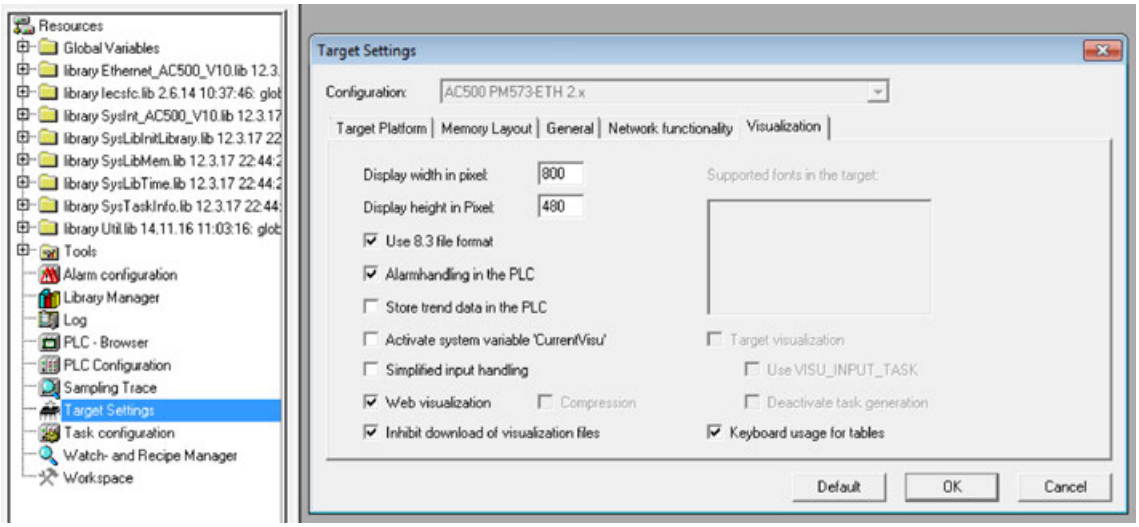
6. Click *"Online ➔ Show file information"* to check the *"PLC Project Data Sizes"*.



Alarm configuration

In case of a required alarm configuration via web server, alarms in the web server can be configured.

1. In your PLC project, open the CODESYS.



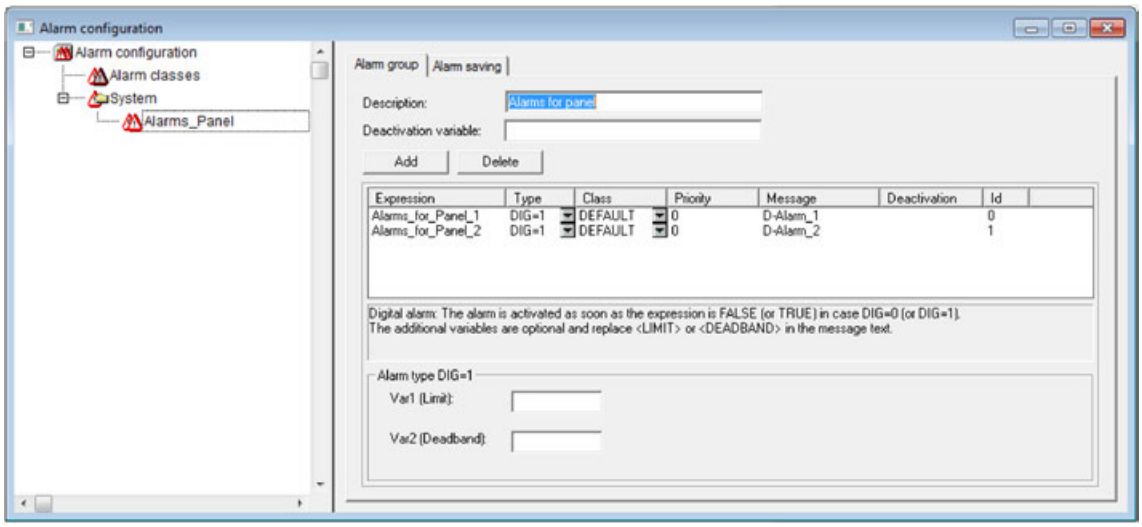
2. Enable *"Alarmhandling in the PLC"*.

   ⇨ In the *"Task configuration"* section a new task ALARM_TASK is created and the following CODESYS libraries are loaded:

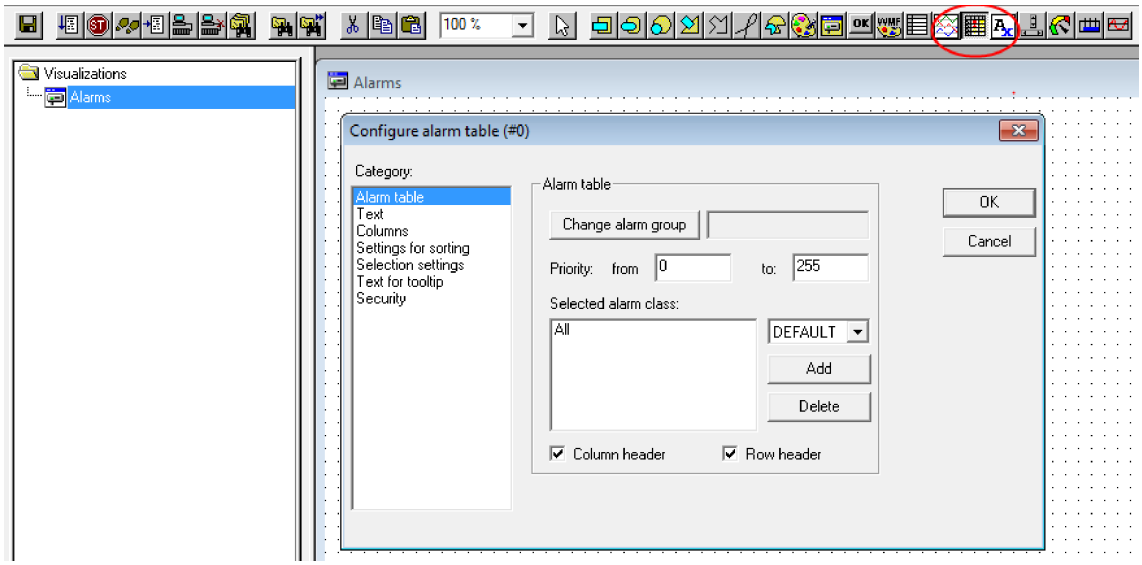
   - SysLibAlarmTrend
   - SysLibFile

3. In the *"Alarm configuration"* modify the default settings for the alarm classes if required.

   Under *"System"* append an alarm group.



4. Open the *"Visualizations"* view and add a new object.
5. If necessary, create a configuration table. However, do not change the alarm group to the option *"All alarm groups"*.



1.6.5.6 Converting an AC500 V2 project to an AC500 V3 project

A project that has been configured for an AC500 V2 PLC can be converted to a project for an AC500 V3 PLC.

Essentially, the conversion is done in Automation Builder, however, some additional actions have to be executed manually. The complete procedure is described in the application example

Instructions on how to convert a V2 project to a V3 project and differences between V2 and V3.

1.6.6 Storage devices for AC500 V2 products

1.6.6.1 Introduction of AC500 storage devices for AC500 Products

1.6.6.1.1 Overview

AC500 PLCs offer a variety of storage devices. The following table gives a short overview and a description on these storage devices:

IEC access means that the storage device can be accessed by function blocks of an IEC program.

FTP access means that the device can be accessed via FTP server on the PLC (if available).

Component	Description	IEC access	FTP access	CPUs AC500 V2
RAM disk	System RAM disk Volatile storage device placed in the SDRAM Used by the firmware during its operation For internal firmware use only!	No	Yes FTP user = "ramdisk"	All
User disk	User RAM disk Volatile storage device placed in the SDRAM Can be used for any application purpose	Yes	Yes FTP user = "userdisk"	All
SRAM disk	Battery-buffered, non-volatile RAM disk placed in the SRAM Can be used for any application purpose	Yes	Yes FTP user = "sramdisk"	All processor modules for AC500 (Standard) V2 *Chapter 1.6.1.7.2 “Device list: Processor modules (CPUs) ” on page 3713*
Internal flash	PLCs persistent memory Used for storing firmware, user application and user data	Yes	No	All

Component	Description	IEC access	FTP access	CPUs AC500 V2
Flash disk	Internal persistent mass storage device Can be used for any application purpose	Yes	Yes FTP user = "flashdisk"	PM592-ETH PM595-4ETH
memory card (removable)	memory card (removable) Removable persistent mass storage device Can be used for any application purpose	Yes	Yes FTP user = "sdcard"	All processor modules with memory card slot *Chapter 1.6.1.7.2 “Device list: Processor modules (CPUs) ” on page 3713*

1.6.6.1.2 Functionalities

The maximum number of files opened at the same time is limited to 1007.

The max. length of the user string (path and filename) is 241 characters.

Component	As of CPU firmware	Functionality
RAM disk	V2.0.2	Load / save boot project Firmware update Internal system files
	V2.1.3	Files via FTP server (e. g. firmware update)
User disk	V2.0.2	WebVisu files and Java applet for web server Symbol file for OPC server and CP600 panels
	V2.1.3	Java script files for web server User data via CAA_File_xxx.lib *Chapter 1.5.4.4 “CAA_File library” on page 789* Files via FTP server
SRAM disk	V2.1.3	User data via CAA_File_xxx.lib *Chapter 1.5.4.4 “CAA_File library” on page 789* Files via FTP server
Internal flash	V2.0.2	CPU firmware Traceability and configuration data User application: boot project, symbol file and WebVisu files (BOOT.ZIP + WEBVISU.ZIP) User data via *Chapter 1.5.4.19 “Internal system library” on page 1500* Internal System Library -> FLASH_<...>

Component	As of CPU firmware	Functionality
Flash disk	V2.1.3	User data via CAA_File_xxx.lib *Chapter 1.5.4.4 “CAA_File library” on page 789* Files via FTP server
Memory card (removable)	V2.0.2	Firmware update Load / save boot project Source code of user project load / save User data via POUs *Chapter 1.5.4.19.2.23 “SD_WRITE” on page 1563* and *Chapter 1.5.4.19.2.22 “SD_READ” on page 1558* Retain data Persistent data
	V2.1.3	User data via CAA_File_xxx.lib *Chapter 1.5.4.4 “CAA_File library” on page 789* Files via FTP server

Unlike the PLC's memory areas like %M or Retain, where 1 byte actually consumes 1 byte, all storage device utilize a file system.

That means there is a difference between a files size and its size on the disk.

On disks the files are stored in so-called clusters which are a group of disk sectors. "Size on disk" refers to the amount of cluster(s) a file is taking up, while "file size" is an actual byte count of the file data. So you will usually find that the size on disk is larger than the file size. This is not an error, but a result of the disk organization via a file system. Since sector and cluster sizes vary depending on a disk's size and the used file system, the ratios between the size on disk and the file size also vary between the various storage devices.

1.6.6.1.3 Memory sizes

PLC type	RAM disk	User RAM disk	SRAM disk	Flash memory		Flash disk	memory card *)
				User Data	User Application		
PM554	2048 kB	256 kB	None	128 kB	704 kB	None	up to 2 GB
PM564	2048 kB	256 kB	None	128 kB	704 kB	None	up to 2 GB
PM554-ETH	4096 kB	512 kB	None	128 kB	1024 kB	None	up to 2 GB
PM556-ETH	4096 kB	512 kB	None	-	-	None	up to 2 GB
PM564-ETH	4096 kB	512 kB	None	128 kB	1024 kB	None	up to 2 GB
PM572	4096 kB	512 kB	32 kB	128 kB	1024 kB	None	up to 2 GB
PM573-ETH	8192 kB	1024 kB	32 kB	128 kB	3072 kB	None	up to 2 GB

PLC type	RAM disk	User RAM disk	SRAM disk	Flash memory		Flash disk	memory card *)
				User Data	User Application		
PM582	8192 kB	512 kB	64 kB	128 kB	2048 kB	None	up to 2 GB
PM583-ETH	8192 kB	4096 kB	64 kB	128 kB	4096 kB	None	up to 2 GB
PM585-ETH	8192 kB	4096 kB	64 kB	-	-	None	up to 2 GB
PM590-ETH	16384 kB	8192 kB	256 kB	128 kB	12288 kB	None	up to 2 GB
PM590-ARC	16384 kB	8192 kB	256 kB	-	-	None	up to 2 GB
PM591-ETH	16384 kB	8192 kB	256 kB	128 kB	16384 kB	None	up to 2 GB
PM591-2 ETH	16384 kB	8192 kB	256 kB	-	-	None	up to 2 GB
PM592-ETH	16384 kB	8192 kB	256 kB	128 kB	16384 kB	4 GB	up to 2 GB
PM595-4 ETH-M	65535 kB	65535 kB	960 kB	-	-	4 GB	up to 2 GB
PM595-4 ETH-F	65535 kB	65535 kB	960 kB	-	-	4 GB	up to 2 GB

*) Size of the MC502 memory card.

It is not possible to use 100 % of a device's memory space. About 10 % of the total available space must remain unused at any time to maintain normal device operation.

1.6.6.1.4 Storage device details

This section contains some details on each storage device. For further details on specific topics please also refer to the following chapters:

Storage device sizes	see *Chapter 1.6.4.1.1 “Inputs, outputs and flags for AC500 V2 products” on page 5395*
IEC access	*Chapter 1.5.4.4 “CAA_File library” on page 789* *Chapter 1.5.4.19 “Internal system library” on page 1500*
FTP access	*Chapter 1.6.5.3.8 “FTP server” on page 6188*
PLC browser commands	*Chapter 1.7.2.5.5 “AC500-specific PLC browser commands” on page 6382*

RAM disk

The RAM disk is a volatile storage device placed in the SDRAM. It is used by the firmware during its operation and download. Via the *Chapter 1.6.5.3.8 “FTP server” on page 6188* it is possible to load new firmware to the PLC. The system files must not be deleted during run time.

Further, the RAM disk is used to temporarily save the PLC's web visualization before it is moved to the User disk.

Size	Product specific, see table Memory Sizes *Chapter 1.6.6.1.3 “Memory sizes” on page 6333*
Diagnosis device number	17

User disk

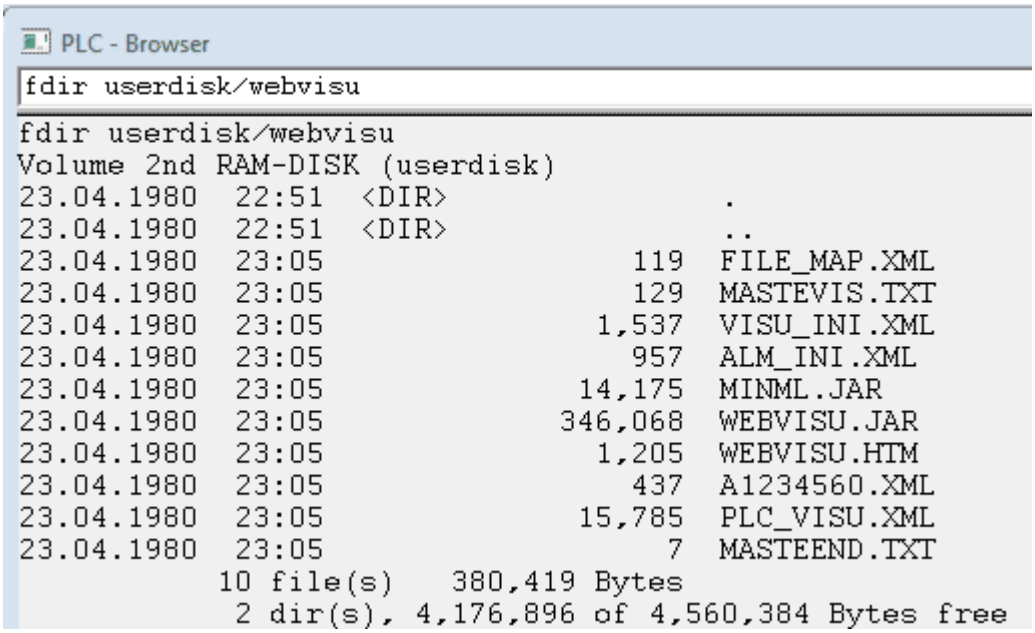
The User disk is a volatile storage device located in the SDRAM. It can be used for any application purpose. If activated, the PLC's web visualization is saved in the User Disk's subdirectory WEBVISU. If configured, the symbol file is also stored in the User disk.

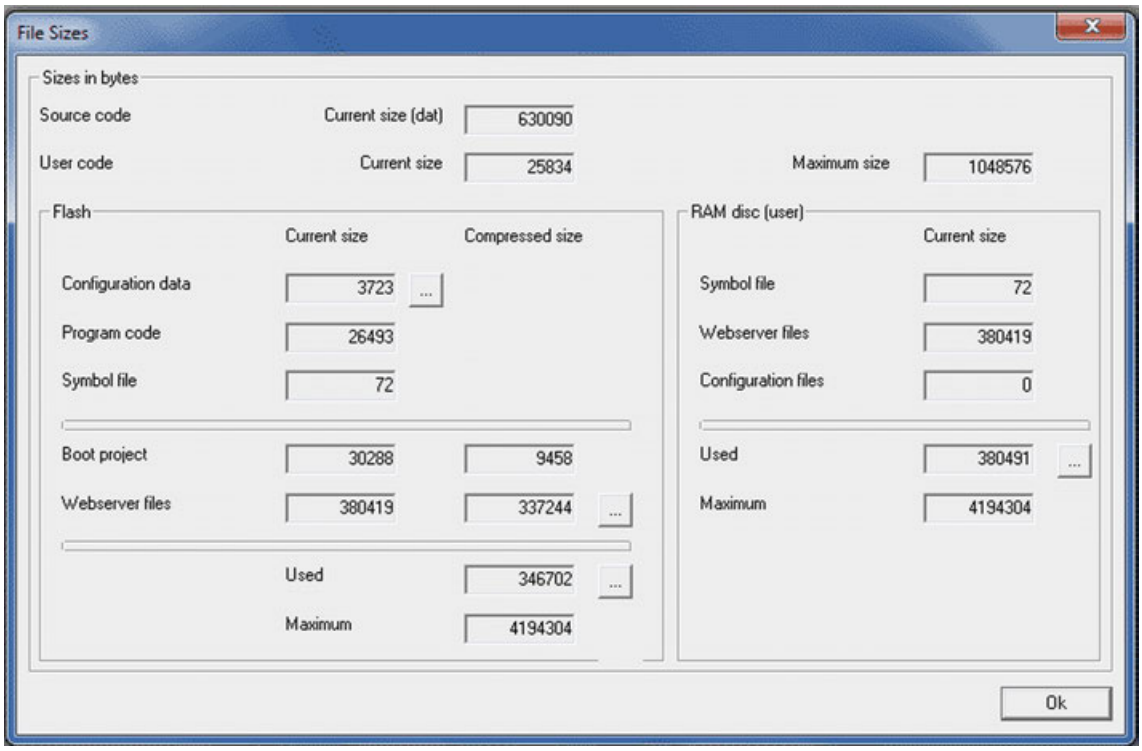
Size	Product specific, see table Memory Sizes *Chapter 1.6.6.1.3 “Memory sizes” on page 6333*
IEC access	*Chapter 1.5.4.4 “CAA_File library” on page 789*
Diagnosis device number	17

Additional hint for the size of the User disk:

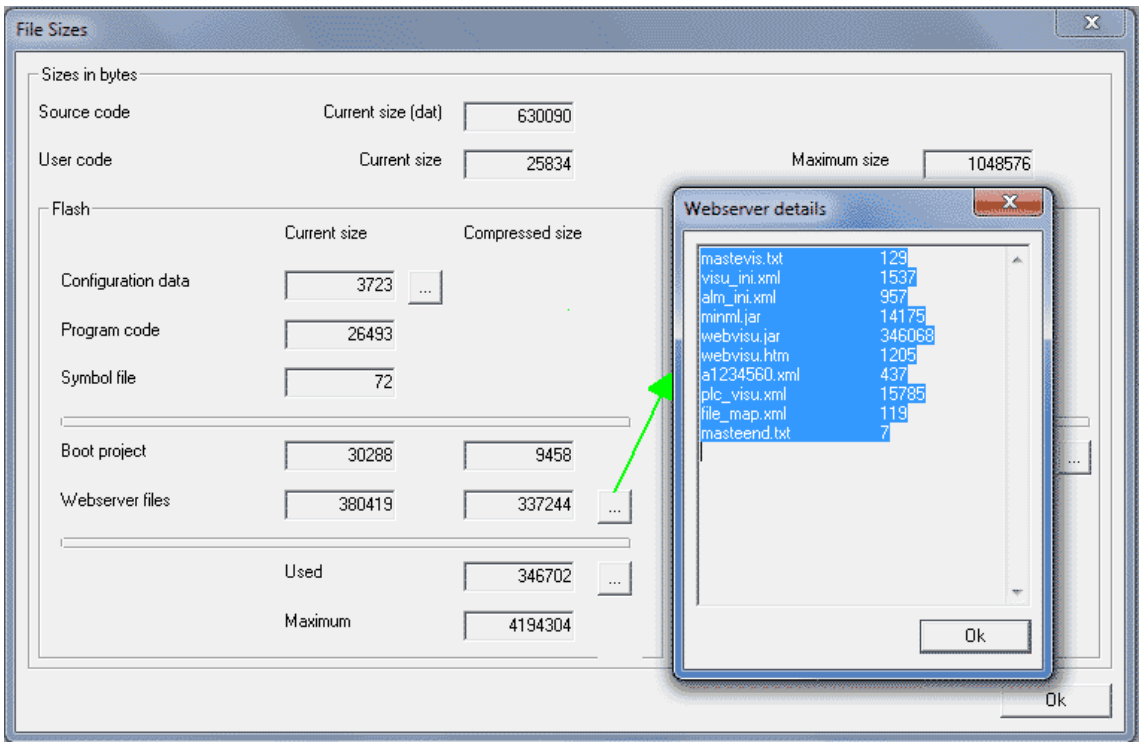
The web visualization's size is the sum of the user's visualization data plus the web visualization overhead (e.g. some Java applets (.jar files)). This means, the actual size of the User disk may differ from the offical documented size.

This difference can be recognized by comparing the sizes of the PLC browser command fdir userdisk and the information in the dialog *“File Sizes”*. This additional space is reserved for the web visualization's overhead.

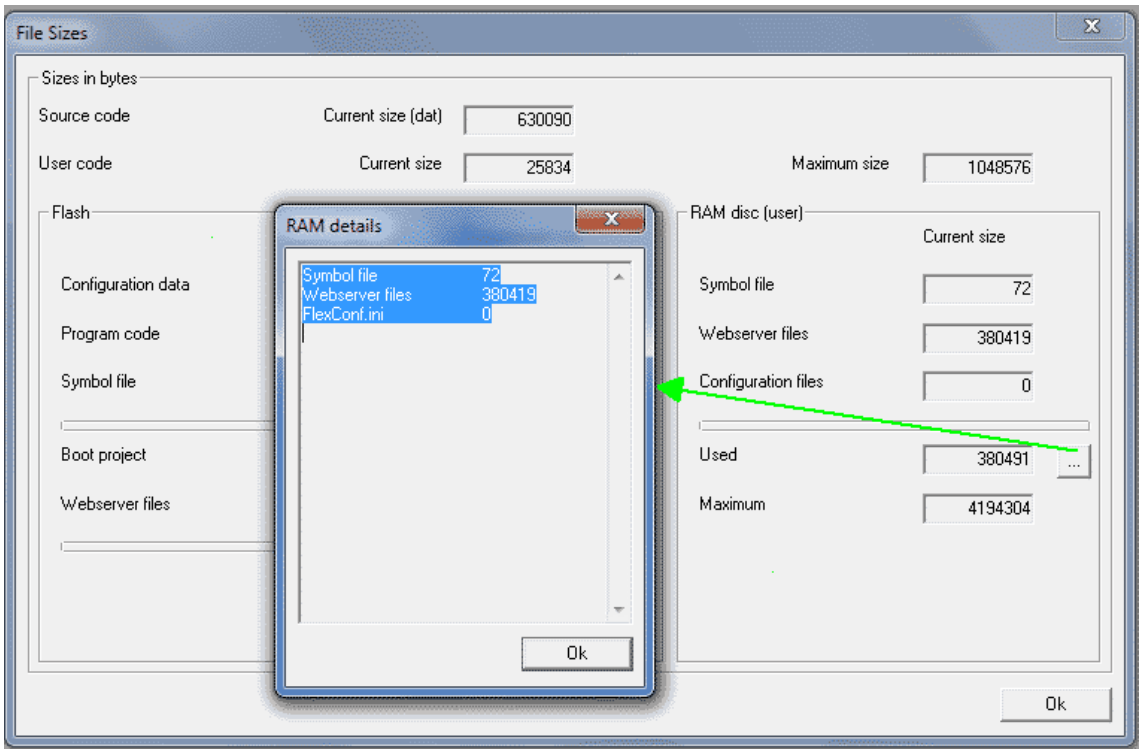




To get detailed information on your web visualization files, click *"..."* in the *"File Sizes"* dialog.



To get a summary of the various data which will be downloaded into the User disk, click *"..."* in line *"Used"* of the *"File Sizes"* dialog.



Thus, the number in line *"Used"* of the *"File Sizes"* dialog, means "total sum of the web visualization in Bytes" and the number in line *"Maximum"* means "Maximum allowed user data for web visualization". So, it is possible that *"Used"* becomes bigger than *"Maximum"*.

SRAM disk

The SRAM disk is a battery-buffered, nonvolatile RAM disk located in the SRAM and can be used for any application purpose. If a battery is inserted into the processor module, the data stored in the SRAM disk will not get lost during a power-down cycle. During PLC startup, the SRAM disk will be formatted automatically if no or an empty battery is inserted into the processor module. In this case, the PLC issues an E4 error (#0153286827 = [4-9-34-31-2-43], i.e. SRAM disk has been formatted). To enable the visibility of errors, set the CPU parameter *"Check for battery"* to TRUE. With this, the system will signal errors during the formatting of the SRAM disk.

Size	Product specific, see table Memory Sizes ↬ *Chapter 1.6.6.1.3 "Memory sizes" on page 6333*
IEC access	↬ *Chapter 1.5.4.4 "CAA_File library" on page 789*
Diagnosis device number	34

Memory card

The memory card is a removable persistent mass storage device and can be used for any application purpose. Both firmware updates and boot project updates can be run from the memory card ↬ *Chapter 1.6.6.2 "Memory card in AC500 V2" on page 6339*.

Size	Product specific, see table Memory Sizes ↳ *Chapter 1.6.6.1.3 “Memory sizes” on page 6333*
IEC access	↳ *Chapter 1.5.4.4 “CAA_File library” on page 789* ↳ *Chapter 1.5.4.19 “Internal system library” on page 1500*
Diagnosis device number	20

Flash

The internal flash is the PLCs persistent memory.

It is used for storing firmware.

It is used for storing firmware, user application and user data (see chapter 'Data Storage in Flash Memory' ↳ *Chapter 1.6.6.3 “Data storage in flash memory for AC500 V2 products” on page 6364*).

Size	Product specific, see table Memory Sizes ↳ *Chapter 1.6.6.1.3 “Memory sizes” on page 6333*
Diagnosis device number	18

Flash disk

The flash disk is an internal persistent mass storage device and can be used for any application purpose.

It has a memory capacity of 4 GB (preformatted).

The flash disk is capable of high data throughput, however, the actual values to be achieved depend on the use cases ↳ *Chapter 1.6.6.4 “Flash disk for AC500 V2 products” on page 6364*. If the performance seems to get insufficient, check the following:

- If the PLCs CPU load is high, reduce overall CPU load of the PLC to have more performance for file operations.
- If the device has low free space, cleanup the disk.
  Please consider the cluster size of 4 kB in your application design to achieve optimal usage of the flash disks space and access performance. For example, 10 files with 10 byte each require 10*4 kB disk space, while 1 file with 100 byte requires only 4 kB.
- If the device is fragmented, cleanup the disk.

Size	Product specific, see table Memory Sizes ↳ *Chapter 1.6.6.1.3 “Memory sizes” on page 6333*
IEC access	↳ *Chapter 1.5.4.4 “CAA_File library” on page 789*
Diagnosis device number	33

1.6.6.2 Memory card in AC500 V2

1.6.6.2.1 Memory card functions for AC500 V2 products

Summary

AC500 processor modules contain a user flash memory card of the type *memory card* as external storage device which is accessed by the CPU like a floppy disk drive. The memory card is used to transfer data between a commercially available PC with memory card interface and the AC500 PLC.

> **NOTICE!**
> **Removal of the (micro) memory card**
> Do not remove the (micro) memory card when it is working!
>
> Remove the memory card or micro memory card with micro memory card adapter only when the RUN LED is not blinking.
>
> Otherwise the (micro) memory card and/or files on it might get corrupted and/or normal PLC operation might be disturbed.

The memory card can be used to:

- update the AC500 CPU processor firmware
- update the CPU boot code
- update the display controller firmware (as of version V2.0)
- update of the RTC_Batt module (eCo)
- update of the onboard I/O firmware (eCo)
- update the Communication Module firmware
- load and save user programs (boot project)
- load and save the source code of the user program
- load and save retentive variables (RETAIN, %R area)
- load and save user data (with blocks)

The memory card can be operated by:

- writing/reading files using a standard PC card reader with memory card interface
- specific PLC-Browser commands
- reading and writing data from the user program using specific blocks
- using FTP client (firmware version > V2.1.x and a Processor Module with onboard Ethernet)

PLC browser commands

For accessing the memory card the following PLC browser commands can be used:

- persistent
- retain
- sdappl
- sdboot x
- sdcoupler x
- sddisplay x
- sdfirm x
- sdfunc
- sdonboardio x
- sdrtcbat x

For details on the PLC browser commands for AC500 V2 products see *Chapter 1.7.2.5.5 “AC500-specific PLC browser commands” on page 6382*

1.6.6.2.2 Firmware and/or application update with memory card

Memory card file structure for AC500 V2 products

File structure as of version V2.x

The following table provides information on the file structure of version V2.x. Depending on the version, the directory names of the actual firmware may differ.

Module Type	Directory (Root)	Data File (SDCARD.INI)	Supported as of Version
CM574-RS CM574-RCOM	FIRMWARE\CM574\2_1_3\	C574.gza C574B.gza RCOM.PRG RCOM.CHK	V1.3.x
PM554	FIRMWARE\PM554\	Pm55x.gza	V1.3.x
	FIRMWARE\PM554\2_4_2\	Pm55x.gza	
	FIRMWARE\PM554\1_3_0\	Pm55xB.gza	
	FIRMWARE\PM554\ONB_IO\1_1_2	5500.app	
	FIRMWARE\PM554\ONB_IO\	5500.app	
	FIRMWARE\PM554\RTC_BATT\1_6\	RtcBatt.app	
PM554-ETH	FIRMWARE\PM554ETH\	Pm55xE.gza	V2.0.x
	FIRMWARE\PM554ETH\2_4_2\	Pm55xE.gza	
	FIRMWARE\PM554ETH\2_0_2\	Pm55xEB.gza	
	FIRMWARE\PM554ETH\ONB_IO\1_1_2\	5505.app	
	FIRMWARE\PM554ETH\ONB_IO\	5505.app	
	FIRMWARE\PM554ETH\RTC_BATT\1_6\	RtcBatt.app	
PM564	FIRMWARE\PM564\	Pm55x.gza	V1.3.x
	FIRMWARE\PM564\2_4_2\	Pm55x.gza	
	FIRMWARE\PM564\1_3_0\	Pm55xB.gza	
	FIRMWARE\PM564\ONB_IO\1_1_2\	5501.app	
	FIRMWARE\PM564\ONB_IO\	5501.app	
	FIRMWARE\PM564\RTC_BATT\1_6\	RtcBatt.app	
PM564-ETH	FIRMWARE\PM564ETH\	Pm55xE.gza	V2.0.x
	FIRMWARE\PM564ETH\2_4_2\	Pm55xE.gza	
	FIRMWARE\PM564ETH\2_0_2\	Pm55xEB.gza	
	FIRMWARE\PM564ETH\ONB_IO\1_1_2\	5501.app	
	FIRMWARE\PM564ETH\ONB_IO\	5501.app	
	FIRMWARE\PM564ETH\RTC_BATT\1_6\	RtcBatt.app	

Module Type	Directory (Root)	Data File (SDCARD.INI)	Supported as of Version
PM572	FIRMWARE\PM572\	PM58xN.gza	V2.0.x
	FIRMWARE\PM572\2_4_2\	PM58xN.gza	
	FIRMWARE\PM572\2_3_0\	PM58xNB.gza	
	FIRMWARE\PM572\Display\2_8	Display.app	
PM573-ETH	FIRMWARE\PM573ETH\	PM58xN.gza	V2.0.x
	FIRMWARE\PM573ETH\2_4_2\	PM58xN.gza	
	FIRMWARE\PM573ETH\2_3_0\	PM58xNB.gza	
	FIRMWARE\PM573ETH\Display\2_8\	Display.app	
PM582	FIRMWARE\PM582\	PM58xN.gza	V2.0.x
	FIRMWARE\PM582\2_4_2\	PM58xN.gza	
	FIRMWARE\PM582\2_3_0\	PM58xNB.gza	
	FIRMWARE\PM582\Display\2_8\	Display.app	
PM583-ETH	FIRMWARE\PM583\	PM58xN.gza	V2.0.x
	FIRMWARE\PM583\2_4_2\	PM58xN.gza	
	FIRMWARE\PM583\2_3_0\	PM58xNB.gza	
	FIRMWARE\PM583\Display\2_8\	Display.app	
PM590-ARC	FIRMWARE\PM590ARC\	Pm59xRD.gza	V2.3.x
	FIRMWARE\PM590ARC\2_4_2\	Pm59xRD.gza	
	FIRMWARE\PM590ARC\2_3_1\	Pm59xRDB.gza	
	FIRMWARE\PM590ARC\Display\2_8\	Display.app	
PM590-ETH	FIRMWARE\PM590ETH\	PM59xRD.gza	V2.0.x
	FIRMWARE\PM590ETH\2_3_1\	PM59xRD.gza	
	FIRMWARE\ PM590ETH\2_4_2\	PM59xRDb.gza	
	FIRMWARE\ PM590ETH\Display\2_8\	Display.app	
PM591-ETH	FIRMWARE\PM591ETH\	PM59xRD.gza	V2.0.x
	FIRMWARE\PM591ETH\2_3_1\	PM59xRD.gza	
	FIRMWARE\PM591ETH\2_4_2\	PM59xRDb.gza	
	FIRMWARE\PM591ETH \Display\2_8\	Display.app	
PM591-2ETH	FIRMWARE\PM591_2.ETH\	Pm59xRD.gza	V2.4.x
	FIRMWARE\PM591_2.ETH\2_4_2\	Pm59xRD.gza	
	FIRMWARE\PM591_2.ETH\2_3_1\	Pm59xRDB.gza	
	FIRMWARE\PM591_2.ETH\Display\2_8\	Display.app	
PM592-ETH	FIRMWARE\PM592ETH\	PM59xRD.gza	V.2.1.x
	FIRMWARE\PM592ETH\2_4_2\	PM59xRD.gza	
	FIRMWARE\PM592ETH\2_3_1\	PM59xRDb.gza	

Module Type	Directory (Root)	Data File (SDCARD.INI)	Supported as of Version
	FIRMWARE\PM591ETH\Display\2_8\	Display.app	
PM595-4ETH	FIRMWARE\PM595_4.ETH\	PM595.gza	V2.4.x
	FIRMWARE\PM595_4.ETH\2_4_1\	PM595B.gza	
	FIRMWARE\PM595_4.ETH\2_4_2\	PM595.gza	
	FIRMWARE\CMETH\2_7_22\	ETHCFG.nxf	
	FIRMWARE\PM595_4.ETH\LEDBOARD\	LEDBoard.app	
	FIRMWARE\PM595_4.ETH\LEDBOARD\1_5\	LEDBoard.app	

Command file *SDCARD.INI* for AC500 V2 Products

FW version V2.0 to V2.4

[Status] FunctionOfCard=2	0 = Perform no function when inserting the memory card or voltage ON 1 = Load user program according to entry in group [UserProg] 2 = Start firmware update according to entry in group [FirmwareUpdate] 3 = Update firmware according to entry in group [FirmwareUpdate] and load user program according to entry in [UserProg]
[FirmwareUpdate] CPUPM5x1=2	0 = No update 1 = Update firmware always from base directory on the CPU (supported for CPU firmware only, but not for other firmware components, e. g. display firmware, firmware of onboard I/Os etc.). 2 = Update with specific version, the update is only performed if the key *version* in a product section (for example [PM583]) and the*) according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed if the key *version* in a product section (for example [PM583]) and the*) according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.
Display=2	0 = No update 2 = Update with specific version, the update is only performed if the key *version* in a product section (for example [PM583]) and the according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed, if the key *version* in a product section (for example [PM583]) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.

OnboardIO=2	0 = No update 2 = Update with specific version, the update is only performed if the key *OnboardIO* in a product section (for example [PM554]) and the according file returns a different result than the version on the CPU. If the key *OnboardIO* is missing, no update is performed. 3 = Update with newer version, the update is only performed if the key *OnboardIO* in a product section (for example [PM554]) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.
RtcBatt=2	0 = No update 2 = Update with specific version, the update is only performed, if the key *RtcBatt* in a product section (for example [PM554]) and the according file returns a different result than the version on the CPU. If the key *OnboardIO* is missing, no update is performed. 3 = Update with newer version, the update is only performed if the key *RtcBatt* in a product section (for example [PM554]) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.
Coupler0=0	Update communication module slot 0 0 = No update 2 = Update with specific version, the update is only performed if the key Version in a product section (for example [PM5x1]) and the according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed if the key *version* in a product section (for example [PM5x1]) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.
Coupler1=0 Coupler2=0 Coupler3=0 Coupler4=0 Coupler5=0 Coupler6=0	Update module slot 1; available modes: 0/2/3; see description Coupler0) Update module slot 2; available modes: 0/2/3; see description Coupler0) Update module slot 3; available modes: 0/2/3; see description Coupler0) Update module slot 4; available modes: 0/2/3; see description Coupler0) Update module slot 5; available modes: 0/2/3; see description Coupler0) Update module slot 6; available modes: 0/2/3; see description Coupler0)
LedBoard=0	Update LED Board of PM595-4ETH CPU 0 = No update 2 = Update with specific version, the update is only performed, if the key *version* in the product section (for PM595-4ETH]) and the according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed if the key *version* in the product section (for PM595-4ETH) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed.
[UserProg]	

UserProgram=0	Update user program: 0 = no update / 1 = update)
RetainData=0	Update retain variables: 0 = no update / 1 = update)
PersistentData=0	Update persistant variables: 0 = no update / 1 = update)
ConfData=0	Update configuration data of the onboard Ethernet CPU´s and the communication module´s: 0 = no update / 1 = update*)
CouplerConfig0=0	Update coupler configuration slot 0 : 0 = no update / 1 = update*)
CouplerConfig1=0	Update coupler configuration slot 1 : 0 = no update / 1 = update*)
CouplerConfig2=0	Update coupler configuration slot 2 : 0 = no update / 1 = update*)
CouplerConfig3=0	Update coupler configuration slot 3 : 0 = no update / 1 = update*)
CouplerConfig4=0	Update coupler configuration slot 4 : 0 = no update / 1 = update*)
[PM554]	CPU type
VERSION=2_4_2	CPU firmware version
PLCBOOT=1_3_0	CPU bootcode version
ONB_IO=1_1_2	Onboard IO version
RTC_BATT=1_6	RTC_Batt version
[PM554ETH]	
VERSION=2_4_2	
PLCBOOT=2_0_2	
ONB_IO=1_1_2	
RTC_BATT=1_6	
[PM564]	
VERSION=2_4_2	
PLCBOOT=1_3_0	
ONB_IO=1_1_2	
RTC_BATT=1_6	
[PM564ETH]	
VERSION=2_4_2	
PLCBOOT=2_0_2	
ONB_IO=1_1_2	
RTC_BATT=1_6	
[PM572]	CPU type
VERSION=2_4_2	CPU firmware version
PLCBOOT=2_3_0	CPU bootcode version
DISPLAY=2_8	CPU display version
[PM573ETH]	
VERSION=2_4_2	
PLCBOOT=2_3_0	
DISPLAY=2_8	
[PM582]	
VERSION=2_4_2	
PLCBOOT=2_3_0	
DISPLAY=2_8	

[PM583] VERSION=2_4_2 PLCBOOT=2_3_0 DISPLAY=2_8	
[PM590ARC] VERSION=2_4_2 PLCBOOT=2_3_1 DISPLAY=2_8	
[PM590ETH] VERSION=2_4_2 PLCBOOT=2_3_1 DISPLAY=2_8	
[PM591ETH] VERSION=2_4_2 PLCBOOT=2_3_1 DISPLAY=2_8	
[PM591-2ETH] VERSION=2_4_2 PLCBOOT=2_3_1 DISPLAY=2_8	
[PM592ETH] VERSION=2_4_2 PLCBOOT=2_3_1 DISPLAY=2_8	
[PM595-4ETH] VERSION=2_4_2 PLCBOOT=2_4_1 LEDBOARD=1_5	
[CM574] VERSION=2_1_3	
[CM578] VERSION=1_109	
[CMETH] VERSION=2_7_22	

FW version V2.5 As of firmware version 2.5 all firmware updates are triggered by the command file *SDCARD.INI*. This is independent from the way of the firmware update (memory card, FTP, *write file to plc*, …). In addition a result file of the firmware update is generated (*SDCARD.RDY*, identical path as *SDCARD.INI*, see *Initializing an memory card* ↬ *Chapter 1.6.6.2.2.4.1 “Using the AC500 PLC for AC500 V2 Products” on page 6349*). For the group [FirmwareUpdate] the new parameters 11, 12 and 13 are defined.

SDCARD.INI for memory card PM595 in part:

```
[FirmwareUpdate]
CPUPM5x1=11
Coupler1=11
Coupler2=11
Coupler5=11
Coupler6=11
[CPU]
Firmware=FIRMWARE\PM595ETH\2_5_0\PM595.gza
[Coupler1]
Firmware=FIRMWARE\CM597\V1_0_1\AC500eth.nxf
[Coupler2]
Firmware=FIRMWARE\SM560\1_0_0\S560_1.gza
Firmware2=FIRMWARE\SM560\1_0_0\S560_2.gza
[Coupler5]
Firmware=FIRMWARE\CM597\2_7_5\AC500pnm.nxf
[Coupler6]
Firmware=FIRMWARE\CM597\4_1_3_7\NX_ECM.nxf
```

SDCARD.INI for update via FTP in part:

In this case the files *PM595.gza*, *AC500eth.nxf*, *S560_1.gza*, *S560_2.gza*, *AC500pnm.nxf* and *NX_ECM.nxf* are transferred via FTP to the RAM disk of the PLC. The last step is the transfer of the file *SDCARD.INI* to the RAM disk of the PLC. At the end of the update the result file *SDCARD.RDY* is generated. The evaluation of this file shows the results of the updates.

```
[FirmwareUpdate]
CPUPM5x1=11
Coupler1=11
Coupler2=11
Coupler5=11
Coupler6=11
[CPU]
Firmware=PM595.gza
[Coupler1]
Firmware=AC500eth.nxf
[Coupler2]
Firmware=S560_1.gza
Firmware2=S560_2.gza
[Coupler5]
Firmware=AC500pnm.nxf
[Coupler6]
Firmware=NX_ECM.nxf
```

[FirmwareUpdate] CPUPM5x1=2	In addition to the unchanged parameters 0, 1, 2 and 3 the parameters 11, 12 and 13 are defined. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *Boot* or/and *Firmware* (for SM560 the component's path keys *Boot2* and *Firmware2* are defined additionally). 12 = Update with specific version, the update is only performed if the version of the file specified by the component path key *Boot* and/or *Firmware* in module's section [CPU] differs from the current version of the CPU. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *Boot* and/or *Firmware* in module's section [CPU] is newer than the current version of the CPU.
Display=2	In addition to the unchanged parameters 0, 2 and 3 the parameters 11, 12 and 13 are defined. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *Display*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *Display* in module's section [CPU] differs from the current version of the Display. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *Display* in module's section [CPU] is newer than the current version of the Display.
OnboardIO=2	In addition to the unchanged parameters 0, 2 and 3 the parameters 11, 12 and 13 are defined. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *OnboardIO*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *OnboardIO* in module's section [CPU] differs from the current version of the OnboardIO. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *OnboardIO* in module's section [CPU] is newer than the current version of the OnboardIO.
OnboardCan=2	Update OnboardCan of PM595-4ETH CPU 0 = No update 2 = Update with specific version, the update is only performed, if the key Version in the product section (for PM595-4ETH]) and the according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed, if the key *version* in the product section (for PM595-4ETH) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *OnboardCan*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *OnboardCan* in module's section [CPU] differs from the current version of the OnboardCan. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *OnboardCan* in module's section [CPU] is newer than the current version of the OnboardCan.

RtcBatt=2	In addition to the unchanged parameters 0, 2 and 3 the parameters 11, 12 and 13 are defined. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *RtcBatt*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *RtcBatt* in module's section [CPU] differs from the current version of the RtcBatt. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *RtcBatt* in module's section [CPU] is newer than the current version of the RtcBatt.
LedBoard=0	Update LED Board of PM595-4ETH CPU 0 = No update 2 = Update with specific version, the update is only performed, if the key *version* in the product section (for PM595-4ETH]) and the according file returns a different result than the version on the CPU. If the key *version* is missing, no update is performed. 3 = Update with newer version, the update is only performed, if the key *version* in the product section (for PM595-4ETH) and the according file returns a newer version than the version on the CPU. If the key *version* is missing, no update is performed. 11 = Update firmware always with the file specified in module's section [CPU] and component's path key *LedBoard*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *LedBoard* in module's section [CPU] differs from the current version of the Led-Board. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *LedBoard* in module's section [CPU] is newer than the current version of the LedBoard.
Coupler0=0	In addition to the unchanged parameters 0, 2 and 3 the parameters 11, 12 and 13 are defined. 11 = Update firmware always with the file specified in module's section [Coupler0] and component's path key *Firmware*. 12 = Update with specific version, the update is only performed, if the version of the file specified by the component path key *Firmware* in module's section [Coupler0] differs from the current version of the Coupler. 13 = Update with newer version, the update is only performed, if the version of the file specified by the component key *Firmware* in module's section [Coupler0] is newer than the current version of the Coupler.
Coupler1=0	Update module slot 1; see description Coupler0)
Coupler2=0	Update module slot 2; see description Coupler0)
Coupler3=0	Update module slot 3; see description Coupler0)
Coupler4=0	Update module slot 4; see description Coupler0)
Coupler5=0	Update module slot 5; see description Coupler0)
Coupler6=0	Update module slot 6; see description Coupler0)

Description of LEDs

The LEDs below the display indicate the status of the processor module:

LED	State	Color	LED = ON	LED = OFF	LED flashes
Power LED (PWR)	Denotes the power supply state of the processor module	Green	Voltage is present (24 V DC)	Voltage is missing	-
Run LED (RUN)	Denotes the activity state of the processor module	Green	Processor module is in RUN mode	Processor module is in STOP mode	If the Run LED (RUN) flashes fast (4 Hz), the processor module is reading/writing the memory card. Together with a flashing Error LED (ERR), the processor module is writing the internal Flash. If the Run LED flashes slowly (1 Hz), a firmware update from the memory card is finished without errors.
Error LED (ERR)	Denotes an error	Red	An error has occurred. After pressing the DIAG function key, the error type and the error code will be displayed. The error codes can be shown by means of the DIAG and OK function keys.	No errors or only warnings have occurred.	If the Error LED flashes fast (4 Hz) together with a flashing Run LED, the firmware is updated and a Flash is written. If the Error LED flashes slowly (1 Hz) a firmware update from the memory card is finished with errors.

A running processor module is indicated with the state RUN on the display, a deactivated processor module is indicated with the state STOP. In both cases the display's backlight is off.

Initializing an memory card

Using the AC500 PLC for AC500 V2 Products

The file structure is created on the memory card when a formatted memory card is inserted into the AC500 PLC. The file SDCARD.INI contains the following entries:

```
[Status]FunctionOfCard=0
[FirmwareUpdate]
CPUPM5x1=0
Display=0
Coupler0=0
Coupler1=0
Coupler2=0
Coupler3=0
Coupler4=0
Coupler5=0
Coupler6=0
OnboardIO=0
```

```
RtcBatt=0
OnboardCAN=0
LEDboard=0
[UserProg]
UserProgram=0
RetainData=0
AddressData=0
```

By default, not all AC500 processor modules are listed in the SDCARD.INI file. Not until the memory card is inserted in the processor module, all data is added to the SDCARD.ini file and the folders are created.

Table 759: The following error message is displayed, if you insert a non-formatted memory card:

152369164	EC4 Warning	Unable to read the memory card

Other files or subdirectories in the root directory on the memory card are kept unchanged.

The file SDCARD.INI as of V2.5 looks as follows:

```
;File sdcard.ini to be used by CPU FW version >= 2.5.
;For usage in CPU FW version < 2.5 delete all comments and
whitespaces.
;
[Status]
;FunctionOfCard
;0 = Perform no function when inserting the card or voltage ON
;1 = Load user program according to entry in group |UserProg|
;2 = Start firmware update according to entry in group |
FirmwareUpdate|
;3 = Update firmware according to entry in group |FirmwareUpdate|
;    and load user program according to entry in |UserProg|
FunctionOfCard=0

;Note: Any ?? below to be replaced by real CPU's / couplers'
identifiers

[FirmwareUpdate]
; 0 = No update
; 1 = Supported for CPU firmware and for other components of CPU
;     (e. g. display, Onboard I/O, etc.).
;     Not supported for couplers.
;     CPU:
;     Update firmware always from base directory of the CPU (e.g.
FIRMWARE\PM590ETH\).
;     Components:
;     Update firmware always from subdirectory specified by <compkey>
within
;     CPU specific base directory of the component <pathkey>
;     (e.g. FIRMWARE\PM590ETH\LedBoard\2_8_0\).
; 2 = Update with specific version, the update is only performed if
the key <compkey>
;     in a product section <prodsec> according file returns a
;     different result than the current version.
;     If the key <compkey> is missing, no update is performed.
; 3 = Like 2, but check version of file to be newer (instead of
different) than current one.
;     If the key <compkey> is missing, no update is performed.
;11 = Update with file specified in module's section <modsec>,
component's path key
<pathkey>
```

```
;12 = Like 11, but check version of file to be updated differs from
current one.
;13 = Like 11, but check version of file to be updated is newer than
current one.
CPUPM5x1=0       ;<prodsec>=|PM5??|,<compkey>=VERSION, <modsec>=|CPU|,
<pathkey>=Firmware
Boot=0       ;<prodsec>=|PM5??|,<compkey>=PLCBOOT, <modsec>=|CPU|,
<pathkey>=Boot
Display=0       ;<prodsec>=|PM5??|,<compkey>=DISPLAY, <modsec>=|CPU|,
<pathkey>=Display
OnboardIO=0       ;<prodsec>=|PM5??|,<compkey>=ONB_IO, <modsec>=|CPU|,
<pathkey>=OnboardIO
OnboardCAN=0       ;<prodsec>=|PM5??|,<compkey>=ONB_CAN, <modsec>=|
CPU|,
<pathkey>=OnboardCAN
RtcBatt=0       ;<prodsec>=|PM5??|,<compkey>=RTC_BATT, <modsec>=|CPU|,
<pathkey>=RtcBatt
LedBoard=0       ;<prodsec>=|PM5??|,<compkey>=LEDBOARD, <modsec>=|
CPU|,
<pathkey>=LedBoard
Coupler=0       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler0|,
<pathkey>=Firmware
Coupler=1       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler1|,
<pathkey>=Firmware
Coupler=2       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler2|,
<pathkey>=Firmware
Coupler=3       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler3|,
<pathkey>=Firmware
Coupler=4       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler4|,
<pathkey>=Firmware
Coupler=5       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler5|,
<pathkey>=Firmware
Coupler=6       ;<prodsec>=|CM???|,<compkey>=VERSION, <modsec>=|
Coupler6|,
<pathkey>=Firmware

[UserProg]
UserProgram=0    ;Update user program: 0 = no update / 1 = update)
RetainData=0     ;Update retain variables: 0 = no update / 1 = update)
PersistentData=0 ;Update persistent variables: 0 = no update / 1 =
update)
ConfData=0       ;Update configuration data of the onboard Ethernet
CPUs
;                 and the communication modules: 0 = no update / 1 =
update)
CouplerConfig0=0 ;Update internal coupler's configuration: 0 = no
update / 1 = update)
CouplerConfig1=0 ;Update external coupler's configuration slot 1 : 0
= no update / 1 = update)
CouplerConfig2=0 ;Update external coupler's configuration slot 2 : 0
= no update / 1 = update)
CouplerConfig3=0 ;Update external coupler's configuration slot 3 : 0
= no update / 1 = update)
CouplerConfig4=0 ;Update external coupler's configuration slot 4 : 0
= no update / 1 = update)
CouplerConfig5=0 ;Update coupler's config. slot 5 (builtin/fct.): 0 =
no update / 1 = update)
CouplerConfig6=0 ;Update coupler's config. slot 6 (builtin/fct.): 0 =
no update / 1 = update)
```

```
[PM5??]
VERSION=
PLCBOOT=
ONB_IO=
ONB_CAN=
RTC_BATT=
DISPLAY=
LEDBOARD=

[PM5??ETH]
VERSION=
PLCBOOT=
ONB_IO=
ONB_CAN=
RTC_BATT=
DISPLAY=
LEDBOARD=

[CM5??]
VERSION=

[CMETH]
VERSION=

[CPU];
Boot=            ;Path/file of CPU's bootcode to update
Firmware=        ;Path/file of CPU's firmware to update
Display=         ;Path/file of Display's firmware to update
OnboardIO=       ;Path/file of OnboardIO's firmware to update
OnboardCAN=      ;Path/file of OnboardCAN's firmware to update
RtcBatt=         ;Path/file of Rtc/Battery module's firmware to update
LedBoard=        ;Path/file of LED board's firmware to update

[Coupler0]
Firmware=        ;Path/file of internal coupler's firmware to update

[Coupler1]
Firmware=        ;Path/file of external coupler's firmware slot 1 to
update

[Coupler2]
Firmware=        ;Path/file of external coupler's firmware slot 2 to
update

[Coupler3]
Firmware=        ;Path/file of external coupler's firmware slot 1 to
update

[Coupler4]
Firmware=        ;Path/file of external coupler's firmware slot 4 to
update

[Coupler5]
Firmware=        ;Path/file of coupler's firmware slot 5 (builtin/
fct.) to update

[Coupler6]
Firmware=        ;Path/file of coupler's firmware slot 6 (builtin/
fct.) to update
```

The generated result file *SDCARD.RDY* looks as follows:

```
[CPU]
;Result of CPU's bootcode to update
Boot=0
;Result of CPU's firmware to update
Firmware=0
;Result of Display's firmware to update
Display=0
;Result of OnboardIO's firmware to update
OnboardIO=0
;Result of OnboardCAN's firmware to update
OnboardCAN=0
;Result of Rtc/Battery module's firmware to update
RtcBatt=0
;Result of LED board's firmware to update
LedBoard=0

[Coupler0]
;Result of internal coupler's firmware to update
Firmware=0

[Coupler1]
;Result of external coupler's firmware slot 1 to update
Firmware=0

[Coupler2]
;Result of external coupler's firmware slot 2 to update
Firmware=0

[Coupler3]
;Result of external coupler's firmware slot 3 to update
Firmware=0

[Coupler4]
;Result of external coupler's firmware slot 4 to update
Firmware=0

[Coupler5]
;Result of coupler's firmware slot 5 (builtin/fct.) to update
Firmware=0

[Coupler6]
;Result of coupler's firmware slot 6 (builtin/fct.) to update
Firmware=0

;Meaning of results
;  1    Update not yet started
;  2    Update pending, analyzing source file
;  3    Update pending, transmitting source file to destination
component
;  4    Update pending, destination component storing source file\
;  0    No Update to be performed
;100    Update successfully performed
;  5    Update not performed since at least one condition not
fulfilled (e.g. comparison of versions)
;  6    Update not allowed in current PLC state (e.g. RUN)
;  7    Unknown update mode
;  8    Key specifying path/file not found
;  9    Source file not found
; 10    Source file corrupt
; 11    Source file does not match component
; 12    Error reading source file
; 13    Internal Error
; 14    Error writing file
; 15    Not supported by this type of CPU (e.g. Display, Rtc/Battery,
```

```
LED board)
;       or not available (e.g. coupler not present)
; 16    Skipped update of this file due to preceding failure
;       (e.g. failed version check of an associated file)
;-<num> Individual error code on performing update
```

1.6.6.2.3 Content of the memory card for firmware/application update

Only advanced users should apply the instructions in this chapter.

1.6.6.2.4 Storing/Loading the Firmware to the memory card for AC500 V2 products

1. Initialize the memory card, i.e., create the file structure the CPU requires by, for example, inserting a new memory card into the AC500 PLC *Chapter 1.6.6.2.2.4.1 “Using the AC500 PLC for AC500 V2 Products” on page 6349*.
2. Copy the firmware files into the corresponding directory:

AC500 CPU	Directory	File
PM5xy	FIRMWARE\<PLC type>\<firmware version number> Example: FIRMWARE\PM564\2_4_4	*.GZA

3. Change the command file *SDCARD.INI* located in the root directory on the memory card as follows:
   - In section `[Status]` set the key FunctionOfCard to "2", if you only want to update the firmware or "3" for firmware and user program update (i.e. `FunctionOfCard=2`).
   - In section `[FirmwareUpdate]` set the key CPUPM5x1 to "1" [=update always] (i.e. `CPUPM5x1=1`)

A specific firmware version can be loaded. This is done by setting the key CPUPM5x1 to "2" [= update, if versions are different] or "3" [= update, if version on the memory card is newer] and creating an according key for the CPU in the product specific section of *SDCARD.INI*. The firmware has to be copied to the according directory *Chapter 1.6.6.2.2 “Firmware and/or application update with memory card” on page 6340*.

1.6.6.2.5 Storing/Loading the user program to/from the memory card for AC500 V2 products

Storing the user program to the memory card

Online

To store the user program to the memory card, proceed as follows:

1. Build the complete project. *“Project ➔ Clean all and Project ➔ Rebuild all”*.
2. Load the project into the AC500 PLC.
3. Create the bootproject on the controller using *“Online ➔ Create boot project”*.

   The bootproject files (<project name>.PRG.ZIP and <project name>.WEB.ZIP) are loaded into the AC500 PLC and flashed.

   The RUN LED on the AC500 PLC flashes while data flashing is in progress.

4. Insert the memory card. If the memory card does not already contain the required file structure, the structure will be created ↳ *Chapter 1.6.6.2.2.4.1 “Using the AC500 PLC for AC500 V2 Products” on page 6349*.

> **NOTICE!**
> **Overwriting of files on the memory card**
> If a user program is already stored on the memory card, i.e., the directory UserData\PM5x1\UserPrg already contains the files DEFAULT.PRG and DEFAULT.CHK, these files will be overwritten without any warning.
>
> If you want to store several user programs to the memory card, you have to copy them into other directories using the PC.

5. Open the PLC Browser/ PLC Shell and enter the command **"sdappl"<ENTER>**.

   The files <project name>.PRG.ZIP and <project name>.WEB.ZIP are loaded from the user flash memory and stored as BOOT.ZIP / WEBVISU.ZIP to the directory User-Data\PM5x1\UserPrg on the memory card.

   In the file SDCARD.INI, the parameter `FunctionOfCard` is set to 1 (bit 0 = 1) and the parameter `UserProgram` is set to 1, i.e. the function "Load the user program" is activated.

   The RUN LED on the AC500 PLC flashes while writing to the memory card is in progress.

   If you insert the memory card containing the user program into the AC500 PLC, the memory card is loaded into the user flash memory of the AC500 PLC.

Offline

Furthermore it is possible to create a project offline. Before creating it offline the project has to be compiled. After creating the bootproject the files are available at according storage folders of the project.

- PM5xy: <project name>.PRG.ZIP / <project name>.WEB.ZIP has to be renamed to BOOT.ZIP / WEBVISU.ZIP and copied to the according PM5xy CPU folder: USER-DATA\PM5xx\USERPRG folder.
- CM574-RS: Copy the created CM574 ZIP file into the according PM5xy CPU folder: USER-DATA\PM5xx\USERPRG and rename it to CM574_X1.bpz (X = Slot number).
-

Loading the user program from the memory card to the AC500 PLC

If a memory card is inserted into the AC500 PLC when the CPU is in STOP mode, or if the memory card is already inserted when switching on the control voltage, the file structure on the memory card is checked. If the file structure exists, the file SDCARD.INI is read. If the parameter "FunctionOfCard" is set to 1 (bit 0 = 1) and the parameter "UserProgram" = 1 and for the CM574-RS (CouplerConfigX=1 X = Slotnumber), the files DEFAULT.PRG and DEFAULT.CHK (CM574_X1.PRG and CM574_X2.CHK (X=Slotnumber) (*PS501V1*) or Boot.zip (CM574_X1.bpz - X = Slot number) (*PS501V2.x*) in the directory UserData\PM5xy\UserPrg on the memory card are loaded into the user flash memory of the AC500 PLC.

The RUN LED on the AC500 PLC flashes while loading. Flashing the user program is in progress.

The loaded program is activated after a CPU restart. If the user program cannot be loaded (for example, due to missing files, wrong directory structure or mismatching project for the controller), a corresponding error message appears. A summary of the memory card errors can be found in the section memory card error messages.

If you insert the memory card into the AC500 PLC when the CPU is in RUN mode, the user program is not loaded independent of the settings for the parameters "FunctionOfCard" (bit 0=1) and "UserProgram" (=1). Thus, the function "Load user program" can be deactivated with the PLC Browser "sdfunc 0" even if no PC card reader is available.

1.6.6.2.6 Storing/Reading user data for AC500 V2 products

Structure of data files stored on the memory card

Depending on the AC500 CPU type, the data is stored in the following memory card directory:

AC500 CPU	Directory	File
PM5xy	..\UserData\PM5xy\UserDat	USRDATxx.DAT
PM5xy-ETH	..\User-Data\PM5xyETH\UserDat	USRDATxx.DAT

A maximum of 100 files (USRDAT00.DAT...USRDAT99.DAT) can be stored in one directory.

Each data file USRDATxx.DAT can be divided into individual sectors, if necessary. The "sector label" enclosed in square brackets (such as [Sector_01]<CR><LF>) indicates the start of the sector. Within a sector, data are saved as data sets in ASCII format. The individual values of a data set are separated by semicolon. Each data set is closed with <CR><LF> (0Dhex, 0Ahex).

This enables the direct import/export of the data from/to MS Excel. The data files can be viewed and edited using a standard ASCII editor (such as Notepad).

When saving/loading data files, observe the following rules:

- Data sets within a sector must always have the same number of values.
- Data sets in different sectors can have a different number of values.
- Values of integer data types can be stored. REAL or LREAL variables cannot be stored.
- The values of a data set must have the same data format (BYTE, WORD, INT,..).
- A sector can have data sets with different data format.
  Warning: The user has to know the structure of the data for reading them.
- The data sets are always appended to the end of the file when writing.
- Searching for a "sector label" within a file is possible when reading it.
- Data sets can be read starting from a particular "sector label".
- A particular data set of a sector cannot be read or written.
- If you want to read each data set individually, a "sector label" must be inserted before each data set.
- Reading and writing the data with help of the user program is done with the blocks SD_READ and SD_WRITE.
- The values of a data set must be available in variables successively stored in the CPU (e.g., ARRAY, STRING, %M area).
- A data file can be deleted with help of the CPU program.
- Individual data sets and/or sectors cannot be deleted with the user program. This has to be done on the PC using an ASCII editor such as Notepad.

Data file examples

Example 1: Data file USRDAT5.DAT without sectors:

-> 5 data sets, each with 10 DINT values:

600462;430;506;469;409;465;466;474;476;-1327203

600477;446;521;484;425;480;482;490;491;-1327187

600493;461;537;499;440;496;497;505;507;-1327172

600508;477;552;515;456;511;513;521;522;-1327156

600524;492;568;530;471;527;528;536;538;-1327141

Example 2: Data file USRDAT7.DAT with sectors:

-> 3 sectors, each with 3 data sets and 10 DINT values per data set:

[Sector_01]

610439;10408;10483;10446;10387;10442;10444;10452;10453;-1317225

610455;10423;10499;10462;10402;10458;10460;10467;10469;-1317209

610476;10445;10520;10483;10424;10479;10481;10489;10490;-1317188

[Sector_02]

610570;10539;10614;10577;10518;10573;10575;10583;10584;-1317094

610585;10554;10630;10592;10533;10589;10591;10598;10600;-1317078

610602;10571;10646;10609;10550;10605;10607;10615;10616;-1317062

[Sector_03]

610701;10670;10746;10708;10649;10704;10706;10714;10715;-1316963

610717;10686;10761;10724;10665;10720;10722;10730;10731;-1316947

610739;10708;10783;10746;10686;10742;10744;10751;10753;-1316926

Function blocks for storing/reading user data to/from the memory card

The following function blocks from the SysInt_AC500_Vxx.lib library are used to write and read user data from the CPU program to/from the memory card:

- *Chapter 1.5.4.19.2.23 “SD_WRITE” on page 1563* - writes user data
- *Chapter 1.5.4.19.2.22 “SD_READ” on page 1558* - reads user data

The SysInt_AC500_Vxx.lib library and the SysExt_AC500_Vxx.lib library are loaded automatically when creating a project for an AC500 CPU.

Storage folder: ..\Data Storage\SD memory card.

Table 760: Inputs and outputs of SD_WRITE

Name	Type	Assignment
Inputs		
EN	BOOL	The FALSE->TRUE edge starts the write process
ATTRIB	BYTE	Write attribute of the function block: 1 - Delete file 2 - Write append 3 - Write sector label
FILENO	BYTE	Consecutive file number 0 <=xx<=99 (USRDATxx.DAT)
SEG	POINTER TO STRING	Pointer to sector label string (via ADR operator)

Name	Type	Assignment
FORMAT	BYTE	Data format: 00 hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT
NVAR	WORD	Number of variables to be written
ADRVAR	DWORD	Address of the first variable, starting from which the data are available in the CPU (via ADR operator)
Outputs		
DONE	BOOL	Function completed
ERR	BOOL	Error: FALSE=no error, TRUE=error
ERNO	INT	Error number

Table 761: Inputs and outputs of SD_READ

Name	Type	Assignment
Inputs		
EN	BOOL	The FALSE->TRUE edge starts the read process
ATTRIB	BYTE	Read attribute of the function block: 1 - Open file, seek sector, read data set 2 - Open file, read data set 3 - Open and read next data set 4 - Read data set, close file 5 - Close file
FILENO	BYTE	Consecutive file number 0 <=xx<=99 (USRDATxx.DAT)
SEG	POINTER TO STRING	Pointer to sector label string (via ADR operator)
FORMAT	BYTE	Data format: 00 hex - 0 - BYTE 01 hex - 1 - CHAR 10 hex - 16 - WORD 11 hex - 17 - INT 20 hex - 32 - DWORD 21 hex - 33 - DINT
NVAR	WORD	Number of variables to be read
ADRVAR	DWORD	Address of the first variable, starting from which the data are stored to the CPU (via ADR operator)
Outputs		
DONE	BOOL	Function completed

Name	Type	Assignment
ERR	BOOL	Error: FALSE=no error, TRUE=error
ERNO	INT	Error number

Deleting a data file stored on the memory card

To delete a data file from the memory card, proceed as follows:

1. Insert the memory card.
2. Call the function block *Chapter 1.5.4.19.2.23 “SD_WRITE” on page 1563* with the following settings:

EN	:= TRUE	
ATTRIB	:= 1	(* delete *)
FILENO	:= 0...99	(* number of the file to be deleted *)
SEG, FORMAT, NVAR, ADRVAR - any		

Storing user data to the memory card

Data file without sectors

To store user data to the memory card in a data file without sectors, proceed as follows:

1. Insert the memory card.
2. Write a data set by calling the function block *Chapter 1.5.4.19.2.23 “SD_WRITE” on page 1563* with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 2	(* write append *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	
If no corresponding file exists, then it is created. The write process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0		

3. Further data sets can be written with the same function block settings after the completion message is displayed (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN.

The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process. An "Open file / Write file / Close file" procedure is performed.

Data file with sectors

Proceed as follows to store user data to the memory card in a data file with sectors:

1. Insert the memory card.
2. Write the sector label by calling the function block ↳ *Chapter 1.5.4.19.2.23 “SD_WRITE” on page 1563* with the following settings:

EN	:= TRUE	
ATTRIB	:= 3	(* write sector *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	
If no corresponding file exists, then it is created. The sector is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0		

3. Write a data set by calling the function block SD_WRITE with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 2	(* write append *)
FILENO	:= 0...99	(* number of the file to which the data set is to be appended *)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable to be written	
The write process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A write error is indicated by ERR:=TRUE and ERNO<>0		

4. Further data sets can be written with the same function block settings after the completion message is displayed (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN.
5. If you want to write further sectors and data sets, repeat the steps as of step 2.

The file USRDATxx.DAT is saved as USRDATxx.BAK for each write process. An "Open file / Write file / Close file" procedure is performed.

Loading user data from the memory Card for AC500 V2 products

Data file without sectors

Proceed as follows to read user data from a data file without sectors on the memory card and write them to the CPU:

1. Insert the memory card.
2. Read a data set by calling the function block *Chapter 1.5.4.19.2.22 “SD_READ” on page 1558* with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 2	(* open / read *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
The read process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A read error is indicated by ERR:=TRUE and ERNO<>0		

3. Further data sets can be read with the following settings after the completion message is displayed (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 3	(* continue read *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.		

4. To read a further data set and to close the file afterwards, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 4	(* read / close *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.		

5. To close the file, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 4	(* close *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set (*any*)	
ADRVAR	:= Address of the first variable (*any*)	

Data file with sectors

Proceed as follows to read user data from a data file with sectors on the memory card and write them to the CPU:

1. Insert the memory card.
2. Seek a sector label and read a data set by calling the function block ↳ *Chapter 1.5.4.19.2.22 “SD_READ” on page 1558* with the following settings:

EN	:= TRUE	(* FALSE/TRUE edge starts writing *)
ATTRIB	:= 1	(* open / seek / read *)
FILENO	:= 0...99	(* number of the file which is to be read *)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
The read process is successfully completed when output DONE:=TRUE and output ERR:=FALSE. A seek error is indicated by ERR:=TRUE and ERNO<>0		

3. Further data sets can be read with the following settings after the completion message is displayed (output DONE=TRUE). This process is started with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 3	(* continue read *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label (*any*)	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.		

4. If you want to read further sectors / data sets, close the file and repeat the steps 2 and 3.

5. To read a further data set and to close the file afterwards, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 4	(* read / close *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set	
ADRVAR	:= Address of the first variable into which data are to be written	
If an unexpected sector label or the end of file (EOF) is detected when reading, a corresponding error message is generated.		

6. To close the file, call the function block SD_READ with the following settings after the completion message (output DONE=TRUE) and start the process with a FALSE/TRUE edge at input EN:

EN	:= TRUE	(* FALSE/TRUE edge starts reading *)
ATTRIB	:= 4	(* close *)
FILENO	:= 0...99	(* number of the file which is to be read*)
SEG	:= Address of the variable of the sector label	
FORMAT	:= Data format	
NVAR	:= Number of values in data set (*any*)	
ADRVAR	:= Address of the first variable (*any*)	

1.6.6.2.7 Storing force values onto memory card

The force values can be written onto a memory card to reactivate them after a loss of power supply voltage. A precondition for storing is that a memory card is plugged into the CPU.

For storing the force values, there is no need for a special program. It is performed automatically if the respective options are configured in the CODESYS project.

1. In the CODESYS open the *“Resources”* tab. Then, open the *“Target Settings”*.
2. In the *“General”* tab, activate the check boxes *“Retain forcing”* and *“Save”*.
3. Compile the project and store it to the user flash memory of the CPU (use *“Create boot project”*).
   ⇨ The current force values are stored onto the memory card at every forcing and a warning message is displayed with every logout from the PLC. To keep the forced values, confirm this message with **No**.

After a loss of power supply voltage, the values are read from the memory card automatically and the forcing is activated.

The force values are only deleted from the memory card by deactivation of the forcing function.

1.6.6.3 Data storage in flash memory for AC500 V2 products

Used function blocks

The library "SysInt_AC500_V10.lib" ↳ *Chapter 1.5.4.19 “Internal system library” on page 1500* contains the following function blocks which are used to store data in the flash memory:

Block	Function
↳ *Chapter 1.5.4.19.2.16 “FLASH_DEL” on page 1542*	Deletes a data segment in the flash memory
↳ *Chapter 1.5.4.19.2.17 “FLASH_READ” on page 1544*	Reads a data segment from the flash memory
↳ *Chapter 1.5.4.19.2.18 “FLASH_WRITE” on page 1547*	Writes a data segment to the flash memory

1.6.6.4 Flash disk for AC500 V2 products

Some AC500 CPUs (e.g. PM592-ETH, PM595) are equipped with an non-removable and non-volatile flash disk. It contains no moving parts. The flash disk is designed for operation in industrial environments and applications. A wear levelling algorithm and a power-fail protected file system provide industrial robustness. To write on the flash disk the ↳ *Chapter 1.5.4.4 “CAA_File library” on page 789* is required.

NOTICE!

The flash disk has a finite number of write cycles.

A warning will be generated when the flash disk reaches 80 % of its max. number of write cycles. The flash disk will be set into read-only mode to avoid data loss, when the max. number of write cycles is reached.

Number of max. write cycles

Technically, the flash chip used in flash disk has 100000 Erase-Cycles (Write cycles).

Due to the produced write overhead, the optimum achievable number of write cycles is 50000 (for typical payload sizes of 256 kB).

Example

The write overhead is indicated by the *write amplification factor* (WAF).

$$\mathrm{WAF} = \frac{\text{Flash Write (in Bytes)}}{\text{Host Write (in Bytes)}}$$

Table 762: Rule of thumb for assessing the flash lifetime for an application:

Typical payload sizes	WAF	Max. write cycles
256 kB	2	50000
128 kB	4	25000
64 kB	8	12500
...	...	...
1024 Byte	512	< 200
512 Byte	1024	< 100

Lifetime of flash disk will also depend on the operating environment.

E.g. high ambient temperatures will impose stress on the user flash memory and reduce the total overwrites achievable.

Read cycles are unlimited.

Important hints

- Programmer should keep the amount of cyclic written data low to assure long availability.
- To increase the number of write cycles, the programmer could choose to use only 90 %, 75 % or even 50 % of the flash disk capacity. The disk space left free will automatically be used by the wear levelling algorithm.
- The wear levelling information of the flash disk can be read from the PLC browser with the command `diskcfg settings flash disk`. In applications with high data generating processes (typical more then 1,5 GByte/day) this information should be monitored during implementation.

1.7 Diagnosis and debugging for AC500 V2 products

1.7.1 The diagnosis system

The AC500 contains a diagnosis system that allows to manage up to 100 error messages in a circular buffer. For each of these events, a time stamp with date and time based on the controller's real-time clock (RTC) is generated in the runtime system.

Each error message has a unique internal error number. The error number provides the following information:

- State: error occured (come), error disappeared (gone), error acknowledged ↳ *Chapter 1.7.1.5.1 “State (come, gone, acknowledged)” on page 6370*
- Error class ↳ *Chapter 1.7.1.5.2 “Error severity” on page 6371*
- Faulty component or interface
- Faulty device
- Faulty module
- Faulty channel
- Error identifier ↳ *Chapter 1.7.1.5 “Structure of error numbers” on page 6369*

↳ *Chapter 1.7.1.5 “Structure of error numbers” on page 6369*

1.7.1.1 Access to diagnosis data

Diagnosis data of the devices can be accessed by:

- CPU display ↳ *Chapter 1.7.1.2 “Diagnosis in CPU display” on page 6365*
- Automation Builder: status line ↳ *Chapter 1.7.1.3 “Diagnosis in Automation Builder” on page 6367*
- Automation Builder: PLC commands ↳ *“PLC browser commands” on page 6368*
- IEC application ↳ *Chapter 1.7.1.4 “Diagnosis in IEC application” on page 6369*

1.7.1.2 Diagnosis in CPU display

1.7.1.2.1 Device state

If there is at least one active diagnosis message, the error LED ERR is on.

The behavior of the error LED depends on the setting of CPU parameter *“Error LED”* ↳ *Chapter 1.6.5.2.3.3 “Parameters of the processor module” on page 5839*.

Diagnosis of AC500-eCo CPUs can only be shown by LED ERR at CPU. No display is available.

General information on the LEDs, the display and the function keys can be found in chapter ↳ *Chapter 1.6.4.1.5 “LEDs, display and function keys on the front panel” on page 5422*.

1.7.1.2.2 Diagnosis descriptions

If one or several unacknowledged errors exist, the errors can be displayed and acknowledged using the *[DIAG]* key. The latest unacknowleged error is displayed first (LIFO-principle: last in first out). Pressing the *[DIAG]* key the first time displays the error class ↳ *Chapter 1.7.1.5.2 “Error severity” on page 6371* and error identifier ↳ *Chapter 1.7.1.5.3 “Error identifiers” on page 6371*. After this, pressing the *[DIAG]* key several times browses through the detailed information:

d1 - Component

d2 - Device

d3 - Module

d4 - Channel

If the "d4" information is displayed and the *[DIAG]* key is pressed once more, the error class/ error identifier is re-displayed.

If you quit the diagnosis display by pressing *[ESC]*, the error remains unacknowledged and will be displayed again when pressing the *[DIAG]* key.

If you quit the diagnostic display with the *[OK]* key, the error is acknowledged.

The LED ERR goes off when all errors are acknowledged.

Example Battery empty or not installed

Key	Display	Description
[DIAG]	E4 008	E4 = warning Error identifier 008 = empty or missing
[DIAG]	d1 009	d1 = component 009 = CPU
[DIAG]	d2 022	d2 = device 022 = battery
[DIAG]	d3 031	d3 = module 031 = no specification
[DIAG]	d4 031	d4 = channel 031 = no specification
[DIAG]	E4 008	See above
[ESC]	run/StoP	Diagnosis display is quit without error acknowledgement.
[DIAG]	E4 008	See above
[OK]	run/StoP	Diagnosis display is quit with error acknowledgement. If no further unacknowledged errors exist, the LED ERR goes off.

1.7.1.3 Diagnosis in Automation Builder

When the Automation Builder is switched to online mode, incoming error messages or state changes of an error message (come, gone, acknowledged) are displayed as plain-text in the status line *Chapter 1.7.2 “Online diagnosis in Automation Builder” on page 6374*.

Example The error "Battery empty or not installed" is displayed in online mode as follows:

#152502216: x 1970-01-01 06:33:53 E4 : No battery or battery empty

#152502216	-	Error number
x	-	State: + = error occured (come), - = error removed (gone) x = error acknowledged
1970-01-01 06:33:53	-	Time stamp: date and time of acknowledgement
E4 :	-	Error class 4 = Warning
No battery or battery empty	-	Error description

The error description is read from the file Errors.xml and displayed according to the error number. The language of the error description is dependent of the language in Automation Builder. Errors which do not have an entry in the file Errors.xml are displayed without error description. The file Errors.xml is part of the target support package (TSP) and located in the directory ..\Targets\ABB_AC500 or ..\Targets\ABB_AC500\AC500_V12.

PLC browser commands

With the PLC browser commands all errors or errors of a certain error class can be displayed and/or acknowledged, the complete diagnosis system can be deleted *Chapter 1.7.2.5.5 “AC500-specific PLC browser commands” on page 6382*.

The error information of the internal error number and the plain-text and online text is diplayed as plain-text interpretation of component/interface, device, module and channel.

1.7.1.3.1 Diagnosis history

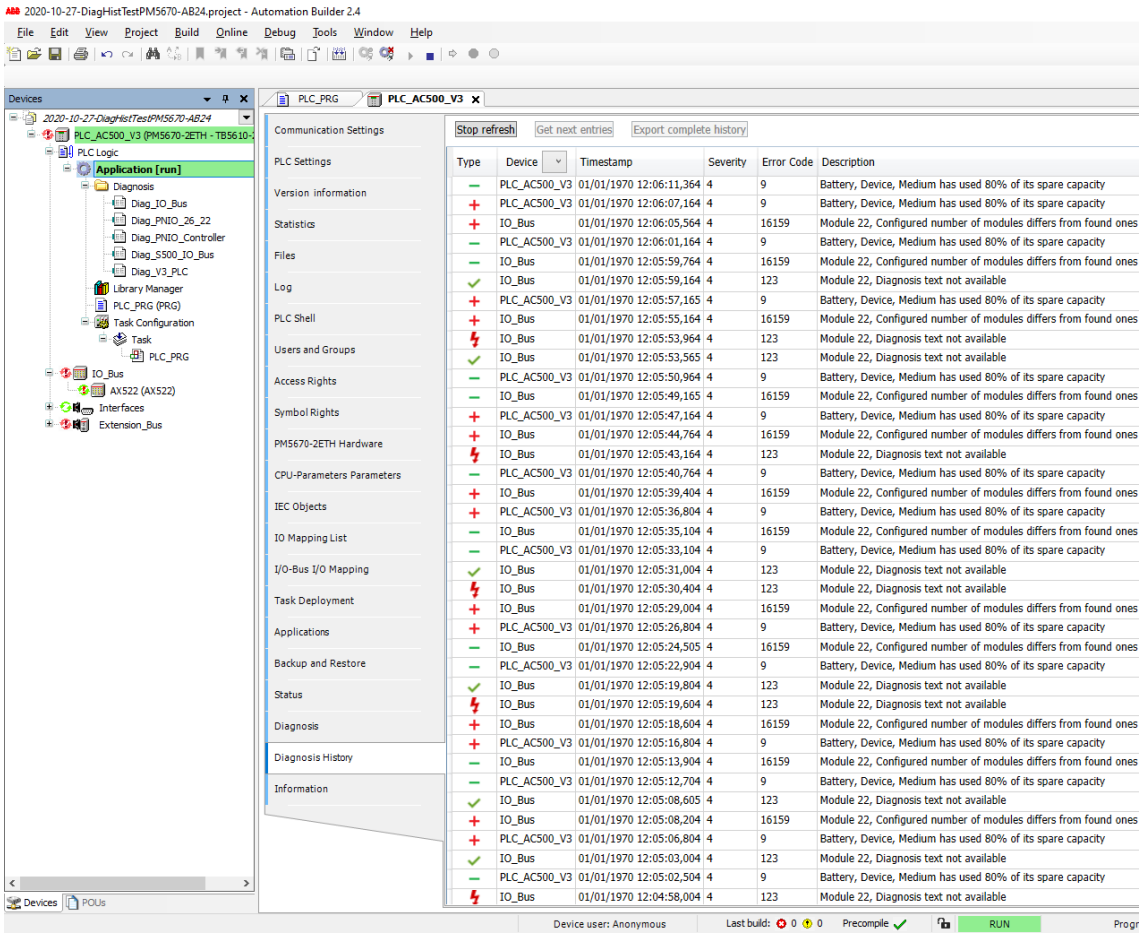
Diagnosis history is available as of Automation Builder 2.4.0 / System FW 3.4.0 the diagnosis system has been extended with diagnosis history.

The 'Diagnosis History' view provides an overview of the current and past system events that resulted in a diagnosis event.

- Incoming diagnosis events are indicated with +.
  After the problem that causes a diagnosis event has been resolved, this diagnosis event is indicated automatically with −.
- Alarm events, e.g. PROFINET alarms are indicated with .
  In the 'Diagnosis' view the user can acknowledge an alarm. Note that an alarm event can be acknowledged though the problem that causes the alarm still persists.
  The acknowledge action is indicated with on the concerning event entry. If the icon changes to , the acknowledge action has been completed by the PLC.

The following buttons are available in the 'Diagnosis History' view:

- Start/Stop refresh:
  Enables or disables the automatic refresh mode. In refresh mode new diagnosis events will be displayed automatically. Only the last 100 entries are shown in this view, the latest events on top of the list.
- Get next entries:
  Adds the previous (older) 100 diagnosis events at the bottom of the list.
- Export complete history:
  Creates a csv file with all events from the diagnosis history (not only the visible ones).



1.7.1.4 Diagnosis in IEC application

The diagnosis messages can be accessed with the library SysInt_AC500_Vxx.lib. All function blocks from the folder "Diagnosis" can be used for diagnosis. ↳ *Chapter 1.5.4.19 “Internal system library” on page 1500*

1.7.1.5 Structure of error numbers

For each error, an internal error number is stored in the firmware. This error number is coded as follows:

Table 763: Error number, 32 bit

Bit	Bit length	Possible values	Definition	Description
-	4 bits	-	State	↳ *Chapter 1.7.1.5.1 “State (come, gone, acknowledged)” on page 6370*
28 ... 29	2 bits	0 ... 3	Error class	E1 ... E4 ↳ *Chapter 1.7.1.5.2 “Error severity” on page 6371*

Bit	Bit length	Possible values	Definition	Description
24 ... 27	4 bits	0 ... 15	Component	1 ... 6 = external communication module 8 = local I/O 9 = CPU 10 = internal communication module 11 = COM1 12 = COM2 13 = FBP 14 = I/O bus 15 = user *Chapter 1.7.3.1 "Possible error combinations" on page 6429*
16 ... 23	8 bits	0 ... 255	Device	Component specific *Chapter 1.7.3.1 "Possible error combinations" on page 6429*
11 ... 15	5 bits	0 ... 31	Module	Device specific, but mostly: 1 = initialization 2 = runtime 3 = project / configuration 4 = protocol 31 = device itself
6 ... 10	5 bits	0 ... 31	Channel	Module specific *Chapter 1.7.3.1 "Possible error combinations" on page 6429*
0 ... 5	6 bits	0 ... 63	Error identifier	0 ... 63 *Chapter 1.7.1.5.3 "Error identifiers" on page 6371*

1.7.1.5.1 State (come, gone, acknowledged)

In addition to the error information, the diagnosis message also contains state information (1 bit per state). Each state is set by a specific event:

State value	Description
Bit 0	not used
Bit 1	Error occured (come)
Bit 2	Error removed (gone)
Bit 3	Error acknowledged

The diagnosis message is generated when an error occurs. In this case, the state bit 1 is set. If this error is acknowledged or removed afterwards, the corresponding state bits are set additionally.

1.7.1.5.2 Error severity

Bit 29	Bit 28	Error class	Type	Description	Example
0	0	E1	Fatal errors	Safe operation of the operating system is no longer ensured.	Checksum error in system flash, RAM error
0	1	E2	Severe error	The operating system works correctly, but the error-free execution of the user program is not ensured.	Checksum error in user flash, task cycle times exceeded
1	0	E3	Minor error	It depends on the application whether the user program has to be stopped by the operating system or not. The user decides which reaction is to be done.	Flash memory cannot be programmed, I/O module failed
1	1	E4	Warnings	Errors that occur on peripheral devices or that will have an effect only in the future. The user decides which reactions are to be done.	Short circuit in an I/O module, battery empty/not installed

Errors with error severitiy 1 - fatal errors

Errors with error severity 1 are not entered in the diagnosis system. These errors do not allow normal operation of the PLC. These errors are detected during PLC start-up and stop the PLC immediately.

Examples are RAM errors or checksum errors when starting the firmware.

Such errors are indicated by rapid flashing of the ERR LED.

1.7.1.5.3 Error identifiers

The error identifier specifies which kind of error occured. It is kept generally in order to reach a maximum systematic. The exact meaning of each error depends on the information provided by the error messages. The error message is a combination of the error identifier and the information where the error occured.

Error identifier	Description
0	General
1	Wrong value

Error identifier	Description
2	Invalid value
3	Timeout
4	Highest level
5	High level
6	Low level
7	Lowest level
8	Empty or missing
9	Full
10	Too big / overflow
11	Too small
12	Read
13	Write
14	Delete
15	Alloc memory
16	Free memory
17	Access
18	Test
19	Checksum
20	Message
21	Put message
22	Get message
23	Wait message
24	Message deleted
25	Wait answer
26	Config data
27	No config
28	Different config
29	Write config
30	Read config
31	Wrong type or model
32	Unknown type or model
33	Wait reset
34	Wait ready
35	Wait run
36	Wait com
37	Cycle time
38	Exception
39	Unknown POU
40	Version
41	Transmit

Error identifier	Description
42	Receive
43	Internal
44	No adjustment values
45	Cut wire
46	Overload
47	Short circuit
48	Overload / Cut wire
49	Short-circuit / Cut wire
50	Overload / Short-circuit
51	Overload / Short-circuit / Cut wire
52	Lost value
53	Changed
54	Conflict
55	Tolerance
56	
57	
58	
59	
60	
61	
62	
63 (max.)	others
63 is the absolute maximum	

1.7.1.6 Diagnosis history file

Diagnosis history is available as of Automation Builder 2.4.0 / System FW 3.4.0 the diagnosis system has been extended with diagnosis history.

Diagnosis history is the entry of all diagnoses into a file according to their time of occurrence.

The diagnosis history file is in the root directory of the user disk and has the name *“DiagHistory.csv”*. The max. number of entries is 2000. When 2000 entries are reached, the oldest entry is overwritten. The max. size of the extended data is 32 bytes.

An entry consists of following data:

Name	Type	Comment	Example
timestamp	ARRAYDT OF BYTE	RTC time of event in milliseconds consists of diTimestamp in DT format and uiMs milliseconds.	1603371910177
event	BYTE	Event type (1=comes, 2=gone).	1
class	BYTE	Severity of error event.	4
compID	UDINT	Component ID	270540802

Name	Type	Comment	Example
conn	UDINT	Connector	0xb17777ac
connIdx	UDINT	Connector index	0
sub	DWORD	SubsystemID: Any number describing detail/location within device, device specific	369098752
addl	DWORD	AdditionalID: Additional number describing detail/location within device, optional, device specific	0
error	DWORD	Error code	9
extended data	ARRAYDT OF BYTE	Extended diagnosis data, max. 32 bytes	

As shown in the example data of the diagnosis history file is not easily readable. The entries must be interpreted according to device and/or fieldbus. Therefor the Automation Builder consists a special view for diagnosis history ↳ *Chapter 1.7.1.3.1 “Diagnosis history” on page 6368*.

With the entries CompID, conn and connID, the device generating the event is clearly identified in the device tree.

If the PLC configuration is changed, the values of this entries may be changed also.

Therefore, the diagnosis history will be deleted during each download.

1.7.2 Online diagnosis in Automation Builder

1.7.2.1 Short description and overview

To use the diagnosis system in Automation Builder, login to the online mode is required ↳ *Chapter 1.7.2.2 “Entering/leaving the online mode” on page 6375*. The online diagnosis in Automation Builder consists of a set of partly animated, mostly read only views. They can be invoked by a double-click on a project tree element which shows a circle indicating that this element is able to show diagnosis messages ↳ *Chapter 1.7.2.3 “Project tree in online mode” on page 6375*.

Available online diagnosis and statistics:

- **Error messages**
  When the Automation Builder is switched to online mode, incoming error messages or status changes of an error message (come, gone, acknowledged) are displayed as plain-text in the status line. ↳ *Chapter 1.7.2.4 “Error messages, warnings and notes (dialogs)” on page 6376*
- **CPU/PLC diagnosis**
  ↳ *Chapter 1.7.2.5 “CPU diagnosis views” on page 6378*.
  All errors or errors of a certain error class can be displayed and/or acknowledged using the PLC browser. Also the complete diagnosis system can be deleted ↳ *Chapter 1.7.2.5.4 “PLC browser” on page 6381*.
- **I/O module diagnosis**
  ↳ *Chapter 1.7.2.6 “Live values in views with I/O components” on page 6391*.
- **Communication module and fieldbus diagnosis**
  ↳ *Chapter 1.7.2.7 “Communication module and fieldbus diagnosis” on page 6392*

1.7.2.2 Entering/leaving the online mode

Prerequisite: Set the gateway before entering the online mode. ↳ *Chapter 1.6.5.2.13 “Gateway configuration” on page 6116*

Enter the online mode

Right-click the PLC tree node and select *“Login”*.

The Automation Builder project login to online mode updates the latest changes of the project.

> *The online mode can be entered or left for each PLC in the project separately.*

Leave the online mode

Right-click the PLC tree node and select *Logout*.

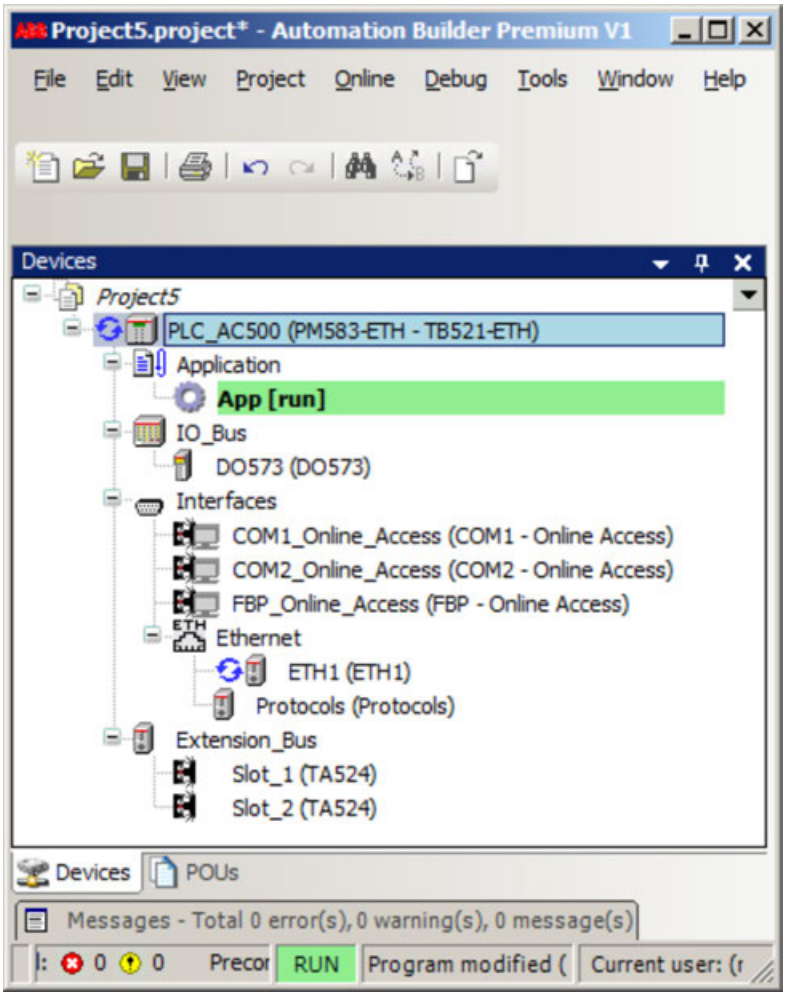
When online mode is active, a thread is running on Automation Builder project which sends cyclically a message to the PLC and expects a response. If the PLC does not respond, the online mode is left programmatically.

1.7.2.3 Project tree in online mode

When Automation Builder enters the online mode internally, an identification message is sent to all configured communication modules.

The connection's status can be recognized by a symbol in the device tree:

	Module responds to identification message and is available for online connection.
	Module responds to identification message, but some warning messages are available. See ↳ *Chapter 1.7.2.4 “Error messages, warnings and notes (dialogs)” on page 6376*.
	Module does not respond to identification message and is not available for online connection.



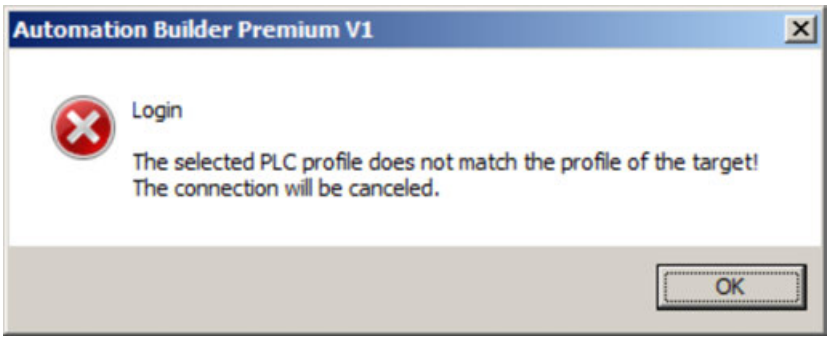
The identification is done once when switching from offline to online mode.

If a communication module is a master unit and contains several configured slave devices below the level (e.g., PROFIBUS, CANopen, PROFINET) will always have the same master, i.e. there is no identification check for the slaves of the fieldbus.

Apart from the online availability the blue circle icon indicates the possibility to activate a diagnosis view on the right screen by a double-click the tree node with the green circle.

1.7.2.4 Error messages, warnings and notes (dialogs)

Wrong target



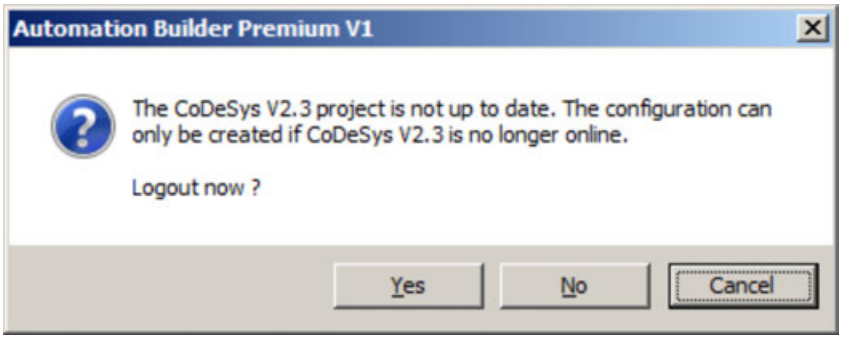
This error message appears if there is a mismatch between the selected target in Automation Builder and the CPU hardware. For example, the selected CPU type is PM583-ETH and in Automation Builder project, the configured CPU type is PM573-ETH.

A login to the CPU is not possible.

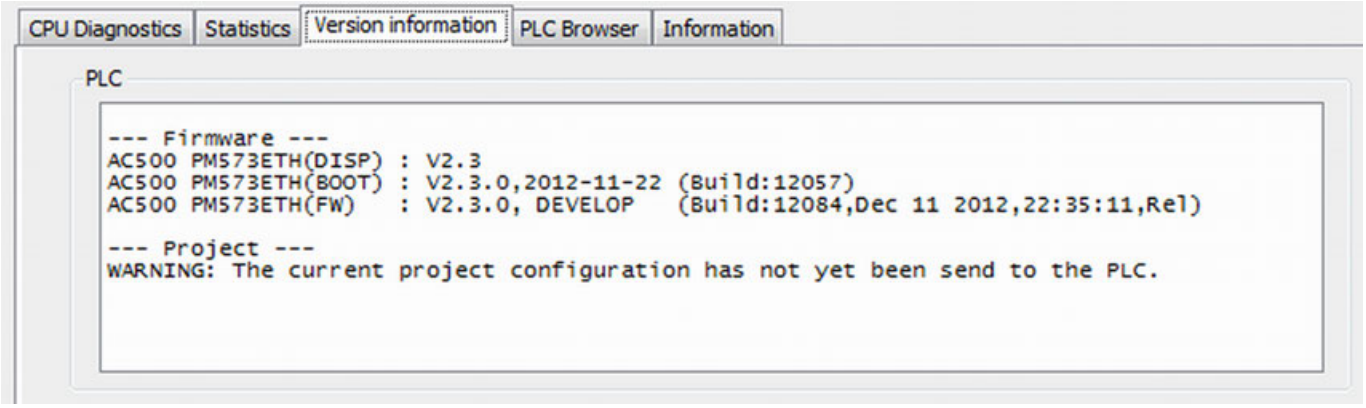
To avoid this error message, correct the project to the appropriate CPU target.

Configuration not yet in PLC

The following warning message will only appear with CPU firmware V2.3.0 or higher.



This warning message appears if the current project in Automation Builder and the project located inside the PLC are different.

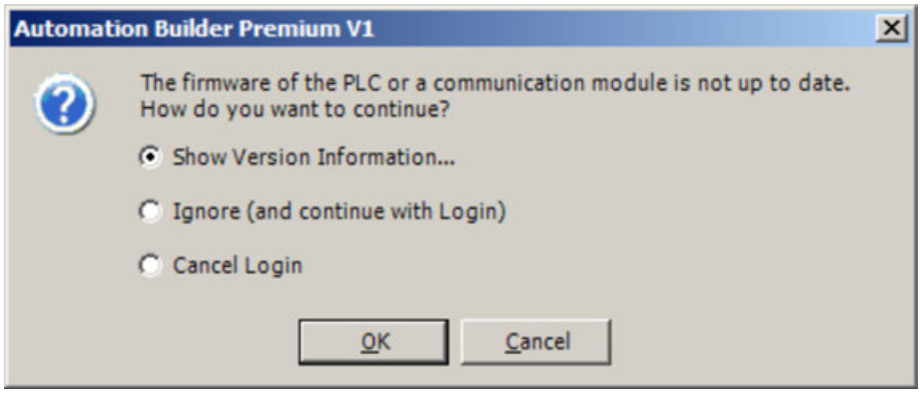


To avoid this warning message, perform the following tasks:

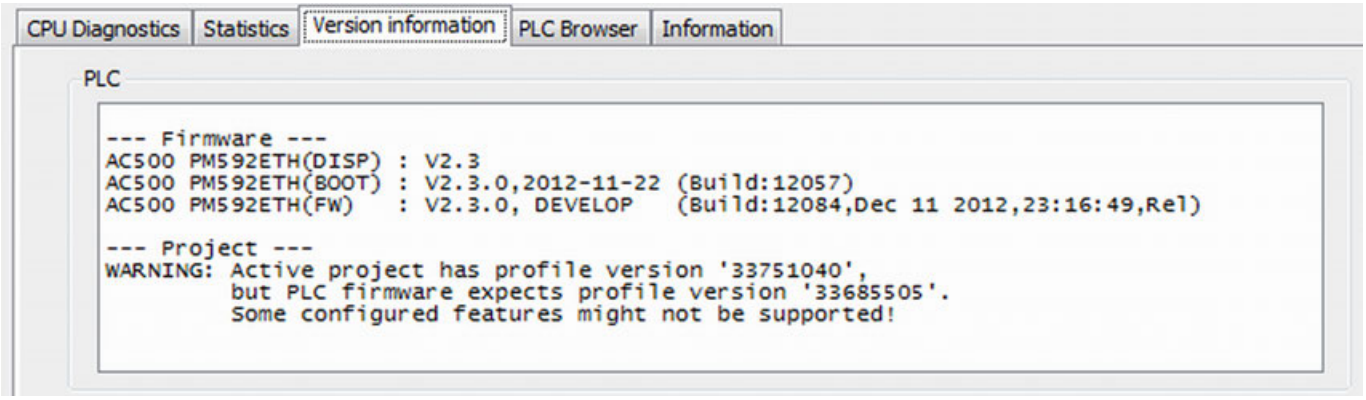
- Log out from the PLC.
- Re-create configuration data (right-click on *"AC500 ➔ Check configuration"*).
- Download the IEC application program to the PLC via CODESYS V2.3.x (a full download is needed).
- Log in for Online Diagnosis again.

Old profile in firmware

The following warning message will only appear with CPU firmware V2.3.0 or higher.



This warning messages appears, if the firmware version inside the PLC is older than the version profile of Automation Builder and therefore, the CPU's firmware does not support all features of the current version profile of Automation Builder.

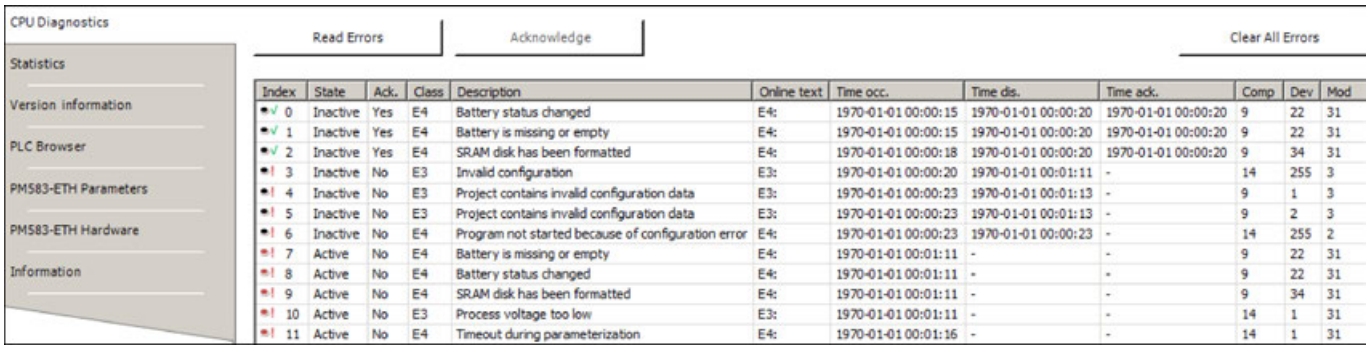


To avoid this warning message, perform one of the following tasks:

- Update the CPU's firmware to the version which is equivalent to the Automation Builder version profile.
- Select a version profile in Automation Builder which is equivalent to the firmware version of the CPU.

1.7.2.5 CPU diagnosis views

1.7.2.5.1 CPU diagnostics (error log)



“Read Errors”: fetches the actual contents of the CPU diagnosis buffer (max. 100 entries possible at the moment).

“Acknowledge”: acknowledges one selected entry. The display will refresh automatically and the time of the acknowledge is shown.

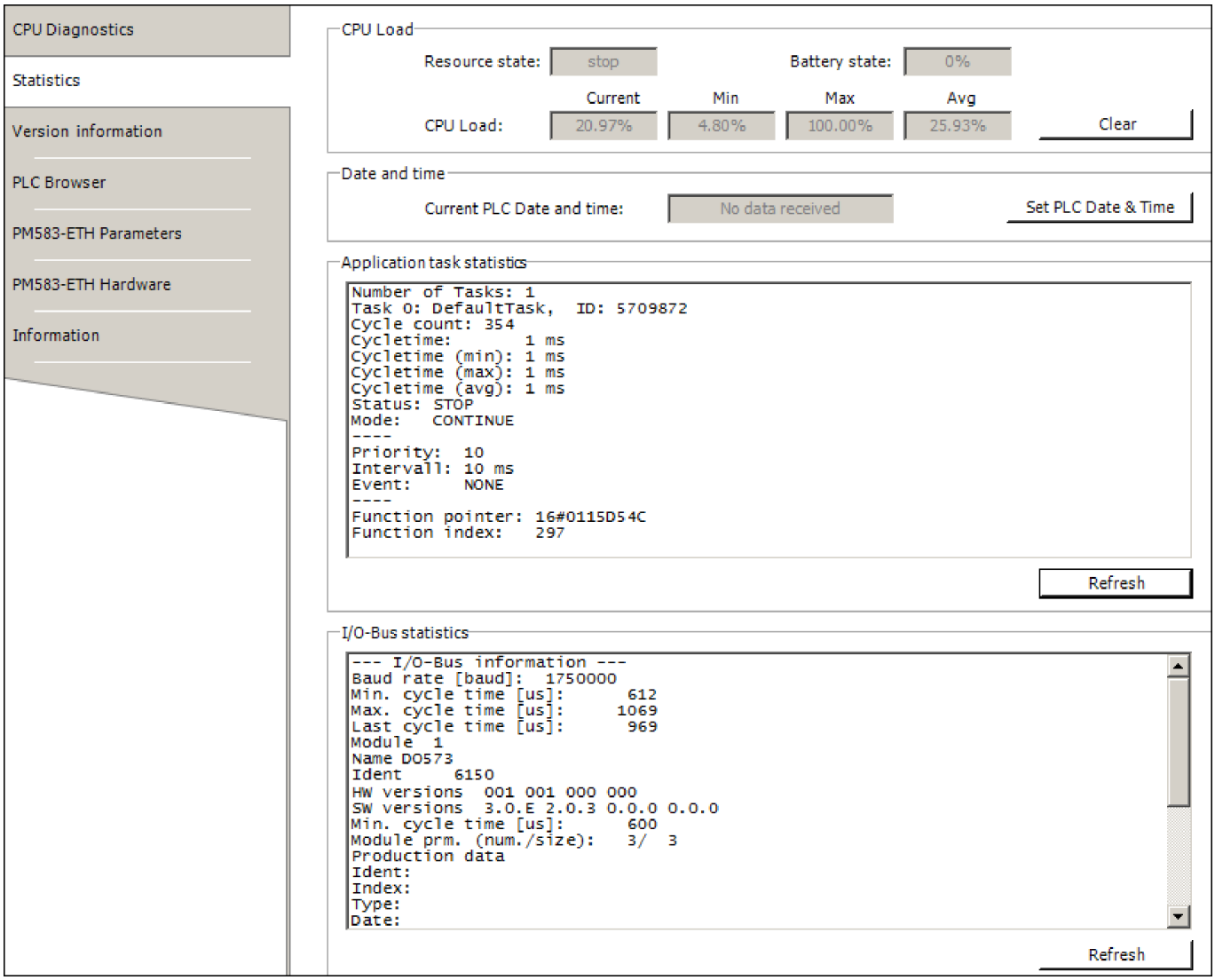
“Clear All Errors”: clears the error buffer on the PLC.

As a result of the Clear Errors command the PLC could go into RUN after next run command.

Description of the columns:

- *Index:* Sequence of the errors entered in the diagnosis buffer, with icon reflecting the states active/inactive/acknowledged/not acknowledged.
- *State:* Active/inactive: When inactive the reason for the entry is gone "“ with timestamp ("Time dis.").
- *Acknowledged:* If yes, the timestamp in *“Time ack.”* column is also displayed.
- *Class:* Severity classes E1-E4 (E1 is most severe).
- *Description:* A description for this error code is displayed (multilingual). The last columns Comp, Dev till Error number are needed to decode this message.
- *Online Text:* Error or warning message delivered from the firmware (non multilingual).
- *Mostly:* Indicates the Communication Module position (Ext.1-4) or I/O Bus Module where a problem exists.

1.7.2.5.2 Statistics



The *"Statistics"* tab shows the following information:

- *CPU Load:* The actual resource run state and the battery load state are shown. The CPU load values "Min", "Max" and "Avg." can be cleared with the *"Clear"* button.
- *Date and time:* The actual date and time of the PLC is shown. It can be set or synchronized with the date/time of the PC via *"Set PLC Date & Time"* button.
- *Application task statistics:* Information on the number of application tasks.
- *Local I/O statistics:* Information on the locally connected I/O Modules.

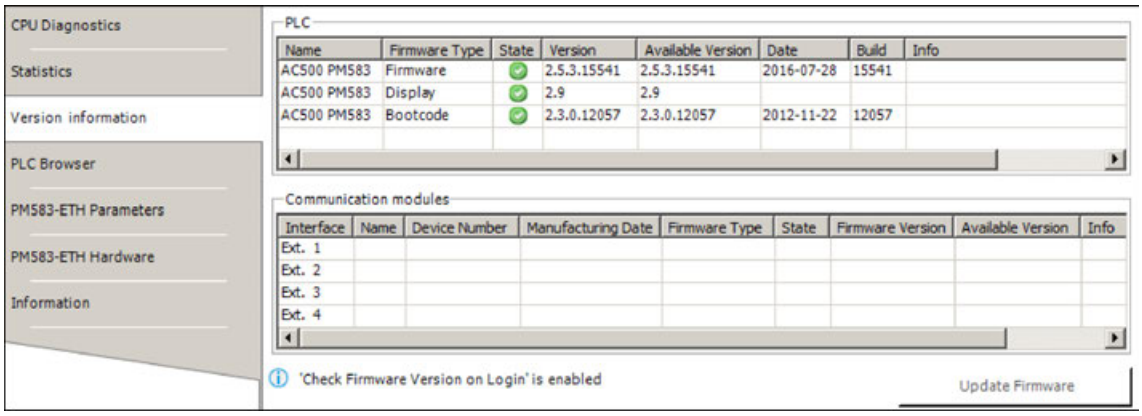
> *This complete view is refreshed cyclically (refresh rate is depending on the underlying communication protocol).*

1.7.2.5.3 Version information

Information on the firmware versions of the processor modules or communication modules, is provided on the *"Version information"* tab.

Remarks:

- The *"Version information"* tab displays the version identified on the device and the version provided with Automation Builder.
- The firmware on the devices must match to the Automation Builder version. Upgrade or downgrade to version supplied with Automation Builder is recommended (especially for CPUs) to ensure correct functionality.
- The firmware type can be changed to the type required by the hardware configuration for devices that support changing the firmware type. E.g., the onboard field bus communication modules of PM595 that may be used as PROFINET, Ethernet or EtherCAT communication module.



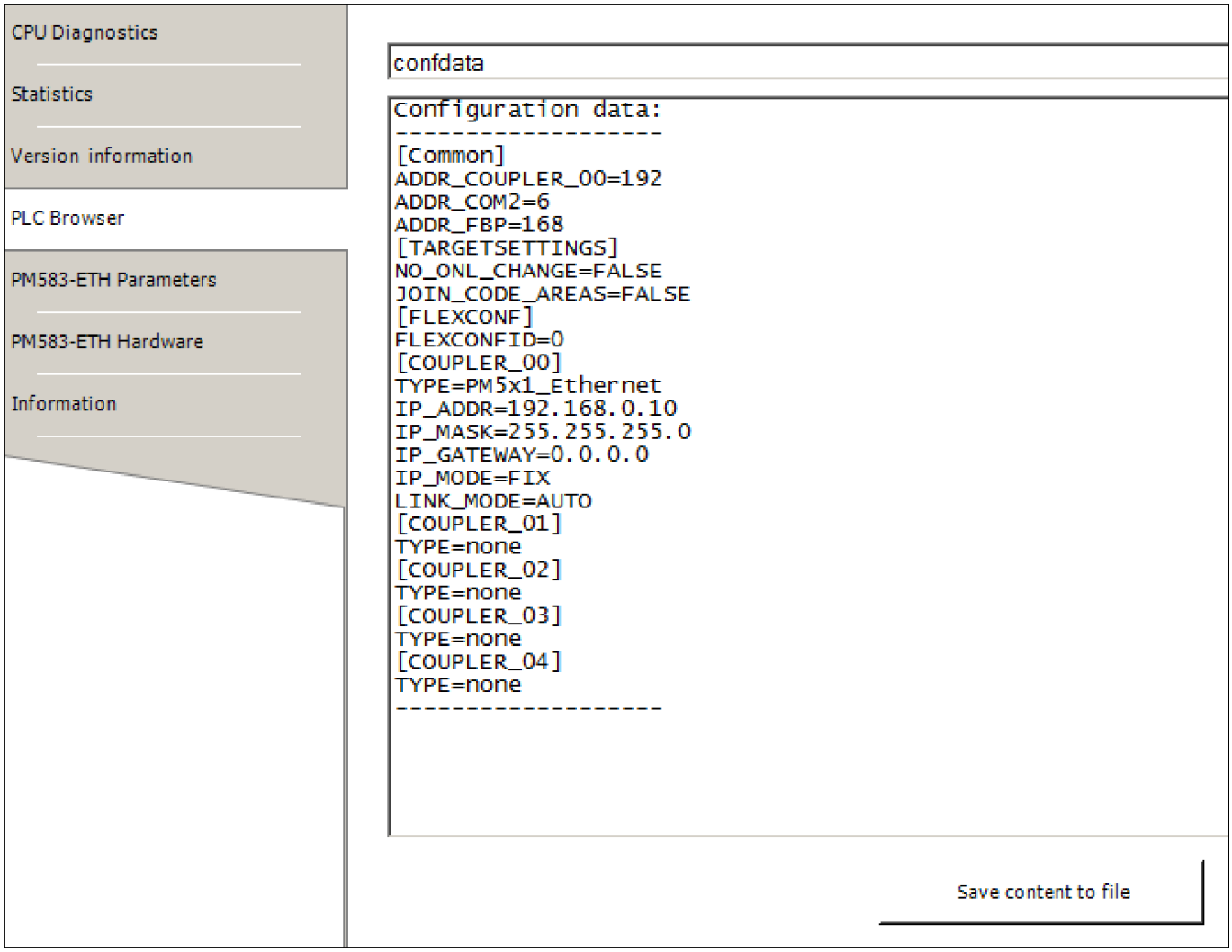
State icons

	Firmware version on device matches version supplied with Automation Builder.
	Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.
	Only for communication modules if CPU firmware must be updated first. This happens when CPU firmware has version below 2.5.0.0. Firmware version (or type) on device is different from version supplied with Automation Builder. Upgrade/downgrade to version supplied with Automation Builder is recommended.

	Identified device is different from configured device, thus no firmware update is possible. Happens only for Communication Modules.
No icon	Firmware of device is not updateable or no newer firmware than the initial version is available.

The [Update Firmware] button to download the new firmware is only enabled if there is updateable firmware.

1.7.2.5.4 PLC browser



In the *"PLC Browser"* tab all supported PLC browser commands can be entered.

Via *"Save content to file"* the content of the output can be saved to a text file.

This view has the same functionality as the PLC browser in CODESYS V2.3.

See *Chapter 1.7.2.5.5 "AC500-specific PLC browser commands" on page 6382*.

1.7.2.5.5 AC500-specific PLC browser commands

Automation Builder provides IEC 61131 standard commands as well as AC500-specific commands.

Online help is available for all commands. The help information is displayed language-dependent by entering command `"?"` when operating in online mode. The command `"?"` lists all available firmware commands.

The commands listed in online mode can differ from the commands shown when pressing the button *[...]* as Automation Builder version and firmware version can differ.

Depending on the device, the PLC browser provides the following commands:

Command	Description	Implementation
?	Displays all implemented commands	Standard
mem	Memory dump of an area Usage: mem [from-addr] [to-addr]	Standard
memc	Memory dump relative to code area	Standard
memd	Memory dump relative to data area	Standard
reflect	Reflect current command (for test purposes)	Standard
dpt	Displays the data pointer table	Standard
ppt	Displays the block pointer table	Standard
pid	Displays the project ID	Standard

pinf	Displays project information in the format:
	`pinf`
	`Address of Structure: 16#0013CF74`
	`Date: 4213949F`
	`Project Name: MODBUS_Test_BB.pro`
	`Project Title: Test MODBUS`
	`Project Version: V1.0`
	`Project Author: Test User`
	`Project Description:`
	`Test of serial interfaces`
	`End of Project-info.`
		Standard

Command	Description	Implementation

tsk	Displays the IEC task list with task information in the format:
	`tsk`
	`Number of Tasks: 1`
	`Task 0: Main program, ID: 1519472`
	`Cycle count: 45402`
	`Cycle time: 1 ms`
	`Cycle time (min): 1 ms`
	`Cycle time (max): 1 ms`
	`Cycle time (avg): 1 ms`
	`Status: RUN`
	`Mode: CONTINUE`
	`----`
	`Priority: 10`
	`Interval: 5 ms`
	`Event: NONE`
	`----`
	`Function pointer: 16#00601584`
	`Function index: 131`
		Standard

tskclear	Clears IEC Task information (cycle count & overall maximum and minimum cycle time)	Specific as of V2.0
startprg	Starts the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
stopprg	Stops the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
resetprg	Resets the user program	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
resetprgcold	Resets the user program (cold)	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)

Command	Description	Implementation
resetprgorg	Resets the user program (origin)	Standard (with CPU firmware below V2.3.0) No (with CPU firmware V2.3.0 and higher)
reload	Reloads the boot project from user flash memory	Standard (not supported with CPU firmware V2.2 or higher)

getprgprop	Displays program properties in the format:
	`getprgprop`
	`Name: MODBUS_FBP_Test_BB.pro`
	`Title: Test MODBUS`
	`Version: V1.0`
	`Author: Test User`
	`Date: 4213949F`
		Standard

getprgstat	Displays the program status in the format:
	`getprgstat`
	`Status: Run`
	`Last error: Id 00000000 TimeStamp 000055F3 Parameter 00000000 Text`
	`Flags:`
		Standard

filecopy	File command copy	No
filerename	File command rename	No
filedelete	File command delete	No
filedir	File command dir	No
saveretain	In V1.0 and V1.1: Saves the RETAIN variables to the memory card. As of V1.2: Writes the RETAIN variables to RAM (same as retain save)	Specific
restoreretain	In V1.0 and V1.1: Restores the RETAIN variables from the memory card. As of V1.2: Restores the RETAIN variables from RAM (same as retain restore)	Specific

Command	Description	Implementation
setpwd	Sets the PLC password (required at logon!) Note: From CPU firmware V2.3.0 and higher, this command works as follows: ● If there is no password set, a password can be set with: setpwd <new_password> ● If a [new] password has been set, the old password must also be inserted.: setpwd <old_password> <new_password>	Standard (with CPU firmware to V2.3.0) Specific (with CPU firmware V2.3.0 and higher)
delpwd	Deletes the PLC password Note: From CPU firmware V2.3.0 and higher, this command works only if a password has been set. Also, you have to specify the old password to delete it, i. e. syntax is: delpwd <current_password>	Standard (with CPU firmware to V2.3.0) Specific (with CPU firmware V2.3.0 and higher)
plcload	Displays the PLC utilization (system + IEC + tasks + communication)	Standard

Command	Description	Implementation

rtsinfo	Displays the firmware information (version, driver) in the format:
	
	`rtsinfo`
	`rts version: 2.4.7.24`
	`OS version: SMX smxPPC 3.5.2`
	`uses IO driver interface`
	`rts api version: 2.408`
	`4 driver(s) loaded`
	`driver 1: AC500 CPU driver, device interface version: 2.403`
	`driver 2: AC500 I/O-BUS driver, device interface version: 2.403`
	`driver 3: AC500 COM driver, device interface version: 2.403`
	`driver 4: AC500 Coupler driver, device interface version: 2.403`
	`AC500 PM___(DISP) : V2.1`
	`AC500 PM___(BOOT) : V2.0.5, 2017-10-26 (Build: 9603, 13:55:09, Rel)`
	`AC500 PM___(FW) : V2.0.4, 2017-10-12 (Build: 9530, 14:30:50, Rel)`
		Specific

traceschedon	Enables task tracing	No
traceschedoff	Disables task tracing	No
traceschedstore	Stores task trace to RAM	No
fdir <path>	Show content of a drive or directory <path> (e.g. `fdir userdisk`, `fdir sdcard/userdata`)	Specific as of V2.1
fread <path>	Dump a file's content	Specific as of V2.1
fmove <path>	Move a file to a directory	Specific as of V2.1
mkdir <path>	Create a directory	Specific as of V2.1
deldir <path>	Delete an empty directory	Specific as of V2.1
rndir <old path> <new path>	Rename a directory	Specific as of V2.1

Command	Description	Implementation
ipaddr	Sets the IP address of the CPU	No
basetick	Sets the basetick to µs	No
diagreset	Resets the diagnosis system	Specific
diagack all	Acknowledges all errors	Specific
diagack x	Acknowledges all errors of the class X (with X= 1...4)	Specific

diagshow all	Shows all errors in the format:
	`diagshow all`
	`--- All errors ---`
	`State Class Comp Dev Mod Ch Err`
	`0152502216`
	`active and acknowledged 4 9 22 31 31 8`
	`1970-01-01 00:00:08 occurred`
	`disappeared`
	`1970-01-01 00:00:15 ack.`
	`0152369165`
	`active not acknowledged 49 2031013`
	`1970-01-01 01:19:12 occurred`
	`- disappeared`
	`- ack.`
	`--- end ---`
		Specific

time	Displays and sets the time of the real-time clock. If no battery is inserted in the PLC and the control voltage is switched on, the PLC clock is set to "01. January 1970, 00:00".	Specific
date	Displays and sets the date of the real-time clock. If no battery is inserted in the PLC and the control voltage is switched on, the PLC clock is set to "01. January 1970, 00:00".	Specific
batt	Polls the battery status	Specific

Command	Description	Implementation
sdappl	Saves the boot project to the memory card	Specific
sdclone	Copys the user program and the user data to the memory card.	For AC500-S only!
sdfunc	Displays and changes the memory card function	Specific
sdsys	Save firmware to memory card	Specific
sdboot	Updates the bootcode from the memory card	Specific
sddisplay	Updates the MMI firmware from the memory card	Specific
sdfirm	Updates the firmware from the memory card	Specific
sdcoupler x	Updates the firmware of Communication Module x from the memory card	Specific
cpuload	Displays the CPU load (current, min., max., average)	Specific
delappl	Deletes the user program in the user flash memory	Specific
retain	Saving and restoring the RETAIN variables: retain clear -> Clears all RETAIN variables retain save -> Saves the RETAIN variables to the RAM disk retain restore -> Restores the RETAIN variables from the RAM disk retain export -> Exports the RETAIN variables from the RAM disk to the memory card retain import -> Imports the RETAIN variables from the memory card to the RAM disk	Specific as of V1.2

Command	Description	Implementation
persistent	Saving and restoring the persistent area %R area: persistent clear -> Clears the %R area persistent save -> Saves the buffered %R area to the RAM disk persistent restore -> Restores the buffered %R area from the RAM disk persistent export -> Exports the buffered %R area from the RAM disk to the memory card persistent import -> Imports the buffered %R area from the memory card to the RAM disk	Specific as of V1.2
cfginfo	Print expected and active configuration version. This is for internal use.	Specific
hashappl	Hash the user program	Specific
io-bus stat	Displays the I/O bus statistic	Specific
io-bus desc	Displays the I/O bus configuration	Specific
com protocols	Displays the protocols available for the serial interfaces	Specific
com settings	Displays the serial interface settings	Specific
coupler desc	Displays information on the communication module interfaces (type, firmware, serial number, date)	Specific
coupler settings	Displays the current communication module settings, for example, IP address and socket assignment	Specific as of V1.2
ping	Ping a address, usage: ping <ipaddr> <couplerid> <ms>	Specific as of V2.1.3
reboot	Reboots the PLC (IEC 61131 performs a logout when restarting or logout possible up to 3 seconds after command input) (This command is not available for CM574-RS as of firmware revision V2.1.x)	Specific

Command	Description	Implementation
diskcfg	Access to drive maintenance Command Syntax: `diskcfg [option] [drivename]` Command Options: ● `unlock` : unlock a drive for writing MBR or formating ● `lock` : lock drive again ● `writembr` : write clean MBR (unlock required) ● `format` : write clean file system (unlock required) ● `settings` : show drive configuration details ● `desc` : show drive overview ● `help` : show this help ● `[none]` : no option shows this help Available Drives (not all commands are supported on all drives): ● Flash memory ● Memory card ● RAM disk ● Userdisk ● Flash disk ● SRAM disk	Specific as of V2.1

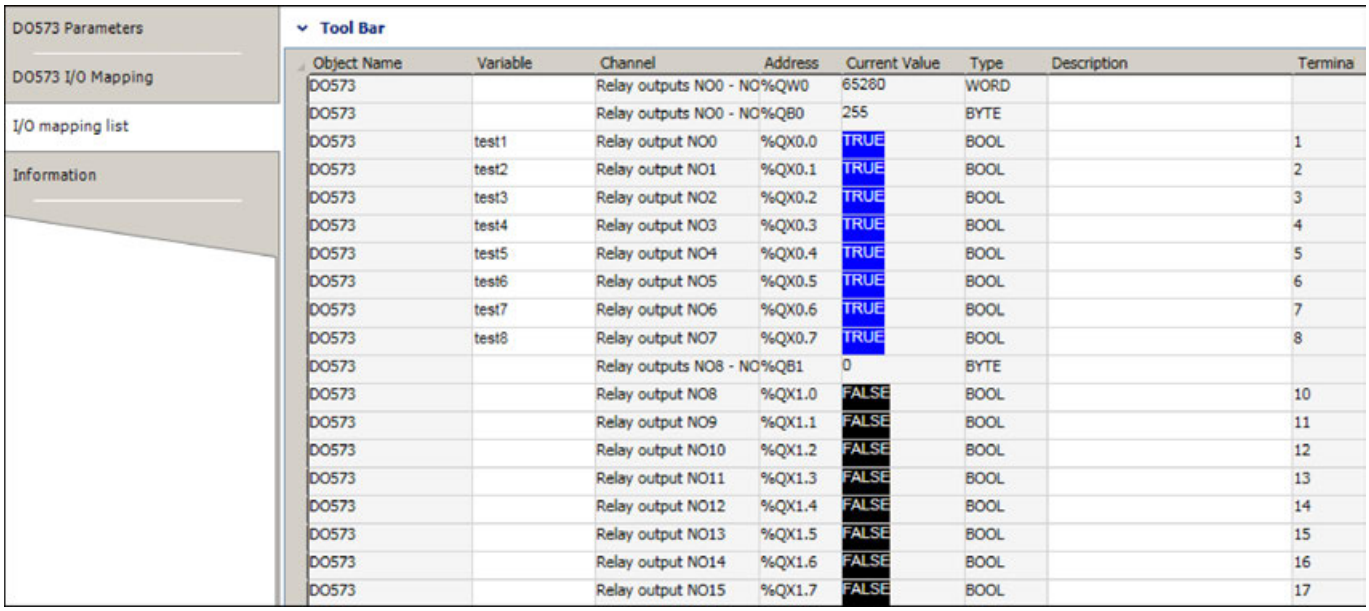
Command	Description	Implementation

proddata	Shows Production Data of PLC
	`proddata`
	`Production data`
	`---------------`
	`Ident : 1SAP123456R0001`
	`Index : B1`
	`Type : PM___`
	`Date : 0447`
	`BA-Inst : 1S120`
	`Factory : 05`
	`Year : 17`
	Serial No. : 00007134
	MAC-Addr : "
		Specific as of V2.0

confdata	Shows Configuration Data of PLC It is possible to save and load PLC specific configuration of any kind with the function blocks (from SysInt_AC500_V10.lib): *Chapter 1.5.4.19.2.3 "CPU_CONFIG_READ" on page 1508* *Chapter 1.5.4.19.2.4 "CPU_CONFIG_WRITE" on page 1511* The AC500 firmware also uses this INI file for settings like IP addresses.	Specific

1.7.2.6 Live values in views with I/O components

"I/O mapping list" tab: In online mode, all Automation Builder views, which contain I/O component mapping tables, show animated live values which are updated every second.



Object Name	Variable	Channel	Address	Current Value	Type	Description	Termina
DO573		Relay outputs NO0 - NO	%QW0	65260	WORD		
DO573		Relay outputs NO0 - NO	%QB0	255	BYTE		
DO573	test1	Relay output NO0	%QX0.0	TRUE	BOOL		1
DO573	test2	Relay output NO1	%QX0.1	TRUE	BOOL		2
DO573	test3	Relay output NO2	%QX0.2	TRUE	BOOL		3
DO573	test4	Relay output NO3	%QX0.3	TRUE	BOOL		4
DO573	test5	Relay output NO4	%QX0.4	TRUE	BOOL		5
DO573	test6	Relay output NO5	%QX0.5	TRUE	BOOL		6
DO573	test7	Relay output NO6	%QX0.6	TRUE	BOOL		7
DO573	test8	Relay output NO7	%QX0.7	TRUE	BOOL		8
DO573		Relay outputs NO8 - NO	%QB1	0	BYTE		
DO573		Relay output NO8	%QX1.0	FALSE	BOOL		10
DO573		Relay output NO9	%QX1.1	FALSE	BOOL		11
DO573		Relay output NO10	%QX1.2	FALSE	BOOL		12
DO573		Relay output NO11	%QX1.3	FALSE	BOOL		13
DO573		Relay output NO12	%QX1.4	FALSE	BOOL		14
DO573		Relay output NO13	%QX1.5	FALSE	BOOL		15
DO573		Relay output NO14	%QX1.6	FALSE	BOOL		16
DO573		Relay output NO15	%QX1.7	FALSE	BOOL		17

1.7.2.7 Communication module and fieldbus diagnosis

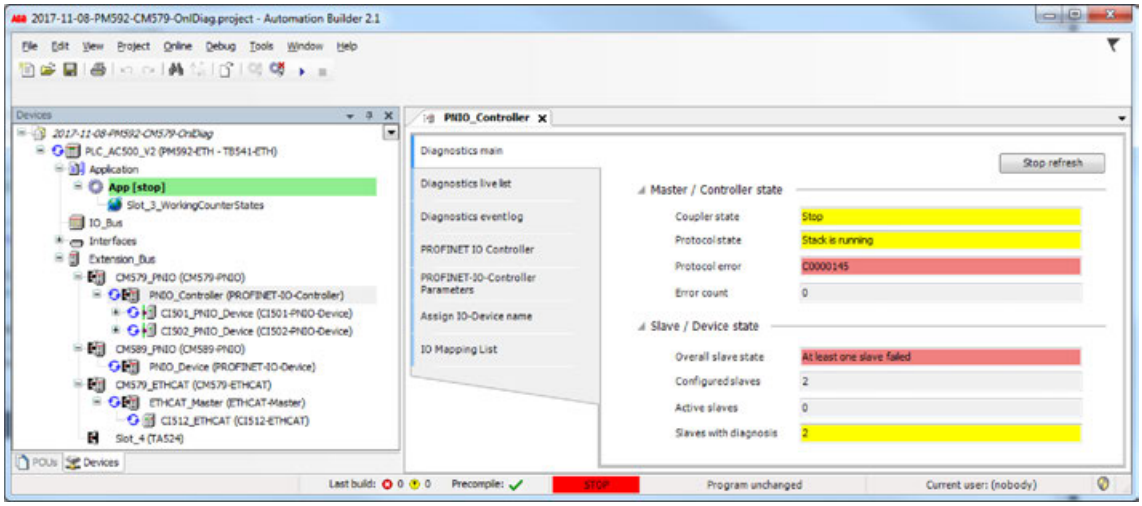
1.7.2.7.1 Fieldbus commissioning

Common online diagnosis views for all netX-based communication modules (e. g. CM579-ETHCAT, CM579-PNIO) can be accessed whenever the related PLC is in online mode ↳ *Chapter 1.7.2.2 “Entering/leaving the online mode” on page 6375*.

Master/controller modules

Master/controller modules like CM579-ETHCAT or CM579-PNIO, provide the following diagnosis views:

- *“Diagnostics main”*: provides diagnosis messages which are common for all protocols (e.g., protocol state and error)
- *“Diagnostics live list”*: provides a list of connected slaves/devices and their state ↳ *Chapter 1.7.2.7.1.1.1 “ PROFINET scan and comparison view” on page 6393*
- *“Diagnostics eventlog”*: provides diagnosis messages from the master/controller and its connected slaves/devices



PROFINET scan and comparison view

PNIO_Controller

1. After going online, double-click on *"PNIO_Controller (PROFINET-IO-Controller)"* in the device tree.

   ⇨ The editor *"PNIO_Controller"* is displayed.

2. Select tab *"Diagnostics live list"* and click *[Scan]* to find all hardware devices that exist.

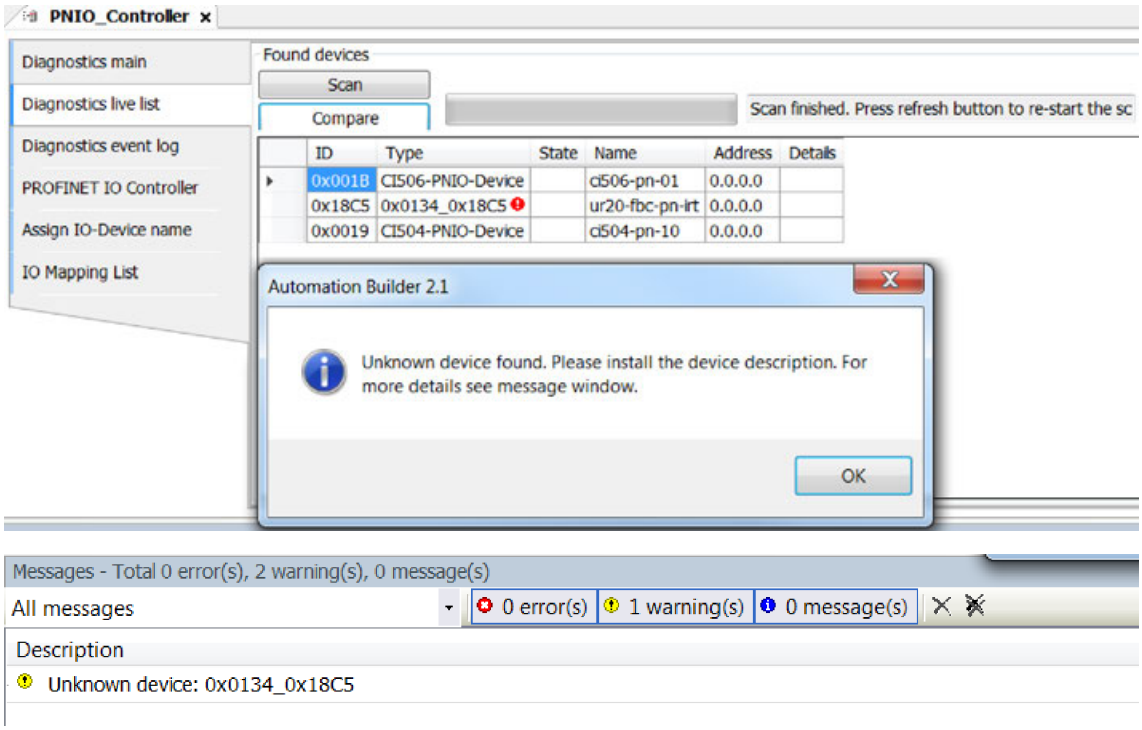
   ⇨ The found devices are listed in a table.

3. Click *[Compare]* to compare the found hardware I/O devices with the current project configuration.

Unknown hardware

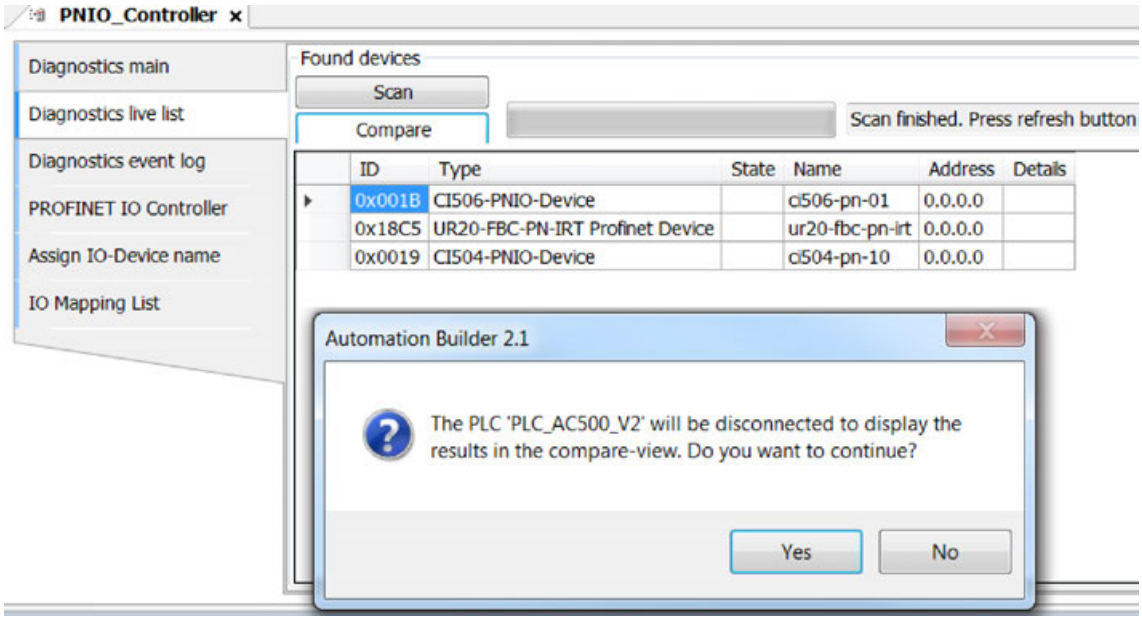
If any I/O hardware device is unknown:

- The devices will be marked with a red exclamation mark.
- A message box will be appear for each unknown device.
- Automation Builder generates a message with information about its vendor ID and device ID.



Comparison view

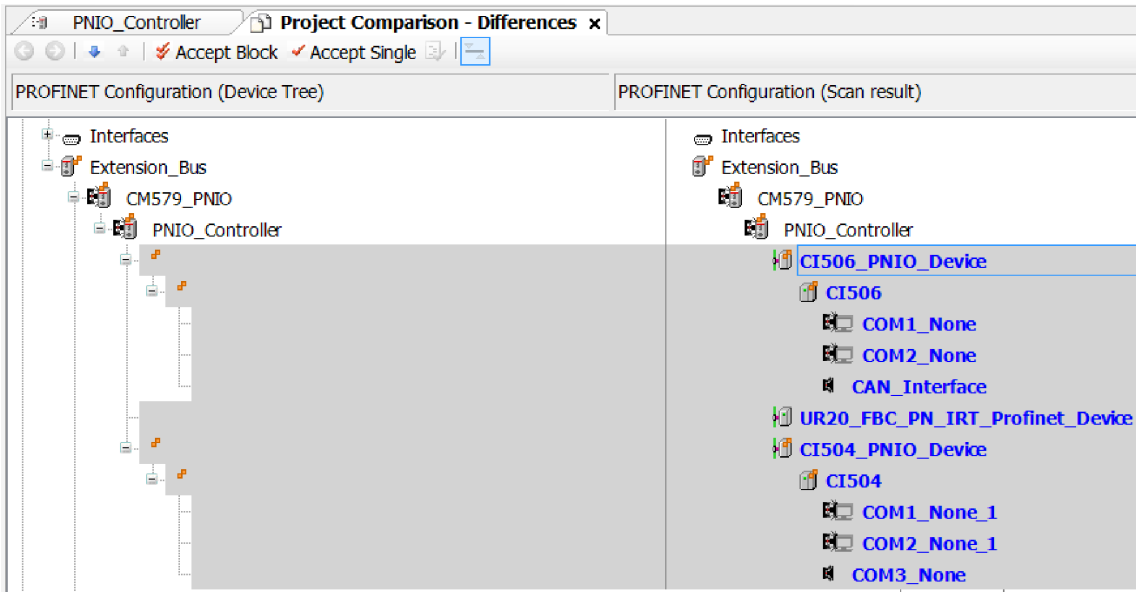
1. To display the comparison view, install the device description for the unknown device.
2. After installing the device description, click *[Scan]* and click *[Compare]*.



⇨ The message box informs you, that the application will go offline to display the comparison view.

3. Click *[Yes]*.

⇨ The *"Project Comparison - Differences"* tab displays the difference between the PROFINET configuration in Automation Builder (left side) and the real hardware configuration (right side).

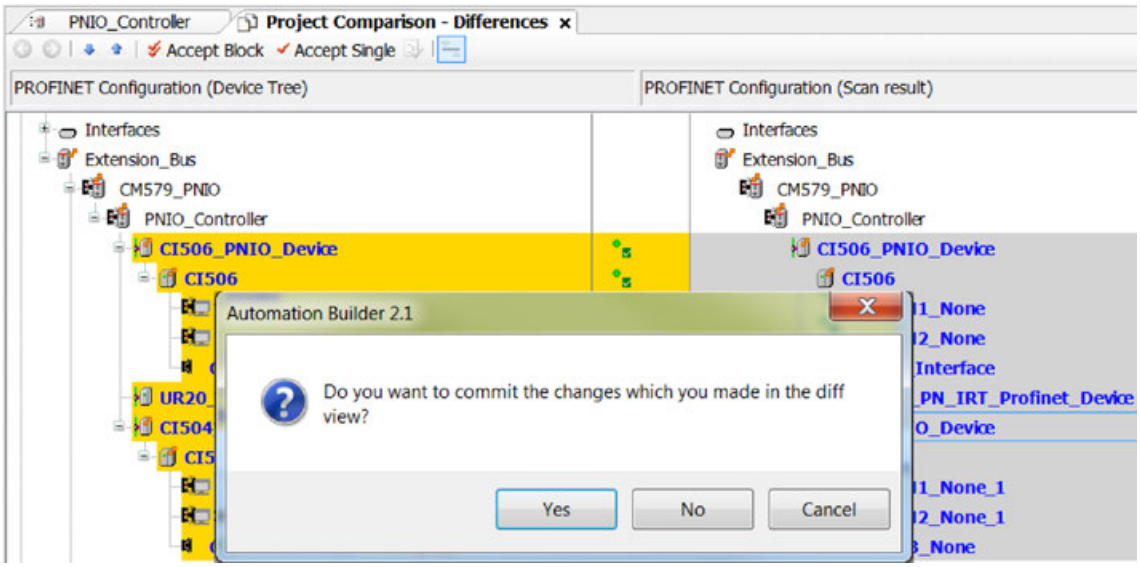


4. Click *[Accept Single]* to accept only a part of the differences or *[Accept Block]* to accept all differences.

⇨ After clicking on the Button *[Accept Single]* or *[Accept Block]* the found devices will be moved from the right side to the left side.

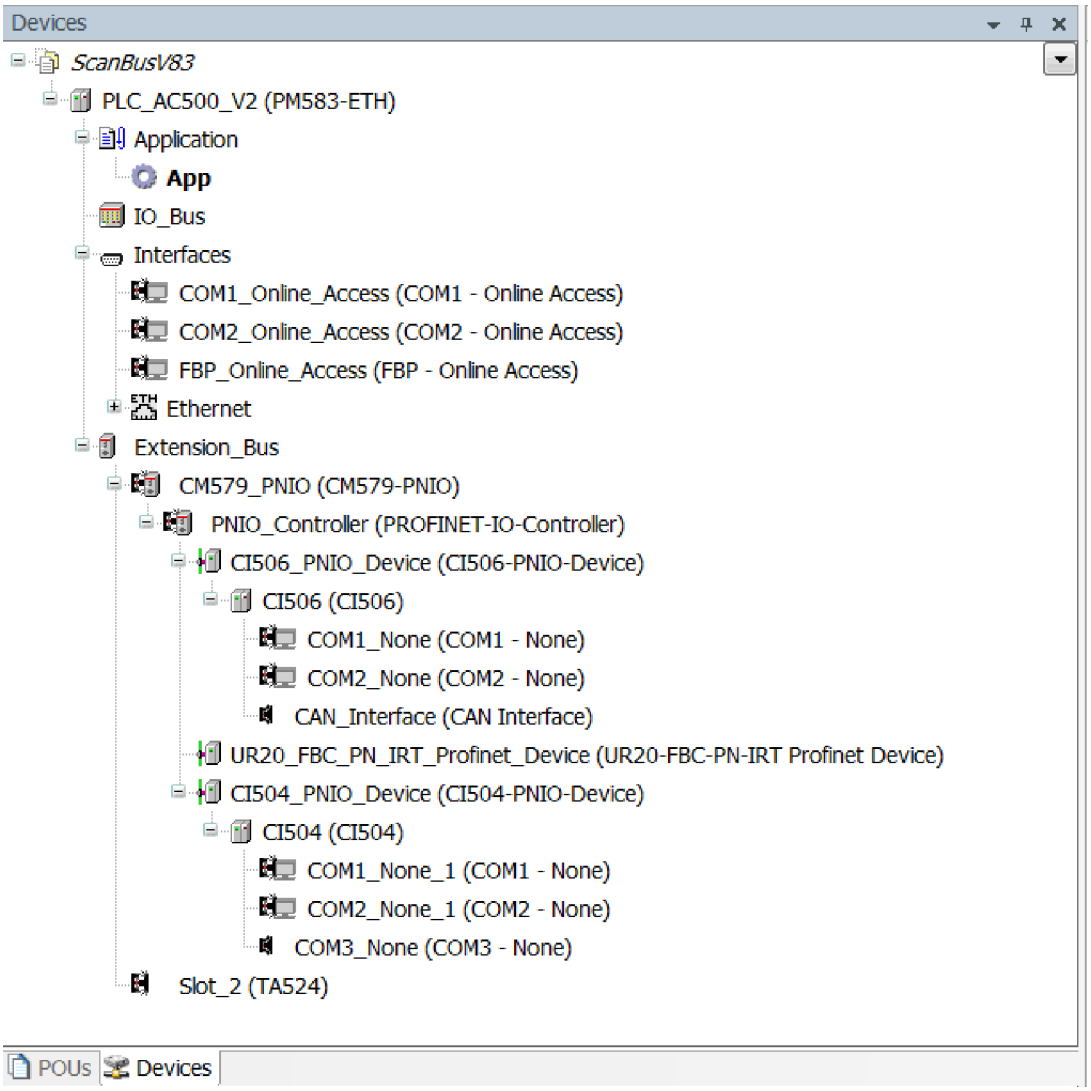
5. Close tab *“Project Comparison - Differences”*.

   ⇨ A message will be displayed to ask if you want to commit the new changes into project.



6. Click *[Yes]*.

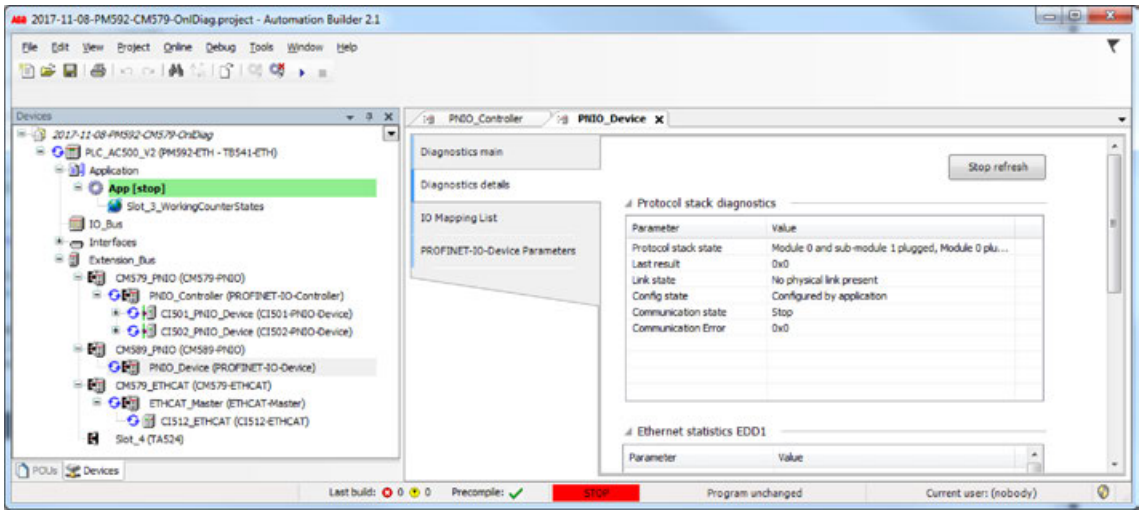
   ⇨ The changes will be saved and the devices will be added to the project.



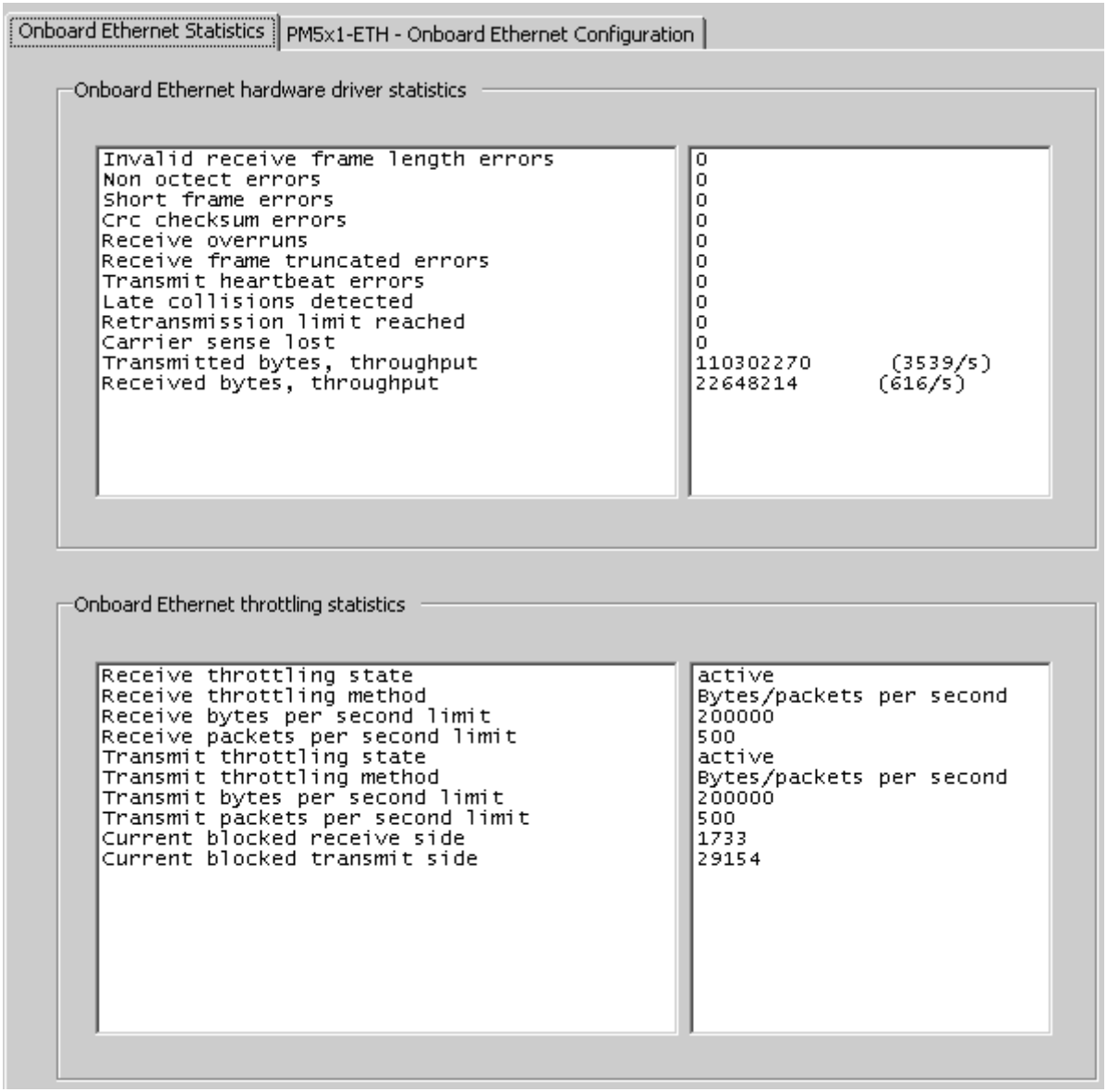
Slave/device communication modules

Diagnosis views for slave/device communication modules like CM589-PNIO:

- *"Diagnostics main"*: provides diagnosis messages which are common for all protocols
- *"Diagnostics details"*: provides protocol specific diagnosis messages



1.7.2.7.2 Onboard Ethernet statistic



This view is refreshed cyclically and shows in above part the common Ethernet errors like late collisions, invalid frame errors.

If these error counters are growing, this indicates problems with the Ethernet cabling or overload.

Transmitted and received bytes are shown as a sum and bytes per second.

The lower part shows information about the throttling status.

Throttling is a mechanism that could protect the PLC from network bursts, "brute force" or DoS (denial of service) attacks.

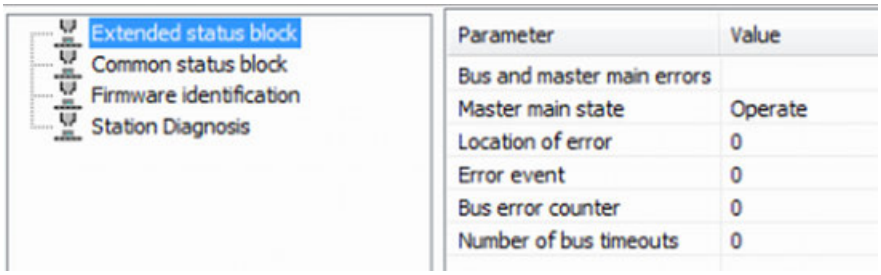
The receive/transmit parts of the network driver are handled separately.

If throttling is active (default) depending on the throttling method (which can be "bytes/second", "packets/second" or both) every cycle (depends on CPU type, about 10 ms) the firmware decides, if the limits (bytes per second limit or packets per second limit) are exceeded.

If this happens, the receiver or transmitter will ignore the data at least for 1 cycle and then calculates again.

1.7.2.7.3 CM592-DP PROFIBUS DP communication module statistic views

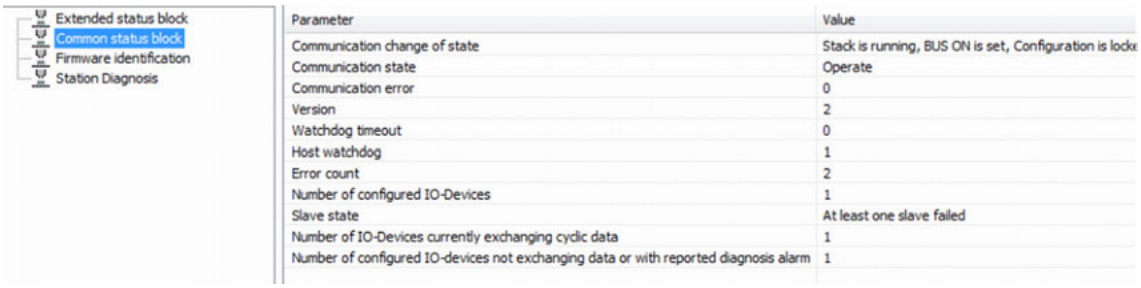
Extended status block view



Parameter	Values	Description
Bus and master main errors	MSK_PROFIBUS_APM_EXT_STA_CTRL_ERR	CONTROL-ERROR: This error is caused by incorrect parameterization.
	MSK_PROFIBUS_APM_EXT_STA_ACLR_ERR	AUTO-CLEAR-ERROR: The device stopped the communication to all slaves and reached the auto-clear end state
	MSK_PROFIBUS_APM_EXT_STA_NEXC_ERR	NON-EXCHANGE-ERROR: At least one slave has not reached the data exchange state and no process data are exchanged with it.
	MSK_PROFIBUS_APM_EXT_STA_FATL_ERR	FATAL-ERROR: Because of a severe bus error, no bus communication is possible any more
	MSK_PROFIBUS_APM_EXT_STA_NRDY	Host-NOT-READY-NOTIFICATION: Indicates, if the host program has set its state to operative or not. If the bit is set, the host program is not ready to communicate
	MSK_PROFIBUS_APM_EXT_STA_TOUT	TIMEOUT-ERROR: The device has detected an overstepped timeout supervision time because of rejected PROFIBUS telegrams. It's an indication for bus short circuits while the master interrupts the communication. The number of detected timeouts are fixed in the Time_out_cnt variable. The bit will be set when the first timeout was detected and will not be deleted any more.

Parameter	Values	Description
Master main state	Offline	In OFFLINE state, there is no communication (data transfer) permitted at all. This is the state after initialization. This means, the master is waiting for a signal to start and does not participate in the token ring of the PROFIBUS access control mechanism
	Stop	In STOP state, there is no data transfer permitted between master and slaves. Data transfer to other masters in multi-master system is allowed, however. The bus parameter set has been loaded successfully in order to get into STOP state.
	Clear	In CLEAR state, the master is able to read the input data from the DP slaves. The master forces the outputs to the slaves to be in a safe state (i.e. they contain only the value 0). For instance, incorrect data transfer of a slave can cause the PROFIBUS DP master to fall back from OPERATE state to CLEAR state. Parameterization and configuration checks are possible in this state.
	Operate	In OPERATE state, unrestricted data transfer is possible. This data transfer is cyclic, i.e. periodically, the input values are read from the slaves and the output data are written to the slaves.
Location of error	0..125	Address of the slave with error
Error event		Not used currently
Bus error counter		Counter for the bus error events
Number of bus timeouts		Counter for bus timeouts

Common status block view



Parameter	Description
Communication change of state	The communication change of state register contains information about the current operating status of the communication channel and its firmware ↳ *Table 764 "Communication change of state flags" on page 6401*.
Communication state	The communication state field contains information regarding the current network status of the communication channel. Depending on the implementation all or a subset of the following definitions is supported: ● UNKNOWN ● NOT_CONFIGURED ● STOP ● IDLE ● OPERATE
Communication error	This field holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to 0 (= RCX_SYS_SUCCESS) again. Depending on the implementation protocol stacks use all or a subset of the following error codes: ● SUCCESS 0x00000000 ● Runtime Failures: – WATCHDOG TIMEOUT 0xC000000C ● Initialization Failures: – INITIALIZATION FAULT 0xC0000100 – DATABASE ACCESS FAILED 0xC0000101 ● Configuration Failures – NOT CONFIGURED 0xC0000119 – CONFIGURATION FAULT 0xC0000120 – INCONSISTENT DATA SET 0xC0000121 – DATA SET MISMATCH 0xC0000122 – INSUFFICIENT LICENSE 0xC0000123 – PARAMETER ERROR 0xC0000124 – INVALID NETWORK ADDRESS 0xC0000125 – NO SECURITY MEMORY 0xC0000126 ● Network Failures – NETWORK FAULT 0xC0000140 – CONNECTION CLOSED 0xC0000141 – CONNECTION TIMED OUT 0xC0000142 – LONELY NETWORK 0xC0000143 – DUPLICATE NODE 0xC0000144 – CABLE DISCONNECT 0xC0000145
Version	The version field holds version of this structure. It starts with 1. 0 is not defined. Version should be 1.
Host watchdog	This field holds the configured watchdog timeout value in milliseconds. The application may set its watchdog trigger interval accordingly. If the application fails to copy the value from the host watchdog location to the device watchdog location, the protocol stack will interrupt all network connections immediately regardless of their current state.

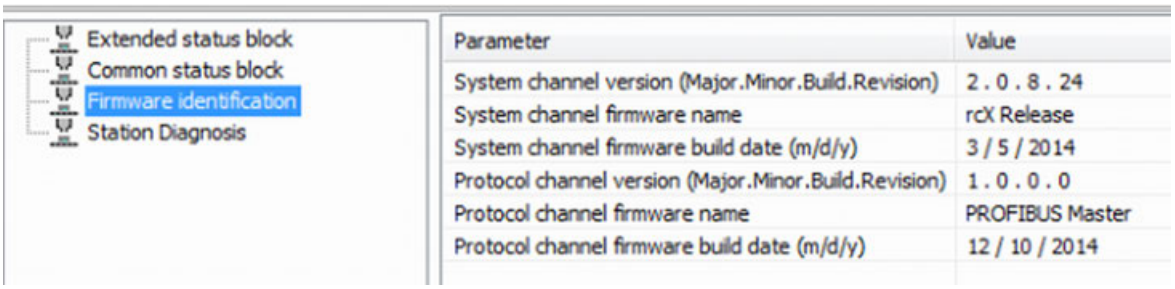
Parameter	Description
Error count	This field holds the total number of errors detected since power-up, respectively after reset. The protocol stack counts all sorts of errors in this field no matter if they were network related or caused internally.
Number of configured I/O devices	The firmware maintains a list of slaves to which the master has to open a connection. This list is derived from the configuration database. This field holds the number of configured slaves.
Slave state	The slave state field indicates whether the master is in cyclic data exchange to all configured slaves. In case there is at least 1 slave missing or if the slave has a diagnostic request pending, the status is set to FAILED. For protocols that support non-cyclic communication only, the slave state is set to OK as soon as a valid configuration is found. (UNDEFINED, NO_FAULT, FAILED)
Number of I/O devices currently exchanging cyclic data	The firmware maintains a list of slaves to which the master has successfully opened a connection. Ideally, the number of active slaves is equal to the number of configured slaves. For certain fieldbus systems it could be possible that the slave is shown as activated, but still has a problem in terms of a diagnostic issue. This field holds the number of active slaves.
Number of configured I/O devices not exchanging data or with reported diagnosis alarm	If a slave encounters a problem, it can provide an indication of the new situation to the master in certain fieldbus systems. As long as those indications are pending and not serviced, the field holds a value unequal 0. If no more diagnosis information is pending, the field is set to 0.

Table 764: Communication change of state flags

Value	Description
Ready	0 - ... 1 - The *Ready* flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the *Running* flag is set, too.
Running	0 - ... 1 - The *Running* flag is set if the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the *Ready* flag and the *Running* flag are set.
Bus On	0 - ... 1 - The *Bus On* flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network. If cleared, the permission was denied and the protocol stack will not open network connections.

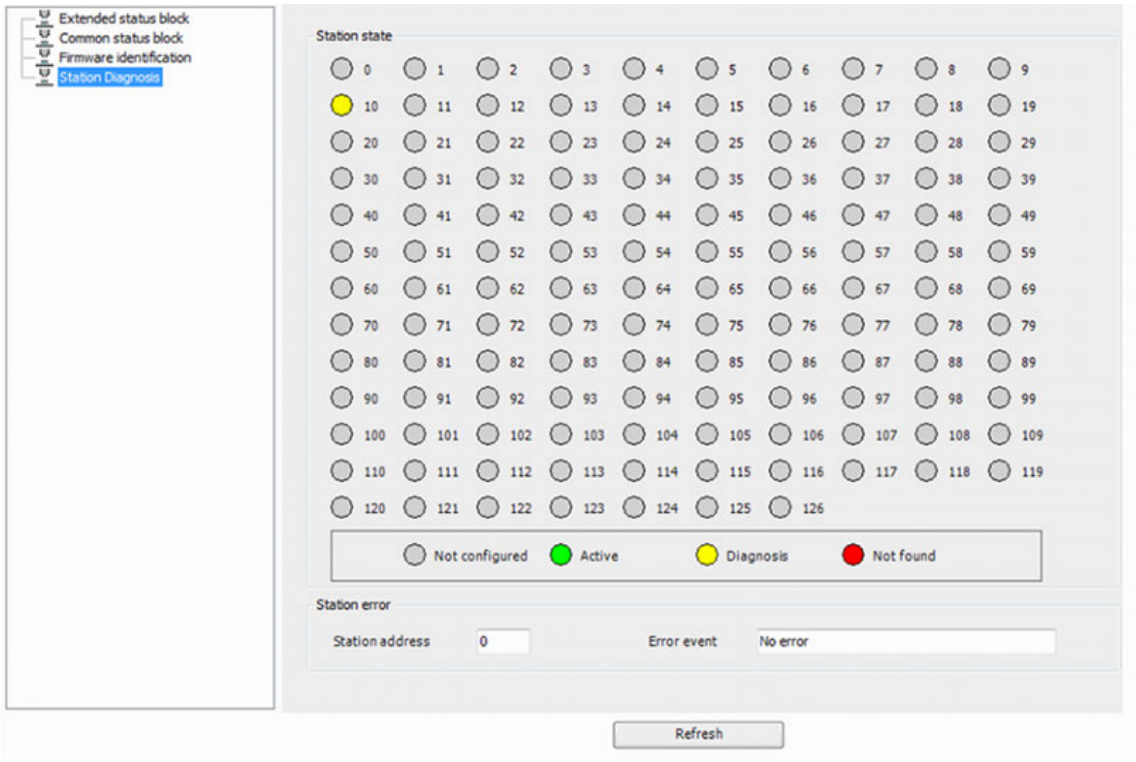
Value	Description
Configuration Locked	0 - ... 1 - The *Configuration Locked* flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the *Configuration Locked* flag in the control block.
Configuration New	0 - ... 1 - The *Configuration New* flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the Restart Required flag.
Restart Required	0 - ... 1 - The *Restart Required* flag is set if the channel firmware requests to be restarted. This flag is used together with the Restart Required Enable flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
Restart Required Enable	0 - ... 1 - The *Restart Required Enable* flag is used together with the Restart Required flag above. If set, this flag enables the execution of the Restart Required command in the netX firmware (for details on the Enable mechanism see section 2.3.2 of the netX DPM Interface Manual).

Firmware identification view



Parameter	Description
System channel version	Version of the RcX operating system
System channel firmware name	Name of the operating system
System channel firmware build date	Date of the operating system
Protocol channel version	Version of the communication module
Protocol channel name	Name of the communication module
Protocol channel date	Date of the communication module

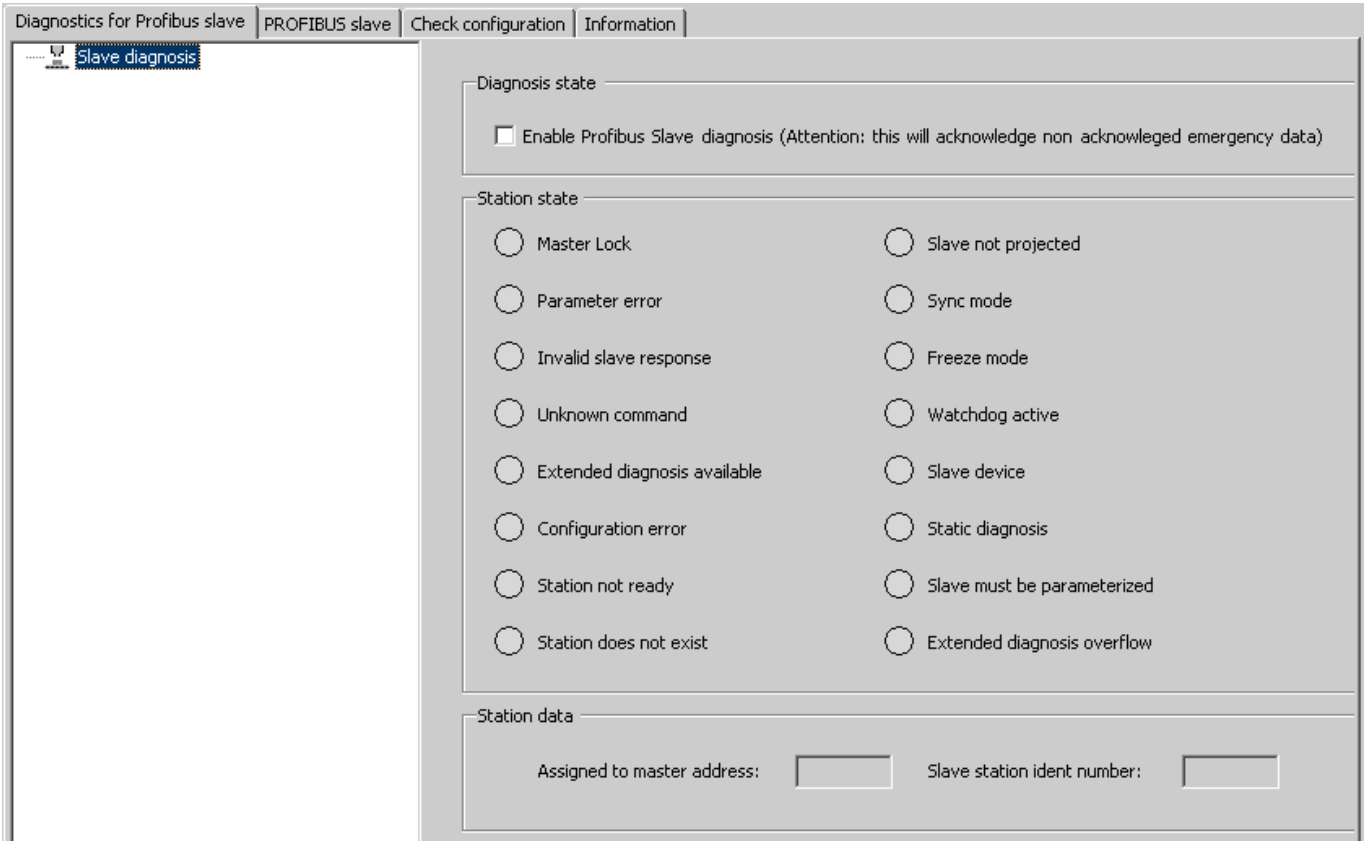
Station diagnosis view



For every configured slave the following data is displayed:

Value	Description
Not configured	Not configured
Active	All OK: slave is in data exchange
Diagnosis	Configured, active has diagnosis
Not found	Configured but not found

1.7.2.7.4 CM592-DP PROFIBUS DP slave view



The PROFIBUS Slave diagnosis must be enabled explicit by the user every time he opens this view (the enabled state will not be saved). This has the following reason:

Each time, when a diagnosis read request is sent to a slave which has actual diagnosis data, the diagnosis data is automatically acknowledged. Thus, an alarm handling function block running on PLC, will possibly miss this alarm. So enabling this view should only be done, if no function blocks are used to handle alarms/diagnosis data.

Table 765: Meaning of Station Status 1

Station Status 1	Activated by	Description and Remedy
Master Lock (Bit 7)	Master	Description: The slave has been parameterized by another master, and is locked for accesses by the selected master.
		Remedy: This is a safety mechanism of PROFIBUS DP. Firstly, you should decide which master should have access to the slave. After that, the slave has to be added into the configuration of that master, which should have access to the slave. Finally, the slave must be deleted in the configuration of the other master(s).

Station Status 1	Activated by	Description and Remedy
Parameter Error (Bit 6)	Slave	Description: This bit is set by the slave automatically, if the parameter, which the master has output, is wrong or incomplete. Each received partner telegram is checked completely by the slave. If the slave recognises an error, it will report an parameter error. Furthermore, the slave verifies its ident number with the ident number the master has sent.
		Remedy: Firstly, verify the ident number inside the device with the ident number inside the GSD file. They must me equal. If they are different, there is either a the wrong GSD file used or a wrong device has been connected to the bus. If both ident numbers are equal, check the parameter data.
Invalid Slave answer (Bit 5)	Master	Description: This bit is set by the master, if an invalid reply from the slave has been received. The physical connection to the slave has been established, but the logical answer could not be understood.
		Remedy: There can be an error on the physical cable line, e. g. inter-changed wires, missing bus termination or missing shield connection. Use a PROFIBUS DP slave, which is corre-sponding to the norm. This error can also happen, if a PROFIBUS-FMS slave is connected instead of a PROFIBUS DP slave to the DP master. Then the slave cannot understand the TP telegram and returns it to the sender. It will be recognized by the master as an invalid slave reply.
Function not sup-ported (Bit 4)	Slave	Description: This bit is set by the slave, if it should process a function it does not support. No version of slaves normally support Sync- and Freeze commands. This is mentioned inside the GSD file and is read by Automation Builder and sent as parameter telegram to the slave.
		Remedy: If this error occurs, the GSD file contains at least 1 function, which is not supported by the slave. Ask the device vendor for the correct GSD file which belongs to the slave.
Enhanced device diagnosis available (Bit 3)	Slave	Description: This bit is set by the slave, if enhanced diagnosis data have been read. These data are optional and are used by the slave to output vendor-specific diagnosis messages.
		Remedy: Activate the enhanced diagnosis to display the enhanced diagnosis data and read the manual of the device vendor for the meaning. If the GSD file contains information about the enhanced device diagnosis, the evaluation can also be threatened by the DTM.

Station Status 1	Activated by	Description and Remedy
Configuration error (Bit 2)	Slave	Description: During the PROFIBUS DP initialization, the slave compares its internal I/O configuration to the configuration of the master. If the slave recognises a difference, it will report a configuration error. This means the master has an other I/O constellation than the slave.
		Remedy: First of all, check whether the I/O modules detected by the slave and the connected I/O modules match. Make sure that the sequence of the modules also matches. At the beginning some slaves need virtual I/O modules or empty modules to achieve an even number of modules. This slave-specific behavior should be mentioned in the vendor manual as it cannot be read inside the GSD file. Pay attention to the configuration hints of the device vendor.
Station not ready (Bit 1)	Slave	Description: The PROFIBUS DP slave is not ready for data exchange yet.
		Remedy: The norm does not specify why and when a slave sets this bit; therefore, there may be several reasons. Mostly this error occurs combined with other errors. Verify the parameters and the configuration. The error *Station not ready* is often a consecution of a parameter or configuration error. Eventually, the power supply of the slave has just been turned on. Wait until the device has started.
Station is not existing (Bit 0)	Master	Description: This bit is set automatically by the master if the slave in the bus does not reply or is not reachable.
		Remedy: Check the PROFIBUS cable. Both signal lines must be connected correctly between all devices. Additionally, both cable ends must be equipped with terminators. Check the power supply connection of the slave. Check the slave station address and the configuration of the master. Verify, whether the slave supports the configured transmission rate. Some older slaves will only work up to 1.5 MBaud or must be set to a special PROFIBUS DP conform behavior.

Table 766: Meaning of Station Status 2

Station Status 2	Activated by	Description
Slave deactivated (Bit 7)	Master	This bit is set by the master, if the parameterization of the slave marks itself as inactive. Thus, it will be excluded from the cyclically data exchange.
Reserved (Bit 6)	-	-
Sync Mode (Bit 5)	Slave	This bit is set by the slave, if a Sync command has been received.

Station Status 2	Activated by	Description
Freeze Mode (Bit 4)	Slave	This bit is set by the slave, if a Freeze command has been received.
Watchdog active (Bit 3)	Slave	This bit is set by the slave, if time monitoring is activated for controlling the communication with the associated master.
Slave device (Bit 2)	Slave	This bit is always set by the slave.
Static diagnosis (Bit 1)	Slave	This bit is set by the slave to signal the master, that it has a general error and is out of order. Typically, the slave is not available for the exchange of user data. In this case, the master should request diagnosis data as long as the bit is set to 0 again. The event or point of time the bit is set, is not described in the norm and cannot be specified more precisely.
Re-parameterization requested (Bit 0)	Slave	This bit is set by the slave to signal the master, that it asks for a new parameterization. The bit stays set until the parameterization is processed. If this error occurs, firstly compare the ident number of the device with the ident number inside the GSD file. These numbers must match. Additionally, check the parameter data.

Table 767: Meaning of Station Status 3

Station Status 3	Activated by	Description
Overflow enhanced diagnosis (Bit 7)	Master Slave	This bit is set, if more enhanced diagnosis data are sent to the master than match into a diagnosis telegram. For example, the slave sets this bit, if more diagnosis data are pending than fit into its buffer.
Reserved (Bit 6)	-	-

The display of the extended diagnosis of a PROFIBUS slave is not supported.

Diagnosis	Description
Assigned to master address	At Assigned to master address, the address of the master, which has parameterized and configured the slave, is displayed. The value 255 shows that the slave is neither parameterized nor configured or the received parameterization and configuration information was rejected with errors.
Slave station ident number	At slave station ident number, the (real) ident number of the used slave is displayed. If the value 0000 is shown, the master has no connection to the PROFIBUS DP slave yet.
Update	Updates the displayed diagnosis conditions.

The ident number, deposited in the device, must match to the ident number in the GSD file. If they are different, either a wrong GSD file is used or the wrong device is connected to the PROFIBUS.

1.7.2.7.5 CM579-PNIO PROFINET IO controller views

IO controller views

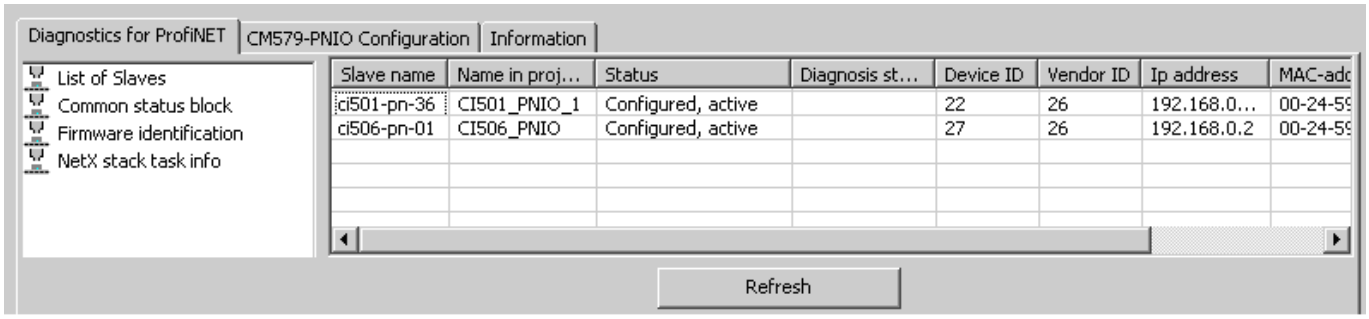
The views in PROFINET master and slave are not refreshed cyclically, because of the big communication load this would impose. For updating the data the Refresh button is available.

Chapter 1.7.2.7.5.2 “IO controller list of slaves view” on page 6408

Chapter 1.7.2.7.5.4 “IO controller common status block view” on page 6409

Chapter 1.7.2.7.5.3 “IO controller firmware and task info view” on page 6408

IO controller list of slaves view



This is the main diagnosis view for the PROFINET IO Controller: List of slaves table view.

For every configured slave the following data are displayed:

Slave name: This is the main identifier on the PROFINET. (This name can be set or changed using the Assign station name editor tab available at the CM579 master level)

Name in Project: Name of the node in the Project Tree where this Device is configured

Status:

- *configured, active* -> all OK, slave is in data exchange
- *configured, inactive* -> configured, but not found
- *configured, inactive, faulted* -> configured, not in data exchange but with errors
- *configured, active, has active alarm, not acknowledged*
- *configured, active, has active alarm, acknowledged*

Diagnosis state: special diagnosis error messages (independent of alarms)

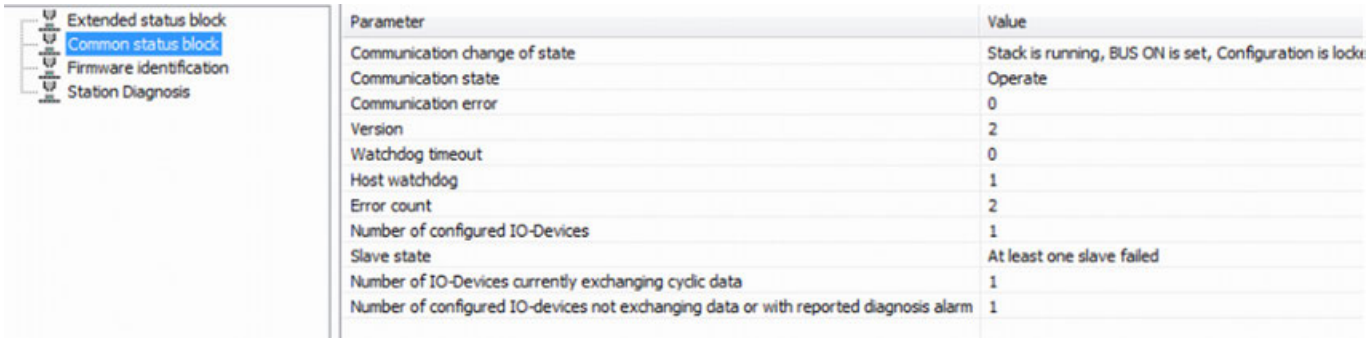
Device ID: type ID given by the Vendor

VendorID: identifies the Vendor, ID is unique

IO controller firmware and task info view

The last 2 sections provide information about the communication module's firmware versions, build dates and internal information about the different running tasks on the communication module side.

IO controller common status block view



Parameter	Description
Communication state of change	The communication change of state register contains information about the current operating status of the communication channel and its firmware. Possible state changes (see below table for details) are: ● *Restart Required Enable* ● *Restart Required* ● *Configuration New* ● *Configuration Locked* ● *Bus On* ● *Running* ● *Ready*
Communication state	The communication state field contains information regarding the current network status of the communication channel. Depending on the implementation, all or a subset of the definitions below is supported: UNKNOWN NOT_CONFIGURED STOP IDLE OPERATE

Parameter	Description
Communication error	This field holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to 0 (= RCX_SYS_SUCCESS) again. Not all of the error codes are supported in every implementation. Protocol stacks may use a subset of the error codes below: SUCCESS 0x00000000 **Runtime Failures:** WATCHDOG TIMEOUT 0xC000000C **Initialization Failures:** INITIALIZATION FAULT 0xC0000100 DATABASE ACCESS FAILED 0xC0000101 **Configuration Failures** NOT CONFIGURED 0xC0000119 CONFIGURATION FAULT 0xC0000120 INCONSISTENT DATA SET 0xC0000121 DATA SET MISMATCH 0xC0000122 INSUFFICIENT LICENSE 0xC0000123 PARAMETER ERROR 0xC0000124 INVALID NETWORK ADDRESS 0xC0000125 NO SECURITY MEMORY 0xC0000126 **Network Failures** NETWORK FAULT 0xC0000140 CONNECTION CLOSED 0xC0000141 CONNECTION TIMED OUT 0xC0000142 LONELY NETWORK 0xC0000143 DUPLICATE NODE 0xC0000144 CABLE DISCONNECT 0xC0000145
Version	The version field holds version of this structure. It starts with 1; 0 is not defined. Version should be 1.
Host Watchdog	This field holds the configured watchdog timeout value in milliseconds. The application may set its watchdog trigger interval accordingly. If the application fails to copy the value from the host watchdog location to the device watchdog location, the protocol stack will interrupt all network connections immediately regardless of their current state.
Error Count	This field holds the total number of errors detected since power-up, respectively after reset. The protocol stack counts all sorts of errors in this field no matter if they were network related or caused internally.
Number of configured slaves	The firmware maintains a list of slaves to which the master has to open a connection. This list is derived from the configuration database. This field holds the number of configured slaves.

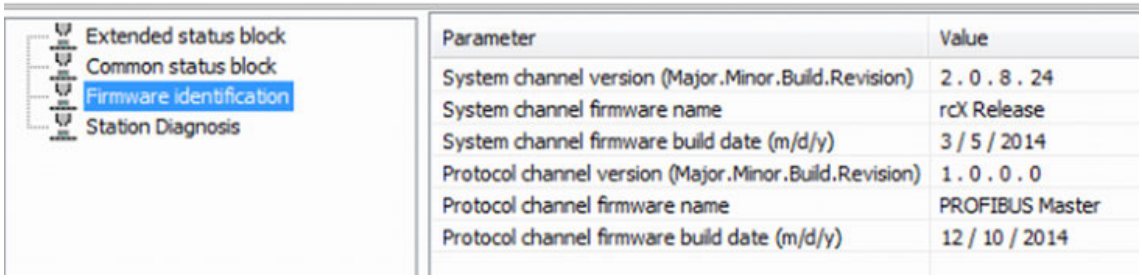
Parameter	Description
Slave state	The slave state field indicates whether the master is in cyclic data exchange to all configured slaves. In case there is at least 1 slave missing or if the slave has a diagnostic request pending, the status is set to FAILED. For protocols that support non-cyclic communication only, the slave state is set to OK as soon as a valid configuration is found. (UNDEFINED, NO_FAULT,FAILED)
Number of IO-Devices currently exchanging cyclic data	The firmware maintains a list of slaves to which the master has successfully opened a connection. Ideally, the number of active slaves is equal to the number of configured slaves. For certain fieldbus systems it could be possible that the slave is shown as activated, but still has a problem in terms of a diagnostic issue. This field holds the number of active slaves.
Number of configured IO-Devices not exchanging …	If a slave encounters a problem, it can provide an indication of the new situation to the master in certain fieldbus systems. As long as those indications are pending and not serviced, the field holds a value unequal 0. If no more diagnosis information is pending, the field is set to 0.

Table 768: Communication change of state flags

Bit Definition	Description
Ready	0 - … 1 - The *Ready* flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the Running flag is set, too.
Running	0 - … 1 - The *Running* flag is set if the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the Ready flag and the Running flag are set.
Bus On	0 - … 1 - The *Bus On* flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
Configuration Locked	0 - … 1 - The *Configuration Locked* flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the Lock Configuration flag in the control block.
Configuration New	0 - … 1 - The *Configuration New* flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the Restart Required flag.

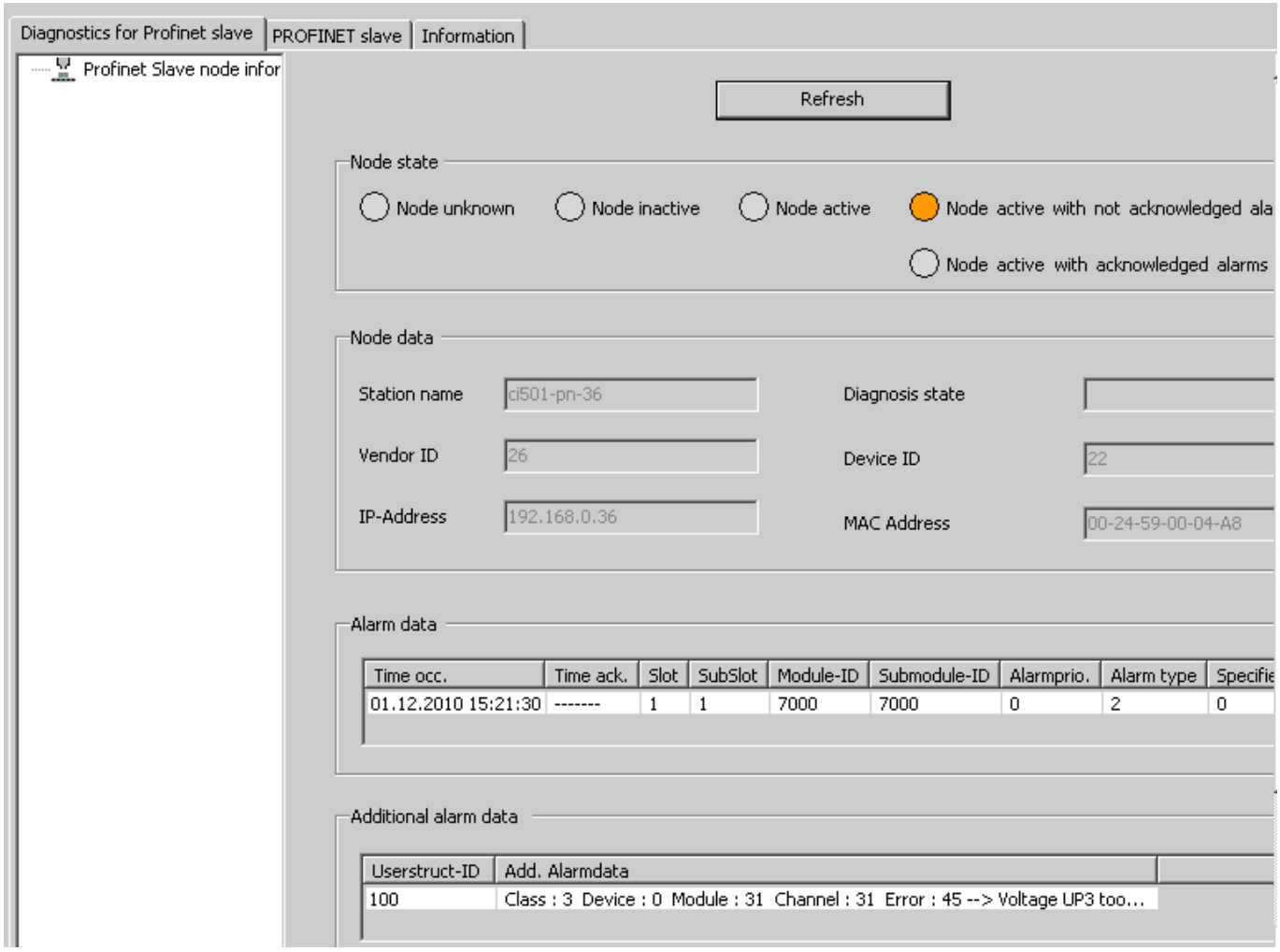
Bit Definition	Description
Restart Required	0 - … 1 - The *Restart Required* flag is set if the channel firmware requests to be restarted. This flag is used together with the Restart Required Enable flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
Restart Required Enable	0 - … 1 - The *Restart Required Enable* flag is used together with the Restart Required flag above. If set, this flag enables the execution of the Restart Required command in the netX firmware (for details on the Enable mechanism see section 2.3.2 of the netX DPM Interface Manual).

IO controller firmware identification view



Parameter	Description
System channel version	Version of the RcX operating system
System channel firmware name	Name of the operating system
System channel firmware build date	Date of the operating system
Protocol channel version	Version of the communication module
Protocol channel name	Name of the communication module
Protocol channel date	Date of the communication module

1.7.2.7.6 CM579-PNIO PROFINET IO device views



The different states for node state and the entries for node data are the same data already described in CM579_PROFINET_CM_controller_list_slaves_views ↬ *Chapter 1.7.2.7.5.2 “IO controller list of slaves view” on page 6408*.

Alarm data

In PROFINET every device can have only 1 active alarm. The PROFINET IO device diagnosis view displays the alarm data of the last active alarm of the selected device.

> *The read out of these diagnosis data does NOT affect the alarm function blocks like it does in CANopen and PROFIBUS. Automation Builder reads only a local copy of the alarm data.*

Time occurred

Date and time the alarm was triggered.

Time acknowledged

Date and time, the user has acknowledged the alarm via alarm function block.

There is no "alarm off" notification, but it is possible to get this information:

During the time the alarm condition is still active, the PROFINET IO device repeats its alarm message to the master continuously. So, if the alarm function block acknowledges an alarm, and shortly afterwards the alarm is again in not acknowledged state -> this means, the alarm cause is not gone. Otherwise it is gone.

If there is more than 1 alarm triggered on 1 slave, it could happen that after the acknowledgement of 1 alarm the other alarm gets the active alarm. So the data displayed could switch between the different alarm causes.

Slot / SubSlot / ModuleID and *SubModuleID* are addressing information to locate the alarm cause/area on a PROFINET IO device.

Alarm priority There are 2 alarm priorities : ALARM_HIGH and ALARM_LOW

Alarm type (hexadecimal)

0x0000	Reserved
0x0001	Diagnosis
0x0002	Process
0x0003	Pull a)
0x0004	Plug
0x0005	Status
0x0006	Update
0x0007	Redundancy
0x0008	Controlled by supervisor
0x0009	Released
0x000A	Plug Wrong Submodule
0x000B	Return of Submodule
0x000C	Diagnosis disappears
0x000D	Multicast communication mismatch notification
0x000E	Port data change notification
0x000F	Sync data changed notification
0x0010	Isochronous mode problem notification
0x0011	Network component problem notification
0x0012	Time data changed notification
0x0013	0x001D Reserved
0x001E	Upload and retrieval notification
0x001F	Pull module b)
0x0020	0x007F Manufacturer specific
0x0080	0x00FF Reserved for profiles
0x0100	0xFFFF Reserved

Alarm specifier Bit Meaning Value

0 ... 10	Alarm Sequence Number 0 … 2047
11	Channel Diagnosis 0 ... 1
12	Manufacturerspecific Diagnosis 0 ... 1
13	Submodule Diagnosis State 0 ... 1
14	Reserved 0
15	AR Diagnosis State 0 ... 1

The additional alarm data are completely manufacturer specific data.

Every manufacturer can specify a User structure-ID which describes a structured data called *Additional Alarmdata.*

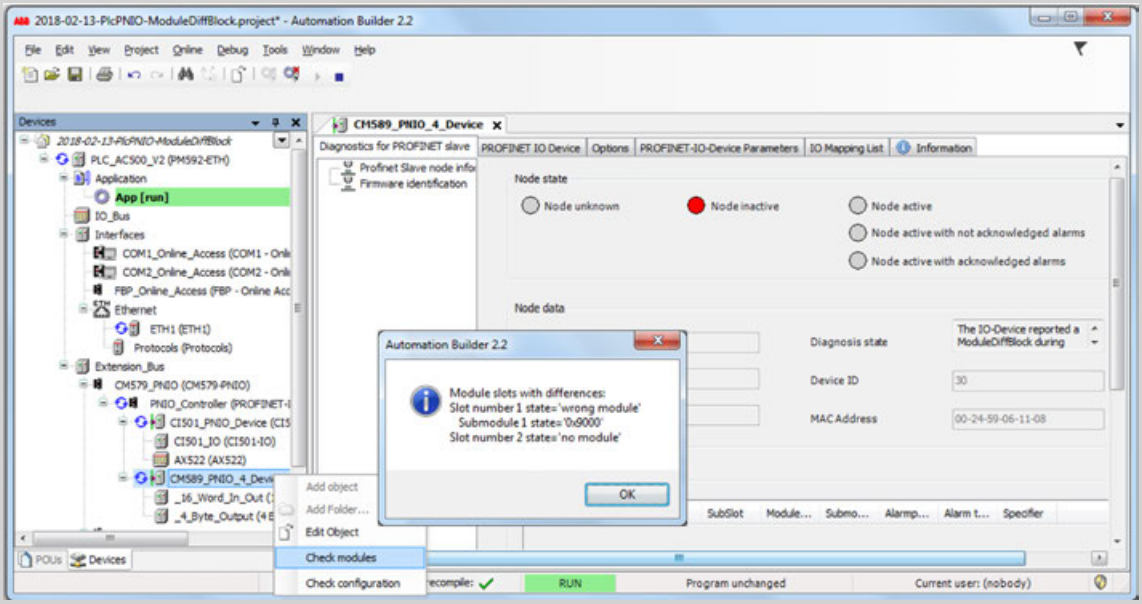
For ABB PROFINET IO devices the structure is well known and the add. Alarm data can be displayed decoded into details and error codes.

ModuleDiffBlock error

As of Automation Builder 2.1.1 the user can read diagnosis information about modules below any PROFINET IO device (e.g. CI50x or CM589).

If a modular PNIO device detects an issue with one of its modules, like plugged module does not match configuration, it reports a ModuleDiffBlock error.

Example



1. Double-click the node *“CM589_PNIO_4_Device”* and open view *“Diagnosis for PROFINET slave”*.

   ⇨ In the section *Node state* appears *Node inactive* and in the section *Node data* the *Diagnosis state* >The IO-Device reported a ModuleDiffBlock...during...<.

2. Right-click the node *“CM589_PNIO_4_Device”* and click *“Check modules”*

   ⇨ A popup-window appears.

   It shows a list of all modules that are different from the current configuration.

   The list contains the slot number(s) of the erroneous module so that a user knows which module(s) causes the issue.

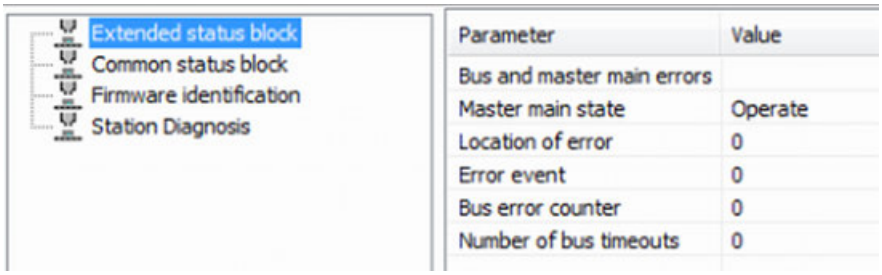
The related context menu item “Check modules” is visible in online mode of V2 PLCs only.

Automation Builder does not show the popup-window in case of no differences.

In any case the Automation Builder adds a message with the result in the message window.

1.7.2.7.7 CM598-CN CANopen communication module statistic views

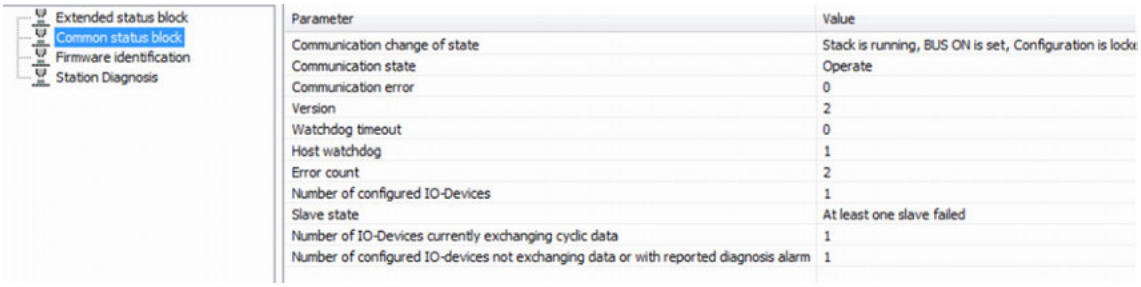
Extended status block view



Parameter	Values	Description
Bus and master main errors	TIMEOUT-ERROR	The DEVICE has detected an overstepped timeout supervision time of at least one CAN message to be sent. The transmission of this message was aborted. The data ist lost. It is an indication that no other CAN device was connected or responsive at this time to acknowledge the sent messages requests. The number of detected timeouts is fixed in the Msg_Time_Out variable. The bit will be sent when the first timeout was detected and will not be deleted anymore.
	HOST-NOT-READY-NOTIFICATION	Indicates if the HOST program has set its state to operate or not. If the bit is set the HOST program is not ready to communicate.
	EVENT-ERROR	The DEVICE has detected transmission errors. The number of detected events are fixed in the Bus_Error_Cnt and Bus Off Cnt variables. The bit will be sent when the first event was detected and will not be deleted anymore.
	NON-EXCHANGE-ERROR	At least one node hast not reached the data exchange state and no process data is exchanged with it.
	AUTO-CLEAR-ERROR	Device stopped the communication to all nodes and reached the autoclear end state.
	CONTROL-ERROR	Parameterization error or severe run time error.
Master main state	Offline	In OFFLINE state, there is no communication (data transfer) permitted at all. This is the state after initialization. This means, the master is waiting for a signal to start and does not participate in the token ring of the PROFIBUS access control mechanism

Parameter	Values	Description
	Stop	In STOP state, there is no data transfer permitted between master and slaves. Data transfer to other masters in multi-master system is allowed, however. The bus parameter set has been loaded successfully in order to get into STOP state.
	Clear	In CLEAR state, the master is able to read the input data from the DP slaves. The master forces the outputs to the slaves to be in a safe state (i.e. they contain only the value 0). For instance, incorrect data transfer of a slave can cause the PROFIBUS DP master to fall back from OPERATE state to CLEAR state. Parameterization and configuration checks are possible in this state.
	Operate	In OPERATE state, unrestricted data transfer is possible. This data transfer is cyclic, i.e. periodically, the input values are read from the slaves and the output data are written to the slaves.
Location of error	0..125	Address of the slave with error
Error event		not used currently
Bus error counter		Counter for the bus error events
Number of bus timeouts		Counter for bus timeouts

Common status block view



Parameter	Description
Communication state of change	The communication change of state register contains information about the current operating status of the communication channel and its firmware. Possible state changes (see below table for details) are: ● *Restart Required Enable* ● *Restart Required* ● *Configuration New* ● *Configuration Locked* ● *Bus On* ● *Running* ● *Ready*
Communication state	The communication state field contains information regarding the current network status of the communication channel. Depending on the implementation, all or a subset of the definitions below is supported: UNKNOWN NOT_CONFIGURED STOP IDLE OPERATE

Parameter	Description
Communication error	This field holds the current error code of the communication channel. If the cause of error is resolved, the communication error field is set to 0 (= RCX_SYS_SUCCESS) again. Not all of the error codes are supported in every implementation. Protocol stacks may use a subset of the error codes below: SUCCESS 0x00000000 **Runtime Failures:** WATCHDOG TIMEOUT 0xC000000C **Initialization Failures:** INITIALIZATION FAULT 0xC0000100 DATABASE ACCESS FAILED 0xC0000101 **Configuration Failures** NOT CONFIGURED 0xC0000119 CONFIGURATION FAULT 0xC0000120 INCONSISTENT DATA SET 0xC0000121 DATA SET MISMATCH 0xC0000122 INSUFFICIENT LICENSE 0xC0000123 PARAMETER ERROR 0xC0000124 INVALID NETWORK ADDRESS 0xC0000125 NO SECURITY MEMORY 0xC0000126 **Network Failures** NETWORK FAULT 0xC0000140 CONNECTION CLOSED 0xC0000141 CONNECTION TIMED OUT 0xC0000142 LONELY NETWORK 0xC0000143 DUPLICATE NODE 0xC0000144 CABLE DISCONNECT 0xC0000145
Version	The version field holds version of this structure. It starts with 1; 0 is not defined. Version should be 1.
Host Watchdog	This field holds the configured watchdog timeout value in milliseconds. The application may set its watchdog trigger interval accordingly. If the application fails to copy the value from the host watchdog location to the device watchdog location, the protocol stack will interrupt all network connections immediately regardless of their current state.
Error Count	This field holds the total number of errors detected since power-up, respectively after reset. The protocol stack counts all sorts of errors in this field no matter if they were network related or caused internally.
Number of configured slaves	The firmware maintains a list of slaves to which the master has to open a connection. This list is derived from the configuration database. This field holds the number of configured slaves.

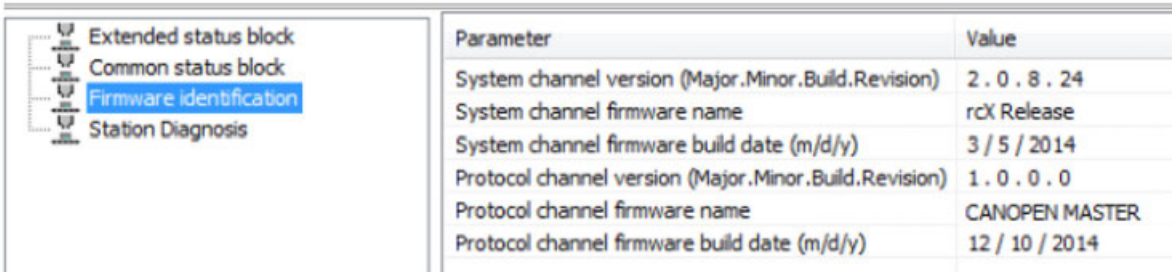
Parameter	Description
Slave state	The slave state field indicates whether the master is in cyclic data exchange to all configured slaves. In case there is at least 1 slave missing or if the slave has a diagnostic request pending, the status is set to FAILED. For protocols that support non-cyclic communication only, the slave state is set to OK as soon as a valid configuration is found. (UNDEFINED, NO_FAULT,FAILED)
Number of IO-Devices currently exchanging cyclic data	The firmware maintains a list of slaves to which the master has successfully opened a connection. Ideally, the number of active slaves is equal to the number of configured slaves. For certain fieldbus systems it could be possible that the slave is shown as activated, but still has a problem in terms of a diagnostic issue. This field holds the number of active slaves.
Number of configured IO-Devices not exchanging …	If a slave encounters a problem, it can provide an indication of the new situation to the master in certain fieldbus systems. As long as those indications are pending and not serviced, the field holds a value unequal 0. If no more diagnosis information is pending, the field is set to 0.

Table 769: Communication change of state flags

Bit Definition	Description
Ready	0 - … 1 - The *Ready* flag is set as soon as the protocol stack is started properly. Then the protocol stack is awaiting a configuration. As soon as the protocol stack is configured properly, the Running flag is set, too.
Running	0 - … 1 - The *Running* flag is set if the protocol stack has been configured properly. Then the protocol stack is awaiting a network connection. Now both the Ready flag and the Running flag are set.
Bus On	0 - … 1 - The *Bus On* flag is set to indicate to the host system whether or not the protocol stack has the permission to open network connections. If set, the protocol stack has the permission to communicate on the network; if cleared, the permission was denied and the protocol stack will not open network connections.
Configuration Locked	0 - … 1 - The *Configuration Locked* flag is set, if the communication channel firmware has locked the configuration database against being overwritten. Re-initializing the channel is not allowed in this state. To unlock the database, the application has to clear the Lock Configuration flag in the control block.
Configuration New	0 - … 1 - The *Configuration New* flag is set by the protocol stack to indicate that a new configuration became available, which has not been activated. This flag may be set together with the Restart Required flag.

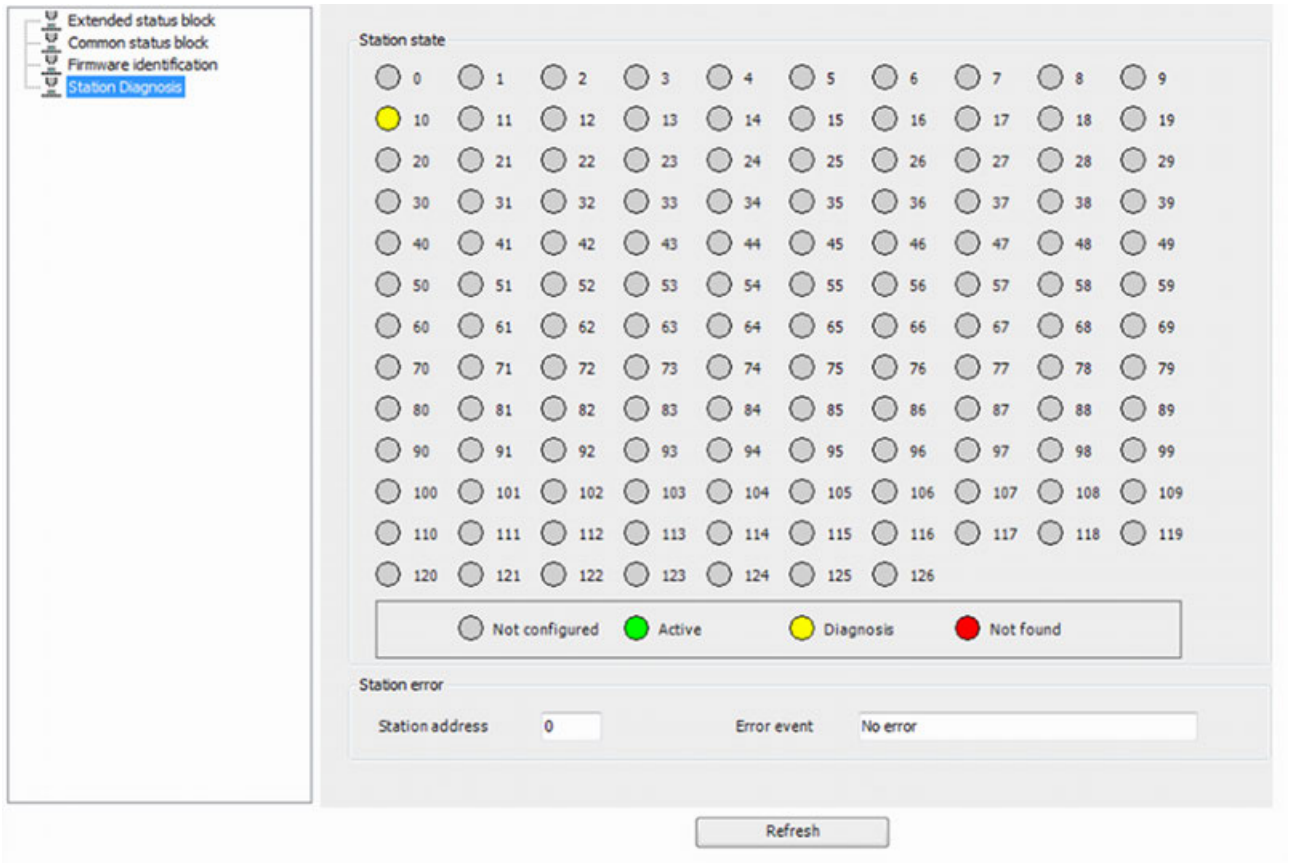
Bit Definition	Description
Restart Required	0 - … 1 - The *Restart Required* flag is set if the channel firmware requests to be restarted. This flag is used together with the Restart Required Enable flag below. Restarting the channel firmware may become necessary, if a new configuration was downloaded from the host application or if a configuration upload via the network took place.
Restart Required Enable	0 - … 1 - The *Restart Required Enable* flag is used together with the Restart Required flag above. If set, this flag enables the execution of the Restart Required command in the netX firmware (for details on the Enable mechanism see section 2.3.2 of the netX DPM Interface Manual).

Firmware identification view



Parameter	Description
System channel version	Version of the RcX operating system
System channel firmware name	Name of the operating system
System channel firmware build date	Date of the operating system
Protocol channel version	Version of the communication module
Protocol channel name	Name of the communication module
Protocol channel date	Date of the communication module

Station diagnosis view



For every configured slave the following data is displayed:

Value	Description
Not configured	Not configured
Active	All OK: slave is in data exchange
Diagnosis	Configured, active has diagnosis
Not found	Configured but not found

1.7.2.7.8 CI506-PNIO communication interface CANopen master view

Master view

The PROFINET slave CI506-PNIO has a built-in CANopen gateway. The user can plug a CANopen Master.

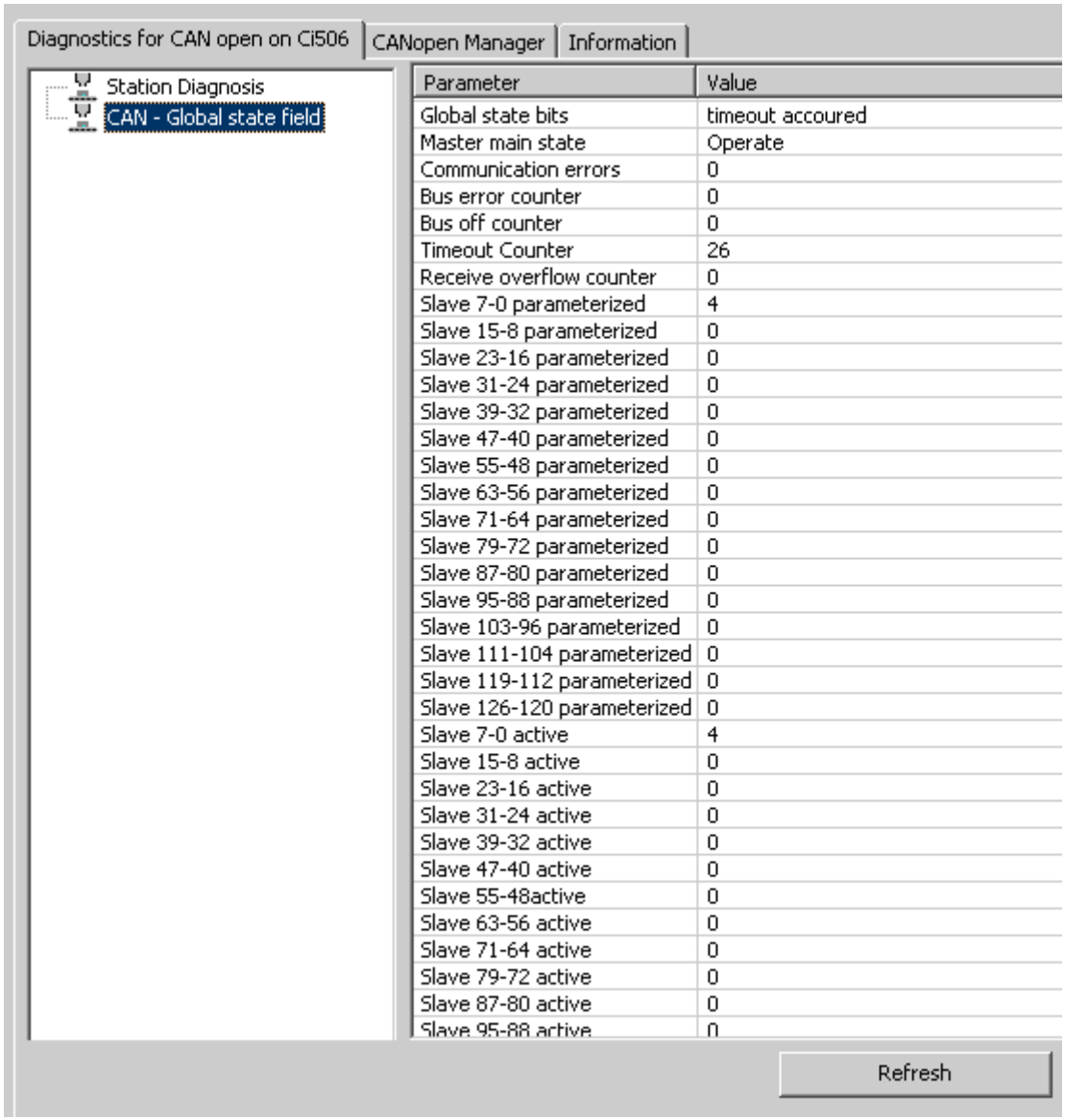
Because of diverse components and assemblies some diagnosis views are different and there are differences in operation.

All these CI506 CANopen views have a refresh button and are not refreshed cyclically.

↳ *Chapter 1.7.2.7.8.2 “Station view” on page 6423* ↳ *Chapter 1.7.2.7.8.3 “Global state field view” on page 6424*

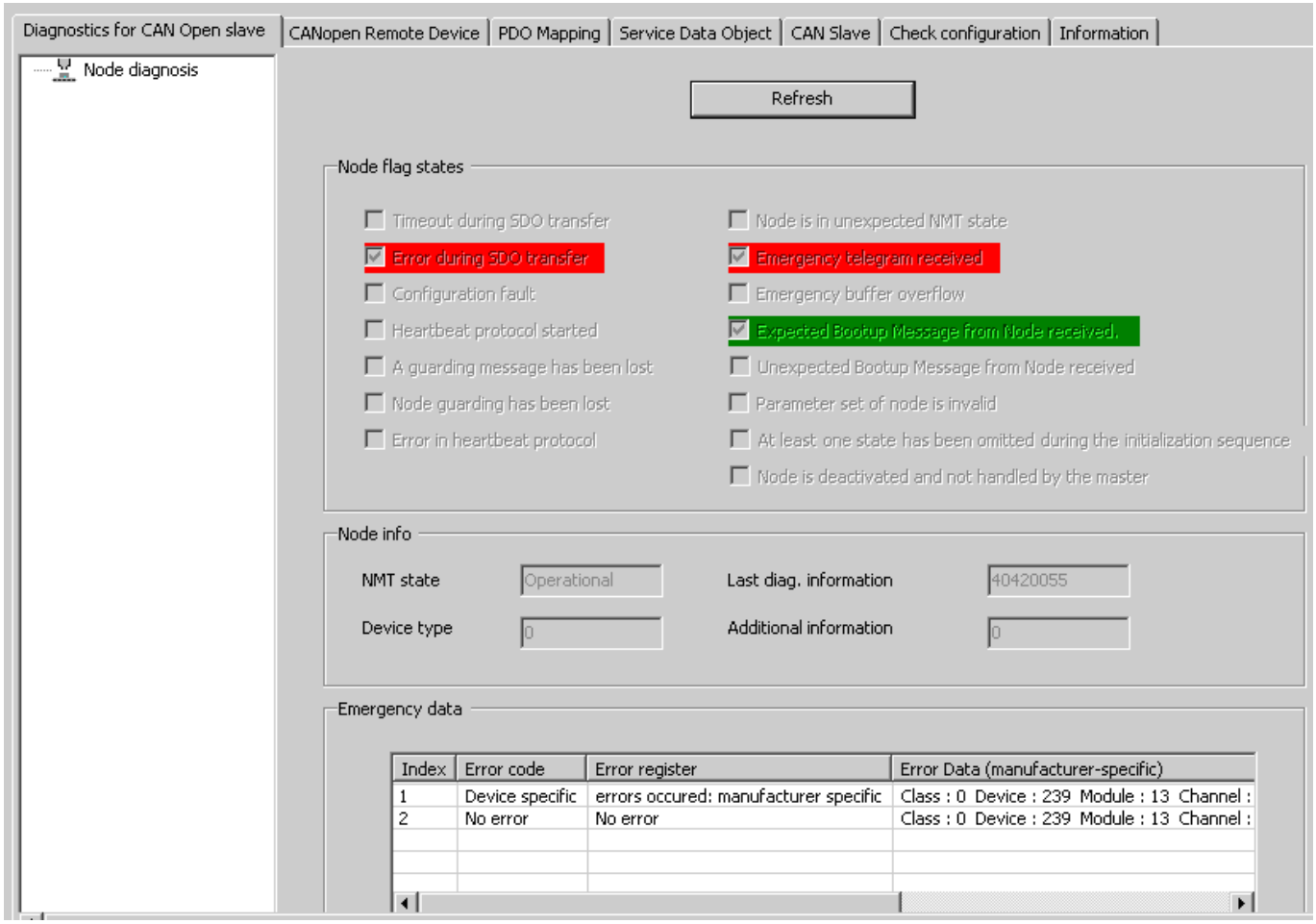
Station view

Global state field view



Parameter	Value
Global state bits	timeout accoured
Master main state	Operate
Communication errors	0
Bus error counter	0
Bus off counter	0
Timeout Counter	26
Receive overflow counter	0
Slave 7-0 parameterized	4
Slave 15-8 parameterized	0
Slave 23-16 parameterized	0
Slave 31-24 parameterized	0
Slave 39-32 parameterized	0
Slave 47-40 parameterized	0
Slave 55-48 parameterized	0
Slave 63-56 parameterized	0
Slave 71-64 parameterized	0
Slave 79-72 parameterized	0
Slave 87-80 parameterized	0
Slave 95-88 parameterized	0
Slave 103-96 parameterized	0
Slave 111-104 parameterized	0
Slave 119-112 parameterized	0
Slave 126-120 parameterized	0
Slave 7-0 active	4
Slave 15-8 active	0
Slave 23-16 active	0
Slave 31-24 active	0
Slave 39-32 active	0
Slave 47-40 active	0
Slave 55-48active	0
Slave 63-56 active	0
Slave 71-64 active	0
Slave 79-72 active	0
Slave 87-80 active	0
Slave 95-88 active	0

1.7.2.7.9 CI506-PNIO communication interface CANopen slave view



The node flag is an unsigned integer (32 bits) which represents different states of the Slave:

States with green background are normal and expected. All other states have a red background when active indicating that something unexpected has happened or diagnosis data is available.

Node flag This variable is organized as a bitfield as described in the table below:

Node Diagnostic Flags	Bit Name Description
D31 FLAG_DEACTIVATED	Node is deactivated and not handled by the Master. This bit does not generate an entry in the diagnostic list.
D30 FLAG_STATE_NOT_HANDLED	At least one state has been omitted during the initialization sequence of the node. This bit does not generate an entry in the diagnostic list.
D13 ..	Reserved
D29 ..	Reserved
D12 FLAG_INVALID_PARAMETER	Parameter set of node is invalid.
D11 FLAG_UNEXPECTED_BOOTUP	Unexpected Bootup Message from node received.
D10 FLAG_BOOTUP	Expected Bootup Message from node received.
D9 FLAG_EMCY_BUFF_OVER	Emergency buffer overflow
D8 FLAG_EMCY_RECEIVED	Emergency telegram received.
D7 FLAG_UNEXPECTED_STATE	Node is in unexpected NMT state.

Node Diagnostic Flags	Bit Name Description
D6 FLAG_HEARTBEAT_ERROR	Error in heartbeat protocol.
D5 FLAG_CON_LOST	Node guarding has been lost.
D4 FLAG_GUARD_ERROR	A guarding message has been lost. This bit does not generate an entry in the diagnostic list.
D3 FLAG_HEARTBEAT_STARTED	Heartbeat protocol started. This bit does not generate an entry in the diagnostic list.
D2 FLAG_CFG_FAULT	Configuration fault.
D1 FLAG_SDO_ERROR	Error during SDO transfer.
D0 FLAG_SDO_TIMEOUT	Timeout during SDO transfer.

Nmt (network management) state

This variable holds the internal NMT state of the node according to the following table:

NMT State	Value
CANOPEN_MASTER_NODE_NMT_STATE_UNKNOWN	0x00000000L
CANOPEN_MASTER_NODE_NMT_STATE_INITIALISING	0x00000001L
CANOPEN_MASTER_NODE_NMT_STATE_STOPPED	0x00000002L
CANOPEN_MASTER_NODE_NMT_STATE_OPERATIONAL	0x00000003L
CANOPEN_MASTER_NODE_NMT_STATE_PRE_OPERATIONAL	0x00000004L
CANOPEN_MASTER_NODE_NMT_STATE_RESET_APPLICATION	0x00000005L
CANOPEN_MASTER_NODE_NMT_STATE_RESET_COMM	0x00000006L

Last diag info

In this information the last diagnosis information of this node station is held down.

Device type valid

This variable indicates whether the device type in variable DeviceType is valid or not.

DeviceType

These 4 bytes are read out from the node while startup. There are several predefined profile numbers existing, each described in an own specification manual. Here is an extract of these:

	decimal
Device Profile for I/O modules	401
Device Profile for Drives and Motion Control	402
Device Profile for Encode	406

Emergency data

Each slave can held up to 5 emergency data structures which consists of 8 bytes each.

An Emergency data structure consist of a 2 byte error code, a 1 byte error register and 5 bytes emergency data (which are always manufacturer specific):

Emergency Error Codes (hex)	Error Code Description
00xx	Error Reset or No Error
10xx	Generic Error
20xx	Current
21xx	Current, device input side
22xx	Current inside the device
23xx	Current, device output side
30xx	Voltage
31xx	Mains Voltage
32xx	Voltage inside the device
33xx	Output Voltage
40xx	Temperature
41xx	Ambient Temperature
42xx	Device Temperature
50xx	Device Hardware
60xx	Device Software
61xx	Internal Software
62xx	User Software
63xx	Data Set
70xx	Additional Modules
80xx	Monitoring
81xx	Communication
8120	CAN in Error Passive Mode
8130	Life Guard Error or Heartbeat Error
8140	Recovered from bus off
8150	Transmit COB-ID collision
82xx	82xx
8210	PDO not processed due to length error
8220	PDO length exceeded
90xx	External Error
F0xx	Additional Functions
FFxx	FFxx Device specific

The bits of the error register have the following meaning:

Error Code	
CANOPEN_MASTER_ERROR_REGISTER_GENERIC_BIT	0x01
CANOPEN_MASTER_ERROR_REGISTER_CURRENT_BIT	0x02
CANOPEN_MASTER_ERROR_REGISTER_VOLTAGE_BIT	0x04
CANOPEN_MASTER_ERROR_REGISTER_TEMPERATURE_BIT	0x08
CANOPEN_MASTER_ERROR_REGISTER_COMM_ERROR_BIT	0x10
CANOPEN_MASTER_ERROR_REGISTER_DEV_PROFILE_BIT	0x20

Error Code	
CANOPEN_MASTER_ERROR_REGISTER_RESERVED_BIT	0x40
CANOPEN_MASTER_ERROR_REGISTER_MANU_SPEC_BIT	0x80

For some ABB CANopen Slaves the emergency data is well known and can be displayed in a decoded way.

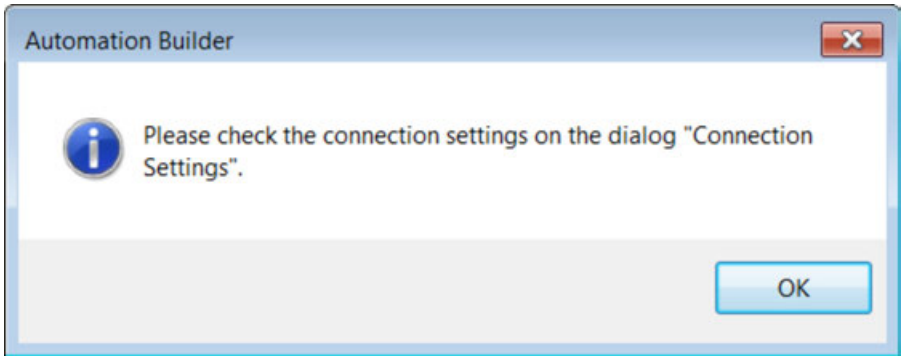
ABB CANopen Slaves are signalling the "Alarm Off/Gone" situation by sending a second emergency data with error code "No Error"

1.7.2.7.10 CI52x Modbus diagnosis

1. Double-click node *"CI52x_MODTCP"* in the device tree.
2. Select *"CI52x Diagnosis"* tab.
   ⇨ The button *[Get Diagnosis]* appears in the tab view.
3. Click on the button *[Get Diagnosis]*.
   ⇨ One of the following use cases will be displayed:
   - Device not connected *"Device not connected" on page 6428*
   - No Errors on the device *"No errors on the device" on page 6428*
   - Diagnosis list *"Diagnosis list" on page 6429*

Device not connected

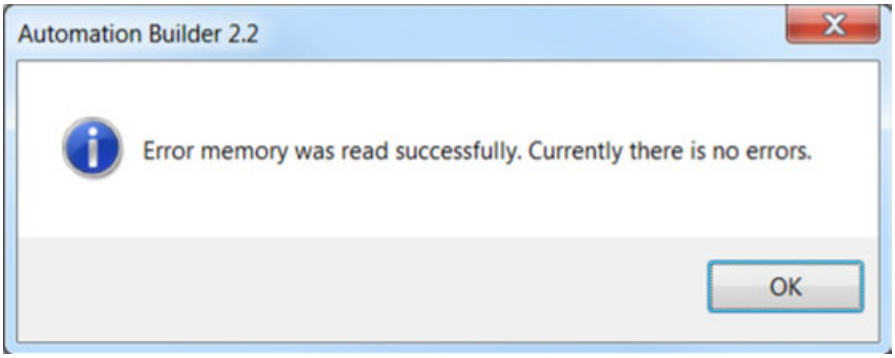
If there is no device connected to the project, the following dialog will be displayed:



1. Select tab *"Connection Settings"* and enter the IP address for the device.
2. Click again button *[Get Diagnosis]*.

No errors on the device

If there are no errors on the device the following dialog will be displayed:



Diagnosis list If the device is not correctly configured the errors will be displayed with *"Error Code"* and *"Code Description"*.

	Error Code	Code Description
▸	234946507	Process voltage too low
	234946541	Process voltage switched off
	234881072	Module number: 1: Measurement overflow or cut wire at the analog input
*		

1.7.3 Diagnosis messages

1.7.3.1 Possible error combinations

The following tables contain the possible combinations of error numbers.

Component		**Device**		**Module or type**		**Channel**		**Remark**
Number	**Comp**	**Number**	**Dev**	**Number**	**Mod**	**Number**	**Ch**	**<- Software view**
	d1		**d2**		**d3**		**d4**	**<- Device display**
9	CPU	0	CPU	1	Operating system	1	Initialization error	
						2	Runtime error	
						3	Configuration error	
						31	Operating system	
				2	Runtime system	1	Initialization error	
						2	Runtime error	
						3	Configuration error	
						31	Operating system	
				4	IEC task online display %s	1	Initialization error	
						2	Runtime error	
		1	External Communication Module 1...6 or internal	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	Communication Module			
		2	External Communication Module 2	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	Communication Module			

Component		Device		Module or type		Channel		Remark
Number	Comp	Number	Dev	Number	Mod	Number	Ch	<- Software view
	d1		d2		d3		d4	<- Device display
		3	External Communication Module 3	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	Communication Module			
		4	External Communication Module 4	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	Communication Module			
		8	Local I/O	1	Initialization	0	not used	
				2	Runtime error			
				3	Project			26
				4	Protocol			
				31	local I/O itself			
		10	Internal Communication Module	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	Communication Module			
		11	COM1	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	COM			
		12	COM2	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26
				4	Protocol			
				31	COM			
		13	FBP	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			26,
				4	Protocol			
				31	FBP			
		14	I/O bus	1	Initialization	0	not used	18, 15

Component		Device		Module or type		Channel		Remark
Number	Comp	Number	Dev	Number	Mod	Number	Ch	<- Software view
	d1		d2		d3		d4	<- Device display
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	I/O bus			
		16	System Flash EPROM	0... 31	Sector, block no. or similar	0... 31	Sector, block no.	
		17	RAM	1	Initialization	0	ramdisk	
				2	Runtime error	3	webvisu files	
				3	Project	4	bootproject	
				4	Protocol			
				31	RAM itself			
		18	Data Flash EPROM	1	Initialization	0	cfg file	
				2	Runtime error	1	production data header	
				3	Project	2	production data	
				4	Protocol	3	webvisu files	
				31	Data Flash EPROM itself	4	bootproject	
						31	module itself	
		19	HW watch dog	31	Watchdog	31	Watchdog	
		20	SD Memory Card	1	Initialization	0	bootcode file	
				2	Runtime error	1	firmware file	
				3	Configuration	2	sdcard.ini file	
				4	Protocol	3	formatting failed	
				31	SDcard	4	bootproject file	
						5	persistent file	
						6	retain data file	
						7	OEM file	
						8	MMI FW file	
						9	local I/O file	
						10 +	10 + communication module communication module cfg file	
						19	any cfg file	

Component		Device		Module or type		Channel		Remark
Number	Comp	Number	Dev	Number	Mod	Number	Ch	<- Software view
	d1		d2		d3		d4	<- Device display
						20 +	20 + communication module communication module FW file	
						29	any FW file	
				31		31	module itself	
		21	Display	1	Initialization	0	any	
				2	Runtime error	1	FW version obsolete	
				4	Protocol			
				31	Display			
		22	Battery	31	Battery	31	Battery	8
		23	Clock	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	Clock			
		24	FPU	1	Initialization	0	no error	
				2	Runtime error	1	division by zero	
						2	overflow	
						3	underflow	
						4	invalid	
						5	inexact	
						6	function	
		25	Host	1	Initialization	0	any	
				2	Runtime error	x	channel	
				3	Project			
		26	User					
		27	Power Supply	1	Initialization	0	no error	
				2	Runtime error	1	voltage dip	
						2	reset after dip	
		28	Production Data	1	Initialization	0	any	
				2	Runtime error	1	key	
						2	data	
		29	RUN/StopSwitch	31	device itself	0	PLC in STOP	
						1	PLC in RUN	
		30	Emulated SRAM	1	Initialization	0	unknown	
				2	Runtime error	1	retain	

Component		Device		Module or type		Channel		Remark
Number	Comp	Number	Dev	Number	Mod	Number	Ch	<- Software view
	d1		d2		d3		d4	<- Device display
						2	persistent	
						4	coredump	
		31	Internal PHY	1	Initialization	0	unknown	
				2	Runtime error	0	any	
		32	Configuration Data	1	Initialization	2	data	
				2	Runtime error			
		33	Flashdisk	1	Initialization	0	any	
				2	Runtime error	1	flashdisk init failed	
				31	Flashdisk itself	2	no filesystem	
						3	formatting failed	
						4	MBR writing failed	
						5	threshold reached	
						6	write protection	
		34	SRAM disk	1	Initialization	0	any	
				2	Runtime error	2	no filesystem	
				31	SRAMdisk itself			

Component		Device		Module or type		Channel		Remark
	Comp		Dev		Mod		Ch	<- Software view
	d1		d2		d3		d4	<- Device display
1..4 10	External Communication Module 1..4 or internal Communication module	0..254	Address/ Socket: Fieldbus: Slave ARCNET: ID partner - 1 Modbus: Comm partner	0..29	Module number	0..31	Channel number	
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	Communication Module	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			

Component		Device		Module or type		Channel		Remark
	Comp		**Dev**		**Mod**		**Ch**	**<- Software view**
	d1		**d2**		**d3**		**d4**	**<- Device display**
				5	Operating system Communication Module	0... 15	Bit 0...3 of the error number reported by the Communication Module	see chapter Communication Module errors *Chapter 1.7.3.4 "Communication modules diagnosis" on page 6489*
				6	Task 1 Communication Module		Bit 4...7 of the error number reported by the Communication Module	
				7	Task 2 Communication Module			
				8	Task 3 Communication Module			
				9	Task 4 Communication Module			
				10	Task 5 Communication Module			
				11	Task 6 Communication Module			
				12	Task 7 Communication Module			
				13	Watchdog Communication Module			
				31	Communication Module			
11	COM1	0..254	Address: CS31: Slave Dec. expansion: Slave Modbus : Comm partner	0..29	Module number CS31: Module type: 00 - Digital input 01 - Analog input 02 - Digital output 03 - Analog output 04 - Digital in/output 05 - Analog in/output	0..31	Channel number	8, 48
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	COM	1	Initialization	0	not used	
				2	Runtime error			

Component		Device		Module or type		Channel		Remark
	Comp		**Dev**		**Mod**		**Ch**	**<- Soft-ware view**
	d1		**d2**		**d3**		**d4**	**<- Device display**
				3	Configuration			
				4	Protocol			
				31	COM			
12	COM2	0..254	Address: Slave Modbus: Comm partner	0..29	Module number	0..31	Channel number	
				30	Module number > 29	0..31	Channel number	
				31	Slave device	31	Slave device	
		255	COM	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	COM			

Component		Device		Module or type		Channel		Remark
	Comp		**Dev**		**Mod**		**Ch**	**<- Soft-ware view**
	d1		**d2**		**d3**		**d4**	**<- Device display**
8	Local I/O	1	PWM	31	local I/O itself	0	any	
		2	interrupt			1	Initialization	
		3	fast counter			2	Runtime	
		4	analog input			3	Project	
		5	analog output			4	Protocol	
		255	local I/O itself			31	local I/O itself	
13	FBP	0...254	Module number P arameter number	0..30	Slot number	0..31	Channel number	
		255	FBP	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	FBP			

Component		Device		Module or type		Channel		Remark
	Comp		**Dev**		**Mod**		**Ch**	**<- Soft-ware view**
	d1		**d2**		**d3**		**d4**	**<- Device display**
14	I/O bus	0... 254	I/O bus module	0..6	Module type: 00 - Digital input 01 - Analog input 02 - Digital output 03 - Analog output 04 - Digital in/ output 05 - Analog in/ output 06 - others (e.g., fast counter)	0..31	Channel number	%s
				31	Module	1	Initialization error	
						2	Runtime error	
						3	Configuration	26
						4	Protocol	
						31	Module	
		255	I/O bus	1	Initialization	0	not used	
				2	Runtime error			
				3	Configuration			
				4	Protocol			
				31	I/O bus			
15	User	0... 255	any	0..31	any	0..31	any, meaning is project-specific	

Remarks for FBP diagnosis block:

- Display:
  - Error class = Byte 6 Bit 6..7
  - Device = Byte 3
  - Module = Byte 4
  - Channel = Byte 5
  - Error identifier = Byte 6 Bit 0..5
- FBP diagnosis blocks do not contain the interface identification.

1.7.3.2 CPU diagnosis

Table 770: Diagnosis messages directly reported by the CPU

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	0	2	0	15	Not enough memory to generate the external reference list		150 999 055	E2:
E2	9	0	2	1	40	Expected and current FWAPI version are incompatible	Update to newer version of PLC firmware	150 999 144	E2:
E2	9	0	2	2	2	Heap check failed / heap is corrupt	1) Check project regarding usage of pointers for write access. 2) Check project regarding usage of loops containing write accesses to variables probably exceeding their borders. 3) Check project regarding usage of dynamically allocated memory. 4) Check stack size of application task.	150 999 170	E2:
E2	9	0	2	2	37	Cycle time is greater than the set watchdog time	Change task configuration	150 999 205	E2:
E2	9	0	2	2	38	Access violation by an IEC task, e. g. zero pointer (details in call hierarchy) Exception cannot be assigned to an IEC task	Correct program	150 999 206	E2:
E2	9	0	2	3	27	No configuration available	Please contact AC500 support	150 999 259	E2:
E2	9	0	2	x ≥ 4	38	x = 2: Exception cannot be assigned to IEC Task x ≥ 4: Index of the faulty IEC task, Task Index = x - 4	Correct program	150 999 334	E2:
E2	9	0	x ≥4	2	38	x = 2: Exception cannot be assigned to IEC Task x ≥ 4: Index of the faulty IEC task, Task Index = x - 4	Correct program	151 003 302	E2: IEC Task 1

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	0	x	2	37	Cycle time of IEC task index x - 4 is greater than watchdog time (x = 31 -> IEC task index > 27)	Change task configuration	151 003 301	E2: IEC Task 0
E2	9	0	5	2	37	Cycle time is greater than the set watchdog time Cycle time exceeded, but shorter than watchdog time	Change task configuration	151 003 301	E2: IEC Task 1
E2	9	1..6/ 10	1	0	15	• Watchdog task could not be installed • Installation of the Communication Modbus bus driver failed • Initialization error, not enough memory	Check communication module Check CPU firmware version	151 652 367	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	1	0	17	Access test failed.	Check communication module Check CPU FW version. For safety PLC: Check safety PLC switch address setting. Restart safety PLC. If this error persists, replace safety PLC.	151 652 369	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER
E2	9	1..6/ 10	1	0	18	Watchdog test for the Communication Module failed	Check communication module	151 652 370	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	1	0	30	Error in configuration data, PLC cannot read configuration data	Create new configuration data	151 652 382	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	1	0	34	Timeout when setting the warm start parameters of the Communication Module	Check communication module	151 652 386	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	1	0	38	Installation of the Communication Module driver failed	Check communication module and FW version	151 652 390	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	1..6	1	0	43	Internal error	For safety PLC: Check safety PLC switch address setting. Restart safety PLC. If this error persists, replace safety PLC.	151 652 395	E2: Ext. [1..6] [COUPLER]
E2	9	1..6	2	0..1	0	This error occurs, if the CPU was stopped but a coupler does not stop the communication.	Restart CPU or call support	151 654 400 151 654 464	E2: Ext. [1..6] [COUPLER]
E2	9	1..6/ 10	2	0	15	Error occurred when creating the I/O description list of the Communication Module	Check communication module and FW version	151 654 415	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6	3	0	15	There is not enough memory available for configuration of PROFINET Controller	Reboot PLC, reduce PROFINET configuration and reload project	151 656 463	E2: Ext. [1..6] [COUPLER]
E2	9	1..6/ 10	3	0	27	Error configuration or no configuration available	Check configuration	151 656 475	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	3	0	40	Error in firmware version of Communication Module (version not supported/too old)	Update firmware	151 656 488	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6	3	0	43	Internal error occured during configuration of PROFINET Controller	Reboot PLC and reload project	151 656 491	E2: Ext. [1..6] [COUPLER]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	1..6/ 10	31	0..7	3	Watchdog error Communication Module/ channel	Check communication module and FW version	151 713 795 151 713 859 151 713 923 151 713 987 151 714 051 151 714 115 151 714 179 151 714 243	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6/ 10	4	0	15	Installation of a driver for the Communication Module failed	Check communication module and FW version	151 658 511	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E2	9	1..6	31	0	43	Internal error	Replace module	151 713 835	E2: Ext. [1..6] [COUPLER]
E2	9	1..6	31	1	17	Internal error, no access to IO data	Restart CPU or call support	151 713 873	E2: Ext. [1..6] [COUPLER]
E2	9	1..6	31	2	17	Internal error, no access to IO data	Restart CPU or call support	151 713 937	E2: Ext. [1..6] [COUPLER]
E2	9	8	4	0	0	Maximum errors in series detected 50 telegrams in sequence are invalid or corrupted. Onboard I/O module is shut down	Restart PLC. If error still exists, replace PLC.	151 527 424	E2: Onboard I/O
E2	9	11.. 12	2	0	43	Watchdog error of CS31 protocol task.	Check CS31 configuration and task configuration	151 719 979	E2: COM[1|2]
E2	9	11.. 13	1	0	15	Installation of a protocol driver for the serial interface failed, not enough memory	Check CPU FW version	151 717 903	E2: COM[1|2] E2: FBP

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	11..13	1	0	17	Initialization of the protocol driver for the serial interface failed	Check CPU FW version	151717905	E2: COM[1|2] E2: FBP
E2	9	11..13	1	0	38	Installation of the hardware for the serial interface failed	Check CPU FW version	151717926	E2: COM[1|2] E2: FBP
E2	9	11..13	3	0	27	No configuration available	Please contact AC500 support	151722011	E2:
E2	9	14	1	0	15	Not enough resources for the I/O-Bus	Check CPU FW version	151914511	E2: I/O-Bus
E2	9	14	1	0	17	Installation of the I/O-Bus driver failed	Check CPU FW version	151914513	E2: I/O-Bus
E2	9	14	1	0	43	Incorrect data format of the hardware driver of the I/O-Bus	Check CPU FW version	151914539	E2: I/O-Bus
E2	9	14	2	0	43	I/O-Bus communication breakdown	Restart PLC	151916587	E2: I/O-Bus
E2	9	20	31	2	2	Error factory test	Check memory card	152369282	E2: SD card
E2	9	20	31	31	18	Error update	Check memory card	152371154	E2: SD card
E2	9	21	31	1	2	Display firmware is below version 2.2 AC500-eCo: Firmware of RTC module too old	Update display firmware AC500-eCo: Update firmware of RTC module	152434754	E2:
E2	9	24	2	1	38	FPU division by zero	Correct program	152572006	E2:
E2	9	24	2	2	38	FPU overflow	Correct program	152572070	E2:
E2	9	24	2	3	38	FPU underflow	Correct program	152572134	E2:
E2	9	24	2	4	38	Forbidden FPU operation	Correct program	152572198	E2:
E2	9	24	2	6	38	FPU library function generated	Correct program	152572326	E2:

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	9	25	3	1..2	27	No configuration available	Please contact AC500 support	152 639 579 152 639 643	E2:
E2	9	31	1	0	3	Error internal Ethernet	Replace module	153 028 611	E2:COM [1|2] E2: FBP
E3	9	0	2	1	15	Init error Web Server, error allocating memory	Make web visualization smaller, clean up user RAM disk	150 999 119	E3:
E3	9	0	2	1	17	Registering the handler for persistent data areas, areas do not work correctly	Check CPU FW version	150 999 121	E3:
E3	9	0	2	2	20	Program not started because of an existing error (see PLC configuration CPU parameters, stop on error class)	Eliminate error and acknowledge	150 999 188	E3:
E3	9	0	2	2	37	User program contains an endless loop, a stop by hand is necessary	Correct user program	150 999 205	E3:
E3	9	0	2	3	26	Configuration error	Adapt PLC configuration	150 999 258	E3:
E3	9	0	2	3	1	Loading of boot project failed due to invalid target ID	Use correct target	150 999 233	E3:
E3	9	0	2	3	27	No configuration available	Create new configuration, check configuration	150 999 259	E3:
E3	9	0	3	0	40	Expected and current major configuration version are not equal	Check version of PLC firmware and PLC configuration	151 001 128	E3:
E3	9	0	4	0	3	Initialization of shared interface failed.	Check CPU and communication module configuration.	151 003 139	E3: IEC Task 0
E3	9	0	4	1	3	Initialization of shared interface failed.	Check CPU and communication module configuration.	151 003 203	E3: IEC Task 0
E3	9	0	4.. 31	3	26	Event-controlled task, unknown external event	Check task configuration	151 003 354	E3: IEC Task [0..27]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	0	5	2	37	CPU will not run when IRQ task is configured	Check task configuration	151 003 301	E3: IEC Task 1
E3	9	1..6/10	2	0	12	Error occurred when reading the I/O description	Check communication module configuration	151 654 412	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	1..6/10	2	0	26	Program was not started because of invalid configuration data of the Communication Module	Configure communication module	151 654 426	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	1..6/10	2	0	29	Error occurred when deleting the configuration	Check communication module	151 654 429	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	1..6/10	2	0	30	Exporting coupler configuration failed	Contact service	151 654 430	E3: Ext. [1..6] [COUPLER]
E3	9	1..6/10	3	0	26	In the PLC configuration, the Communication Module was not configured correctly or not at all	Adapt PLC configuration	151 656 474	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	1..6/10	4	0	26	IEC60870: Queue overrun	On IEC60870 try to increase sendbuffer k value	015 165 822	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	1..6/10	31	0	40	Ethernet: Wrong firmware version of Ethernet Communication Module PROFINET Wrong firmware version of PROFINET Communication Module or wrong configuration	Ethernet: Update firmware PROFINET: Update firmware. It is not possible to use the 2.1 configuration format with firmware version 2.0.1	151 713 832	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	9	8	1	0	3	Timeout occurred while downloading configuration to Onboard I/O.	Restart PLC	151 521 283	E3:
E3	9	8	1	0	8	Onboard I/O module is defective, missing or firmware (bootcode/application) is invalid or not supported	Restart PLC, if error still exists replace PLC	151 521 288	E3:

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	8	1	0	31	Wrong or unknown Onboard I/O module installed. Wrong Onboard I/O module in PLC configuration selected (e .g. module 8DI+6DO selected for PM564) or mismatch between production data and installed module detected.	Check PLC configuration	151 521 311	E3:
E3	9	8	4	0	0	Maximum errors in series detected. More than 50 telegrams in sequence are invalid or corrupted. Onboard I/O module is shut-down	Restart PLC	151 527 424	E3:
E3	9	8	31	0	43	Internal error in Onboard I/O module detected, e. g. resource problem or error of device driver.	Restart PLC	151 582 763	E3:
E3	9	10	4	0	9	IEC60870: Command queue within the IEC60870 Protocol ran out of resources.	Please contact AC500 support	151 658 505	E3: Int. [COUPLER]
E3	9	11.. 13	3	0	26	Configuration error of the serial interface	Check configuration	151 722 010	E3: COM[1|2] E3: FBP
E3	9	11.. 13	3	4	26	Configuration error at protocol configuration	Check configuration	151 722 266	E3: COM[1|2]
E3	9	14	3	0	26	Parameter "Error LED"=Failsafe is only allowed, if parameter "Behaviour of outputs in stop=actual state in hardware and online"	Change configuration	151 918 618	E3: I/O-Bus
E3	9	16	1	0	13	Writing of the boot project failed (possibly no valid boot project available)	Reload project	152 045 581	E3:
E3	9	18	2	4	14	Failed to delete boot project		152 045 582	E3:
E3	9	20	1	2	13	Failed to write to / read from SD card	Check write protection and file system consistenc	152 307 853	E3: SD card

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	1	20..24	8	Error firmware update of the SD card, file could not be opened	Check memory card	152309000 152309064 152309128 152309192 152309256	E3: SD card Int/Ext i <Name> FW
E3	9	20	1	20..24	12	Error firmware update of the SD card, error while reading the file	Check memory card	152309004 152309068 152309132 152309196 152309260	E3: SD card Int/Ext i <Name> FW
E3	9	20	1	20..24	13	Error firmware update of the SD card, error while writing the file	Check CPU + communication module	152309005 152309069 152309133 152309197 152309261	E3: SD card Int/Ext i <Name> FW

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	1	20..24	17	Error firmware update of the SD card, error while accessing the Communication Module	Check communication module	152309009 152309073 152309137 152309201 152309265	E3: SD card Int/Ext i <Name> FW
E3	9	20	1	20..24	31	Error firmware update of the SD card, file does not match the Communication Module type	Check memory card	152309023 152309087 152309151 152309215 152309279	E3: SD card Int/Ext i <Name> FW
E3	9	20	1	31	43	Error creating the SD card task, involving no more use of the SD card	Restart PLC, check memory card	152309739	E3: SD card
E3	9	20	3	10..14	8	Error while reading/ writing the configuration data from/to the SD card, file could not be opened	Check memory card	152312456 152312520 152312584 152312648 152312712	E3: SD card Int/Ext i <Name> CFG

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	3	10..14	12	Error while reading/writing the configuration data from/to the SD card, error while reading the file	Check memory card	152312460 152312524 152312588 152312652 152312716	E3: SD card Int/Ext i <Name> CFG
E3	9	20	3	10..14	13	Error while reading/writing the configuration data from/to the SD card, error while writing the file	Check memory card	152312461 152312525 152312589 152312653 152312717	E3: SD card Int/Ext i <Name> CFG
E3	9	20	3	10..14	17	Error while reading/writing the configuration data from/to the SD card, error while accessing the Communication Module	Check memory card and hardware configuration	152312465 152312529 152312593 152312657 152312721	E3: SD card Int/Ext i <Name> CFG

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	3	10.. 14	31	Error while reading/ writing the configuration data from/to the SD card, file does not match the Communication Module type	Check memory card and hardware configuration	152 312 479 152 312 543 152 312 607 152 312 671 152 312 735	E3: SD card Int/Ext i <Name> CFG
E3	9	20	31	0	2	File does not exist	Check memory card	152 369 154	E3: SD card Bootcode
E3	9	20	31	0	8	Error: Invalid file	Check memory card	152 369 160	E3: SD card Bootcode
E3	9	20	31	0	13	Error while updating the bootcode. Perhaps bootcode is invalid or some errors occurred while loading the bootcode into the PLC.	Check memory card, reload	152 369 165	E3: SD card Bootcode
E3	9	20	31	0	19	Error while updating the bootcode. Checksum error has occur while loading the new bootcode into the PLC.	Check memory card, reload	152 369 171	E3: SD card Bootcode
E3	9	20	31	0	40	Version is not supported, e.g. obsolete firmware	Check firmware version, update to latest version	152 369 192	E3: SD card Bootcode
E3	9	20	31	0	43	Other error, e. g. not enough memory or file system error	Update CPU firmware	152 369 195	E3: SD card Firmware
E3	9	20	31	1	2	File does not exist	Check memory card	152 369 218	E3: SD card Firmware
E3	9	20	31	1	8	Error: Invalid file	Check memory card	152 369 224	E3: SD card Firmware

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	31	1	13	SD card errors: SDCARD.INI on SD card is missing, default was generated Copying the boot project from the SD card failed (may be that there is no valid boot project) Creating of the boot project failed (may be that there is no valid boot project)	Check memory card	152 369 229	E3: SD card Firmware
E3	9	20	31	1	40	Version is not supported, e.g. obsolete firmware	Check firmware version, update to latest version	152 369 256	E3: SD card Firmware
E3	9	20	31	1	41	Copying the firmware from the PLC to the SD card failed	Check memory card	152 369 257	E3: SD card Firmware
E3	9	20	31	1	43	Other error, e .g. not enough memory or file system error	Update CPU firmware	152 369 259	E3: SD card Firmware
E3	9	20	31	4	12	Copying the boot project and/or (at least) the Communication Module configuration from PLC to SD card failed	Check memory card	152 369 420	E3: SD card Bootproject
E3	9	20	31	4	13	Copying the boot project and/or (at least) the Communication Module configuration from PLC to SD card failed	Check memory card	152 369 421	E3: SD card Bootproject
E3	9	20	31	5	13	Error while writing the source file	Check memory card	152 369 485	E3: SD card Sourcecode
E3	9	20	31	7	8	Error image update, file could not be found	Check memory card	152 369 608	E3: SD card Image
E3	9	20	31	7	12	Error while copying the boot project, retains, persistent and/or (at least) Communication Module from SD card to PLC	Check memory card	152 369 612	E3: SD card Image
E3	9	20	31	7	13	Error while updating the image, perhaps firmware is defective or an error occurred while writing the new image	Check memory card and firmware version, retry	152 369 613	E3: SD card Image

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	9	20	31	7	19	Error while updating the image. Checksum error has occur while loading the new image into the PLC	Check memory card and firmware version, retry	152 369 619	E3: SD card Image
E3	9	20	31	7	43	Other error, e. g. not enough memory or file system error	Update CPU firmware	152 369 643	E3: SD card Image
E3	9	20	31	29	0	Error while updating the image of (at least) 1 Communication Module	Check memory card and FW version, retry	152 371 008	E3: SD card Firmware Communication Module
E3	9	21	31	0	17	Display could not be installed	Check FW	152 434 705	E3:
E3	15	1	2	0	26	Different configuration values at function block HA_CS31_CONTROL in CPU A and CPU B	Correct Application -> HA_CS31_CONTROL	-	-
E3	15	1	3	0	26	Different configuration values at function block HA_DATA_SYNC in CPU A and CPU B	Correct Application -> HA_DATA_SYNC	-	-
E4	1	255	30	2	39	More than one instance of SF_WDOG_TIME_SET or SF_MAX_POWER_DIP_SET	Correct user program	184 545 447	E4:
E4	9	0	2	2	9	Diagnosis buffer full	Clear all errors	150 999 177	E4:
E4	9	0	2	2	10	Error while copying web visualization/IEC RAM disc, RAM disc to small	Make web visualization/IEC program smaller, clean up user RAM disk	150 999 178	E4:
E4	9	0	2	2	20	Program not started because of an existing error (see PLC configuration CPU parameters, stop on error class)	Eliminate error and acknowledge	150 999 188	E4:
E4	9	0	2	2	37	Cycle time exceeded, but shorter than watchdog time	Adapt task configuration	150 999 205	E4:
E4	9	0	2	2	43	Control system was restarted by EC2 or power dip according to PLC configuration		150 999 211	E4:

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	0	2	3	17	Internal Error	Contact ABB Technical Support	150 999 249	
E4	9	0	2	3	54	FlexConfID is greater than the number of alternative configuration files.	Change FlexConf ID	150 999 286	E4:
E4	9	0	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151 001 116	E4:
E4	9	0	3	1	40	Boot code versions V1.1.3. (or older versions) support smaller RAM disk	Update boot code to 1.2.0	151 001 192	E4:
E4	9	1..6/ 10	1	0	32	Installation of the Communication Module driver failed, unknown Communication Module type	Check configuration	151 652 384	E4: Ext. [1..6] [COUPLER] E4: Int. [COUPLER]
E4	9	1..6/ 10	2	0	3	Within the specified time, connection could not be established to all of the slaves, I/O data may be (partly) invalid	Check slave for existence or set times according to the slave behavior	151 654 403	E4: Ext. [1..6] [COUPLER] E4: Int. [COUPLER]
E4	9	1..6/ 10	2	0	4	No socket available	Check communication module settings with PLC browser	151 654 404	E4: Ext. [1..6] [COUPLER] E4: Int. [COUPLER]
E4	9	1..6	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151 656 476	E4: Ext. [1..6] [COUPLER]
E4	9	1..6	3	0	40	Selected configuration is not supported by the current coupler firmware.	Update the communication module firmware so that this configuration can be used	151 656 488	E4: Ext. [1..6] [COUPLER]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	8	2	0	52	Interrupt lost. This happens when some input channels are configured as Interrupt on rising edge or Interrupt on falling edge and the function block ONB_IO_INT_IN is not called fast enough to acknowledge incoming interrupts events.	Check Task configuration and PLC program	151523380	E4:
E4	9	10	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151656476	E4: Int. [COUPLER]
E4	9	11/12	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151722012	E4: COM1
E4	9	13	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151722012	E4: FBP
E4	9	17	2	3	10	Web visualization exceeds size of RAM disk	Build smaller web visualization	152113354	E4:
E4	9	17	2	4	10	Boot project exceeds size of RAM disc	Build smaller boot project	152113418	E4:
E4	9	17	3	3	10	Web visualization exceeds size of user RAM disk	Build smaller web visualization	152115402	E4:
E4	9	17	3	3	8	Incomplete download of web visualization	Try to download web visualization again	152115400	E4:
E4	9	17	3	3	14	Failed to cleanup unused parts of web visualization	not available	152115406	E4:
E4	9	18	1	0	17	PLC Configuration file could not be read	Reload project	152176657	E4:
E4	9	18	2	0	8	Failed to write to / read from SD card	Check write protection and file system consistency	152178696	E4:

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	18	2	4	10	Boot project exceeds size of flash	Build smaller boot project	152 178 954	E4:
E4	9	18	31	3	12	Failed to read web visualization, power-off while web visualization has been flashed	Recreate web visualization	152 238 284	E4:
E4	9	18	31	4	12	Failed to read boot project from flash; power-off while boot project has been flashed	Recreate boot project	152 238 348	E4:
E4	9	20	1	2	2	Invalid value for FunctionOfCard in SDCARD.INI, value will be ignored	Check file SDCARD.INI on the memory card	152 307 842	E4: SD card
E4	9	20	1	2	13	Error of the key actualization of the SDCARD.ini file	Check memory card	152 307 853	E4: SD card
E4	9	20	1	10.. 14	8	Error Firmware update of the SD card, file could not be opened	Check memory card	152 308 360 152 308 424 152 308 488 152 308 552 152 308 616	E4: SD card
E4	9	20	1	10.. 14	12	Error Firmware update of the SD card, error while reading the file	Check memory card	152 308 364 152 308 428 152 308 492 152 308 556 152 308 620	E4: SD card

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	1	10..14	13	Error Firmware update of the SD card, error while writing the file	Check CPU + communication module	152308365 152308429 152308493 152308557 152308621	E4: SD card
E4	9	20	1	10..14	17	Error Firmware update of the SD card, error while accessing the Communication Module	Check communication module	152308369 152308433 152308497 152308561 152308625	E4: SD card
E4	9	20	1	10..14	31	Error Firmware update of the SD card, file does not match the Communication Module type	Check memory card	152308383 152308447 152308511 152308575 152308639	E4: SD card
E4	9	20	1	31	3	static error - Failed to deinit filesystem stack for SD card!	PLC restart required	152309699	E4: SD card

Error class	Com-ponent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	1	31	17	static error - SD card removed during access!	PLC restart required	152 309 713	E4: SD card
E4	9	20	2	4	13	Failed to copy boot project to SD Card	Check memory card for free space	152 310 029	E4: SD card
E4	9	20	3	10.. 14	1	Error updating the Communication Module configuration, file does not match the Communication Module type	Check memory card	152 312 449 152 312 513 152 312 577 152 312 641 152 312 705	E4: SD card CFG Int. E4: SD card Ext.1..4 CFG
E4	9	20	3	10.. 14	8	Error updating the Communication Module configuration, file could not be found	Check memory card	152 312 456 152 312 520 152 312 584 152 312 648 152 312 712	E4: SD card CFG Int. E4: SD card Ext.1..4 CFG

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	3	10..14	27	Error updating the Communication Module configuration, error while accessing the Communication Module	Check CPU + communication module	152 312 475 152 312 539 152 312 603 152 312 667 152 312 731	E4: SD card CFG Int. E4: SD card Ext.1..4 CFG
E4	9	20	3	10..14	35	Error updating the Communication Module configuration error while reading the file	Check communication module	152 312 483 152 312 547 152 312 611 152 312 675 152 312 739	E4: SD card CFG Int. E4: SD card Ext.1..4 CFG
E4	9	20	3	10..14	36	Error updating the Communication Module configuration, error while writing the file	Check memory card	152 312 484 152 312 548 152 312 612 152 312 676 152 312 740	E4: SD card CFG Int. E4: SD card Ext.1..4 CFG

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	3	20..24	8	Error while reading/writing the configuration data from/to the SD card, file could not be opened	Check memory card	152313096 152313160 152313224 152313288 152313352	E4: SD card
E4	9	20	3	20..24	12	Error while reading/writing the configuration data from/to the SD card, error while reading the file	Check memory card	152313100 152313164 152313228 152313292 152313356	E4: SD card
E4	9	20	3	20..24	13	Error while reading/writing the configuration data from/to the SD card, error while writing the file	Check memory card	152313101 152313165 152313229 152313293 152313357	E4: SD card

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	3	20..24	17	Error while reading/ writing the configuration data from/to the SD card, error while accessing the Communication Module	Check memory card and hardware configuration	152 313 105 152 313 169 152 313 233 152 313 297 152 313 361	E4: SD card
E4	9	20	3	20..24	31	Error while reading/ writing the configuration data from/to the SD card, file does not match the Communication Module type	Check memory card and hardware configuration	152 313 119 152 313 183 152 313 247 152 313 311 152 313 375	E4: SD card
E4	9	20	31	0..15	0..15	Communication Module update failed, error message of the Communication Module: Channel = Bit 4..7; Error = Bit 0..3	*Chapter 1.7.3.4 "Communication modules diagnosis" on page 6489*		
E4	9	20	31	0	2	Error while updating the bootcode, *.gza is incorrect or not supported	Check memory card, update CPU firmware	152 369 154	E4: SD card Bootcode
E4	9	20	31	0	8	Error while updating the bootcode, *.gza file could not be opened, e. g. file not found	Check memory card	152 369 160	E4: SD card Bootcode
E4	9	20	31	0	13	Error while reading/ writing the bootcode from/to the SD card, error while writing the file	Check memory card	152 369 165	E4: SD card Bootcode

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	31	0	19	Error while updating the bootcode, checksum failure of the written file	Update CPU firmware, check memory card	152 369 171	E4: SD card Bootcode
E4	9	20	31	0	40	Error while updating the bootcode, version is not supported by hardware	Update CPU firmware	152 369 192	E4: SD card Bootcode
E4	9	20	31	0	43	Error while updating the bootcode, internal error, e. g. not enough memory or file system error	Update CPU firmware	152 369 195	E4: SD card Bootcode
E4	9	20	31	1	2	Error while updating the firmware, *.gza is incorrect or not supported	Check memory card, update CPU firmware	152 369 218	E4: SD card Firmware
E4	9	20	31	1	8	Error while updating the firmware, *.gza file could not be opened, e. g. file not found	Check memory card	152 369 224	E4: SD card Firmware
E4	9	20	31	1	13	Error while reading/ writing the firmware from/to the SD card, error while writing the file	Check memory card	152 369 229	E4: SD card Firmware
E4	9	20	31	1	19	Error while updating the firmware, checksum failure of the written file	Check memory card, reload	152 369 235	E4: SD card Firmware
E4	9	20	31	1	40	Error while updating the firmware, version is not supported by hardware	Update CPU firmware	152 369 256	E4: SD card Firmware
E4	9	20	31	1	41	Error while exporting the firmware to the SD card	Check memory card, reload	152 369 257	E4: SD card Firmware
E4	9	20	31	1	43	Error while updating the firmware, internal error, e. g. not enough memory or file system error	Update CPU firmware	152 369 259	E4: SD card Firmware
E4	9	20	31	2	12	No SD card inserted or SDCARD.INI file not found	Check memory card	152 369 292	E4: SD card

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	31	3	13	SD card errors: ● SDCARD.INI on SD card is missing, default was generated ● Copying the boot project from the SD card failed ● Copying the boot project from the SD card failed (may be that there is no valid boot project) ● Creation of the boot project failed (may be that there is no valid boot project)	Check memory card	152 369 357	E4: SD card
E4	9	20	31	4	12	Error while exporting the user program and/or the Communication Module's configuration to the SD card	Check memory card, reload	152 369 420	E4: SD card Bootproject
E4	9	20	31	4	13	Error while importing the user program and/or the Communication Module's configuration from SD card	Check memory card, reload	152 369 421	E4: SD card Bootproject
E4	9	20	31	5	12	Error while importing the persistent data from the SD card	Check memory card, reload	152 369 484	E4: SD card Persistents
E4	9	20	31	5	13	Loading the source code failed	Check memory card, reload	152 369 485	E4: SD card Persistents
E4	9	20	31	6	12	Error while importing the retain data from the SD card	Check memory card, reload	152 369 548	E4: SD card Retains
E4	9	20	31	7	8	Error while updating the OEM-image, file not found	Check memory card	152 369 608	E4: SD card Image
E4	9	20	31	7	12	Error while updating the OEM-image, error while reading the file	Check memory card	152 369 612	E4: SD card Image
E4	9	20	31	7	13	Error while updating the OEM-image, error while writing the file	Check memory card	152 369 613	E4: SD card Image
E4	9	20	31	7	19	Error while updating the OEM-image, checksum failure	Check memory card, reload	152 369 619	E4: SD card Image

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	31	7	43	Error while updating the OEM-image, internal error (e. g. not enough memory)	Check memory card, update CPU firmware	152 369 643	E4: SD card Image
E4	9	20	31	8	1	Error Firmware update of the SD card, no version could be found at the sdcard.ini	Check memory card	152 369 665	E4: SD card Firmware MMI
E4	9	20	31	8	2	Error Firmware update of the SD card, file is incorrect or does not exist	Check memory card	152 369 666	E4: SD card Firmware MMI
E4	9	20	31	8	8	Error Firmware update of the SD card, file could not be opened	Check memory card	152 369 672	E4: SD card Firmware MMI
E4	9	20	31	8	27	Error Firmware update, wrong PLC mode (RUN)	Check PLC mode	152 369 691	E4: SD card Firmware MMI
E4	9	20	31	8	35	Error Firmware update of the SD card, error while reading the file	Check memory card	152 369 699	E4: SD card Firmware MMI
E4	9	20	31	8	36	Error Firmware update of the SD card, error while writing the file	Check memory card	152 369 700	E4: SD card Firmware MMI
E4	9	20	31	8	40	Error Firmware update of the SD card, PLC version does not support the update	Update CPU firmware	152 369 701	E4: SD card Firmware MMI
E4	9	20	31	9	1	Error Firmware update of the SD card, no version could be found at the sdcard.ini	Check memory card	152 369 729	E4: SD card Firmware IO
E4	9	20	31	9	2	Error Firmware update of the SD card, file is incorrect or does not exist	Check memory card	152 369 730	E4: SD card Firmware IO
E4	9	20	31	9	8	Error Firmware update of the SD card, file could not be opened	Check memory card	152 369 736	E4: SD card Firmware IO
E4	9	20	31	9	12	Error Firmware update of the SD Card, file could not be read	Check memory card	152 369 740	E4: SD card Firmware IO
E4	9	20	31	9	17	Error Firmware update, wrong PLC mode (RUN)	Check PLC mode	152 369 745	E4: SD card Firmware IO
E4	9	20	31	9	36	Error Firmware update of the SD card, error while writing the file	Check memory card	152 369 764	E4: SD card Firmware IO

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	20	31	9	40	Error Firmware update of the SD card, PLC version does not support the update	Update CPU firmware	152 369 768	E4: SD card Firmware IO
E4	9	20	31	9	43	Other error, e. g. not enough memory or file system error	Update CPU firmware	152 369 771	E4: SD card Firmware IO
E4	9	20	31	29	0	Error while updating the image of (at least) 1 Communication Module	Check memory card and FW version, retry	152 371 008	E4: SD card Firmware Communication Module
E4	9	20	31	41	0	Error Firmware update of the Communication Module	Update CPU firmware, reload	152 369 728	E4: SD card Firmware Coupler
E4	9	21	31	1	2	Error display firmware	Update display firmware	152 434 754	E4:
E4	9	22	31	31	8	Missing or exhausted battery	Insert battery or set parameter "Check Battery" to "Off"	152 502 216	E4:
E4	9	22	31	31	53	Battery status changed		152 502 261	E4:
E4	9	25	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	152 639 516	E4:
E4	9	27	1	2	0	PLC performed reboot due to power dip	Check PLC's power supply	152 766 592	E4:
E4	9	29	31	0	2	RUN/STOP switch in STOP position and trying to run user program via CODESYS	Check position of RUN/STOP switch	152 502 216	E4:
E4	9	33	31	1	6 8 15	Startup/Restart of flashdisk failed	Check flash disk	153 221 190 153 221 192 153 221 199	E4:
E4	9	33	31	2	43	No filesystem found	Format flash disk manually (PBC)	153 221 291	E4:
E4	9	33	31	3	43	Format failed	Check flash disk	153 221 355	E4:

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	33	31	4	43	Writing MBR failed	Check flash disk	153 221 419	E4:
E4	9	33	31	5	43	Flashdisk has aged (close to end of life)	Prepare replacement of CPU	153 221 483	E4:
E4	9	33	31	6	43	Flashdisk has reached end of life, switched to read-only mode	Replace CPU	153 221 547	E4:
E4	9	34	31	2	43	SRAM disk has been formatted		153 286 827	E4:
E4	9	255	30	2	39	More than one instance of SF_WDOG_TIME_SET or SF_MAX_POWER_DIP_SET	Correct user program	167 768 231	E4:

Table 771: Diagnosis messages of the file system

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	17	31	2	43	No filesystem found	Contact support	152 172 715	--
E4	9	17	31	3	43	Formating memory location failed	Contact support	152 172 779	--
E4	9	20	31	2	43	No filesystem found	Contact support	152 369 323	--
E4	9	20	31	3	43	Formating memory location failed	Contact support	152 369 387	Online text --
E4	9	33	31	1	8	No filesystem found	Contact support	153 221 192	Startup/Restart of flashdisk failed
E4	9	33	31	1	15	Could not create drive instance	Contact support	153 221 199	Startup/Restart of flashdisk failed
E4	9	33	31	1	6	Can't open device driver	Contact support	153 221 190	Startup/Restart of flashdisk failed
E4	9	33	31	1	43	Unknown	Contact support	153 221 227	--

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	9	33	31	2	43	No filesystem found	Contact support	153 221 291	--
E4	9	33	31	3	43	Formating memory location failed	Contact support	153 221 355	Format failed -> Check flashdisk
E4	9	33	31	5	43	Flashdisk has aged	Contact support	153 221 483	Flashdisk has reached end of life, switched to read-only mode -> Replace CPU
E4	9	33	31	6	43	Flashdisk is at end of (write) lifetime	Contact support	153 221 547	--
E4	9	34	31	2	43	No filesystem found	Contact support	153 286 827	SRAM disk has been formatted
E4	9	34	31	3	43	Formating memory location failed	Contact support	153 286 891	--

Table 772: Diagnosis messages of the I/O bus

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	14	1..10	31	1	34	Timeout while initializing an I/O Module	Replace module	234 944 610	E2: I/O-Bus, Mod. [1..10]
E2	14	1..10	31	3	27	No configuration available	Please contact AC500 support	234 944 731	E2: I/O-Bus, Mod. 1..10
E2	14	1..10	31	4	33	Timeout while waiting for Reset	Check I/O module	234 944 801	E2: I/O-Bus, Mod. [1..10]
E2	14	1..10	31	4	42	Failure of the module, more than the max. permissible communication errors have occurred in sequence	Check module, FW version (PLC-browser: I/O bus desc)	234 944 810	E2: I/O-Bus, Mod. [1..10]
E2	14	255	3	0	27	No configuration available	Please contact AC500 support	251 598 875	E2: I/O-Bus

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	31	1	32	Master and module could not agree on any protocol variant, no variant found which is supported by both the master and the module	Check FW version CPU / I/O module	234 944 608	E3: I/O-Bus, Mod. [1..10]
E3	14	1..10	31	3	26	Configuration error PLC configuration master	Check configuration	234 944 730	E3: I/O-Bus, Mod. [1..10]
E3	14	255	2	0	3	Timeout while updating the I/O data at the program start	Check FW version CPU / I/O modules	251 596 803	E3: I/O-Bus
E3	14	255	2	0	26	Program was not started because of configuration error of the I/O-Bus	Check configuration	251 596 826	E3: I/O-Bus
E3	14	255	3	0	26	Configuration error PLC configuration master	Check configuration	251 598 874	E3: I/O-Bus
E4	14	0.. 254	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	234 881 052	E4: I/O-Bus, Mod. 0..254, 3, 0
E4	14	1..10	31	1	34	Timeout during parameterization	Check FW version CPU / I/O modules	234 944 610	E4: I/O-Bus, Mod. [1..10]
E4	14	1..10	31	1	35	Timeout during parameterization	Check FW version CPU / I/O modules	234 944 611	E4: I/O-Bus, Mod. [1..10]
E4	14	1..10	31	31	44	Module has not passed factory test	Replace module	234 946 540	E4: I/O-Bus, Mod. [1..10]
E4	14	255	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	251 598 876	E4: I/O-Bus

Table 773: Diagnosis messages of the onboard I/O modules

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	8	0..7	3	0	27	No configuration available	Please contact AC500 support	134 223 899 134 289 435 134 354 971 134 420 507 134 486 043 134 551 579 134 617 115 134 682 651	"E2: Onboard I/O E2: Onboard I/O PWM E2: Onboard I/O IRQ E2: Onboard I/O FC E2: Onboard I/O AI E2: Onboard I/O AO E2: Onboard I/O E2: Onboard I/O
E2	8	255	3	0	27	No configuration available	Please contact AC500 support	150 935 579	E2: Onboard I/O
E3	8	255	2	0	3	MaxWaitRun for onboard I/O Module has expired, when PLC is put into RUN state	Reboot and try it again. If the error still exists, replace CPU for testing	150 933 507	E3: Onboard I/O Ch: 0
E3	8	255	3	0	26	Invalid configuration of Onboard I/O Module, e. g. 2 input channels are configured as fast counter and interrupt input at the same time.	Correct PLC configuration	150 935 578	E3: Onboard I/O Ch: 0
E3	9	8	2	0	52	Frequency on interrupt input pin too high and interrupt events are missed	Correct frequency	150 933 556	E3: Onboard I/O Ch: 0
E4	8	1	2	0..1	4	PWM channel frequency or cycle time too high	Correct frequency or cycle time	134 287 364 134 287 428	E4: Onboard I/O PWM Ch: 0/1

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	8	1	2	0..1	7	PWM channel frequency or cycle time too low	Correct frequency or cycle time	134 287 367 134 287 431	E4: Onboard I/O PWM Ch: 0/1
E4	8	1	2	0	52	Frequency at interrupt input too high	Check task configuration and PLC program	151 523 380	E3: Onboard I/O
E4	8	1	2	1	2	Invalid configuration value for PWM channel. Frequency / cycletime for the PWM channel of the 8DI+6DO and 8DI+6DO+2AI+1AO module are common and if both channel are configured for PWM, the frequency of the second channel must be set to 0.	Correct frequency	134 287 426	E4: Onboard I/O PWM Ch: 1
E4	8	4	2	0..1	48	Analog input value too high	Correct value	134 484 016 134 484 080	E4: Onboard I/O AI Ch: 0/1
E4	8	5	2	0	48	Analog output value too high	Correct value	134 549 552	E4: Onboard I/O AO Ch: 0
E4	8	255	0	0	43	Unspecified or internal error occured	Replace CPU	150 929 451	E4: Onboard I/O Ch: 0
E4	8	255	2	0	26	PLC was put into RUN state, although a configuration error is present, because parameter Run on config fault is set to YES	Correct PLC configuration	150 933 530	E4: Onboard I/O Ch: 0
E4	9	8	3	0	28	Different configuration -> old firmware with new Automation Builder or new firmware with old Automation Builder	Update CPU firmware	151 525 404	E4: Onboard I/O

Table 774: Diagnosis messages of the communication module interface [1])

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	1..6	11/12	3	0	27	Error configuration	Check configuration	167 772 187	E2: Ext. [1..6] [COUPLER], [COUPLER] 11..12, 3, 0
E2	1..6/ 10	255	3	0	2	Same Node ID twice in the net	Use node IDs only once	184 489 986	E2: Ext. [1..6] [COUPLER] E2: Int. [COUPLER]
E3	1..6	0.. 254	31	31	20	Slave-To-Slave communication fails	Check configuration and version	167 837 652	E3: Ext. [1..6] [COUPLER]
E3	1..6	0.. 254	31	31	41	Error on distributed clocks	Check configuration and version	167 837 673	E3: Ext. [1..6] [COUPLER]
E3	1..6/ 10	255	3	0	26	Incorrect or missing Communication Module configuration. Setting of IP parameters failed.	Configure communication module. Check the IP.	184 490 010	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	1..6/ 10	255	5	0.. 15	0..15	Error message of the operating system of the Communication Module: Channel = Bit 4..7 Error = Bit 0..3	↳ *Chapter 1.7.3.4 “Communication modules diagnosis” on page 6489*	184 494 080	E3: Ext. [1..6] [COUPLER] OS E3: Int. [COUPLER] OS
E3	1..6/ 10	255	6.. 12	0.. 15	0..15	Error message of the task x of the Communication Module: Task x = 'Module' - 5 Channel = Bit 4..7 Error = Bit 0..3	↳ *Chapter 1.7.3.4 “Communication modules diagnosis” on page 6489*	184 496 128	E3: Ext. [1..6] [COUPLER] Task [1..7] E3: Int. [COUPLER] Task [1..7]
E3	1..6/ 10	255	31	0	33	Timeout while waiting for reset of the Communication Module	Check communication module	184 547 361	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	1..6/ 10	255	31	0	34	Timeout while waiting for readiness of the Communication Module	Check communication module	184 547 362	E3: Ext. [1..6] [COUPLER] E3: Int. [COUPLER]
E3	10	255	1	1	25	Error timeout DHCP (≥ 30 s); fallback IP set by configurator	Check network (IP address, wiring, ...), check DHCP server	184 485 977	E3: Int. [COUPLER]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	10	255	3	0	28	Error: changing IP parameter while using this interface; no IP parameters can be changed if the device is in use	Log out, log in and set to run	184 490 012	E3: Int. [COUPLER]
E3	10	255	4	5	25	SNTP: Error - Timeout, no answer in PRESync timeout, program does not start	Check network (IP address, wiring, ...), change PREsync timeout in configuration	184 492 377	E3: Int. [COUPLER]
E3	10	255	4	5	38	SNTP: Error - No answer from server (normal and backup), try again	Check network (IP address, wiring, ...)	184 492 390	E3: Int. [COUPLER]
E4	1..6/ 10	0..7	31	31	47	Short-circuit Communication Module/ channel	Fix short-circuit	167 837 679	E4: Ext. [1..6] [COUPLER], [COUPLER] [1..7] E4: Int. [COUPLER], [COUPLER] [1..7]
E4	1..6/ 10	0.. 254	31	0.. 15	0..15	Communication error to the slave, Channel = Bit 4..7 Error = Bit 0..3	*Chapter 1.7.3.4 "Communication modules diagnosis" on page 6489*	167 835 648	E4: Ext. [1..6] [COUPLER], [COUPLER] [1..254] E4: Int. [COUPLER], [COUPLER] [1..254]
E4	1..6	2	31	1	1	PROFIBUS Slave device <device number>: connection lost Error initalization of slave configuration	Check connection Check configuration values of slave	167 835 713	E4: PROFIBUS Slave device 4 E4: Ext. [1..6] [COUPLER] , [COUPLER] 2
E4	1..6	2	31	1	14	Error initialization of slave configuration	Check slave configuration	167 835 726	E4: Ext. [1..6] [COUPLER] , [COUPLER] 2
E4	1..6	2	31	2	2	Runtime error of slave	Check slave configuration	167 835 778	E4: Ext. [1..6] [COUPLER] , [COUPLER] 2
E4	1..6/ 10	255	2	0.. 15	0..15	Communication error of the Communication Module, Channel = Bit 4..7 Error = Bit 0..3	*Chapter 1.7.3.4 "Communication modules diagnosis" on page 6489*	184 487 936	E4: Ext. [1..6] [COUPLER] E4: Int. [COUPLER]
E4	10	255	4	5	3	SNTP: Error finding connection to server; tried 5 times to get connection	Check network (IP address, wiring, ...)	184 492 355	E4: Int. [COUPLER]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	10	255	4	5	10	SNTP: Error time; time difference makes time jump necessary, but too big / not allowed	Set time manually or allow bigger time jumps	184 492 362	E4: Int. [COUPLER]
E4	10	255	4	5	34	SNTP: Could not sync time before configured timeout	Check connection to time server and/or configure bigger timeout	184 482 386	E4: Int. [COUPLER]

[1]) Communication module errors: 1 = UDP DataExchange, 2 = UDP no AC31 Header, 3 = Online Protocol, 4 = Netconfig, 5 = SNTP, 6 = Modbus

Table 775: Diagnosis messages of the serial interfaces

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E2	11.. 13	255	31	0	17	Access errors: ● Interface could not be closed ● Interface could not be opened ● Timeslotmode could not be activated	Check FW version, replace CPU if necessary	201 324 561 234 878 993	E2: COM [1..2] E2: FBP
E4	13	255	4	0	42	Receiving error or timeout of the FBP slave interface		234 823 722	E4: FBP

Table 776: Diagnosis messages of the CS31 bus (COM1 = CS31 master)

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	11	0..61	0..5	0	8	No module found on the CS31 bus	Adapt configuration	184 549 384	E3: COM1 [PROTOCOL] [0..61], [0..5], 0
E3	11	0..61	1..8	0.. 31	0..63	S500 class 3 diagnostic sent by DC551	↳ *Chapter 1.7.3.3 “S500 I/O modules diagnosis” on page 6472*	184 549 376 to 184 549 439	E3: COM1 [PROTOCOL] [0..61], [1..8], [0..30 | (≥) 31]
E3	11	255	1	0	8	No module found on the CS31 Bus	Check configuration	201 263 112	E3: COM1

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	11	0..61	0..5	0	8	ICMK 14 with extensions configured, the extensions was not found on the bus	Check configuration	184 549 384	E4: COM1 [PROTOCOL] [0..61], [0..5], 0
E4	11	0..61	0..5	0	28	Module discarded and registered again; is only reported at the start		184 549 404	E4: COM1 [PROTOCOL] [0..61], [0..5], 0
E4	11	0..61	0..5	0	32	Not configured module found on the bus, again discarded	Check CS31 bus, insert CS31_DIAG function block into the project	184 549 408	E4: COM1 [PROTOCOL] [0..61], [0..5], 0
E4	11	0..61	0..5	0.. 15	1	Internal error (error 1), reported by an AC31 I/O module	Check module	184 549 377	E4: COM1 [PROTOCOL] [0..61], [0..5], [0..15]
E4	11	0..61	0..5	0.. 15	28	Configured module does not match the module registered to the bus	Check PLC configuration, insert CS31_DIAG function block into the project	184 549 404	E4: COM1 [PROTOCOL] [0..61], [0..5], [0..15]
E4	11	0..61	0..5	0.. 15	32	Not configured module registered to the bus	Check PLC configuration, insert CS31_DIAG function block into the project	184 549 408	E4: COM1 [PROTOCOL] [0..61], [0..5], [0..15]
E4	11	0..61	0..5	0.. 15	47	Short-circuit on CS31 module/channel	Fix short-circuit	184 549 423	E4: COM1 [PROTOCOL] [0..61], [0..5], [0..15]
E4	11	0..61	0, 2, 4	0.. 15	2	Cut wire (Error 2) of an AC31 I/O module	Remove error at the module or the channel	184 549 378	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]
E4	11	0..61	0, 2, 4	0.. 15	4	Overload (Error 4) of an AC31 I/O module		184 549 380	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]
E4	11	0..61	0, 2, 4	0.. 15	6	Overload + cut wire (Error 6) of an AC31 I/O module		184 549 382	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]
E4	11	0..61	0, 2, 4	0.. 15	10	Short-circuit + cut wire or "out of range" at analog modules (Error 10) of an AC31 I/O module		184 549 386	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]
E4	11	0..61	0, 2, 4	0.. 15	12	Overload + short-circuit (Error 12) of an AC31 I/O module		184 549 388	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	11	0..61	0, 2, 4	0.. 15	14	Short-circuit + overload + cut wire (Error 14) of an AC31 I/O module		184 549 390	E4: COM1 [PROTOCOL] [0..61], [0|2|4], [0..15]
E4	11	0..61	1, 3, 5	0.. 15	3	Analog value exceeded (Error 3) of an AC31 I/O Module		184 549 379	E4: COM1 [PROTOCOL] [0..61], [1|3|5], [0..15]
E4	11	0..61	1, 3, 5	0.. 15	9	Cut wire (Error 9) of an AC31 analog module		184 549 385	E4: COM1 [PROTOCOL] [0..61], [1|3|5], [0..15]
E4	11	0..61	1..8	0.. 31	0..63	E4 error messages of DC551 and S500 I/O Modules, see table below	↳ *Chapter 1.7.3.3 "S500 I/O modules diagnosis" on page 6472*	184 549 376	E4: COM1 [PROTOCOL] [0..61], [1..8], [0..30 | (≥) 31]
E4	11	0..61	31	31	9	Impossible configuration DC551 and S500 I/O Modules (too many I/Os in one cluster)	Change configuration	184 614 857	E4: COM1 [PROTOCOL] [0..61]
E4	11	0..61	31	31	31	Impossible configuration DC551 and S500 I/O Modules (too many parameters in one cluster)	Change configuration	184 614 879	E4: COM1 [PROTOCOL] [0..61]
E4	11	0..61	31	31	34	Outputs are written before the configuration of the modules DC551 + S500 I/O have been finished		184 614 882	E4: COM1 [PROTOCOL] [0..61]

1.7.3.3 S500 I/O modules diagnosis

Table 777: S500 I/O module errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	1	0..7	4	Measurement overflow at the I/O module	Check channel wiring and sensor power supply	234881028	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549380	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	31	31	3	Timeout in the I/O Module	Replace I/O module	234946499	E3: I/O-Bus, Mod. [1..10]
	11 / 12	ADR	1..10					184549379	E3: COM [1|2] [PROTOCOL] [ADR], [1..10], (≥)31
E3	14	1..10	31	31	9	Overflow diagnosis buffer	Restart	234946505	E3: I/O-Bus, Mod. [1..10]
	11 / 12	ADR	1..10					184549385	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E3	14	1..10	31	31	10	Process voltage too high	Check process voltage	234946506	E3: I/O-Bus, Mod. 1..10
	1..4	ADR	1..10						
E3	14	1..10	31	31	11	Process voltage too low	Check process voltage	234946507	E3: I/O-Bus, Mod. [1..10]
	11 / 12	ADR	1..10					184549387	E3: COM [1|2] [PROTOCOL] [ADR] [1.. 10], (≥) 31
E3	14	1..10	31	31	18	Plausibility check failed (iParameter)	Check configuration	234946514	E3: I/O-Bus, Mod. 1..10
	1..4	ADR	1..10						
E3	14	1..10	31	31	19	Checksum error in the I/O Module	Non-safety I/O: Replace I/O Module Safety-I/O: Check safety configu-	234946515	E3: I/O-Bus, Mod. [1..10]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
	11 / 12	ADR	1..10				ration and CRCs for iParameters and F-Parameters	184549395	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥) 31
E3	14	1..10	31	31	20	PROFIsafe communication error	Restart I/O Module. If this error persists, contact ABB technical support.	234946516	E3: I/O-Bus, Mod. 1..10
	1..4	ADR	1..10						
E3	14	1..10	31	31	25	PROFIsafe watchdog timed out	Restart I/O Module. If this error persists, increase PROFIsafe watchdog time.	234946521	E3: I/O-Bus, Mod. 1..10
	1..4	ADR	1..10						
E3	14	1..10	31	31	26	Parameter value	Check master or configuration	234946522	E3: I/O-Bus, Mod. [1.. 10]
	11 / 12	ADR	1..10					184549402	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E3	14	1..10	31	31	28	F-Parameter configuration and address switch value do not match	Check I/O Module's F-Parameter configuration and module address switch value	234946524	E3: I/O-Bus, Mod. 1..10
	1..4	ADR	1..10						
E3	14	1..10	31	31	36	Internal data interchange disturbed	Replace I/O Module	234946532	E3: I/O-Bus, Mod. [1.. 10]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
	11 / 12	ADR	1..10					184549412	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E3	14	1..10	31	31	40	Different hardware and firmware versions in the module	Replace I/O Module	234946536	E3: I/O-Bus, Mod. [1.. 10]
	11 / 12	ADR	1..10					184549416	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E3	14	1..10	31	31	43	Internal error in the device	Replace I/O Module	234946539	E3: I/O-Bus, Mod. [1.. 10]
	11 / 12	ADR	1..10					184549419	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E3	14	1..10	31	31	47	Sensor voltage too low	Check sensor voltage	234946543	E3: I/O-Bus, Mod. [1.. 10]
	11 / 12	ADR	1..10					184549423	E3: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31
E4	14	1..10	31	31	45	Process voltage switched off (ON->OFF)	Process voltage ON	234946541	E4: I/O-Bus, Mod. [1.. 10]
	11 / 12	ADR	1..10					184549421	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], (≥)31

Table 778: Channel errors of the S500 I/O modules

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	14	1..10	1	0..7	1	Wrong measurement; wrong temperature at the compensations channel	Check temperature compensation channel	234881025	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549376	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	2	AI531: Wrong measurement; potential difference is to high CD522: PWM duty cycle out of duty area	AI531: Check potential difference CD522: Check min/max values	234881026	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549378	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	4	Measurement overflow at the I/O module	Check channel wiring and sensor power supply	234881028	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549380	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	7	Measurement underflow at the analog input	Check input value	234881031	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549383	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	4	1..10	1	0..7	10	Input/output value to high	Check input/ output value	234881034	E4: I/O-Bus, Mod. [1..10], 1, [0..7]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
	11 / 12	ADR	1..10					184549386	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	11	AI531:Output voltage 10 V to small CD522: Output voltage 5 V to small (sensor)	AI531: Check charge of the output voltage CD522: Check connection	234881035	E4: I/O-Bus, Mod. [1.. 10], 1, [0..7]
	11 / 12	ADR	1..10					184549387	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
4	14	1..10	1	0..7	18	Internal fuse to 0 V blown, 0 V not connected to ZP	Replace I/O Module	234881042	E4: I/O-Bus, Mod. [1.. 10], 1, [0..7]
	11 /12	ADR	1..10					184549394	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	47	Short-circuit at the analog input	Check terminal	234881071	E4: I/O-Bus, Mod. [1.. 10], 1, [0..7]
	11 / 12	ADR	1..10					184549423	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	1	0..7	48	Measurement overflow or cut wire at the analog input	Check input value and terminal	234881072	E4: I/O-Bus, Mod. [1..10], 1, [0..7]
	11 / 12	ADR	1..10					184549424	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	14	1..10	2	0..23	47	Short-circuit at the digital output	Check terminal	234881071	E4: I/O-Bus, Mod. [1..10], 2, [0..7]
	11 / 12	ADR	1..10					184549423	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	3	0..7	48	Measurement overflow at the analog output	Check output value	234881072	E4: I/O-Bus, Mod. [1..10], 3, [0..7]
	11 / 12	ADR	1..10					184549424	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	3	0..7	49	Short-circuit / Cut wire	Check terminal	234881073	E4: I/O-Bus, Mod. [1..10], 3, [0..7]
	11/12	ADR	1..10					184549425	E4: COM [1|2] [PROTOCOL] [ADR], [1.. 10], [0..7]
E4	14	1..10	3	0..7	7	Measurement underflow at the analog output	Check output value	234881031	E4: I/O-Bus, Mod. [1..10], 3, [0..7]
E4	14	1..10	31	5	8	I/O module removed from hot-swap terminal unit or defective module on hot-swap terminal unit [1)]	Plug I/O module, replace I/O module	234944840	E4: I/O-Bus, Mod. [1..10], 31, [5]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	14	1..10	31	5	28	Wrong I/O module plugged on hot-swap terminal unit [1])	Remove wrong I/O module and plug projected I/O module	234944860	E4: I/O-Bus, Mod. [1..10], 31, [5]
E4	14	1..10	31	5	42	No communication with I/O module on hot-swap terminal unit [1])	Replace I/O module	234944874	E4: I/O-Bus, Mod. [1..10], 31, [5]
E4	14	1..10	31	5	54	I/O module does not support hot swap [1]) [2])	Power off system and replace I/O module	234944886	E4: I/O-Bus, Mod. [1..10], 31, [5]
E4	14	1..10	31	6	8	Hot-swap terminal unit required but not found [1])	Replace terminal unit with hot-swap terminal unit and restart	234944904	E4: I/O-Bus, Mod. [1..10], 31, [6]
E4	14	1..10	31	6	42	No communication with hot-swap terminal unit [1])	Restart, if error persists replace terminal unit	234944938	E4: I/O-Bus, Mod. [1..10], 31, [6]

[1]) Diagnosis for hot swap available as of version index F0.

[2]) In case of an I/O module doesn't support hot swapping, do not perform any hot-swap operations (also not on any other terminal units (slots)) as modules may be damaged or I/O bus communication may be disturbed.

Table 779: Module errors DC551-CS31 and CI590

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	11	ADR	31	31	3	Timeout in the I/O Module	Replace I/O Module	184614851	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	9	Overflow of diagnosis buffer	Restart	184614857	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	11	Process voltage too low	Check process voltage	184614859	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	1..7	31	17	No communication with the I/O Module	Replace I/O Module	184549393	E3: COM1 [PROTOCOL] [ADR], [1..7], (≥)31
E3	11	ADR	31	31	19	Checksum error in the I/O Module	Replace I/O Module	184614867	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	26	Parameter error	Check master	184614874	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	28	Configuration of PLC 1 and PLC 2 are different	Check PLC CS31 Module configuration	184614876	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	36	Internal data interchange disturbed	Check PLC program	184614884	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	40	Different hardware and firmware versions in the module	Replace I/O Module	184614888	E3: COM1 [PROTOCOL] [ADR]
E3	11	ADR	31	31	43	Internal error in the module	Replace I/O Module	184614891	E3: COM1 [PROTOCOL] [ADR]

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	11	ADR	31	31	33	PLC conflict 3)	Check PLC program	184614881	E4: COM1 [PROTOCOL] [ADR]
E4	11	ADR	31	31	34	Outputs are different (synchronization error) 4)	Check PLC program and synchronize	184614882	E4: COM1 [PROTOCOL] [ADR]
E4	11	ADR	31	31	45	Process voltage ON/OFF	Process voltage ON	184614893	E4: COM1 [PROTOCOL] [ADR]
E4	11	ADR	31	31	32	Wrong I/O Module on the slot	Replace I/O Module and check configuration	184614882	E4: COM1 [PROTOCOL] [ADR], [1..10], (≥)31
			1..10					184549410	E4: COM1 [PROTOCOL] [ADR]
E4	11	ADR	31	31	34	No response during initialization of the I/O Module	Replace I/O Module	184614880	E4: COM1 [PROTOCOL] [ADR]
			1..10					184549408	E4: COM1 [PROTOCOL] [ADR], [1..10], (≥)31

[3]) FB HA control mode is both primary or both secondary. Could be only an error output of FB if too much diagnosis

[4]) if no other error occurs, the programs are not identical and parameter is set to report an error in this case

Table 780: Channel errors DC551-CS31 and CI590

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	11	ADR	1..10/31	8..23	47	Short-circuit at the digital output	Check terminal	184549423	E4: COM1 [PROTOCOL] [ADR] E4: COM1 [PROTOCOL] [ADR], [1..10], (≥)31
E4	11	ADR	1..10/31	8..23	7	Measurement underflow at the analog in/output	Check value	1846133383	E4: COM1 [PROTOCOL] [ADR] E4: COM1 [PROTOCOL] [ADR], [1..10], (≥)31

Table 781: Parameter errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	31	3	1	Wrong parameter value	Check configuration	234944705	
E3	14	1..10	31	3	19	Checksum error has occured in iParameter or F-Parameters	Check safety configuration and CRCs for iParameter and F-Parameters	234944723	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	3	28	F-Parameter configuration and address switch value do not match	Check I/O Module F-Parameter configuration and module address switch value	234944732	E3: I/O-Bus, Mod. 1

Table 782: Runtime errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	31	2	3	PROFIsafe watchdog timed out	Restart I/O Module. If this error persists, increase PROFIsafe watchdog time	234944643	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	19	Checksum error has occured in iParameters	Restart I/O Module. If this error persists, replace I/O module. Contact ABB technical support	234944659	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	20	PROFIsafe communication error	Restart I/O Module. If this error persists, contact ABB technical support	234944660	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	28	Internal error	Contact ABB technical support. Replace I/O Module	234944668	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	43	Internal runtime error in I/O module	Contact ABB technical support. Replace I/O module	234944683	E3: I/O-Bus, Mod. 1

Table 783: Background test errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	31	2	4	I/O Module overvoltage	Check I/O module power supply	234944644	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	7	I/O Module undervoltage	Check I/O module power supply	234944647	E3: I/O-Bus, Mod. 1
E3	14	1..10	31	2	18	Internal error	Restart I/O module. If this error persists, replace I/O module. Contact ABB technical support	234944658	E3: I/O-Bus, Mod. 1

Table 784: Specific errors AI, DI and DO

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	0	0..15	3	Discrepancy time expired	Check discrepancy time value, channel wiring and sensor	234881027	E3: I/O-Bus, Mod. 1-10, 0, 0-15
E3	14	1..10	0	0..15	12	Module 0: Test pulse error	Check wiring and sensor	234881036	E3: I/O-Bus, Mod. 1-10, 0, 0-15

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identi-fier	Error message	Remedy	Error number	Online text
			2	0..8		Module 2: Channel stuck-at error	Check I/O Module wiring. Restart I/O Module, if needed. If this error persists, replace I/O Module.		E3: I/O-Bus, Mod. 1..10, 2, 0..8
E3	14	1..10	0	0..15	13	Module 0: Channel test pulse cross-talk error	Check wiring and sensor. If this error persists, replace I/O Module. Contact ABB technical support	234881037	E3: I/O-Bus, Mod. 1-10, 0, 0-15
			2	0..7		Module 2: Channel readback error	Check I/O Module wiring. Restart I/O Module, if needed. If this error persists, replace I/O Module.		E3: I/O-Bus, Mod. 1..10, 2, 0..8

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identi-fier	Error message	Remedy	Error number	Online text
E3	14	1..10	0	0..15	18	Channel test error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O Module	234881042	E3: I/O-Bus, Mod. 1-10, 0, 0-15
E3	14	1..10	0	0..15	25	Channel stuck - at error	Check I/O Module wiring. Restart I/O Module, if needed. If this error persists, replace I/O Module.	234881049	E3: I/O-Bus, Mod. 1..10, 0, 0..15
E3	14	1..10	0	0..15	28	Channel cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O Module	234881052	E3: I/O-Bus, Mod. 1-10, 0, 0-15
E3	14	1..10	1	0..3	4	Measurement overflow at the I/O Module	Check channel wiring and sensor power supply	234881028	E4: I/O-Bus, Mod. 1..10, 1, 0..3

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	1	0..3	7	Measurement underflow at the I/O Module	Check channel wiring and sensor power supply	234881031	E3: I/O-Bus, Mod. 1..10, 1, 0..3
E3	14	1..10	1	0..3	45	Cut wire	Check channel wiring and sensor power supply	234881069	E3: I/O-Bus, Mod. 1, 1, 1 - 3
E3	14	1..10	1	0..3	51	Overload	Check channel wiring and sensor power supply	234881075	E3: I/O-Bus, Mod. 1, 1, 1 - 3
E3	14	1..10	1	0..3	55	Channel value difference too high	Adjust tolerance window for channels. Check channel wiring and sensor configuration.	234881079	E3: I/O-Bus, Mod. 1, 1, 1 - 3
E3	14	1..10	2	0..7	18	Channel cross-talk error	Check I/O Module wiring. Restart I/O Module, if needed. If this error persists, replace I/O Module.	234881042	E3: I/O-Bus, Mod. 1..10, 4, 0..8
E3	14	1..10	2	0..8	43	Internal error	Contact ABB technical support. Replace I/O Module.	234881067	E3: I/O-Bus, Mod. 1, 4, 0-7

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E3	14	1..10	4	0..7	13	Channel test error	Check I/O module wiring. Restart I/O Module, if needed. If this error persists, replace I/O module	234881037	E3: I/O-Bus, Mod. 1, 4, 0-7
E3	14	1..10	4	0..7	17	Channel test error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O Module	234881025	E3: I/O-Bus, Mod. 1, 4, 0-7

1.7.3.4 Communication modules diagnosis

1.7.3.4.1 Errors of the communication module's operating system

Table 785: General operating system errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	0	1	01hex = 1dec Error priority MAX	
E3	1..4/10	255	5	0	2	02hex = 2dec Error priority NULL	
E3	1..4/10	255	5	0	3	03hex = 3dec Error priority DOUBLE	
E3	1..4/10	255	5	0	4	04hex = 4dec Stack size error	
E3	1..4/10	255	5	0	5	05hex = 5dec EPROM size error	
E3	1..4/10	255	5	0	6	06hex = 6dec RAM size error	
E3	1..4/10	255	5	0	7	07hex = 7dec Segment counter error	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	0	8	08hex = 8dec Segment size error	
E3	1..4/10	255	5	0	9	09hex = 9dec Cycle time error	
E3	1..4/10	255	5	0	10	0Ahex = 10dec Fre-quency error	
E3	1..4/10	255	5	0	11	0Bhex = 11dec Trace buffer size error	
E3	1..4/10	255	5	0	12	0Chex = 12dec Error min. RAM	
E3	1..4/10	255	5	0	13	0Dhex = 13dec Device address error	
E3	1..4/10	255	5	0	14	0Ehex = 14dec MCL token error	
E3	1..4/10	255	5	0	15	0Fhex = 15dec Driver type error	
E3	1..4/10	255	5	1	0	10hex = 16dec SCC error	
E3	1..4/10	255	5	1	1	11hex = 17dec Flash type OPT error	
E3	1..4/10	255	5	1	2	12hex = 18dec Flash type BSL error	
E3	1..4/10	255	5	1	3	13hex = 19dec Flash DIR name error	
E3	1..4/10	255	5	1	4	14hex = 20dec Function table error	
E3	1..4/10	255	5	1	5	15hex = 21dec RAM type error	
E3	1..4/10	255	5	1	6	16hex = 22dec Flash DIR type error	
E3	1..4/10	255	5	3	2	32hex = 50dec RAM test error	
E3	1..4/10	255	5	3	3	33hex = 51dec Data segment error	
E3	1..4/10	255	5	3	4	34hex = 52dec RAM error	
E3	1..4/10	255	5	3	5	35hex = 53dec EPROM error	
E3	1..4/10	255	5	3	6	36hex = 54dec DONGLE error	
E3	1..4/10	255	5	3	7	37hex = 55dec Wrong RCS identifier error	
E3	1..4/10	255	5	3	8	38hex = 56dec Error allocating memory	
E3	1..4/10	255	5	6	4	64hex = 100dec RCS task not ready	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	6	5	65hex = 101dec Task 1 not ready	
E3	1..4/10	255	5	6	6	66hex = 102dec Task 2 not ready	
E3	1..4/10	255	5	6	7	67hex = 103dec Task 3 not ready	
E3	1..4/10	255	5	6	8	68hex = 104dec Task 4 not ready	
E3	1..4/10	255	5	6	9	69hex = 105dec Task 5 not ready	
E3	1..4/10	255	5	6	10	6Ahex = 106dec Task 6 not ready	
E3	1..4/10	255	5	6	11	6Bhex = 107dec Task 7 not ready	
E3	1..4/10	255	5	6	12	6Chex = 108dec Task 8 not ready	
E3	1..4/10	255	5	6	13	6Dhex = 109dec Task 9 not ready	
E3	1..4/10	255	5	6	14	6Ehex = 110dec Task 10 not ready	
E3	1..4/10	255	5	6	15	6Fhex = 111dec Task 11 not ready	
E3	1..4/10	255	5	7	0	70hex = 112dec Task 12 not ready	
E3	1..4/10	255	5	7	1	71hex = 113dec Task 13 not ready	
E3	1..4/10	255	5	7	2	72hex = 114dec Task 14 not ready	
E3	1..4/10	255	5	7	3	73hex = 115dec Task 15 not ready	
E3	1..4/10	255	5	7	8	78hex = 120dec MCL 0 missing	
E3	1..4/10	255	5	7	9	79hex = 121dec MCL 1 missing	
E3	1..4/10	255	5	7	10	7Ahex = 122dec MCL 2 missing	
E3	1..4/10	255	5	8	0	80hex = 128dec MCL double	
E3	1..4/10	255	5	8	1	81hex = 129dec MCL start address	
E3	1..4/10	255	5	8	2	82hex = 130dec MCL 0 error	
E3	1..4/10	255	5	8	3	83hex = 131dec MCL 1 error	
E3	1..4/10	255	5	8	4	84hex = 132dec MCL 2 error	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	8	10	8Ahex = 138dec MCL mode	
E3	1..4/10	255	5	8	12	8Chex = 140dec RCS 0 missing	
E3	1..4/10	255	5	8	13	8Dhex = 141dec RCS 1 missing	
E3	1..4/10	255	5	8	14	8Ehex = 142dec RCS 2 missing	
E3	1..4/10	255	5	8	15	8Fhex = 143dec RCS 3 missing	
E3	1..4/10	255	5	9	0	90hex = 144dec RCS 4 missing	
E3	1..4/10	255	5	9	1	91hex = 145dec RCS 5 missing	
E3	1..4/10	255	5	9	2	92hex = 146dec RCS 6 missing	
E3	1..4/10	255	5	9	3	93hex = 147dec RCS 7 missing	
E3	1..4/10	255	5	9	4	94hex = 148dec RCS double	
E3	1..4/10	255	5	9	5	95hex = 149dec RCS start address	
E3	1..4/10	255	5	9	6	96hex = 150dec RCS 0 error	
E3	1..4/10	255	5	9	7	97hex = 151dec RCS 1 error	
E3	1..4/10	255	5	9	8	98hex = 152dec RCS 2 error	
E3	1..4/10	255	5	9	9	99hex = 153dec RCS 3 error	
E3	1..4/10	255	5	9	10	9Ahex = 154dec RCS 4 error	
E3	1..4/10	255	5	9	11	9Bhex = 155dec RCS 5 error	
E3	1..4/10	255	5	9	12	9Chex = 156dec RCS 6 error	
E3	1..4/10	255	5	9	13	9Dhex = 157dec RCS 7 error	
E3	1..4/10	255	5	10	0	A0hex = 160dec LIB 0 missing	
E3	1..4/10	255	5	10	1	A1hex = 161dec LIB 1 missing	
E3	1..4/10	255	5	10	2	A2hex = 162dec LIB 2 missing	
E3	1..4/10	255	5	10	3	A3hex = 163dec LIB 3 missing	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	10	4	A4hex = 164dec LIB 4 missing	
E3	1..4/10	255	5	10	5	A5hex = 165dec LIB 5 missing	
E3	1..4/10	255	5	10	6	A6hex = 166dec LIB 6 missing	
E3	1..4/10	255	5	10	7	A7hex = 167dec LIB 7 missing	
E3	1..4/10	255	5	10	8	A8hex = 168dec LIB double	
E3	1..4/10	255	5	10	9	A9hex = 169dec LIB start address	
E3	1..4/10	255	5	10	10	AAhex = 170dec LIB 0 error	
E3	1..4/10	255	5	10	11	ABhex = 171dec LIB 1 error	
E3	1..4/10	255	5	10	12	AChex = 172dec LIB 2 error	
E3	1..4/10	255	5	10	13	ADhex = 173dec LIB 3 error	
E3	1..4/10	255	5	10	14	AEhex = 174dec LIB 4 error	
E3	1..4/10	255	5	10	15	AFhex = 175dec LIB 5 error	
E3	1..4/10	255	5	11	0	B0hex = 176dec LIB 6 error	
E3	1..4/10	255	5	11	1	B1hex = 177dec LIB 7 error	
E3	1..4/10	255	5	12	8	C8hex = 200dec unknown IRQ	
E3	1..4/10	255	5	12	9	C9hex = 201dec Watchdog	
E3	1..4/10	255	5	12	10	CAhex = 202dec SCC TX IRQ	
E3	1..4/10	255	5	12	11	CBhex = 203dec SCC RX IRQ	
E3	1..4/10	255	5	12	12	CChex = 204dec Task state	
E3	1..4/10	255	5	14	6	E6hex = 230dec Task 0	
E3	1..4/10	255	5	14	7	E7hex = 231dec Task 1	
E3	1..4/10	255	5	14	8	E8hex = 232dec Task 2	
E3	1..4/10	255	5	14	9	E9hex = 233dec Task 3	
E3	1..4/10	255	5	14	10	EAhex = 234dec Task 4	
E3	1..4/10	255	5	14	11	EBhex = 235dec Task 5	
E3	1..4/10	255	5	14	12	EChex = 236dec Task 6	

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	5	14	13	EDhex = 237dec Task 7	
E3	1..4/10	255	5	15	0	F0hex = 240dec DBG task 0 segment	
E3	1..4/10	255	5	15	1	F1hex = 241dec DBG task 1 segment	
E3	1..4/10	255	5	15	2	F2hex = 242dec DBG task 2 segment	
E3	1..4/10	255	5	15	3	F3hex = 243dec DBG task 3 segment	
E3	1..4/10	255	5	15	4	F4hex = 244dec DBG task 4 segment	
E3	1..4/10	255	5	15	5	F5hex = 245dec DBG task 5 segment	
E3	1..4/10	255	5	15	6	F6hex = 246dec DBG task 6 segment	
E3	1..4/10	255	5	15	7	F7hex = 247dec DBG task 7 segment	

Table 786: General task errors

Error class	Component	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	6..12	0	1	01hex = 1dec No communication	
E3	1..4/10	255	6..12	0	2	02hex = 2dec Idle	
E3	1..4/10	255	6..12	3	2	32hex = 50dec Base initialization	
E3	1..4/10	255	6..12	6	4	64hex = 100dec Parity error	
E3	1..4/10	255	6..12	6	5	65hex = 101dec Frame error	
E3	1..4/10	255	6..12	6	6	66hex = 102dec Overrun	
E3	1..4/10	255	6..12	6	7	67hex = 103dec Data count	
E3	1..4/10	255	6..12	6	8	68hex = 104dec Checksum error	
E3	1..4/10	255	6..12	6	9	69hex = 105dec Timeout	
E3	1..4/10	255	6..12	6	10	6Ahex = 106dec Protocol error	
E3	1..4/10	255	6..12	6	11	6Bhex = 107dec Data error	
E3	1..4/10	255	6..12	6	12	6Chex = 108dec NACK	
E3	1..4/10	255	6..12	6	14	6Ehex = 110dec Protocol base	

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	6..12	9	6	96hex = 150dec Invalid message header	
E3	1..4/10	255	6..12	9	7	97hex = 151dec Invalid message length	
E3	1..4/10	255	6..12	9	8	98hex = 152dec Invalid message command	
E3	1..4/10	255	6..12	9	9	99hex = 153dec Invalid message structure	
E3	1..4/10	255	6..12	9	10	9Ahex = 154dec Mes-sage error	
E3	1..4/10	255	6..12	9	11	9Bhex = 155dec Mes-sage timeout	
E3	1..4/10	255	6..12	9	12	9Chex = 156dec Invalid message sequence	
E3	1..4/10	255	6..12	9	13	9Dhex = 157dec Invalid message number	
E3	1..4/10	255	6..12	9	14	9Ehex = 158dec Unable to run the command, since execution of the previous command is not yet finished	
E3	1..4/10	255	6..12	10	0	A0hex = 160dec Error in telegram header	
E3	1..4/10	255	6..12	10	1	A1hex = 161dec Invalid device address	
E3	1..4/10	255	6..12	10	2	A2hex = 162dec Wrong address data area	
E3	1..4/10	255	6..12	10	3	A3hex = 163dec Data address and data count cause a buffer overflow	
E3	1..4/10	255	6..12	10	4	A4hex = 164dec Invalid data index	
E3	1..4/10	255	6..12	10	5	A5hex = 165dec Invalid data count	
E3	1..4/10	255	6..12	10	6	A6hex = 166dec Unknown data type	
E3	1..4/10	255	6..12	10	7	A7hex = 167dec Unknown function	
E3	1..4/10	255	6..12	10	10	AAhex = 170dec Mes-sage base	
E3	1..4/10	255	6..12	12	8	C8hex = 200dec Task not initialized, Commu-nication Module not configured	
E3	1..4/10	255	6..12	12	9	C9hex = 201dec Busy	
E3	1..4/10	255	6..12	12	10	CAhex = 202dec No segment of RCS received	

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	6..12	12	11	CBhex = 203dec Unknown or wrong sender of a command message	
E3	1..4/10	255	6..12	13	2	D2hex = 210dec No database	
E3	1..4/10	255	6..12	13	3	D3hex = 211dec Error writing the database	
E3	1..4/10	255	6..12	13	4	D4hex = 212dec Error reading the database	
E3	1..4/10	255	6..12	13	5	D5hex = 213dec Error registering the diag-nosis structure	
E3	1..4/10	255	6..12	13	6	D6hex = 214dec Parameter error	
E3	1..4/10	255	6..12	13	7	D7hex = 215dec Con-figuration	
E3	1..4/10	255	6..12	13	8	D8hex = 216dec Func-tion list	
E3	1..4/10	255	6..12	13	9	D9hex = 217dec System	
E3	1..4/10	255	6..12	13	10	DAhex = 218dec Not enough internal memory available	
E3	1..4/10	255	6..12	13	11	DBhex = 219dec No DPR	
E3	1..4/10	255	6..12	13	12	DChex = 220dec System base	

1.7.3.4.2 Ethernet communication module errors

Table 787: OMB task (Modbus TCP) errors (task 3) [1])

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	8	3	4	34hex = 52dec Invalid configuration data, server connections	
E3	1..4/10	255	8	3	5	35hex = 53dec Invalid configuration data, task timeout	
E3	1..4/10	255	8	3	6	36hex = 54dec Invalid configuration data, OMB timeout	
E3	1..4/10	255	8	3	7	37hex = 55dec Invalid configuration data, mode	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	8	3	8	38hex = 56dec Invalid configuration data, send timeout	
E3	1..4/10	255	8	3	9	39hex = 57dec Invalid configuration data, con-nect timeout	
E3	1..4/10	255	8	3	10	3Ahex = 58dec Invalid configuration data, close timeout	
E3	1..4/10	255	8	3	11	3Bhex = 59dec Invalid configuration data, swap	
E3	1..4/10	255	8	3	12	3Chex = 60dec TCP_UDP task not found, TCP task not ready	
E3	1..4/10	255	8	3	13	3Dhex = 61dec PLC task not found, PLC task not ready	
E3	1..4/10	255	8	3	14	3Ehex = 62dec Error initializing OMB task	
E3	1..4/10	255	8	3	15	3Fhex = 63dec Error initializing PLC task mode	
E3	1..4/10	255	8	6	15	6Fhex = 111dec Unknown sender of a response	
E3	1..4/10	255	8	7	0	70hex = 112dec Error code in response	
E3	1..4/10	255	8	7	1	71hex = 113dec No socket found in searched status	
E3	1..4/10	255	8	7	2	72hex = 114dec Invalid value in request	
E3	1..4/10	255	8	7	3	73hex = 115dec Error message of TCP task	
E3	1..4/10	255	8	7	4	74hex = 116dec Modbus error	
E3	1..4/10	255	8	7	5	75hex = 117dec No socket available	
E3	1..4/10	255	8	7	6	76hex = 118dec Invalid socket handle	
E3	1..4/10	255	8	7	7	77hex = 119dec Timeout in client socket	
E3	1..4/10	255	8	7	8	78hex = 120dec Socket closed, without response to command	
E3	1..4/10	255	8	7	9	79hex = 121dec Not ready flag set	

Error class	Compo-nent d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy
E3	1..4/10	255	8	7	10	7Ahex = 122dec TCP task no longer ready	
E3	1..4/10	255	8	7	11	7Bhex = 123dec Watchdog event	
E3	1..4/10	255	8	7	12	7Chex = 124dec Device in reconfiguration	
E3	1..4/10	255	8	7	13	7Dhex = 125dec PLC task not initialized	
E3	1..4/10	255	8	7	14	7Ehex = 126dec OMB server socket closed	

Table 788: TCP/UDP task (task 6) errors [1])

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	11	3	2	32hex = 50dec Init of IP task not completed	
E3	1..4/10	255	11	3	3	33hex = 51dec Error when initializing the task configuration	
E3	1..4/10	255	11	3	4	34hex = 52dec Init of IP task failed	
E3	1..4/10	255	11	3	7	37hex = 55dec No memory available for init	
E3	1..4/10	255	11	6	14	6Ehex = 110dec Timeout	
E3	1..4/10	255	11	6	15	6Fhex = 111dec Invalid timeout	
E3	1..4/10	255	11	7	0	70hex = 112dec Invalid socket	
E3	1..4/10	255	11	7	1	71hex = 113dec Socket status	
E3	1..4/10	255	11	7	3	73hex = 115dec Target not reachable	
E3	1..4/10	255	11	7	4	74hex = 116dec Option not supported	
E3	1..4/10	255	11	7	5	75hex = 117dec Invalid parameter	
E3	1..4/10	255	11	7	6	76hex = 118dec Invalid IP address	
E3	1..4/10	255	11	7	7	77hex = 119dec Invalid port	
E3	1..4/10	255	11	7	8	78hex = 120dec CONN closed	
E3	1..4/10	255	11	7	9	79hex = 121dec CONN reset	

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	11	7	10	7Ahex = 122dec Unknown protocol	
E3	1..4/10	255	11	7	11	7Bhex = 123dec No sockets	
E3	1..4/10	255	11	8	2	82hex = 130dec Unknown mode	
E3	1..4/10	255	11	8	3	83hex = 131dec Max. data length exceeded	
E3	1..4/10	255	11	8	4	84hex = 132dec Message count exceeded	
E3	1..4/10	255	11	8	5	85hex = 133dec Max. group exceeded	
E3	1..4/10	255	11	9	5	95hex = 149dec Unexpected response message	

Table 789: IP task (task 7) errors [1])

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	12	3	2	32hex = 50dec No TCP task initialized	
E3	1..4/10	255	12	3	3	33hex = 51dec Error when initializing the task configuration	
E3	1..4/10	255	12	3	4	34hex = 52dec No Ethernet address	
E3	1..4/10	255	12	3	5	35hex = 53dec Wait for warm start	
E3	1..4/10	255	12	3	6	36hex = 54dec Invalid flags	
E3	1..4/10	255	12	3	7	37hex = 55dec Invalid IP address	
E3	1..4/10	255	12	3	8	38hex = 56dec Invalid net mask	
E3	1..4/10	255	12	3	9	39hex = 57dec Invalid gateway	
E3	1..4/10	255	12	3	11	3Bhex = 59dec Unknown hardware	
E3	1..4/10	255	12	3	12	3Chex = 60dec No IP address	
E3	1..4/10	255	12	3	13	3Dhex = 61dec Error initializing the driver	
E3	1..4/10	255	12	3	14	3Ehex = 62dec No IP address configuration	
E3	1..4/10	255	12	3	15	3Fhex = 63dec Invalid serial number	
E3	1..4/10	255	12	4	0	40hex = 64dec No memory on chip	

Error class	Compo-nent	Device	Module	Channel	Error identifier	Error message	Remedy
E3	1..4/10	255	12	6	14	6Ehex = 110dec Timeout	
E3	1..4/10	255	12	6	15	6Fhex = 111dec Timeout invalid	
E3	1..4/10	255	12	7	3	73hex = 115dec Target not reachable	
E3	1..4/10	255	12	7	6	76hex = 118dec IP address invalid	
E3	1..4/10	255	12	7	12	7Chex = 124dec Ethernet address invalid	
E3	1..4/10	255	12	8	2	82hex = 130dec Unknown mode	
E3	1..4/10	255	12	8	3	83hex = 131dec ARP cache full	
E3	1..4/10	255	12	8	6	86hex = 134dec No ARP entry found	
E3	1..4/10	255	12	9	5	95hex = 149dec Unex-pected response	

[1]):

- The error information is also available at the output ERNO of the blocks used for the Communication Module.
- The following applies: ERNO := 6000hex OR error.

1.7.3.4.3 CM579-ETHCAT

Status codes

Hexadecimal Value	Definition	Description
0x00000000	TLR_S_OK	Status ok
0xC0650005	TLR_E_ETHERCAT_MASTER_ERROR_BUSSCAN_FAILED	Existing bus does not match config-ured bus.
0xC0650006	TLR_E_ETHERCAT_MASTER_NOT_ALL_SLAVES_AVAIL	Not all slaves are available.
0xC065000B	TLR_E_ETHERCAT_MASTER_INVALID_BUSCYCLETIME	The requested bus cycle time is invalid.
0xC065000C	TLR_E_ETHERCAT_MASTER_INVALID_BROKEN_SLAVE_BEHAV-IOUR_PARA	Invalid parameter for broken slave behavior.
0xC065000F	TLR_E_ETHERCAT_MASTER_COE_INVALID_SLAVEID	Invalid SlaveId was used for CoE.
0xC0650012	TLR_E_ETHERCAT_MASTER_COE_INVALID_INDEX	Invalid Index on slave requested.
0xC0650013	TLR_E_ETHERCAT_MASTER_COE_INVALID_COMMUNICA-TION_STATE	Invalid bus communication state for CoE-Usage.
0xC0650014	TLR_E_ETHERCAT_MASTER_COE_FRAME_LOST	Frame with CoE data is lost.

Hexadecimal Value	Definition	Description
0xC0650015	TLR_E_ETHERCAT_MASTER_COE_TIMEOUT	Timeout during CoE service.
0xC0650016	TLR_E_ETHERCAT_MASTER_COE_SLAVE_NOT_ADDRESSABLE	Slave is not addressable (not on bus or power down?).
0xC0650017	TLR_E_ETHERCAT_MASTER_COE_INVALID_LIST_TYPE	Invalid list type requested (during GetOdList).
0xC0650018	TLR_E_ETHERCAT_MASTER_COE_SLAVE_RESPONSE_TOO_BIG	Data in slave response is too big for confirmation packet.
0xC0650019	TLR_E_ETHERCAT_MASTER_COE_INVALID_ACCESSBITMASK	Invalid access mask selected (during GetEntryDesc).
0xC065001A	TLR_E_ETHERCAT_MASTER_COE_WKC_ERROR	Slave Working Counter Error during CoE service.
0xC065001C	TLR_E_ETHERCAT_MASTER_INVALID_COMMUNICATION_STATE	Command is not usable in the communication state.
0xC065001E	TLR_E_ETHERCAT_MASTER_BUS_SCAN_CURRENTLY_RUNNING	The scan is already running. It cannot be started twice at the same time.
0xC065001F	TLR_E_ETHERCAT_MASTER_BUS_SCAN_TIMEOUT	Timeout during bus scan. But at least a link is established.
0xC0650020	TLR_E_ETHERCAT_MASTER_BUS_SCAN_NOT_READY_YET	The bus scan was not started before or is not finish yet.
0xC0650021	TLR_E_ETHERCAT_MASTER_BUS_SCAN_INVALID_SLAVE	The requested slave is invalid.
0xC0650022	TLR_E_ETHERCAT_MASTER_COE_INVALIDACCESS	Slave does not allow reading or writing (CoE-Access).
0xC0650023	TLR_E_ETHERCAT_MASTER_COE_NO_MBX_SUPPORT	Slave does not support a mailbox.
0xC0650024	TLR_E_ETHERCAT_MASTER_COE_NO_COE_SUPPORT	Slave does not support CoE.
0xC0650025	TLR_E_ETHERCAT_MASTER_TASK_CREATION_FAILED	Task could not be created during run time.
0xC0650026	TLR_E_ETHERCAT_MASTER_INVALID_SLAVE_SM_CONFIGURATION	The Sync Manager configuration of a slave is invalid.
0xC0650027	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TOGGLE	SDO abort code: Toggle bit not alternated.
0xC0650028	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TIMEOUT	SDO abort code: SDO protocol timed out.
0xC0650029	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_CCS_SCS	SDO abort code: Client/server command specifier not valid or unknown.
0xC065002A	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_BLK_SIZE	SDO abort code: Invalid block size (block mode only).
0xC065002B	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_SEQNO	SDO abort code: Invalid sequence number (block mode only).
0xC065002C	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_CRC	SDO abort code: CRC error (block mode only).
0xC065002D	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_MEMORY	SDO abort code: Out of memory.
0xC065002E	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_ACCESS	SDO abort code: Unsupported access to an object.

Hexadecimal Value	Definition	Description
0xC065002F	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_WRITEONLY	SDO abort code: Attempt to read a write only object.
0xC0650030	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_READONLY	SDO abort code: Attempt to write a read only object.
0xC0650031	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_INDEX	SDO abort code: Object does not exist in the object dictionary.
0xC0650032	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_MAP	SDO abort code: Object cannot be mapped to the PDO.
0xC0650033	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_PDO_LEN	SDO abort code: The number and length of the objects to be mapped would exceed PDO length.
0xC0650034	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_P_INCOMP	SDO abort code: General parameter incompatibility reason.
0xC0650035	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_I_INCOMP	SDO abort code: General internal incompatibility in the device.
0xC0650036	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_HARDWARE	SDO abort code: Access failed due to an hardware error.
0xC0650037	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE	SDO abort code: Data type does not match, length of service parameter does not match.
0xC0650038	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE1	SDO abort code: Data type does not match, length of service parameter too high.
0xC0650039	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_SIZE2	SDO abort code: Data type does not match, length of service parameter too low.
0xC065003A	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_OFFSET	SDO abort code: Sub-index does not exist.
0xC065003B	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE	SDO abort code: Range of values of parameter exceeded (only for write access).
0xC065003C	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE1	SDO abort code: Value of parameter written too high.
0xC065003D	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DATA_RANGE2	SDO abort code: Value of parameter written too low.
0xC065003E	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_MINMAX	SDO abort code: Maximum value is less than minimum value.
0xC065003F	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_GENERAL	SDO abort code: general error.
0xC0650040	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER	SDO abort code: Data cannot be transferred or stored to the application.
0xC0650041	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER1	SDO abort code: Data cannot be transferred or stored to the application because of local control.
0xC0650042	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_TRANSFER2	SDO abort code: Data cannot be transferred or stored to the application because of the present device state.

Hexadecimal Value	Definition	Description
0xC0650043	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_DICTIONARY	SDO abort code: Object dictionary dynamic generation fails or no object dictionary is present (e.g. object dictionary is generated from file and generation fails because of an file error).
0xC0650044	TLR_E_ETHERCAT_MASTER_SDO_ABORTCODE_UNKNOWN	SDO abort code: unknown code.
0xC0CC0001	ECM_ERROR_LLD_TIMEOUT	LLD: Timeout
0xC0CC0003	ECM_ERROR_LLD_UNSUPPORTED_COMMAND	LLD: Unsupported command
0xC0CC0004	ECM_ERROR_LLD_DUPLICATE_FIXED_STATION_ADDRESS	LLD: Duplicate fixed station address
0xC0CC0005	ECM_ERROR_LLD_SII_CHECKSUM_ERROR	LLD: SII Checksum Error
0xC0CC0006	ECM_ERROR_LLD_SII_EEPROM_LOADING_ERROR	LLD: SII EEPROM Loading Error
0xC0CC0007	ECM_ERROR_LLD_SII_MISSING_ERROR_ACK	LLD: SII Missing Error Ack
0xC0CC0008	ECM_ERROR_LLD_STATE_CHANGE_FAILED	LLD: State Change Failed
0xC0CC0009	ECM_ERROR_LLD_UNEXPECTED_AL_STATUS	LLD: Unexpected AL Status
0xC0CC000A	ECM_ERROR_LLD_UNEXPECTED_WKC	LLD: Unexpected WKC
0xC0CC000B	ECM_ERROR_LLD_MAILBOX_NOT_AVAILABLE	LLD: Mailbox not available
0xC0CC000C	ECM_ERROR_LLD_MAILBOX_MESSAGE_TOO_LARGE	LLD: Mailbox message too large
0xC0CC000D	ECM_ERROR_LLD_CONFIGURATION_IN_PROGRESS	LLD: Configuration in progress
0xC0CC000E	ECM_ERROR_LLD_TOO_MANY_CYCLIC_FRAMES	LLD: Too many cyclic frames
0xC0CC000F	ECM_ERROR_LLD_CYCLIC_FRAME_EXCEEDS_MTU	LLD: Cyclic frame exceeds MTU
0xC0CC0010	ECM_ERROR_LLD_INVALID_CYCLIC_TELEGRAM_CONFIG	LLD: Invalid cyclic telegram config
0xC0CC0011	ECM_ERROR_LLD_BUILDING_COPY_ROUTINES_FAILED	LLD: Building copy routines failed
0xC0CC0012	ECM_ERROR_LLD_UNSUPPORTED_SLAVE_STATION_ADDRESS	LLD: Unsupported slave station address
0xC0CC0013	ECM_ERROR_LLD_STATION_ADDRESS_NOT_ALLOWED	LLD: Station Address not allowed
0xC0CC0014	ECM_ERROR_LLD_INVALID_STD_TX_MBX_PHYS_OFFSET	LLD: Invalid Std TxMbx PhysOffset
0xC0CC0015	ECM_ERROR_LLD_INVALID_STD_RX_MBX_PHYS_OFFSET	LLD: Invalid Std Rx Mbx PhysOffset
0xC0CC0016	ECM_ERROR_LLD_INVALID_BOOT_TX_MBX_PHYS_OFFSET	LLD: Invalid BOOT Rx Mbx PhysOffset

Hexadecimal Value	Definition	Description
0xC0CC0017	ECM_ERROR_LLD_INVALID_BOOT_RX_MBX_PHYS_OFFSET	LLD: Invalid BOOT Tx Mbx Phys-Offset
0xC0CC0018	ECM_ERROR_LLD_INVALID_STD_TX_MBX_SM_NO	LLD: Invalid Std Tx Mbx SmNo
0xC0CC0019	ECM_ERROR_LLD_INVALID_STD_RX_MBX_SM_NO	LLD: Invalid Std Rx Mbx SmNo
0xC0CC001A	ECM_ERROR_LLD_INVALID_BOOT_TX_MBX_SM_NO	LLD: Invalid BOOT Tx Mbx SmNo
0xC0CC001B	ECM_ERROR_LLD_INVALID_BOOT_RX_MBX_SM_NO	LLD: Invalid BOOT Rx Mbx SmNo
0xC0CC001C	ECM_ERROR_LLD_UNCON-FIGURED_SLAVE_STA-TION_ADDRESS	LLD: Unconfigured slave station address
0xC0CC001D	ECM_ERROR_LLD_WRONG_SLAVE_STATE	LLD: Wrong slave state
0xC0CC001E	ECM_ERROR_LLD_CYCLE_TIME_TOO_SMALL	LLD: Cycle time too small
0xC0CC001F	ECM_ERROR_LLD_REPETI-TION_COUNT_NOT_SUPPORTED	LLD: Repetition count not supported
0xC0CC0020	ECM_ERROR_LLD_INVALID_CALLBACK_TYPE	LLD: Invalid callback type
0xC0CC0021	ECM_ERROR_LLD_INVALID_CYCLE_MULTIPLIER	LLD: Invalid cycle multiplier
0xC0CC0022	ECM_ERROR_LLD_UNKNOWN_ERROR	LLD: Unknown Error
0xC0CC0023	ECM_ERROR_LLD_INVALID_REG_LENGTH	LLD: Invalid reg length
0xC0CC0024	ECM_ERROR_LLD_INVALID_PARAMETER	LLD: Invalid parameter
0xC0CC0025	ECM_ERROR_LLD_IRQ_NOT_AVAILABLE	LLD: IRQ not available
0xC0CC0026	ECM_ERROR_LLD_IOMEM_IRQ_NOT_AVAILABLE	LLD: IOMem Irq not available
0xC0CC0027	ECM_ERROR_LLD_HW_INIT_FAILED	LLD: Hardware init failed
0xC0CC0028	ECM_ERROR_LLD_MUTEX_CRE-ATION_FAILED	LLD: Mutex creation failed
0xC0CC0029	ECM_ERROR_LLD_DC_RX_LATCH_COM-MAND_REQUIRED_FOR_DC	LLD: DC Rx Latch command is not configured within cyclic frames
0xC0CC002A	ECM_ERROR_LLD_TX_PROCESS_IMAGE_EXCEEDED	LLD: Transmit process image is exceeded
0xC0CC002B	ECM_ERROR_LLD_RX_PROCESS_IMAGE_EXCEEDED	LLD: Receive process image is exceeded
0xC0CC002C	ECM_ERROR_LLD_MBX_STATE_IMAGE_EXCEEDED	LLD: Mailbox State image is exceeded
0xC0CC002D	ECM_ERROR_LLD_RESULT_DUPLICATE_BWR_RX_LATCH_CMD	LLD: Duplicate BWR Rx DC Latch command detected in cyclic frames

Hexadecimal Value	Definition	Description
0xC0CC002E	ECM_ERROR_LLD_RESULT_DUPLICATE_EXT_SYSTIME_CONTROL_CMD	LLD: Duplicate External Sync SysTime Control command detected in cyclic frames
0xC0CC002F	ECM_ERROR_LLD_CC_PROCESS_IMAGE_EXCEEDED	LLD: Cross Communication Process image exceeded
0x40CD0017	ECM_INFO_EMC_BUS_IS_OFF	Bus is off
0xC0CD0001	ECM_ERROR_EMC_REQUEST_DESTINATION_PROBLEM	Request Destination Problem
0xC0CD0002	ECM_ERROR_EMC_INVALID_SLAVE_STATION_ADDRESS	Invalid slave station address
0xC0CD0003	ECM_ERROR_EMC_CONFIGURATION_BUFFER_IS_OPEN	Configuration buffer is open
0xC0CD0004	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION	Wrong state for reconfiguration
0xC0CD0005	ECM_ERROR_EMC_CONFIGURATION_BUFFER_IS_NOT_OPEN	Configuration buffer is not open
0xC0CD0006	ECM_ERROR_EMC_SLAVE_STATION_ADDRESS_ALREADY_IN_CONFIG	Slave station address already in config
0xC0CD0007	ECM_ERROR_EMC_INVALID_STD_MBX_PARAMETERS	Invalid Std Mbx parameters
0xC0CD0008	ECM_ERROR_EMC_INVALID_BOOT_MBX_PARAMETERS	Invalid BOOT Mbx parameters
0xC0CD0009	ECM_ERROR_EMC_STD_MBX_SM_ARE_OVERLAPPING	Std Mbx SMs are overlapping
0xC0CD000A	ECM_ERROR_EMC_BOOT_MBX_SM_ARE_OVERLAPPING	BOOT Mbx SMs are overlapping
0xC0CD000B	ECM_ERROR_EMC_SM_PARAMS_ALREADY_ADDED	SM Params already added
0xC0CD000C	ECM_ERROR_EMC_INVALID_SM_NUMBER	Nvalid SM number
0xC0CD000D	ECM_ERROR_EMC_FMMU_PARAMS_ALREADY_ADDED	FMMU params already added
0xC0CD000E	ECM_ERROR_EMC_INVALID_FMMU_NUMBER	Invalid FMMU number
0xC0CD000F	ECM_ERROR_EMC_INVALID_MIN_STATE	Invalid min state
0xC0CD0010	ECM_ERROR_EMC_CYCLE_FRAME_AMOUNT_EXCEEDED	Cycle frame amount exceeded
0xC0CD0011	ECM_ERROR_EMC_INVALID_CYCLIC_FRAME_IN_CONFIGURATION	Invalid cycle frame in configuration
0xC0CD0012	ECM_ERROR_EMC_CYCLE_FRAME_INDEX_NOT_VALID	Cycle frame index not valid
0xC0CD0013	ECM_ERROR_EMC_INVALID_TELEGRAM_LENGTH	Invalid telegram length
0xC0CD0014	ECM_ERROR_EMC_CYCLE_FRAME_LENGTH_EXCEEDED	Cycle frame length exceeded

Hexadecimal Value	Definition	Description
0xC0CD0015	ECM_ERROR_EMC_AMOUNT_OF_TELE-GRAMS_IN_CYCLIC_FRAME_EXCEEDED	Amount of telegrams in cyclic frame exceeded
0xC0CD0016	ECM_ERROR_EMC_STATE_CHANGE_IN_PROGRESS	State change in progress
0xC0CD0018	ECM_ERROR_EMC_TOO_MANY_SLAVES_GIVEN	Too many slaves given
0xC0CD0019	ECM_ERROR_EMC_DUPLI-CATE_STA-TION_ADDRESS_IN_LIST	Duplicate station address in list
0xC0CD001A	ECM_ERROR_EMC_COM-MAND_TYPE_NOT_ALLOWED_FOR_SLAVE_FSM	Command type not allowed for slave FSM
0xC0CD001B	ECM_ERROR_EMC_CONFIGURA-TION_DATA_INCORRECT	Configuration data incorrect
0xC0CD001C	ECM_ERROR_EMC_VEN-DORID_MISMATCH	VendorID mismatch
0xC0CD001D	ECM_ERROR_EMC_PRODUCT-CODE_MISMATCH	ProductCode mismatch
0xC0CD001E	ECM_ERROR_EMC_REVI-SIONNO_MISMATCH	Revision number mismatch
0xC0CD001F	ECM_ERROR_EMC_SERI-ALNO_MISMATCH	Serial number mismatch
0xC0CD0020	ECM_ERROR_EMC_LOST_CON-NECTION	Lost connection
0xC0CD0021	ECM_ERROR_EMC_UNKNOWN_STATE_CHANGE_HAPPENED	Unknown state change happened
0xC0CD0022	ECM_ERROR_EMC_UNEX-PECTED_STATE_CHANGE_HAP-PENED	Unexpected state change happened
0xC0CD0023	ECM_ERROR_EMC_SLAVE_CHANGED_STATE	Slave changed state
0xC0CD0026	ECM_ERROR_EMC_DC_RX_TIME-STAMP_ERROR	DC Rx Timestamp error
0xC0CD0027	ECM_ERROR_EMC_DC_MASTER_PORT_TIMESTAMP_ERROR	DC master port timestamp error
0xC0CD0028	ECM_ERROR_EMC_INVALID_SLAVE_INDEX	Invalid slave index
0xC0CD0029	ECM_ERROR_EMC_WRONG_MASTER_STATE	
0xC0CD002A	ECM_ERROR_EMC_INVALID_TRANSFER_ID	Invalid Transfer Id
0xC0CD002B	ECM_ERROR_EMC_INVALID_SEGMENTATION	Invalid Segmentation
0xC0CD002C	ECM_ERROR_EMC_IP_PARAMS_ALREADY_ADDED	EoE IP Params already added
0xC0CD002D	ECM_ERROR_EMC_EOE_SUP-PORT_NOT_AVAILABLE	EoE support not available

Hexadecimal Value	Definition	Description
0xC0CD002E	ECM_ERROR_EMC_END_CONFIGURATION_IN_PROGRESS	End configuration in progress
0xC0CD002F	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_IS_ON	Wrong state for reconfiguration (Bus Is On)
0xC0CD0030	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_BUS_SCAN_ACTIVE	Wrong state for reconfiguration (Bus Scan Active)
0xC0CD0031	ECM_ERROR_EMC_WRONG_STATE_FOR_RECONFIGURATION_IN_PROGRESS_TO_BUSOFF	Wrong state for reconfiguration (In Progress to Bus off)
0xC0CD0032	ECM_EROR_EMC_NO_DIAG_ENTRY_AVAILABLE	No Diag Entry available
0xC0CD0033	ECM_ERROR_EMC_SLAVE_SYNC_PARAMS_NOT_POSSIBLE_WITHOUT_WORKING_DC	A slave has been configured to have SYNC0 and/or SYNC1 but does not support DC at all.
0xC0CD0034	ECM_ERROR_EMC_MANDATORY_SLAVE_MISSING	At least one required slave for boot up is missing.
0xC0CD0035	ECM_ERROR_EMC_WRONG_SLAVE_AT_POSITION	A wrong slave at a specific position has been detected.
0xC0CD0036	ECM_ERROR_EMC_NO_DC_REF_CLOCK	No DC reference clock
0xC0CD0037	ECM_ERROR_EMC_DC_REF_CLOCK_DOES_NOT_PROVIDE_64BIT	DC Reference clock does not provide 64 Bit
0xC0CD0038	ECM_ERROR_EMC_INVALID_DC_REF_CLOCK	Invalid DC Reference clock
0xC0CD0039	ECM_ERROR_EMC_COE_SUPPORT_NOT_AVAILABLE	CoE support not available
0xC0CD003A	ECM_ERROR_EMC_SOE_SUPPORT_NOT_AVAILABLE	SoE support not available
0xC0CD003B	ECM_ERROR_EMC_FOE_SUPPORT_NOT_AVAILABLE	FoE support not available
0xC0CD003C	ECM_ERROR_EMC_AOE_SUPPORT_NOT_AVAILABLE	AoE support not available
0x40CD003E	ECM_INFO_EMC_RECONNECTED	Reconnected
0x80CD003F	ECM_WARN_EMC_DC_STOPPED	DC stopped
0xC0CD0040	ECM_ERROR_EMC_STOPPED_DUE_SYNC_ERROR	Stopped due Sync Error
0xC0CD0041	ECM_ERROR_EMC_MANDATORY_SLAVE_NOT_IN_OP	At least one mandatory slave is not in OP
0xC0CD0042	ECM_ERROR_EMC_BUS_CYCLE_TIME_NOT_POSSIBLE	Bus Cycle Time not possible
0xC0CD0043	ECM_ERROR_EMC_TOPOLOGY_ERROR_DETECTED	Topology error detected
0xC0CD0044	ECM_ERROR_EMC_TOPOLOGY_MISMATCH_DETECTED	Topology mismatch detected
0xC0CD0045	ECM_ERROR_EMC_NO_VALID_TOPOLOGY_CONFIGURATION_DATA	No valid topology configuration data

Hexadecimal Value	Definition	Description
0xC0CD0046	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0	Unexpected slave at port 0 of slave.
0xC0CD0047	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT1	Unexpected slave at port 1 of slave.
0xC0CD0048	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT2	Unexpected slave at port 2 of slave.
0xC0CD0049	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT3	Unexpected slave at port 3 of slave.
0xC0CD004A	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_RECONNECTED	-
0xC0CD004B	ECM_ERROR_EMC_UNEX-PECTED_MISSING_SLAVE_AT_PORT0	Missing slave at port 0 of slave.
0xC0CD004C	ECM_ERROR_EMC_UNEX-PECTED_MISSING_SLAVE_AT_PORT1	Missing slave at port 1 of slave.
0xC0CD004D	ECM_ERROR_EMC_UNEX-PECTED_MISSING_SLAVE_AT_PORT2	Missing slave at port 2 of slave.
0xC0CD004E	ECM_ERROR_EMC_UNEX-PECTED_MISSING_SLAVE_AT_PORT3	Missing slave at port 3 of slave.
0xC0CD004F	ECM_ERROR_EMC_SLAVE_NOT_CHECKED	Slave is not checked.
0xC0CD0050	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_1	Unexpected slave at port 0 and 1 of slave.
0xC0CD0051	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_2	Unexpected slave at port 0 and 2 of slave.
0xC0CD0052	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_3	Unexpected slave at port 0 and 3 of slave.
0xC0CD0053	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT1_2	Unexpected slave at port 1 and 2 of slave.
0xC0CD0054	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT1_3	Unexpected slave at port 1 and 3 of slave.
0xC0CD0055	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT2_3	Unexpected slave at port 2 and 3 of slave.
0xC0CD0056	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_1_2	Unexpected slave at port 0, 1 and 2 of slave.
0xC0CD0057	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_1_3	Unexpected slave at port 0, 1 and 3 of slave.
0xC0CD0058	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT0_2_3	Unexpected slave at port 0, 2 and 3 of slave.
0xC0CD0059	ECM_ERROR_EMC_UNEX-PECTED_SLAVE_AT_PORT1_2_3	Unexpected slave at port 1, 2 and 3 of slave.
0xC0CD005A	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1	Missing slave at port 0 and 1 of slave.
0xC0CD005B	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_2	Missing slave at port 0 and 2 of slave.
0xC0CD005C	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_3	Missing slave at port 0 and 3 of slave.

Hexadecimal Value	Definition	Description
0xC0CD005D	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_2	Missing slave at port 1 and 2 of slave.
0xC0CD005E	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_3	Missing slave at port 1 and 3 of slave.
0xC0CD005F	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT2_3	Missing slave at port 2 and 3 of slave.
0xC0CD0060	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1_2	Missing slave at port 0, 1 and 2 of slave.
0xC0CD0061	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_1_3	Missing slave at port 0, 1 and 3 of slave.
0xC0CD0062	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT0_2_3	Missing slave at port 0, 2 and 3 of slave.
0xC0CD0063	ECM_ERROR_EMC_MISSING_SLAVE_AT_PORT1_2_3	Missing slave at port 1, 2 and 3 of slave.
0xC0CD0065	ECM_ERROR_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MANDATORY_SLAVE_LIST	A Hot Connect group participant is not allowed to be configured a mandatory slave
0xC0CD0066	ECM_ERROR_EMC_HC_PARTICIPANT_NOT_ALLOWED_IN_MULTIPLE_HC_GROUPS	A Hot Connect group participant is not allowed to be configured in multiple Hot Connect groups
0xC0CD0067	ECM_ERROR_EMC_GC_GROUP_HEAD_IS_NOT_LISTED_FOR_HC_DETECTION	Hot Connect group head is not listed for Hot Connect detection
0xC0CD0068	ECM_ERROR_EMC_DC_SETUP_CALCULATION_ERROR	DC Setup calculation has encountered an error
0xC0CD0069	ECM_ERROR_EMC_NON_DC_SLAVE_MORE_THAN_2_PORTS_IN_DC_SETUP	A slave, which does not support DC, has more than 2 ports in a DC setup
0xC0CD006A	ECM_ERROR_EMC_HC_GROUP_CONTAINS_NOT_CONFIGURED_SLAVE	A Hot Connect group has been defined to include a slave address that has no configuration
0xC0CD006B	ECM_ERROR_EMC_ALCONTROL_TIMEOUT	AL Control Timeout happened i.e. a slave ESM state change was not completed in time
0xC0CD006C	ECM_ERROR_EMC_DC_MEASUREMENT_ERROR	DC measurement encountered an error
0xC0CD006D	ECM_ERROR_EMC_RX_DESTINATION_EXCEEDS_RX_IMAGE_SIZE	Receive destination exceeds receive image size
0xC0CD006E	ECM_ERROR_EMC_TX_SOURCE_EXCEEDS_TX_IMAGE_SIZE	Transmit source exceeds transmit image size
0xC0CD006F	ECM_ERROR_EMC_WCSTATEBIT_EXCEEDS_RX_IMAGE_SIZE	WcState bit placement exceeds receive image size
0xC0CD0070	ECM_ERROR_EMC_WKC_MAPPING_EXCEEDS_RX_IMAGE_SIZE	Wkc value placement exceeds receive image size
0xC0CD0071	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0	DC Latch Error detected at port 0 of slave
0xC0CD0072	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1	DC Latch Error detected at port 1 of slave
0xC0CD0073	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT2	DC Latch Error detected at port 2 of slave

Hexadecimal Value	Definition	Description
0xC0CD0074	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT3	DC Latch Error detected at port 3 of slave
0xC0CD0075	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1	DC Latch Error detected at ports 0 and 1 of slave
0xC0CD0076	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_2	DC Latch Error detected at ports 0 and 2 of slave
0xC0CD0077	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_3	DC Latch Error detected at ports 0 and 3 of slave
0xC0CD0078	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1_2	DC Latch Error detected at ports 1 and 2 of slave
0xC0CD0079	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT1_3	DC Latch Error detected at ports 1 and 3 of slave
0xC0CD007A	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT2_3	DC Latch Error detected at ports 2 and 3 of slave
0xC0CD007B	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_2	DC Latch Error detected at ports 0, 1 and 2 of slave
0xC0CD007C	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORT0_1_3	DC Latch Error detected at ports 0, 1 and 3 of slave
0xC0CD007D	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS0_2_3	DC Latch Error detected at ports 0, 2 and 3 of slave
0xC0CD007E	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS1_2_3	DC Latch Error detected at ports 1, 2 and 3 of slave
0xC0CD007F	ECM_ERROR_EMC_DC_RX_LATCH_ERROR_AT_PORTS0_1_2_3	DC Latch Error detected at ports 0, 1, 2 and 3 of slave
0xC0CD0080	ECM_ERROR_EMC_ASSIGN_PDO_IS_MISSING_PDO_MAPPING	AssignPDO data is missing related PDO mapping data
0xC0CD0081	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_SM	Parts of Ext Sync object are not mapped to the same SyncManager
0xC0CD0082	ECM_ERROR_EMC_DUPLICATE_EXT_SYNC_OBJ	Duplicate Ext Sync object mapping
0xC0CD0083	ECM_ERROR_EMC_UNSUPPORTED_EXT_SYNC_OBJ_RECORD	Unsupported Ext Sync object record detected
0xC0CD0084	ECM_ERROR_EMC_UNSUPPORTED_MAPPING_OF_EXT_SYNC_OBJ_RECORD	Unsupported mapping of Ext Sync object record detected
0xC0CD0085	ECM_ERROR_EMC_MISSING_MAPPING_OF_EXT_SYNC_OBJ_RECORD	Missing mapping of Ext Sync object record detected
0xC0CD0086	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_TO_SAME_FMMU	Parts of Ext Sync object are not mapped to the same FMMU
0xC0CD0087	ECM_ERROR_EMC_EXT_SYNC_OBJ_INTERNAL_ERROR	Internal error detected regarding Ext Sync object
0xC0CD0088	ECM_ERROR_EMC_EXT_SYNC_OBJ_IS_NOT_MAPPED_IN_ONE_CYCLIC_CMD	Parts of Ext Sync object are not mapped within the same cyclic command

Hexadecimal Value	Definition	Description
0xC0CD0089	ECM_ERROR_EMC_UNSUP-PORTED_FMMU_MAP-PING_OF_EXT_SYNC_OBJ_RECORD	Unsupported FMMU mapping of Ext Sync object detected
0xC0CD008A	ECM_ERROR_EMC_EXT_SYNC_REQUIRES_ADJUST_EXT_SYNC_CMD	Unicast Ext Sync control (APWR/FPWR 0x910) is required
0xC0CD008B	ECM_ERROR_EMC_EXT_SYNC_CMD_DOES_NOT_MATCH_XRMW_CMD	Unicast Ext Sync control does not match xRMW command
0xC0CD008C	ECM_ERROR_EMC_EXT_SYNC_REQUIRES_XRMW_CMD	Ext Sync requires DC configuration (xRMW command to 0x910)
0xC0CD008D	ECM_ERROR_EMC_EXPLICIT_DEV_IDENT_FAILED_ALSTATUS	Explicit Device identification via ALSTATUS failed
0xC0CD008E	ECM_ERROR_EMC_EXPLICIT_DEV_IDENT_FAILED_REG	Explicit Device identification via register failed
0xC0CD008F	ECM_ERROR_EMC_COPY_INFOS_FOUND_AT_UNMAPPED_RECEIVE_DATA	CopyInfos found at unmapped receive data
0xC0CD0090	ECM_ERROR_EMC_COPY_INFO_RECEIVE_DATA_AREA_NOT_MATCHING	CopyInfo receive data area is not matching
0xC0CD0091	ECM_ERROR_EMC_SDO_UPLOAD_TOO_LONG	SDO Upload data too long
0xC0CD0092	ECM_ERROR_EMC_SDO_UPLOAD_TOO_SHORT	SDO Upload data too short
0xC0CD0093	ECM_ERROR_EMC_SDO_UPLOAD_COM-PARE_DOES_NOT_MATCH_EXPECTATION	SDO Upload compare does not match expectation
0xC0CD0094	ECM_ERROR_EMC_SOE_READ_TOO_LONG	SoE Read IDN data too long
0xC0CD0095	ECM_ERROR_EMC_SOE_READ_TOO_SHORT	SoE Read IDN data too short
0xC0CD0096	ECM_ERROR_EMC_SOE_READ_COM-PARE_DOES_NOT_MATCH_EXPECTATION	SoE Read compare does not match expectation
0xC0CD0097	ECM_ERROR_EMC_REG_INITCMD_COM-PARE_DOES_NOT_MATCH_EXPECTATION	Register read compare does not match expectation
0xC0CD0098	ECM_ERROR_EMC_REDUN-DANCY_PORT_ONLY_POS-SIBLE_ONCE	Redundancy port can only be placed once into configuration
0xC0CD0099	ECM_ERROR_EMC_STARTUP_SCAN_SII_FAILED	Startup scan of SII failed
0xC0CD009A	ECM_ERROR_EMC_STARTUP_VERIFY_SII_FAILED	Startup verification of SII failed
0xC0CD009B	ECM_ERROR_EMC_MAIN_PORT_NOT_CONNECTED	Main port not connected during topology scan

Hexadecimal Value	Definition	Description
0xC0CD009C	ECM_ERROR_EMC_BUS_SCAN_TOO_MANY_SLAVES	Bus scan detects too many slaves
0xC0CD009D	ECM_ERROR_EMC_BUS_SCAN_SPLIT_RING_NOT_SUPPORTED	Bus Scan detects unsupported split ring topology
0xC0CD009E	ECM_ERROR_EMC_BUS_SHUT-DOWN	Bus is shutting down
0xC0CD009F	ECM_ERROR_EMC_MASTER_ADDRESS_NOT_ALLOWED_AS_STATION_ADDRESS	Master address (0) is not allowed as station address
0xC0CD00A0	ECM_ERROR_EMC_FIRST_STATION_HAS_INVALID_PORT_0	First station has invalid port 0
0xC0CD00A1	ECM_ERROR_EMC_STATION_HAS_INVALID_PORT	Station has invalid port
0xC0CD00A2	ECM_ERROR_EMC_STATION_HAS_NOT_LISTED_STATION_ADDRESS_IN_PORT	Station has not listed station address in port
0xC0CD00A3	ECM_ERROR_EMC_PORT_CONNECTION_BETWEEN_STATIONS_DOES_NOT_MATCH	Port connection between stations does not match
0xC0CD00A4	ECM_ERROR_EMC_STATION_HAS_ALREADY_USED_STATION_ADDRESS_IN_PORT	Station has already used station address in port
0xC0CD00A5	ECM_ERROR_EMC_INVALID_SM_PHYS_START_ADDRESS	Invalid Sm physical start address
0xC0CD00A6	ECM_ERROR_EMC_DC_TOPOLOGY_ON_REDUNDANCY_PORT_NOT_SUPPORTED	DC topology on redundancy port connection not supported. DC slaves having AutoIncrement positions behind redundancy port
0xC0CD00A7	ECM_ERROR_EMC_SM_ASSIGN_PDO_ALREADY_ADDED	Sm AssignPdo already added
0xC0CD00A8	ECM_ERROR_EMC_BASE_SYNC_OFFSET_PERCENTAGE_OUT_OF_RANGE	Base Sync Offset percentage out of range
0xC0CF0001	ECM_ERROR_COE_INITIALIZATION_ERROR	CoE: Initialization Error
0xC0CF0002	ECM_ERROR_COE_INVALID_TRANSFER_HANDLE	CoE: Invalid transfer handle used
0xC0CF0003	ECM_ERROR_COE_NO_MAILBOX_AVAILABLE	CoE. No mailbox available
0xC0CF0004	ECM_ERROR_COE_INVALID_TRANSFER_STATE	CoE: Invalid transfer state
0xC0CF0005	ECM_ERROR_COE_TRANSFER_SEGMENT_TOO_LONG	CoE: Transfer segment is too long
0xC0CF0006	ECM_ERROR_COE_SHUTTING_DOWN	CoE is shutting down.
0xC0CF0007	ECM_ERROR_COE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES	CoE: Maximum total bytes is smaller than actual total bytes.
0xC0CF0008	ECM_ERROR_COE_MAILBOX_TRANSMIT_FAILED	CoE: Mailbox transmit failed

Hexadecimal Value	Definition	Description
0xC0CF0009	ECM_ERROR_COE_TRANSFER_ABORTED	CoE: Transfer has been aborted.
0xC0CF000A	ECM_ERROR_COE_SDOINFO_INITIALIZATION_ERROR	0xC0CF000B
0xC0CF000C	ECM_ERROR_COE_PROTOCOL_ERROR	CoE Protocol Error
0xC0CF000D	ECM_ERROR_COE_NO_AOE_AVAILABLE	CoE: No AoE available
0xC0CF000F	ECM_ERROR_COE_INVALID_SLAVE_STATION_ADDRESS	CoE: Invalid slave station address
0xC0CF8000	ECM_ERROR_COE_ABORTCODE_TOGGLE_BIT_NOT_ALTERNATED	SDO Abort Code: Toggle Bit not alternated
0xC0CF8001	ECM_ERROR_COE_ABORTCODE_COMMAND_SPECIFIER_NOT_VALID	SDO Abort Code: Command specifier not valid
0xC0CF8002	ECM_ERROR_COE_ABORTCODE_PROTOCOL_TIMEOUT	SDO Abort Code: Protocol Timeout
0xC0CF8003	ECM_ERROR_COE_ABORTCODE_OUT_OF_MEMORY	SDO Abort Code: Out Of Memory
0xC0CF8004	ECM_ERROR_COE_ABORTCODE_UNSUPPORTED_ACCESS	SDO Abort Code: Unsupported access
0xC0CF8005	ECM_ERROR_COE_ABORTCODE_OBJECT_IS_WRITE_ONLY	SDO Abort Code: Object is write only
0xC0CF8006	ECM_ERROR_COE_ABORTCODE_OBJECT_IS_READ_ONLY	SDO Abort Code: Object is read only
0xC0CF8007	ECM_ERROR_COE_ABORTCODE_SUBINDEX_CANNOT_BE_WRITTEN_SI0_NZ	SDO Abort Code: Subindex cannot be written if subindex 0 is not zero
0xC0CF8008	ECM_ERROR_COE_ABORTCODE_COMPLETE_ACCESS_NOT_SUPPORTED	SDO Abort Code: Complete access not supported
0xC0CF8009	ECM_ERROR_COE_ABORTCODE_OBJECT_LENGTH_EXCEEDS_MAILBOX_SIZE	SDO Abort Code: Object length exceeds mailbox size
0xC0CF800A	ECM_ERROR_COE_ABORTCODE_OBJECT_MAPPED_TO_RXPDO_NO_WRITE	SDO Abort Code: Object mapped to RxPDO, SDO Download blocked
0xC0CF800B	ECM_ERROR_COE_ABORTCODE_OBJECT_DOES_NOT_EXIST	SDO Abort Code: Object does not exist
0xC0CF800C	ECM_ERROR_COE_ABORTCODE_OBJECT_CANNOT_BE_PDO_MAPPED	SDO Abort Code: Object cannot be mapped to PDO
0xC0CF800D	ECM_ERROR_COE_ABORTCODE_PDO_LENGTH_WOULD_EXCEED	SDO Abort Code: PDO Length would exceed maximum size
0xC0CF800E	ECM_ERROR_COE_ABORTCODE_GEN_PARAM_INCOMPATIBILITY	SDO Abort Code: General parameter incompatibility

Hexadecimal Value	Definition	Description
0xC0CF800F	ECM_ERROR_COE_ABORT-CODE_ACCESS_FAILED_DUE_TO_HW_ERROR	SDO Abort Code: Access failed due to hardware error
0xC0CF8010	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_DOES_NOT_MATCH	SDO Abort Code: Data type does not match
0xC0CF8011	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_LENGTH_TOO_LONG	SDO Abort Code: Data type length too long
0xC0CF8012	ECM_ERROR_COE_ABORT-CODE_DATA-TYPE_LENGTH_TOO_SHORT	SDO Abort Code: Data type length too short
0xC0CF8013	ECM_ERROR_COE_ABORT-CODE_SUB-INDEX_DOES_NOT_EXIST	SDO Abort Code: Subindex does not exist
0xC0CF8014	ECM_ERROR_COE_ABORT-CODE_RANGE_OF_PARAM-ETER_EXCEEDED	SDO Abort Code: Range of parameter exceeded
0xC0CF8015	ECM_ERROR_COE_ABORT-CODE_VALUE_OF_PARAM_WRITTEN_TOO_HIGH	SDO Abort Code: Value of parameter written too high
0xC0CF8016	ECM_ERROR_COE_ABORT-CODE_VALUE_OF_PARAM_WRITTEN_TOO_LOW	SDO Abort Code: Value of parameter written too low
0xC0CF8017	ECM_ERROR_COE_ABORT-CODE_MIN_VALUE_IS_LESS_THAN_MAX_VALUE	SDO Abort Code: Minimum value is less than maximum value
0xC0CF8018	ECM_ERROR_COE_ABORT-CODE_GENERAL_ERROR	SDO Abort Code: General Error
0xC0CF8019	ECM_ERROR_COE_ABORT-CODE_NO_TRANSFER_TO_APP	SDO Abort Code: Data cannot be transferred or stored to the application
0xC0CF801A	ECM_ERROR_COE_ABORT-CODE_LOCAL_CONTROL	SDO Abort Code: Data cannot be transferred or stored to the application because of local control
0xC0CF801B	ECM_ERROR_COE_ABORT-CODE_NO_TRANSFER_DUE_TO_CURRENT_STATE	SDO Abort Code: Data cannot be transferred or stored to the applcation because of the present device state
0xC0CF801C	ECM_ERROR_COE_ABORT-CODE_NO_OBJECT_DIC-TIONARY_PRESENT	SDO Abort Code: Object dictionary dynamic generation fails or no object dictionary is present
0xC0CF801D	ECM_ERROR_COE_ABORT-CODE_UNKNOWN_ABORT_CODE	SDO Abort Code: Unknown abort code
0xC0CF801E	ECM_ERROR_COE_ABORT-CODE_GEN_INTERNAL_COMPAT	SDO Abort Code: General internal incompatibility in the device
0xC0D00001	ECM_ERROR_EOE_INVALID_MAC_ADDRESS	Invalid MAC address
0xC0D00002	ECM_ERROR_EOE_INVALID_CALLBACK_TYPE	Invalid callback type
0xC0D00003	ECM_ERROR_EOE_DESTINA-TION_UNREACHABLE	Destination unreachable

Hexadecimal Value	Definition	Description
0xC0D00004	ECM_ERROR_EOE_INVALID_EOE_RESPONSE	Invalid EoE Response
0xC0D00005	ECM_ERROR_EOE_UNKNOWN_ERROR	SetIPParam/SetFilterParam: Unknown error
0xC0D00006	ECM_ERROR_EOE_UNSPECIFIED_ERROR	SetIPParam/SetFilterParam: Unspecified Error
0xC0D00007	ECM_ERROR_EOE_UNSUPPORTED_FRAME_TYPE	SetIPParam/SetFilterParam: Unsupported frame type
0xC0D00008	ECM_ERROR_EOE_NO_IP_SUPPORT	SetIPParam/SetFilterParam: No IP support
0xC0D00009	ECM_ERROR_EOE_DHCP_NOT_SUPPORTED	SetIPParam/SetFilterParam: DHCP not supported
0xC0D0000A	ECM_ERROR_EOE_NO_FILTER_SUPPORT	SetIPParam/SetFilterParam: No filter supported
0xC0D0000B	ECM_ERROR_EOE_TIMEOUT	EoE Timeout
0xC0D0000C	ECM_ERROR_EOE_SHUTTING_DOWN	EoE is shutting down
0xC0D0000D	ECM_ERROR_EOE_MASTER_ADDRESS_NOT_ALLOWED	EoE: Master address is not allowed to use here
0xC0D0000E	ECM_ERROR_EOE_CONFIGURATION_IS_NOT_OPEN	EoE: Configuration is not open
0xC0D0000F	ECM_ERROR_EOE_CONFIGURATION_IS_ALREADY_OPEN	EoE: Configuration is already open
0xC0D00010	ECM_ERROR_EOE_DUPLICATE_IP_ADDRESS	EoE: Duplicate IP address
0xC0D00011	ECM_ERROR_EOE_DUPLICATE_MAC_ADDRESS_ON_MULTIPLE_PORTS	EoE: Duplicate MAC address on multiple ports
0xC0D00012	ECM_ERROR_EOE_FRAME_TOO_LARGE	EoE: Frame too large
0xC0D00013	ECM_ERROR_EOE_IF_INITIALIZATION_ERROR	EoE: Interface initialization error
0xC0D00014	ECM_ERROR_EOE_IF_NO_FRAME_AVAILABLE	EoE: No Frame available
0xC0D00015	ECM_ERROR_EOE_LINK_DOWN	EoE: Link down
0xC0D10002	ECM_ERROR_FOE_ERROR_UNKNOWN_ERROR	-
0xC0D10003	ECM_ERROR_FOE_INVALID_TRANSFER_HANDLE	FoE: Invalid transfer handle
0xC0D10004	ECM_ERROR_FOE_INVALID_TRANSFER_STATE	FoE: Invalid transfer state
0xC0D10005	ECM_ERROR_FOE_INVALID_SLAVE_STATION_ADDRESS	FoE: Invalid slave station address
0xC0D10006	ECM_ERROR_FOE_WRONG_SLAVE_STATE	FoE: Wrong slave state
0xC0D10007	ECM_ERROR_FOE_NO_MAILBOX_AVAILABLE	FoE: No mailbox available
0xC0D10008	ECM_ERROR_FOE_TRANSFER_ABORTED	FoE: Transfer has been aborted

Hexadecimal Value	Definition	Description
0xC0D10009	ECM_ERROR_FOE_PROTOCOL_TIMEOUT	FoE: Protocol Timeout
0xC0D1000A	ECM_ERROR_FOE_TRANSFER_SEGMENT_TOO_LONG	FoE: Transfer segment is too long
0xC0D1000B	ECM_ERROR_FOE_MAILBOX_TRANSMIT_FAILED	FoE: Mailbox transmit failed
0xC0D1000C	ECM_ERROR_FOE_FILENAME_TOO_LONG	FoE: Filename is too long
0xC0D1000D	ECM_ERROR_FOE_BUFFER_EXCEEDED	FoE: Buffer is exceeded
0xC0D1000E	ECM_ERROR_FOE_FIRST_SEGMENT_SHOULD_NOT_BE_EMPTY	FoE: First segment should not be empty
0xC0D1000F	ECM_ERROR_FOE_SEGMENT_SHOULD_BE_EMPTY	FoE: Segment should be empty
0xC0D18000	ECM_ERROR_FOE_ERROR_NOT_DEFINED	FoE: Error Response: not defined
0xC0D18001	ECM_ERROR_FOE_ERROR_NOT_FOUND	FoE: Error Response: Not Found
0xC0D18002	ECM_ERROR_FOE_ACCESS_DENIED	FoE: Error Response: Access Denied
0xC0D18003	ECM_ERROR_FOE_ERROR_DISK_FULL	FoE: Error Response: Disk full
0xC0D18004	ECM_ERROR_FOE_ERROR_ILLEGAL	FoE: Error Response: Illegal
0xC0D18005	ECM_ERROR_FOE_ERROR_PACKET_NUMBER_WRONG	FoE: Error Response: Packet number is wrong
0xC0D18006	ECM_ERROR_FOE_ERROR_ALREADY_EXISTS	FoE: Error Response: Already exists
0xC0D18007	ECM_ERROR_FOE_ERROR_NO_USER	FoE: Error Response: No User
0xC0D18008	ECM_ERROR_FOE_ERROR_BOOTSTRAP_ONLY	FoE: Acces to specified file is only allowed in BOOT state
0xC0D18009	ECM_ERROR_FOE_ERROR_NOT_BOOTSTRAP	FoE: Access to specified file is only allowed when in PREOP, SAFEOP or OP
0xC0D1800A	ECM_ERROR_FOE_ERROR_NO_RIGHTS	FoE: No Rights
0xC0D1800B	ECM_ERROR_FOE_ERROR_PROGRAM_ERROR	FoE: Program Error
0xC0D20001	ECM_ERROR_SOE_UNKNOWN_SOE_ERROR	SoE: Unknown SoE Error
0xC0D20002	ECM_ERROR_SOE_INITIALIZATION_ERROR	SoE: Initialization error
0xC0D20003	ECM_ERROR_SOE_INVALID_TRANSFER_HANDLE	SoE: Invalid transfer handle
0xC0D20004	ECM_ERROR_SOE_NO_MAILBOX_AVAILABLE	SoE: No Mailbox available
0xC0D20005	ECM_ERROR_SOE_INVALID_TRANSFER_STATE	SoE: Invalid transfer state

Hexadecimal Value	Definition	Description
0xC0D20006	ECM_ERROR_SOE_TRANSFER_SEGMENT_TOO_LONG	SoE: Transfer segment is too long
0xC0D20007	ECM_ERROR_SOE_SHUTTING_DOWN	SoE is shutting down
0xC0D20008	ECM_ERROR_SOE_MAX_TOTAL_BYTES_SMALLER_THAN_ACTUAL_TOTAL_BYTES	SoE: Maximum total bytes is smaller than actual total bytes
0xC0D20009	ECM_ERROR_SOE_MAILBOX_TRANSMIT_FAILED	SoE: Mailbox transmit failed
0xC0D2000A	ECM_ERROR_SOE_INVALID_SOE_HEADER	SoE: Invalid SoE header
0xC0D2000B	ECM_ERROR_SOE_PROTOCOL_TIMEOUT	SoE: Protocol Timeout
0xC0D2000C	ECM_ERROR_SOE_PROTOCOL_ERROR	SoE: Protocol Error
0xC0D2000D	ECM_ERROR_SOE_TRANSFER_ABORTED	SoE: Transfer has been aborted
0xC0D2000E	ECM_ERROR_SOE_WRONG_SLAVE_STATE	SoE: Wrong slave state
0xC0D2000F	ECM_ERROR_SOE_NO_AOE_AVAILABLE	SoE: No AoE available
0xC0D20010	ECM_ERROR_SOE_INVALID_SLAVE_STATION_ADDRESS	SoE: Invalid slave station address
0xC0D21001	ECM_ERROR_SOE_SSC_NO_IDN	SoE: No IDN
0xC0D21009	ECM_ERROR_SOE_SSC_INVALID_ACCESS_TO_ELEMENT_1	SoE: Invalid access to element 1
0xC0D22001	ECM_ERROR_SOE_SCC_NO_NAME	SoE: IDN has no name
0xC0D22002	ECM_ERROR_SOE_SSC_NAME_TRANSMISSION_IS_TOO_SHORT	SoE: Name transmission is too short
0xC0D22003	ECM_ERROR_SOE_SSC_NAME_TRANSMISSION_IS_TOO_LONG	SoE: Name transmission is too long
0xC0D22004	ECM_ERROR_SOE_SSC_NAME_CANNOT_BE_CHANGED	SoE: Name cannot be changed
0xC0D22005	ECM_ERROR_SOE_SSC_NAME_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Name is write protected at this time
0xC0D23002	ECM_ERROR_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_SHORT	SoE: Attribute transmission is too short
0xC0D23003	ECM_ERROR_SOE_SSC_ATTRIBUTE_TRANSMISSION_IS_TOO_LONG	SoE: Attribute transmission is too long
0xC0D23004	ECM_ERROR_SOE_SSC_ATTRIBUTE_CANNOT_BE_CHANGED	SoE: Attribute cannot be changed
0xC0D23005	ECM_ERROR_SOE_SSC_ATTRIBUTE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Attribute is write protected at this time
0xC0D24001	ECM_ERROR_SOE_SSC_NO_UNIT	SoE: IDN has no unit

Hexadecimal Value	Definition	Description
0xC0D24002	ECM_ERROR_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_SHORT	SoE: Unit transmission is too short
0xC0D24003	ECM_ERROR_SOE_SSC_UNIT_TRANSMISSION_IS_TOO_LONG	SoE: Unit transmission is too long
0xC0D24004	ECM_ERROR_SOE_SSC_UNIT_CANNOT_BE_CHANGED	SoE: Unit cannot be changed
0xC0D24005	ECM_ERROR_SOE_SSC_UNIT_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Unit is write protected at this time
0xC0D25001	ECM_ERROR_SOE_SSC_NO_MAXIMUM_VALUE	SoE: IDN has no maximum value
0xC0D25002	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT	SoE: Minimum value transmission is too short
0xC0D25003	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_TRANSMISSION_IS_TOO_LONG	SoE: Minimum value transmission is too long
0xC0D25004	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_CANNOT_BE_CHANGED	SoE: Minimum value cannot be changed
0xC0D25005	ECM_ERROR_SOE_SSC_MINIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Mimum value is write protected at this time
0xC0D26001	ECM_ERROR_SOE_SSC_NO_MAXIMUM_VALUE	SoE: IDN has no maximum value
0xC0D26002	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_SHORT	SoE: Maximum value transmission is too short
0xC0D26003	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_TRANSMISSION_IS_TOO_LONG	SoE: Maximum value transmission is too long
0xC0D26004	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_CANNOT_BE_CHANGED	SoE: Maximum value cannot be changed
0xC0D26005	ECM_ERROR_SOE_SSC_MAXIMUM_VALUE_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: Maximum value is write protected at this time
0xC0D27002	ECM_ERROR_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_SHORT	SoE: OpData transmission is too short
0xC0D27003	ECM_ERROR_SOE_SSC_OPDATA_TRANSMISSION_IS_TOO_LONG	SoE: OpData transmission is too long
0xC0D27004	ECM_ERROR_SOE_SSC_OPDATA_CANNOT_BE_CHANGED	SoE: OpData cannot be changed
0xC0D27005	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_THIS_TIME	SoE: OpData is write protected at this time
0xC0D27006	ECM_ERROR_SOE_SSC_OPDATA_IS_LOWER_THAN_MINIMUM_VALUE	SoE: OpData is lower than minimum value

Hexadecimal Value	Definition	Description
0xC0D27007	ECM_ERROR_SOE_SSC_OPDATA_IS_HIGHER_THAN_MAXIMUM_VALUE	SoE: OpData is higher than maximum value
0xC0D27008	ECM_ERROR_SOE_SSC_OPDATA_IS_INVALID	SoE: OpData is invalid
0xC0D27009	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_BY_PASSWORD	SoE: OpData is write protected by password
0xC0D2700A	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_CYCLICALLY_CONFIGURED	SoE: OpData is write protected due to being cyclically configured
0xC0D2700B	ECM_ERROR_SOE_SSC_OPDATA_INVALID_DIRECT_ADDRESSING	SoE: Invalid direct addressing
0xC0D2700C	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_DUE_OTHER_SETTINGS	SoE: OpData is write protected due to other settings.
0xC0D2700D	ECM_ERROR_SOE_SSC_OPDATA_INVALID_FLOATING_POINT_NUMBER	SoE: Invalid floating point number
0xC0D2700E	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_PARAMETERIZATION_LEVEL	SoE: OpData is write protected at parameterization level
0xC0D2700F	ECM_ERROR_SOE_SSC_OPDATA_IS_WRITE_PROTECTED_AT_OPERATION_LEVEL	SoE: OpData is write protected at operation level
0xC0D27010	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_ALREADY_ACTIVE	SoE: Procedure command already active
0xC0D27011	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_INTERRUPTIBLE	SoE: Procedure command not interruptible
0xC0D27012	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_AT_THIS_TIME	SoE: Procedure command is not executable at this time
0xC0D27013	ECM_ERROR_SOE_SSC_OPDATA_PROCEDURE_COMMAND_NOT_EXECUTABLE_INVALID_PARAM	SoE: Procedure command is not executable due to invalid parameter
0xC0D4005C	ECM_ERROR_ENI_NO_SLAVES_IN_ENI	ENI does not contain any slaves
0xC0D50001	ECM_ERROR_ALSTATCODE_UNSPECIFIED_ERROR	ALStatusCode: Unspecified error
0xC0D50002	ECM_ERROR_ALSTATCODE_NO_MEMORY	ALStatusCode: No memory
0xC0D50003	ECM_ERROR_ALSTATCODE_INVALID_DEVICE_SETUP	ALStatusCode: Invalid Device Setup
0xC0D50011	ECM_ERROR_ALSTATCODE_INVALID_REQUESTED_STATE_CHANGE	ALStatusCode: Invalid requested state change

Hexadecimal Value	Definition	Description
0xC0D50012	ECM_ERROR_ALSTAT-CODE_UNKNOWN_REQUESTED_STATE	ALStatusCode: Unknown requested state
0xC0D50013	ECM_ERROR_ALSTAT-CODE_BOOTSTRAP_NOT_SUP-PORTED	ALStatusCode: Bootstrap not supported
0xC0D50014	ECM_ERROR_ALSTAT-CODE_NO_VALID_FIRMWARE	ALStatusCode: No valid firmware
0xC0D50015	ECM_ERROR_ALSTAT-CODE_INVALID_BOOT_MAILBOX_CONFIGURATION	ALStatusCode: Invalid BOOT mailbox configuration
0xC0D50016	ECM_ERROR_ALSTAT-CODE_INVALID_PREOP_MAILBOX_CONFIGURATION	ALStatusCode: Invalid PREOP mailbox configuration
0xC0D50017	ECM_ERROR_ALSTAT-CODE_INVALID_SYNC_MAN-AGER_CONFIGURATION	ALStatusCode: Invalid sync manager configuration
0xC0D50018	ECM_ERROR_ALSTAT-CODE_NO_VALID_INPUTS_AVAIL-ABLE	ALStatusCode: No valid inputs available
0xC0D50019	ECM_ERROR_ALSTAT-CODE_NO_VALID_OUTPUTS	ALStatusCode: No valid outputs
0xC0D5001A	ECM_ERROR_ALSTAT-CODE_SYNCHRONIZA-TION_ERROR	ALStatusCode: Synchronization error
0xC0D5001B	ECM_ERROR_ALSTAT-CODE_SYNC_MAN-AGER_WATCHDOG	ALStatusCode: Sync Manager watchdog
0xC0D5001C	ECM_ERROR_ALSTAT-CODE_INVALID_SYNC_MAN-AGER_TYPES	ALStatusCode: Invalid Sync Manager Types
0xC0D5001D	ECM_ERROR_ALSTAT-CODE_INVALID_OUTPUT_CON-FIGURATION	ALStatusCode: Invalid output configuration
0xC0D5001E	ECM_ERROR_ALSTAT-CODE_INVALID_INPUT_CONFIGU-RATION	ALStatusCode: Invalid input configuration
0xC0D5001F	ECM_ERROR_ALSTAT-CODE_INVALID_WATCHDOG_CONFIGURATION	ALStatusCode: Invalid Watchdog configuration
0xC0D50020	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_COLD_START	ALStatusCode: Slave needs cold start
0xC0D50021	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_INIT	ALStatusCode: Slave needs INIT
0xC0D50022	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_PREOP	ALStatusCode: slave needs PREOP
0xC0D50023	ECM_ERROR_ALSTAT-CODE_SLAVE_NEEDS_SAFEOp	ALStatusCode: slave needs SAFEOP
0xC0D50024	ECM_ERROR_ALSTAT-CODE_INVALID_INPUT_MAPPING	ALStatusCode: Invalid Input Mapping

Hexadecimal Value	Definition	Description
0xC0D50025	ECM_ERROR_ALSTAT-CODE_INVALID_OUTPUT_MAP-PING	ALStatusCode: Invalid Output Map-ping
0xC0D50026	ECM_ERROR_ALSTAT-CODE_INCONSISTENT_SET-TINGS	ALStatusCode: Inconsistent settings
0xC0D50027	ECM_ERROR_ALSTAT-CODE_FREERUN_NOT_SUP-PORTED	ALStatusCode: FreeRun not sup-ported
0xC0D50028	ECM_ERROR_ALSTAT-CODE_SYNCMODE_NOT_SUP-PORTED	ALStatusCode: SyncMode not sup-ported
0xC0D50029	ECM_ERROR_ALSTAT-CODE_FREERUN_NEEDS_3BUFF ER_MODE	ALStatusCode: FreeRun needs 3Buffer mode
0xC0D5002A	ECM_ERROR_ALSTAT-CODE_BACK-GROUND_WATCHDOG	ALStatusCode: Background Watchdog
0xC0D5002B	ECM_ERROR_ALSTAT-CODE_NO_VALID_INPUTS_AND_ OUTPUTS	ALStatusCode: No valid Inputs and Outputs
0xC0D5002C	ECM_ERROR_ALSTAT-CODE_FATAL_SYNC_ERROR	ALStatusCode: Fatal Sync error
0xC0D5002D	ECM_ERROR_ALSTAT-CODE_NO_SYNC_ERROR	ALStatusCode: No Sync error
0xC0D50030	ECM_ERROR_ALSTAT-CODE_INVALID_DC_SYNC_CON-FIGURATION	ALStatusCode: Invalid DC SYNC configuration
0xC0D50031	ECM_ERROR_ALSTAT-CODE_INVALID_DC_LATCH_CON-FIGURATION	ALStatusCode: Invalid DC Latch configuration
0xC0D50032	ECM_ERROR_ALSTAT-CODE_PLL_ERROR	ALStatusCode: PLL error
0xC0D50033	ECM_ERROR_ALSTAT-CODE_DC_SYNC_IO_ERROR	ALStatusCode: DC Sync IO error
0xC0D50034	ECM_ERROR_ALSTAT-CODE_DC_SYNC_TIMEOUT_ERR OR	ALStatusCode: DC Sync Timeout Error
0xC0D50035	ECM_ERROR_ALSTAT-CODE_DC_INVALID_SYNC_CYCL E_TIME	ALStatusCode: DC Invalid Sync Cycle Time
0xC0D50036	ECM_ERROR_ALSTAT-CODE_DC_SYNC0_CYCLE_TIME	ALStatusCode: DC Sync0 Cycle Time
0xC0D50037	ECM_ERROR_ALSTAT-CODE_DC_SYNC1_CYCLE_TIME	ALStatusCode: DC Sync1 Cycle Time
0xC0D50041	ECM_ERROR_ALSTAT-CODE_MBX_AOE	ALStatusCode: MBX_AOE
0xC0D50042	ECM_ERROR_ALSTAT-CODE_MBX_EOE	ALStatusCode: MBX_EOE
0xC0D50043	ECM_ERROR_ALSTAT-CODE_MBX_COE	ALStatusCode: MBX_COE

Hexadecimal Value	Definition	Description
0xC0D50044	ECM_ERROR_ALSTAT-CODE_MBX_FOE	ALStatusCode: MBX_FOE
0xC0D50045	ECM_ERROR_ALSTAT-CODE_MBX_SOE	ALStatusCode: MBX_SOE
0xC0D5004F	ECM_ERROR_ALSTAT-CODE_MBX_VOE	ALStatusCode: MBX_VOE
0xC0D50050	ECM_ERROR_ALSTAT-CODE_EEPROM_NO_ACCESS	ALStatusCode: EEPROM no access
0xC0D50051	ECM_ERROR_ALSTAT-CODE_EEPROM_ERROR	ALStatusCode: EEPROM error
0xC0D50060	ECM_ERROR_ALSTAT-CODE_SLAVE_RESTARTED_LOCALLY	ALStatusCode: Slave restarted locally
0xC0D50061	ECM_ERROR_ALSTAT-CODE_DEVICE_IDENTIFICA-TION_VALUE_UPDATED	ALStatusCode: Device identificatin value updated
0xC0D500F0	ECM_ERROR_ALSTAT-CODE_APPLICATION_CON-TROLLER_AVAILABLE	ALStatusCode: Application controller available
0xC0D58000	ECM_ERROR_ALSTAT-CODE_VENDOR_SPE-CIFIC_CODE_START	Begin of vendor-specific ALStatus-Code mapping
0xC0D5FFFF	ECM_ERROR_ALSTAT-CODE_VENDOR_SPE-CIFIC_CODE_END	End of vendor-specific ALStatus-Code mapping
0xC0D60001	ECM_ERROR_IF_COE_SUP-PORT_NOT_AVAILABLE	CoE support is not configured
0xC0D60002	ECM_ERROR_IF_SOE_SUP-PORT_NOT_AVAILABLE	SoE support is not configured
0xC0D60003	ECM_ERROR_IF_FOE_SUP-PORT_NOT_AVAILABLE	FoE support is not configured
0xC0D60004	ECM_ERROR_IF_AOE_SUP-PORT_NOT_AVAILABLE	AoE support is not configured
0xC0D60005	ECM_ERROR_IF_INVALID_TRANS PORT_TYPE	Invalid transfer type
0xC0D60006	ECM_ERROR_IF_SOE_INVALID_D RIVE_NO	SoE: Invalid drive number
0xC0D60007	ECM_ERROR_IF_SOE_INVALID_E LEMENT_FLAGS	SoE: invalid element flags
0xC0D60008	ECM_ERROR_IF_INVALID_SOE_T RANSFER_ID	SoE: Invalid transfer ID
0xC0D60009	ECM_ERROR_IF_TRANSFER_AB ORTED	Transfer aborted
0xC0D6000A	ECM_ERROR_IF_OUT_OF_PACKE TS	Out of packets
0xC0D6000B	ECM_ERROR_IF_OUT_OF_TRAN SFER_CONTEXTS	Out of transfer contexts
0xC0D6000C	ECM_ERROR_IF_INVALID_SUB-INDEX_FOR_COMPLETE_ACCESs	CoE: Invalid subindex for Complete Access

Hexadecimal Value	Definition	Description
0xC0D6000D	ECM_ERROR_IF_INVALID_COE_TRANSFER_ID	CoE: Invalid transfer ID
0xC0D6000E	ECM_ERROR_IF_INVALID_COE_SDOINFO_LISTTYPE	CoE: Invalid SDOINFO ListType
0xC0D6000F	ECM_ERROR_IF_FILE_READ_ERROR	File Read Error
0xC0D60010	ECM_ERROR_IF_COULD_NOT_OPEN_FILE	Could not open file
0xC0D60011	ECM_ERROR_IF_INVALID_CONFIG_NXD	Invalid config.nxd detected
0xC0D60012	ECM_ERROR_IF_CONFIG_NXD_WITHOUT_SLAVES	Config.nxd does not contain any slaves
0xC0D60013	ECM_ERROR_IF_INVALID_FILE_NAME	Invalid file name
0xC0D60014	ECM_ERROR_IF_INVALID_FOE_TRANSFER_ID	Invalid FoE transfer id
0xC0D60015	ECM_ERROR_IF_INVALID_GET_TOPOLOGY_TRANSFER_ID	Invalid GetTopology transfer id

1.7.3.4.4 CM589-PNIO(-4) errors

Error class	Component d1	Device d2	Module d3	Channel d4	Error identifier	Error message	Remedy	Error number	Online text
E4	1..6	255	2	0	45	CM589 PNIO device communication module has no connection to network	Check cabeling	184487981	E4: Ext. [1..6] [COUPLER], E4: Int. [COUPLER]

1.7.3.4.5 CM598-CN errors

CM598-CN Errors can be read out with function blocks:

↳ *Chapter 1.5.4.7.1.9 “CANOM_NODE_DIAG_EXT” on page 937*

↳ *Chapter 1.5.4.7.1.14 “CANOM_STATE ” on page 952*

↳ *Chapter 1.5.4.7.1.15 “CANOM_SYS_DIAG” on page 957*.

1.7.3.4.6 AC500-S: errors from safety CPU and safety I/O modules

Table 790: Common error messages for SM560-S / SM560-S-FD-1 / SM560-S-FD-4 safety CPUs

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E2	1 ... 4	255	30	1	0	Operation finished.	Change safety PLC switch address setting or remove memory card from non-safety PLC. Restart safety PLC. If this error persists, replace safety PLC.
E2	1 ... 4	255	30	1	1	Wrong user data	Delete user data from safety PLC. Restart safety PLC and write user data again.
E2	1 ... 4	255	30	1	2	Internal PROFIsafe initialization error	Restart safety PLC. If this error persists, replace safety PLC. Contact ABB technical support.
E2	1 ... 4	255	30	1	12	Flash read error	Restart safety PLC. If this error persists, replace safety PLC. Contact ABB technical support.
E2	1 ... 4	255	30	1	18	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	1	28	Boot project download error	Reload boot project. If this error persists, replace safety PLC.
E2	1 ... 4	255	30	1	40	Wrong firmware version	Update safety PLC firmware. Restart safety PLC. If this error persists, replace safety PLC.
E2	1 ... 4	255	30	1	43	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	1	48	Overvoltage or undervoltage detected	Restart safety PLC. Check safety PLC setting for power supply error. If this error persists, replace safety PLC.
E2	1 ... 4	255	30	1	52	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	0	User program triggered safe stop	Check user program
E2	1 ... 4	255	30	2	1	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	2	Internal PROFIsafe error	Restart safety PLC. If this error persists, replace safety PLC. Contact ABB technical support.
E2	1 ... 4	255	30	2	3	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	10	Internal error	Contact ABB technical support. Replace safety PLC.

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E2	1 ... 4	255	30	2	13	Flash write error	Restart safety PLC. If this error persists, replace safety PLC. Contact ABB technical support.
E2	1 ... 4	255	30	2	17	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	18	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	19	Checksum error has occurred in safety PLC.	Restart safety PLC. If this error persists, replace safety PLC.
E2	1 ... 4	255	30	2	25	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	37	Cycle time error in safety PLC	Check safety PLC watchdog time.
E2	1 ... 4	255	30	2	38	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	42	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	43	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	52	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	2	54	Internal error	Contact ABB technical support. Replace safety PLC.
E2	1 ... 4	255	30	3	30	PROFIsafe configuration error	Check F-Parameter configuration of I/O module and reload boot project.
E2	9	1 ... 4	1	0	17	Access test failed	Check safety PLC switch address setting. Restart safety PLC. If this error persists, replace safety PLC.
E2	9	1 ... 4	1	0	43	Internal error	Check safety PLC switch address setting. Restart safety PLC. If this error persists, replace safety PLC
E2	9	1 ... 4	31	0	43	Internal error	Replace module
E3	1 ... 4	255	30	1	26	Error in configuration data, safety PLC cannot read configuration data	Create new configuration data
E3	1 ... 4	255	30	1	27	Error in configuration data, safety PLC cannot read configuration data	Create boot project

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E4	1 ... 4	255	30	1	0	Operation finished	Change safety PLC switch address setting or remove memory card from non-safety PLC. Restart safety PLC. If this error persists, replace safety PLC.
E4	1 ... 4	255	30	1	4	Boot project not loaded, maximum power dip reached	Restart safety PLC
E4	1 ... 4	255	30	1	8	Power dip data missed or corrupted. Default power dip data was flashed by safety PLC.	Warning
E4	1 ... 4	255	30	1	19	Checksum error has occured in safety PLC configuration.	Create new boot project and restart safety PLC
E4	1 ... 4	255	30	2	13	Flash write error (production data)	Warning
E4	1 ... 4	255	30	2	26	No or wrong configuration data from PM5x, run state not possible	Create correct boot project at PM5x
E4	1 ... 4	255	30	2	39	More than one instance of SF_WDOG_TIME_SET or SF_MAX_POWER_DIP_SET	Warning
E4	1 ... 4	255	30	4	13	Flash write error (boot project)	Warning
E4	1 ... 4	255	30	5	13	Flash write error (boot code)	Warning
E4	1 ... 4	255	30	6	13	Flash write error (firmware)	Warning
E4	1 ... 4	255	30	7	13	Flash write error (password)	Warning
E4	1 ... 4	255	30	8	13	Flash write error (user data)	Warning
E4	1 ... 4	255	30	9	13	Flash write error (user data)	Warning
E4	1 ... 4	255	30	10	13	Flash write error (internal)	Warning
E4	1 ... 4	255	30	11	13	Flash write error (internal)	Warning
E4	1 ... 4	255	30	12	13	Flash write error (internal)	Warning

Table 791: Specific error messages for SM560-S-FD-1 / SM560-S-FD-4 safety CPUs

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E2	1 ... 4	255	28	0 ... 31	43	Internal PROFIsafe F-Device error	Restart safety PLC. If this error persists, replace safety PLC. Contact ABB technical support.
E3	1 ... 4	255	28	0 ... 31	1	Safety destination address not valid (F_Dest_Add)	Check safety PLC configuration or switch address setting. Restart safety PLC. If this error persists, replace safety PLC.
E3	1 ... 4	255	28	0 ... 31	2	Safety source address not valid (F_Source_Add)	Check safety PLC configuration.
E3	1 ... 4	255	28	0 ... 31	10	Parameter "F_SIL" exceeds SIL from specific device application	Check safety PLC configuration.
E3	1 ... 4	255	28	0 ... 31	11	Safety watchdog time value is 0 ms (F_WD_Time)	Check safety PLC configuration.
E3	1 ... 4	255	28	0 ... 31	19	CRC1-Fault	Check safety PLC configuration. If this error persists, contact ABB technical support.
E3	1 ... 4	255	28	0 ... 31	28	Mismatch of safety destination address (F_Dest_Add)	Check safety PLC configuration or switch address setting. Restart safety PLC. If this error persists, replace safety PLC.
E3	1 ... 4	255	28	0 ... 31	42	Parameter "F_CRC_Length" does not match the generated values	Check safety PLC configuration.
E3	1 ... 4	255	28	0 ... 31	40	Version of F-Parameter set incorrect	Check safety PLC configuration.
E3	1 ... 4	255	30	1	17	Safety source addresses cannot be checked	Check PROFIsafe F-Host library version (2.0.0 or above). If this error persists, contact ABB technical support.
E3	1 ... 4	255	30	1	54	PROFIsafe F_Dest_Add rules are violated	Check safety PLC configuration or switch address setting against PROFIsafe F_Dest_Add configuration rules. Restart safety PLC. If this error persists, contact ABB technical support.

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E3	1…4	255	28	0…31	26	F_Block_ID not supported	Check safety PLC configuration
E3	1…4	255	28	0…31	20	Transmission error: data inconsistent (CRC2 error)	Check installation and wiring
E3	1…4	255	28	0…31	25	Transmission error: timeout (F_WD_Time or F_WD_Time_2 elapsed)	Check safety PLC configuration

Table 792: Error messages for safety I/O modules (channel or module reintegration is possible)

Error severity	Component or interface	Device	Module	Channel	Error	Error text	Remedy
E3	14	1..10	0	0..15	3	Discrepancy time expired	Check discrepancy time value, channel wiring and sensor.
E3	14	1..10	0	0..15	12	Test pulse error	Check wiring and sensor.
E3	14	1..10	0	0..15	13	Channel test pulse cross-talk error	Check wiring and sensor. If this error persists, replace I/O module. Contact ABB technical support.
E3	14	1..10	0	0..15	25	Channel stuck-at error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
E3	14	1..10	0	0..15	28	Channel cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
E3	14	1..10	1	0..3	4	Measurement overflow at the I/O module	Check channel wiring and sensor power supply.
E3	14	1..10	1	0..3	7	Measurement underflow at the I/O module	Check channel wiring and sensor power supply.
E3	14	1..10	1	0..3	55	Channel value difference too high	Adjust tolerance window for channels. Check channel wiring and sensor configuration.
E3	14	1..10	2	0..7	13	Channel read-back error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.
E3	14	1..10	2	0..7	18	Channel cross-talk error	Check I/O module wiring. Restart I/O module, if needed. If this error persists, replace I/O module.

Error severity	Compo-nent or interface	Device	Module	Channel	Error	Error text	Remedy
E3	14	1..10	31	31	10	Process voltage too high	Check process voltage
E3	14	1..10	31	31	11	Process voltage too low	Check process voltage
E3	14	1..10	31	31	20	PROFIsafe communication error	Restart I/O module. If this error persists, contact ABB technical support.
E3	14	1..10	31	31	25	PROFIsafe watchdog timed out	Restart I/O module. If this error persists, increase PROFIsafe watchdog time.
E3	14	1..10	31	31	43	Internal error in the device	Replace I/O module

Table 793: Error messages for safety I/O modules (channel or module reintegration is not possible)

Error severity	Compo-nent or interface	Device	Module	Channel	Error	Error text	Remedy
E3	14	1..10	31	31	18	Plausibility check failed (iParameter)	Check configuration
E3	14	1..10	31	31	19	Checksum error in the I/O module	Check safety configuration and CRCs for I- and F-Parameters.
E3	14	1..10	31	31	26	Parameter error	Check master or configuration
E3	14	1..10	31	31	28	F-Parameter configuration and address switch value do not match.	Check I/O module F-Param-eter configuration and module address switch value.

1.7.3.5 Error messages of the AC500 V2 function block libraries

1.7.3.5.1 0000hex...0FFFhex - telegram error

DEC	HEX	Error description
0	0000	No error
1	0001	COM_MOD_MAST: Error message from slave ILLEGAL FUNCTION ETH_MOD_MAST:Error message from slave ILLEGAL FUNCTION
2	0002	COM_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS ETH_MOD_MAST: Error message from slave ILLEGAL DATA ADDRESS
3	0003	COM_MOD_MAST: Error message from slave ILLEGAL DATA VALUE ETH_MOD_MAST: Error message from slave ILLEGAL DATA VALUE
4	0004	COM_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE ETH_MOD_MAST: Error message from slave SLAVE DEVICE FAILURE
5	0005	COM_MOD_MAST: Error message from slave ACKNOWLEDGE ETH_MOD_MAST: Error message from slave ACKNOWLEDGE

DEC	HEX	Error description
6	0006	COM_MOD_MAST: Error message from slave SLAVE DEVICE BUSY ETH_MOD_MAST: Error message from slave SLAVE DEVICE BUSY
8	0008	COM_MOD_MAST: Error message from slave MEMORY PARITY ERROR ETH_MOD_MAST: Error message from slave MEMORY PARITY ERROR
9	0009	COM_MOD_MAST: Error message from slave SEE SLAVE DESCRIPTION ETH_MOD_MAST: Error message from slave SEE SLAVE DESCRIPTION
10	000A	COM_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE ETH_MOD_MAST: Error message from slave GATEWAY PATH UNAVAILABLE
11	000B	COM_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND ETH_MOD_MAST: Error message from slave GATEWAY TARGET DEVICE FAILED TO RESPOND
4095	0FFF	FLASH_READ, FLASH_WRITE FLASH_DEL while ERR=FALSE: Block execution is in process

1.7.3.5.2 1000hex...1FFFhex - device error

DEC	HEX	Error description
4097	1001	Device does not exist
4098	1002	Command not supported by the device. The function is not supported by the device firmware/hardware. Block library newer than the device firmware. FC..: No high-speed counter available at the given module.
4100	1004	Error operating mode. FC..: Operating mode "0" -> No counter set in the PLC configuration.
4101	1005	Invalid status FLASH_READ: Block is not written yet FLASH_WRITE: Block was already written RETAIN..: No program loaded Library PROFINET IO: PNIO_WRITE, PNIO_READ: Internal error. Restart the PLC.
4117	1015	Format error SD..: File cannot be read because of an invalid format. Data could not be read or not be read completely.
4119	1017	Incorrect length PERSISTENT..: Data have an incorrect length RETAIN..: Data have an incorrect length
4120	1018	Checksum error
4122	101A	FC..: Internal error (e.g. no CS31 Adr parameter found, wrong size of CS31 Adr) HA..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in configuration found)

DEC	HEX	Error description
4123	101B	Device access error Flash..: Resources are not available HA..: Remote CPU failure: Other CPU is off or out of order PERSISTENT..: Data could not be copied, access error or no data do exist RETAIN..: Data could not be copied, access error or no data do exist SD..: Access to the memory card is not possible (e.g. memory exhausted, file already opened, etc.
4124	101C	Incorrect number PERSISTENT..: Because of the current CPU parameters, data are loaded only partly
4127	101F	Access protection SD..: Memory card is write protected
4128	1020	Error when opening SD..: Error when opening a file stored on the memory card
4129	1021	Not found FC..: CS31 Bus Module not found EtherCAT Modul not found HA_CS31_CONTROL: Own CI590-CS31-HA slave failure (missing module) HA_CS31_DIAG: One or more CI590-CS31-HA slave(s) is/are inactive HA_CS31_DIAG_VIA_CM574-RS: One or more CI590 are inactive SD..: The searched sector could not be found in the file TASK_INFO: Unknown task
4130	1022	End reached SD..: Section end or end of file reached PERSISTENT..: Because of the current CPU parameters, data are not loaded
4131	1023	Reading error FLASH..: Reading error in data segment: Incorrect checksum
4132	1024	Writing error FLASH..: Block cannot be programmed SD..: File could not be deleted or written
4137	1029	FC..: Wrong configuration (e.g. no CM574-RS configured on specified slot) HA..: Remote CI590-CS31-HA slave failure (missing module)
8191	1FFF	Not ready Flash..: The command is already executed by an other instance SD..: Command cannot be executed. Another instance is already active.

1.7.3.5.3 2000hex...2FFFhex - interface error

DEC	HEX	Error description
8193	2001	Invalid interface, Communication Module number or slot ID COM: Interface is not configured in the "free mode" HA: Wrong COM number at input COM ETH-MOD-MAST: Invalid interface or slot ID
8194	2002	Command not supported by the interface. The function is not supported by the device firmware. Block library newer than the device firmware.
8195	2003	Invalid interface or Communication Module type. Block is not suitable for this type.
8197	2005	HA..: CS31 Bus failure
8198	2006	Fault on both local and remote CPU.
8211	2013	Timeout COM_MOD_MAST: Slave did not respond within the specified time HA..: No Ethernet link, Error in DPRAM communication between CM574-RS and AC500 CPUs. ECAT_COE_READ, ECAT_COE_Write: Response timeout occurs
8212	2014	Framing error (incorrect transmission rate, number of stop bits and/or bits per character)
8213	2015	Parity error HA: Remote CS31 slave sync error. Difference in digital/ analog output buffer of PLC A and B
8214	2016	Idle error COM..: Character timeout occurred
8215	2017	Invalid length COM_MOD_MAST: Invalid data length received COM_REC: Received more data than expected
8216	2018	Checksum error
8217	2019	Handshake error
8218	201A	Service failed COM_REC: Unknown error message of the interface COM_SET_PROT: Interface hardware not accessible. Initialization already failed during system start-up. COM_MOD_SLV_SET_ADDR..: Internal error (e.g. wrong return value of internal function, null pointer received) CS31..: Internal error (e.g. CS31 communication failed) FC..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in config found) HA..: Internal error (e.g. null pointer received, wrong return value of internal function, no entry in configuration found)
8219	201B	Access error HA..: Remote CS31 Bus failure: Other CPUs' CS31 Bus is out of order IO..: Module number does not exist

DEC	HEX	Error description
8220	201C	Incorrect number (quantity) COM_SET_PROT: Invalid protocol index. Index is not supported by the device firmware. COM_MOD_SLV_SET_ADDR..: Invalid protocol index. HA..: Numbers of CI590-CS31-HA devices configured in line A and line B are different IO..: Invalid module number
8223	201F	Access denied COM..: Access to the interface is not possible at present. Automation Builder, OPC or another program is logged in via the interface.
8226	2022	COM_MOD_SLV_SET_ADDR..: Wrong configuration (No protocols configured). FC..: Wrong configuration (e.g. no Automation Builder configuration, no CS31 configured, no CS31 modules found). HA..: Overflow of HA_DATA reference table
8233	2029	● HA_CS31_CONTROL: – CI590-CS31-HA slave configuration not complete – CI590-CS31-HA slaves in Bus1 and Bus2 are mix-wired – Remote CI590-CS31-HA Failure – CS31 Master Cross Wired / No CS31 configuration / No submodules on CS31 bus ● HA_CS31_DIAG/ HA_CS31_DIAG_VIA_CM574-RS – CI590-CS31-HA slaves in line A and line B are mix-wired.
8234	202A	HA: configuration error: ● Wrong CM574-RS communication ● CM574-RS COM interface is not configured for shared communication ● CM574-RS not added to Communication Module slot ● Wrong configuration of DIAG blocks

1.7.3.5.4 3000hex...3FFFhex - protocol error

DEC	HEX	Error description
12289	3001	Unknown protocol or protocol not configured. MqttClient: The Network Connection has been made but the MQTT service is unavailable on the specified port.
12290	3002	Command not supported by the protocol. The function is not supported by the device firmware. Block library newer than the device firmware.
12291	3003	Another protocol is configured. FC_DC551: The selected interface (COMx) is not set to the CS31 protocol. COM_MOD_SLV_SET_ADDR..: The selected interface (COMx) is not set to the Multi protocol. COM_MOD_SLV_SET_ADDR..: Wrong protocol index (No Modbus protocol on specified index). HA: No CS31 protocol at COM. BACnet B-ASC: Multiple function block instances, one instance allowed only.

DEC	HEX	Error description
12292	3004	Operating mode error COM_MOD_MAST: Invalid operating mode (master/slave).
12293	3005	Protocol status error Fieldbus Communication Module .._SYS_DIAG: Master is not in the OPERATE state. ETH_SMTP_EMAIL_SEND: Internal error (ulHandle wrong). BACnet B-ASC: No device object instantiated.
12307	3013	IEC..: ACTCON timeout ETH_SMTP_EMAIL_SEND: Server timeout MqttClient: The timeout value for the communication has been exceeded.
12308	3014	IEC..: NACK received
12309	3015	ETH_SMTP_EMAIL_SEND: Syntax error in mail address
12310	3016	IEC..: Timeout
12311	3017	Incorrect length ARC..: Buffer is full CAN2..: Total length of all messages too high IEC..: Queue overrun ETH_UDP..: Buffer is full. MqttClient: Received topic or payload is too long.
12313	3019	IEC..: ACTERM timeout Wait answer ETH_SMTP_EMAIL_SEND: Could not connect to server. Not reachable or does not answer.
12314	301A	Execution failed COM_SET_PROT: Initialization of the protocol failed IEC..: Send failed due to queue deleted COM_MOD_SLV_SET_ADDR..: Internal error (e.g. null pointer received, wrong return value of internal function) ETH_SMTP_EMAIL_SEND: Internal error. Mail not sent. (e.g. out of resources) ETH_ICMP_PING: Target Host did not answer the ping echo request before the specified timeout MqttClient: MQTT broker did not answer the ping. MQTT client has passed the KeepAlive or MQTT broker is unreachable.
12315	301B	Access error ARC..: Buffer does not exist / is not specified CAN2..: Buffer does not exist / is not specified ETH_UDP..: Buffer does not exist / is not specified IEC..: Protocol access error IO..: There is no module in the selected slot
12316	301C	Wrong number IO..: Invalid module number > max.

DEC	HEX	Error description
12319	301F	Access denied COM..: Access to the interface is not possible at present. Automation Builder, OPC or another program is logged in via the interface. MqttClient: The Client identifier is correct UTF-8 but not allowed by the Server.
12320	3020	Error when opening ARC..: Error during protocol initialization. Protocol not yet ready. CAN2..: Error during protocol initialization. Protocol not yet ready. IEC..: Data type mismatch for this PIN when data arrived. ETH_UDP..: Error during protocol initialization. Protocol not yet ready. ETH_SMTP_EMAIL_SEND: ● File for attachment cannot be opened ● Server not ready (wrong port configured?) MqttClient: The Server does not support the level of the MQTT protocol requested by the Client.
12321	3021	ETH_SMTP_EMAIL_SEND: File for attachment not found CPU_PROD_ENTRY_READ: Entry not found
12325	3025	Address error COM_MOD_MAST: Receive telegram does not contain the expected register address. ETH_MOD_MAST: Cannot connect to server (IP address) OR response telegram contains wrong UNIT_ID (must be the same as in request). ETH_SMTP_EMAIL_SEND: Internal error (Could not bind sockets). ETH_UDP_REC: Wrong INDEX for connection. ETH_UDP_SEND: Wrong INDEX for connection or IP Destination does not contain a valid IP address. MqttClient: Connection refused, maybe the IP address is malformed.
12326	3026	Function error COM_MOD_MAST: The received FCT does not correspond to the sent FCT.
12327	3027	Invalid value COM_MOD_MAST: Receive telegram contains an unexpected value. MqttClient: Network error on MqttSubscribe/MqttUnsubscribe. Maybe the topic is not valid.
12328	3028	ETH_SMTP_EMAIL_SEND: Out of sockets. Mail not sent (resource starvation). ETH_ICMP_PING: Could not create internal Task (not enough resources). BACnet B-ASC: Protocol task start failed.
12329	3029	HA..: CI590-CS31-HA slave configuration is not complete.
12331	302B	Unspecified error MqttClient: Internal library returned an unspecified error.
12333	302D	No connection, sending not possible due to no connection, either closed or not established yet MqttClient: No connection to an MQTT Broker.
12540	30FC	ETH_SMTP_EMAIL_SEND: Mailbox unavailable or not found on target server. SMTP error by server.

DEC	HEX	Error description
12788	31F4	ETH_SMTP_EMAIL_SEND: Syntax Error. SMTP error by server.
12789	31F5	Syntax error in parameters or arguments.
12790	31F6	Command not implemented. SMTP error by server.
12791	31F7	The SMTP server has encountered a bad sequence of commands, or it requires an authentication.
12823	3217	ETH_SMTP_EMAIL_SEND: Wrong user login and/or password. Check configuration. SMTP Error by server. MqttClient: Authentication failed: Bad username, password OR client id.
12838	3226	ETH_SMTP_EMAIL_SEND
12839	3227	ETH_SMTP_EMAIL_SEND: Mailbox unavailable or not found on target server. SMTP Error by server.
12840	3228	ETH_SMTP_EMAIL_SEND: Exceeded mailbox storage on target server. SMTP Error by server.
12841	3229	ETH_SMTP_EMAIL_SEND: Mailbox name not allowed. SMTP Error by server.
12848	3230	MqttClient: Error on TLS handshake.
12849	3231	MqttClient: Server certificate not valid. Check if PLC date has been set correctly.
12850	3232	MqttClient: Server certificate format is not formatted as PEM.
12851	3233	MqttClient: Server certificate has expired.
12852	3234	MqttClient: Client certificate not valid. Check if PLC date has been set correctly.
12853	3235	MqttClient: Client certificate or client key format is not formatted as PEM.
12854	3236	MqttClient: Client certificate has expired.
16383	3FFF	Not ready. Resources currently not available. COM_MOD_MAST: Transmission is not possible at the moment. Another instance of the function block is already transmitting. COM_SEND: Transmission is not possible at the moment. Another instance of the function block is already transmitting. BACnet B-ASC: No memory for BACnet objects.

1.7.3.5.5 4000hex...4FFFhex - block input error

The error 4xxxhex is used in case of detected function block input parameter errors. The error is structured as follows:

4 X1 X2 X3 hex

X...	Value	Error Description
X1 + X2	1....FF	Number of the input
X3	0	Invalid value
	1	Value too low
	2	Value too high
	3	Wrong combination of the parameters

1.7.3.5.6 5000hex...5FFFhex - request error

DEC	HEX	Error description
20482	5002	The request is not supported (e.g. in the simulation mode of the Automation Builder).
20485	5005	Invalid internal state -> function block probably called in invalid manner/ sequence
20499	5013	Timeout error ECAT_COE_READ, ECAT_COE_Write: Request timeout occurs
20503	5017	Incorrect length ARC..: Invalid data length CAN2..: Invalid DLC in message ETH_UDP..: Invalid data length ECAT_COE_READ, ECAT_COE_Write: Request/ Response length invalid
20504	5018	Traceability data is corrupted.
20507	501B	Access error COM_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges. ETH_MOD_MAST: Invalid memory address DATA or DATA + NB. At least one datum is outside of the access range of the user program. The range includes different flag ranges.
20508	501C	Invalid number (quantity) CAN2A_SEND: Invalid number of messages at input NUM COM_MOD_MAST: Invalid number of data at NB (0 or more than permitted). ETH_MOD_MAST: Invalid number of data at NB (0 or more than permitted).20515
20513	5021	IO_PROD_ENTRY_READ: I/O Module not found: input MODULE invalid
20515	5023	Failed to read traceability data from I/O Module.
20517	5025	Address error ARC..: Invalid IP address CAN2..: Invalid identifier in message COM_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_MOD_MAST: Invalid slave address. Broadcast not permitted in connection with the selected function code. ETH_UDP..: Invalid IP address.
20518	5026	Function error COM_MOD_MAST: Invalid function code FCT. ETH_MOD_MAST: Invalid function code FCT.
20735	50FF	DIAG..: Simulation mode
24575	5FFF	Operation pending / busy -> wait and try again later

1.7.3.5.7 6000hex...6FFFhex - communication module errors

DEC	HEX	Error description
24577	6001	EtherCAT: Invalid command received PNIO: Creating a TLR-timer-packet in RPC task failed due to insufficient memory
24578	6002	EtherCAT: No link exists OR the watchdog expired CAN..: Service was rejected by the node with SDO abortion. Index/subindex not valid or no access to the specified node. DNM..: Resource not available or invalid class ID DPM../DPV1..: Resource not available. Free buffer memory in slave is not sufficient for the requested service. PNIO: Generic RPC-error code or not enough memory
24579	6003	EtherCAT: Error during reading the bus configuration OR the requested watchdog time is too small. DPM../DPV1..: Requested service (e.g. DPV1) is not active in the slave.
24580	6004	Communication Module does not support DPRAM message communication function blocks OR There is no receive function block active in IEC project of Communication Module. EtherCAT: Error during processing the bus configuration OR The requested watchdog time is too large.
24581	6005	EtherCAT: Existing bus does not match configured bus OR Error during reset (resetting watchdog).
24582	6006	EtherCAT: Not all slaves are available OR Error during reset (cleanup dynamic resources) DPM../DPV1..: Slave address not configured
24583	6007	EtherCAT: Error during reset (stopping the master) OR master is in critical error state, reset required.
24584	6008	EtherCAT: Error during reset (deinitializing the master) OR error activating the watchdog. DNM..: Service not available in module. Read/write function not supported by the selected class.
24585	6009	EtherCAT: Error during reset (cleanup the dynamic resources) OR size of configured input data is larger than cyclic DPM input data size. DNM..: Attribute invalid or not supported DPM../DPV1..: No data received from slave
24586	600A	EtherCAT: Master is in critical error state. Reset required OR size of configured output data is larger than DPM data output size.
24587	600B	EtherCAT: The requested bus cycle time is invalid DNM..: Request is already in progress
24588	600C	EtherCAT: Invalid parameter for broken slave behavior. DNM..: Conflict of the object status
24589	600D	EtherCAT: Master is in wrong internal state
24590	600E	EtherCAT: The watchdog expired DNM..: Attribute cannot be set or writing is not permitted
24591	600F	EtherCAT: Invalid SlaveID was used for CoE DNM..: Permission check faulty or access denied

DEC	HEX	Error description
24592	6010	EtherCAT: No available resources for CoE Transfer DNM..: Status conflict. Device prohibits execution
24593	6011	EtherCAT: Internal error during CoE usage CAN..: No response from the selected node DNM..: No response from the selected device DPM../DPV1..: No response received from slave.
24594	6012	EtherCAT: Invalid index on slave requested DPM../DPV1..: Master not in logical "token ring". PNIO: Internal error inside Communication Module's firmware
24595	6013	EtherCAT: Invalid bus communication state for CoE-usage CAN..: Selected node is not ready for operation DNM..: Not enough receive data
24596	6014	EtherCAT: Frame with CoE data is lost CAN..: Local resources are not available. Requested bus parameters are not available. Communication Module is not configured. DNM..: Local resources are not available. Requested bus parameters are not available. Communication Module is not configured. PNIO: Internal error inside Communication Module's firmware
24597	6015	EtherCAT: Timeout during CoE service CAN..: Parameter error DNM..: Parameter error
24598	6016	EtherCAT: Slave is not addressable (not on bus or power down) DNM..: Object does not exist
24599	6017	EtherCAT: Invalid list type requested Other Communication Modules: Received data length too big. Internal buffer too small.
24600	6018	EtherCAT: Data in slave response is to large for confirmation packet. PNIO: Internal error inside Communication Module's firmware
24601	6019	EtherCAT: Invalid access mask selected (during GetEntryDesc) DPM../DPV1..: Unexpected reaction of slave or reaction not in accordance with standard.
24602	601A	EtherCAT: Slave Working Counter error during CoE service PNIO: Another request is already running ETH2 not supported on AC500 CPU with Ethernet
24603	601B	EtherCAT: The service is already in use
24604	601C	EtherCAT: Command is not usable in this communication state
24605	601D	EtherCAT: Distributed Clocks must be activated for this command
24606	601E	EtherCAT: The scan is already running. It cannot be started twice at the same time
24607	601F	EtherCAT: Timeout during bus scan, but at least one link is established
24608	6020	EtherCAT: The bus scan was not started before or it is not finished yet
24609	6021	EtherCAT: The requested slave is invalid

DEC	HEX	Error description
24610	6022	EtherCAT: Internal error during CoE usage
24612	6024	PNIO: Internal error inside Communication Module's firmware
24615	6027	EtherCAT: Internal error of SDO protocol
24616	6028	EtherCAT: Internal error of SDO protocol
24617	6029	EtherCAT: Internal error of SDO protocol
24618	602A	EtherCAT: Internal error of SDO protocol
24619	602B	EtherCAT: Internal error of SDO protocol
24620	602C	EtherCAT: Internal error of SDO protocol
24621	602D	EtherCAT: Not enough memory
24622	602E	EtherCAT: Selected object could not be accessed
24623	602F	EtherCAT: Selected object is write only
24624	6030	EtherCAT: Selected object is read only CAN..: Function timeout DNM..: Device not configured PNIO: Internal error inside Communication Module's firmware
24625	6031	EtherCAT: Selected object does not exist PNIO: Internal error inside Communication Module's firmware
24626	6032	EtherCAT: PDO mapping failed DNM..: Format error in the received data ETH_UDP..: TCP/UDP task not available or IP task not ready. PNIO: Internal error inside Communication Module's firmware
24627	6033	EtherCAT: Selceted object is to large to be mapped to PDO CAN..: Maximum buffer size of the receive data exceeded ETH_UDP..: Internal task with configuration data not available PNIO: The ALPMR protocol-machine corresponding to the index in request packet is invalid
24628	6034	EtherCAT: General parameter error occured CAN..: Function not available. Code unknown. DNM..: Code unknown ETH_MOD..: Invalid parameter for "ServerConnection" ETH_UDP..: No MAC address available PNIO: The ALPMR protocol-machine is invalid for the current request
24629	6035	EtherCAT: Internal device error occured CAN..: Unknown area. Buffer exceeded. DNM..: Overflow of buffer length ETH_MOD..: Invalid parameter for "Task Timeout" ETH_UDP..: Waiting for warm start performed by the application

DEC	HEX	Error description
24630	6036	EtherCAT: Hardware error occured CAN..: Unknown function in HOST message or function still active DNM..: Other service still active ETH_MOD..: Invalid parameter for "OBM Timeout" ETH_UDP..: Unknown flag in start parameters DPM../DPV1..: Slave denied access to the requested data
24631	6037	EtherCAT: Invalid data type CAN..: Parameter error DNM..: Parameter error or MAC ID beyond the valid range ETH_MOD..: Invalid parameter for "Mode" ETH_UDP..: Invalid IP address in start parameters PNIO: The index of ALPMR protocol-machine is invalid
24632	6038	EtherCAT: Invalid data type ETH_MOD..: Invalid parameter for "Send Timeout" ETH_UDP..: Invalid subnet mask in start parameters
24633	6039	EtherCAT: Invalid data type CAN..: Sequence error DNM..: Sequence error or one MAC ID was multiple used in one network ETH_MOD..: Invalid parameter for "Connect Timeout" ETH_UDP..: Invalid gateway IP in start parameters
24634	603A	EtherCAT: Invalid sub-index ETH_MOD..: Invalid parameter for "Close Timeout".
24635	603B	EtherCAT: Invalid parameter value CAN..: Data error DNM..: Data error ETH_MOD..: Invalid parameter for "Swab" ETH_UDP..: Unknown device type
24636	603C	EtherCAT: Invalid parameter value CAN..: Node address configured twice DNM..: Display of total number of data sets incorrect ETH_MOD..: TCP task not ready ETH_UDP..: Access to IP address in the specified source failed
24637	603D	EtherCAT: Invalid parameter value CAN..: ADD table incorrect DNM..: ADD table incorrect ETH_MOD..: PLC task not ready ETH_UDP..: Initialization of the driver layer failed

DEC	HEX	Error description
24638	603E	EtherCAT: Invalid parameter value CAN..: Total length of the node parameters incorrect DNM..: Size of the I/O configuration table incorrect ETH_MOD..: Error during initialization ETH_UDP..: No source specified for IP address (BOOTP, DHCP, IP address parameter)
24639	603F	EtherCAT: Internal error of SDO protocol CAN..: Transmission type unknown DNM..: I/O configuration does not match with the ADD table
24640	6040	EtherCAT: Internal device error occured CAN..: Length of the PDO-cfg file too big DNM..: Parameter size incorrct or channel/handler already in use PNIO: Internal error inside Communication Module's firmware
24641	6041	EtherCAT: Internal device error occured CAN..: Unknown transmission rate DNM..: Number of defined inputs in the ADD table does not match with the I/O configuration PNIO: Internal error inside Communication Module's firmware
24642	6042	EtherCAT: Internal device error occured CAN..: COB-ID SYNC beyond the valid range DNM..: Number of defined outputs in the ADD table does not match with the I/O
24643	6043	EtherCAT: Internal device error occured CAN..: Value of the synchronization timer invalid DNM..: Unknown data type in the I/O configuration
24644	6044	EtherCAT: Unknown SDO protocol error CAN..: Input offset of the PDOs too big DNM..: Defined data type of an I/O module does not match with the defined data size
24645	6045	CAN..: Output offset of the PDOs too big DNM..: The configured output address of an I/O module is not within the permitted address range of 3584 bytes
24646	6046	CAN..: Inconsistency between the PDO and the ADD table DNM..: The configured input address of an I/O module is not within the permitted address range of 3584 bytes
24647	6047	CAN..: Length of the ADD table inconsistent DNM..: Unknown connection type
24648	6048	CAN..: Total data length inconsistent DNM..: Several identical connections defined
24649	6049	CAN..: COB-ID Emergency beyond the permitted range DNM..: The configured value of the "Exp_Packet_Rate" of a connection is smaller than the value of the "Prod_Inhibit_Time"

DEC	HEX	Error description
24650	604A	CAN..: COM-ID Node Guard beyond the permitted range DNM..: Inconsistent parameter field "DNM_PRED_MSTSL_CFG_DATA"
24651	604B	CAN..: Configured PDO length greater than 8 DNM..: Device could not perform "Duplicate_MAC-ID check". Incorrect transmission rate or no connection to the device possible.
24652	604C	CAN..: Number of defined objects in SDO data too big DNM..: Value of "usRecFragTimer" beyond the permitted range
24657	6051	PNIO: The current bus state is OFF and no frames can be sent
24659	6053	PNIO: The state of APMS protocol-machine is invalid for the current request
24660	6054	PNIO: APMS was not able to get an Edd_Frame buffer for sending a packet
24661	6055	PNIO: An error occurred while APMS was trying to send an Edd_Frame
24662	6056	PNIO: Device not reachable (DEV_NAME is not projected)
24663	6057	PNIO: Insufficient memory for APMS_send_req_Date() to allocate a timer-indication packet
24672	6060	PNIO: The acyclic service failed. The I/O module answered with an error code. See output STATUS (EtherCAT status) for details.
24679	6067	PNIO: The maximum amount of data supported by this service is exceeded.
24686	606E	ETH_UDP..: Timeout has occurred
24687	606F	ETH_MOD..: Unknown send or receive telegram ETH_UDP..: Invalid timeout parameter
24688	6070	ETH_MOD..: TCP responds with an error ETH_UDP..: Invalid socket
24689	6071	ETH_MOD..: No corresponding socket found ETH_UDP..: Command cannot be executed in the current socket state
24690	6072	ETH_MOD..: Command with invalid value
24691	6073	ETH_MOD..: TCP task status error ETH_UDP..: No access to target IP address
24692	6074	ETH_UDP..: Invalid option parameter
24693	6075	ETH_MOD..: No free socket found ETH_UDP..: Invalid command parameter
24694	6076	ETH_MOD..: TCP command is directed to an unknown socket ETH_UDP..: Invalid IP address or no access to address HA..: Wrong IP address configured
24695	6077	ETH_MOD..: Time for a client job is over ETH_UDP..: Invalid port number or port not available
24696	6078	ETH_MOD..: Socket has been closed unexpectedly ETH_UDP..: Connection closed
24697	6079	ETH_MOD..: Not-Ready flag has been set by the user ETH_UDP..: Connection reset
24698	607A	ETH_MOD..: OMB task cannot open socket ETH_UDP..: Invalid protocol

DEC	HEX	Error description
24699	607B	ETH_MOD..: Watchdog event in PLC task, only in I/O mode ETH_UDP..: No socket available
24700	607C	ETH_MOD..: TCP task in configuration state ETH_UDP..: Invalid MAC address
24701	607D	ETH_MOD..: PLC task not initialized
24702	607E	ETH_MOD..: Server socket was closed without response from the device
24703	607F	ETH_MOD_MAST.
24705	6081	DPM../DPV1..: DPV1 not in "OPEN" state
24706	6082	ETH_UDP..: Invalid mode parameter DPM../DPV1..: Invalid parameters received from slave. Communication stopped.
24707	6083	ETH_UDP..: Maximum data length exceeded or ARP cache full DPM../DPV1..: Service still active. Parallel operation is not possible
24708	6084	ETH_UDP..: Maximum number of messages exceeded DPM../DPV1..: Data length too high for the reserved buffer
24709	6085	ETH_UDP..: Maximum number of IP multicast groups exceeded DPM../DPV1..: Wrong parameter
24710	6086	ETH_UDP..: ARP input not found in ARP cache
24725	6095	ETH_UDP..: Invalid message response received
24727	6097	ETH_MOD..: Invalid message length ETH_UDP..: Invalid message length
24728	6098	CAN..: Unknown message command DNM..: Unknown message command ETH_MOD..: Unknown message command ETH_UDP..: Unknown message command
24730	609A	DPM../DPV1..: Invalid message command
24732	609C	ETH_UDP..: Sequence error during transmission in Sequence Message Mode
24734	609E	ETH_UDP..: Command cannot be executed or command is currently executed
24736	60A0	ETH_MOD..: Error in telegram header
24737	60A1	CAN..: Node address beyond the permitted range DNM..: Device address beyond the permitted range ETH_MOD..: Invalid address detected in the telegram DPM../DPV1..: Invalid slave address
24738	60A2	CAN..: Invalid address range DNM..: Invalid address range
24739	60A3	CAN..: Data buffer overflow DNM..: Data buffer overflow ETH_MOD..: Invalid data address

DEC	HEX	Error description
24741	60A5	CAN..: Incorrect data counter DNM..: Incorrect data counter ETH_MOD..: Invalid data counter
24742	60A6	CAN..: Unknown data type DNM..: Unknown data type
24743	60A7	CAN..: Unknown function DNM..: Unknown function ETH_MOD..: OBM task received an error in the response of the TCP task
24776	60C8	CAN..: Communication Module is not configured DNM..: Communication Module is not configured ETH_UDP..: Task not initialized
24778	60CA	ETH_MOD..: OBM task does not have a segment from RCS
24779	60CB	ETH_MOD..: Unknown or invalid sender specified with the command
24786	60D2	ETH_UDP..: No configuration data available
24788	60D4	ETH_UDP..: Error while reading the configuration data
24789	60D5	ETH_UDP..: Error while creating the diagnosis structure
24794	60DA	ETH_UDP..: Not enough memory available
24832	6100	PNIO: Generic RPC-error code. See output STATUS (PROFINET-status) for details.
24847	610F	ETH_MOD_MAST: Wrong MBAP header received
24848	6110	ETH_MOD..: Invalid Unit Identifier received
24850	6112	ETH_MOD_MAST: Invalid MBAP header Length value
25088	6200	PNIO: Internal error inside Communication Module's firmware
25089	6201	PNIO: Internal error inside Communication Module's firmware
26117	6605	PNIO: Internal error inside Communication Module's firmware
26118	6606	PNIO: Internal error inside Communication Module's firmware
26119	6607	PNIO: Internal error inside Communication Module's firmware

Abbreviations	
RPC	Remote Procedure Call
CMCTL	Controller Context Management
APMS	Acyclic Protocol-Machine sender
APMR	Acyclic Protocol-Machine receiver

1.7.3.5.8 7000hex...7FFFhex - product libraries

Table 794: Drives library

Dec	Hex	Error description
28672	7000	Any activity was NOT completed within an appropiate TIME
28673	7001	Read or write parameter could not be completed

Dec	Hex	Error description
28674	7002	A parameter at the function block is out of the possible range. This does not refer to the parameter range which is allowed for the drive but just to the 32-bit integer which is used for internal calculation
28675	7003	The field bus connection is faulty
28677	7005	Wrong PPO type
28678	7006	Wrong or no adapter type could be detected
28679	7007	Drive type does not match to function block
28680	7008	Function aborted
28681	7009	Error while reading scaling parameter for REF1
28682	700A	Wrong parameter number at read/write parameter
28683	700B	COM interface differs from others on same LINE_TOKEN variable
28684	700C	Profile type error
28685	700D	Function block Read_Parameters has been executed with error
28686	700E	Function block Write_Parameters has been executed with error
28687	700F	No connection to communication block, or error in communication block
28688	7010	Error at reading the drives communication profile value
28689	7011	Drive communication profile can not be used with this function block
28690	7012	PROFINET or PROFIBUS write packet size exceed 240 byte data limit

Table 795: Solar library

Dec	Hex	Error description
28928	7100	ALARM: Limit exceeded. System has moved more than POS_DEG_LIMIT distance without any moving command.
28929	7101	ALARM: Timeout. The system has not reached POS_DEG_LIMIT within the tolerance time (3 * t POS_TIME_LIMIT [ms]).
28930	7102	ALARM: Limit at wrong side reached. System has reached POS_DEG_LIMIT distance in the opposite DIR to actual movement order.
28931	7103	ALARM: Low Limit exceeded. Tracker position is less than VIRTUAL_LIMIT_MIN.
28932	7104	High Limit exceeded. Tracker position is more than VIRTUAL_LIMIT_MAX.
28933	7105	Warning: Interlocking. Try to move BACKWARD while STOP_BWD input is set.
28934	7106	Warning: Interlocking. Try to move FORWARD while STOP_FWD input is set.
28935	7107	Warning: Interlocking. Both STOP_BWD and STOP_FWD input are set.

Table 796: Data logger library

Dec	Hex	Error description
29184	7200	zLOG_ERROR_NO_TYPE_SPECIFIED
29185	7201	zLOG_ERROR_LENGTH_BOOL_EXCEEDED
29186	7202	zLOG_ERROR_LENGTH_BYTE_EXCEEDED
29187	7203	zLOG_ERROR_LENGTH_INT_EXCEEDED
29188	7204	zLOG_ERROR_LENGTH_WORD_EXCEEDED
29189	7205	zLOG_ERROR_LENGTH_DINT_EXCEEDED
29190	7206	zLOG_ERROR_LENGTH_DWORD_EXCEEDED

Dec	Hex	Error description
29191	7207	zLOG_ERROR_LENGTH_REAL_EXCEEDED
29195	720B	zLOG_ERROR_OPEN_ERR
29196	720C	zLOG_ERROR_GETPOS_ERR
29197	720D	zLOG_ERROR_SETPOS_ERR
29198	720E	zLOG_ERROR_WRITE_ERR
29199	720F	zLOG_ERROR_READ_ERR
29200	7210	zLOG_ERROR_CLOSE_ERR
29201	7211	zLOG_ERROR_GETSIZE_ERR
29202	7212	zLOG_ERROR_FLUSH_ERR
29205	7215	zLOG_ERROR_MAX_NUMBER_OF_FILES_EXCEEDED
29206	7216	zLOG_ERROR_MAX_NUMBER_OF_DATASETS_PRO_FILE
29207	7217	EXCEEDED zLOG_ERROR_FILE_WRITE_FAILED
29208	7218	zLOG_ERROR_FILE_READ_FAILED
29210	721A	zLOG_ERROR_CREATE_DIRECTORY
29211	721B	zLOG_ERROR_FILE_MOVE
29212	721C	zLOG_ERROR_NO_CSV_FOUND_IN_DATASET
29213	721D	zLOG_ERROR_DELETE_ACTUAL_FILE
29214	721E	zLOG_ERROR_FORMAT_DISK1
29215	721F	zLOG_ERROR_FORMAT_DISK2
29216	7220	zLOG_ERROR_DELETE_ACTUAL_FILES_NO_FILE_FOUND
29217	7221	zLOG_ERROR_DELETE_SAVE_FILES_NO_FILE_FOUND
29218	7222	zLOG_ERROR_ILLEGAL_MODE
29219	7223	zLOG_ERROR_ILLEGAL_DESIGNATION_ON_DISK1
29220	7224	zLOG_ERROR_ILLEGAL_DESIGNATION_ON_DISK2
29221	7225	zLOG_ERROR_FORMAT_NOT_SUPPORTED
29222	7226	zLOG_ERROR_MODE3_AND_DISK2_EXTENTION_TRUE_NOT_ALLOWED
29230	722E	zLOG_ERROR_29230_TIMEOUT_DIRECTORY_CREATE_FILE_MOVE
29231	722F	zLOG_ERROR_RESERVE
29232	7230	zLOG_ERROR_TIMEOUT_DISK_FORMAT
29233	7231	zLOG_ERROR_TIMEOUT_FILE_DELETE
29234	7232	zLOG_ERROR_TIMEOUT_WRITE_DATASET_IN_FILE
29235	7233	zLOG_ERROR_TIMEOUT_READ_DATASET_IN_CASE_OF_CONNECT
29236	7234	zLOG_ERROR_TIMEOUT_WRITE_DATASET_IN_CASE_OF_CONNECT
29237	7235	zLOG_ERROR_TIMEOUT_READ_DATASET_IN_CASE_OF_NOT_CONNECT
29238	7236	zLOG_ERROR_TIMEOUT_IEC60870_COMMUNICATION

Table 797: Temperature control library

Dec	Hex	Error description
29312	7280	Fault (TuneFault): "Tuning failed": outputs disabled, see Tune_Status Remedy: Check for tune set point and currrent temperatur
29313	7281	Fault (TC_Fault_1): "Bad Thermocouple reading": outputs disabled. Remedy: Bad Thermocouple reading: outputs disabled
29314	7282	Fault (TC_Fault_2): "Plausibility check not passed": outputs disabled. Remedy: Check for inverted connection of Thermocouple.
29315	7283	Fault (HighHighTempFalt): "HighHigh temperature alarm": outputs disabled. Remedy ● Check for defective cooling device. ● Check for correct setting of HIGHHIGH_TEMP and SET_TEMP.
29316	7284	Fault (LowLowTempFault): "LowLow temperature alarm": outputs disabled. Remedy ● Check for defective heating device. ● Check for correct setting of LOWLOW_TEMP and SET_TEMP.
29319	7287	High temperature alarm
29320	7288	Low temperature alarm
29321	7289	High deviation alarm
29330	7292	Zone_size and zone_index are not assigned
29335	7297	No group function block call possible: internal Group function call for TECT_TEMP_CONTROL or TECT_PWM8 failed.
29336	7298	TimeOut of one operation state.
29337	7299	Read data failed: no correct format of data items.
29322	728A	Low deviation alarm
29323	728B	Fault: (NoHighHighLowLow): "No plausible HighHigh or LowLow limit is defined" Remedy: Check the HighHighTemp and low low temp.
29324	728C	Fault (WrongLimits): "No plausible wrong limits defined" Remedy: Check the defined limits.
29325	728D	Heat and Cool are not enabled: Monitoring only.
29326	728E	Faut (NoAutoTune): "AutoTune cannot be started" Remedy: Process in Manual mode or Heat is not enabled
29327	728F	Fault (NoPIDProcess): "PID process cannot be started" Remedy: Process in Manual mode or no KS, TU, TG parameters are accepted.
29328	7290	Fault (Improper setting of HigHigh and High): HighHigh and High limits are not set correctly. Values are not taken by process/ADR_ZONEDATA. Remedy: Check parameter settings.
29329	7291	Fault (Improper setting of LowLow and Low): LowLow and Low limits are not set correct. Values are not taken by process/ADR_ZONEDATA. Remedy: Check parameter settings.
29330	7292	Fault (No assignment for zone index and size): Zone_size and zone_index are not assigned. Remedy: Check parameter settings.

Dec	Hex	Error description
29335	7297	Fault (No Group function possible):no Group function block call possible: internal Group function call for TECT_TEMP_CONTROL or TECT_PWM8 failed.
29336	7298	TimeOut of one operation state.
29337	7299	Fault (Reading failed):Read data failed: no correct format of data items. Remedy: Check parameter settings.
29338	729A	Fault (Incomplete log saving): Restart of the EN is required since last log saving is not complete
29339	729B	Fault (Less zone data available in RECIPE file than requested):Mismatch in number of zones declared and zone data in recipe. Remedy: Check the zone data in CSV file and Num_of_zones declared in RECIPE.

Table 798: CMS/FM502 library

Dec	Hex	Error description
29440	7300	Invalid file format.
29445	7305	Number of channels > 1, not supported by this version of library.
29440	7300	Invalid RIFF Chunk ID
29441	7301	Invalid WAV Chunk ID
29442	7302	Invalid FMT Chunk ID
29443	7303	Invalid DATA Chunk ID
29444	7304	Invalid LABL Chunk ID
29445	7305	Number of channels >1 , not supported by this version of library.
29504	7340	No Channel activated
29505	7341	Internal Error (Start)
29506	7342	Internal Error (Poll)
29507	7343	Internal Error (Finish)
29508	7344	Not possible because of measurement running
29519	734F	No Measurement possible. Device in Failure Loop.
29520	7350	Comunication Timeout to FM502
29521	7351	Internal File Error
29522	7352	Plausibility Check of Configuration wrong
29569	7381	Error in memory allocation to the input signal, due to insufficient memory available.
29571	7383	Wrong intermediate variable value
29572	7384	Error during reading wave file - check wave file for format compatibility.
29573	7385	Could not open indicated wave file.

1.8 Engineering interfaces and tools

1.8.1 Export and import interfaces

1.8.1.1 Exporting and importing ECAD data (PBF)

Automation Builder provides an ECAD interface for exchanging the PLC configuration data with EPLAN Electric P8 and Zuken E3. This feature removes double data entry between electrical engineering in the ECAD tool and the control logic programming in Automation Builder by synchronizing the PLC hardware including topology and I/O signals between these tools.

Automation Builder - ECAD interface supports various flexible workflows:

- Enables PLC hardware planning and configuration in the ECAD tool and allows importing the exported data from the ECAD tool through the *PBF* file (process integration bus interchange format) into the Automation Builder project with diff and merge functionality, providing full control on selective import/merge.
- Enables PLC hardware configuration in Automation Builder and allows exporting the configuration to the ECAD tool through a *PBF* file.
- Supports bi-directional roundtrip engineering with loss less data exchange between Automation Builder and the ECAD tool.

Automation Builder uses the rack information to identify the relations between:

- PLC and devices plugged to I/O bus or extension bus.
- Fieldbus slave and attached IO devices.

It is recommended to assign the PLC, IO devices, communication modules and fieldbus slaves properly to the rack in the ECAD project. If the rack information is missing, devices will be imported to the device pool and must be arranged manually in the Automation Builder project or mapped to already existing devices.

1.8.1.1.1 Requirements on EPLAN electric P8

- EPLAN Electric P8 with PLC and Bus Extension. It is recommended to use version 2.3 or later.
- Use of appropriate part data and macros for ABB devices. This can be achieved by getting the part data and macros from the EPLAN data portal.

1.8.1.1.2 Importing PLC data from the ECAD tool

You can create a new Automation Builder PLC project from the existing PLC hardware configuration in your ECAD tool, by importing the exported *PBF* file to Automation Builder.

Import *PBF* file to Automation Builder

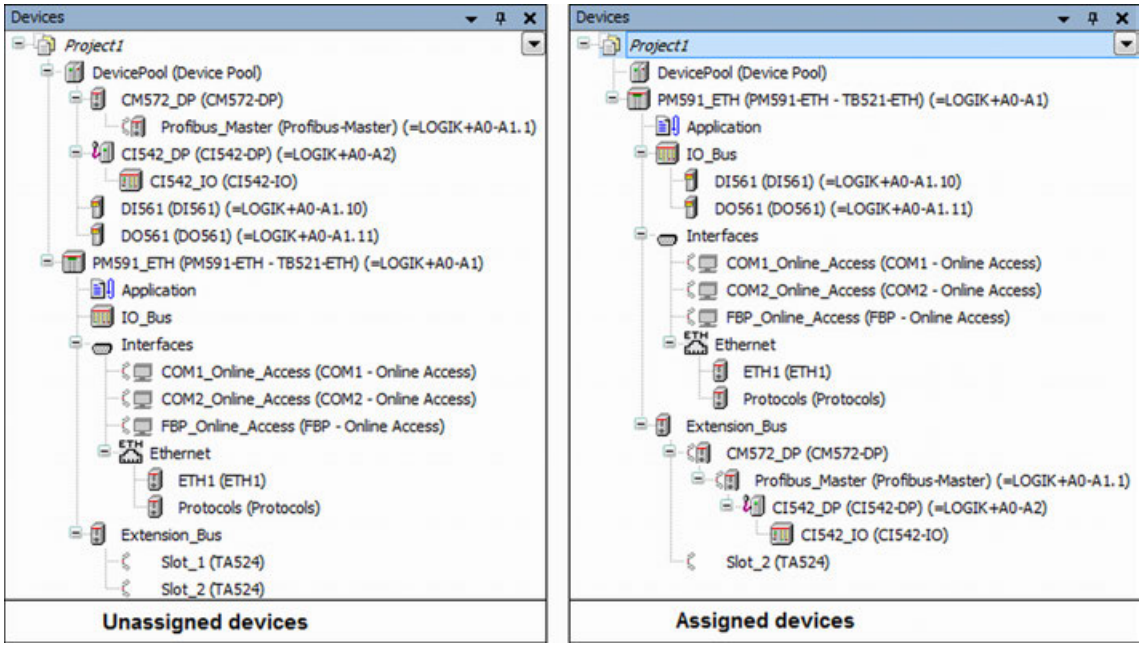
1. From the main menu, select *“Project ➔ Import ➔ ECAD (PBF)”*.
2. From the file system, select the *PBF* file.

   Automation Builder starts importing the devices and its associated signals from the *PBF* file. After a successful import, the result is displayed in the **Project Compare –Differences** view. You can now decide and selectively merge the differences.
3. Select the DevicePool node and click *“Accept Block”* to accept the complete PLC structure in the ECAD tool.
4. Select the PLC node and click *“Accept Block”* to accept all child device nodes.

   The DevicePool node holds all devices coming from the ECAD tool without any hierarchy information. The missing hierarchy information can be defined after closing the editor.
5. Close the **Project Compare – Differences** view to accept the changes.

6. Arrange unassigned devices in the DevicePool to the PLC hardware structure by drag-and-drop.



⇨ The I/O signals assigned to I/O devices in the *PBF* file are imported and allocated to IO devices. IO signals can be viewed in I/O mapping editor of the I/O devices.

1.8.1.1.3 Importing third party devices

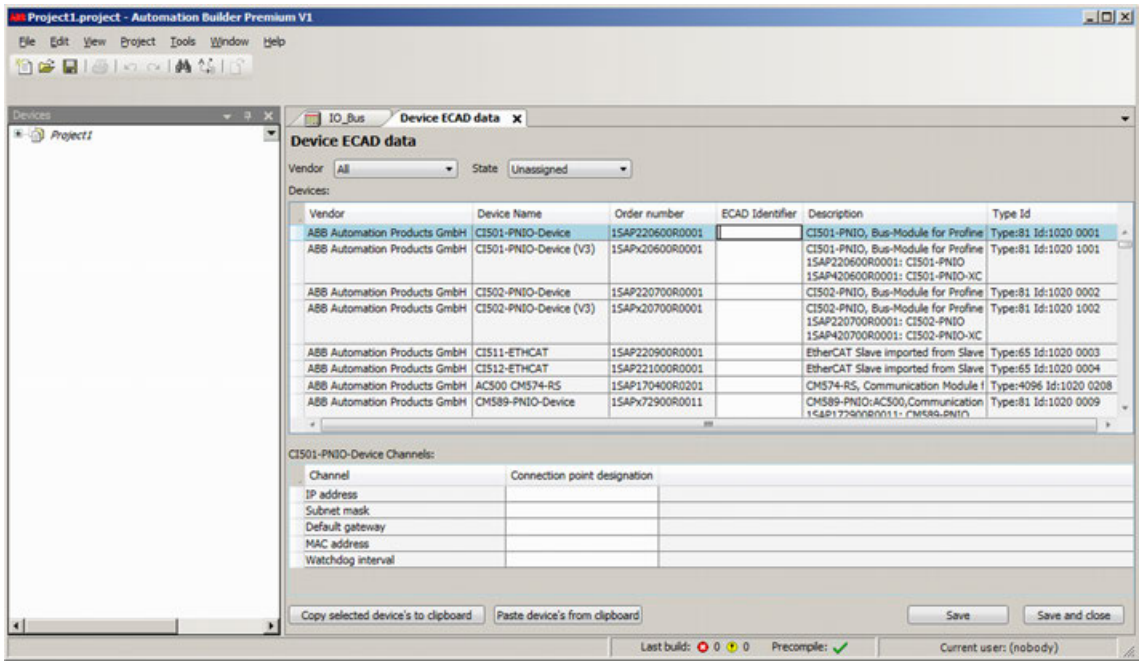
Prerequisite: To import third party devices from ECAD to Automation Builder, install third party fieldbus devices (for example, GSD, GSDML and EDS files) using *"Tools ➔ Device repository"* in Automation Builder.

1. From the main menu, select *"Project ➔ Import ➔ ECAD (PBF)"*.
2. From the file system, select the ECAD *pbf* file which consists of third party devices.

   ⇨ When the device identifier of the third party device installed in Automation Builder does not match with the device identifier of the device imported from ECAD, an error window is shown with the devices which are failed to import with error identifier 14.

   To import third party devices, it is required to assign ECAD identifier (PLC type designation/order number) in Automation Builder in *"Tools ➔ Device ECAD data"*. Click the link in the *Import* window to see the error messages in a text file.
3. Click *"Continue"* in the Import window to import valid devices to the project that are imported successfully or click *"Cancel"* to cancel the import process.
4. In Automation Builder, click *"Tools ➔ Device ECAD data"*.

5. In the Device ECAD data editor, add the ECAD identifier for the devices shown in the import errors window with error identifier 14, to enable these devices for export and import.



⇨ Also, add the ECAD identifiers for all devices which need to support export/import in ECAD.

6. Reimport the *pbf* file to import the third party devices.

1.8.1.1.4 Exporting PLC data to ECAD tool

1. Open the existing PLC project.
2. In the device tree, right-click *“PLC ➔ Export ➔ ECAD (PBF)”*.
3. Select the desired location in the file system to save the *PBF* file.

   The ECAD user can import the exported *PBF* file from Automation Builder and can use the imported PLC data for electrical engineering purpose. If the user modifies imported PLC data in the ECAD project, the data can be imported back to the Automation Builder project which supports the round trip engineering efficiently with loss less synchronization of the data.

1.8.1.1.5 Exporting third party devices

1. Right-click on a PLC device, click *“Export”* and select *“ECAD (PBF)”*.
2. Save the file to the desired location in the file system.

   If the third party devices does not contain assigned ECAD identifiers, a message is displayed showing which devices cannot be exported.

   ⇨ To add ECAD identifiers to the devices, see *Importing third party devices* ↳ *Chapter 1.8.1.1.3 “Importing third party devices” on page 6551*.

   After adding ECAD identifiers to the third party devices, execute *“Export”* to export the devices including third party devices.

1.8.1.1.6 Importing ECAD PLC data to existing AB project

Automation Builder ECAD interface supports concurrent engineering by importing the ECAD data to the existing Automation Builder PLC project.

1. From the main menu, select *“Project ➔ Import ➔ ECAD (PBF)”*.
2. Select the PBF file which has been created during the export from the ECAD tool.
3. Select the PLC from the list and click *“OK”*.
   ⇨ A dialog window is displayed if the Automation Builder project provides PLCs of the identical type as defined in the PBF file.

   By selecting *“None”* in the dialog window a new PLC is defined in the ECAD tool.
4. In the **Project Compare – Differences** view, click  to merge device signals.
   ⇨ The differences between the current PLC hardware configuration in Automation Builder and the ECAD PLC data are displayed.
5. Select the differences as desired and click *“Accept Single”* to accept the selected difference block.
6. Close the **Project Compare – Differences** view to accept the changes.

1.8.1.1.7 Arrange or map devices imported to the device pool

Devices that are imported to the device pool because of missing hierarchy information (mainly rack information) must be arranged manually in the Automation Builder project or mapped to already existing devices.

Arrange devices imported to the device pool

Arrange the unassigned devices in the DevicePool to the PLC hardware structure by drag-and-drop.

Map devices imported to the device pool

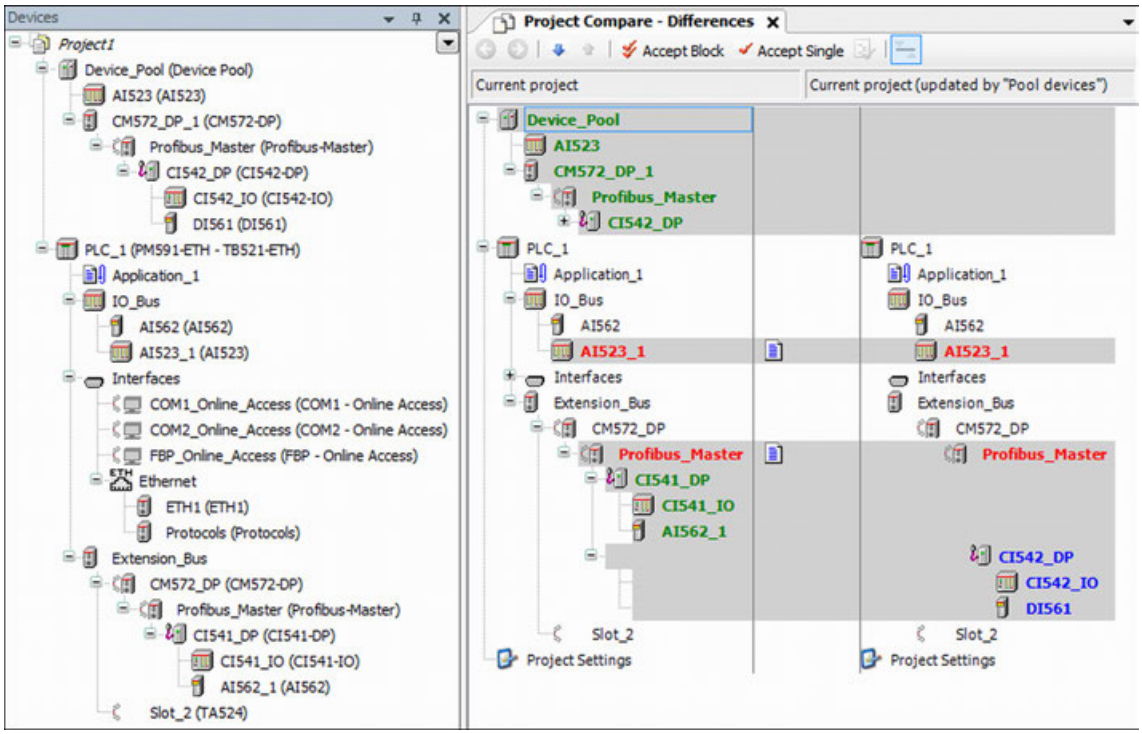
If the devices are already added to the Automation Builder project prior to the import, you have to map the instances of the same type manually (one instance in the Automation Builder project tree and one instance in the DevicePool).

After mapping the devices, you can selectively merge the device parameter or signal in the difference view.

To map pool devices, proceed as follows:

1. In the device tree, select the Device Pool node, click *“Project”* and select *“Map pool devices”*.
2. Map the device pool instances of identical types in the project from the drop-down list and click *“OK”*.

   ⇨ Pool devices which are mapped are removed from the device pool and mapped to the corresponding Automation Builder device. Differences between the signals of the mapped I/O devices are displayed. e.g. AI523_1 device:



1.8.1.1.8 Limitations

The following limitations are considered when working with the Automation Builder ECAD interface:

- The scope of a PBF file is limited to one single PLC including all connected devices.
- There is no representation of XC variants of devices in Automation Builder. Therefore, always use the standard variant for export. This might lead to part data mismatch when importing into the ECAD tool.
- In reimport or round trip import cases, if any changes are made in ECAD by adding a new communication module with connecting to one of the PLC slot or replacing existing communication module, then those device changes to the communication modules are not displayed as connected to PLC slots during the import in Automation Builder diff and import, instead those CM modules are added under the device pool. After merging and importing is completed, to work with device pool devices ↳ *Chapter 1.8.1.1.7 “Arrange or map devices imported to the device pool” on page 6553*.
- IO mapping data cannot be imported for IO devices plugged to an EtherCAT slave when they are imported individually to the device pool because of missing hierarchy information. After arranging the devices properly in the device tree, the import can be done again to import also the IO mapping data.

1.8.1.2 Exporting and importing I/O mapping (CSV)

The I/O module mappings of an Automation Builder project can be exported to CSV for bulk editing in MS Excel or other documentation purposes. I/O mappings can be exported at single I/O module level or at PLC level.

Further, the I/O module mappings can be imported with the option of displaying differences and merging each single changed or import all signals at once by overwriting existing I/O module signals.

1.8.1.2.1 Exporting IO mapping data to CSV

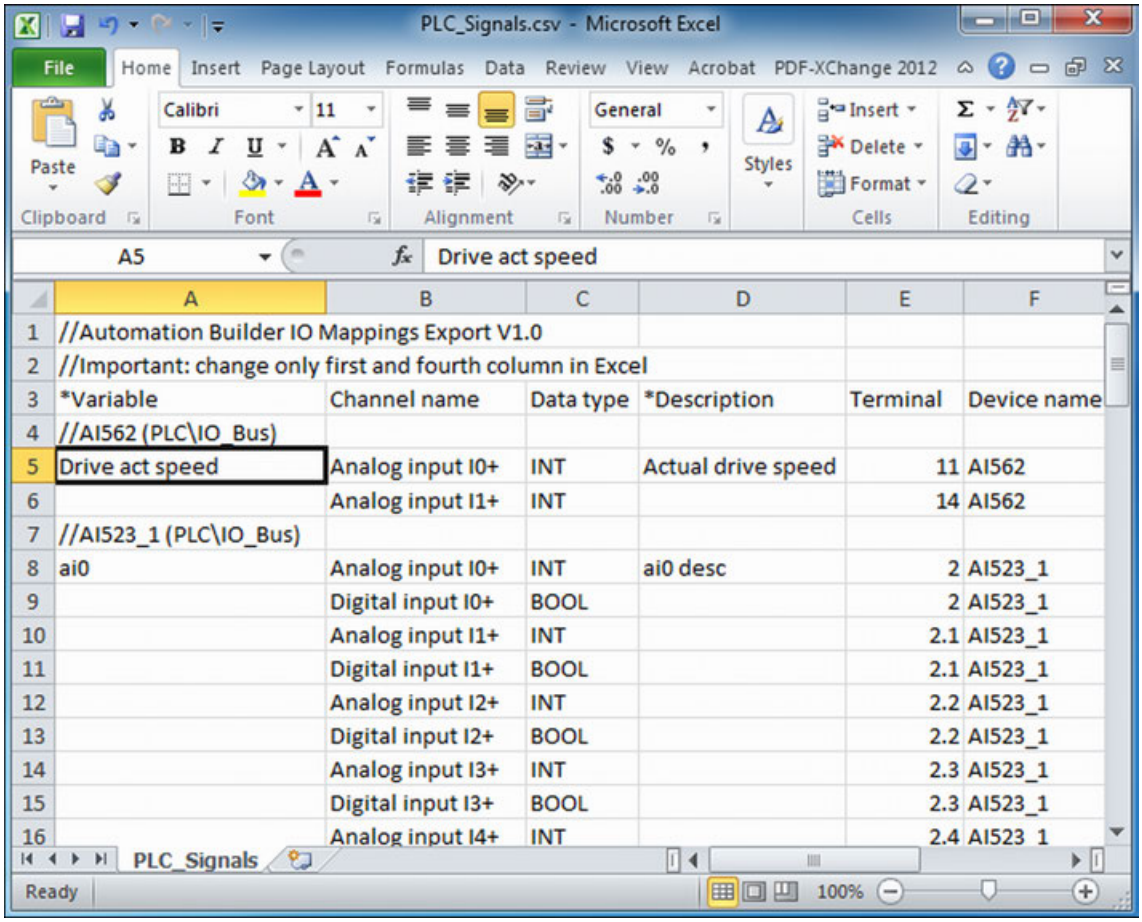
To export I/O mappings to a CSV signal list, proceed as follows:

1. In the device tree, right-click *"PLC ➔ Export ➔ IO mapping (CSV)"*.
2. Save the **IO mappings CSV** to the desired location in the file system.

   If the CSV signal list has been exported successfully, a success message is displayed. The status of the export is shown in the dialog.
3. In the export dialog, click the link to open the exported **IO mapping CSV** file in MS Excel.

> *The template can only be opened if MS Excel is installed and configured to open .csv files.*

4. In the IO mapping (CSV) file, change **Variable** and **Description** fields to edit I/O mappings.



	A	B	C	D	E	F
1	//Automation Builder IO Mappings Export V1.0					
2	//Important: change only first and fourth column in Excel					
3	*Variable	Channel name	Data type	*Description	Terminal	Device name
4	//AI562 (PLC\IO_Bus)					
5	Drive act speed	Analog input I0+	INT	Actual drive speed	11	AI562
6		Analog input I1+	INT		14	AI562
7	//AI523_1 (PLC\IO_Bus)					
8	ai0	Analog input I0+	INT	ai0 desc	2	AI523_1
9		Digital input I0+	BOOL		2	AI523_1
10		Analog input I1+	INT		2.1	AI523_1
11		Digital input I1+	BOOL		2.1	AI523_1
12		Analog input I2+	INT		2.2	AI523_1
13		Digital input I2+	BOOL		2.2	AI523_1
14		Analog input I3+	INT		2.3	AI523_1
15		Digital input I3+	BOOL		2.3	AI523_1
16		Analog input I4+	INT		2.4	AI523_1

⇨

> *Do not modify other field's data in IO mapping (CSV) file.*

1.8.1.2.2 Importing I/O mapping data from CSV

To import an edited I/O mapping (CSV) file, proceed as follows:

1. From the main menu, select *“Project ➔ Import ➔ IO mapping (CSV) ➔ Open”*.
2. A CSV signal list import dialog is displayed.
   ⇨ With *“YES”*, all I/O mappings will be imported without difference view. With *“NO”*, the difference view is displayed with the I/O mapping differences.
3. In the **Project Compare – Differences** view, click  to merge I/O mappings.
4. Select the signal row for which the difference is to be accepted. Select the **Variable** field and click *“Accept Single”* to merge the I/O mappings.
5. Close the **Project compare – Differences** view to accept the changes and merge the I/O mappings with the Automation Builder project.

1.8.1.3 Exporting and importing device list (CSV)

The Automation Builder project devices can be exported to CSV for bulk device renaming or adding device tag labels to devices in MS Excel or other documentation purposes. A devices export is only possible at PLC level.

Automation Builder provides importing devices in bulk based on device type, instance and hierarchy information provided in the CSV file.

1.8.1.3.1 Exporting device list to CSV

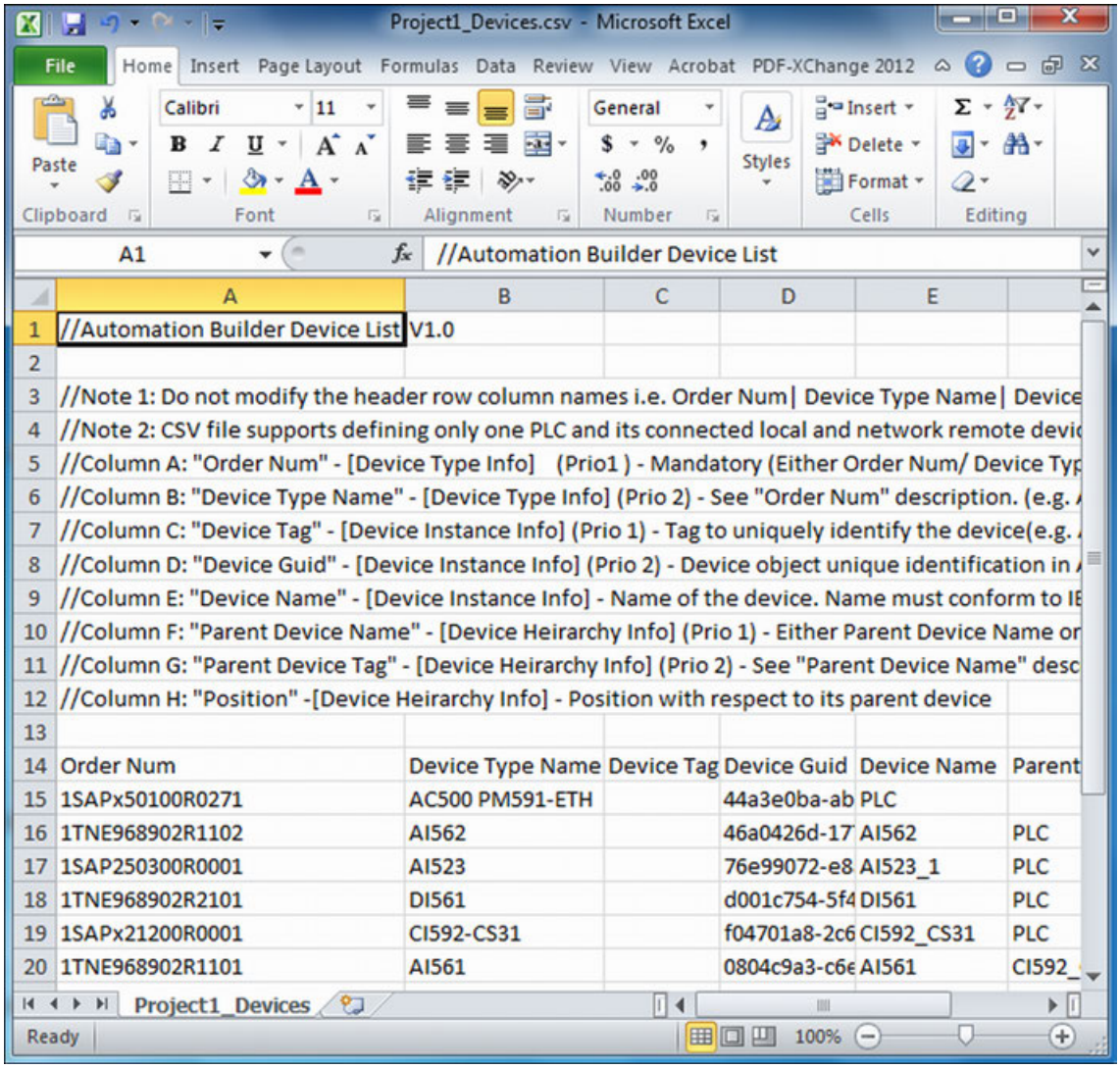
To export a CSV device list, proceed as follows:

1. In the device tree, right-click *“PLC ➔ Export ➔ Device list (CSV)”*.
2. Select the desired location in the file system to save the **Device list (CSV)**.

   If the CSV device list is exported successfully, a success message is displayed.

3. In the Export dialog, click the link to open the exported CSV device list.

   The exported CSV device list consists of all devices connected to the PLC that is exported. Each row represents a device with its device type and hierarchy information.



1.8.1.3.2 Creating CSV device list

To create the devices in CSV, use the device list template provided in Automation Builder.

▷ In the main menu, click *"Tools ➔ Create CSV Device list"*.

⇨ The device list template is opened in the MS Excel.

> *The template can only be opened if MS Excel is installed and configured to open .csv files.*

In this file, add each device in a separate row with device information like Device Type (Order Num or Device Type Name) and instance details (name, tag) and hierarchy information (parent Device name, parent Device Tag, position). The mandatory information required to import CSV is only Device Type. All other fields are optional. After editing the device list CSV file, save it in the file system and close.

1.8.1.3.3 Importing a device list from CSV

To import devices from CSV in bulk, proceed as follows:

1. From the main menu, click *“Project ➔ Import ➔ Device list (CSV)”*.
2. Select the device list CSV file from the file system and click *“Open”* in the Import dialog.

   All devices that are defined in the CSV are imported. The **Project Compare – Differences** view displays the current project and the project that has been updated by the import file.
3. Select the desired devices and click *“Accept Block”* to accept all the devices and its child device nodes or *“Accept Single”* to accept only a single device.
4. After closing the **Project Compare – Differences** view, the devices are imported to the Automation Builder project.

   ⇨ The devices (except PLC) are placed under the device pool if the valid device hierarchy information is not provided in the CSV device list file. By drag-and-drop devices can be assigned to the desired PLC hardware structure ↬ *Chapter 1.8.1.1.7 “Arrange or map devices imported to the device pool” on page 6553*.

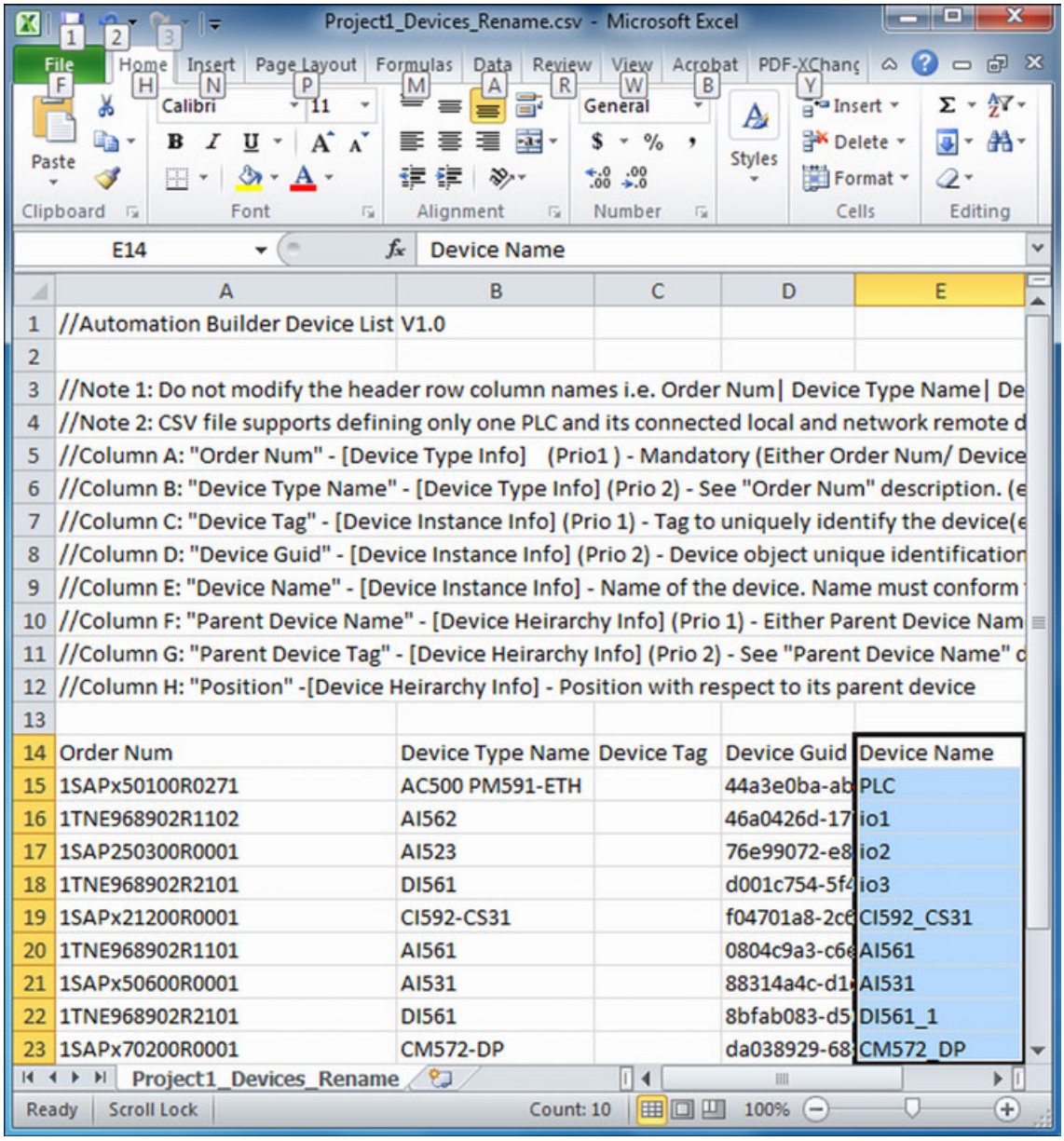
   If a device tag is provided for a device in CSV, it appears next to each device node in the device tree.

1.8.1.3.4 Renaming devices

To rename the devices, proceed as follows:

1. In the device tree, right-click *“PLC ➔ Export ➔ Device list (CSV)”*.
2. Select the desired location from the file system to save the CSV device list.

3. Rename the device names in the column **Device Name**:



4. Click *“Project ➔ Import ➔ Device list (CSV)”*.

5. Select the updated CSV file from the file system.

   Open the **Project Compare – Differences** view. If only the device names have been changed in the CSV file, the difference view does not show the changes.

> *Device Name changes are not displayed as changes in the difference view.*

6. Close the **Project Compare – Differences** view. The Renamed Devices dialog is displayed with the current name and the new name provided in the CSV file.

7. In the Rename Devices window, select the desired devices and click *“OK”*.

   The device names are updated in the Automation Builder project.

1.8.2 Virtual commissioning technology

Virtual Commissioning Technology offers the user to do commissioning of the devices in advance. The user can verify variable data exchange between different devices and run those in simulation. The user need to create system model which represents physical system for commissioning.

Virtual Commissioning Technology supports AC500 V2 PLC, ACS380 Drive and ACS580 Drive. To work with this feature, apart from essential license, the user need to enable Virtual Mode (*"Online ➔ Virtual Mode"*).

Drive parameterization is required to start data exchange between virtual drive and virtual PLC. For information on drive parameterization, see PROFIBUS and PROFINET manuals.

1.8.2.1 Virtual mode

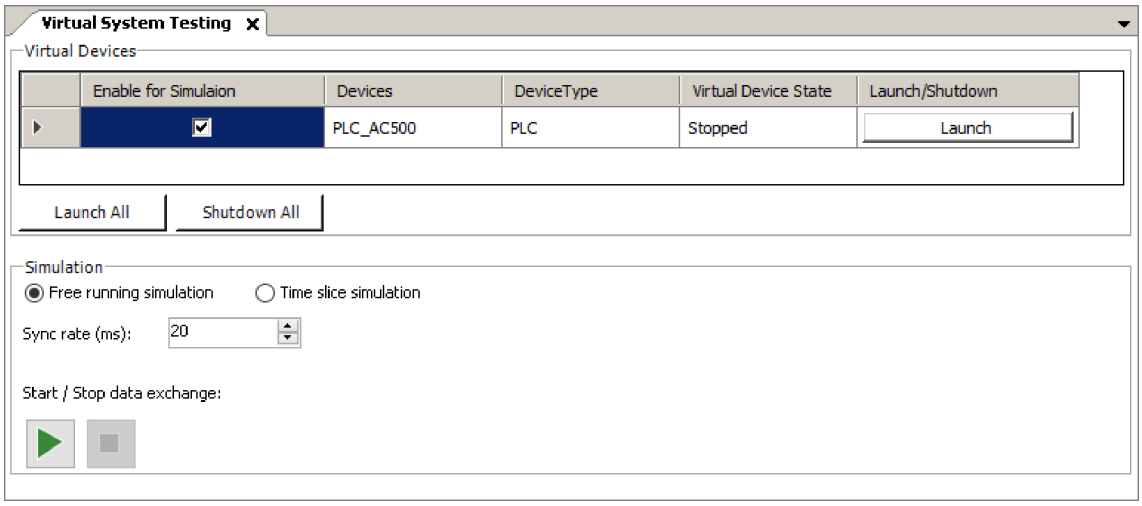
Virtual Mode option enables virtual mode for Automation Builder. This mode is retained even after closing the project. To disable virtual mode, the user need to undo the selection of Virtual Mode option. To enable Virtual Mode, select *"Online ➔ Virtual Mode"* in the Automation Builder main menu.

The Virtual Mode is enabled and the Virtual System Testing selection appears in the menu.

1.8.2.2 Virtual system testing

Virtual System Testing devices (launch/shutdown) monitor its process state and allows simulation of those devices based on selection.

The Virtual System Testing editor contains Virtual Devices and Simulation sections.



Virtual devices

Virtual Devices section helps the user to process and monitor virtual devices by:

- Launching virtual device(s)
- Monitoring state of virtual device
- Shutdown virtual device(s)
- Selection of devices for simulation

Launching virtual devices

The user can launch all the virtual devices together using *"Launch All"* or for individual virtual device using *"Launch"*. The state of buttons changes from launch to shutdown when the selected virtual device starts running. Virtual Device State represents the state of virtual device process and independent of simulation state.

Virtual online mode

The process of logging in to virtual PLC is same as real PLC.

After launching the virtual device, login the application in the device tree to enter online mode.

- In the devices tree, right-click on *"App"* and select *"Login"* to login the application.

The virtual device enters online mode and connected virtually with the PLC device.

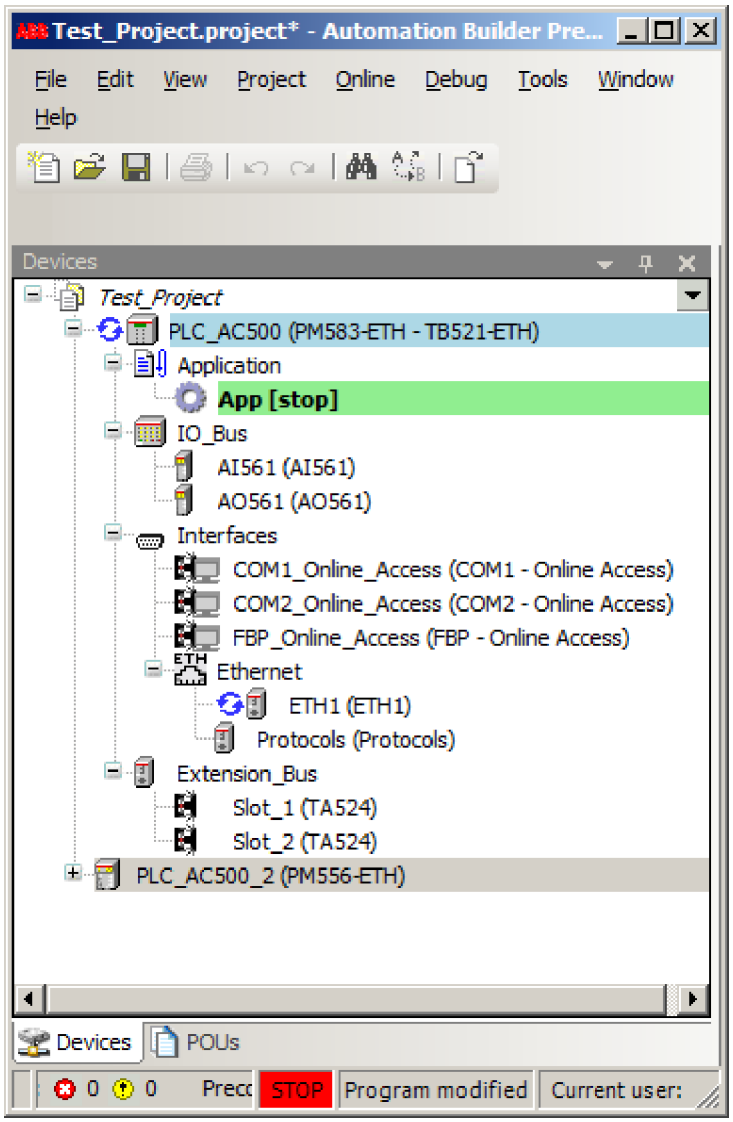


Fig. 1177: Online mode

> *If there is no program on the controller, the application notifies to download a new program.*

Shutdown virtual devices

The user can shutdown all the virtual devices together using *"Shutdown All"* or for individual virtual device using *"Shutdown"*. The state of buttons changes from shutdown to launch when the selected virtual device stops.

1.8.2.3 Simulation

The simulation runs a program in virtual devices which are selected in Virtual Devices section. The configuration of a system model should be correct to proceed with the simulation. The variable exchange between different devices occurs when the simulation is running.

Without system model, the simulation can be run independently for selected devices. The user can do this when there is no variable data exchange desired.

The simulation supports **Free running simulation** and **Time slice simulation** modes. The user need to choose the mode and then launch virtual devices in respective mode. After launching virtual devices, switching to the other mode is not allowed.

Checking signal consistency

Before starting simulation, the signal consistency check is performed on the following criteria.

- In a System Model, the broken signal should not exist. While configuring System Model, if you delete any mapped model block instance then it will create a broken signal. The user need to clear broken connections using **Clear Broken Connection**.
- Any input pin cannot be mapped to multiple output pins.
- Any mapped signal must have variable associated with it.
- Variable associated with mapped signal must have a matching data type.

If a configured system model fails to achieve any of these criteria, an error message is shown to the user and the simulation cannot be allowed to continue.

Free running simulation

In Free running simulation, the virtual devices are not synchronized through virtual time. Each device which has participated is running freely. The user can only Start and Stop the simulation. The time elapsed is displayed to indicate the time for which simulation is running.

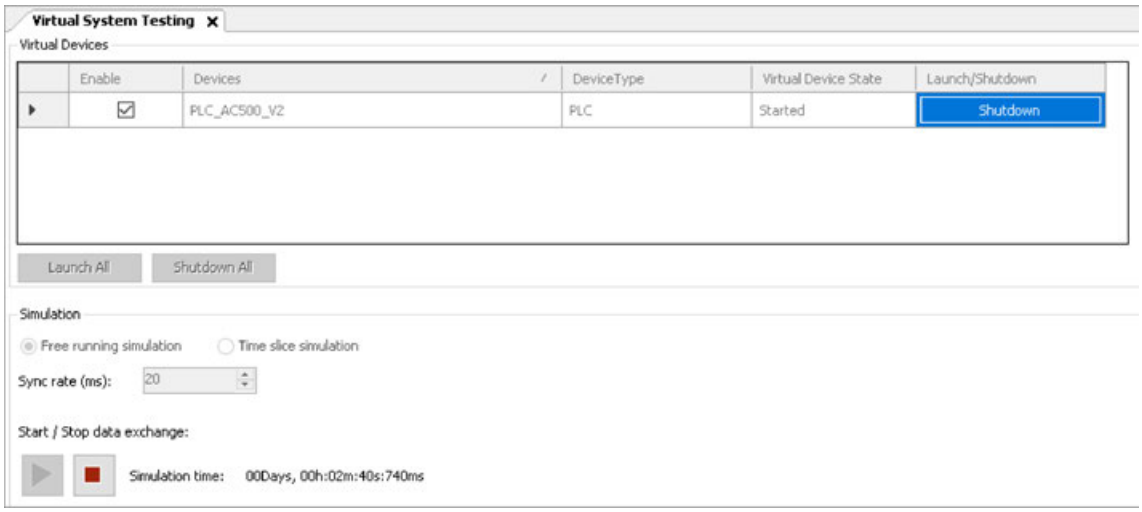


Fig. 1178: Free running simulation

Before starting simulation, the user need to define:

- **Cycle time**: Defines interval for variable data exchange in free running mode.
- **Reset PLC**: If selected, earlier state of virtual device will be reset.

Time slice simulation

In Time slice simulation mode, the virtual devices are always synchronized through virtual devices. In this mode, the Steps and Pause buttons are enabled. The user has the two entry points Start or Steps for simulation.

The Virtual time displays the time for which the virtual devices are running. This time is different than that of real time.

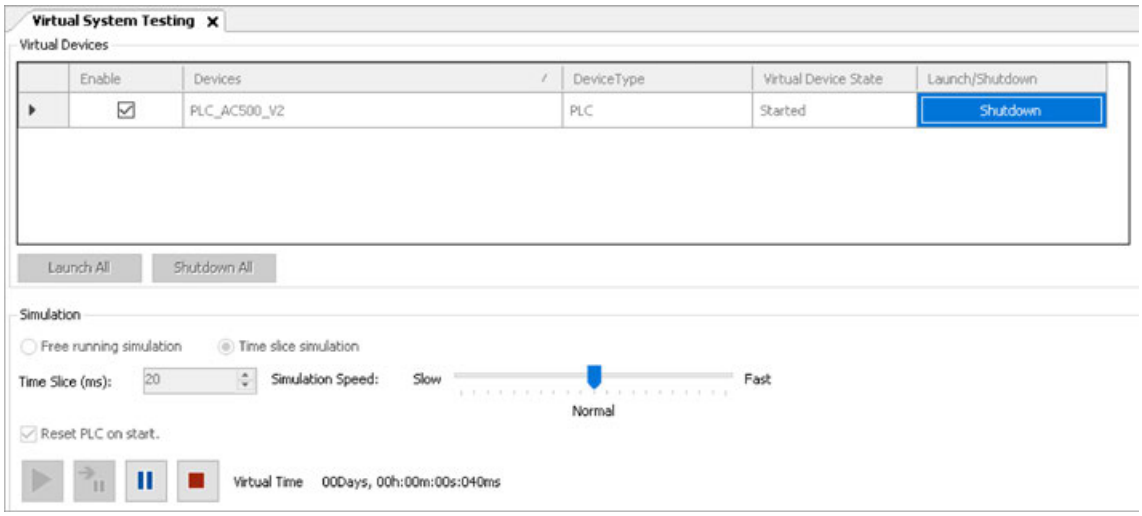


Fig. 1179: Time slice simulation

Before starting simulation, the user need to define:

- **Time Slice (ms)**: Indicates time for which the virtual devices are running before synchronizing with each other. Also, indicates the time interval for variable data exchange in time slice mode. For example, if the selected time-slice for simulation is 20 ms and scan time of PLC is 10 ms, it means in each time-slice, the PLC executes its program twice.
- **Simulation Speed**: Indicates or decides how fast the simulation is executed. The user can change the value even when the simulation is in progress. The simulation speed is independent of time-slice.
- **Reset PLC on start**: If selected, earlier state of virtual device is reset.

> *If variable exchange is not occurring as per expectations, make sure that all the variables defined in system model are available in symbol file. To update symbol file, make sure that the symbol file generation option is selected from CODESYS application ("Project ➔ Options").*

1.8.2.4 Protocol switch

In Automation Builder, the projects can be switched between real PLC mode to/from virtual mode. Automation Builder internally manages the commands and parameters that are given to Panel Builder software to update the changes in protocol to communicate with real or virtual PLC.

Automation Builder verifies the following conditions each time Panel Builder is launched.

- If the project is switched between virtual to normal mode or vice versa, a message is displayed. You can update the panel project to use new protocol or cancel the update.
- If the project is switched between virtual to normal mode or vice versa and the Drive application program is not build, an error message is displayed. The Drive application program must be configured and rebuild to update Panel Builder project.
- When the virtual mode project is switched to normal mode during which Panel Builder is already open, a message is displayed. You can close and relaunch Panel Builder by activating Update Panel Builder project on launch check box.
- When the virtual mode project is switched to normal mode, during which if you try to launch Panel Builder with Update Panel Builder project on launch unchecked, a message is displayed. The tags and protocol information of panel project will not synchronize with Automation Builder.

1.8.2.5 Virtual AC500 V2 extensions

1.8.2.5.1 Introduction

The Virtual AC500 V2 is a virtual PLC running on the PC. The target is to create the engineering and the commissioning in the same way as for a real AC500 V2, with some minor exceptions which are described in the following. Only the AC500 function blocks listed in this documentation are supported by the Virtual AC500 V2.

This documentation provides only the exceptions, limitations and hints with the Virtual AC500 V2. For further information about the function blocks please follow the AC500 documentation.

1.8.2.5.2 General

The PLC configuration created in Automation Builder is not taken into account by the Virtual AC500 V2. Some configuration parameters that are required must be created in a Global Variable List (GVL) “VirtualAC500_Configuration” in CODESYS V2.3.

It is also required that all these variables are added to the symbol file of the PLC.

1.8.2.5.3 UDP/IP function blocks

The supported function block are:

- ETH_UDP_STD_INFO
- ETH_UDP_STD_SEND
- ETH_UDP_STD_REC
- ETH_UDP_STD_INFO
- ETH_UDP_STD_SEND
- ETH_UDP_STD_REC

Function block in- and outputs

Input “SLOT” is ignored in all function blocks.

Only the status information about the level of all receive buffers in bytes (output “LEVR_BY” and “LEV_BY”) is evaluated. All other status informations are always 0.

Configuration

In the Virtual AC500 V2 receiving of broadcast messages is always enabled and cannot be disabled.

The connection parameter “Port” is required and must be added as variable to the GVL “VirtualAC500_Configuration”.

Syntax

ETH_UDP_STD function blocks

CCF_UdpConnection<Index>_Port

- <Index> is the number of the connection.

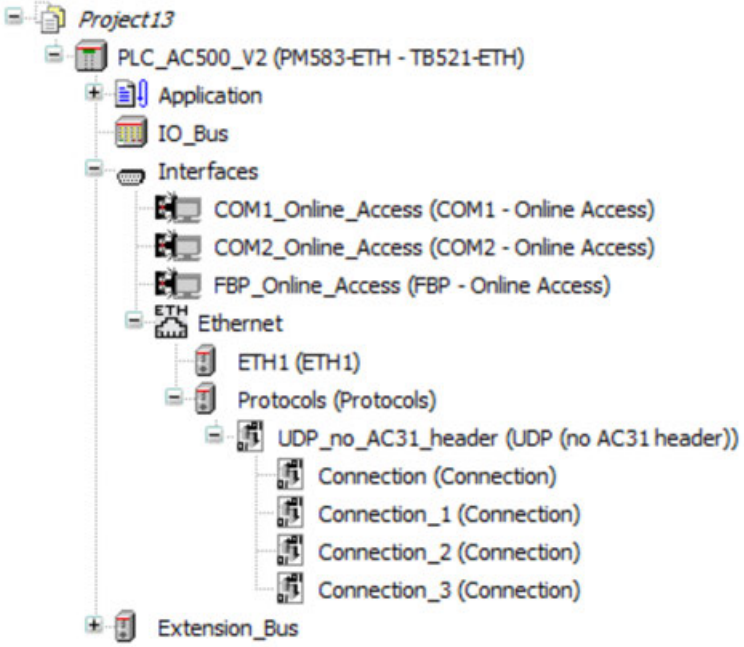
ETHx_UDP_STD function blocks

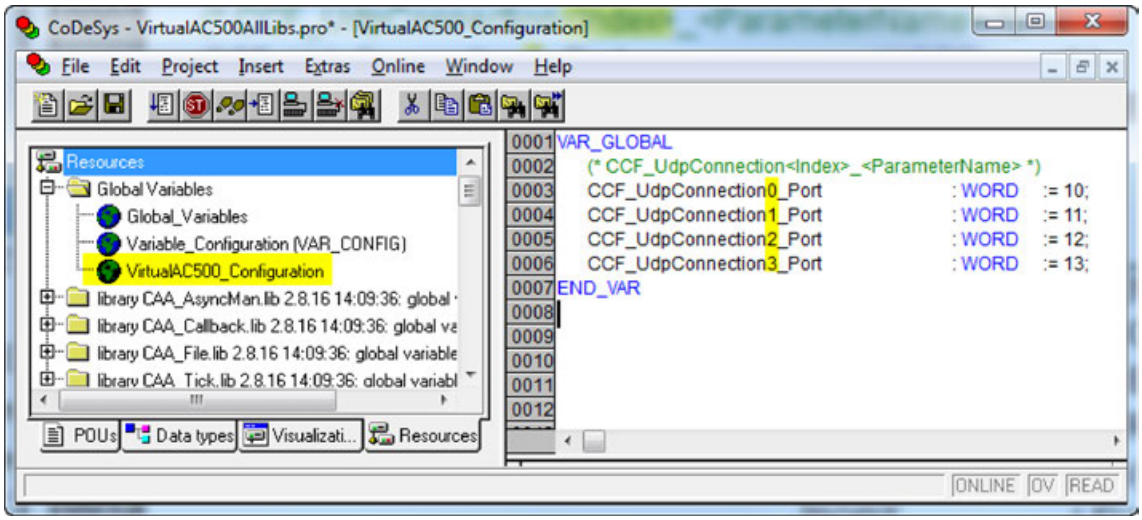
CCF_ETH<ETH index>_UdpConnection<Index>_Port

- <ETH index> is the value of the input variable *ETH* of the UDP function blocks.
- <Index> is the number of the connection.

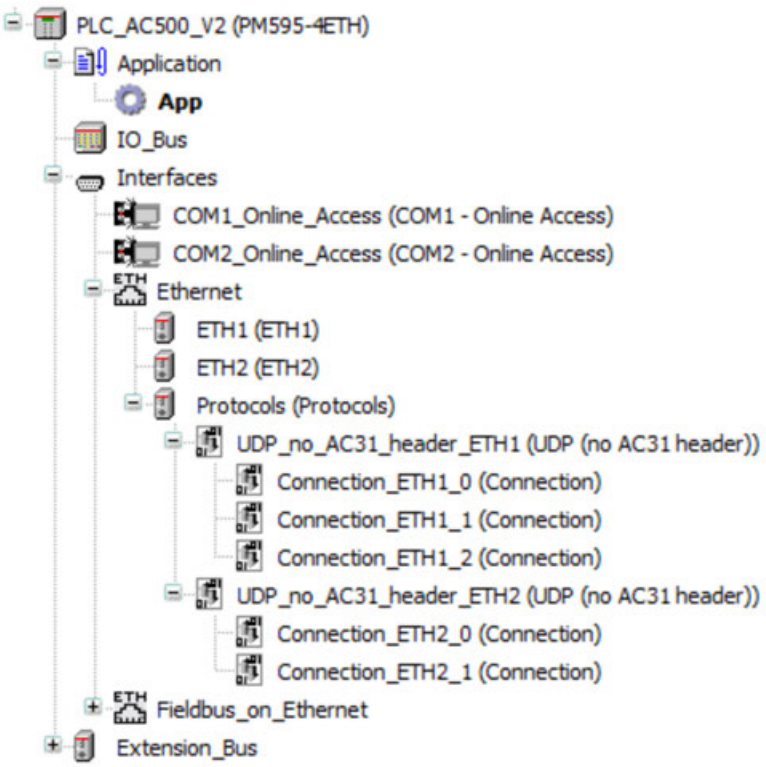
Examples

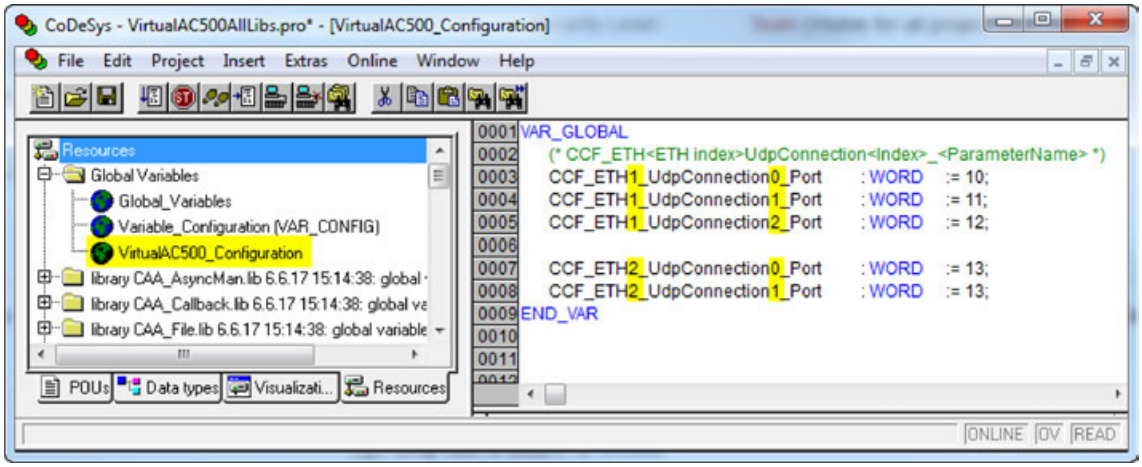
ETH_UDP_STD function blocks





ETHx_UDP_STD function blocks





1.8.2.5.4 CAA_File function blocks

The supported function blocks are:

- FILE_Open
- FILE_Close
- FILE_Read
- FILE_Write

The CAA_File function blocks are used for accessing directories and files. There are several directories from the PLC from where files can be read and written equal to the real AC500, e.g.: flash disk, user disk, memory card. The files and directories are created on the local file system in the working directory of the Virtual AC500 V2.

1.8.2.5.5 Clock function blocks

The supported function blocks are:

- CLOCK
- CLOCK_DT

These two function blocks are used to get and set the current time and date. The current time and date is initialized with the Windows clock during startup of the PLC. All further changes on time and date are independent from the Windows clock, so it is possible to define time and date for each Virtual AC500 V2 instance.

1.8.2.5.6 Ethernet function blocks

The supported function blocks are:

- ETH_OWN_IP
- ETH_ICMP_PING

The function block “ETH_OWN_IP” delivers the IP address of the Ethernet interfaces of the PC where the Virtual AC500 V2 is running. With input “SLOT” it is possible to get the settings from all Ethernet interfaces (0 = first interface, 1 = second, etc.).

The function block “ETH_ICMP_PING” ignores the input value “SLOT”.

1.8.2.5.7 Error numbers

The error numbers of the Virtual AC500 are depending on the associated guest operating system.

Therefore the error numbers of the Virtual AC500 can be different to real AC500 V2.

A list with possible error numbers are to be found in MSDN.

1.8.3 System model

The System Model provides the abstract view of the system which the user wants to configure. It captures physical devices and signals that are exchanged between different devices. Features like virtual commissioning use the system model to capture information needed for variable exchange and simulation.

The System Model object represents all the information configured using system model editors. The user can have only one system model object in Automation Builder project.

To complete the System Model configuration, the user needs to ceate model block class objects and model block instance objects.

1.8.3.1 Creating a system model

1. In the devices tree, right-click on project and click *“Add object”* to add System Model.
2. Select System Modelling in the Categories section and click *“Add object”*.
   ⇨ System Model object is added to Automation Builder project.

3. Under *“System Model”*, add *“Model Block Class”* and their instance objects.

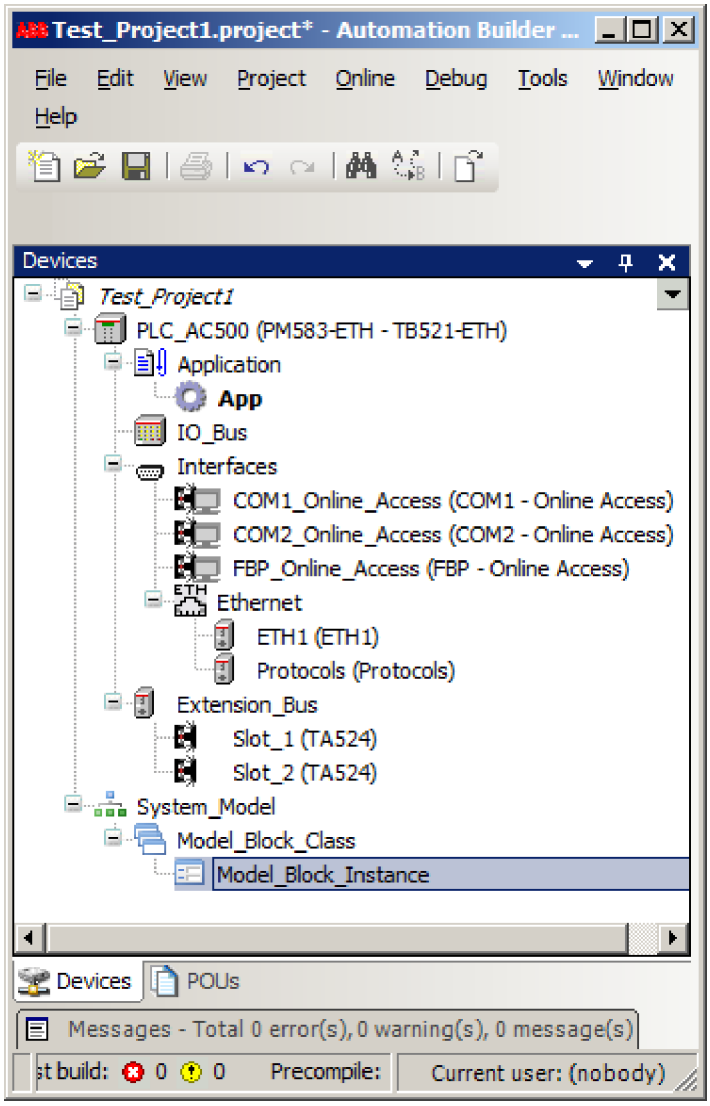
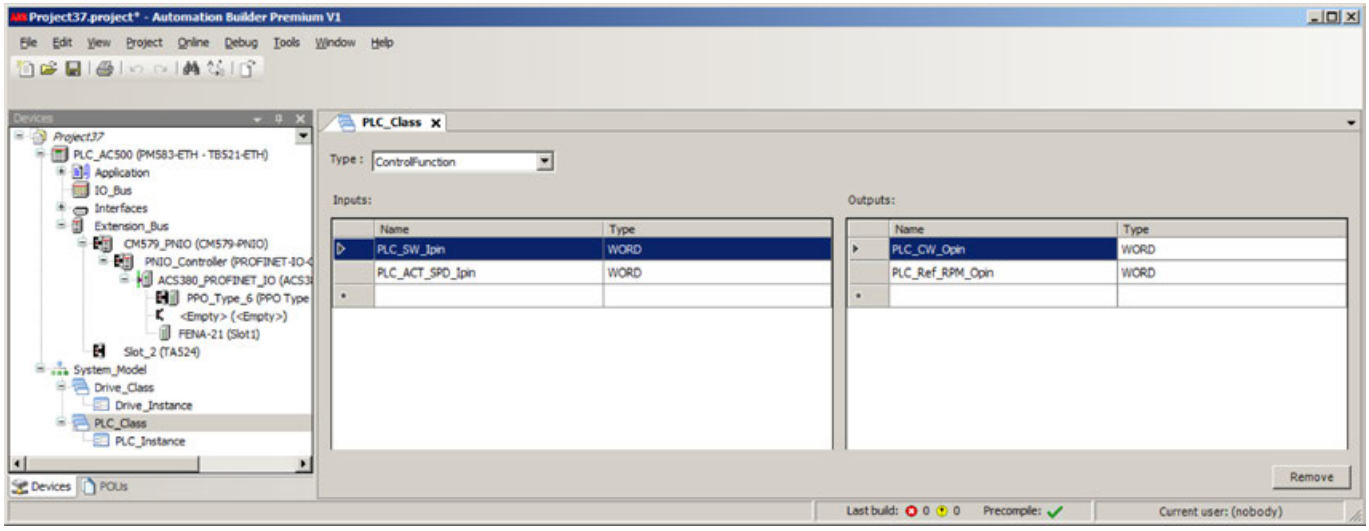


Fig. 1180: Model Block Instance

1.8.3.1.1 Model block class editor

The Model Block Class editor represents model block class object and supports following functions.

- **Create pin definitions** - The user can create definitions for input and output pin. The pin definition includes name and type of a pin. The pin name should be unique.
- **Remove pin definitions** - The user can remove pin definitions either by using 'Remove' button or by using 'Delete' Key. The user may remove multiple pin definitions at once.
- **Auto correcting pin definitions** - The pin name should be unique within a model block class object. If a duplicate pin names exists, the user can auto-correct the names. Appropriate indexes are used to correct duplicate names if the user chooses for auto correction.



1.8.3.1.2 Model block instance editor

The Model Block Instance editor represents the instances of the class object. The instance editor derives all the pin definitions from the class object and enables the user to create pin connections between different instance objects.

Using instance editor, you can do the following:

- Assigning a device
- Assigning variables
- Assigning endpoint
- Clear broken connections

Assigning a device

In the instance editor, first select the virtual device from the 'Assigned Device' drop down. No changes are allowed to the object or in the editor until the user selects the 'Assigned Device'.

Assigning variables

Variables defined in 'IO mapping' or as global variables are created as pins in the model block class object. The user can assign the variables using the inline editor or by using the **Assign Variable** button.

- To assign variables in the inline editor, enter the variable name in the 'Variable' column.
- To assign variables using 'Assign Variable' button, click **Assign Variables** to show a dialog containing all IO mapping and global variables of the corresponding device. Select the required variable and click **OK**. Connect Variable indicates a variable associated with connected endpoint.
- Only variables which match the direction and type of a pin are available for mapping. In case of any incorrect data, an error is displayed in the **Variable** column.

> *Use the Refresh button to fetch if the user makes any changes to variables, either using IO Mapping tab or using a Codesys application.*

Assigning endpoint

Defines the instances and pin connections (Source Block and Source Pin for inputs and Sink Block and Sink Pin for outputs). The user can assign endpoints using Inline editor or by Assign Endpoint button.

- To assign instances and pin connections in an Inline editor, enter the value in Source Block and Source Pin for inputs and Sink Block and Sink Pin for outputs or click on drop-down which displays the list of instances and pins.
- To assign instance and pin connections, click **Assign Endpoint** which displays the endpoint selection dialog. Select the required instance or pin and click **OK**.

Clear broken connections

The Clear Broken Connection clears all the broken source block instance and pin connections between Model Class objects. If there are any broken connection, the cell is indicated with an error icon. When an instance associated with any mapped signal is deleted, it will generate broken signal.

1.8.3.2 Generating system model

System Modelling allows you to generate System Model for Drives and I/O devices which can be readily used for Virtual System Testing.

- System Model generation is only implemented for AC500 V2 products.
- Deleting one node of a generated System Model will result in the entire System Model being deleted.
- Symbols are exposed for the devices which are used in System Model generation.
- It is recommended that the manual changes should not be made to any generated objects.
- If any changes are made to the devices which have a System Model, it is recommended to use ‘Update System Model’ command to keep the System Model up-to-date.

The System Model generation commands can be found in the context menu of a System Model object in the device tree.

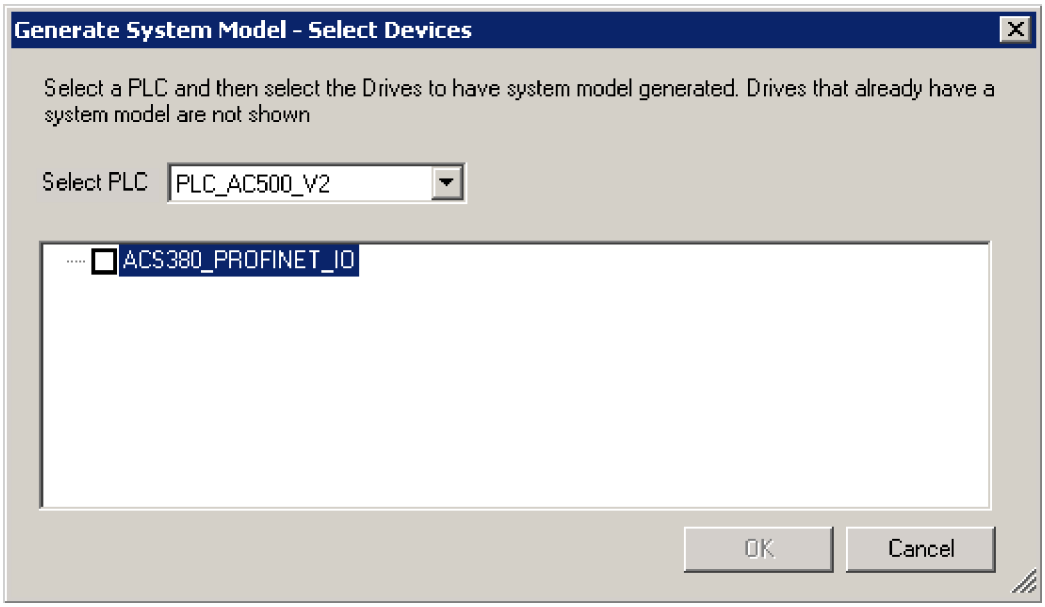
- Generate System Model for Drives
- Generate System Model for I/O
- Update System Model

Generating system model for drives

To generate System Model for the Drives, the user has to configure Automation Builder project with PROFIBUS or PROFINET drive and minimum of one variable mapping. To expose the symbols of a device, launch CODESYS application and generate the symbol file.

The command will generate a System Model for one or more selected drives.

1. Right-click on System Model node and select *“System Modelling ➔ Generate System Model for Drives”*.



2. Select the desired PLC from the drop-down list and select the Drive and click **OK** to generate a System Model.

   ⇨ The System Model is generated for the selected device(s). You can see multiple nodes are been added to the device tree under the System Model node.

   - Two Class and two Instance objects are generated for each device selected for generation.
   - The Drive Class object contain pin definitions for all the mapped variables on the selected device.
   - The Drive Instance object contain information about which pin/variable is mapped to the PLC Instance.

The PLC instance will contain exposed variables and the Drive instance will contain internal drive variables (FBAdatain/FBAdataout), but with the pin directions reversed. (for example, If the drive has an input variable ‘var’ mapped, the PLC equivalent will have an output variable ‘var’ mapped). At this point, the System Model is successfully generated and is ready for use in Virtual System Testing.

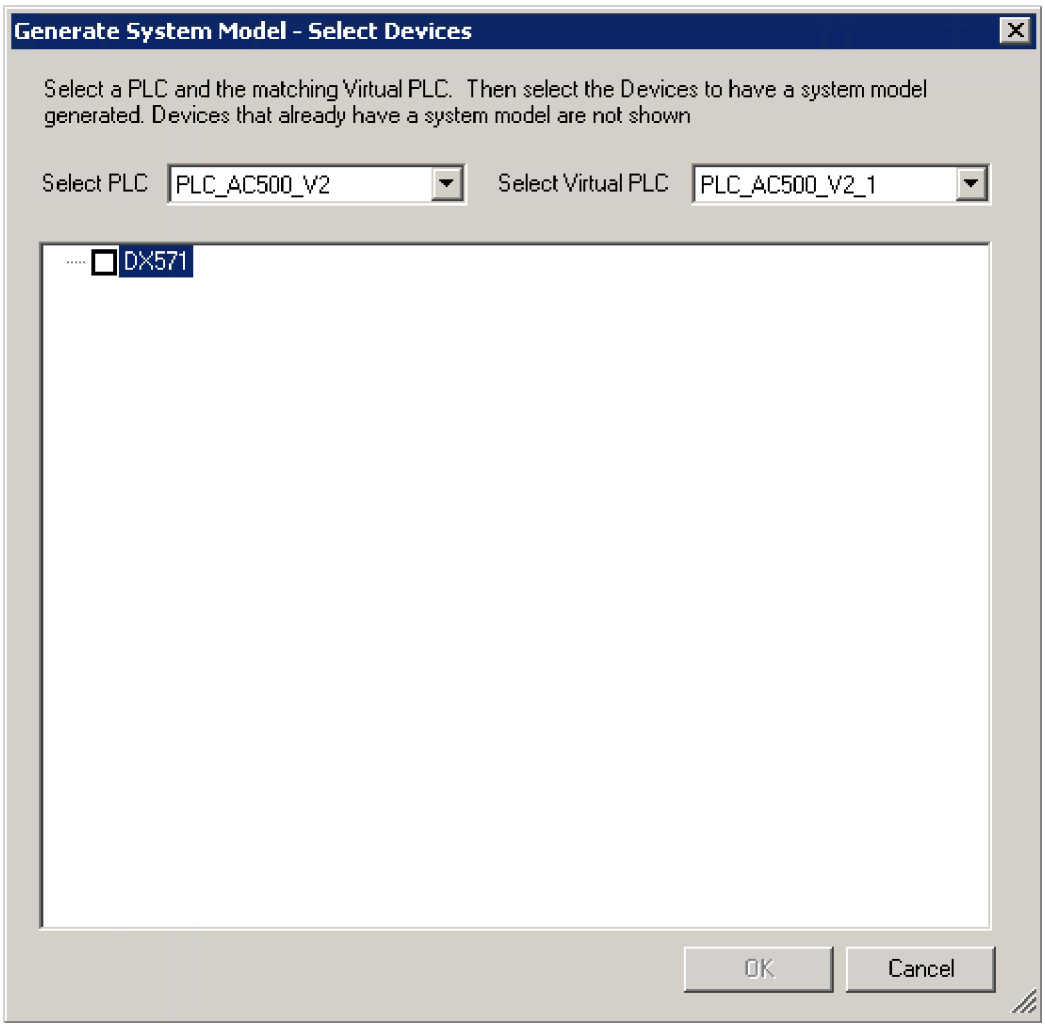
Generating system model for I/O

To generate a System Model for I/O modules, the user has to create an I/O module minimum of one variable mapping. The same type of device have to exist under the virtual PLC and have variable mappings with the same names and types but in opposite directions.

The command will generate a System Model for one or more selected I/O modules.

1. Right-click on System Model node and select *“System Modelling ➔ Generate System Model for I/O”*.

   ⇨ Generate System Model dialog shows the devices which are suitable for generating System Model.



2. From the drop-down list, select the real and virtual PLCs which displays the I/O modules that are associated with the real PLC.

   ⇨ The System Model for I/O mdoules are generated successfully.

Two Class and two Instance objects are generated for each device. One Real Class and Instance, and one Virtual Class and Instance. It is possible that the generation of a System Model can fail or generate warnings or errors. See sections ↳ *“Warnings” on page 6573* and ↳ *“Errors” on page 6573*.

The Class objects contain pin definitions for the variables which are in real and virtual devices. The Instance objects contain the information of pin mappings.

Update system model

Update System Model command updates all the existing System Models to reflect the current state of the devices.

1. Right-click on System Model node and select **System Modelling** -> **Update System Model** to update the System Models.
2. Confirm an upcoming message with *“OK”*.

   ⇨ Any manual changes made to the existing generated System Models are overwritten and they are updated to reflect the information contained within the devices.

Warnings

When a System Model generation is failed due to the selected device with no variable mappings, a warning 'Model not generated for device: No variable mappings found' is displayed.

The device should contain minimum of one variable mapping to generate a System Model.

Errors

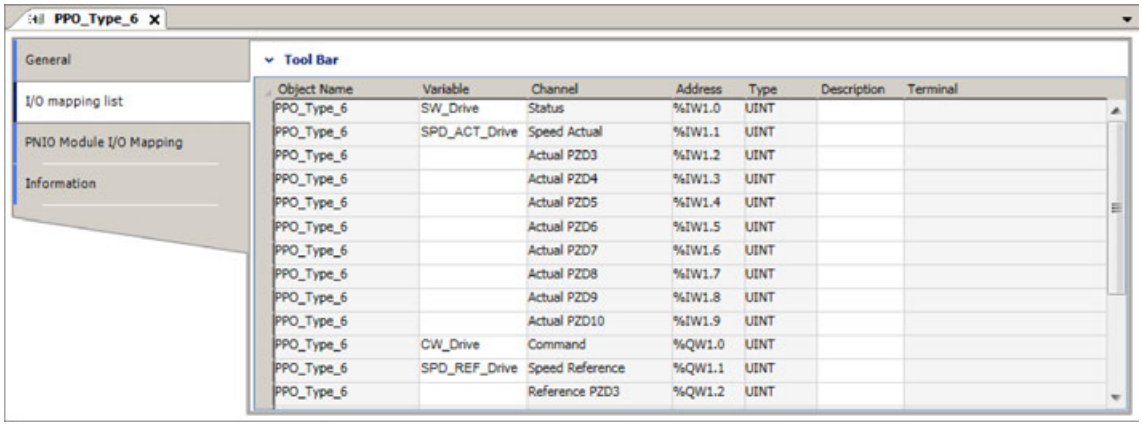
When a System Model is generated for a device which does not have the symbols exposed, an error 'The required symbols are not exposed. Please generate the symbol file for device x' is displayed.

The System Model can be generated without exposing the symbols. However, you cannot use in Virtual System Testing until the symbols of the device are exposed.

1.8.3.3 Example

In the example, the PLC will send Start, Stop (CW) and Speed Reference to the drive and read Actual speed and Drive Status (SW) using the fieldbus (PROFIBUS or PROFINET) device.

1. In the Automation Builder project, create a CODESYS application program and configure ACS380 Drive and its variables in the PPO Type editor.



2. Add a System Model under Automation Builder project.
3. Under System Model, add a model class and their instances for the drive and PLC (virtual devices).
4. Define input and output pins and their data types for the drive.

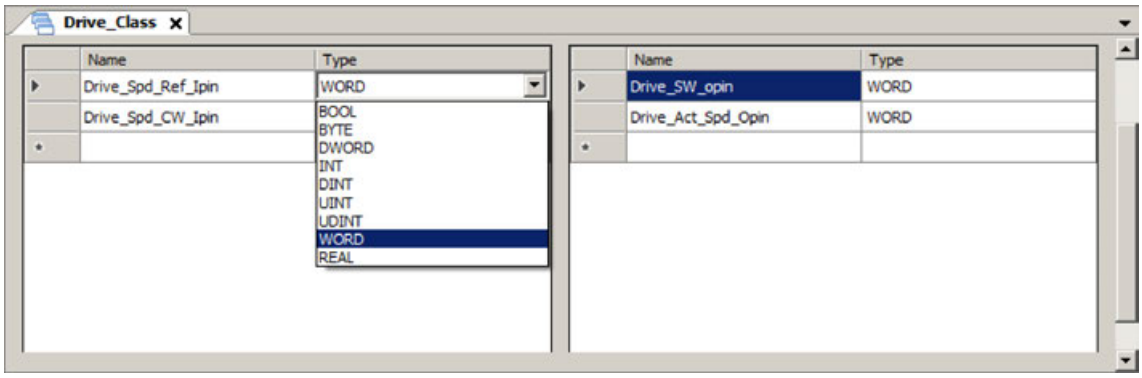


Fig. 1181: Drive Class Inputs and Outputs

⇨ Similarly, define input and output pins and their data types for the PLC.

5. In the Drive Instance editor, select a device (ACS380_PROFINET_IO) from the assigned device drop down which represents an instance.

> *The pins and their data types are defined in the class editor for a drive and PLC are displayed in the drive and PLC editors.*

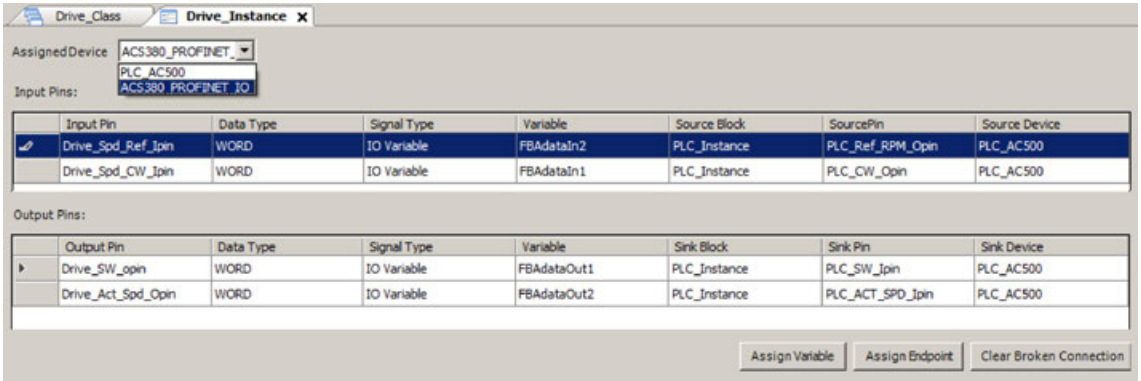


Fig. 1182: Assigned devices

6. Assign a variable using Inline editor or by Assign Variable button.
7. Assign Source Block and Source Pin for inputs and Sink Block and Sink Pin for outputs using the inline editor or by Assign Endpoint button.

> *If the connections between drive and PLC input and output pins are incorrect, click "Clear Broken Connection" to delete the broken connection pins.*

1.8.4 Drive composer pro integration

Drive Composer Pro is a start-up and maintenance tool for ABB's common architecture drives. The tool is used to view and set drive parameters, and to monitor and tune process performance.

Drive Composer Pro provides:

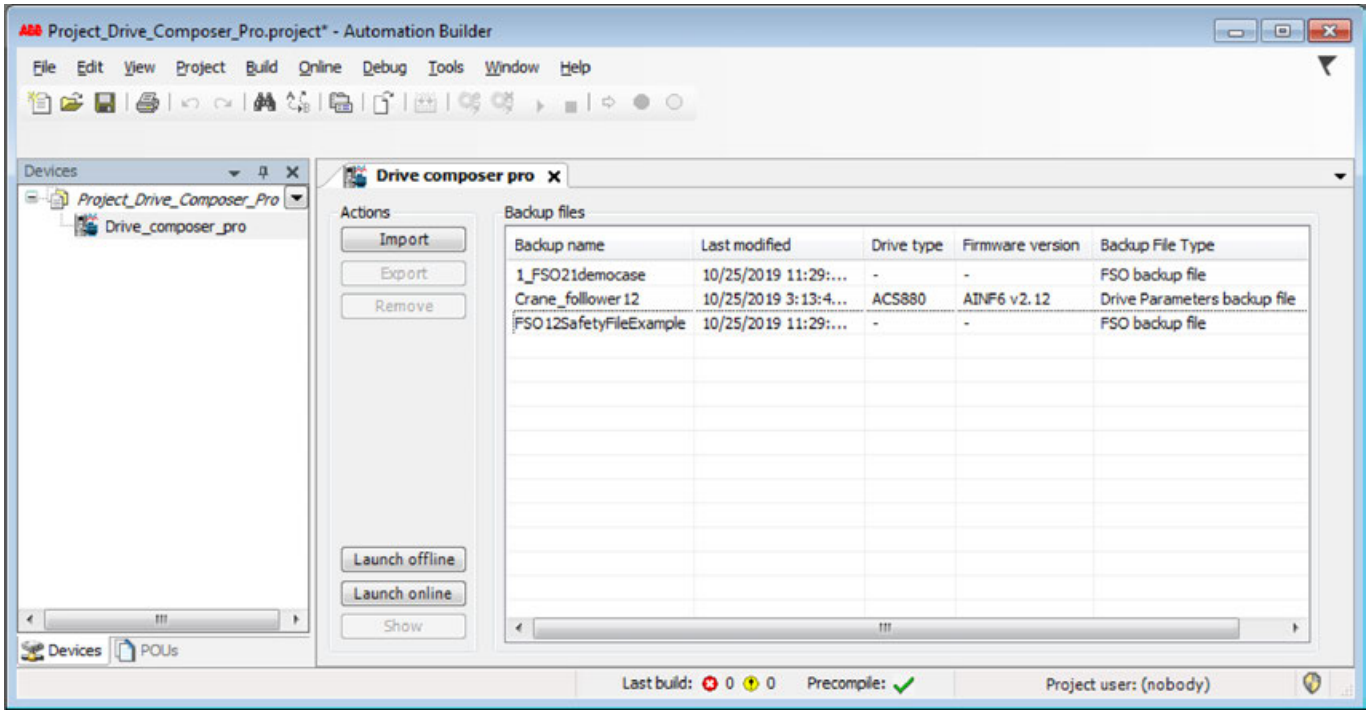
- Setting parameters,
- taking local control of the drive from the PC,
- event logger handling
- control diagrams,
- fast monitoring,
- working with multiple drives on the PC tool network,
- macro script editing for parameters and much more.

1. Add *"Drive Composer Pro"* object into the tree via add object dialog.
2. Open the *"Drive Composer Pro"* with double-click on the object.

In the following section important functions are described.

Import of backup files

1. Import of FSO backup files (*.dcsafety) and Drive Parameters backup files (*.dcparamsbak) into Automation Builder project via the Drive Composer Pro object in the device tree.
2. View of integrated FSO backup files and Drive Parameters backup files in Automation Builder project - refer to figure below.

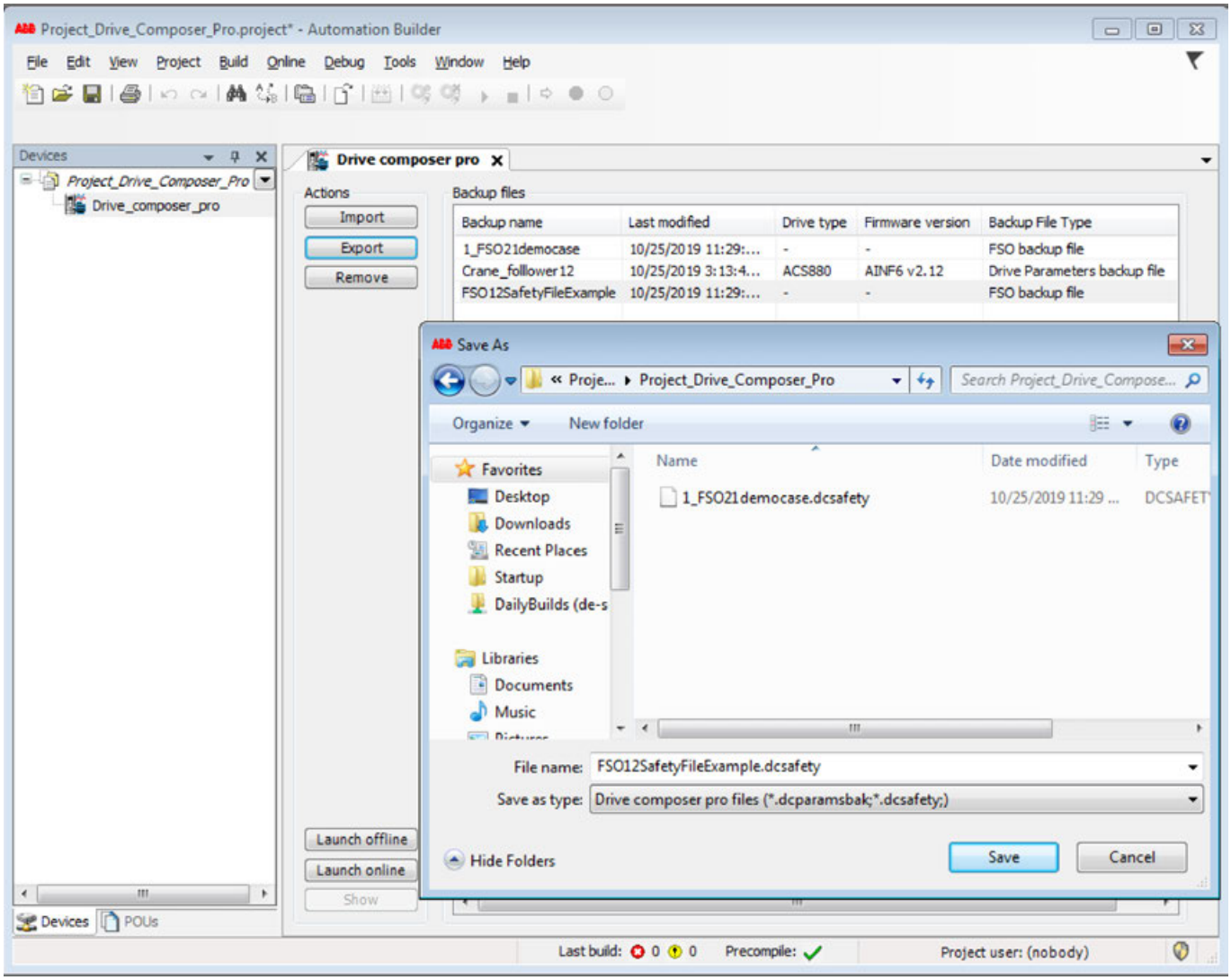


⇨

Drive Composer Pro can't be launched directly with integrated “FSO backup files” but they have to be loaded manually via context menu on the drive in Drive Composer Pro → “Safety Settings”.

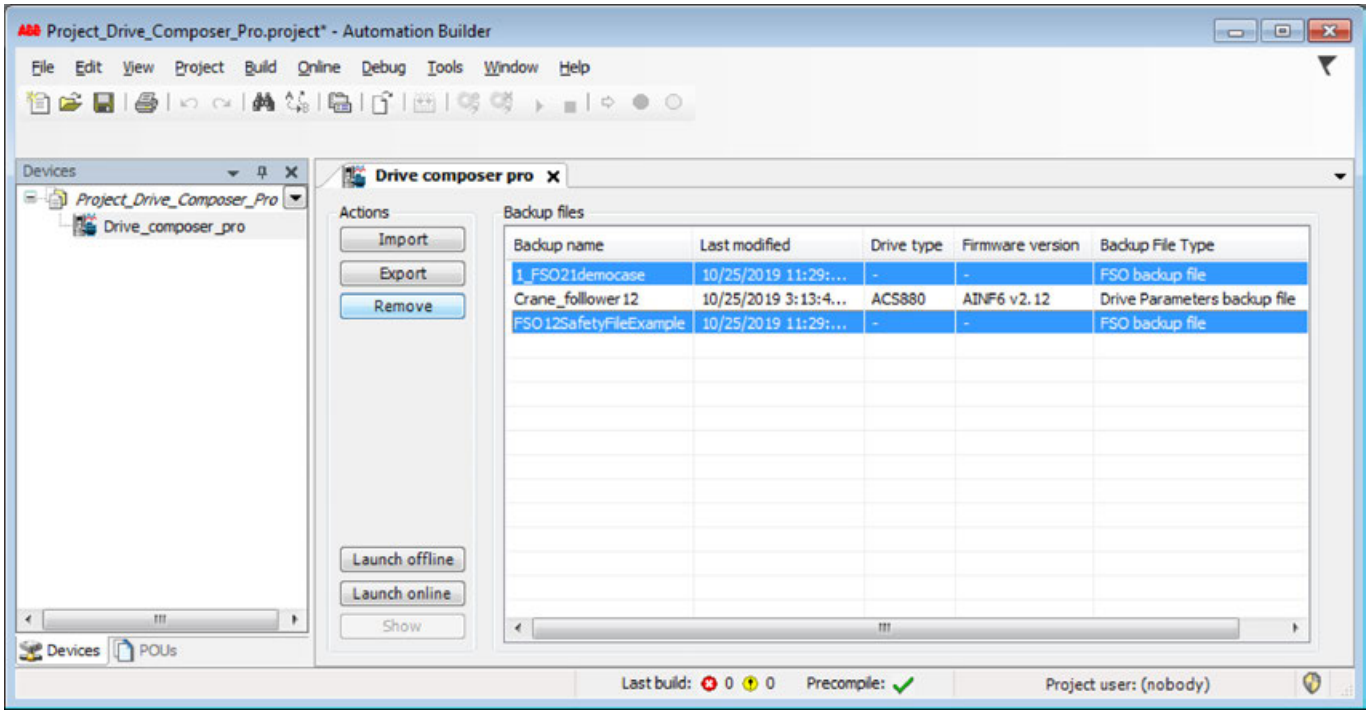
Export of backup files

1. Select the FSO and Drive Parameters backup files.
2. Export the selected file by clicking *[Export]*.
   ⇨ Select the desired storage path.



Remove of backup files

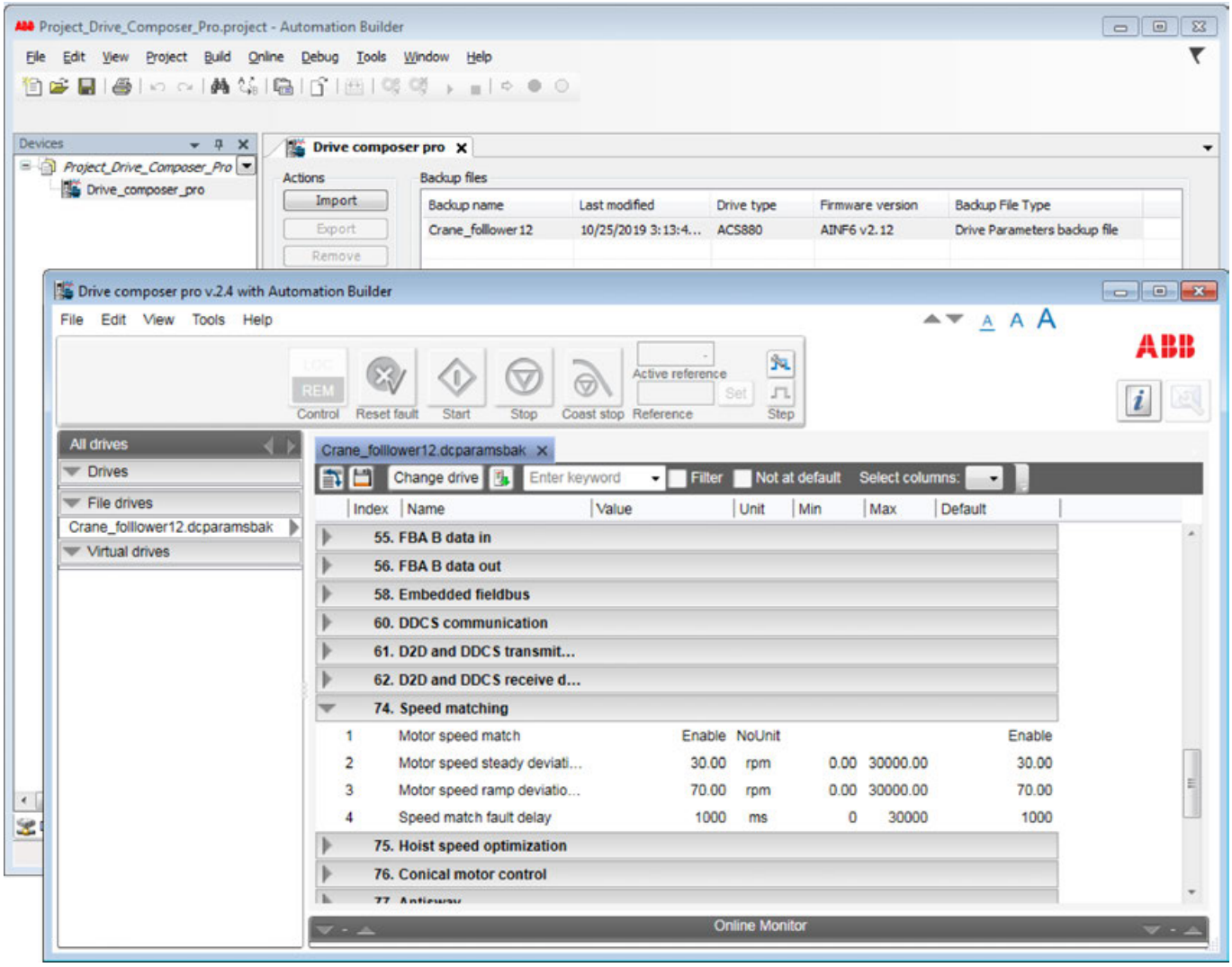
1. Select the FSO and Drive Parameter backup files from Automation Builder project.
2. Remove the selected files by clicking *[Remove]*.



View standard drive parameter backup files

1. Open the *“Crane_follower12.dcparamsbak”* with double-click.
2. The *“Drive Composer Pro”* starts automatically.

> *Standard Drive Parameter backup files (*.dcparamsbak) are automatically displayed under “File Drives”.*



3. Saved changes in the standard drive parameter backup file are automatically updated in the Automation Builder project.

1.8.5 Professional Version Control

SVN integration in CODESYS

Professional Version Control allows for the development of CODESYS projects under version control by Apache™ Subversion®. Professional Version Control provides an SVN client integrated in CODESYS. The objects of your project are versioned in a central *SVN repository*.

As a rule, the SVN repository should be created in a server configuration and located on a server. For testing purposes, you can create a local SVN repository where you can access via `file://`.

Professional Version Control requires a valid license and can be installed using the Automation Builder Installer or the Automation Builder Installation Manager.

1.8.5.1 Getting Started

The following steps are required in order to develop your CODESYS project with Professional Version Control with version control by Apache™ Subversion®:

1. Install the Professional Version Control package in CODESYS.
2. Install an SVN server.
3. Create an SVN repository.
4. Open your CODESYS project in CODESYS.
5. Import the CODESYS project into the SVN project archive.
   ⇨ The CODESYS project is saved in the SVN repository.
6. To edit and further develop the project with SVN version control, the project is edited in CODESYS and then committed to the SVN repository.

A detailed description of these individual steps is located in the following sections.

See also

- *Chapter 1.8.5.3 “Using an SVN Repository” on page 6579*
- *Chapter 1.8.5.4 “Using Working Copies” on page 6581*

1.8.5.2 Version control

What is version control?

Apache™ Subversion® (SVN) is a tool for version and revision management of current and previous versions of files, such as source code, websites, and documentation. Apache™ Subversion® is a registered trademark of the Apache Software Foundation.

Revision management (also known as version control, version management, and source code management) is the management of changes to documents, programs, and other information that is stored as computer files. Version control is employed frequently in software development when a team of employees works on the same files.

Tasks

- Co-writing of changes in revisions: At any time, you can show who made which changes at which time.
- Restoring of old revisions of individual files: At any time, you can reverse accidental changes to files.
- Archiving of special revisions of a project: At any time, you can revert to older versions.
- Coordination of common access of developers to data
- Development of a project simultaneously in multiple branches

Script Engine SVN Add-on API

Professional Version Control provides a scripting-interface for SVN.

1.8.5.3 Using an SVN Repository

An *SVN repository* usually saves information as a file system tree, a hierarchy of files, and directories. Any number of clients connects to the SVN repository and reads or writes changes to the files in revisions.

Creating an SVN repository

NOTICE!

Consult with your IT specialists for more information, for example how to create an SVN repository. For production purposes, we recommend a strictly dedicated administrative SVN server.

We recommend that you create the suggested default directory structure in the SVN repository.

See also

– *http://svnbook.red-bean.com/en/1.8/svn.tour.importing.html#svn.tour.importing.layout*

Creating an SVN repository for testing purposes

NOTICE!

Use the `file://` access method for testing purposes only.

You can reach SVN repositories that were created in format 1.8 or 1.9 via the `file://` protocol.

For testing purposes, you can create a local SVN repository without installing your own server. The SVN repository is accessed via `file://` and provides the same functionality as a server.

Creating a test repository with TortoiseSVN

☑ Requirement: The SVN client TortoiseSVN 1.9 is installed on the development system.

1. Create a new, empty folder on your local file system. The test repository will be created there.
   ⇨ Example: `D:\SVN repository`
2. Click *"TortoiseSVN ➔ Create repository here"*.
   ⇨ The dialog *"Create repository"* opens.
3. Click *"Create directory tree"*.
   ⇨ The `SVN repository` is created.

See also

- Documentation TortoiseSVN *Documentation TortoiseSVN*

Accessing the SVN repository

Table 799: SVN repository URLs

`file:///`	Direct access to an SVN repository (on local hard drive)
`http://`	Access via WebDAV protocol to Apache server that is supported by SVN
`https://`	As `http://`, but with SSL encryption
`svn://`	Access via own protocol to an `svnserve` server
`svn+ssh://`	As `svn://`, but tunneled via SSH

Import the project into the SVN repository.

1. Open the CODESYS project that you want to save in the SVN repository.
   ⇨ Example: `A.project` is open.
2. Click *"Project ➔ SVN ➔ Import project to SVN"*.
   ⇨ The *"Browse SVN repository"* dialog opens.
3. Select the directory `file:///D:/SVN repository/trunk` in the directory tree.
4. Select the command.
   ⇨ The *"Create remote directory"* dialog opens.
5. Specify the URL for the new directory.
   Note: Because the new directory should contain the CODESYS project, specify the project name with extension here.
   ⇨ `file:///D:/SVN%20repository/trunk/A.project`
6. Click *"OK"* to close the dialog.
7. Select the new project and click *"OK"* to exit the *"Browse SVN repository"* dialog.
   ⇨ The *"Import Project to SVN"* dialog opens. The directory `file:///D:/SVN repository/trunk/A.project` is specified in *"URL of SVN repository"*.

See also

- *Chapter 1.8.5.5.1 "Overlay Icons" on page 6582*

1.8.5.4 Using Working Copies

Checking out a project

You can copy CODESYS projects to your development system that are saved in the SVN repository.

Creating a working copy

1. Open CODESYS.
2. Click *"Project ➔ SVN ➔ Checkout"*.
   ⇨ The *"Checkout"* dialog opens.
3. Specify the URL of the SVN repository and select a project in the SVN repository tree.
   If a CODESYS project has the extension `.project` or `_project`, then it is recognized automatically as a project at checkout. If it has the extension `.library` or `_library`, then it is recognized as a library project.
4. In *"Checkout to"*, specify the name and location of the working copy on your development system.
5. Click *"OK"* to close the dialog.
   ⇨ The project opens in CODESYS. In the object tree of the project, the SVN link is shown with overlaid icons. Now the project is saved as a working copy on your development system.

See also

- *Chapter 1.8.5.5.1 "Overlay Icons" on page 6582*

Editing the working copy

Update the working copy before you start editing, especially if the project is revised by a team. This is how you avoid conflicts.

1. Open the working copy.
2. Click *"Project ➔ SVN ➔ Update project"* (symbol:).
   ⇨ You working copy is current.
3. Revise your project.
4. Click *"Project ➔ SVN ➔ Edit SVN working copy"*.
   ⇨ The dialog opens. There you can browse your changes.
5. Close the dialog.
6. If necessary, you can click *"SVN ➔ Revert"* in the context menu.
   ⇨ The file is reverted back to the base revision and your changes are discarded.
7. If necessary, you can click *"Compare"* in the context menu of an edited object.
   ⇨ The compare dialog opens. You can resolve any conflicts here.
8. Close the compare dialog.
9. Click *"Project ➔ SVN ➔ Commit project"* (symbol:).
   ⇨ The *"Commit"* dialog opens.
10. In *"Message"*, specify a log entry that describes your changes. Example: `Changes for customer ABC, request 1234.`
    ⇨ Your changes are saved in the SVN repository as a revision with a revision number.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*

Changed working copy format in Professional Version Control `V4.1.0.0` and later

For projects in version Professional Version Control `V4.1.0.0` and later, the working directory (working copy) has a new format.

If you open a project that was created with `V4.0.4.0` or earlier, then the project is updated automatically to the new format when it is opened.

If you open a project that was created with `V4.0.4.0` or earlier and the project is based on an older SVN version of `1.7.x` or earlier, then you are prompted whether or not CODESYS should update the format. If you decline the update, then the SVN link of the project is deactivated. You can still load and edit the project.

The update does not have an effect on saving to the SVN server. You can also checkout projects with earlier versions of the client. The new format affects only the local working directory.

See also

- *http://svnbook.red-bean.com/en/1.8/svn.ref.svn.c.upgrade.html*

1.8.5.5 Reference, User Interface

1.8.5.5.1 Overlay Icons

Every object in CODESYS has a status value in the SVN repository. This status value is displayed in the object tree (in the *"POUs"*, *"Devices"*, or *"Modules"* views) for each object by overlay icons.

Table 800: Overlay icons

	Object is planned to be added to the SVN repository.
	Object conflicted
	Object deleted
	Object modified
	Object with modification in the metadata
	Object with modifications in the memory format
	Object normal
	Object write-protected (read-only)
	Object locked
	Object with deleted subobjects
	Object ignored on commit
	External object
	Ignored object
	Unversioned object
	Object with modified subobjects
	The object is not saved in the SVN repository. It will be created again when loaded from SVN.
	`SVN_VERSION_INFO` temporarily unavailable, for example as with interface libraries
	The status of the object is not updated.
	The object was modified on the server (**U**pdate available).
	The object was locked on the server by another user (or in another working directory).
	Tree conflict by changes to the structure of the project

1.8.5.5.2 Commands

Not all commands are available in the logged in state because some SVN commands of the project could be changed.

Table 801: Availability of commands

Command	Not Logged In	Logged In
Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 6585	X	X
Chapter 1.8.5.5.2.2 “Command 'Edit SVN working copy'” on page 6586	X	
Chapter 1.8.5.5.2.3 “Command 'Import project to SVN'” on page 6589	X	X
Chapter 1.8.5.5.2.4 “Command 'Checkout'” on page 6589	X	X
Chapter 1.8.5.5.2.5 “Command 'Commit', Command 'Commit Project'” on page 6591	X	X
Chapter 1.8.5.5.2.6 “Command 'Compare'” on page 6594	X	
Chapter 1.8.5.5.2.7 “Command 'Compare with HEAD revision'” on page 6594	X	
Chapter 1.8.5.5.2.8 “Command 'Compare with revision'” on page 6594	X	
Chapter 1.8.5.5.2.9 “Command 'Compare to remote project...'” on page 6595	X	
Chapter 1.8.5.5.2.10 “Command 'Include externals to project', Command 'Include externals'” on page 6596	X	
Chapter 1.8.5.5.2.11 “Command 'Ignore on commit'” on page 6597	X	X
Chapter 1.8.5.5.2.12 “Command 'SVN Info'” on page 6598	X	X
Chapter 1.8.5.5.2.13 “Command 'Show properties'” on page 6598	X	X
Chapter 1.8.5.5.2.14 “Command 'Get lock'” on page 6599	X	X
Chapter 1.8.5.5.2.15 “Command 'Steal locks'” on page 6599	X	X
Chapter 1.8.5.5.2.16 “Command 'Release lock'” on page 6600	X	X
Chapter 1.8.5.5.2.17 “Command 'Release locks recursively'” on page 6600	X	X
Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 6600	X	X
Chapter 1.8.5.5.2.19 “Command 'Revert', Command 'Revert project'” on page 6602	X	
Chapter 1.8.5.5.2.20 “Command 'Revert to revision', Command 'Revert project to revision'” on page 6603	X	
Chapter 1.8.5.5.2.21 “Command 'Update', Command 'Update project' ” on page 6603	X	
Chapter 1.8.5.5.2.22 “Command 'Update to revision'” on page 6604	X	
Chapter 1.8.5.5.2.23 “Command 'Update only this'” on page 6605	X	

Command	Not Logged In	Logged In
Chapter 1.8.5.5.2.24 “Command 'Disconnect project from SVN'” on page 6605	X	X
Chapter 1.8.5.5.2.25 “Command 'Switch'” on page 6605	X	
Chapter 1.8.5.5.2.26 “Command 'Un-Ignore on commit'” on page 6606	X	X
Chapter 1.8.5.5.2.27 “Command 'SVN Cleanup'” on page 6606	X	X
Chapter 1.8.5.5.2.28 “Command 'Clear authentication data' ” on page 6607	X	X
Chapter 1.8.5.5.2.29 “Command 'Merge changes'” on page 6607	X	
Chapter 1.8.5.5.2.30 “Command 'Connect to existing project'” on page 6608	X	X
Chapter 1.8.5.5.2.31 “Command 'Resolve conflict' ” on page 6609	X	
Chapter 1.8.5.5.2.32 “Command 'Work in offline mode'” on page 6609	X	X
Chapter 1.8.5.5.2.33 “Command 'Copy (Branch/Tag)'” on page 6610	X	
Chapter 1.8.5.5.2.34 “Command 'Pending Changes'” on page 6611	X	X

Command 'SVN Repository Browser'

Symbol: 

Function: This command opens the SVN repository browser. The contents of an SVN repository is shown in a tree structure here. You can search through the repository in the browser.

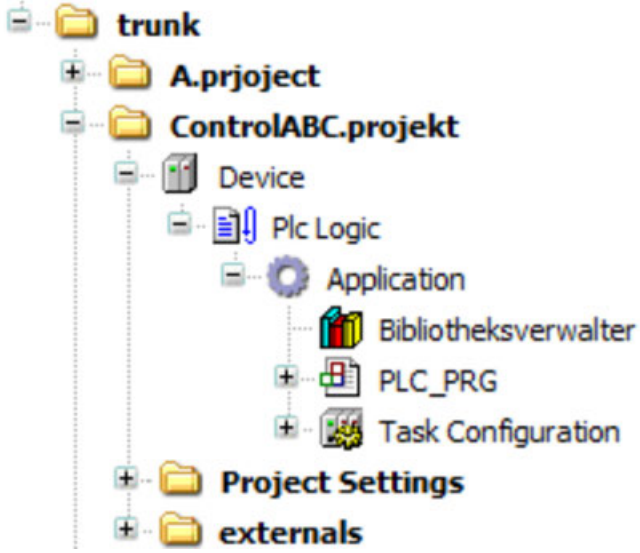
Call: Menu bar: *“Project ➔ SVN”*.

Depending on the selected object, the following commands are available in the context menu:

- *“Show log”*
- *“Checkout”*
- *“Create folder”*
- *“Copy to”*
- *“Rename”*
- *“Delete”*

Double-clicking the object with the right mouse button opens the log dialog.

Dialog 'SVN Repository Browser'

"URL"	URL in SVN repository Example: `https://svnserver/repository/trunk/ControlABC.project` Tip: As soon as a valid SVN repository is specified, you can browse and select a specific project by means of the adjacent button.
15	Opens the dialog *"Select revision"*. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"3"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
	Updates the browser view by rescanning the SVN repository.
	Navigates the URL address up by one folder.
Left area	Directory tree in the SVN repository. Project nodes are shown in bold.  Note: In this view, you can directly edit the project name and the name of the superordinate folder.
Right area	List of objects of the selected directory
"Close"	Closes the dialog

See also

- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 6614*

Command 'Edit SVN working copy'

Symbol:

Function: This command opens the dialog *"Edit SVN working copy"* and displays the working copy in a browser from the SVN view.

Call: Menu bar: *"Project ➔ SVN"*.

The functionality of the browser allows for:

- Access to and actions on objects that are not displayed in the *"Devices"* view.
- Actions on objects that can lead to exceptions in the *"Devices"* view.
- Editing of global objects that are modified, in conflict, or blocked.

Dialog 'Edit SVN working copy'

Table 802: "Edit SVN working copy: <project name> - <project URL>"

"Path in SVN repository"	Display of working copy from SVN view. The file and folder structure of the objects in the project are presented in a tree view. In this way, the recursion depth of an object is clear. : Object selected for the following menu command
"Name of object"	File name of the object Example: `Application`
" Node type"	The top node is the project root directory.
"Text status"	Object status: ● *"modified"* ● *"added"* ● *"deleted"* ● *"non-versioned"* ● *"Conflicted"*
"Property status"	Status in SVN repository: ● *"modified"* ● *"added"* ● *"deleted"* ● *"Conflicted"* ● *"normal"*
"Revision"	Revision number
"Conflict information"	File conflict, property conflict, or tree conflict
"Lock"	For locked objects, the user who applied the lock is displayed. Example: `b.mayer`
"Lock comment"	Lock message. Implicit, normal, or stolen lock.
"URL"	URL of the object

Table 803: Menu commands

"Select ➔ All"	Selects all files.
"Select ➔ None"	Deselects all files.
"Select ➔ Modified"	Selects the modified files.
"Select ➔ Conflicted"	Selects the conflicted files.
"Select ➔ Locked"	Selects the locked files.
	Updates the working copy. Changes made by others are added from the SVN repository to your working copy.
"Update ➔ Project"	Updates all files of the project.
"Update ➔ Selected nodes"	Updates only the selected files.
"Update ➔ Selected nodes and children"	Updates the selected files and subordinate files.
"Reset"	Discards your changes to the working copy. Then the object corresponds to the revision in the repository.

"Delete ➔ Selected nodes"	Deletes the selected objects from the working copy.

"Commit"	Commits your changes to the SVN repository. Any locked objects will be unlocked.
"Commit ➔ Project"	Commits all files in the project.
"Commit ➔ Selected nodes"	Commits only the selected files.
"Commit ➔ Selected nodes and children"	Commits the selected files and subordinate files.

	Commands for managing locks.
"Locks ➔ Revalidate all"	Checks the validity of locks in the working copy. Any invalid locks will be unlocked.
"Locks ➔ Release locks"	Releases the lock.
"Locks ➔ Acquire locks"	Locks the object from editing by others.
"Locks ➔ Steal locks"	Locks the file for you and removes the lock of another user. Tip: Avoid stealing a lock because the changes made by another user can be lost.

	Commands to resolve conflicts.
"Conflicts ➔ Mark as resolved"	Indicates a displayed conflict in the SVN repository as marked and resolved. Note: Select the command if you edited and resolved the displayed conflict. Then you can commit changes again.
"Conflicts ➔ Resolve using theirs"	Resolves the conflict: In the SVN repository, the changes are accepted that were committed by other users. Your changes are discarded.
"Conflicts ➔ Resolve using mine"	Resolves the conflict: In the SVN repository, the changes to your working copy are accepted and the changes by other users are discarded.

"Show log"	Opens the dialog *"Log - Application"*. The history of the selected node is shown here. The previous revisions are displayed with the respective actions.

"Change location"	Changes the storage location of the selected object within the working copy. Example: You can resolve a tree conflict by saving the local object to another location. Then update the parent object to apply it to the locked children.

"Update"	Updates the browser view by rescanning the working copy.

"Cleanup"	Executes an SVN cleanup operation on the working copy.

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 6603*
- *Chapter 1.8.5.5.2.19 "Command 'Revert', Command 'Revert project'" on page 6602*
- *Chapter 1.8.5.5.2.5 "Command 'Commit', Command 'Commit Project'" on page 6591*
- *Chapter 1.8.5.5.2.31 "Command 'Resolve conflict' " on page 6609*
- *Chapter 1.8.5.5.2.14 "Command 'Get lock'" on page 6599*
- *Chapter 1.8.5.5.2.16 "Command 'Release lock'" on page 6600*
- *Chapter 1.8.5.5.2.15 "Command 'Steal locks'" on page 6599*
- *Chapter 1.8.5.5.2.25 "Command 'Switch'" on page 6605*
- *Chapter 1.8.5.5.2.18 "Command 'Show log', Command 'Show project log'" on page 6600*
- *Chapter 1.8.5.5.2.27 "Command 'SVN Cleanup'" on page 6606*

Command 'Import project to SVN'

Symbol: 

Function: This command opens the *"Import Project to SVN"* dialog for importing a CODESYS project to the SVN repository.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement

- You have access to an SVN repository and you know its URL.
- You have read access to the entire project.

> **NOTICE!**
> Projects are always saved unencrypted on the server. Therefore, take appropriate security measures (for example, respective access rights to the SVN server) for protecting your projects.

See also

- User and access management in *Protect and save project*

Dialog 'Import Project to SVN'

"URL of SVN repository"	URL of the SVN repository with the new project folder where the files are imported Example: `https://svnserver/repository/trunk/ControlABC.project` Hint: When importing libraries, specify the extension `.library` or `_library`. For projects, specify the extension `.project` or `_project`. Then the project type is recognized automatically at checkout and the options are set accordingly in the *"Checkout"* dialog.
	Opens the *"SVN Repository Browser"* dialog The previous project structure is displayed and you can edit them here.
"Import message"	Text for use as log message Example: `Control project for customer A`
"Recent messages"	Opens the *"Recent Messages"* dialog. There you can reuse the last log messages.
"Generate SVN_VERSION_INFO"	☐: The object `SVN_VERSION_INFO` is not created automatically during the import operation. Therefore, the project does not get any global constants or variables for the project metadata.
"OK"	Creates the current project in the SVN repository and imports the project objects. The local project in CODESYS Development System is linked to the SVN repository. Overlay icons show this in the object trees.

See also

- *Chapter 1.8.5.5.2.4 "Command 'Checkout'" on page 6589*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*
- *Chapter 1.8.5.5.1 "Overlay Icons" on page 6582*

Command 'Checkout'

Symbol: 

Function: This command opens the *"Checkout"* dialog. Here you can checkout a project stored in the SVN repository as a working copy.

Call: Menu bar: *"Project ➔ SVN"*.

Dialog 'Check-out'

Table 804: "URL of SVN repository"

https://svnserver/repository/trunk/ControlABC.project	URL of the project in the SVN repository Example: `https://svnserver/repository/trunk/ControlABC.project` Tip: As soon as a valid SVN repository is specified, you can click the adjacent button or use the options to browse in *"Revision"* and select a specific project.
15	Opens the dialog *"Select revision"*. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
...	Opens the *"SVN repository browser"* dialog Here you can browse the SVN repository.

Table 805: "Checkout to"

"Name"	Name of the working copy Example: `ControlABC.project`
"Location"	Storage location of the working copy Example: `/D:/svn/repository/trunk/ControlABC.project`

Table 806: "Checkout as"

"Project"	The project is saved as a CODESYS project "`<project name>.project`".
"Library"	The project is saved as a CODESYS library file "`<project name>.library`".
"Auto-detect"	CODESYS attempts to recognize the project type by means of the extension. The current implementation checks whether the URL of the project ends with "`_library`" or "`.library`". In this case, the project is recognized as a library or a project.

Table 807: "Checkout options"

"Omit externals":	☑: Externals (external objects) are not copied to the working directory.

Table 808: "Revision"

For a description, refer to the section "Dialog 'Select revision'". Note: The group provides the same options as the *"Revision"* dialog.

"OK"	Checks out the project from the SVN repository, saves it locally to the specified location, and opens it in CODESYS as the primary project.

If files were encrypted when imported to the SVN repository, or if they have been committed, then note the following:

When committing to the SVN repository, the information about an encrypted project file is included. However, the type of encryption is not included (password, Wibu security key, X509 certificate). Therefore, it may be necessary to encrypt the working copy again in the project settings. In this case, a dialog opens when exiting the command to notify you of this. Then you are able to switch directly to the project settings.

See also

- *Chapter 1.8.5.5.3.3 “Dialog 'Select revision'” on page 6614*
- *Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 6585*
- *"Version control with Subversion", Section "Revision identifier"*

Command 'Commit', Command 'Commit Project'

Symbol:

Function: The command commits changes that were made in CODESYS to the SVN repository. The *“Commit”* dialog opens for this purpose.

Call:

- Context menu: *“SVN”* to commit exactly this object
- *“Project ➔ SVN ➔ Commit Project”* to commit all changes in the project at the same time

Requirement: At least one object was modified. An object whose contents have been modified is overlaid in the object tree with the , , or symbol.

When you execute the command, the lock on the objects to be committed is lifted automatically.

See also

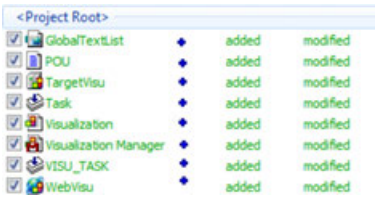
- *Chapter 1.8.5.5.1 “Overlay Icons” on page 6582*

Dialog 'Commit'

Table 809: “Commit to: <URL project/object>”

	URL in SVN repository Example: `file:///D:/SVN repository/trunk/ControlABC.project`
“Log message”	Type in a log message that comments your change. Example: `Bug fix error 123`
“Recent Messages”	Opens the *“Recent Messages”* dialog for displaying the last log messages. You can click a log message to accept it.

Table 810: "Changes made (double-click on object for compare, right-click on object for more operations)"

	List of objects that were changed and can therefore be committed. The SVN URLs mirror the hierarchy of the object in the SVN repository. The objects are highlighted in color according to the object status: ● Blue: Modified ● Green: Added ● Dark red: Deleted ● Red: Conflicted ● Black: Non-versioned (not in SVN repository) Note: These objects are displayed when the *"Show non-versioned objects"* option is selected. ● Gray: Excluded from commit Note: This is the case when the *"Ignore during commit"* option is selected. The list also contains objects which have not been modified but have a lock. This helps to prevent locking from going unnoticed in the repository. Double-click an object in order to open the compare dialog. The revision of the working copy is compared with the base revision. The compare dialog also opens when you click *"Compare"* in the context menu. Right-click an object in order to open the context menu. Note: When the *"Commit Project"* command has been executed, a list of objects is shown here. When the *"Commit"* command is applied to a specific object, only this object is shown (if modified or locked) and its modified or locked child objects.
"Object"	☑: The object is selected for the commit. Example: ☑ Device\Plc Logic\Application\PLC_PRG
"Text status"	Object status in CODESYS ● *"Modified"* ● *"Added"* ● *"Deleted"* ● *"Non-versioned"* ● *"Conflicted"*
"Property status"	Status of the metadata of the object ● *"Modified"* ● *"Added"* ● *"Deleted"* ● *"Conflicted"* ● *"Normal"*
"Lock"	If the object has a lock, then it is shown here the user who applied the lock. Example: `b.mayer`
"Description"	Display of the log message
"Select/Deselect All"	☑: All objects in the list are selected.

"Keep Locks"	☑: Your locked object remains in locked after the commit.
"Keep Change Lists":	☑: The change list also remains after the commit. ☐: The change list is not deleted after the commit.
"Update After Commit (recommended)"	☑: The object/project is updated after the commit. Select this check box to ensure that the project is up-to-date and to prevent conflicts resulting from mixed revisions of working copies.

Button *"Update Project"*	Updates the project Hint: Prevent conflicts by committing a previously updated project/object.
"OK" Keyboard shortcut *[Ctrl]+[Enter]* Keyboard shortcut *[Ctrl]+[Enter]*	Checks the working copy first. Starts the commit of changes when the working copy is current. Opens a dialog when the working copy is outdated. You can then select from the following: ● *"Abort the commit, I want to investigate the issue."* ● *"Yes, I want to update this project now."* ● *"Continue with the commit, I know what I do."* Note: The history of the commit is displayed in the *"Messages"* view. The messages are highlighted in color. ● Blue: Commit a change ● Green: Add an object ● Dark red: Delete/replace an object ● Black: Other messages (summary)

Handling external objects

If the external object is in the same SVN repository, then changes in this external object are listed in the commit dialog and committed together with the internal project. If an external object is in another SVN repository, then you are notified about changes in the external project and you have to commit these separately.

An external object has the "externals" property.

See also

- *Chapter 1.8.5.5.2.6 "Command 'Compare'" on page 6594*
- SVN help: *http://svnbook.red-bean.com/en/1.7/svn.basic.in-action.html#svn.basic.in-action.mixedrevs)*

Context menu (right-click on object)

"Compare"	Opens the compare dialog to compare the working copy with the top-level revision.
"Compare with HEAD version"	Opens the compare dialog to compare the working copy with the HEAD revision.
"Compare with Revision"	The list entries are highlighted in color according to the object status: ● Blue: Modified ● Green: Added ● Dark red: Deleted ● Red: Conflicted ● Black: Non-versioned (not in SVN repository) Note: These objects are displayed when the *"Show non-versioned objects"* option is selected. ● Gray: Excluded from commit Note: This is the case when the *"Ignore during commit"* option is selected for the object.
"Revert"	Discards your changes to the working copy. Then the object corresponds to the revision in the SVN repository.
"Show log"	Shows the version history of the selected object.

"Properties"	Opens the *"SVN Properties"* dialog. The properties are displayed there and you can edit them.
Move to change list	Note: This command has not been implemented yet.

Command 'Compare'

Symbol: 🔍

Function: This command opens a tab that shows the result of the comparison of your working copy and the BASE revision. The base revision is the top-level revision in the SVN repository.

Call:

- Menu bar: *"Project ➔ SVN"*.
- Context menu

Requirement: The object is versioned, it was modified locally, and it does not contain any conflicts.

Multiple tabs can be open at the same time with the comparison of different objects.

Comparison by object type

The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.

See also

- *Chapter 1.6.5.1.1.6 "Comparing projects" on page 5765*

Command 'Compare with HEAD revision'

Symbol: 🔍

Function: This command opens a tab that shows the result of the comparison of your working copy and the HEAD revision. The HEAD revision is the top-level revision in the branch. You can revert specific changes that were committed to the HEAD revision.

Call: Context menu: *"SVN"*

Requirement: The object is versioned and not conflicted.

Multiple tabs can be open at the same time with the comparison of different objects.

Comparison by object type

The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.

Command 'Compare with revision'

Symbol: 🔍

Function: This command opens the *"Project log"* dialog or *"Log - <object> "* where the version history is displayed from the project or an object of the CODESYS project. Here you can select a revision. A tab opens and shows the result of the comparison of your working copy and the revision.

Call: Context menu: *"SVN"*

Requirement: The object is versioned and not conflicted.

Multiple tabs can be open at the same time with the comparison of different objects.

> ***Comparison by object type***
> *The comparison dialog makes use of the functionality of the CODESYS command "Project ➔ Compare". In this way, objects are compared according to their object type.*

See also

- *"Tab 'Project log', Dialog 'Log - <object>'" on page 6600*

Command 'Compare to remote project...'

Symbol:

Function: This command opens the dialog *"Select Remote Project for Comparison"*.

Call: Menu bar: *"Project ➔ SVN"*.

Dialog 'Select Remote Project for Comparison'

Table 811: "URL of SVN repository"

https://svnserver/repository/trunk/ControlABC.project	URL of the project in the SVN repository that is compared. Example: `file:///D:/SVN repository/trunk/ControlDEF.project` As soon as a valid SVN repository is specified, you can click the adjacent button or use the options to browse in *"Revision"* and select a project.
15	The label on the button corresponds to the selected revision: ● *"HEAD"*: Top revision (latest). ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) After clicking the button, the dialog *"Select revision"* opens. Note: The dialog provides the same options as the *"Revision"* group.
...	Opens the dialog *"Browse SVN repository"* to search the SVN repository.

Table 812: "Checkout options"

"Omit externals":	☑: External objects are not compared.

Table 813: "Revision"

Options for selecting a specific revision Note: the current valid selection is also displayed next to the SVN repository URL.	
"HEAD"	◉: The HEAD revision is selected. This is the latest revision (top revision) within a branch.
"Revision"	◉: A specific revision is selected by the revision number. Example: `3`

"Date"	: The specific revision is selected by the modification date. Example: `12/23/2016 11:59:59`
"Use UTC Time":	: Modification date in universal time.

Table 814: "compare options"

"Ignore Whitespace"	: No comparison of whitespace characters. Semantically relevant whitespaces, such as in strings, are compared anyway.
"Ignore Comments"	: No comparison of comments.
"Ignore Properties"	: No comparison of properties. Folders, the property *"Exclude from build"*, and POU images are not compared. See: Dialog 'Properties'

"OK"	Compares the SVN project with the working copy.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*

Command 'Include externals to project', Command 'Include externals'

Symbol:

Function: These commands open the dialog *"Include externals"*.

Call:

- Menu bar: *"Project ➔ SVN"*.
- Context menu: *"SVN"*

Requirement: An object is selected in the object tree. The external objects are linked below that. If you have selected nothing or the project root directory, then the command *"Include externals to project"* is available. If you have selected an object, then the command *"Include externals"* is available.

The same external objects cannot be linked multiple times at different locations in the same project. This leads to problems in CODESYS because of conflicts with the internal identification of the object.

Dialog 'Include externals'

Table 815: "URL of SVN repository"

file:///D:/SVN repo A/trunk/DSTest.project/Source	URL of the external object that is linked. The object to be linked is versioned and can have sub-objects. External objects are located at another location in the SVN repository than the project. It can even be in another SVN repository. Example: `file:///D:/SVN repo A/trunk/DSTest.project/GlobalTextList` Note: The objects that should be linked below the selected object must have a matching object type. For example, only a task can be linked below the *"Task configuration"* object.

15	Opens the dialog *"Select revision"*. Here you can select a revision. The button is labeled with the currently selected revision: ● *"HEAD"*: Top revision (latest). Preset ● *"15"*: Revision number of the selected revision ● *"23.12.2016 11:59:59 (UTC)"*: Change date of the selected revision (UTC) Note: The dialog provides the same options as the *"Revision"* group.
...	Opens the *"SVN repository browser"* dialog Here you can browse the SVN repository.

Table 816: "Revision"

Options for selecting a revision Note: the current valid selection is also displayed in the buttons next to the SVN repository URL.	
"HEAD"	◉: Latest revision (top revision) selected in a branch.
"Revision"	◉: A specific revision by the revision number. Example: `3`
"Date"	◉: A specific revision by the modification date. Example: `12/23/2016 11:59:59`
"Use UTC Time":	☑: Modification date in universal time.

"OK"	Adds the external object and its sub-objects with the property `svn:externals` to your project (below the selected object). The working copy is updated and the external object is overlaid with the ↗ symbol. Example: Source (external device `Source`) Note: If the linking fails (for example when adding a device below a task configuration), then the complete operation fails and reverts back. Note: Renaming or moving individual external objects is permitted inly within an external tree, whereby it is not permitted to move the top object. To move a complete tree, you have to remove it and link it to another location.

"... You should seriously consider using explicit revision numbers in all of your externals definitions. Doing so means that you get to decide when to pull down a different snapshot of external information, and exactly which snapshot to pull. Besides avoiding the surprise of getting changes to third-party repositories that you might not have any control over, using explicit revision numbers also means that as you backdate your working copy to a previous revision, your externals definitions will also revert to the way they looked in that previous revision, which in turn means that the external working copies will be updated to match the way they looked back when your repository was at that previous revision. For software projects, this could be the difference between a successful and a failed build of an older snapshot of your complex codebase. ..."

This is a quote from:

http://svnbook.red-bean.com/nightly/en/svn.advanced.externals.html).

Command 'Ignore on commit'

Function: This command identifies an object and adds it to the "ignore-on-commit" list. Then it is deactivated in the commit dialog by default.

Call: Menu bar: *“SVN”*

Requirement: At least one object is available that is not in the change list `ignore-on-comment`.

Objects of the "ignore-on-commit” list are overlaid with the symbol in the object tree. By default, they are not selected in the commit dialog, unless a dependency of a selected object requires it. These objects can always be selected manually in the dialog.

See also

- *Chapter 1.8.5.5.2.26 “Command 'Un-Ignore on commit'” on page 6606*

Command 'SVN Info'

Function: This command provides information about the selected object in the SVN repository. The *“SVN Information”* dialog opens for this purpose.

Call: Context menu: *“SVN”*

Requirement: A versioned object (with SVN link) is selected in the object tree.

Dialog 'SVN Information'

Example

```
Name: Device_4\Plc Logic\Application\PLC_PRG
URL: file:///D:/SVN repository/trunk/ControlABC.project/Device/Plc
Logic/Application/PLC_PRG/svnobj
Repository Root: file:///D:/SVN repository/
Repository UUID: 185325d7-73eb-e54b-ab50-206aa23c8b42
Revision: 29
Node Kind: File
Schedule: Normal
Last Changed Author: a.mayer
Last Changed Rev: 8
Last Changed Date: 17.01.2017 12:33:51
Text Last Updated: 17.01.2017 12:33:51
Checksum: d5fb4d91ebaea06f26bcdb15942724d57932b6a3
```

Command 'Show properties'

Symbol:

Function: This command opens the *“SVN Properties”* dialog. Here you can edit the properties of the versioned object.

Call: Context menu: *“SVN”*

Requirement: A versioned, unlocked object is selected.

Dialog 'SVN Properties'

Table 817: "properties for: <object name>"

"Name"	Name of the property Example: `myprop:customer-number` Note: SVN has some reserved properties. Example: `svn:mime-type`
"Value"	Example: `1234` Double-click in the field to edit the value.
"Add"	Opens a dialog to define another property with its value.
"Remove"	Deletes the selected property.
"Show binary properties"	☑: The binary properties are also displayed.
"Reset"	Resets the changes displayed in green.
"OK"	Accepts the changes.

See also

- *http://svnbook.red-bean.com*.

Command 'Get lock'

Symbol:

Function: This command locks the object explicitly for you. The *"Lock Message"* dialog opens for this purpose.

Call: Context menu: *"SVN"*

Requirement: The versioned object is not locked (not overlaid with the symbol).

Dialog 'Lock Message'

"Enter the reason why you lock the object:"	Lock message Example: `Locked for processing task 123`
Button *"Recent Message"*	Shows message in the dialog that have already been used. There you select one in order to use the lock message.
"Recursive"	☑: The object is locked with all subordinate child objects.
"OK"	Locks the object When the lock is successful, the object (in the object tree) is overlaid with the symbol.

Command 'Steal locks'

Symbol:

Function: This command steal the lock of the object. The *"Lock Message"* dialog opens for this purpose.

Call: Context menu: *"SVN"*

Requirement: The versioned object is locked by someone else (overlaid with the symbol).

Dialog 'Lock Message'

"Enter the reason why you lock the object:"	Lock message Example: `a.mayer had to steal the lock because the changes need to be implemented so urgently.`
"Recent Message"	Shows message in the dialog that have already been used. There you select one in order to use the lock message.
"Recursive"	☑: The lock is stolen by the object and all subordinate child objects.
"OK"	Steals the lock. When the stolen lock is successful, the object (in the object tree) is overlaid with the symbol.

Command 'Release lock'

Symbol:

Function: This command releases the lock of an object.

Call: *"Context menu ➔ SVN"*

Requirement: The object is locked.

Command 'Release locks recursively'

Symbol:

Function: This command releases the lock of an object explicitly with all of its subordinate objects.

Call: *"Context menu ➔ SVN"*

Requirement: The object is locked.

Command 'Show log', Command 'Show project log'

Symbol:

Function: These commands open the tab *"Project log"* or *"Log - <object>"*. The version history of the project or an object of the CODESYS project is displayed in the tab.

Call:

- Menu bar: *"Project ➔ SVN"*.
- *"Context menu ➔ SVN"*

If you select nothing or the base node in the object tree, then the history of the entire project is displayed (*"Show project log"*). If you select one or more objects, then the history of these elements is displayed (*"Show log"*).

Multiple tabs can be open at the same time with the version history of different objects.

Tab 'Project log', Dialog 'Log - <object>'

Upper area ● *“Revision”*: Revision number ● *“Author”* ● *“Date”* ● *“Message”*: Message entered at commit	List of all revisions of the project or the selected objects in the information. The first 100 revisions are displayed by default. The *“Next 100”* and *“All”* buttons are provided for displaying more or all revisions. Several commands are available in the context menu of each revision. These context menu commands are described below.
Middle area	Display of the *“Message”* of the revision that is selected in the upper area.
Lower area ● *“Action”* ● *“Path”*: Object path in SVN ● *“Copy from path”* ● *“Copy from revision”*	List of actions that were performed on the objects of the project in the selected revision:
“Hide unrelated changed paths”	☑: All changes of this revision are hidden that do not have any relevance to the object.
“Stop on copy/rename”	☑: If the object was copied from another location in the SVN repository, then no more log messages are retrieved. This is especially beneficial when branches or tags are monitored and only changes within the branch are relevant.
“Filter/Range”	Opens the *“Filter”* dialog
“All”	All revisions are listed.
“Next 100”	The next 100 revisions are listed.

Table 818: Dialog “Filter”

“Revision range”	The displayed revisions can be filtered by *“Head”*, *“Revision”*, or *“Date”*. ☑: The option fields for *“Start revision”* and *“End revision”* are editable. *“Use UTC time”*: Date display in universal time. For more detailed information, refer to the description “Dialog ‘Select revision’“.
“Message contains”	Display of revision logs that contain a special text in the *“Message”*
“ Author contains”	Display of revision logs of the specified author
“Path contains”	Display of revision logs of the specified path

Table 819: Context menu commands of the revisions

“Compare with base working copy”	Compares the selected revision of the object with the base working copy (without local changes).
“Com with working copy”	Compares the selected revision of the object with the working copy.
“Compare with HEAD revision”	Compares the selected revision of the object with the HEAD revision.
“Compare with previous revision”	Compares the selected revision of the object with the previous revision.
“Update item to revision”	Updates the object to the selected revision. Note: Changes of the project by this command cannot be committed. For VSS users: This is comparable to loading an older version without checkout. To revert a previous commit, the command *“Revert to this revision”* has to be used.

"Revert to this revision"	Reverts the object to the selected revision. This command does not have an effect on the SVN repository as long as the changes are not committed. Internally, SVN reverts the merges for all changes that were made after the selected revision in order to revert the changes of the preceding commits.
"Edit author"	Opens a dialog for changing the author of the revision.
"Edit log message"	Opens a dialog for changing the log message of the revision.
"Revision properties"	Opens the dialog *"Revision properties"* where the properties are displayed. In the dialog, you can activate the *"Add"* and *"Remove"* properties and the option *"Show binary properties"*.
"Create branch/tag from this revision"	Creates a branch or tag from the selected revision.
"Browse SVN repository"	Opens the *"SVN repository browser"* dialog
"Copy to clipboard"	Copies log details of the selected revision to the clipboard This is the revision number, author, date of revision, log message, and the list of changes objects for each revision.

See also

- *Chapter 1.8.5.5.2.6 "Command 'Compare'" on page 6594*
- *Chapter 1.8.5.5.2.7 "Command 'Compare with HEAD revision'" on page 6594*
- *Chapter 1.8.5.5.2.8 "Command 'Compare with revision'" on page 6594*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*
- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 6614*

Command 'Revert', Command 'Revert project'

Symbol: 

Function: This command opens the *"Revert"* dialog. In the dialog, select the objects whose local changes should be reverted, and those that are reverted to the state of the base revision of the working copy.

Call:

- Menu bar: *"Project ➔ SVN"*.
- *"Context menu ➔ SVN"*

If you select nothing or the main node in the device tree, then all modified objects are listed in this dialog (*"Revert project"*). If you selected one or more objects, then only the changes to this object are listed and recursively their sub-objects (*"Revert"*).

Dialog 'Revert'

"Group externals"	☑: The external definitions are grouped by their external storage locations.
"Keep locks"	☑: The lock is retained for all files that are modified by the revert command.
"Select/deselect all"	

When external objects are deleted, Professional Version Control cannot restore this data in SVN offline mode. The user is prompted how to proceed:

- Switch back to SVN online mode and call the external objects.
- Connect now to the SVN server one time in order to complete the current operation, but afterwards switch back to SVN offline mode.
- Skip the retrieval of the external objects. They can be fetched later by updating the project.

See also

- *Chapter 1.8.5.5.2.20 “Command 'Revert to revision', Command 'Revert project to revision'” on page 6603*

Command 'Revert to revision', Command 'Revert project to revision'

Symbol:

Function: This command opens the *“Select revision”* dialog. In this dialog, you select the revision to which the project or the selected objects revert.

Call:

- *“Project ➔ SVN”*
- *“Context menu ➔ SVN”*

If nothing or the base node is marked in the object tree, then the entire project is reverted to a specific revision (*“Revert project to revision”*). If one or more objects are selected, then these objects and their sub-objects are reverted (*“Revert to revision”*).

Dialog 'Select revision'

For a description of the dialog, refer to the section "Select revision".

See also

- *Chapter 1.8.5.5.3.3 “Dialog 'Select revision'” on page 6614*
- *Chapter 1.8.5.5.2.19 “Command 'Revert', Command 'Revert project'” on page 6602*
- *Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 6600*

Command 'Update', Command 'Update project'

Symbol:

Function: This command commits changes in the SVN repository to the project. The update is performed with the HEAD revision.

Call:

- Menu bar: *“Project ➔ SVN”*.
- *“Context menu ➔ SVN”*

If nothing or the main node is selected, then the entire project is updated (*“Update project”*). If one or more objects are selected, then these objects and their sub-objects are updated (*“Update SVN”*).

The following cases are possible:

- Projects are added to the project that are present in the SVN repository, but not in the project. In this case, the message "Added <object>" is issued to the message view.
- Objects that no longer exist in the SVN repository, but are present in the project locally (and not marked as "added"), are treated according to the Subversion standard procedure: If local changes are present, then the object remains in the project as unversioned. If there are no local changes, then the object is also deleted locally because the user can retrieve the object from an older version at any time. In this case, "Deleted object" is issued to the message view.
- Versioned objects that exist in both the SVN repository and the project are updated if they are different. Three cases to observe:
  - No local changes have been made since the last update: In this case, the local object is overwritten by the contents from the SVN repository. The message *"Object updated"* is issued to the message view.
  - Local changes have been made since the last update and the corresponding object type can be merged. When versions have been merged successfully, the message *"Objects merged"* is issued to the message view. If the command is not executed successfully, then the object is marked as "Conflicted object" in the object tree and the message *"Conflicted object"* is issued.
  - Local changes have been made since the last update and the corresponding object type cannot be merged. In this case, the object is marked as "Conflicted object" in the object tree and the message *"Conflicted object"* is issued.

If only some of the objects are updated, it may be that objects with the same name already exist. For example, this situation can come from moving objects to a folder.

For this conflict, you can react in the following ways:

- Do nothing and leave the conflict-causing objects as they are.
- Update (and remove) the conflicting objects in order to correct the conflict.
- Update the entire project in order to remove all conflicting objects and correct the conflict.

See also

- *Chapter 1.8.5.5.2.22 "Command 'Update to revision'" on page 6604*

Command 'Update to revision'

Symbol:

S**Function**: This command opens the *"Update"* dialog. In the dialog, the revision is defined for updating the project.

Call:

- *"Project ➔ SVN"*
- *"Context menu ➔ SVN"*

If you select nothing or the base node in the object tree, then the entire project is updated to a revision (*"Update project to revision"*). If you select one or more objects, then these objects are updated and their sub-objects are updated recursively (*"Update to revision"*). As an option, you can define that the sub-objects are not updated.

The behavior of the updating process (for example merging of conflicts) is similar to the *"Update project"* and *"Update"* commands.

Dialog 'Update'

"HEAD"	: This command behaves the same as the *"Update"* and *"Update project"* commands.
"Revision"	: The revision to which was last updated is selected by the revision number. ...: Opens the dialog *"Log"* for selecting the revision.

"Date"	: The revision to which was last updated is selected by the modification date. *"Use UTC time"*: : The date is displayed in universal time.
"Recursive"	: Default setting. The selected part is updated recursively. This means that all elements below the selected object are also updated.
"Omit external objects"	: External objects are not updated.

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 6603*
- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 6614*

Command 'Update only this'

Symbol:

Function: The command updates the selected objects. In contrast to the *"Update"* and *"Update to Revision"* commands, the child objects are not updated.

Call: *"Context menu ➔ SVN"*

See also

- *Chapter 1.8.5.5.2.21 "Command 'Update', Command 'Update project' " on page 6603*
- *Chapter 1.8.5.5.2.22 "Command 'Update to revision'" on page 6604*

Command 'Disconnect project from SVN'

Symbol:

Function: This command deletes all connections of the current project to SVN by converting the project into a non-versioned project.

Call: Menu bar: *"Project ➔ SVN"*.

Because this operation cannot be reversed, the operation must be confirmed before the command is executed.

Use the command "Connect to existing project" to connect to the SVN repository again at a later time.

See also

- *Chapter 1.8.5.5.2.30 "Command 'Connect to existing project'" on page 6608*

Command 'Switch'

Symbol:

Function: This command opens the *"SVN switch"* dialog. In this dialog, you specify a URL in the SVN repository to which the current working copy of the project is updated. The command switches a project from a branch or tag to another.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement: The project is versioned.

Dialog 'SVN switch'

"From"	Current SVN URL of the project
"To"	Input field for the target URL in SVN ● *"HEAD"*: The *"Select revision"* dialog opens. ● ...: The *"SVN Repository Browser"* dialog opens. There you select the target URL in the SVN repository.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*

Command 'Un-Ignore on commit'

Function: This command removes an unversioned object from the ignore list so that the object is checked by default on commit.

Call: Context menu: *"SVN"*

Requirement: The command *"Ignore on commit"* was executed for the object. The object is marked with the symbol.

See also

- *Chapter 1.8.5.5.2.11 "Command 'Ignore on commit'" on page 6597*

Command 'SVN Cleanup'

Function: This command opens the *"SVN Cleanup"* dialog. In the dialog, you define actions that are performed when cleaning up the SVN working copy.

Call: Menu bar: *"Project ➔ SVN"*.

Dialog 'SVN Cleanup'

Table 820

"Internal SVN working copy"	
"Update time stamps (speeds up SVN status display)"	☑: Corrects recorded time stamps for unchanged files in the working directory. This leads to a reduction in the compare time for future checks. It is not necessary to execute this in regular intervals in the normal workflow.
"Vacuum cached pristine copies (may reduce the size of your project file)"	☑: Cleans the buffer for the original copies by deleting older versions that are no longer referenced by the current project. Advantage: The size of the project file is reduced. Disadvantage: If you downgrade to older revisions, or if you switch between different branches, then the retrieved data size will become larger.
"Clear work queue and force unlock of SVN internal data structures (emergency only!)"	☑: Cleans up the internal SVN task queues and unlocks internal SVN data structures. This should never be necessary during normal work by Professional Version Control. Note: Use this option only if errors occur for SVN commands due to locked working copies. When this is the case, it refers to an error in Professional Version Control. Then please send us an error report (if possible with steps to repeat) to the CODESYS support. Info: These are administrative locks that are internal locks in the SVN working copy. These locks are not set up by context menu commands. For more information, refer to the section "The three meanings of locks" in: *http://svnbook.red-bean.com/en/1.8/svn.advanced.locking.html*
"Project contents"	

“Revert all local changes (use with care!)”	Reverts all local changes to the original status in the SVN repository.
“Release all locks”	Releases all advisory locks in the project (locks visible to the user). These locks are activated by *“Acquire lock”* and *“Steal lock”*.
“Revalidate all locks against the repository (they could have been stolen)”	Checks whether the locally available advisory locks are still valid or have been stolen by someone else for example. All invalid locks are removed.
“Status caches”	
“Clear all caches and refresh status icons”	Deletes all internal caches that Professional Version Control has and updates the status icons. Required only if it issues an error in Professional Version Control through which the caches or the status display are inconsistent.

Command 'Clear authentication data'

Function: This command opens the *“CODESYS”* dialog. In this dialog, define the caches that will be deleted.

Call: Menu bar: *“Project ➔ SVN”*.

Dialog 'CODE-SYS'

The authentication memory contains the authentication data of all SVN repositories for which the user has selected for saving the authorization data. This memory is deleted completely by this command.	
“Clear the shared on-disk cache.”	☑: The data saved on the computer is deleted.
“Clear the RAM cache of this instance.”	☑: The data saved in the RAM is deleted.

The authentication data saved on the computer is stored in `%APPDATA%\Subversion\auth`. This memory path is also used for most other Subversion client applications (for example, TortoiseSVN and AnkhSVN). Therefore, deleting the authentication data affects these applications as well.

Command 'Merge changes'

Symbol: 

Function: This command opens the *“Merge”* dialog. In this dialog, you determine the revisions with the changes to be merged with the working copy of the project.

Call: Menu bar: *“Project ➔ SVN”*.

Requirement: The project is linked to SVN.

Dialog 'Merge'“”

"Kind of merge"	• *"Sync/Reintegrate/Symmetric merge"*: Synchronizes all missing changes from trunk (or a different branch) into this branch. • *"Cherry pick"*: Integrates specifically selected revisions from one branch to another branch. This is necessary, for example, if any error trapping has to be ported back to an older version.
"Merge source"	SVN URL of the SVN repository • Input field • *"HEAD"*: HEAD revision • ...: Dialog *"SVN Repository Browser"* opens for selecting the SVN repository.
"Define start and end revision"	Select this option to merge a cohesive range of revisions with the working copy.
"Start revision"	Defines the range of revisions that are merged with the working copy: • *"HEAD"*: HEAD revision • *"Revision"*: Start and end revision of the range • *"Date"*: Date of the start and end revisions
"End revision"	
"Define revision range"	Select this option to merge individual revisions with the working copy. You can also highlight the individual revisions in the *"Log"* dialog. Note: When defining ranges, CODESYS SVN behaves like other graphical clients, such as Tortoise SVN), and not like the command-line client. Example: For a range of `4-7`, revisions 4, 5, 6, and 7 are merged. See also: *Merging a Range of Revisions*
"Dry run (simulation)"	☑: This command is executed without changing the working copy. Files that are changed during an actual merge are displayed, as well as ranges where conflicts occur.
"Record only"	☑: The revision is marked as "merged" without actually performing the merge.
"Ignore ancestry "	☑: SVN uses path-based differences only, not history-based differences.

See also

- *Chapter 1.8.5.5.3.3 "Dialog 'Select revision'" on page 6614*
- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*

Command 'Connect to existing project'

Symbol:

Function: This command opens the *"Connect to SVN repository"* dialog. In the dialog, you define the URL and the revision of the SVN repository with which the unversioned project is connected.

Call: Menu bar: *"Project ➔ SVN"*.

Requirement: The project is disconnected from SVN.

NOTICE!
Only users who have read access to the entire project (see the CODESYS user and access management) can import the project into the SVN repository or can link to an existing database project.

NOTICE!
This command functions reliably only when the project has already been imported into SVN and then disconnected with the command *"Disconnect project from SVN"*.

Dialog 'Connect to SVN repository'

"URL of existing project"	URL of the SVN repository *"HEAD"*: Selection of the revision in the *"Select revision"* dialog ...: Selection of the SVN repository in the *"SVN Repository Browser"*
"Checkout options"	*"Omit externals"*: External objects are not checked out.
"Revision"	● *"HEAD"*: HEAD revision ● *"Revision"*: Number of the revision ● *"Date"*: Date of the revision *"Use UTC time"*: ☑: Date display in universal time.

See also

- *Chapter 1.8.5.5.2.1 "Command 'SVN Repository Browser'" on page 6585*

Command 'Resolve conflict'

Symbol: 🔍

Function: This command opens the *"<object>"* dialog. In the dialog, the conflicts are displayed and functions for resolving conflicts are prepared in order to merge changes.

Call: Context menu of the object.

Requirement: The object has a conflict that has occurred by updating the object with local changes.

Dialog '<object>'

"Compare"	The local objects are displayed on the left side, and the version from the SVN repository is displayed on the right side.
"Use mine"	A local change is used.
"Use yours"	A change of the version from the SVN repository is changed.
"Apply"	All changes are accepted that you made in this dialog. The status of the object is changed.
"Cancel"	Cancels all changes that you made in this dialog. But the object keeps the conflicted status.

Command 'Work in offline mode'

Function: This command switches to SVN offline mode. In SVN offline mode, the implicit locking and all commands that access the SVN repository are not possible.

Call:

- Menu bar: *"Project ➔ SVN"*.
- Context menu: *"SVN"*

Requirement: The project is linked to SVN.

When switching back to SVN online mode, all present locks on the working copy are checked against the server. If this locking is invalid, then it is released.

Uses case

The user on a machine wants to make changes to the project without disconnecting the connection. At the moment, there is not connection to the server. Despite this, when automatic locking is activated, work is possible because the SVN offline mode deactivates the automatic lock temporarily.

Command 'Copy (Branch/Tag)'

Symbol: 

Function: This command opens the *“SVN Copy Branch/Tag”* dialog. There you can *“Branch”* or *“Tag”* a revision of your project. A specific revision of your project is saved there at this position. A branch is normally used in order to save changes isolated in one version. A tag is used for marking a specific state, for example a shipping version. Internally, it is copied not in the actual sense, but more refers to the revision.

Call: Menu bar: *“Project ➔ SVN”*.

Requirement: The project is versioned.

Dialog 'SVN Copy (Branch/Tag)'

Table 821: “SVN repository”

“From”	SVN path of the current project Example: `https://svnserver/repository/trunk/ControlABC.project`
“To”	Target path in the SVN repository for the copy operation Example of tag: `https://svnserver/repository/tags/V4.4.4.4/ControlABC.project` ...: Dialog *“SVN Repository Browser”* opens for selecting the target path.

Table 822: “Log message”

Input field	Comment the change in a log message. Example: `Tag for version 4.4.4.4 created.`
“ Recent Messages”	Opens the dialog *“Recent Messages”* to display the last log messages. You can click a log message to accept it.

Table 823: “Create copy from”

“Working copy (including local changes)”	The new branch/tag refers to the working copy including all local changes. The local changes are committed to the SVN repository for this purpose.
“Base revision of working copy (<revision number>)”	The new branch/tag refers to the base revision of your working copy whose revision number is displayed in the parentheses. If the working copy already contains local changes, then these are not committed to the SVN repository.
“HEAD revision of the repository”	The new branch/tag refers to the HEAD revision of your project.
“Specific revision in SVN repository” 	The new branch/tag refers to a revision that is displayed on the adjacent button. Click the button to change the revision. The *“ dialog opens.”*.

“Switch to new location ”	☑: After the dialog is confirmed, the working copy switches to the new branch/ tag.

“OK”	The target path is created (as a new tag `../repository/tags/V4.4.4.4` or as a new branch `../repository/branches/new_feature`). Then the revision specified in *“Create copy from”* is copied there.

See also

- *Chapter 1.8.5.5.2.1 “Command 'SVN Repository Browser'” on page 6585*

Command 'Pending Changes'

Symbol:

Function: The command opens the *“Pending Changes”* view. All objects are listed there which have changed from the base revision or which are locked.

Call: *“View ➔ Pending Changes”*

View 'Pending Changes'

The modified or locked objects are shown in the lower half of the view. You can use the *“Commit”*, *“Revert”*, and *“Update”* commands on single or multiple objects. You will find commands for comparing and displaying the version history in the context menu of a selected object.

Double-clicking the object opens the project comparison.

“Select”	Selection or clearing of all objects
“Commit”	Commits local changes to the SVN repository
“Revert”	Reverts the local changes to the state of the base revision of the working copy
“Update”	The command commits changes in the SVN repository to the project. The update is performed with the HEAD revision.
“Keep Locks”	☑ Lock is not released automatically after commit
“Recent Messages”	Shows the last used log messages. You can click a log message to accept it.
“Messages”	Type in a log message that comments your change. Example: `Bug fix error 123`

See also

- *Chapter 1.8.5.5.2.5 “Command 'Commit', Command 'Commit Project'” on page 6591*
- *Chapter 1.8.5.5.2.19 “Command 'Revert', Command 'Revert project'” on page 6602*
- *Chapter 1.8.5.5.2.21 “Command 'Update', Command 'Update project' ” on page 6603*

1.8.5.5.3 Dialogs

Dialog 'Options' - 'SVN Settings'

Tab 'General' Symbol: 

Function: This tab includes the basic settings for Professional Version Control.

Call: Menu bar: *"Tools ➔ Options"*.

Table 824: "Automatic locking and merging"

"Merge"	Behavior for the commands *"Update"*, *"Merge"*, or *"Switch"*, when both sides (working copy and SVN repository) have changed from the base version. ● *"Mark all colliding changes as conflicts"*: The objects are not merged automatically. All changes with a conflict are marked as "With conflict", even if some of them can be merged automatically. ● *"Merge mergeable changes, mark the others as conflicts"*: Changes that can be merged are merged automatically. All others are marked as "With conflict". ● *"Merge mergeable changes, ask the user for the others"*: Changes that can be merged are merged automatically. The user is prompted for all others. ● *"Always ask the user, even for mergeable changes"*: For all changed objects, the user is prompted, even if some of them can be merged automatically.
"Locks"	Behavior such as Professional Version Control objects when they are changed locally. ● *"Always try to lock before modification"*: All objects are locked before they are changed, even if they can be merged. ● *"Only lock the objects which don't support merging"*: Only those objects are locked that cannot be merged automatically. ● *"Never acquire a lock automatically"*: No objects are locked, not even if they can be merged automatically.
"Marker"	● *"Use conflict markers when merging objects"*: If objects with conflicts exist that cannot be merged, then these conflicts are marked in the source code with conflict markers. In addition, the object itself is marked as being merged successfully (no conflict). ● *"Leave non-mergeable objects as conflicted"*: No conflict marker is set. Objects that cannot be merged remain in the status "With conflict".
"Prompt the user when automatic locking fails."	☑: If it is not possible, to lock the object, then the dialog *"Automatic locking failed"* opens (see dialog description).

Example of conflict markers

```
Foo();
>>>>>>>>>>>>>>>
I := I + 1;
===============
I := I + 2;
<<<<<<<<<<<<<<<
```

Table 825: "Server check"

"Check server for updates and locks"	☑: Professional Version Control checks in the specified time interval that objects have been updated on the server. In addition, it checks whether objects are locked or locks have been stolen.
"Check interval (minutes)"	Example: 10

Table 826: "Ignore for comparison"

Ignore whitespace	☑: Whitespace differences between the current project and the reference project are ignored.
Ignore comments	☑: Comments in the programming code are excluded from the comparison.
Ignore Properties	☑: Object properties are excluded from the comparison.

> *Some of the SVN options can be overwritten by the project-specific settings.*
>
> *Project-specific settings are defined in the menu "Project ➔ Project settings", category "SVN Settings".*

See also

- *Chapter 1.8.5.5.3.5 "Dialog 'Automatic locking failed'" on page 6617*
- *Chapter 1.8.5.5.3.2 "Dialog 'Project Settings' - 'SVN Settings'" on page 6613*

Tab 'SSH'

Symbol:

Function: This tab contains the settings for the SSH protocol.

Call: Menu bar: *"Tools ➔ Options"*.

Table 827: "SSH client implementation"

"libssh2 (recommended)"	Professional Version Control uses Libssh2 for establishing a connection via SSH protocol. This is the recommended setting.
"SharpPlink (backwards compatibility)"	Professional Version Control uses plink.exe for establishing a connection with SSH servers. This option is required only for communication with outdated servers that support the deprecated SSH-1 protocol.

> *The SSH configuration can be overwritten by means of the environment variable `SVN_SSH` or server-specific by means of the SVN configuration file.*

See also

- *Tunneling via SSH*

Dialog 'Project Settings' - 'SVN Settings'

Symbol:

Function: The behavior of the integrated SVN version control system is configured in this dialog.

Call: Menu bar: *"Project ➔ Project Settings"* (*"SVN Settings"*).

Requirement: A project is open.

Table 828: "Automatic locking and merging"

With these settings, you can overwrite the default settings that were made in the dialog *"Tools ➔ Options"*, category *"SVN Settings"*.	
"Merge"	Behavior for the commands *"Update"*, *"Merge"*, or *"Switch"*, when both sides (working copy and SVN repository) have changed from the base version.

“Locks”	Behavior such as Professional Version Control objects when they are changed locally.
“Marker”	Behavior for conflicts

Table 829: “Settings SVN version info”

“Create SVN_VERSION_INFO constants for IEC access”	☑: The object `SVN_VERSION_INFO` is created and includes global constants or variables for the project metadata. ☐: The object `SVN_VERSION_INFO` is not available. When you activate the option, the object is created automatically. When you deactivate the option, the object is removed from the project automatically.

See also

- *Chapter 1.8.5.5.3.1 “Dialog 'Options' - 'SVN Settings'” on page 6612*

Dialog 'Select revision'

Function: This dialog shows the currently selected revision. You can edit the selection there.

“Revision”	
“HEAD”	◉: The latest revision (top revision) within a branch is displayed.
“Revision”	◉: A specific revision is displayed by the revision number. Example: `3` Tip: Click … to show the revisions. Then the *“Log”* dialog opens to display the revisions and the associated actions. The revision that you select there is applied.
“Date”	◉: A specific revision is checked out by the modification date. This is the highest revision at the given time (the last revision before that time). Example: `12/23/2016 11:59:59` Tip: See section "Revision identifiers" in "Version control with Subversion"
“Use UTC Time”:	☑: Modification date in universal time is used.
“Reset recursively”	☑: All objects below the selected object are also reset. The action fails if ● Objects have been moved in or out of the hierarchy below ● Objects outside of the hierarchy would be changed by implicit dependencies

See also

- *Chapter 1.8.5.5.2.18 “Command 'Show log', Command 'Show project log'” on page 6600*
- *"Version control with Subversion", Section "Revision identifier"*

Dialog 'Subversion Authentication'

The dialogs are used for authenticating the server/client connection. A server or client authentication is performed depending on the initial situation and protocol.

Overview of possible protocols and dialogs

- `svn://`: The SVN protocol; either unencrypted or SSL/TLS encrypted
  - Can prompt for user name and password (even for an unencrypted connection)
  - Can prompt for a server certificate from the dialog for authentication in order to confirm the server if a certificate is unknown, defective, or invalid (for TLS/SSL encryption)
  - As an alternative or in addition to the user name and password prompt, the client can also be authenticated with client certificates (for TLS/SSL encryption). The dialogs for authentication open with the client certificate.
- `http://`: SVN via http, unencrypted
  - Can prompt for user name and password
- `https://`: SVN via http, SSL/TLS encrypted.
  - Can prompt for user name and password
  - Can prompt for a server certificate from the dialog for authentication in order to confirm the server if a certificate is unknown, defective, or invalid.
  - As an alternative or in addition to the user name and password, the client can also be authenticated with client certificates. The dialogs for authentication open with the client certificate.
- `svn+ssh://`: The SVN protocol, encrypted through an SSH tunnel. SSH (Secure Shell) is the usual networking tool in Linux/Unix for accessing other computers.
  - Can prompt for user name and password
  - Prompts for server certificate in the dialog for authentication if the server is still unknown in order to be sure that it is the correct server.

Dialog for authentication with a server certificate

Initial situation: CODESYS (as a client) receives an unknown or defective server certificate.

This dialog shows information about the certificate. There you can confirm the identity of the server.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 830: “Certificate information” (for SSL/TLS connections)

“Host name”	Example: `svn repository`
“Thumbprint”	
“Valid from”	
“Valid to”	
“Issuer”	Example: `ABB AG`
“Certificate”	

Table 831: “SSH server key information” (for SSH connections)

“Key type”	
“Key size (bits)”	
“Key thumbprint”	

“Save information to RAM”	☑: The certificate is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The certificate is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates and established the connection.

The certificate memory is secured cryptographically and distributed with other SVN clients.

See also

- *Version Control with Subversion*

Dialog for authentication with a client certificate

Initial situation: The SVN server requires a client certificate for authentication.

In this dialog, you select the client certificate in order to confirm the identity.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 832: “The SSL server requires a client certificate file.”

“File”	Client certificate file

“Save information to RAM”	☑: The certificate is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The certificate is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates and established the connection.

Dialog for authentication with a pass phrase

Initial situation: The SVN server is configured so that it demands a client certificate for authentication. The applied certificate is protected by a pass phrase.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

Table 833: “A pass phrase is needed to unlock the certificate.”

“Pass phrase”	Example: `***`

“Save information to RAM”	☑: The pass phrase is saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: The pass phrase is saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Authenticates with client certificates by means of a pass phrase and establishes the connection.

Dialog for authentication with a user name and password

Initial situation: The SVN server is configured so that it demands a user name and password for authentication.

“Authentication area”	Connection that is secured Example: `https://svn repository:443`

“User name”	Example: `a.mayr`
“Password”	Example: `***`

“Save information to RAM”	☑: Saved to the working memory. Then the client recognizes in the current CODESYS session for future connections. If you restart CODESYS, then you have to accept the certificate again.
“Save to disk”	☑: Saved on the computer and it is available for future connections. If you restart CODESYS, then the saved certificate is used.

“OK”	Establishes the connection and authenticates it.

Dialog 'Automatic locking failed'

The dialog shows a list of all objects for which an automatic locking was not possible. In the optoins you define how Professional Version Control will resolve the confilict.

Table 834: “Automatic Locking and Merging”

• *“Try to steal the lock for the affected objects”* • *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if another user has locked the object.
• *“Update the affected objects to the newest revision”* • *“Update the whole project to the newest revision”* • *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if there exists a more current version of the object on the server.
• *“Activate the "Offline Mode" to temporarily suppress locking”*	These options are displayed if no connection can be established to the server.
“SVN Project Settings”	Opens the SVN project settings dialog (menu *“Project ➔ Project Settings”*). There you can change the settings for the automatic locking.
“SVN Settings”	Opens the general SVN project settings dialog (menu *“Tools ➔ Options”*).

See also

- *Chapter 1.8.5.5.3.1 “Dialog 'Options' - 'SVN Settings'” on page 6612*
- *Chapter 1.8.5.5.3.2 “Dialog 'Project Settings' - 'SVN Settings'” on page 6613*

1.8.5.5.4 Objects

Object 'SVN_VERSION_INFO'

Symbol: 

The object contains the SVN metadata of the project as global constants or variables in a variable list. It is located in the *“POUs”* view. You can specifically retrieve the data of the global constants or variables by the application. By calling specific data, you can also reduce the memory usage on the controller.

The SVN metadata is provided for this purpose, subdivided over multiple global variable lists (GVLs):

- *“SVN_VERSION_INFO”*
- *“SVN_Info_Summary”*
- *“SVN_Info_SummaryW”*
- *“SVN_Info_URI”*
- *“SVN_Info_Revisions”*
- *“SVN_Info_Flags”*
- *“SVN_info_LastChange”*

The SVN_VERSION_INFO object is created automatically when a project is imported to a SVN repository. To do so the option *“Create SVN_VERSION_INFO”* in the dialog *“Import project to SVN”* must be activated.

Furthermore you can create the object or remove it from the project with the option *“Generate SVN_VERSION_INFO constants for IEC Access”* (Dialog *“Project ➔ Project Settings”*, category *“SVN Settings”*).

Table 835: Global Constants

Name	Data type	Description
MINREVISION	LINT	Lowest revision number of the working copy
MAXREVISION	LINT	Highest revision number of the working copy
PARTIAL	BOOL	TRUE: The working copy is incomplete. Example: Cancellation during the last update due to a network error or a checkout.
MODIFIED	BOOL	TRUE: Local changes were made.
SWITCHED	BOOL	TRUE: Parts of the project were branched (with the *"Switch"* command).
VERSION	STRING	Version identification, similar to Apache™ Subversion® (subversion.exe) Example: 12:34M, means MINREVISION = 12, MAXREVISION = 34, MODIFIED = TRUE For more information, refer to the documentation for Apache™ Subversion®.
CLEAN	BOOL	TRUE: The version is clean. This is the case when MINREVISION is equal to MAXREVISION, the working copy is complete, and non-versioned, and is was not switched.
URL	WSTRING	SVN-URL of the project Example: https://svnserver/repository/trunk/ControlABC.project

If a controller does not support the data type WSTRING, then a compiler error is issued when accessing the object SVN_VERSION_INFO.

See also

- *Chapter 1.8.5.5.3.2 "Dialog 'Project Settings' - 'SVN Settings'" on page 6613*
- *Chapter 1.8.5.5.2.3 "Command 'Import project to SVN'" on page 6589*

1.8.6 Subversion

1.8.6.1 Project Version Control with Subversion

Introduction

Automation Builder projects can be stored in Subversion (SVN) repositories by using the Project Version Control. The Project Version Control can be used to track changes on a project and to have access to historic versions of the whole project or objects in the project. It is possible to hold different versions of a project in branches and to compare these versions. The Project Version Control enables multiple engineers to work collaboratively on the same project.

Basic knowledge

Make yourself familiar with the concepts of SVN.

This manual about Project Version Control is additionally to the following information and describes mainly the specific behavior of Subversion in Automation Builder.

- Homepage of Subversion: *http://subversion.apache.org/*
- Online user manual for Subversion: *http://svnbook.red-bean.com/*
- Documentation on SVN integration in Automation Builder: Refer to subfolder .

1.8.6.1.1 Preconditions

Automation Builder

- In Automation Builder, the Project Version Control must be installed.
- A valid license for the Project Version Control must be activated.
- All collaborating users working on the same project need:
  - Automation Builder installed in the same version with the same features.
  - License for same edition.
  - Same set of optional third party device descriptions.
  - Same set of optional customer specific packages.

SVN server

The Project Version Control can be used in combination with an SVN server in version 1.6 or newer, the repository format should be 1.5, 1.6 or 1.7. Newer repository formats are not yet supported.

The usage of local repositories in the local file system or even on a network share is strongly discouraged.

1.8.6.1.2 Working with Project Version Control

- All objects in the device tree or POU tree are represented by an object in the SVN repository, there might be hidden objects that are not visible in the tree but that exist in SVN.
- The smallest unit in the SVN repository is one object including all its data like name, parameters, device identification.
- Objects are identified in the SVN repository by their name. Renaming one object in Automation Builder means to delete it from the SVN repository and add a new one to the SVN repository. Renaming an object causes a break in the history of that object.
- By default objects are locked before they are changed to prevent other users from changing the object. The locking strategy can be changed in the user options.
- Objects can be compared to other versions of the same object, many differences/changes between the current object in the Automation Builder project and the compared object can be merged into the object in the Automation Builder project. Merging changes could be used to resolve conflicts in case concurrent changed can not be avoided.

- To ensure consistency it is required and also enforced that some changes can be committed or reverted only together.
  - All changes to device objects in the hardware tree that are sub-nodes to the same top level device. Note: Objects that are not devices are excluded, e.g. the application node.
  - All changes below the AC500 PLC application node.
- Most SVN operations can not be performed while other external applications like CODESYS or Panel Builder work on files that are embedded in Automation Builder project.
- Some operations like changing the target or updating the project to the latest device (description) versions do a recursive lock of the whole AC500 PLC. If the lock can’t be obtained the operation is aborted.
- Some objects contain internal data that has no meaning to the end user but is also important. Changes on such data are not shown in the compare dialog or are summarized by a placeholder like "There are hidden changes".
- Including externals is not supported.

1.8.6.1.3 Recommendations on Working with Project Version Control

Be collaborative

- Multiple users that work collaboratively on the same project should agree on their responsibility for certain parts of the project where they do changes to avoid conflicts and tree conflicts.
- Agree on locking strategy used by all users working on the same project.
- Distribute the work between multiple users meaningful.
  - It is suggested to setup the hardware structure at first before other users checkout a project to work on it and limit structure changes in the hardware tree to the minimum.
  - Before adding objects, especially top level objects, users should agree that only one user adds objects at top level or below the same parent, or agree on unique names for the objects to add. The default naming scheme for new objects bears the risk of name conflicts. These conflicts could be resolved only by reverting the changes of the user who later tried to commit the changes.

Be careful

- The SVN integration (and also project compare) gives lot of power to the user, users should be sure to do only things they fully understand. Especially by merging changes incomplete it is possible to create inconsistent data.
- Adding devices, removing devices or even changing parameters can have side effects to other devices, do not change objects/parameters to their original state by merging that were not done explicitly.
- Commit changes frequently to SVN.
  - To release locks that you don’t longer need.
  - To reduce the risk of conflict with co-workers.
  - To keep the sets of changes to commit small.
- Do frequent updates when collaborating in a team.
  - To be up-to date.
  - To keep the sets of changes to get from SVN small.
  - To reduce the risk of losing work results in case of conflicts.
- To avoid conflicts, it is suggested to stay with the default setting to automatically lock objects before doing changes. Consider explicit recursive locks of sub-trees where you plan bigger changes.
- Prefer a clean checkout over using the switch command to change between different branches.
- Do not use the switch command to change between unrelated projects, this could corrupt the Automation Builder project (local copy, not in SVN) easily.
- Commit local changes to the SVN repository before creating a branch.

Be effective

- Give objects good/correct names after adding them and use renaming of objects already committed to SVN sparely to maintain a continuous history in the SVN repository.
- The goal to revert only single changes of all changes done that must be committed/reverted together, could be achieved by using project compare or the object compare dialog.
- If changes can’t be committed to the SVN repository because of locks hold by other users, it is possible to create a branch, use the switch command to change to this branch and commit the changes there. The branch and base line could be merged together later.

1.8.6.1.4 Known Issues and Troubleshooting

Not all changes are shown for all objects, but hidden changes are also important.

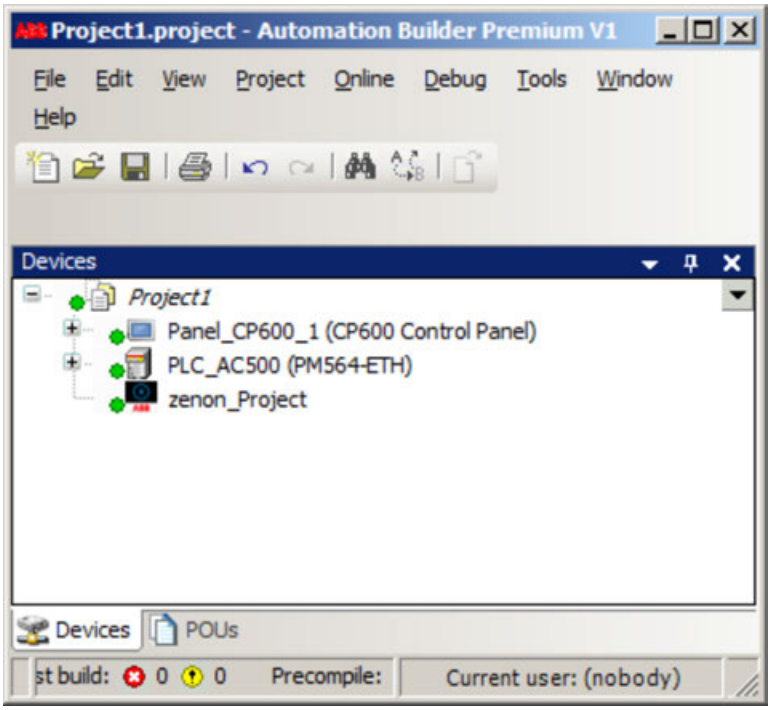
The device pool may be changed as side effect of several operations, including opening the project.

When a project was corrupted (by performing an update that tried to add an AC500 Communication Module) it is possible to save this project and merge changes to a project that has been cleanly checked out by project compare.

1.8.6.2 SVN Support Examples

1.8.6.2.1 Importing Automation Builder Project to SVN Repository

1. In the Automation Builder main menu, go to *“Project ➔ SVN ➔ Import Project to Subversion”*.
2. Enter user credentials and click *“OK”*.
3. Select SVN server repository to import Automation Builder project and click *“OK”*.
   ⇨ The Automation Builder project is imported into the selected repository and connected automatically to the repository. The imported project nodes are identified with green indicators.



1.8.6.2.2 Logging in User2

1. In the Automation Builder main menu, go to *“Project ➔ SVN ➔ Checkout”*.
2. Enter user credentials and click **OK**.
3. Select the repository location, project folder and revision if any and click *“OK”*.
   ⇨ The project will checked out of the repository, saved in the selected location and opened as a primary project.

The tables below provide the descriptions of the options availlable in the check-out dialog.

Checkout options	
Omit externals	Do not checkout external objects.
As library project	Saves the project as a CODESYS library file.

Revisions	
HEAD	Checks out the Head revision.
Revision	Allows to select the required revision of the project.
Date	Allows to select a revision date of the project.

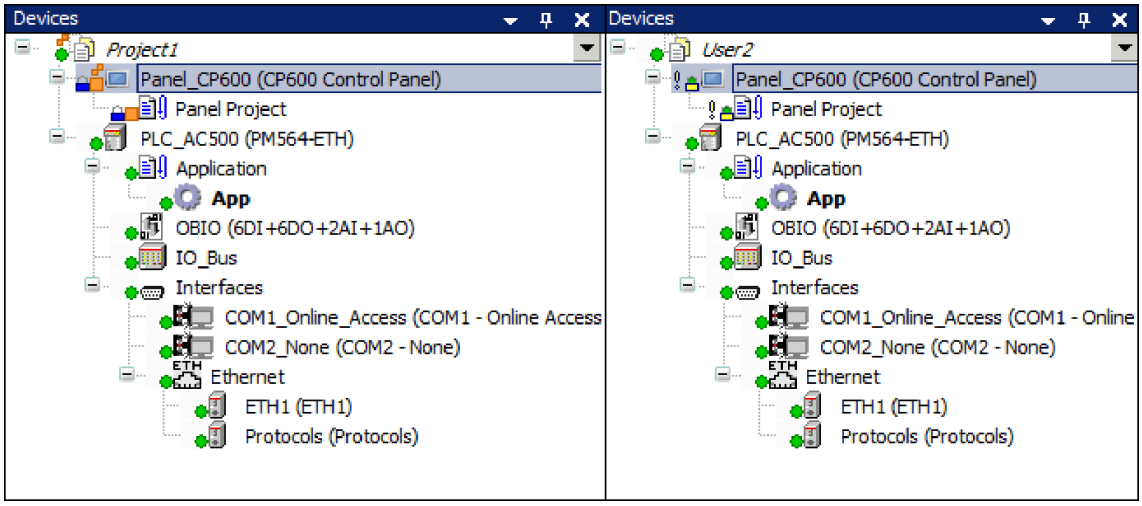
The following instances can occur in the workflow.

- If the project contains any updates, the specific project level node is indicated with .
- If a new object is added to the project, the newly added node is indicated with .
- If the project node is deleted, the specific node is indicated with .

1.8.6.2.3 Examples

Example 1

If User1 modifies Panel_CP600 project, then the node indicator turns to orange with lock symbols. If User2 need to modify the same Panel_CP600 project, the Panel_CP600 project appears with a lock symbol.

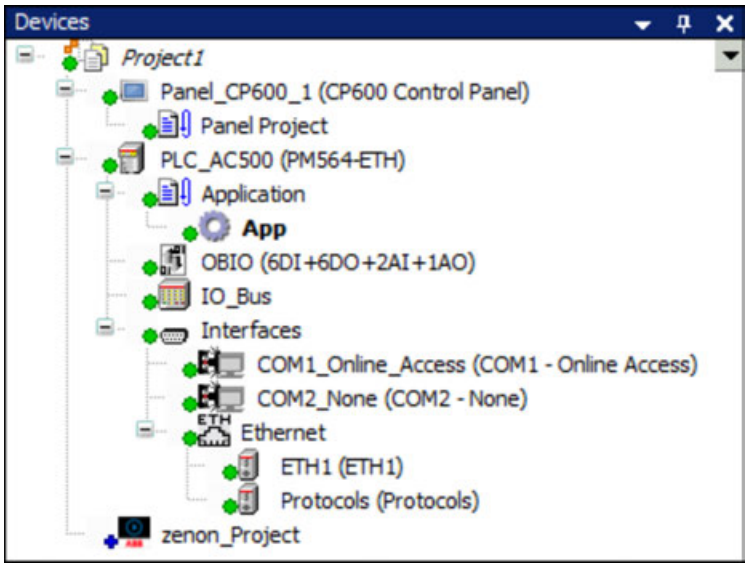


To steal the lock of an affected object, proceed as follows:

1. Double-click *"Panel_CP600"* project.
   ⇨ Automatic lock failed dialog is displayed.
2. Enable *"Try to steal the lock for the affected objects"* and click *"OK"* to steal the lock.
3. In the Lock Message window, enter the reason to steal the lock and click *"OK"*.
4. User2 can modify and commit the project.

Example 2

If User1 adds a new object to the project and commit the changes, then User2 can update the project to see the latest modifications.



Example 3

The user can revert to any of the available project revisions.

1. Right-click on object node and select *"SVN ➔ Revert to Revision"*.
2. Select or enter the revision number and click *"OK"*.
   ⇨ The revision command reverts local changes of this object back to the specific revision of the working copy.
3. Right-click on the object node and select *"SVN ➔ Commit"*.
4. In the commit window, enter the reason to change the project and click *"OK"* to make the changes.
   ⇨ The project node is updated with the latest changes.

Example 4

SVN server allows to select the required revisions of Automation Builder project. You can checkout the project using *"Project ➔ SVN ➔ Checkout"* and then enter the credentials and click *"OK"*.

In the check-out dialog, do the following:

1. Select the project repository.
2. Activate *"Revision"* and select or enter the revision number and click *"OK"*.

The user can work on the selected revision. To commit the changes to the project, right-click on project and select *"SVN ➔ Commit Project"*.

1.8.7 Python

1.8.7.1 Python script support

Using scripts

Scripting allows python scripts to be used to automate project configuration in Automation Builder. Parameters can be added to scripts, so that a generic script can be customized before execution. The user can add a script to most parts of the device tree. A script can be started either from the user interface (by a command or with the python scripting editor) or from the Windows command line and is saved with the project.

With the scripting feature commands or complex program operations can be automated.

Examples of use cases:

- Integration of Automation Builder in automatic build server environments:
  - continuous integration (CI)
  - continuous delivery (CD)
  - continuous testing
- Integration with third-party software, for example:
  - code generators
  - creation of projects that are custom tailored to a specific machine configuration
- Creation of documentation
- Updating of libraries: Setting of project information during the release process
- Automatic testing: Mostly in connection with the Professional Test Manager
- Outputting variables via monitoring APIs

Licensing

A valid license is required to use the scripting. If you open a project with the existing script object without a valid license, you are not allowed to add or edit the scripts. However, the scripts are not removed from the project.

Scripting language

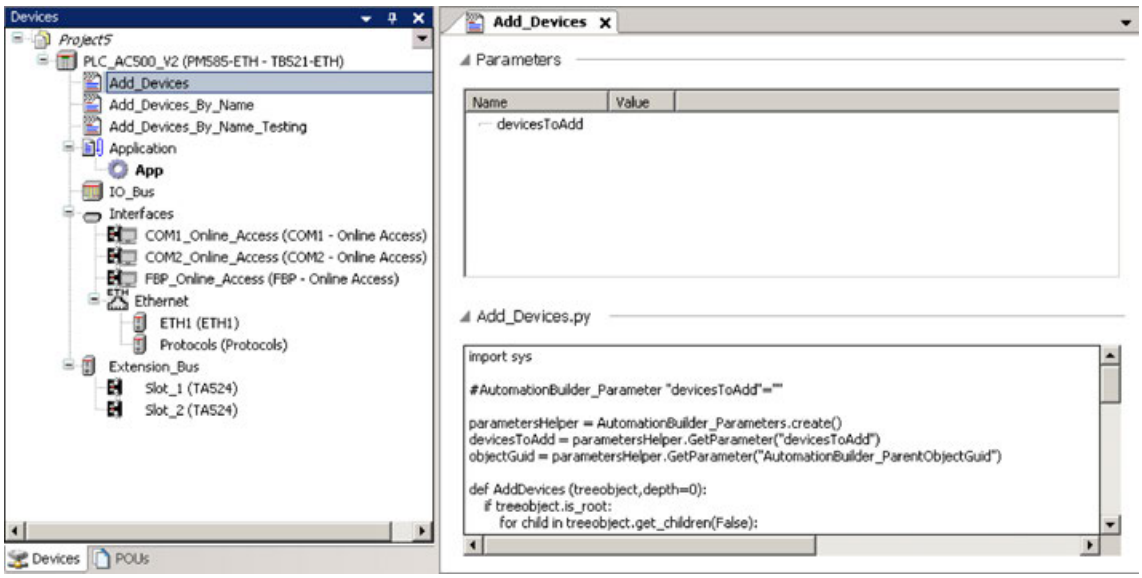
The Automation Builder scripting language is modular and based on IronPython. For this purpose, the Automation Builder “ScriptEngine” component combines the IronPython interpreter with the Automation Builder development environment which makes the extensive python framework libraries available including file access in networks and much more.

1.8.7.2 Working with script objects

Scripts to execute can be added to and stored in the Automation Builder project. Additionally, parameters can be added to scripts, so that generic scripts can be customized before execution.

Adding a script object to the project

1. In the device tree, right-click on a node (e.g. a PLC node) and click *“Add object”*.
2. Under *“Scripting category”* select *“Script ➔ Add object”*.
   ⇨ The 'Add Script' dialog is displayed.
3. Browse and select a script from the file system or create a new script by clicking *[Add]*.
   ⇨ A script is added below the selected node and the editor is opened.



4. The default parameter values are read from the script. The user can edit the default values as required.

Editing scripts within Automation Builder is not supported. You can use an external editor to edit the script and then import it to Automation Builder.

The script objects can be reused within the project via copy-and-paste around the device tree.

Execution

The user can execute the script with the parameter values via the execute button in the editor or via right-click on the script object in the device tree by selecting *“Script ➔ Execute”*.

Import

The user can import the script from the file system. This will replace the contents of the current script object with the contents of the imported file. Optionally, parameter values will be preserved if the imported script has a matching named parameter. In the device tree, right-click on a script object and select *“Script ➔ Import”*.

Export

The user can export the selected script and saved it as a new file in the file system. The exported file does not include any edited parameter values. In the device tree, right-click on a script object and select *“Script ➔ Export”*.

Parameters

The following instructions help the user to create parameters in the python script:

- Parameters must be defined in the script.
- Parameters and values are optional.
- The `ParameterName` and the `ParameterValue` must be delimited with symbols. The format must be as follows:
  "#AutomationBuilder_Parameter {"ParameterName"} {= "ParameterValue"}
- {ParameterName} is the name given to the parameter. This allows the values to be referenced in the python script.
- {ParameterValue} is the default value given to the parameter. This value can be modified in the editor.

The example below shows the format of the `ParameterName` and `ParameterValue` in the script.

- #AutomationBuilder_Parameter "numWidgets": creates a new parameter called `numWidgets`.
- #AutomationBuilder_Parameter "numWidgets" = "4": creates a new parameter called `numWidgets` and initializes to the value 4.

Using parameters within the python script:

- Parameters can be used in the script by creating an instance of the parameter helper: parameterHelper = AutomationBuilder_Parameters.create()
- Individual parameters are retrieved by calling: GetParameter(parameterName). devicename = parameterHelper.GetParameter("Name")

Python script examples

A set of python script examples are available in the path %Public%\Documents\Automation-Builder\Examples\Python scripts.

1.8.7.3 Python script editor

In Automation Builder a browser-based python script editor is integrated. This allows the user to modify the existing python script, to create a python script from the scratch and to finally execute the script. Moreover, it assists the user in writing the script with the following features:

- Auto suggest
  - IntelliSense suggestions for the python syntax during typing.
  - IntelliSense for CODESYS script engine and Automation Builder injected script objects.
  - Built-in language service that provides complete code intelligence for objects, properties and methods.
  - Details of the object with *[CTRL]* + *[spacebar]*.
- Auto completion
  Press the Enter key on a function suggested by IntelliSense in order to insert it.
- Python syntax highlighting (basic syntax colorization)
  The function and its respective namespace is automatically colored in order to match colors.
- Matching brackets
  Matching brackets are highlighted as soon as the cursor is near to one of them using the command palette.
- Zoom
  Changes the font size of the editor's content.
- Find and replace
  Support of 'Find' (search for a keyword) and 'Find and replace' (search and replace a keyword). This feature is supported in the editor, however not integrated in Automation Builder platform.
- Minimap
  High level overview of the script for a quick navigation and code understanding.
- Copy/paste
  Support of 'copy and paste' of the script text within and into the editor.
- Undo/redo
  Support of 'undo/redo' for editing actions. This feature is supported in the editor, however not integrated in Automation Builder platform.
- Keyboard shortcuts
  Keyboard shortcuts allow to perform most tasks directly from the keyboard (e.g. *[CTRL]* + *[Z]*, *[CTRL]* + *[Y]*) including copy and paste. For further keyboard shortcuts refer to the command palette (*[F1]*).
- Folding
  Support of folding and expanding script regions.
- Comment/uncomment the code
  Support of commenting (*[CTRL]* + *[K]*) and uncommenting (*[CTRL]* + *[C]*) code through shortcuts.
- 'Execution' button
  Executes the script directly in the editor window.
- In order to start a new script from the scratch the user can start with an empty editor. This can be done via the 'Add script' dialog without script file selection.

For further features that can be used in the python script editor refer to the command palette (*[F1]*).

Limitations with CODESYS script engine IntelliSense

- No IntelliSense available for return type of a property.
- No support of IntelliSense for keyword “None”.
- No IntelliSense support for method overloading.
- No IntelliSense support for methods that return an object.
- Private methods are also part of IntelliSense. Refer to the CODESYS script engine document to verify the access modifier.

1.9 Human machine interface

1.9.1 Panel Builder interface

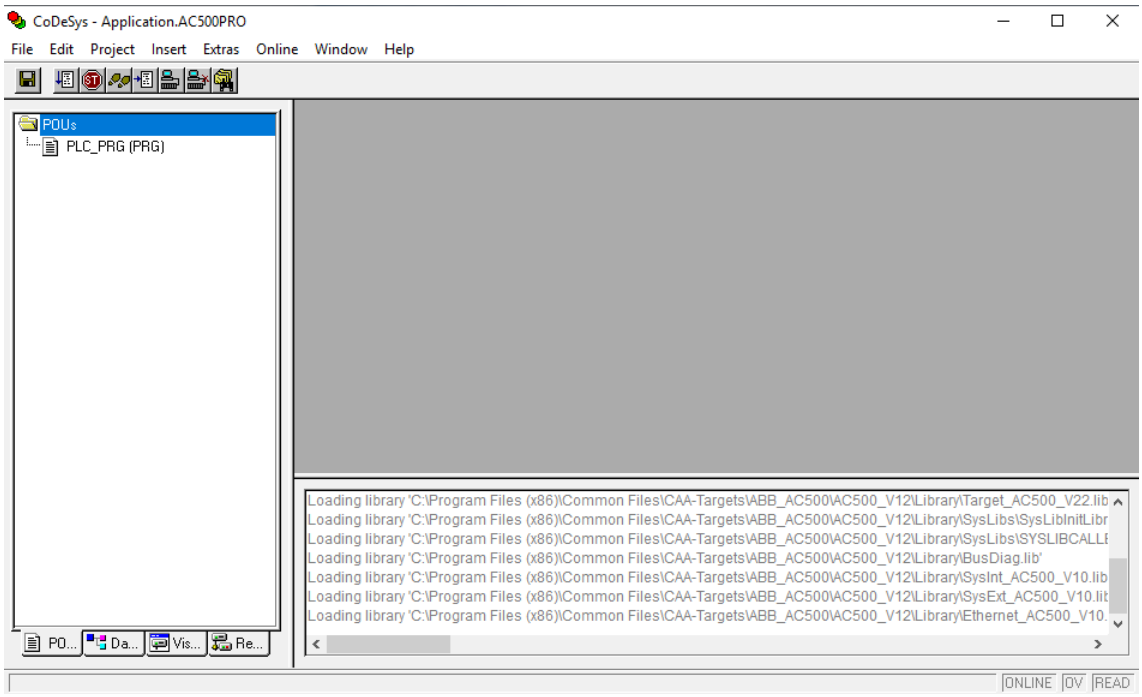
This document describes HMI CP600 Control Panel configuration in Automation Builder and starting HMI configuration and programming software Panel Builder 600 from Automation Builder. The Panel Builder project created for the HMI CP600 is stored within the Automation Builder project.

1.9.1.1 Adding desired AC500 PLC to the project

Basic settings in CODESYS application

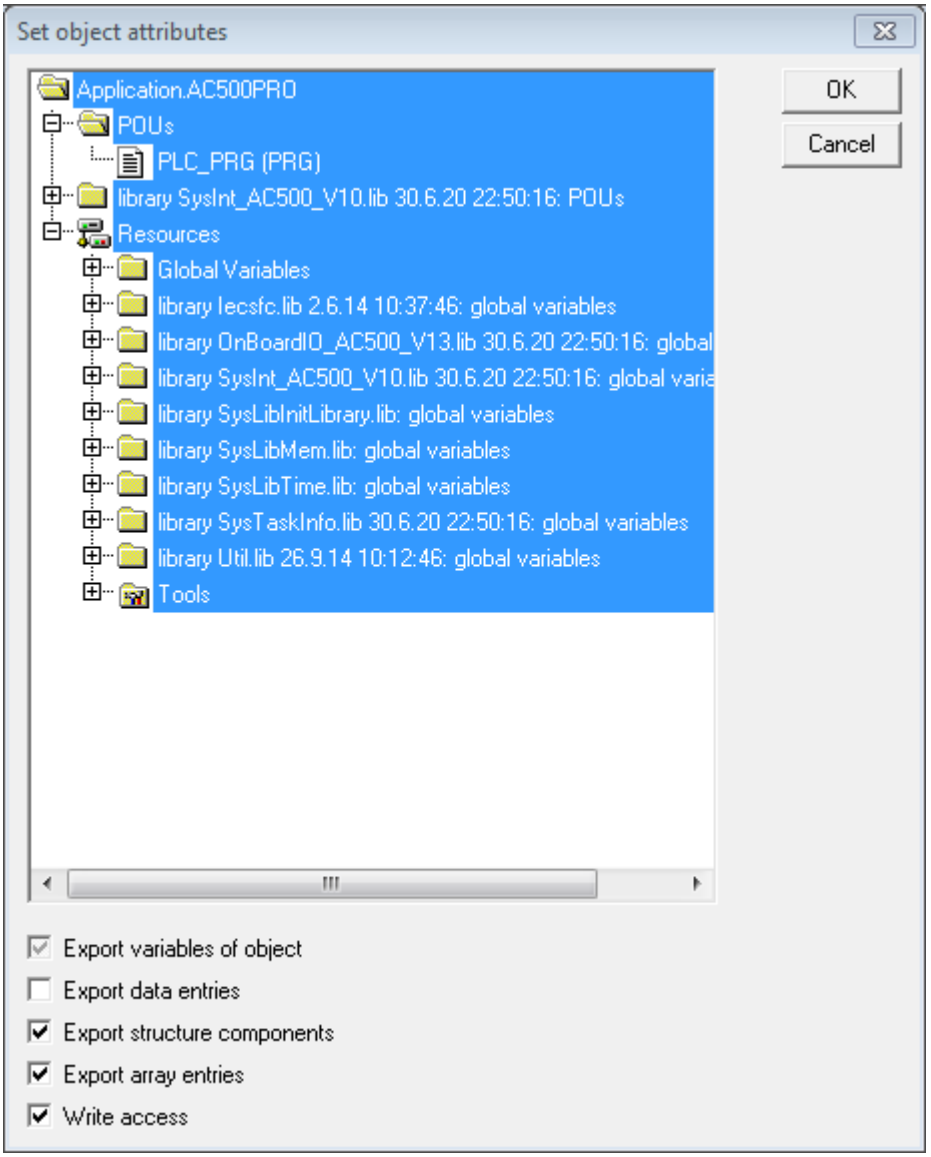
Before configuring basic settings, define the variables in CODESYS application.

1. In the Automation Builder device tree double-click the *“Application”* node, to start the CODESYS application.



2. In the CODESYS application main menu, click *“Project ➔ Options”*.
3. In the 'Options dialog', click *“Symbol configuration”* and enable the options *“Dump symbol entries”* and *“Dump XML symbol table”*. Then, click *“Configure symbol file”*.

4. Enable the following checkboxes and click *[OK]*.



⇨

> *Export variables of the object will be in gray background. Double-click to activate.*

5. In the CODESYS application window, click *"Resources ➔ Target settings ➔ General tab"*. Enable *"Download symbol file"* and then click *[OK]*.
6. In the CODESYS application main menu, select *"Project ➔ Build"* to compile the project.

After adding/modifying the variables, update the Panel Builder project. For further information, see *"Project information" on page 6632*.

Exporting symbol file

1. In the device tree, right-click the *"Application"* node and click *"Export ➔ Symbol file"*.

> *The precondition to generate a symbol file is to create the application and to perform PLC program build in CODESYS.*

2. Select the desired location in the file system to save the symbol file.
3. In the device tree, double-click *"Application"* to start CODESYS application.

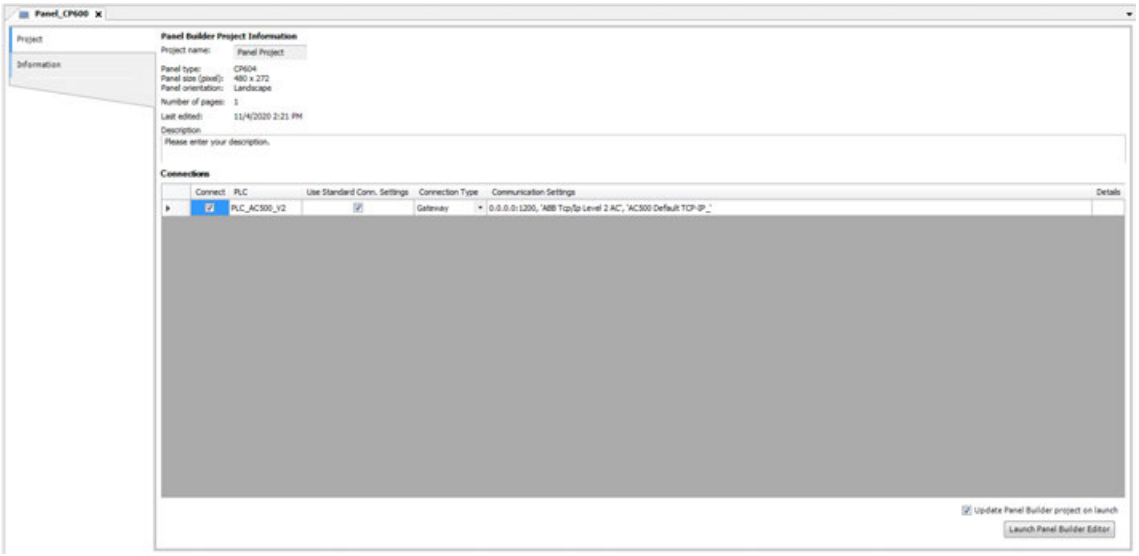
1.9.1.2 Creating a Panel Builder project

Adding a panel object

1. Right-click in the Automation Builder device tree and click *"Add object ➔ Panel-CP600"*
2. Click on *"CP600 Control Panel"* and click *"Add object"*
   ⇨ A Control Panel object is added to the Automation Builder device tree.

Starting a Panel Builder project

1. In the device tree, double-click *"Panel CP600"* object to start Panel CP600 screen.



2. Select the required PLC and enable the checkbox in the 'Use Standard Connection Settings' column to use it as a standard gateway connection.

   You can set communication settings using the application program or by creating custom communication settings. Custom communication settings can be configured by clicking the button in the 'Details' column.
3. Enable the *"Update Panel Builder project on launch"* checkbox and click *[Launch Panel Builder Editor]*.

> *If you update Automation Builder project with new variables and data types or if there are changes in existing Automation Builder project variables and data types (new, modified, deleted), recompile CODESYS application to refresh the symbol file, then launch Panel Builder editor.*

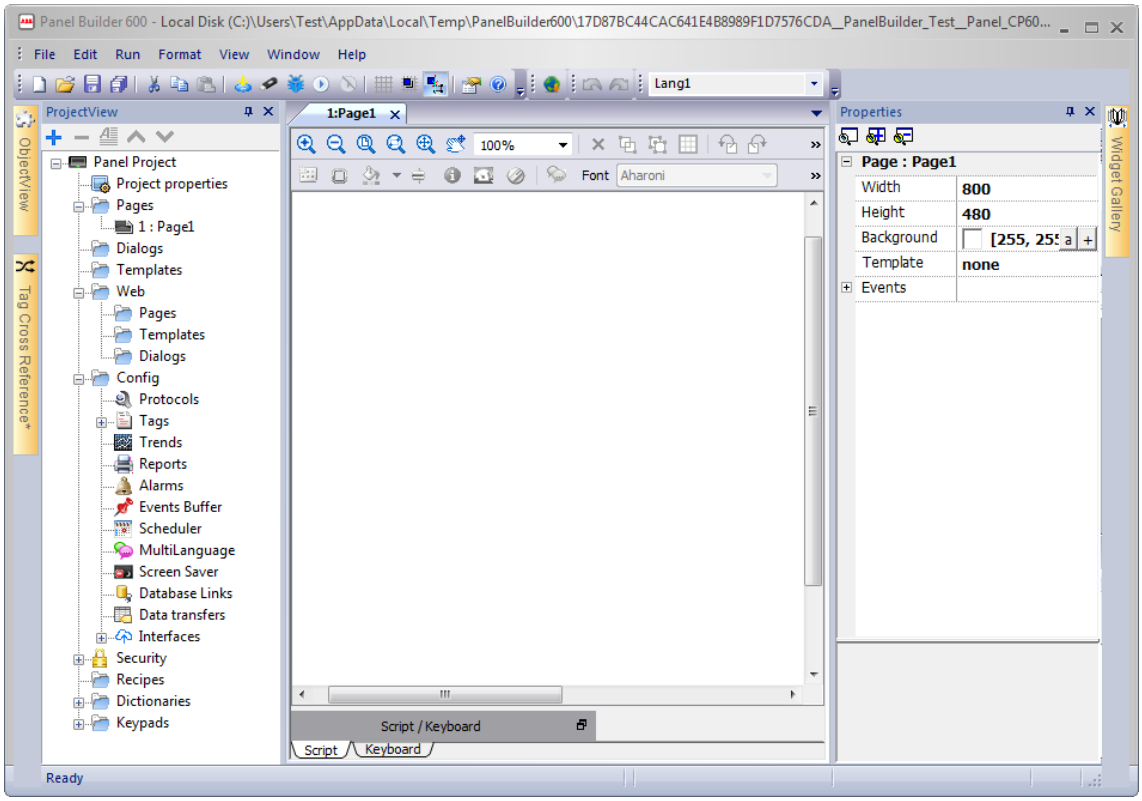
4. Select *“New”* and click *“Open”* to create a new HMI project.

   ⇨ A project wizard is displayed.

> *If you want to import an already existing Panel Builder project file from the file system, select “Import existing project file” and proceed.*

5. Select the required panel type and orientation and click *“Finish”*.

   ⇨ A new project wizard starts only if the Panel project is empty.

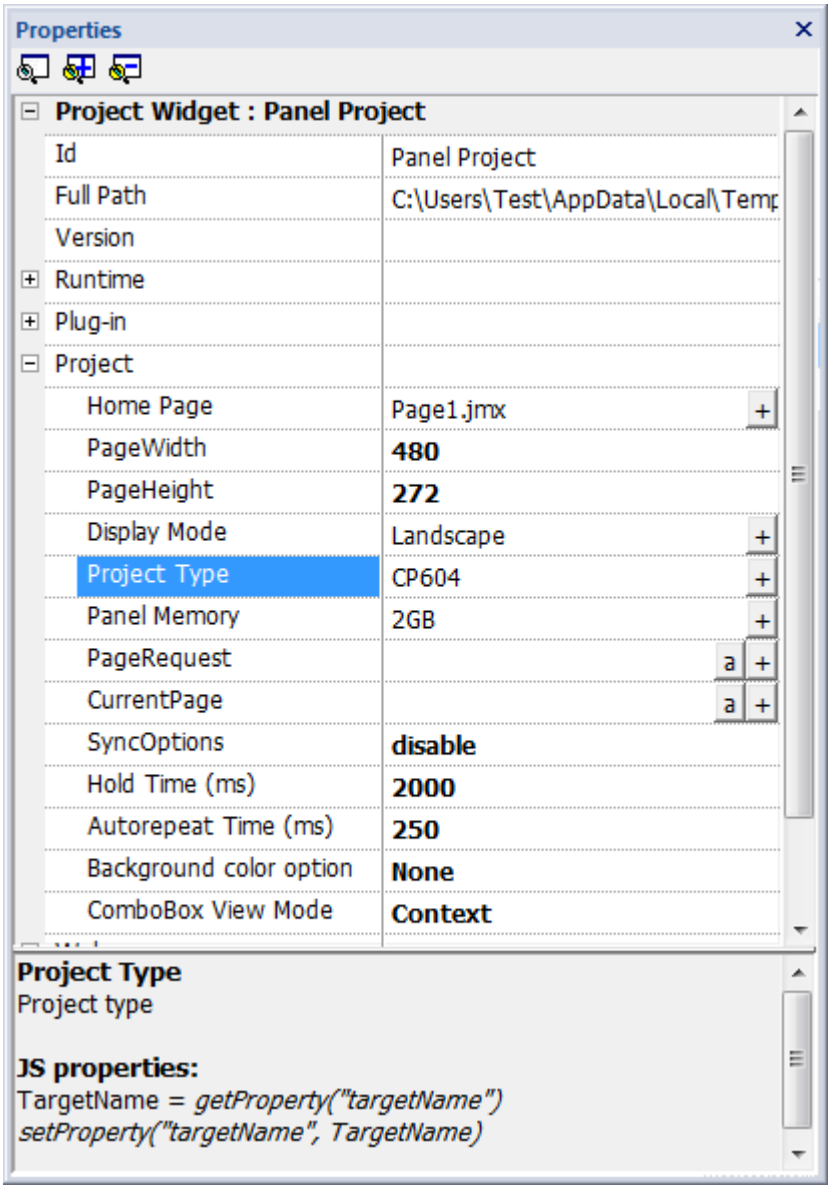


> *The panel projects can be compared in Automation Builder using the “Compare Objects” option.*

Changing panel type

1. In the Panel project, double-click *"Project properties"* to change the panel type to the panel which is used.

   ⇨ The Properties dialog is displayed.



2. In the Properties dialog, expand *"Project"* and click *"Project Type "*.

   ⇨ A project wizard dialog is displayed.

3. Select the desired panel type and click *"Finish"*.

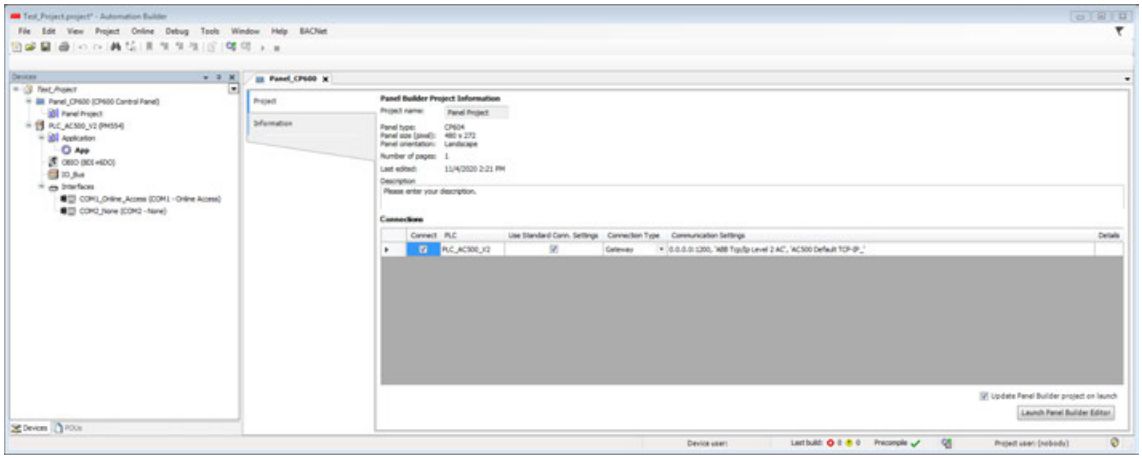
Project information

The project information view provides an overview of the Panel Builder project without opening the project. To open the project information, double-click the *"Panel_CP600"* object.

The project information is updated every time the Panel Builder project is edited. You can rename the Panel Builder project via context menu.

> *The project name is internally used as a base for the Panel Builder project file name. Therefore, the project name has to comply with general file name restrictions.*

The Panel Builder project information shows the list of PLCs added to the project.

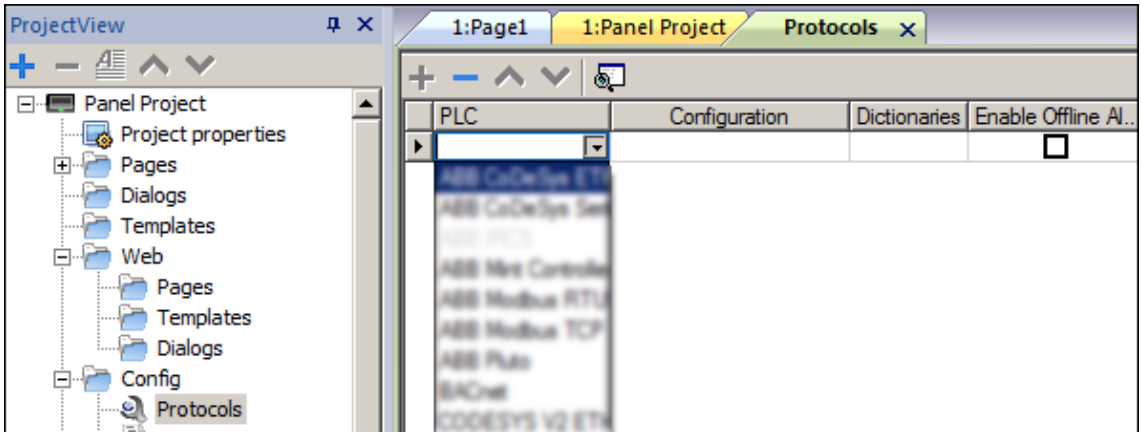


1.9.1.3 Configuring Panel Builder

Configuring communication protocols

> *The user can configure a panel project manually in Panel Builder editor when there is a need to create individual panel projects. Otherwise, the configuration is updated in the panel project while launching Panel Builder editor in Automation Builder.*

1. In the Panel Builder project structure, double-click *"Config ➔ Protocols"*.
2. Click + to add a protocol.



3. Select the desired protocol and set the IP address, port, protocol type and PLC models. Click *[OK]*.

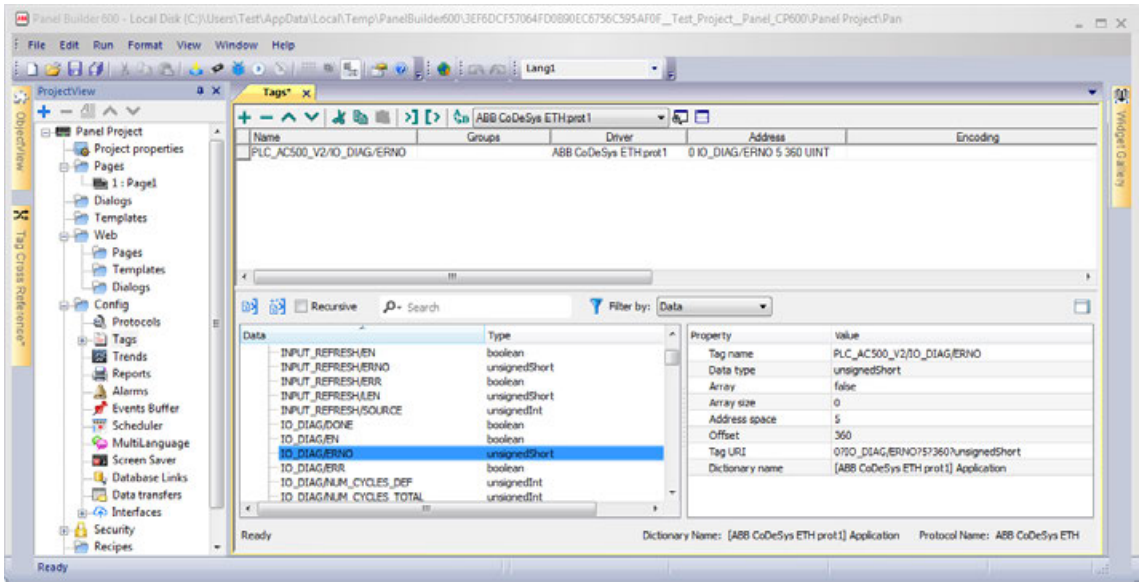
Importing tags

1. In the Panel project view, click *"Config ➔ Tags"*.
2. Select the protocol from the drop-down list and click ›] to import tags.

> *If the Panel Builder contains multiple tag importers, a dialog is displayed to select the required importer type.*

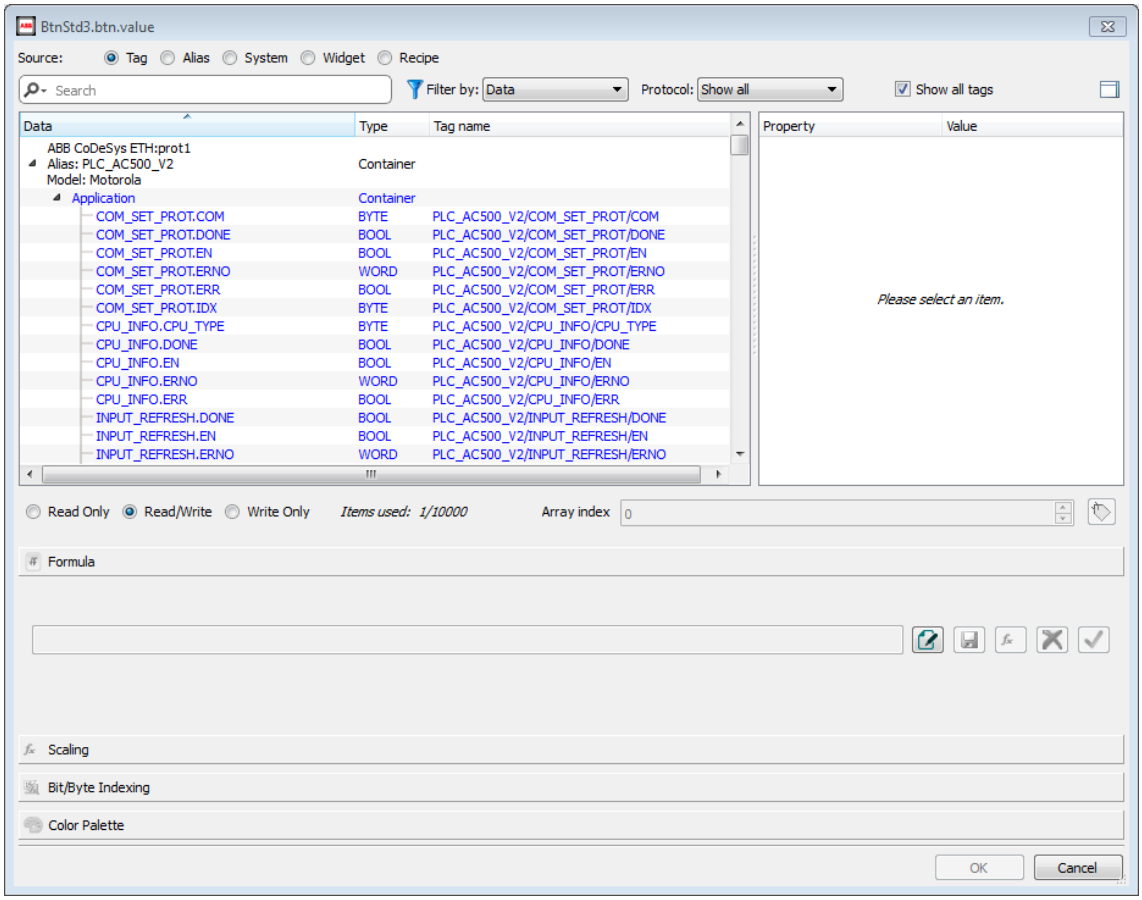
3. Select the symbol file which was exported to the file system.

4. In the lower part of the tag editor, mark the desired tags and click *"Import Tag (s)"* to import the tags to the Panel Builder project.



Attaching tags to widgets

1. In the project view, expand *"Pages"* and double-click **Page1**.
2. In the Panel Builder 600 main menu, select *"View* ➔ *Toolbars and Docking Windows* ➔ *Widget Gallery"*.
3. Drag-and-drop the desired widget to the page editor.
4. Right-click on the widget value and select *"Attach To"* to attach a tag to the widget.



5. Select the desired tag and select the desired option for the authorization *“Read Only”* or *“Read/ Write”* or *“Write Only”*. Then, click *[OK]*.

Downloading a project to panel

1. In the Panel Builder main menu, click *“Run ➔ Download To Target”*.
2. Select the CP600 project from the drop-down list and click *“Download”*.

Importing an existing Panel Builder project

1. In the Automation Builder device tree, right-click the Panel project and click *“Import ➔ Panel Builder Project”*.

   System prompts to overwrite the exiting project object data.
2. Click *“Yes”* to confirm.
3. Select the existing Panel Builder 600 project from the file system and click *“Open”*.

   ⇨ The imported project is displayed.

Exporting Panel Builder project

1. In the Automation Builder device tree, right-click the Panel Builder 600 project and click *“Export ➔ Panel Builder Project”*.
2. Click *“Browse”* and select the desired location in the file system and save the project file.

   ⇨ A success message is displayed, if the project file exports successfully.

When you double-click the Panel Builder project node, the compressed information of the node is extracted into a temporary folder and then the external Panel Builder program is started. After the external Panel Builder program is closed, the corresponding Panel Builder files can be compressed back into the node and saved in the Automation Builder project.

We recommend to edit the Panel Builder project by starting Panel Builder through the Automation Builder. You can also export a Panel Builder project to the file system to edit the project by using the external Panel Builder. Then, reimport it to Automation Builder.

1.9.2 SCADA Integration

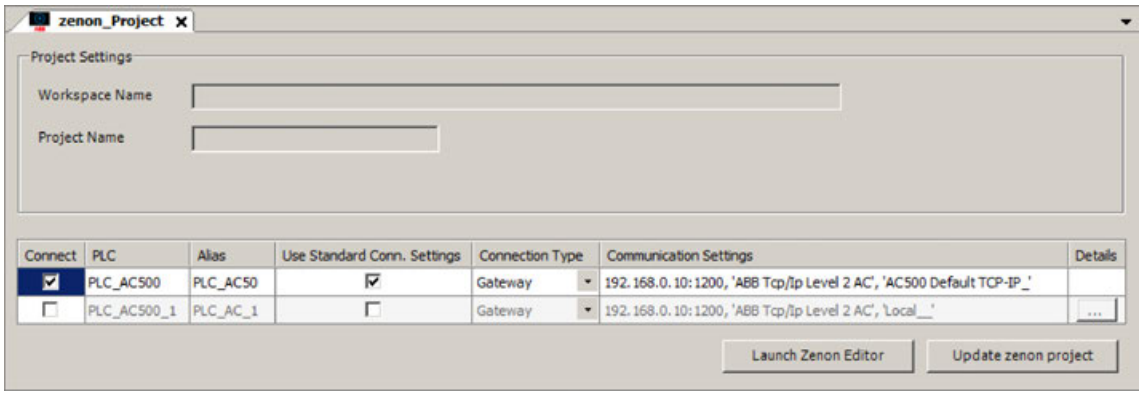
Overview

This document describes SCADA integration configuration in Automation Builder using zenon editor. The configured device network address information and variables are synchronized with zenon editor to avoid double entry.

The Automation Builder supports both standard and multi-user functionality.

1.9.2.1 Creating Workspace and Project

1. In the device tree, double-click *“zenon_Project”*.



   ⇨ To launch the zenon editor, click *[Launch Zenon Editor]*.

   To update the zenon project with latest changes of application program, click *[Update zenon project]*.

2. Select the required PLC and select the *“Use Standard Conn. Settings”* option to use as a standard gateway connection.

   This enables the user to use the same communication settings that Automation Builder uses to communicate to the PLC.

> *The configured gateway communication settings made in Automation Builder are displayed in the column 'Connection Type'.*

   As an alternative you can create custom communication settings: Deselect the *“Use Standard Conn. Settings”* option and click the button in the 'Details' column.

3. Click *[Launch Zenon Editor]* to create a new workspace and project.

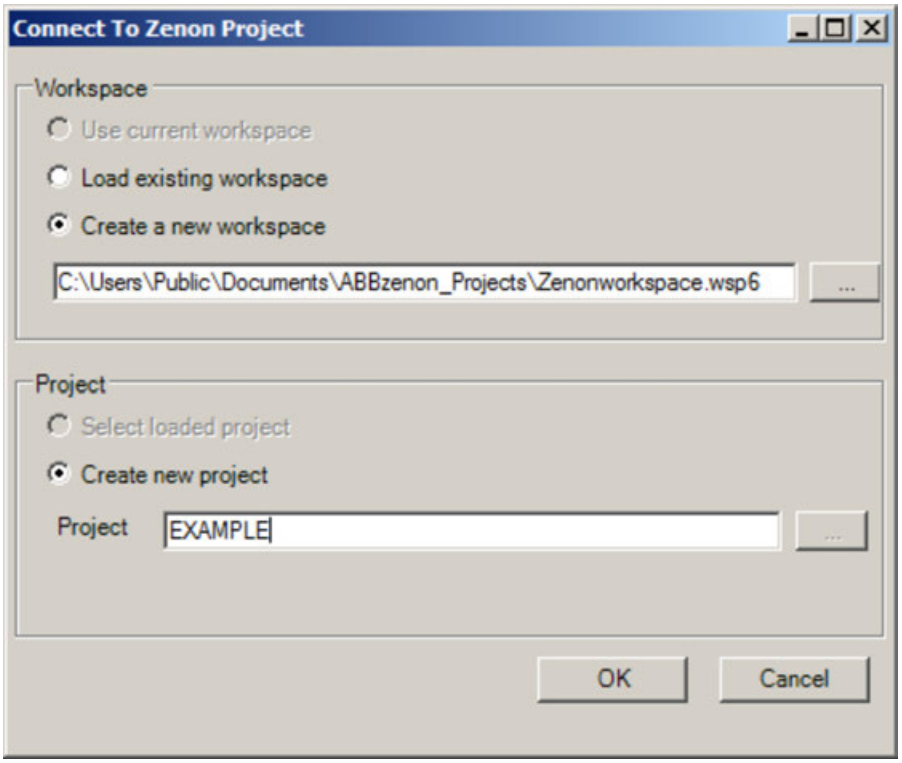


Fig. 1183: Connect to zenon project

⇨

> *If Zenon Editor is already running, then select the "Use current workspace" option.*

4. Select the *"Create a new workspace"* option and select the file location to create a new workspace.
5. Select the *"Create new project"* option to create a project.
   ⇨ ABB zenon editor is displayed.

> *If you update or change an Automation Builder project with new variables or data types (new, modified, deleted), recompile the application to refresh the symbol file and click [Update zenon project].*

After creating the project and workspace in Automation Builder, it is not required to set it again for the zenon object. A double-clicking on the zenon project shows the previously configured zenon project and the workspace.

1.9.2.2 Loading existing Workspace and Project

You can load an existing workspace and project to ABB zenon supervisor.

1. In the zenon_Project screen, click *[Update zenon project]*.
   ⇨ Connection to the zenon project dialog is displayed.
2. In the workspace area, enable *"Load existing workspace"* and select the location.

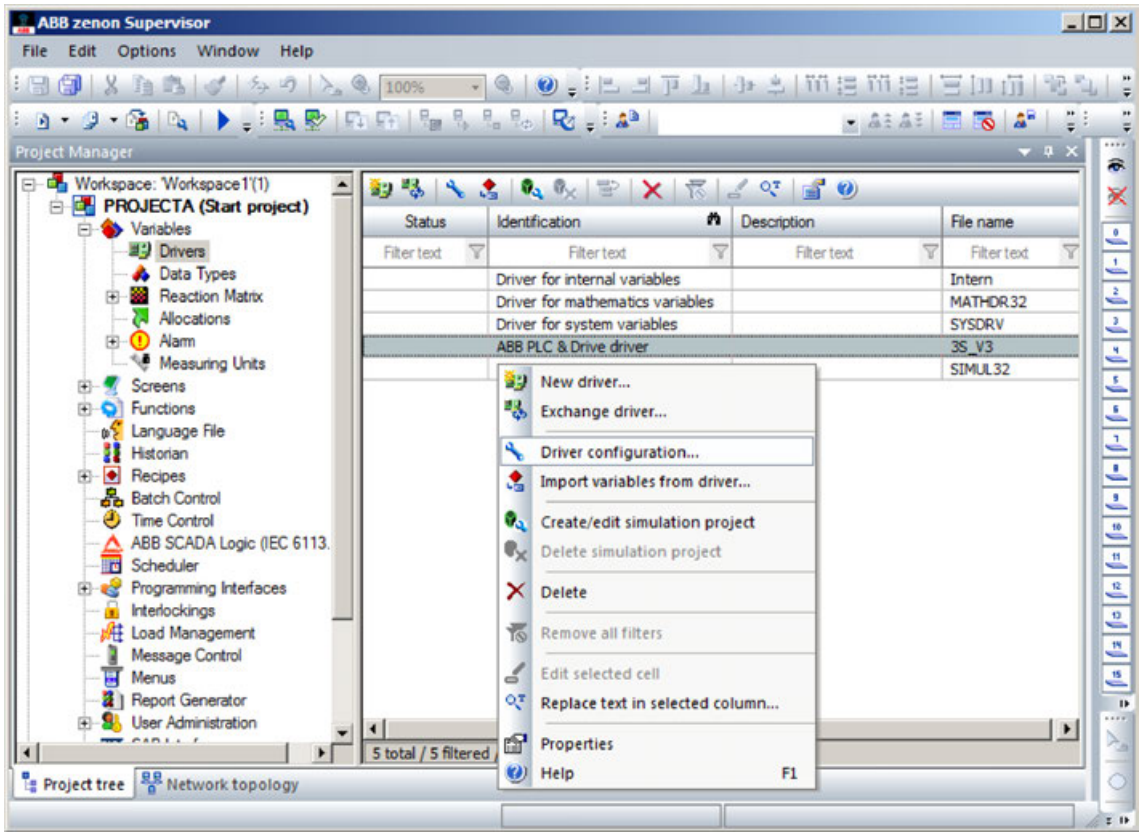
3. In the project area, enable *“Select loaded project”* and click *[OK]*.

   ⇨ Zenon editor loads the selected existing workspace and the project.

1.9.2.3 Checking the Gateway Settings in a Zenon Project

The gateway settings configured in Automation Builder can be checked in a zenon project. The IP address configured in Automation Builder are displayed in the zenon driver configuration.

In the Project Manager structure of the zenon editor, click *“Variables ➔ Drivers”* to configure the driver configuration.



The *“Settings”* tab shows all gateway settings based on the number of configured PLCs in Automation Builder. The IP address should be similar to the project gateway settings in Automation Builder.

> *In the zenon project window, the **Connect** column should be checked to transfer the desired number of PLC connection settings to the zenon editor.*

1.9.2.4 Generating a Symbol File

Before generating the symbol file, define the variables in theCODESYS application.

1. In the CODESYS application main menu, click *“Project ➔ Options”*.
2. In the *“Options”* dialog, click *“Symbol configuration”*. Enable *“Dump symbol entries”* and *“Dump XML symbol table”* and click *“Configure symbol file”*.

   ⇨ Set object attributes dialog is displayed.

3. Enable *“Export variables of object”*. If this option has a gray background, double-click on it to activate.

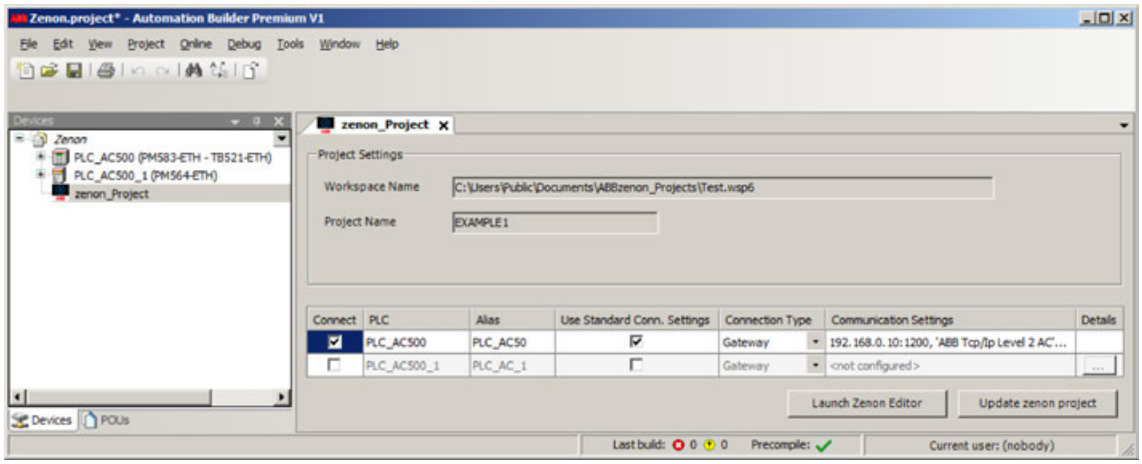
4. In the CODESYS application window, click at the bottom of the window and click *"Resources ➔ Tools ➔ Target settings"*.
5. In the target settings dialog, open the *"General"* tab and enable *"Download symbol file"*.
6. From the CODESYS application main menu, select *"Project ➔ Build"* to compile the project.

> *Precondition to generate a symbol file is to create the application and perform a PLC program build in CODESYS application.*

The symbol file is generated after the build. The data exchange can be transferred to the zenon project by clicking *[Update zenon project]* in Automation Builder.

1.9.2.5 Updating Standard Data Types

The standard data types created in CODESYS application can be updated to the zenon project by clicking on *"Update zenon project"*.



> *Data types and variables can be updated from the desired number of PLCs configured in the zenon project of Automation Builder.*

In the zenon project, double-click *"Variables"* and check the updated standard data type.

1.9.2.6 Creating Data Types

1. In the CODESYS application open the *"Data types"* tab. Right-click *"Data types ➔ Add object"* to create a new data type.
2. Enter the user defined data type name.
3. In *"POUs"* tab, add the user defined variable data type and compile.

   ⇨ The user defined data type is created and can be imported in the zenon editor.

> *If you modify or delete the data types in CODESYS application, compile with "Rebuild all option".*

1.9.2.7 Importing Data Types in zenon Editor

1. In the zenon project, click *[Update zenon project]* to update the data types.
2. Click *"Update"* to update the variables and data types to the zenon project.
   ⇨ The user defined variables and data types are imported to the zenon project.

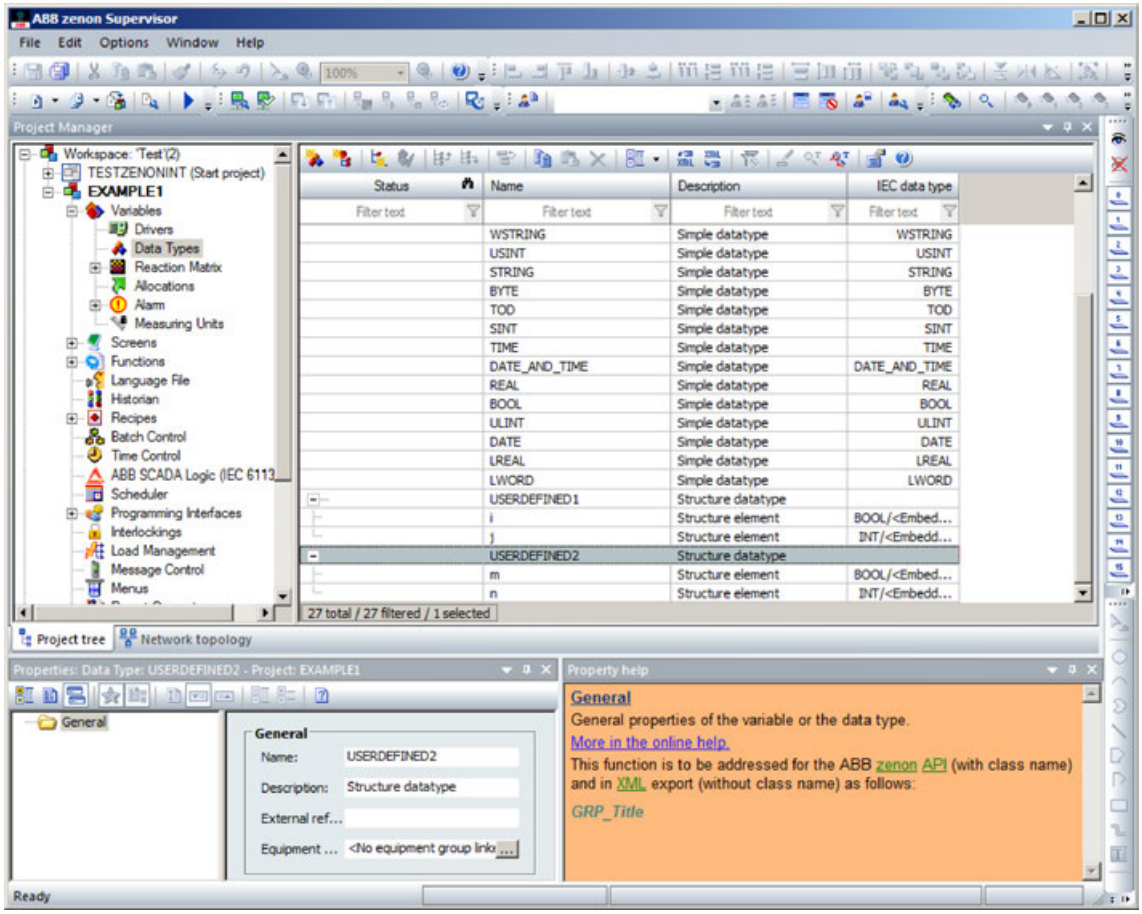


Fig. 1184: User defined variables

1.10 Contact ABB

ABB AG
Eppelheimer Str. 82
69123 Heidelberg, Germany

PLC support: +49 (0)6221 701 1444, plc.support@de.abb.com

abb.com/plc
abb.com/automationbuilder

abb.com/contacts

2 Index

1, 2, 3 ...

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z

—
ABB AG
Eppelheimer Str. 82
69123 Heidelberg, Germany
Telephone: +49 (0)6221 701 1444
E-mail: plc.support@de.abb.com
abb.com/plc
abb.com/automationbuilder
abb.com/contacts

3ADR010582, 3, en_US